FUJITSU

FUJITSU Software
Interstage Application Server

アプリケーション作成ガイド（CORBAサービス編）

Windows/Solaris/Linux

B1WS-1020-04Z0(00)
2014年10月

まえがき

本書の目的

本書は、マルチ言語サービスのCORBAサービスのアプリケーションを作成するためのマニュアルです。CORBAサービスを利用してアプリケーション開発を行うために必要なプログラミングの方法と、業務運用を実現するためプログラミングの手法、手順、および定義を説明しています。

Java EEのアプリケーションを作成する場合は、“Java EE運用ガイド”を参照してください。

前提知識

本書を読む場合、以下の知識が必要です。

- 使用するOSに関する基本的な知識
- C言語に関する基本的な知識
- C++言語に関する基本的な知識
- COBOLに関する基本的な知識
- OOCOBOLに関する基本的な知識
- Java言語に関する基本的な知識
- インターネットに関する基本的な知識
- オブジェクト指向技術に関する基本的な知識
- 分散オブジェクト技術(CORBA)に関する基本的な知識
- リレーショナルデータベースに関する基本的な知識

本書の構成

本書は以下の構成になっています。

第1部 アプリケーション開発(基本編)

第1章 CORBAアプリケーションの基礎知識
CORBAアプリケーションを作成するための基礎知識について説明します。

第2章 CORBAアプリケーション開発時の留意事項
CORBAアプリケーションを開発する際の留意事項について説明します。

第3章 アプリケーションの開発（C言語）
CORBAアプリケーションをC言語で開発する手順について説明します。

第4章 アプリケーションの開発（C++言語）
CORBAアプリケーションをC++言語で開発する手順について説明します。

第5章 アプリケーションの開発（Java言語）
CORBAアプリケーションをJava言語で開発する手順について説明します。

第6章 アプリケーションの開発（COBOL）
CORBAアプリケーションをCOBOLで開発する手順について説明します。

第7章 CORBAアプリケーションの基本的な機能
CORBAアプリケーションの基本的な機能について説明します。

第2部 アプリケーション開発(応用編)

第8章 動的起動インタフェース
動的起動インタフェースのプログラミングについて説明します。

第9章 ネーミングサービスのプログラミング
ネーミングサービスが提供するインタフェースとプログラミングについて説明します。

第10章 インタフェースリポジトリサービスのプログラミング
インタフェースリポジトリサービスが提供するインタフェースとプログラミングについて説明します。

第11章 CORBAアプリケーションの高度な使い方
アプリケーションの高度な使い方について説明します。

第12章 CORBAインタフェース
動的にデータを操作するためのインタフェースについて説明します。

第13章 ネーミングサービスのイニシャルリファレンスの獲得
ネーミングサービスのイニシャルリファレンスの獲得について説明します。

付録A IDL定義

IDL定義ファイルについて説明します。

付録B 提供プログラム

提供しているプログラムについて説明します。

付録C インタフェース定義情報の移出入

動的起動インタフェースを使用する場合に必要な、運用サーバのインタフェースリポジトリへのインタフェース定義情報の方法を説明します。

付録D サンプルプログラム

CORBAサービスおよびPortable-ORBで提供しているサンプルプログラムについて説明します。

付録E プロセスバインド機能を用いたセション管理の例

プロセスバインド機能を用いたセション管理について、例を挙げて説明します。

付録F 旧バージョンからの移行上の注意

旧バージョンから移行する際の注意事項について説明します。

付録G 旧互換機能

CORBAサービスの旧バージョン互換用の機能について説明します。

輸出許可

本ドキュメントを輸出または第三者へ提供する場合は、お客様が居住する国および米国輸出管理関連法規等の規制をご確認のうえ、必要な手続きをおとりください。

著作権

Copyright 1999-2014 FUJITSU LIMITED

2014年10月　第4版
2012年8月　初版

目　次

第1部 アプリケーション開発(基本編)....1

第1章 CORBAアプリケーションの基礎知識....2
1.1 CORBAアプリケーションの基本モデル....2
1.1.1 2階層モデル....2
1.1.2 3階層モデル....3
1.2 CORBAサービスの運用形態....3
1.2.1 クライアントの運用形態....4
1.2.2 サーバの運用形態....5
1.3 CORBAアプリケーションの設計....7
1.3.1 アプリケーションの開発言語....7
1.3.2 サーバアプリケーションの起動タイプ....7
1.3.3 多重制御....8
1.3.4 インスタンス管理システム....9
1.3.5 クライアントアプリケーションのインタフェース....9
1.3.6 サーバアプリケーションのインタフェース....10
1.4 CORBAアプリケーションの開発手順....10
1.5 CORBAアプリケーションの運用手順....12

第2章 CORBAアプリケーション開発時の留意事項....14
2.1 コーディング....14
2.1.1 データ領域の獲得/解放処理（C, C++, COBOL）....14
2.1.2 シグナル処理（C, C++, Java, COBOL）....15
2.1.3 子プロセス/スレッドの生成終了処理（C, C++, Java, COBOL）....15
2.1.4 ORBの初期化処理（C, C++, COBOL）....17
2.1.5 サーバアプリケーションの例外処理（C, C++, COBOL）....17
2.1.6 サーバ・クライアント機能の同時実装....17
2.1.7 oneway機能....17
2.1.8 その他....18
2.2 コンパイル・リンク....18
2.2.1 スレッドモードとプロセスモード....18
2.2.2 CORBAサーバアプリケーション作成時の共通留意事項....21
2.2.3 Windows(R)アプリケーションに関する注意事項....21
2.2.4 Solarisアプリケーションに関する注意事項....24
2.2.5 Linuxアプリケーションに関する注意事項....24
2.2.6 サーバアプリケーションのコンパイル・リンク例 (Solaris)....25
2.2.6.1 Cサーバアプリケーション(スレッドモード)の場合....25
2.2.6.2 Cサーバアプリケーション(プロセスモード)の場合....25
2.2.6.3 COBOLサーバアプリケーション(スレッドモード)の場合....26
2.2.6.4 COBOLサーバアプリケーション(プロセスモード)の場合....26
2.2.7 Javaアプリケーションのコンパイル....27
2.3 トラブル発生時の原因調査....27
2.3.1 保守情報....27
2.4 トランザクションアプリケーションからCORBAアプリケーションへ移行....28
2.5 ネットワーク制御....32
2.5.1 クライアントが使用するIPアドレス....32
2.5.2 コネクション接続....32

第3章 アプリケーションの開発（C言語）....33
3.1 アプリケーションの開発手順....33
3.1.1 アプリケーションのインタフェース定義(IDL)の記述....33
3.1.2 IDLファイルのコンパイル....33
3.1.3 サーバアプリケーションのプログラミング....33
3.1.4 クライアントアプリケーションのプログラミング....34
3.1.5 アプリケーションのコンパイルとリンク....34

3.1.6 サーバアプリケーションの登録....34
3.2 サーバアプリケーションのプログラミング（静的スケルトンインタフェース）....34
3.2.1 初期化....35
3.2.2 サーバの活性化....35
3.2.3 インタフェース実装関数....36
3.2.4 サーバの非活性化....37
3.3 サーバアプリケーションの例外処理....37
3.4 サーバアプリケーションの登録....39
3.4.1 インプリメンテーションリポジトリへの登録....39
3.4.2 ネーミングサービスへの登録....40
3.5 クライアントアプリケーションのプログラミング（静的起動インタフェース）....43
3.5.1 初期化....43
3.5.2 ネーミングサービスのオブジェクトリファレンスの獲得....43
3.5.3 サーバアプリケーションのオブジェクトリファレンスの獲得....44
3.5.4 メソッドの呼出し....44
3.6 クライアントアプリケーションの例外処理....44
3.7 データ型に対するマッピング....46
3.7.1 基本データ型....47
3.7.2 文字列型....47
3.7.3 ワイド文字列型....49
3.7.4 any型....52
3.7.5 シーケンス型....55
3.7.6 構造体....58
3.7.7 共用体....61
3.7.8 配列....65
3.7.9 属性宣言(attribute)のマッピング....68
3.7.10 パラメタ受渡しで使用するデータ型....70
3.8 any型、sequence型のリリースフラグ....71
3.9 アプリケーション作成時の注意事項....72

第4章 アプリケーションの開発（C++言語）....73
4.1 アプリケーションの開発手順....73
4.1.1 アプリケーションのインタフェース定義(IDL)の記述....73
4.1.2 IDLファイルのコンパイル....73
4.1.3 サーバアプリケーションのプログラミング....74
4.1.4 クライアントアプリケーションのプログラミング....74
4.1.5 アプリケーションのコンパイルとリンク....74
4.1.6 サーバアプリケーションの登録....74
4.2 サーバアプリケーションのプログラミング（Basic Object Adapter：BOA）....75
4.2.1 静的スケルトンインタフェース....75
4.2.1.1 初期化....75
4.2.1.2 サーバの活性化....76
4.2.1.3 インタフェース実装関数....77
4.2.1.4 サーバの非活性化....80
4.2.2 サーバアプリケーションの例外処理....80
4.3 サーバアプリケーションのプログラミング（Portable Object Adapter：POA）....82
4.3.1 インスタンス管理とアプリケーション形態....82
4.3.1.1 アプリケーション形態の種別....82
4.3.1.2 各アプリケーション形態....86
4.3.1.3 アプリケーション形態の比較....91
4.3.2 POA概要....91
4.3.2.1 POAとは....91
4.3.2.2 POAのアーキテクチャ....92
4.3.2.3 POAオブジェクト....94
4.3.2.4 オブジェクトリファレンスの生成....99
4.3.2.5 オブジェクトリファレンス、オブジェクトID、Servantオブジェクトの関連....100
4.3.2.6 オブジェクトの活性化....101

4.3.2.7 リクエスト処理.....102
4.3.2.8 暗黙的活性化(Implicit Activation).....103
4.3.2.9 POAManagerオブジェクト.....103
4.3.2.10 ServantManagerオブジェクト.....104
4.3.2.11 AdapterActivatorオブジェクト.....108
4.3.3 サーバアプリケーションと環境設定との関連付け.....109
4.3.3.1 インプリメンテーション情報との関連付け.....109
4.3.3.2 オブジェクトリファレンスの生成方法.....110
4.3.3.3 POA使用方法例.....111
4.3.4 静的スケルトンインタフェース.....113
4.3.4.1 初期化.....114
4.3.4.2 RootPOAのオブジェクトリファレンスの獲得.....115
4.3.4.3 子孫POAの作成.....115
4.3.4.4 インタフェースの実装.....116
4.3.4.5 AOMへの登録(活性化).....117
4.3.4.6 ネーミングサービスへの登録.....117
4.3.4.7 オブジェクトリファレンスの事前生成／登録.....118
4.3.4.8 POAManagerの活性化と終了待機.....118
4.3.4.9 アプリケーション作成時の注意点.....119
4.3.5 サーバアプリケーションの実装アプローチ.....119
4.3.5.1 継承方式と代理方式.....120
4.3.5.2 継承方式のServant実装.....120
4.3.5.3 代理方式のServant実装.....121
4.3.5.4 継承方式と代理方式の比較.....123
4.3.6 クライアントとのコネクション切断時のインスタンス解放.....123
4.3.6.1 インスタンス解放処理を行うクラスの作成.....124
4.3.6.2 インスタンス解放処理の実装.....124
4.3.6.3 インスタンス解放処理を行うクラスインスタンスの登録.....124
4.3.6.4 インスタンス解放処理を行うクラスインスタンスの変更・削除.....124
4.3.7 サーバアプリケーションのプログラミング例.....125
4.3.7.1 Default Servant使用例(デフォルトインスタンス方式).....125
4.3.7.2 Active Object Map(AOM)使用例 (Factory-1方式).....128
4.3.7.3 Servant Activator使用例 (Factory-2方式).....133
4.3.7.4 Servant Locator使用例 (ユーザインスタンス管理方式).....139
4.3.7.5 AdapterActivator使用例 (find_POA時).....146
4.3.7.6 AdapterActivator使用例 (リクエスト受信時).....154
4.3.7.7 代理方式の実装例(デフォルトインスタンス方式の例).....159
4.3.7.8 Active Object Map(AOM)使用例(Factory-1方式＋クライアントとのコネクション切断時のインスタンス解放).....163
4.4 サーバアプリケーションの登録.....172
4.4.1 インプリメンテーションリポジトリへの登録.....172
4.4.2 ネーミングサービスへの登録.....173
4.5 クライアントアプリケーションのプログラミング.....176
4.5.1 静的起動インタフェース.....176
4.5.1.1 初期化.....177
4.5.1.2 ネーミングサービスのオブジェクトリファレンスの獲得.....177
4.5.1.3 サーバアプリケーションのオブジェクトリファレンスの獲得.....177
4.5.1.4 メソッドの呼出し.....178
4.5.2 クライアントアプリケーションの例外処理.....178
4.6 データ型に対するマッピング.....181
4.6.1 基本データ型.....181
4.6.2 文字列型.....182
4.6.3 ワイド文字列型.....184
4.6.4 any型.....186
4.6.5 シーケンス型.....195
4.6.6 構造体.....204
4.6.7 共用体.....207
4.6.8 配列.....214

4.6.9 インタフェース宣言(interface)のマッピング....217
4.6.10 属性宣言(attribute)のマッピング....218
4.6.11 パラメタ受渡しで使用するデータ型....220
4.6.12 any型、sequence型のリリースフラグ....221
4.7 varクラス....223
4.7.1 String_varクラス....224
4.7.2 WString_varクラス....227
4.7.3 Any_varクラス....231
4.7.4 ユーザ定義varクラス....234
4.8 アプリケーション作成時の注意事項....239

第5章 アプリケーションの開発(Java言語)....240
5.1 Javaアプレットの開発手順(プレインストール型Javaライブラリ)....240
5.1.1 HTMLファイルの記述....241
5.1.2 アプレットのプログラミング....241
5.1.3 IDL生成ファイルとアプレットのコンパイル....242
5.1.4 Javaクラスファイルのアーカイブ....242
5.2 Javaアプレットの開発手順(Portable-ORB)....244
5.2.1 HTMLファイルの記述....244
5.2.2 アプレットのプログラミング....246
5.2.3 IDL生成ファイルとアプレットのコンパイル....246
5.2.4 Javaクラスファイルのアーカイブ....247
5.2.5 WebサーバへのJavaアプレットの登録....248
5.3 サーバアプリケーションの開発手順....248
5.3.1 IDLファイルの作成とコンパイル....249
5.3.2 サーバアプリケーションの作成....249
5.3.3 Javaファイルのコンパイル....250
5.3.4 起動用ファイルの作成....250
5.3.5 インプリメンテーションリポジトリの登録....251
5.3.6 セキュリティーマネージャーの設定....252
5.4 CORBAアプリケーションの実行....252
5.4.1 Java実行環境....252
5.4.2 ORB(Object Request Broker)の指定....253
5.5 クライアントの環境設定(プレインストール型Javaライブラリ)....255
5.5.1 HTMLファイルの記述....255
5.5.2 Javaライブラリに対する権限の設定....255
5.5.3 ライブラリの設定....257
5.5.4 セキュリティーマネージャーの設定....257
5.6 クライアントの環境設定(Portable-ORB)....257
5.6.1 HTMLファイルの記述....257
5.6.2 Javaライブラリに対する権限の設定....258
5.6.3 Javaライブラリに対する署名の設定....260
5.6.4 ライブラリの設定....260
5.6.5 Portable-ORB動作環境ファイルの指定....261
5.7 サーバの環境設定....265
5.8 アプレットのデジタル署名....265
5.8.1 デジタル署名の概要....266
5.8.2 ダウンロードオブジェクトと署名対象オブジェクト....266
5.8.3 デジタル署名手順....267
5.8.4 policytoolコマンドの設定....270
5.9 クライアントアプリケーションのプログラミング(静的起動インタフェース)....281
5.9.1 初期化....282
5.9.2 ネーミングサービスのオブジェクトリファレンスの獲得....283
5.9.3 サーバアプリケーションのオブジェクトリファレンスの獲得....283
5.9.4 メソッドの呼び出し....283
5.10 クライアントアプリケーションの例外処理....284
5.11 簡単なサーバアプリケーションの作成例....287

5.11.1 作業の流れ....288
5.11.2 IDLファイルの作成とコンパイル....288
5.11.3 サーバアプリケーションのソースファイル作成....289
5.11.4 クライアントアプリケーションのソースファイル作成....289
5.11.5 Javaファイルのコンパイル....289
5.11.6 defファイルの作成と登録....289
5.11.7 起動用ファイルの作成....290
5.11.8 アプリケーションの実行....290
5.11.9 アプリケーションの実行結果....291
5.11.10 サーバアプリケーション情報の削除....291
5.12 インスタンス管理とアプリケーション形態....291
5.12.1 アプリケーション形態の種別....291
5.12.2 各アプリケーション形態....296
5.12.3 アプリケーション形態の比較....300
5.13 サーバアプリケーションのプログラミング（POA概要）....300
5.13.1 POAとは....300
5.13.2 POAのアーキテクチャ....301
5.13.3 POAオブジェクト....303
5.13.4 オブジェクトリファレンスの生成....308
5.13.5 オブジェクトリファレンス、オブジェクトID、Servantオブジェクトの関連....309
5.13.6 オブジェクトの活性化....310
5.13.7 リクエスト処理....311
5.13.8 暗黙的活性化（Implicit Activation）....312
5.13.9 POAManagerオブジェクト....312
5.13.10 ServantManagerオブジェクト....313
5.13.11 AdapterActivatorオブジェクト....316
5.14 サーバアプリケーションと環境設定との関連付け....318
5.14.1 インプリメンテーション情報との関連付け....318
5.14.2 オブジェクトリファレンスの生成方法....318
5.14.3 POA使用方法例....319
5.15 サーバアプリケーションのプログラミング（静的スケルトンインタフェース）....322
5.15.1 初期化....322
5.15.2 RootPOAのオブジェクトリファレンスの獲得....323
5.15.3 子孫POAの作成....323
5.15.4 インタフェースの実装....324
5.15.5 AOMへの登録（活性化）....324
5.15.6 ネーミングサービスへの登録....325
5.15.7 オブジェクトリファレンスの事前生成／登録....325
5.15.8 POAManagerの活性化と終了待機....325
5.16 サーバアプリケーションの例外処理....326
5.17 サーバアプリケーションの実装アプローチ....328
5.17.1 継承方式と代理方式....328
5.17.2 継承方式のServant実装....328
5.17.3 代理方式のServant実装....329
5.17.4 継承方式と代理方式の比較....331
5.18 クライアントとのコネクション切断時のインスタンス解放....331
5.18.1 インスタンス解放処理を行うクラスの作成....332
5.18.2 インスタンス解放処理の実装....332
5.18.3 インスタンス解放処理を行うクラスインスタンスの登録....332
5.18.4 インスタンス解放処理を行うクラスインスタンスの変更・削除....332
5.19 サーバアプリケーションのプログラミング例....332
5.19.1 Default Servant使用例（デフォルトインスタンス方式）....332
5.19.2 Active Object Map（AOM）使用例（Factory-1方式）....336
5.19.3 Servant Activator使用例（Factory-2方式）....340
5.19.4 Servant Locator使用例（ユーザインスタンス管理方式）....345
5.19.5 AdapterActivator使用例（find_POA時）....351
5.19.6 AdapterActivator使用例（リクエスト受信時）....358

5.19.7 代理方式の実装例（デフォルトインスタンス方式の例）....362
5.19.8 Active Object Map（AOM）使用例（Factory-1方式＋インスタンス解放）....366
5.20 データ型に対するマッピング....372
5.20.1 基本データ型....372
5.20.2 文字列型....372
5.20.3 ワイド文字列型....375
5.20.4 列挙型....378
5.20.5 any型....383
5.20.6 シーケンス型....389
5.20.7 構造体....394
5.20.8 共用体....401
5.20.9 配列....409
5.20.10 属性宣言(attribute)のマッピング....416
5.20.11 定数宣言(const)のマッピング....419
5.20.12 パラメタ受渡しで使用するデータ型....420
5.21 注意事項....422
5.21.1 アプリケーション作成時の注意事項....422
5.21.2 クライアント環境でのアプレット運用時の注意事項....422
5.21.3 その他の注意事項....423

第6章 アプリケーションの開発（COBOL）....424
6.1 アプリケーションの開発手順....424
6.1.1 アプリケーションのインタフェース定義(IDL)の記述....424
6.1.2 IDLファイルのコンパイル....424
6.1.3 サーバアプリケーションのプログラミング....425
6.1.4 クライアントアプリケーションのプログラミング....425
6.1.5 アプリケーションのコンパイルとリンク....425
6.1.6 サーバアプリケーションの登録....425
6.2 サーバアプリケーションのプログラミング（静的スケルトンインタフェース）....425
6.2.1 初期化....426
6.2.2 サーバの活性化....428
6.2.3 インタフェース実装関数....429
6.2.4 サーバの非活性化....431
6.3 サーバアプリケーションの例外処理....432
6.4 サーバアプリケーションの登録....435
6.4.1 インプリメンテーションリポジトリへの登録....435
6.4.2 ネーミングサービスへの登録....438
6.5 クライアントアプリケーションのプログラミング（静的起動インタフェース）....442
6.5.1 初期化....442
6.5.2 ネーミングサービスのオブジェクトリファレンスの獲得....444
6.5.3 サーバアプリケーションのオブジェクトリファレンスの獲得....444
6.5.4 メソッドの呼び出し....445
6.6 クライアントアプリケーションの例外処理....446
6.7 アプリケーションのテスト....448
6.7.1 サーバアプリケーションのテスト方法....448
6.7.2 クライアントアプリケーションのテスト方法....450
6.8 データ型に対するマッピング....450
6.8.1 文字列型....452
6.8.2 ワイド文字列型....457
6.8.3 any型....462
6.8.4 シーケンス型....468
6.8.5 構造体....479
6.8.6 共用体....484
6.8.7 固定小数点型....491
6.8.8 配列....497
6.8.9 属性宣言(attribute)のマッピング....505
6.8.10 パラメタ受渡しで使用するデータ型....509

6.9 any型、sequence型のリリースフラグ....512
6.10 COBOLアプリケーション使用時の注意事項....513
6.11 COBOL用登録集について....531
6.11.1 使用例....531
6.11.2 登録集原文....533

第7章 CORBAアプリケーションの基本的な機能....544
7.1 クライアントタイムアウト機能....544
7.2 CORBAワークユニット....544

第2部 アプリケーション開発(応用編)....546

第8章 動的起動インタフェース....547
8.1 動的起動インタフェースについて....547
8.2 C言語の開発....547
8.2.1 初期化....547
8.2.2 ネーミングサービスのオブジェクトリファレンスの獲得....548
8.2.3 インタフェースリポジトリからサーバアプリケーション情報の獲得....548
8.2.4 パラメタの組立て....549
8.2.5 リクエストの作成....550
8.2.6 リクエストの送信....551
8.2.7 リクエストの削除....551
8.2.8 動的インタフェースでのパラメタの獲得／解放....552
8.3 C++言語の開発....553
8.3.1 初期化....554
8.3.2 ネーミングサービスのオブジェクトリファレンスの獲得....554
8.3.3 サーバアプリケーションのオブジェクトリファレンスの獲得....554
8.3.4 インタフェースリポジトリからサーバアプリケーション情報の獲得....554
8.3.5 パラメタの組立て....555
8.3.6 リクエストの作成....556
8.3.7 リクエストの送信....557
8.3.8 リクエストの削除....558
8.3.9 動的インタフェースでのパラメタの獲得／解放....558
8.4 Java言語の開発....567
8.4.1 初期化....567
8.4.2 ネーミングサービスのオブジェクトリファレンスの獲得....568
8.4.3 サーバアプリケーションのオブジェクトリファレンスの獲得....568
8.4.4 インタフェースリポジトリからサーバアプリケーション情報の獲得....568
8.4.5 パラメタの組立て....569
8.4.6 リクエストの作成....569
8.4.7 リクエストの送信....570
8.5 COBOL言語の開発....570
8.5.1 初期化....571
8.5.2 ネーミングサービスのオブジェクトリファレンスの獲得....572
8.5.3 インタフェースリポジトリからサーバアプリケーション情報の獲得....573
8.5.4 パラメタの組み立て....575
8.5.5 リクエストの作成....576
8.5.6 リクエストの送信....578
8.5.7 リクエストの削除....579

第9章 ネーミングサービスのプログラミング....580
9.1 ネーミングサービスの概要....580
9.2 ネーミングコンテキストインタフェース....584
9.2.1 ネーミングコンテキストインタフェースで扱うデータ型....584
9.2.2 ネーミングコンテキストインタフェースの呼び出しで発生する例外....585
9.2.3 バインディングの作成....587
9.2.4 バインディングの検索....587
9.2.5 バインディングの削除....588

9.2.6 ネーミングコンテキストの作成....588
9.2.7 ネーミングコンテキストの削除....588
9.2.8 バインディングの一覧獲得....589
9.3 バインディングアイテレータインタフェース....589
9.4 文字列表記のバインディング名....590
9.4.1 文字列表記バインディング名の基本表記....590
9.4.2 文字列表記バインディング名のエスケープ機構....591
9.5 URLスキーマ....591
9.5.1 IOR URLスキーマ....591
9.5.2 corbaloc URLスキーマ....591
9.5.3 corbaname URLスキーマ....592
9.6 バインディング名、URL、IOR間の変換....593
9.7 ネーミングサービスのプログラミング例....593
9.7.1 コンテキスト配下のオブジェクトの検索....593
9.7.2 コンテキスト配下のオブジェクトの登録....600
9.7.3 コンテキスト配下のオブジェクトのリストの取得....608

第10章 インタフェースリポジトリサービスのプログラミング....618
10.1 インタフェースリポジトリサービスで管理するオブジェクトの種類....618
10.2 インタフェースリポジトリオブジェクトの包含／継承関係....618
10.3 インタフェースリポジトリサービスインタフェース....619
10.3.1 インタフェースリポジトリサービスが提供するインタフェース....619
10.3.2 インタフェースの継承関係....626
10.4 インタフェースリポジトリサービスのプログラミング....628
10.4.1 ルートリポジトリのオブジェクトリファレンスを獲得....629
10.4.2 インタフェースリポジトリのオブジェクトの検索....629
10.4.3 インタフェース情報の獲得....629
10.4.4 インタフェースリポジトリサービスのプログラミング例....630

第11章 CORBAアプリケーションの高度な使い方....645
11.1 Factory....645
11.1.1 サーバアプリケーションのプログラミング....645
11.1.2 クライアントアプリケーションのプログラミング....647
11.1.3 private領域の実装....647
11.1.4 クライアントごとのprivateデータの初期化....649
11.1.5 終了処理のプログラミング....651
11.1.6 クライアントとの接続切断時の処理関数の登録....651
11.2 オブジェクト単位のプロセスバインド....653
11.2.1 定義情報....653
11.2.2 使用するAPI....653
11.2.3 オブジェクトとインスタンスの関係....653
11.2.4 リクエスト振り分け方式....654
11.2.5 セションタイムアウト機能....654
11.2.6 プロセスの終了....654
11.3 複数のIDLファイルのコンパイル方法....655
11.3.1 #include文....655
11.3.2 -noincludeオプションの使用方法....656
11.4 複数インタフェースの1プロセスへの実装....658
11.4.1 同一プログラム内の別オブジェクトの呼び出し....658
11.4.2 1プロセス－1インタフェースの場合....659
11.4.3 1プロセス－複数インタフェースの場合....659
11.5 ライブラリの作成方法....661
11.5.1 ライブラリの作成方法....661
11.5.2 インタフェースの継承とライブラリ....664
11.5.3 ライブラリ作成時の注意事項....666
11.5.4 サーバアプリケーションのライブラリ化の例....667
11.6 スレッドモード・プロセスモードのアプリケーション....671
11.6.1 スレッドモード・プロセスモードのアプリケーションの作成....671

11.6.2 サーバ・クライアント同時実装時の注意事項....673
11.7 サーバアプリケーションの複数ホストへの配置....674
11.7.1 環境設定....674
11.7.2 サーバアプリケーションのプログラミング....674
11.7.3 サーバアプリケーションの登録....674
11.7.4 クライアントアプリケーションのプログラミング....675
11.8 ファイルによるオブジェクトリファレンスの配布....675
11.8.1 オブジェクトリファレンスのファイルへの出力....675
11.8.2 ファイルの配布....676
11.8.3 クライアントアプリケーションのプログラミング....676
11.9 通信資源の解放....676
11.10 出口関数....677
11.10.1 出口関数の登録....677
11.10.2 出口関数の最大処理時間監視....678
11.10.3 出口関数使用時の注意事項....678
11.10.4 出口関数の例....678
11.11 Interstage for GSとの連携....680

第12章 CORBAインタフェース....681
12.1 TypeCodeオブジェクト....681
12.1.1 TypeCodeオブジェクトとは....681
12.1.2 TypeCodeインタフェース....682
12.2 NVListオブジェクト....684
12.2.1 NVListオブジェクトとは....684
12.2.2 NVListインタフェース....684
12.3 Contextオブジェクト....685
12.3.1 Contextオブジェクトとは....685
12.3.2 Contextインタフェース....685
12.3.3 Contextインタフェースの使用例....688

第13章 ネーミングサービスのイニシャルリファレンスの獲得....692
13.1 ORBInitRef....692
13.2 ORBDefaultInitRef....692
13.3 イニシャルサービスの検索順序....693
13.4 corbaloc URLスキーマ....693
13.4.1 corbaloc URLスキーマ....693
13.4.2 corbaloc:rir URL....694
13.4.3 corbaloc:iiop URL....694

付録A IDL定義....696
A.1 IDLの書式....696
A.1.1 コメント....696
A.1.2 識別子....697
A.1.3 定数....697
A.1.4 区切り文字....698
A.1.5 名前とスコープ....698
A.1.6 C++との違い....701
A.1.7 プリプロセス....701
A.2 モジュール宣言....702
A.3 インタフェース宣言....703
A.3.1 インタフェースの継承....704
A.4 オペレーション宣言....706
A.5 属性宣言....707
A.6 定数宣言....708
A.7 データ型と型宣言....709
A.7.1 IDLでサポートしているデータ型....709
A.7.2 基本データ型....711
A.7.3 シーケンス型....712

A.7.4 構造体 713
A.7.5 共用体....714
A.7.6 固定小数点型....715
A.7.7 オブジェクトリファレンス....716
A.7.8 タイプコード....716
A.7.9 配列....717
A.8 例外宣言....717
A.9 IDLのフォーマット....718
A.10 コンポーネントトランザクションサービスにおけるIDLの利用範囲....721

付録B 提供プログラム....725
B.1 CORBAサービスの提供プログラム....725
B.1.1 インクルードファイル....725
B.1.2 ライブラリ....727
B.1.2.1 サーバ用ライブラリ....727
B.1.2.2 クライアント用ライブラリ....730
B.2 Portable-ORBの提供プログラム....731

付録C インタフェース定義情報の移出入....732
C.1 インタフェース定義情報の登録....732
C.2 プログラム開発から運用への手順....732
C.3 実行例....732
C.3.1 インタフェース定義情報の移出....732
C.3.2 インタフェース定義情報の移入....733

付録D サンプルプログラム....734
D.1 サンプルプログラムについて....734
D.1.1 サンプルプログラムの種別....734
D.1.2 サンプルプログラムの概要....736
D.2 使用時の注意事項....740
D.3 実行手順(Windows(R))....741
D.3.1 静的インタフェース....741
D.3.1.1 静的インタフェース／各種データ型(C言語)....741
D.3.1.2 静的インタフェース／各種データ型(C++言語)....744
D.3.1.3 静的インタフェース／各種データ型(Java言語)....744
D.3.1.4 静的インタフェース／四則演算(C言語)....745
D.3.1.5 静的インタフェース／四則演算(C++言語)....745
D.3.1.6 静的インタフェース／四則演算(Java言語)....746
D.3.2 動的起動インタフェース....747
D.3.2.1 動的起動インタフェース／基本(C言語)....748
D.3.2.2 動的起動インタフェース／基本(C++言語)....748
D.3.2.3 動的起動インタフェース／四則演算(C言語)....748
D.3.2.4 動的起動インタフェース／四則演算(C++言語)....749
D.3.3 ネーミングサービス....750
D.3.3.1 ネーミングサービス／listメソッド....751
D.3.3.2 ネーミングサービス／bind, resolve, destroyメソッド....751
D.3.4 インタフェースリポジトリ....752
D.3.4.1 インタフェースリポジトリ....752
D.3.5 POAインタフェース....753
D.3.5.1 POAインタフェース(C++言語)....753
D.3.5.2 POAインタフェース(Java言語)....754
D.3.6 プロセスバインド....754
D.3.6.1 プロセスバインド....754
D.3.7 各種API....754
D.3.7.1 net_disconnect....755
D.3.7.2 set_client_request_timer....757
D.3.7.3 string_to_object....757
D.3.8 Portable-ORB....758

D.3.8.1 Portable-ORB／各種データ型 759
D.3.8.2 Portable-ORB／POAインタフェース 760
D.3.8.3 Portable-ORB／四則演算 760
D.3.8.4 Portable-ORB／各種API（net_disconnect） 762
D.3.8.5 Portable-ORB／各種API（set_client_request_timer） 764
D.3.8.6 Portable-ORB／各種API（string_to_object） 764
D.3.9 その他 766
D.3.9.1 その他（C言語） 766
D.3.9.2 その他（C++言語） 766
D.3.9.3 その他（Java言語） 766
D.4 実行手順（Solaris/Linux） 767
D.4.1 静的インタフェース 768
D.4.1.1 静的インタフェース／各種データ型（C言語） 768
D.4.1.2 静的インタフェース／各種データ型（C++言語） 770
D.4.1.3 静的インタフェース／各種データ型（Java言語） 770
D.4.1.4 静的インタフェース／各種データ型（COBOL） 771
D.4.1.5 静的インタフェース／各種データ型（Windows(R)クライアント） 773
D.4.1.6 静的インタフェース／四則演算（C言語） 774
D.4.1.7 静的インタフェース／四則演算（C++言語） 775
D.4.1.8 静的インタフェース／四則演算（Java言語） 775
D.4.1.9 静的インタフェース／四則演算（COBOL） 777
D.4.2 動的起動インタフェース 777
D.4.2.1 動的起動インタフェース／基本（C言語） 778
D.4.2.2 動的起動インタフェース／基本（C++言語） 778
D.4.2.3 動的起動インタフェース／四則演算（C言語） 778
D.4.2.4 動的起動インタフェース／四則演算（C++言語） 779
D.4.3 ネーミングサービス 780
D.4.3.1 ネーミングサービス／listメソッド 780
D.4.3.2 ネーミングサービス／bind, resolve, destroyメソッド 781
D.4.4 インタフェースリポジトリ 782
D.4.4.1 インタフェースリポジトリ 782
D.4.5 POAインタフェース 783
D.4.5.1 POAインタフェース（C++言語） 783
D.4.5.2 POAインタフェース（Java言語） 783
D.4.6 プロセスバインド 783
D.4.6.1 プロセスバインド 783
D.4.7 各種API 784
D.4.7.1 net_disconnect 784
D.4.7.2 set_client_request_timer 786
D.4.7.3 string_to_object 786
D.4.8 Portable-ORB 787
D.4.8.1 Portable-ORB／各種データ型 788
D.4.8.2 Portable-ORB／POAインタフェース 791
D.4.8.3 Portable-ORB／四則演算 791
D.4.8.4 Portable-ORB／各種API（net_disconnect） 793
D.4.8.5 Portable-ORB／各種API（set_client_request_timer） 795
D.4.8.6 Portable-ORB／各種API（string_to_object） 796
D.4.9 その他 797
D.4.9.1 その他（C言語） 797
D.4.9.2 その他（C++言語） 797
D.4.9.3 その他（Java言語） 798

付録E プロセスバインド機能を用いたセション管理の例 799
E.1 クライアントアプリケーションのプログラミング 799
E.1.1 初期化 799
E.1.2 ネーミングサービスのオブジェクトリファレンスの獲得 800
E.1.3 サーバアプリケーションのオブジェクトリファレンスの獲得 800

E.1.4 セション継続用オブジェクトリファレンスの獲得....800
E.1.5 メソッドの呼出し....800
E.1.6 セション破棄用のメソッドの呼出し....801
E.2 サーバアプリケーションのプログラミング....801
E.2.1 初期化....801
E.2.2 サーバの活性化....802
E.2.3 セション継続用のオブジェクトリファレンス生成....803
E.2.4 セション継続用のオブジェクトリファレンスのバインド関係登録....803
E.2.5 セション継続メソッドの実行....803
E.2.6 セション継続用のオブジェクトリファレンスのバインド関係登録削除....803

付録F 旧バージョンからの移行上の注意....805
F.1 Javaインタフェース(JDK/JRE 1.3以前からの移行)....805
F.2 ネーミングサービスに関する注意事項（V2以前のバージョンからの移行）....811

付録G 旧互換機能....815
G.1 COM/CORBA連携プログラミング....815
G.1.1 OLE-CORBAゲートウェイとは....815
G.1.2 OLEクライアントの処理....815
G.1.2.1 OLE-CORBAゲートウェイの起動....816
G.1.2.2 サーバアプリケーションのオブジェクトリファレンスの検索....816
G.1.2.3 メソッドの実行....816
G.1.3 サーバアプリケーションへのパラメタ受渡し....817
G.1.4 基本データ型の指定方法....817
G.1.4.1 文字列型....818
G.1.4.2 列挙型....819
G.1.4.3 any型....819
G.1.5 その他のデータ型の指定方法....820
G.1.5.1 シーケンス型....820
G.1.5.2 構造体....821
G.1.5.3 共用体....821
G.1.5.4 配列....822
G.1.6 サーバの処理結果....823
G.1.7 属性の設定・参照....823
G.1.8 例外....824
G.2 ロードバランス機能のプログラミング....826
G.2.1 ロードバランスの運用....826
G.2.2 ロードバランス機能インタフェース....827
G.2.3 ロードバランスオプションインタフェース....829
G.2.3.1 ロードバランスオブジェクトグループへのオブジェクトの登録....829
G.3 アプリケーションの開発(OOCOBOL)....836
G.3.1 クライアントアプリケーションのプログラミング（静的起動インタフェース）....837
G.3.1.1 初期化....837
G.3.1.2 ネーミングサービスのオブジェクトリファレンスの獲得....838
G.3.1.3 サーバアプリケーションのオブジェクトリファレンスの獲得....839
G.3.1.4 メソッドの呼び出し....839
G.3.2 クライアントアプリケーションのプログラミング（動的起動インタフェース）....840
G.3.2.1 初期化....841
G.3.2.2 ネーミングサービスのオブジェクトリファレンスの獲得....842
G.3.2.3 サーバアプリケーションのオブジェクトリファレンスの獲得....842
G.3.2.4 インタフェースリポジトリからサーバアプリケーション情報の獲得....843
G.3.2.5 パラメタの組立て....844
G.3.2.6 リクエストの作成....845
G.3.2.7 リクエストの送信....846
G.3.3 クライアントアプリケーションの例外処理....846
G.3.3.1 静的起動インタフェース例外....848
G.3.3.2 動的起動インタフェース例外....848
G.3.4 サーバアプリケーションのプログラミング（静的スケルトンインタフェース）....848

G.3.4.1 初期化....849
G.3.4.2 サーバの活性化....850
G.3.4.3 インタフェース実装クラス....851
G.3.4.4 サーバの非活性化....852
G.3.5 サーバアプリケーションの例外処理....853
G.3.6 サーバアプリケーションの登録....854
G.3.6.1 インプリメンテーションリポジトリへの登録....854
G.3.6.2 ネーミングサービスへの登録....856
G.3.7 アプリケーションのテスト....860
G.3.8 データ型に対するマッピング....861
G.3.8.1 文字列型....862
G.3.8.2 ワイド文字列型....865
G.3.8.3 シーケンス型....868
G.3.8.4 構造体....877
G.3.8.5 共用体....883
G.3.8.6 固定小数点型....891
G.3.8.7 属性宣言(attribute)のマッピング....895
G.3.8.8 パラメタ受渡しで使用するデータ型....901
G.3.9 OOCOBOLアプリケーション使用時の注意事項....902
G.3.10 OOCOBOL用登録集について....909
G.3.10.1 使用例....909
G.3.10.2 登録集原文....910
G.3.10.3 リポジトリファイル....910
G.4 動的スケルトンインタフェースのプログラミング....917
G.4.1 C言語の開発....917
G.4.1.1 初期化....917
G.4.1.2 ゲートウェイの登録....918
G.4.1.3 サーバの活性化....918
G.4.1.4 ゲートウェイの処理....918
G.4.1.5 サーバの非活性化....920
G.4.1.6 パラメタの獲得／解放....920
G.4.2 C++言語の開発....921
G.4.2.1 Basic Object Adapter : BOA....921
G.4.2.1.1 初期化....922
G.4.2.1.2 ゲートウェイの登録....922
G.4.2.1.3 サーバの活性化....922
G.4.2.1.4 ゲートウェイの処理....922
G.4.2.1.5 サーバの非活性化....925
G.4.2.2 Portable Object Adpter : POA....925
G.4.2.2.1 初期化....925
G.4.2.2.2 ゲートウェイの登録....925
G.4.2.2.3 サーバの活性化....925
G.4.2.2.4 ゲートウェイの処理....925
G.4.2.2.5 DSI使用時の注意点....928
G.4.2.3 パラメタの獲得／解放....929
G.4.3 Java言語の開発....930
G.4.3.1 初期化....930
G.4.3.2 ゲートウェイの登録....930
G.4.3.3 サーバの活性化....930
G.4.3.4 ゲートウェイの処理....930
G.4.4 COBOLの開発....933
G.4.4.1 初期化....933
G.4.4.2 サーバの活性化....934
G.4.4.3 ゲートウェイの処理....934
G.4.4.4 サーバの非活性化処理....940
G.4.5 OOCOBOLの開発....940
G.4.5.1 初期化....941

G.4.5.2 ゲートウェイの登録....942
G.4.5.3 サーバの活性化....942
G.4.5.4 ゲートウェイの処理....942

索引....947

第1部 アプリケーション開発(基本編)

第1章 CORBAアプリケーションの基礎知識.....2

第2章 CORBAアプリケーション開発時の留意事項.....14

第3章 アプリケーションの開発（C言語）.....33

第4章 アプリケーションの開発（C++言語）.....73

第5章 アプリケーションの開発（Java言語）.....240

第6章 アプリケーションの開発（COBOL）.....424

第7章 CORBAアプリケーションの基本的な機能.....544

第1章 CORBAアプリケーションの基礎知識

CORBAアプリケーションを作成するための基礎知識について説明します。

注意

本マニュアルは、マルチ言語サービスのCORBAサービスのアプリケーションを作成するためのマニュアルです。Java EEのアプリケーションを作成する場合は、“Java EE運用ガイド”を参照してください。

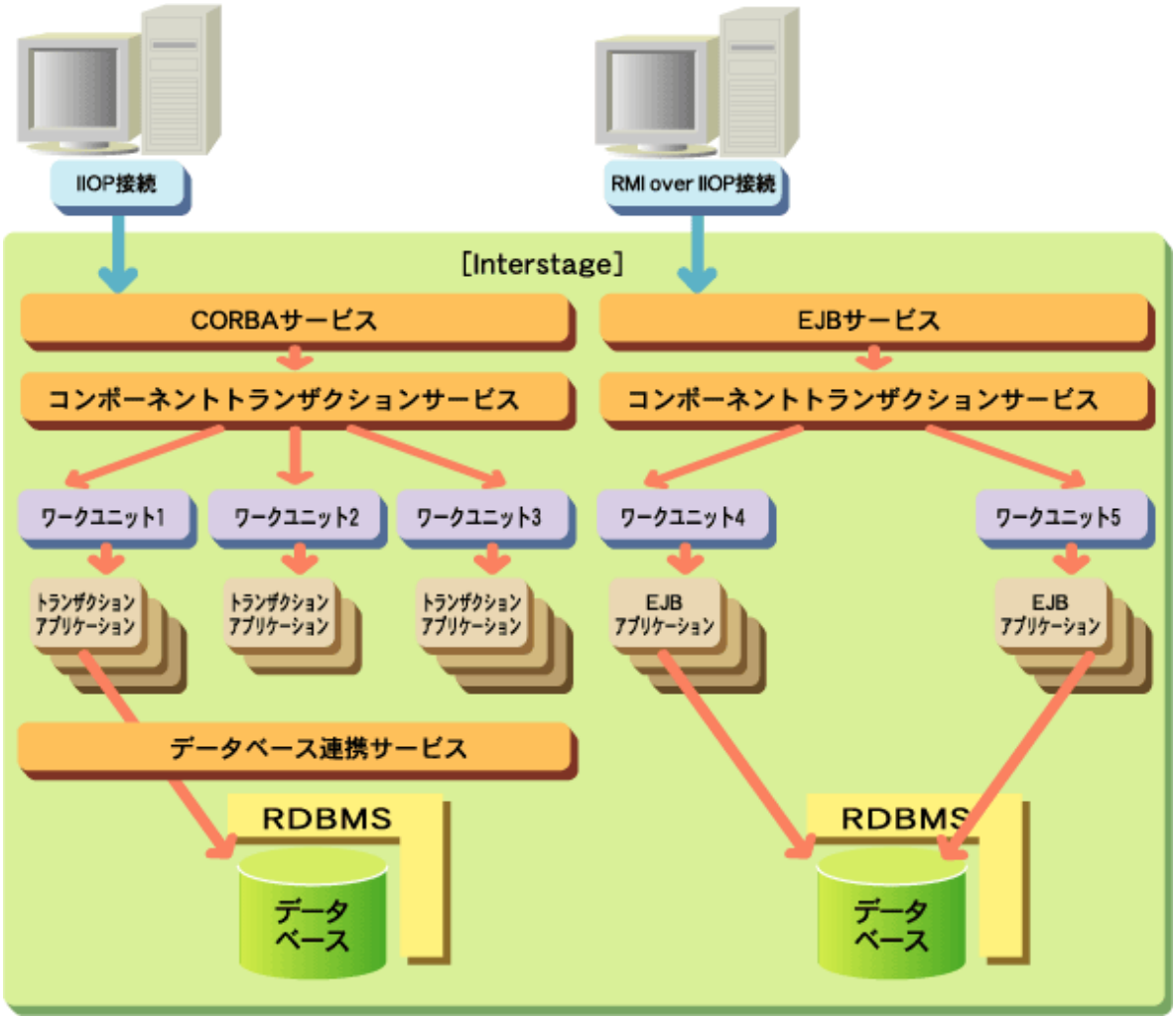
1.1 CORBAアプリケーションの基本モデル

Interstageシステムを設計するための基本モデルとして、以下のようなモデルがあります。

- 2階層モデル
- 3階層モデル

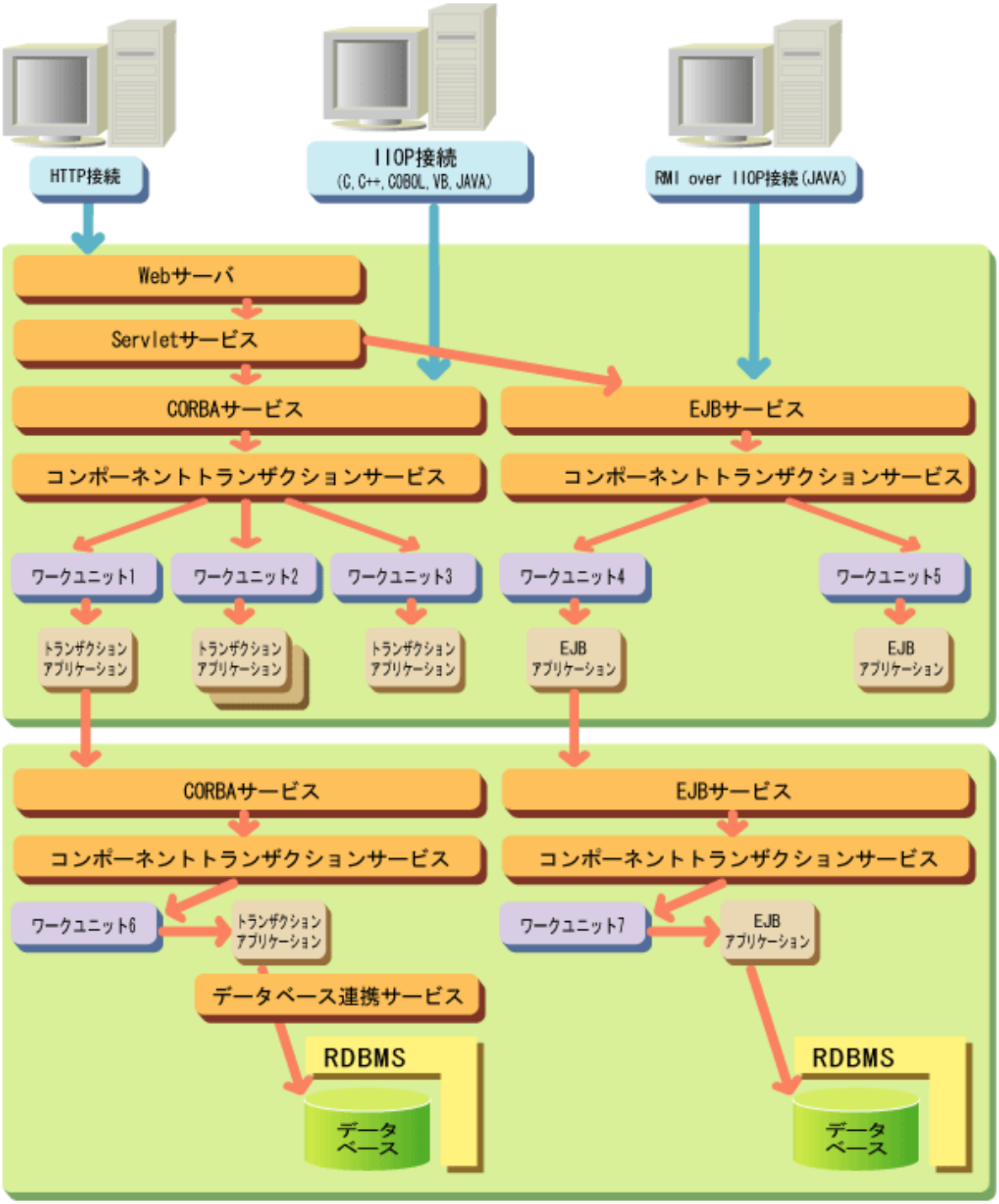
1.1.1 2階層モデル

2階層モデルは、クライアント／サーバ間で処理を行うモデルであり、クライアント／サーバ間の通信は、ORB(Object Request Broker)によって仲介されています。
以下に2階層モデルの概要図を示します。



1.1.2 3階層モデル

3階層モデルは、クライアント／サーバシステムのサーバを、アプリケーションサーバとデータベースサーバのように処理単位を分けて構築したシステムモデル（2階層モデルのサーバを分けた拡張モデル）です。
以下に3階層モデルの概要図を示します。



1.2 CORBAサービスの運用形態

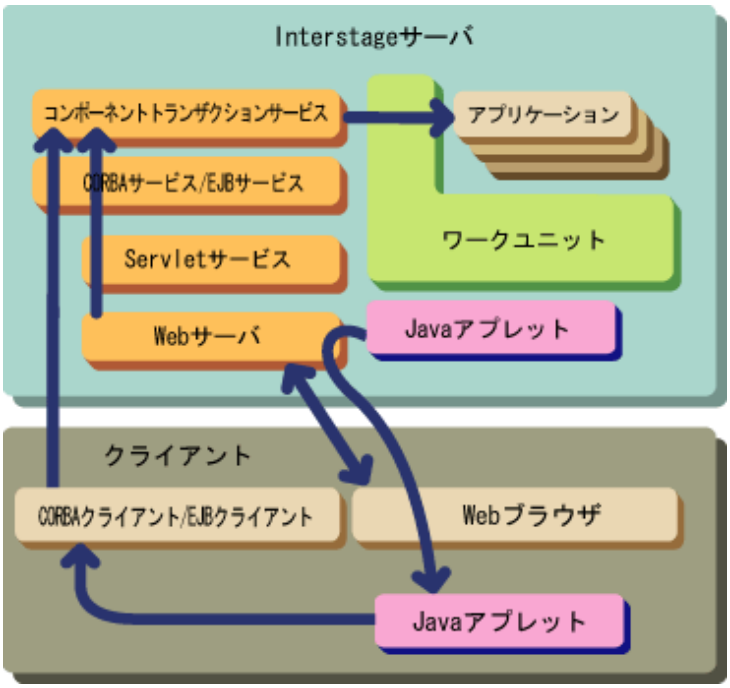
CORBAアプリケーションの運用形態について説明します。

1.2.1 クライアントの運用形態

クライアントの運用には、以下の2つの形態があります。Java言語では、プレインストール型／Portable-ORBを選択して運用できます。

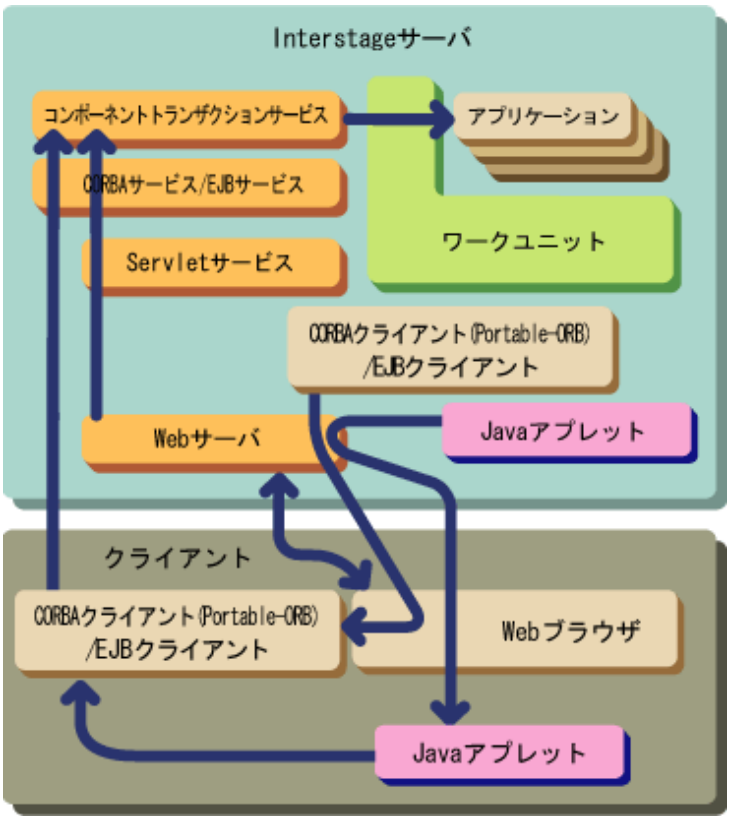
プレインストール型（すべての開発言語で使用可能）

あらかじめクライアントマシンにクライアントランタイム（CORBAサービスクライアント）をインストールして、運用する形態です。プレインストール型は、すでにクライアントマシンにインストールされているため、実行時にクライアントランタイムをダウンロードするための時間はかかりません。



Portable-ORB（Java言語のみ使用可能）

あらかじめサーバマシンにクライアントランタイム（Portable-ORB）をインストールして、実行時にサーバマシン（Webサーバ）からアプレットとともにクライアントランタイム（Portable-ORB）をダウンロードして、運用する形態です。Portable-ORBは、個々のクライアント端末に事前にJavaランタイムをインストールする必要がないため、クライアントの運用・保守コストを削減できます。



1.2.2 サーバの運用形態

サーバの運用では、複数サーバで運用できます。複数サーバで運用する場合は、サーバオブジェクトの配置とネーミングサービスの配置を検討する必要があります。

サーバオブジェクトの配置

複数サーバで運用する場合は、サーバオブジェクトを複数のサーバマシン上に分散して配置します。この運用は、負荷の高いオブジェクト同士を同一サーバマシン上に配置すると、負荷の集中によりサーバマシンの性能に影響を及ぼす可能性があります。したがって、設計時には複数のサーバマシンに均等に負荷が分散されるようにオブジェクトの配置を考慮する必要があります。
同じ機能を提供するオブジェクトを異なる名前のオブジェクトリファレンスで登録することにより、特定の機能の負荷を分散することが可能になります。“11.7 サーバアプリケーションの複数ホストへの配置”を参照してください。
また、同じサーバオブジェクト(インタフェースが同一で、同じ機能を提供するオブジェクト)を複数のサーバ上に配置することにより、以下の機能を使用することが可能となります。

IPCOM連携機能

大規模の場合はIPCOM連携機能を使用することを推奨します。IPCOM連携機能については、“高信頼性システム運用ガイド”の“IPCOMを利用した負荷分散”を参照してください。

ロードバランス機能 Windows32 Solaris32 Linux32

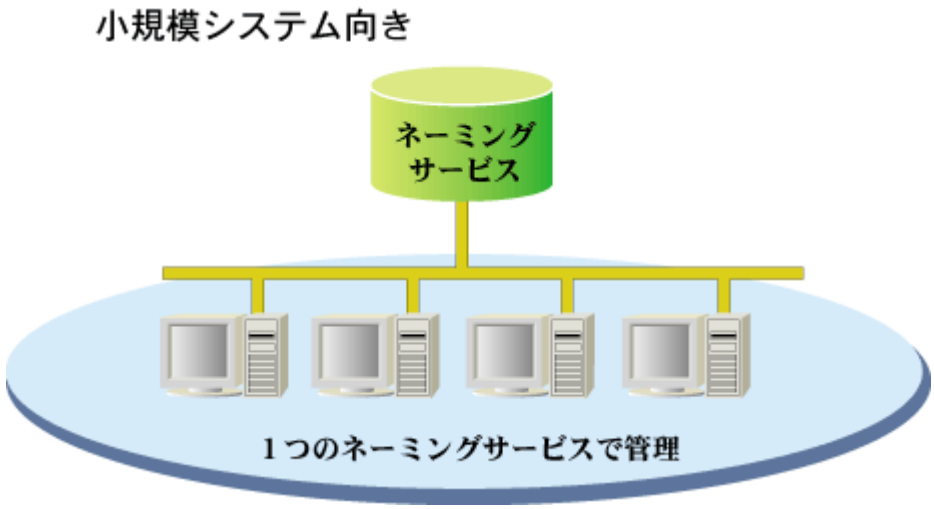
ロードバランス機能については、“高信頼性システム運用ガイド”の“ロードバランス機能を利用した場合の設計”を参照してください。

ネーミングサービスの配置

ネーミングサービスを複数のサーバマシンに分散して配置します。この運用は、負荷分散や配置による物理的なトラフィックやレスポンスを考慮して、以下の3つの配置方式からシステム規模に合ったものを選択してください。

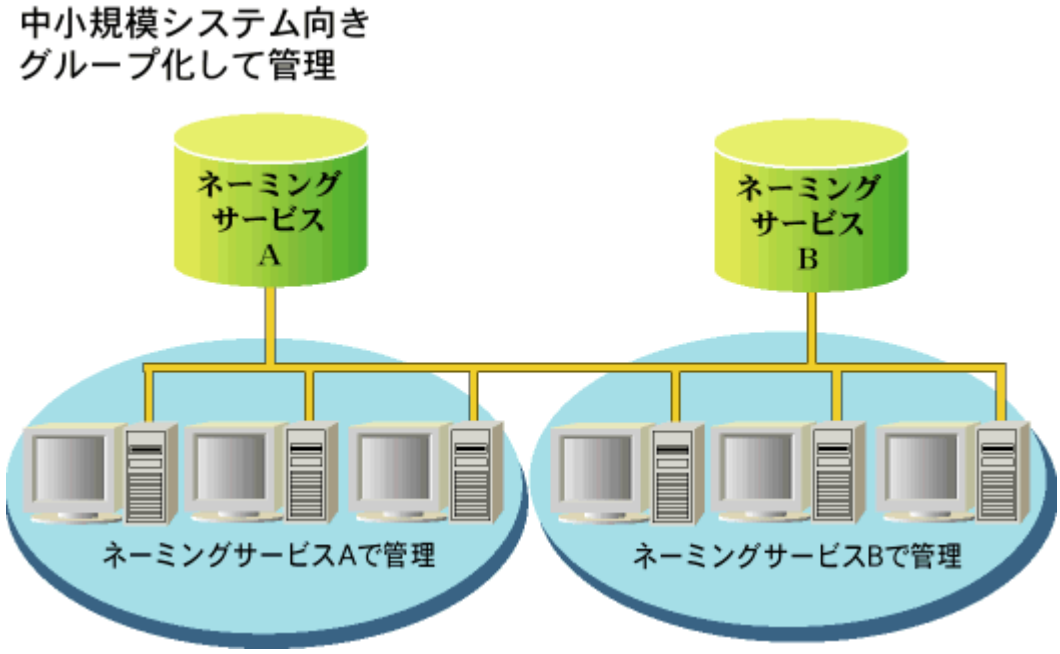
集中型

システム全体のオブジェクトを1つのネーミングサービスで管理します。この方式は、1つのネーミングサービスに負荷が集中するため、小規模システムに向いています。



分散型

システム全体のオブジェクトを複数のネーミングサービスで管理します。また、各ネーミングサービスでは、オブジェクトをグループ化して管理します。この方式は、特定のオブジェクトへのアクセスの負荷が1つのネーミングサービスに集中するため、中小規模システムに向いています。

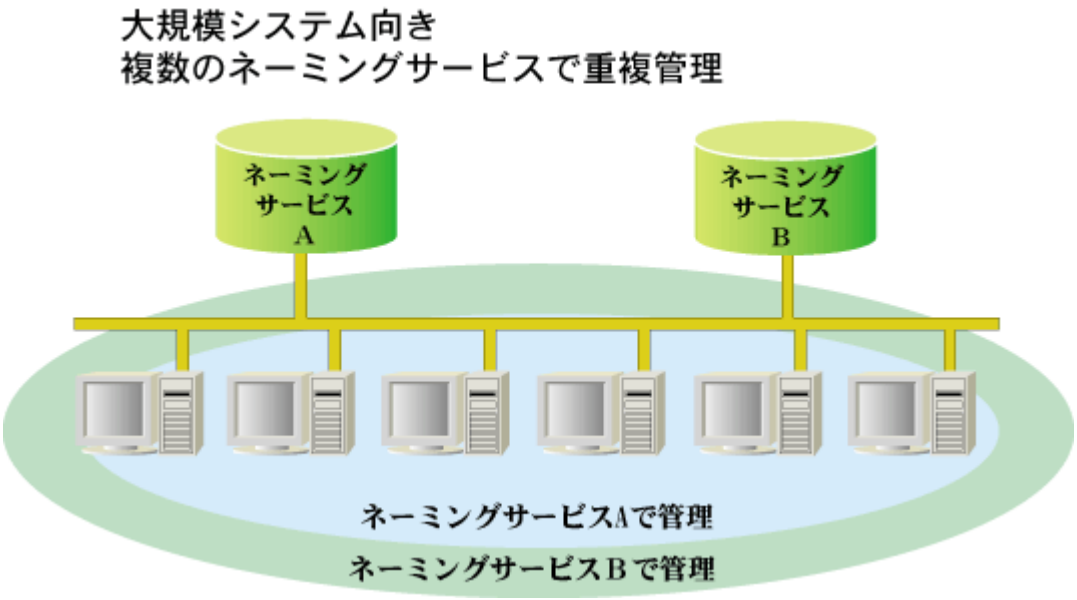


レプリケーション型

システム全体のオブジェクトを管理するネーミングサービスが複数存在します。また、各ネーミングサービスでは、すべて同じオブジェクトを管理します。この方式は、システム全体を管理するネーミングサービスを複数持つことができるため、クライアント数の多い大規模システムに向いています。

この配置方式ではIPCOM連携機能を使用することが可能となります。IPCOM連携機能については、“高信頼性システム運用ガイド”の“IPCOMを利用した負荷分散”を参照してください。

IPCOM連携機能を使用せず、クライアントアプリケーションが使用するネーミングサービスを選択することも可能です。詳細については、“第13章 ネーミングサービスのイニシャルリファレンスの獲得”を参照してください。



1.3 CORBAアプリケーションの設計

CORBAアプリケーションを設計する際に考慮すべき内容について説明します。

1.3.1 アプリケーションの開発言語

アプリケーションを開発するには、以下の言語を使用して作成できます。

プログラム

- C言語
- C++言語
- Java言語
- COBOL
- OOCOBOL

注意

Windows64 Solaris64 Linux64

OOCOBOLは、Windows(R)クライアントアプリケーションで使用できます。

1.3.2 サーバアプリケーションの起動タイプ

サーバアプリケーションのオブジェクト処理は、サーバアプリケーションの起動タイプを設定することで、以下の3つから選択できます。作成するシステム設計にしたがって、インプリメンテーションリポジトリへの登録時に設定します。

shared

1つのサーバアプリケーションが同時に複数のオブジェクトを処理します。サーバオブジェクトは、複数のクライアントからの要求を処理します。この起動タイプを指定すると、クライアントから要求があった時点でサーバアプリケーションは自動起動します。

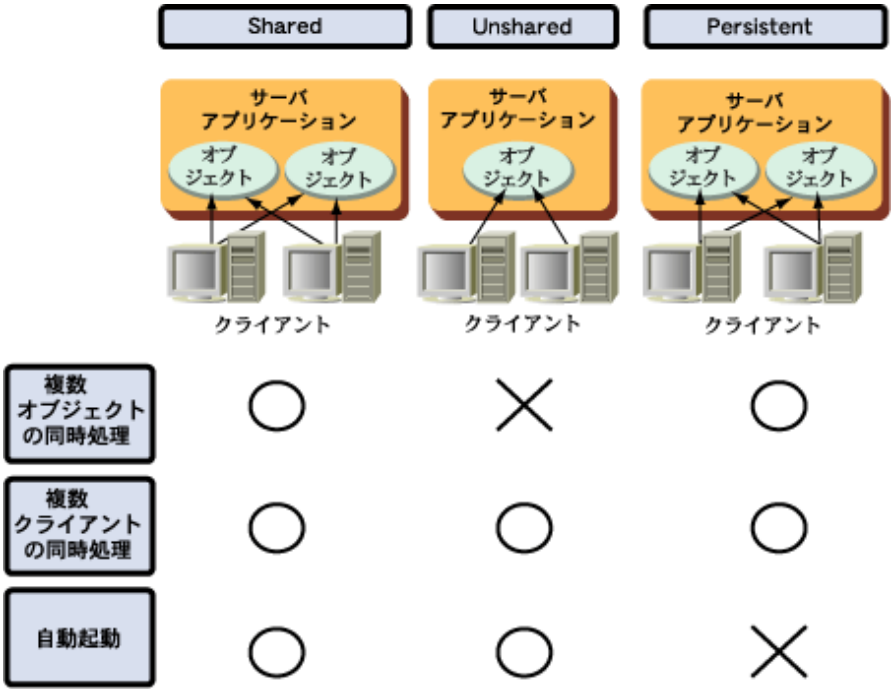
unshared

1つのサーバアプリケーションが1つのオブジェクトだけを処理します。サーバオブジェクトは、複数のクライアントからの要求を処理します。この起動タイプを指定すると、クライアントから要求があった時点でサーバアプリケーションは自動起動します。

persistent

1つのサーバアプリケーションが同時に複数のオブジェクトを処理します。サーバオブジェクトは、複数のクライアントからの要求を処理します。この起動タイプを指定すると、クライアントから要求がきてもサーバアプリケーションは自動起動しません。クライアントからの要求を処理するには、事前にサーバアプリケーションを起動する必要があります。

サーバアプリケーションのオブジェクトを共用する場合は、「Shared」や「Persistent」を指定します。サーバアプリケーションのオブジェクトを排他的に利用する場合は、「Unshared」を指定します。なお、Persistent以外の起動タイプの場合、サーバアプリケーションが起動されていない場合でも、要求があった時点でサーバアプリケーションは自動的に起動します。
以下を参照し、利用目的に応じて選択してください。

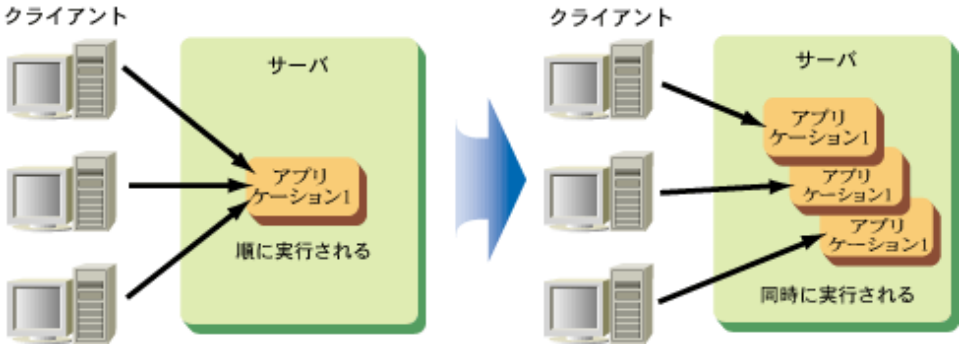


注意

特別な要件がない限り、persistentサーバを使用することを推奨します。

1.3.3 多重制御

多重制御により、アプリケーションは複数クライアントからの要求を処理できます。多重制御の運用イメージを以下に示します。



CORBAアプリケーションの場合は、インプリメンテーションリポジトリを利用して、多重制御を行うことができます。
また、同一アプリケーションの多重制御には、プロセスを多重化するプロセス多重とスレッドによるスレッド多重があり、プロセス最大多重度およびスレッド初期多重度を設定することにより複数のクライアントからの要求を処理できます。

注意

サーバアプリケーションの起動タイプが“shared”／“unshared”の場合、プロセス最大多重度を設定しても複数のプロセスが起動することはありません。

1.3.4 インスタンス管理システム

クライアントアプリケーションごとのオブジェクトの作業域を管理できます。サーバアプリケーションでFactoryインタフェースを実装して、各クライアントアプリケーションに対応するオブジェクトを動的に生成します。なお、本機能は、以下の言語でサポートされています。

- C++言語 Windows32 Solaris32 Linux32
  詳細については、“11.1 Factory”を参照してください。
- Java言語
  詳細については、“5.12 インスタンス管理とアプリケーション形態”を参照してください。

1.3.5 クライアントアプリケーションのインタフェース

クライアントアプリケーションは、サーバアプリケーションの呼出し方法により、大きく以下の2つのインタフェースに分けることができます。なお、サーバ側のインタフェースと異なる組合せになっても、問題ありません。

静的起動インタフェース

クライアントアプリケーションは、スタブからインタフェース情報を取得します。サーバオブジェクトはコンパイル時に認識されるため、簡易に作成することが可能であり、性能面に優れています。

動的起動インタフェース

クライアントアプリケーションは、インタフェースリポジトリから動的にインタフェース情報を取得します。クライアントアプリケーションは、連携時にサーバオブジェクトを認識するため、最新のサーバオブジェクトの情報を受け取ることが可能です。また、サーバ側では、クライアントへの影響を考慮することなくオペレーションの追加／変更／削除を行うことが可能であるため、柔軟性のあるシステムを実現できます。

注意

IDL定義ファイルの構成によっては、インタフェース独自の領域獲得変数などがスタブに定義されることがあります。この場合、コンパイル時にアプリケーションとスタブをリンクする必要があります。

1.3.6 サーバアプリケーションのインタフェース

サーバアプリケーションは、クライアントに対するインタフェース情報の公開方法により、大きく以下の2つのインタフェースに分けることができます。なお、クライアント側のインタフェースと異なる組合せになっても、問題ありません。

静的スケルトンインタフェース

スケルトンによりインタフェースを公開します。IDL定義ファイルから生成されたスケルトンとアプリケーションをコンパイル時にリンクすることにより、サーバオブジェクトとオペレーションを固定化するため、簡易に作成することが可能であり、性能面に優れています。

動的スケルトンインタフェース

DSI(Dynamic Skelton Interface)関数によりインタフェースを公開します。スケルトンを使用せず、サーバアプリケーション起動時に、DSI関数がORBに通知されます。クライアントからの要求に対し、どのサーバオブジェクトに送信された要求であるかを動的に判断することができるため、柔軟性のあるシステムを実現できます。

注意

IDL定義ファイルの構成によっては、インタフェース独自の領域獲得変数などがスケルトンに定義されることがあります。この場合、コンパイル時にアプリケーションとスケルトンをリンクする必要があります。

1.4 CORBAアプリケーションの開発手順

CORBAアプリケーションの開発手順を以下に示します。

開発手順の詳細については、開発言語により以下を参照してください。

開発言語	参照先
C	第3章 アプリケーションの開発（C言語）
C++	第4章 アプリケーションの開発（C++言語）
Java	第5章 アプリケーションの開発（Java言語）
COBOL	第6章 アプリケーションの開発（COBOL）
OOCOBOL	G.3 アプリケーションの開発(OOCOBOL)

注意

Windows64 Solaris64 Linux64

OOCOBOLは、Windows(R)クライアントアプリケーションで使用できます。

1.5 CORBAアプリケーションの運用手順

CORBAアプリケーションをワークユニットで運用させる場合は、“7.2 CORBAワークユニット”を参照してください。ワークユニットでないCORBAアプリケーションを運用させる場合の運用手順を以下に示します。

(1)CORBAアプリケーションの作成

CORBAアプリケーションの作成については、“1.4 CORBAアプリケーションの開発手順”を参照してください。

(2)サーバアプリケーション情報のインプリメンテーションリポジトリへの登録

OD_impl_instコマンドを使用してインプリメンテーションリポジトリにサーバアプリケーション情報を登録します。OD_impl_instコマンドの詳細については、“リファレンスマニュアル(コマンド編)”を参照してください。

(3)CORBAアプリケーションの起動

業務運用を開始させるため、CORBAアプリケーションを起動します。

サーバアプリケーションの起動

インプリメンテーションリポジトリに登録されているサーバアプリケーションの起動タイプが「persistent」の場合は、作成したアプリケーションをコマンドラインやスクリプト/バッチなどにより起動してください。「persistent」以外の場合は、クライアントから要求があった時点で自動的に起動するため、ユーザのオペレーションは必要ありません。

クライアントアプリケーションの起動

作成したアプリケーションをコマンドラインやスクリプト/バッチなどにより起動してください。

(4)CORBAアプリケーションの停止

業務運用を終了させるため、起動中のCORBAアプリケーションを停止します。

サーバアプリケーションの停止

サーバアプリケーションを停止する場合は、以下の手順で停止してください。odcntlqueコマンドの詳細については、“リファレンスマニュアル(コマンド編)”を参照してください。

```
odcntlque -d -que Impl-ID (サーバアプリケーションのインプリメンテーションリポジトリID)
odcntlque -c Impl-ID
odcntlque -s Impl-ID
```

ポイント

上記を実行することで、サーバアプリケーション活性化時の動作モード(OD_impl_instコマンドの“mode”(定義ファイル)の指定)により、アプリケーションの状態は以下のように遷移します。

“mode=SYNC_END”の場合

サーバアプリケーション活性化メソッド(CORBA_BOA_impl_is_ready()など)が復帰し、任意の終了処理が実行可能です。プロセスを終了する場合は、exit関数を発行してください。

“mode=COMPATIBLE”の場合

サーバアプリケーションは、終了します。

注意

Windows32/64

以下の条件のとき、活性化メソッドが復帰せずにサーバアプリケーションが終了します。

- サーバアプリケーションをプロセスモードで作成している場合 かつ
- サーバアプリケーションをCORBAワークユニット運用している場合 かつ
- CORBAワークユニットを停止した場合

活性化メソッドの後に後処理を行う場合は、サーバアプリケーションをスレッドモードで作成するか、またはワークユニットの出口機能を利用してください。ワークユニットの出口機能に関しては、“OLTPサーバ運用ガイド”の“ワークユニットの出口機能”を参照してください。

クライアントアプリケーションの停止

クライアントアプリケーションの停止処理を行う場合は、exit関数を発行してください。

第2章 CORBAアプリケーション開発時の留意事項

ここでは、CORBAアプリケーションを開発する際に留意する事項について説明します。

注意

本マニュアルは、マルチ言語サービスのCORBAサービスのアプリケーションを作成するためのマニュアルです。Java EEのアプリケーションを作成する場合は、“Java EE運用ガイド”を参照してください。

2.1 コーディング

注意

Windows64 Solaris64 Linux64

OOCOBOLは、Windows(R)クライアントアプリケーションで使用できます。

2.1.1 データ領域の獲得/解放処理（ C, C++, COBOL ）

以下のデータ型を使用する場合は、各データ型に応じた専用の関数により領域を獲得する必要があります。専用の関数で獲得していない場合は、アプリケーション動作時にメモリアクセスエラーを引き起こす要因となります。また、クライアントおよびスタブ内で動的に獲得した領域は、不要になった時点で各データ型に応じた開放関数および方法により明示的な開放操作が基本的に必要です。開放漏れの場合、メモリリークの要因となります。

- 文字列型
- ワイド文字列型
- any型
- シーケンス型
- 構造体
- 共用体
- 配列

any型およびシーケンス型に関してデータ域も合わせて解放する場合は、専用関数により事前にリリースフラグを設定する必要があります。明示的に設定しない場合、デフォルトは解放しない設定となっているため、メモリリークの要因となります。リリースフラグの詳細については、各開発言語の“any型、sequence型のリリースフラグ”を参照してください。

以下のデータ型を使用する場合は、クライアントアプリケーションのinパラメタ／inoutパラメタ、サーバアプリケーションのoutパラメタ／inoutパラメタ、復帰値として、NULLポインタを設定できません。誤って設定した場合、メモリアクセスエラーを招く要因となります。

- 文字列型
- シーケンス型
- 構造体
- 共用体
- 配列

COBOLアプリケーションにおいて以下の型のデータ設定・取得操作は、必ず専用APIを使用してください。各APIの詳細については、“リファレンスマニュアル(API編)”の該当する記事を参照してください。

データ型	データ設定API	データ取得API
文字列型	CORBA-STRING-SET	CORBA-STRING-GET
ワイド文字列型	CORBA-WSTRING-SET	CORBA-WSTRING-GET
シーケンス型	CORBA-SEQUENCE-ELEMENT-SET	CORBA-SEQUENCE-ELEMENT-GET

参照

- “アプリケーションの開発(C言語)”の“3.7 データ型に対するマッピング”、“3.8 any型、sequence型のリリースフラグ”
- “アプリケーションの開発(C++言語)”の“4.6 データ型に対するマッピング”、“4.6.12 any型、sequence型のリリースフラグ”
- “アプリケーションの開発(COBOL)”の“6.8 データ型に対するマッピング”、“6.9 any型、sequence型のリリースフラグ”

2.1.2 シグナル処理(C, C++, Java, COBOL)

Solaris32/64

C言語のCORBAアプリケーションでは、SIGCHLDシグナルを受信できます。その他の言語では、シグナルを受信するアプリケーションを作成できません。

シグナルハンドラで使用できる関数は、「非同期シグナル安全」(Async-Signal-Safe)関数だけです。Async-Signal-Safeの関数は、オンラインマニュアルattributes(5)で確認できます。printf、malloc、freeなどの標準Cライブラリ関数やCORBAのAPIを呼び出すことは許されていないため、注意してください。

Linux32/64

シグナルを受信するアプリケーションを作成できません。

いくつかのOS関数は、シグナルを受信するとエラー情報にEINTRを設定して失敗します。CORBAのライブラリでは、OS関数がEINTRで失敗した場合もリトライ処理などを対応していますが、LinuxではOS自体の修正が行われた場合に、シグナル受信時のOS関数の動作が変わる可能性があります。この場合、シグナル受信時にCORBAライブラリの制御が正常に行えなくなる可能性があります。

2.1.3 子プロセス/スレッドの生成終了処理(C, C++, Java, COBOL)

CORBAワークユニット／IJServerのアプリケーションから生成される子プロセスでは、CORBA機能を使用しないことを推奨します。
CORBAワークユニット／IJServerのアプリケーションから生成される子プロセスにおいてCORBA機能を使用する必要がある場合は、子プロセスに環境変数が引き継がれないようにしてください。CORBAワークユニット／IJServerのアプリケーションにはInterstageによって設定されている環境変数があり、環境変数が引き継がれると子プロセスのCORBA機能が正しく動作しない可能性があります。

Solaris32/64 Linux32/64

CORBAアプリケーションより生成し、exec系の関数でプロセスイメージを変更していない子プロセスを終了する際には、exit()ではなく、_exit() を使用してください。exit() を使用した場合、子プロセスが終了した際に親プロセスも終了したと誤認されて、その後の動作は保証されません。
また、他言語(COBOLなど)のランタイムを呼び出す場合、その言語の機能／APIでプロセスを終了させると、ランタイム内でexit()が使用される可能性があります。exec系の関数でプロセスイメージを変更していない子プロセスでは、他言語の機能／APIでプロセスを終了させないでください。
なお、exec系の関数でプロセスイメージを変更している場合は、プロセスを終了する際にexit()が使用可能です。

CORBAアプリケーションより生成した子プロセスでCORBA機能を使用する場合は、exec系の関数でプロセスイメージを変更してから、CORBA_ORB_init関数(C++ではCORBA::ORB::init関数)を発行して初期化処理を行ってください。

Solaris32/64

プロセスモードのサーバアプリケーションから子プロセスを生成する場合は、fork()を使用してください。スレッドモードのサーバアプリケーションの場合は、fork()ではなくfork1()を使用してください。

スレッドモードのサーバアプリケーション内から、さらにスレッドを生成する場合は、thr_create() の引数としてTHR_NEW_LWPおよびTHR_BOUNDフラグを指定する必要があります。

Windows32/64

CORBA_ORB_init関数(C++ではCORBA::ORB::init関数)発行後に子プロセスを生成する場合は、_spawn系関数(_spawn関数、_spawnl関数など)は使用しないでください。_spawn系関数はハンドルなどを親プロセスから引き継ぐ仕様となっているため、CORBAサービスの制御が正常に行えなくなります。
CORBA_ORB_init関数(C++ではCORBA::ORB::init関数)発行後にプロセスを生成する場合は、CreateProcess関数を使用してください。

Solaris32/64 Linux32/64

動作モードがSYNC_ENDのサーバアプリケーションの場合、CORBA_BOA_impl_is_ready関数(C++ではCORBA::BOA::impl_is_ready関数)復帰後に、リクエスト処理スレッドがリクエストを処理することはありませんが、OSレベルでスレッドが終了していることは保証しません。そのため、dlopen関数で動的に組み込んだライブラリ上の関数をpthread_key_create関数などでスレッド終了時のデストラクタ関数に登録して、CORBA_BOA_impl_is_ready関数復帰後に動的に組み込んだライブラリをdlclose関数で解放した場合、解放済みのライブラリの関数をリクエスト処理スレッドが終了時に発行しようとしてプロセスが異常終了する可能性があります。デストラクタ関数の存在するライブラリは、dlclose関数で解放しないでください。

Linux32/64

初期スレッドでpthread_exit()を発行して初期スレッドを終了させた場合、Linuxの仕様によりpsコマンドでプロセスの情報を確認すると<defunct>の表示が現れ、gcoreコマンドやstraceコマンドなどでプロセスの情報を採取することができなくなります。
gcoreコマンドやstraceコマンドなどでプロセスの情報を採取できないと、トラブルが発生した場合に調査が難しくなるため、アプリケーションを開発する場合は、初期スレッドを終了させないロジックにすることを推奨します。
たとえば、インプリメンテーションの定義で動作モードに“COMPATIBLE”を設定したサーバアプリケーションの場合、サーバアプリケーションの活性化後は初期スレッドが不要となることが一般的です。この場合、動作モードを“SYNC_END”に設定して活性化関数復帰後にプロセスを終了させるように修正するか、初期スレッドでpthread_exit()を発行せずにsleep()を無限ループするように修正します。

例

修正前のソース(動作モード:COMPATIBLE)

```
int main( int argc, char *argv[] )
{
    /* ORBの初期化処理など */
    CORBA_BOA_impl_is_ready(...); /* 活性化関数(引数は省略) */

    pthread_exit(NULL);
}
```

動作モードをSYNC_ENDに設定した場合の修正後のソース

```
int main( int argc, char *argv[] )
{
    /* ORBの初期化処理など */
    CORBA_BOA_impl_is_ready(...); /* 活性化関数(引数は省略) */

    return 0;
}
```

動作モードをCOMPATIBLEから変更しない場合の修正後のソース

```
int main( int argc, char *argv[] )
{
    /* ORBの初期化処理など */
    CORBA_BOA_impl_is_ready(...); /* 活性化関数(引数は省略) */

    while ( 1 ){
        sleep( 1000 );
```

```
    }
}
```

2.1.4 ORBの初期化処理(C, C++, COBOL)

ORBの初期化関数(C言語の場合は、CORBA_ORB_init)に渡す引数には、main関数に渡されるargc／argvをそのまま渡すようにしてください。

Windows32/64

アプリケーション	初期化方法
コンソールアプリケーション	main()関数の引数argc、argvを、CORBA_ORB_init()に渡します。詳細については、サンプルプログラム(C言語の場合はsimple_sのソースsimple_s.c)を参照してください。
Windows(R)アプリケーション	WinMain()関数の第3引数から、CORBA_ORB_init()に渡す形式にデータを変換します。詳細は、サンプルプログラムmsg_sのソースmsg_s.cを参照してください。データを変換するルーチンget_arg()は、arg.cに記述されています。

Solaris32/64 Linux32/64

アプリケーション	初期化方法
システムのアプリケーション	main()関数の引数argc、argvを、CORBA_ORB_init()に渡します。詳細については、サンプルプログラム(C言語の場合はsimple_sのソースsimple_s.c)を参照してください。
Windows(R)アプリケーション	WinMain()関数の第3引数から、CORBA_ORB_init()に渡す形式にデータを変換します。

Windows(R)アプリケーションでは、EXE部分以外でCORBA_ORB_init()を呼び出す場合、またはVisual Basicから呼び出されるDLLでCORBA_ORB_init()を呼び出す場合は、ODWIN.DLLをあらかじめEXE部分／Visual Basicの処理内でロード処理を行い、ODWIN.DLLのロードカウンタに1を加算する必要があります。また、アプリケーションの終了時には、1を加算したロードカウンタを保持している必要があります。

Windows(R)アプリケーションでは、DllMain関数内でCORBA_ORB_init()およびその他のCORBAの関数を発行してはいけません。

ORBの初期化関数は、CORBAの機能を使用するプロセスで1回だけ呼び出すようにしてください。

クライアントアプリケーションにおいてサーバメソッドの呼出しは、ORBの初期化関数が正常終了した後に行ってください。ORBの初期化処理が完了する前にサーバメソッドなどを呼び出した場合、“UNKNOWN”例外が通知されます。

2.1.5 サーバアプリケーションの例外処理(C, C++, COBOL)

サーバメソッドの引数に指定したCORBA_Environment構造体(C言語の場合)は、そのサーバメソッド内から呼び出す他のメソッドの引数として使用しないでください。そのまま使用した場合、該当サーバメソッドの例外情報に、他のメソッドで発生した例外情報が設定され、本来の呼出元の例外情報が得られないことになります。サーバメソッド内から他のメソッドを呼び出す際は、別のCORBA_Environment構造体を使用してください。

2.1.6 サーバ・クライアント機能の同時実装

アプリケーションにサーバ機能とクライアント機能を同時に実装する場合、サーバ機能のIDLファイルから生成されるスケルトンとクライアント機能のIDLファイルから生成されるスタブとで同一の関数名が生成されることがあります。これらを結合すると二重定義となるため、IDLファイルの作成には注意する必要があります。

2.1.7 oneway機能

IDL定義のオペレーション宣言でonewayを指定した場合、サーバは呼び出されたメソッドの実行結果をクライアントに通知しません。onewayを指定したメソッド呼び出す場合は、以下の点に注意してください。

- リクエストがサーバアプリケーションで処理される保証はありません。
  サーバ側で資源不足が発生した場合、またはネットワーク異常が発生した場合、リクエストはサーバアプリケーションで処理されません。このとき、クライアントは異常を検出できません。
- ネットワーク異常などの要因によりonewayメソッドに例外が通知される場合があります。
  onewayメソッドでは受信処理は行わないため、サーバで発生した異常を検出できませんが、リクエストの送信処理で異常を検出した場合は、onewayメソッドにも例外が通知されます。
- サーバの資源を大量に消費してしまう可能性があります。
  onewayメソッドではサーバ側の処理完了を待ち合わせないため、連続して発行した場合、繰り返し連続で発行した場合、サーバの資源を大量に消費してしまう可能性があるため、注意してください。

2.1.8 その他

- アプリケーション状態が無応答状態となった過去事例におけるユーザ論理ミスの要因として、次のようなものが挙げられます。
  - アプリケーションの処理内でsleep()により停止していた。
  - メッセージキュー(IPC資源)枯渇により、OS自体がスローダウンしていた。
- サーブレット／JSPで指定する日本語コード系が統一されているかを確認してください。
- CORBAのライブラリ(Windows(R)の場合：odsv.dll/odwin.dll、Solaris/Linuxの場合：libOM.so)をdlopen関数(Windows(R)の場合：LoadLibrary関数)プログラムに動的に組み込んでいる場合は、dlclose関数(Windows(R)の場合：FreeLibrary関数)で解放しないでください。
- CORBAのJava言語インタフェースを使用している場合、Javaのfinalizeメソッドではcorba通信を行わないでください。Javaのfinalizeメソッドでcorba通信を行っている場合、COMM_FAILURE例外が発生する可能性があります。
- プレインストール型JavaおよびJavaサーバでは、リクエストの送受信時にCORBAサービスが獲得するCヒープ領域は、FullGC処理により解放されます。そのため、Javaヒープに大きいサイズを指定している場合、FullGC処理を行う間隔が長くなり、Cヒープのサイズ不足によるメモリ不足が発生する可能性があります。
  このようなメモリ不足の発生を防止するため、CORBA-Javaアプリケーションにおいて以下のタイミングでSystem.gc()を呼び出し、FullGC処理を動作させてください。
  - サーバからのリプライ受信後
  - 一定時間間隔ごと
- Solaris32/64
  CORBAアプリケーションでは、CORBA通信のためにファイルディスクリプタを使用します。しかし、fopen関数などの標準Cライブラリstdio(3C)の関数では、ファイルディスクリプタが255までしか使用できません。CORBAアプリケーションにおいて、標準Cライブラリstdio(3C)の関数は使用しないことを推奨します。
- Solaris32/64 Linux32/64
  CORBAサービスでは以下の変数名を予約語としています。CORBAアプリケーションでは使用しないでください。
  - "OM_"で始まる変数名
  - "OD_"で始まる変数名
  - "ISP_"で始まる変数名
  - "g_"で始まる変数名

2.2 コンパイル・リンク

2.2.1 スレッドモードとプロセスモード

CORBAサービスのサポートするサーバアプリケーションのモードには、プロセスモードとスレッドモードがあります。

注意

Solaris32/64 Linux32/64

環境変数LD_LIBRARY_PATHにプロセスモード用の設定をしている状態で、Interstage Application Serverのコマンド(isstartなど)を実行しないでください。

スレッドモード

アプリケーションプロセスがマルチスレッドで動作する形態です。通常、本タイプを選択します。アプリケーションから呼び出す併用製品のライブラリがマルチスレッド対応している場合は、必ず本タイプを選択してください。

サーバアプリケーションでスレッドモードを使用する際の設定方法を以下に示します。

Windows32/64

インプリメンテーションリポジトリに登録したサーバアプリケーションの定義情報によって設定します。インプリメンテーションリポジトリへの定義情報の登録には、OD_impl_instコマンドを使用します。OD_impl_instコマンドの詳細については、“リファレンスマニュアル(コマンド編)”を参照してください。

定義情報では、thr_conc_initに2以上の値を設定します。OD_impl_instコマンドで指定する定義ファイルの例を以下に示します。

```
rep_id           = IDL:test1/intf1:1.0
type             = persistent
proc_conc_max    = 1
thr_conc_init    = 16
thr_conc_maximum = 32
```

Solaris32/64 Linux32/64

サーバアプリケーションをスレッドモード用のライブラリと結合します。スレッドモード用のライブラリは、以下のとおりです(インストールパスはデフォルト)。

Solaris32/64

```
/opt/FSUNod/lib/libOM.so
```

Linux32/64

```
/opt/FJSVod/lib/libOM.so
```

注意

Solaris32/64 Linux32/64

- 併用製品(言語ランタイムやDBMSなど)のライブラリをリンクする際は、必ずマルチスレッド対応された(スレッドセーフの)ライブラリを指定してください。誤ってマルチスレッド未対応(スレッドアンセーフ)のライブラリをリンクした場合、アプリケーションプロセス全体としてマルチスレッド動作が保証されなくなるため、アプリケーションの動作は不定となります(たまたま正常動作するケースもあれば、異常終了等、突然誤動作を起こすケースもあります)。アプリケーションのコンパイル・リンクの際には、スレッドセーフなライブラリとスレッドアンセーフなライブラリが混在して結合されないよう、細心の注意が必要です。アプリケーションのリンク状態は、lddコマンドにより確認できます。
- スレッドモードのアプリケーションを動作させる場合は、環境変数LD_LIBRARY_PATHに“/opt/FSUNod/lib/nt”(Solarisの場合)または“/opt/FJSVod/lib/nt”(Linuxの場合)が設定されていてはいけません。設定されている場合、アプリケーションが誤動作する要因となります。プロセスモードのアプリケーションとスレッドモードのアプリケーションを混在して運用する場合は、スレッドモードのアプリケーション起動時に環境変数LD_LIBRARY_PATHに上記が設定されていないことを確認してください。

プロセスモード

アプリケーションプロセスがシングルスレッドで動作する形態です。本タイプは、アプリケーションが呼び出す併用ライブラリがマルチスレッド対応していない場合に選択します。通常、スレッドモードを選択してください。

サーバアプリケーションでプロセスモードを使用する際の設定方法を以下に示します。

Windows32/64

インプリメンテーションリポジトリに登録したサーバアプリケーションの定義情報によって設定します。インプリメンテーションリポジトリへの定義情報の登録には、OD_impl_instコマンドを使用します。OD_impl_instコマンドの詳細については、“リファレンスマニュアル(コマンド編)”を参照してください。

定義情報では、proc_conc_maxに2以上の値を設定します。また、thr_conc_initとthr_conc_maximumに1を設定します。OD_impl_instコマンドで指定する定義ファイルの例を以下に示します。

```
rep_id           = IDL:test1/intf1:1.0
type             = persistent
proc_conc_max    = 8
thr_conc_init    = 1
thr_conc_maximum  = 1
```

なお、プロセスモードの場合は、サーバアプリケーションの活性化後の動作モードは、modeの設定内容にかかわらずSYNC_ENDになります。

Solaris32/64 Linux32/64

サーバアプリケーションをプロセスモード用のライブラリと結合します。プロセスモード用のライブラリは、以下のとおりです(インストールパスはデフォルト)。

Solaris32/64

```
/opt/FSUNod/lib/nt/libOM.so
```

Linux32/64

```
/opt/FJSVod/lib/nt/libOM.so
```

注意

Solaris32/64 Linux32/64

- アプリケーションのリンク時にスレッドライブラリを指定していた場合、誤動作の要因となります。このため、スレッドライブラリは指定しないでください。アプリケーションのリンク状態は、lddコマンドにより確認できます。
- サーバアプリケーションの活性化後の動作モードは、OD_impl_instコマンドを使用して登録したmodeの設定内容にかかわらず、SYNC_ENDとなります。
- プロセスモードのアプリケーションを動作させる場合は、環境変数LD_LIBRARY_PATHに“/opt/FSUNod/lib/nt”(Solarisの場合)／“/opt/FJSVod/lib/nt”(Linuxの場合)を設定しておく必要があります。設定されていない場合、誤動作の要因となります。
  設定方法を以下に示します。

  Solaris32/64

  bsh系の場合

  ```
  LD_LIBRARY_PATH = /opt/FSUNod/lib/nt:$LD_LIBRARY_PATH
  export LD_LIBRARY_PATH
  ```

  csh系の場合

  ```
  setenv LD_LIBRARY_PATH /opt/FSUNod/lib/nt:$LD_LIBRARY_PATH
  ```

Linux32/64

bsh系の場合

```
LD_LIBRARY_PATH = /opt/FJSVod/lib/nt:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

csh系の場合

```
setenv LD_LIBRARY_PATH /opt/FJSVod/lib/nt:$LD_LIBRARY_PATH
```

2.2.2 CORBAサーバアプリケーション作成時の共通留意事項

- Solaris32/64 Linux32/64
  CORBAアプリケーションは、動的リンク可能な実行可能ファイル／共有ライブラリ(オブジェクト)として作成する必要があります。リンク時に、静的リンクオプション(Solarisの場合：-dnオプション)を付加しないでください。

  アプリケーションのリンク時は、例示するライブラリに加えて、併用製品(DBMSなど)のライブラリなど、アプリケーションが必要とするすべてのライブラリを漏れなくリンクする必要があります。また、リンクしたライブラリ群の格納先は、アプリケーション実行時に環境変数LD_LIBRARY_PATH に設定しておく必要があります。なお、必要なライブラリのリンク漏れ／環境変数LLD_LIBRARY_PATH の設定漏れがあった場合、アプリケーション実行時に“BAD_OPERATION”などの例外が発生します。

- 動的スケルトンインタフェースを使用したCORBAサーバアプリケーションを運用する場合、インタフェース情報がインタフェースリポジトリに登録されている必要があります。このインタフェースリポジトリ上の登録情報と、IDLc コマンドにより生成されたスタブ内の情報間に不整合が生じたまま、クライアント・サーバ通信を行った場合、以下のような異常事象が発生します。

  ― サーバアプリケーションで受信した、クライアントアプリケーションからの要求データが不当な値となる。

  ― メモリ不足エラーが発生する。

  ― 処理要求が無応答となる。

  運用に際し、事前にインタフェースリポジトリ上の登録情報とスタブ内の情報に不整合がないかを確認してください。インタフェースリポジトリ上の登録内容は、odlistirコマンドにより確認できます。インタフェース情報の不整合をアプリケーション通信過程で自動検出する“インタフェース情報チェック機能”が提供されています(本機能はデフォルトの状態では有効になっていません)。前述したトラブルを未然防止するためにも、本機能の適用を推奨します。本機能の詳細については、“OLTPサーバ運用ガイド”の“インタフェース情報チェック機能を使用した運用”を参照してください。

2.2.3 Windows(R)アプリケーションに関する注意事項

(1) コンパイル時のオプションについて

(a) コンパイルに関する設定

コンパイルを実行する場合は、以下のオプションを設定してください。

Microsoft(R) Visual C++(R) .NET／Microsoft(R) Visual Studio(R)の場合:

[プロジェクト]-[プロパティ]-[構成プロパティ]-[C/C++] から、以下の表に示すオプションを設定してください。

カテゴリ	項目	設定値(32bit)	設定値(64bit)
コード生成	構造体メンバのアライメント	4 バイト	8 バイト
	使用するランタイム	マルチスレッド(DLL)	
最適化		デフォルトを推奨	
プリプロセッサ	プリプロセッサの定義	"OM_PC","OM_WIN32_BUILD", "__STDC__" を追加 (注1)	
言語	wchar_t をビルトイン型として扱う	いいえ	
詳細	呼び出し規約	_cdecl	

注1)
"__STDC__"を設定することにより、Microsoft拡張機能を前提とする処理でコンパイルエラーが発生することがあります。この場合、CORBAアプリケーション用の処理とMicrosoft拡張機能を前提とする処理のソースを分けて、Microsoft拡張機能を前提とする処理のソースの先頭に「#undef __STDC__」を追加して対処してください。

(b) リンクに関する設定

ライブラリのリンク設定は、以下から行ってください。

Microsoft(R) Visual C++(R) .NET／Microsoft(R) Visual Studio(R)の場合:

[プロジェクト]-[プロパティ]- [構成プロパティ]-[リンカ]-[入力]

リンクするライブラリについては、“提供プログラム”の“CORBAサービスの提供プログラム”－“B.1.2 ライブラリ”を参照してください。

また、Visual C++(R)の場合は、[ツール]-[オプション]-[ディレクトリ]の「インクルード」、「ライブラリ」にそれぞれインストールフォルダの下の"INCLUDE"、"LIB"フォルダを登録してください。

登録例)

- インクルードファイル
  C:¥MSDEV¥INCLUDE;C:¥MSDEV¥MFC¥INCLUDE;C:¥Interstage¥ODWIN¥INCLUDE;
  C:¥Interstage¥TD¥INCLUDE;C:¥Interstage¥ots¥INCLUDE;
  C:¥Interstage¥ESWIN¥INCLUDE
  (Visual C++(R)およびInterstageを、それぞれC:¥MSDEVおよびC:¥Interstageにインストールした場合)
- ライブラリファイル
  C:¥MSDEV¥LIB;C:¥MSDEV¥MFC¥LIB;C:¥Interstage¥ots¥LIB;
  C:¥Interstage¥ODWIN¥LIB;C:¥Interstage¥EXTP¥LIB;C:¥Interstage¥TD¥LIB;
  C:¥Interstage¥ESWIN¥LIB
  (Visual C++(R)およびInterstageを、それぞれC:¥MSDEVおよびC:¥Interstageにインストールした場合)

(2) コンパイル時に発生するエラーについて

(a) コンパイル時のエラー

Microsoft(R) Visual Studio(R)を使用してC++アプリケーションのコンパイルを行った場合、以下のようなエラーメッセージが表示され、コンパイルに失敗することがあります。

```
error C4980: '__value' : このキーワードを使用するには、/clr:oldSyntax コマンド ライン オプションが必要です
```

この場合、IDLファイルの変数名を変更する必要があります。上記の例の場合は、IDLファイルにおいて変数名が“value”となっている変数を別名に変更してください。
変更例を以下に示します。

例

修正前のIDLファイル

```
module ODsample{
    interface intf{
        readonly attribute long value;
        ・・・
    };
};
```

修正後のIDLファイル

```
module ODsample{
    interface intf{
        readonly attribute long value1;    /* value -> value1 */
        ・・・
    };
};
```

(b) リンク時のエラー

ODSV.LIB／ODWIN.LIBを使用して、Visual C++(R)でコンパイルを実行すると、以下のようなエラーメッセージが表示されることがあります。

```
リンク中...
odsv.lib(ODSV.dll) : error LNK2005: _CORBA_sequence_string_allocはすでにsimple_skel.objで定義されています。
odsv.lib(ODSV.dll) : warning LNK4006: _CORBA_sequence_string_allocはsimple_skel.objで定義されています。2つ目以降の定義は無視されます。
```

この場合、Visual C++(R)で以下のように設定し、再度コンパイルを実行してください。

Microsoft(R) Visual C++(R) .NETまたはMicrosoft(R) Visual Studio(R)の場合:

[プロジェクト]-[プロパティ]- [構成プロパティ]-[リンカ]-[コマンドライン]-[追加のオプション]に、/FORCEオプションを追加します。

また、C++マッピングの場合、以下のラーメッセージが表示されることがありますが、本メッセージは問題ありません。

```
リンク中...
LINK : warning LNK4103:
削除するデストラクタ "public: virtual void * __thiscall CORBA::Object::`vector deleting destructor'(unsigned int)"
のインポートです; イメージは正しく動作しない可能性があります
```

(3) アプリケーションのDLL化について

Visual C++(R)で、IDLコンパイラから生成されたファイルを使用してDLLを作成する方法について説明します。

(a) IDLコンパイル

DLL作成用のソースを生成する場合は、IDLコンパイル時に-dyオプションを指定してIDLコンパイルを行ってください。

(b) プリプロセッサの定義

DLL作成時およびDLLリンク時に、Visual C++(R)で以下のような手順で表に示すプリプロセッサの定義を設定してください。

Microsoft(R) Visual C++(R) .NETまたはMicrosoft(R) Visual Studio(R)の場合:

[プロジェクト]-[プロパティ]- [構成プロパティ]-[C/C++]-[プリプロセッサ]-[プロセッサの定義]

言語	DLL作成/リンク	プリプロセッサ定義
C	DLL作成時	"OM_USRDLL_MAKE"
	DLLリンク時	"OM_USRDLL_LINK"
C++	スケルトンDLL作成時 (注1)	"OM_SKDLL_MAKE"
	スケルトンDLLリンク時	"OM_SKDLL_LINK"
	スタブDLL作成時 (注2)	"OM_STDLL_MAKE"
	スタブDLLリンク時	"OM_STDLL_LINK"

注1)スケルトンファイルおよびメソッド実装部からDLLを作成する場合
注2)スタブファイルからDLLを作成する場合

(c) 注意事項

上記のようにプロジェクトの設定でプリプロセッサを定義する方法の他に、IDLコンパイラ生成ヘッダファイル(*.h)をインクルードするソースファイルで、#define文を使用して指定することもできます。
複数のIDLファイルから生成されたヘッダファイルを使用してDLLのリンクと作成を両方行う場合の例を以下に示します。

```
#define OM_USRDLL_LINK
#include "a.h" // IDLコンパイラ生成ヘッダ
#undef OM_USRDLL_LINK

#define OM_USRDLL_MAKE
#include "b.h" // IDLコンパイラ生成ヘッダ
#undef OM_USRDLL_MAKE

// 以下ソースコード
...
```

(4) リンクするライブラリについて

一つのアプリケーションには、サーバ用ライブラリかクライアント用ライブラリのどちらか一方のライブラリをリンクしてください。

2.2.4 Solarisアプリケーションに関する注意事項

Solaris32/64

Oracle Solaris Studio 12.2以降を使用してアプリケーションを作成する場合に以下のコンパイルエラーとなる場合があります。

```
Error: …… requires an lvalue.
エラー: …… を初期化するには左辺値が必要です.
```

CCコマンドに"-features=rvalueref"を指定してコンパイルしてください。

Solaris64

アプリケーションをコンパイル・リンクする際には64ビット対応するためのオプションを指定して下さい。指定するオプションは使用するコンパイラのバージョンにより異なるため、コンパイラのマニュアルを確認して下さい。

2.2.5 Linuxアプリケーションに関する注意事項

Linux32

サーバアプリケーションを作成する場合、ldコマンド実行時は“-E”オプションを、gcc/g++コマンド実行時は“-Wl,-E”オプションを指定する必要があります。サンプルプログラムsimpleのMakefileを参考にしてください。

なお、Linux for Intel64（32ビット互換)版Interstage Application Serverでアプリケーションを作成する場合は、gcc/g++コマンドに“-m32 -mtune=i386”オプションを指定する必要があります。

Linux64

サーバアプリケーションを作成する場合、ldコマンド実行時は“-E”オプションを、gcc/g++コマンド実行時は“-Wl,-E”オプションを指定する必要があります。サンプルプログラムsimpleのMakefileを参考にしてください。

Linux32/64

アプリケーションは、実行するOSと同じディストリビューション・同じバージョンのOSで作成する必要があります。OSをバージョンアップする場合は、新しいバージョンのOS環境でアプリケーションを再作成する必要があります。

2.2.6 サーバアプリケーションのコンパイル・リンク例 (Solaris)

2.2.6.1 Cサーバアプリケーション(スレッドモード)の場合

■実行可能ファイルとして作成する場合

例

odsvapl.c:サーバアプリケーションソース
IDL_skel.c:スケルトンソース
odsvapl:サーバアプリケーション実行可能ファイル

```
% IDLc IDL.idl
% cc -c -D_REENTRANT -DNeedFunctionPrototypes -I/opt/FSUNod/include -o odsvapl.o odsvapl.c
% cc -c -D_REENTRANT -DNeedFunctionPrototypes -I/opt/FSUNod/include -o IDL_skel.o IDL_skel.c
% cc -lthread -lsocket -lnsl -L/opt/FSUNod/lib -lOM -o odsvapl odsvapl.o IDL_skel.o
```

■共有オブジェクトとして作成する場合

例

odsvapl.c:サーバアプリケーションソース
IDL_A_B_skel.c:スケルトンソース
libA_B.so:サーバアプリケーション共有オブジェクト

```
% cat IDL.idl
  module A {
      interface B {
          :
      }
  }
% IDLc -dy IDL.idl
% cc -c -D_REENTRANT -DNeedFunctionPrototypes -I/opt/FSUNod/include -o odsvapl.o odsvapl.c
% cc -c -D_REENTRANT -DNeedFunctionPrototypes -I/opt/FSUNod/include -o IDL_A_B_skel.o IDL_skel.c
% cc -G -Kpic -lthread -lsocket -lnsl -L/opt/FSUNod/lib -lOM -o libA_B.so odsvapl.o IDL_A_B_skel.o
```

使用可能なコンパイラについては、“システム設計ガイド”の“アプリケーション開発時に必要なソフトウェア”を参照してください。

スレッドライブラリ(libthread.so)は、他のライブラリ群の中で必ず先頭にくるよう指定してください。リンク後、lddコマンドによりシステムライブラリ"libc.so"より上方に"libthread.so"がリンクされていることを確認してください。"libthread.so"が"libc.so"より下方にリンクされている場合は、スレッドライブラリが正常に機能せず、アプリケーション誤動作の要因となるため、注意してください。

```
% ldd odsvapl
libthread.so.1 =>       /usr/lib/libthread.so.1
        :                       :
libc.so.1 =>            /usr/lib/libc.so.1
        :                       :
```

2.2.6.2 Cサーバアプリケーション(プロセスモード)の場合

例

odsvapl.c:サーバアプリケーションソース
IDL_skel.c:スケルトンソース
odsvapl:サーバアプリケーション実行可能ファイル

```
% IDLc IDL.idl
% cc -c -DNeedFunctionPrototypes -I/opt/FSUNod/include -o odsvapl.o odsvapl.c
% cc -c -DNeedFunctionPrototypes -I/opt/FSUNod/include -o IDL_skel.o IDL_skel.c
% cc -lsocket -lnsl -L/opt/FSUNod/lib/nt -lOM -o odsvapl odsvapl.o IDL_skel.o
```

使用可能なコンパイラについては、“システム設計ガイド”の“アプリケーション開発時に必要なソフトウェア”を参照してください。

プロセスモードのアプリケーションでは、リンク時にスレッドライブラリ(libthread.so)を付加しないでください。誤ってスレッドライブラリを付加した場合は、アプリケーションが誤動作する要因となります。lddコマンドによりスレッドライブラリが混在していないかを確認してください。

プロセスモードのアプリケーションを動作させる場合は、環境変数LD_LIBRARY_PATHに“/opt/FSUNod/lib/nt”を設定しておく必要があります。設定されていない場合、アプリ誤動作の要因となります。

2.2.6.3 COBOLサーバアプリケーション(スレッドモード)の場合

例

odsvapl.cbl:主プログラムソース(初期化処理部)
odsvapl_sub.cbl:副プログラムソース(I/F実装部)
IDL_A_B_skel.cbl:スケルトンソース
IDL_cdr.cbl:CDRソース
odsvapl:サーバアプリケーション実行可能ファイル
libA-B.so:サーバアプリケーション共有オブジェクト

```
% cat IDL.idl
  module A {
      interface B {
          :
      }
  }
% IDLc -cobol IDL.idl
% CORBA=/opt/FSUNod/include/COBOL
% export CORBA
% cobol -c -M -Tm odsvapl.cbl
% cobol -Tm -L/opt/FSUNod/lib -lOMcblMT -o odsvapl odsvapl.o
% cobol -G -Tm -o libIDL_cdr.so IDL_cdr.cbl
% cobol -G -Tm -o libIDL_A_B_skel.so IDL_A_B_skel.cbl
% cobol -G -Tm -L/opt/FSUNod/lib -lOMcblMT -lIDL_cdr -lIDL_A_B_skel -o libA-B.so odsvapl_sub.cbl
```

COBOLで作成するCORBAアプリケーションは、動的リンク構造である必要があります。動的プログラム構造は動作保証されないため、翻訳オプションとして“DLOAD”を指定しないでください

COBOLソースのコンパイル時、“-dy”オプションを指定しないでください。誤って指定した場合、アプリケーション実行時にCORBAサービスが提供するCOBOL用ライブラリ(libOMcblMT.so／libOMircblMT.so)の呼出しに失敗し、“BAD_OPERATION”例外が発生します。

2.2.6.4 COBOLサーバアプリケーション(プロセスモード)の場合

例

odsvapl.cbl：主プログラムソース(初期化処理部)
odsvapl_sub.cbl：副プログラムソース(I/F実装部)
IDL_A_B_skel.cbl：スケルトンソース
IDL_cdr.cbl：CDRソース
odsvapl：サーバアプリケーション実行可能ファイル
libA-B.so：サーバアプリケーション共有オブジェクト

```
% cat IDL.idl
  module A {
      interface B {
          :
      }
  }
% IDLc -cobol IDL.idl
% CORBA=/opt/FSUNod/include/COBOL
% export CORBA
% cobol -c -M odsvapl.cbl
% cobol -L/opt/FSUNod/lib -lOMcbl -o odsvapl odsvapl.o
% cobol -G -o libIDL_cdr.so IDL_cdr.cbl
% cobol -G -o libIDL_A_B_skel.so IDL_A_B_skel.cbl
% cobol -G -L/opt/FSUNod/lib -lOMcbl -lIDL_cdr -lIDL_A_B_skel -o libA-B.so odsvapl_sub.cbl
```

COBOLで作成するCORBAアプリケーションは、動的リンク構造である必要があります。動的プログラム構造は動作保証されないため、翻訳オプションとして“DLOAD”を指定しないでください

COBOLソースのコンパイル時に、“-dy”オプションを指定しないでください。誤って指定した場合、アプリケーション実行時にCORBAサービスが提供するCOBOL用ライブラリ(libOMcbl.so／libOMircbl.so)の呼出しに失敗し、“BAD_OPERATION”例外が発生します。

プロセスモードのアプリケーションを動作させる場合は、環境変数LD_LIBRARY_PATHに“/opt/FSUNod/lib/nt”を設定しておく必要があります。設定されていない場合、アプリ誤動作の要因となります。

2.2.7 Javaアプリケーションのコンパイル

Javaアプリケーションのコンパイル時に使用するjavacコマンドは、Interstage Application Serverが提供するJDKに含まれているコマンドを使用してください。

2.3 トラブル発生時の原因調査

2.3.1 保守情報

CORBAサービスでは、CORBAアプリケーション運用におけるトラブル発生時など、原因調査のための情報として、以下の情報を採取できます。それぞれの参照先を以下に示します。

参照

- CORBAサービスのトレース情報
  “トラブルシューティング集”の“障害調査資料の採取”－“CORBAサービスのトレース情報の採取”
- CORBAサービスのログ情報
  “トラブルシューティング集”の“障害調査資料の採取”－“CORBAサービスのログ情報の採取”
- CORBAサービスのスナップショット
  “トラブルシューティング集”の“障害調査資料の採取”－“CORBAサービスのスナップショットの採取”

2.4 トランザクションアプリケーションからCORBAアプリケーションへ移行

トランザクションアプリケーションをCORBAアプリケーションへ移行する場合の手順を説明します。

(1) APIの移行

トランザクションアプリケーションで使用しているAPIを、CORBAアプリケーション用のAPIに変更します。

トランザクションアプリケーションとCORBAアプリケーションの各言語のAPI対比表を以下に示します。

API対比表（C言語）

トランザクションアプリケーションAPI名	CORBAアプリケーションAPI名	備考
TD_xxx_alloc()	CORBA_xxx_alloc()	
TD_free()	CORBA_free()	
TD_get_smo_name()	－	CORBAアプリケーションでは未サポート
TD_get_client_id()	－	CORBAアプリケーションでは未サポート
TD_get_user_information()	－	CORBAアプリケーションでは未サポート
TD_getsessionid()	－	CORBAアプリケーションでは未サポート
TD_setcontcvt()	－	CORBAアプリケーションでは未サポート
TD_refsessionid()	－	CORBAアプリケーションでは未サポート

API対比表（C++言語）

トランザクションアプリケーションAPI名	CORBAアプリケーションAPI名	備考
TD::xxx_alloc()	CORBA::xxx_alloc()	
TD::free()	CORBA::free()	
TD::string_free()	CORBA::string_free()	

トランザクションアプリケーションAPI名	CORBAアプリケーションAPI名	備考
TD::wstring_free()	CORBA::wstring_free()	
TD::get_smo_name()	－	CORBAアプリケーションでは未サポート
TD::get_client_id()	－	CORBAアプリケーションでは未サポート
TD::get_user_information()	－	CORBAアプリケーションでは未サポート
TD::getsessionid()	－	CORBAアプリケーションでは未サポート
TD::setcontcvt()	－	CORBAアプリケーションでは未サポート
TD::refsessionid()	－	CORBAアプリケーションでは未サポート

API対比表（COBOL）

トランザクションアプリケーションAPI名	CORBAアプリケーションAPI名	備考
TDXXXALLOC	CORBA-XXX-ALLOC	
TDSTRINGSET	CORBA-STRING-SET	
TDSTRINGGET	CORBA-STRING-GET	
TDWSTRINGSET	CORBA-WSTRING-SET	
TDWSTRINGGET	CORBA-WSTRING-GET	
TDFREE	CORBA-FREE	
TDGETSMONAME	－	CORBAアプリケーションでは未サポート
TDGETCLIENTID	－	CORBAアプリケーションでは未サポート
TDGETUSERINFORMATION	－	CORBAアプリケーションでは未サポート
TDGETSESSIONID	－	CORBAアプリケーションでは未サポート
TDSETCONTCVT	－	CORBAアプリケーションでは未サポート
TDREFSESSIONID	－	CORBAアプリケーションでは未サポート
TDFREESEQUENCEBUF	CORBA-FREE	
TDSEQUENCEELEMENTGET	CORBA-SEQUENCE-ELEMENT-GET	
TDSEQUENCEELEMENTSET	CORBA-SEQUENCE-ELEMENT-SET	

注意

CORBAアプリケーション未サポート機能を使用する場合は、トランザクションアプリケーションを使用してください。

また、any型、sequence型のデータを利用している場合、必要に応じて領域の開放設定を行ってください。各言語の対応方法を以下に示します。

C言語

any型を利用している場合、CORBA_any_set_releaseを利用して領域の開放設定を行ってください。

sequence型を利用している場合、CORBA_sequence_set_releaseを利用して領域の開放設定を行ってください。

詳細は、“3.7.4 any型”、“3.7.5 シーケンス型”、“3.8 any型、sequence型のリリースフラグ”を参照してください。

C++言語

any型を利用している場合、CORBA_Anyクラスのインスタンス生成時または、replaceメソッドを利用して領域の開放設定を行ってください。

sequence型を利用している場合、シーケンスクラスのインスタンス生成時に領域の開放設定を行ってください。

詳細は、“4.6.4 any型”、“4.6.5 シーケンス型”、“4.6.12 any型、sequence型のリリースフラグ”を参照してください。

COBOL

any型を利用している場合、CORBA-ANY-SET-RELEASEを利用して領域の開放設定を行ってください。

sequence型を利用している場合、CORBA-SEQUENCE-SET-RELEASEを利用して領域の開放設定を行ってください。

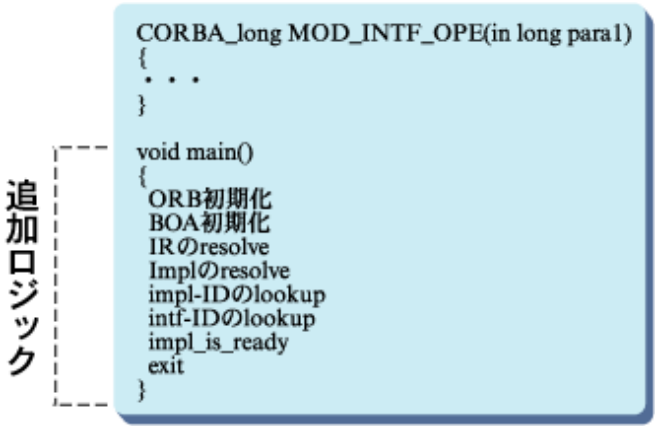
詳細は、“6.8.3 any型”、“6.8.4 シーケンス型”、“6.9 any型、sequence型のリリースフラグ”を参照してください。

(2) インクルードファイルの修正

トランザクションアプリケーションでは、トランザクションアプリケーション用スケルトンのインクルードファイル(モジュール名_インタフェース名.h)をインクルードしていましたが、CORBAアプリケーションでは、CORBAアプリケーション用スケルトンのヘッダファイル(IDLファイル名.h)をインクルードするように修正します。

(3) mainロジックの追加

CORBAアプリケーションでは、main関数を持つ実行モジュールを作成する必要があります。
そのため、main関数を作成し、CORBAアプリケーションの初期化処理を行う必要があります。



【移行例】

トランザクションアプリケーションからCORBAアプリケーションに移行するためのアプリケーションソースの修正例を、IDL定義例とともに以下に示します。

■IDL定義(test.idl)

```
module MOD {
  interface INTF {
    long OPE(in string para1);
  };
};
```

■トランザクションアプリケーションのソース

```
#include "orb.h"
#include "MOD_INTF.h"

CORBA_long MOD_INTF_OPE(CORBA_string para1)
{
  CORBA_long ret = 0;
  if (strcmp(para1," test" ) == 0) {
    ret = 0;
  } else {
    ret = 1;
  }
  TD_free(para1);
  return ret;
}
```

■CORBAアプリケーションへのソース修正例

上記トランザクションアプリケーションをCORBAアプリケーションに移行する場合は、下線部分を変更します。

```
#include  "orb.h"
#include  "test.h"

CORBA_long MOD_INTF_OPE(CORBA_string para1)
{
  CORBA_long ret = 0;
  if (strcmp(para1,"test") == 0) {
    ret = 0;
  } else {
    ret = 1;
  }
  CORBA_free(para1);
  return ret;
}

void main(int argc,char **argv)
{
  int current_argc = argc;
  orb = CORBA_ORB_init(....);

  boa = CORBA_ORB_BOA_init(....);

  impl_ret = CORBA_ORB_resolve_initial_references(....);

  impl = FJ_Implementation_lookup_id(....);
```

```
  intf = CORBA_Repository_lookup_id(....);

  CORBA_BOA_impl_is_ready(....);

  exit();
}
```

2.5 ネットワーク制御

Interstage Application ServerのCORBAサービスのネットワーク制御について説明します。

2.5.1 クライアントが使用するIPアドレス

クライアント環境で物理IPアドレスと仮想IPアドレスをあわせて複数のネットワークインタフェースがある場合、どのIPアドレスを使用してサーバと通信するかはOSの制御に依存します。
クライアント・サーバ間にファイヤウォールを配置してIPアドレスで通信をブロックするなど、クライアントのIPアドレスを意識して運用する場合は、注意してください。

2.5.2 コネクション接続

クライアントアプリケーションがサーバアプリケーションと通信する際、最初にコネクション接続を行います。サーバマシンが未起動状態などの理由でパケットが接続先に到達できない場合、一般的にコネクション接続がタイムアウトするまでに3分程度かかります(タイムアウト時間は、OSの制御に依存します)。
コネクション接続が完了するまでは、クライアントタイムアウト機能が有効となりません。通信時にサーバマシンが未起動状態などの可能性がある場合は、コネクション接続がタイムアウトするまでの時間を考慮した運用を行ってください。

第3章 アプリケーションの開発(C言語)

3.1 アプリケーションの開発手順

3.1.1 アプリケーションのインタフェース定義(IDL)の記述

サーバアプリケーションが提供するインタフェースを定義するために、IDLファイルを作成します。IDLファイルの書式については、“IDL定義”を参照してください。

例

```
module ODdemo{
  interface calculator{
    exception ZEROPARAM {};
    struct result {
      long    add_result;
      long    subtract_result;
      long    multiple_result;
      float   divide_result;
    };

    result calculate( in long a, in long b )
        raises( ZEROPARAM );
  };
};
```

3.1.2 IDLファイルのコンパイル

IDLcコマンドでIDLファイルをコンパイルします。この結果、IDL定義をC言語にマッピングした、複数の*.c,*.hファイルが出力されます。IDLcコマンドの詳細については、“リファレンスマニュアル(コマンド編)”の“IDLc”を参照してください。

例

```
IDLc simple.idl
```

simple.idlというIDLファイルをIDLcコマンドでオプションを指定せずにコンパイルすると、以下のファイルが出力されます。

ファイル名	ファイル種別
simple.h	共通ヘッダファイル
simple_cdr.h	CDRヘッダファイル
simple_cdr.c	CDRソースファイル
simple_stub.c	スタブファイル
simple_skel.c	スケルトンファイル

3.1.3 サーバアプリケーションのプログラミング

サーバアプリケーションのコーディングを行います。サーバアプリケーションは、「初期化処理部」と「インタフェース実装部」に分けられます。詳細については、“3.2 サーバアプリケーションのプログラミング(静的スケルトンインタフェース)”を参照してください。また、アプリケーション開発時の留意事項として“2.1 コーディング”を参照してください。

3.1.4 クライアントアプリケーションのプログラミング

クライアントアプリケーションのコーディングを行います。クライアントアプリケーションは、「初期化処理部」と「メソッド呼出し部」に分けられます。詳細については、“3.5 クライアントアプリケーションのプログラミング（静的起動インタフェース）”を参照してください。また、アプリケーション開発時の留意事項として“2.1 コーディング”を参照してください。

3.1.5 アプリケーションのコンパイルとリンク

開発環境を使用してアプリケーションのコンパイル・リンクを行います。
サーバアプリケーションは、IDLコンパイルで生成されたスケルトンファイルとユーザが作成したサーバアプリケーション用のファイルをコンパイル・リンクします。
クライアントアプリケーションは、IDLコンパイルで生成されたスタブファイルとユーザが作成したクライアントアプリケーション用のファイルをコンパイル・リンクします。
コンパイル・リンク時の注意事項については、“2.2 コンパイル・リンク”を参照してください。

3.1.6 サーバアプリケーションの登録

OD_impl_instコマンドを使用して、インプリメンテーションリポジトリにサーバアプリケーション情報を登録します。
また、OD_or_admコマンドを使用して、ネーミングサービスにサーバアプリケーションのオブジェクトリファレンスを登録します。

例

OD_impl_instコマンド実行例

```
OD_impl_inst -ax def
```

defファイルの内容

```
rep_id  = IDL:ODdemo/calculator:1.0
type    = persistent
mode    = SYNC_END
```

OD_or_admコマンド実行例

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -n ODdemo::calculator
```

3.2 サーバアプリケーションのプログラミング（静的スケルトンインタフェース）

サーバアプリケーションは、初期化処理部とインタフェースの実装部分から構成されます。初期化処理部では、以下に示す処理を行います。

3.2.1 初期化

CORBAの初期化メソッドCORBA_ORB_init()を呼び出し、初期化処理を行います。結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
main( argc, argv )
    int     argc;
    char    *argv[];
{
    int                 current_argc = argc;
    CORBA_ORB           orb;    /* ORBのオブジェクトリファレンス */
    CORBA_Environment   env;    /* 例外情報 */

    orb = CORBA_ORB_init( &current_argc,argv, FJ_OM_ORBid, &env );
```

また、CORBA_ORB_BOA_init()で基本オブジェクトアダプタの初期化処理を行います。

```
     CORBA_BOA  boa;            /* BOA用オブジェクトリファレンス */

    /* BOAの取得 */
    boa = CORBA_ORB_BOA_init( orb, &current_argc, argv, CORBA_BOA_OAid, &env );
```

なお、必要であれば、サーバアプリケーションの初期化も行います。

3.2.2 サーバの活性化

サーバアプリケーションの初期化が完了すると、ORBに対してその旨を通知します。ORBは、この命令が発行された時点で、クライアントからの要求をサーバアプリケーションに伝えます。活性化の方法は、サーバのタイプにより以下のように異なります。

サーバのタイプ	メソッド
shared server	CORBA_BOA_impl_is_ready
unshared server	CORBA_BOA_obj_is_ready
persistent server	CORBA_BOA_impl_is_ready

(1)インプリメンテーションリポジトリのオブジェクトリファレンスの獲得

CORBA_ORB_resolve_initial_references()を使用して、インプリメンテーションリポジトリのオブジェクトリファレンスを獲得します。第2パラメタには、CORBA_ORB_ObjectId_ImplementationRepositoryを指定します。

(2) ImplementationDefオブジェクトのオブジェクトリファレンスの検索

FJ_ImplementationRep_lookup_id()を使用して、ImplementationDefオブジェクトのオブジェクトリファレンスを求めます。第1パラメタには、サーバアプリケーションのFJ_ImplementationRepオブジェクトを指定します。

(3) サーバの活性化

CORBA_BOA_impl_is_ready()／CORBA_BOA_obj_is_ready()を使用して、サーバの活性化を行います。

サーバアプリケーションはプロセスで実装するか、スレッドで実装するかにより、メソッドの動作が異なります。また、スレッドで実装する場合、インプリメンテーションリポジトリ内に設定されている情報によっても異なります。動作の違いを以下に示します。

条件		動作
プロセスで実装	OD_impl_instで mode=SYNC_END	メソッドは即時に復帰せず、このオブジェクトの停止がオペレータにより指示されたときに復帰します。(注)
スレッドで実装	OD_impl_instで mode=SYNC_END	同上

条件		動作
	OD_impl_instで mode=COMPATIBLE	メソッドは即時に復帰します。 その後、thr_exitを呼び出す必要があります。

注) メソッドの復帰後は、初期化時にオープンしたファイルのクローズ処理、および領域の解放処理などの非活性化処理を行う必要があります。詳細については、“3.2.4 サーバの非活性化”を参照してください。

注意

Windows32/64

以下の条件のとき、CORBA_BOA_impl_is_ready()／CORBA_BOA_obj_is_ready()が復帰せずにサーバアプリケーションが終了します。

- サーバアプリケーションをプロセスモードで作成している場合 かつ
- サーバアプリケーションをCORBAワークユニット運用している場合 かつ
- CORBAワークユニットを停止した場合

CORBA_BOA_impl_is_ready()／CORBA_BOA_obj_is_ready()の後に後処理を行う場合は、サーバアプリケーションをスレッドモードで作成するか、またはワークユニットの出口機能を利用してください。ワークユニットの出口機能に関しては、“OLTPサーバ運用ガイド”の“ワークユニットの出口機能”を参照してください。

以下の例で使用されている“_IMPL_ODdemo_calculator”は、IDLコンパイラによって生成されたヘッダファイルに定義されている“_IMPL_”で始まる文字列です。例ではヘッダファイルに定義されている文字列を使用していますが、活性化対象のインプリメンテーションリポジトリIDを指定する必要があります。

```
CORBA_ImplementationDef  impl;       /* ImplementationDefオブジェクトリファレンス */
FJ_ImplementationRep     impl_rep;   /* ImplementationRepオブジェクトリファレンス */

/* インプリメンテーション情報のオブジェクトリファレンス検索 */
impl_rep = CORBA_ORB_resolve_initial_references(
    orb,
    CORBA_ORB_ObjectId_ImplementationRepository,
    &env);

/* ImplementationRepオブジェクトリファレンス検索 */
impl = FJ_ImplementationRep_lookup_id(
    impl_rep,
    _IMPL_ODdemo_calculator,          /*リポジトリID */
    &env );

/* サーバの活性化 */
CORBA_BOA_impl_is_ready(boa, impl, &env );
```

3.2.3 インタフェース実装関数

初期化処理の後、サーバアプリケーションで実装するインタフェースの処理を記述します。インタフェース実装関数の第1パラメタにはオブジェクトリファレンスが渡され、次にIDLで定義したパラメタ、最後にCORBA_Environment構造体へのポインタが渡されます。
なお、アプリケーションで使用可能なCORBAのデータ型については、“3.7 データ型に対するマッピング”を参照してください。

```
ODdemo_calculator_result            /* 復帰値の型 */
ODdemo_calculator_calculate(        /* calculateメソッドの実装関数 */
    ODdemo_calculator   obj,        /* サーバのオブジェクトリファレンス */
    CORBA_long          a,          /* 入力値 */
    CORBA_long          b,          /* 入力値 */
    CORBA_Environment   *env )      /* エラー値 */
{
```

```
    ODdemo_calculator_result  res;  /* 復帰値 */

    res.add_result = 0;
    res.subtract_result = 0;
    res.multiple_result = 0;
    res.divide_result = 0;

    if( b == 0 ){                   /* 除算チェック */
        CORBA_BOA_set_exception(
            boa,
            CORBA_USER_EXCEPTION,
            ex_ODdemo_calculator_ZEROPARAM,
            NULL,
            env );
        return( res ) ;
    }
    res.add_result = a+b;                   /* 加算結果設定 */
    res.subtract_result = a-b;              /* 減算結果設定 */
    res.multiple_result = a*b;              /* 乗算結果設定 */
    res.divide_result = (CORBA_float)a/b;  /* 除算結果設定 */
    return res;
}
```

3.2.4 サーバの非活性化

サーバアプリケーションは、利用者などからの停止要求を受けた場合、クライアントからの要求を以降受け付けない旨をORBに対して通知します。ORBはこの通知を受けて、クライアントからの処理要求をサーバアプリケーションに伝えず、クライアントに例外を返します。非活性化の方法は、サーバタイプにより以下のように異なります。

サーバのタイプ	メソッド
shared server	CORBA_BOA_deactivate_impl
unshared server	CORBA_BOA_deactivate_obj
persistent server	CORBA_BOA_deactivate_impl

CORBA_BOA_deactivate_impl()では、サーバアプリケーションのImplementationRepオブジェクトをパラメタとして指定します。
CORBA_BOA_deactivate_obj()では、オブジェクトリファレンスをパラメタとして指定します。

```
CORBA_BOA_deactivate_impl( boa, impl, &env );    /* サーバの非活性化 */
```

なお、ワークユニット運用している場合は、isstopwuコマンド／Interstage管理コンソールから停止操作が実行されることにより、クライアントからの要求を以降受け付けない旨をORBに対して通知します。そのため、アプリケーション側で非活性化用のメソッドを発行する必要はありません。
ワークユニット運用でないCORBAアプリケーションの場合も、odcntlqueコマンドでサーバアプリケーションを終了する場合は、アプリケーション側で非活性化用のメソッドを発行する必要はありません。

3.3 サーバアプリケーションの例外処理

(1)例外情報の設定

ユーザ例外を設定する場合は、CORBA_string_alloc()でユーザ例外を識別する文字列を設定する領域を獲得し、この領域に文字列をコピーします。続いて、IDLコンパイラで生成されたalloc関数でユーザ例外を設定する領域を獲得します。alloc関数名は、IDLで指定したインタフェース名、例外の識別名、およびallocを、アンダースコア("_")でつなげた形式で指定します。IDLで定義した例外情報はC言語では構造体になるので、構造体のメンバ変数に値を設定します。なお、今回使用した例では、ユーザ例外の識別情報を設定していません。最後に、CORBA_BOA_set_exception()でCORBA_Environment構造体に例外情報を設定します。

```
ODdemo_calculator_result
ODdemo_calculator_calculate(
    ODdemo_calculator   obj,
    CORBA_long          a,
    CORBA_long          b,
    CORBA_Environment   *env )
{
    ODdemo_calculator_result   res;

            :
    if( b == 0 ){
         /* エラーZEROPARAMをenvに設定します */
         CORBA_BOA_set_exception(
             boa,                              /* BOAオブジェクトリファレンス */
             CORBA_USER_EXCEPTION,             /* ユーザ例外 */
             ex_ODdemo_calculator_ZEROPARAM,   /* ユーザ例外の識別子 */
             NULL,                             /* ユーザ例外の識別情報 */
             env );                            /* CORBA_Environment */
        return( 0 ) ;
    }
        :
    return res;
}
```

注意

サーバメソッドの引数に渡されるCORBA_Environment構造体を、サーバメソッド内部から他のメソッドを呼び出す場合の引数として使用しないでください。サーバメソッドの例外情報に、他のメソッドで発生したサーバメソッド内部の例外情報が設定されることになります。
サーバメソッド内部から他のメソッドを呼び出す場合は、別のCORBA_Environment構造体を使用してください。
以下に正しいプログラム例、および誤ったプログラム例を示します。

[正しいプログラム例]

```
ODdemo_calculator_result
ODdemo_calculator_calculate(
    ODdemo_calculator   obj,
    CORBA_long          a,
    CORBA_long          b,
    CORBA_Environment   *env )
{
    CosNaming_NamingContext  nc;
    CosNaming_Name  name;
    CORBA_Object  obj;
    CORBA_Environment  local_env; /* 別のCORBA_Environment構造体を用意 */

    memset( &local_env, 0, sizeof(CORBA_Environment) );
        :

    /*
     * CosNaming_NamingContext_bind()の例外情報は、local_envに設定されるため、
     * ODdemo_calculator_calculate()の例外情報として扱われることはない
     */
    CosNaming_NamingContext_bind( nc, &name, obj, &local_env );
        :
}
```

[誤ったプログラム例]

```
ODdemo_calculator_result
ODdemo_calculator_calculate(
```

```
    ODdemo_calculator   obj,
    CORBA_long          a,
    CORBA_long          b,
    CORBA_Environment   *env )
{
    CosNaming_NamingContext  nc;
    CosNaming_Name           name;
    CORBA_Object             obj;
        :

    /*
     * CosNaming_NamingContext_bind()が例外復帰した場合、
     * CosNaming_NamingContext_bind()の例外情報がODdemo_calculator_calculate()の
     * 例外情報として扱われてしまう
     */
    CosNaming_NamingContext_bind( nc, &name, obj, env );
        :
}
```

(2)例外情報の獲得

サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“3.6 クライアントアプリケーションの例外処理”を参照してください。

3.4 サーバアプリケーションの登録

作成したアプリケーションをインプリメンテーションリポジトリ、ネーミングサービスに登録します。

3.4.1 インプリメンテーションリポジトリへの登録

OD_impl_instコマンドを使用してインプリメンテーションリポジトリにサーバアプリケーション情報を登録します。
OD_impl_instコマンドでの登録処理の例と指定する情報について以下に示します。

例

Windows32/64

インプリメンテーションリポジトリへの登録

```
OD_impl_inst -a -r IDL:ODdemo/calculator:1.0 -t S -f D:¥server¥simple_s.exe
```

-a

サーバアプリケーション情報を登録します。

-r IDL:ODdemo/calculator:1.0

インプリメンテーションリポジトリIDを指定します。

-t S

サーバタイプを英大文字1字で指定します。指定できるタイプには、“P”(persistent)、“S”(shared)、“U”(unshared)があります。

-f D:¥server¥simple_s.exe

サーバアプリケーションのパス名を指定します。

Solaris32/64 Linux32/64

インプリメンテーションリポジトリへの登録

```
OD_impl_inst -a -r IDL:ODdemo/calculator:1.0 -t S -f /home/guest/simple_s -u user -g group
```

-a

サーバアプリケーション情報を登録します。

-r IDL:ODdemo/calculator:1.0

インプリメンテーションリポジトリIDを指定します。

-t S

サーバタイプを英大文字1字で指定します。指定できるタイプには、“P”(persistent)、“S”(shared)、“U”(unshared)があります。

-f /home/guest/simple_s

サーバアプリケーションのパス名を指定します。

-u user

サーバアプリケーション実行時のユーザIDを指定します。

-g group

サーバアプリケーション実行時のグループIDを指定します。

3.4.2 ネーミングサービスへの登録

作成したサーバアプリケーションを他のアプリケーションからオブジェクトとしてアクセスできるようにするために、そのオブジェクトを識別するためのオブジェクトリファレンスを作成します。同時に作成したオブジェクトリファレンスをネーミングサービスに登録します。
オブジェクトを作成する方法として以下の方法があります。

- OD_or_admコマンドを使用します。
- サーバアプリケーションでBOAの機能を使用してオブジェクトリファレンスを作成し、ネーミングサービスに登録します。

■OD_or_admコマンドを使用する方法

OD_or_admコマンドは、オブジェクトリファレンス作成後、ネーミングサービスに登録します。OD_or_admコマンドでの登録例および指定する情報を以下に示します。

例

Windows32/64

ネーミングサービスへの登録

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -L SJIS -n ODdemo::calculator
```

-c IDL:ODdemo/calculator:1.0

指定されたインタフェースリポジトリIDでオブジェクトリファレンスを登録します。

-L SJIS

ネーミングサービスに登録するオブジェクトに対してコード系を設定します。

-n ODdemo::calculator

ネーミングサービスに登録するオブジェクト名を指定します。

Solaris32/64 Linux32/64

ネーミングサービスへの登録

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -L EUC -n ODdemo::calculator
```

-c IDL:ODdemo/calculator:1.0

指定されたインタフェースリポジトリIDでオブジェクトリファレンスを登録します。

-L EUC

ネーミングサービスに登録するオブジェクトに対してコード系を設定します。

-n ODdemo::calculator

ネーミングサービスに登録するオブジェクト名を指定します。

■サーバアプリケーションで作成する方法

以下の流れで処理を行います。

以下にプログラミングの例を示します。

```
CORBA_ORB              orb;         /* ORB_initで取得したObject */
CORBA_BOA              boa;         /* ORB_BOA_initで取得したObject */
CORBA_Repository       intf_rep;   /* インタフェースリポジトリのオブジェクトリファレンス */
CORBA_InterfaceDef     intf;        /* インタフェースリポジトリのInterfaceDef */
FJ_ImplementationRep   impl_rep;   /* インプリメンテーションリポジトリのオブジェクトリファレンス */
CORBA_ImplementationDef impl;       /* インプリメンテーションリポジトリのImplementationDef */
CORBA_Object           new_obj;     /* 作成したオブジェクトリファレンス */
CORBA_ReferenceData    id;          /* ReferenceData格納域 */
CosNaming_NamingContext cos_naming; /* NamingServiceのオブジェクトリファレンス */
CosNaming_Name         name;        /* 検索した情報格納域 */
CORBA_Environment      env;         /* 例外情報 */

/* ObjectDirectorの初期化（省略） */
      :

/* ReferenceDataの初期化 */
id._length  = 0;
id._maximum = 0;
id._buffer  = NULL;

/* インタフェースリポジトリのオブジェクトリファレンスの獲得 */
intf_rep = CORBA_ORB_resolve_initial_references(
        orb,
        CORBA_ORB_ObjectId_LightInterfaceRepository,
        &env );

/* InterfaceDefオブジェクトのオブジェクトリファレンスの獲得 */
intf = CORBA_Repository_lookup_id(
        intf_rep,
        "...",  /* インタフェースリポジトリID */
        &env );

/* インプリメンテーションリポジトリのオブジェクトリファレンスの獲得 */
impl_rep = CORBA_ORB_resolve_initial_references(
        orb,
        CORBA_ORB_ObjectId_ImplementationRepository,
        &env );

/* ImplementationDefオブジェクトのオブジェクトリファレンスの獲得 */
impl = FJ_ImplementationRep_lookup_id(
        impl_rep,
        "...",  /* インプリメンテーションリポジトリID */
        &env );

/* オブジェクトリファレンスの作成 */
new_obj = CORBA_BOA_create( boa, &id, intf, impl, &env );

/* NamingServiceのリファレンスを獲得 */
cos_naming = CORBA_ORB_resolve_initial_references(
    orb,
    CORBA_ORB_ObjectId_NameService,
    &env);

name._length = name._maximum = 1;          /* オブジェクト名の数 */
name._buffer = &name_component;            /* オブジェクト名格納域 */
name_component.id = "ODdemo::calculator";  /* オブジェクト名 */
name_component.kind = "";                  /* オブジェクトのタイプ */

/* ネーミングサービスへの登録 */
CosNaming_NamingContext_bind (cos_naming, &name, new_obj, &env );
```

注意

CORBA_BOA_create()呼出し時にインプリメンテーションリポジトリに定義されているデフォルトコード系でオブジェクトリファレンスを作成します。デフォルトコード系は、OD_impl_inst／OD_set_envコマンドで設定します。

3.5 クライアントアプリケーションのプログラミング（静的起動インタフェース）

静的起動インタフェースを使用する場合の、クライアントアプリケーションの処理の流れを以下に示します。

3.5.1 初期化

CORBAの初期化メソッドCORBA_ORB_init()を呼び出し、初期化処理を行います。結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
main( argc, argv )
    int    argc;
    char   *argv[];
{
    int                current_argc = argc;
    CORBA_ORB          orb;      /* ORBのオブジェクトリファレンス */
    CORBA_Environment  env;      /* 例外情報 */

    orb = CORBA_ORB_init( &current_argc, argv, FJ_OM_ORBid, &env );
```

3.5.2 ネーミングサービスのオブジェクトリファレンスの獲得

ネーミングサービスから実行するオブジェクトを検索するために、ネーミングサービスのオブジェクトリファレンスが必要となります。CORBAインタフェースのオブジェクトリファレンスの取出しメソッドCORBA_ORB_resolve_initial_references()によりネーミングサービスのオブジェクトリファレンスを取り出します。このとき、メソッドのパラメタとしてCORBA_ORB_ObjectId_NameServiceを指定します。

```
CosNaming_NamingContext  cos_naming;      /* NamingServiceのオブジェクトリファレンス */
CosNaming_Name           name;            /* 検索したい情報格納域 */
CosNaming_NameComponent  name_component;  /* ネームコンポーネント */

/* NamingServiceのオブジェクトリファレンスを獲得 */
cos_naming = CORBA_ORB_resolve_initial_references (
        orb,
```

```
        CORBA_ORB_ObjectId_NameService,
        &env);
```

3.5.3 サーバアプリケーションのオブジェクトリファレンスの獲得

ネーミングサービスのメソッドCosNaming_NamingContext_resolve()により、呼び出すサーバアプリケーションのオブジェクトリファレンスを取り出します。本メソッドのパラメタとして検索するオブジェクト名を指定します。

```
CORBA_Object        obj;                    /* サーバアプリケーションのオブジェクトリファレンス */

name._length = name._maximum = 1;           /* オブジェクト名の数 */
name._buffer = &name_component;             /* オブジェクト名格納域 */
name_component.id = "ODdemo::calculator";   /* オブジェクト名 */
name_component.kind = "";                   /* オブジェクトのタイプ */

/* サーバアプリケーションのオブジェクトリファレンスを獲得 */
obj = CosNaming_NamingContext_resolve( cos_naming, &name, &env );
```

3.5.4 メソッドの呼出し

サーバプログラムのメソッドを呼び出します。メソッド名は、IDLで指定したモジュール名、インタフェース名、およびオペレーション名を、アンダースコア(_)でつなげた形式で指定します。この例では、ODdemo、calculator、calculateがそれにあたります。なお、メソッド呼出し時にネーミングサービスで求めたサーバアプリケーションのオブジェクトリファレンスとサーバプログラムで例外が発生した場合に例外情報を獲得するためにCORBA_Environment構造体を指定します。
なお、アプリケーションで使用可能なCORBAのデータ型については、“3.7 データ型に対するマッピング”を参照してください。

```
CORBA_long                  a,b;    /* 入力パラメタ */
ODdemo_calculator_result    res;    /* 復帰値*/

a = 100; b = 20;

/* メソッドの呼出し */
res = ODdemo_calculator_calculate( obj, a, b, &env );
```

3.6 クライアントアプリケーションの例外処理

クライアントアプリケーションは、サーバアプリケーションの処理が正常終了したか異常終了したかを知ることができます。異常終了時は、システムの異常／サーバアプリケーションの異常かを知ることができ、前者をシステム例外、後者をユーザ例外と呼びます。
メソッド呼出し時に指定したCORBA_Environment構造体に、エラー情報が格納されます。
CORBA_Environment構造体を以下に示します。

```
typedef struct  {
    CORBA_enum  _major;
    CORBA_unsigned_long _minor;
        .
        .
} CORBA_Environment;
```

アプリケーションは、メソッドを呼び出す前にCORBA_Environment構造体の領域を初期化する必要があります。

```
CORBA_Environment env;

memset( &env, 0, sizeof(CORBA_Environment));
```

_majorには、以下の値が設定されます。

```
CORBA_NO_EXCEPTION      : 正常終了
CORBA_SYSTEM_EXCEPTION  : システム例外
CORBA_USER_EXCEPTION    : ユーザ例外
```

(1)システム例外

次に示すようなシステム例外の値が設定されます。システム例外で通知されるデータ型は、文字列です。CORBA_exception_id()でCORBA_Environment構造体から文字列を取り出します。
システム例外を以下に示します。例外の意味については、“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。

例外情報	例外コード
BAD_CONTEXT	ex_CORBA_StExcep_BAD_CONTEXT
BAD_INV_ORDER	ex_CORBA_StExcep_BAD_INV_ORDER
BAD_OPERATION	ex_CORBA_StExcep_BAD_OPERATION
BAD_PARAM	ex_CORBA_StExcep_BAD_PARAM
BAD_QOS	ex_CORBA_StExcep_BAD_QOS
BAD_TYPECODE	ex_CORBA_StExcep_BAD_TYPECODE
CODESET_INCOMPATIBLE	ex_CORBA_StExcep_CODESET_INCOMPATIBLE
COMM_FAILURE	ex_CORBA_StExcep_COMM_FAILURE
CONTEXT	ex_CORBA_StExcep_CONTEXT
DATA_CONVERSION	ex_CORBA_StExcep_DATA_CONVERSION
FREE_MEM	ex_CORBA_StExcep_FREE_MEM
IMP_LIMIT	ex_CORBA_StExcep_IMP_LIMIT
INITIALIZE	ex_CORBA_StExcep_INITIALIZE
INTERNAL	ex_CORBA_StExcep_INTERNAL
INTF_REPOS	ex_CORBA_StExcep_INTF_REPOS
INV_FLAG	ex_CORBA_StExcep_INV_FLAG
INV_IDENT	ex_CORBA_StExcep_INV_IDENT
INV_OBJREF	ex_CORBA_StExcep_INV_OBJREF
INV_POLICY	ex_CORBA_StExcep_INV_POLICY
MARSHAL	ex_CORBA_StExcep_MARSHAL
NO_IMPLEMENT	ex_CORBA_StExcep_NO_IMPLEMENT
NO_MEMORY	ex_CORBA_StExcep_NO_MEMORY
NO_PERMISSION	ex_CORBA_StExcep_NO_PERMISSION
NO_RESOURCES	ex_CORBA_StExcep_NO_RESOURCES
NO_RESPONSE	ex_CORBA_StExcep_NO_RESPONSE
OBJ_ADAPTER	ex_CORBA_StExcep_OBJ_ADAPTER
PERSIST_STORE	ex_CORBA_StExcep_PERSIST_STORE
REBIND	ex_CORBA_StExcep_REBIND
TIMEOUT	ex_CORBA_StExcep_TIMEOUT
TRANSIENT	ex_CORBA_StExcep_TRANSIENT

例外情報	例外コード
UNKNOWN	ex_CORBA_StExcep_UNKNOWN
INVALID_TRANSACTION	ex_CORBA_StExcep_INVALID_TRANSACTION
TRANSACTION_MODE	ex_CORBA_StExcep_TRANSACTION_MODE
TRANSACTION_REQUIRED	ex_CORBA_StExcep_TRANSACTION_REQUIRED
TRANSACTION_ROLLEDBACK	ex_CORBA_StExcep_TRANSACTION_ROLLEDBACK
TRANSACTION_UNAVAILABLE	ex_CORBA_StExcep_TRANSACTION_UNAVAILABLE

システム例外発生時には、CORBA_Environment構造体の_minorにマイナーコードが設定されます。マイナーコードの値については、“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。

(2)ユーザ例外

ユーザ例外でも文字列でエラー情報が通知されるため、CORBA_exception_id()でCORBA_Environment構造体から取り出します。また、IDLで定義したユーザ定義情報の詳細については、CORBA_exception_value()でCORBA_Environment構造体から取り出します。

(3)例外情報の獲得

例外情報を獲得する例を以下に示します。

```
/* メソッドの呼出し */
ret = ODdemo_calculator_calculate(obj, a, b, &env);

switch(env._major){
    case CORBA_SYSTEM_EXCEPTION:
        /* システム例外のエラー処理 */
        id = CORBA_exception_id(&env);

        if ( strcmp (id, ex_CORBA_StExcep_COMM_FAILURE) == 0 ) {
          /* マイナーコードの獲得 */
          fprintf( stderr, "SystemException:COMM_FAILURE (minor = %x)¥n", env._minor);
             .
             .
        }
             .
             .
    case CORBA_USER_EXCEPTION:
        /* ユーザ例外のエラー処理 */
        id = CORBA_exception_id(&env);

        if ( strcmp (id, ex_exp_exc) == 0 ) {
            uexc  =  (exp_exc *)CORBA_exception_value(&env);
             .
             .
        }
             .
             .
```

3.7 データ型に対するマッピング

ここでは、クライアント／サーバアプリケーションをC言語で作成する場合に使用するデータ型について説明します。クライアント／サーバアプリケーション間の関数呼出しで使用するパラメタのデータ型については、“3.7.10 パラメタ受渡しで使用するデータ型”で説明します。

3.7.1 基本データ型

CORBAで定義されている基本データ型とC言語でのデータ型の対応を以下に示します。

CORBAデータ型		C言語での定義	備考
整数型	long	CORBA_long	
	short	CORBA_short	
	unsigned long	CORBA_unsigned_long	
	unsigned short	CORBA_unsigned_short	
	long long	CORBA_long_long	
浮動小数点型	float	CORBA_float	
	double	CORBA_double	
	long double	CORBA_long_double	Windows32/64 Solaris32/64
文字型	char	CORBA_char	
	wchar	CORBA_wchar	
オクテッド型	octet	CORBA_octet	
ブーリアン型	boolean	CORBA_boolean	
文字列型	string	CORBA_string	“3.7.2 文字列型”参照
	wstring	CORBA_wstring	“3.7.3 ワイド文字列型”参照
列挙型	enum	CORBA_enum	
any型	any	CORBA_any	“3.7.4 any型”参照
オブジェクトリファレンス	Object	CORBA_Object	
タイプコード	TypeCode	CORBA_TypeCode	

3.7.2 文字列型

(1)IDLマッピング

IDL言語で文字列型stringを指定した場合、C言語ではCORBA_stringにマッピングされます。CORBA_stringは以下のように定義されます。

```
typedef  char  *CORBA_string;    /* stringの定義*/
```

以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface  stringtest{
       string op1(in string str1, out string str2, inout string str3);
    };
};
```

C言語

```
CORBA_string
ODsample_stringtest_op1(
    ODsample_stringtest obj;   /* オブジェクトリファレンス */
    CORBA_string str1,         /* inパラメタ */
```

```
    CORBA_string *str2,         /* outパラメタ */
    CORBA_string *str3,         /* inoutパラメタ */
    CORBA_Environment *env )    /* 例外情報 */
```

(2)クライアントアプリケーションで扱うパラメタ

クライアントアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	CORBA_string_alloc()で領域を獲得します。	－
inout	(inパラメタと同じ)	領域はスタブで自動的に獲得されます。
out 復帰	－	(inoutパラメタと同じ)

注意

クライアントおよびスタブで獲得した領域は、不要になった時点でCORBA_free()で解放する必要があります。

クライアントアプリケーションでの処理例を以下に示します。

```
CORBA_string          str1, str2, str3, ret;
CORBA_Environment     env;

/* inパラメタ */
str1 = CORBA_string_alloc(2);   /* 領域獲得 */
strcpy( str1 , "IN" );          /* パラメタ設定 */

/* inoutパラメタ */
str3 = CORBA_string_alloc(7);   /* 領域獲得 */
strcpy( str3 , "INOUT:1" );     /* パラメタ設定 */

/* サーバ呼出し */
ret = ODsample_stringtest_op1(obj, str1, &str2, &str3, &env );

/* 領域解放 */
CORBA_free( str1 );             /* inパラメタ */
CORBA_free( str2 );             /* outパラメタ */
CORBA_free( str3 );             /* inoutパラメタ */
CORBA_free( ret );              /* 復帰パラメタ */
```

(3)サーバアプリケーションで扱うパラメタ

サーバアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
in	文字列域は、スケルトンで自動的に獲得/解放されます。	－
inout	文字列域は、スケルトンで自動的に獲得されます。	・渡されたパラメタより短い文字列を返す場合 渡された文字列域に文字列を設定します。 ・渡されたパラメタより長い文字列を返す場合 渡された文字列域をCORBA_free()で一度解放し、CORBA_string_alloc()で再獲得します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
		文字列域は、スケルトンで自動的に解放されます。
out 復帰	―	文字列域をCORBA_string_alloc()で獲得します。文字列域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
CORBA_string
ODsample_stringtest_op1(
    ODsample_stringtest obj;          /* オブジェクトリファレンス */
    CORBA_string str1,                /* inパラメタ */
    CORBA_string *str2,               /* outパラメタ */
    CORBA_string *str3,               /* inoutパラメタ */
    CORBA_Environment *env )          /* 例外情報 */
{
    CORBA_string str;

/* outパラメタ */
    *str2 = CORBA_string_alloc(3);    /* 領域獲得 */
    strcpy( *str2, "OUT" );           /* パラメタ設定 */

/* inoutパラメタ */
    CORBA_free( *str3 );              /* 領域解放 */
    *str3 = CORBA_string_alloc(7);    /* 領域再獲得 */
    strcpy( *str3, "INOUT:2" );       /* パラメタ設定 */

/* 復帰パラメタ */
    str = CORBA_string_alloc(6);      /* 領域獲得 */
    strcpy( str, "RETURN" );          /* 復帰パラメタ設定 */
    return( str );
}
```

3.7.3 ワイド文字列型

(1)IDLマッピング

IDL言語で文字列型wstringを指定した場合、C言語ではCORBA_wstringにマッピングされます。CORBA_wstringは以下のように定義されています。

```
typedef CORBA_wchar    *CORBA_wstring;
```

以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface   wstringtest{
        wstring op1(in wstring str1, out wstring str2, inout wstring str3);
    };
};
```

C言語

```
CORBA_wstring
ODsample_wstringtest_op1(
    ODsample_wstringtest obj;  /* オブジェクトリファレンス */
    CORBA_wstring str1,        /* inパラメタ */
```

```
    CORBA_wstring *str2,        /* outパラメタ */
    CORBA_wstring *str3,        /* inoutパラメタ */
    CORBA_Environment *env )    /* 例外情報 */
```

(2)クライアントアプリケーションで扱うパラメタ

クライアントアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	CORBA_wstring_alloc()で領域を獲得します。	－
inout	(inパラメタと同じ)	領域はスタブで自動的に獲得されます。
out 復帰	－	(inoutパラメタと同じ)

クライアントおよびスタブで獲得した領域は、不要になった時点でCORBA_free()で解放する必要があります。

クライアントアプリケーションでの処理例を以下に示します。

```
CORBA_wstring          str1, str2, str3, ret;
CORBA_Environment      env;

/* inパラメタ */
str1 = CORBA_wstring_alloc(2);  /* 領域獲得 */
str1[0] = ...;                  /* パラメタ設定(“(4)文字列の設定方法”参照) */
    :

/* inoutパラメタ */
str3 = CORBA_wstring_alloc(5);  /* 領域獲得 */
str3[0] = ...;                  /* パラメタ設定(“(4)文字列の設定方法”参照) */
    :

/* サーバ呼出し */
ret = ODsample_wstringtest_op1(obj, str1, &str2, &str3, &env );

/* 領域解放 */
CORBA_free( str1 );             /* inパラメタ */
CORBA_free( str2 );             /* outパラメタ */
CORBA_free( str3 );             /* inoutパラメタ */
CORBA_free( ret );              /* 復帰パラメタ */
```

(3)サーバアプリケーションで扱うパラメタ

サーバアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
in	文字列域は、スケルトンで自動的に獲得/解放されます。	－
inout	文字列域は、スケルトンで自動的に獲得されます。	・ 渡されたパラメタより短い文字列を返す場合 渡された文字列域に文字列を設定します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
		・ 渡されたパラメタより長い文字列を返す場合 渡された文字列域をCORBA_free()で一度解放し、CORBA_wstring_alloc()で再獲得します。 文字列域は、スケルトンで自動的に解放されます。
out 復帰	―	文字列域をCORBA_wstring_alloc()で獲得します。文字列域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
CORBA_wstring
ODsample_wstringtest_op1(
    ODsample_stringtest  obj;   /* オブジェクトリファレンス */
    CORBA_wstring     wstr1,    /* inパラメタ */
    CORBA_wstring     *wstr2,   /* outパラメタ */
    CORBA_wstring     *wstr3,   /* inoutパラメタ */
    CORBA_Environment *env )    /* 例外情報 */
{
    CORBA_wstring    wstr;

/* outパラメタ */
    *wstr2 = CORBA_wstring_alloc(3);   /* 領域獲得 */
    (*wstr2)[0] = ...;                 /* パラメタ設定：(4)参照 */
      :

/* inoutパラメタ */
    CORBA_free( *wstr3 );              /* 領域解放 */
    *wstr3 = CORBA_wstring_alloc(5);   /* 領域再獲得 */
    (*wstr3)[0] = ...;                 /* パラメタ設定：(4)参照 */
      :

/* 復帰パラメタ */
    wstr = CORBA_wstring_alloc(4);     /* 領域獲得 */
    wstr[0] = ...;                     /* パラメタ設定：(4)参照 */
      :
    return( wstr );
}
```

(4)文字列の設定方法

ワイド文字列型への文字列の設定方法を以下に示します。変数wstrは、CORBA_wchar*型です。

EUCまたはShiftJISを使用する場合

文字を1文字ずつ設定します(EUCの場合)。

```
wstr[0] = 0xc6fc;  /*  "日"    */
wstr[1] = 0xcbdc;  /*  "本"    */
wstr[2] = 0xb8ec;  /*  "語"    */
wstr[3] = 0x0000;  /*  終端子  */
```

Windows(R)でUNICODEを使用する場合

""で囲んだ文字列にL修飾子を付けると、UNICODEになります。

```
wstr = L"日本語";
```

Solaris/LinuxでUNICODEを使用する場合 Solaris32/64 Linux32/64

UNICODEの値を1文字ずつ直接数値で設定します。

```
wstr[0] = 0x65e5;  /*  "日"    */
wstr[1] = 0x672c;  /*  "本"    */
wstr[2] = 0x8a9e;  /*  "語"    */
wstr[3] = 0x0000;  /*  終端子  */
```

ワイド文字列型をコンソールなどに出力する場合、出力処理は各OSに依存するため、そのままでは正しい文字が出力できないことがあります。出力の方法については、各OSのマニュアルを参照してください。

Solaris32/64 Linux32/64

L"..."の形式の文字列を処理することはできません。

3.7.4 any型

(1)IDLマッピング

IDL言語でany型を指定した場合、C言語ではCORBA_any構造体にマッピングされます。また、データ域(_value)の獲得関数(関数名:“モジュール名_インタフェース名_構造体名_alloc”。以降、XX_alloc関数と呼ぶ)が生成されます。
CORBA_any構造体の構造を以下に示します。

```
typedef struct any {
    CORBA_TypeCode    _type;      /* データ型識別情報(TypeCode) */
    void              *_value;    /* データ域 */
} CORBA_any;
```

以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    struct sample1 {
        long    para1;
        string  para2;
    };
    struct sample2 {
        char    para1;
        float   para2;
    };
    struct sample3 {
        char    para1;
        double  para2;
    };
    interface   anytest{
        any     op1(in any any1, out any any2, inout any any3 );
    };
};
```

(2)クライアントアプリケーションで扱うパラメタ

クライアントアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	領域(CORBA_any構造体/_value域)を動的に獲得する場合は、CORBA_any_alloc()/XX_alloc関数を使用します。	―
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。
out 復帰	―	(inoutパラメタと同じ)

注意

クライアントおよびスタブで獲得した領域は、不要になった時点でCORBA_free()で解放する必要があります。このとき、_valueの指す領域を解放するかどうかを、リリースフラグで指定できます。
リリースフラグの参照／設定は、以下の関数／フラグを使用します。

[関数]

```
CORBA_boolean CORBA_any_get_release( CORBA_any * );
        void          CORBA_any_set_release( CORBA_any *, CORBA_boolean );
```

[フラグ]

- CORBA_TRUE
  CORBA_free()発行時、_valueの指す領域も解放されます。
- CORBA_FALSE
  CORBA_free()発行時、_valueの指す領域は解放されません(デフォルト)・

なお、スタブで獲得されたパラメタ(out,復帰)のリリースフラグは、CORBA_TRUEに設定されます。

クライアントアプリケーションの処理例を以下に示します。

```
CORBA_any            *any1, *any2, *any3, *ret;
CORBA_Object         obj;
CORBA_Environment    env;
ODsample_sample1     *smp1;
ODsample_sample2     *smp2;
ODsample_sample3     *smp3;

/* inパラメタ */
any1 = CORBA_any_alloc();              /* 領域獲得 */
any1->_type = TC_ODsample_sample3;     /* TypeCode設定 */
smp3 = any1->_value = ODsample_sample3_alloc();
smp3->para1 = 'a';
smp3->para2 = 0.00001;

/* inoutパラメタ */
any3 = CORBA_any_alloc();              /* 領域獲得 */
any3->_type = TC_ODsample_sample2;     /* TypeCode設定 */
smp2 = any3->_value = ODsample_sample2_alloc();
smp2->para1 = 'c';
smp2->para2 = 0.0001;

/* サーバ呼出し */
ret = ODsample_anytest_op1( obj, any1, &any2, any3, &env );

/* 領域解放 */
CORBA_free( any1 );             /* inパラメタ */
CORBA_free( any2 );             /* outパラメタ */
```

```
CORBA_free( any3 );             /* inoutパラメタ */
CORBA_free( ret );              /* 復帰パラメタ */
```

(3)サーバアプリケーションで扱うパラメタ

サーバアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
in	各領域(CORBA_any構造体/データ域)は、スケルトンで自動的に獲得/解放されます。	－
inout	各領域(CORBA_any構造体/データ域)は、スケルトンで自動的に獲得されます。	データ域(_value)をCORBA_free()で一度解放し、XX_alloc関数で再獲得します。 領域は、スケルトンで自動的に解放されます。
out 復帰	－	CORBA_any構造体/データ域(_value)をCORBA_any_alloc()/XX_alloc関数で獲得します。 領域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
CORBA_any
*ODsample_anytest_op1(
    ODsample_anytest   obj,
    CORBA_any          *any1,
    CORBA_any          **any2,
    CORBA_any          *any3,
    CORBA_Environment  *env )
{
    CORBA_any  *p;
    ODsample_sample1   *smp1;
    ODsample_sample2   *smp2;
    ODsample_sample3   *smp3;

/* outパラメタ */
    p = *any2 = CORBA_any_alloc();                  /* 領域獲得 */
    p->_type = TC_ODsample_sample2;                 /* TypeCode設定 */
    smp2 = p->_value = ODsample_sample2_alloc();    /* データ域獲得 */
    smp2->para1 = 'x';                              /* パラメタ設定 */
    smp2->para2 = 0.001;
    CORBA_any_set_release( p, CORBA_TRUE );         /* リリースフラグ設定 */

/* inoutパラメタ */
    any3->_type = TC_ODsample_sample3;              /* TypeCodeの設定 */
    CORBA_free( any3->_value );                     /* 領域解放(データ域) */
    smp3 = any3->_value = ODsample_sample3_alloc(); /* 領域再獲得(データ域) */
    smp3->para1 = 'y';                              /* パラメタ設定 */
    smp3->para2 = 0.0001;

/* 復帰パラメタ */
    p = CORBA_any_alloc();                          /* 領域獲得 */
    p->_type = TC_ODsample_sample1;                 /* TypeCode設定 */
    smp1 = p->_value = ODsample_sample1_alloc();    /* データ域獲得 */
    smp1->para1 = 300;                              /* パラメタ設定 */
    smp1->para2 = CORBA_string_alloc(4);
    strcpy( smp1->para2, "test" );
    CORBA_any_set_release( p, CORBA_TRUE );         /* リリースフラグ設定 */
    return( p );
}
```

3.7.5 シーケンス型

(1)IDLマッピング

IDL言語でシーケンス型sequenceを指定した場合、C言語では以下の構造体(シーケンス構造体)にマッピングされます。また、シーケンス構造体の領域獲得関数(関数名:“モジュール名_インタフェース名_構造体名_alloc”。以降XX_alloc関数と呼ぶ)と、データ域(_buffer)の領域獲得関数(関数名:“モジュール名_インタフェース名_構造体名_allocbuf”。以降XX_allocbuf関数と呼ぶ)が生成されます。
シーケンス構造体の構造を以下に示します。

```
struct
{
    CORBA_unsigned_long    _maximum;    /* シーケンス配列の最大個数 */
    CORBA_unsigned_long    _length;     /* シーケンス配列の使用個数 */
    void                   *_buffer;    /* データ域 */
};
```

以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface  seqtest{
        typedef sequence<long>  sampleseq;
        sampleseq op1(in sampleseq seq1, out sampleseq seq2, inout sampleseq seq3 );
    };
};
```

C言語

```
typedef struct
{
    CORBA_unsigned_long    _maximum;
    CORBA_unsigned_long    _length;
    CORBA_long             *_buffer;
} sampleseq;
```

(2)クライアントアプリケーションで扱うパラメタ

クライアントアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	シーケンス構造体/データ域(_buffer)を獲得する場合は、XX_alloc関数/XX_allocbuf関数を使用します。 メンバにシーケンス配列の最大個数/使用個数/データ域アドレスを設定します。	－
inout	シーケンス構造体/データ域(_buffer)を獲得する場合は、XX_alloc関数/XX_allocbuf関数を使用します。 メンバにシーケンス配列の最大個数/使用個数/データ域アドレスを設定します。 リリースフラグにCORBA_TRUEが設定されていると、データ部の領域はスタブにより解放されます。 リリースフラグにCORBA_TRUEが設定されていないと、スタブではデータ部の領域を解放しません。この場合、アプリケーション内で領域を保持し、不要になった段階でデータ部の解放処理を行ってください。	領域は、スタブで自動的に獲得されます(各メンバも設定されます)。 リリースフラグは、サーバへ渡すパラメタに設定された値から変化しません。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
out 復帰	―	領域は、スタブで自動的に獲得されます(各メンバも設定されます)。このとき、リリースフラグにCORBA_TRUEが設定されます。 (inoutパラメタと同じ)

注意

クライアントおよびスタブで獲得した領域は、不要になった時点でCORBA_free()で解放する必要があります。このとき、データ域の領域(XX_allocbuf関数で獲得)を解放するかどうかを、リリースフラグで指定します。

リリースフラグの参照／設定は、以下の関数／フラグを使用します。

[関数]

```
CORBA_boolean CORBA_sequence_get_release( void * );
      void          CORBA_sequence_set_release( void *, CORBA_boolean );
```

[フラグ]

- CORBA_TRUE
  CORBA_free()発行時、シーケンスデータ領域も解放されます。
- CORBA_FALSE
  CORBA_free()発行時、シーケンスデータ領域は解放されません(デフォルト)。

なお、スタブで獲得されたパラメタ(out,復帰)のリリースフラグは、CORBA_TRUEに設定されます。
ただし、シーケンス型／any型のsequenceをin/inoutに使用する場合は、XX_allocbuf関数で獲得したシーケンス型／any型の領域に対して、リリースフラグにCORBA_TRUEを設定してください。また、out/inout/復帰値に使用する場合、シーケンス型の_bufferの型がシーケンス型／any型の場合、かつ_bufferがNULLでない場合は、CORBA_free()で領域を解放する前に、_buffer領域に対してリリースフラグにCORBA_TRUEを設定してください。

クライアントアプリケーションでの処理例を以下に示します。

```
ODsample_seqtest_sampleseq  *seq1, *seq2, *seq3, *ret;
CORBA_Object         obj;
CORBA_Environment    env;
CORBA_long           *smp;
int                  i;

/* inパラメタ */
seq1 = ODsample_seqtest_sampleseq_alloc();   /* 領域獲得 */
seq1->_maximum = seq1->_length = 2;
seq1->_buffer = ODsample_seqtest_sampleseq_allocbuf(2);
for( i = 0; i < 2; i++ ){
    smp = &(seq1->_buffer[i]);
    *smp = i;
}
CORBA_sequence_set_release( seq1, CORBA_TRUE ) ;  /* リリースフラグ設定 */

/* inoutパラメタ */
seq3 = ODsample_seqtest_sampleseq_alloc();
seq3->_maximum = seq3->_length = 3;
seq3->_buffer = ODsample_seqtest_sampleseq_allocbuf(3);
for( i = 0; i < 3; i++ ){
    smp = &(seq3->_buffer[i]);
    *smp = i*10;
}
```

```
CORBA_sequence_set_release( seq3, CORBA_TRUE ) ;   /* リリースフラグ設定 */

/* サーバ呼出し */
ret = ODsample_seqtest_op1( obj, seq1, &seq2, seq3, &env );

/* 領域解放 */
CORBA_free( seq1 );             /* inパラメタ */
CORBA_free( seq2 );             /* outパラメタ */
CORBA_free( seq3 );             /* inoutパラメタ */
CORBA_free( ret );              /* 復帰パラメタ */
```

(3)サーバアプリケーションで扱うパラメタ

サーバアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
in	各領域(シーケンス構造体/データ域)は、スケルトンで自動的に獲得/解放されます。 データ域(_buffer)がシーケンス型／any型の場合、かつ_bufferがNULLでない場合は、_buffer領域に対して、リリースフラグにCORBA_TRUEを設定してください。	―
inout	各領域(シーケンス構造体/データ域)は、スケルトンで自動的に獲得されます。このとき、リリースフラグにCORBA_TRUEが設定されます。 データ域(_buffer)がシーケンス型／any型の場合、かつ_bufferがNULLでない場合は、inの領域を解放する前に、_buffer領域に対して、リリースフラグにCORBA_TRUEを設定してください。	inoutデータの領域を更新する場合は、以下の処理を行ってください。 CORBA_free()でデータ域(_buffer)を一度解放し、XX_allocbuf関数で再獲得します。 データ域(_buffer)がシーケンス型／any型の場合は、XX_allocbuf関数で獲得した領域に対してリリースフラグにCORBA_TRUEを設定してください。 リリースフラグにCORBA_TRUEを設定する、またはリリースフラグを設定しない場合、領域はスケルトンで自動的に解放されます。 リリースフラグにCORBA_FALSEを設定する場合、スケルトンではデータ域を解放しません。この場合、不要になった時点で領域の解放処理を行ってください。
out 復帰	―	XX_alloc関数/XX_allocbuf関数でシーケンス構造体/データ域(_buffer)を獲得します。 データ域(_buffer)がシーケンス型／any型の場合は、XX_allocbuf関数で獲得した領域に対してリリースフラグにCORBA_TRUEを設定してください。 リリースフラグにCORBA_TRUEを設定する場合、領域はスケルトンで自動的に解放されます。 リリースフラグにCORBA_FALSEを設定する、またはリリースフラグを設定しない場合、スケルトンではデータ域を解放しません。この場合、不要になった時点で領域の解放処理を行ってください。

サーバアプリケーションでの処理例を以下に示します。

```
ODsample_seqtest_sampleseq
*ODsample_seqtest_op1(
    ODsample_seqtest            obj,
    ODsample_seqtest_sampleseq  *seq1,
    ODsample_seqtest_sampleseq  **seq2,
```

```
    ODsample_seqtest_sampleseq  *seq3,
    CORBA_Environment  *env )
{
    ODsample_seqtest_sampleseq  *seq;
    CORBA_long                  *smp;
    int                         i;

/* outパラメタ */
    seq = *seq2 = ODsample_seqtest_sampleseq_alloc();       /* 領域獲得 */
    seq->_maximum = seq->_length = 4;                       /* メンバ設定 */
    seq->_buffer = ODsample_seqtest_sampleseq_allocbuf(4);  /* データ域獲得 */
    for( i = 0; i < 4; i++ ) {
        smp = &(seq->_buffer[i]);
        *smp = i*100;
    }
    CORBA_sequence_set_release( *seq2, CORBA_TRUE );         /* リリースフラグ設定 */

/* inoutパラメタ */
    seq3->_maximum = seq3->_length = 5;
    CORBA_free( seq3->_buffer );                            /* 領域解放(データ域) */
    seq3->_buffer = ODsample_seqtest_sampleseq_allocbuf(5); /* 領域再獲得(データ域) */
    for( i = 0; i < 5; i++ ) {                              /* パラメタ設定 */
        smp = &(seq3->_buffer[i]);
        *smp = i*1000;
    }
    CORBA_sequence_set_release( seq3, CORBA_TRUE );         /* リリースフラグ設定 */

/* 復帰パラメタ */
    seq = ODsample_seqtest_sampleseq_alloc();               /* 領域獲得 */
    seq->_maximum = seq->_length = 6;                       /* メンバ設定 */
    seq->_buffer = ODsample_seqtest_sampleseq_allocbuf(6);  /* データ域獲得 */
    for( i = 0; i < 6; i++ ) {
        smp = &(seq->_buffer[i]);
        *smp = i*10000;
    }
    CORBA_sequence_set_release( seq, CORBA_TRUE );          /* リリースフラグ設定 */
    return( seq );
}
```

3.7.6 構造体

(1)IDLマッピング

IDL言語で構造体structを指定した場合、C言語でもstructにマッピングされます。また、可変長の構造体については、領域獲得関数(関数名:“モジュール名_インタフェース名_構造体名_alloc”。以降XX_alloc関数と呼ぶ)が生成されます。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    struct samplefix {    /* 構造体(固定長) */
        long    para1;
        long    para2;
    };
    struct samplevar {    /* 構造体(可変長) */
        long    para1;
        string  para2;
    };

    interface  structtest{
        samplefix        op2(
```

```
            in samplefix str1,
            out samplefix str2,
            inout samplefix str3
        );
        samplevar         op1(
            in samplevar str1,
            out samplevar str2,
            inout samplevar str3
        );
    };
};
```

C言語

```
typedef struct ODsample_samplefix{    /* 構造体(固定長) */
    CORBA_long    para1;
    CORBA_long    para2;
} ODsample_samplefix;

typedef struct ODsample_samplevar{    /* 構造体(可変長) */
    CORBA_long    para1;
    CORBA_char    *para2;
} ODsample_samplevar;
```

(2)クライアントアプリケーションで扱うパラメタ(固定長)

構造体(固定長)のパラメタ(in、out、inout)を扱う場合、構造体変数をパラメタに指定します。領域の獲得／解放を行う必要はありません。

```
ODsample_samplefix  fix1, fix2, fix3, ret;
CORBA_Environment   env;
CORBA_Object        obj;

/* inパラメタ */
fix1.para1 = 10;
fix1.para2 = 11;

/* inoutパラメタ */
fix3.para1 = 20;
fix3.para2 = 21;

/* サーバ呼出し */
ret = ODsample_structtest_op2( obj, &fix1, &fix2, &fix3, &env );
```

(3)クライアントアプリケーションで扱うパラメタ(可変長)

クライアントアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	構造体領域/可変長データ域を獲得する場合、 XX_alloc関数/データ域獲得関数を使用します。	－
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。
out 復帰	－	(inoutパラメタと同じ)

注意

クライアントおよびスタブで動的に獲得した領域は、不要になった時点でCORBA_free()で解放する必要があります。CORBA_free()を発行することで可変長データ域も解放されます。

クライアントアプリケーションでの処理例を以下に示します。

```
ODsample_samplevar   *var1, *var2, *var3, *ret;
CORBA_Object         obj;
CORBA_Environment    env;

/* inパラメタ */
var1 = ODsample_samplevar_alloc();          /* 領域獲得 */
var1->para1 = 5;                            /* パラメタ設定 */
var1->para2 = CORBA_string_alloc(4);
strcpy( var1->para2, "test" );

/* inoutパラメタ */
var3 = ODsample_samplevar_alloc();          /* 領域獲得 */
var3->para1 = 4;                            /* パラメタ設定 */
var3->para2 = CORBA_string_alloc(3);
strcpy( var3->para2, "pro" );

/* サーバ呼出し */
ret = ODsample_structtest_op1( obj, var1, &var2, var3, &env );

/* 領域解放 */
CORBA_free( var1 );                         /* inパラメタ */
CORBA_free( var2 );                         /* outパラメタ */
CORBA_free( var3 );                         /* inoutパラメタ */
CORBA_free( ret );                          /* 復帰パラメタ */
```

(4)サーバアプリケーションで扱うパラメタ(固定長)

構造体(固定長)のパラメタ(in、out、inout)を扱う場合、構造体領域はスケルトンで自動的に獲得／解放されます。
サーバアプリケーションでの処理例を以下に示します。

```
ODsample_samplefix
ODsample_structtest_op2(
    CORBA_Object        obj,
    ODsample_samplefix  *str1,
    ODsample_samplefix  *str2,
    ODsample_samplefix  *str3,
    CORBA_Environment   *env )
{
    ODsample_samplefix  fix;

    /* outパラメタ */
    str2->para1 = 0;
    str2->para2 = 1;

    /* inoutパラメタ */
    str3->para1 = 2;
    str3->para2 = 3;

    /* 復帰パラメタ */
    fix.para1 = 4;
    fix.para2 = 5;
    return( fix );
}
```

(5)サーバアプリケーションで扱うパラメタ(可変長)

サーバアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	クライアントからのパラメタ領域	クライアントへのパラメタ領域
in	構造体領域/可変長データ域は、スケルトンで自動的に獲得/解放されます。	―
inout	構造体領域/可変長データ域は、スケルトンで自動的に獲得されます。	CORBA_free()で可変長データ域を一度解放し、データ域獲得関数で再獲得します。 領域は、スケルトンで自動的に解放されます。
out 復帰	―	XX_alloc関数/データ域獲得関数で構造体/可変長データ域を獲得します。 領域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
ODsample_samplevar
*ODsample_structtest_op1(
    ODsample_structtest  obj,
    ODsample_samplevar   *str1,
    ODsample_samplevar   **str2,
    ODsample_samplevar   *str3,
    CORBA_Environment    *env )
{
    ODsample_samplevar  *str;

    print_strvar( str1 );
    print_strvar( str3 );

    /* outパラメタ */
    *str2 = ODsample_samplevar_alloc();          /* 領域獲得 */
    (*str2)->para1 = 12;                         /* メンバ設定 */
    (*str2)->para2 = CORBA_string_alloc( 16 );
    strcpy( (*str2)->para2, "(*str2)->para2" );

    /* inoutパラメタ */
    str3->para1 = 12;                            /* メンバ設定 */
    CORBA_free( str3->para2 );                   /* 領域解放(メンバ域) */
    str3->para2 = CORBA_string_alloc(16);        /* 領域再獲得(メンバ域) */
    strcpy( str3->para2, "str3->para2" );

    /* 復帰パラメタ */
    str = ODsample_samplevar_alloc();            /* 領域獲得 */
    str->para1 = 11;                             /* メンバ設定 */
    str->para2 = CORBA_string_alloc(16);
    strcpy( str->para2, "str->para2" );
    return( str );
}
```

3.7.7 共用体

(1)IDLマッピング

IDL言語で共用体unionを指定すると、C言語ではデータ型を識別する弁別情報_dと共用体データ域_uから構成される構造体(共用型構造体)にマッピングされます。また、共用型構造体の領域獲得関数(関数名:"モジュール名_インタフェース名_共用体名_alloc"。以降XX_alloc関数と呼ぶ)が生成されます。
共用型構造体の構造を以下に示します。

```
struct {                          /* 共用型構造体 */
    CORBA_long  _d;               /* データ型弁別情報 */
    union {                       /* 共用体データ域 */
        CORBA_long    xxx;
         :
    } _u;
} ;
```

以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{

    union samplefix switch(long){    /* 共用体(固定長) */
        case 1: long    para1;
        case 2: long    para2;
    };
    union samplevar switch(long){    /* 共用体(可変長) */
        case 1: long    para1;
        case 2: string  para2;
    };
    interface  uniontest{
        samplefix  op2(
                        in samplefix uni1,
                        out samplefix uni2,
                        inout samplefix uni3
                    );
        samplevar  op1(
                        in samplevar uni1,
                        out samplevar uni2,
                        inout samplevar uni3
                    );
    };
};
```

C言語

```
typedef struct ODsample_samplefix {   /* 共用型構造体(固定長) */
    CORBA_long  _d;
    union {
        CORBA_long    para1;
        CORBA_long    para2;
    } _u;
} ODsample_samplefix;

typedef struct ODsample_samplevar{    /* 共用型構造体(可変長) */
    CORBA_long  _d;
    union {
        CORBA_long    para1;
        CORBA_char    *para2;
    } _u;
} ODsample_samplevar;
```

(2)クライアントアプリケーションで扱うパラメタ(固定長)

共用型構造体(固定長)のパラメタ(in、out、inout)を扱う場合、構造体変数をパラメタとして渡します。このとき、弁別情報にデータ型識別値、データ域にデータを設定します。
クライアントアプリケーションでの処理例を以下に示します。

```
CORBA_Object         obj;
CORBA_Environment    env;
ODsample_samplefix   unif1, unif2, unif3, ref;

/* inパラメタ */
unif1._d = 2;          /* 弁別子設定 */
unif1._u.para2 = 'z';  /* パラメタ値設定 */

/* inoutパラメタ */
unif3._d = 2;          /* 弁別子設定 */
unif3._u.para2 = 'y';  /* パラメタ値設定 */

/* サーバ呼出し */
ref = ODsample_uniontest_op2( obj, &unif1, &unif2, &unif3, &env );
```

(3)クライアントアプリケーションで扱うパラメタ(可変長)

クライアントアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	構造体領域/可変長データ域を獲得する場合、XX_alloc関数/データ領域獲得関数を使用します。 データ型識別値とデータを設定します。	―
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。
out 復帰	―	(inoutパラメタと同じ)

注意

クライアントおよびスタブで動的に獲得した領域は、不要になった時点でCORBA_free()で解放する必要があります。CORBA_free()を発行することで可変長データ域も解放されます。

クライアントアプリケーションでの処理例を以下に示します。

```
ODsample_samplevar   *uni1, *uni2, *uni3, *ret;
CORBA_Object         obj;
CORBA_Environment    env;

/* inパラメタ */
uni1 = ODsample_samplevar_alloc();            /* 領域獲得 */
uni1->_d = 1;                                 /* 弁別子設定 */
uni1->_u.para1 = 10;                          /* パラメタ設定 */

/* inoutパラメタ */
uni3 = ODsample_samplevar_alloc();            /* 領域獲得 */
uni3->_d = 2;                                 /* 弁別子設定 */
uni3->_u.para2 = CORBA_string_alloc(11);      /* パラメタ設定 */
strcpy( uni3->_u.para2, "INOUT:para2" );

/* サーバ呼出し */
ret = ODsample_uniontest_op1( obj, uni1, &uni2, uni3, &env );
```

```
/* 領域解放 */
CORBA_free( uni1 );                         /* inパラメタ */
CORBA_free( uni2 );                         /* outパラメタ */
CORBA_free( uni3 );                         /* inoutパラメタ */
CORBA_free( ret );                          /* 復帰パラメタ */
```

(4)サーバアプリケーションで扱うパラメタ(固定長)

共用型構造体(固定長)のパラメタ(in,out,inout)を扱う場合、構造体領域はスケルトンで自動的に獲得／解放されます。
サーバアプリケーションでの処理例を以下に示します。

```
ODsample_samplefix
ODsample_uniontest_op2(
    CORBA_Object          obj,
    ODsample_samplefix    *uni1,
    ODsample_samplefix    *uni2,
    ODsample_samplefix    *uni3,
    CORBA_Environment     *env )
{
    ODsample_samplefix    uni;

    /* outパラメタ */
    uni2->_d = 1;           /* 弁別情報設定 */
    uni2->_u.para1 = 10;    /* データ設定 */

    /* inoutパラメタ */
    uni3->_d = 2;           /* 弁別情報設定 */
    uni3->_u.para2 = 'c';   /* データ設定 */

    /* 復帰パラメタ */
    uni._d = 1;             /* 弁別情報設定 */
    uni._u.para1 = 100;     /* データ設定 */
    return( uni );
}
```

(5)サーバアプリケーションで扱うパラメタ(可変長)

サーバアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	クライアントからのパラメタ領域	クライアントへのパラメタ領域
in	構造体領域/可変長データ域は、スケルトンで自動的に獲得/解放されます。	－
inout	構造体領域/可変長データ域は、スケルトンで自動的に獲得されます。	CORBA_free()で可変長データ域を一度解放し、データ域獲得関数で再獲得します。 領域は、スケルトンで自動的に解放されます。
out 復帰	－	XX_alloc関数/データ域獲得関数で構造体/可変長データ域を獲得します。 領域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
ODsample_samplevar
*ODsample_uniontest_op1(
    ODsample_uniontest    obj,
    ODsample_samplevar    *uni1,
    ODsample_samplevar    **uni2,
```

```
    ODsample_samplevar    *uni3,
    CORBA_Environment     *env )
{
    ODsample_samplevar    *uni;

    /* outパラメタ */
    *uni2 = ODsample_samplevar_alloc();          /* 領域獲得 */
    (*uni2)->_d = 2;                             /* 弁別情報設定 */
    (*uni2)->_u.para2 = CORBA_string_alloc(9);   /* データ設定 */
    strcpy( (*uni2)->_u.para2, "OUT:param" );

    /* inoutパラメタ */
    if( uni3->_d == 2 )                          /* 弁別情報 */
        CORBA_free( uni3->_u.para2 );
    uni3->_d = 1;                                /* 弁別情報設定 */
    uni3->_u.para1 = 10;                         /* データ設定 */

    /* 復帰パラメタ */
    uni = ODsample_samplevar_alloc();            /* 領域獲得 */
    uni->_d = 1;                                 /* 弁別情報設定 */
    uni->_u.para1 = 30;                          /* データ設定 */
    return( uni );
}
```

3.7.8 配列

(1)IDLマッピング

IDL言語で配列を指定すると、C言語でも配列にマッピングされます。また、配列スライス(配列の先頭の次元を減らしたもの。名前は“モジュール名_インタフェース名_配列名_slice”)と配列スライスの領域獲得関数(関数名:“モジュール名_インタフェース名_配列名_alloc”。以降XX_alloc関数と呼ぶ)が生成されます。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface  arraytest{
        typedef long    fix[4][3][2];         /* 配列(固定長) */
        typedef string  str[2][3][4];         /* 配列(可変長) */
        fix    op1(in fix para1, out fix para2, inout fix para3 );
        str    op2(in str para1, out str para2, inout str para3 );
    };
};
```

C言語

```
typedef CORBA_long  ODsample_arraytest_fix[4][3][2];      /* 配列(固定長) */
typedef CORBA_long  ODsample_arraytest_fix_slice[3][2];   /* 配列スライス */

typedef CORBA_char  *ODsample_arraytest_str[2][3][4];     /* 配列(可変長) */
typedef CORBA_char  *ODsample_arraytest_str_slice[3][4];  /* 配列スライス */
```

(2)クライアントアプリケーションで扱うパラメタ(固定長)

配列(固定長)のパラメタ(in、out、inout)を扱う場合、宣言した配列変数をパラメタに指定します。復帰パラメタは、スタブで領域獲得された配列スライスで返されます。この領域は、アプリケーションで不要になった時点でCORBA_free()で解放する必要があります。
クライアントアプリケーションでの処理例を以下に示します。

```
ODsample_arraytest_fix        fix1, fix2, fix3;
ODsample_arraytest_fix_slice  *ret;
CORBA_Object         obj;
CORBA_Environment    env;
int                  i, j, k;

for( i = 0; i < 4; i++ )
for( j = 0; j < 3; j++ )
for( k = 0; k < 2; k++ ){
    fix1[i][j][k] = i;        /* inパラメタ */
    fix3[i][j][k] = i*10;     /* inoutパラメタ */
}

/* サーバ呼出し */
ret = ODsample_arraytest_op1( obj, fix1, fix2, fix3, &env );

/* 領域解放 */
CORBA_free( ret );           /* 復帰パラメタ */
```

(3)クライアントアプリケーションで扱うパラメタ(可変長)

配列(可変長)のin／inoutパラメタを扱う場合、配列変数をパラメタで指定します。out／復帰パラメタは、スタブで領域獲得された配列スライスに返されます。この領域は、アプリケーションで不要になった時点でCORBA_free()で解放する必要があります。

```
ODsample_arraytest_str          str1, str3;
ODsample_arraytest_str_slice    *str2, *ret;
CORBA_Object         obj;
CORBA_Environment    env;
int                  i, j, k;
CORBA_string         buf[];

for( i = 0; i < 2; i++ )
for( j = 0; j < 3; j++ )
for( k = 0; k < 4; k++ ) {

    /* inパラメタ */
    sprintf( buf, "str1[i][j][k]", i, j, k );
    str1[i][j][k] = CORBA_string_alloc(strlen(buf));
    strcpy( str1[i][j][k], buf );

    /* inoutパラメタ */
    sprintf( buf, "str3[i][j][k]", i, j, k );
    str3[i][j][k] = CORBA_string_alloc(strlen(buf));
    strcpy( str3[i][j][k], buf );
}
ret = ODsample_arraytest_op2( obj, str1, &str2, str3, &env );

/* 領域解放 */
for( i = 0; i < 2; i++ )
for( j = 0; j < 3; j++ )
for( k = 0; k < 4; k++ ) {
    CORBA_free(str1[i][j][k]);            /* inパラメタ */
    CORBA_free(str3[i][j][k]);            /* inoutパラメタ */
}
CORBA_free( ret );                        /* 復帰パラメタ */
CORBA_free( str2 );                       /* outパラメタ */
```

(4)サーバアプリケーションで扱うパラメタ(固定長)

配列(固定長)のパラメタ(in、out、inout)を扱う場合、配列領域はスケルトンで自動的に獲得／解放されます。out／inoutパラメタへのデータ設定は、その領域を使用します。復帰パラメタについては、XX_alloc関数で領域を獲得し、その領域にデータを設定し返します。獲得した領域は、スケルトンで解放されます。
サーバアプリケーションの処理例を以下に示します。

```
ODsample_arraytest_fix_slice
*ODsample_arraytest_op1(
    ODsample_arraytest          obj,
    ODsample_arraytest_fix      para1,
    ODsample_arraytest_fix      para2,
    ODsample_arraytest_fix      para3,
    CORBA_Environment           *env )
{
    ODsample_arraytest_fix_slice  *para;
    int i, j, k;

    /* outパラメタ */
    for( i = 0; i < 4; i++ )                /* データ設定 */
    for( j = 0; j < 3; j++ )
    for( k = 0; k < 2; k++ )
        para2[i][j][k] = i+j+k;

    /* inoutパラメタ */
    for( i = 0; i < 4; i++ )                /* データ設定 */
    for( j = 0; j < 3; j++ )
    for( k = 0; k < 2; k++ )
        para3[i][j][k] = (i+j+k)*10;

    /* 復帰パラメタ */
    para = ODsample_arraytest_fix_alloc();  /* 領域獲得 */
    for( j = 0; j < 3; j++ )                /* データ設定 */
    for( k = 0; k < 2; k++ )
        para[j][k] = j*k;
    return( para );
}
```

(5)サーバアプリケーションで扱うパラメタ(可変長)

配列(可変長)のin／inoutパラメタを扱う場合、配列領域はスケルトンで自動的に獲得／解放されます。out／復帰パラメタについては、XX_alloc関数で配列スライスの領域を獲得し、その領域にデータを設定します。inoutパラメタの領域を拡張する場合は、一度CORBA_free()でデータ域を解放し、再度領域を獲得します。ここで獲得した領域は、スケルトンで自動的に解放されます。
サーバアプリケーションでの処理例を以下に示します。

```
ODsample_arraytest_str_slice
*ODsample_arraytest_op2(
    CORBA_Object                obj,
    ODsample_arraytest_str      para1,
    ODsample_arraytest_str_slice  **para2,
    ODsample_arraytest_str      para3,
    CORBA_Environment           *env )
{
    CORBA_string                str;
    ODsample_arraytest_str_slice  *para;
    int                         i, j, k;
    char                        buf[120];

    /* outパラメタ */
    *para2 = ODsample_arraytest_str_alloc();   /* 領域獲得 */
    for( j = 0; j < 3; j++ )
    for( k = 0; k < 4; k++ ){
```

```
        sprintf(buf, "(*para2)[%d][%d]", j, k );
        (*para2)[j][k] = CORBA_string_alloc(strlen(buf));
        strcpy( (*para2)[j][k], buf );
    }

    /* inoutパラメタ */
    for( i = 0; i < 2; i++ )                    /* 領域解放 */
    for( j = 0; j < 3; j++ )
    for( k = 0; k < 4; k++ )
        CORBA_free( para3[i][j][k] );
    for( i = 0; i < 2; i++ )
    for( j = 0; j < 3; j++ )
    for( k = 0; k < 4; k++ ) {
        sprintf(buf, "para3[%d][%d][%d]", i, j, k );
        para3[i][j][k] = CORBA_string_alloc(strlen(buf));  /* 領域再獲得 */
        strcpy( para3[i][j][k], buf );
    }

    /* 復帰パラメタ */
    para = ODsample_arraytest_str_alloc();      /* 領域獲得 */
    for( j = 0; j < 3; j++ )
    for( k = 0; k < 4; k++ ) {
        sprintf(buf, "para[%d][%d]", j, k );
        para[j][k] = CORBA_string_alloc(strlen(buf));
        strcpy( para[j][k], buf );
    }
    return( para );
}
```

3.7.9 属性宣言(attribute)のマッピング

(1)IDLマッピング

IDL言語で属性宣言(attribute)を指定すると、オブジェクトのデータ設定／取得を行う関数(関数名:“モジュール名_インタフェース名_set(get)_変数名”。以降、データ設定関数／データ設定関数と呼ぶ)が生成されます。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface    attrtest{
        attribute long          para1;
        attribute string        para2;
        readonly attribute long  para3;
    };
};
```

C言語

```
ODsample_attrtest_set_para1( obj, 2, &env );
ODsample_attrtest_get_para1( obj, &env );

ODsample_attrtest_set_para2( obj, "test", &env );
ODsample_attrtest_get_para2( obj, &env );

ODsample_attrtest_get_para3( obj, &env );
```

(2)クライアントアプリケーションで扱うパラメタ

データ設定関数でデータ設定に使用するデータ域は、inパラメタと同様に扱われます。また、データ取得関数でデータ取得に使用するデータ域は、復帰パラメタと同様に扱われます。このため、可変長データでは、その領域が不要になった時点でCORBA_free()で領域を解放する必要があります。

```
CORBA_Object         obj;
CORBA_Environment    env;
CORBA_long           ret;
CORBA_string         str;

/* para1: データ設定関数 */
ODsample_attrtest_set_para1( obj, 2, &env );

/* para1: データ取得関数 */
ret = ODsample_attrtest_get_para1( obj, &env );
printf( "ODsample_attrtest_get_para1 returns [%d]¥n", ret );

/* para2: データ設定関数 */
ODsample_attrtest_set_para2( obj, "test", &env );

/* para2: データ取得関数 */
str = ODsample_attrtest_get_para2( obj, &env );
env_check( "ODsample_attrtest_get_para2" );
printf( "ODsample_attrtest_get_para2 returns [%s]¥n", str );
CORBA_free( str );                  /* 領域解放 */

/* para3: データ取得関数 */
ret = ODsample_attrtest_get_para3( obj, &env );
printf( "ODsample_attrtest_get_para3 returns [%d]¥n", ret );
```

(3)サーバアプリケーションで扱うパラメタ

データ設定関数でデータ設定に使用するデータ域はinパラメタと同様に扱われ、スケルトンで自動的に獲得／解放されます。データ取得関数でデータ取得に使用するデータ域は、復帰パラメタと同様に扱われます。このため、可変長データでは、データ域獲得関数で領域獲得する必要があります。獲得した領域は、スケルトンで自動的に解放されます。

```
/* para1: データ設定関数 */
void
ODsample_attrtest_set_para1(
      ODsample_attrtest   obj,
      CORBA_long          para,
      CORBA_Environment   *env )
{
      para1 = para;
}

/* para1: データ取得関数 */
CORBA_long
ODsample_attrtest_get_para1(
      ODsample_attrtest   obj,
      CORBA_Environment   *env )
{
      return para1 ;
}

/* para2: データ設定関数 */
void
ODsample_attrtest_set_para2(
      ODsample_attrtest   obj,
      CORBA_string        para,
      CORBA_Environment   *env )
```

```
{
      CORBA_free( para2 );
      para2 = CORBA_string_alloc( strlen( para ) );
      strcpy( para2, para );
      return;
}

/* para2: データ取得関数 */
CORBA_string
ODsample_attrtest_get_para2(
      ODsample_attrtest   obj,
      CORBA_Environment   *env )
{
      CORBA_string    ret;

      if( para2 == NULL ) {
              ret = CORBA_string_alloc(1);
              ret[0] = '¥0';
              return( ret );
      }
      else {
              ret = CORBA_string_alloc( strlen(para2)+1 );
              strcpy( ret, para2 );
              return( ret );
      }
}

/* para3: データ取得関数 */
CORBA_long
ODsample_attrtest_get_para3(
      ODsample_attrtest   obj,
      CORBA_Environment   *env )
{
      return( para3 );
}
```

3.7.10 パラメタ受渡しで使用するデータ型

クライアント／サーバアプリケーションでパラメタ受渡しに使用するデータ型を以下に示します。

CORBAデータ型	in	out	inout	復帰
long	long	long*	long*	long
short	short	short*	short*	short
unsigned long	unsigned long	unsigned long*	unsigned long*	unsigned long
unsigned short	unsigned short	unsigned short*	unsigned short*	unsigned short
long long	long long	long long*	long long*	long long
float	float	float*	float*	float
double	double	double*	double*	double
long double Windows32/64 Solaris32/64	long double	long double*	long double*	long double
char	char	char*	char*	char
wchar	wchar	wchar*	wchar*	wchar
octet	octet	octet*	octet*	octet
Boolean	boolean	boolean*	boolean*	boolean
enum	enum	enum*	enum*	enum

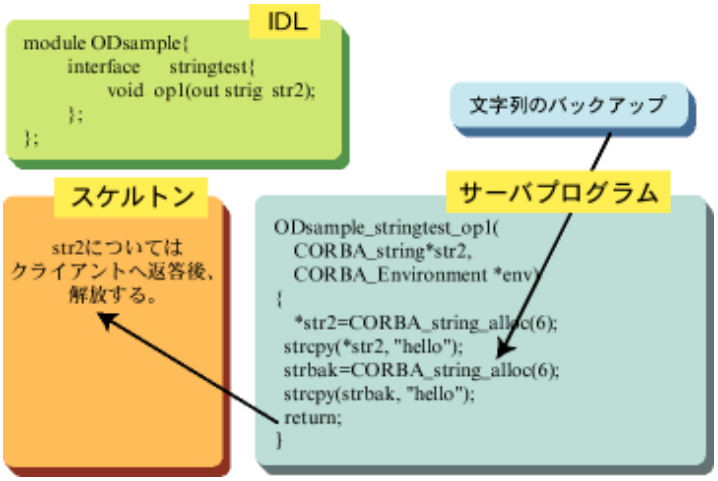
CORBAデータ型		in	out	inout	復帰
string		char*	char**	char**	char*
wstring		wchar*	wchar**	wchar**	wchar*
any		any*	any**	any*	any*
sequence		sequence*	sequence**	sequence*	sequence*
struct	固定長	struct*	struct*	struct*	struct
	可変長	struct*	struct**	struct*	struct*
union	固定長	union*	union*	union*	union
	可変長	union*	union**	union*	union*
array(配列)	固定長	array	array	array	array slice*
	可変長	array	array slice**	array	array slice*
Object		CORBA_Object	CORBA_Object*	CORBA_Object*	CORBA_Object
TypeCode		CORBA_TypeCode	CORBA_TypeCode*	CORBA_TypeCode*	CORBA_TypeCode

(1)NULLポインタの設定について

クライアントアプリケーションのin、inoutパラメタ、およびサーバアプリケーションのout、inoutパラメタ復帰値では、文字列型、シーケンス型、構造体、共用型、配列データ型でNULLポインタを設定できません。

3.8 any型、sequence型のリリースフラグ

可変長データは、データをクライアントへ返却後、スケルトン中で自動的に解放します。そのため、解放したくないデータについては、どこか別の領域にそのデータを複写しておく必要があります。文字列(string)を例に説明します。



しかし、any、sequenceのような複雑な型の場合は、コピーする処理が複雑なため、リリースフラグを設定し、解放する／しないを設定できるようになっています。
リリースフラグの設定方法について以下に示します。

変数型	設定方法
any	CORBA_any_set_releaseを使用。デフォルトは、CORBA_FALSE (注) 例） CORBA_any *any = CORBA_any_alloc(); CORBA_any_set_release(any, CORBA_TRUE);
sequence	CORBA_sequence_set_releaseを使用。デフォルトは、CORBA_FALSE (注) 例） struct sequence<long> data; data = CORBA_data_alloc(); CORBA_sequence_set_release(data, CORBA_TRUE);

注) CORBA_TRUEの場合は、解放されます。CORBA_FALSEの場合は、解放されません。

3.9 アプリケーション作成時の注意事項

アプリケーションを作成する場合の注意事項については、“第2章 CORBAアプリケーション開発時の留意事項”を参照してください。

第4章 アプリケーションの開発(C++言語)

4.1 アプリケーションの開発手順

4.1.1 アプリケーションのインタフェース定義(IDL)の記述

サーバアプリケーションが提供するインタフェースを定義するために、IDLファイルを作成します。IDLファイルの書式については、“IDL定義”を参照してください。

例

```
module ODdemo{
  interface calculator{
    exception ZEROPARAM {};
    struct result {
      long    add_result;
      long    subtract_result;
      long    multiple_result;
      float   divide_result;
    };

    result calculate( in long a, in long b )
        raises( ZEROPARAM );
  };
};
```

4.1.2 IDLファイルのコンパイル

IDLcコマンドでIDLファイルをコンパイルします。この結果、IDL定義をC++言語にマッピングした複数のファイルが出力されます。IDLcコマンドの詳細については、“リファレンスマニュアル(コマンド編)”の“IDLc”を参照してください。

例

Windows32/64

```
IDLc -vcpp simple.idl
```

Solaris32/64 Linux32/64

```
IDLc -cpp simple.idl
```

simple.idlというIDLファイルをIDLcコマンドで-cpp／-vcppにコンパイルすると、以下のファイルが出力されます。

[-cppオプション]

ファイル名	ファイル種別
simple.H	共通ヘッダファイル
simple.h	共通ヘッダファイル
simple_cdr.h	CDRヘッダファイル

ファイル名	ファイル種別
simple_cdr.c	CDRソースファイル
simple_c++.C	クラスメソッド定義ファイル
simple_stub_c++.C	スタブファイル
simple_skel_c++.C	スケルトンファイル

[-vcppオプション]

ファイル名	ファイル種別
simple.h	共通ヘッダファイル
simple_cdr.h	CDRヘッダファイル
simple_cdr.cpp	CDRソースファイル
simple_c++.cpp	クラスメソッド定義ファイル
simple_stub.cpp	スタブファイル
simple_skel.cpp	スケルトンファイル

4.1.3 サーバアプリケーションのプログラミング

サーバアプリケーションのコーディングを行います。サーバアプリケーションは、「初期化処理部」と「インタフェース実装部」に分けられます。詳細については、“4.2 サーバアプリケーションのプログラミング（Basic Object Adapter：BOA）”を参照してください。また、アプリケーション開発時の留意事項として“2.1 コーディング”を参照してください。

4.1.4 クライアントアプリケーションのプログラミング

クライアントアプリケーションのコーディングを行います。クライアントアプリケーションは、「初期化処理部」と「メソッド呼出し部」に分けられます。詳細については、“4.5 クライアントアプリケーションのプログラミング”を参照してください。また、アプリケーション開発時の留意事項として“2.1 コーディング”を参照してください。

4.1.5 アプリケーションのコンパイルとリンク

開発環境を使用してアプリケーションのコンパイル・リンクを行います。
サーバアプリケーションは、ユーザが作成したサーバアプリケーション用のファイルとIDLコンパイルで生成されたスケルトンファイル、クラスメソッド定義ファイル、CDRソースファイルをコンパイル・リンクします。
クライアントアプリケーションは、ユーザが作成したクライアントアプリケーション用のファイルとIDLコンパイルで生成されたスタブファイル、クラスメソッド定義ファイル、CDRソースファイルをコンパイル・リンクします。
コンパイル・リンク時の注意事項については、“2.2 コンパイル・リンク”を参照してください。

4.1.6 サーバアプリケーションの登録

OD_impl_instコマンドを使用して、インプリメンテーションリポジトリにサーバアプリケーション情報を登録します。
また、OD_or_admコマンドを使用してネーミングサービスにサーバアプリケーションのオブジェクトリファレンスを登録します。

例

OD_impl_instコマンド実行例

```
OD_impl_inst -ax def
```

defファイルの内容

```
rep_id  = IDL:ODdemo/calculator:1.0
type    = persistent
mode    = SYNC_END
```

OD_or_admコマンド実行例

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -n ODdemo::calculator
```

4.2 サーバアプリケーションのプログラミング（Basic Object Adapter：BOA）

4.2.1 静的スケルトンインタフェース

サーバアプリケーションは、初期化処理部とインタフェースの実装部分から構成されます。初期化処理部では、以下に示す処理を行います。

4.2.1.1 初期化

CORBAの初期化メソッドCORBA::ORB_init()を呼び出して、初期化処理を行います。結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
main ( int argc, char *argv[] )
{
    CORBA::ORB_ptr          orb;    // ORBのオブジェクトリファレンス
    CORBA::Environment      env;    // 例外情報
    int                     current_argc = argc;

    env = new CORBA::Environment;
    orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );
```

また、CORBA::ORB::BOA_init()の呼出しにより、基本オブジェクトアダプタの初期化処理を行います。

```
    CORBA::BOA_ptr  boa;                // BOA用オブジェクトリファレンス
    boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, env );
```

なお、必要であれば、サーバアプリケーションの初期化処理も行います。

4.2.1.2 サーバの活性化

サーバアプリケーションの初期化が完了すると、ORBに対してその旨を通知します。ORBは、この命令が発行された時点で、クライアントからの要求をサーバアプリケーションに伝えます。活性化の方法は、サーバのタイプにより以下のように異なります。

サーバのタイプ	メソッド
shared server	CORBA::BOA::impl_is_ready
unshared server	CORBA::BOA::obj_is_ready
persistent server	CORBA::BOA::impl_is_ready

(1)インプリメンテーションリポジトリのオブジェクトリファレンスの獲得

インプリメンテーションリポジトリのオブジェクトリファレンスを取り出します。メソッドのパラメタには、CORBA_ORB_ObjectId_ImplementationRepositoryを指定します。

(2)ImplementationDefオブジェクトのオブジェクトリファレンスの検索

FJ::ImplementationRep::lookup_id()を使用して、ImplementationDefオブジェクトのオブジェクトリファレンスを求めます。パラメタには、サーバアプリケーションのImplementationRepオブジェクトを指定します。

(3)サーバの活性化

CORBA::BOA::impl_is_ready()／CORBA::BOA::obj_is_ready()を使用して、サーバの活性化を行います。

サーバアプリケーションはプロセスで実装するかスレッドで実装するかにより、メソッドの動作が異なります。また、スレッドで実装する場合は、インプリメンテーションリポジトリ内に設定されている情報によっても異なります。動作の違いを以下に示します。

条件	動作	
プロセスで実装	OD_impl_instでmode=SYNC_END	メソッドは即時に復帰せず、このオブジェクトの停止がオペレータにより指示されたときに復帰します。(注)
スレッドで実装	OD_impl_instでmode=SYNC_END	同上
	OD_impl_instで mode=COMPATIBLE	メソッドは、即時に復帰します。 その後、thr_exitを呼び出す必要があります。

注) メソッドの復帰後は、初期化時にオープンしたファイルのクローズ処理／領域の解放処理などの非活性化処理を行う必要があります。詳細については、“4.2.1.4 サーバの非活性化”を参照してください。

Windows32/64

以下の条件のとき、CORBA::BOA::impl_is_ready()／CORBA::BOA::obj_is_ready()が復帰せずにサーバアプリケーションが終了します。

- サーバアプリケーションをプロセスモードで作成している場合 かつ
- サーバアプリケーションをCORBAワークユニット運用している場合 かつ
- CORBAワークユニットを停止した場合

CORBA::BOA::impl_is_ready()／CORBA::BOA::obj_is_ready()の後に後処理を行う場合は、サーバアプリケーションをスレッドモードで作成するか、またはワークユニットの出口機能を利用してください。ワークユニットの出口機能に関しては、“OLTPサーバ運用ガイド”の“ワークユニットの出口機能”を参照してください。

```
CORBA::ImplementationDef_ptr impl;     // インプリメンテーション情報検索用
ImplementationRep_ptr        impl_rep; // ImplementationRepオブジェクトリファレンス
CORBA::Object_ptr            o;        // オブジェクトリファレンスのポインタ

// インプリメンテーション情報のオブジェクトリファレンス検索
o = orb->resolve_initial_references(
    CORBA_ORB_ObjectId_ImplementationRepository, env );

// ImplementationRepクラスへの変換
impl_rep = FJ::ImplementationRep::_narrow(o);

// サーバのImplementationRepオブジェクトの検索
o = impl_rep->lookup_id( _IMPL_ODdemo_calculator, env );

// ImplementationDefクラスへの変換
impl = CORBA::ImplementationDef::_narrow(o);

// サーバの活性化
boa->impl_is_ready( impl, env );
```

4.2.1.3 インタフェース実装関数

初期化処理の後、サーバアプリケーションで実装するインタフェースの処理を記述します。インタフェースを実装する方法として、以下の2種類があります。

- implクラス方式
- TIEクラス方式

implクラス方式

派生クラスのメソッドとして実装する方法です。以下に特長を示します。

- インタフェース実装クラスは、IDLコンパイラで生成されます。ユーザは、インタフェース実装クラスのメソッドとして、インタフェース実装関数だけを作成すればよいため、サーバアプリケーションの作成が容易です。
- インタフェース実装クラスは、IDLコンパイラで生成されて固定化されるため、IDLに記述できない機能を追加できません。

TIEクラス方式

テンプレートクラス機能を利用して実装する方法です。以下に特長を示します。

- ユーザは、インタフェース実装クラスとそのメンバ関数としてのインタフェース実装関数を作成します。
- インタフェース実装関数は、IDLコンパイラから生成されたTIEクラスを介して呼び出されます。
- インタフェース実装クラスには、IDLで定義したインタフェース(クラス)のほか、任意の機能を同時に実装できます。

TIEクラス方式を使用する場合は、IDLcコマンド実行時に-tieオプションを指定する必要があります。

それぞれの実装方法について、以下のIDL定義をもとに説明します。
なお、アプリケーションで使用可能なCORBAのデータ型については、“4.6 データ型に対するマッピング”を参照してください。

IDLファイル

```
module ODdemo{
    interface  calculator{
```

```
        exception ZEROPARAM {};
        struct result {
            long     add_result;
            long     subtract_result;
            long     multiple_result;
            float    divide_result;
        };
        result calculate( in long a, in long b )
            raises( ZEROPARAM );
    };
};
```

(1)implクラス方式

implクラスを使用する場合、以下を記述します。

- インタフェース実装関数

インタフェース実装関数

インタフェース実装関数は、“モジュール名_インタフェース名_impl”という名前で定義されたインタフェース実装クラスのメンバ関数として実装します。

```
// IDLで生成されるヘッダ定義
class ODdemo_calculator_impl {
public:
    ODdemo::calculator::result calculate( CORBA::Long, CORBA::Long, CORBA::Environment & )
          throw ( CORBA::Exception );
}

// インタフェースの実装
ODdemo::calculator::result                  // 復帰値の型
ODdemo_calculator_impl::calculate(          // addメソッドの実装
    CORBA::Long  a,                         // 入力値
    CORBA::Long  b,                         // 入力値
    CORBA::Environment & )                  // エラー値
    throw( CORBA::Exception )
{
    if( b == 0 ){                           // 0除算チェック
        ODdemo::calculator::ZEROPARAM  exc;
        throw( exc );
    }

    ODdemo::calculator::result res;
    res.add_result = a+b;                   // 加算結果設定
    res.subtract_result = a-b;              // 減算結果設定
    res.multiple_result = a*b;              // 乗算結果設定
    res.divide_result = (CORBA::Float)a/b;  // 除算結果設定
    return res;
}
```

(2)TIEクラス方式

TIEクラスを使用する場合、以下を記述します。

- インタフェース実装クラス
- インタフェース実装関数
- インタフェース実装クラス割当て関数

インタフェース実装クラス

インタフェース実装クラスには、以下の定義をpublic属性として含める必要があります。

- デフォルトコンストラクタ
- デストラクタ
- オペレーション／アトリビュートの実装関数

オペレーションとアトリビュートの実装関数は、IDLコンパイラが生成するヘッダに定義されているインタフェースと同一の名前にします。ただし、“throw(CORBA::Exception)”は必要ありません。

その他のメンバ定義やクラス名は、適宜追加できます。

```
class  CalculatorImpl {
public:
    CalculatorImpl();           //コンストラクタの宣言
    ~CalculatorImpl();          //デストラクタの宣言
    ODdemo::calculator::result
    calculate  (                //オペレーションの宣言
          CORBA::Long    a,
          CORBA::Long    b,
          CORBA::Environment  & );
};
```

インタフェース実装関数

インタフェース実装関数は、インタフェース実装クラスのメンバ関数として定義します。
デフォルトコンストラクタとデストラクタでは、特定の処理を記述する必要はありませんが、独自の処理を記述することもできます。

```
CalculatorImpl::CalculatorImpl()
{
}

CalculatorImpl::~CalculatorImpl()
{
}

ODdemo::calculator::result                  // 復帰値の型
CalculatorImpl::calculate(                  // メソッドの実装
    CORBA::Long    a,                       // 入力値
    CORBA::Long    b,                       // 入力値
    CORBA::Environment   & )                // エラー値
{
    if( b == 0 ){                           // 0除算チェック
        ODdemo::calculator::ZEROPARAM  exc;
        throw( exc );
    }
    ODdemo::calculator::result res;
    res.add_result = a+b;                   // 加算結果設定
    res.subtract_result = a-b;              // 減算結果設定
    res.multiple_result = a*b;              // 減算結果設定
    res.divide_result = (CORBA::Float)a/b;  // 除算結果設定
    return res;
}
```

インタフェース実装クラス割当て関数

インタフェース実装クラス割当て関数は、インタフェース実装クラスを生成し、TIEクラスにインスタンスを設定します。この関数は、クライアントからリクエストを受け取ったときに自動的に呼び出され、インタフェース実装クラスのインスタンスが生成されます。
TIEクラスとインタフェース実装クラス割当て関数の名前は、以下のようになります。

TIEクラス

_tie_インタフェース名

インタフェース実装クラス割当て関数

インタフェースクラス_ptr _ct_インタフェース名();

インスタンはデフォルトコンストラクタで生成する必要はなく、パラメタ付きのコンストラクタやoperator演算子などを利用して生成することもできます。TIEクラスにインスタンスを設定するためのコンストラクタは、1つに限定されいます。TIEクラスを割り当てるときは、引数として必ずインタフェース実装クラスのインスタンスを渡す必要があります。

```
ODdemo::calculator_ptr  _ct_ODdemo_calculator()
{
    CalculatorImpl  *impl = new CalculatorImpl();
    _tie_ODdemo_calculator<CalculatorImpl>    *tie
           = new _tie_ODdemo_calculator<CalculatorImpl>(*impl);
    return  tie;
}
```

4.2.1.4 サーバの非活性化

サーバアプリケーションは、利用者などからの停止要求を受けた場合、クライアントからの要求を以降受け付けない旨をORBに対して通知します。ORBはこの通知を受けて、サーバアプリケーションにクライアントからの処理要求を伝えず、クライアントに例外を返します。非活性化の方法は、サーバタイプにより以下のように異なります。

サーバのタイプ	メソッド
shared server	CORBA::BOA::deactivate_impl
unshared server	CORBA::BOA::deactivate_obj
persistent server	CORBA::BOA::deactivate_impl

deactivate_implでは、サーバアプリケーションのImplementationRepオブジェクトをパラメタとして指定します。deactivate_objでは、オブジェクトリファレンスをパラメタとして指定します。

```
boa->deactivate_impl( impl, env );    // サーバの非活性化
```

なお、ワークユニット運用している場合は、isstopwuコマンドやInterstage管理コンソールから停止操作が実行されることにより、クライアントからの要求を以降受け付けない旨をORBに対して通知します。そのため、アプリケーション側で非活性化用のメソッドを発行する必要はありません。
ワークユニット運用でないCORBAアプリケーションの場合も、odcntlqueコマンドでサーバアプリケーションを終了する場合は、アプリケーション側で非活性化用のメソッドを発行する必要はありません。

4.2.2 サーバアプリケーションの例外処理

(1)例外情報の設定

IDLコンパイラで、インタフェース名::例外の識別名の例外クラス名が生成されます。このクラスでは、例外メンバをprivateデータとして保持します。この例では、ODdemo、calculator、ZEROPARAMがそれにあたります。例外情報を設定するためには、まず、例外クラスのインスタンスをC++の演算子newで生成し、続いて例外クラスのメンバにデータを設定します。最後に、CORBA::Environment::exception()を呼び出して、Environmentクラスのインスタンスにユーザ例外を設定します。

```
ODdemo::calculator::result              // 復帰値
ODdemo_calculator_impl::calculate(      // divideメソッドの実装
      CORBA::Long      a,
      CORBA::Long      b,
      CORBA::Environment & )
```

```
      throw( CORBA::Exception )
{
        :
    if( b == 0 ){
        // 例外クラスをthrowする。
        ODdemo::calculator::ZEROPARAM  exc;
        throw( exc );
    }
        :

    return res;
}
```

注意

サーバメソッド内部から他のメソッドを呼び出す場合の引数として、サーバメソッドの引数に渡されるCORBA::Environmentクラスのインスタンスを使用しないでください。サーバメソッドの例外情報に、他のメソッドで発生したサーバメソッド内部の例外情報が設定されることになります。サーバメソッド内部から他のメソッドを呼び出す場合は、別のCORBA::Environmentクラスのインスタンスを使用してください。以下に正しいプログラム例、および誤ったプログラム例を示します。

[正しいプログラム例]

```
ODdemo::calculator::result
ODdemo_calculator_impl::calculate(
    CORBA::Long        a,
    CORBA::Long        b,
    CORBA::Environment &env )
    throw( CORBA::Exception )
{
    CosNaming::NamingContext_ptr  nc;
    CosNaming::Name               name;
    CORBA::Object_ptr             obj;
    CORBA::Environment            local_env;
                                /* 別のCORBA::Environmentクラスのインスタンスを用意 */
        :

    /*
     * CosNaming::NamingContext::bind()の例外情報は、local_envに設定されるため、
     * ODdemo::calculator_impl::calculate()の例外情報として扱われることはない
     */
    nc->bind( name, obj, local_env );
        :
}
```

[誤ったプログラム例]

```
ODdemo::calculator::result
ODdemo_calculator_impl::calculate(
    CORBA::Long        a,
    CORBA::Long        b,
    CORBA::Environment &env )
    throw( CORBA::Exception )
{
    CosNaming::NamingContext_ptr  nc;
    CosNaming::Name               name;
    CORBA::Object_ptr             obj;
        :

    /*
     * CosNaming::NamingContext::bind()が例外復帰した場合、
     * CosNaming::NamingContext::bind()の例外情報が
```

```
     * ODdemo::calculator_impl::calculate()の例外情報として扱われてしまう
     */
    nc->bind( name, obj, env );
        :
}
```

(2)例外情報の獲得

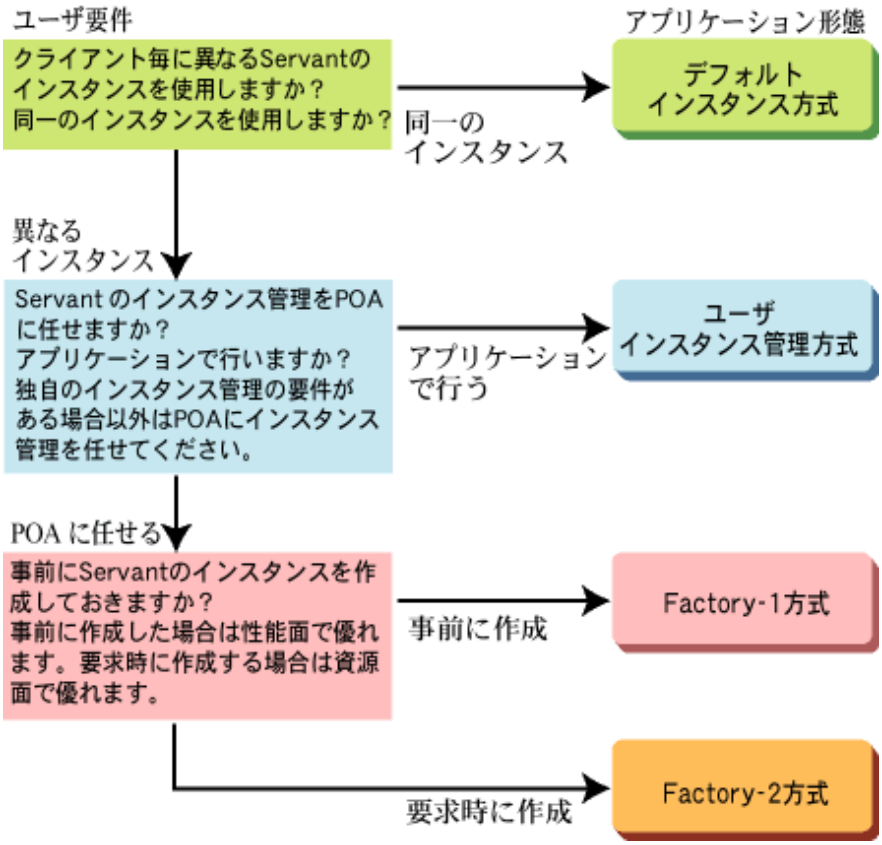
サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“4.5.2 クライアントアプリケーションの例外処理”を参照してください。

4.3 サーバアプリケーションのプログラミング（Portable Object Adapter：POA）

4.3.1 インスタンス管理とアプリケーション形態

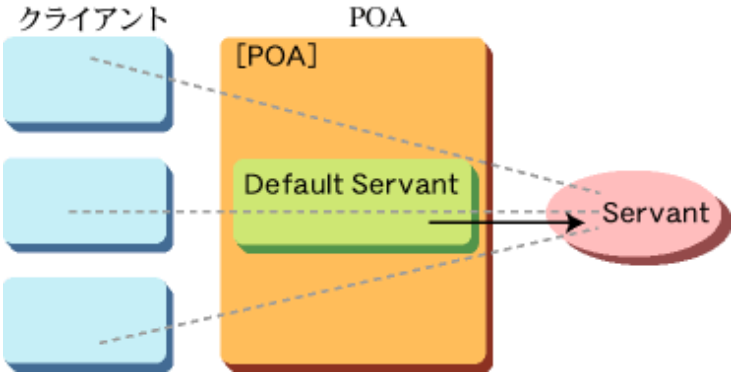
4.3.1.1 アプリケーション形態の種別

サーバアプリケーションは、Servant(インタフェース実装部)のインスタンスの管理方法により以下の4つの形態をとることができます。以下のように要件によりアプリケーション形態を選択してください。



デフォルトインスタンス方式

各クライアントが同一のServantオブジェクトを使用する方式です。つまり、クライアントごとのインスタンス管理をしない方式です。各クライアントは、以下のように同一のServantのインスタンスを使用します。



Factory-1方式

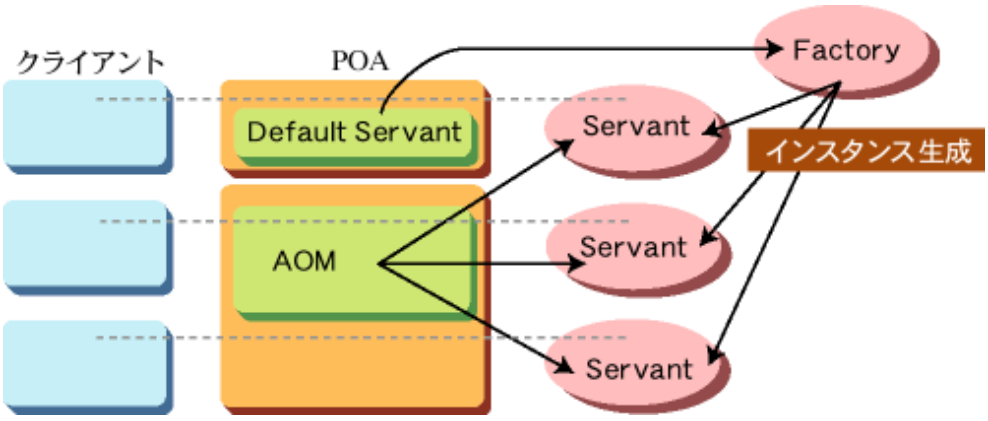
クライアントごとに異なるServantオブジェクトを使用する方式です。
クライアントごとのインスタンス管理をPOAオブジェクトで行う方式です。Factory内で、オブジェクトリファレンスおよびインスタンスを作成します。

Factory-2方式

クライアントごとに異なるServantオブジェクトを使用する方式です。
クライアントごとのインスタンス管理をPOAオブジェクトで行う方式です。Factory内でオブジェクトリファレンスを作成し、リクエスト受信時にServantManagerオブジェクト内でインスタンスを作成します。

Factoryはリクエストごとにオブジェクトリファレンスやインスタンスを作成するオブジェクトであり、インスタンス管理を行う場合に必要となるオブジェクトです。
AOMはActive Object Mapのことであり、インスタンスの管理テーブルのことです。
AOM、ServantManagerなどの詳細については、“4.3.2 POA概要”を参照してください。

Factory-1方式／Factory-2方式の各クライアントは、以下のようにクライアントごとに異なるServantのインスタンスを使用します。

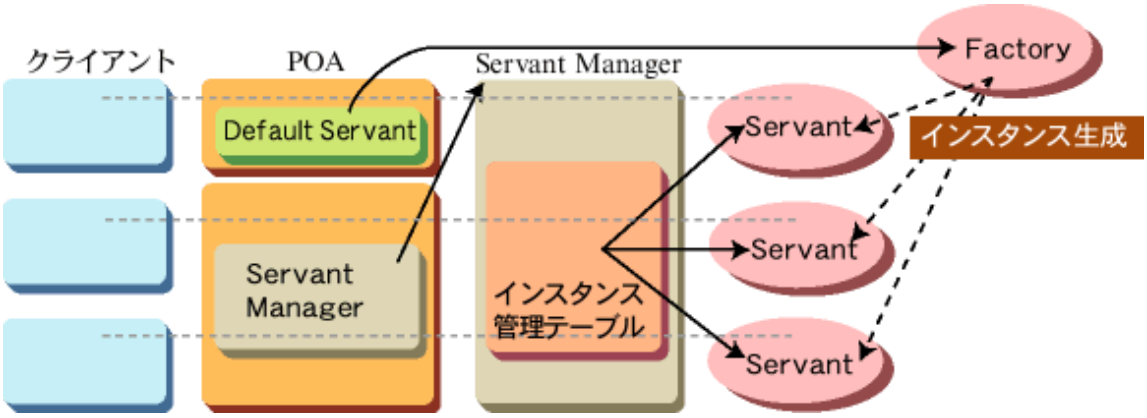


[Factory-1方式とFactory-2方式の違い]

Factory-1方式はFactory内で事前にインスタンスを作成しておくことができるため、リクエストを受信した時点でインスタンスを作成するFactory-2より性能的には若干優れます。
Factory-2方式はリクエストを受信した時点でインスタンスを作成するため、Factory内で事前のインスタンスを作成しておくFactory-1より資源的には若干優れます。

ユーザインスタンス管理方式

インスタンス管理をユーザの作成するServantManagerオブジェクトに任せる方式です。
各クライアントは、以下のようにクライアントごとに異なるServantのインスタンスを使用します。インスタンスは、ユーザが作成したServantManagerオブジェクト内のインスタンス管理テーブルで行われます。



[アプリケーション形態の選択]

アプリケーション形態は、Servantオブジェクトの属するPOAオブジェクトのServantRetentionポリシ／RequestProcessingポリシの指定により決まります。組合せ方を以下に示します。詳細については、“4.3.2.3 POAオブジェクト”を参照してください。

アプリケーション形態	ServantRetentionポリシ	RequestProcessingポリシ
デフォルトインスタンス方式	NON_RETAIN	USE_DEFAULT_SERVANT
Factory-1方式	RETAIN	任意
Factory-2方式	RETAIN	USE_SERVANT_MANAGER
ユーザインスタンス管理方式	NON_RETAIN	USE_SERVANT_MANAGER

注意

表内のポリシは、Servantオブジェクトを管理するPOAポリシに対する設定値を示しています。

[Factoryについて]

Factoryとは、Servantオブジェクトのインスタンスを生成し、そのオブジェクトリファレンスをクライアントに返すことを目的としたインタフェースです。
これにより、そのクライアントが専用で使用できるServantオブジェクトのインスタンスをサーバ側に持たせることが可能となります。
Factoryの概念は、以下のIDL定義によって示されます。前述したFactory-1方式／Factory-2方式を使用する際も、IDL定義で他のインタフェースとともにFactoryインタフェースを定義します。

【FactoryのIDL定義例(a)】

```
// 何らかのインタフェース
interface intf{
    void  op();
};

// Factoryインタフェース
interface Factory {
```

```
    intf   create();
};
```

上記のIDL定義で示されるように、クライアントはFactoryオブジェクトのオペレーションcreate()を発行することにより、そのクライアントで使用するintfオブジェクトを取得することが可能となります。
次に、クライアントは取得したintfオブジェクトに対してop()オペレーションを発行することで、本来の目的の処理を行います。

一方、サーバ側では、以下の一連の処理をcreate()オペレーションの実装として記述する必要があります。

1. Servantオブジェクト(intfインタフェースの実装クラスのインスタンス)を生成(new)します。
2. POAのAOMに生成したインスタンスとオブジェクトリファレンスとの関連付けを登録します。
3. 関連付けられたオブジェクトリファレンスを返します。

Factory-2方式では、create()オペレーションの実装として3.のオブジェクトリファレンスの生成／復帰だけを行います。1.と2.は、要求時にServantManagerオブジェクト内で行います。しかし、クライアント側から見たcreate()オペレーションの機能は同じです。

Factoryとは、このようなインタフェースの機能上の概念です。したがって、IDL定義では、Factoryインタフェースの名前(上記の例の場合:“Factory”)、実装するオペレーションの名前(上記の例の場合:、“create”)、種類は、自由です。しかし、Factoryの目的上、上記のcreate()のような「戻り値としてオブジェクトリファレンスを返す」機能をもつオペレーションが最低限必要となります。

実際のアプリケーション開発では、生成したServantオブジェクトのインスタンスを解放するためのオペレーションもFactoryに実装することが推奨されます。これによりサーバ資源の節約が可能となります。
以下のIDL定義の例では、Servantオブジェクトのインスタンスを解放するためのオペレーションとしてdestroy()を定義しています。

【FactoryのIDL定義例(b)】

```
// 何らかのインタフェース
interface intf{
    void  op();
};

// Factoryインタフェース
interface Factory{
    intf  create();
    void  destroy(in intf obj);
};
```

クライアント側では目的の処理が終了した時点で、以前にcreate()メソッドで取得したオブジェクトリファレンスを引数としてdestroy()オペレーションをコールします。一方、サーバ側では、渡されたオブジェクトリファレンスに対応するServantオブジェクトをAOMから削除する処理を行います。

上記のIDLに対するFactoryオブジェクトのcreate()オペレーション、destoroy()オペレーションの実装例を以下に示します。ここでは、intfインタフェースの実装クラスをUserServantクラスとしています。処理の詳細については、“4.3.2 POA概要”以降を参照してください。

【オペレーションの実装例(Factory-1方式)】

```
intf_ptr  Factory_impl::create( CORBA::Environment&  env )
          throw( CORBA::Exception )
{
    intf_ptr       ior; // Servantオブジェクトのオブジェクトリファレンス
    try {
        // 1. Servantオブジェクトの生成
        intf_impl* svt = new intf_impl();
        // 2. AOMへ登録
        poa->activate_object( svt, env );
        //3.オブジェクトリファレンスの生成
        CORBA::Object_var _tmpObj = poa.servant_to_reference( svt, env );
```

```
        // intf型に変換
        ior = intf::_narrow( _tmpObj );
    }
    catch( CORBA::SystemException&  e ) {  /* 例外処理 */  }
    return( ior );
}

void  Factory_impl::destroy( intf_ptr obj,
CORBA::Environment&  env )
throw( CORBA::Exception )
{
    try {
        // オブジェクトリファレンスからオブジェクトIDを求める
        PortableServer::ObjectId_var  oid = poa->reference_to_id( obj, env );
        // オブジェクトIDからServantを求める
        PortableServer::Servant  svt = poa->id_to_servant( *oid, env );
        // Servantをdeactiveにする（AOMより削除）
        poa->deactivate_object( *oid, env );
        // Servantの削除
        svt->servant_delete();
    }
    catch( CORBA::SystemException&  e ) {  /* 例外処理 */  }
}
```

注) ソース上のpoa:POAオブジェクトのインスタンス

[Factoryオブジェクトのインスタンスの扱いについて]

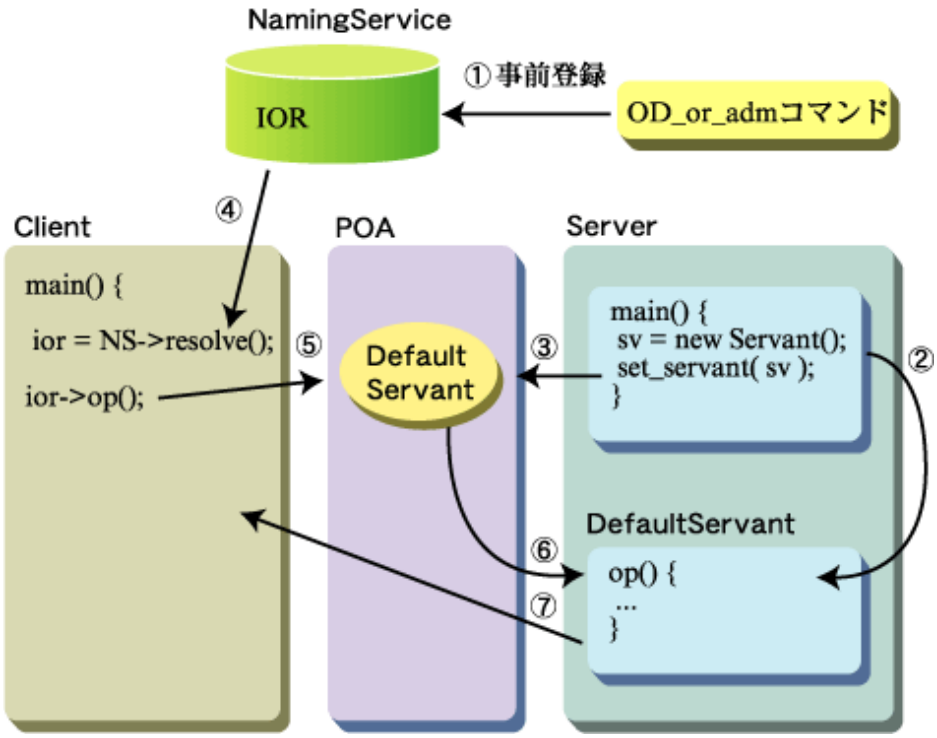
Factoryオブジェクトのインスタンスも、通常のServantオブジェクトのインスタンスと同様に、POAに登録する必要があります。
“4.3.7.2 Active Object Map（AOM）使用例（Factory-1方式）”や“4.3.7.3 Servant Activator使用例（Factory-2方式）”などに示した例では、Factoryオブジェクトは「Factory専用のPOA」を使用してDefault Servantとして登録しています。加えて、Factory用のPOAはAOMを使用しない設定（NON_RETAINポリシ）としています。
これは、以下の理由によるものです。

- 同一のPOAのAOM上で、Factoryオブジェクトとそのfactoryが生成するServantオブジェクトを管理させることは可能です。しかし、この場合、Factoryオブジェクトへの要求に対してもAOM内の検索が行われるため、効率的ではありません。
- ServantオブジェクトをPOAのDefault ServantとしてFactoryオブジェクトを登録することは可能です。しかし、AOMとDefault Servantを同時に使用する場合は、Default Servantへの要求はAOMの検索の後となります。このため、Factoryオブジェクトへの要求は、効率的ではありません。

4.3.1.2 各アプリケーション形態

(1)デフォルトインスタンス方式

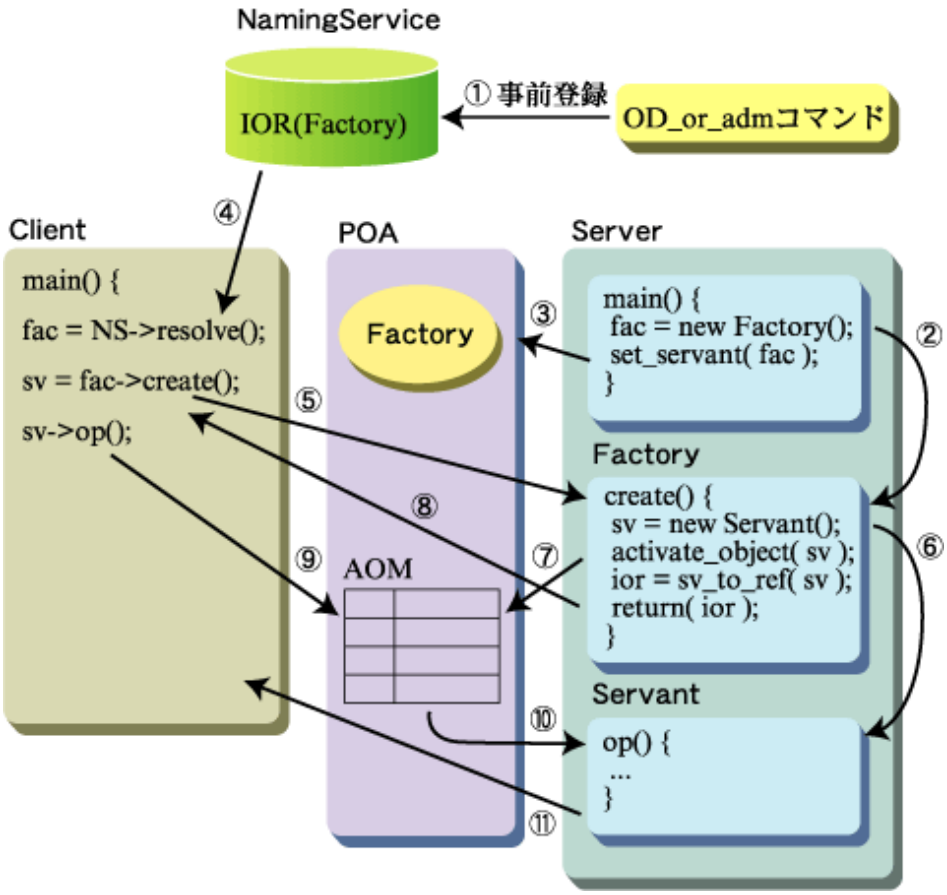
クライアントごとのインスタンス管理をしない方式です。



1. OD_or_admコマンドにより事前にServantのオブジェクトリファレンスをNamingServiceに登録します。
2. サーバアプリケーションは、Servantを作成します。
3. サーバアプリケーションは、ServantをPOAにDefault Servantとして登録します。
4. クライアントアプリケーションは、NamingServiceからオブジェクトリファレンスを獲得します。
5. クライアントアプリケーションは、オペレーションop()を要求します。
6. POAは、登録されているDefault Servantを起動します。
7. POAは、オペレーションop()の結果をクライアントに通知します。

(2)Factory-1方式

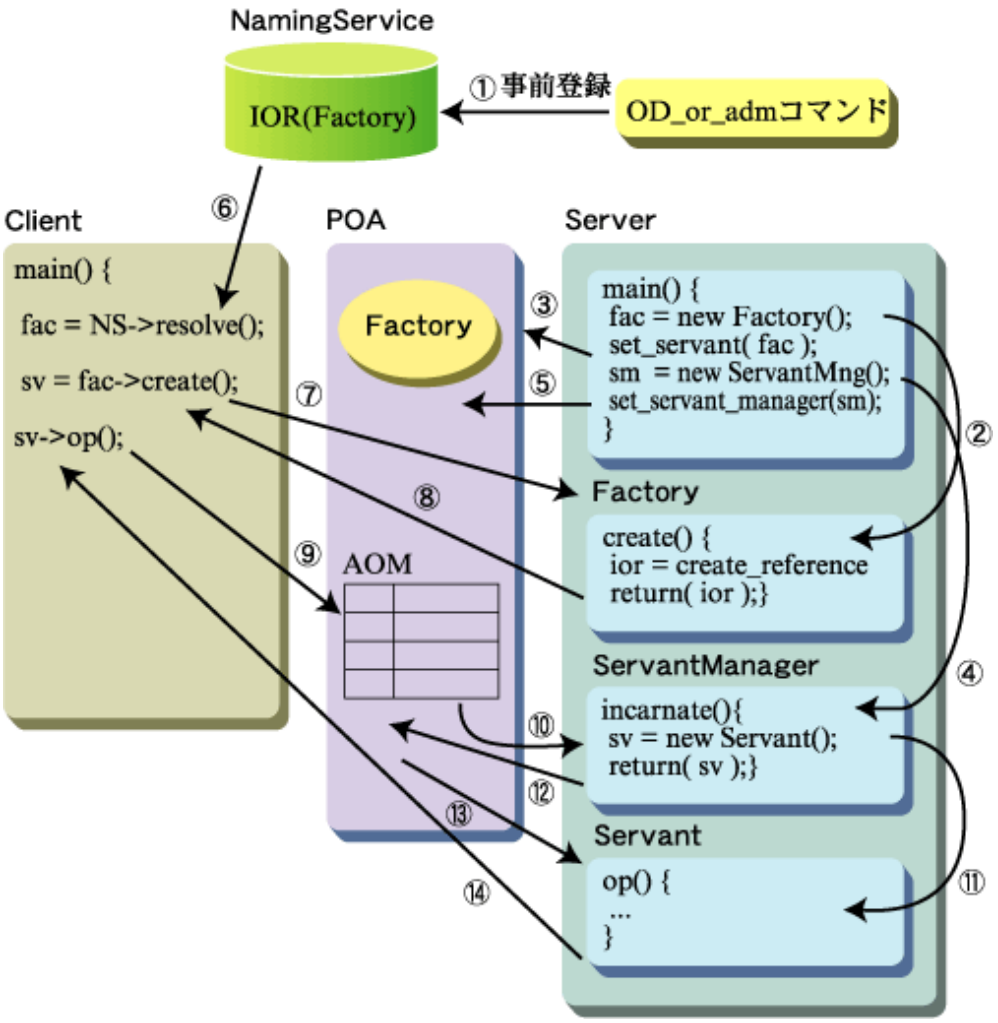
クライアントごとでインスタンス管理をPOAオブジェクトで行う方式です。Factory内で、オブジェクトリファレンスおよびインスタンスを作成します。



1. OD_or_admコマンドで事前にFactoryのオブジェクトリファレンスをNamingServiceに登録します。
2. サーバアプリケーションは、Factoryを作成します。
3. サーバアプリケーションは、FactoryをDefault ServantとしてPOAに登録します。
4. クライアントアプリケーションは、NamingServiceからFactoryのオブジェクトリファレンスを獲得します。
5. クライアントアプリケーションは、オペレーションcreateをFactoryに要求します。
6. Factoryは、Servantのインスタンスを作成します。
7. Factoryは、AOMにインスタンスを登録します。
8. Factoryは、オブジェクトリファレンスを作成してクライアントに通知します。
9. クライアントアプリケーションは、オペレーションop()を要求します。
10. POAは、AOMを検索して、獲得したインスタンスでServantに対してオペレーションop()の実行を依頼します。
11. POAは、オペレーションop()の結果をクライアントに通知します。

(3)Factory-2方式

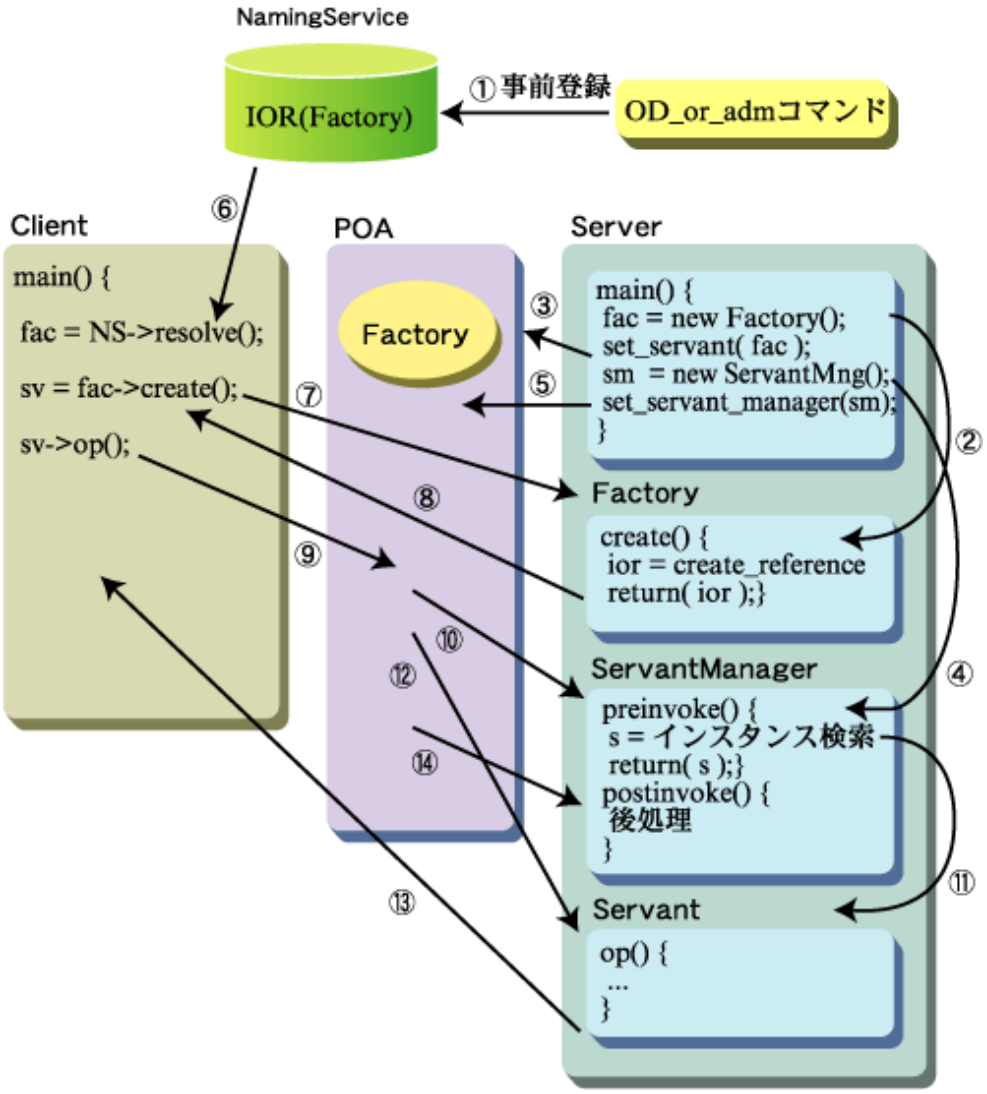
クライアントごとでインスタンス管理をPOAオブジェクトで行う方式です。Factory内でオブジェクトリファレンスを作成し、リクエスト受信時にServantManagerオブジェクト内でインスタンスを作成します。



1. OD_or_admコマンドで事前にFactoryのオブジェクトリファレンスをNamingServiceに登録します。
2. サーバアプリケーションは、Factoryを作成します。
3. サーバアプリケーションは、FactoryをDefault ServantとしてPOAに登録します。
4. サーバアプリケーションは、ServantManagerを作成します。
5. サーバアプリケーションは、ServantManagerをPOAに登録します。
6. クライアントアプリケーションは、NamingServiceからFactoryのオブジェクトリファレンスを獲得します。
7. クライアントアプリケーションは、オペレーションcreateをFactoryに要求します。
8. Factoryは、Servantのオブジェクトリファレンスを作成して、オブジェクトリファレンスをクライアントに通知します。
9. クライアントアプリケーションは、オペレーションop()を要求します。
10. POAはAOMを検索するが、未登録の要求であるため、ServantManagerを呼び出します。
11. ServantManagerは、該当するServantを検索して返します。
12. POAは、ServantManagerからServantを受け取り、AOMに登録します。
13. POAは、Servantに対してオペレーションop()の実行を依頼します。
14. POAは、オペレーションop()の結果をクライアントに通知します。

(4)ユーザインスタンス管理方式

インスタンス管理をユーザの作成するServantManagerオブジェクトに任せる方式です。



1. OD_or_admコマンドで事前にFactoryのオブジェクトリファレンスをNamingServiceに登録します。
2. サーバアプリケーションは、Factoryを作成します。
3. サーバアプリケーションは、FactoryをDefault ServantとしてPOAに登録します。
4. サーバアプリケーションは、ServantManagerを作成します。
5. サーバアプリケーションは、ServantManagerをPOAに登録します。
6. クライアントアプリケーションは、NamingServiceからFactoryのオブジェクトリファレンスを獲得します。
7. クライアントアプリケーションは、オペレーションcreateをFactoryに要求します。
8. FactoryはServantのオブジェクトリファレンスを作成して、オブジェクトリファレンスをクライアントに通知します。
9. クライアントアプリケーションは、オペレーションop()を要求します。
10. POAは、ServantManagerのpreinvoke()を呼び出します。
11. ServantManagerは、Servantのインスタンスを検索して、POAに通知します。

12. POAは、獲得したインスタンスでServantに対してオペレーションop()の実行を依頼します。
13. POAは、オペレーションop()の結果をクライアントに通知します。
14. POAは、ServantManagerのpostinvoke()を呼び出します。

4.3.1.3 アプリケーション形態の比較

	デフォルトインスタンス方式	Factory-1方式	Factory-2方式	ユーザインスタンス管理方式
インスタンス管理	しない	する	する	する
インスタンス管理者	—	POAオブジェクト	POAオブジェクト	ユーザが作成したServantManagerオブジェクト
性能	優れている	Factory-2方式より優れる	Factory-1方式より劣る	ユーザアプリによる
資源	少ない	Factory-2方式より多い	Factory-1方式より少ない	ユーザアプリによる
開発の難易度	やさしい	普通	普通	難しい

4.3.2 POA概要

4.3.2.1 POAとは

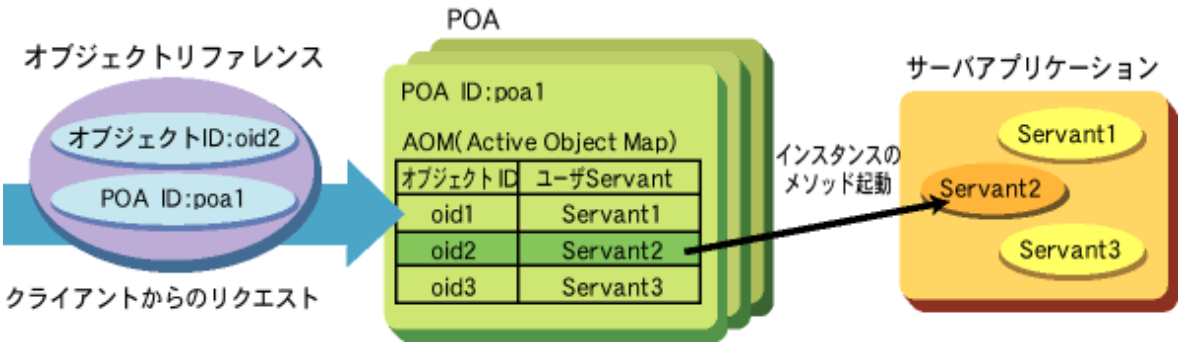
POA (Portable Object Adapter)は、CORBA2.2で新しく標準的なオブジェクトアダプタとして採用された仕様です。POAとしてオブジェクトアダプタの仕様が明確化されたことにより、ORB製品ごとの仕様上の差違が排除されるため、異なるORB製品へのサーバアプリケーションの移植が容易となります。

POAの特徴

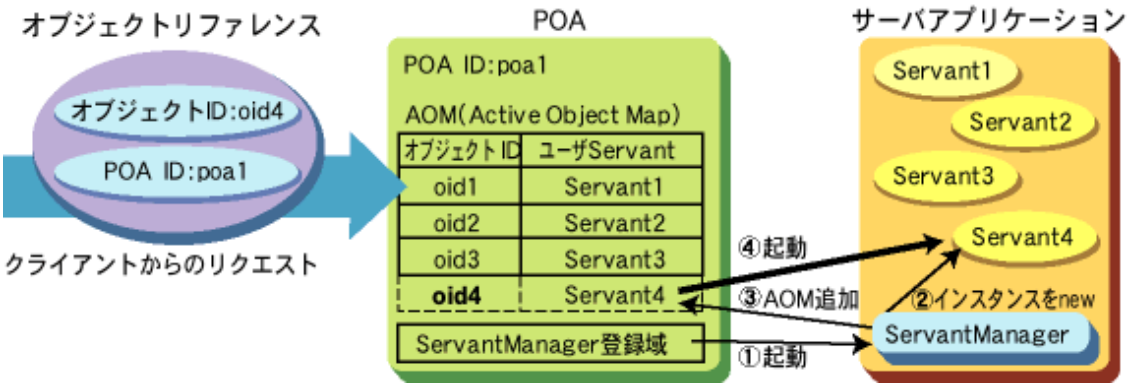
インスタンス管理

POAオブジェクト内にAOM(Active Object Map:インスタンスの管理テーブル)を持ち、インスタンス管理を行います。サーバアプリケーションは、以下の2とおりのインスタンス管理を行うことが可能です。

- サーバアプリケーションは、事前にインスタンスを作成してAOMに登録しておきます。クライアントからのリクエスト受信時に、AOMのインスタンスが検索され、Servantオブジェクトが実行されます。



- クライアントからのリクエスト受信時に、POAオブジェクトはユーザ実装のServantManagerを呼び出します。ServantManagerでは、インスタンスを作成してAOMに登録します。次に、Servantオブジェクトが実行されます。



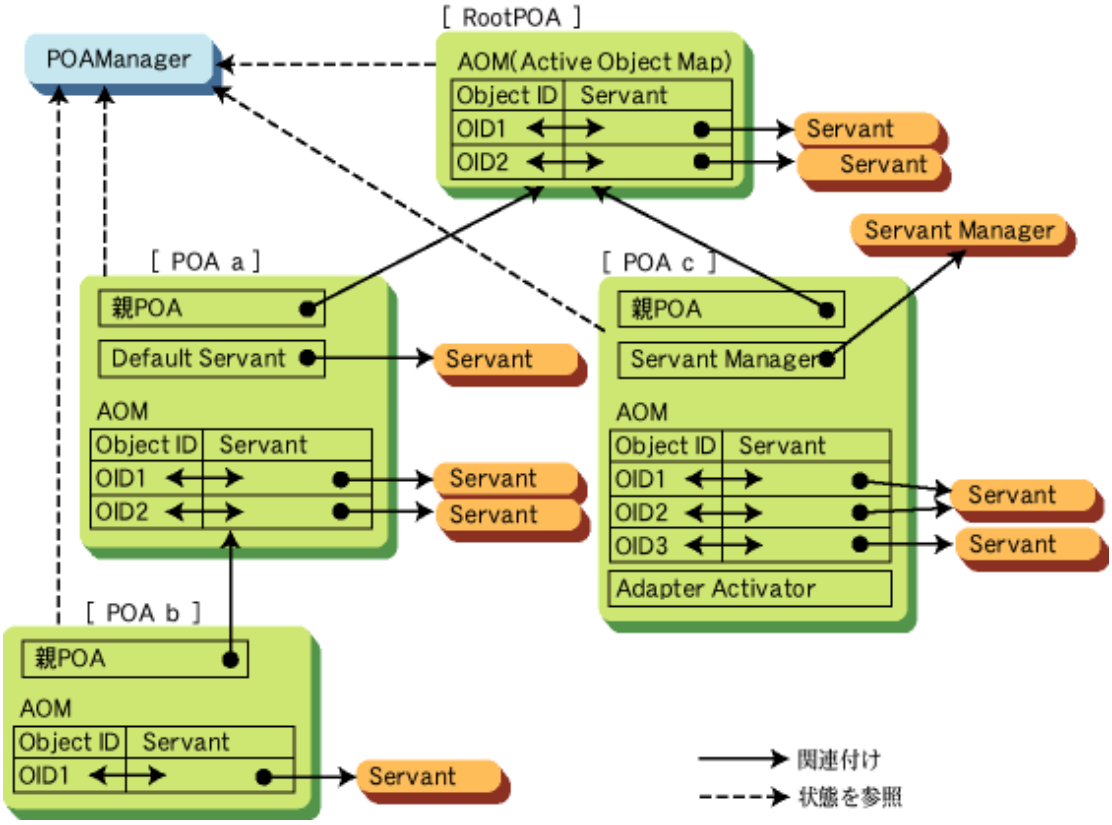
POAポリシ

POAに対してポリシを設定することで、POAごとにインスタンス管理方法、オブジェクトの生存期間、多重動作などのポリシを指定できます。POAポリシとして、以下の種類があります。
詳細については、“4.3.2.3 POAオブジェクト”を参照してください。

- Servant関連付けポリシ: ServantRetentionPolicy
- リクエスト処理ポリシ: RequestProcessingPolicy
- 暗黙的活性化ポリシ: ImplicitActivationPolicy
- ID割り当てポリシ: IdAssignmentPolicy
- オブジェクトID一意性ポリシ: IdUniquenessPolicy
- 生存期間ポリシ: LifespanPolicy
- スレッドポリシ: ThreadPolicy

4.3.2.2 POAのアーキテクチャ

POAは、以下のような仕組みを提供します。



POAオブジェクト

POAオブジェクトは、オブジェクトID、Servantオブジェクト(ユーザ提供のサーバ実装のインスタンス)の関係をAOM(Active Object Map)上で管理し、クライアントからの要求メッセージに従って、Servantオブジェクトのメソッドを実行します。

POAManagerオブジェクト

POAの要求メッセージ処理の受付けに関する状態を制御します。

Servantオブジェクト

サーバアプリケーションが提供するインタフェースを実装し、リクエストに対する処理を実際に行う、ユーザ作成のオブジェクトを示します。

Default Servantオブジェクト

Default Servantオブジェクトは、RequestProcessingポリシにUSE_DEFAULT_SERVANTが設定されています。さらに、リクエストに対応するServantオブジェクトがPOAのAOM(Active Object Map)に存在しない場合に、POAはDefault Servantオブジェクトのメソッドを実行します。

ServantManagerオブジェクト

ServantManagerオブジェクトはPOAオブジェクトにより呼び出され、Servantオブジェクトの作成および検索を行います。
ServantManagerオブジェクトは、RequestProcessingポリシにUSE_SERVANT_MANAGERが設定されている場合にPOAオブジェクトが呼び出しますが、以下の2種類があります。

- **ServantActivator**（ServantRetentionポリシがRETAINの場合）

  POAオブジェクトは、POAオブジェクトのAOM(ActiveObjectMap)に、リクエストに対応するServantオブジェクトが存在しない場合に、ServantActivatorオブジェクトにServantオブジェクトの生成を要求します。ServantActivatorオブジェクトはServantオブジェクトを返し、POAオブジェクトはServantオブジェクトのメソッドを実行します。
  このServantActivatorオブジェクトは、ユーザが作成する必要があります。

- **ServantLocator**（ServantRetentionポリシがNON_RETAINの場合）

  POAオブジェクトは、リクエスト受信時、ServantLocatorオブジェクトに対応するServantオブジェクトを問い合わせます。ServantLocatorオブジェクトはServantオブジェクトを返し、POAオブジェクトはServantオブジェクトのメソッドを実行します。
  このServantLocatorオブジェクトは、ユーザが作成する必要があります。また、対応するServantオブジェクトを管理する必要があります。

詳細については、“4.3.2.10 ServantManagerオブジェクト”を参照してください。

AdapterActivatorオブジェクト

リクエストの受信時、対応するPOAオブジェクトが存在しない場合に、POAオブジェクトを生成します。
このAdapterActivatorオブジェクトは、ユーザが作成する必要があります。

オブジェクトID

オブジェクトIDは、POAオブジェクトが対応するServantオブジェクトを特定するための値です。オブジェクトIDの値は、IdAssignmentポリシによって、POAオブジェクトによって割り当てられる場合と、ユーザアプリケーションにより割り当てられる場合があります。オブジェクトIDは、オブジェクトリファレンス中に含められるため、クライアントから意識する必要はありません。

POA ID

POA IDは、リクエストの受信時にPOAオブジェクトを特定するための値です。POA IDは、オブジェクトリファレンス中に含められるため、クライアントから意識する必要はありません。

4.3.2.3 POAオブジェクト

POAオブジェクト作成時は、まず、POAポリシリストを作成する必要があります。次に、ポリシリストを指定してPOAオブジェクトを作成します。

POAポリシ

POAオブジェクトの作成時に、以下のPOAポリシを設定できます。

POAポリシ	意味	Value	Valueの説明	備考
ServantRetention: Servant関連付けポリシ	AOMテーブルをPOAオブジェクト内で持つか持たないかを指定します。つまり、POAオブジェクト内でインスタンス管理を行うかどうかを指定できます。 “4.3.2.7 リクエスト処理”参照。	RETAIN (注1)	ActiveObjectMapにActiveなServantオブジェクトを憶えます。	
		NON_RETAIN	ActiveObjectMapを使いません。	
RequestProcessing: リクエスト処理ポリシ	リクエスト受信時に対象インスタンスをAOM内に持ってない、もしくはAOM自身を保持しないときの振る舞いを規定するポリシです。	USE_ACTIVE_OBJECT_MAP_ONLY (注1)	リクエスト処理にActiveObjectMapだけ使用します。	

POAポリシ	意味	Value	Valueの説明	備考
	“4.3.2.7 リクエスト処理”参照。	USE_DEFAULT_SERVANT	リクエストの処理をDefault Servantオブジェクトに任せます。	
		USE_SERVANT_MANAGER	ServantManagerオブジェクトによって目的のServantオブジェクトを見つけます。必要であればactivateも行われます。	
ImplicitActivation: 暗黙的活性化ポリシ	Servantの活性化のモードを以下のように指定可能です。 ・自動的にServantがactivateされます。 ・自動的にServantがactivateされません(アプリケーションが明示的にServantをPOAに登録します)。 “4.3.2.8 暗黙的活性化（Implicit Activation）”参照。	IMPLICIT_ACTIVATION (注1)	自動的にServantオブジェクトがactivateされます。	
		NO_IMPLICIT_ACTIVATION	自動的にServantオブジェクトがactivateされません(サーバアプリケーションが明示的にServantオブジェクトを生成して、POAオブジェクトに登録します)。	
IdAssignment: ID割り当てポリシ	オブジェクトIDをシステム(POAオブジェクト)が自動設定するか、ユーザが設定するかを指定できます。USER_IDはアプリケーションで、オブジェクトIDに固有の意味を持たせたいときに有効です。 “4.3.2.6 オブジェクトの活性化”参照。	SYSTEM_ID (注1)	POAオブジェクトがオブジェクトIDを付けます。	
		USER_ID	ユーザがオブジェクトIDを付けます。	
IdUniqueness: オブジェクトID一意性ポリシ	Servantにユニークなを付けるか、重複したIDを許すかを指定できます。 “4.3.2.5 オブジェクトリファレンス、オブジェクトID、Servantオブジェクトの関連”参照。	UNIQUE_ID (注1)	Servantオブジェクトにユニークなを付けます。	
		MULTIPLE_ID	Servantオブジェクトに重複したIDを許します。	
Lifespan: 生存期間ポリシ	トランジェントタイプのオブジェクトを扱うかパーシステントタイプのオブジェクトを扱うかを指定可能です。	TRANSIENT (注1)	トランジェントタイプのオブジェクトを扱います(サーバlifespan = オブジェクトlifespan)。	PERSISTENTは、指定不可
		PERSISTENT	パーシステントタイプのオブジェクトを扱います(サーバlifespan < オブジェクトlifespan)。	
Thread: スレッドポリシ	スレッド処理をORBに任せるかシングルスレッドで動作させるかを指定できます。	ORB_CTRL_MODEL (注1)	スレッド処理をORBに任せます。	このPOAポリシの設定値は、無効(注2)
		SINGLE_THREAD_MODEL	シングルスレッドで動作させます。	

注1)デフォルト値
注2)インプリメンテーションリポジトリ登録時のプロセス最大多重度(proc_conc_max)、スレッド初期多重度(thr_conc_init)の設定内容に依存します。

POAポリシの生成例を以下に示します。

例

```
// POAポリシリスト作成
CORBA::PolicyList   policies( 4 );  // POAのポリシリスト
CORBA::Environment  env;            // 例外情報

policies.length( 4 );
policies[0] = Poa->create_servant_retention_policy(
                  PortableServer::NON_RETAIN, env );
policies[1] = Poa->create_request_processing_policy(
                  PortableServer::USE_DEFAULT_SERVANT, env );
policies[2] = Poa->create_id_assignment_policy(
                  PortableServer::SYSTEM_ID, env );
policies[3] = Poa->create_id_uniqueness_policy(
                  PortableServer::MULTIPLE_ID, env );
```

注) ソース内のPoa:POAクラスのインスタンス

POAオブジェクトの生成

RootPOAオブジェクト

POAオブジェクトを使用するためには、まずRootPOAオブジェクトを生成する必要があります。POAオブジェクトは、RootPOAオブジェクトをルートとして、子POAオブジェクトや孫POAオブジェクトを生成することでツリー構成とすることができます。RootPOAオブジェクトは、ORBがサーバの初期化時に生成します。RootPOAオブジェクトのPOAオブジェクトは、以下のように取得できます。

例

```
main( int argc, char* argv[] )
{
    CORBA::ORB_ptr      orb;          // ORBのオブジェクトリフレンス
    CORBA::Object_ptr   obj;          // オブジェクトリファレンスの格納域
    CORBA::Environment  env;          // 例外情報
    int                 current_argc = argc;

    // ORBの生成と初期化
    orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

    // RootPOAオブジェクトのオブジェクトリファレンスの取得
    obj = orb->resolve_initial_references("RootPOA", env );

    // RootPOAオブジェクトのPOAオブジェクト獲得
    PortableServer::POA_ptr  rootPOA  =
                      PortableServer::POA::_narrow( obj );
```

RootPOAオブジェクトには、以下のPOAポリシが設定されています。

- ServantRetentionPolicy: RETAIN
- RequestProcessingPolicy: USE_ACTIVE_OBJECT_MAP_ONLY
- ImplicitActivationPolicy: IMPLICIT_ACTIVATION
- IdAssignmentPolicy: SYSTEM_ID
- IdUniquenessPolicy: UNIQUE_ID
- LifespanPolicy: TRANSIENT
- ThreadPolicy: ORB_CTRL_MODEL

ユーザアプリケーションで上記のPOAポリシを使用してインスタンス管理を行う場合は、RootPOAオブジェクトをそのまま使用することが可能です。しかし、新しいPOAオブジェクト(RootPOAオブジェクトの子孫POAオブジェクト)を生成し、固有のPOAポリシを設定することによりRootPOAオブジェクトとは異なるPOAポリシでインスタンス管理を行うことが可能となります。

子孫POAオブジェクト

新しいPOAオブジェクトを生成するには、すでに存在するPOAオブジェクト上でcreate_POA()メソッドを呼び出します。この結果、create_POA()を発行したPOAオブジェクトと新しく生成されたPOAオブジェクトは親子関係となります。アプリケーションで最初に生成するPOAオブジェクトはRootPOAオブジェクトであるため、その他のPOAオブジェクトはすべてRootPOAオブジェクトの子孫POAオブジェクトとなります。
create_POA()の引数としてPOAポリシリスト(Policyクラスの配列)を渡すことで、生成した子POAオブジェクトのポリシを設定することが可能です。

例

```
// 子POAオブジェクト(childPOA1)の生成
PortableServer::POA_ptr     childPOA1
        = rootPOA->create_POA( "poa_1", poamanager, policies1 );
// 孫POAオブジェクト(childPOA2)の生成
PortableServer::POA_ptr     childPOA2
        = childPOA1->create_POA("poa_2", poamanager, policies2 );
```

注)ソース内のpoa_1,poa_2:生成するPOAの名前
ソース内のpoamanager:POAManagerクラスのインスタンス
policies1/policies2:Policyクラスのインスタンス

上記のように、POAオブジェクトは階層構造をもたせることが可能です。オブジェクトの生成、オブジェクトと管理するPOAオブジェクトとの関係については、"4.3.3 サーバアプリケーションと環境設定との関連付け"を参照してください。destroy()メソッドによりあるPOAオブジェクトが破棄された場合、その子孫POAオブジェクトはすべて破棄されます。

POAポリシの設定値の組合せ

ポリシリストに設定したPOAポリシの各設定値は、create_POA()メソッド発行時にそれらの組合せが妥当であるかチェックされます。もし、設定値の組合せが妥当でない場合は、例外が発生します。
以下にPOAポリシの設定値の組合せを示します。妥当性欄で"*"の部分はこの値が設定された場合、create_POA()メソッド実行時にPortableServer::POA::InvalidPolicy例外が発生することを示しています。
なお、LifespanPolicyの設定値は、TRANSIENTとしてください。また、ThreadPolicyの設定値は、無視されます。

ポリシ	Servant Retention	Request Processing	Implicit Activation	IdAssignment	IdUniqueness	妥当性
設定値	RETAIN	USE_ACTIVE_OBJECT_MAP_ONLY	IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	

ポリシ	Servant Retention	Request Processing	Implicit Activation	IdAssignment	IdUniqueness	妥当性
					MULTIPLE_ID	
				USER_ID	―	*
			NO_IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	
					MULTIPLE_ID	
		USE_DEFAULT_SERVANT	IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	*
					MULTIPLE_ID	
				USER_ID	―	*
			NO_IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	*
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	*
					MULTIPLE_ID	
		USE_SERVANT_MANAGER	IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	―	*
			NO_IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	
					MULTIPLE_ID	
	NON_RETAIN	USE_ACTIVE_OBJECT_MAP_ONLY	―	―	―	*
		USE_DEFAULT_SERVANT	― (注)	SYSTEM_ID	UNIQUE_ID	*
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	*
					MULTIPLE_ID	
		USE_SERVANT_MANAGER	― (注)	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	

ポリシ	Servant Retention	Request Processing	Implicit Activation	IdAssignment	IdUniqueness	妥当性
					MULTIPLE_ID	

注) ServantRetentionPolicyにNON_RETAINが設定されている場合、ImplicitActivationPolicyの設定値は無視されます。

POAポリシのデフォルト設定値

POAは、デフォルトのポリシ設定値として以下の値を持っています。

ポリシ	デフォルト値
ServantRetention	RETAIN
RequestProcessing	USE_ACTIVE_OBJECT_MAP_ONLY
ImplicitActivation	NO_IMPLICIT_ACTIVATION
IdAssignment	SYSTEM_ID
IdUniqueness	UNIQUE_ID
Lifespan	TRANSIENT
Thread	ORB_CTRL_MODEL

create_POA()メソッドを使用して新しいPOAを生成する際は、デフォルトの設定値から変更したいポリシだけ設定することが可能です。以下の例では、USE_DEFAULT_SERVANTにRequestProcessingPolicyを、MULTIPLE_IDにIdUniquenessPolicyを設定した子POAを生成しています。この場合、他のポリシの設定値は、上記のデフォルトの設定値のままとなります。

例

```
CORBA::Environment  env;    // 例外情報

// POAポリシリスト作成
CORBA::PolicyList   policies( 2 )
policies[0] =  Poa->create_request_processing_policy(
                   PortableServer::USE_DEFAULT_SERVANT, env );
policies[1] =  Poa->create_id_uniqueness_policy(
                   PortableServer::MULTIPLE_ID, env );

// 子POAオブジェクトの生成
PortableServer::POA_ptr  Poa1
    = Poa->create_POA( "poa_1", poamanager, policies );
```

注)ソース内のPoaは:POAクラスのインスタンス
　ソース内のpoamanager:POAManagerクラスのインスタンス

なお、create_POA()メソッドの第3パラメタにnullを指定した場合は、生成される子POAのポリシはデフォルトの設定値となります。

4.3.2.4 オブジェクトリファレンスの生成

オブジェクトリファレンスは、以下の方法で生成します。

Servantオブジェクトの活性化前に作成する方法(直接作成)

サーバアプリケーションは、直接オブジェクトリファレンスをPOAオブジェクトのcreate_reference()やcreate_reference_with_id()メソッドで作成できます。この操作は、オブジェクトリファレンスだけを作成します。実際のアクティブなServantオブジェクトは、後からServantManagerオブジェクトで生成される場合もあります。

例

```
CORBA::Environment  env;   // 例外情報

// オブジェクトリファレンスの生成
CORBA::Object  _bind_obj =
          Poa->create_reference( "IDL:Intfid11:1.0", env );

// ID指定によるオブジェクトリファレンスの生成
long  len = strlen( "USERID" );
CORBA::Octet*  buf = PortableServer::ObjectId::allocbuf( len );
memcpy( buf, "USERID", len );
PortableServer::ObjectId_var  userid =
          new PortableServer::ObjectId( len, len, buf, CORBA_TRUE );
CORBA::Object  _bind_obj =
          Poa->create_reference_with_id( *userid, "IDL:Intfid11:1.0", env );
```

注)ソース内のPoa:POAクラスのインスタンス
IDL:Intfid11:1.0:インタフェースリポジトリID

Servantオブジェクトの活性化後に作成する方法(間接作成)

サーバアプリケーションは、activate_object()／activate_object_with_id()メソッドよりServantオブジェクトを活性化します。一度Servantオブジェクトが活性化されると、サーバアプリケーションはServantオブジェクト／オブジェクトIDを指定して、servant_to_reference()／id_to_reference()メソッドを発行することにより容易にオブジェクトリファレンスを作成できます。また、POAオブジェクトがIMPLICIT_ACTIVATIONポリシで作成されていれば、servant_to_reference()を使用してオブジェクトリファレンスを生成した場合、自動的にServantオブジェクトがactivateされ、オブジェクトリファレンスが有効となります。

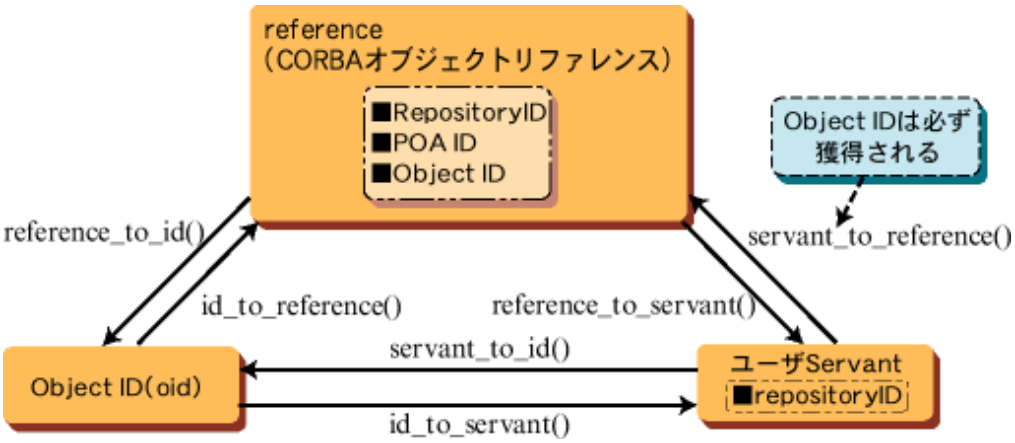
例

```
// Servantの生成
ODsample_intf_impl*  svt = new ODsample_intf_impl();

// IMPLICIT_ACTIVATIONポリシを指定している場合、自動的にactivateされる
CORBA::Object_ptr obj =  Poa->servant_to_reference( svt, env );
```

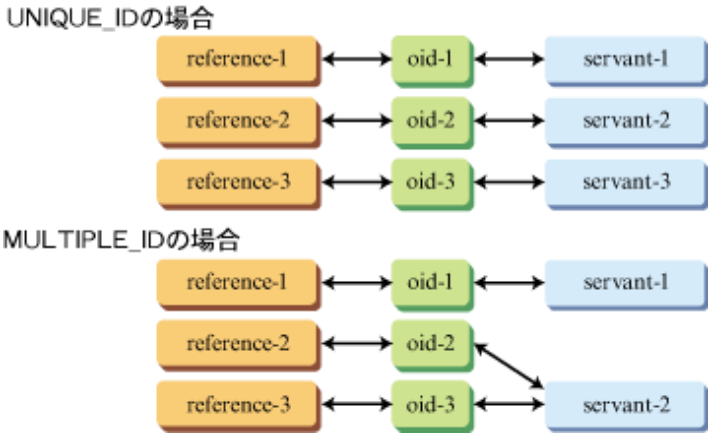
注) ソース内のPoaオブジェクト:POAクラスのインスタンス

一度オブジェクトリファレンスがクライアントに通知されると、クライアントの側から見るとそのオブジェクトリファレンスはオブジェクトの身元を持っていることになります。クライアントプログラムがそのオブジェクトリファレンスを使う限り、そのオブジェクトリファレンスで作られるリクエストは同一のオブジェクトのインスタンスに送信されます。

4.3.2.5 オブジェクトリファレンス、オブジェクトID、Servantオブジェクトの関連



オブジェクトリファレンス、オブジェクトID(oid)、Servantオブジェクトの関係は、IdUniquenessPolicyによって決まります。オブジェクトリファレンスは、oidを包含しているため、referenceとoidは、1対1の関係となります。oidとServantオブジェクトの関係は、UNIQUE_IDの場合では、Servantオブジェクトごとに一意のoidが割り当てられます。MULTIPLE_IDの場合は、同じServantオブジェクトへの複数のoidが割り当て可能です。このことは、具体的にはoidを生成するservant_to_id()／servant_to_reference()の動作に影響を与えます。



4.3.2.6 オブジェクトの活性化

オブジェクトが活性状態とは、POAオブジェクトの管理するAOM(Active Object Map)に、あるServantオブジェクトがオブジェクトIDと関連付けられて登録されている状態のことを指します。
Servantオブジェクトは、インスタンスとして生成されただけでは、POAオブジェクトに認識されないため、activate_object()／activate_object_with_id()メソッドを使用して、オブジェクトIDとの関連付けをPOAオブジェクトに登録する必要があります。Servantオブジェクトを活性化する方法を以下に示します。

明示的にServantオブジェクトを活性化する方法

Servantオブジェクトの作成(new)後に、activate_object()／activate_object_with_id()メソッドでAOMに登録できます。

例

```
CORBA::Environment  env;  // 例外情報

// Servantの生成
```

```
ODsample_intf_impl*  svt = new ODsample_intf_impl();

// Servantの活性化
Poa->activate_object ( svt, env );

// ID指定によるServantの活性化
long  len = strlen( "USERID" );
CORBA::Octet*  buf = PortableServer::ObjectId::allocbuf( len );
memcpy( buf, "USERID", len );
PortableServer::ObjectId_var  userid =
          new PortableServer::ObjectId( len, len, buf, CORBA_TRUE );
Poa->activate_object_with_id( *userid, svt, env );
```

注) ソース内のPoaオブジェクト:POAクラスのインスタンス

なお、活性化の際にはオブジェクトIDが必要となりますが、オブジェクトIDの割付けは、IdAssignmentPolicyに従って処理されます。

- SYSTEM_IDの場合
  POAオブジェクトが自動的にオブジェクトIDを割り付けます。Servantオブジェクトを活性化する場合に、activate_object()を使用してください。
- USER_IDの場合
  ユーザがオブジェクトIDを割り付けます。Servantオブジェクトを活性化する場合は、activate_object_with_id()を使用してください。

暗黙的にServantオブジェクトを活性化する方法

POAオブジェクトがIMPLICIT_ACTIVATIONポリシを持っている場合、servant_to_reference()メソッドを使用してリファレンスを生成すると、自動的にServantオブジェクトが活性化されて、オブジェクトリファレンスが通知されます。暗黙的活性化については、“4.3.2.8 暗黙的活性化(Implicit Activation)”を参照してください。

例

```
CORBA::Environment  env;  // 例外情報

// Servantの生成
ODsample_intf_impl* svt = new ODsample_intf_impl();

// IMPLICIT_ACTIVATIONポリシを指定している場合、自動的にactivateされる
CORBA::Object Obj =  Poa->servant_to_reference( svt, env );
```

注) ソース内のPoaオブジェクト:POAクラスのインスタンス

なお、暗黙的活性化の場合、オブジェクトIDはPOAオブジェクトにより自動的に割り付けます。このため、IdAssignmentPolicyは、SYSTEM_IDに設定しておく必要があります。

4.3.2.7 リクエスト処理

POAオブジェクトの検索

クライアントからのリクエスト受信時に、インタフェースリポジトリIDに対応するPOAオブジェクトがサーバプロセス上に存在しなければ、親のPOAオブジェクトはAdapterActivatorオブジェクトを呼び出します。AdapterActivatorオブジェクトでは、要求されたPOAオブジェクトを作成します。

AdapterActivatorオブジェクトは、ユーザが作成する必要があります。また、事前にPOAオブジェクトに登録しておく必要があります。もし、AdapterActivatorオブジェクトが登録されていない場合、クライアントはOBJECT_NOT_EXIST例外を受け取ります。

Servantオブジェクトの検索

クライアントからのリクエスト受信時に、POAオブジェクトはAOMに登録されているServantオブジェクトを検索します。なお、AOMを使用するかどうかは、対象のPOAオブジェクトがRETAINポリシ(使用する)とNON_RETAINポリシ(使用しない)のどちらが指定されているかによります。
RETAINポリシが指定され、目的のServantオブジェクトがAOM内にみつかった場合、検索処理はそこで終了してServantオブジェクトのメソッドを実行します。Servantオブジェクトがみつからなかった場合や、NON_RETAINの場合は、以下のRequestProcessingポリシに従って処理を行います。

- **USE_DEFAULT_SERVANT**
  対象のServantオブジェクトがみつからなかった場合、デフォルトのServantオブジェクトに処理を任せる方法です。この場合、あらかじめset_servant()によりデフォルトのServantオブジェクトを登録しておく必要があります。もし、Default Servantオブジェクトが登録されていなければ、OBJ_ADAPTERエラーとなります。

- **USE_SERVANT_MANAGER**
  対象のServantオブジェクトがみつからなかった場合、ServantManagerオブジェクトにServantオブジェクトを検索する処理を任せる方法です。
  リクエストを扱えるServantオブジェクトを見つけるため、POAオブジェクトは、incarnate()／preinvoke()を呼び出します（メソッド選択は、POAオブジェクトのNON_RETAIN／RETAINポリシに依存します）。ServantManagerオブジェクトは、見つけたServantオブジェクトをPOAオブジェクトに通知します。この場合、あらかじめ、set_servant_manager()によりServantManagerオブジェクトを登録しておく必要があります。もし、ServantManagerオブジェクトが登録されていなければ、OBJ_ADAPTERエラーとなります。

- **USE_ACTIVE_OBJECT_MAP_ONLY**
  この場合、OBJECT_NOT_EXISTエラーとなります。

4.3.2.8 暗黙的活性化（Implicit Activation）

Servantオブジェクトの活性化方法には、アプリケーションが明示的にServantオブジェクトを活性化させる方法に加え、POAオブジェクトにより暗黙的（Implicit）にServantオブジェクトを活性化させるIMPLICIT_ACTIVATIONポリシがあります。
IMPLICIT_ACTIVATIONポリシを使用するためには、SYSTEM_IDポリシおよびRETAINポリシを指定する必要があります。IMPLICIT_ACTIVATIONポリシを指定することにより、servant_to_reference()／servant_to_id()などのServantオブジェクトからオブジェクトリファレンスやオブジェクトIDに変換するメソッドを使用した際に、POAオブジェクトがAOMにServantオブジェクトおよびオブジェクトIDを登録し、Servantオブジェクトを活性化できます。このときにUNIQUE_IDポリシが指定されていた場合、再度、servant_to_reference()／servant_to_id()を使用した場合、前回と同じリファレンスやオブジェクトIDが返されます。MULTIPLE_IDポリシが指定されていた場合、毎回、異なったオブジェクトリファレンスやオブジェクトIDが返されます。これは、同じServantオブジェクトに複数のオブジェクトリファレンスやオブジェクトIDが割り付けられることを意味します。

4.3.2.9 POAManagerオブジェクト

POAManagerオブジェクトは、POAオブジェクトの生成時、create_POA()メソッドの引数として渡すことにより、POAオブジェクトと関連付けられます。POAManagerは、以下の4つの状態を持っています。POAオブジェクトは、POAManagerの状態に従って処理を決定します。この状態は、activate()、hold_requests()、discard_requests()、deactivate()などのメソッドで変更できます。

holding状態

受け取ったメッセージをキューイングします。

active状態

受け取ったメッセージを処理します。

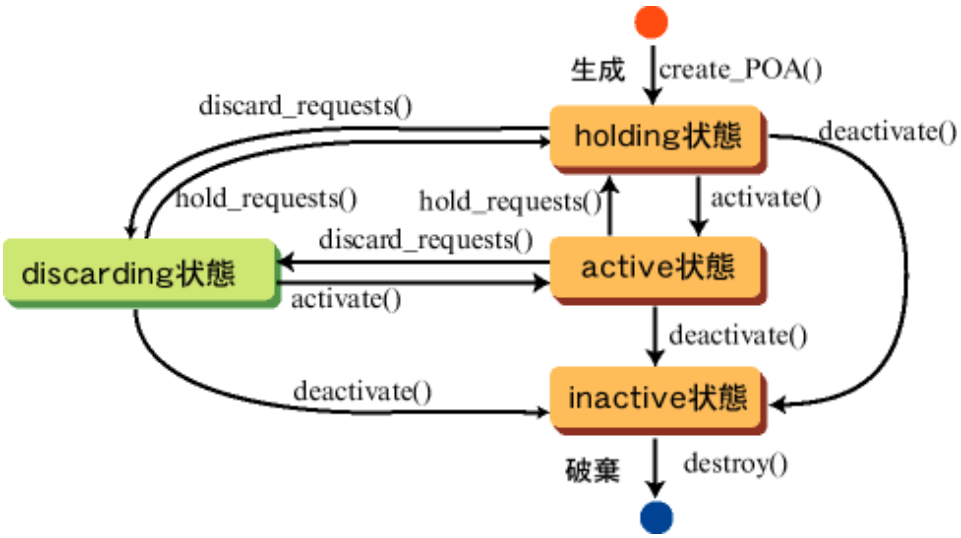
discarding状態

受け取ったメッセージを捨てます。この場合、クライアントへはエラーが通知されます。

inactive状態

shutdown直前の状態。

POAオブジェクトの状態遷移を以下に示します。



4.3.2.10 ServantManagerオブジェクト

ServantManagerオブジェクトは、リクエスト受信時に、Servantオブジェクトの生成／削除／検索を可能にする機能です。ServantManagerオブジェクトの各メソッドの実装は、ユーザ自身で作成します。作成したServantManagerオブジェクトは、POAオブジェクトに登録しておきます。
POAオブジェクトがリクエストを受信した際、AOMにオブジェクトIDが一致するServantオブジェクトが登録されていない場合、POAオブジェクトはServantManagerオブジェクトのメソッドを呼び出し、Servantオブジェクトの生成／削除／検索を依頼します。
ServantManagerオブジェクトは、以下のの2種類が存在します。

- Servantオブジェクトの生成／削除を行うServantActivatorオブジェクト
- Servantオブジェクトを検索するServantLocatorオブジェクト

POAオブジェクトは、以下のオブジェクトを呼び出します。

- ServerRetentionポリシがRETAINの場合：ServantActivatorオブジェクト
- ServerRetentionポリシがNON_RETAINの場合：ServantLocatorオブジェクト

(1)ServantActivatorオブジェクト

作成するServantActivatorクラスは、以下のようにPortableServer::ServantActivatorクラスを継承させる必要があります。また、メソッドとしてincarnate()／etherealize()を実装している必要があります。

【ServantActivatorクラスの実装形式】

```
class UserServantActivator : public PortableServer::ServantActivator
{
public :
    PortableServer::Servant incarnate(...) {...}
    void etherealize(...) {...}
};
```

POAオブジェクトは、以下のメソッドを呼び出します。

- Servantオブジェクトの生成を依頼する場合：ServantActivatorオブジェクトのincarnate()メソッド
- Servantオブジェクトの削除を依頼する場合：ServantActivatorオブジェクトのetherealize()メソッド

POAオブジェクトから渡されるパラメタは、必要に応じて各メソッドの実装部で使用できます。

- **incarnate (Servantオブジェクト生成)**
  POAは、リクエスト受信時にAOMに対象のServantオブジェクトが未登録の場合、incarnate()メソッドを呼び出します。incarnate()メソッドは、POAオブジェクトにServantオブジェクトを生成させ返します。

```
PortableServer::Servant  incarnate(
    const PortableServer::ObjectId&   oid,
    PortableServer::POA_ptr           adapter,
    CORBA::Environment                env )
```

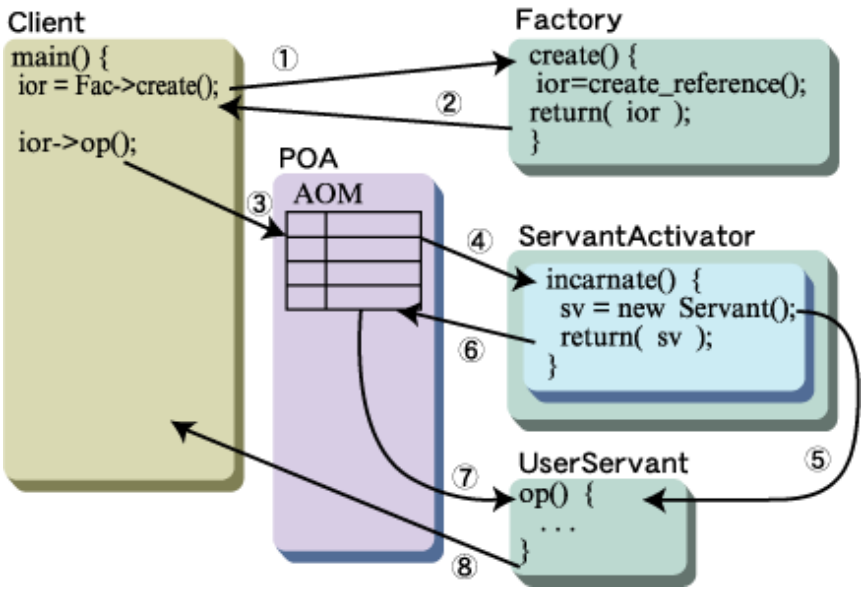
  oid：リクエスト対象のオブジェクトのオブジェクトID
  adapter：呼び出し元のPOAオブジェクト

- **etherealize (Servantオブジェクト削除)**
  POAは、アプリケーションからdeactivate_object()メソッドが呼び出されたときに、etherealize()メソッドを呼び出します。POAオブジェクトが削除するServantオブジェクトをパラメタで指定します。etherealize()メソッドは、指定されたServantオブジェクトを削除します。

```
void etherealize(
    const PortableServer::ObjectId&  oid,
    PortableServer::POA_ptr          adapter,
    PortableServer::Servant          serv,
    CORBA::Boolean                   cleanup_in_progress,
    CORBA::Boolean                   remaining_activations,
    CORBA::Environment               env );
```

  oid：削除対象のServantオブジェクトに対応するオブジェクトID
  adapter：呼び出し元のPOAオブジェクト
  serv：削除対象のServantオブジェクト
  cleanup_in_progress：POA->destoroy()から本メソッドが呼ばれた場合にtrue
  remaining_activations：本メソッド呼出し時、常にfalse

ServantActivatorオブジェクトの具体的な処理の流れを以下に示します。



1. FactoryにServantのオブジェクトリファレンスの生成を依頼します。(注)
2. Factoryで生成したオブジェクトリファレンスをクライアントに返します。
3. Servantに、オペレーションop()を要求します。
4. POAでは、AOMに未登録の要求であるため、ServantActivatorを呼び出します。
5. ServantActivatorでは、該当するServantを検索して返します。
6. ServantActivatorからServantを受け取り、AOMに登録します。
7. Servantに対してオペレーションop()の実行を依頼します。
8. クライアントに、オペレーションop()の結果を通知します。

注) Factoryは、リクエストごとにオブジェクトリファレンスやインスタンスを作成するオブジェクトです。インスタンス管理を行う場合に必要となるオブジェクトです。

(2)ServantLocatorオブジェクト

作成するServantLocatorクラスは、以下のようにPortableServer::ServantLocatorクラスを継承させる必要があります。また、メソッドとしてpreinvoke()／postinvoke()を実装している必要があります。

【ServantLocatorクラスの実装形式】

```
class UserServantLocator : public PortableServer::ServantLocator
{
public :
    PortableServer::Servant preinvoke(...) {...}
    void postinvoke(...) {...}
};
```

POAオブジェクトはクライアントからの要求を処理するため、毎回ServantLocatorオブジェクトのpreinvoke()メソッドを呼び出し、処理対象のServantオブジェクトを獲得します。POAオブジェクトはそのServantオブジェクトに処理を要求し、その処理が終わったら、ServantLocatorオブジェクトのpostinvoke()メソッドを呼び出し、ServantLocatorオブジェクトに後処理を依頼します。
POAオブジェクトから渡されるパラメタは、必要に応じて各メソッドの実装部で使用できます。

- **preinvoke**
  POAオブジェクトは、処理を要求すべきServantオブジェクトを獲得するために、ServantLocatorオブジェクトのpreinvoke()メソッドを呼び出します。preinvoke()メソッドは、POAオブジェクトにServantオブジェクトを返すと同時に、postinvoke()のための情報(cookie)も返します。

```
PortableServer::Servant  preinvoke(
    const PortableServer::ObjectId          oid,
    PortableServer::POA_ptr                 adapter,
    CORBA::Identifier                       operation,
    PortableServer::ServantLocator::Cookie  the_cookie,
    CORBA::Environment& = CORBA::Environment() )
```

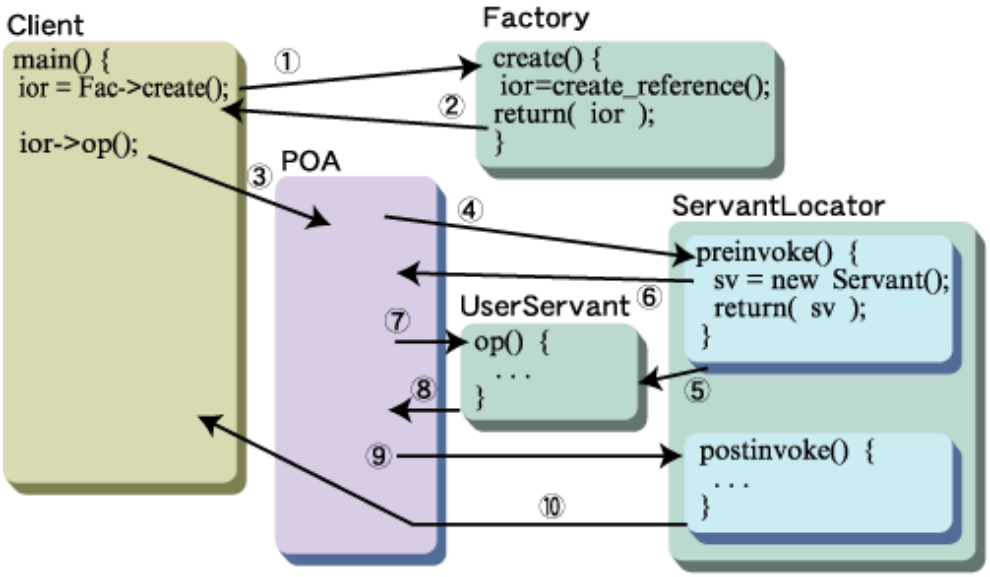
  oid:リクエスト対象のオブジェクトのオブジェクトID
  adapter:呼び出し元のPOAオブジェクト
  operation:リクエスト対象のオペレーションの名前
  the_cookie:Cookieオブジェクトを格納する

- **postinvoke**
  POAオブジェクトは、Servantオブジェクトの要求処理の終了直後に、ServantLocatorオブジェクトのpostinvoke()を呼び出します。このとき、preinvoke()との対応をとるため、preinvoke()で返されたcookieを指定します。

```
void  postinvoke(
    const PortableServer::ObjectId  oid,
    PortableServer::POA_ptr         adapter,
    CORBA::Identifier               operation,
    CORBA::Object                   the_cookie,
    PortableServer::Servant         the_servant,
    CORBA::Environment& = CORBA::Environment() )
```

  oid:リクエスト対象のオブジェクトのオブジェクトID
  adapter:呼び出し元のPOAオブジェクト
  operation:リクエスト対象のオペレーションの名前
  the_cookie:preinvokeで返されたCookieオブジェクト
  the_servant:リクエストに対する処理を行ったServantオブジェクト

ServantLocatorの具体的な処理の流れを以下に示します。



1. Factoryに、Servantのオブジェクトリファレンスの生成を依頼します。
2. Factoryで生成したオブジェクトリファレンスをクライアントに返します。
3. Servantに、オペレーションop()を要求します。
4. POAでは、ServantLocatorを呼び出します。
5. ServantLocatorのpreinvoke()でServantを検索します。
6. Servantを受け取ります。
7. Servantに対してオペレーションop()の実行を依頼します。
8. クライアントに、オペレーションop()の結果を通知します。
9. ServantLocatorのpostinvoke()を呼び出します。
10. クライアントへオペレーションを復帰します。

4.3.2.11 AdapterActivatorオブジェクト

AdapterActivatorオブジェクトは、リクエストを受信時にPOAオブジェクトを生成するために使用します。POAオブジェクトが生成したオブジェクトリファレンスには、オブジェクトIDに加え、生成元POAオブジェクトを示すPOAidが埋め込まれています。クライアントからのリクエストが送られてきた際、POA idに対応するPOAオブジェクトが検索され、そのPOAオブジェクトがリクエストに対する処理を行います。もし、みつからない場合、検索対象のPOAの親POAオブジェクトにAdapterActivatorオブジェクトが設定されているならば、そのAdapterActivatorオブジェクトに子POAオブジェクトの生成を依頼します。親POAにAdapterActivatorオブジェクトが設定されていない場合は、CORBA::StExcep::OBJ_ADAPTER例外がクライアントに通知されます。
AdapterActivatorオブジェクトは、ユーザが作成し、POAに登録する必要があります。
作成するAdapterActivatorクラスは、PortableServer::AdapterActivatorクラスを継承させる必要があります。また、メソッドとしてunknown_adapter()を実装している必要があります。

【AdapterActivatorクラスの実装形式】

```
class UserAdapterActivator : public PortableServer::AdapterActivator {
public :
    CORBA::Boolean unknown_adapter(...){...}
}
```

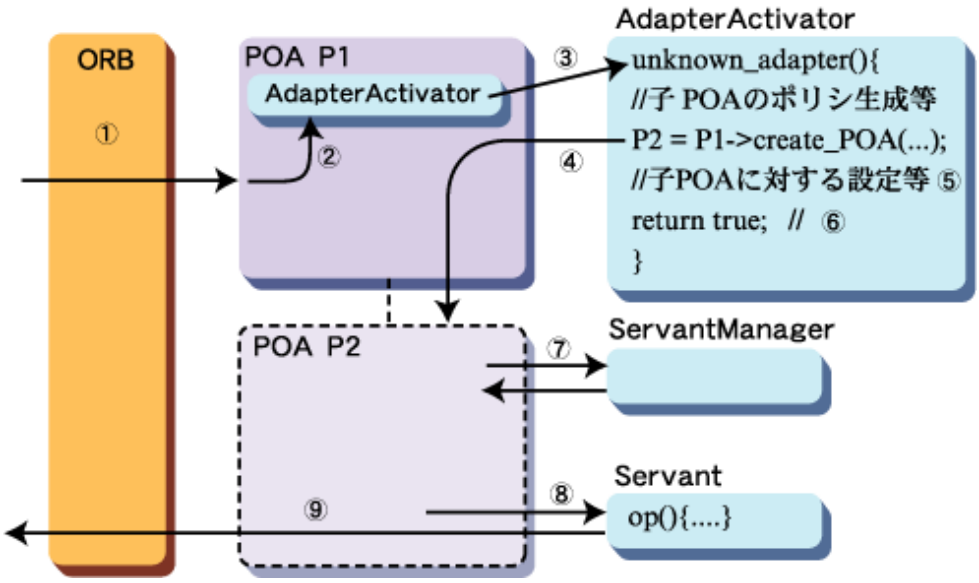
- **unknown_adapter**
  POAidに対応するPOAが検出できなかった場合は、AdapterActivatorオブジェクトのunknown_adapter()メソッドが呼び出されます。
  unknown_adapter()には、パラメタとしてPOAオブジェクトとStringオブジェクトが渡されます。
  unknown_adapter()内では、渡されたPOAオブジェクトの子POAをcreate_POA()を使用して生成します。このときのアダプタ名として、渡されたStringオブジェクトを使用します。
  子POAに対するポリシ設定や、その他必要な処理(ServantやServantManagerの生成・登録など)もこのメソッド内に記述することが可能です。
  子POAの生成およびその他の実装した処理が成功した場合は、戻り値としてtrueを返します。そうでない場合は、falseを返します。trueを返した場合は、生成した子POAからServantのオペレーションが起動されます。falseを返した場合は、CORBA::StExcep::OBJECT_NOT_EXIST例外がクライアントに通知されます。

```
CORBA::Boolean unknown_adapter(
    PortableServer::POA_ptr  parent,
    CORBA::Char*             name,
    CORBA::Environment&      env );
```

  parent:生成するPOAの親POA
  name:生成するPOAのアダプタ名

AdapterActivatorオブジェクトのunknown_adapterメソッドは、PortableServer::POA::find_POAメソッド発行時に呼び出すことも可能です。

AdapterActivatorの処理の流れを以下に示します。



1. クライアントからのリクエストを受信します。リクエストの対象となるオブジェクトリファレンスは、POAidを内包しています。
2. POAidに対応するPOAオブジェクトを検索します。存在しない場合は、既存のPOAから対象のPOAの親POAを検索します。
3. 親POAオブジェクトに設定されているAdapterActivatorクラスのインスタンスに対して、unknown_adapeter()メソッドが発行されます。
4. unknown_adapeter()の実装部では、パラメタで渡されたPOA（親POA）の子POAオブジェクトを生成します。以降、同じオブジェクトリファレンスに対するリクエストの処理は、このPOAが行います。
5. 生成した子POAに対する設定などを行います。ここでは、直接Servantを生成して子POAに登録することも可能であり、ServantManagerを生成し登録しておくことも可能です。
6. 子POAの生成に成功した場合は、trueを返します。
7. Servantの生成／登録を行います（ServantManager使用時）。
8. Servantのオペレーションを起動します。
9. オペレーションの実行結果を返します。

4.3.3 サーバアプリケーションと環境設定との関連付け

インプリメンテーション情報を登録するためのOD_impl_instコマンドとネーミングサービスへ登録するためのOD_or_admコマンドがあります。ここでは、サーバアプリケーションとOD_impl_instコマンド／OD_or_admコマンドとの関連付けについて説明します。

4.3.3.1 インプリメンテーション情報との関連付け

POAオブジェクトを使用したサーバアプリケーションを動作させるためには、OD_impl_instコマンドに指定したインプリメンテーションリポジトリIDとの関連付けを行う必要があります。この関連付けを行うには、以下の2とおりの方法があります。

1. 環境変数OD_IMPLIDへの指定
   プログラムの外からインプリメンテーションリポジトリIDを指定できるのでプログラムの独立性が高まります。
2. POAオブジェクトのアダプタ名への指定
   プログラムごとの環境変数の指定が不要であり、環境設定が容易となります。

(1)環境変数OD_IMPLIDへの指定

サーバアプリケーションの実行環境では、環境変数OD_IMPLIDへインプリメンテーションリポジトリID名を設定することで可能となります。

例

Windows32/64

バッチファイル(xxx.bat)でサーバアプリケーションを起動する場合

```
set OD_IMPLID=Implid1  ← OD_IMPLIDにインプリメンテーションリポジトリ設定
       :
simple_s.exe           ← サーバアプリケーションを起動
```

Solaris32/64 Linux32/64

シェルでサーバアプリケーションを起動する場合

```
OD_IMPLID=Implid1    ← OD_IMPLIDにインプリメンテーションリポジトリ設定
export OD_IMPLID
       :
simple_s             ← サーバアプリケーションを起動
```

(2)POAオブジェクトのアダプタ名への指定

環境変数OD_IMPLIDを設定しないで、アプリケーション上でPOAオブジェクトとインプリメンテーション情報との関連付けを明示的に行うことが可能です。この方式では、RootPOAオブジェクト直下の子POAオブジェクトのアダプタ名にインプリメンテーションリポジトリIDを使用することで可能となります。

4.3.3.2 オブジェクトリファレンスの生成方法

オブジェクトリファレンスの生成およびネーミングサービスへの登録には、以下の2とおりの方法があります。

- 動的生成
- 事前生成

動的生成

POAオブジェクト上(サーバアプリケーション内)でPOAオブジェクトのcreate_reference()メソッドにより実行時に生成します。作成したオブジェクトリファレンスは アプリケーション内でネーミングサービスへ登録するか、またはパラメタとしてクライアントアプリケーションに通知することにより、クライアントアプリケーションへ渡されます。
この方法はクライアントからの要求ごとにオブジェクトリファレンスを作成できるため、クライアントごとのインスタンス管理を動的に行うことができます。
“4.3.1.2 各アプリケーション形態”に示す「Factory-1方式」、「Factory-2方式」、「ユーザインスタンス管理法式」のServantオブジェクトのオブジェクトリファレンスは、動的生成で作成しています。
なお、オブジェクトリファレンスの動的生成方法については、“4.3.2.4 オブジェクトリファレンスの生成”を参照してください。

事前生成

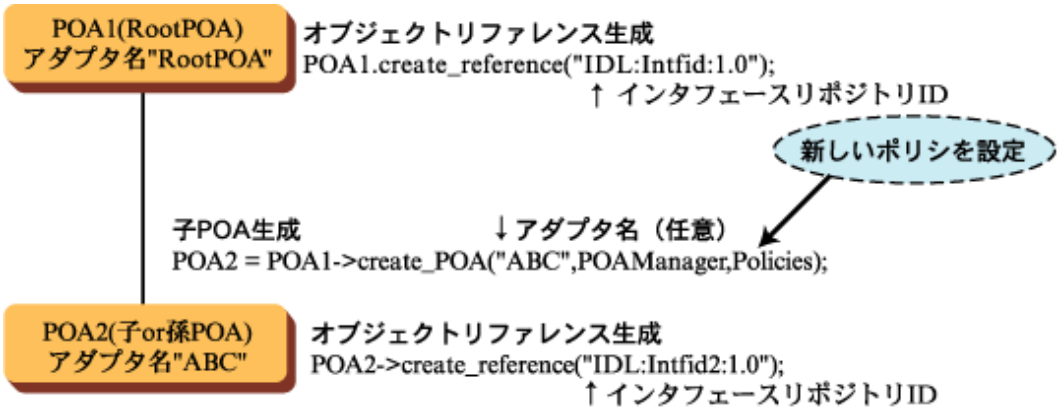
OD_or_admコマンドを使用し、事前にオブジェクトリファレンスの生成とネーミングサービスへの登録を行います。
この方法は事前にオブジェクトリファレンスを登録しておくため、動的生成に比較して性能／資源面で優れています。
“4.3.1.2 各アプリケーション形態”に示す「デフォルトインスタンス方式」のServantオブジェクトのオブジェクトリファレンス、および「Factory-1

方式」、「Factory-2方式」、「ユーザインスタンス管理法式」のFactoryオブジェクトのオブジェクトリファレンスは、事前生成で作成しています。

4.3.3.3 POA使用方法例

サーバアプリケーションが上記のどちらの方法を使用するかにより、POAオブジェクトの構成方法が異なります。上記のそれぞれの場合について、POAオブジェクトの使用方法を説明します。

POAオブジェクト使用方法1：動的生成／環境変数OD_IMPLID使用



POAオブジェクトの構成方法として一般的な方法です。

1. POAを生成（任意のアダプタ名）します。
2. create_referenceメソッドで、指定したインタフェースをもつオブジェクトリファレンスを生成します。
3. Servantを作成します。
4. 生成したオブジェクトリファレンスのオブジェクトIDを取得します。
5. POAのAOMに、Servantのインスタンスとオブジェクトの関連付けを登録します（活性化）。
6. ネーミングサービスへオブジェクトリファレンスを登録(bind)します。

上図では、インタフェースリポジトリID“IDL:Intfid1:1.0”のインタフェースをもつオブジェクトのインスタンスは、POA1オブジェクト(RootPOA)上で管理されます。インタフェースリポジトリID“IDL:Intfid2:1.0”のインタフェースをもつオブジェクトのインスタンスは、POA2オブジェクト(子／子孫POAオブジェクト)上で管理されます。

RootPOAオブジェクトで複数のオブジェクトを管理することも可能ですが、上記のような構成とすることにより、インタフェースごとに異なるPOAポリシで管理することが可能となります。

アプリケーションを作成する場合は、create_reference()の代わりに、同じくPOAオブジェクトのメソッドであるservant_to_reference()を使用すると便利です。

1. POAを生成します（任意のアダプタ名）。
2. Servantを作成します。
3. servant_to_reference()を発行し、オブジェクトリファレンスを取得します。
4. ネーミングサービスへオブジェクトリファレンスを登録(bind)します。

servant_to_reference()の結果、POAオブジェクトにServantオブジェクトとオブジェクトIDの関連付けの登録（活性化）が行われ、さらに、オブジェクトIDを内包するオブジェクトリファレンスが返されます。Servantオブジェクトはインタフェースリポジトリを内包しているため、create_reference()を使用した場合のように明示的に指定する必要がありません。

また、オブジェクトIDは活性化時に自動生成されるため、意識する必要がありません。ただし、POAポリシにIMPLICIT_ACTIVATIONを指定する必要があります。

POAオブジェクト使用方法2：事前生成／環境変数OD_IMPLID使用

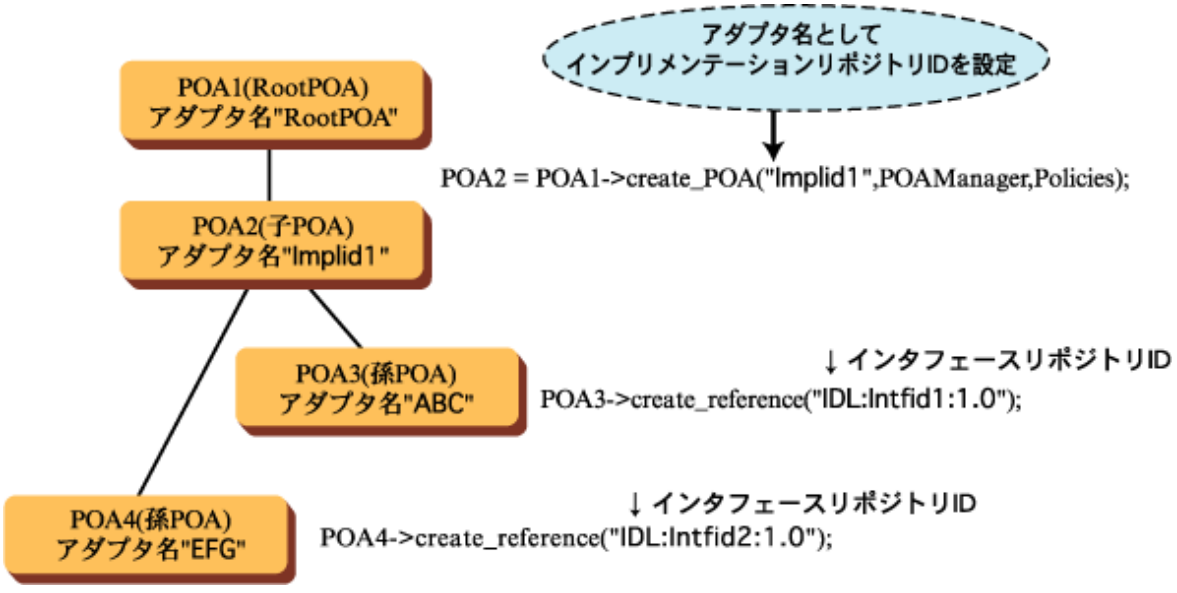
この方法は、OD_or_admコマンドを使用し、事前にオブジェクトリファレンスの生成とネーミングサービスへの登録を行っている場合に使用します。インタフェースリポジトリID“IDL:Intfid1:1.0”のインタフェースをもつオブジェクトを管理するために、「子POAオブジェクト」を生成します。この際に、インタフェースリポジトリIDと同じ文字列（Stringオブジェクト）を設定します。

Servantオブジェクトの活性化は、以下のように行います。

1. Servantオブジェクトを作成します。
2. Servantオブジェクトに対応するインタフェースリポジトリIDをアダプタ名としたPOAに対し、activate_object()を発行します。

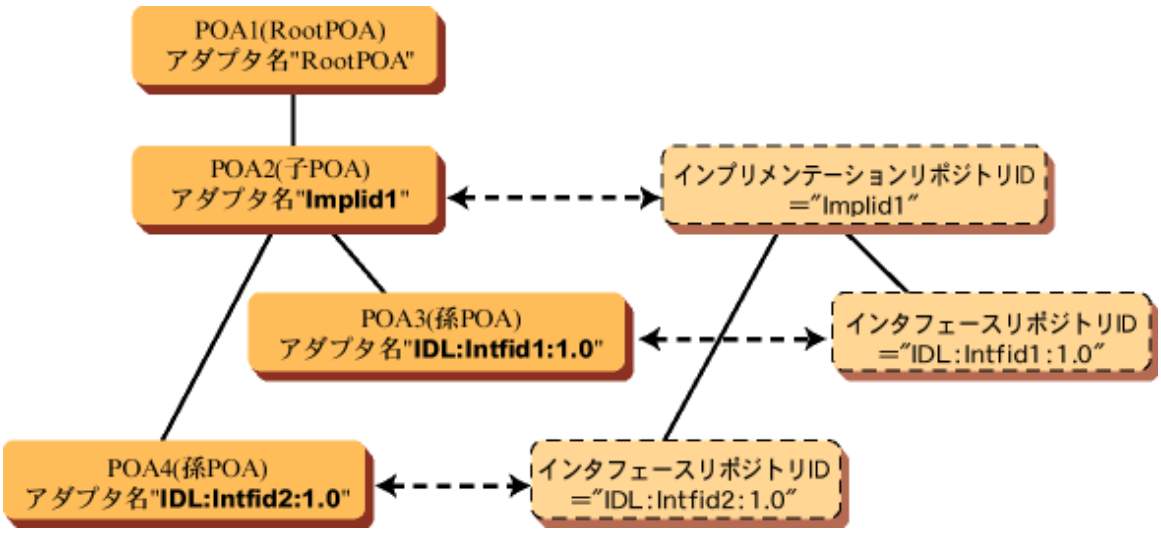
OD_or_admコマンドにより事前にオブジェクトリファレンスの生成とネーミングサービスへの登録を行っているため、アプリケーション上での生成／登録を行う必要はありません。

POAオブジェクト使用方法3：動的生成／環境変数OD_IMPLID未使用



インプリメンテーション情報との関連付けを行うために、アダプタ名にインプリメンテーションリポジトリIDを指定して、RootPOAオブジェクトに対する子POAオブジェクトを作成します。
次に、孫（またはその子孫）POAオブジェクトを作成し（アダプタ名は任意）、そのPOAオブジェクト上でcreate_reference()メソッド／servant_to_reference()メソッドを使用してオブジェクトリファレンスを作成します。
Servantオブジェクトの活性化後、オブジェクトのインスタンス管理はこの孫（またはその子孫）POAオブジェクト上で行われます。この方法の場合、RootPOAオブジェクトではインスタンス管理を行うことはできません。

POAオブジェクト使用方法4：事前生成／環境変数OD_IMPLID未使用



インプリメンテーション情報との関連付けを行うためには、上図の左側のように3階層でPOAオブジェクトを生成します。このとき、RootPOAオブジェクトの直下の子POAオブジェクト（POA2）は、インプリメンテーションリポジトリIDをアダプタ名に指定して生成します。次に、子POAオブジェクトのさらに子POAオブジェクト（孫POA：POA3, POA4）は、インタフェースリポジトリIDをアダプタ名に指定して生成します。この結果、インプリメンテーション情報、インタフェース情報は、上図のようにPOAと関連付けられます。リポジトリID“IDL:Intfid2:1.0”のインタフェースをもつオブジェクトのインスタンス管理は、POA4オブジェクトで行われます。

4.3.4 静的スケルトンインタフェース

静的起動インタフェースを使用する場合のサーバアプリケーションの処理の流れを以下に示します。

4.3.4.1 初期化

CORBAの初期化メソッドCORBA::ORB.init()を呼び出し、初期化処理を行います。本メソッドの結果として、ORBのオブジェクトリファレンスが通知されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
int main( int argc, char* argv[] ) {
    CORBA::ORB          orb;   // ORB用オブジェクトリファレンス
    CORBA::Environment  env;   // 例外情報

    try {
        // ORBの生成と初期化
        ob = CORBA::ORB.init( current_argc, argv, FJ_OM_ORBid, env );
            ...
    }
    catch( CORBA::Exception& e ) {
        ... // 例外処理
    }
}
```

例外処理try-catchについては、“4.5.2 クライアントアプリケーションの例外処理”を参照してください。

4.3.4.2 RootPOAのオブジェクトリファレンスの獲得

サーバアプリケーションでPOAオブジェクトを使用するために、イニシャルサービスのオブジェクトリファレンスの取出しメソッドresolve_initial_references()によりRootPOAのオブジェクトリファレンスを求めます。

```
try {
    // RootPOAのオブジェクトリファレンスの取得
    CORBA::Object_ptr  _tmpObj = orb->resolve_initial_references( "RootPOA", env );
    // RootPOAのPOAオブジェクト獲得
    PortableServer::POA_ptr  rootPOA  = PortableServer::POA::_narrow( _tmpObj );
}
catch( CORBA::Exception& e ) {
    ... // 例外処理
}
```

4.3.4.3 子孫POAの作成

RootPOAとは、異なるポリシでインスタンスを管理する場合や、インタフェースごとなど、複数のPOAにインスタンスの管理を分散させたい場合は、新しいPOAを生成します。

以下の例では、取得したRootPOAのメソッドを使用して、子POAを作成しています。
create_POA()の第1パラメタには、アダプタ名(生成するPOAの名前)を指定します。アダプタ名は、任意の文字列(Stringオブジェクト)を指定します。“4.3.3 サーバアプリケーションと環境設定との関連付け”で示すように、インプリメンテーションリポジトリID／インタフェースリポジトリIDを指定することも可能です。
create_POA()の第3パラメタのポリシには、アプリケーション形態にあったポリシリストを指定します。この例では、USE_DEFAULT_SERVANTを使用するPOAを作成しています。

```
CORBA::PolicyList   policies( 4 );   // ポリシリスト
CORBA::Environment  env;             // 例外情報

try {
    // ポリシリスト作成
    policies.length( 4 );
    policies[0] = rootPOA->create_servant_retention_policy(
           PortableServer::NON_RETAIN, env );
    policies[1] = rootPOA->create_request_processing_policy(
           PortableServer::USE_DEFAULT_SERVANT, env );
    policies[2] = rootPOA->create_id_assignment_policy(
           PortableServer::SYSTEM_ID, env );
    policies[3] = rootPOA->create_id_uniqueness_policy(
           PortableServer::MULTIPLE_ID, env );

    // インタフェースリポジトリID に対応するのPOAオブジェクト獲得
    PortableServer::POA_ptr  POA1 =
            rootPOA->create_POA( "childPOA", rootPOA->the_POAManager(), policies, env );
}
catch( CORBA::Exception& e ) {
    ... // 例外情報
}
```

孫以下のPOAを作成する場合も上記と手順は、同じです。子POAのインスタンス(上記の例の場合:POA1)上でcreate_POA()メソッドを発行することにより、孫POAを生成することが可能です。

注意

POAのアダプタ名(上記例では“childPOA”)に、日本語などのマルチバイト文字は使用できません。

4.3.4.4 インタフェースの実装

IDLコンパイラで生成されたスケルトン(以下参照)内に定義してあるPOA_xxxクラス(xxx:<モジュール名>_<インタフェース名>)を継承したxxx_implクラス(xxx:<モジュール名>_<インタフェース名>)内のオペレーションの実装を記述します。オペレーションのC++マッピング後のシグネチャは、yyy.hファイル(yyy:IDLファイル名)内の記述を参照することでわかります。

Windows32/64

```
yyy_skel.cpp
```

yyy:IDLファイル名

Solaris32/64 Linux32/64

```
yyy_skel_c++.C
```

yyy:IDLファイル名

注意

この実装方法は、継承方式(Inheritance-Based Implementation)に基づくものです。代理方式(Delegation-Based Implementation)を使用する場合は、"4.3.5 サーバアプリケーションの実装アプローチ"を参照してください。

【IDL定義】

```
module ODsample{
    interface intf{
        long add( in long a, in long b );
    };
};
```

【スケルトン】

```
class POA_ODsample_intf : public virtual PortableServer::ServantBase
{
     :
 };

class ODsample_intf_impl :  public virtual POA_ODsample_intf
{
    public :
        CORBA::Long  add( ... );
       :
};
```

【メソッドの実装】

```
CORBA::Long  Odsample_intf_impl::add(
    CORBA::Long a,
    CORBA::Long b,
    CORBA::Environment& env )
    throw( CORBA::Exception )
{
    return( a + b );
}
```

4.3.4.5 AOMへの登録(活性化)

Servantオブジェクトを生成し、POAのAOMに登録します。以下の例では、RootPOAのAOMにServantオブジェクトを登録しています。servant_to_reference()メソッドを使用することで、Servantオブジェクトを登録すると同時に対応するオブジェクトリファレンスを生成・獲得しています(POAがIMPLICIT_ACTIVATIONポリシを持つ場合)。

【Servantオブジェクトのインスタンス生成とAOMへの登録】

```
try {

    // Servantの生成
    ODsample_intf_impl* svt = new ODsample_intf_impl();

    // AOMにServantを登録 (活性化),オブジェクトリファレンス取得
    CORBA::Object_ptr c_obj =
        rootPOA->servant_to_reference( svt, env );

}
catch( CORBA::SystemException& e ) {
    ...  // 例外処理
}
```

4.3.4.6 ネーミングサービスへの登録

生成したオブジェクトリファレンスをネーミングサービスへ登録します。OD_or_admコマンドでオブジェクトリファレンスの事前生成／登録を行っている場合は、必要ありません。

【ネーミングサービスへの登録】

```
#include "CosNaming_cplus.h"
   :
CORBA::Environment  env;   // 例外情報
:
try {
    // ネーミングサービスのオブジェクトリファレンスの取得
    CORBA::Object_ptr  obj = orb->resolve_initial_references( "NameService", env );
    // NamingContext型に変換
    CosNaming::NamingContext_ptr cos_naming =
                    CosNaming::NamingContext::_narrow( obj );

    CosNaming::Name                 name;

    name.length( 1 );
    name[0]->id   = ( const CORBA::Char* )"ODsample::intf";
    name[0]->kind = ( const CORBA::Char* )"";

    try {
        cos_naming->unbind( name, env ); //同名の古いオブジェクトリファレンスがあれば削除
    }
    catch( CORBA::Exception& e )
    {
        :
    }
    cos_naming->bind( name, env ); //オブジェクトリファレンスの登録
}
catch( CORBA::Exception& e ) {
```

```
    ...   //例外処理
}
```

4.3.4.7 オブジェクトリファレンスの事前生成／登録

“4.3.4.6 ネーミングサービスへの登録”の例では、アプリケーション内でオブジェクトリファレンスの生成とネーミングサービスへの登録を行っていますが、OD_or_admコマンドを使用することで、事前にオブジェクトリファレンスの生成／登録を行うことが可能となります。この場合、アプリケーション内での生成／登録は不要です。POA構成は、「POA構成方法2」／「POA構成方法4」を使用することになります。

例

```
OD_or_adm -c IDL:ODsample/intf:1.0  -a Imple_sample1  -n ODsample::sample1
```

-c IDL:ODsample/intf:1.0

指定されたインタフェースリポジトリIDでオブジェクトリファレンスを登録します。

-a Imple_sample1

指定されたインプリメンテーションリポジトリIDでオブジェクトリファレンスを登録します。

-n ODsample::sample1

ネーミングサービスに登録するオブジェクト名を指定します。

4.3.4.8 POAManagerの活性化と終了待機

POAに関連付けられたPOAManagerのactivate()を発行することで、クライアントからの受信待ち状態となります。activate()発行直後に、ORBクラスのrun()メソッドを発行します。これにより、サーバアプリケーションは終了せず待機状態となります。

【POAManagerの活性化】

```
try {
    // POAマネージャのactivate
    PortableServer::POAManager_ptr  poa_manager;
    poa_manager = rootPOA->the_POAManager( env );
    poa_manager->activate();
    orb->run( env );
}
catch( CORBA::Exception& e ) {
    ...//例外処理
}
```

注意

動作上、activate()発行時に終了待機状態となるため、run()メソッドには制御は移りません。このため、run()メソッドは機能的には何も行いません。しかし、アプリケーションの可搬性の面からrun()メソッドを記述することが推奨されます。

注意

Windows32/64

以下の条件のとき、POAManagerのactivate()が復帰せずにサーバアプリケーションが終了します。

- サーバアプリケーションをプロセスモードで作成している場合 かつ
- サーバアプリケーションをCORBAワークユニット運用している場合 かつ
- CORBAワークユニットを停止した場合

POAManagerのactivate()の後に後処理を行う場合は、サーバアプリケーションをスレッドモードで作成するか、またはワークユニットの出口機能を利用してください。ワークユニットの出口機能に関しては、“OLTPサーバ運用ガイド”の“ワークユニットの出口機能”を参照してください。

4.3.4.9 アプリケーション作成時の注意点

POAアプリケーションを作成する場合、以下のように設定する必要があります。

Windows32/64

【Visual C++(R)の場合】

1. 「プロジェクトの設定」の「プリプロセッサの定義」に“_MULTI_LINK_”を追加します。
2. プロジェクトに“xx_stub.cpp”(xx:IDLファイル名)を追加してビルドします。

【Makefileでnmakeを使用する場合】

1. コンパイラオプションに“_MULTI_LINK_”を追加します。
2. “xx_stub.cpp”(xx:IDLファイル名)を追加してコンパイルし、ビルドします。

Solaris32/64 Linux32/64

1. コンパイラオプションに“_MULTI_LINK_”を追加します。
2. “xx_stub_c++.C”(xx:IDLファイル名)を追加してコンパイルし、ビルドします。

注意

Linux32/64

コンパイラにg++を使用する場合、プログラミング例では省略されていることがありますが、メソッドによっては引数にCORBA::Environmentのパラメタが必要となることがあります。

4.3.5 サーバアプリケーションの実装アプローチ

静的スケルトンインタフェースを使用したサーバアプリケーションの実装上のアプローチとして、以下の2つの方式が存在します。

- 継承方式
- 代理方式

ここでは、これら2つの実装方式とその特徴について説明します。

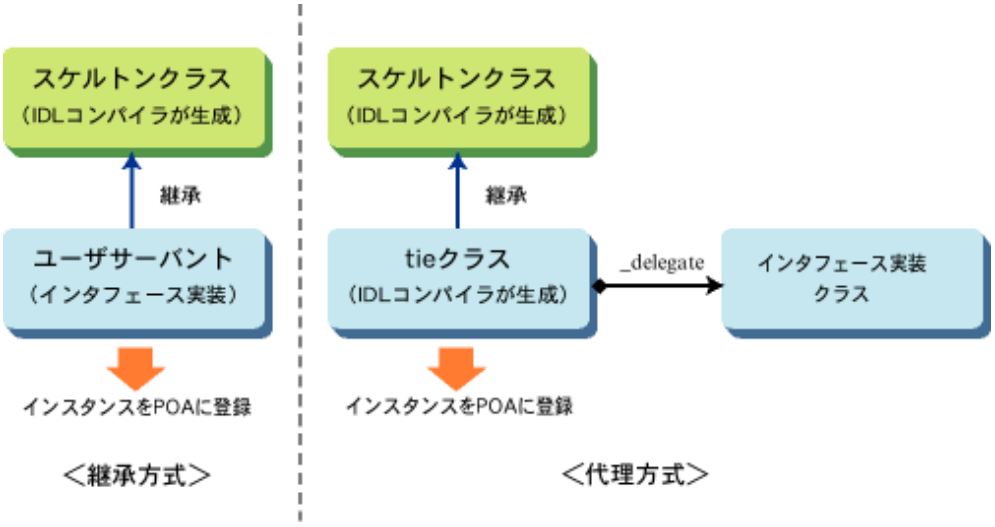
4.3.5.1 継承方式と代理方式

継承方式（Inheritance-Based Implementation）

新規にサーバアプリケーションを作成する場合に使用する一般的な実装方法です。サーバアプリケーションでスケルトンのメソッドを実装します。POAに対しては、スケルトンのクラス自身を登録します。

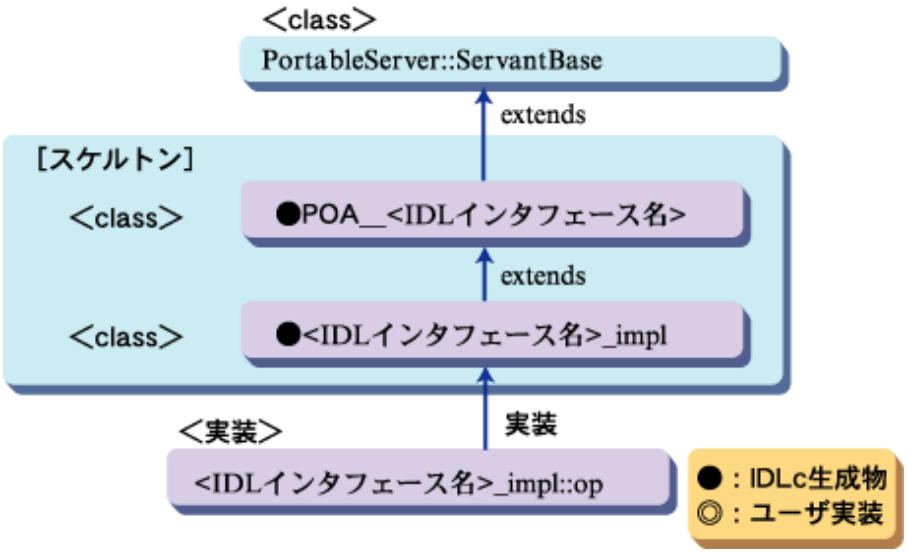
代理方式（Delegation-Based Implementation）

ユーザが開発した既存のクラスなどをServantとして使用する場合に有効な方法です。スケルトンを継承し、かつPOAの登録対象となるServantとして**tieクラス**が用意されます。IDLコンパイラによりtemplate<class T> POA_<interface名>_tieクラスが生成されます。
ユーザが作成したインタフェース実装クラスのインスタンスは、tieクラス内のメンバオブジェクトとして登録します。オペレーションの起動は、いったんtieクラスを経由して登録したクラスのインスタンスに対して発行されます。



4.3.5.2 継承方式のServant実装

継承方式によるServantは、以下のような継承関係となります。サーバアプリケーションのメイン部では、このServantをPOAに登録します。



上図のとおり、継承方式ではServantは以下のようにスケルトンクラスを継承する形式で実装します。

【IDL定義】

```
module ODsample{
    interface   intf{
        long    add(in long a, in long b);
    };
};
```

【スケルトン】

```
class POA_ODsample_intf : public virtual PortableServer::ServantBase
{
       :
};
class ODsample_intf_impl : public virtual POA_ODsample_intf{
      :
    CORBA::Long add(
        CORBA::Long a
        CORBA::Longint b
        CORBA::Environment& = CORBA::Environment() )
        throw( CORBA::Exception );
               :
};
```

【サーバアプリケーションでの実装例】

```
CORBA::Long  ODsample_intf_impl::add{
    return( a + b );
}
```

メイン処理部では、スケルトンクラスをPOAに登録します。以下の例では、Default Servantとして登録しています。

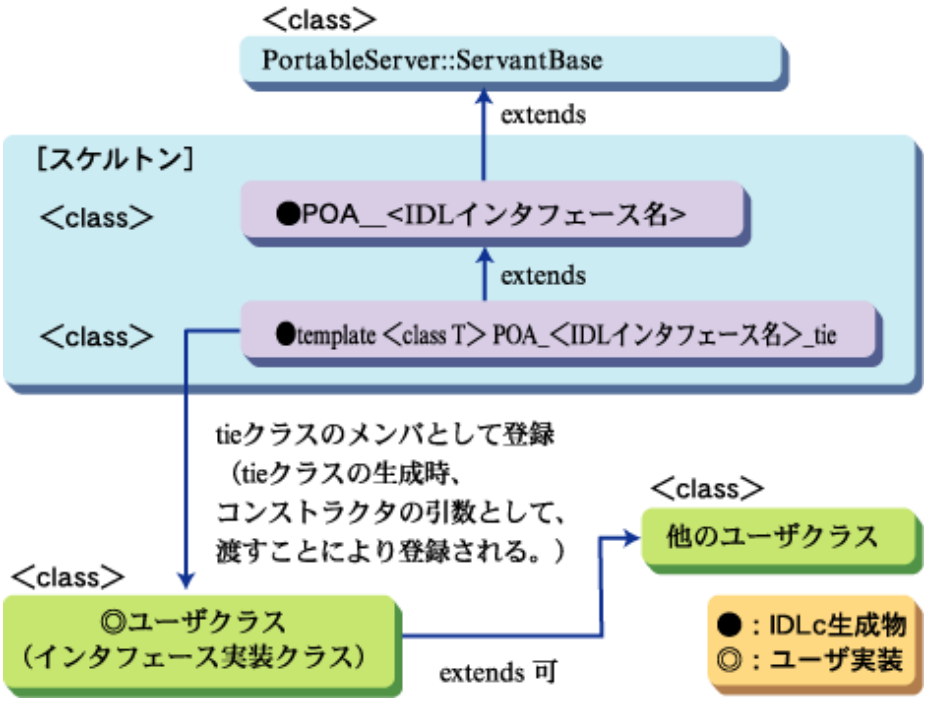
【メイン部の例】

```
// Servantの生成
ODsample_intf_impl* svt = new ODsample_intf_impl();
// Default Servantに設定
aPOA->set_servant( svt );
```

注) aPOA：POAクラスのインスタンス

4.3.5.3 代理方式のServant実装

代理方式によるユーザクラス(インタフェース実装クラス)は、以下のような継承関係で実装します。サーバアプリケーションのメイン部では、tieクラス(POA_<インタフェース名>_tie)のインスタンスをPOAに登録します。継承方式を使用した場合のような規定の継承関係から解放されるため、実装の自由度が高くなります。ただし、tieクラスのメンバとして登録するため、インタフェース実装クラスをtieクラスと関連付ける必要があります。代理方式のアプリケーションの例については、“4.3.7 サーバアプリケーションのプログラミング例”を参照してください。



上図のとおり、代理方式では、ユーザクラスは以下のようにPOA_<interface名>tieインタフェースを実装するように記述します。

【IDL定義】

```
module ODsample{
    interface   intf{
        long    add(in long a, in long b);
    };
};
```

【スケルトンクラス】

```
class POA_ODsample_intf : public virtual PortableServer::ServantBase
{
     :
}
template <class T>
class POA_ODsample_intf_tie : public virtual POA_ODsample_intf
{
        :
    CORBA::Long  add( CORBA::Long  a,
                      CORBA::Long  b,
                      CORBA::Environment  &_env = CORBA::Environment() )
                      throw( CORBA::Exception ) { return _ptr->add( _env ); }
        :
};
```

【ユーザクラスの例】

```
class UserClass
{
    public :
    CORBA::Long add(
              CORBA::Long a,
              CORBA::Long b,
```

```
                CORBA::Environment&env )
                throw( CORBA::Exception )
    {
        return( a + b );
    }
};
```

メイン処理部では、いったんこのユーザクラスのインスタンスをtieクラスのインスタンスに登録します(tieクラスのコンストラクタの引数としてユーザクラスインスタンスを渡します)。さらに、tieクラスのインスタンスをPOAにServantとして登録します。以下の例では、Default Servantとして登録しています。

【メイン部の例】

```
// ユーザクラスのインスタンス生成
UserClass*  uc = new UserClass();
// ユーザクラスとtieオブジェクトの関連付け
POA_ODsample_intf_tie< UserClass >*  svt = new POA_ODsample_intf_tie< UserClass >( uc );
// Default Servantに設定
aPOA->set_servant( svt );
```

注) aPOA:POAクラスのインスタンス

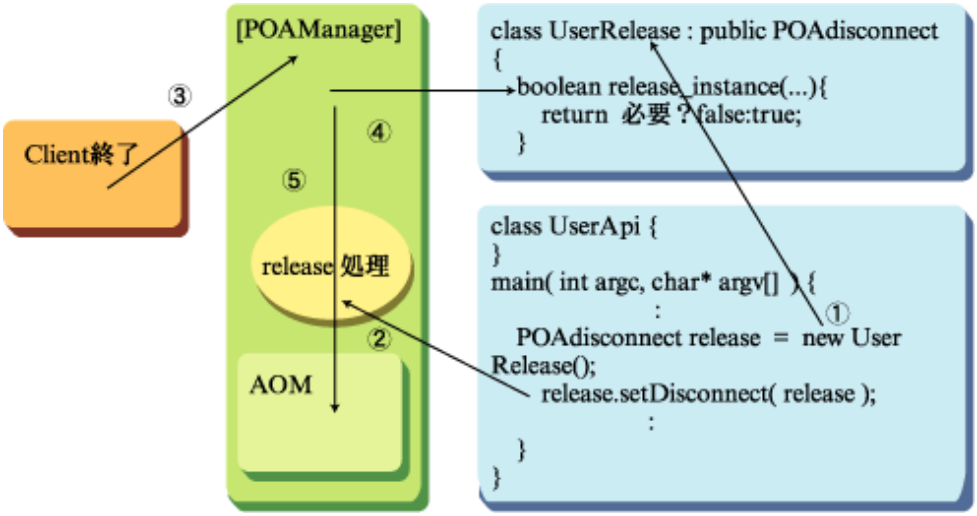
4.3.5.4 継承方式と代理方式の比較

継承方式、代理方式の特徴を以下の表に示します。
サーバアプリケーションの設計で、どちらの実装方式を使用するかはユーザの選択に任されます。

	継承方式	代理方式
主な用途	新規開発	既存資産の利用
実装の自由度	小さい。 ServantにはIDLc が生成する実装クラスを指定し、実装クラスは他のクラスを継承することができないため。	大きい。 ユーザが定義するインタフェース実装クラスはtieクラスとの関連付けに必要なインタフェース部分以外に関して、自由に作成することができます。
コーディングの複雑さ	比較的簡潔となります。	複雑になりやすい。
性能	オペレーションの起動に必要なスケルトン機能はServant自身に継承されているため、比較的よい。	オペレーションの起動はtieクラスを経由するため、オブジェクトの参照が増加します。これにより多少のオーバヘッドがありえます。

4.3.6 クライアントとのコネクション切断時のインスタンス解放

クライアントとのコネクションが切断された場合に、サーバアプリケーション側の不要なインスタンスを解放する方法について説明します。この機能は、インスタンス管理機能を利用し、かつPOAがActiveObjectMapを使用してインスタンス管理を行っている場合にだけ利用できます。



1. Servantインスタンス解放用クラスのインスタンスを生成します。
2. 生成したインスタンスを登録します。
3. クライアントとのコネクション切断が通知されます。
4. サーバントインスタンス解放の是非を問い合わせます。
5. サーバントインスタンスが不要である場合、AOMから削除します。

4.3.6.1 インスタンス解放処理を行うクラスの作成

クライアントとのコネクション切断時にインスタンス解放を行うクラスを作成します。このクラスは、PortableServer::POAdisconnectクラスを継承して作成します。

4.3.6.2 インスタンス解放処理の実装

クライアントとのコネクションが切断されたときに、作成したクラスのrelease_instance()メソッドが呼び出されます。親クラス（POAdisconnectクラス）のrelease_instance()メソッドは、常にfalse（解放しない）を返します。
release_instance()メソッドが呼び出されたとき、コネクションが切断されたクライアントが使用した、POA、ServantのインスタンスおよびオブジェクトIDがパラメタとして通知されます。オーバーライドされたrelease_instance()メソッドでは、これらの情報をもとにして、通知されたServantが必要なものであるかどうかを判断し、true（解放する）／false（解放しない）を返します。

trueの場合、暗黙的にPortableServer::ServantBase::servant_delete()メソッドが実行されます。対象のServantが実行中の場合、コネクションが切断されたクライアント以外のクライアントが対象のServantからのレスポンスを受け取った後に、そのServantのインスタンスが削除され、それ以降、そのServantへはアクセスできなくなります。

4.3.6.3 インスタンス解放処理を行うクラスインスタンスの登録

ユーザが作成したインスタンス解放クラスをPOAManagerに登録します。登録には、POAdisconnectクラスのstaticメソッドsetDisconnect()を呼び出します。

4.3.6.4 インスタンス解放処理を行うクラスインスタンスの変更・削除

すでに登録したインスタンス解放クラスを変更または取り消す場合は、POAdisconnectクラスのstaticメソッドresetDisconnect()を呼び出します。

4.3.7 サーバアプリケーションのプログラミング例

4.3.7.1 Default Servant使用例（デフォルトインスタンス方式）

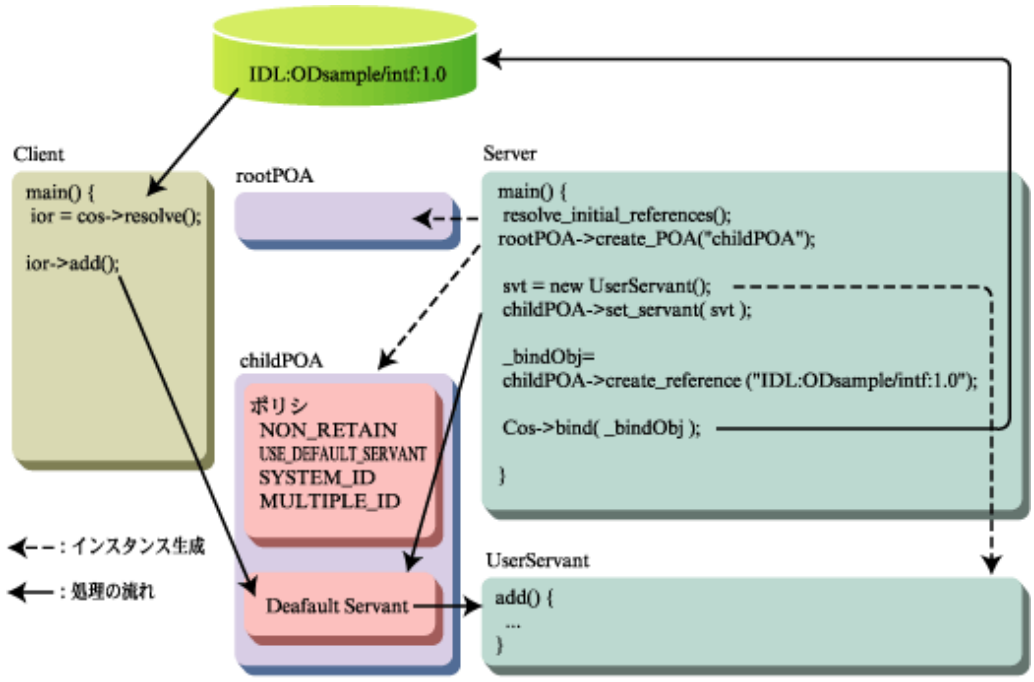
(1)IDL定義

```
module ODsample{
    interface  intf{
        long  add( in  long  a,  in  long  b );
    };
};
```

(2)ネーミングサービスの登録

動的登録

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
#include <stdlib.h>
#include <iostream>
#include "orb_cplus.h"
#include "CosNaming_cplus.h"
#include "simple.h"

using std::cout;
using std::endl;

static  CORBA::Object_var            obj;
static  CORBA::ORB_var               orb;
static  CORBA::Environment           env;
static  CosNaming::NamingContext_var  cos_naming;
```

```
static  CosNaming::Name              name;

int main( int argc, char *argv[] )
{

    try {
        // ORBの生成と初期化
        orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, env );

        // ネーミングサービスのオブジェクトリファレンスの取得
        obj = orb->resolve_initial_references(
                      CORBA_ORB_ObjectId_NameService, env );
        cos_naming = CosNaming::NamingContext::_narrow( obj );

        // ネーミングサービスのresolveメソッドを発行し、
        // サーバアプリケーションのオブジェクトリファレンスの獲得
        name.length( 1 );
        name[0]->id   = (const CORBA::Char *)"ODsample::POAsample1";
        name[0]->kind = (const CORBA::Char *)"";
        obj = cos_naming->resolve( name, env );

        // サーバアプリケーションのオブジェクトリファレンス獲得
        ODsample::intf* obj2 = ODsample::intf::_narrow( obj );

        // サーバアプリケーションのメソッド呼出し
        CORBA::Long ret = obj2->add( 1, 2, env );

        CORBA::release( obj2 );

    }catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);

    }catch( CORBA::Exception& e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }

    return 0;
}
```

(5)サーバアプリケーション

```
#include <iostream>
#include <stdlib.h>

#include "orb_cplus.h"
#include "poa.h"
#include "CosNaming_cplus.h"

#include "simple.h"

using std::cout;
using std::endl;

static CORBA::ORB_var                   orb;
static CORBA::Object_var                obj;
static CORBA::Environment               env;

static CosNaming::NamingContext_var     cos_naming;
static CosNaming::Name                  name;
```

```
static PortableServer::POA_var          rootPOA;
static PortableServer::POA_var          childPOA;
static CORBA::PolicyList                policies( 4 );
static PortableServer::POAManager_var   poa_manager;

// ユーザアプリケーション：メイン処理クラス
int main( int argc, char*argv[] )
{
    int current_argc = argc;

    try{
        // ORBの生成と初期化
        orb         = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

        // RootPOAのオブジェクトリファレンスの取得
        obj         = orb->resolve_initial_references( "RootPOA", env );
        // RootPOAのPOAオブジェクト獲得
        rootPOA     = PortableServer::POA::_narrow( obj );

        // インタフェース用のPOA作成
        // ポリシリスト作成
        policies.length( 4 );
        policies[0] = rootPOA->create_servant_retention_policy(
                            PortableServer::NON_RETAIN, env );
        policies[1] = rootPOA->create_request_processing_policy(
                            PortableServer::USE_DEFAULT_SERVANT, env );
        policies[2] = rootPOA->create_id_assignment_policy(
                            PortableServer::SYSTEM_ID, env );
        policies[3] = rootPOA->create_id_uniqueness_policy(
                            PortableServer::MULTIPLE_ID, env );

        // Default Servant用のPOAオブジェクトの作成
        childPOA   = rootPOA->create_POA( "childPOA",
                           PortableServer::POAManager::_nil(), policies, env );

        // Servantの生成
        ODsample_intf_impl* svt = new ODsample_intf_impl();
        // Default Servantに設定
        childPOA->set_servant( svt, env );

        CORBA::Object_var _bind_obj = childPOA->create_reference(
                                    "IDL:ODsample/intf:1.0", env );

        // ネーミングサービスのオブジェクトリファレンスの取得
        CORBA::Object_var _tmpObj = orb->resolve_initial_references(
                                    "NameService", env );
        cos_naming = CosNaming::NamingContext::_narrow( _tmpObj );

        // ネーミングサービスのresolveメソッドを発行し、サーバアプリケーションの
        // オブジェクトリファレンスの獲得
        name.length( 1 );
        name[0]->id   = ( const CORBA::Char* )"ODsample::POAsample1";
        name[0]->kind = ( const CORBA::Char* )"";

        try {
            cos_naming->unbind( name, env );
        } catch( CORBA::Exception& e ) {
            ;
        }
        cos_naming->bind( name, _bind_obj, env );

        // POAマネージャの獲得
        poa_manager = childPOA->the_POAManager( env );
```

```
        // POAマネージャのactivate
        poa_manager->activate( env );

        orb->run( env );

    } catch( CORBA::Exception& e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }

    return 0;
}

// Servant：メソッド実装
CORBA::Long ODsample_intf_impl::add(
    CORBA::Long a,
    CORBA::Long b,
    CORBA::Environment& env)
    throw( CORBA::Exception ){

    CORBA::Long    result;

    cout << "ODsample::intf::add()" << endl;
    cout << "  a      : " << a << endl;
    cout << "  b      : " << b << endl;
    result = a + b;
    cout << "  result: " << result << endl;
    return result;
};
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法の詳細については、“4.2.2 サーバアプリケーションの例外処理”を参照してください。

4.3.7.2 Active Object Map（AOM）使用例（Factory-1方式）

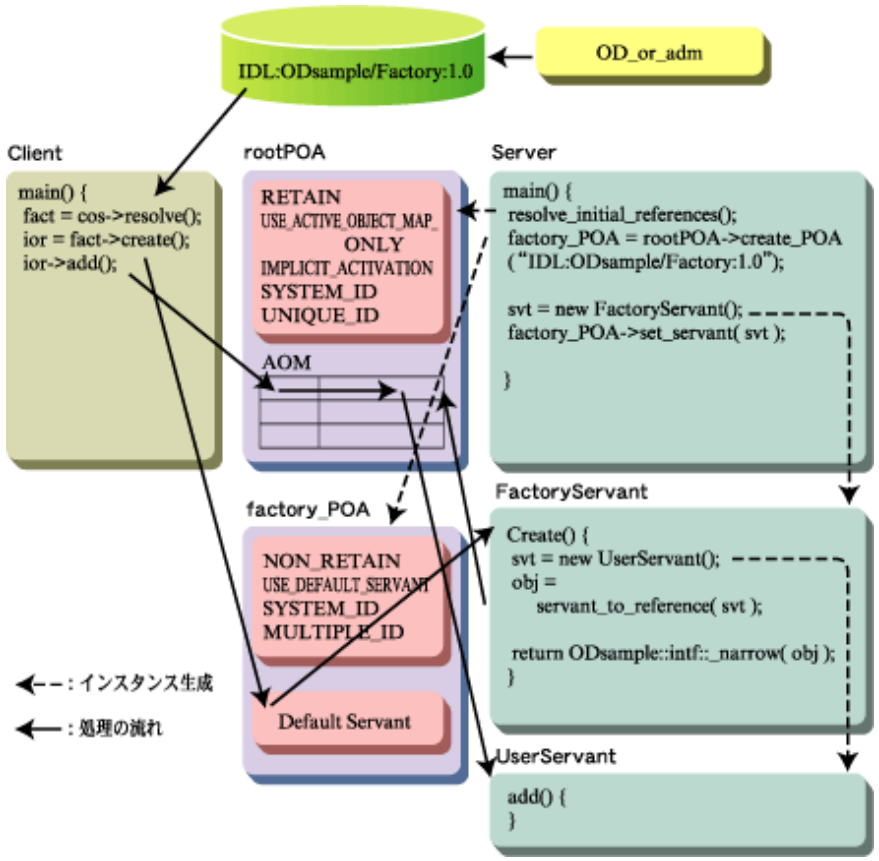
(1)IDL定義

```
module ODsample{
    interface intf{
        readonly  attribute  long  value1;
        void  add( in  long  a );
        void  sub( in  long  b );
    };
    interface Factory {
        intf   create();
        void   destroy( in  intf  obj );
    };
};
```

(2)ネーミングサービスの登録

```
OD_or_adm -c IDL:ODsample/Factory:1.0 -a Imple_POAsample3 -n ODsample::POAsample3
```

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
#include <stdlib.h>
#include <iostream>

#include "orb_cplus.h"
#include "CosNaming_cplus.h"
#include "simple.h"

using std::cout;
using std::endl;

static  CORBA::Object_var            obj;
static  CORBA::Environment           env;
static  CORBA::ORB_var               orb;
static  CosNaming::NamingContext_var  cos_naming;
static  CosNaming::Name              name;

int main( int argc, char *argv[] )
{
    try {
        // ORBの生成と初期化
        orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, env );

        // ネーミングサービスのオブジェクトリファレンスの取得
        obj = orb->resolve_initial_references(
                        CORBA_ORB_ObjectId_NameService, env );
        cos_naming = CosNaming::NamingContext::_narrow( obj );
```

```
        // ネーミングサービスのresolveメソッドを発行し、
        // サーバアプリケーションのオブジェクトリファレンスの獲得
        name.length( 1 );
        name[0]->id   = ( const CORBA::Char* )"ODsample::POAsample3";
        name[0]->kind = ( const CORBA::Char* )"";

        obj = cos_naming->resolve( name, env );

        // Factoryのオブジェクトリファレンス獲得
        ODsample::Factory* target = ODsample::Factory::_narrow( obj );

        // Factoryでインタフェース用オブジェクトリファレンス作成
        ODsample::intf* obj2 = target->create( env );

        CORBA::Long  a = 10;    // inパラメタ用変数
        CORBA::Long  b = 5;

        // サーバアプリケーションのメソッド呼出し
        CORBA::Long ret = obj2->value1( env );
        printf( "value = %d¥n", ret );

        obj2->add( a, env );
        printf( "add( %d )¥n", a );

        ret = obj2->value1( env );
        printf( "value = %d¥n", ret );

        obj2->sub( b, env );
        printf( "sub( %d )¥n", b );

        ret = obj2->value1( env );
        printf( "value = %d¥n", ret );

        // インスタンスの解放
        target->destroy( obj2, env );

        CORBA::release( target );
        CORBA::release( obj2 );

    } catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);

    } catch( CORBA::Exception&  e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }
    return 0;
}
```

(5)サーバアプリケーション

```
#include <stdlib.h>
#include <iostream>
#include "orb_cplus.h"
#include "simple.h"

#include "poa.h"
#include "CosNaming_cplus.h"

using std::cout;
```

```
using std::endl;

static CORBA::ORB_var                   orb;
static CORBA::Object_var                obj;
static CORBA::Environment               env;

static PortableServer::POA_var          rootPOA;
static PortableServer::POA_var          factoryPOA;
static CORBA::PolicyList                factory_policies( 4 );
static PortableServer::POAManager_var   poa_manager;

static long                             val = 10;

// ユーザアプリケーション：メイン処理クラス
int main( int argc, char*argv[] )
{
    try {
        // ORBの生成と初期化
        orb         = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

        // RootPOAのオブジェクトリファレンスの取得
        obj         = orb->resolve_initial_references( "RootPOA", env );
        // RootPOAのPOAオブジェクト獲得
        rootPOA     = PortableServer::POA::_narrow( obj );

        // Factoryインタフェース用のPOA作成
        // ポリシリスト作成
        factory_policies.length( 4 );
        factory_policies[0] = rootPOA->create_servant_retention_policy(
                              PortableServer::NON_RETAIN, env );
        factory_policies[1] = rootPOA->create_request_processing_policy(
                              PortableServer::USE_DEFAULT_SERVANT, env );
        factory_policies[2] = rootPOA->create_id_assignment_policy(
                              PortableServer::SYSTEM_ID, env );
        factory_policies[3] = rootPOA->create_id_uniqueness_policy(
                              PortableServer::MULTIPLE_ID, env );

        factoryPOA = rootPOA->create_POA( "IDL:ODsample/Factory:1.0",
             PortableServer::POAManager::_nil(), factory_policies ,env);

        // FactoryServantの生成
        ODsample_Factory_impl* svt = new ODsample_Factory_impl();
        // Factoryインタフェース用POAのDefault Servantに設定
        factoryPOA->set_servant( svt, env );

        // POAマネージャの獲得
        poa_manager = rootPOA->the_POAManager( env );

        // POAマネージャのactivate
        poa_manager->activate( env );

        orb->run( env );

    } catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);
    } catch( CORBA::Exception&  e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }
    return 0;
}
```

```
// Factory_impl::create：Factoryメソッドの実装
ODsample::intf_ptr ODsample_Factory_impl::create(
          CORBA::Environment&  env )
          throw( CORBA::Exception )

    try{
        // Servantの生成
        ODsample_intf_impl*  svt = new ODsample_intf_impl();

        // Servantからオブジェクトリファレンスの生成
        // IMPLICIT_ACTIVATIONポリシなので、AOMへ自動登録される
        CORBA::Object_var _tmpObj = rootPOA->servant_to_reference( svt, env );
        ODsample::intf_ptr ior = ODsample::intf::_narrow( _tmpObj );

        return ior;

    } catch( CORBA::Exception& e ) {
        cout << "exception-id: " << e.id() << endl;
        throw e;
    }
};

// Factory_impl::destroy：Factoryメソッドの実装
void ODsample_Factory_impl::destroy(
          ODsample::intf_ptr obj,
          CORBA::Environment& env )
          throw( CORBA::Exception )
{
    try{
        // オブジェクトリファレンスからオブジェクトIDを求める
        PortableServer::ObjectId_var oid = rootPOA->reference_to_id( obj, env );

        // Servantオブジェクトの取得
        PortableServer::Servant  svt = id_to_servant( *oid, env );

        // Servantをdeactiveにする
        rootPOA->deactivate_object( *oid, env );

        // Servantの削除
        svt->servant_delete();

    } catch( CORBA::Exception& e ) {
        cout << "exception-id: " << e.id() << endl;
        throw e;
    }
};

// ODsample::intf：メソッド実装クラス
void ODsample_intf_impl::add(
            CORBA::Long a,
            CORBA::Environment& env)
            throw( CORBA::Exception ){

        cout << "ODsample::intf::add()" << endl;
        cout << "  val : " << val << endl;
        cout << "  a     : " << a << endl;
        val = val + a;
        cout << "  result: " << val << endl;
};
void ODsample_intf_impl::sub(
            CORBA::Long b,
            CORBA::Environment& env)
            throw( CORBA::Exception ){
```

```
        cout << "ODsample::intf::sub()" << endl;
        cout << "  val : " << val << endl;
        cout << "  b     : " << b << endl;
        val = val - b;
        cout << "  result: " << val << endl;
};
CORBA::Long ODsample_intf_impl::value1(
            CORBA::Environment& env)
            throw( CORBA::Exception ){
        return val;
};
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法の詳細については、“4.2.2 サーバアプリケーションの例外処理”を参照してください。

4.3.7.3 Servant Activator使用例（Factory-2方式）

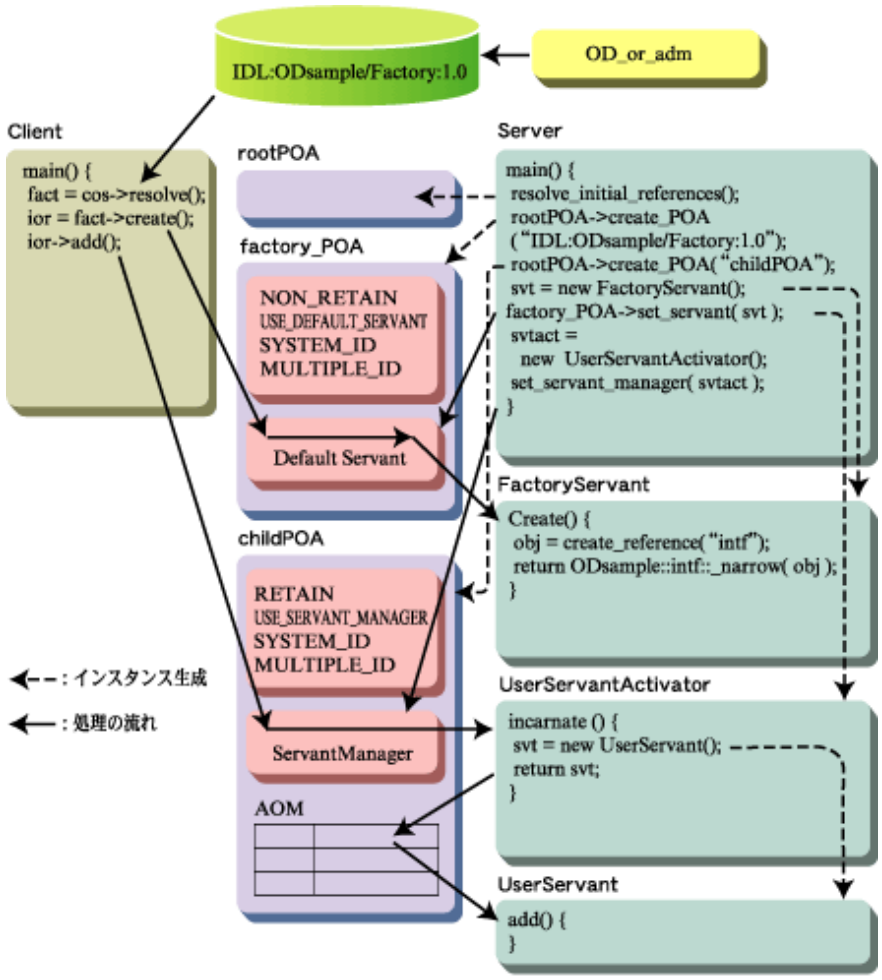
(1)IDL定義

```
module ODsample{
    interface intf{
        readonly  attribute  long  value1;
        void  add( in  long  a );
        void  sub( in  long  b );
    };
    interface Factory {
        intf   create();
        void   destroy( in  intf  obj );
    };
};
```

(2)ネーミングサービスの登録

```
OD_or_adm -c IDL:ODsample/Factory:1.0 -a Imple_POAsample3 -n ODsample::POAsample3
```

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
#include <stdlib.h>
#include <iostream>

#include "orb_cplus.h"
#include "CosNaming_cplus.h"
#include "simple.h"

using std::cout;
using std::endl;

static  CORBA::Object_var             obj;
static  CORBA::Environment            env;
static  CORBA::ORB_var                orb;
static  CosNaming::NamingContext_var  cos_naming;
static  CosNaming::Name               name;

int main( int argc, char *argv[] )
{
    try {
        // ORBの生成と初期化
        orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, env );

        // ネーミングサービスのオブジェクトリファレンスの取得
        obj = orb->resolve_initial_references(
                        CORBA_ORB_ObjectId_NameService, env );
```

```
        cos_naming = CosNaming::NamingContext::_narrow( obj );

        // ネーミングサービスのresolveメソッドを発行し、
        // サーバアプリケーションのオブジェクトリファレンスの獲得
        name.length( 1 );
        name[0]->id   = ( const CORBA::Char* )"ODsample::POAsample3";
        name[0]->kind = ( const CORBA::Char* )"";
        obj = cos_naming->resolve( name, env );

        // Factoryのオブジェクトリファレンス獲得
        ODsample::Factory* target = ODsample::Factory::_narrow( obj );

        // Factoryでインタフェース用オブジェクトリファレンスを作成する
        ODsample::intf* obj2 = target->create( env );

        CORBA::Long  a = 10;    // inパラメタ用変数
        CORBA::Long  b = 5;

        // サーバアプリケーションのメソッド呼出し
        CORBA::Long ret = obj2->value1( env );
        printf( "value = %d¥n", ret );

        obj2->add( a, env );
        printf( "add( %d )¥n", a );

        ret = obj2->value1( env );
        printf( "value = %d¥n", ret );

        obj2->sub( b, env );
        printf( "sub( %d )¥n", b );

        ret = obj2->value1( env );
        printf( "value = %d¥n", ret );

        // インスタンスの解放
        target->destroy( obj2, env );

        CORBA::release( target );
        CORBA::release( obj2 );

    } catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);

    } catch( CORBA::Exception&  e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }
    return 0;
}
```

(5)サーバアプリケーション

```
#include <iostream>
#include <stdlib.h>

#include "poa.h"
#include "CosNaming_cplus.h"
#include "orb_cplus.h"

#include "simple.h"
```

```
using std::cout;
using std::endl;

static CORBA::ORB_var                   orb;
static CORBA::Object_var                obj;
static CORBA::Environment               env;

static PortableServer::POA_var          rootPOA;
static PortableServer::POA_var          factoryPOA;
static PortableServer::POA_var          childPOA;
static CORBA::PolicyList                factory_policies( 4 );
static CORBA::PolicyList                intf_policies( 4 );
static PortableServer::POAManager_var   poa_manager;

static long                             val = 10;

// ユーザアプリケーション：メイン処理クラス
int main( int argc, char*argv[] )
{
    int current_argc = argc;

    try{
        // ORBの生成と初期化
        orb         = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

        // RootPOAのオブジェクトリファレンスの取得
        obj         = orb->resolve_initial_references( "RootPOA", env );
        // RootPOAのPOAオブジェクト獲得
        rootPOA     = PortableServer::POA::_narrow( obj );

        // Factoryインタフェース用のPOA作成
        // ポリシリスト作成
        factory_policies.length( 4 );
        factory_policies[0] = rootPOA->create_servant_retention_policy(
                              PortableServer::NON_RETAIN, env );
        factory_policies[1] = rootPOA->create_request_processing_policy(
                              PortableServer::USE_DEFAULT_SERVANT, env );
        factory_policies[2] = rootPOA->create_id_assignment_policy(
                              PortableServer::SYSTEM_ID, env );
        factory_policies[3] = rootPOA->create_id_uniqueness_policy(
                              PortableServer::MULTIPLE_ID, env );

        factoryPOA = rootPOA->create_POA( "IDL:ODsample/Factory:1.0",
               PortableServer::POAManager::_nil(), factory_policies ,env);

        // インタフェース用のPOA作成
        // ポリシリスト作成
        intf_policies.length( 4 );
        intf_policies[0] = rootPOA->create_servant_retention_policy(
                              PortableServer::RETAIN, env );
        intf_policies[1] = rootPOA->create_request_processing_policy(
                              PortableServer::USE_SERVANT_MANAGER, env );
        intf_policies[2] = rootPOA->create_id_assignment_policy(
                              PortableServer::SYSTEM_ID, env );
        intf_policies[3] = rootPOA->create_id_uniqueness_policy(
                              PortableServer::MULTIPLE_ID, env );

        childPOA = rootPOA->create_POA( "childPOA",
               PortableServer::POAManager::_nil(), intf_policies ,env );

        // FactoryServantの生成
        ODsample_Factory_impl* svt = new ODsample_Factory_impl();
        // Factoryインタフェース用POAのDefault Servantに設定
```

```
        factoryPOA->set_servant( svt, env );

        // ServantActivatorの生成
        PortableServer::ServantActivator_var svtact = new UserServantActivator();
        // インタフェース用POAのServantManagerとして登録
        childPOA->set_servant_manager( svtact, env );

        // POAマネージャの獲得
        poa_manager = rootPOA->the_POAManager( env );

        // POAマネージャのactivate
        poa_manager->activate( env );

        orb->run( env );

    } catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);
    } catch( CORBA::Exception&  e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }
    return 0;
}

// ODsample_Factory_impl::create：Factoryメソッド実装クラス
ODsample::intf_ptr ODsample_Factory_impl::create(
          CORBA::Environment& env )
          throw( CORBA::Exception )
{
    try{
        // オブジェクトリファレンスの生成
        CORBA::Object_var _tmpObj
              = childPOA->create_reference( "IDL:ODsample/intf:1.0", env );
        ODsample::intf_ptr ior = ODsample::intf::_narrow( _tmpObj );
        return ior;

    } catch( CORBA::Exception&  e ) {
        cout << "exception-id: " << e.id() << endl;
        throw( e );
    }
};

// ODsample_Factory_impl::destroy：Factoryメソッド実装クラス
void ODsample_Factory_impl::destroy(
          ODsample::intf_ptr obj,
          CORBA::Environment& env )
          throw( CORBA::Exception )
{
    try{
        // オブジェクトリファレンスからオブジェクトIDを求める
        PortableServer::ObjectId_var oid = childPOA->reference_to_id( obj, env );

        // Servantをdeactiveにする
        childPOA->deactivate_object( *oid, env );

    } catch( CORBA::Exception&  e ) {
        cout << "Exception: " << opname << endl;
        cout << "exception-id: " << e.id() << endl;
        throw( e );
    }
};
```

```
// ServantActivator：Servant生成クラス（ServantActivatorを継承）
class UserServantActivator: public PortableServer::ServantActivator
{
public:
   UserServantActivator() {};
   ~UserServantActivator() {};

    PortableServer::Servant incarnate(
                const PortableServer::ObjectId& oid,
                PortableServer::POA_ptr adapter,
                CORBA::Environment& env )
                throw( CORBA::Exception )
    {
        // Servantの生成
        PortableServer::Servant svt = new ODsample_intf_impl();
        return( svt );
    };

    void etherealize( const PortableServer::ObjectId& oid,
                      PortableServer::POA_ptr adapter,
                      PortableServer::Servant serv,
                      CORBA::Boolean cleanup_in_progress,
                      CORBA::Boolean remaining_activations,
                      CORBA::Environment& env )
                      throw( CORBA::Exception )
    {
        // Servantオブジェクトの削除。
        serv->servant_delete();
    }
};

// ODsample_intf_impl：メソッド実装クラス
void ODsample_intf_impl::add(
          CORBA::Long a,
          CORBA::Environment& env)
          throw( CORBA::Exception ){

    cout << "ODsample::intf::add()" << endl;
    cout << "  val : " << val << endl;
    cout << "  a     : " << a << endl;
    val = val + a;
    cout << "  result: " << val << endl;
};

void ODsample_intf_impl::sub(
          CORBA::Long b,
          CORBA::Environment& env)
          throw( CORBA::Exception ){

    cout << "ODsample::intf::sub()" << endl;
    cout << "  val : " << val << endl;
    cout << "  b     : " << b << endl;
    val = val - b;
    cout << "  result: " << val << endl;
};

CORBA::Long ODsample_intf_impl::value1(
            CORBA::Environment& env)
            throw( CORBA::Exception ){
    return val;
};
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法の詳細については、“4.2.2 サーバアプリケーションの例外処理”を参照してください。

4.3.7.4 Servant Locator使用例（ユーザインスタンス管理方式）

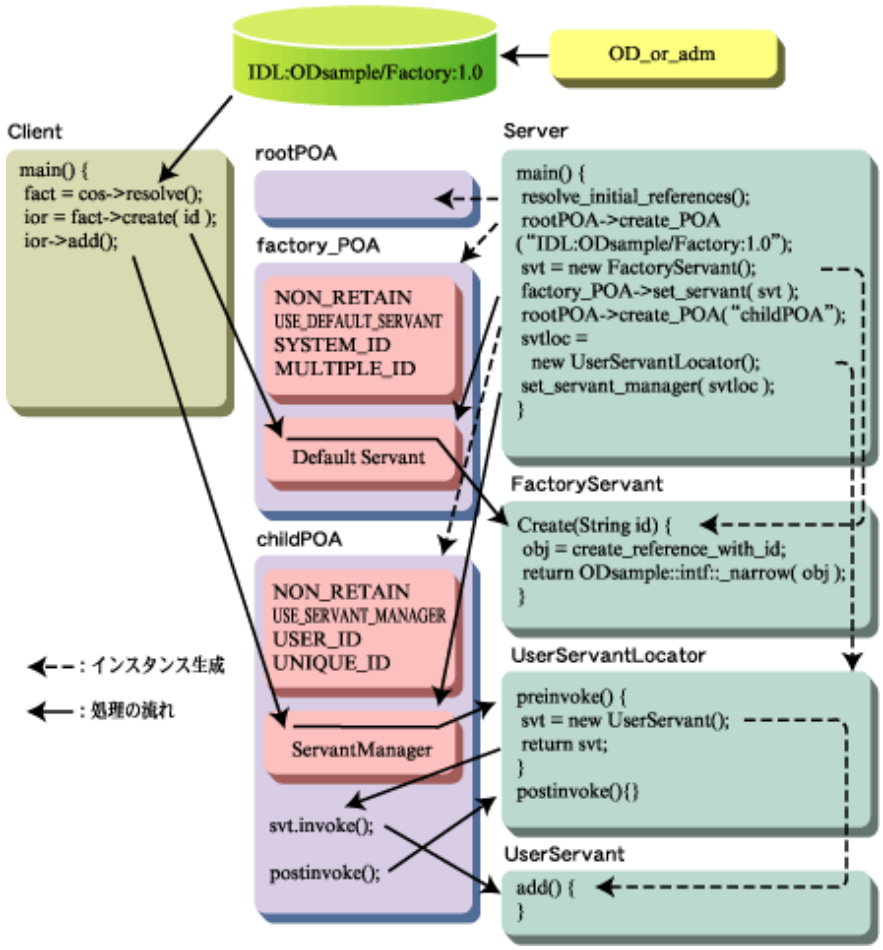
(1)IDL定義

```
module ODsample{
    interface    intf{
        readonly  attribute  long  value1;
        void add( in  long  a );
        void sub( in  long  b );
    };
    interface Factory{
        intf   create( in  string  userid );
        void   destroy( in  intf  obj );
    };
};
```

(2)ネーミングサービスの登録

```
OD_or_adm -c IDL:ODsample/Factory:1.0 -a Imple_POAsample4 -n ODsample::POAsample4
```

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
#include <iostream>
#include <stdlib.h>

#include "orb_cplus.h"
#include "CosNaming_cplus.h"
#include "simple.h"

using std::cout;
using std::endl;

static  CORBA::Object_var            obj;
static  CORBA::Environment           env;
static  CORBA::ORB_var               orb;
static  CosNaming::NamingContext_var cos_naming;
static  CosNaming::Name              name;

int main( int argc, char* argv[] )
{
    try {
        // ORBの生成と初期化
        orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, env );

        // ネーミングサービスのオブジェクトリファレンスの取得
        obj = orb->resolve_initial_references(
                    CORBA_ORB_ObjectId_NameService, env );
        cos_naming = CosNaming::NamingContext::_narrow( obj );
```

```
        // ネーミングサービスのresolveメソッドを発行し、
        // サーバアプリケーションのオブジェクトリファレンスの獲得
        name.length( 1 );
        name[0]->id   = ( const CORBA::Char* )"ODsample::POAsample4";
        name[0]->kind = ( const CORBA::Char* )"";

        obj = cos_naming->resolve( name, env );

        // サーバアプリケーションのインタフェースクラスの獲得
        ODsample::Factory* target = ODsample::Factory::_narrow( obj );

        char userid[1024];
        printf( "USERID -> " );
        userid[0] = 0;
        scanf( "%s", userid );
        if( strlen( userid ) < 1 ){
            return 255;
        }

        // Factoryでインタフェースアプリの IORを作成する
        ODsample::intf* obj2 = target->create( userid, env );

        CORBA::Long  a = 10;
        CORBA::Long  b = 5;

        // サーバアプリケーションのメソッド呼出し
        CORBA::Long ret = obj2->value1( env );
        printf( "value = %d¥n", ret );

        obj2->add( a, env );
        printf( "add( %d )¥n", a );

        ret = obj2->value1( env );
        printf( "value = %d¥n", ret );

        obj2->sub( b, env );
        printf( "sub( %d )¥n", b );

        ret = obj2->value1( env );
        printf( "value = %d¥n", ret );

        target->destroy( obj2, env );

        CORBA::release( target );
        CORBA::release( obj2 );

    } catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);
    } catch( CORBA::Exception&  e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }
    return 0;
}
```

(5)サーバアプリケーション

```
#include <iostream>
#include <stdlib.h>
```

```
#include "orb_cplus.h"
#include "poa.h"
#include "CosNaming_cplus.h"

#include "simple.h"

using std::cout;
using std::endl;

static CORBA::ORB_var                   orb;
static CORBA::Object_var                obj;
static CORBA::Environment               env;

static PortableServer::POA_var          rootPOA;
static PortableServer::POA_var          factory_poa;
static PortableServer::POA_var          childPOA;
static CORBA::PolicyList                policies( 4 );
static CORBA::PolicyList                inf_policies( 4 );
static PortableServer::POAManager_var   poa_manager;

static CORBA::Long                      val = 10;

// ユーザアプリケーション：メイン処理クラス
int main( int argc, char*argv[] )
{
    int current_argc = argc;
    try{
        // ORBの生成と初期化
        orb         = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

        // RootPOAのオブジェクトリファレンスの取得
        obj         = orb->resolve_initial_references( "RootPOA", env );
        // RootPOAのPOAオブジェクト獲得
        rootPOA     = PortableServer::POA::_narrow( obj );

        // Factoryインタフェース用のPOA作成
        // ポリシリスト作成
        policies.length( 4 );
        policies[0] = rootPOA->create_servant_retention_policy(
                            PortableServer::NON_RETAIN, env );
        policies[1] = rootPOA->create_request_processing_policy(
                            PortableServer::USE_DEFAULT_SERVANT, env );
        policies[2] = rootPOA->create_id_assignment_policy(
                            PortableServer::SYSTEM_ID, env );
        policies[3] = rootPOA->create_id_uniqueness_policy(
                            PortableServer::MULTIPLE_ID, env );

        factory_poa = rootPOA->create_POA( "IDL:ODsample/Factory:1.0",
                        PortableServer::POAManager::_nil(), policies ,env);

        // インタフェース用のPOA作成
        // ポリシリスト作成
        inf_policies.length( 4 );
        inf_policies[0] = rootPOA->create_servant_retention_policy(
                            PortableServer::NON_RETAIN, env );
        inf_policies[1] = rootPOA->create_request_processing_policy(
                            PortableServer::USE_SERVANT_MANAGER, env );
        inf_policies[2] = rootPOA->create_id_assignment_policy(
                            PortableServer::USER_ID, env );
        inf_policies[3] = rootPOA->create_id_uniqueness_policy(
                            PortableServer::UNIQUE_ID, env );

        childPOA = rootPOA->create_POA( "childPOA",
```

```
                        PortableServer::POAManager::_nil(), inf_policies ,env);

        // FactoryServantの生成
        ODsample_Factory_impl* svt = new ODsample_Factory_impl();
        // Factoryインタフェース用POAのDefault Servantに設定
        factory_poa->set_servant( svt, env );

        // ServantLocatorの生成
        PortableServer::ServantLocator* svtloc = new UserServantLocator();
        // インタフェース用POAにServantManagerとして登録
        childPOA->set_servant_manager( svtloc, env );

        // POAマネージャの獲得
        poa_manager = rootPOA->the_POAManager( env );

        // POAマネージャのactivate
        poa_manager->activate( env );

        orb->run( env );
    } catch( CORBA::Exception& e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }
    return 0;
}

// 管理テーブル
class ServantList
{
public:
    ServantList() {
        _next = NULL;
        _svr = NULL;
        _oid = NULL;
    };
    ~ServantList() {
        if( _svr != NULL ) {
            _svr->servant_delete();
        }
        if( _oid != NULL ) {
            delete _oid;
        }
    };
    PortableServer::ObjectId* getObjectId() {
        return _oid;
    };
    ServantList* getNext() {
        return _next;
    };
    // 管理テーブルから検索
    PortableServer::ServantBase* getServant( const PortableServer::ObjectId* oid ) {
        if( _svr == NULL ) {
            if( _next == NULL ) {
                return NULL;
            }
            return _next->getServant( oid );
        }
        if( equal( oid ) ) {
            return _svr;
        }
        if( _next ) {
            return _next->getServant( oid );
        }
```

```
        return NULL;
    };
    long equal( const PortableServer::ObjectId* oid ) {
        if( _oid == NULL ) {
            return 0;
        }
        if( oid->length() == oid->length() ) {
            long len = oid->length();
            long i;
            for( i = 0 ; i < len ; i++ ) {
                if( oid->operator[](i) != _oid->operator[](i) ) {
                    return 0;
                }
            }
            return 1;
        }
        return 0;
    };
    // 管理テーブルから削除
    void remove( PortableServer::ObjectId* oid ) {
        if( _next != NULL ) {
            if( _next->equal( oid ) ) {
                ServantList* del = _next;
                _next = _next->getNext();
                 delete del;
            }
        }
    };
    // 管理テーブルへ追加
    void put( const PortableServer::ObjectId* oid, PortableServer::ServantBase* svr ) {
        if( _next != NULL ) {
            _next->put( oid, svr );
        } else {
            _next = new ServantList();
            _next->_svr = svr;
            _next->_oid = new PortableServer::ObjectId( *oid );
        }
    };
private :
    PortableServer::ObjectId*      _oid;
    PortableServer::ServantBase*   _svr;
    ServantList*                   _next;
};

// 管理テーブルのインスタンス
ServantList servant_list;

// ODsample_Factory_impl::create : メソッドの実装
ODsample::intf_ptr ODsample_Factory_impl::create(
          const CORBA::Char* str,
          CORBA::Environment& env)
          throw( CORBA::Exception ){
    try{
        long len = strlen( str );
        CORBA::Octet* buf = PortableServer::ObjectId::allocbuf( len );
        memcpy( buf, str, len );
        PortableServer::ObjectId_var userid = new PortableServer::ObjectId(
                            len, len, buf, CORBA_TRUE );

        // オブジェクトリファレンスの生成
        CORBA::Object_var _tmpObj = childPOA->create_reference_with_id( *userid,
                                    "IDL:ODsample/intf:1.0",env );
        ODsample::intf_ptr ior = ODsample::intf::_narrow( _tmpObj );
```

```
        return ior;
    } catch(CORBA::Exception& e ) {
        cout << "exception-id: " << e.id() << endl;
        throw e;
    }
};

// ODsample_Factory_impl::destroy：メソッドの実装
void ODsample_Factory_impl::destroy(
          ODsample::intf_ptr obj,
          CORBA::Environment& env)
          throw( CORBA::Exception ){
    try{
        // オブジェクトリファレンスからオブジェクトIDを求める
        PortableServer::ObjectId_var  oid = childPOA->reference_to_id( obj, env );

        // Servantを管理テーブルから削除
        servant_list.remove( oid );
    } catch( CORBA::Exception&  e ) {
        cout << "exception-id: " << e.id() << endl;
        throw e;
    }
};

// ServantLocator：Servant生成クラス（ServantLocatorを継承）
class UserServantLocator: public PortableServer::ServantLocator
{
public:
    UserServantLocator() {};
    ~UserServantLocator() {};

    PortableServer::Servant preinvoke(
              const PortableServer::ObjectId&          oid,
              PortableServer::POA_ptr                  adapter,
              const CORBA::Identifier                  operation,
              PortableServer::ServantLocator::Cookie&  cookie,
              CORBA::Environment& = CORBA::Environment() )
              throw( CORBA::Exception )
    {
        // Servantを検索する。みつからなければnewで生成する。
        PortableServer::Servant svt = servant_list.getServant( &oid );
        if( svt == NULL ) {
            svt = new ODsample_intf_impl();
            servant_list.put( &oid, svt );
        }
        // Cookieの作成
        cookie = ( CORBA::ULong )svt;
        return svt;
    };
    void  postinvoke(
          const PortableServer::ObjectId&         oid,
          PortableServer::POA_ptr                 adapter,
          const CORBA::Identifier                 operation,
          PortableServer::ServantLocator::Cookie  cookie,
          PortableServer::Servant                 the_servant,
          CORBA::Environment& = CORBA::Environment() )
          throw( CORBA::Exception )
    {
    };
};

// ODsample_intf_implクラスのメソッド実装
void ODsample_intf_impl::add(
```

```
          CORBA::Long a,
          CORBA::Environment& env )
          throw( CORBA::Exception )
{
    cout << "ODsample::intf::add()" << endl;
    cout << "  value : " << val << endl;
    cout << "  a     : " << a << endl;
    val = val + a;
    cout << "  result: " << val << endl;
};
void ODsample_intf_impl::sub(
          CORBA::Long b,
          CORBA::Environment& env )
          throw( CORBA::Exception )
{
    cout << "ODsample::intf::sub()" << endl;
    cout << "  value : " << val << endl;
    cout << "  b     : " << b << endl;
    val = val - b;
    cout << "  result: " << val << endl;
};
CORBA::Long ODsample_intf_impl::value1(
          CORBA::Environment& env )
          throw( CORBA::Exception )
{
    return val;
};
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法の詳細については、“4.2.2 サーバアプリケーションの例外処理”を参照してください。

4.3.7.5 AdapterActivator使用例（find_POA時）

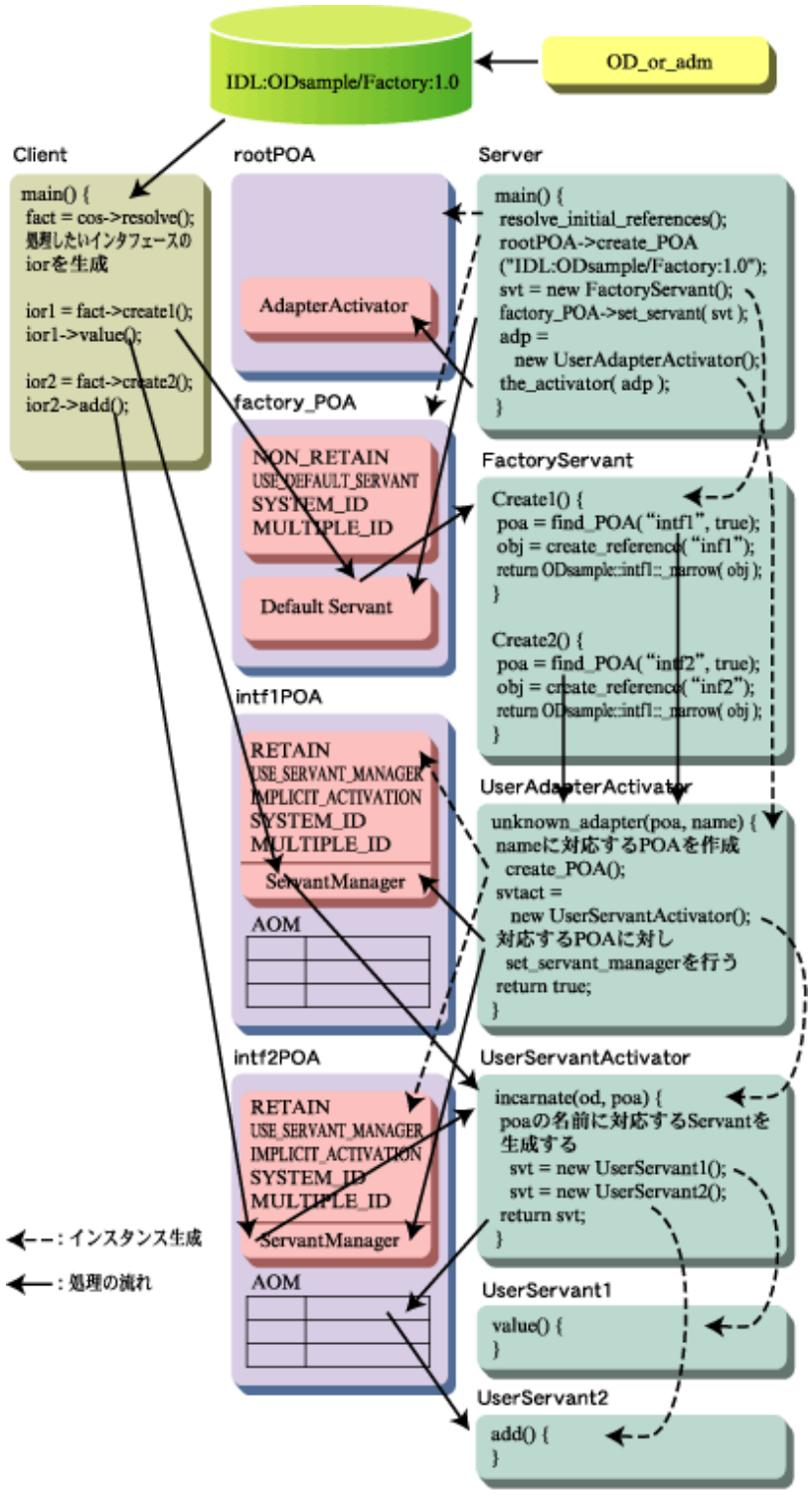
(1)IDL定義

```
module ODsample{
    interface intf1{
        attribute string value1;
    };
    interface intf2{
        readonly attribute long value1;
        void add( in long a );
        void sub( in long b );
    };
    interface Factory {
        intf1  create1 ();
        intf2  create2 ();
        void   destroy1( in intf1 obj );
        void   destroy2( in intf2 obj );
    };
};
```

(2)ネーミングサービスの登録

```
OD_or_adm -c IDL:ODsample/Factory:1.0 -a Imple_POAsample9 -n ODsample::POAsample9
```

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
#include <stdlib.h>
#include <iostream>
#include "orb_cplus.h"
#include "CosNaming_cplus.h"
```

```
#include "simple.h"

using std::cout;
using std::endl;

static  CORBA::Object_var             obj;
static  CORBA::Environment            env;
static  CORBA::ORB_var                orb;
static  CosNaming::NamingContext_var  cos_naming;
static  CosNaming::Name               name;

int main( int argc, char* argv[] )
{
    try {
        // ORBの生成と初期化
        orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, env );

        // ネーミングサービスのオブジェクトリファレンスの取得
        obj = orb->resolve_initial_references( CORBA_ORB_ObjectId_NameService, env );
        cos_naming = CosNaming::NamingContext::_narrow( obj );

        // ネーミングサービスのresolveメソッドを発行し、
        // サーバアプリケーションのオブジェクトリファレンスの獲得
        name.length( 1 );
        name[0]->id   = ( const CORBA::Char* )"ODsample::POAsample9";
        name[0]->kind = ( const CORBA::Char* )"";

        obj = cos_naming->resolve( name, env );

        // Factoryのオブジェクトリファレンス獲得
        ODsample::Factory* target = ODsample::Factory::_narrow( obj );

        // Factoryでインタフェース1用オブジェクトリファレンスを作成する
        ODsample::intf1* obj2 = target->create1( env );

        printf( "String = " );
        char buffer[1024];
        buffer[0] = 0;
        scanf( "%s", buffer );

        // サーバアプリケーションのメソッド呼出し
        obj2->value1( buffer, env );
        printf( "String = %s¥n", obj2->value1( env ) );

        // インスタンスの解放
        target->destroy1( obj2, env );

        // Factoryでインタフェース2用オブジェクトリファレンスを作成する
        ODsample::intf2* obj3 = target->create2( env );

        // サーバアプリケーションのメソッド呼出し
        printf( "value = %d¥n", obj3->value1( env ) );

        CORBA::Long val;
        printf( "add =>" );
        scanf( "%d", &val );
        obj3->add( val, env );

        printf( "  value = %d¥n", obj3->value1( env ) );

        printf( "sub =>");
        scanf( "%d", &val );
        obj3->sub( val, env );
```

```
        printf( "  value = %d¥n", obj3->value1( env ) );

        // インスタンスの解放
        target->destroy2( obj3, env );

        CORBA::release( target );
        CORBA::release( obj2 );
        CORBA::release( obj3 );

    } catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);
    } catch( CORBA::Exception&  e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }
    return 0;
}
```

(5)サーバアプリケーション

```
#include <iostream>
#include <stdlib.h>
#include "orb_cplus.h"
#include "poa.h"
#include "CosNaming_cplus.h"
#include "simple.h"

using std::cout;
using std::endl;

static CORBA::ORB_var                   orb;
static CORBA::Object_var                obj;
static CORBA::Environment               env;
static PortableServer::POA_var          rootPOA;
static PortableServer::POA_var          factoryPOA;
static CORBA::PolicyList                policies( 4 );
static PortableServer::POAManager_var   poa_manager;

static CORBA::Char*                     Str_value = NULL;
static CORBA::Long                      Ln_value = 20;

// ユーザアプリケーション：メイン処理クラス
int main( int argc, char* argv[] )
{
    int current_argc = argc;
    try{
        // ORBの生成と初期化
        orb         = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

        // RootPOAのオブジェクトリファレンスの取得
        obj         = orb->resolve_initial_references( "RootPOA", env );
        // RootPOAのPOAオブジェクト獲得
        rootPOA     = PortableServer::POA::_narrow( obj );

        // Factoryインタフェース用のPOA作成
        // ポリシリスト作成
        policies.length( 4 );
        policies[0] = rootPOA->create_servant_retention_policy(
                                PortableServer::NON_RETAIN, env );
        policies[1] = rootPOA->create_request_processing_policy(
```

```
                                PortableServer::USE_DEFAULT_SERVANT, env );
        policies[2] = rootPOA->create_id_assignment_policy(
                                PortableServer::SYSTEM_ID, env );
        policies[3] = rootPOA->create_id_uniqueness_policy(
                                PortableServer::MULTIPLE_ID, env );
        factoryPOA = rootPOA->create_POA(
                               "IDL:ODsample/Factory:1.0",
                               PortableServer::POAManager::_nil(),
                               policies,
                               env );

        // FactoryServantの生成（コンストラクタにインプリ用POAを渡す）
        ODsample_Factory_impl* svt = new ODsample_Factory_impl();
        // Factoryインタフェース用POAのDefault Servantに設定
        factoryPOA->set_servant( svt, env );

        // AdapterActivatorの生成
        PortableServer::AdapterActivator_var adp = new UserAdapterActivator();
        // インプリ用POAのAdapterActivatorに設定
        rootPOA->the_activator( adp, env );

        // POAマネージャの獲得
        poa_manager = factoryPOA->the_POAManager( env );

        // POAマネージャのactivate
        poa_manager->activate( env );

        orb->run( env );
    } catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);
    } catch( CORBA::Exception&  e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }
    return 0;
}

// ODsample_Factory_impl::create1：メソッドの実装
ODsample::intf1_ptr ODsample_Factory_impl::create1(
                                            CORBA::Environment&  env )
                                            throw( CORBA::Exception )
{
    try{
        // インタフェースリポジトリIDに対応するのPOAオブジェクト獲得
        PortableServer::POA_var poa = rootPOA->find_POA( "intf1", CORBA_TRUE, env );

        // オブジェクトリファレンスの生成
        CORBA::Object_var _tmpObj = poa->create_reference( "IDL:ODsample/intf1:1.0", env );

        ODsample::intf1_ptr ior = ODsample::intf1::_narrow( _tmpObj );

        return ior;
    } catch( CORBA::Exception& e ) {
        cout << "exception-id: " << e.id() << endl;
        throw e;
    }
};

// ODsample_Factory_impl::create2：メソッドの実装
ODsample::intf2_ptr ODsample_Factory_impl::create2(
                                              CORBA::Environment&  env )
                                              throw( CORBA::Exception )
```

```
{
    try{
        // インタフェースリポジトリIDに対応するのPOAオブジェクト獲得
        PortableServer::POA_var poa = rootPOA->find_POA( "intf2", CORBA_TRUE, env );

        // オブジェクトリファレンスの生成
        CORBA::Object_var _tmpObj = poa->create_reference( "IDL:ODsample/intf2:1.0", env );

        ODsample::intf2_ptr ior = ODsample::intf2::_narrow( _tmpObj );

        return ior;
    } catch( CORBA::Exception& e ) {
        cout << "exception-id: " << e.id() << endl;
        throw e;
    }
};

// ODsample_Factory_impl::destroy1：メソッドの実装
void ODsample_Factory_impl::destroy1(
                                ODsample::intf1_ptr  obj,
                                CORBA::Environment&  env )
                                throw( CORBA::Exception )
{
    try{
        // インタフェースIDに対応するのPOAオブジェクト獲得
        PortableServer::POA_var poa = rootPOA->find_POA( "intf1", CORBA_TRUE, env );

        // オブジェクトリファレンスからオブジェクトIDを求める
        PortableServer::ObjectId_var oid = poa->reference_to_id( obj, env );

        // Servantをdeactiveにする
        poa->deactivate_object( *oid, env );

    } catch( CORBA::Exception& e ) {
        cout << "exception-id: " << e.id() << endl;
        throw e;
    }
};

// ODsample_Factory_impl::destroy2：メソッドの実装
void ODsample_Factory_impl::destroy2(
                                 ODsample::intf2_ptr  obj,
                                 CORBA::Environment&  env )
                                 throw( CORBA::Exception )
{
    try{
        // インタフェースIDに対応するのPOAオブジェクト獲得
        PortableServer::POA_var poa = rootPOA->find_POA( "intf2", CORBA_TRUE, env );

        // オブジェクトリファレンスからオブジェクトIDを求める
        PortableServer::ObjectId_var oid = poa->reference_to_id( obj, env );

        // Servantをdeactiveにする
        poa->deactivate_object( *oid, env );

    } catch( CORBA::Exception& e ) {
        cout << "exception-id: " << e.id() << endl;
        throw e;
    }
};

// AdapterActivator：POA生成クラス（AdapterActivatorを継承）
class UserAdapterActivator : public PortableServer::AdapterActivator
```

```
{
public:
    UserAdapterActivator() {};
    ~UserAdapterActivator() {};

    CORBA::Boolean unknown_adapter(
                            PortableServer::POA_ptr parent,
                            const CORBA::Char*      name,
                            CORBA::Environment&     env )
                            throw( CORBA::Exception )
    {
        // ポリシリスト作成
        CORBA::PolicyList    policies( 4 );
        policies.length( 4 );
        policies[0] = rootPOA->create_servant_retention_policy(
                                PortableServer::RETAIN, env );
        policies[1] = rootPOA->create_request_processing_policy(
                                PortableServer::USE_SERVANT_MANAGER, env );
        policies[2] = rootPOA->create_id_assignment_policy(
                                PortableServer::SYSTEM_ID, env );
        policies[3] = rootPOA->create_id_uniqueness_policy(
                                PortableServer:: MULTIPLE_ID, env );

        // ServantActivatorの生成
        PortableServer::ServantActivator_ptr svtact = new UserServantActivator();

        try {
            if( !strcmp( name, "intf1" ) ) {
                // intf1用のPOA作成
                PortableServer::POA_ptr intf1POA
                    = parent->create_POA( name, NULL, policies, env );

                // ServantManager登録
                intf1POA->set_servant_manager( svtact , env );

                return( CORBA_TRUE );
            }
            else if( !strcmp( name, "intf2" ) ) {
                // intf2用のPOA作成
                PortableServer::POA_ptr intf2POA
                    = parent->create_POA( name, NULL, policies, env );

                // ServantManager登録
                intf2POA->set_servant_manager( svtact , env );

                return( CORBA_TRUE );
            }
        } catch( CORBA::SystemException&  se ) {
            cout << "exception-id: " << se.id() << endl;
            return( CORBA_FALSE );
        } catch( CORBA::UserException&  ue ) {
            cout << "exception-id: " << ue.id() << endl;
            return( CORBA_FALSE );
        } catch( CORBA::Exception&  e ) {
            cout << "exception-id: " << e.id() << endl;
            return( CORBA_FALSE );
        }
        return CORBA_TRUE;
    };
};

// ServantActivator：Servant生成クラス（ServantActivatorを継承）
class UserServantActivator : public PortableServer::ServantActivator
```

```
{
public:
    UserServantActivator() {};
    ~UserServantActivator() {};

    PortableServer::Servant incarnate(
                                const PortableServer::ObjectId& oid,
                                PortableServer::POA_ptr         adapter,
                                CORBA::Environment&             env )
                                throw( CORBA::Exception )
    {
        PortableServer::Servant svt = NULL;

        CORBA::Char* name = adapter->the_name( env );
        if( !strcmp( name, "intf1" ) ) {
            svt = new ODsample_intf1_impl();
        } else if( !strcmp( name, "intf2" ) ) {
            svt = new ODsample_intf2_impl();
        } else {
            CORBA::StExcep::BAD_PARAM ex;
            throw ex;
        }
        CORBA::string_free( name );
        return( svt );
    };

    void etherealize(
               const PortableServer::ObjectId& oid,
               PortableServer::POA_ptr         adapter,
               PortableServer::Servant         serv,
               CORBA::Boolean                  cleanup_in_progress,
               CORBA::Boolean                  remaining_activations,
               CORBA::Environment&             env )
               throw( CORBA::Exception )
    {
        serv->servant_delete();
    }
};

// メソッドの実装
void  ODsample_intf1_impl::value1(
                            const CORBA::Char*   a,
                            CORBA::Environment&  env )
                            throw( CORBA::Exception )
{
    cout << "ODsample::intf1::value(Char*)" << endl;
    cout << "  value : " << a << endl;
    if( Str_value != NULL ) {
        CORBA::string_free( Str_value );
    }
    Str_value = CORBA::string_alloc( strlen( a ) + 1 );
    strcpy( Str_value, a );
};

CORBA::Char*  ODsample_intf1_impl::value1(
                                  CORBA::Environment&  env )
                                  throw( CORBA::Exception )
{
    if( Str_value == NULL ) {
        Str_value = CORBA::string_alloc( 1 );
        Str_value[0] = 0;
    }
    CORBA::Char* ret = CORBA::string_alloc( strlen( Str_value ) + 1 );
```

```
    strcpy( ret, Str_value );
    return ret;
};

CORBA::Long  ODsample_intf2_impl::value1(
                                  CORBA::Environment&  env )
                                  throw( CORBA::Exception )
{
    return Ln_value;
};

void  ODsample_intf2_impl::add(
                               CORBA::Long          a,
                               CORBA::Environment&  env )
                               throw( CORBA::Exception )
{
    cout << "ODsample::intf2::add()" << endl;
    cout << "  value : " << Ln_value << endl;
    cout << "  a     : " << a << endl;
    Ln_value = Ln_value + a;
    cout << "  result: " << Ln_value << endl;
};

void  ODsample_intf2_impl::sub(
                               CORBA::Long          b,
                               CORBA::Environment&  env )
                               throw( CORBA::Exception )
{
    cout << "ODsample::intf2::sub()" << endl;
    cout << "  value : " << Ln_value << endl;
    cout << "  b     : " << b << endl;
    Ln_value = Ln_value - b;
    cout << "  result: " << Ln_value << endl;
};
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法の詳細については、“4.2.2 サーバアプリケーションの例外処理”を参照してください。

4.3.7.6 AdapterActivator使用例（リクエスト受信時）

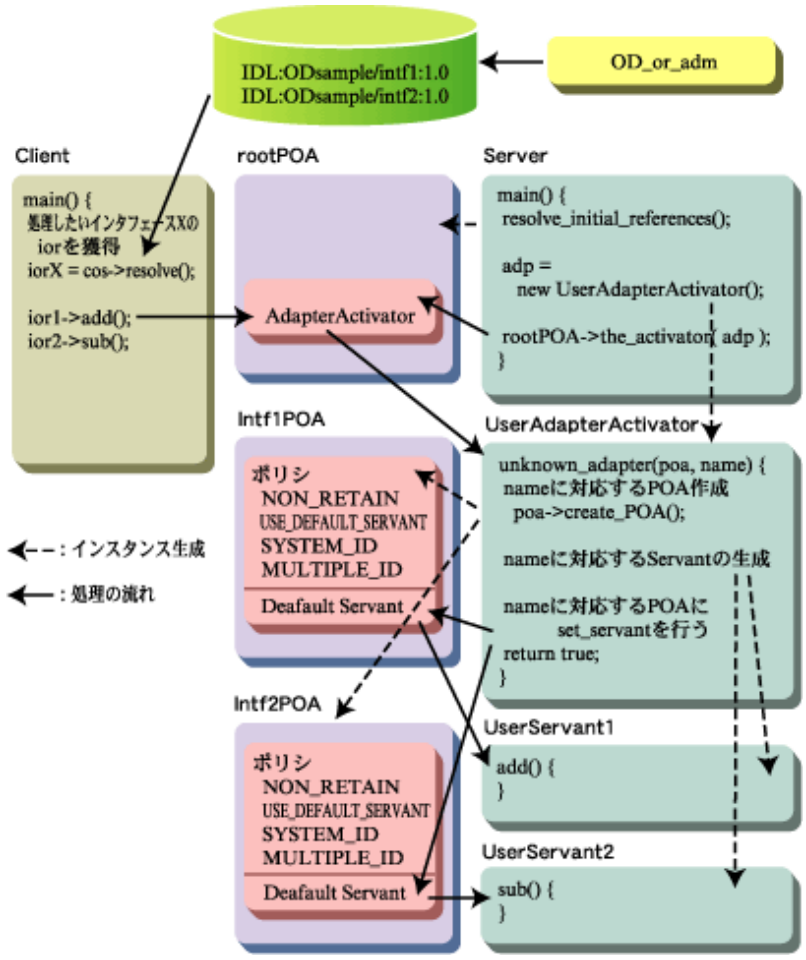
(1)IDL定義

```
module ODsample{
    interface    intf1{
        long  add( in long a, in long b );
    };
    interface  intf2{
        long  sub( in long a, in long b );
    };
};
```

(2)ネーミングサービスの登録

```
OD_or_adm -c IDL:ODsample/intf1:1.0  -a Imple_POAsample10 -n ODsample::POAsample10-1
OD_or_adm -c IDL:ODsample/intf2:1.0  -a Imple_POAsample10 -n ODsample::POAsample10-2
```

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
#include <stdlib.h>
#include <iostream>
#include "orb_cplus.h"
#include "CosNaming_cplus.h"
#include "simple.h"

using std::cout;
using std::endl;

static  CORBA::Object_var             obj;
static  CORBA::Environment            env;
static  CORBA::ORB_var                orb;
static  CosNaming::NamingContext_var  cos_naming;
static  CosNaming::Name               name;
static  CosNaming::Name               name2;

int main( int argc, char* argv[] )
{
    CORBA::Long   val1 = 0;      // inパラメタ用変数
    CORBA::Long   val2 = 0;      // inパラメタ用変数

    try {
        // ORBの生成と初期化
```

```
        orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, env );

        // ネーミングサービスのオブジェクトリファレンスの取得
        obj = orb->resolve_initial_references( CORBA_ORB_ObjectId_NameService, env );
        cos_naming = CosNaming::NamingContext::_narrow( obj );

        // ネーミングサービスのresolveメソッドを発行し、
        // サーバアプリケーションのオブジェクトリファレンスの獲得
        name.length( 1 );
        name[0]->id   = ( const CORBA::Char* )"ODsample::POAsample10-1";
        name[0]->kind = ( const CORBA::Char* )"";
        obj = cos_naming->resolve( name, env );

        // サーバアプリケーションのオブジェクトリファレンス獲得
        ODsample::intf1* target1 = ODsample::intf1::_narrow( obj );

        printf( "val1=>" );
        scanf( "%d", &val1 );
        printf( "val2=>" );
        scanf( "%d", &val2 );

        // サーバアプリケーションのメソッド呼出し
        CORBA::Long  ret1 = target1->add( val1, val2, env );

        // メソッドの結果表示
        printf( "val1 + val2 = %d¥n", ret1 );

        CORBA::release( target1 );

        // ネーミングサービスのresolveメソッドを発行し、
        // サーバアプリケーションのオブジェクトリファレンスの獲得
        name2.length( 1 );
        name2[0]->id   = ( const CORBA::Char* )"ODsample::POAsample10-2";
        name2[0]->kind = ( const CORBA::Char* )"";
        obj = cos_naming->resolve( name2, env );

        // サーバアプリケーションのオブジェクトリファレンス獲得
        ODsample::intf2* target2 = ODsample::intf2::_narrow( obj );

        printf( "val1=>" );
        scanf( "%d", &val1 );
        printf( "val2=>" );
        scanf( "%d", &val2 );

        // サーバアプリケーションのメソッド呼出し
        CORBA::Long  ret2 = target2->sub( val1, val2, env );

        // メソッドの結果表示
        printf( "val1 - val2 = %d¥n", ret2 );

        CORBA::release( target2 );
    } catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);
    } catch( CORBA::Exception&  e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }
    return 0;
}
```

(5)サーバアプリケーション

```
#include <iostream>
#include <stdlib.h>
#include "orb_cplus.h"
#include "poa.h"
#include "CosNaming_cplus.h"
#include "simple.h"

using std::cout;
using std::endl;

static CORBA::ORB_var                   orb;
static CORBA::Object_var                obj;
static CORBA::Environment               env;
static PortableServer::POA_var          rootPOA;
static CORBA::PolicyList                policies( 4 );
static PortableServer::POAManager_var   poa_manager;

// ユーザアプリケーション：メイン処理クラス
int main( int argc, char* argv[] )
{
    int current_argc = argc;
    try{
        // ORBの生成と初期化
        orb         = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

        // RootPOAのオブジェクトリファレンスの取得
        obj         = orb->resolve_initial_references( "RootPOA", env );
        // RootPOAのPOAオブジェクト獲得
        rootPOA     = PortableServer::POA::_narrow( obj );

        // AdapterActivatorの生成
        PortableServer::AdapterActivator_var adp = new UserAdapterActivator();
        // インプリ用POAのAdapterActivatorに設定
        rootPOA->the_activator( adp, env );

        // POAマネージャの獲得
        poa_manager = rootPOA->the_POAManager( env );
        // POAマネージャのactivate
        poa_manager->activate( env );

        orb->run( env );
    } catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);
    } catch( CORBA::Exception&  e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }
    return 0;
}

// AdapterActivator：POA生成クラス（AdapterActivatorを継承）
class UserAdapterActivator : public PortableServer::AdapterActivator
{
public:
    UserAdapterActivator() {};
    ~UserAdapterActivator() {};

    CORBA::Boolean  unknown_adapter(
                            PortableServer::POA_ptr parent,
                            const CORBA::Char*      name,
```

```
                            CORBA::Environment&     env )
                            throw( CORBA::Exception )
    {
        // ポリシリスト作成
        CORBA::PolicyList    policies( 4 );
        policies.length( 4 );
        policies[0] = rootPOA->create_servant_retention_policy(
                            PortableServer::NON_RETAIN, env );
        policies[1] = rootPOA->create_request_processing_policy(
                            PortableServer::USE_DEFAULT_SERVANT, env );
        policies[2] = rootPOA->create_id_assignment_policy(
                            PortableServer::SYSTEM_ID, env );
        policies[3] = rootPOA->create_id_uniqueness_policy(
                            PortableServer::MULTIPLE_ID, env );
        try {
            if( !strcmp( name, "IDL:ODsample/intf1:1.0" ) ) {
                // intf1用のPOA作成
                PortableServer::POA_ptr intf_POA
                  = parent->create_POA( name, NULL, policies, env );

                // Servantの生成
                ODsample_intf1_impl* svt = new ODsample_intf1_impl();

                // Default Servantに設定
                intf_POA->set_servant( svt, env );

                return( CORBA_TRUE );
            }
            else if( !strcmp( name, "IDL:ODsample/intf2:1.0" ) ) {
                // intf2用のPOA作成
                PortableServer::POA_ptr intf_POA
                    = parent->create_POA( name, NULL, policies, env );

                // Servantの生成
                ODsample_intf2_impl* svt = new ODsample_intf2_impl();

                // Default Servantに設定
                intf_POA->set_servant( svt, env );

                return( CORBA_TRUE );
            }
        } catch( CORBA::SystemException&  se ) {
            cout << "exception-id: " << se.id() << endl;
            return( CORBA_FALSE );
        } catch( CORBA::UserException&  ue ) {
            cout << "exception-id: " << ue.id() << endl;
            return( CORBA_FALSE );
        } catch( CORBA::Exception&  e ) {
            cout << "exception-id: " << e.id() << endl;
            return( CORBA_FALSE );
        }
        return CORBA_TRUE;
    };
};

// メソッド実装
CORBA::Long  ODsample_intf1_impl::add(
                                    CORBA::Long          a,
                                    CORBA::Long          b,
                                    CORBA::Environment&  env )
                                    throw( CORBA::Exception )
{
    return a + b;
```

```
};

CORBA::Long  ODsample_intf2_impl::sub(
                                    CORBA::Long  a,
                                    CORBA::Long  b,
                                    CORBA::Environment&  env )
                                    throw( CORBA::Exception )
{
    return a - b;
};
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法の詳細については、“4.2.2 サーバアプリケーションの例外処理”を参照してください。

4.3.7.7 代理方式の実装例(デフォルトインスタンス方式の例)

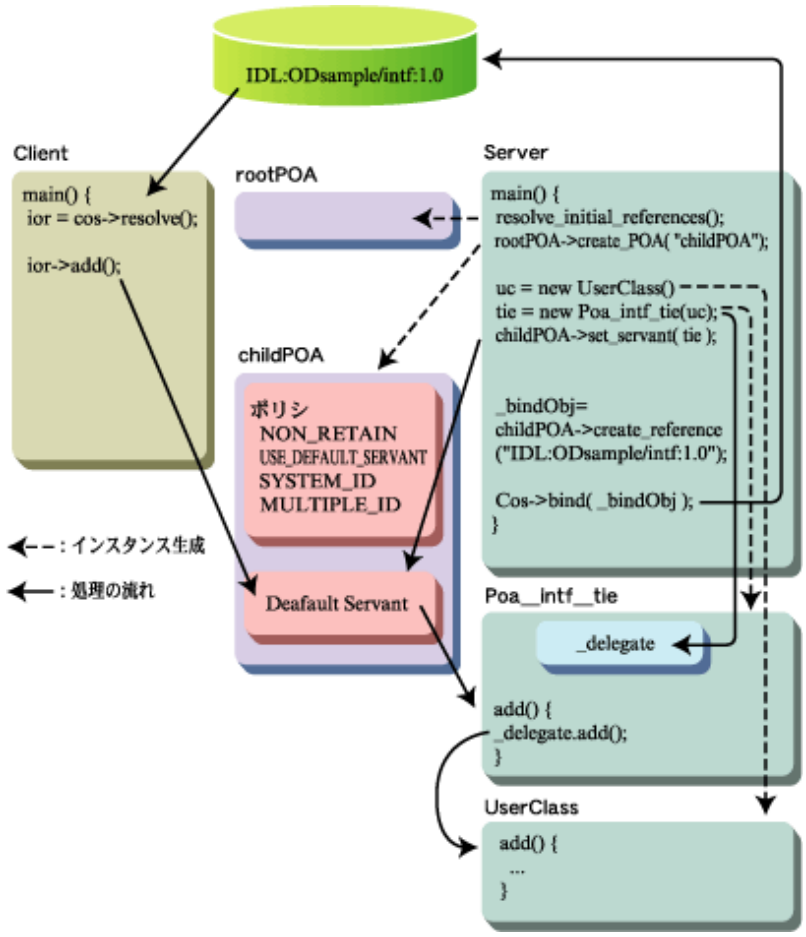
(1)IDL定義

```
module ODsample{
    interface  intf{
        long  add( in  long  a,  in  long  b );
    };
};
```

(2)ネーミングサービスの登録

動的登録

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
#include <stdlib.h>
#include <iostream>
#include "orb_cplus.h"
#include "CosNaming_cplus.h"
#include "simple.h"

using std::cout;
using std::endl;

static  CORBA::Object_var            obj;
static  CORBA::ORB_var               orb;
static  CORBA::Environment           env;
static  CosNaming::NamingContext_var  cos_naming;
static  CosNaming::Name              name;

int main( int argc, char* argv[] )
{
    try {
        // ORBの生成と初期化
        orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, env );

        // ネーミングサービスのオブジェクトリファレンスの取得
        obj = orb->resolve_initial_references(
                    CORBA_ORB_ObjectId_NameService, env );
        cos_naming = CosNaming::NamingContext::_narrow( obj );

        // ネーミングサービスのresolveメソッドを発行し、
```

```
        // サーバアプリケーションのオブジェクトリファレンスの獲得
        name.length( 1 );
        name[0]->id   = ( const CORBA::Char* )"ODsample::POAsample1";
        name[0]->kind = ( const CORBA::Char* )"";
        obj = cos_naming->resolve( name, env );

        // サーバアプリケーションのオブジェクトリファレンス獲得
        ODsample::intf* obj2 = ODsample::intf::_narrow( obj );

        CORBA::Long  a;     // inパラメタ用変数
        CORBA::Long  b;     // inパラメタ用変数
        CORBA::Long  ret;   // 復帰値用変数

        printf( "  a => " );
        scanf( "%d", &a );
        printf( "  b => " );
        scanf( "%d", &b );

        // サーバアプリケーションのメソッド呼出し
        ret = obj2->add( a, b, env );

        // メソッドの結果表示
        printf( "  a + b = %d", ret );

        CORBA::release( obj2 );
    }catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);
    }catch( CORBA::Exception& e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }
    return 0;
}
```

(5)サーバアプリケーション

```
#include <iostream>
#include <stdlib.h>
#include "orb_cplus.h"
#include "poa.h"
#include "CosNaming_cplus.h"
#include "simple.h"

using std::cout;
using std::endl;

static CORBA::ORB_var                   orb;
static CORBA::Object_var                obj;
static CORBA::Environment               env;
static CosNaming::NamingContext_var     cos_naming;
static CosNaming::Name                  name;
static PortableServer::POA_var          rootPOA;
static PortableServer::POA_var          childPOA;
static CORBA::PolicyList                policies( 4 );
static PortableServer::POAManager_var   poa_manager;

// ユーザアプリケーション：メイン処理クラス
int main( int argc, char*argv[] )
{
    int current_argc = argc;
```

```
    try{
        // ORBの生成と初期化
        orb         = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

        // RootPOAのオブジェクトリファレンスの取得
        obj         = orb->resolve_initial_references( "RootPOA", env );

        // RootPOAのPOAオブジェクト獲得
        rootPOA     = PortableServer::POA::_narrow( obj );

        // インタフェース用のPOA作成
        // ポリシリスト作成
        policies.length( 4 );
        policies[0] = rootPOA->create_servant_retention_policy(
                            PortableServer::NON_RETAIN, env );
        policies[1] = rootPOA->create_request_processing_policy(
                            PortableServer::USE_DEFAULT_SERVANT, env );
        policies[2] = rootPOA->create_id_assignment_policy(
                            PortableServer::SYSTEM_ID, env );
        policies[3] = rootPOA->create_id_uniqueness_policy(
                            PortableServer::MULTIPLE_ID, env );

        // Default Servant用のPOAオブジェクトの作成
        childPOA   = rootPOA->create_POA( "childPOA",
                        PortableServer::POAManager::_nil(), policies, env );

        // tieオブジェクト作成とユーザクラスの登録
        PortableServer::Servant svt = _ct_ODsample_intf();

        // POAのDefault Servantに設定
        childPOA->set_servant( svt, env );

        CORBA::Object_var _bind_obj = childPOA->create_reference( "IDL:ODsample/intf:1.0", env );

        // ネーミングサービスのオブジェクトリファレンスの取得
        CORBA::Object_var _tmpObj = orb->resolve_initial_references( "NameService", env );
        cos_naming = CosNaming::NamingContext::_narrow( _tmpObj );

        // サーバアプリケーションのオブジェクトリファレンスの獲得
        name.length( 1 );
        name[0]->id   = ( const CORBA::Char* )"ODsample::POAsample1";
        name[0]->kind = ( const CORBA::Char* )"";
        try{
          cos_naming->unbind( name, env );
        } catch( CORBA::Exception& e ) {
            ;
        }
        cos_naming->bind( name, _bind_obj, env );

        // POAマネージャの獲得
        poa_manager = childPOA->the_POAManager( env );

        // POAマネージャのactivate
        poa_manager->activate( env );

        orb->run( env );
    } catch( CORBA::Exception& e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }
    return 0;
}
```

```
//ユーザクラス（インタフェース実装クラス）
class UserServant
{
    public:
    UserServant(){};
    ~UserServant(){};
    CORBA::Long add(
                    CORBA::Long a,
                    CORBA::Long b,
                    CORBA::Environment& env )
                    throw( CORBA::Exception )
    {

        CORBA::Long    result;

        cout << "ODsample::intf::add()" << endl;
        cout << "  a     : " << a << endl;
        cout << "  b     : " << b << endl;
        result = a + b;
        cout << "  result: " << result << endl;
        return result;
    };
};

POA_ODsample_intf* _ct_ODsample_intf()
{
    // ユーザクラスのインスタンス生成
    UserServant* impl = new UserServant();
    // tieオブジェクト作成とユーザクラスの登録
    POA_ODsample_intf_tie<UserServant>* tie =
        new POA_ODsample_intf_tie<UserServant>( impl );
    return tie;
};
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法の詳細については、“4.2.2 サーバアプリケーションの例外処理”を参照してください。

4.3.7.8 Active Object Map（AOM）使用例（Factory-1方式＋クライアントとのコネクション切断時のインスタンス解放）

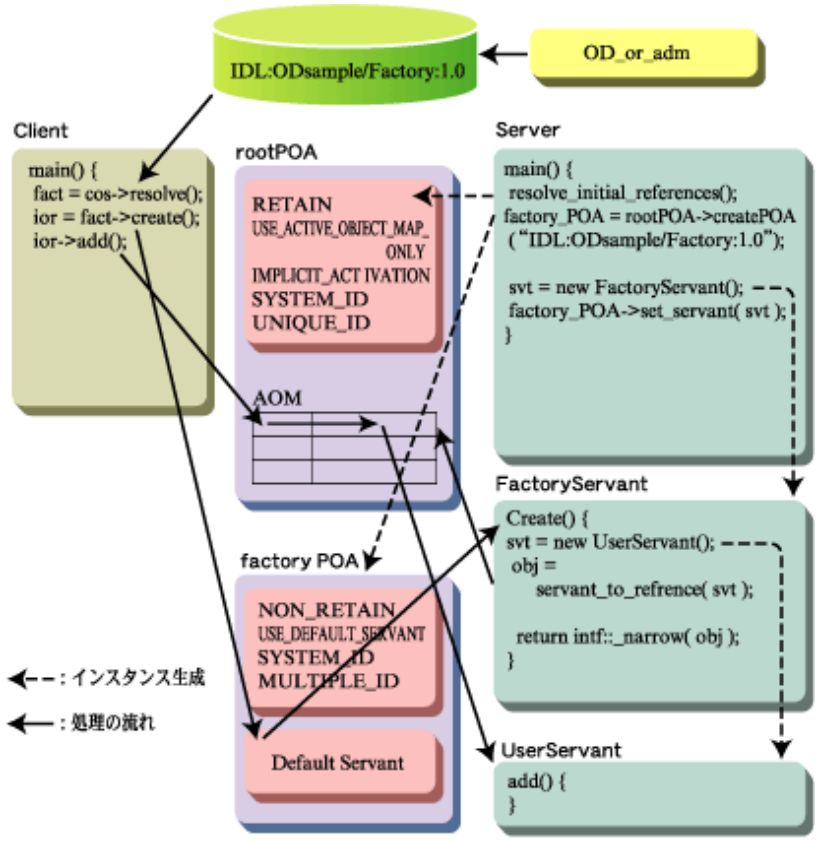
(1)IDL定義

```
module ODsample{
    interface intf{
        readonly  attribute  long  value1;
        void  add( in  long  a );
        void  sub( in  long  b );
    };
    interface Factory {
        intf   create( in string username );
        void   destroy( in  intf  obj );
    };
};
```

(2)ネーミングサービスの登録

```
OD_or_adm -c IDL:ODsample/Factory:1.0 -a Imple_POAsample5 -n ODsample::POAsample5
```

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
#include <iostream>
#include <stdlib.h>
#include "orb_cplus.h"
#include "CosNaming_cplus.h"
#include "simple.h"
#include <string.h>

using std::cin;
using std::cout;
using std::endl;

int main( int argc, char *argv[] )
{
    CORBA::Environment env;
    try {
        // ORBの生成と初期化
        CORBA::ORB_ptr orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, env );

        // ネーミングサービスのオブジェクトリファレンスの取得
        CORBA::Object_ptr obj = orb->resolve_initial_references(
            CORBA_ORB_ObjectId_NameService, env );
        CosNaming::NamingContext_ptr ctx = CosNaming::NamingContext::_narrow( obj );
        CORBA::release( obj );

        // ネーミングサービスのresolveメソッドを発行し、
```

```
        // サーバアプリケーションのオブジェクトリファレンスの獲得
        CosNaming::Name name;
        name.length(1);
        name[0]->id   = ( const CORBA::Char* )"ODsample::POAsample5";
        name[0]->kind = ( const CORBA::Char* )"";

        obj = ctx->resolve( name, env );

        // Factoryのオブジェクトリファレンスの獲得
        ODsample::Factory_ptr factory = ODsample::Factory::_narrow( obj );
        CORBA::release( obj );

        // ユーザIDの入力
        char _username[128];
        cout << "input your name  => ";
        cin >> _username;

        // Factoryでインタフェース用オブジェクトリファレンス作成
        ODsample::intf_ptr target = factory->create( _username );

        // サーバアプリケーションのメソッド呼出し
        cout << "value = " << target->value1() << endl;
        cout << "add => ";
        CORBA::Long a;
        cin >> a;

        target->add( a );

        cout << "value = " << target->value1() << endl;
        cout << "sub => ";
        CORBA::Long b;
        cin >> b;

        target->sub( b );

        cout << "value = " << target->value1() << endl;

        CORBA::release( factory );
        CORBA::release( target );

    } catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);

    } catch( CORBA::Exception&  e ) {
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }

    return 0;
}
```

(5)サーバアプリケーション

```
#include <iostream>
#include <stdlib.h>

#include "orb_cplus.h"
#include "simple.h"
#include "poa.h"
#include "CosNaming_cplus.h"
```

```
using std::cout;
using std::endl;

static CORBA::ORB_var                   orb;
static CORBA::Object_var                obj;
static CORBA::Environment               env;

static PortableServer::POA_var          rootPOA;
static PortableServer::POA_var          factoryPOA;
static CORBA::PolicyList                factory_policies;
static PortableServer::POAManager_var   poa_manager;

// ユーザアプリケーション：メイン処理クラス
int main( int argc, char*argv[] )
{
    int current_argc = argc;

    try{
        // ORBの生成と初期化
        orb         = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

        // RootPOAのオブジェクトリファレンスの取得
        obj         = orb->resolve_initial_references( "RootPOA", env );

        // RootPOAのPOAオブジェクト獲得
        rootPOA     = PortableServer::POA::_narrow( obj );

        // Factoryインタフェース用のPOA作成
        // ポリシリスト作成
        factory_policies.length( 4 );
        factory_policies[0] = rootPOA->create_servant_retention_policy(
                                PortableServer::NON_RETAIN, env );
        factory_policies[1] = rootPOA->create_request_processing_policy(
                                PortableServer::USE_DEFAULT_SERVANT, env );
        factory_policies[2] = rootPOA->create_id_assignment_policy(
                                PortableServer::SYSTEM_ID, env );
        factory_policies[3] = rootPOA->create_id_uniqueness_policy(
                                PortableServer::MULTIPLE_ID, env );

        factoryPOA = rootPOA->create_POA( "IDL:ODsample/Factory:1.0",
                 PortableServer::POAManager::_nil(), factory_policies ,env);

        // FactoryServantの生成
        PortableServer::Servant  svt = _ct_ODsample_Factory( rootPOA );

        // Factoryインタフェース用POAをDefault Servantに設定
        factoryPOA->set_servant( svt, env );

        // POAマネージャの獲得
        poa_manager = rootPOA->the_POAManager( env );

        // インスタンス解放処理の登録
        PortableServer::POAdiconnect::setDisconnect(new instrelease());

        // POAマネージャのactivate
        poa_manager->activate( env );

        orb->run( env );

    } catch( CORBA::SystemException&  se ) {
        cout << "exception-id: " << se.id() << endl;
        exit(1);
    } catch( CORBA::Exception&  e ) {
```

```
        cout << "exception-id: " << e.id() << endl;
        exit(1);
    }

    return 0;
}

// FactoryServant：Factoryメソッド実装クラス
class FactoryServant
{
    public :
    // コンストラクタ
    FactoryServant(
        PortableServer::POA_ptr  poa )
    {
        _poa = poa;
    }

    ~FactoryServant()
    {
        CORBA::release( _poa );
    }

    ODsample::intf_ptr create(
        const CORBA::Char *username,
        CORBA::Environment&  env )
        throw( CORBA::Exception )
    {
        if( !_poa ) return NULL;
        ODsample::intf_ptr _ior = NULL;
        try{
            if( servant_list == NULL ){
                servant_list = new ServantList();
            }

            // Servantの検索
            PortableServer::ServantBase *_svt = servant_list->get( username );

            // みつからない場合、Servantを作成 (guestは毎回作成)
            if( !_svt ){
                _svt = _ct_ODsample_intf( username );
                servant_list->put( username, _svt );
                if( 100 < servant_list->length()){
                  PortableServer::POAdisconnect::setDisconnect(new ALLinstrelease());
                }
            }

            // Servantからオブジェクトリファレンスの生成
            // IMPLICIT_ACTIVATIONポリシを指定しているので、AOMへ自動登録される
            CORBA::Object_ptr _obj = _poa->servant_to_reference( _svt, env );
            _ior = ODsample::intf::_narrow( _obj );

            CORBA::release( _obj );
        } catch( CORBA::Exception &e ) {
            cout << "exception-id: " << e.id() << endl;
            throw e;
        }
        return _ior;
    };


    void destroy(
        ODsample::intf_ptr obj,
```

```
        CORBA::Environment& env )
        throw( CORBA::Exception )
    {
        if( !_poa ) return;
        try{
            // オブジェクトリファレンスからオブジェクトIDを求める
            PortableServer::ObjectId_var _oid = _poa->reference_to_id( obj, env );

            // オブジェクトIDからServantを求める
            PortableServer::Servant _svt = _poa->id_to_servant( *_oid, env );

            // Servantをdeactiveにする
            _poa->deactivate_object( *_oid, env );

            // Servantの削除
            _svt->servant_delete();
        } catch( CORBA::Exception& e ) {
            cout << "exception-id: " << e.id() << endl;
            throw e;
        }
    };
    private :
    FactoryServant() {}
    PortableServer::POA_ptr    _poa;
};

// FactoryServantをtieオブジェクトに関連付けるためのメソッド（delegate方式の場合）
POA_ODsample_Factory*  _ct_ODsample_Factory(
    PortableServer::POA_ptr poa )
{
    FactoryServant*  impl = new FactoryServant( poa );
    POA_ODsample_Factory_tie< FactoryServant >*  tie
        = new POA_ODsample_Factory_tie< FactoryServant >( impl );
    return tie;
}

// Servant：メソッド実装クラス（スケルトンを継承）
class UserServant
{
    public :
    UserServant(
        const CORBA::Char *username )
    {
        _username = new char[ strlen(username) + 1 ];
        strcpy( _username, username );
        _val = 10;
    }
    ~UserServant()
    {
        if( _username )
            delete [] _username;
    }

    void add(
        CORBA::Long a,
        CORBA::Environment& env)
        throw( CORBA::Exception )
    {
        cout << "ODsample::intf::add()" << endl;
        cout << "  val : " << _val << endl;
        cout << "  a     : " << a << endl;
        _val = _val + a;
        cout << "  result: " << _val << endl;
```

```
    }

    void sub(
        CORBA::Long b,
        CORBA::Environment& env)
        throw( CORBA::Exception )
    {
        cout << "ODsample::intf::sub()" << endl;
        cout << "  val : " << _val << endl;
        cout << "  b     : " << b << endl;
        _val = _val - b;
        cout << "  result: " << _val << endl;
    }

    CORBA::Long value1(
        CORBA::Environment& env)
        throw( CORBA::Exception )
    {
        return _val;
    }

    const char *name()
    {
        return _username;
    }

private :
    UserServant() {}
    char *_username;
    long _val;
};

// UserServantをtieオブジェクトに関連付けるためのメソッド（delegate方式の場合）
POA_ODsample_intf *_ct_ODsample_intf(
      const CORBA::Char *username )
{
    UserServant*  impl = new UserServant( username );
    POA_ODsample_intf_tie< UserServant >*  tie
          = new POA_ODsample_intf_tie< UserServant >( impl );
    return tie;
}

// 条件付きインスタンス解放クラス（POAdisconnectを継承）
class instrelease : public virtual PortableServer::POAdisconnect
{
    public :
    instrelease() {}
    ~instrelease() {}

    CORBA::Boolean release_instance(
        PortableServer::POA_ptr poa,
        PortableServer::ServantBase *svt,
        PortableServer::ObjectId&  oid,
        CORBA::Environment&  env  )
    {
        cout << "instrelease::release_instance called..." << endl;
        POA_ODsample_intf_tie< UserServant >*  _tie
                        = dynamic_cast< POA_ODsample_intf_tie< UserServant >* >( svt );
        UserServant*  _svt = _tie->_tied_object();
        cout << "    servant->name : " << _svt->name() << endl;
        if( !strcmp( _svt->name(), "guest" )){
            ServantList::remove( servant_list, "guest" );
            return true;
```

```
        }
        return false;
    }
};

// 無条件インスタンス解放クラス（POAdisconnectを継承）
class ALLinstrelease : public virtual PortableServer::POAdisconnect
{
    public :
    ALLinstrelease() {}
    ~ALLinstrelease() {}

    CORBA::Boolean release_instance(
        PortableServer::POA_ptr poa,
        PortableServer::ServantBase *svt,
        PortableServer::ObjectId&  oid,
        CORBA::Environment&  env  )
    {
        cout << "ALLinstrelease::release_instance called..." << endl;
        POA_ODsample_intf_tie< UserServant >*  _tie
                        = dynamic_cast< POA_ODsample_intf_tie< UserServant >* >( svt );
        UserServant*  _svt = _tie->_tied_object();
        cout << "    servant->name : " << _svt->name() << endl;
        ServantList::remove( servant_list, _svt->name());
        return true;
    }
};

// Servant管理テーブル
class ServantList
{
    public:
    ServantList()
    {
        _svt = NULL;
        _username = NULL;
        _next = NULL;
    };
    ServantList(
        const CORBA::Char *username,
        PortableServer::ServantBase *svt )
    {
        _username = new char[ strlen(username) + 1];
        strcpy( _username, username );
        _svt = svt;
        _next = NULL;
    };
    ~ServantList()
    {
        if( _username )
            delete [] _username;
        if( _next )
            delete _next;
    };

    PortableServer::ServantBase *get(
        const CORBA::Char *username )
    {
        if( !username || !_username )
            return NULL;
        if( !strcmp( username, _username )){
            return _svt;
        }
```

```
        if( _next )
            return _next->get(username);
        return NULL;
    }

    void put(
        const CORBA::Char *username,
        PortableServer::ServantBase *svt )
    {
        if( _next != NULL ){
            _next->put( username, svt );
        }
        else{
            if( _svt == NULL ){
                _svt = svt;
                _username = new char[ strlen(username) + 1];
                strcpy( _username, username );
            }
            else{
                _next = new ServantList( username, svt );
            }
        }
    }

    static void remove(
        ServantList *&list,
        const CORBA::Char *username )
    {
        ServantList *_back = NULL;
        for( ServantList *_list = list; _list; _list = _list->_next ){
            if( !strcmp( _list->_username, username )){
                if( _back ){
                    _back->_next = _list->_next;
                }
                else{
                    list = _list->_next;
                }
                _list->_next = NULL;
                delete _list;
                break;
            }
            _back = _list;
        }
        return;
    }

    int length()
    {
        ServantList *_p = _next;
        int _count = 1;
        while( _p ){
            _p = _p->_next;
            ++_count;
        }
        return _count;
    }

    private :
    PortableServer::ServantBase *_svt;
    char *_username;
    ServantList *_next;
};
```

```
// 管理テーブルのインスタンス
ServantList *servant_list = NULL;
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法の詳細については、“4.2.2 サーバアプリケーションの例外処理”を参照してください。

4.4 サーバアプリケーションの登録

インプリメンテーションリポジトリ／ネーミングサービスに、作成したアプリケーションを登録します。

4.4.1 インプリメンテーションリポジトリへの登録

OD_impl_instコマンドを使用してインプリメンテーションリポジトリにサーバアプリケーション情報を登録します。
OD_impl_instコマンドでの登録処理の例と指定する情報について以下に示します。

例

インプリメンテーションリポジトリへの登録

```
OD_impl_inst -ax def
```

-ax def

定義ファイルで指定した定義情報をサーバアプリケーション情報として登録します。

Windows32/64

defファイルの内容

```
rep_id = IDL:ODdemo/calculator:1.0
type   = shared
binary = D:¥server¥simple_s.exe
mode   = SYNC_END
ior    = 1.1
locale = SJIS
```

rep_id = IDL:ODdemo/calculator:1.0

インプリメンテーションリポジトリIDを指定します。

type = shared

サーバタイプを指定します。指定できるタイプには、persistent、shared、unsharedがあります。

binary = D:¥server¥simple_s.exe

サーバアプリケーションのパス名を指定します。

mode = SYNC_END

サーバアプリケーションがCORBA::BOA::impl_is_ready()／CORBA::BOA::obj_is_ready()発行時の動作モードを指定します。指定できる動作モードには、COMPATIBLE、SYNC_ENDがあります。

ior = 1.1

連携するIORのバージョンを指定します。指定できるバージョンには、1.0、1.1があります。

locale = SJIS

サーバアプリケーションのコード系を指定します。

Solaris32/64 Linux32/64

defファイルの内容

```
rep_id = IDL:ODdemo/calculator:1.0
type   = shared
binary = /home/guest/simple_s
uid    = user
gid    = group
mode   = SYNC_END
ior    = 1.1
locale = EUC
```

rep_id = IDL:ODdemo/calculator:1.0

インプリメンテーションリポジトリIDを指定します。

type = shared

サーバタイプを指定します。指定できるタイプには、persistent、shared、unsharedがあります。

binary = /home/guest/simple_s

サーバアプリケーションのパス名を指定します。

uid = user

サーバアプリケーション実行時のユーザIDを指定します。

gid = group

サーバアプリケーション実行時のグループIDを指定します。

mode = SYNC_END

サーバアプリケーションがCORBA::BOA::impl_is_ready()／CORBA::BOA::obj_is_ready()発行時の動作モードを指定します。指定できる動作モードには、COMPATIBLE、SYNC_ENDがあります。

ior = 1.1

連携するIORのバージョンを指定します。指定できるバージョンには、1.0、1.1があります。

locale = EUC

サーバアプリケーションのコード系を指定します。

4.4.2 ネーミングサービスへの登録

作成したサーバアプリケーションを他のアプリケーションからオブジェクトとしてアクセスできるようにするために、そのオブジェクトを識別するためのオブジェクトリファレンスを作成します。同時に作成したオブジェクトリファレンスをネーミングサービスに登録します。
オブジェクトを作成する方法として以下の方法があります。

- OD_or_admコマンドを使用します。
- サーバアプリケーションでBOAの機能を使用してオブジェクトリファレンスを作成し、ネーミングサービスに登録します。

注意

POAを使用してサーバアプリケーションを作成する場合は、環境設定を行う必要があります。詳細については、“POA使用方法例”を参照してください。

■OD_or_admコマンドを使用する方法

OD_or_admコマンドは、オブジェクトリファレンス作成後、ネーミングサービスに登録します。
OD_or_admコマンドでの登録例と指定する情報を以下に示します。

例

Windows32/64

ネーミングサービスへの登録

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -L SJIS -n ODdemo::calculator
```

-c IDL:ODdemo/calculator:1.0

指定されたインタフェースリポジトリIDでオブジェクトリファレンスを登録します

-L SJIS

ネーミングサービスに登録するオブジェクトに対してコード系SJISを設定します。

-n ODdemo::calculator

ネーミングサービスに登録するオブジェクト名を指定します。

Solaris32/64 Linux32/64

ネーミングサービスへの登録

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -L EUC -n ODdemo::calculator
```

-c IDL:ODdemo/calculator:1.0

指定されたインタフェースリポジトリIDでオブジェクトリファレンスを登録します

-L EUC

ネーミングサービスに登録するオブジェクトに対してコード系EUCを設定します。

-n ODdemo::calculator

ネーミングサービスに登録するオブジェクト名を指定します。

■サーバアプリケーションで作成する方法

以下の流れで処理を行います。

プログラミングの例を以下に示します。

```
CORBA::ORB_ptr            orb;       // ORB_initで取得したObject
CORBA::BOA_ptr            boa;       // ORB::BOA__initで取得したObject
CORBA::Repository_ptr     intf_rep;  // インタフェースリポジトリのオブジェクトリファレンス
CORBA::InterfaceDef_ptr   intf;      // インタフェースリポジトリのInterfaceDef
FJ::ImplementationRep_ptr impl_rep;  // インプリメンテーションリポジトリのオブジェクトリファレンス
CORBA::ImplementationDef_ptr impl;   // インプリメンテーションリポジトリのImplementationDef
CORBA::Object_ptr           new_obj; // 作成したオブジェクトリファレンス
CORBA::ReferenceData        id;      // ReferenceData格納域
CORBA_Environment           *env = new CORBA::Environment();  // 例外情報

// ObjectDirectorの初期化（省略）
           :

// インタフェースリポジトリのオブジェクトリファレンスの獲得
intf_rep = orb->resolve_initial_references(
        CORBA_ORB_ObjectId_LightInterfaceRepository,
        *env );

// InterfaceDefオブジェクトのオブジェクトリファレンスの獲得
intf = intf_rep->lookup_id(
        "...",  // インタフェースリポジトリID
```

```
        &env );

// インプリメンテーションリポジトリのオブジェクトリファレンスの獲得
impl_rep = orb->resolve_initial_references(
        CORBA_ORB_ObjectId_ImplementationRepository,
        *env );

// ImplementationDefオブジェクトのオブジェクトリファレンスの獲得
impl = impl_rep->lookup_id(
        "...",  // インプリメンテーションリポジトリID
        *env );

// オブジェクトリファレンスの作成
new_obj = boa->create( id, intf, impl , *env );

CORBA::Object_ptr obj;                  // NamingServiceのオブジェクトリファレンス

// NamingServiceのオブジェクトリファレンス獲得
obj = orb->resolve_initial_references( CORBA_ORB_ObjectId_NameService, *env );

// NamingContextクラスへの変換
CosNaming::NamingContext_ptr cos_naming =
      CosNaming::NamingContext::_narrow( obj );

CosNaming::Name              name;              // CosNaming::Nameのインスタンス

// オブジェクト名
name[0]->id = (const CORBA::Char *)"ODdemo::calculator";
// オブジェクトのタイプ
name[0]->kind = (const CORBA::Char *)"";

// サーバアプリケーションのオブジェクトリファレンスを登録
cos_naming-> bind( name, obj, *env );
```

CORBA::BOA::create関数呼出し時に、インプリメンテーションリポジトリに定義されているデフォルトコード系でオブジェクトリファレンスを作成します。デフォルトコード系は、OD_impl_inst／OD_set_envコマンドで設定します。

4.5 クライアントアプリケーションのプログラミング

4.5.1 静的起動インタフェース

静的起動インタフェースを使用する場合の、クライアントアプリケーションの処理の流れを以下に示します。

4.5.1.1 初期化

CORBAの初期化メソッドCORBA::ORB_init()を呼び出し、初期化処理を行います。結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
main( int argc, char *argv[] )
{
    CORBA::ORB_ptr          orb;                    // ORBのオブジェクトリファレンス
    CORBA::Environment      env;                    // 例外情報
    int                     current_argc = argc;

    env = new CORBA::Environment;                   // 例外情報
    orb = CORBA::ORB_init(current_argc, argv, FJ_OM_ORBid, env);
```

4.5.1.2 ネーミングサービスのオブジェクトリファレンスの獲得

実行するオブジェクトをネーミングサービスから検索するために、ネーミングサービスのオブジェクトリファレンスが必要となります。CORBAインタフェースのオブジェクトリファレンスの取出しメソッドCORBA::ORB::resolve_initial_references()によりネーミングサービスのオブジェクトリファレンスを取り出します。このとき、CORBA_ORB_ObjectId_NameServiceをメソッドのパラメタとして指定します。

```
// NamingServiceのオブジェクトリファレンスを獲得
CORBA::Object_ptr
obj = orb->resolve_initial_references( CORBA_ORB_ObjectId_NameService, env );

// NamingContextクラスへの変換
CosNaming::NamingContext_ptr
cos_naming = CosNaming::NamingContext::_narrow( obj );
```

4.5.1.3 サーバアプリケーションのオブジェクトリファレンスの獲得

ネーミングサービスのメソッドCosNaming::NamingContext::resolve()により、これから実行したいサーバアプリケーションのオブジェクトリファレンスを取り出します。検索するオブジェクト名を本メソッドのパラメタとして指定します。

```
CosNaming::Name             name;  // CosNaming::Nameのインスタンス
CORBA::Object_ptr           obj;   // サーバアプリケーションのオブジェクトリファレンス

name.length(1);
// オブジェクト名
```

```
name[0]->id = (const CORBA::Char *)"ODdemo::calculator";
// オブジェクトのタイプ
name[0]->kind = (const CORBA::Char *)"";

// サーバアプリケーションのオブジェクトリファレンスを獲得
obj = cos_naming-> resolve( name, env );

// ODdemo::calculatorクラスへの変換
ODdemo::calculator_ptr ap = ODdemo::calculator::_narrow( obj );
```

4.5.1.4 メソッドの呼出し

サーバプログラムのメソッドを呼び出します。メソッド名は、IDLで指定したモジュール名、インタフェース名、およびオペレーション名を、2つのコロン(::)でつなげた形式で指定します。この例では、ODdemo、calculator、calculateがそれにあたります。なお、メソッド呼出し時に、ネーミングサービスで求めたサーバアプリケーションのオブジェクトリファレンスとサーバプログラムで例外が発生した場合に例外情報を獲得するためにCORBA::Environment構造体を指定します。
なお、アプリケーションで使用可能なCORBAのデータ型については、“4.6 データ型に対するマッピング”を参照してください。

```
ODdemo::calculator::result    res;   // 復帰値
CORBA::Long                   a, b;  // パラメタ

a = 10;
b = 5;

// メソッドの呼出し
res = ap->calculate( a, b, *env );
```

4.5.2 クライアントアプリケーションの例外処理

クライアントアプリケーションは、サーバアプリケーションの処理が正常終了したか異常終了したかを知ることができます。異常終了した場合は、システムの異常／サーバアプリケーションの異常かを知ることができ、前者をシステム例外、後者をユーザ例外と呼びます。システム例外クラスを以下に示します。例外の意味については、“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。

例外情報	例外クラス
BAD_CONTEXT	CORBA::StExcep::BAD_CONTEXT
BAD_INV_ORDER	CORBA::StExcep::BAD_INV_ORDER
BAD_OPERATION	CORBA::StExcep::BAD_OPERATION
BAD_PARAM	CORBA::StExcep::BAD_PARAM
BAD_QOS	CORBA::StExcep::BAD_QOS
BAD_TYPECODE	CORBA::StExcep::BAD_TYPECODE
CODESET_INCOMPATIBLE	CORBA::StExcep::CODESET_INCOMPATIBLE
COMM_FAILURE	CORBA::StExcep::COMM_FAILURE
CONTEXT	CORBA::StExcep::CONTEXT
DATA_CONVERSION	CORBA::StExcep::DATA_CONVERSION
FREE_MEM	CORBA::StExcep::FREE_MEM
IMP_LIMIT	CORBA::StExcep::IMP_LIMIT
INITIALIZE	CORBA::StExcep::INITIALIZE
INTERNAL	CORBA::StExcep::INTERNAL
INTF_REPOS	CORBA::StExcep::INTF_REPOS

例外情報	例外クラス
INV_FLAG	CORBA::StExcep::INV_FLAG
INV_IDENT	CORBA::StExcep::INV_IDENT
INV_OBJREF	CORBA::StExcep::INV_OBJREF
INV_POLICY	CORBA::StExcep::INV_POLICY
MARSHAL	CORBA::StExcep::MARSHAL
NO_IMPLEMENT	CORBA::StExcep::NO_IMPLEMENT
NO_MEMORY	CORBA::StExcep::NO_MEMORY
NO_PERMISSION	CORBA::StExcep::NO_PERMISSION
NO_RESOURCES	CORBA::StExcep::NO_RESOURCES
NO_RESPONSE	CORBA::StExcep::NO_RESPONSE
OBJ_ADAPTER	CORBA::StExcep::OBJ_ADAPTER
PERSIST_STORE	CORBA::StExcep::PERSIST_STORE
REBIND	CORBA::StExcep::REBIND
TIMEOUT	CORBA::StExcep::TIMEOUT
TRANSIENT	CORBA::StExcep::TRANSIENT
UNKNOWN	CORBA::StExcep::UNKNOWN
INVALID_TRANSACTION	CORBA::StExcep::INVALID_TRANSACTION
TRANSACTION_MODE	CORBA::StExcep::TRANSACTION_MODE
TRANSACTION_REQUIRED	CORBA::StExcep::TRANSACTION_REQUIRED
TRANSACTION_ROLLEDBACK	CORBA::StExcep::TRANSACTION_ROLLEDBACK
TRANSACTION_UNAVAILABLE	CORBA::StExcep::TRANSACTION_UNAVAILABLE

システム例外発生時は、CORBA::SystemException::minor()を使用してマイナーコードを獲得します。マイナーコードの値については、“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。

エラーは、以下の方法で判定します。

- try-catchを使用する方法
- メソッド呼出し時に指定したCORBA::Environmentクラスに格納された情報を使用する方法

通常、try-catchを使用してエラーを判定する必要があります。IDLcでスタブを作成する際にnoexオプションを指定した場合は、サーバアプリケーションのメソッド呼出し時のエラー判定に、CORBA::Environmentクラスに格納された情報を使用します。IDLcの詳細については、“リファレンスマニュアル(コマンド編)”を参照してください。

(1)tryーcatchを使用する場合

tryブロックで例外をハンドルするために、例外クラスに対応するcatchブロックを定義しておきます。例外が発生すると、例外クラスに対応するcatchブロックが実行されます。
システム例外は、親クラスとしてCORBA::SystemExceptionを継承し、その派生したクラスとして各詳細情報を定義したクラスCORBA::StExcep::UNKNOWN、CORBA::StExcep::BAD_PARAMなどがあります。
システム例外を受ける場合は、catch文でCORBA::SystemExceptionクラスを指定します。もし、システム例外／ユーザ例外で、エラー処理の区別が必要でなければ、上位クラスであるCORBA::Exceptionをcatch文で指定すれば、すべて一括して受けることができます。

```
// ユーザ例外の参照
CORBA::Environment    env;
CORBA::Long           l;
CORBA::ULong          minor;  // マイナーコード

try {
```

```
    obj->op(env);
} catch ( CORBA::SystemException &systemerr ) {
    minor = systemerr.minor(); // マイナーコードの獲得
    // エラー処理
} catch( ODdemo::calculator::ZEROPARAM &errcode ){
    // エラー処理
}

または

CORBA::Environment    env;
CORBA::Long           l;

try {
    obj->op(env);
} catch ( CORBA::Exception &e ) {
    // エラー処理
}
```

(2)CORBA::Environmentクラスによる判定方法

例外処理用のCORBA::Environmentクラスを使用して、例外を判定します。
CORBA::Environmentクラスを以下に示します。

```
class Environment
{
public:
    void exception(Exception *);
    Exception *exception() const;
    void clear();
}
```

メンバ関数のexception()に対して、エラーが発生したかどうかを問い合わせます。結果として復帰値にNULLが設定された場合は、正常終了です。異常終了の場合は、以下のどちらかの例外クラスへのポインタが設定されます。

- CORBA::SystemException:システム例外
- CORBA::UserException:ユーザ例外

システム例外

システム例外が発生したかどうかをSystemExceptionクラスの_narrowメソッドを呼び出して、その結果を判定します。結果が真の場合、システム例外と判断できます。また、どのエラーが発生したかを知るには、システム例外の詳細クラスの_narrowメソッドを呼び出して、その結果を判定します。結果が真の場合、そのエラー詳細クラスと判断できます。

ユーザ例外

ユーザ例外が発生したかどうかをUserExceptionクラスの_narrowメソッドを呼び出して、その結果を判定します。結果が真の場合、ユーザ例外と判断できます。その後、_valueメソッドを呼び出して、ユーザ例外の詳細を取り出します。

例外情報を獲得する例を以下に示します。

```
// 例外の参照
CORBA::Environment      *env;           // CORBA::Environmentクラス
CORBA::Exception        *exc;           // CORBA::Exceptionクラス
CORBA::SystemException  *systemerr;     // CORBA::SystemExceptionクラス
CORBA::UNKNOWN          *sys_unknown;   // 未知のエラー
CORBA::BAD_PARAM        *sys_bad_param; // パラメタエラー
           :
           :
CORBA::UserException    *usererr;       // CORBA::UserExceptionクラス
```

```
ODdemo::calculator::ZEROPARAM *errcode; // エラー詳細
CORBA::ULong            minor;          // マイナーコード

env = new CORBA::Environment;
result = ap->divide(2,  3, *env);       // メソッドの呼出し
if( exc = env->exception() ){
    if( systemerr = SystemException::_narrow(exc)) {    // システム例外
        minor = systemerr->minor();     // マイナーコードの獲得
        if( sys_unknown = CORBA::UNKNOWN::_narrow( systemerr) ){
            :                           // エラー処理
            :
        }
        if( sys_bad_param = CORBA::BAD_PARAM::_narrow( systemerr) ){
            :                           // エラー処理
            :
        }
    } else if( usererr = UserException::_narrow(exc)) { // ユーザ例外
        if( errcode = ODdemo::calculator::ZERODIVIDE::_narrow( usererr ) ){
            :                           // エラー処理
            :
        }
    }
}
```

4.6 データ型に対するマッピング

本節では、クライアント／サーバアプリケーションをC++言語で作成する場合に使用するデータ型について説明します。クライアント／サーバアプリケーション間の関数呼出しで使用するパラメタのデータ型については、“4.6.11 パラメタ受渡しで使用するデータ型”で説明します。

4.6.1 基本データ型

CORBAで定義されている基本データ型とC++言語でのデータ型の対応を以下に示します。

CORBAデータ型		C++言語での定義	備考
整数型	long	CORBA::Long	
	short	CORBA::Short	
	unsigned long	CORBA::ULong	
	unsigned short	CORBA::UShort	
	long long	CORBA::LongLong	
浮動小数点型	float	CORBA::Float	
	double	CORBA::Double	
	long double	CORBA::LongDouble	Windows32/64 Solaris32/64
文字型	char	CORBA::Char	
	wchar	CORBA::WChar	
オクテッド型	octet	CORBA::Octet	
ブーリアン型	boolean	CORBA::Boolean	
文字列型	string	CORBA::Char *	“4.6.2 文字列型”参照
	wstring	CORBA::WChar *	“4.6.3 ワイド文字列型”参照
列挙型	enum	enum	
any型	any	CORBA::Any	“4.6.4 any型”参照

CORBAデータ型		C++言語での定義	備考
オブジェクトリファレンス	Object	CORBA::Object_ptr	
タイプコード	TypeCode	CORBA::TypeCode_ptr	

4.6.2 文字列型

(1)IDLマッピング

IDL言語で文字列型stringを指定した場合、C++言語ではCORBA::Char *にマッピングされます。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface  stringtest{
        string op1(in string str1, out string str2, inout string str3);
    };
};
```

C++言語

```
CORBA::Char *
ODsample::stringtest::op1(
    const CORBA::Char *str1,       // inパラメタ
    CORBA::Char  *&str2,           // outパラメタ
    CORBA::Char  *&str3,           // inoutパラメタ
    CORBA::Environment  &env )     // 例外情報
```

(2)クライアントアプリケーションで扱うパラメタ

クライアントアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	CORBA::string_alloc()で領域を獲得します。	－
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。
out 復帰	－	(inoutパラメタと同じ)

注意

クライアントおよびスタブで獲得した領域は、不要になった時点でCORBA::string_free()で解放する必要があります。

クライアントアプリケーションでの処理例を以下に示します。

```
CORBA::Environment  env;
CORBA::Char         *str1, *str2, *str3, *ret;

// inパラメタ
str1 = CORBA::string_alloc(2);          // 領域獲得
strcpy( str1 ,"IN" );                   // パラメタ設定
```

```
// inoutパラメタ
str3 = CORBA::string_alloc(7);          // 領域獲得
strcpy( str3 ,"INOUT:1" );              // パラメタ設定

// サーバ呼出し
ret = obj->op1( str1, str2, str3, env );

// 領域解放
CORBA::string_free( ret );              // 復帰パラメタ
CORBA::string_free( str1 );             // inパラメタ
CORBA::string_free( str2 );             // outパラメタ
CORBA::string_free( str3 );             // inoutパラメタ
```

(3)サーバアプリケーションで扱うパラメタ

サーバアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
in	領域は、スケルトンで自動的に獲得/解放されます。	－
inout	領域は、スケルトンで自動的に獲得されます。	・ 渡されたパラメタより短い文字列を返す場合: 渡された領域に文字列を設定します。 ・ 渡されたパラメタより長い文字列を返す場合: 渡された領域をCORBA::string_free()で解放し、CORBA::string_alloc()で新しい領域を獲得します。 渡した領域は、スケルトンで自動的に解放されます。
out 復帰	－	領域は、CORBA::string_alloc()で獲得します。 この領域は、スケルトンで自動的に解放されます

サーバアプリケーションでの処理例を以下に示します。

```
CORBA::Char *
ODsample_stringtest_impl::op1(
    const CORBA::Char  *str1,           // inパラメタ
    CORBA::Char        *&str2,          // outパラメタ
    CORBA::Char        *&str3,          // inoutパラメタ
    CORBA::Environment &env )           // 例外情報
    throws( CORBA::Exception )
{
    // outパラメタ
    str2 = CORBA::string_alloc(3);      // 領域獲得
    strcpy( str2, "OUT" );              // パラメタ設定

    // inoutパラメタ
    CORBA::string_free( str3 );         // 領域解放
    str3 = CORBA::string_alloc(7);      // 領域再獲得
    strcpy( str3, "INOUT:2" );          // パラメタ設定

    // 復帰パラメタ
    str = CORBA::string_alloc(6);       // 領域獲得
    strcpy( str, "RETURN" );            // パラメタ設定
    return( str );
}
```

4.6.3 ワイド文字列型

(1)IDLマッピング

IDL言語でワイド文字列型wstringを指定した場合、C++言語ではCORBA::WChar *でデータを宣言します。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface  wstringtest{
        wstring op1(in wstring str1, out wstring str2, inout wstring str3);
    };
};
```

C++言語

```
CORBA::WChar *
ODsample::wstringtest::op1(
    const CORBA::WChar *str1,     // inパラメタ
    CORBA::WChar  *&str2,         // outパラメタ
    CORBA::WChar  *&str3,         // inoutパラメタ
    CORBA::Environment  &env )    // 例外情報
```

(2)クライアントアプリケーションで扱うパラメタ

クライアントアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	CORBA::wstring_alloc()で領域を獲得します。	－
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。
out 復帰	－	(inoutパラメタと同じ)

クライアントおよびスタブで獲得した領域は、不要になった時点でCORBA::wstring_free()で解放する必要があります。

クライアントアプリケーションでの処理例を以下に示します。

```
CORBA::Environment  env;
CORBA::WChar        *str1, *str2, *str3, *ret;

// inパラメタ
str1 = CORBA::wstring_alloc(2);      // 領域獲得
str1[0] = ...;                       // パラメタ設定：(4)参照
    :
// inoutパラメタ
str3 = CORBA_wstring_alloc(5);       // 領域獲得
str3[0] = ...;                       // パラメタ設定：(4)参照
    :
// サーバ呼出し
ret = obj->op1( str1, str2, str3, env );

// 領域解放
CORBA::wstring_free( ret );          // 復帰パラメタ
```

```
CORBA::wstring_free( str1 );        // inパラメタ
CORBA::wstring_free( str2 );        // outパラメタ
CORBA::wstring_free( str3 );        // inoutパラメタ
```

(3)サーバアプリケーションで扱うパラメタ

サーバアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
in	領域は、スケルトンで自動的に獲得/解放されます。	－
inout	領域は、スケルトンで自動的に獲得されます。	・ 渡されたパラメタより短い文字列を返す場合: 渡された領域に文字列を設定します。 ・ 渡されたパラメタより長い文字列を返す場合: 渡された領域をCORBA::wstring_free()で解放し、CORBA::wstring_alloc()で新しい領域を獲得します。 渡した領域は、スケルトンで自動的に解放されます。
out 復帰	－	CORBA::wstring_alloc()で獲得します。 この領域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
CORBA::WChar *
ODsample_wstringtest_impl::op1(
    const CORBA::WChar  *str1,          // inパラメタ
    CORBA::WChar        *&str2,         // outパラメタ
    CORBA::WChar        *&str3,         // inoutパラメタ
    CORBA::Environment  &env )          // 例外情報
    throws( CORBA::Exception )
{
    // outパラメタ
    str2 = CORBA::wstring_alloc(3);     // 領域獲得
    str2[0] = ...;                      // パラメタ設定：(4)参照
       :

    // inoutパラメタ
    CORBA::wstring_free( str3 );        // 領域解放
    str3 = CORBA::wstring_alloc(5);     // 領域再獲得
    str3[0] = ...;                      // パラメタ設定：(4)参照
        :

    // 復帰パラメタ
    str = CORBA::wstring_alloc(4);      // 領域獲得
    str[0] = ...;                       // パラメタ設定：(4)参照
        :
    return( str );
}
```

(4)文字列の設定方法

ワイド文字列型への文字列の設定方法を以下に示します。変数wstrは、CORBA::WChar*型です。

EUCまたはShiftJISを使用する場合

文字を1文字ずつ設定します(EUCの場合)。

```
wstr[0] = 0xc6fc;  //  "日"
wstr[1] = 0xcbdc;  //  "本"
wstr[2] = 0xb8ec;  //  "語"
wstr[3] = 0x0000;  //  終端子
```

Windows(R)でUNICODEを使用する場合

""で囲んだ文字列にL修飾子を付けると、UNICODEになります。

```
wstr = L"日本語";
```

Solaris/LinuxでUNICODEを使用する場合 Solaris32/64 Linux32/64

UNICODEの値を1文字ずつ直接数値で設定します。

```
wstr[0] = 0x65e5;  //  "日"
wstr[1] = 0x672c;  //  "本"
wstr[2] = 0x8a9e;  //  "語"
wstr[3] = 0x0000;  //  終端子
```

注意

- ワイド文字列型をコンソールなどに出力する場合、出力処理は各OSに依存するため、そのままでは正しい文字が出力できないことがあります。出力の方法については、各OSのマニュアルを参照してください。
- Solaris32/64 Linux32/64
  L"..."の形式の文字列を処理することはできません。

4.6.4 any型

(1)IDLマッピング

IDL言語でany型を指定した場合、C++言語では、データ型を識別するTypeCode(_tc)、データ域のアドレス(_value)、リリースフラグ(_release)をprivateデータに持つCORBA::Anyクラスにマッピングされます。CORBA::Anyクラスのインスタンスは、new(C++の演算子)で生成します。privateデータのアクセスは、CORBA::Anyクラスのメソッドを使用します。
インスタンス生成の記述形式を以下に示します。

```
CORBA::Any  *data = new CORBA::Any(
        CORBA::TypeCode_ptr  _tc,        // データ型識別TypeCode
        void                 *_value,    // データ域アドレス
        CORBA::Boolean       release );  // リリースフラグ
```

IDL言語

```
class CORBA {
    class Any {
        public:
            Any();                                          // デフォルトコンストラクタ
            Any( const Any & );                             // コピーコンストラクタ
            Any( TypeCode_ptr tc, void *value, Boolean release = CORBA_FALSE );
                                                            // コンストラクタ(型定義されていない値用)
            ~Any();                                         // デストラクタ

            Any     &operator=( const Any & );              // 代入演算子
            // 左シフト代入演算子
            void    operator<<=( Short );                   // Short型
```

```
void    operator<<=( UShort );            // UShort型
void    operator<<=( Long );              // Long型
void    operator<<=( ULong );             // ULong型
void    operator<<=( LongLong );          // LongLong型
void    operator<<=( Float );             // Float型
void    operator<<=( Double );            // Double型
void    operator<<=( LongDouble );        // LongDouble型  (Windows(R)、Solaris のみ)
void    operator<<=( const Any & );       // Any型
void    operator<<=( const Char * );      // Char * (String) 型
void    operator<<=( const WChar * );     // WChar * (WString) 型

// 右シフト代入演算子
Boolean operator>>=( Short & ) const;     // Short型
Boolean operator>>=( UShort & ) const;    // UShort型
Boolean operator>>=( Long & ) const;      // Long型
Boolean operator>>=( ULong & ) const;     // ULong型
Boolean operator>>=( LongLong & ) const;  // LongLong型
Boolean operator>>=( Float & ) const;     // Float型
Boolean operator>>=( Double & ) const;    // Double型
Boolean operator>>=( LongDouble & ) const; // LongDouble型  (Windows(R)、Solaris のみ)
Boolean operator>>=( Any & ) const;       // Any型
Boolean operator>>=( Char *& ) const;     // Char * (String) 型
Boolean operator>>=( WChar *& ) const;    // WChar * (WString) 型

// boolean, octet, char の設定に必要な特別な helper 型
struct  from_boolean {
    from_boolean( Boolean b ) : val(b) {}
    Boolean    val;
};
struct  from_octet {
    from_octet( Octet o ) : val(o) {}
    Octet      val;
};
struct  from_char {
    from_char( Char c ) : val(c) {}
    Char       val;
};
void    operator<<=( from_boolean );
void    operator<<=( from_octet );
void    operator<<=( from_char );

// boolean, octet, char の抽出に必要な特別な helper 型
struct  to_boolean {
    to_boolean( Boolean &b ) : ref(b) {}
    Boolean    &ref;
};
struct  to_octet {
    to_octet( Octet &o ) : ref(o) {}
    Octet       &ref;
};
struct  to_char {
    to_char( Char &c ) : ref(c) {}
    Char       &ref;
};
Boolean operator>>=( to_boolean ) const;
Boolean operator>>=( to_octet ) const;
Boolean operator>>=( to_char ) const;

void  replace( TypeCode_ptr tc, void *value,
               Boolean release = CORBA_FALSE );

TypeCode_ptr    type() const;
const  void     *value() const;
```

```
            private:
                TypeCode_ptr    _tc;        // データ型識別TypeCode
                void            *_value;    // データ域アドレス
                Boolean         _release;   // リリースフラグ
    };
};
```

以降では、以下のany型のIDL定義例をもとに、クライアント／サーバアプリケーションの記述例を示します。

IDL言語

```
module ODsample{
    struct sample1 {
        long    para1;
        string  para2;
    };
    struct sample2 {
        char    para1;
        float   para2;
    };
    struct sample3 {
        char    para1;
        double  para2;
    };
    interface  anytest{
        any    op1(in any any1, out any any2, inout any any3 );
    };
};
```

(2)クライアントアプリケーションで扱うパラメタ

クライアントアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	領域を動的に獲得する場合は、インスタンスをnewで生成します。	－
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。
out 復帰	－	(inoutパラメタと同じ)

クライアントおよびスタブで獲得した領域は、不要になった時点で、delete(C++演算子)で解放する必要があります。このとき、データ域(_value)を解放するかどうかをリリースフラグで指定します。リリースフラグは、CORBA::Anyクラスのインスタンス生成時に3番目のパラメタで指定します。

- CORBA_TRUE:delete発行時、_valueの領域も解放されます。
- CORBA_FALSE:delete発行時、_valueの領域は解放されません(デフォルト)。

なお、スタブで獲得されたパラメタ(inout,out,復帰)のリリースフラグは、CORBA_TRUEに設定されます。リリースフラグの詳細については、“4.6.12 any型、sequence型のリリースフラグ”を参照してください。

クライアントアプリケーションの処理例を以下に示します。

```
CORBA::Environment     env;
ODsample::anytest_ptr  obj;
```

```
ODsample::sample1      *smp1;
ODsample::sample2      *smp2;
ODsample::sample3      *smp3;
CORBA::Any             *any0, *any1, *any2, *any3;

// inパラメタ
ODsample::sample2 *smp2 = new ODsample::sample2; // データ域獲得(ODsample::sample2構造体)
smp2->para1 = 'z';                               // データ域(構造体)設定
smp2->para2 = 0.001;
// インスタンス生成/パラメタ設定
any1 = new CORBA::Any( _tc_ODsample_sample2, smp2, CORBA_TRUE );

// inoutパラメタ
ODsample::sample3 *smp3 = new ODsample::sample3; // データ域獲得(ODsample::sample3構造体)
smp3->para1 = 'x';                               // データ域(構造体)設定
smp3->para2 = 0.0001;
// インスタンス生成/パラメタ設定
any3 = new CORBA::Any();
any3->replace( _tc_ODsample_sample3, smp3, CORBA_TRUE);

// サーバ呼出し
any0 = obj->op1( *any1, any2, *any3, env );

// 領域解放
delete any0;                    // 復帰パラメタ
delete any1;                    // inパラメタ
delete any2;                    // outパラメタ
delete any3;                    // inoutパラメタ
```

(3)サーバアプリケーションで扱うパラメタ

サーバアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
in	領域は、スケルトンで自動的に獲得/解放されます。	－
inout	領域は、スケルトンで自動的に獲得されます。	replaceメソッドでデータを置き換えます。 この領域は、スケルトンで自動的に解放されます。
out 復帰値	－	newで獲得します。 この領域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
CORBA::Any*
ODsample_anytest_impl::op1(
    const CORBA::Any   &any1,
    CORBA::Any         *&any2,
    CORBA::Any         &any3,
    CORBA::Environment &env )
    throw( CORBA::Exception )
{
    // outパラメタ
    ODsample::sample1 *smp1 = new ODsample::sample1; // データ域獲得(ODsample::sample1構造体)
    smp1->para1 = 100;                               // データ域(構造体)設定
    CORBA::Char *str = CORBA::string_alloc(3);
    strcpy( str, "OUT" );
    smp1->para2 = str;
    // インスタンス生成/パラメタ設定
    any2 = new CORBA::Any( _tc_ODsample_sample1, smp1, CORBA_TRUE);
```

```
    // inoutパラメタ
    ODsample::sample2 *smp2 = new ODsample::sample2; // データ域獲得(ODsample::sample2構造体)
    smp2->para1 = 'x';                               // データ域(構造体)設定
    smp2->para2 = 0.01;
    // パラメタ設定
    any3->replace( _tc_ODsample_sample2, smp2, CORBA_TRUE );

    // 復帰値の処理
    ODsample::sample3 *smp3 = new ODsample::sample3; // データ域獲得(ODsample::sample3構造体)
    smp3->para1 = 'z';                               // データ域(構造体)設定
    smp3->para2 = 0.001;
    // インスタンス生成/パラメタ設定
    CORBA::Any *any = new CORBA::Any( _tc_ODsample_sample3, smp3, CORBA_TRUE );

    return( any );
}
```

(4)CORBA::Anyクラスのメソッド

CORBA::Anyクラスで定義されているメソッドの意味を以下に示します。

CORBA::Anyクラスのメンバ	意味
デフォルトコンストラクタ	インスタンス生成時、_tcを0、_valueを0で初期化します。リリースフラグには、CORBA_FALSEが設定されます。
コピーコンストラクタ	インスタンス生成時、データ域用の領域を獲得し、パラメタで指定された値を複写します。リリースフラグには、CORBA_TRUEが設定されます。 なお、メモリ不足によりデータ域用の領域獲得に失敗した場合、_valueはNULLに設定されます。
コンストラクタ (型定義されていない値用)	インスタンス生成時、パラメタで指定された値を_tcと_valueに設定します。リリースフラグは、指定された値が設定されます。リリースフラグの指定を省略した場合は、CORBA_FALSEが設定されます。
デストラクタ	リリースフラグがCORBA_TRUEの場合、_tcと_valueの値に対する領域を解放します。
代入演算子	リリースフラグがCORBA_TRUEの場合、現在設定されている_tcと_valueの値に対する領域を解放します。その後、データ域用の領域を獲得し、パラメタで指定された値を設定します。リリースフラグには、CORBA_TRUEが設定されます。 なお、メモリ不足によりデータ域用の領域獲得に失敗した場合、_valueにNULLが設定されます。
左シフト代入演算子	リリースフラグがCORBA_TRUEの場合、現在設定されている_tcと_valueの値に対する領域を解放します。その後、データ域用の領域を獲得し、パラメタで指定された値を設定します。リリースフラグには、CORBA_TRUEが設定されます。 なお、メモリ不足によりデータ域用の領域獲得に失敗した場合、_valueにNULLが設定されます。
右シフト代入演算子	_valueに設定されている値を取り出します。
helper関数	boolean、char、octetの場合、unsigned charとなるため、これらの型を識別します。
replace関数	リリースフラグがCORBA_TRUEの場合は、現在設定されている_valueの値に対する領域を解放します。その後、パラメタで指定された値を_tcと_valueに設定します(新たに領域は獲得しません)。リリースフラグは、指定された値が設定されます。リリースフラグの指定を省略した場合は、CORBA_FALSEが設定されます。
type関数	_tcのTypeCodeオブジェクトのオブジェクトリファレンスを返します。
value関数	_valueに設定されているデータ域へのポインタをvoid*型で返します。

デフォルトコンストラクタ

デフォルトコンストラクタは、_tcを0(TypeCode::_nil())、_valueを0で初期化し、また、リリースフラグをCORBA_TRUEに設定します。
デフォルトコンストラクタの使用例と処理内容を以下に示します。

```
// 使用例
CORBA::Any a;                        // デフォルトコンストラクタで初期化

または

CORBA::Any *b = new CORBA::Any;      // デフォルトコンストラクタで初期化

// デフォルトコンストラクタの処理内容
Any::Any()
{
      _tc = TypeCode::_nil();
      _value = 0;
      _release = CORBA_TRUE;
};
```

コピーコンストラクタ

コピーコンストラクタは、データ域用の領域を獲得し、パラメタで指定された値を複写します。リリースフラグにはCORBA_TRUEが設定されます。
コピーコンストラクタの使用例と処理内容を以下に示します。

```
// 使用例
CORBA::Any a;                // デフォルトコンストラクタで初期化

CORBA::Long x = 3;
a <<= x;                     // 左シフト代入演算子。_tcに_tc_longを、
                             // _valueにCORBA::Longの領域を獲得し値3を設定、
                             // リリースフラグをCORBA_TRUEに設定

CORBA::Any *b = new CORBA::Any(a);
        // コピーコンストラクタにより、aのprivateデータ(_tc,_value)のデータを複写し、
        // また、リリースフラグをCORBA_TRUEに設定

if ( b->value() == NULL ) {  // 領域の獲得の成否を判定
   ...                       // 獲得に失敗している場合、エラー処理
}

CORBA::Any c(*b);            // コピーコンストラクタ
if ( c.value() == NULL ) {   // 領域の獲得の成否を判定
    ...                      // 獲得に失敗している場合、エラー処理
}

// コピーコンストラクタの処理内容
Any::Any( const Any &a )
{
    _tc = TypeCode::_duplicate(a.type());
    _value = ...     // 領域を獲得し、aの_valueデータを複写する処理
    _release = CORBA_TRUE;
};
```

コンストラクタ

このコンストラクタは、パラメタで指定したTypeCodeオブジェクトリファレンスtcの値をコピーし、valueの指すポインタを_valueに代入します。パラメタのreleaseがCORBA_TRUEであれば、デストラクタ起動時に_valueの指す領域は解放されます。
release=CORBA_FALSEの場合は、アプリケーションが_valueでポイントされる領域を管理しなければなりません。

```
// 使用例
Long l = 1;
CORBA::Any *a = new CORBA::Any(tc_long, &l, CORBA_TRUE);
                        // 以下のコンストラクタが呼ばれます。

// コンストラクタの処理内容
Any::Any( TypeCode_ptr tc, void *value, Boolean release = CORBA_FALSE)
{
    _tc = tc;
    _value = value;
    _release = release;
}
```

デストラクタ

デストラクタの使用例と処理内容を以下に示します。

```
// 使用例
void X()
{
    CORBA::Any *a = new CORBA::Any();

    CORBA::Long x = 3;
    (*a) <<= x;  // 左シフト代入演算子

    delete a;    // デストラクタ
}

// デストラクタの処理内容
Any::~Any()
{
    if( _release && _value )
        delete _value;  // リリースフラグが設定されている場合、削除
    CORBA::release( _tc );
}
```

代入演算子

代入演算子の使用例と処理内容を以下に示します。

```
// 使用例
CORBA::Any a1;              // デフォルトコンストラクタで初期化
CORBA::Any a2;              // デフォルトコンストラクタで初期化

CORBA::Long x = 10;
a1 <<= x;                   // 左シフト代入演算子で、_tcに_tc_longを、
                            // _valueにCORBA::Longの領域を獲得し値10を、
                            // リリースフラグにCORBA_TRUEを設定
a2 = a1;                    // 代入演算子で、a2の_valueにCORBA::Longの領域を獲得し、
                            // a1のデータ(_tc,_value)の値をa2に複写し、
                            // リリースフラグにCORBA_TRUEを設定
if ( a2.value() == NULL ) {  // 領域の獲得の成否を判定
    ...                     // 獲得に失敗している場合、エラー処理
}

// 代入演算子の処理内容
Any &
Any::operator=( const Any &r )
{
    if( _release && _value )
        delete value;       // リリースフラグがCORBA_TRUEなら、既存データを削除
```

```
    CORBA::release( _tc );

    _tc = TypeCode::_duplicate(a.type());
    _value = ...              // 領域を獲得し、rの_valueの内容を複写する処理
    _release = CORBA_TRUE;
    return this;
}
```

左シフト代入演算子

左シフト代入演算子の使用例と処理内容を以下に示します。

```
// 使用例
CORBA::Long value = 42;
Any a;
a <<= value;                // _tcに_tc_longを、_valueにCORBA::Longの領域を獲得し値42を、
                            // リリースフラグにCORBA_TRUEを設定
if ( a.value() == NULL ) {  // 領域の獲得の成否を判定
    ...                     // 獲得に失敗している場合、エラー処理
}

// 左シフト代入演算子の処理内容
void CORBA::Any::operator<<=( Long v )
{
    if( _release && _value )
        delete value;       // リリースフラグがCORBA_TRUEなら、既存データを削除
    CORBA::release( _tc );

    _tc = .... ( _tc_longを設定する処理)
    _value = new Long(v);   // _valueにvの値を設定
    _release = CORBA_TRUE;
}
```

右シフト代入演算子

右シフト代入演算子の使用例と処理内容を以下に示します。

```
// 使用例
CORBA::Long value;
Any a;
a <<= CORBA::Long(42);          // aの_tcに_tc_longを_valueに値42を代入
                                // リリースフラグはCORBA_TRUEを設定
if(a >>= value ) {
                                // aの_tcが_tc_longを指している場合は、CORBA_TRUEを返し、
                                // valueに_valueの値を代入。
}

// 右シフト代入演算子の処理内容
CORBA::Boolean
CORBA::Any::operator>>=( Long &r )const
{
    if( _tc->kind != tk_long )  // _tcがtc_longを指しているかチェック
        return CORBA_FALSE
    r = ...                     // _valueの値を代入する処理
    return CORBA_TRUE;
}
```

文字列と配列に関して取り出された値を使用するときは、文字列／配列のサイズ指定を超えてはいけないため、アプリケーションはAny型のTypeCodeを確認する必要があります。

helper関数

boolean、char、octetがすべてC++でunsigned charにマッピングされるので、以下のように、直接Any型変数にboolean、octet、char型の変数を代入／取出しを行うと、コンパイルエラーとなります。

```
Octet oct = 040;
Any any;
any <<= oct;                    // この行はコンパイルが通らない
```

boolean、char、octetを区別する方法として、helper関数およびoperator<<=とoperator>>=が提供されます。
helper関数の使用例を以下に示します。

```
CORBA::Boolean b = CORBA_TRUE;
Any any;
any <<= CORBA::Any::from_boolean(b);  // _tcに_tc_booleanを、_valueにbの値を代入
    ...
if( any >>= CORBA::Any::to_boolean(b)){
    // anyの_valueはCORBA::Boolean型
    // bにanyの_valueの値(ここではCORBA_TRUE)を代入
}
```

replace関数

replace関数は、パラメタで指定したTypeCodeオブジェクトリファレンスtcと値valueを_tcと_valueに代入します。パラメタのreleaseがCORBA_TRUEであれば、デストラクタ起動時に、_valueの指す領域を解放します。release=CORBA_FALSEの場合は、アプリケーションが_valueでポイントされる領域を解放する必要があります。

```
// 使用例
Long l = 1;
CORBA::Any *a = new CORBA::Any();

a->replace(_tc_long, &l, CORBA_FALSE);    // 以下の関数が呼ばれます。

// replace関数の処理内容
Any::Any::replace( TypeCode_ptr tc, void *value, Boolean release = CORBA_FALSE)
{
    if( _release && _value )
        delete value;  // リリースフラグがCORBA_TRUEなら、既存データを削除
    CORBA::release( _tc );

    _tc = tc;
    _value = value;
    _release = release;
}
```

type関数

type関数は、Anyに関連づけられたCORBA::TypeCodeオブジェクトのオブジェクトリファレンスCORBA::TypeCode_ptrを返します。

```
CORBA::Any *a = new CORBA::Any();
CORBA::Long x;
(*a) <<= x;
    ...
CORBA::TypeCode_ptr tc = a->type(); // _tcを取り出します。
if( tc->kind() == tk_long ){
      ...               // このプログラムでは、等しいためここが処理されます。
}
```

value関数

value関数は、Any型変数に設定されている_valueをvoid*型で返します。_valueに値が設定されていなければ、NULLポインタを返します。

```
CORBA::Any any;
CORBA::Long x;
x = 3;
any <<= x;      // 左シフト代入演算子で_tcに_tc_longを_valueに値3を代入

CORBA::Long *p = (CORBA::Long *)any.value();
                // _valueを取り出します。void*型なのでCORBA::Long *型にキャスト

count << "(*p)=" << *p << endl;
```

4.6.5 シーケンス型

(1)IDLマッピング

IDL言語でシーケンス型sequenceを指定した場合、C++言語では、配列の最大個数(_maximum)、配列の使用個数(_length)、データ域のアドレス(_buffer)、リリースフラグ(_release)をprivateデータに持つクラス(シーケンスクラス)にマッピングされます。また、データ域(_buffer)の獲得関数(関数名:“モジュール名::インタフェース名::シーケンス名::allocbuf”。以降、XX::allocbuf関数と呼ぶ)が生成されます。
シーケンスクラスのインスタンスは、new(C++の演算子)で生成します。パラメタには、配列の最大個数、使用個数、データ域のアドレス(XX::allocbuf関数で獲得)、リリースフラグを指定します。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface  seqtest{
        typedef sequence<long>  sampleseq;
        sampleseq op1(in sampleseq seq1, out sampleseq seq2,
                      inout sampleseq seq3 );
    };
};
```

C++言語

```
class sampleseq
{
    public:
       sampleseq();                                    // デフォルトコンストラクタ
       sampleseq( CORBA::ULong max);                   // maximumコンストラクタ
       sampleseq( CORBA::ULong max,
                  CORBA::ULong length,
                  CORBA::Long *data,
                  CORBA::Boolean release = CORBA_FALSE );
       // T *dataコンストラクタ
       sampleseq( const sampleseq &s );                // コピーコンストラクタ
       ~sampleseq();                                   // デストラクタ

       static CORBA::Long *allocbuf( CORBA::ULong );   // allocbuf
       static void freebuf( CORBA::Long* );            // freebuf

       sampleseq &operator=( const sampleseq &s );     // 代入演算子

       CORBA::ULong maximum() const;                   // maximumアクセス関数

       void length( CORBA::ULong );                    // lengthアクセス関数
```

```
        CORBA::ULong length() const;                    // lengthアクセス関数

        CORBA::Long &operator[]( CORBA::ULong index ); // _bufferのindex番目の要素を取得
        const CORBA::Long &operator[]( CORBA::ULong index )
                                                const;   // _bufferのindex番目の要素を取得

    private:
        CORBA::ULong _maximum;          // 配列の最大個数
        CORBA::ULong _length;           // 配列の個数
        CORBA::Long *_buffer;           // 配列の値
        CORBA::Boolean _release;        // リリースフラグ
};

// 左シフト代入演算子
void operator<<=( CORBA::Any&, const ODsample::seqtest::sampleseq& );
void operator<<=( CORBA::Any&, ODsample::seqtest::sampleseq* );
void operator<<=( CORBA::Any_var&, const ODsample::seqtest::sampleseq& );

// 右シフト代入演算子
CORBA::Boolean operator>>=( const CORBA::Any&, ODsample::seqtest::sampleseq*& );
CORBA::Boolean operator>>=( const CORBA::Any_var&, ODsample::seqtest::sampleseq*& );
```

(2)クライアントアプリケーションで扱うパラメタ

クライアントアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	領域を動的に獲得する場合、シーケンスクラスのインスタンスは、new演算子を使用し、データ域はXX::allocbuf関数を使用します。	ー
inout	シーケンスクラスのインスタンスを動的に獲得する場合は、new演算子を使用します。データ域を獲得する場合は、XX::allocbuf関数を使用します。 リリースフラグには、CORBA_TRUEを設定してください。 データ域は、スタブで自動的に解放されます。	領域は、スタブで自動的に獲得されます(構造体メンバも設定されます)。このとき、獲得した領域には、リリースフラグにCORBA_TRUEが設定されます。
out 復帰	ー	(inoutパラメタと同じ)

注意

クライアントおよびスタブで獲得した領域は、不要になった時点でdelete(C++演算子)で解放する必要があります。このとき、データ域(_buffer)を解放するかどうかをリリースフラグで指定します。リリースフラグは、シーケンスクラスのインスタンス生成時に4番目のパラメタで指定します。

- CORBA_TRUE:delete発行時、_bufferの領域も解放されます。
- CORBA_FALSE:delete発行時、_bufferの領域は解放されません(デフォルト)。

なお、スタブで獲得されたパラメタ(inout、outm復帰)のリリースフラグは、CORBA_TRUEに設定されます。リリースフラグの詳細については、“4.6.12 any型、sequence型のリリースフラグ”を参照してください。

クライアントアプリケーションの処理例を以下に示します。

```
ODsample::sampleseq *seq = new ODsample::sampleseq(
    CORBA::ULong maximum,
    CORBA::ULong minimum,
    CORBA::Long *data,
    CORBA::Boolean release = CORBA_FALSE );

CORBA::Environment     env;
ODsample::seqtest_ptr  obj;
ODsample::sampleseq *seq0, *seq1, *seq2, *seq3;

// inパラメタ
CORBA::Long *p = ODsample::seqtest::sampleseq::allocbuf(3); // データ域獲得
for( int i = 0; i < 3; i++ )                                // データ域設定
    p[i] = i*10;
// インスタンス生成/パラメタ設定
seq1 = new ODsample::seqtest::sampleseq( 3, 3, p, CORBA_TRUE );

// inoutパラメタ
CORBA::Long *q = ODsample::seqtest::sampleseq::allocbuf(5); // データ域獲得
for( i = 0; i < 5; i++ )                                    // データ域設定
    q[i] = i*100;
// インスタンス生成/パラメタ設定
seq3 = new ODsample::seqtest::sampleseq( 5, 5, q, CORBA_TRUE );

// サーバ呼出し
seq0 = obj->op1( obj, *seq1, seq2, *seq3, env );

// 領域解放
delete seq0;                    // 復帰パラメタ
delete seq1;                    // inパラメタ
delete seq2;                    // outパラメタ
delete seq3;                    // inoutパラメタ
```

(3)サーバアプリケーションで扱うパラメタ

サーバアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
in	領域は、スケルトンで自動的に獲得/解放されます。	－
inout	領域は、スケルトンで自動的に獲得されます。このとき、リリースフラグにはCORBA_TRUEが設定されます。	inoutデータの領域を更新する場合は、代入演算子でデータを複写します。 データ域は、スケルトンで自動的に解放されます。
out 復帰値	－	シーケンスクラスのインスタンスをnew演算子で、データ域をXX::allocbuf関数で獲得します。 リリースフラグにCORBA_TRUEを設定する場合、この領域はスケルトンで自動的に解放されます。 リリースフラグにCORBA_FALSEを設定する、またはリリースフラグを設定しない場合、スケルトンではデータ域を解放しません。この場合、不要になった時点で領域の解放処理を実施してください。

サーバアプリケーションでの処理例を以下に示します。

```
ODsample::seqtest::sampleseq*
ODsample_seqtest_impl::op1(
    const ODsample::seqtest::sampleseq  &seq1,
    ODsample::seqtest::sampleseq        *&seq2,
```

```
    ODsample::seqtest::sampleseq        &seq3,
    CORBA::Environment                  &env )
    throw( CORBA::Exception )
{
    // outパラメタ
    CORBA::Long *p = ODsample::seqtest::sampleseq::allocbuf(2); // データ域獲得
    for( int i = 0; i < 2; i++ )                                // データ域設定
        p[i] = i*1000;
    // インスタンス生成/パラメタ設定
    seq2 = new ODsample::seqtest::sampleseq( 2, 2, p, CORBA_TRUE );

    // inoutパラメタ
    CORBA::Long *q = ODsample::seqtest::sampleseq::allocbuf(3); // データ域獲得
    for( i = 0; i < 3; i++ )                                    // データ域設定
        q[i] = i;
    // 代入演算子でパラメタ設定
    seq3 = ODsample::seqtest::sampleseq( 3, 3, q, CORBA_TRUE );

    // 復帰パラメタ
    CORBA::Long *r = ODsample::seqtest::sampleseq::allocbuf(1);
    r[0] = 0;                                                   // データ域獲得/設定
    // インスタンス生成/パラメタ設定
    ODsample::seqtest::sampleseq_ptr seq =
        new ODsample::seqtest::sampleseq( 1, 1, r, CORBA_TRUE );

    return seq;
}
```

(4)シーケンスクラスのメソッド

シーケンスクラスで定義されているメソッドとその意味を以下に示します。

メソッド	意味
デフォルトコンストラクタ	インスタンス生成時、シーケンス長_lengthを0で初期化します。また、サイズ指定なしのシーケンスの場合、最大長_maximumも0で初期化し、リリースフラグはCORBA_FALSEに設定します。サイズ指定ありのシーケンスの場合、最大長_maximumはIDLの定義値で初期化され、リリースフラグはCORBA_TRUEに設定します。 サイズ指定ありのシーケンスの場合は、_bufferの領域も獲得します。メモリ不足により領域の獲得に失敗した場合、_bufferはNULLに設定されます。
コピーコンストラクタ	インスタンス生成時、_bufferの領域を獲得し、パラメタで指定された値を_length、_maximum、_bufferに設定します。リリースフラグは、CORBA_TRUEに設定します。 なお、メモリ不足により領域の獲得に失敗した場合、サイズ指定なしのシーケンスでは_lengthと_maximumが0に設定され、_bufferがNULLに設定されます。サイズ指定ありのシーケンスでは_lengthが0に設定され、_bufferがNULLに設定されます。
maximumコンストラクタ	サイズ指定なしのシーケンスの場合に提供されます。インスタンス生成時、パラメタで指定された値を_maximumに設定します。また、_bufferの領域を獲得します。リリースフラグは、CORBA_TRUEに設定します。 なお、メモリ不足により領域の獲得に失敗した場合は、_maximumが0に設定され、_bufferがNULLに設定されます。
T*dataコンストラクタ	インスタンス生成時、サイズ指定あり/なしの両方で、パラメタで指定したシーケンス長、最大長、バッファポインタおよびリリースフラグを_length、_maximum、_buffer、_releaseに設定します。リリースフラグの指定を省略した場合は、CORBA_FALSEを設定します。
デストラクタ	リリースフラグがCORBA_TRUEの場合は、freebuf関数で_bufferの値に対する領域を解放します。
代入演算子	リリースフラグがCORBA_TRUEの場合は、freebuf関数で_bufferの値に対する領域を解放します。データ域の領域を獲得し、代入元のデータを複写します。リリースフラグは、

メソッド	意味
	CORBA_TRUEに設定します。 なお、メモリ不足により領域の獲得に失敗した場合、サイズ指定なしのシーケンスでは_lengthと_maximumが0に設定され、_bufferにNULLが設定されます。サイズ指定ありのシーケンスでは_lengthが0に設定され、_bufferにNULLが設定されます。
添字演算子	与えられたインデックスに対応する_bufferの内容を返します。
maximumアクセス関数	サイズ指定のないシーケンスの場合は、現在使用可能なバッファの総数を返します。サイズが指定されたシーケンスの場合は、IDLで与えられたシーケンスの大きさを返します。
lengthアクセス関数	length(ULong)関数は、パラメタで指定した値をシーケンス長に設定します。この際、必要に応じて_bufferの領域を獲得します。メモリ不足により領域の獲得に失敗した場合、シーケンスの値(シーケンス長、_buffer)は変更されません。 length()は、シーケンス長を返します。
allocbuf関数	T*dataコンストラクタに渡すことができるシーケンスの要素の配列を割り当てます。
freebuf関数	allocbuf関数によって割り当てられた配列を解放します。
左シフト代入演算子	Any／Any_varへのシーケンスを代入します。Any変数のリリースフラグには、CORBA_TRUEを設定します。
右シフト代入演算子	Any／Any_varからシーケンスへのポインタを取り出します。

デフォルトコンストラクタ

サイズ指定あり／なしの両方で、デフォルトコンストラクタはシーケンス長を0に初期化します。サイズ指定ありでは最大長はメンバイニシャライザで初期化されるので、アプリケーションは変更できません。サイズ指定なしのシーケンスでは、デフォルトコンストラクタで最大長を0に初期化します。サイズ指定ありのシーケンスではリリースフラグはCORBA_TRUEに設定され、サイズ指定なしのシーケンスではリリースフラグはCORBA_FALSEに設定されます。また、サイズ指定ありのシーケンスでは、_bufferの領域が獲得されます。なお、メモリ不足により領域の獲得に失敗した場合、_bufferにNULLが設定されます。このため、メモリ不足が発生する可能性がある環境でサイズ指定ありのシーケンスを使用する場合は、デフォルトコンストラクタ以外のコンストラクタを使用することを推奨します。デフォルトコンストラクタの使用例と処理内容を以下に示します。

```
// IDL
typedef sequence< long > LongSeq1;      // サイズ指定なしの場合
typedef sequence< long, 10 > LongSeq2; // サイズ指定あり(この例では10)の場合

//使用例
LongSeq1 *seq1 = new LongSeq1();        // 以下のデフォルトコンストラクタLongSeq1()が呼ばれ
                                        // seq1 が初期化されます。
LongSeq2 *seq2 = new LongSeq2();        // 以下のデフォルトコンストラクタLongSeq2()が呼ばれ
                                        // seq2 が初期化されます。

// デフォルトコンストラクタ (サイズ指定なし)
LongSeq1::LongSeq1()
{
    _length = 0;
    _maximum = 0;
    _release = CORBA_FALSE;
}
// デフォルトコンストラクタ(サイズ 指定あり)
LongSeq2::LongSeq2() : _maximum(10)         // IDLで指定したサイズ
{
    _length = 0;
    _buffer = LongSeq2::allocbuf(10);
    _release = CORBA_TRUE;
}
```

コピーコンストラクタ

コピーコンストラクタは、パラメタで指定されたシーケンスと同じ最大長と長さを持つ新しいシーケンスを生成し、パラメタで指定されたシーケンスの要素データをコピーします。コピーコンストラクタでは、リリースフラグはCORBA_TRUEに設定されます。
なお、コピーコンストラクタでは_bufferの領域が獲得されます。メモリ不足により領域の獲得に失敗した場合、サイズ指定なしのシーケンスでは_lengthと_maximumが0に設定され、_bufferがNULLに設定されます。サイズ指定ありのシーケンスでは_lengthが0に設定され、_bufferがNULLに設定されます。
コピーコンストラクタの使用例と処理内容を以下に示します。

```
// 使用例
long data[] = {1, 2, 3, 4, 5};
LongSeq s1(10, 5, data);
LongSeq s2(s1)       // 以下のコピーコンストラクタが呼ばれ、s1のデータがs2にコピーされます
if ( s2.length() == 0 ) {   // 領域の獲得の成否を判定（_lengthが0の場合、失敗）
    ...                     // 獲得に失敗している場合、エラー処理
}

// コピーコンストラクタ
LongSeq::LongSeq( const LongSeq &s )
{
    _length = s.length();
    _maximum = s.maximum();
    _buffer = allocbuf( _maximum );
    for( int i = 0; i < _length; i++ )
       // s の内容から *( _buffer + i ) へのコピー
    _release = CORBA_TRUE;
}
```

maximumコンストラクタ

サイズ指定なしのシーケンスでは、最大長として指定された個数分の_bufferの領域を獲得するコンストラクタが提供されます。
maximumコンストラクタは、リリースフラグにCORBA_TRUEを設定します。なお、メモリ不足により_bufferの領域獲得に失敗した場合、_maximumに0が設定され、_bufferにNULLが設定されます。
maximumコンストラクタの使用例と処理内容を以下に示します。

```
// IDL
typedef sequence< long > LongSeq;

// 使用例
LongSeq seq(10);     // 以下のmaximumコンストラクタが呼ばれ、seqの最大長が10に設定されます
if ( seq.maximum() == 0 ) {   // 領域の獲得の成否を判定（_maximumが0の場合、失敗）
    ...                       // 獲得に失敗している場合、エラー処理
}

// maximumコンストラクタ
LongSeq::LongSeq(ULong max)
{
    _maximum = max;
    _buffer = LongSeq::allocbuf(max);
    release = CORBA_TRUE;
}
```

T*dataコンストラクタ

T*dataコンストラクタは、サイズ指定あり／なしの両方で、パラメタで指定した値をprivateデータに設定します。サイズ指定のないシーケンスに対しては最大長を設定することも可能です。
このコンストラクタは、releaseパラメタによって_bufferの扱いが異なります。パラメタがCORBA_FALSEである場合、_bufferの指す領域はアプリケーションが管理する必要があります。一方、CORBA_TRUEである場合、_bufferの指す領域はシーケンスクラスの変数が解放されるときに解放されます。
T*dataコンストラクタの使用例と処理内容を以下に示します。

```
// 使用例
long data[] = {1, 2, 3, 4, 5};
LongSeq seq = new LongSeq(10, 5, data); // 以下のT *dataコンストラクタが呼ばれ、seqが初期化されます

// T *dataコンストラクタ
LongSeq::LongSeq( ULong max, ULong length, Long *value,
                              Boolean release = CORBA_FALSE )
{
    _length = length;
    _maximum = max;
    _buffer = value;
    _release = release;
}
```

T *dataコンストラクタで指定するvalueの値がString型の場合はString_var*を渡し、オブジェクトリファレンス型の場合はObject_var*を渡します。

デストラクタ

```
ODsample::seqtest::sampleseq::~sampleseq()
{
    if( _release == CORBA_TRUE )
        ODsample::seqtest::sampleseq::freebuf( _buffer );
}
```

代入演算子

代入演算子は、リリースフラグがCORBA_TRUEに設定されている場合、_bufferの指す領域を解放します。リリースフラグがCORBA_FALSEに設定されている場合、解放しません。その後、_bufferの領域を新規に獲得し、パラメタで指定されたシーケンスの要素データをコピーします。なお、メモリ不足により領域の獲得に失敗した場合、サイズ指定なしのシーケンスでは_lengthと_maximumが0に設定され、_bufferがNULLに設定されます。サイズ指定ありのシーケンスでは、_lengthに0が設定され、_bufferにNULLが設定されます。
代入演算子では、リリースフラグはCORBA_TRUEに設定されます。
代入演算子の処理内容を以下に示します。

```
// 使用例
long data[] = {1, 2, 3, 4, 5};
LongSeq s1(10, 5, data);
LongSeq s2;

LongSeq s2 = s1;    // 以下の代入演算子が呼ばれ、s1のデータがs2にコピーされます
if ( s2.length() == 0 ) {   // 領域の獲得の成否を判定（_lengthが0の場合、失敗）
    ...                     // 獲得に失敗している場合、エラー処理
}

// 代入演算子
LongSeq &
LongSeqV1::operator=( const LongSeqV1 &s )
{
    if( _release )
        LongSeq::freebuf( _buffer );
    _length = s.length();
    _maximum = s.maximum();
    _buffer = LongSeq::allocbuf( _maximum );
    for( int i = 0; i < _length; i++ )
      //  *( _buffer + i )  へ s[i] をコピー
    _release =CORBA_TRUE;
    return *this;
}
```

添字演算子

添字演算子は、与えられたインデックスに対応する_bufferの内容を返します。
添字演算子の使用例を以下に示します。

```
// 使用例
Long l;
long data[] = {1, 2, 3, 4, 5};
LongSeq seq(10, 5, data,CORBA_TRUE);

l = seq[3];    // lの値は4
```

シーケンスの要素がString型の場合はString_varが返され、オブジェクトリファレンス型の場合はObject_var型が返されます。

maximumアクセス関数

サイズ指定のないシーケンスに対するmaximum()アクセス関数は、現在使用可能なバッファの総数を返します。アプリケーションは、サイズ指定のないシーケンスにどのくらいのアイテムが設定できるかを知ることができます。サイズが指定されたシーケンスに対するmaximum()は、そのIDLで与えられたシーケンスの大きさを返します。
maximumアクセス関数の使用例を以下に示します。

```
// IDL
typedef sequence< long > LongSeq1;
typedef sequence< long, 10 > LongSeq2;

// 使用例
Long l;
LongSeq1 s1(5);
LongSeq2 s2;

l = s1.maximum();            // l の値は5
l = s2.maximum();            // l の値は10
```

lengthアクセス関数

length(ULong)関数は、シーケンス長にパラメタで指定した値を設定します。また、_bufferにlength分の領域を獲得します。メモリ不足により領域の獲得に失敗した場合、シーケンスの値(シーケンス長、_buffer)は変更されません。
length()は、シーケンス長を返します。
lengthアクセス関数の使用例を以下に示します。

```
// 使用例
Long l;
long data[] = {1, 2, 3, 4, 5};
LongSeq s1 = new LongSeq(10, 5, data, CORBA_FALSE);

l = s1->length();           // lの値は5
s1->length(6);              // シーケンス長を6 に設定
if ( s1->length() == l ) {  // 領域の獲得の成否を判定 (_lengthが変更されていない場合、失敗)
    ...                     // 獲得に失敗している場合、エラー処理
}
```

allocbuf関数とfreebuf関数

allocbuf関数は、T*dataコンストラクタに渡すことができるシーケンスの要素の配列を割り当てます。配列の長さは、パラメタにより与えられます。
使用例を以下に示します。

```
// 使用例
Long *data = LongSeq::allocbuf(3);
```

```
data[0] = 0;
data[1] = 1;
data[2] = 2;

LongSeq *seq = new LongSeq(10, 3, data, CORBA_TRUE);
if ( seq == NULL ) {    // 領域の獲得の成否を判定
    ...                 // 獲得に失敗している場合、エラー処理
}
```

allocbuf関数は、要求されたメモリを割り当てられない場合、NULLポインタを返します。allocbuf関数によって割り当てられた配列は、リリースフラグがCORBA_TRUEの場合、sequence解放時に、デストラクタにより解放されます。リリースフラグがCORBA_FALSEの場合は、freebuf関数によって、アプリケーションにより解放しなければなりません。

左シフト代入演算子

左シフト代入演算子は、Anyの_valueにシーケンスのコピー／ポインタを代入します。このとき、AnyのリリースフラグにはCORBA_TRUEが設定されます。
シーケンスのコピーを代入する場合、Anyの_valueの領域が新規に獲得されます。領域の獲得に失敗した場合、_valueはNULLに設定されます。
シーケンスのコピーの代入は、Any_varに対しても実行できます。
使用例を以下に示します。

```
// 使用例
long data1[] = {1, 2, 3, 4, 5};
ODsample::seqtest::sampleseq *seq1 = new ODsample::seqtest::sampleseq( 10, 5, data1 );
CORBA::Any a1;
a1 <<= *seq1;  // a1の_valueにseq1のコピーを設定
               // TypeCodeに_tc_ODsample_seqtest_sampleseq、リリースフラグにCORBA_TRUEを設定
if ( a1.value() == NULL ) {   // 領域の獲得の成否を判定
    ...                       // 獲得に失敗している場合、エラー処理
}

long data2[] = {1, 2, 3, 4};
ODsample::seqtest::sampleseq *seq2 = new ODsample::seqtest::sampleseq( 10, 4, data2 );
CORBA::Any a2;
a2 <<= seq2;   // a2の_valueにseq2のポインタを設定
               // TypeCodeに_tc_ODsample_seqtest_sampleseq、リリースフラグにCORBA_TRUEを設定
               // seq2を解放後は、a2の_valueにはアクセス不可

long data3[] = {1, 2, 3};
ODsample::seqtest::sampleseq *seq3 = new ODsample::seqtest::sampleseq( 10, 3, data3 );
Any a3;
Any_var a3_v( &a3 );
a3_v <<= *seq3; // a3の_valueにseq3のコピーを設定
                // TypeCodeに_tc_ODsample_seqtest_sampleseq、リリースフラグにCORBA_TRUEを設定
if ( a3.value() == NULL ) {   // 領域の獲得の成否を判定
    ...                       // 獲得に失敗している場合、エラー処理
}
```

右シフト代入演算子

右シフト代入演算子は、Anyの_valueに設定されているシーケンスを取り出します。引数としてシーケンスのポインタを設定します。
AnyのTypeCodeが引数のTypeCodeと等しければ、_valueの値を引数に代入し、CORBA_TRUEで復帰します。
右シフト代入演算子は、Any_varに対しても使用できます。
使用例を以下に示します。

```
// 使用例
long data[] = {1, 2, 3, 4, 5};
ODsample::seqtest::sampleseq *seq1 = new ODsample::seqtest::sampleseq (10, 5, data);
```

```
Any a;
a <<= *seq1; // aの_valueにseq1のコピーを設定
             // TypeCodeに_tc_ODsample_seqtest_sampleseq、リリースフラグにCORBA_TRUEを設定

ODsample::seqtest::sampleseq *seq2, *seq3;
if( a >>= seq2 ){      // aのTypeCodeが_tc_ODsample_seqtest_sampleseqの場合、
                       // CORBA_TRUEを返し、seq2にaの_valueの値（seq1）を代入
}

Any_var a_var( &a );
if( a_var >>= seq3 ){  // aのTypeCodeが_tc_ODsample_seqtest_sampleseqの場合、
                       // CORBA_TRUEを返し、seq3にaの_valueの値（seq1）を代入
}
```

4.6.6 構造体

(1)IDLマッピング

IDL言語で構造体structを指定した場合、C++言語でもstructにマッピングされます。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    struct samplefix {  // 構造体(固定長)
        long    para1;
        long    para2;
    };
    struct samplevar {  // 構造体(可変長)
        long    para1;
        string  para2;
    };
    interface  structtest{
        samplefix  op2(
                        in samplefix str1,
                        out samplefix str2,
                        inout samplefix str3
        );
        samplevar  op1(
                        in samplevar str1,
                        out samplevar str2,
                        inout samplevar str3
        );
    };
};
```

C++言語

```
struct samplefix{    // 構造体(固定長)
    CORBA::Long    para1;
    CORBA::Long    para2;
} ;

struct samplevar{    // 構造体(可変長)
    CORBA::Long        para1;
    CORBA::String_var  *para2;
} ;
```

(2)クライアントアプリケーションで扱うパラメタ(固定長)

構造体(固定長)のパラメタ(in、out、inout)を扱う場合は、構造体変数をパラメタに指定します。領域の獲得／解放を行う必要はありません。

```
ODsample::samplefix    strf0, strf1, strf2, strf3;
CORBA::Environment     env;
ODsample::structtest_ptr    obj;

// inパラメタ
strf1.para1 = 10;      // パラメタ設定
strf1.para2 = 11;

// inoutパラメタ
strf3.para1 = 20;      // パラメタ設定
strf3.para2 = 21;

// サーバ呼出し
fix0 = obj->op2( fix1, fix2, fix3, env );
```

(3)クライアントアプリケーションで扱うパラメタ(可変長)

クライアントアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	領域を動的に獲得する場合は、new、データ域獲得関数を使用します。	―
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。
out 復帰	―	(inoutパラメタと同じ)

注意

クライアントおよびスタブで動的に獲得した領域は、不要になった時点で、delete(C++演算子)で解放する必要があります。deleteを発行すると、構造体の可変長領域も解放されます。

クライアントアプリケーションの処理例を以下に示します。

```
ODsample_samplevar         *str0, str1, *str2, str3;
CORBA::Environment         env;
ODsample::structtest_ptr   obj;

// inパラメタ
str1.para1 = 5;                              // パラメタ設定
str1.para2 = (const CORBA::Char *)"xxxx";

// inoutパラメタ
str3.para1 = 6;                              // パラメタ設定
CORBA::Char *str = CORBA::sring_alloc(6);    // 領域獲得
strcpy( str, "yyyyy" );
str3.para2 = str;                            // パラメタ設定

// サーバ呼出し
str0 = obj->op1( str1, str2, str3, env );

// 領域解放
delete( str0 );                              // 復帰パラメタ
delete( str2 );                              // outパラメタ
```

(4)サーバアプリケーションで扱うパラメタ(固定長)

構造体(固定長)のパラメタ(in、out、inout)を扱う場合は、構造体変数をパラメタに指定します。領域の獲得／解放を行う必要はありません。
サーバアプリケーションでの処理例を以下に示します。

```
ODsample::samplefix
ODsample_structtest_impl::op2(
    const ODsample::samplefix  &str1,
    ODsample::samplefix        &str2,
    ODsample::samplefix        &str3,
    CORBA::Environment         &env )
    throw( CORBA::Exception )
{
    // outパラメタ
    str2.para1 = 1;
    str2.para2 = 2;

    // inoutパラメタ
    str3.para1 = 11;
    str3.para2 = 12;

    // 復帰パラメタ
    ODsample::samplefix fix;
    fix.para1 = 21;
    fix.para2 = 22;

    return( fix );
}
```

(5)サーバアプリケーションで扱うパラメタ(可変長)

サーバアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	クライアントからのパラメタ領域	クライアントへのパラメタ領域
in	スケルトンで自動的に獲得/解放されます。	－
inout	スケルトンで自動的に獲得されます。	渡されたパラメタに新しいデータを設定します。古い領域は解放され、新しいデータに置き換わります。
out 復帰値	－	構造体/可変長データの領域をnew/データ域獲得関数で獲得します。 この領域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
ODsample_samplevar *
ODsample_structtest_impl::op1(
    const ODsample_samplevar  &str1,
    ODsample::samplevar       *&str2,
    ODsample::samplevar       &str3,
    CORBA::Environment        &env )
    throw( CORBA::Exception )
{
    // outパラメタ
    str2 = new ODsample::samplevar;                    // 領域獲得
    str2->para1 = 12;                                  // パラメタ設定
    str2->para2 = (const CORBA::Char *)"aaa";

    // inoutパラメタ
```

```
    str3->para1 = 4;                                    // パラメタ設定
    str3->para2 = ( const CORBA::Char *)"bbb";

    // 復帰パラメタ
    ODsample::samplevar *smp = new ODsample::samplevar;  // 領域獲得
    smp->para1 = 5;                                     // パラメタ設定
    smp->para2 = ( const CORBA::Char *)"bbb";

    return( smp );
}
```

4.6.7 共用体

(1)IDLマッピング

IDL言語で共用体unionを指定した場合、C++言語ではデータ型を識別する弁別情報_dと共用体データ域_ptrから構成される共用体クラスにマッピングされます。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    union samplefix switch(long){    // 共用体(固定長)
        case 1: long    para1;
        case 2: long    para2;
    };
    union samplevar switch(long){    // 共用体(可変長)
        case 1: long    para1;
        case 2: string  para2;
    };
    interface  uniontest{
        samplefix  op2(
                        in samplefix uni1,
                        out samplefix uni2,
                        inout samplefix uni3
        );
        samplevar  op1(
                        in samplevar uni1,
                        out samplevar uni2,
                        inout samplevar uni3
        );
    };
};
```

C++言語

```
class samplefix
{
    public:
        samplefix();                    // デフォルトコンストラクタ
        samplefix( const samplefix& ); // コピーコンストラクタ
        ~samplefix();                   // デストラクタ

        samplefix &operator=( const samplefix & );  // 代入演算子

        void   _d( CORBA::Long );    // 弁別子情報の設定
        CORBA::Long  _d() const;     // 弁別子情報の取得

        void   para1( CORBA::Long ); // para1データの設定
        CORBA::Long  para1() const;  // para1データの取得
```

```
        void   para2( CORBA::Long );  // para2データの設定
        CORBA::Long  para2() const;   // para2データの取得

    private:
        CORBA::Long  __d;             // 弁別情報
        void         *_ptr;           // データ域
};

class samplevar
{
    public:
        samplevar();                              // デフォルトコンストラクタ
        samplevar( const samplevar& );            // コピーコンストラクタ
        ~samplevar();                             // デストラクタ

        samplevar &operator=( const samplevar & ); // 代入演算子
        // 弁別子アクセス関数
        void    _d( CORBA::Long );                // 弁別子情報の設定
        CORBA::Long    _d() const;                // 弁別子情報の取得
        // メンバアクセス関数
        void   para1( CORBA::Long );              // para1データの設定
        CORBA::Long    para1() const;             // para1データの取得

        void   para2( CORBA::Char* );             // para2データの設定
        void   para2( const CORBA::Char* );       // para2データの設定
        void   para2( const CORBA::String_var& ); // para2データの設定
        CORBA::Char *   para2() const;            // para2データの取得

    private:
        CORBA::Long    __d;                       // 弁別情報
        void           *_ptr;                     // データ格納域
};
```

(2)クライアントアプリケーションで扱うパラメタ(固定長)

共用体(固定長)のパラメタ(in、out、inout)を扱う場合は、パラメタに宣言した共用体変数を指定します。領域の獲得／解放を行う必要はありません。データ型識別情報とデータ値を、メンバ関数を使用して設定します。
クライアントアプリケーションでの処理例を以下に示します。

```
ODsample::uniontest_ptr    obj;
ODsample::samplefix        unif0, unif1, unif2, unif3;
CORBA::Environment         env;

unif1.para2(100);                  // inパラメタの設定
unif3.para1(200);                  // inoutパラメタの設定

// サーバ呼出し
unif0 = obj->op2( unif1, unif2, unif3, env );
```

(3)クライアントアプリケーションで扱うパラメタ(可変長)

クライアントアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	可変長データ域を動的に獲得する場合は、newを使用します。	－
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
out 復帰値	―	(inoutパラメタと同じ)

注意

クライアントおよびスタブで動的に獲得した領域は、不要になった時点で、delete(C++演算子)で解放する必要があります。deleteを発行すると、構造体の可変長データの領域も解放されます。

クライアントアプリケーションでの処理例を以下に示します。

```
ODsample::uniontest_ptr      obj;
ODsample::samplevar          *uni0, uni1, *uni2, uni3;
CORBA::Environment           env;

uni1.para1(10);                                    // inパラメタ設定
uni3.para2( (const CORBA::Char *)"INOUT::para2" );  // inoutパラメタ設定

// サーバ呼出し
uni0 = obj->op1(  uni1, uni2, uni3, env );

// 領域解放
CORBA_free( uni0 );          // 復帰パラメタ
CORBA_free( uni2 );          // outパラメタ
```

(4)サーバアプリケーションで扱うパラメタ(固定長)

共用体(固定長)のout、inoutパラメタを扱う場合は、データ型の識別情報とデータの値を、メンバ関数を使用して共用体クラスのインスタンスに設定します。共用体の領域の獲得／解放を行う必要はありません。

```
ODsample::samplefix
ODsample_uniontest_impl::op2(
    const ODsample::samplefix  &uni1,
    ODsample::samplefix        &uni2,
    ODsample::samplefix        &uni3,
    CORBA::Environment         &env
 )
throw( CORBA::Exception )
{
    // outパラメタの処理
    uni2.para1(10);              // データの値の設定

    // inoutパラメタの処理
    uni3.para2(100);             // データの値の設定

    // 復帰値の処理
    ODsample::samplefix uni;
    uni.para1(100);              // データの値の設定
    return( uni );
  }
```

(5)サーバアプリケーションで扱うパラメタ(可変長)

サーバアプリケーションのパラメタの扱いについて以下に示します。

パラメタ	クライアントからのパラメタ領域	クライアントへのパラメタ領域
in	スケルトンで自動的に獲得/解放されます。	―
inout	スケルトンで自動的に獲得されます。	メンバアクセス関数で新しいデータを設定します。古い領域は、自動的に解放されます。 可変長データの場合は、データ域獲得関数でデータ域を獲得します。 この領域は、スケルトンで自動的に解放されます。
out 復帰値	―	new/データ域獲得関数で共用体領域/可変長データ域を獲得します。 この領域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
ODsample_samplevar *
ODsample_uniontest_impl::op1(
    const ODsample::samplevar  &uni1,
    ODsample::samplevar        *&uni2,
    ODsample::samplevar        &uni3,
    CORBA::Environment         &env )
{
    // outパラメタ
    uni2 = new ODsample::samplevar;                      // 領域獲得
    uni2->para2( (const CORBA::Char *)"OUT::param" );    // パラメタ設定

    // inoutパラメタ
    uni3.para1(10);                                      // パラメタ設定

    // 復帰パラメタ
    ODsample::samplevar *uni = ODsample::samplevar;      // 領域獲得
    uni.para1(30);                                       // パラメタ設定

    return( uni );
}
```

(6)共用体クラスのメソッド

共用体クラスで定義されているメソッドと意味を以下に示します。

シーケンスクラスのメンバ	意味
デフォルトコンストラクタ	インスタンス生成時、データ格納域_ptrを0で初期化します。
コピーコンストラクタ	インスタンス生成時、_ptrの領域を獲得し、パラメタで指定された値を_ptr、__dに設定します。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定され、__dに0が設定されます。
デストラクタ	インスタンス破壊時、_ptr、_dの値に対する領域を解放します。
代入演算子	_ptrの領域を獲得し、パラメタで指定された値を_ptr、_dに設定します。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定され、__dに0が設定されます。
弁別子アクセス関数	_d(Long)関数は、パラメタで指定した値を弁別子に設定します。 _d()は、弁別子を返します。
メンバアクセス関数	“メンバ名(変数)”の形式で呼び出すことによりメンバの値をデータ格納域に設定します。メモリ不足によりデータ格納域の獲得に失敗した場合は、_ptrにNULLが設定され、__dに0が設定されます。 また、“メンバ名()”の形式で呼び出すことによりデータ格納域の設定されている変数の値を取得します。

デフォルトコンストラクタ

デフォルトコンストラクタは、データ格納域_ptrを0に初期化します。
デフォルトコンストラクタの使用例と処理内容を以下に示します。

```
// IDL
union UnionSmp switch(long){
    case 1: long para1;
    case 2: string para2;
};

// 使用例
UnionSmp *uni1 = new UnionSmp();
        // 以下のデフォルトコンストラクタUnionSmp()が呼ばれ、uni1 が初期化されます。
UnionSmp   uni2;
        // 以下のデフォルトコンストラクタUnionSmp()が呼ばれ、uni2 が初期化されます。

// デフォルトコンストラクタ
UnionSmp::UnionSmp()
{
    _ptr = (void *)0;
}
```

コピーコンストラクタ

コピーコンストラクタは、パラメタで指定された共用体のデータ__d(弁別情報)、_ptr(データ格納域)のデータを複写します。このとき、_ptrの領域が獲得されます。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定され、__dに0が設定されます。
コピーコンストラクタの使用例と処理内容を以下に示します。

```
// IDL
union UnionSmp switch(long){
    case 1: long para1;
    case 2: string para2;
};

// 使用例
UnionSmp *uni1 = new UnionSmp();
uni1->para1(10); // メンバアクセス関数
UnionSmp uni2 = new UnionSmp(*uni1);
                // コピーコンストラクタにより、__dに1、_ptrに値10が設定されます
if ( uni2._d() == 0 ) {   // 成否判定（__dが0の場合、失敗）
    ...                   // 失敗している場合、エラー処理
}

// コピーコンストラクタ
UnionSample::UnionSmp(const UnionSmp &_UnionSmp )
{
    switch( _UnionSmp.__d ) {
       case 1:
           para1( _UnionSmp.para1() );
           break;
       case 2:
           para2( (const CORBA::Char *)_UnionSmp.para2);
           break;
    }
    __d = _UnionSmp._d();
}
```

デストラクタ

デストラクタは、_ptrの領域を解放します。
デストラクタの使用例を以下に示します。

```
// IDL
union UnionSmp switch(long){
    case 1: long para1;
    case 2: string para2;
};

// 使用例
UnionSmp *uni1 = new UnionSmp();
CORBA::Char *str = CORBA::string_alloc(5);
strcpy( str, "data");
uni1->para2(str);    // 共用体メンバ関数。strのポインタが_ptrに渡されます

delete  uni1;        // デストラクタが呼ばれ_ptr(str)の領域も解放されます。
```

代入演算子

```
// IDL
union UnionSmp switch(long){
    case 1: long para1;
    case 2: string para2;
};

// 使用例
UnionSmp *uni1 = new UnionSmp();
uni1->para1(10);        // メンバアクセス関数
UnionSmp uni2 = *uni1;  // 代入演算子により、uni2の_dに1、_ptrに値10が設定されます。
if ( uni2._d() == 0 ) { // 成否判定（__dが0の場合、失敗）
    ...                 // 失敗している場合、エラー処理
}

// 代入演算子
パラメタで指定された値を_ptr,_dに設定します。

UnionSmp&
UnionSmp::operator=( const UnionSmp &_UnionSmp )
{
    if( _ptr ){
        switch( __d ){
            case    1 :
            {
                delete _ptr;
                _ptr = 0;
             }
             break;
             case    2 :
             {
                 CORBA::String_var  *_tmp = ( CORBA::String_var* )_ptr;
                 delete _tmp;
                 _ptr = 0;
             }
        }
    }
    switch( _UnionSmp.__d ){
        case 1:
            para1( _UnionSmp.para1() );
        case 2:
            para2( (const CORBA::Char *)_UnionSmp.para2() );
```

```
            break;
    }
    __d = _UnionSmp._d();
    return *this;
}
```

弁別子アクセス関数

弁別子アクセス関数は、弁別情報(__d)への設定および弁別情報獲得のためのメンバ関数です。
弁別子アクセス関数の使用例と処理内容を以下に示します。

```
// IDL
union UnionSmp switch(long){
    case 1: long para1;
    case 2: string para2;
};

// 使用例
UnionSmp *uni1 = new UnionSmp();
uni1->para1(10);
CORBA::Long l = uni1->_d();     // lの値は1
uni1->_d(2);                    // __dに2が設定されます。
l = uni1->_d();                 // lの値は2
```

メンバアクセス関数

メンバアクセス関数は、メンバ設定関数とメンバ取得関数が提供されます(関数名はメンバ名と同じ)。
メンバアクセス関数の使用例と処理内容を以下に示します。

```
// IDL
typedef long Bytes[64];
struct S {
    long len;
};

union UnionSmp switch(long){
    case 1: long x;
    case 2: Bytes y;
    case 3: string z;
    case 4: S w;
};

// 使用例
UnionSmp *uni1 = new UnionSmp();
uni1->x(10);                // メンバ設定関数。__dに1、_ptrに値10が設定されます。
if ( uni1->_d() == 0 ) {    // 成否判定 (__dが0の場合、失敗)
    ...                     // 失敗している場合、エラー処理
}

UnionSmp *uni2 = new UnionSmp();
Bytes data;
for ( int i = 0; i < 64; i++ )
    data[i] = i;
uni2->y(data);              // __dに2、_ptrにdataを複写します。
if ( uni2->_d() == 0 ) {    // 成否判定 (__dが0の場合、失敗)
    ...                     // 失敗している場合、エラー処理
}

UnionSmp uni3(*uni2);
Bytes_slice *slice = uni3->y();  // _ptrに設定されているポインタを返します。
```

```
UnionSmp uni4;
CORBA::Char *str = CORBA::string_alloc(4);
strcpy( str, "ZZZ" );
uni4.z( str );            // __dに3、_ptrにstrのポインタを設定します。
uni4.z( (const CORBA::Char *)"XXX" );
                          // _ptrに設定されている領域を解放し、データ”XXX”を複写します
                          // (const付き：データの複写，なし：ポインタ渡し)
if ( uni4._d() == 0 ) {   // 成否判定（__dが0の場合、失敗）
    ...                   // 失敗している場合、エラー処理
}

UnionSmp  uni5;
S  str_data;
str_data.len = 5;
uni5.w( data_len );       // __dに4、_ptrにdata_lenを複写します。
if ( uni5._d() == 0 ) {   // 成否判定（__dが0の場合、失敗）
    ...                   // 失敗している場合、エラー処理
}
S  str_data2;
str_data2 = uni5.w();     // str_data2に_ptrに設定されているデータが複写されます。
```

4.6.8 配列

(1)IDLマッピング

IDL言語で配列を指定すると、C++言語でも配列にマッピングされます。また、配列スライス(配列の先頭の次元を減らしたもの。名前:“モジュール名::インタフェース名::配列名_slice”)と配列スライスの領域獲得関数(関数名:“モジュール名::インタフェース名::配列名_alloc”。以降XX_alloc関数と呼ぶ)が生成されます。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface  arraytest{
        typedef long    fix[4][3][2];          // 配列(固定長)
        typedef string  str[2][3][4];          // 配列(可変長)
        fix    op1(in fix para1, out fix para2, inout fix para3 );
        str    op2(in str para1, out str para2, inout str para3 );
    };
};
```

C++言語

```
typedef CORBA::Long           fix[4][3][2];       // 配列(固定長)
typedef CORBA::Long           fix_slice[4][3];    // 配列スライス(固定長)
typedef CORBA::String_var     str[2][3][4];       // 配列(可変長)
typedef CORBA::String_var     str_slice[2][3];    // 配列スライス(可変長)
```

(2)クライアントアプリケーションで扱うパラメタ(固定長)

配列(固定長)のパラメタ(in、out、inout)を扱う場合は、パラメタに宣言した配列変数を指定します。復帰パラメタは、スタブで領域獲得された配列スライスで返されます。この領域は、アプリケーションで不要になった時点で、データ域解放関数(XX_free関数)で解放する必要があります。
クライアントアプリケーションでの処理例を以下に示します。

```
CORBA::Environment             env;
ODsample::arraytest_ptr        obj;
ODsample::arraytest::fix_slice  *fix0;
```

```
ODsample::arraytest::fix        fix1, fix2, fix3;
int i, j, k;

for( i = 0; i < 4; i++ )
for( j = 0; j < 3; j++ )
for( k = 0; k < 2; k++ ) {
    fix1[i][j][k] = i;       // inパラメタ
    fix3[i][j][k] = i*10;    // inoutパラメタ
}

// サーバ呼出し
fix0 = obj->op1( fix1, fix2, fix3, env );

// 領域解放
ODsample::arraytest::fix_free(fix0);
```

(3)クライアントアプリケーションで扱うパラメタ(可変長)

配列(可変長)のin、inoutパラメタを扱う場合は、宣言した配列変数をパラメタで指定します。out、復帰パラメタは、スタブで領域獲得された配列スライスで返されます。この領域は、アプリケーションで不要になった時点で、データ域解放関数(XX_free関数)で解放する必要があります。
クライアントアプリケーションの処理例を以下に示します。

```
CORBA::Environment              env;
ODsample::arraytest_ptr         obj;
ODsample::arraytest::str_slice  *str0, *str2;
ODsample::arraytest::str        str1, str3;
int     i, j, k;
char    buf[120];

for( i = 0; i < 2; i++ )
for( j = 0; j < 3; j++ )
for( k = 0; k < 4; k++ ) {
    sprintf( buf, "str1[%d][%d][%d]", i, j, k );
    str1[i][j][k] = (const CORBA::Char *)buf;       // inパラメタ設定
    sprintf( buf, "str3[%d][%d][%d]", i, j, k );
    str3[i][j][k] = (const CORBA::Char *)buf;       // inoutパラメタ設定
}

// サーバ呼出し
str0 = obj->op2( str1, str2, str3, env );

// 領域解放
ODsample::arraytest::str_free( str0 );
ODsample::arraytest::str_free( str2 );
```

(4)サーバアプリケーションで扱うパラメタ(固定長)

配列(固定長)のパラメタ(in、out、inout)を扱う場合、配列領域はスケルトンで自動的に獲得／解放されます。out／inoutパラメタへのデータ設定は、その領域を使用します。復帰パラメタについては、XX_alloc関数で領域を獲得し、その領域にデータを設定し返します。獲得した領域は、スケルトンで解放されます。なお、領域の獲得に失敗した場合、XX_alloc関数はNULLを返します。
サーバアプリケーションの処理例を以下に示します。

```
ODsample::arraytest::fix_slice *
ODsample_arraytest_impl::op1(
    ODsample::arraytest::fix   para1,
    ODsample::arraytest::fix   para2,
    ODsample::arraytest::fix   para3,
    CORBA::Environment         &env,
    throw( CORBA::Exception )
```

```
{
    int     i, j, k;

    // outパラメタ
    for( i = 0; i < 4; i++ )
    for( j = 0; j< 3; j++ )
    for( k = 0; k < 2; k++ )
        para2[i][j][k] = i+j+k;

    // inoutパラメタ
    for( i = 0; i < 4; i++ )
    for( j = 0; j < 3; j++ )
    for( k = 0; k < 2; k++ )
        para3[i][j][k] = (i+j+k)*10;

    // 復帰パラメタ
    ODsample::arraytest::fix_slice *fix =       // 領域獲得
            ODsample::arraytest::fix_alloc();
    if ( fix != NULL ){
        for( i = 0; i < 4; i++ )
        for( j = 0; j < 3; j++ )
        for( k = 0; k < 2; k++ )
            fix[i][j][k] = i*j*k;
    }

    return( fix );
}
```

(5)サーバアプリケーションで扱うパラメタ(可変長)

配列(可変長)のin／inoutパラメタを扱う場合、配列領域はスケルトンで自動的に獲得／解放されます。out／復帰パラメタについては、XX_alloc関数で配列スライスの領域を獲得し、その領域にデータを設定します。inoutパラメタにデータを設定する場合、新しいデータを代入することで古いデータ域は自動的に解放され、新しいデータ域に置き換わります。ここで獲得した領域は、スケルトンで自動的に解放されます。なお、領域の獲得に失敗した場合、XX_alloc関数はNULLを返します。
サーバアプリケーションでの処理例を以下に示します。

```
ODsample::arraytest::str_slice *
ODsample_arraytest_impl::op2(
    ODsample::arraytest::str        para1,
    ODsample::arraytest::str_slice  *&para2,
    ODsample::arraytest::str        para3,
    CORBA::Environment              &env,
    throw( CORBA::Exception )
{
    int   i, j, k;
    char  buf[120];

    // outパラメタ
    para2 = ODsample::arraytest::str_alloc();           // 領域獲得
    if ( para2 != NULL ){
        for( i = 0; i < 2; i++ )                        // パラメタ設定
        for( j = 0; j< 3; j++ )
        for( k = 0; k < 4; k++ ){
            sprintf( buf, "str2[%d][%d][%d]", i, j, k );
            para2[i][j][k] = (const CORBA::Char *)buf;
        }
    }

    // inoutパラメタ
    for( i = 0; i < 2; i++ )                            // パラメタ設定
    for( j = 0; j < 3; j++ )
    for( k = 0; k < 4; k++ ){
```

```
        sprintf( buf, "str3[%d][%d][%d]", i, j, k );
        para3[i][j][k] = (const CORBA::Char *)buf;      // 新データ代入
    }

    // 復帰パラメタ
    ODsample::arraytest::str_slice *str =
                        ODsample::arraytest::str_alloc();
    if ( str != NULL ){
        for( i = 0; i < 4; i++ )
        for( j = 0; j < 3; j++ )
        for( k = 0; k < 2; k++ ){
            sprintf( buf, "str[%d][%d][%d]", i, j, k );
            str[i][j][k] = (const CORBA::Char *)buf;
        }
    }

    return( str );
}
```

4.6.9 インタフェース宣言(interface)のマッピング

(1)IDLマッピング

IDL言語でインタフェース宣言を定義すると、C++言語ではクラスにマッピングされます。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
interface A {
    ...    // オペレーション定義
};
```

C++言語

```
typedef A    *A_ptr;
class A : public virutual CORBA::Object {
    public:
        static A_ptr        _duplicate( A_ptr );
        static A_ptr        _narrow( CORBA::Object_ptr );
        static A_ptr        _nil();

       // オペレーション定義
       ...
};
```

CORBA::Objectクラスを継承しているため、CORBA::Objectクラスのメンバ関数も使用できます。

メソッド	意味
_duplicate()	パラメタで指定されたインタフェースオブジェクトの複製を作成します。
_narrow()	パラメタで指定されたオブジェクトの情報を複製し、そのオブジェクトをインタフェースオブジェクト型に変換して返します。
_nil()	NILオブジェクトを返します。
is_nil()	指定されたオブジェクトがNILかを調べます(CORBA::Objectクラス継承関数)。
release()	指定されたオブジェクトを解放します(CORBA::Objectクラス継承関数)。

(2)インタフェースクラスのメソッドの使用方法

_duplicate()

パラメタで指定されたインタフェースオブジェクトの複製を作成します。メモリ不足により複製の作成に失敗した場合、_duplicate()はNULLを返します。

```
// 使用例
A_ptr  obj;
obj ...      // objectを何らかの方法で獲得
A_ptr obj_copy = A::_duplicate( obj );
```

_narrow()

パラメタで指定されたオブジェクトの情報を複製し、そのオブジェクトをインタフェースオブジェクト型に変換して返します。メモリ不足により複製の作成に失敗した場合、_narrow()はNULLを返します。

```
// 使用例
CORBA::Object_ptr obj = NamingContext_obj->resolve( *name , env );
                                  // NamingServiceからオブジェクトを取得
A_ptr target = A::_narrow( obj ); // CORBA::Object_ptr型をA_ptrに変換
```

_nil()

NILオブジェクトリファレンスを返します。

```
// 使用例
A_ptr obj;
obj = A::_nil(); // NILを設定
```

is_nil()

指定されたオブジェクトがNILかを調べます。NILであればCORBA_TRUEを、そうでない場合はCORBA_FALSEを返します。

```
// 使用例
A_ptr obj;
CORBA::Environment env;
...
obj->is_nil(env);
```

release()

指定されたオブジェクトを解放します。

```
A_ptr obj;
CORBA::Environment env;
...
obj->release(env);
```

4.6.10 属性宣言(attribute)のマッピング

(1)IDLマッピング

IDL言語で属性宣言(attribute)を定義すると、オブジェクトのデータ設定／取得を行う関数(関数名:IDLで定義された変数名)にマッピングされます。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface       attrtest{
        attribute long  para1;
        attribute string para2;
        readonly attribute long para3;
    };
};
```

C++言語

```
CORBA::Long   para1;
av->para1(100,*env);            // データ設定関数
para1 = av->para1(*env);        // データ取得関数

CORBA::Char*  p;
p = CORBA::string_alloc(7);
av->para2( p,*env );            // データ設定関数
p = av->para2( *env );          // データ取得関数

CORBA::Long   para3;
para3 = av->para3(*env);        // データ取得関数
```

(2)クライアントアプリケーションで扱うパラメタ

データ設定関数でデータ設定に使用するデータ域は、inパラメタと同様に扱われます。また、データ取得関数でデータ取得に使用するデータ域は、復帰パラメタと同様に扱われます。このため、可変長データでは、その領域が不要になった時点で領域を解放する必要があります。

```
ODsample::attrtest_var  av
env = new CORBA::Environment;

CORBA::Long   para1;
para1 = av->para1(*env);
av->para1(100, *env);
para1 = av->para1(*env);
cout << para1 << endl;

CORBA::Char*  p;
p = CORBA::string_alloc(7);
strcpy( p, "STRING" );
av->para2( p,*env );
CORBA::string_free( p );

p = av->para2( *env );
cout << p << endl;
CORBA::string_free( p );

para3 = av->para3(*env);
cout << para3 << endl;
```

(3)サーバアプリケーションで扱うパラメタ

データ設定関数でデータ設定に使用するデータ域はinパラメタと同様に扱われ、スケルトンで自動的に獲得／解放されます。データ取得関数でデータ取得に使用するデータ域は、復帰パラメタと同様に扱われます。このため、可変長データでは、データ域獲得関数で領域獲得する必要があります。獲得した領域は、スケルトンで自動的に解放されます。

```
CORBA::Long
ODsample_attrtest_impl::para1(      // データ取得関数
    CORBA::Environment      & )
    throw( CORBA::Exception )
{
    CORBA::Long   para1;
    return( para1 );
}

void
ODsample_attrtest_impl::para1(      // データ設定関数
    CORBA::Long             data,
    CORBA::Environment      & )
    throw( CORBA::Exception )
{
    CORBA::Long   para1;
    para1 = data;
}

CORBA::Char*
ODsample_attrtest_impl::para2(      // データ取得関数
    CORBA::Environment      & )
    throw( CORBA::Exception )
{
    static CORBA::Char* p;
    p = "test";
    return( p );
}

void
ODsample_attrtest_impl::para2(      // データ設定関数
    const CORBA::Char* data,
    CORBA::Environment      & )
    throw( CORBA::Exception )
{
    CORBA::String_var  para2("");
    para2 = data;
    return;
}

CORBA::Long
ODsample_attrtest_impl::para3(      // データ取得関数
    CORBA::Environment      & )
    throw( CORBA::Exception )
{
    CORBA::Long        para3;
    return( para3 );
}
```

4.6.11 パラメタ受渡しで使用するデータ型

クライアント／サーバアプリケーションでパラメタ受渡しに使用するデータ型を以下に示します。

CORBAデータ型	in	out	inout	復帰
long	Long	Long&	Long&	Long
short	Short	Short&	Short&	Short
unsigned long	ULong	ULong&	ULong&	ULong
unsigned short	UShort	UShort&	UShort&	UShort
long long	LongLong	LongLong&	LongLong&	LongLong

CORBAデータ型		in	out	inout	復帰
float		Float	Float&	Float&	Float
double		Double	Double&	Double&	Double
long double Windows32/64 Solaris32/64		LongDouble	LongDouble&	LongDouble&	LongDouble
char		Char	Char&	Char&	Char
wchar		WChar	WChar&	WChar&	WChar
octet		Octet	Octet&	Octet&	Octet
boolean		Boolean	Boolean&	Boolean&	Boolean
enum		enum	enum&	enum&	enum
string		const Char*	Char*&	Char*&	Char*
wstring		const WChar*	WChar*&	WChar*&	WChar*
any		const any&	any*&	any&	any*
sequence		const sequence&	sequence*&	sequence&	sequence*
struct	固定長	const struct&	struct&	struct&	struct
	可変長	const struct&	struct*&	struct&	struct*
union	固定長	const union&	union&	union&	union
	可変長	const union&	union*&	union&	union*
array(配列)	固定長	const array	array	array	array slice*
	可変長	const array	array slice*&	array	array slice*
Object		CORBA::Object_ptr	CORBA::Object_ptr&	CORBA::Object_ptr&	CORBA::Object_ptr
TypeCode		CORBA::TypeCode_ptr	CORBA::TypeCode_ptr&	CORBA::TypeCode_ptr&	CORBA::TypeCode_ptr

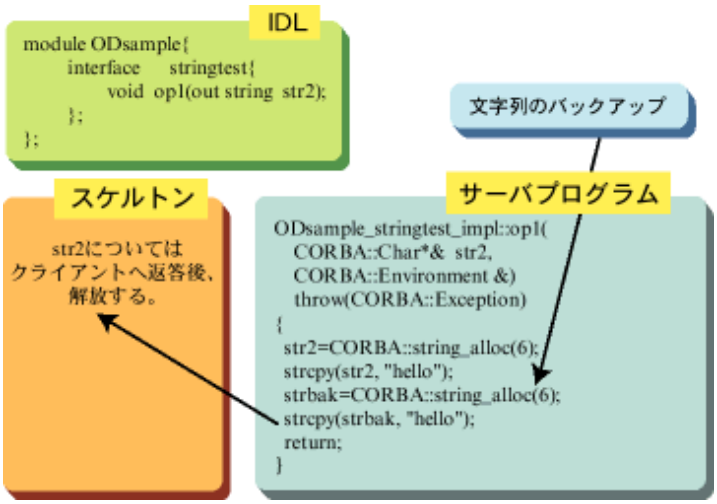
注意

NULLポインタの設定について

クライアントアプリケーションのin、inoutパラメタ、およびサーバアプリケーションのout、inoutパラメタ復帰値では、文字列型、シーケンス型、構造体、共用体、配列データ型でNULLポインタを設定できません。

4.6.12 any型、sequence型のリリースフラグ

可変長データは、データをクライアントへ返却後、スケルトン中で自動的に解放します。
そのため、解放したくないデータは、そのデータをどこか別の領域に複写しておく必要があります。文字列(string)を例に説明します。



しかし、any、sequenceのような複雑な型の場合は、コピーする処理が複雑なため、リリースフラグを設定し、解放する／しないを設定できるようになっています。
リリースフラグの設定方法について以下に示します。

変数型	設定方法
any	コンストラクタ／replaceメソッドを使用ます。デフォルトは、CORBA_FALSEです。(注) 例） CORBA::Any_ptr p = new CORBA::Any(_typecode, value, CORBA_TRUE); CORBA::Any_ptr p = new CORBA::Any; p->replace(_typecode, value, CORBA_TRUE);

sequence	コンストラクタを使用します。デフォルトは、CORBA_FALSEです。(注)
	
	例）
	struct sample {
	string para1;
	long para2;
	};
	typedef sequence<sample> data;
	－＞
	sample_var *p = sample::allocbuf(2);
	data *q = new data(2, 2, p, CORBA_TRUE);
	(2,2はそれぞれmaximum, length)

注) CORBA_TRUEの場合は、解放されます。CORBA_FALSEの場合は、解放されません。

注意

Windows32/64

any型が基本データ型（CORBA::Octet、CORBA::Char、CORBA::Longなど）のデータ領域を保持している場合は、リリースフラグの設定にかかわらず、データ領域を解放しません。以下の例のように、データ領域が必要なくなった時点で、明示的に開放してください。

```
CORBA::Long   *ptr = new CORBA::Long[10]; // データ領域の獲得
for( int i = 0;  i < 10; i++ )
    ptr[i] = i;                           // 値の設定

CORBA::Any    *p = new CORBA::Any(_tc_long, ptr, CORBA_FALSE);
                                          // リリースフラグをCORBA_FALSEに設定

// 領域の解放
```

```
delete  p;                              // any型の解放
delete  ptr;                            // データ領域の解放
```

4.7 varクラス

CORBAでは、クライアント上の可変長データを以下のように扱います。

- in／inoutはユーザが明示的に演算子newなどを使用して領域を獲得し、データ使用後、ユーザが演算子deleteなどを使用してその領域を解放します。
- out／復帰値は、スタブ中で領域が獲得されます。データ使用後、ユーザが演算子deleteなどを使用してその領域を解放します。

ユーザミスにより上記解放処理が行われなかった場合は、解放漏れとなり、無駄な資源がメモリ中に残ることになります。
CORBAでは、上記解放処理の簡易化のため、varクラスが提供されています。varクラスは、CORBAの可変長データを自動変数化して扱うための機能を提供します。varクラスで提供している主な機能は、以下のとおりです。

- 可変長データのポインタを格納し、デストラクタ中でそのデータを解放します。
- 格納しているデータのポインタを取り出しデータを変更します。

varクラスは、以下の4とおりあります。

String_varクラス

文字列を扱うためのvarクラス

WString_varクラス

ワイド文字列を扱うためのvarクラス

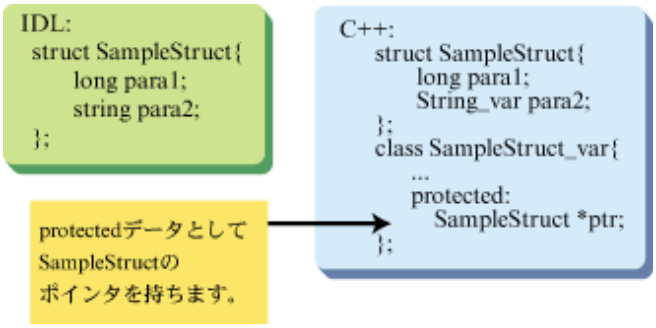
Any_varクラス

Any型を扱うためのvarクラス

ユーザ定義型varクラス

ユーザ定義により作成されるデータ型に対するvarクラス。シーケンス型、構造体、共用体、配列、各インタフェースにより作成されるObject型。

varクラスの使用イメージを以下に示します。




```
func()
{
    SampleStruct *data = new SampleStruct;
    ...                     // dataに値を設定。
    SampleStruct_var  var_data
    var_data = data;        // dataのポインタをvar_dataに格納。
    ...
```

```
    var_data->para1 = 3;  // var_dataに格納されているポインタを取り出し、
                          // 構造体の要素para1のデータを書き換えます。
    ...
}    // 関数終了時に自動変数var_dataが解放されます。その際、デストラクタが起動され、
     // newで獲得したSampleStructの領域も同時に解放されます。
```

4.7.1 String_varクラス

String_varクラスは、文字列操作を簡易化するためのいくつかのメンバ関数を提供します。その他のvarクラスとの大きな違いは、const あり、constなしの場合の2種類のメンバ関数が定義されていることです。
String_varクラスのクラス定義を以下に示します。

```
class String_var {
    public :
        String_var();                                   // デフォルトコンストラクタ
        String_var( char * );                           // char * コンストラクタ
        String_var( const char * );                     // char *コンストラクタ
        String_var( const String_var & );               // コピーコンストラクタ
        ~String_var();                                  // デストラクタ

        String_var    &operator=( char * );             // char *代入演算子
        String_var    &operator=( const char * );       // char *代入演算子
        String_var    &operator=( const String_var & ); // 代入演算子

        operator      char*();                          // 変換オペレータ
        operator      const char*() const;              // 変換オペレータ

        char          &operator[]( ULong );             // [ ]オペレータ
        char          operator[]( ULong ) const;        // [ ]オペレータ
    private:
        char          *_ptr;
};
```

String_varクラスのメンバ	意味
デフォルトコンストラクタ	インスタンス生成時、_ptrをNULLに初期化します。
char *コンストラクタ	constなし: インスタンス生成時、_ptrに指定されたパラメタのポインタを設定します。 constあり: インスタンス生成時、指定されたパラメタのコピーを作成し、_ptrに設定します。_ptrの領域を新規獲得します。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定されます。
コピーコンストラクタ	インスタンス生成時、指定されたパラメタの_ptrのデータのコピーを作成し、_ptrに設定します。_ptrの領域を新規獲得します。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定されます。
デストラクタ	CORBA::string_freeにより_ptrを解放します。
char *代入演算子	constなし: 右辺で指定されたchar*ポインタを_ptrに代入します。 constあり: 右辺で指定されたchar*ポインタデータのコピーを_ptrに代入します。_ptrの領域を新規獲得します。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定されます。
代入演算子	右辺で指定されたString_var変数の_ptrのデータのコピーを作成し、_ptrに代入します。_ptrの領域を新規獲得します。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定されます。
交換演算子	_ptrに設定されているポインタを取り出します。

String_varクラスのメンバ	意味
[ ]演算子	パラメタで指定されている_ptr番目の要素の参照値を取り出します。

デフォルトコンストラクタ

デフォルトコンストラクタは、_ptrをNULLに初期化します。
デフォルトコンストラクタの使用例と処理内容を以下に示します。

```
String_var  data;

CORBA::String_var::String_var()
{
    _ptr = NULL;
}
```

char*コンストラクタ

char*コンストラクタは、constなしの場合、インスタンス生成時、指定されたパラメタのポインタを_ptrに設定します。constありの場合は、インスタンス生成時、指定されたパラメタのコピーを_ptrに設定します。
char *コンストラクタの使用例と処理内容を以下に示します。

```
CORBA::Char * str = CORBA::string_alloc(4);
strcpy( str, "test" );

CORBA::String_var data1( str );                      // char*コンストラクタ(constなし)
CORBA::String_var data2( (const CORBA::Char *)str ); // char *コンストラクタ(constあり)

CORBA::String_var::String_var( char *p )
{
    _ptr = p;
}

CORBA::String_var::String_var( const char *p )
{
    ...  // _ptrにpの内容を複写します
}
```

コピーコンストラクタ

コピーコンストラクタは、インスタンス生成時、指定されたパラメタの_ptrに設定されているデータ自身の_ptrに複写します。
コピーコンストラクタの使用例と処理内容を以下に示します。

```
CORBA::Char * str = CORBA::string_alloc(4);
strcpy( str, "test" );

CORBA::String_var data1( str );      // char*コンストラクタ(constなし)
CORBA::String_var data2( data1 );    // コピーコンストラクタ

CORBA::String_var::String_var(
const CORBA::String_var &r )
{
    ...    // rの_ptrを取り出し、そのコピーを_ptrに設定します。
}
```

デストラクタ

デストラクタは、インスタンス解放時、_ptrに設定されているデータを解放します。
デストラクタの使用例と処理内容を以下に示します。

```
CORBA::Char * str = CORBA::string_alloc(4);
strcpy( str, "test" );

CORBA::String_var *data = new CORBA::String_var( str );  // char*コンストラクタ(constなし)

delete data;    // デストラクタが起動され_ptr(この例ではstrの指す領域)が解放されます

CORBA::String_var::~String_var()
{
    CORBA::string_free( _ptr );
}
```

char *代入演算子

char*代入演算子は、constなしの場合は、右辺で指定されたchar*ポインタを_ptrに代入します。constありの場合は、右辺で指定されたchar*ポインタの指すデータのコピーを作成し、そのポインタを_ptrに設定します。すでに_ptrにデータが設定されている場合は、constなし、constありいずれの場合も、すでに設定されている_ptrのデータを解放した後で、それぞれの処理を実行します。
char*代入演算子の使用例と処理内容を以下に示します。

```
CORBA::Char * str = CORBA::string_alloc(4);
strcpy( str, "test" );

CORBA::String_var data;

data = test;                              // char*代入演算子(constなし)
data = (const CORBA::Char *)"new data";  // char *代入演算子(constあり)

CORBA::String_var  &
CORBA::String_var::operator=( char *p )
{
    if( _ptr )
        CORBA::string_free( _ptr );
    _ptr = p;
    return *this;
}

CORBA::String_var  &
CORBA::String_var::operator=( const char *p )
{
    if( _ptr )
        CORBA::string_free( _ptr );
    ...    // _ptrにpのコピーを作成し、そのポインタを設定
    return *this;
}
```

代入演算子

代入演算子は、右辺で指定されたString_var変数の_ptrの設定されているデータのコピーを作成し、_ptrに設定します。すでに_ptrにデータが設定されている場合は、そのデータを解放した後、処理を行います。
代入演算子の使用例と処理内容を以下に示します。

```
CORBA::Char * str = CORBA::string_alloc(4);
strcpy( str, "test" );

CORBA::String_var data;

data = test;        // char*代入演算子(constなし)

CORBA::String_var  data2;

data2 = data1;        // 代入演算子
```

```
CORBA::String_var  &
CORBA::String_var::operator=( const CORBA::String_var &r )
{
    if( _ptr )
        CORBA::string_free( _ptr );
        ...// rの_ptrを取り出し、そのコピーを_ptrに設定します。
    return *this;
}
```

[ ]演算子

[ ]演算子は、_ptrに設定されているパラメタで指定されたデータ+1番目のデータの参照値を返します。
[ ]演算子の使用例と処理内容を以下に示します。

```
CORBA::Char * str = CORBA::string_alloc(4);
strcpy( str, "test" );

CORBA::String_var data;

data = test;                 // char*代入演算子(constなし)

CORBA::Char x = data[2];     // 3文字目's'が返ります。

char   &
CORBA::String_var::operator[]( ULong index )
{
    return _ptr[index];
}
```

4.7.2 WString_varクラス

WString_varクラスは、文字列操作を簡易化するためのいくつかのメンバ関数を提供します。その他のvarクラスとの大きな違いは、constあり、constなしの場合の2種類のメンバ関数が定義されていることです。
WString_varクラスのクラス定義を以下に示します。

```
class WString_var {
    public :
        WString_var();                                  // デフォルトコンストラクタ
        WString_var( WChar * );                         // WChar * コンストラクタ
        WString_var( const WChar * );                   // WChar *コンストラクタ
        WString_var( const WString_var & );             // コピーコンストラクタ
        ~WString_var();                                 // デストラクタ

        WString_var    &operator=( WChar * );           // WChar *代入演算子
        WString_var    &operator=( const WChar * );     // WChar *代入演算子
        WString_var    &operator=( const WString_var & ); // 代入演算子

        operator       WChar*();                        // 変換オペレータ
        operator       const WChar*()  const;           // 変換オペレータ

        WChar          &operator[]( ULong );            // [ ]オペレータ
        WChar          operator[]( ULong )  const;      // [ ]オペレータ
    private:
        WChar          *_ptr;
};
```

WString_varクラスのメンバ	意味
デフォルトコンストラクタ	インスタンス生成時、_ptrをNULLに初期化します。

WString_varクラスのメンバ	意味
WChar *コンストラクタ	constなし: インスタンス生成時、指定されたパラメタのポインタを_ptrに設定します。 constあり: インスタンス生成時、指定されたパラメタのコピーを作成し、_ptrに設定します。_ptrの領域を新規獲得します。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定されます。
コピーコンストラクタ	インスタンス生成時、指定されたパラメタの_ptrのデータのコピーを作成し、_ptrに設定します。_ptrの領域を新規獲得します。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定されます。
デストラクタ	_ptrをCORBA::wstring_freeによって解放します。
WChar *代入演算子	constなし: 右辺で指定されたWChar*ポインタを_ptrに代入します。 constあり: 右辺で指定されたWChar*ポインタデータのコピーを_ptrに代入します。_ptrの領域を新規獲得します。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定されます。
代入演算子	右辺で指定されたWString_var変数の_ptrのデータのコピーを作成し、_ptrに代入します。_ptrの領域を新規獲得します。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定されます。
交換演算子	_ptrに設定されているポインタを取り出します。
[ ]演算子	パラメタで指定されている_ptr番目の要素の参照値を取り出します。

デフォルトコンストラクタ

デフォルトコンストラクタは、_ptrをNULLに初期化します。
デフォルトコンストラクタの使用例と処理内容を以下に示します。

```
WString_var  data;

CORBA::WString_var::WString_var()
{
    _ptr = NULL;
}
```

WChar*コンストラクタ

WChar*コンストラクタは、constなしの場合、インスタンス生成時、指定されたパラメタのポインタを_ptrに設定します。constありの場合は、インスタンス生成時、指定されたパラメタのコピーを_ptrに設定します。
WChar *コンストラクタの使用例と処理内容を以下に示します。

```
CORBA::WChar * str = CORBA::wstring_alloc(3);
(*str)[0] = 'テ';
(*str)[1] = 'ス';
(*str)[2] = 'ト';
(*str)[3] = '¥0';

CORBA::WString_var data1( str );                        // WChar*コンストラクタ(constなし)
CORBA::WString_var data2( (const CORBA::WChar *)str ); // WChar *コンストラクタ(constあり)

CORBA::WString_var::WString_var( WChar *p )
{
    _ptr = p;
}
```

```
CORBA::WString_var::WString_var( const WChar *p )
{
    ...  // _ptrにpの内容を複写します
}
```

コピーコンストラクタ

コピーコンストラクタは、インスタンス生成時、指定されたパラメタの_ptrに設定されているデータ自身の_ptrに複写します。
コピーコンストラクタの使用例と処理内容を以下に示します。

```
CORBA::WChar * str = CORBA::wstring_alloc(3);
(*str)[0] = 'テ';
(*str)[1] = 'ス';
(*str)[2] = 'ト';
(*str)[3] = '¥0';

CORBA::WString_var data1( str );      // WChar*コンストラクタ(constなし)
CORBA::WString_var data2( data1 );    // コピーコンストラクタ

CORBA::WString_var::WString_var(
const CORBA::WString_var &r )
{
    ...    // rの_ptrを取り出し、そのコピーを_ptrに設定します。
}
```

デストラクタ

デストラクタは、インスタンス解放時、_ptrに設定されているデータを解放します。
デストラクタの使用例と処理内容を以下に示します。

```
CORBA::WChar * str = CORBA::wstring_alloc(3);
(*str)[0] = 'テ';
(*str)[1] = 'ス';
(*str)[2] = 'ト';
(*str)[3] = '¥0';

CORBA::WString_var *data = new CORBA::WString_var( str );  // WChar*コンストラクタ(constなし)

delete data;   // デストラクタが起動され_ptr(この例ではstrの指す領域)が解放されます

CORBA::WString_var::~WString_var()
{
    CORBA::wstring_free( _ptr );
}
```

WChar *代入演算子

WChar*代入演算子は、constなしの場合、右辺で指定されたWChar*ポインタを_ptrに代入します。constありの場合は、右辺で指定されたWChar*ポインタの指すデータのコピーを作成し、そのポインタを_ptrに設定します。すでに_ptrにデータが設定されている場合は、constなし、constありいずれの場合も、すでに設定されている_ptrのデータを解放した後で、それぞれの処理を実行します。
WChar*代入演算子の使用例と処理内容を以下に示します。

```
CORBA::WChar * str = CORBA::wstring_alloc(3);
(*str)[0] = 'テ';
(*str)[1] = 'ス';
(*str)[2] = 'ト';
(*str)[3] = '¥0';

CORBA::WString_var data;
```

```
data = (CORBA::WChar *)str;        // WChar*代入演算子(constなし)
data = (const CORBA::WChar *)str;  // WChar*代入演算子(constあり)

CORBA::WString_var  &
CORBA::WString_var::operator=( WChar *p )
{
    if( _ptr )
        CORBA::wstring_free( _ptr );
    _ptr = p;
    return *this;
}

CORBA::WString_var  &
CORBA::WString_var::operator=( const WChar *p )
{
    if( _ptr )
        CORBA::wstring_free( _ptr );
    ...    // _ptrにpのコピーを作成し、そのポインタを設定
    return *this;
}
```

代入演算子

代入演算子は、右辺で指定されたWString_var変数の_ptrの設定されているデータのコピーを作成し、_ptrに設定します。すでに_ptrにデータが設定されている場合は、そのデータを解放した後、処理を行います。
代入演算子の使用例と処理内容を以下に示します。

```
CORBA::WChar * str = CORBA::wstring_alloc(3);
(*str)[0] = 'テ';
(*str)[1] = 'ス';
(*str)[2] = 'ト';
(*str)[3] = '¥0';

CORBA::WString_var data;

data = test;    // WChar*代入演算子(constなし)

CORBA::WString_var  data2;

data2 = data1;  // 代入演算子

CORBA::WString_var  &
CORBA::WString_var::operator=( const CORBA::WString_var &r )
{
    if( _ptr )
        CORBA::wstring_free( _ptr );
        ...    // rの_ptrを取り出し、そのコピーを_ptrに設定します。
    return *this;
}
```

[ ]演算子

[ ]演算子は、_ptrに設定されているパラメタで指定されたデータ+1番目のデータの参照値を返します。
[ ]演算子の使用例と処理内容を以下に示します。

```
CORBA::WChar * str = CORBA::wstring_alloc(3);
strcpy( str, "テスト" );

CORBA::WString_var data;

data = test;                    // WChar*代入演算子(constなし)
```

```
CORBA::WChar x = data[2];      // 3文字目'ト'が返ります。

WChar   &
CORBA::WString_var::operator[]( ULong index )
{
    return _ptr[index];
}
```

4.7.3 Any_varクラス

Any_varクラスは、Any型データを扱うためのいくつかのメンバ関数を提供します。

```
class Any_var {
    public:
        Any_var();                              // デフォルトコンストラクタ
        Any_var( Any *);                        // Any*コンストラクタ
        Any_var( const Any_var & );             // コピーコンストラクタ
        ~Any_var();                             // デストラクタ

        Any_var  &operator=( Any * );           // Any*代入演算子
        Any_var  &operator=( const Any_var & ); // 代入演算子

        Any       *operator->();                // ポインタ演算子
        operator  Any*()    const;              // 交換演算子

    private:
        Any       *_ptr;
};
```

Any_varクラスのメンバ	意味
デフォルトコンストラクタ	インスタンス生成時、_ptrを0に初期化します。
Any*コンストラクタ	インスタンス生成時、パラメタで指定されたAnyポインタを_ptrに代入します。
コピーコンストラクタ	インスタンス生成時、指定されたパラメタの_ptrのデータのコピーを作成し、自身の_ptrに設定します。_ptrの領域を新規獲得します。メモリ不足により領域の獲得に失敗した場合、_ptrにNULLが設定されます。
デストラクタ	_ptrの指す領域を解放します。
Any* 代入演算子	右辺で指定されたAny*ポインタを_ptrに代入します。
代入演算子	右辺で指定されたAny_varの_ptrに設定されているデータのコピーを作成し、そのポインタを_ptrに設定します。_ptrの領域を新規獲得します。メモリ不足により領域の獲得に失敗した場合、_ptrにNULLが設定されます。
ポインタ演算子	_ptrに設定されているデータを取り出します。
交換演算子	_ptrに設定されているデータを取り出します。

デフォルトコンストラクタ

デフォルトコンストラクタは、インスタンス生成時、_ptrに0を設定します。
デフォルトコンストラクタの使用例と処理内容を以下に示します。

```
CORBA::Any_var any;        // デフォルトコンストラクタが起動されます。

CORBA::Any_var::CORBA::Any_var()
{
```

```
    _ptr = 0;
}
```

Any*コンストラクタ

Any*コンストラクタは、インスタンス生成時、パラメタで指定されたAnyポインタを_ptrに代入します。
Any*コンストラクタの使用例と処理内容を以下に示します。

```
CORBA::Any *data = new CORBA::Any();
CORBA::Long x = 3;
(*data) <<= x;

CORBA::Any_var  any(data);    // Any*コンストラクタが起動されます。

CORBA::Any_var::Any_var( Any *p )
{
    _ptr = p;
}
```

コピーコンストラクタ

コピーコンストラクタは、インスタンス生成時、パラメタで指定されたAny_var変数に設定されている_ptrのコピーを作成し、自身の_ptrに代入します。
コピーコンストラクタの使用例と処理内容を以下に示します。

```
CORBA::Any *data = new CORBA::Any();
CORBA::Long x = 3;
(*data) <<= x;

CORBA::Any_var  any(data);     // Any*コンストラクタが起動されます。

CORBA::Any_var  any2( any );   // コピーコンストラクタが起動されます。

CORBA::Any_var::Any_var( const CORBA::Any_var &r )
{
    ...    // rの_ptrを取り出し、そのコピーを_ptrに設定します。
}
```

デストラクタ

デストラクタは、インスタンス解放時、_ptrに設定されている領域を解放します。
デストラクタの使用例と処理内容を以下に示します。

```
func()
{
    CORBA::Any *data = new CORBA::Any();
    CORBA::Long x = 3;
    (*data) <<= x;

    CORBA::Any_var  any(data);  // Any*コンストラクタが起動されます。
    CORBA::Any_var  *any2 = new Any_var( any );

    // コピーコンストラクタが起動されます。

    delete any2;    // デストラクタが起動されます。
}   // 自動変数anyの解放時にデストラクタが起動されます。

CORBA::Any_var::~Any_var()
{
```

```
    delete _ptr;
}
```

Any*代入演算子

Any*代入演算子は、右辺で指定されたAny*ポインタを_ptrに設定します。すでに_ptrにデータが設定されている場合は、解放した後、処理を行います。
Any*代入演算子の使用例と処理内容を以下に示します。

```
CORBA::Any_var  var_data;
CORBA::Any *data = new CORBA::Any();
CORBA::Long x = 3;

(*data) <<= x;

var_data = data; // Any *代入演算子が起動されます。

CORBA::Any_var::operator=( Any *p )
{
    if( _ptr )
        delete _ptr;
    _ptr = p;
}

CORBA::Any_var::operator=( const Any_var &r )
{
    if( _ptr )
        delete _ptr;
    ...    // rの_ptrを取り出し、そのコピーを_ptrに設定します。
}
```

代入演算子

代入演算子は、右辺で指定されたAny_varのptrに設定されているデータのコピーを作成し自身の_ptrに代入します。すでに_ptrにデータが設定されている場合は、解放した後、処理を行います。
代入演算子の使用例と処理内容を以下に示します。

```
CORBA::Any_var  var_data;
CORBA::Any *data = new CORBA::Any();
CORBA::Long x = 3;

(*data) <<= x;

var_data = data;                    // Any *代入演算子が起動されます。

CORBA::Any_var var_data2 = var_data; // 代入演算子が起動されます。
```

ポインタ演算子

ポインタ演算子は、_ptrに設定されているポインタを返します。
ポインタ演算子の使用例と処理内容を以下に示します。

```
CORBA::Any_var  var_data;
CORBA::Any *data = new CORBA::Any();
CORBA::Long x = 3;

(*data) <<= x;

CORBA::Long  *p = var_data->value();
// ポインタ演算子が起動され、CORBA::Any *データが取り出され、
```

```
// そのCORBA::Any*データに対してvalueメソッドを起動します。

CORBA::Any_var::operator->()
{
    return _ptr;
}
```

交換演算子

交換演算子は、_ptrに設定されているポインタを返します。
交換演算子の使用例と処理内容を以下に示します。

```
CORBA::Any_var  var_data;
CORBA::Any *data = new CORBA::Any();
CORBA::Long x = 3;

(*data) <<= x;

CORBA::Any  *p = (CORBA::Any*)var_data;
// 交換演算子が起動され、CORBA::Any *データが取り出されます。

CORBA::Any_var::operator CORBA::Any*()
{
    return _ptr;
}
```

4.7.4 ユーザ定義varクラス

ユーザ定義型データ(シーケンス、構造体、共用体、配列型、IDLで定義された各インタフェースのObjectクラス)は、IDLコンパイラにより、各定義型に応じたクラスとそのクラス名に_varを付けたvarクラスが作成されます。Tをユーザ定義したデータ型名とした場合、このvarクラスには、以下のメンバ関数が定義されます。

```
class T_var {
    public:
        T_var();                          // デフォルトコンストラクタ
        T_var( T* );                      // T*コンストラクタ
        T_var( const T_var & );           // コピーコンストラクタ
        ~T_var();                         // デストラクタ

        T_var &operator=( T* );           // T*代入演算子
        T_var &operator=( const T_var & ); // 代入演算子

        operator T*()const;               // 交換演算子
        T* operator->() const;            // ポインタ演算子

        // 以下はシーケンスの場合のみ
        T_var &operator[]( CORBA::ULong );

        // T(可変長)のsequenceのパラメタ＋1番目の要素の取り出し
        const T_var &operator[]( CORBA::ULong );
        // T(可変長)のsequenceのパラメタ＋1番目の要素の取り出し
        T_var     &operator[]( CORBA::ULong );
        // [ ]演算子
        const T     &operator[]( CORBA::ULong );
        // [ ]演算子

    protected:
        T *ptr;
}
```

T_varクラスのメンバ	意味
デフォルトコンストラクタ	インスタンス生成時、新規T*データを作成し初期化します。
T*コンストラクタ	インスタンス生成時、パラメタで指定されたTポインタを_ptrに代入します。
コピーコンストラクタ	インスタンス生成時、指定されたパラメタの_ptrのデータのコピーを作成し、自身の_ptrに設定します。_ptrの領域を新規獲得します。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定されます。
デストラクタ	_ptrの指す領域を解放します。
T*代入演算子	右辺で指定されたT*ポインタを_ptrに代入します。
代入演算子	右辺で指定されたT_varの_ptrに設定されているデータのコピーを作成し、そのポインタを_ptrに設定します。_ptrの領域を新規獲得します。メモリ不足により領域の獲得に失敗した場合は、_ptrにNULLが設定されます。
交換演算子	_ptrに設定されているデータを取り出します。
ポインタ演算子	_ptrに設定されているデータを取り出します。
[ ]演算子	_ptrに設定されているパラメタ+1番目の要素を取り出します。

デフォルトコンストラクタ

デフォルトコンストラクタは、インスタンス生成時_ptrに0を設定します。
デフォルトコンストラクタの使用例と処理内容を以下に示します。

```
// IDL
struct X {
    long x;
    string y;
};

// C++
X_var x;

X_var::X_var()
{
    _ptr = new X;
}
```

T*コンストラクタ

T*コンストラクタは、インスタンス生成時、パラメタで指定されたTポインタを_ptrに代入します。
T*コンストラクタの使用例と処理内容を以下に示します。

```
// IDL
struct X {
    long x;
    string y;
};
typedef sequence<long>  LongSeq;

// C++
X *data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";

X_var x( data );            // T*コンストラクタ

CORBA::Long *buffer = LongSeq::allocbuf(2);
for( int i = 0; i < 2; i++ )
```

```
    buffer[i] = i;
LongSeq *seq = new LongSeq(2,2,buffer,CORBA_TRUE);

LongSeq_var seq_var(seq);   // T*コンストラクタ

X_var::X_var( X *p )
{
    _ptr = p;
}

LongSeq_var::Long_var( LongSeq *p )
{
    _ptr = p;
}
```

コピーコンストラクタ

コピーコンストラクタは、インスタンス生成時、パラメタで指定されたT_varデータの_ptrのコピーを作成し、自身の_ptrに代入します。
コピーコンストラクタの使用例と処理内容を以下に示します。

```
// IDL
struct X {
    long x;
    string y;
};
typedef sequence<long>  LongSeq;

// C++
X *data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";

X_var x_data1( data );              // T*コンストラクタが起動されます。
X_var x_data2( x_data1 );           // コピーコンストラクタが起動されます。

CORBA::Long *buffer = LongSeq::allocbuf(2);
for( int i = 0; i < 2; i++ )
    buffer[i] = i;
LongSeq *seq = new LongSeq(2,2,buffer,CORBA_TRUE);

LongSeq_var seq_var1(seq);          // T*コンストラクタが起動されます。
LongSeq_var seq_var2( seq_var1 );   // コピーコンストラクタが起動されます。

X_var::X_var( const X_var &r )
{
    ...    // rの_ptrを取り出し、そのコピーを_ptrに設定します。
}

LongSeq_var::LongSeq_var( const LongSeq_var &r )
{
    ...    // rの_ptrを取り出し、そのコピーを_ptrに設定します。
}
```

デストラクタ

デストラクタは、インスタンス破壊時、_ptrの指すデータを解放します。
デストラクタの使用例と処理内容を以下に示します。

```
// IDL
struct X {
    long x;
```

```
    string y;
};

// C++
X *data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";
X_var *var_data = new X_var(data);

delete var_data;

X_var::~X_var()
{
    delete _ptr;
}
```

T*代入演算子

T*代入演算子は、パラメタで指定されたTポインタを_ptrに代入します。
T*代入演算子の使用例と処理内容を以下に示します。

```
// IDL
struct X {
    long x;
    string y;
};

// C++
X *str_data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";
X_var x_var;

x_var = data; // 代入演算子が起動されます。

X_var::X_var::operator=( T* p)
{
    _ptr = p;
}
```

代入演算子

代入演算子は、パラメタで指定されたT_varの_ptrに設定されているデータのコピーを作成し、_ptrに代入します。
代入演算子の使用例と処理内容を以下に示します。

```
// IDL
struct X {
    long x;
    string y;
};

// C++
X *data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";
X_var x_var;

x_var = data;          // T*代入演算子が起動されます。

X_var x_var2 = x_var;  // 代入演算子が起動されます。
```

```
X_var::X_var::operator=( const X_var &r )
{
    ...    // rの_ptrを取り出し、そのコピーを_ptrに設定します。
}
```

交換演算子

交換演算子は、_ptrに設定されているデータを取り出します。
交換演算子の使用例と処理内容を以下に示します。

```
// IDL
struct X {
    long x;
    string y;
};

// C++
X *str_data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";
X_var x_var;

x_var = data;       // T*代入演算子が起動されます。

X *p = (X *)x_var;  // 交換演算子が起動されます。

X * X_var::operator X*()
{
    return _ptr;
}
```

ポインタ演算子

ポインタ演算子は、_ptrに設定されているデータを取り出します。
ポインタ演算子の使用例と処理内容を以下に示します。

```
// IDL
struct X {
    long x;
    string y;
};

// C++
X *data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";
X_var x_var;

x_var = data;   // T*代入演算子が起動されます。

CORBA::Long z = x_var->x;
               // 交換演算子が起動されます。zには、3が代入されます。
x_var->x = 5;  // 交換演算子が起動され、_ptrの指す構造体の要素xに5が代入されます。
x_var->y = (const CORBA::Char *)"test";
               // 交換演算子が起動され、_ptrの指す構造体の要素yに文字列"test"が代入されます。

X * X_var::operator->()
{
    return _ptr;
}
```

[ ]演算子

[ ]演算子は、0から数えて[ ]で指定された番号番目の要素を返します。
[ ]演算子の使用例と処理内容を以下に示します。

```
// IDL
typedef sequence<long>    LongSeq;
typedef sequence<string>  StringSeq;

// C++
CORBA::Long *p = LongSeq::allocbuf(2);
p[0] = 2; p[1] = 3;

LongSeq *long_seq = new LongSeq(2,2,p,CORBA_TRUE);

LongSeq_var long_seq_var = long_seq;    // 代入演算子

CORBA::Long y = long_seq_var[1];        // [ ]演算子。yには3が代入されます。

CORBA::String_var *q = StringSeq::allocbuf(2);
q[0] = (const CORBA::Char *)"test";
q[1] = (const CORBA::Char *)"data";

StringSeq *str_seq = new StringSeq(2,2,q,CORBA_TRUE);

StringSeq_var str_seq_var = str_seq;    // 代入演算子

CORBA::String_var str = str_seq_var[1]; // strの_ptrには、"data"が代入されます。

CORBA::Long &
LongSeq_var::operator[](CORBA::ULong index)
{
    return _ptr->operator[]( index );
}

CORBA::String_var &
StringSeq_var::operator[](CORBA::ULong index)
{
    return (const CORBA::String_var &)
    _ptr->operator[](index);
}
```

4.8 アプリケーション作成時の注意事項

アプリケーションを作成する場合の注意事項については、“第2章 CORBAアプリケーション開発時の留意事項”を参照してください。

第5章 アプリケーションの開発(Java言語)

注意

本マニュアルは、マルチ言語サービスのCORBAサービスのアプリケーションを作成するためのマニュアルです。Java EEのアプリケーションを作成する場合は、“Java EE運用ガイド”を参照してください。

ポイント

- Java言語アプリケーションを開発する場合は、IJServerクラスタによる運用が可能であるため、Java EEによる開発を推奨します。IJServerクラスタによる運用は、CORBAワークユニットによる運用と比較して以下のメリットがあります。IJServerクラスタの詳細については、“Java EE運用ガイド”を参照してください。
  - ー GlassFish v2.1をベースとしたJava EE機能を提供します。
  - ー Interstage管理コンソールを使用してJava VMの以下の情報を参照できます。
    - ヒープ情報
    - ガーベジコレクション情報
  - ー 起動時実行クラス/停止時実行クラスを指定できます。

  Java言語のCORBAアプリケーションのワークユニット運用を行う場合、Interstage管理コンソールを使用してヒープ情報を参照できませんが、jheapコマンドを使用してヒープ情報を参照できます。jheapコマンドの詳細については、“トラブルシューティング集”の“Javaツール機能”ー“jheap”を参照してください。
- 本マニュアルでは、製品“Interstage Studio”を略称“Studio”で表記しています。
- 文中のJBKプラグインは、本製品のクライアントパッケージに同梱されているJBKプラグインを示します。

5.1 Javaアプレットの開発手順(プレインストール型Javaライブラリ)

ここでは、CORBAクライアントとしてのJavaアプレットの開発手順について説明します。
プレインストール型Javaライブラリ使用時のJavaアプレットの開発手順を以下に示します。

5.1.1 HTMLファイルの記述

アプレットを実行するには、HTMLファイルで<applet>タグを使用してアプレットを指定します。

Internet Explorerにプラグインすることにより、ブラウザのJava VMをJDKに置き換えて運用します。アプレット運用時の注意事項については、“5.21.2 クライアント環境でのアプレット運用時の注意事項”を参照してください。

HTMLファイルの記述例(タイトル: Java sample Applet、Javaアプレット: Sample)を以下に示します。

(1)JBKプラグインを使用する場合

HTMLファイルを以下のように記述します。詳細については、Studioの“J Business Kit オンラインマニュアル”を参照してください。

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin" NAME="Sample" CODE="Sample.class" WIDTH=300 HEIGHT=250>
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

(2)JBKプラグインを使用する場合(jar形式アーカイブファイル使用)

<PARAM>タグのARCHIVE指定、および<EMBED>タグのARCHIVE指定で、Sample.jarをダウンロードするように指定します。

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="Sample.jar">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin" CODE="Sample.class" ARCHIVE="Sample.jar" WIDTH=300 HEIGHT=250>
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

5.1.2 アプレットのプログラミング

サーバ上のCORBA準拠のアプリケーションと連携するJavaアプレットでは、その先頭でimport文を使用してorg.omg.CORBAクラスを参照することを宣言します。また、HTMLファイルの<applet>タグで記述したクラス名でクラス宣言します。

```
import org.omg.CORBA.*;                    // org.omg.CORBAクラス
import java.awt.*;                         // 抽象ウィンドウツールキットクラス
```

```
public class Sample extends java.applet.Applet // appletクラスの宣言
{
    ...                                         // Java言語による処理の記述
}
```

注意

Javaアプレットのファイル名は、アプレット内で宣言したクラス名と同じ(大文字／小文字の区別あり)にする必要があります。

5.1.3 IDL生成ファイルとアプレットのコンパイル

IDLコンパイラを実行し、Javaアプレット用のスタブを生成します。また、生成されたスタブとJavaアプレットをコンパイルします。コンパイルには、本製品が提供しているjavacコマンドを使用します。

1. インタフェース(xxx.java,xxxOperations.java)
2. Helperクラス(xxxHelper.java)
3. Holderクラス(xxxHolder.java)
4. スタブクラス(_xxxStub.java)
5. ユーザの作成したJavaアプレット

注意事項

コンパイルの実行時、環境変数CLASSPATHにカレントディレクトリ(.)およびJavaライブラリを指定する必要があります。
設定例を以下に示します。

例

Windows32/64(OD_HOME:C:¥Interstage¥ODWIN)

```
set CLASSPATH=.;%OD_HOME%¥etc¥class¥ODjava4.jar;%CLASSPATH%
```

Solaris32/64(OD_HOME:/opt/FSUNod) Linux32/64(OD_HOME:/opt/FJSVod)

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export CLASSPATH
```

5.1.4 Javaクラスファイルのアーカイブ

JavaクラスファイルをWebサーバに登録する場合、ダウンロード時間を短縮するため、複数のファイルを一度にダウンロードし、クラスファイルをまとめたアーカイブファイルを作成します。アーカイブファイルは、jarコマンド(Java Development Kit(以降JDK)に含まれる)を使用して作成します。
ここで作成したJavaクラスファイルのアーカイブファイル(アプレット)をWebサーバからダウンロードして実行する場合は、アーカイブファイルに対して署名を行う必要があります。署名については、“5.8 アプレットのデジタル署名”を参照してください。

参照

jarコマンドの使用方法などについては、JDKのドキュメントを参照してください。

サブディレクトリを含むクラスファイルからアーカイブファイルを作成する場合のコマンド使用例を以下に示します。

■jarコマンド使用例

作成するアーカイブファイル名、およびアーカイブするクラスファイルを指定します。作成されたアーカイブファイルには、サブディレクトリ内のファイルが含まれています。

```
jar cvf Sample.jar *.class Samplemod¥*.class
    adding: Samplemod/_SampleintfStub.class (in=1282) (out=704) (deflated 45%)
    adding: Samplemod/Sampleintf.class (in=302) (out=215) (deflated 28%)
    adding: Samplemod/SampleintfHelper.class (in=2175) (out=994) (deflated 54%)
    adding: Samplemod/SampleintfHolder.class (in=907) (out=461) (deflated 49%)
```

注意

アーカイブファイルとするクラスファイルを指定する場合は、パス指定の情報もアーカイブファイル内に含まれます。カレントディレクトリをアーカイブファイルとするクラスファイルが格納される最上位ディレクトリに移動後、クラスファイルの構成に従って指定し、アーカイブファイルを作成してください。

■HTMLファイル記述例

Javaアプレット名“Sample”およびIDL生成ファイル“Sample.jar”としてアーカイブした場合のHTMLファイル記述例を、“5.1.1 HTMLファイルの記述”に示したHTMLファイルを例として以下に示します。

JBKプラグインを使用する場合(jar形式アーカイブファイル使用)

<PARAM>タグのARCHIVE指定、および<EMBED>タグのARCHIVE指定で、Sample.jarをダウンロードするように指定します。

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="NAME" VALUE="Sample">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="Sample.jar">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin" NAME="Sample" CODE="Sample.class" ARCHIVE="Sample.jar" WIDTH=300 HEIGHT=250>
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

5.2 Javaアプレットの開発手順(Portable-ORB)

ここでは、CORBAクライアントとしてのJavaアプレットの開発手順について説明します。
Portable-ORB使用時のJavaアプレットの開発手順を以下に示します。

5.2.1 HTMLファイルの記述

アプレットを実行するには、HTMLファイルで<applet>タグを使用してアプレットを指定します。

Internet Explorerにプラグインすることによりブラウザの Java VMをJDKに置き換えて運用します。アプレット運用時の注意事項については、"5.21.2 クライアント環境でのアプレット運用時の注意事項"を参照してください。

また、Portable-ORBをダウンロードする運用では、Javaアプレットを実行するHTMLファイルで、<APPLET ARCHIVE>タグまたは<PARAM>タグ(cabbase)を使用して以下のファイルを記述します。

ORBコア	ネーミングサービス (注)	インタフェースリポジトリ(注)
ODporb4_plugin.jar	CosNaming4_plugin.jar	InterfaceRep4_plugin.jar

注) ネーミングサービス・インタフェースリポジトリが必要な場合は、HTMLファイルに記述します。

使用するPortable-ORBのファイル名は、運用形態により異なります。Javaライブラリについては、"B.2 Portable-ORBの提供プログラム"を参照してください。
HTMLファイルの記述例(タイトル: Java sample Applet、Javaアプレット名: Sample)を以下に示します。

(1)JBKプラグインを使用する場合

HTMLファイルを以下のように記述します。詳細については、Studioの"J Business Kit オンラインマニュアル"を参照してください。

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
```

```
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar">
<PARAM NAME="PORB_HOME" VALUE="PORBDIR">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
    CODE="Sample.class" WIDTH=300 HEIGHT=250
    ARCHIVE="ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar"
    PORB_HOME="PORBDIR">
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

(2)JBKプラグインを使用する場合（jar形式アーカイブファイル使用）

<PARAM>タグのARCHIVE指定、および<EMBED>タグのARCHIVE指定で、Sample.jarをダウンロードするように指定します。

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="Sample.jar,ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar">
<PARAM NAME="PORB_HOME" VALUE="PORBDIR">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
    CODE="Sample.class" WIDTH=300 HEIGHT=250
    ARCHIVE="Sample.jar,ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar"
    PORB_HOME="PORBDIR">
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

(3) Portable-ORBをダウンロードしない場合(JBKプラグインを使用する場合）

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="Sample.jar">
<PARAM NAME="PORB_HOME" VALUE="PORBDIR">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
    CODE="Sample.class" WIDTH=300 HEIGHT=250
    ARCHIVE="Sample.jar" PORB_HOME="PORBDIR">
</EMBED>
</COMMENT>
</OBJECT>
```

```
</BODY>
</HTML>
```

ポイント

- 上記のPortable-ORBをダウンロードする場合の例では、Portable-ORBのファイルがHTMLファイルと同じ格納先ディレクトリにあるものと仮定しています。同じディレクトリにない場合は、ARCHIVE/VALUEでパスを指定します。Solaris/LinuxシステムをWebサーバとして使用する場合は、パスを設定する代わりにリンクファイルを作成して代行することもできます。
- <PARAM NAME>タグで指定するPORB_HOMEには、動作環境ファイルを検索する際のサブディレクトリ名を指定します。上記の例の動作環境ファイル格納ディレクトリは、以下です。パスの指定については、“5.6.5 Portable-ORB動作環境ファイルの指定”を参照してください。

```
Webサーバのドキュメントルートディレクトリ/PORBDIR/etc
```

5.2.2 アプレットのプログラミング

サーバ上のCORBA準拠のアプリケーションと連携するJavaアプレットでは、その先頭でimport文を使用してorg.omg.CORBAクラスを参照することを宣言します。また、HTMLファイルの<applet>タグで記述したクラス名でクラス宣言します。

```
import org.omg.CORBA.*;                          // org.omg.CORBAクラス
import java.awt.*;                               // 抽象ウィンドウツールキットクラス

public class Sample extends java.applet.Applet // appletクラスの宣言
{
    ...                                          // Java言語による処理の記述
}
```

注意

Javaアプレットのファイル名は、アプレット内で宣言したクラス名と同じ(大文字／小文字の区別あり)にする必要があります。

5.2.3 IDL生成ファイルとアプレットのコンパイル

IDLコンパイラを実行し、Javaアプレット用のスタブを生成します。また、生成されたスタブとJavaアプレットをコンパイルします。コンパイルには、本製品が提供しているjavacコマンドを使用します。

1. インタフェース(xxx.java,xxxOperations.java)
2. Helperクラス(xxxHelper.java)
3. Holderクラス(xxxHolder.java)
4. スタブクラス(_xxxStub.java)
5. ユーザの作成したJavaアプレット

注意事項

コンパイルの実行時、環境変数CLASSPATHにカレントディレクトリ(.)およびJavaライブラリを指定する必要があります。なお、運用形態により環境変数CLASSPATHに設定するJavaライブラリが異なります。Javaライブラリについては、“B.2 Portable-ORBの提供プログラム”を参照してください。
設定例を以下に示します。

例

Windows32/64 (PORB_HOME:C:¥Interstage¥PORB)

```
set CLASSPATH=.;%PORB_HOME%¥lib¥ODporb4.jar;%PORB_HOME%¥lib¥CosNaming4.jar;
    %PORB_HOME%¥lib¥InterfaceRep4.jar;%CLASSPATH%
```

Solaris32/64 Linux32/64 (PORB_HOME:/opt/FJSVporb)

```
CLASSPATH=.:$PORB_HOME/lib/ODporb4.jar:$PORB_HOME/lib/CosNaming4.jar:
    $PORB_HOME/lib/InterfaceRep4.jar:$CLASSPATH
export CLASSPATH
```

5.2.4 Javaクラスファイルのアーカイブ

JavaクラスファイルをWebサーバに登録する場合、ダウンロード時間を短縮するため、複数のファイルを一度にダウンロードし、クラスファイルをまとめたアーカイブファイルを作成します。アーカイブファイルは、Java Development Kit(以降JDK)に含まれるjarコマンドを使用して作成します。
ここで作成したJavaクラスファイルのアーカイブファイル(アプレット)をWebサーバからダウンロードして実行する場合は、アーカイブファイルに対して署名を行う必要があります。署名については、“5.8 アプレットのデジタル署名”を参照してください。

参照

jarコマンドの使用方法などについては、JDKのドキュメントを参照してください。

サブディレクトリを含むクラスファイルからアーカイブファイルを作成する場合のコマンド使用例を以下に示します。

■jarコマンド使用例

作成するアーカイブファイル名とアーカイブするクラスファイルを指定します。作成されたアーカイブファイルには、サブディレクトリ内のファイルが含まれています。

```
jar cvf Sample.jar *.class Samplemod¥*.class
    adding: Samplemod/_SampleintfStub.class (in=1282) (out=704) (deflated 45%)
    adding: Samplemod/Sampleintf.class (in=302) (out=215) (deflated 28%)
    adding: Samplemod/SampleintfHelper.class (in=2175) (out=994) (deflated 54%)
    adding: Samplemod/SampleintfHolder.class (in=907) (out=461) (deflated 49%)
```

注意

- アーカイブファイルとするクラスファイルを指定する場合、パス指定の情報もアーカイブファイル内に含まれます。カレントディレクトリをアーカイブファイルとするクラスファイルが格納される最上位ディレクトリに移動後、クラスファイルの構成に従って指定し、アーカイブファイルを作成してください。
- HTMLファイル内の<PARAM>タグでcabbaseを指定する場合、<APPLET>タグのARCHIVE指定は無効です。

■HTMLファイル記述例

JavaアプレットがSampleとIDL生成ファイルをSample.jarとしてアーカイブした場合のHTMLファイル記述例を、“5.2.1 HTMLファイルの記述”に記述されているHTMLファイルを例にして以下に示します。

(1)JBKプラグインを使用する場合(jar形式アーカイブファイル使用)

<PARAM>タグのARCHIVE指定、および<EMBED>タグのARCHIVE指定で、Sample.jarをダウンロードするように指定します。

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="NAME" VALUE="Sample">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="Sample.jar,ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar">
<PARAM NAME="PORB_HOME" VALUE="PORBDIR">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
       NAME="Sample" CODE="Sample.class" WIDTH=300 HEIGHT=250
       ARCHIVE="Sample.jar,ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar"
       PORB_HOME="PORBDIR">
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

(2)Portable-ORBをダウンロードしない場合

上記の(1)および(2)のHTMLの記述例からPortable-ORBのファイルの記述を削除したものがPortable-ORBをダウンロードしない場合の記述となります。記述を削除するPortable-ORBのファイルは、“5.2.1 HTMLファイルの記述”に説明されている表を参照してください。

5.2.5 Webサーバへの Javaアプレットの登録

HTMLファイル、Javaアプレット、IDL生成ファイルクラスをWebサーバ上の同じディレクトリに格納します。

5.3 サーバアプリケーションの開発手順

ここでは、静的スケルトンインタフェースを使用する場合を例として、サーバアプリケーションの開発手順の概要について説明します。また、サーバアプリケーションの開発と実行に必要な環境設定について説明します。

5.3.1 IDLファイルの作成とコンパイル

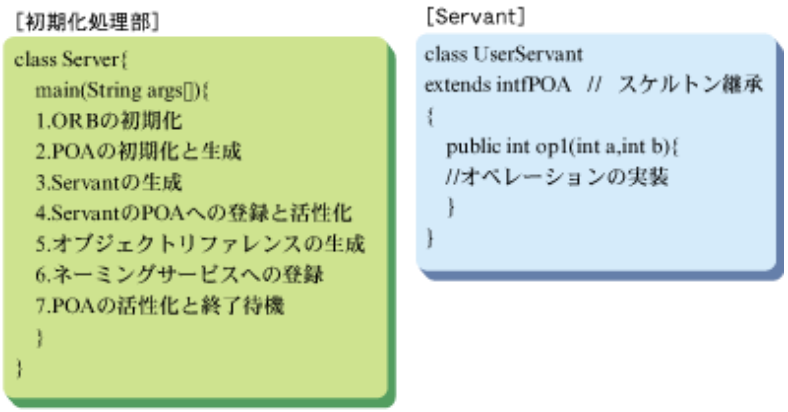
1. サーバアプリケーションが提供するインタフェースを定義するために、IDLファイルを作成します。
2. IDLcコマンドでIDLファイルをコンパイルします。この結果、IDL定義をjavaの記述にマッピングした、複数の*.javaファイルが出力されます。
   IDLcコマンドにより、以下のファイルが出力されます。

<IDL定義のinterface名>Operations.java <IDL定義のinterface名>.java	インタフェース
<IDL定義のinterface名>POA.java	スケルトンクラス
<IDL定義のinterface名>POATie.java	tieクラス(代理方式で使用)(注1)
<xxx>Holder.java	Holderクラス
<xxx>Helper.java	Helperクラス
_<IDL定義のinterface名>Stub.java	スタブクラス(注2)

注1)代理方式については、“5.17 サーバアプリケーションの実装アプローチ”を参照してください。
注2)サーバアプリケーションでも使用する場合があります。

5.3.2 サーバアプリケーションの作成

サーバアプリケーションは「初期化処理部」と「インタフェース実装部(Servant)」に分けられます。それぞれの部分は以下のような機能を実装します。詳細については、“5.15 サーバアプリケーションのプログラミング(静的スケルトンインタフェース)”を参照してください。



（注：5 6は　OD_or_admコマンドを使用した場合不要）

5.3.3 Javaファイルのコンパイル

IDLコンパイラにより生成された*.javaファイルおよびユーザが作成したサーバアプリケーションを合わせてjavacコマンドでコンパイルします。

注意

コンパイル時、環境変数CLASSPATHにJavaライブラリを指定する必要があります。Javaライブラリについては、“B.1 CORBAサービスの提供プログラム”－“B.1.2 ライブラリ”を参照してください。

設定例については、“5.1.3 IDL生成ファイルとアプレットのコンパイル”の“注意事項”を参照してください。

5.3.4 起動用ファイルの作成

サーバアプリケーションを起動するには、通常バッチファイルやシェルスクリプトを用意します。起動時に必要な環境変数OD_IMPLIDへのインプリメンテーションリポジトリIDの設定も、起動用ファイルに設定します。ただし、POAオブジェクトのアダプタ名にインプリメンテーションリポジトリIDを指定する場合は、環境変数OD_IMPLIDを設定する必要はありません。詳細については、“5.14 サーバアプリケーションと環境設定との関連付け”を参照してください。

なお、Javaアプリケーションの実行時は、環境変数CLASSPATHにJavaのライブラリを設定する必要があります。
環境変数CLASSPATHの設定と起動用ファイルの例を以下に示します。

例

Windows32/64 (OD_HOME: C:¥Interstage¥ODWIN)

```
set CLASSPATH=.;%OD_HOME%¥etc¥class¥ODjava4.jar;%CLASSPATH%
```

【起動用バッチファイル例】

```
set OD_IMPLID=IDL:ODdemo/calculator:1.0  インプリメンテーションリポジトリIDを設定
                                        必要に応じて環境変数の設定などを記述
set CLASSPATH=<Server.classのパス>;%CLASSPATH%;
java Server                             サーバクラスを引数に、Java VM起動
```

Solaris32/64 (OD_HOME: /opt/FSUNod) Linux32/64 (OD_HOME: /opt/FJSVod)
環境変数CLASSPATHの他に、環境変数LD_LIBRARY_PATHに“$OD_HOME/lib”を設定する必要があります。

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export CLASSPATH
LD_LIBRARY_PATH=$OD_HOME/lib
export LD_LIBRARY_PATH
```

【起動用シェルスクリプト例】

```
OD_IMPLID= IDL:ODdemo/calculator:1.0  インプリメンテーションリポジトリIDを設定
export OD_IMPLID
                                      必要に応じて環境変数の設定などを記述
CLASSPATH=<Server.classのパス>:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH:
export CLASSPATH
LD_LIBRARY_PATH=$OD_HOME/lib
export LD_LIBRARY_PATH
java Server                           サーバクラスを引数に、Java VM起動
```

5.3.5 インプリメンテーションリポジトリの登録

インプリメンテーションリポジトリにサーバアプリケーション情報を登録するために、OD_impl_instコマンドを使用します。OD_impl_instコマンドでの登録処理の例と指定する情報について以下に示します。

例

```
OD_impl_inst -ax def
```

-ax def

定義ファイルで指定した定義情報でサーバアプリケーション情報を登録することを示します。

defファイルの内容

```
rep_id        = IDL:ODdemo/calculator:1.0
type          = persistent
mode          = SYNC_END
iswitch       = ON
proc_conc_max = 1
thr_conc_init = 1
ior           = 1.1
locale        = UNICODE
```

rep_id = IDL:ODdemo/calculator:1.0

インプリメンテーションリポジトリIDを指定します。

type = persistent

サーバタイプを指定します。Javaサーバアプリケーションの場合は、persistentを指定します。

mode = SYNC_END

リクエスト受付準備完了通知のモードです。Javaサーバアプリケーションの場合は、SYNC_ENDを指定します。

iswitch = ON

サーバアプリケーションがクライアントアプリケーションごとにインスタンスデータを保持するかどうかを指定します(ON／OFF)。
アプリケーション形態としてFactory方式およびユーザインスタンス管理方式をとる場合は、ONを指定してください。
アプリケーション形態については、“5.12.1 アプリケーション形態の種別”を参照してください。

proc_conc_max = 1

プロセス最大多重度を指定します。

thr_conc_init = 1

スレッド初期多重度を指定します(省略時:16)。

ior = 1.1

連携するIORのバージョンを指定します。locale = UNICODEを指定する場合は、1.1を指定してください(省略時:1.0)。

locale = UNICODE

サーバアプリケーションのコード系を指定します。日本語データを扱う場合は、UNICODEを指定してください。その他のコード系は、指定できません。
なお、使用するJDKのバージョンによって、取り扱うUNICODEのバージョンが異なる場合があるため、注意してください。

5.3.6 セキュリティーマネージャーの設定 Solaris64 Linux32/64

セキュリティーマネージャーを使用する場合は、以下のどちらかを設定します。

■システムプロパティの設定

システムプロパティに、以下を設定します。

```
jdk.corba.allowOutputStreamSubclass=true
jdk.corba.allowInputStreamSubclass=true
```

■Javaライブラリの権限の設定

以下のJavaライブラリに、SerializablePermission enableSubclassImplementation権限を設定します。

- Solaris64
  /opt/FSUNod/etc/class/ODjava4.jar
- Linux32/64
  /opt/FJSVod/etc/class/ODjava4.jar
- ユーザアプリケーションのクラスおよびライブラリ

注意

Linux32/64
RHEL5の場合、設定する必要はありません。

5.4 CORBAアプリケーションの実行

ここでは、CORBAアプリケーションの実行について説明します。

5.4.1 Java実行環境

CORBAアプリケーションの実行環境として提供しているJava実行環境について説明します。

JBKプラグインおよびStudioのバージョンごとに添付されているJDK/JREの動作可能な組合せを以下に示します。

JDK/JREのバージョン	JBKプラグイン	Studioのバージョン		
		V9.2	V10	V11以降
JDK/JRE 7	○	－	－	○
JDK/JRE 6	○	○	○	○

○: サポート可
ー: 未サポート

5.4.2 ORB(Object Request Broker)の指定

アプリケーション(またはアプレット)を起動するための環境設定として、使用するORBを選択します。
CORBA規約では、システム上に存在する複数のORB(Object Request Broker)から、使用するORBを選択する方法が規約化されています。

ここでは、この仕様に基づいて、使用するORBとしてCORBAサービス(ObjectDirector)を指定する方法について説明します(クライアント/サーバ共通)。

注意

ORBは、必ず指定してください。なお、Portable-ORBのアプレット運用を行う場合は、プロパティ名"org.omg.CORBA.ORBSingletonClass"を指定しないでください。

■ORBの指定方法の種類

ORBの指定方法として以下の方法があります。以下のどちらかの方法で、使用するORBを指定してください。

- アプリケーション起動時に指定する方法
- 環境設定ファイルを用意する方法

どちらの方法も、Java実行環境の「プロパティ情報」として以下の値を設定することにより、CORBAサービスのORBを指定します。

プロパティ名	設定値(ObjectDirectorを使用)
org.omg.CORBA.ORBClass	com.fujitsu.ObjectDirector.CORBA.ORB
org.omg.CORBA.ORBSingletonClass	com.fujitsu.ObjectDirector.CORBA.SingletonORB

それぞれの指定方法について、以下に説明します。

(1)アプリケーション起動時に指定する方法

Javaアプリケーション実行時に、javaコマンドのパラメタとしてプロパティ情報を設定します。以下のように-Dオプションに続けて必要な情報を記述します。バッチファイルおよびシェルスクリプトを使用してアプリケーションを起動する場合は、これらのファイル内に以下を指定してください。

例

プロパティ情報の指定

```
java -Dorg.omg.CORBA.ORBClass=com.fujitsu.ObjectDirector.CORBA.ORB
     -Dorg.omg.CORBA.ORBSingletonClass=com.fujitsu.ObjectDirector.CORBA.SingletonORB
     〈アプリケーションのクラス名〉
```

注) -Dの直後に、空白文字を入れないでください。

(2)環境設定ファイルを用意する方法

「プロパティ情報」を記述したテキストファイル(ファイル名:orb.properties)を作成し、インストールパッケージおよび使用するJDK/JREにより以下のディレクトリに格納します。
Javaアプレットの場合は、実行するマシン環境で設定してください。

Windows32/64

【Interstage Studioクライアント運用パッケージの場合】

```
JRE 7系使用時（注1）:
<Studioのインストールパス>¥jre7¥lib
JRE 6系使用時:
<Studioのインストールパス>¥jre6¥lib
```

注1)Interstage Studioクライアント運用パッケージ V11.0以降の場合

【Interstage Studio Standard-J Editionの場合】

```
JRE 7系使用時（注2）:
<Studioのインストールパス>¥JDK7¥jre¥lib
JRE 6系使用時:
<Studioのインストールパス>¥JDK6¥jre¥lib
```

注2)Interstage Studio Standard-J Edition V11.0以降の場合

【サーバパッケージ内の“JDK/JRE”の場合】

```
JRE 7系使用時:
C:¥Interstage¥jre7¥lib
JRE 6系使用時:
C:¥Interstage¥jre6¥lib
JDK 7系使用時:
C:¥Interstage¥jdk7¥jre¥lib
JDK 6系使用時:
C:¥Interstage¥jdk6¥jre¥lib
```

【クライアントパッケージ内の“JBKプラグイン”の場合】

```
JRE 7系使用時:
C:¥Interstage¥JBKDI¥jre7¥lib
JRE 6系使用時:
C:¥Interstage¥JBKDI¥jre6¥lib
JDK 7系使用時:
C:¥Interstage¥JBKDI¥jdk7¥jre¥lib
JDK 6系使用時:
C:¥Interstage¥JBKDI¥jdk6¥jre¥lib
```

Solaris32/64 Linux32/64

```
JRE 7系使用時:
<Java実行環境のインストールパス>/jre7/lib
JRE 6系使用時:
<Java実行環境のインストールパス>/jre6/lib
JDK 7系使用時:
<Java実行環境のインストールパス>/jdk7/jre/lib
```

```
JDK 6系使用時：
<Java実行環境のインストールパス>/jdk6/jre/lib
```

【ファイルorb.propertiesの内容】

```
org.omg.CORBA.ORBClass=com.fujitsu.ObjectDirector.CORBA.ORB
org.omg.CORBA.ORBSingletonClass=com.fujitsu.ObjectDirector.CORBA.SingletonORB
```

【Portable-ORBアプレット運用時のファイルorb.propertiesの内容】

```
org.omg.CORBA.ORBClass=com.fujitsu.ObjectDirector.CORBA.ORB
```

5.5 クライアントの環境設定（プレインストール型Javaライブラリ）

プレインストール型Javaライブラリ使用時のクライアントの環境設定について、以下に説明します。
クライアントの環境設定では、CORBAアプリケーションの実行環境に応じたアプレットのコンパイル環境の設定およびHTMLの記述を行います。それぞれの設定方法および記述方法を以降に示します。クライアントアプリケーションのコンパイル／実行時に使用するJavaライブラリについては、“B.1 CORBAサービスの提供プログラム”－“B.1.2 ライブラリ”を参照してください。

参照

Solaris32/64
プレインストール型Javaクライアントを動作させる場合は、“5.8 アプレットのデジタル署名”を参照してください。

5.5.1 HTMLファイルの記述

CORBAアプリケーション実行環境としてJBKプラグイン単位でJava VMを切り替えるには、アプレットのパラメタ“@JBKPLGPROP”を使用します。HTMLファイルの記述については、“5.1.1 HTMLファイルの記述”を参照してください。また、パラメタなどの詳細については、Studioの“J Business Kit オンラインマニュアル”を参照してください。
なお、アプレットに対するデジタル署名が必要です。デジタル署名の方法については、“5.8 アプレットのデジタル署名”を参照してください。

注意

HTML文書にプレインストール型Javaクライアントが提供するJavaライブラリを記述する必要はありません。

5.5.2 Javaライブラリに対する権限の設定

Javaアプレットを実行する場合は、Javaライブラリに対する権限を設定します。
policytool（JDK添付）によるJavaライブラリに対する権限の設定方法を以下に示します。

1. policytoolを起動します。

```
policytool
```

2. 起動後、最初に表示される[Policy Tool]の画面上で[ポリシーエントリの追加]ボタンを押下します。

3. 表示された[ポリシーエントリ]の画面上で、各項目の値を以下のように設定します。

項目	設定値
CodeBase:	file:<CORBAサービスクライアントインストールディレクトリ>/etc/class/ODjava4.jar (注1)
signed by:	(なし)

注1) 区切り文字として“/”を使用してください。

4. [ポリシーエントリ]の画面上で[アクセス権の追加]ボタンを押下します。

5. 表示された[アクセス権]の画面上で、各項目に以下の表の値を設定します。
一組の[アクセス権:／ターゲット名:／アクション:]の値を設定し、[了解]ボタンを押下して権限を設定します。[了解]ボタンを押下すると、[ポリシーエントリ]の画面に戻ります。次の一組の値を設定するには、再度、[アクセス権の追加]ボタンを押下します。これを繰り返し、必要な情報を設定します。
設定後は、[ポリシーエントリ]の画面上で[完了]ボタンを押下します。

◇通常運用で必要な権限

通常の運用では、セキュリティ上の安全性を確保するため、本権限以外は設定しないでください。

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
ランタイム権限	RuntimePermission (java.lang.RuntimePermission)	loadLibrary.DLL名 (注2)	設定不要
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	com.fujitsu.*	read

注2) インストールしている機能により以下のダイナミックリンクライブラリ(DLL)名を指定します。拡張子は、指定不要です。

インストールしている機能	ダイナミックリンクライブラリ
CORBAサービスクライアント(クライアント機能)	ODjava4
CORBAサービス(サーバ機能)	ODjavas4

◇CORBAサービスの内部ログを採取する場合に必要な権限 (注3)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	user.dir	read
		java.class.path	read
ファイル権限	FilePermission (java.io.FilePermission)	${user.dir}¥*	read, write
		%OD_HOME%¥etc¥config (注4)	read

注3) CORBAサービスの内部ログの詳細については、“チューニングガイド”の“config”を参照してください。なお、ログ採取後は追加した権限を削除してください。
注4) %OD_HOME%には、CORBAサービス、CORBAサービスクライアントのインストールパス(デフォルト:C:¥Interstage¥ODWIN)を指定します。

6. [Policy Tool]の画面上で、メニュー[ファイル]→[保存]を選択します。

7. メニュー[ファイル]→[終了]を選択し、policytoolを終了します。

注意

policytoolの初回実行時は、policytool終了前に、[Policy Tool]の画面からメニュー[ファイル]→[別名保存]を選択し、ポリシーファイルの名前と格納位置を指定する必要があります。ポリシーファイル指定の詳細については、“5.8.3 デジタル署名手順”を参照してください。

5.5.3 ライブラリの設定 Solaris32/64 Linux32/64

プレインストール型Javaライブラリを動作させる場合は、環境変数LD_LIBRARY_PATHに<CORBAサービスのインストールディレクトリ>/libを設定します。

5.5.4 セキュリティーマネージャーの設定 Solaris64 Linux32/64

セキュリティーマネージャーを使用する場合は、以下のどちらかを設定します。

■システムプロパティの設定

システムプロパティに、以下を設定します。

```
jdk.corba.allowOutputStreamSubclass=true
jdk.corba.allowInputStreamSubclass=true
```

■Javaライブラリの権限の設定

以下のJavaライブラリに、SerializablePermission enableSubclassImplementation権限を設定します。

- Solaris64
  /opt/FSUNod/etc/class/ODjava4.jar
- Linux32/64
  /opt/FJSVod/etc/class/ODjava4.jar
- ユーザアプリケーションのクラスおよびライブラリ

注意

Linux32/64
RHEL5の場合、設定する必要はありません。

5.6 クライアントの環境設定（Portable-ORB）

Portable-ORB使用時のクライアントの環境設定について、以下に説明します。
クライアント環境設定では、CORBAアプリケーションの実行環境に応じたアプレットのコンパイル環境の設定およびHTMLの記述を行います。それぞれの設定方法および記述方法を以降に示します。クライアントアプリケーションのコンパイル／実行時に使用するJavaライブラリについては、“B.2 Portable-ORBの提供プログラム”を参照してください。

5.6.1 HTMLファイルの記述

CORBAアプリケーション実行環境としてJBKプラグイン単位でJava VMを切り替えるには、アプレットのパラメタ“@JBKPLGPROP”を使用します。HTMLファイルの記述については、“5.2.1 HTMLファイルの記述”を参照してください。また、パラメタなどの詳細については、Studioの“J Business Kit オンラインマニュアル”を参照してください。

なお、アプレットに対するデジタル署名が必要です。デジタル署名の方法については、“5.8 アプレットのデジタル署名”を参照してください。

5.6.2 Javaライブラリに対する権限の設定

Portable-ORBをダウンロードしないで（クライアントにあらかじめ配置して）Javaアプレットを実行する場合は、Javaライブラリに対する権限を設定します。
policytool（JDK添付）によるJavaライブラリに対する権限の設定方法を以下に示します。

1. policytoolを起動します。

```
policytool
```

2. 起動後、最初に表示される[Policy Tool]の画面上で[ポリシーエントリの追加]ボタンを押下します。
3. 表示された[ポリシーエントリ]の画面上で、各項目の値を以下のように設定します。

項目	設定値
CodeBase:	file:<Portable-ORB Jarファイル格納ディレクトリ>/* (注1)
signed by:	(なし)

注1) 区切り文字として'/'を使用してください。

4. [ポリシーエントリ]の画面上で[アクセス権の追加]ボタンを押下します。
5. 表示された[アクセス権]の画面上で、各項目の値を以下の表のように設定します。
一組の[アクセス権:／ターゲット名:／アクション:]の値を設定し、[了解]ボタンを押下して権限を設定します。[了解]ボタンを押下すると、[ポリシーエントリ]の画面に戻ります。次の一組の値を設定するには、再度、[アクセス権の追加]ボタンを押下します。これを繰り返し、必要な情報を設定します。
設定後は、[ポリシーエントリ]の画面上で[完了]ボタンを押下します。

◇通常運用で必要な権限

通常の運用では、セキュリティ上の安全性を確保するため、本権限以外は設定しないでください。

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
通信権限	SocketPermission (java.net.SocketPermission)	通信先サーバ名 (注2)	connect

注2) Javaアプレットから通信する通信先サーバ分の通信先サーバ名を設定します。
通信先サーバ名として、以下のホスト名を指定します。

- porbeditenvコマンドの「ホスト情報」で設定したホスト名
  詳細については、“リファレンスマニュアル（コマンド編）”の“porbeditenv”を参照してください。
- 通信するサーバアプリケーションのオブジェクトリファレンスに設定されているホスト名
  odlistnsコマンド(-lオプション指定)で表示される「オブジェクトのホスト名」です。
- URLスキーマに指定したホスト名
  詳細については、“9.5 URLスキーマ”および“13.4 corbaloc URLスキーマ”を参照してください。

なお、通信先サーバとして、serverA.interstage.co.jpとserverB.interstage.co.jpが存在する場合は、通信先サーバ名としてserverA.interstage.co.jpとserverB.interstage.co.jpの2組の通信権限を設定します。または、信頼できる範囲を確認した上でワイルドカード(*)を使用して、*.interstage.co.jpと設定することもできます。ただし、ワイルドカード(*)のみの指定（不特定のサーバマシンとの通信）は、セキュリティ上の安全を確保するため、指定しないでください。

◇EJBアプリケーションを使用する場合に必要な権限 (注3)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	com.fujitsu.*	read
		java.class.path	read
ランタイム権限	RuntimePermission (java.lang.RuntimePermission)	getClassLoader	(指定不要)

注3) EJBアプリケーションの詳細については、“J2EE ユーザーズガイド”の“EJB編”を参照してください。

◇Portable-ORBのログ情報を採取する場合に必要な権限 (注4)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
ファイル権限	FilePermission (java.io.FilePermission)	ログ採取ファイル (注5)	read, write, delete
		ログ採取ディレクトリ (注6)	read

注4) Portable-ORBのログ情報の詳細については、“リファレンスマニュアル(コマンド編)”の“porbeditenv”を参照してください。
なお、ログを採取する場合は、porbeditenvコマンドの「ログ格納ディレクトリ」で指定したディレクトリを事前に作成しておく必要があります。また、「ログ格納ディレクトリ」にはログ採取ファイル以外のユーザ資源などを格納しないでください。ログ採取後は、追加した権限を削除してください。
注5) porbeditenvコマンドの「ログ格納ディレクトリ」で指定したディレクトリ名に、「¥*」を付加したパスを指定します。「ログ格納ディレクトリ」に「c:¥log¥porb」と指定した場合は、「c:¥log¥porb¥*」です。
注6) porbeditenvコマンドの「ログ格納ディレクトリ」で指定したディレクトリ名を指定します。

◇SSL連携機能を使用する場合に必要な権限 (注7)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
ファイル権限	FilePermission (java.io.FilePermission)	keystoreディレクトリ (注8)	read

注7) SSL連携機能の詳細については、“セキュリティシステム運用ガイド”の“Portable-ORBでSSLを利用する方法”を参照してください。
注8) porbeditenvコマンドの「キーストア格納位置」で指定した格納位置、または-ORB_FJ_PORB_SSLPathパラメタで指定した格納位置を指定します。格納位置が「C:¥Interstage¥PORB¥etc¥keystore」の場合は、「C:¥Interstage¥PORB¥etc¥keystore」です。キーストアの格納位置がネットワーク上(HTTP指定)である場合、指定する必要はありません。

◇サーバアプリケーションでユーザ情報獲得用のAPIを使用する場合に必要な権限 (注9)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	user.name	read

注9)ユーザ情報獲得用のAPIの詳細については、“リファレンスマニュアル(API編)”の“TD_get_user_information”／“TD::get_user_information”／“TDGETUSERINFORMATION”を参照してください。

6. [Policy Tool]の画面上で、メニュー[ファイル]→[保存]を選択します。
7. メニュー[ファイル]→[終了]を選択し、policytoolを終了します。

注意

policytoolの初回実行時は、policytool終了前に、[Policy Tool]の画面からメニュー[ファイル]→[別名保存]を選択し、ポリシーファイルの名前と格納位置を指定する必要があります。ポリシーファイル指定の詳細については、“5.8.3 デジタル署名手順”を参照してください。

5.6.3 Javaライブラリに対する署名の設定

Portable-ORBをダウンロードしてJavaアプレットを実行する場合は、Portable-ORBのJavaライブラリに対してデジタル署名の設定、およびJavaアプレットを実行するクライアントマシンで署名されたJavaライブラリに対する権限を設定します。
“5.8 アプレットのデジタル署名”を参照し、署名および権限を設定してください。
なお、このとき、ダウンロードして実行するアプレットの署名に使用した証明書を使用して、Portable-ORBのJavaライブラリに対してデジタル署名を行ってください。

5.6.4 ライブラリの設定

(1) Portable-ORBをダウンロードする運用形態

Portable-ORBをダウンロードする運用形態では、Portable-ORBをダウンロードするWebサーバ上に署名済みのJavaライブラリを格納します。
HTMLファイルの記述例(タイトル: Java sample Applet、Javaアプレット名: Sample)を以下に示します。

以下のHTML文書例の場合は、このHTMLファイルが格納されているディレクトリにJavaライブラリを格納します。

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="NAME" VALUE="Sample">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="Sample.jar,ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar">
<PARAM NAME="PORB_HOME" VALUE="PORBDIR">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
       NAME="Sample" CODE="Sample.class" WIDTH=300 HEIGHT=250
       ARCHIVE="Sample.jar,ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar"
       PORB_HOME="PORBDIR">
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

(2) Portable-ORBをダウンロードしない運用形態

Portable-ORBをダウンロードしない運用形態では、Portable-ORBを動作する(Portable-ORBをインストールした)クライアントで環境変数CLASSPATHにJavaライブラリを設定します。プレインストール型Javaライブラリ、およびPortable-ORBをダウンロードする形態では、本設定を行う必要はありません。なお、運用形態により環境変数CLASSPATHに設定するJavaライブラリが異なります。Javaライブラリについては、“B.2 Portable-ORBの提供プログラム”を参照してください。
設定例を以下に示します。

例

Windows32/64 (PORB_HOME:C:¥Interstage¥PORB、インストールパスはデフォルト)

```
set CLASSPATH=.;%PORB_HOME%¥lib¥ODporb4.jar;%PORB_HOME%¥lib¥CosNaming4.jar;
    %PORB_HOME%¥lib¥InterfaceRep4.jar;%CLASSPATH%
```

Solaris32/64 (PORB_HOME:/opt/FJSVporb、インストールパスはデフォルト)

```
CLASSPATH=.:$PORB_HOME/lib/ODporb4.jar:$PORB_HOME/lib/CosNaming4.jar:
    $PORB_HOME/lib/InterfaceRep4.jar:$CLASSPATH
export CLASSPATH
```

Linux32/64 (PORB_HOME:/opt/FJSVporb)

```
CLASSPATH=.:$PORB_HOME/lib/ODporb4.jar:$PORB_HOME/lib/CosNaming4.jar:
    $PORB_HOME/lib/InterfaceRep4.jar:$CLASSPATH
export CLASSPATH
```

注意

環境変数CLASSPATHに他のORBのJavaライブラリ(プレインストール型Javaライブラリを含む)が設定されている場合は、そのORBが動作する可能性があります。Portable-ORB以外のORBのJavaライブラリは、環境変数CLASSPATHに設定しないでください。

5.6.5 Portable-ORB動作環境ファイルの指定

Portable-ORBを使用する場合、動作環境ファイルの格納位置を指定するため、HTMLファイルでPORB_HOMEパラメタを設定します。Portable-ORBには、以下の動作環境ファイルがあります。

動作環境ファイル	ファイル名
環境定義ファイル	config
オブジェクトリファレンス情報格納ファイル	initial_services
オブジェクトリファレンス検索情報ファイル	initial_hosts

これらの動作環境ファイルは、Webサーバのドキュメントルート配下に格納する必要があります。

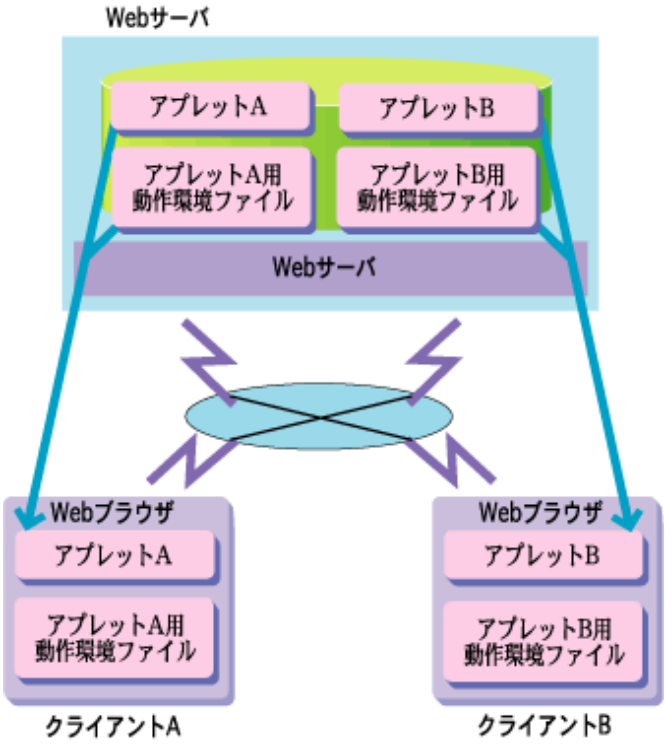
注意

Webサーバでユーザ名およびパスワードによる認証を行う場合、ユーザ認証ディレクトリ配下に動作環境ファイルを格納しないでください。

動作環境ファイルとして、アプレットごとに異なる動作環境ファイルを指定する方法、および複数のアプレットで同一の動作環境ファイルを指定する方法があります。それぞれの場合の動作環境ファイルの使用イメージ・指定方法を以下に示します。

■アプレットごとに異なる動作環境を使用する場合

動作環境ファイルの使用イメージを以下に示します。



クライアントAで動作するアプレットAはアプレットA用の動作環境ファイルを、クライアントBで動作するアプレットBはアプレットB用の動作環境ファイルを使用します。
アプレット単位に動作環境ファイルを指定する方法は、PORB_HOMEを指定する方法、およびPORB_HOMEを指定しない方法の2種類があります。

PORB_HOMEを指定する場合

PORB_HOMEを指定する場合、各アプレットの動作環境ファイルはetcディレクトリを作成してその配下に格納します。動作環境ファイルの格納位置として、HTMLファイルの<PARAM>タグで、PORB_HOMEパラメタにWebサーバのドキュメントルートからの相対パスを指定します。この指定では、etcディレクトリは含めません。
ディレクトリ構成例を以下に示します。Webサーバのドキュメントルートディレクトリが/WWWで、アプレットAの動作環境ファイルはaplAenv/etc配下、アプレットBの動作環境ファイルはaplBenv/etc配下に格納されています。

```
/-  ...
 - [WWW] - [envfile] - [aplAenv] - [etc] - アプレットA用動作環境ファイル
                     - [aplBenv] - [etc] - アプレットB用動作環境ファイル
         - [applet ] - [appletA] - アプレットA.html
                                 - アプレットA.class
                       [appletB] - アプレットB.html
                                 - アプレットB.class
```

上記の例で、それぞれのアプレットのPORB_HOMEパラメタは<PARAM>タグで以下のように指定します。

【アプレットAの<PARAM>タグ】

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<H1>Java sample Applet</H1>
<applet code="アプレットA.class" width=300 height=250>
<PARAM NAME=PORB_HOME VALUE=envfile/aplAenv>
</applet><BR>
</BODY>
</HTML>
```

【アプレットBの<PARAM>タグ】

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<H1>Java sample Applet</H1>
<applet code="アプレットB.class" width=300 height=250>
<PARAM NAME=PORB_HOME VALUE=envfile/aplBenv>
</applet><BR>
</BODY>
</HTML>
```

PORB_HOMEを指定しない場合

PORB_HOMEを指定しない場合、各アプレットを格納したディレクトリにetcディレクトリを作成し、動作環境ファイルを格納します。
ディレクトリ構成例を以下に示します。

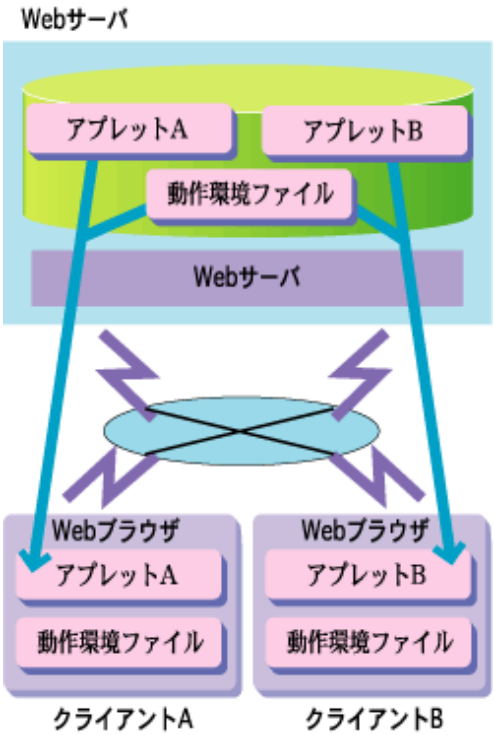
```
/- ...
 - [WWW] - [applet  ] - [appletA] - アプレットA.html
                                  - アプレットA.class
                                  - [etc] - アプレットA用動作環境ファイル
                        [appletB] - アプレットB.html
                                  - アプレットB.class
                                  - [etc] - アプレットB用動作環境ファイル
```

ポイント

fileプロトコルを使用して動作させる場合は、“PORB_HOMEを指定しない場合”の方法で動作させます。ブラウザ、およびJDKのツールであるappletviewerを使用し、Webサーバを使用せずにローカルディスクに格納されたHTMLファイルを直接指定した場合は、fileプロトコルとなります。

■複数のアプレットで同一の動作環境を使用する場合

動作環境ファイルの使用イメージを以下に示します。



クライアントAで動作するアプレットA、クライアントBで動作するアプレットBで共通の動作環境ファイルを使用します。
動作環境ファイルの格納位置を指定するには、アプレットAのHTMLファイルおよびアプレットBのHTMLファイルの<PARAM>タグで、PORB_HOMEパラメタにWebサーバのドキュメントルートからの相対パスを設定します。この指定では、etcディレクトリは含めません。
ディレクトリの構成例を以下に示します。/WWW(Webサーバのドキュメントルートディレクトリ)配下のporbディレクトリにPortable-ORBがインストールされ、その配下のetcディレクトリに動作環境ファイルが格納されています。

```
/- ...
 - [WWW] - [porb   ] - [lib] - ODporb4.jar...
                     - [etc] - 動作環境ファイル
         - [applet ] - [appletA] - アプレットA.html
                                 - アプレットA.class
                       [appletB] - アプレットB.html
                                 - アプレットB.class
```

アプレットAのHTMLファイル、およびアプレットBのHTMLファイルに/WWW/porb/etc配下の動作環境ファイルを使用するように/WWWから/WWW/porbまでのパスを指定します。

それぞれの<PARAM>タグは、以下のようになります。

【アプレットAの<PARAM>記述】

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<H1>Java sample Applet</H1>
<applet code="アプレットA.class" width=300 height=250>
<PARAM NAME=PORB_HOME VALUE=porb>
</applet><BR>
```

```
</BODY>
</HTML>
```

【アプレットBの<PARAM>記述】

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<H1>Java sample Applet</H1>
<applet code="アプレットB.class" width=300 height=250>
<PARAM NAME=PORB_HOME VALUE=porb>
</applet><BR>
</BODY>
</HTML>
```

注意

- 動作環境ファイルは、環境設定コマンド(porbeditenv)を使用して作成／更新を行います。
- 動作環境ファイルを格納するetcディレクトリを作成する場合は、必ず英小文字で作成してください。

5.7 サーバの環境設定

サーバの環境設定について、以下に説明します。
サーバアプリケーションのコンパイル／実行時に使用するJavaライブラリについては、“B.1 CORBAサービスの提供プログラム”－“B.1.2 ライブラリ”を参照してください。

例

■環境変数の設定例 Windows32/64

起動用のバッチファイル内で環境変数CLASSPATHを設定している場合は、バッチファイル内の記述を以下のように変更します。

```
set CLASSPATH=.;%OD_HOME%¥etc¥class¥ODjava4.jar;%CLASSPATH%;
```

■環境変数の設定例 Solaris32/64 Linux32/64

起動用のシェルスクリプト内で環境変数CLASSPATHを設定している場合は、シェルスクリプト内の記述を以下のように変更します。
また、環境変数CLASSPATHの他に、環境変数LD_LIBRARY_PATHに“<CORBAサービスのインストールディレクトリ>/lib”(OD_HOME:CORBAサービスのインストールディレクトリ)を設定する必要があります。

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export CLASSPATH
LD_LIBRARY_PATH=$OD_HOME/lib
export LD_LIBRARY_PATH
```

5.8 アプレットのデジタル署名

Javaアプレットをダウンロードして運用する場合、アプレット本体／Portable-ORBに対してデジタル署名を行います。CORBAクライアントとしてのJavaアプレットは、Webサーバからダウンロードされて、CORBAのサーバアプリケーションが動作するリモート上のマシンに

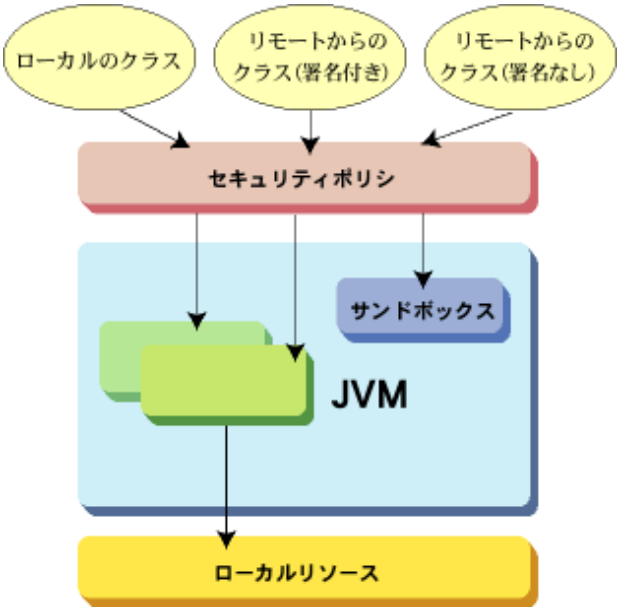
ネットワークを介してアクセスします。したがって、プレインストール型Javaライブラリ、Portable-ORBいずれを使用する際も、Javaアプレットをダウンロードして運用する場合は、デジタル署名を行う必要があります。

5.8.1 デジタル署名の概要

Webサーバからダウンロードしたクラスは、サンドボックスというセキュリティ機構の管理下に置かれ、ファイルアクセスなどのローカルコンピュータの資源へのアクセスが制限されます。
しかし、署名（権限）を施されたJavaクラスは、このサンドボックスの管理下に置かれないため、ローカルコンピュータ資源へのアクセスが可能となります。
プレインストール型Javaライブラリは、ローカル資源へアクセス（ダイナミックライブラリのロードやファイルへのアクセス）を内部的に行うため、プレインストール型JavaライブラリのAPIを呼び出しているクラスに署名を行うことが必要です。

また、セキュリティポリシという定義ができます。セキュリティポリシとは、実行するコードが実行可能であるか、またはアクセス可能なファイルのパスの制限を定義するものです。Javaクラスは、実行される前にセキュリティポリシに基づき、どのように実行するか、どこまでローカル資源のアクセスが可能であるかを確認します。

セキュリティ機構のイメージ図を以下に示します。



5.8.2 ダウンロードオブジェクトと署名対象オブジェクト

ダウンロードオブジェクトとデジタル署名を行うオブジェクトの関係を以下に示します。

ダウンロードオブジェクト (注)	署名対象オブジェクト (注)	署名形式
アプレットのみ	アプレット	X.509形式
アプレット/Portable-ORB	アプレット/Portable-ORB	

注）“アプレット”は、ユーザ作成クラスとIDLファイルから生成されたクラスを含みます。

参照

Portable-ORBのファイルは、“5.2.1 HTMLファイルの記述”を参照してください。

5.8.3 デジタル署名手順

デジタル署名手順について、Windows(R)での手順をもとに説明します。署名ツールとして以下のJDKのツールを使用します。

- keytool
- jarsigner
- policytool

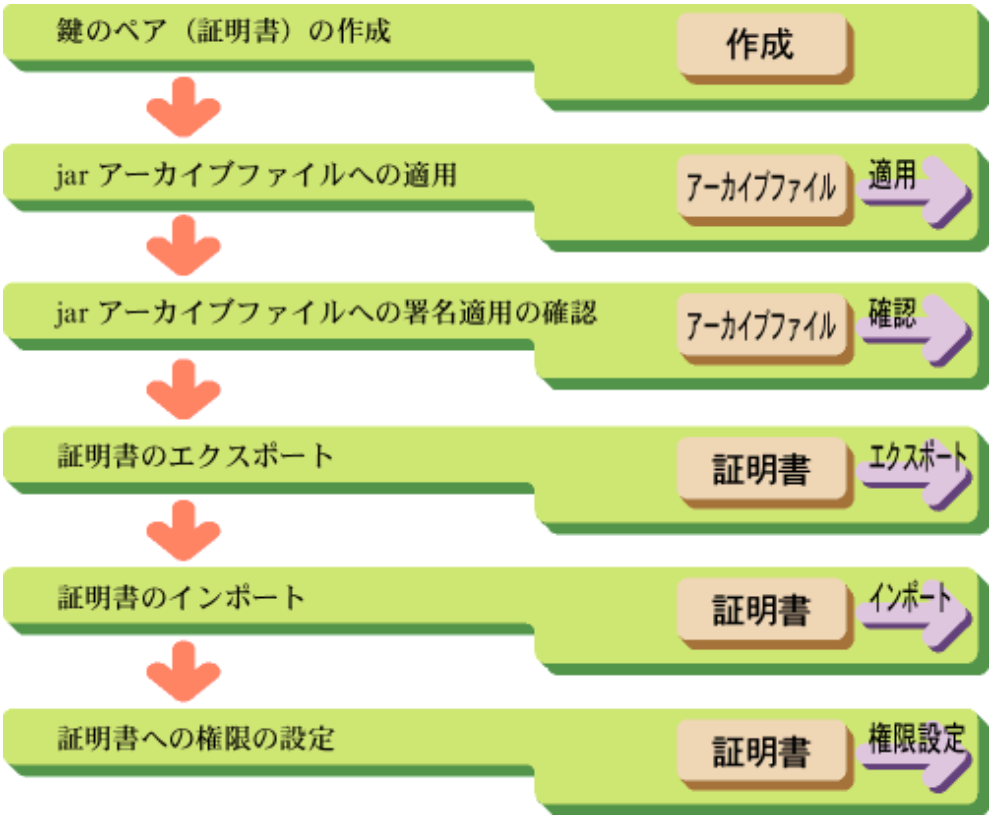
注意

Portable-ORBをダウンロードしない運用で使用する場合、またはプレインストール型Javaクライアントを使用する場合は、それぞれの環境のJavaライブラリに対しても、policytoolによる権限を設定する必要があります。Javaライブラリに対する権限の設定については、“5.5.2 Javaライブラリに対する権限の設定”を参照してください。

参照

署名ツールの詳細については、JDKのドキュメントを参照してください。J Business Kitを使用している場合は、Studioの“J Business Kit オンラインマニュアル”を参照してください。

デジタル署名は、以下の手順で行います。



(1)鍵のペア(証明書)の作成

証明書に付加する情報、証明書にアクセスするための別名を指定して鍵のペア(証明書)を作成します。
別名“samplesigner”、証明書有効期限“365日”の場合の実行例を以下に示します。

```
keytool -genkeypair -alias samplesigner -dname "cn=samplesigner, ou=JAVA PROJECT, o=FUJITSU, c=JA" -validity 365
```

-genkeypair

鍵のペア(証明書)の生成を指定します。

-alias

作成する鍵のペア(証明書)にアクセスするための別名を指定します。

-dname

署名者名、組織名、会社名、国名などを指定します。

-validity

証明書の有効期限を指定します。

コマンド入力時、キーストアのパスワード入力、作成する鍵のペア(証明書)のパスワード入力が求められます。これらのパスワードは、以降のキーストアへのアクセス、および作成した鍵のペア(証明書)へのアクセスで必要となります。
キーストアは、鍵のペア(証明書)情報などを管理するデータベースです。JDK/JREのインストール時には、存在していません。keytoolコマンドを実行することにより作成されます。
ここで作成された証明書と各クライアントマシンでインポートする証明書の作成時刻は、同一である必要があります。-dnameオプションに指定した内容が同一でも異なる時刻に作成された証明書は、別の証明書として認識されるため、注意してください。

(2)jarアーカイブファイルへの適用

作成した証明書を使用して、jarアーカイブファイルへデジタル署名を適用します。
Sample.jarに、(1)で作成した証明書を署名する場合の実行例を以下に示します。

```
jarsigner -signedjar Sample.jar.sig  Sample.jar  samplesigner
```

-signedjar

署名を適用したjarファイルの名前を指定します(省略時:署名元jarファイルと同一名)。

Sample.jar

証明元のjarファイルを指定します。

samplesigner

署名を行う証明書の別名を指定します。

コマンド入力時、キーストアのパスワード入力、鍵のペア(証明書)のパスワード入力が求められます。(1)で指定したパスワードを指定します。
jarアーカイブファイルの作成方法については、“5.1.4 Javaクラスファイルのアーカイブ”(プレインストール型Javaライブラリ)/“5.2.4 Javaクラスファイルのアーカイブ”(Portable-ORB)を参照してください。
同一の署名者で複数のjarアーカイブファイルを作成する場合は、(2)の処理を繰り返してください。

(3)jarアーカイブファイルへのデジタル署名適用の確認

デジタル署名を適用したjarアーカイブファイルに正常に署名が行われているかを確認します。
(2)でデジタル署名を適用したjarアーカイブファイルを確認する場合の実行例を以下に示します。

```
jarsigner -verify Sample.jar.sig
```

-verify

jarアーカイブファイルのデジタル署名の検証を指定します。

Sample.jar.sig

デジタル署名の検証を行うjarアーカイブファイルを指定します。

コマンドを実行すると、以下のメッセージが表示されます。詳細なデジタル署名情報を表示する場合は、-verbose/-certsオプションを指定してコマンドを実行してください。

- デジタル署名が正常に行われた場合:「jar verified」
- デジタル署名が正常に行われなかった場合:「jar is unsigned.(signatures missing or not parsable)」

(4)jarアーカイブファイルのファイル名の変更

(2)でデジタル署名を適用したjarファイルを署名元jarファイルと別名を指定している場合は、jarファイルとして使用するため、拡張子を“.jar”に変更します。
たとえば、(2)で別名に“Sample.jar.sig”を指定した場合は、デジタル署名を行う前のファイル“Sample.jar”を削除したあと、ファイル“Sample.jar.sig”のファイル名を“Sample.jar”に変更します。

(5)証明書のエクスポート

アプレットをダウンロードするクライアントマシンに、インポートするための証明書のエクスポートを行います。
(1)で指定した別名が“samplesigner”の場合の実行例を以下に示します。

```
keytool -exportcert -alias samplesigner -file samplesign.cer
```

-exportcert

証明書の取出しを指定します。

-alias

取り出す証明書の別名を指定します。

-file

取り出した証明書を格納するファイル名を指定します。

コマンド入力時、キーストアのパスワード入力が求められます。(1)で指定したパスワードを指定します。

(6)証明書のインポート

クライアントマシン上で、アプレットをダウンロードするクライアントマシンに証明書をインポートします。証明書“samplesign.cer”を事前にクライアントマシンにコピーしてください。
(5)でエクスポートした証明書“samplesign.cer”の場合の実行例を以下に示します。

```
keytool -importcert -alias sampleuser -file samplesign.cer
```

-importcert

証明書のインポートを指定します。

-alias

インポートする証明書の別名を指定します。

-file

インポートする証明書のファイル名を指定します。

コマンド入力時、キーストアのパスワード入力が求められます。これらのパスワードは、以降のキーストアへのアクセスで必要となります。
また、インポートする証明書を信頼するかどうかの入力が求められるので、「yes」と入力します。
コマンド入力時に指定した別名（-aliasで指定した別名）は、以降の操作で、証明書（-fileで指定した証明書）を指定するために必要となります。“(7)証明書への権限の設定”で権限を設定する際に、権限を設定する証明書の別名を指定してください。

(7)証明書への権限の設定

クライアントマシンにインポートした証明書に対して、権限を設定します。権限は、クライアントマシン上でpolicytoolコマンド（セキュリティポリシを定義するGUIツール）を使用して設定します。

このコマンド使用して署名されているJavaクラス、および任意の場所に格納されているJavaクラスの権限の設定または変更を行います。
policytoolコマンドの設定については、“5.8.4 policytoolコマンドの設定”を参照してください。

5.8.4 policytoolコマンドの設定

policytoolコマンドの具体的な設定方法について、以下の環境を例に説明します。

使用ライブラリ

プレインストール型Javaライブラリ

オペレーティングシステム

Windows(R) 2003

Javaのバージョン

JDK/JRE 6

CORBAクライアントのインストールディレクトリ

C:¥Interstage¥ODWIN

ユーザ名

guest

インポートした証明書の別名

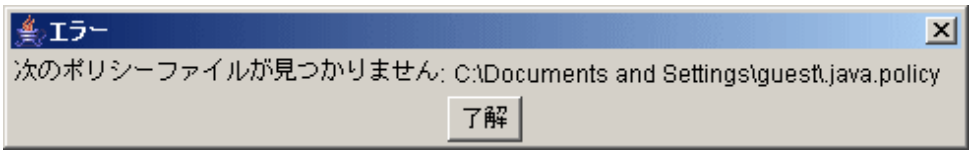
sampleuser

(1)policytoolコマンドの起動

policytoolコマンドを起動します。
表示される「Policy Tool」画面を以下に示します。

注意

policytoolコマンドの初回起動時に、以下のようなエラーを示すダイアログが表示されることがあります。これは、policytoolコマンドが起動と同時にユーザのセキュリティポリシーファイルを読み込むが、初回起動時にセキュリティポリシーファイルが存在しないためであり、動作上の問題はありません。
ただし、2回目以降のpolicytoolコマンドの起動でこのダイアログが表示される場合は、セキュリティポリシーファイルが指定された格納位置にあるかを確認してください。



(2)キーストア(証明書データベース)の格納位置の指定

キーストアファイルの格納位置、およびキーストアのタイプを指定します。
policytool起動後、最初に表示される「Policy Tool」画面内で、プルダウンメニュー「編集」→「キーストアの変更」を選択すると、以下のような「キーストア」画面が表示されます。

「新規キーストアのURL」フィールドへ使用するキーストアの格納位置を指定し、「新規キーストアのタイプ」フィールドにはキーストアタイプを指定します。それぞれのフィールドへ指定する内容を以下に示します。
設定後は、「キーストア」画面上の「了解」ボタンを押下してください。

「新規キーストアのURL」フィールド

“(6)証明書のインポート”で作成されたキーストアの格納位置を、キーストアファイル名も含めてURL形式で指定します。
デフォルトの場合の格納先を以下に示します。

プラットフォーム	格納先
Windows32/64	ログインユーザのプロファイルディレクトリ
Solaris32/64 Linux32/64	ログインユーザのホームディレクトリ

例

- 「Windows Server(R) 2012」または「Windows(R) 8」、ログイン名「guest」の場合

  キーストア格納位置

  ```
  file:c:/Users/guest/.keystore
  ```

  ポリシーファイル格納位置

  ```
  c:¥Users¥guest¥.java.policy
  ```

- 「Windows Server(R) 2003」、ログイン名「guest」の場合

  キーストア格納位置

  ```
  file:c:/Documents and Settings/guest/.keystore
  ```

  ポリシーファイル格納位置

  ```
  c:¥Documents and Settings¥guest¥.java.policy
  ```

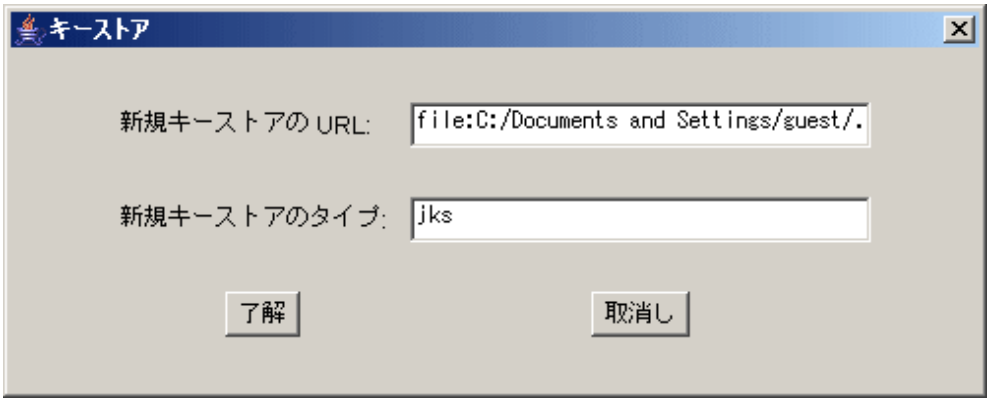
ポイント

- キーストアの格納位置には、URL上のキーストアを指定することもできます。URL上のキーストアを指定する場合、“(5)証明書のエクスポート”、“(6)証明書のインポート”の手順は必要ありません。
- キーストアのファイル名は、デフォルト名で「.keystore」です。
- ポリシーファイルのファイル名は、デフォルト名で「.java.policy」です。

「新規キーストアのタイプ」フィールド

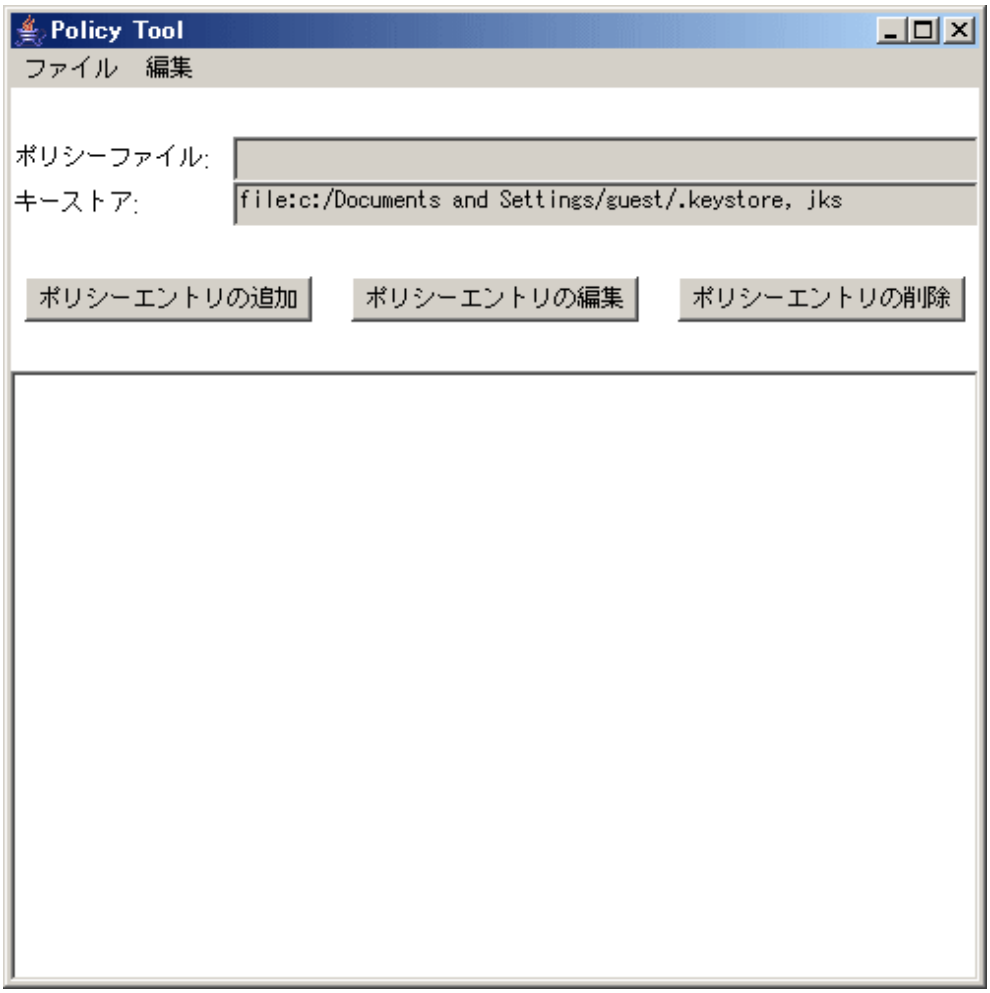
「jks」を指定します。

設定画面の例を以下に示します。

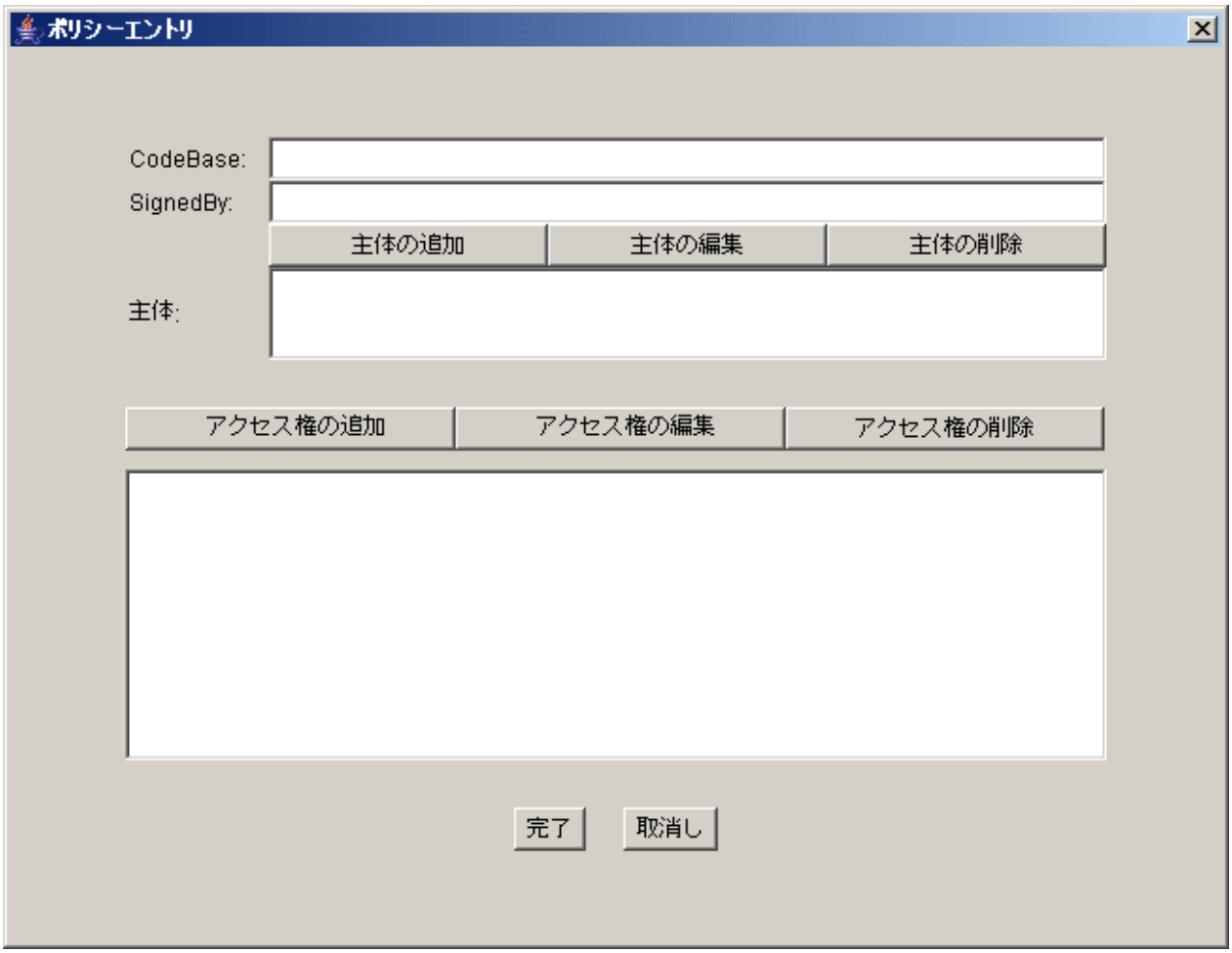


(3)エントリの作成

キーストアの格納位置を指定すると、「キーストア:」フィールドに設定した内容が反映されます。

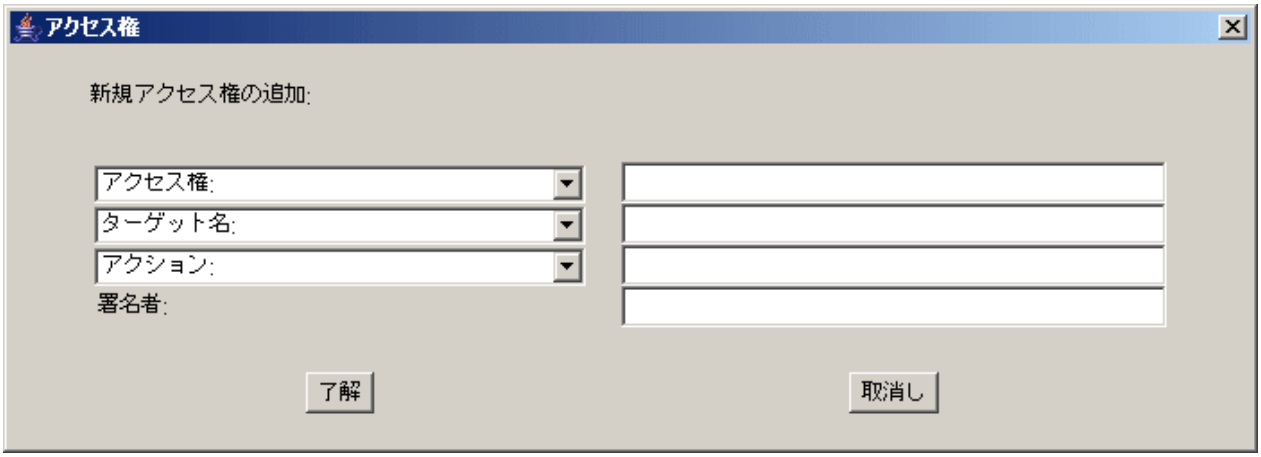


「Policy Tool」画面上の「ポリシーエントリの追加」ボタンを押下し、以下のような「ポリシーエントリ」画面で各権限を設定します。



(4)権限の設定

「ポリシーエントリ」画面上の「アクセス権の追加」ボタンを押下し、「アクセス権」画面を表示します。この画面上で権限を設定します。



一組の「アクセス権:／ターゲット名:／アクション:」の値を設定し、「了解」ボタンを押下して権限を設定します。「了解」ボタンを押下すると、「ポリシーエントリ」画面に戻ります。次の一組の値を設定するには、再度「アクセス権の追加」ボタンを押下します。これを繰り返し、必要な情報を設定します。

設定する権限は「プレインストール型Javaライブラリ」を使用する場合と「Portable-ORB」を使用する場合で異なります。
設定が必要となる権限を以下に示します。

プレインストール型Javaライブラリの場合

◇通常運用で必要な権限

通常の運用では、セキュリティ上の安全性を確保するため、本権限以外は設定しないでください。

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
ランタイム権限	RuntimePermission (java.lang.RuntimePermission)	loadLibrary.<ライブラリ名> (注1)	設定不要
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	com.fujitsu.*	read

注1) インストールしている機能により以下のダイナミックリンクライブラリ(DLL)名を指定します。なお、拡張子を指定する必要がありません。

インストールしている機能	ダイナミックリンクライブラリ
CORBAサービスクライアント(クライアント機能)	ODjava4
CORBAサービス(サーバ機能)	ODjavas4

◇CORBAサービスの内部ログを採取する場合に必要な権限 (注2)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	user.dir	read
		java.class.path	read
ファイル権限	FilePermission (java.io.FilePermission)	${user.dir}¥*	read, write
		%OD_HOME%¥etc¥config (注3)	read

注2) CORBAサービスの内部ログの詳細については、“チューニングガイド”の“config”を参照してください。なお、ログ採取後は追加した権限を削除してください。
注3) %OD_HOME%は、CORBAサービス、CORBAサービスクライアントのインストールパス(デフォルト:C:¥Interstage¥ODWIN)を指定します。

Portable-ORBの場合

◇通常運用で必要な権限

通常の運用では、セキュリティ上の安全性を確保するため、本権限以外は設定しないでください。

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
通信権限	SocketPermission (java.net.SocketPermission)	通信先サーバ名 (注1)	connect

注1) Javaアプレットをダウンロードした Webサーバ以外のサーバマシンと通信する場合に、通信先サーバ分の通信先サーバ名を設定します。通信するサーバマシンがJavaアプレットをダウンロードしたWebサーバだけの場合、設定する必要はありません。
通信先サーバ名として、以下のホスト名を指定します。

- porbeditenvコマンドの「ホスト情報」で設定したホスト名
  詳細については、“リファレンスマニュアル(コマンド編)”の“porbeditenv”を参照してください。
- 通信するサーバアプリケーションのオブジェクトリファレンスに設定されているホスト名
  odlistnsコマンド(-lオプション指定)で表示される「オブジェクトのホスト名」です。
- URLスキーマに指定したホスト名
  詳細については、“9.5 URLスキーマ”および“13.4 corbaloc URLスキーマ”を参照してください。

なお、通信先サーバとして、serverA.interstage.co.jpとserverB.interstage.co.jpが存在する場合、通信先サーバ名としてserverA.interstage.co.jpとserverB.interstage.co.jpの2組の通信権限を設定します。または、信頼できる範囲を確認した上でワイルドカード(*)を使用して、*.interstage.co.jpと設定することもできます。ただし、ワイルドカード(*)のみの指定(不特定のサーバマシンとの通信)は、セキュリティ上の安全を確保するため、指定しないでください。

◇EJBアプリケーションを使用する場合に必要な権限 (注2)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	com.fujitsu.*	read
		java.class.path	read
ランタイム権限	RuntimePermission (java.lang.RuntimePermission)	getClassLoader	(指定不要)

注2) EJBアプリケーションの詳細については、“J2EE ユーザーズガイド”の“EJB編”を参照してください。

◇Portable-ORBのログ情報を採取する場合に必要な権限 (注3)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
ファイル権限	FilePermission (java.io.FilePermission)	ログ採取ファイル (注4)	read, write, delete
		ログ採取ディレクトリ (注5)	read

注3) Portable-ORBのログ情報の詳細については、“リファレンスマニュアル(コマンド編)”の“porbeditenv”を参照してください。なお、ログを採取する場合は、porbeditenvコマンドの「ログ格納ディレクトリ」で指定したディレクトリを事前に作成しておく必要があります。また、「ログ格納ディレクトリ」にはログ採取ファイル以外のユーザ資源などを格納しないでください。ログ採取後は、追加した権限を削除してください。
注4) porbeditenvコマンドの「ログ格納ディレクトリ」で指定したディレクトリ名に「¥*」を付加したパスを指定します。「ログ格納ディレクトリ」に「c:¥log¥porb」と指定した場合は、「c:¥log¥porb¥*」です。
注5) porbeditenvコマンドの「ログ格納ディレクトリ」で指定したディレクトリ名を指定します。

◇SSL連携機能を使用する場合に必要な権限 (注6)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
ファイル権限	FilePermission (java.io.FilePermission)	keystoreディレクトリ (注7)	read

注6) SSL連携機能の詳細については、“セキュリティシステム運用ガイド”の“Portable-ORBでSSLを利用する方法”を参照してください。
注7) porbeditenvコマンドの「キーストア格納位置」で指定した格納位置、または-ORB_FJ_PORB_SSLPathパラメタで指定した格納位置

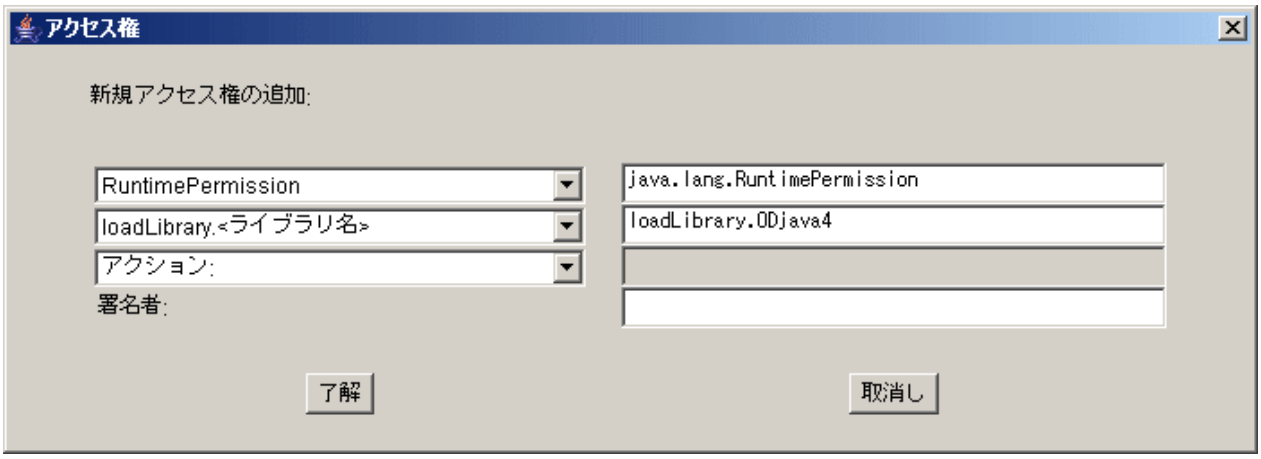
を指定します。たとえば、格納位置が「C:¥Interstage¥PORB¥etc¥keystore」の場合は、「C:¥Interstage¥PORB¥etc¥keystore」です。キーストアの格納位置がネットワーク上（HTTP指定）である場合、指定する必要はありません。

◇サーバアプリケーションでユーザ情報獲得用のAPIを使用する場合に必要な権限 (注8)

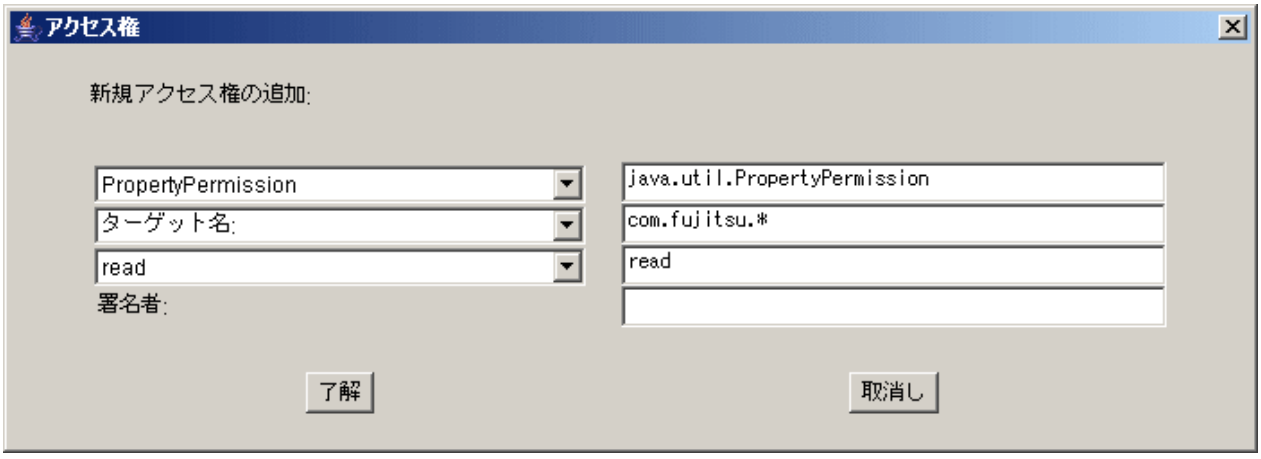
権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	user.name	read

注８）ユーザ情報獲得用のAPIの詳細については、“リファレンスマニュアル（API編）”の“TD_get_user_information”、“TD::get_user_information”、“TDGETUSERINFORMATION”を参照してください。

ランタイム権限の設定画面の例を以下に示します。

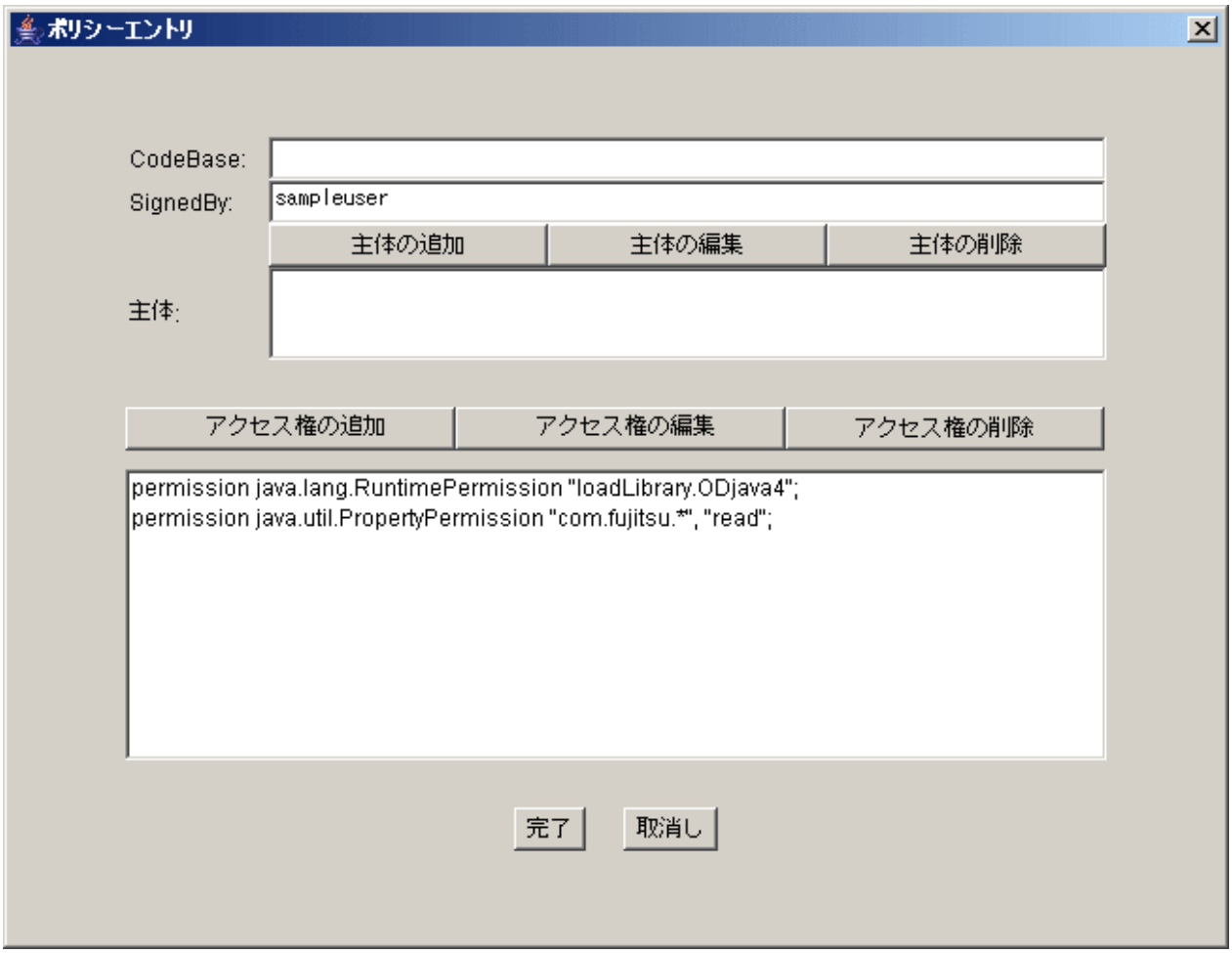


プロパティ権限の設定画面の例を以下に示します。



(5)署名付きJavaクラスのための証明書の指定

どの証明書に対する権限かを指定するため、「SignedBy:」フィールドにキーストアにインポートした名前（別名）を指定します。入力後、「完了」ボタンを押下します。

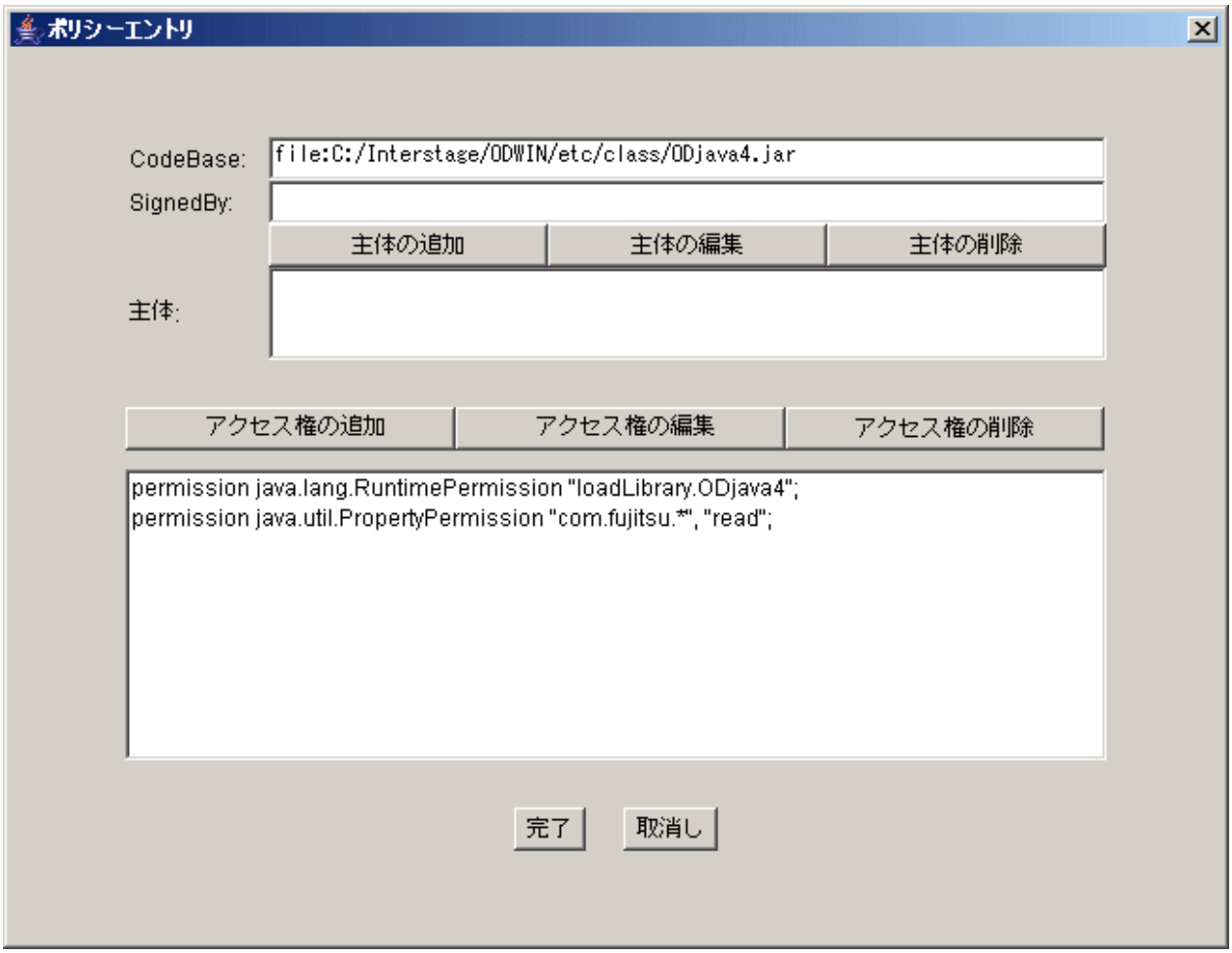


(6)プレインストール型Javaライブラリの権限の設定

プレインストール型Javaライブラリの権限を設定します。

「ポリシーエントリの追加」ボタンを押下して、ランタイム権限／プロパティ権限／ファイル権限をそれぞれ設定します（“(3)エントリの作成”、“(4)権限の設定”）。

それぞれの権限の設定後、「ポリシーエントリ」画面の「CodeBase:」フィールドにプレインストール型Javaライブラリを指定します。入力後、「完了」ボタンを押下します。



(7)保存

設定した権限をセキュリティポリシーファイルとして保存します。
初回作成時は、プルダウンメニュー「ファイル」の「別名保存」を選択してポリシーファイルの名前と格納位置を指定します。2回目以降は、「Policy Tool」画面上の「ポリシーファイル:」フィールドで指定します。
デフォルトで指定するディレクトリは使用するオペレーティングシステムにより異なります。
各オペレーティングシステムでの格納ディレクトリは、「新規キーストアのURL」フィールドで説明されている格納ディレクトリの表を参照してください。

プルダウンメニューの「ファイル」→「保存」を選択します。保存が完了すると、ポリシーファイルから以下のようなダイアログが表示されます。

(8)policytoolコマンドの終了

保存後、「ポリシーファイル:」フィールドに保存したポリシーファイルが入っていることが確認できます。
「Policy Tool」画面上でプルダウンメニュー「ファイル」→「終了」を選択し、終了します。

5.9 クライアントアプリケーションのプログラミング（静的起動インタフェース）

静的起動インタフェースを使用する場合の、クライアントアプリケーションの処理の流れを以下に示します。

5.9.1 初期化

CORBAの初期化メソッドorg.omg.CORBA.ORB.init()を呼び出し、初期化処理を行います。本メソッドの結果として、ORBオブジェクトが返されます。このオブジェクトは、以降で呼び出すORBインタフェースを使用する場合に指定します。

例

■アプリケーションの場合

```
public class Sample { // クラスの宣言
    public static void main( String args[] ) {
        org.omg.CORBA.ORB   Orb;     // ORBオブジェクト

        try {
            // ORBの生成と初期化
            Orb = org.omg.CORBA.ORB.init( args, null );
               ...
        }
        catch( java.lang.Exception e ) {
            ... // 例外処理
        }
    }
}
```

■アプレットの場合

```
public class Sample extends java.applet.Applet { // appletクラスの宣言

    org.omg.CORBA.ORB   Orb;     // ORB用オブジェクト

    public void init() {

        try {
            // ORBの生成と初期化
            Orb = org.omg.CORBA.ORB.init( this, null );
                ...
        }

        catch( java.lang.Exception e ) {
              ... // 例外処理
        }

        ... // 画面表示処理
    }

    public void start() {
        ...
    }
}
```

参照

例外処理try-catchについては、“5.10 クライアントアプリケーションの例外処理”を参照してください。

5.9.2 ネーミングサービスのオブジェクトリファレンスの獲得

実行するオブジェクトをネーミングサービスから検索するために、ネーミングサービスのオブジェクトリファレンスが必要となります。
CORBAインタフェースのオブジェクトリファレンスの取り出しメソッドorg.omg.CORBA.ORB.resolve_initial_references()によりネーミングサービスのオブジェクトリファレンスを取り出します。

```
// NamingService用オブジェクトリファレンス
org.omg.CosNaming.NamingContextExt  Cos;

// ネーミングサービスのオブジェクトリファレンスの取得
try {
   org.omg.CORBA.Object _tmpObj =
                         Orb.resolve_initial_references( "NameService" );
// NamingContext型に変換
   Cos  = org.omg.CosNaming.NamingContextExtHelper.narrow( _tmpObj );
          ...
}
catch( java.lang.Exception e ) {
         ... // 例外処理
}
```

5.9.3 サーバアプリケーションのオブジェクトリファレンスの獲得

ネーミングサービスのメソッドCosNaming.NamingContext.resolve()により、これから実行したいサーバアプリケーションのオブジェクトリファレンスを取り出します。検索したいオブジェクト名を本メソッドのパラメタとして指定します。

```
// ネーミングサービスのresolveメソッドを発行し、
// サーバアプリケーションのオブジェクトリファレンスの獲得
String NCid   = new String( "ODdemo::calculator" );   // オブジェクト名
String NCkind = new String( "" );                     // オブジェクトタイプ
try {
    org.omg.CosNaming.NameComponent nc =  new NameComponent( NCid, NCkind );
    org.omg.CosNaming.NameComponent NCo[] = { nc };
    // NamingServiceのresolveメソッドを発行
    org.omg.CORBA.Object  Obj = Cos.resolve( NCo );
          ...
}
catch( java.lang.Exception e ) {
         ... // 例外処理
}
```

5.9.4 メソッドの呼び出し

サーバプログラムのメソッドを呼び出します。メソッド名は、IDLで指定したモジュール名、インタフェース名、およびオペレーション名をドット(.)でつなげた形式で指定します。この例では、ODdemo、calculator、calculateがそれにあたります。なお、メソッド呼出し時にネーミングサービスで求めたサーバアプリケーションのオブジェクトリファレンスとサーバプログラムで例外が発生した場合に例外情報を設定するためにtry-catchブロックを指定します。

```
ODdemo.calculator target;    // アプリケーション用オブジェクトリファレンス
try {
    // ODdemo.calculatorクラスへの変換
    target = ODdemo.calculatorHelper.narrow(obj);
    ODdemo.calculatorPackage.result res = new ODdemo.calculatorPackage.result();
    res = target.calculate( a, b );
          ...
}
catch( NumberFormatException e ){
```

```
        ... // 例外処理
}
catch( ODdemo.calculatorPackage.ZEROPARAM e ){
        ... // 例外処理
}
catch( org.omg.CORBA.SystemException err ){
        ... // 例外処理
}
```

5.10 クライアントアプリケーションの例外処理

クライアントアプリケーションは、サーバアプリケーションの処理結果が正常終了か異常終了かを知ることができます。また、異常終了した場合は、システムの異常終了か、サーバアプリケーションの異常終了かを知ることができます。前者をシステム例外、後者をユーザ例外と呼びます。システム例外クラスを以下の表に示します。
システム例外は、親クラスとしてorg.omg.CORBA.SystemExceptionが定義され、その継承クラスとして各詳細情報を定義したクラスorg.omg.CORBA.UNKNOWN 、org.omg.CORBA.BAD_PARAM などがあります。ユーザ例外は、親クラスとしてorg.omg.CORBA.UserExceptionが定義されます。例外の意味については、“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。

例外情報	例外クラス
BAD_CONTEXT	org.omg.CORBA.BAD_CONTEXT
BAD_INV_ORDER	org.omg.CORBA.BAD_INV_ORDER
BAD_OPERATION	org.omg.CORBA.BAD_OPERATION
BAD_PARAM	org.omg.CORBA.BAD_PARAM
BAD_QOS	org.omg.CORBA.BAD_QOS
BAD_TYPECODE	org.omg.CORBA.BAD_TYPECODE
CODESET_INCOMPATIBLE	org.omg.CORBA.CODESET_INCOMPATIBLE
COMM_FAILURE	org.omg.CORBA.COMM_FAILURE
DATA_CONVERSION	org.omg.CORBA.DATA_CONVERSION
FREE_MEM	org.omg.CORBA.FREE_MEM
IMP_LIMIT	org.omg.CORBA.IMP_LIMIT
INITIALIZE	org.omg.CORBA.INITIALIZE
INTERNAL	org.omg.CORBA.INTERNAL
INTF_REPOS	org.omg.CORBA.INTF_REPOS
INV_FLAG	org.omg.CORBA.INV_FLAG
INV_IDENT	org.omg.CORBA.INV_IDENT
INV_OBJREF	org.omg.CORBA.INV_OBJREF
INV_POLICY	org.omg.CORBA.INV_POLICY
MARSHAL	org.omg.CORBA.MARSHAL
NO_IMPLEMENT	org.omg.CORBA.NO_IMPLEMENT
NO_MEMORY	org.omg.CORBA.NO_MEMORY
NO_PERMISSION	org.omg.CORBA.NO_PERMISSION
NO_RESOURCES	org.omg.CORBA.NO_RESOURCES
NO_RESPONSE	org.omg.CORBA.NO_RESPONSE
OBJ_ADAPTER	org.omg.CORBA.OBJ_ADAPTER

例外情報	例外クラス
PERSIST_STORE	org.omg.CORBA.PERSIST_STORE
REBIND	org.omg.CORBA.REBIND
TIMEOUT	org.omg.CORBA.TIMEOUT
TRANSIENT	org.omg.CORBA.TRANSIENT
UNKNOWN	org.omg.CORBA.UNKNOWN
INVALID_TRANSACTION	org.omg.CORBA.INVALID_TRANSACTION
TRANSACTION_MODE	org.omg.CORBA.TRANSACTION_MODE
TRANSACTION_REQUIRED	org.omg.CORBA.TRANSACTION_REQUIRED
TRANSACTION_ROLLEDBACK	org.omg.CORBA.TRANSACTION_ROLLEDBACK
TRANSACTION_UNAVAILABLE	org.omg.CORBA.TRANSACTION_UNAVAILABLE

tryブロックで例外をハンドルするために例外クラスに対応するcatchブロックを定義します。例外が発生すると、例外クラスに対応するcatchブロックが実行されます。
詳細エラーごとにエラー処理を記述したい場合は、以下のように詳細エラーごとにcatch文で指定します。共通なエラー処理は、システム例外の場合、org.omg.CORBA.SystemExceptionクラスをcatch文で利用すれば、それ以前のcatchで指定されている詳細エラー以外のすべてのエラー情報を一括して受けることができます。もし、システム例外、ユーザ例外、各例外のエラー処理の区別が必要でない場合、上位クラスであるjava.lang.Exceptionをcatch文で指定すれば、すべて一括して受けることができます。
また、システム例外発生時は、org.omg.CORBA.SystemException.minorを使用してマイナーコードを獲得することができます。マイナーコードの値については、“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。

```
try {
    fret =target.divide( inArg1,inArg2 );
    // divideメソッドを実行
}
catch( org.omg.CORBA.UNKNOWN e ){
    // org.omg.CORBA.UNKNOWNの場合の処理
    System.out.println("ERROR : " + e.getClass().getName()
      + ": Minor = 0x" + java.lang.Integer.toHexString(e.minor)); // マイナーコードの獲得
}
catch( org.omg.CORBA.BAD_PARAM e ){
    // org.omg.CORBA.BAD_PARAMの場合の処理
    System.out.println("ERROR : " + e.getClass().getName()
      + ": Minor = 0x" + java.lang.Integer.toHexString(e.minor)); // マイナーコードの獲得
}
catch( org.omg.CORBA.SystemException e ){
    System.out.println("ERROR : " + e.getClass().getName()
      + ": Minor = 0x" + java.lang.Integer.toHexString(e.minor)); // マイナーコードの獲得
    // 上記以外のシステム例外
}
catch( demo.calc.ZERODIVIDE e ){
    // ユーザ例外の処理
}
```

または

```
try {
    fret =target.divide( inArg1, inArg2 ); // divideメソッドを実行
}
```

```
catch( java.lang.Exception e ){
    if ( e instanceof demo.calc.ZERODIVIDE ){
        // ユーザ例外の処理
    }
    if ( e instanceof org.omg.CORBA.SystemException ){
        // システム例外の処理
        System.out.println("ERROR : " + e.getClass().getName()
          + ": Minor = 0x"                  // マイナーコードの獲得
          + java.lang.Integer.toHexString(((org.omg.CORBA.SystemException)e).minor));
              :
    }
    // その他の例外の処理など
}
```

IDL言語でユーザ例外が定義されていた場合のクライアントアプリケーションプログラムの例を以下に示します。

(1)IDLマッピング

【IDL言語】

```
module ODsample {
    interface  exptest{
        exception testException { string reason; };
        void  op1() raises( testException );
    };
};
```

これをJava言語で記述すると、以下のようになります。

【Java言語】

＜インタフェース＞

```
package ODsample;
public interface exptest  extends org.omg.CORBA.Object  {
    public void op1()  throws ODsample.exptestPackage.testException;
}
```

＜ユーザ例外クラス＞

```
package ODsample.exptestPackage;
public class testException extends org.omg.CORBA.UserException {
    public java.lang.String reason;
    public testException()  {
        reason = null;
    }
    public testException( java.lang.String _reason )  {
        reason = _reason;
    }
}
```

(2)クライアントアプリケーションでの処理

```
import org.omg.CORBA.*;
import ODsample.*;
```

```
public class expClient {
    public static void main( String args[] ) {
        // ORBの前処理
        // オブジェクトリファレンスの獲得
            :
         try {
            // サーバアプリケーションのメソッド呼出し
            target.op1( );
        }

        catch ( ODsample.exptestPackage.testException e ) {
            System.out.println( "UserException : " );
            System.out.println( "   " + e  + " -> " + e.reason );
        }

        catch (java.lang.Exception e) {
            if ( e instanceof demo.calc.ZERODIVIDE ){
                // ユーザ例外の処理
            }
            if ( e instanceof org.omg.CORBA.SystemException )
                // システム例外の処理
                System.out.println("ERROR : " + e.getClass().getName()
                 + ": Minor = 0x"                    // マイナーコードの獲得
                 + java.lang.Integer.toHexString(((org.omg.CORBA.SystemException)e).minor));
                   :
            }
        }
    }
}
```

5.11 簡単なサーバアプリケーションの作成例

これまでの説明をもとに、簡単なサーバアプリケーションの開発から実行までの流れを具体的な例を使用して説明します。
“5.19.1 Default Servant使用例(デフォルトインスタンス方式)”に記載したアプリケーション例を使用し、起動までの過程を説明します。

ポイント

ここでは、以下の環境を使用した場合をもとに説明します。また、クライアントアプリケーションは、サーバアプリケーションと同一システム上で動作させるものとしています。

Windows32/64

カレントディレクトリ(作業用ディレクトリ)

C:¥home¥example

コマンド入力画面

DOSウィンドウ

Solaris32/64 Linux32/64

カレントディレクトリ(作業用ディレクトリ)

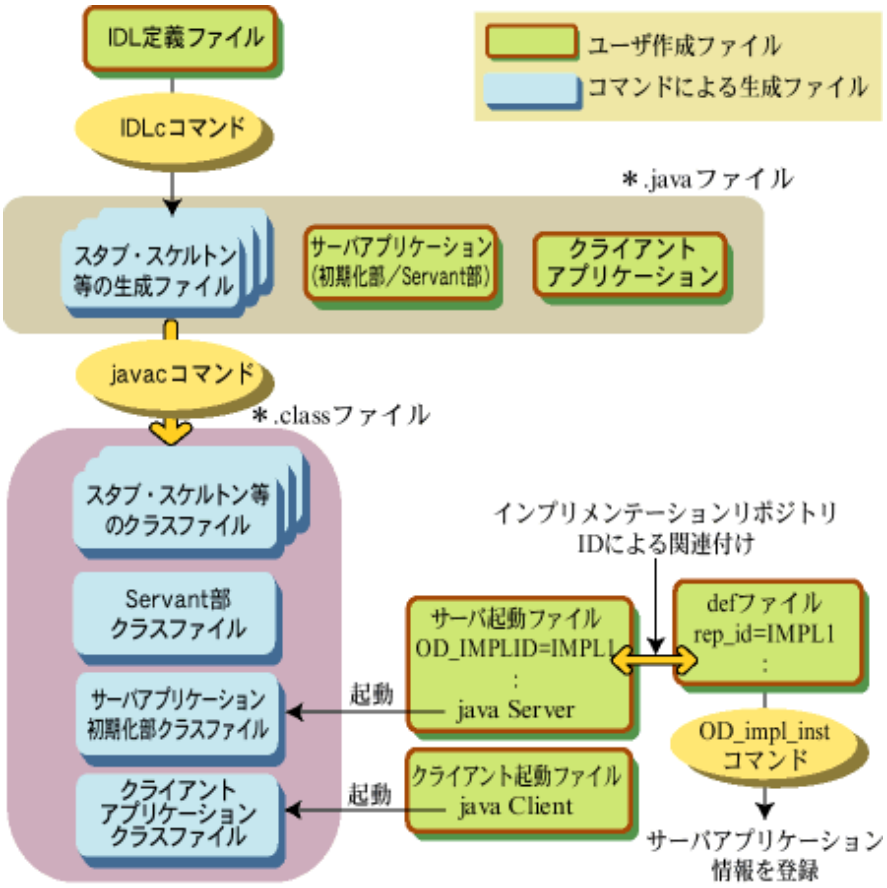
/home/example

コマンド入力画面

端末画面

5.11.1 作業の流れ

全体の作業の流れとファイル間の関連は、以下のようになります。



ユーザ自身で作成するファイルは、以下のものとなります([ ]内は本節の例で使用するファイル名)。

1. IDLファイル [TEST.idl]
2. サーバアプリケーション(初期化処理部、Servant部)のソースファイル [Server.java]
3. クライアントアプリケーションのソースファイル [Client.java]
4. defファイル [def]
5. 起動用ファイル(バッチ/シェルスクリプト) [exec-SV.bat/exec-SV、exec-CL.bat/exec-CL]

5.11.2 IDLファイルの作成とコンパイル

以下のような内容のIDL定義ファイルを作成します。

IDL定義ファイル(TEST.idl)

```
module ODsample{
    interface  intf{
        long  add( in  long  a,  in  long  b );
    };
};
```

このファイルをファイル名TEST.idlとしてカレントディレクトリに格納します。
次に、IDLcコマンドで作成したファイルTEST.idlをコンパイルします。これによりJavaアプリケーションで使用するスタブ・スケルトン、その他のファイルが生成されます。以下のコマンドを入力してください。

```
IDLc -java TEST.idl
```

この結果、カレントディレクトリにODsampleディレクトリが生成され、その配下にスタブ、スケルトンなどの*.javaファイルが生成されます。

5.11.3 サーバアプリケーションのソースファイル作成

サーバアプリケーションは、IDLで定義したインタフェースを実装する「Servant部」と、ORBの初期化やServantオブジェクトの生成・登録などを行う「初期化処理部」に分けられます。ここでは、簡単化のため、「Servant部」と「初期化処理部」は、同一のファイル上に記述するものとします（同一ファイルである必要はありません）。
“5.19.1 Default Servant使用例（デフォルトインスタンス方式）”のサーバアプリケーション部分を記述したソースファイルを作成し、ファイル名Server.javaとしてカレントディレクトリに格納してください。

5.11.4 クライアントアプリケーションのソースファイル作成

“5.19.1 Default Servant使用例（デフォルトインスタンス方式）”のクライアントアプリケーション部分を記述したソースファイルを作成し、ファイル名Client.javaとしてカレントディレクトリに格納してください。

5.11.5 Javaファイルのコンパイル

カレントディレクトリ配下のJavaファイル（IDLcコマンドが生成した*.javaファイルおよびServer.java,Client.java）をjavacコマンドでコンパイルします。javacコマンド実行時には、環境変数CLASSPATHを設定する必要があります。以下のコマンドを入力してください。

例

Windows32/64 (OD_HOME: C:¥Interstage¥ODWIN)

```
set CLASSPATH=.;%OD_HOME%¥etc¥class¥ODjava4.jar;%CLASSPATH%;
javac  -d  .  *.java
```

Solaris32/64 (OD_HOME: /opt/FSUNod) Linux32/64 (OD_HOME: /opt/FJSVod)

```
CLASSPATH=./:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH:
export CLASSPATH
javac  -d  .  *.java
```

5.11.6 defファイルの作成と登録

サーバアプリケーションに関する情報を記述したファイルを作成します。以下の内容のファイルを作成し、ファイル名defとしてカレントディレクトリに格納してください。

defファイル（ファイル名def）

```
rep_id        = IMPL1
type          = persistent
mode          = SYNC_END
proc_conc_max = 1
thr_conc_init = 1
```

```
ior           = 1.1
locale        = UNICODE
```

次に、OD_impl_instコマンドを使用して、defファイルに記述した情報を登録します。以下のコマンドを入力してください。

```
OD_impl_inst  -ax  def
```

5.11.7 起動用ファイルの作成

アプリケーションを起動するための起動用ファイルを作成します。以下のファイル名で作成し、カレントディレクトリに格納してください。

注意

Solaris32/64
exec-CL／ exec-SVは、実行可能ファイルに設定してください。

■サーバ起動用ファイル

exec-SV.bat Windows32/64

```
set OD_IMPLID=IMPL1
java Server
```

exec-SV Solaris32/64 Linux32/64

```
OD_IMPLID=IMPL1
export OD_IMPLID
java Server
```

■クライアント起動用ファイル

exec-CL.bat Windows32/64

```
java Client
```

exec-CL Solaris32/64 Linux32/64

```
java Client
```

5.11.8 アプリケーションの実行

クライアント・サーバのコマンド入力画面から、以下の環境変数を設定し、アプリケーションを起動します。

■環境変数の設定（コマンド入力）

例

Windows32/64 (OD_HOME: C:¥Interstage¥ODWIN)

```
set CLASSPATH=.¥;%OD_HOME%¥etc¥class¥ODjava4.jar;%CLASSPATH%;
```

Solaris32/64 (OD_HOME: /opt/FSUNod) Linux32/64 (OD_HOME: /opt/FJSVod)
環境変数CLASSPATHの他、環境変数LD_LIBRARY_PATHに“$OD_HOME/lib”を設定する必要があります。

```
CLASSPATH=./:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH:
export CLASSPATH
LD_LIBRARY_PATH=$OD_HOME/lib
export LD_LIBRARY_PATH
```

■サーバアプリケーションの起動(コマンド入力)

```
exec-SV
```

■クライアントアプリケーションの起動(コマンド入力)

```
exec-CL
```

注意

- サーバアプリケーション実行時は、スレッドの多重度を16以下にしてください。
- サーバアプリケーション実行時は、クライアントの総接続数を100以内にしてください。

5.11.9 アプリケーションの実行結果

アプリケーションの実行に成功すると、以下のような実行結果が得られます。

【実行結果(クライアント側の表示)】

```
in1 => 888  (値を入力)
in2 => 111  (値を入力)
888 + 111 = 999   (結果の表示)
```

サーバアプリケーションを停止させるには、[Ctrl+C]を入力してください。

5.11.10 サーバアプリケーション情報の削除

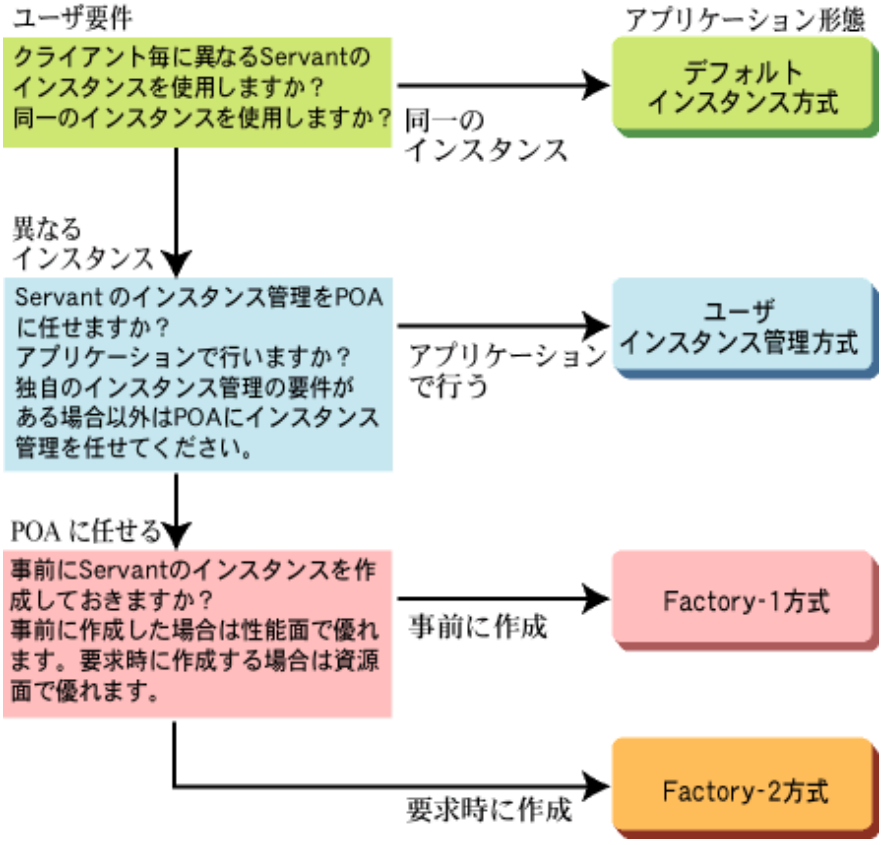
サーバアプリケーション終了後、以下のコマンドを実行してサーバアプリケーション情報を削除します。

```
OD_impl_inst  -d  -r  IMPL1
```

5.12 インスタンス管理とアプリケーション形態

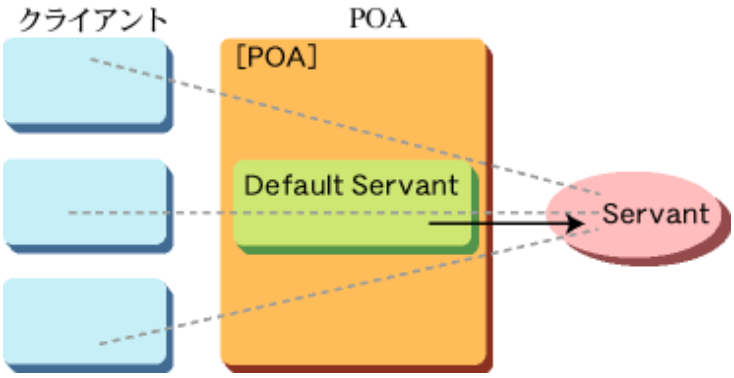
5.12.1 アプリケーション形態の種別

サーバアプリケーションは、Servant(インタフェース実装部)のインスタンスの管理方法により以下の4つの形態をとることができます。
以下のように要件によりアプリケーション形態を選択してください。



■デフォルトインスタンス方式

各クライアントが同一のServantオブジェクトを使用する方式です。つまり、クライアントごとのインスタンス管理をしない方式です。
各クライアントは、以下のように同一のServantのインスタンスを使用します。



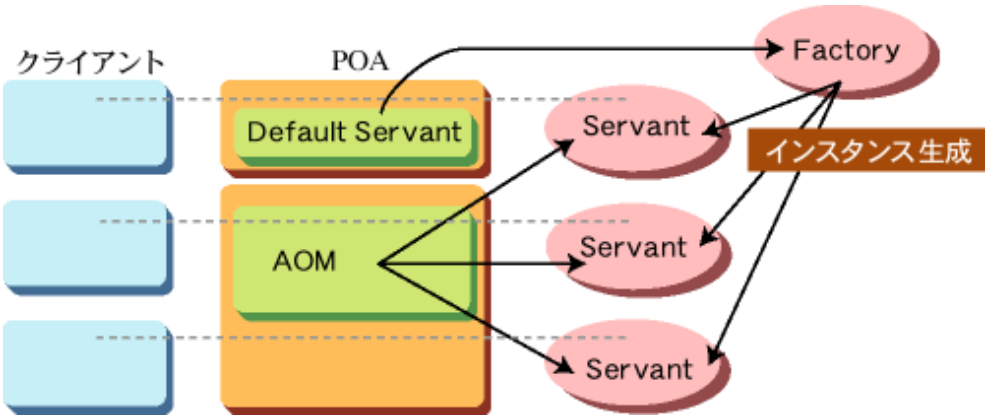
■Factory-1方式

クライアントごとに異なるServantオブジェクトを使用する方式です。
クライアントごとのインスタンス管理をPOAオブジェクトで行う方式です。Factory内で、オブジェクトリファレンスおよびインスタンスを作成します。

■Factory-2方式

クライアントごとに異なるServantオブジェクトを使用する方式です。
クライアントごとのインスタンス管理をPOAオブジェクトで行う方式です。Factory内でオブジェクトリファレンスを作成し、リクエスト受信時にServantManagerオブジェクト内でインスタンスを作成します。

Factoryは、リクエストごとにオブジェクトリファレンスやインスタンスを作成するオブジェクトであり、インスタンス管理を行う場合に必要となるオブジェクトです。
AOMはActive Object Mapのことであり、インスタンスの管理テーブルのことです。
AOM、ServantManagerなどの詳細については、“5.13 サーバアプリケーションのプログラミング（POA概要）”を参照してください。
Factory-1方式およびFactory-2方式の各クライアントは、以下のようにクライアントごとに異なるServantのインスタンスを使用します。

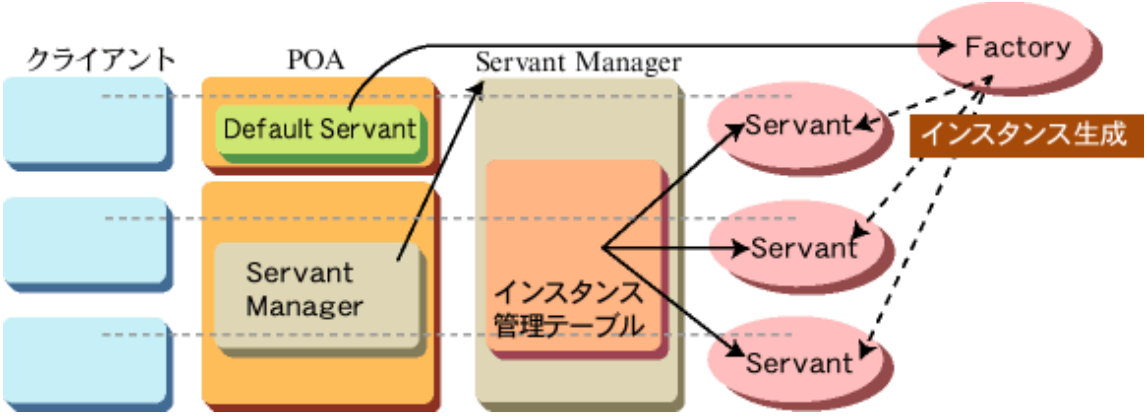


Factory-1方式とFactory-2方式の違い

Factory-1方式は、Factory内で事前にインスタンスを作成しておくことができるため、リクエストを受信した時点でインスタンスを作成するFactory-2より性能には若干優れます。
Factory-2方式は、リクエストを受信した時点でインスタンスを作成するため、Factory内で事前のインスタンスを作成しておくFactory-1より資源的には若干優れます。

■ユーザインスタンス管理方式

インスタンス管理をユーザの作成するServantManagerオブジェクトに任せる方式です。
各クライアントは、以下のように、クライアントごとに異なるServantのインスタンスを使用します。インスタンスは、ユーザの作成したServantManagerオブジェクト内のインスタンス管理テーブルで行われます。



アプリケーション形態の選択

アプリケーション形態は、Servantオブジェクトの属するPOAオブジェクトのServantRetentionポリシ、RequestProcessingポリシの指定により決まります。組合せ方を以下に示します。詳細については、“5.13.3 POAオブジェクト”を参照してください。

アプリケーション形態	ServantRetentionポリシ	RequestProcessingポリシ
デフォルトインスタンス方式	NON_RETAIN	USE_DEFAULT_SERVANT
Factory-1方式	RETAIN	任意
Factory-2方式	RETAIN	USE_SERVANT_MANAGER
ユーザインスタンス管理方式	NON_RETAIN	USE_SERVANT_MANAGER

注）表内のポリシは、Servantオブジェクトを管理するPOAポリシに対する設定値を示しています。

Factoryについて

Factoryとは、Servantオブジェクトのインスタンスを生成し、そのオブジェクトリファレンスをクライアントに返すことを目的としたインタフェースです。
これにより、そのクライアントが専用で使用できるServantオブジェクトのインスタンスをサーバ側に持たせることが可能となります。
Factoryの概念は、以下のIDL定義により示されます。前述したFactory-1方式、Factory-2方式を使用する際も、IDL定義で他のインタフェースとともにFactoryインタフェースを定義します。

【FactoryのIDL定義例(a)】

```
// 何らかのインタフェース
interface intf{
    void  op();
};

// Factoryインタフェース
interface Factory {
    intf   create();
};
```

上記のIDL定義で示されるように、クライアントはFactoryオブジェクトのオペレーションcreate()を発行することにより、そのクライアントで使用するintfオブジェクトを取得することが可能となります。
次に、クライアントは取得したintfオブジェクトに対して、op()オペレーションを発行することで本来の目的の処理を行います。

一方、サーバ側では、以下の一連の処理をcreate()オペレーションの実装として記述する必要があります。

1. Servantオブジェクト（intfインタフェースの実装クラスのインスタンス）を生成（new）します。
2. POAのAOMに、生成したインスタンスとオブジェクトリファレンスとの関連付けを登録します。
3. 関連付けられたオブジェクトリファレンスを返します。

Factory-2方式では、create()オペレーションの実装として3.のオブジェクトリファレンスの生成と復帰だけを行い、1.と2.は要求時にServantManagerオブジェクト内で行います。しかし、クライアント側から見たcreate()オペレーションの機能は同じです。

Factoryとは、このようなインタフェースの機能上の概念です。したがって、IDL定義では、Factoryインタフェースの名前（上記の例：“Factory”）や実装するオペレーションの名前（上記の例：“create”）、種類は自由です。しかし、Factoryの目的上、上記のcreate()のような「戻り値としてオブジェクトリファレンスを返す」機能をもつオペレーションが最低限必要となります。

実際のアプリケーション開発では、生成したServantオブジェクトのインスタンスを解放するためのオペレーションもFactoryに実装することが推奨されます。これによりサーバ資源の節約が可能となります。
以下のIDL定義の例では、Servantオブジェクトのインスタンスを解放するためのオペレーションとしてdestroy()を定義しています。

【FactoryのIDL定義例(b)】

```
// 何らかのインタフェース
interface intf{
    void  op();
};

// Factoryインタフェース
interface Factory{
    intf  create();
    void  destroy(in intf obj);
};
```

クライアント側では、目的の処理が終了した時点で、以前にcreate()メソッドで取得したオブジェクトリファレンスを引数としてdestroy()オペレーションをコールします。一方、サーバ側では、渡されたオブジェクトリファレンスに対応するServantオブジェクトをAOMから削除する処理を行います。

上記のIDLに対するFactoryオブジェクトのcreate()オペレーション、destoroy()オペレーションの実装例を以下に示します。ここでは、intfインタフェースの実装クラスをUserServantクラスとしています。処理の詳細については、“5.13 サーバアプリケーションのプログラミング(POA概要)”以降を参照してください。

【オペレーションの実装例(Factory-1方式)】

```
public intf create() {
    intf       ior; // Servantオブジェクトのオブジェクトリファレンス
    try {
        // 1. Servantオブジェクトの生成
        org.omg.PortableServer.Servant svt = new UserServant();
        // 2. AOMへ登録
        poa.activate_object( svt );
        // 3. オブジェクトリファレンスの生成
        org.omg.CORBA.Object Obj = poa.servant_to_reference( svt );
        // intf型に変換
        ior = intfHelper.narrow( Obj );
    }
    catch( Exception e ) {  /* 例外処理 */  }
    return( ior );
}
public void destroy( intf obj ) {
    try {
        // オブジェクトリファレンスからオブジェクトIDを求める
        byte oid[] = poa.reference_to_id( obj );
        // Servantをdeactiveにする (AOMより削除)
        poa.deactivate_object( oid );
    }
    catch( Exception e ) {  /* 例外処理 */ }
}
```

注) ソース上のpoa:POAオブジェクトのインスタンス

Factoryオブジェクトのインスタンスの扱いについて

Factoryオブジェクトのインスタンスについても、通常のServantオブジェクトのインスタンスと同様に、POAに登録します。
“5.19.2 Active Object Map(AOM)使用例(Factory-1方式)”や“5.19.3 Servant Activator使用例(Factory-2方式)”などに示した例で

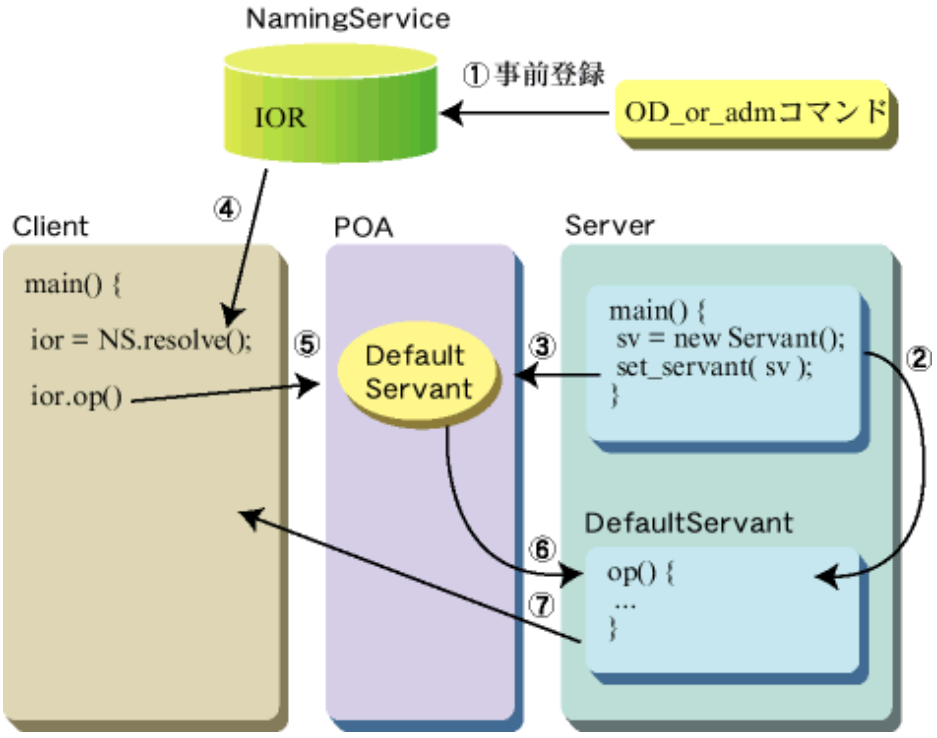
は、Factoryオブジェクトは「Factory専用のPOA」を使用してDefault Servantとして登録しています。加えて、Factory用のPOAはAOMを使用しない設定（NON_RETAINポリシ）としています。
これは、以下の理由によるものです。

- FactoryオブジェクトとそのFactoryが生成するServantオブジェクトを、同一のPOAのAOM上で管理させることは可能です。しかし、この場合、Factoryオブジェクトへの要求に対してもAOM内の検索が行われるため、効率的ではありません。
- Servantオブジェクトを管理するPOAのDefault ServantとしてFactoryオブジェクトを登録することは可能です。しかし、AOMとDefault Servantを同時に使用する場合は、Default Servantへの要求はAOMの検索の後となります。このため、Factoryオブジェクトへの要求は効率的ではありません。

5.12.2 各アプリケーション形態

(1)デフォルトインスタンス方式

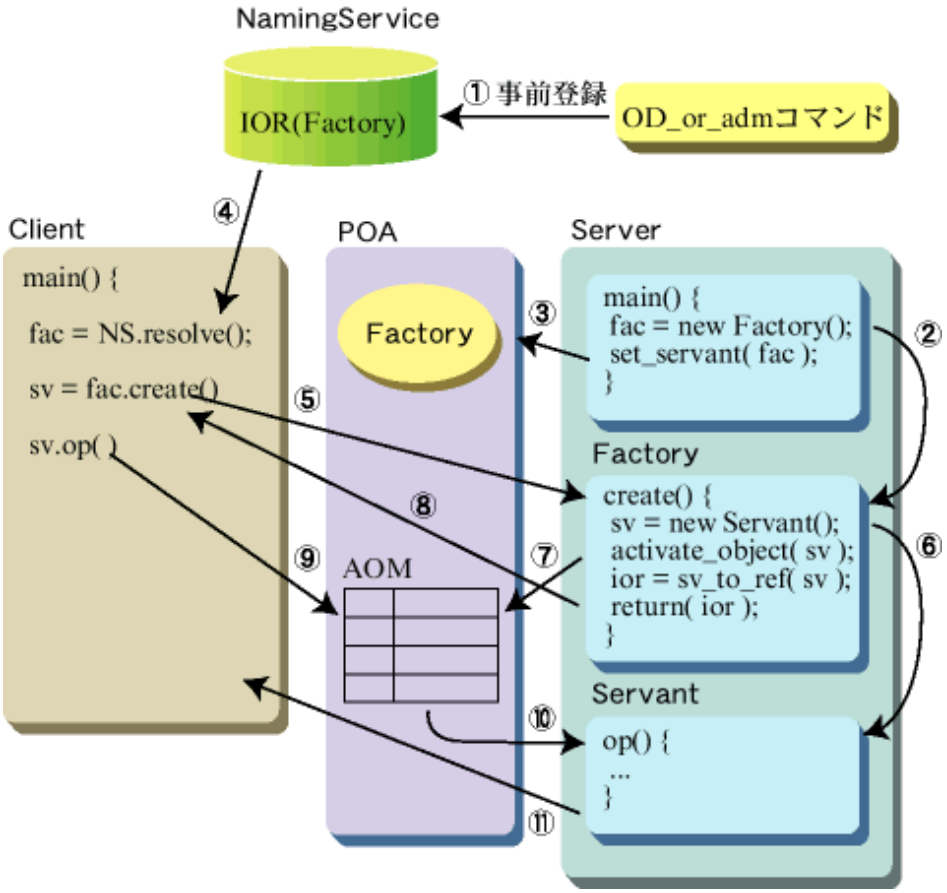
クライアントごとのインスタンス管理をしない方式です。



1. OD_or_admコマンドで事前にServantのオブジェクトリファレンスをNamingServiceに登録します。
2. サーバアプリケーションは、Servantを作成します。
3. サーバアプリケーションは、ServantをPOAにDefault Servantとして登録します。
4. クライアントアプリケーションは、NamingServiceからオブジェクトリファレンスを獲得します。
5. クライアントアプリケーションは、オペレーションop()を要求します。
6. POAは、登録されているDefault Servantを起動します。
7. POAは、オペレーションop()の結果をクライアントに通知します。

(2)Factory-1方式

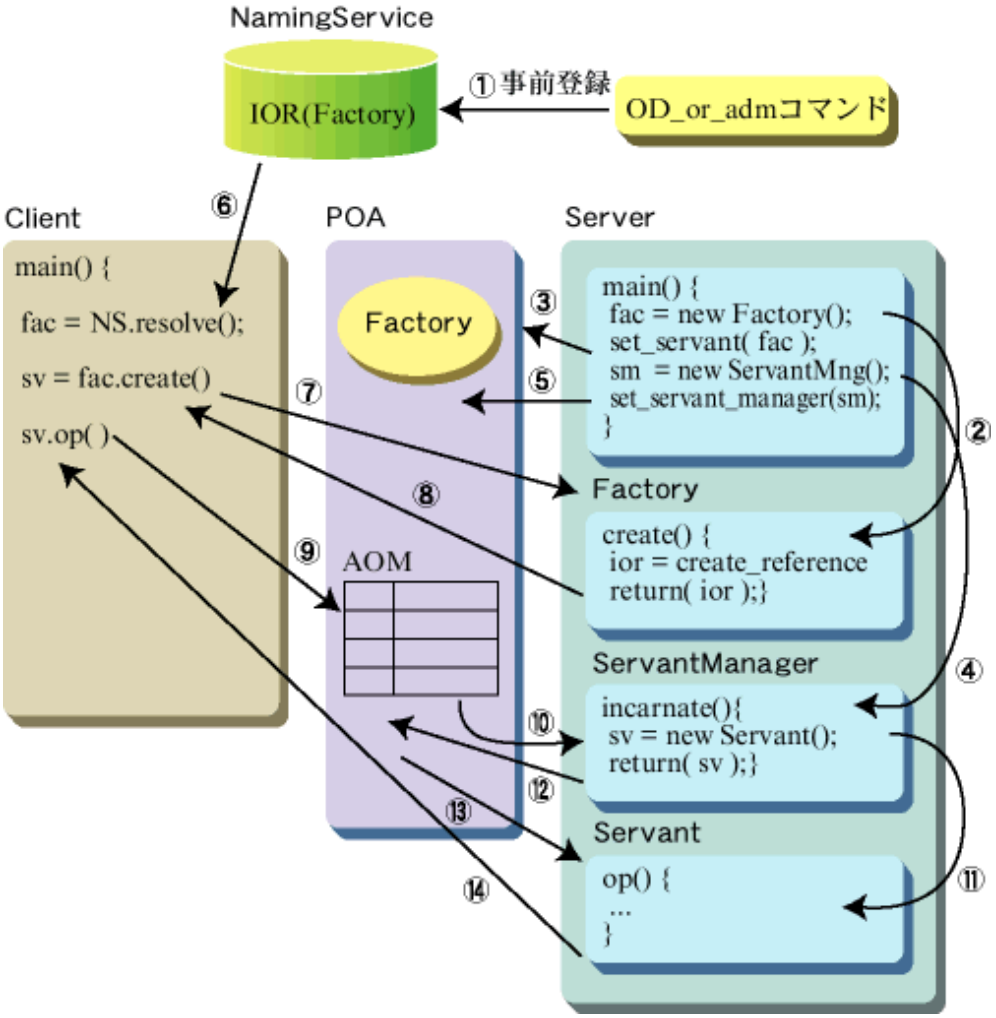
クライアントごとでインスタンス管理をPOAオブジェクトで行う方式です。Factory内で、オブジェクトリファレンスおよびインスタンスを作成します。



1. OD_or_admコマンドで事前にFactoryのオブジェクトリファレンスをNamingServiceに登録します。
2. サーバアプリケーションは、Factoryを作成します。
3. サーバアプリケーションは、FactoryをDefault ServantとしてPOAに登録します。
4. クライアントアプリケーションは、NamingServiceからFactoryのオブジェクトリファレンスを獲得します。
5. クライアントアプリケーションは、オペレーションcreateをFactoryに要求します。
6. Factoryは、Servantのインスタンスを作成します。
7. Factoryは、AOMにインスタンスを登録します。
8. Factoryは、オブジェクトリファレンスを作成してクライアントに通知します。
9. クライアントアプリケーションは、オペレーションop()を要求します。
10. POAは、AOMを検索し、獲得したインスタンスでServantに対してオペレーションop()の実行を依頼します。
11. POAは、オペレーションop()の結果をクライアントに通知します。

(3)Factory-2方式

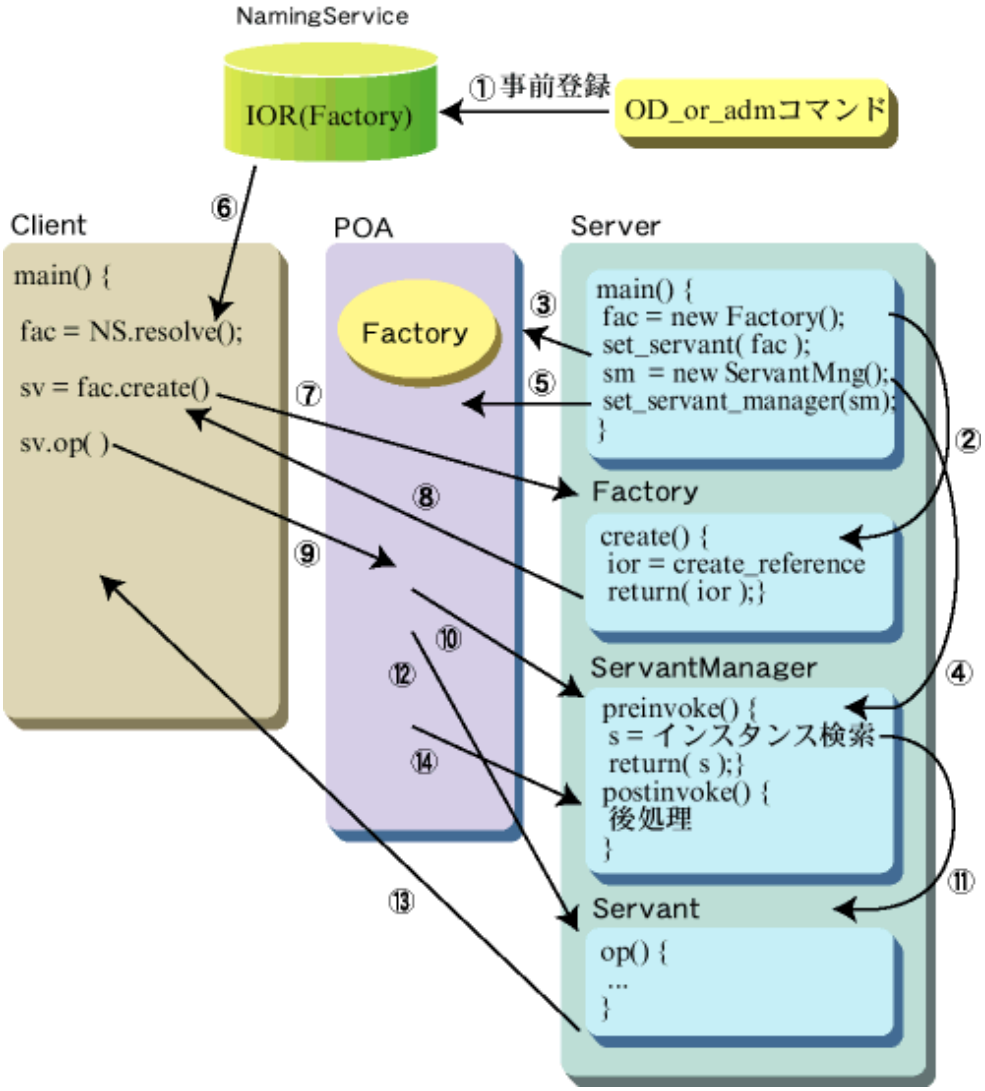
クライアントごとにインスタンス管理をPOAオブジェクトで行う方式です。Factory内でオブジェクトリファレンスを作成し、リクエスト受信時にServantManagerオブジェクト内でインスタンスを作成します。



1. OD_or_admコマンドで事前にFactoryのオブジェクトリファレンスをNamingServiceに登録します。
2. サーバアプリケーションは、Factoryを作成します。
3. サーバアプリケーションは、FactoryをDefault ServantとしてPOAに登録します。
4. サーバアプリケーションは、ServantManagerを作成します。
5. サーバアプリケーションは、ServantManagerをPOAに登録します。
6. クライアントアプリケーションは、NamingServiceからFactoryのオブジェクトリファレンスを獲得します。
7. クライアントアプリケーションは、オペレーションcreateをFactoryに要求します。
8. Factoryは、Servantのオブジェクトリファレンスを作成し、オブジェクトリファレンスをクライアントに通知します。
9. クライアントアプリケーションは、オペレーションop()を要求します。
10. POAはAOMを検索するが、未登録の要求であるため、ServantManagerを呼び出します。
11. ServantManagerは、該当するServantを検索して返します。
12. POAは、ServantManagerからServantを受け取り、AOMに登録します。
13. POAは、Servantに対してオペレーションop()の実行を依頼します。
14. POAは、オペレーションop()の結果をクライアントに通知します。

(4)ユーザインスタンス管理方式

インスタンス管理をユーザの作成するServantManagerオブジェクトに任せる方式です。



1. OD_or_admコマンドで事前にFactoryのオブジェクトリファレンスをNamingServiceに登録します。
2. サーバアプリケーションは、Factoryを作成します。
3. サーバアプリケーションは、FactoryをDefault ServantとしてPOAに登録します。
4. サーバアプリケーションは、ServantManagerを作成します。
5. サーバアプリケーションは、ServantManagerをPOAに登録します。
6. クライアントアプリケーションは、NamingServiceからFactoryのオブジェクトリファレンスを獲得します。
7. クライアントアプリケーションは、オペレーションcreateをFactoryに要求します。
8. FactoryはServantのオブジェクトリファレンスを作成し、オブジェクトリファレンスをクライアントに通知します。
9. クライアントアプリケーションは、オペレーションop()を要求します。
10. POAは、ServantManagerのpreinvoke()を呼び出します。
11. ServantManagerはServantのインスタンスを検索し、POAに通知します。

12. POAは、獲得したインスタンスでServantに対してオペレーションop()の実行を依頼します。
13. POAは、オペレーションop()の結果をクライアントに通知します。
14. POAは、ServantManagerのpostinvoke()を呼び出します。

5.12.3 アプリケーション形態の比較

	デフォルトインスタンス方式	Factory-1方式	Factory-2方式	ユーザインスタンス管理方式
インスタンス管理	しない	する	する	する
インスタンス管理者	—	POAオブジェクト	POAオブジェクト	ユーザが作成したServantManagerオブジェクト
性能	優れている	Factory-2方式より優れる	Factory-1方式より劣る	ユーザアプリによる
資源	少ない	Factory-2方式より多い	Factory-1方式より少ない	ユーザアプリによる
開発の難易度	やさしい	普通	普通	難しい

5.13 サーバアプリケーションのプログラミング（POA概要）

5.13.1 POAとは

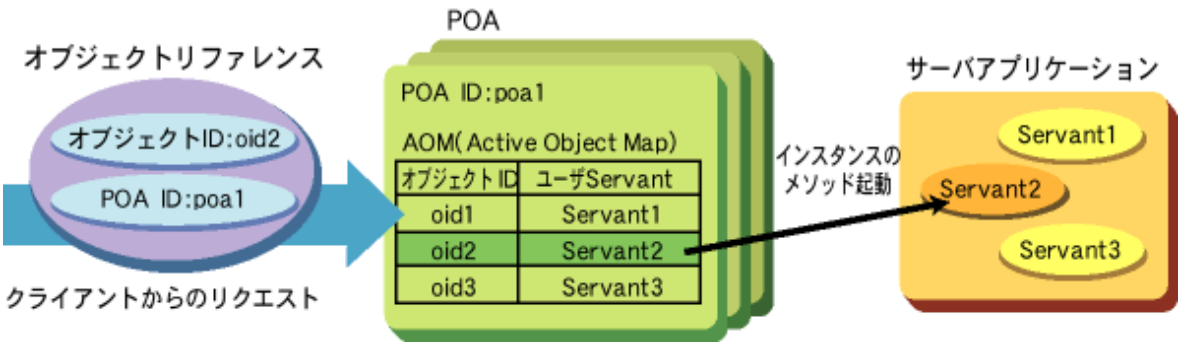
POA (Portable Object Adapter)は、CORBA2.2で新しく標準的なオブジェクトアダプタとして採用された仕様です。POAとしてオブジェクトアダプタの仕様が明確化されたことにより、ORB製品ごとの仕様上の差違が排除されるため、異なるORB製品へのサーバアプリケーションの移植が容易となります。

■POAの特徴

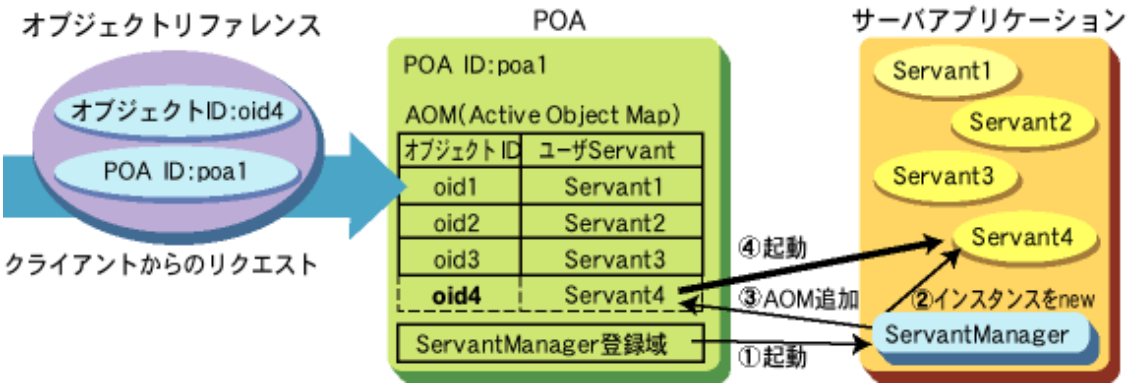
インスタンス管理

POAオブジェクト内にAOM(Active Object Map:インスタンスの管理テーブル)を持ち、インスタンス管理を行います。サーバアプリケーションは、以下の2とおりのインスタンス管理を行うことが可能です。

- サーバアプリケーションは、インスタンスを事前に作成してAOMに登録しておきます。クライアントからのリクエストを受信したときに、AOMのインスタンスが検索されServantオブジェクトが実行されます。



- クライアントからのリクエストを受信したときに、POAオブジェクトはユーザ実装のServantManagerを呼び出します。ServantManagerでは、インスタンスを作成してAOMに登録します。次に、Servantオブジェクトが実行されます。



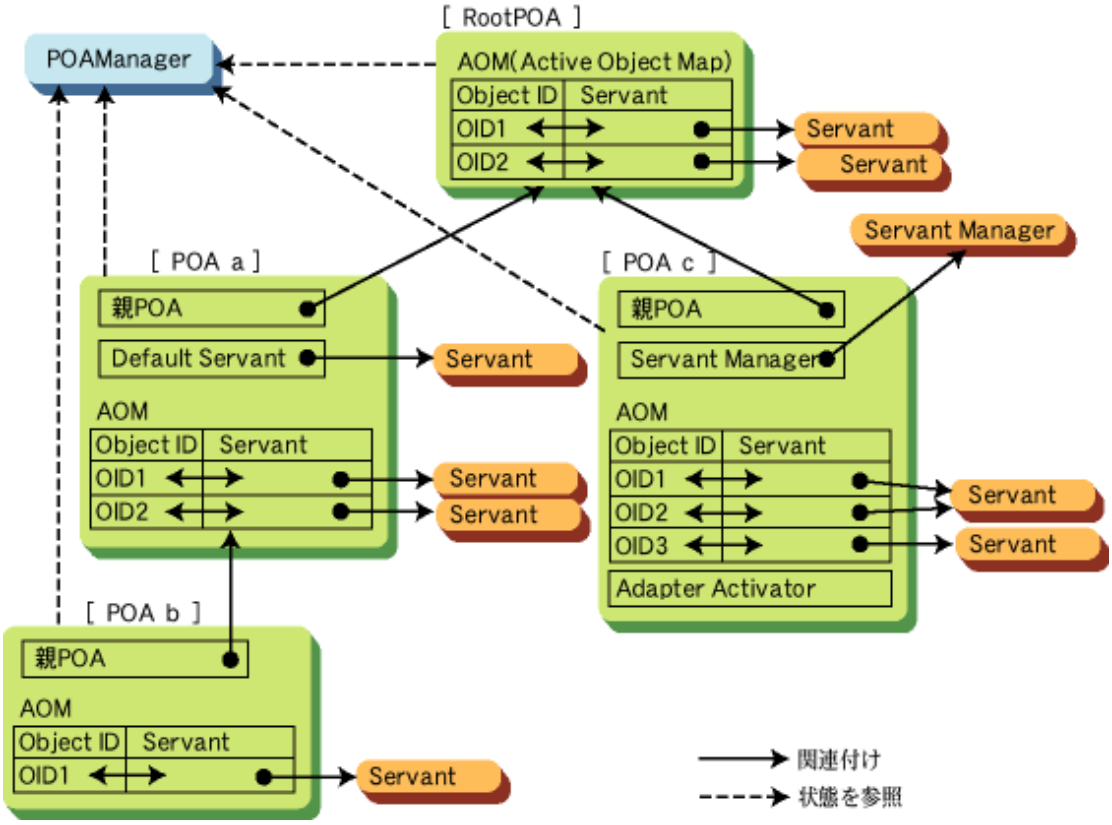
POAポリシ

POAに対してポリシを設定することで、POAごとにインスタンス管理方法、オブジェクトの生存期間、多重動作などのポリシを指定できます。POAポリシとして、以下の種類があります。詳細については、“5.13.3 POAオブジェクト”を参照してください。

- Servant関連付けポリシ: ServantRetentionPolicy
- リクエスト処理ポリシ: RequestProcessingPolicy
- 暗黙的活性化ポリシ: ImplicitActivationPolicy
- ID割り当てポリシ: IdAssignmentPolicy
- オブジェクトID一意性ポリシ: IdUniquenessPolicy
- 生存期間ポリシ: LifespanPolicy
- スレッドポリシ: ThreadPolicy

5.13.2 POAのアーキテクチャ

POAは、以下のような仕組みを提供します。



■POAオブジェクト

POAオブジェクトは、オブジェクトID、Servantオブジェクト（ユーザ提供のサーバ実装のインスタンス）の関係をAOM（Active Object Map）上で管理し、クライアントからの要求メッセージに従ってServantオブジェクトのメソッドを実行します。

■POAManagerオブジェクト

POAの要求メッセージ処理の受付けに関する状態を制御します。

■Servantオブジェクト

サーバアプリケーションが提供するインタフェースを実装し、リクエストに対する処理を実際に行う、ユーザ作成のオブジェクトを示します。

■Default Servantオブジェクト

Default Servantオブジェクトは、RequestProcessingポリシにUSE_DEFAULT_SERVANTが設定されて、さらに、リクエストに対応するServantオブジェクトがPOAのAOM(Active Object Map)に存在しない場合に、POAはDefault Servantオブジェクトのメソッドを実行します。

■ServantManagerオブジェクト

ServantManagerオブジェクトは、POAオブジェクトより呼び出されて、Servantオブジェクトの作成／検索を行います。
ServantManagerオブジェクトは、RequestProcessingポリシにUSE_SERVANT_MANAGERが設定されているときにPOAオブジェクトが呼び出しますが、以下の2種類があります。

- **ServantActivator**（ServantRetentionポリシがRETAINの場合）

  リクエストに対応するServantオブジェクトがPOAオブジェクトのAOM（ActiveObjectMap）に存在しない場合に、POAオブジェクトはServantActivator オブジェクトにServantオブジェクトの生成を要求します。ServantActivatorオブジェクトはServantオブジェクトを返し、POAオブジェクトはServantオブジェクトのメソッドを実行します。
  このServantActivatorオブジェクトは、ユーザが作成する必要があります。

- **ServantLocator**(ServantRetentionポリシがNON_RETAINの場合)

  リクエスト受信時に、POAオブジェクトは対応するServantオブジェクトをServantLocatorオブジェクトに問い合わせます。ServantLocatorオブジェクトはServantオブジェクトを返し、POAオブジェクトはServantオブジェクトのメソッドを実行します。
  このServantLocatorオブジェクトは、ユーザが作成する必要があり、対応するServantオブジェクトを管理する必要があります。

詳細については、“5.13.10 ServantManagerオブジェクト”を参照してください。

■AdapterActivatorオブジェクト

リクエストの受信時に、対応するPOAオブジェクトが存在しない場合に、POAオブジェクトを生成します。このAdapterActivatorオブジェクトは、ユーザが作成する必要があります。

■オブジェクトID

オブジェクトIDは、POAオブジェクトが対応するServantオブジェクトを特定するための値です。オブジェクトIDの値は、IdAssignmentポリシによって、POAオブジェクトによって割り当てられる場合と、ユーザアプリケーションにより割り当てられる場合があります。オブジェクトIDは、オブジェクトリファレンス中に含められるため、クライアントからは意識する必要はありません。

■POA ID

POA IDは、リクエストの受信時にPOAオブジェクトを特定するための値です。POA IDは、オブジェクトリファレンス中に含められるため、クライアントからは意識する必要はありません。

5.13.3 POAオブジェクト

POAオブジェクトを作成する際には、まずPOAポリシリストを作成する必要があります。次に、ポリシリストを指定してPOAオブジェクトを作成します。

■POAポリシ

POAオブジェクトの作成時に、以下のPOAポリシを設定できます。

POAポリシ	意味	Value	Valueの説明	備考
ServantRetention: Servant関連付けポリシ	AOMテーブルをPOAオブジェクト内で持つか持たないかを指定します。つまり、POAオブジェクト内でインスタンス管理を行うかどうかを指定できます。 “5.13.7 リクエスト処理”参照。	RETAIN (注1)	ActiveObjectMapにActiveなServantオブジェクトを憶えます。	
		NON_RETAIN	ActiveObjectMapを使いません。	
RequestProcessing: リクエスト処理ポリシ	リクエスト受信時に対象インスタンスをAOM内に持ってない、またはAOM自身を保持しないときの振る舞いを規定するポリシです。 “5.13.7 リクエスト処理”参照。	USE_ACTIVE_ OBJECT_MAP_ ONLY (注1)	リクエスト処理にActiveObjectMapだけ使用します。	

POAポリシ	意味	Value	Valueの説明	備考
		USE_DEFAULT_SERVANT	リクエストの処理をDefault Servantオブジェクトに任せます。	
		USE_SERVANT_MANAGER	ServantManagerオブジェクトにより目的のServantオブジェクトを見つけます。必要であれば、activateも行われます。	
ImplicitActivation: 暗黙的活性化ポリシ	Servantの活性化のモードを以下のように指定可能です。 ・自動的にServantがactivateされます。 ・自動的にServantがactivateされません(アプリケーションが明示的にServantをPOAに登録します)。 “5.13.8 暗黙的活性化（Implicit Activation）”参照。	IMPLICIT_ACTIVATION (注1)	自動的にServantオブジェクトがactivateされます。	
		NO_IMPLICIT_ACTIVATION	自動的にServantオブジェクトがactivateされません。サーバアプリケーションが明示的にServantオブジェクトを生成し、POAオブジェクトに登録します。	
IdAssignment: ID割り当てポリシ	オブジェクトIDをシステム(POAオブジェクト)が自動設定するか、ユーザが設定するかを指定できます。 USER_IDはアプリケーションで、オブジェクトIDに固有の意味を持たせたいときに有効です。 “5.13.6 オブジェクトの活性化”参照。	SYSTEM_ID (注1)	POAオブジェクトがオブジェクトIDを付けます。	
		USER_ID	ユーザがオブジェクトIDを付けます。	
IdUniqueness: オブジェクトID一意性ポリシ	Servantにユニークなを付けるか、重複したIDを許すかを指定できます。 “5.13.5 オブジェクトリファレンス、オブジェクトID、Servantオブジェクトの関連”参照。	UNIQUE_ID (注1)	Servantオブジェクトにユニークなを付けます。	
		MULTIPLE_ID	Servantオブジェクトに重複したIDを許します。	
Lifespan: 生存期間ポリシ	トランジェントタイプのオブジェクトを扱うか、パーシステントタイプのオブジェクトを扱うかを指定可能です。	TRANSIENT (注1)	トランジェントタイプのオブジェクトを扱います(サーバlifespan = オブジェクトlifespan)。	PERSISTENTは指定不可
		PERSISTENT	パーシステントタイプのオブジェクトを扱います(サーバlifespan < オブジェクトlifespan)。	
Thread: スレッドポリシ	スレッド処理をORBに任せるか、シングルスレッドで動作させるかを指定できます。	ORB_CTRL_MODEL(注1)	スレッド処理をORBに任せます。	このPOAポリシの設定値は無効(注2)
		SINGLE_THREAD_MODEL	シングルスレッドで動作させます。	

注1）デフォルト値
注2）インプリメンテーションリポジトリ登録時のプロセス最大多重度(proc_conc_max)、スレッド初期多重度(thr_conc_init)の設定内容に依存します。

POAポリシの生成例を以下に示します。

例

```
// POAポリシリスト作成
org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[4];
policies[0] =  Poa.create_servant_retention_policy(
    org.omg.PortableServer.ServantRetentionPolicyValue.NON_RETAIN );
policies[1] =  Poa.create_request_processing_policy(
    org.omg.PortableServer.RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
policies[2] =  Poa.create_id_assignment_policy(
    org.omg.PortableServer.IdAssignmentPolicyValue.SYSTEM_ID );
policies[3] =  Poa.create_id_uniqueness_policy(
    org.omg.PortableServer.IdUniquenessPolicyValue.MULTIPLE_ID );
```

注) ソース内のPoa:POAクラスのインスタンス

■POAオブジェクトの生成

RootPOAオブジェクト

POAオブジェクトを使用するためには、まずRootPOAオブジェクトを生成する必要があります。POAオブジェクトは、RootPOAオブジェクトをルートとして、子POAオブジェクトや孫POAオブジェクトを生成することでツリー構成とすることができます。RootPOAオブジェクトは、ORBがサーバの初期化時に生成します。RootPOAオブジェクトのPOAオブジェクトは、以下のように取得します。

例

```
// ORBの生成と初期化
org.omg.CORBA.ORB  Orb = org.omg.CORBA.ORB.init( args, null );

// RootPOAオブジェクトのオブジェクトリファレンスの取得
org.omg.CORBA.Object  _tmpObj = Orb.resolve_initial_references( "RootPOA" );
// RootPOAオブジェクトのPOAオブジェクト獲得
org.omg.PortableServer.POA  rootPOA  = org.omg.PortableServer.POAHelper.narrow( _tmpObj );
```

RootPOAオブジェクトには、以下のPOAポリシが設定されています。

- ServantRetentionPolicy: RETAIN
- RequestProcessingPolicy: USE_ACTIVE_OBJECT_MAP_ONLY
- ImplicitActivationPolicy: IMPLICIT_ACTIVATION
- IdAssignmentPolicy: SYSTEM_ID
- IdUniquenessPolicy: UNIQUE_ID
- LifespanPolicy: TRANSIENT
- ThreadPolicy: ORB_CTRL_MODEL

ユーザアプリケーションで上記のPOAポリシを使用してインスタンス管理を行う場合は、RootPOAオブジェクトをそのまま使用することが可能です。しかし、新しいPOAオブジェクト(RootPOAオブジェクトの子孫POAオブジェクト)を生成して固有のPOAポリシを設定することにより、RootPOAオブジェクトとは異なるPOAポリシでインスタンス管理を行うことが可能となります。

子孫POAオブジェクト

新しいPOAオブジェクトを生成するには、すでに存在するPOAオブジェクト上でcreate_POA()メソッドを呼び出します。この結果、create_POA()を発行したPOAオブジェクトと新しく生成されたPOAオブジェクトは親子関係となります。アプリケーションで最初に生成するPOAオブジェクトはRootPOAオブジェクトであるため、その他のPOAオブジェクトはすべてRootPOAオブジェクトの子孫POAオブジェクトとなります。
create_POA()の引数としてPOAポリシリスト(Policyクラスの配列)を渡すことで、生成した子POAオブジェクトのポリシを設定することが可能です。

例

```
// 子POAオブジェクト(Poa1)の生成
orb.omg.PortableServer.POA     Poa1 = rootPOA.create_POA("poa_1", poamanager, policies1);
// 孫POAオブジェクト(Poa2)の生成
orb.omg.PortableServer.POA     Poa2 = Poa1.create_POA("poa_2", poamanager, policies2);
```

注)ソース内のpoa_1、poa_2:生成するPOAの名前
ソース内のpoamanager:POAManagerクラスのインスタンス
policies1、policies2:Policyクラスのインスタンス

上記のように、POAオブジェクトは階層構造をもたせることが可能です。オブジェクトの生成、オブジェクトと管理するPOAオブジェクトとの関係については、“5.14 サーバアプリケーションと環境設定との関連付け”を参照してください。destroy()メソッドによりあるPOAオブジェクトが破棄された場合、その子孫POAオブジェクトはすべて破棄されます。

■POAポリシの設定値の組み合わせ

ポリシリストに設定したPOAポリシの各設定値は、それらの組み合わせが妥当であるかどうかがcreate_POA()メソッド発行時にチェックされます。もし、設定値の組み合わせが妥当でない場合は、例外が発生します。
POAポリシの設定値の組合せを以下に示します。妥当性欄で“*”の部分はこの値が設定された場合、create_POA()メソッド実行時にorg.omg.PortableServer.POAPackage.InvalidPolicy例外が発生することを示しています。
なお、LifespanPolicyの設定値はTRANSIENTとしてください。また、ThreadPolicyの設定値は無視されます。

ポリシ	Servant Retention	Request Processing	Implicit Activation	IdAssignment	IdUniqueness	妥当性
設定値	RETAIN	USE_ACTIVE_OBJECT_MAP_ONLY	IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	—	*
			NO_IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	
					MULTIPLE_ID	

ポリシ	Servant Retention	Request Processing	Implicit Activation	IdAssignment	IdUniqueness	妥当性
		USE_DEFAULT_SERVANT	IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	*
					MULTIPLE_ID	
				USER_ID	－	*
			NO_IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	*
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	*
					MULTIPLE_ID	
		USE_SERVANT_MANAGER	IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	－	*
			NO_IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	
					MULTIPLE_ID	
	NON_RETAIN	USE_ACTIVE_OBJECT_MAP_ONLY	－	－	－	*
		USE_DEFAULT_SERVANT	－ (注)	SYSTEM_ID	UNIQUE_ID	*
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	*
					MULTIPLE_ID	
		USE_SERVANT_MANAGER	－ (注)	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	
					MULTIPLE_ID	

注) ServantRetentionPolicyにNON_RETAINが設定されている場合、ImplicitActivationPolicyの設定値は無視されます。

■POAポリシのデフォルト設定値

POAはデフォルトのポリシ設定値として以下の値を持っています。

ポリシ	デフォルト値
ServantRetention	RETAIN
RequestProcessing	USE_ACTIVE_OBJECT_MAP_ONLY
ImplicitActivation	NO_IMPLICIT_ACTIVATION
IdAssignment	SYSTEM_ID
IdUniqueness	UNIQUE_ID
Lifespan	TRANSIENT
Thread	ORB_CTRL_MODEL

create_POA()メソッドを使用して新しいPOAを生成する際は、デフォルトの設定値から変更したいポリシだけを設定することが可能です。以下の例では、USE_DEFAULT_SERVANTにRequestProcessingPolicyを、MULTIPLE_IDにIdUniquenessPolicyを設定した子POAを生成しています。この場合、他のポリシの設定値は、上記のデフォルトの設定値のままとなります。

例

```
// POAポリシリスト作成
org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[2];
policies[0] =  Poa.create_request_processing_policy(
    org.omg.PortableServer.RequestProcessingPolicyValue.USE_DEFAULT_SERVANT);
policies[1] =  Poa.create_id_uniqueness_policy(
    org.omg.PortableServer.IdUniquenessPolicyValue.MULTIPLE_ID );
// 子POAオブジェクトの生成
org.omg.PortableServer.POA  newPOA = Poa.create_POA("poa_1", poamanager, policies);
```

注)ソース内のPoa:POAクラスのインスタンス
　ソース内のpoamanager:POAManagerクラスのインスタンス

なお、create_POA()メソッドの第3パラメタにnullを指定した場合、生成される子POAのポリシはデフォルトの設定値となります。

5.13.4 オブジェクトリファレンスの生成

オブジェクトリファレンスは、以下の方法で生成できます。

■Servantオブジェクトの活性化前に作成する方法(直接作成)

サーバアプリケーションは、直接オブジェクトリファレンスをPOAオブジェクトのcreate_reference()／create_reference_with_id()メソッドで作成できます。この操作は、オブジェクトリファレンスだけを作成します。実際のアクティブなServantオブジェクトは、後からServantManagerオブジェクトで生成される場合もあります。

例

```
// オブジェクトリファレンスの生成
org.omg.CORBA.Object  Obj = Poa.create_reference( "IDL:Intfid11:1.0" );

// ID指定によるオブジェクトリファレンスの生成
String userid = "USERID";
org.omg.CORBA.Object Obj = Poa.create_reference_with_id( userid.getBytes(), "IDL:Intfid11:1.0" );
```

注)ソース内のPoa:POAクラスのインスタンス
“IDL:Intfid11:1.0”:インタフェースリポジトリID

■Servantオブジェクトの活性化後に作成する方法(間接作成)

サーバアプリケーションは、activate_object()／activate_object_with_id()メソッドよりServantオブジェクトを活性化します。一度、Servantオブジェクトが活性化されると、サーバアプリケーションはServantオブジェクト／オブジェクトIDを指定し、servant_to_reference()／id_to_reference()メソッドを発行することにより容易にオブジェクトリファレンスを作成できます。また、POAオブジェクトがIMPLICIT_ACTIVATIONポリシで作成されていれば、servant_to_reference()を使用してリファレンスを生成した場合、自動的にServantオブジェクトがactivateされて、リファレンスが有効となります。

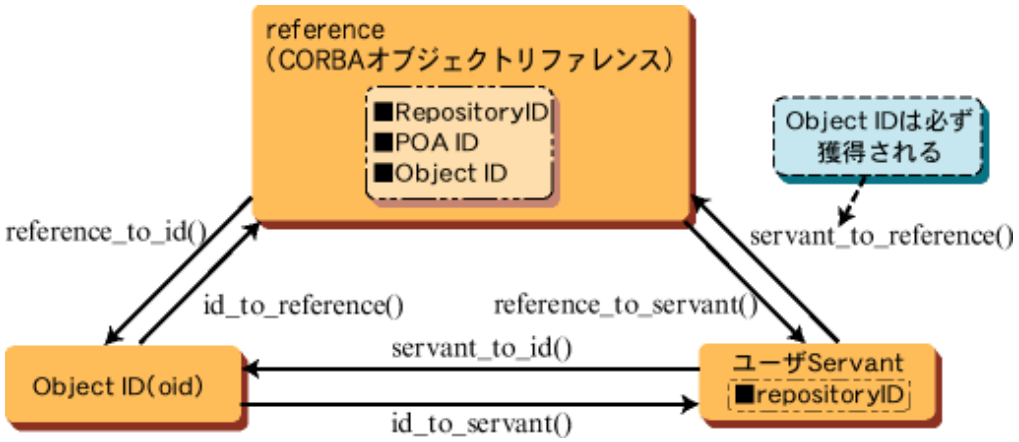
例

```
// Servantの生成
org.omg.PortableServer.Servant svt = new UserServant();

// IMPLICIT_ACTIVATIONポリシを指定している場合、自動的にactivateされる
org.omg.CORBA.Object Obj =  Poa.servant_to_reference( svt );
```

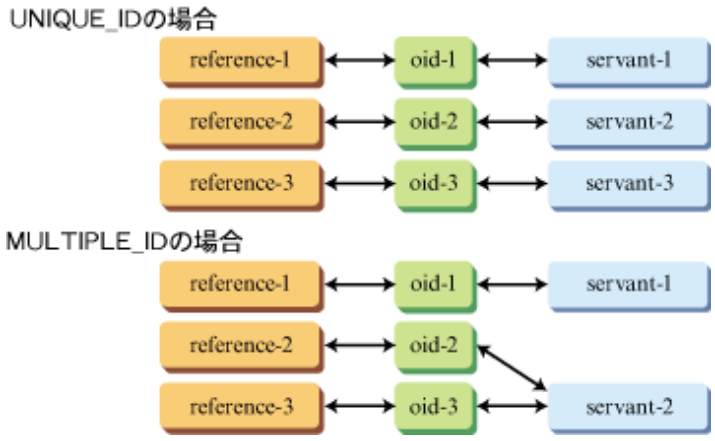
注)ソース内のPoaオブジェクト:POAクラスのインスタンス

一度オブジェクトリファレンスがクライアントに通知されると、クライアントの側からみるとそのオブジェクトリファレンスはオブジェクトの身元を持っていることになります。クライアントプログラムがそのリファレンスを使う限り、そのオブジェクトリファレンスで作られるリクエストは同一のオブジェクトのインスタンスに送信されます。

5.13.5 オブジェクトリファレンス、オブジェクトID、Servantオブジェクトの関連



オブジェクトリファレンス、オブジェクトID(oid)、Servantオブジェクトの関係は、IdUniquenessPolicyにより決まります。オブジェクトリファレンスはoidを包含しているため、referenceとoidは1対1の関係となります。oidとServantオブジェクトの関係は、UNIQUE_IDの場合、Servantオブジェクトごとに一意のoidが割り当てられます。MULTIPLE_IDの場合、同じServantオブジェクトへの複数のoidが割り当て可能です。このことは、具体的にはoidを生成するservant_to_id()／servant_to_reference()の動作に影響を与えます。



5.13.6 オブジェクトの活性化

オブジェクトが活性状態とは、POAオブジェクトの管理するAOM(Active Object Map)に、あるServantオブジェクトがオブジェクトIDと関連付けられて登録されている状態のことです。
Servantオブジェクトは、インスタンスとして生成されただけでは、POAオブジェクトに認識されないため、activate_object()／activate_object_with_id()メソッドを使用して、オブジェクトIDとの関連付けをPOAオブジェクトに登録する必要があります。Servantオブジェクトを活性化する方法を以下に示します。

■明示的にServantオブジェクトを活性化する方法

Servantオブジェクトの作成(new)後に、activate_object()／activate_object_with_id()メソッドでAOMに登録します。

例

```
// Servantの生成
org.omg.PortableServer.Servant svt = new UserServant();

// Servantの活性化
Poa.activate_object ( svt );

// ID指定によるServantの活性化
String userid = "USERID";
Poa.activate_object_with_id( userid.getBytes(), svt );
```

注)ソース内のPoaオブジェクト:POAクラスのインスタンス

なお、活性化の際にはオブジェクトIDが必要となりますが、オブジェクトIDの割り付けは、IdAssignmentPolicyに従って処理されます。

- SYSTEM_IDの場合
  POAオブジェクトが自動的にオブジェクトIDを割り付けます。Servantオブジェクトを活性化する場合は、activate_object()を使用します。
- USER_IDの場合
  ユーザがオブジェクトIDを割り付けます。Servantオブジェクトを活性化する場合は、activate_object_with_id()を使用します。

■暗黙的にServantオブジェクトを活性化する方法

POAオブジェクトがIMPLICIT_ACTIVATIONポリシを持っている場合は、servant_to_reference()メソッドを使用してリファレンスを生成すると、自動的にServantオブジェクトが活性化されて、オブジェクトリファレンスが通知されます。暗黙的活性化については、“5.13.8 暗黙的活性化(Implicit Activation)”を参照してください。

例

```
// Servantの生成
org.omg.PortableServer.Servant svt = new UserServant();

// IMPLICIT_ACTIVATIONポリシを指定している場合、自動的にactivateされる
org.omg.CORBA.Object Obj =  Poa.servant_to_reference( svt );
```

注) ソース内のPoaオブジェクト:POAクラスのインスタンス

なお、暗黙的活性化の場合は、オブジェクトIDはPOAオブジェクトにより自動的に割り付けます。このため、SYSTEM_IDにIdAssignmentPolicyを設定しておく必要があります。

5.13.7 リクエスト処理

■POAオブジェクトの検索

クライアントからのリクエスト受信時に、インタフェースリポジトリIDに対応するPOAオブジェクトがサーバプロセス上に存在しなければ、親のPOAオブジェクトはAdapterActivatorオブジェクトを呼び出します。AdapterActivatorオブジェクトでは、要求されたPOAオブジェクトを作成します。AdapterActivatorオブジェクトはユーザが作成する必要があり、事前にPOAオブジェクトに登録しておく必要があります。もし、AdapterActivatorオブジェクトが登録されていない場合、クライアントはOBJECT_NOT_EXIST例外を受け取ります。

■Servantオブジェクトの検索

クライアントからのリクエストを受信時に、POAオブジェクトはAOMに登録されているServantオブジェクトを検索します。なお、AOMを使用するかどうかは、対象のPOAオブジェクトがRETAINポリシ(使用する)とNON_RETAINポリシ(使用しない)のどちらが指定されているかによります。
RETAINポリシが指定され、目的のServantオブジェクトがAOM内にみつかった場合、検索処理はそこで終了し、Servantオブジェクトのメソッドを実行します。Servantオブジェクトがみつからなかった場合や、NON_RETAINの場合は、以下のRequestProcessingポリシに従って処理します。

- **USE_DEFAULT_SERVANT**
  対象のServantオブジェクトがみつからなかった場合は、デフォルトのServantオブジェクトに処理を任せる方法です。この場合、あらかじめset_servant()によりデフォルトのServantオブジェクトを登録しておく必要があります。もし、Default Servantオブジェクトが登録されていなければ、OBJ_ADAPTERエラーとなります。

- **USE_SERVANT_MANAGER**
  対象のServantオブジェクトがみつからなかった場合は、ServantManagerオブジェクトにServantオブジェクトを検索する処理を任せる方法です。
  リクエストを扱えるServantオブジェクトを見つけるため、POAオブジェクトはincarnate()／preinvoke()を呼び出します(メソッド選択は、POAオブジェクトのNON_RETAIN／RETAINポリシに依存します)。ServantManagerオブジェクトは、見つけたServantオブジェクトをPOAオブジェクトに通知します。この場合、あらかじめset_servant_manager()によりServantManagerオブジェクトを登録しておく必要があります。もし、ServantManagerオブジェクトが登録されていなければ、OBJ_ADAPTERエラーとなります。

- **USE_ACTIVE_OBJECT_MAP_ONLY**
  この場合、OBJECT_NOT_EXISTエラーとなります。

5.13.8 暗黙的活性化（Implicit Activation）

Servantオブジェクトの活性化方法には、アプリケーションが明示的にServantオブジェクトを活性化させる方法に加え、POAオブジェクトによって暗黙的（Implicit）にServantオブジェクトを活性化させるIMPLICIT_ACTIVATIONポリシがあります。
IMPLICIT_ACTIVATIONポリシを使用するためには、SYSTEM_IDポリシとRETAINポリシを指定する必要があります。
IMPLICIT_ACTIVATIONポリシを指定することにより、servant_to_reference()やservant_to_id()などのServantオブジェクトからオブジェクトリファレンスやオブジェクトIDに変換するメソッドを使用した際に、POAオブジェクトがAOMにServantオブジェクトおよびオブジェクトIDを登録し、Servantオブジェクトを活性化できます。このときにUNIQUE_IDポリシが指定されていた場合、再度servant_to_reference()やservant_to_id()を使用した場合、前回と同じリファレンスやオブジェクトIDが返されます。MULTIPLE_IDポリシが指定されていた場合、毎回、異なったオブジェクトリファレンスやオブジェクトIDが返されます。これは、同じServantオブジェクトに複数のオブジェクトリファレンスやオブジェクトIDが割り付けられることを意味します。

5.13.9 POAManagerオブジェクト

POAManagerオブジェクトは、POAオブジェクトの生成時、create_POA()メソッドの引数として渡すことによりPOAオブジェクトと関連付けられます。POAManagerは、以下の4つの状態を持っています。POAオブジェクトは、POAManagerの状態に従って処理を決定します。この状態は、activate()、hold_requests()、discard_requests()、deactivate()などのメソッドで変更することが可能です。

holding状態

受け取ったメッセージをキューイングします。
このとき使用されるキューは、インプリメンテーションリポジトリで定義されているインプリメンテーション単位に作成されます。“OLTPサーバ運用ガイド”の“ワークユニットの機能”－“CORBAワークユニット”－“キュー制御”を参照してください。

active状態

受け取ったメッセージを処理します。

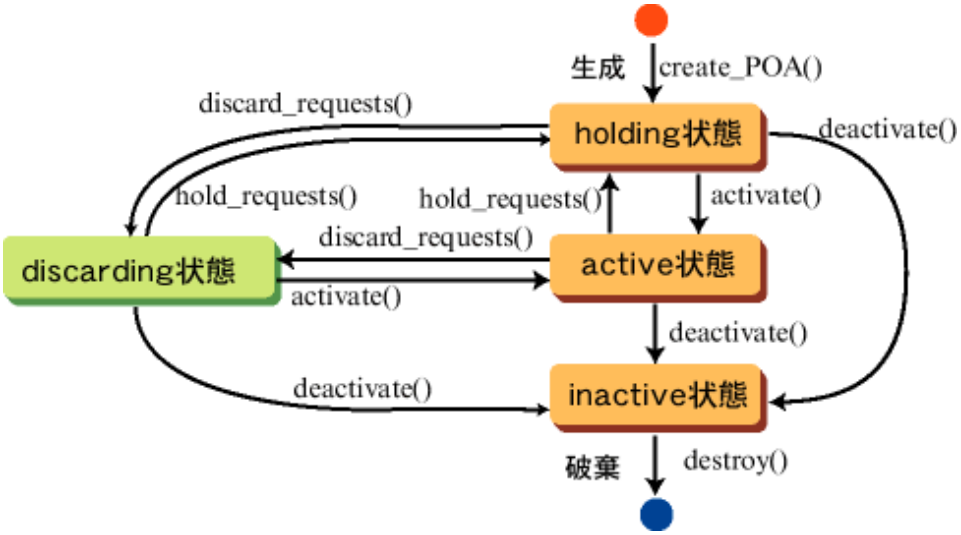
discarding状態

受け取ったメッセージを捨てます。この場合、クライアントへはエラーが通知されます。

inactive状態

shutdown直前の状態です。

POAManagerオブジェクトの状態遷移を以下に示します。



注意

POAManagerオブジェクトの初期状態は、Holding状態です。
CORBAサービスのJava言語マッピングでは、初期状態からactive状態を経由せずに、discarding状態にすることはできません。active状態とした後は、上図のように状態遷移を行うことができます。

5.13.10 ServantManagerオブジェクト

ServantManagerオブジェクトは、リクエスト受信時にServantオブジェクトの生成／削除や検索を可能にする機能です。ServantManagerオブジェクトの各メソッドの実装は、ユーザ自身で作成します。作成したServantManagerオブジェクトは、POAオブジェクトに登録しておきます。
POAオブジェクトがリクエストを受信した際、AOMにオブジェクトIDが一致するServantオブジェクトが登録されていない場合に、POAオブジェクトはServantManagerオブジェクトのメソッドを呼び出し、Servantオブジェクトの生成／削除、検索を依頼します。
ServantManagerオブジェクトは、Servantオブジェクトの生成／削除を行うServantActivatorオブジェクトとServantオブジェクトを検索するServantLocatorオブジェクトの2種類が存在します。POAオブジェクトは、ServerRetentionポリシがRETAINの場合はServantActivatorオブジェクトを呼び出し、NON_RETAINの場合はServantLocatorオブジェクトを呼び出します。

(1)ServantActivatorオブジェクト

作成するServantActivatorクラスは、以下のようにorg.omg.PortableServer.ServantActivatorクラスを継承させる必要があります。また、メソッドとしてincarnate()、etherealize()を実装している必要があります。

【ServantActivatorクラスの実装形式】

```
class UserServantActivator extends org.omg.CORBA.LocalObject
      implements org.omg.PortableServer.ServantActivator{
    public org.omg.PortableServer.Servant incarnate(...) {...}
    public void etherealize(...) {...}
}
```

POAオブジェクトは、Servantオブジェクトの生成を依頼する場合、ServantActivatorオブジェクトのincarnate()メソッドを呼び出し、Servantオブジェクトの削除を依頼する場合は、etherealize()メソッドを呼び出します。POAオブジェクトから渡されるパラメタは、必要に応じて各メソッドの実装部で使用できます。

- **incarnate (Servantオブジェクト生成)**
  POAは、リクエスト受信時にAOMに対象のServantオブジェクトが未登録の場合にincarnate()メソッドを呼び出します。incarnate()メソッドは、POAオブジェクトにServantオブジェクトを生成させ返します。

```
org.omg.PortableServer.Servant  incarnate(
    byte[] oid,
    org.omg.PortableServer.POA adapter )
```

  oid:リクエスト対象のオブジェクトのオブジェクトID
  adapter:呼び出し元のPOAオブジェクト

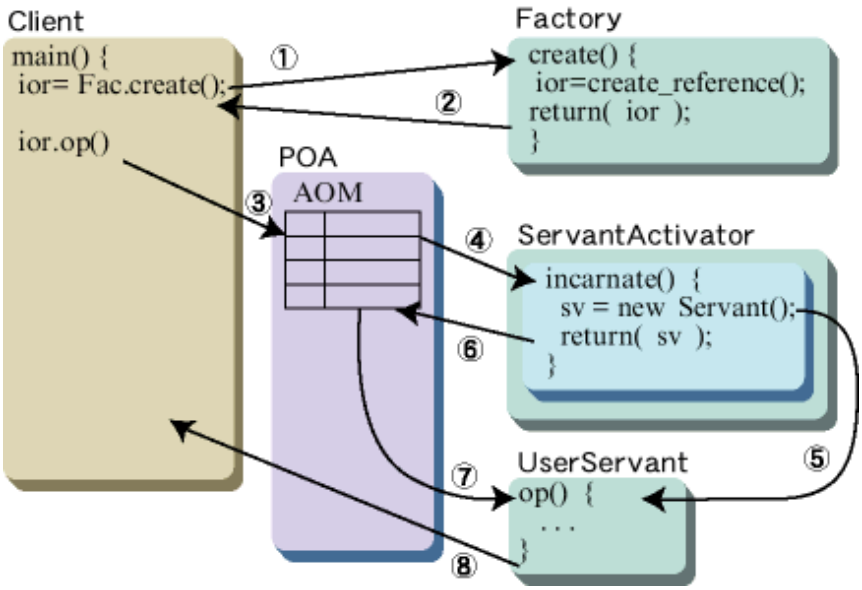
- **etherealize (Servantオブジェクト削除)**
  POAは、アプリケーションからdeactivate_object()メソッドが呼び出されたときに、etherealize()メソッドを呼び出します。POAオブジェクトが削除するServantオブジェクトをパラメタで指定します。etherealize()メソッドは、指定されたServantオブジェクトを削除します。

```
void etherealize(
    byte[] oid,
    org.omg.PortableServer.POA adapter,
    org.omg.PortableServer.Servant serv,
```

```
    boolean cleanup_in_progress,
    boolean remaining_activations );
```

oid：削除対象のServantオブジェクトに対応するオブジェクトID
adapter：呼び出し元のPOAオブジェクト
serv：削除対象のServantオブジェクト
cleanup_in_progress：POA.destoroy()から本メソッドが呼ばれた場合にtrue
remaining_activations：本メソッド呼出し時、常にfalse

ServantActivatorオブジェクトの具体的な処理の流れを以下に示します。



1. FactoryにServantのオブジェクトリファレンスの生成を依頼します。(注)
2. Factoryで生成したオブジェクトリファレンスをクライアントに返します。
3. Servantにオペレーションop()を要求します。
4. POAでは、AOMに未登録の要求であるためServantActivatorを呼び出します。
5. ServantActivatorでは、該当するServantを検索して返します。
6. ServantActivatorからServantを受け取り、AOMに登録します。
7. Servantに対して、オペレーションop()の実行を依頼します。
8. オペレーションop()の結果をクライアントに通知します。

注) Factoryは、リクエストごとにオブジェクトリファレンスやインスタンスを作成するオブジェクトであり、インスタンス管理を行う場合に必要となるオブジェクトです。

(2)ServantLocatorオブジェクト

作成するServantLocatorクラスは、以下のようにorg.omg.PortableServer.ServantLocatorクラスを継承させる必要があります。また、メソッドとしてpreinvoke()、postinvoke()を実装している必要があります。

【ServantLocatorクラスの実装形式】

```
class UserServantLocator extends org.omg.CORBA.LocalObject
      implements org.omg.PortableServer.ServantLocator{
    public org.omg.PortableServer.Servant preinvoke(...) {...}
    public void postinvoke(...) {...}
}
```

POAオブジェクトはクライアントからの要求を処理するため、毎回ServantLocatorオブジェクトのpreinvoke()メソッドを呼び出し、処理対象のServantオブジェクトを獲得します。POAオブジェクトはそのServantオブジェクトに処理を要求し、その処理が終わったら、ServantLocatorオブジェクトのpostinvoke()メソッドを呼び出し、ServantLocatorオブジェクトに後処理を依頼します。
POAオブジェクトから渡されるパラメタは、必要に応じて各メソッドの実装部で使用することができます。

- **preinvoke**
  POAオブジェクトは、処理を要求すべきServantオブジェクトを獲得するためにServantLocatorオブジェクトのpreinvoke()メソッドを呼び出します。preinvoke()メソッドは、POAオブジェクトにServantオブジェクトを返すと同時に、postinvoke()のための情報(cookie)も返します。

```
org.omg.PortableServer.Servant  preinvoke(
      byte[] oid,
      org.omg.PortableServer.POA adapter,
      String operation,
      org.omg.PortableServer.ServantLocatorPackage.CookieHolder the_cookie )
```

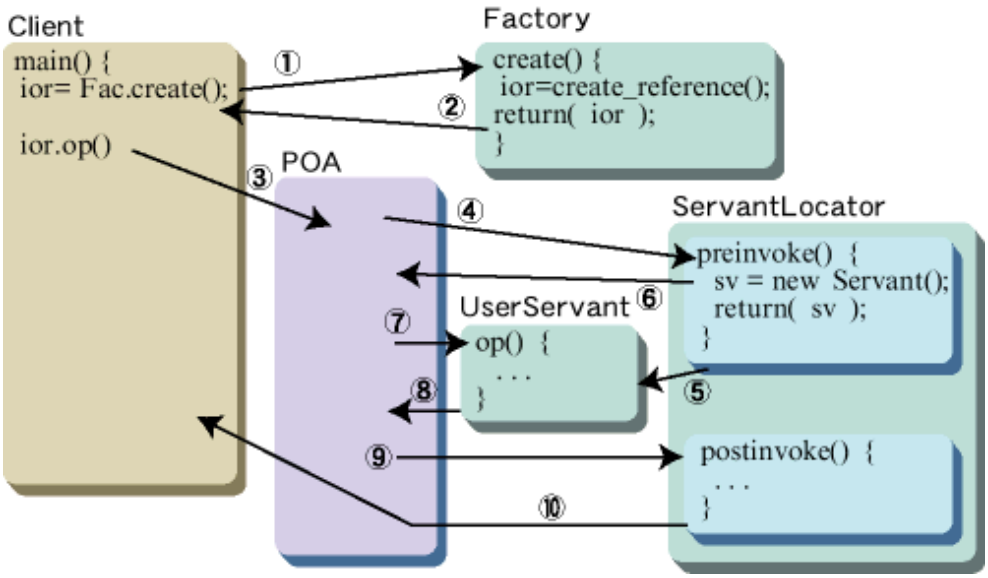
  oid:リクエスト対象のオブジェクトのオブジェクトID
  adapter:呼び出し元のPOAオブジェクト
  operation:リクエスト対象のオペレーションの名前
  the_cookie:Cookieオブジェクトを格納するためのCookieHolderオブジェクト

- **postinvoke**
  POAオブジェクトは、Servantオブジェクトの要求処理の終了直後にServantLocatorオブジェクトのpostinvoke()を呼び出します。その際、preinvoke()との対応をとるため、preinvoke()で返されたcookieを指定します。

```
void  postinvoke(
      byte[] oid,
      org.omg.PortableServer.POA adapter,
      String operation,
      java.lang.Object the_cookie,
      org.omg.PortableServer.Servant the_servant );
```

  oid:リクエスト対象のオブジェクトのオブジェクトID
  adapter:呼び出し元のPOAオブジェクト
  operation:リクエスト対象のオペレーションの名前
  the_cookie:preinvokeで返されたCookieオブジェクト
  the_servant:リクエストに対する処理を行ったServantオブジェクト

ServantLocatorの具体的な処理の流れを以下に示します。



1. FactoryにServantのオブジェクトリファレンスの生成を依頼します。
2. Factoryで生成したオブジェクトリファレンスをクライアントに返します。
3. Servantにオペレーションop()を要求します。
4. POAでは、ServantLocatorを呼び出します。
5. ServantLocatorのpreinvoke()でServantを検索します。
6. Servantを受け取ります。
7. Servantに対して、オペレーションop()の実行を依頼します。
8. オペレーションop()の結果をクライアントに通知します。
9. ServantLocatorのpostinvoke()を呼び出します。
10. クライアントへオペレーションを復帰します。

5.13.11 AdapterActivatorオブジェクト

AdapterActivatorオブジェクトは、リクエストを受信時にPOAオブジェクトを生成するために使用します。POAオブジェクトが生成したオブジェクトリファレンスには、オブジェクトIDに加え、生成元POAオブジェクトを示すPOAidが埋め込まれています。クライアントからのリクエストが送られてきた際、POA idに対応するPOAオブジェクトが検索され、そのPOAオブジェクトがリクエストに対する処理を行います。もし、みつからない場合、検索対象のPOAの親POAオブジェクトにAdapterActivatorオブジェクトが設定されているならば、そのAdapterActivatorオブジェクトに子POAオブジェクトの生成を依頼します。親POAにAdapterActivatorオブジェクトが設定されていない場合は、org.omg.CORBA.OBJ_ADAPTER例外がクライアントに通知されます。
AdapterActivatorオブジェクトは、ユーザが作成しPOAに登録する必要があります。
作成するAdapterActivatorクラスは、org.omg.PortableServer.AdapterActivatorクラスを継承させる必要があります。また、メソッドとしてunknown_adapter()を実装している必要があります。

【AdapterActivatorクラスの実装形式】

```
class UserAdapterActivator extends org.omg.CORBA.LocalObject
      implements org.omg.PortableServer.AdapterActivator {
    public boolean unknown_adapter(...){...}
}
```

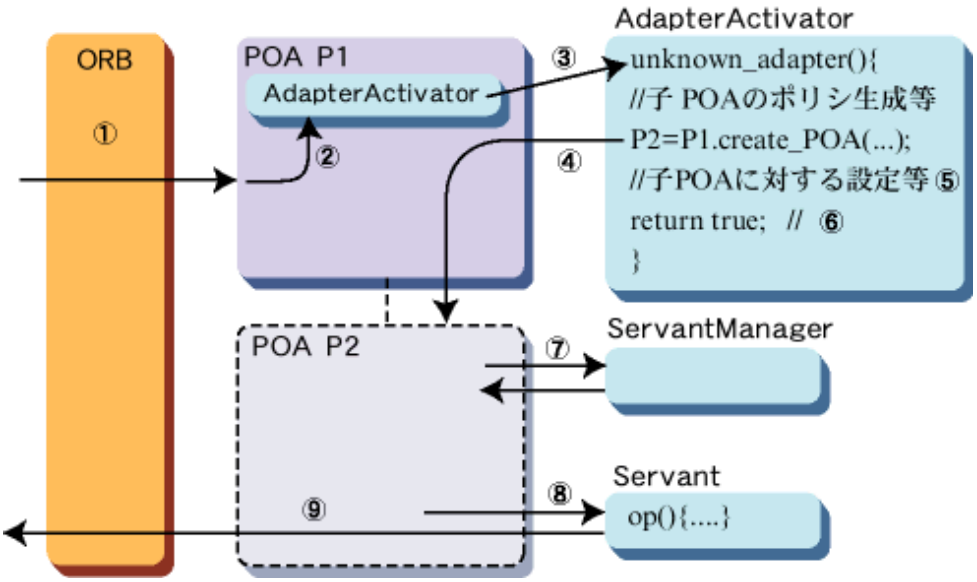
- **unknown_adapter**
  POAidに対応するPOAが検出できなかった場合は、AdapterActivatorオブジェクトのunknown_adapter()メソッドが呼び出されます。
  unknown_adapter()には、パラメタとしてPOAオブジェクトとStringオブジェクトが渡されます。
  unknown_adapter()内では、渡されたPOAオブジェクトの子POAをcreate_POA()を使用して生成します。その際のアダプタ名として、渡されたStringオブジェクトを使用します。
  子POAに対するポリシ設定や、その他の必要な処理(ServantやServantManagerの生成・登録など)もこのメソッド内に記述することが可能です。
  子POAの生成およびその他の実装した処理が成功した場合は、戻り値としてtrueを返します。そうでない場合は、falseを返します。trueを返した場合は、生成した子POAからServantのオペレーションが起動されます。falseを返した場合は、org.omg.CORBA.OBJECT_NOT_EXIST例外がクライアントに通知されます。

```
public boolean unknown_adapter( POA parent, String name )
```

  parent:生成するPOAの親POA
  name:生成するPOAのアダプタ名

AdapterActivatorオブジェクトのunknown_adapterメソッドは、org.omg.PortableServer.POA.find_POAメソッド発行時に呼び出すことも可能です。

AdapterActivatorの処理の流れを以下に示します。



1. クライアントからのリクエストを受信します。リクエストの対象となるオブジェクトリファレンスは、POAidを内包しています。
2. POAidに対応するPOAオブジェクトを検索します。存在しない場合は、既存のPOAから対象のPOAの親POAを検索します。
3. 親POAオブジェクトに設定されているAdapterActivatorクラスのインスタンスに対して、unknown_adapeter()メソッドが発行されます。
4. unknown_adapeter()の実装部では、パラメタで渡されたPOA(親POA)の子POAオブジェクトを生成します。以降、同じオブジェクトリファレンスに対するリクエストは、このPOAが処理します。
5. 生成した子POAに対する設定などを行います。ここでは、直接Servantを生成して子POAに登録することも可能であり、ServantManagerを生成し登録しておくことも可能です。
6. 子POAの生成に成功した場合は、trueを返します。
7. Servantの生成/登録を行います(ServantManager使用時)。

8. Servantのオペレーションを起動します。
9. オペレーションの実行結果を返します。

5.14 サーバアプリケーションと環境設定との関連付け

インプリメンテーション情報を登録するためのOD_impl_instコマンドとネーミングサービスへ登録するためのOD_or_admコマンドがあります。ここでは、サーバアプリケーションとOD_impl_instコマンド、OD_or_admコマンドとの関連付けについて説明します。

5.14.1 インプリメンテーション情報との関連付け

POAオブジェクトを使用したサーバアプリケーションを動作させるためには、OD_impl_instコマンドに指定したインプリメンテーションリポジトリIDとの関連付けを行う必要があります。この関連付けを行うには、以下の2とおりの方法があります。

1. 環境変数OD_IMPLIDへの指定
   プログラムの外からインプリメンテーションリポジトリIDを指定できるため、プログラムの独立性が高まります。
2. POAオブジェクトのアダプタ名への指定
   プログラムごとの環境変数の指定が不要であり、環境設定が容易となります。

(1)環境変数OD_IMPLIDへの指定

サーバアプリケーションの実行環境では、環境変数OD_IMPLIDへインプリメンテーションリポジトリID名を設定することで可能となります。たとえば、Windows(R)システムのバッチファイルでサーバアプリケーションを起動する場合、以下のように指定します。

例

<xxx.bat>

```
set OD_IMPLID=Implid1  ← OD_IMPLIDにインプリメンテーションリポジトリ設定
      :
java.exe  Server       ← サーバアプリケーションクラスを引数にJava VMを起動
```

(2)POAオブジェクトのアダプタ名への指定

環境変数OD_IMPLIDを設定しないで、アプリケーション上でPOAオブジェクトとインプリメンテーション情報との関連付けを明示的に行うことが可能です。この方式は、RootPOAオブジェクト直下の子POAオブジェクトのアダプタ名に、インプリメンテーションリポジトリIDを使用することで可能となります。

5.14.2 オブジェクトリファレンスの生成方法

オブジェクトリファレンスの生成およびネーミングサービスへの登録には、以下の2種類の方法があります。

- 動的生成
- 事前生成

■動的生成

POAオブジェクト上（サーバアプリケーション内）で、POAオブジェクトのcreate_reference()メソッドにより実行時に生成します。作成したオブジェクトリファレンスは アプリケーション内でネーミングサービスへ登録するか、またはパラメタとしてクライアントアプリケーションに通知することにより、クライアントアプリケーションへ渡されます。
この方法は、クライアントからの要求ごとにオブジェクトリファレンスを作成できるため、クライアントごとのインスタンス管理を動的に行うことができます。

“5.12.2 各アプリケーション形態”に示す「Factory-1方式」、「Factory-2方式」、「ユーザインスタンス管理法式」のServantオブジェクトのオブジェクトリファレンスは、動的生成で作成しています。
なお、オブジェクトリファレンスの動的生成方法については、“5.13.4 オブジェクトリファレンスの生成”を参照してください。

■事前生成

OD_or_admコマンドを使用し、事前にオブジェクトリファレンスの生成とネーミングサービスへの登録を行います。
この方法は、事前にオブジェクトリファレンスを登録しておくため、動的生成に比較して性能／資源面で優れています。
“5.12.2 各アプリケーション形態”に示す「デフォルトインスタンス方式」のServantオブジェクトのオブジェクトリファレンス、および「Factory-1方式」、「Factory-2方式」、「ユーザインスタンス管理法式」のFactoryオブジェクトのオブジェクトリファレンスは、事前生成で作成しています。

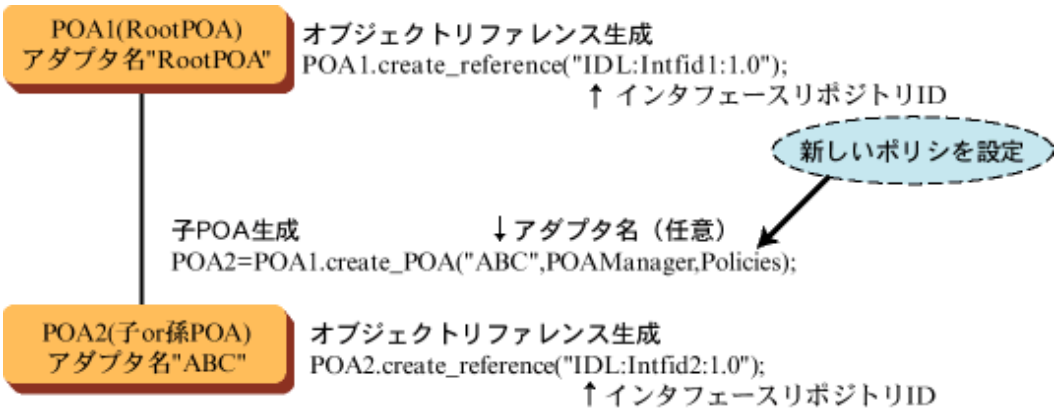
5.14.3 POA使用方法例

POAオブジェクトの構成は、サーバアプリケーションの実装において以下の方法をどのような組合せで使用するかにより異なります。

- インプリメンテーションリポジトリIDとの関連付けの方法（“5.14.1 インプリメンテーション情報との関連付け”を参照）
  - 環境変数OD_IMPLID使用（環境変数OD_IMPLIDへの指定）
  - 環境変数OD_IMPLID未使用（POAオブジェクトのアダプタ名への指定）
- オブジェクトリファレンスの生成方法（“5.14.2 オブジェクトリファレンスの生成方法”を参照）
  - 動的生成
  - 事前生成

それぞれの組合せについて、POAオブジェクトの使用方法と構成を説明します。

■POAオブジェクト使用方法1：動的生成／環境変数OD_IMPLID使用



一般的なPOAオブジェクトの構成です。

1. POAを生成（任意のアダプタ名）します。
2. 指定したインタフェースをもつオブジェクトリファレンスをcreate_referenceメソッドで生成します。
3. Servantを作成します。
4. 生成したオブジェクトリファレンスのオブジェクトIDを取得します。

5. POAのAOMに、Servantのインスタンスとオブジェクトの関連付けを登録します（活性化）。
6. ネーミングサービスへオブジェクトリファレンスを登録(bind)します。

上図では、インタフェースリポジトリID“IDL:Intfid1:1.0”のインタフェースをもつオブジェクトのインスタンスはPOA1オブジェクト(RootPOA)上で管理され、インタフェースリポジトリID“IDL:Intfid2:1.0”のインタフェースをもつオブジェクトのインスタンスはPOA2オブジェクト(子／子孫POAオブジェクト)上で管理されます。

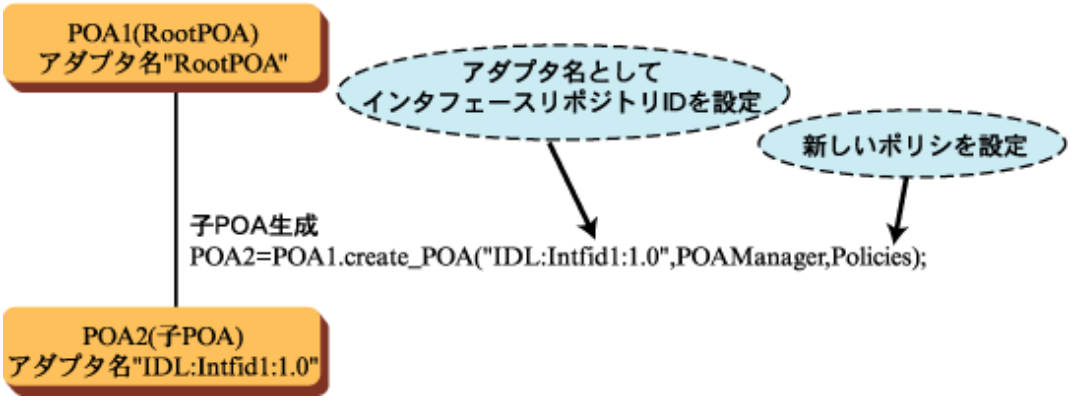
RootPOAオブジェクトで複数のオブジェクトを管理することも可能ですが、上記のような構成とすることにより、インタフェースごとに異なるPOAポリシで管理することが可能となります。

アプリケーションを作成する場合は、create_reference()の代わりに、同じくPOAオブジェクトのメソッドであるservant_to_reference()を使用すると便利です。

1. POAを生成します（任意のアダプタ名）。
2. Servantを作成します。
3. servant_to_reference()を発行し、オブジェクトリファレンスを取得します。
4. ネーミングサービスへオブジェクトリファレンスを登録(bind)します。

servant_to_reference()の結果、POAオブジェクトにServantオブジェクトとオブジェクトの関連付けの登録（活性化）が行われ、さらにオブジェクトIDを内包するオブジェクトリファレンスが返されます。ServantオブジェクトはインタフェースリポジトリIDを内包しているため、create_reference()を使用した場合のように明示的に指定する必要がありません。また、オブジェクトIDは活性化時に自動生成されるため、意識する必要はありません。ただし、POAポリシにIMPLICIT_ACTIVATIONを指定する必要があります。

■POAオブジェクト使用方法2：事前生成／環境変数OD_IMPLID使用



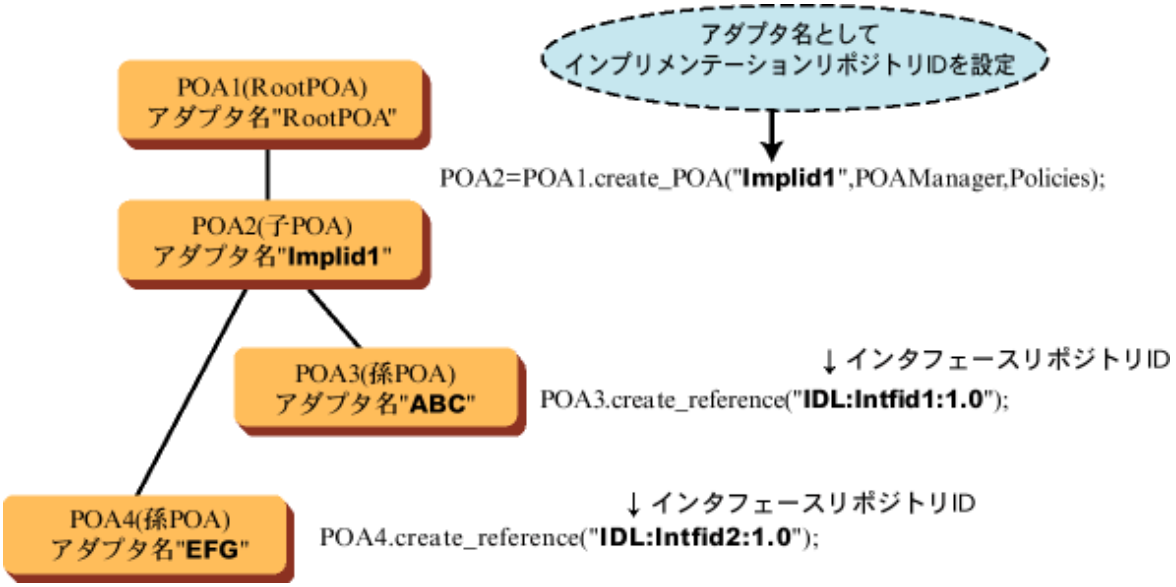
この構成は、OD_or_admコマンドを使用し、事前にオブジェクトリファレンスの生成とネーミングサービスへの登録を行っている場合に使用します。インタフェースリポジトリID“IDL:Intfid1:1.0”のインタフェースをもつオブジェクトを管理するために、「子POAオブジェクト」を生成します。この際に、インタフェースリポジトリIDと同じ文字列（Stringオブジェクト）を設定します。

Servantオブジェクトの活性化は、以下のように行います。

1. Servantオブジェクトを作成します。
2. Servantオブジェクトに対応するインタフェースリポジトリIDをアダプタ名としたPOAに対し、activate_object()を発行します。

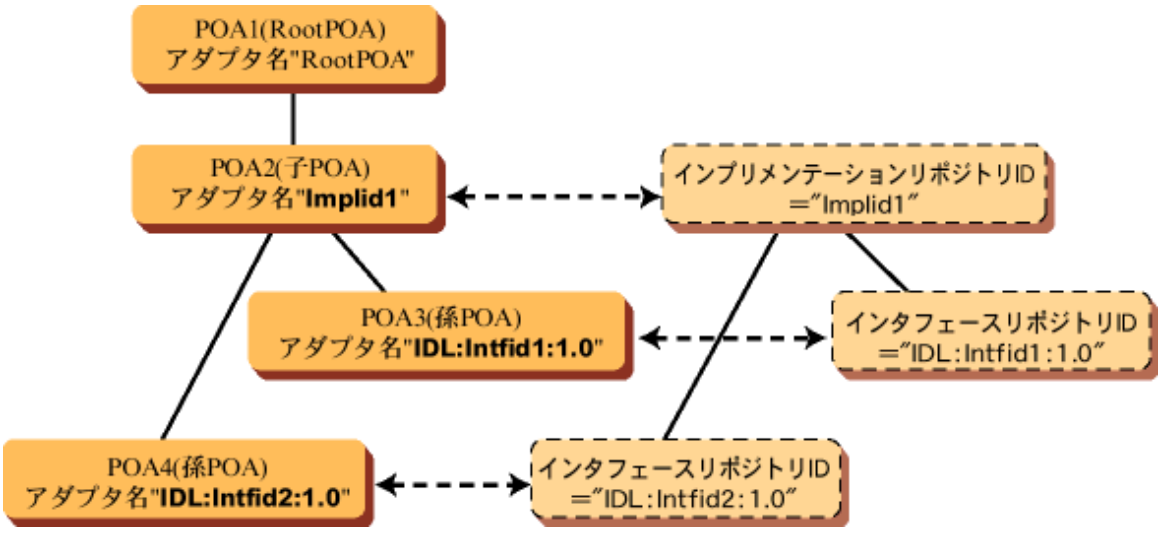
OD_or_admコマンドにより事前にオブジェクトリファレンスの生成とネーミングサービスへの登録を行っているため、アプリケーション上での生成と登録は必要ありません。

■POAオブジェクト使用方法3:動的生成／環境変数OD_IMPLID未使用



インプリメンテーション情報との関連付けを行うために、アダプタ名にインプリメンテーションリポジトリIDを指定し、RootPOAオブジェクトに対する子POAオブジェクトを作成します。
次に、孫（またはその子孫）POAオブジェクトを作成し（アダプタ名は任意）、そのPOAオブジェクト上でcreate_reference()メソッド／servant_to_reference()メソッドを使用してオブジェクトリファレンスを作成します。
Servantオブジェクトの活性化後、オブジェクトのインスタンス管理は、この孫（またはその子孫）POAオブジェクト上で行われます。この構成の場合、RootPOAオブジェクトでは、インスタンス管理を行うことはできません。

■POAオブジェクト使用方法4:事前生成／環境変数OD_IMPLID未使用



インプリメンテーション情報との関連付けを行うためには、上図の左側のように3階層でPOAオブジェクトを生成します。このとき、RootPOAオブジェクトの直下の子POAオブジェクト（POA2）は、インプリメンテーションリポジトリIDをアダプタ名に指定して生成します。
次に、子POAオブジェクトのさらに子POAオブジェクト（孫POA：POA3、POA4）は、インタフェースリポジトリIDをアダプタ名に指定して

生成します。
この結果、インプリメンテーション情報、インタフェース情報は、上図のようにPOAと関連付けられます。リポジトリID“IDL:Intfid2:1.0”のインタフェースをもつオブジェクトのインスタンス管理は、POA4オブジェクトで行われます。

5.15 サーバアプリケーションのプログラミング（静的スケルトンインタフェース）

静的起動インタフェースを使用する場合のサーバアプリケーションの処理の流れを以下に示します。

5.15.1 初期化

CORBAの初期化メソッドorg.omg.CORBA.ORB.init()を呼び出し、初期化処理を行います。本メソッドの結果として、ORBのオブジェクトリファレンスが通知されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
public class Server { // クラスの宣言
    public static void main( String args[] ) {
        org.omg.CORBA.ORB    Orb;      // ORB用オブジェクトリファレンス

        try {
           // ORBの生成と初期化
            Orb = org.omg.CORBA.ORB.init( args, null );
             ...
        }
        catch( java.lang.Exception e ) {
```

```
            ... // 例外処理
        }
    }
}
```

例外処理try-catchについては、“5.16 サーバアプリケーションの例外処理”を参照してください。

5.15.2 RootPOAのオブジェクトリファレンスの獲得

サーバアプリケーションでPOAオブジェクトを使用するため、イニシャルサービスのオブジェクトリファレンスの取り出しメソッドresolve_initial_references()によりRootPOAのオブジェクトリファレンスを求めます。

```
try {
    // RootPOAのオブジェクトリファレンスの取得
    org.omg.CORBA.Object  _tmpObj = Orb.resolve_initial_references( "RootPOA" );
    // RootPOAのPOAオブジェクト獲得
    org.omg.PortableServer.POA  rootPOA  = org.omg.PortableServer.POAHelper.narrow( _tmpObj );
}
catch( java.lang.Exception e ) {
    ... // 例外処理
}
```

5.15.3 子孫POAの作成

RootPOAとは異なるポリシでインスタンスの管理を行う場合や、インタフェースごとなど、複数のPOAにインスタンスの管理を分散させる場合、新しいPOAを生成します。

以下の例では、取得したRootPOAのメソッドを使用し、子POAを作成しています。
create_POA()の第1パラメタには、アダプタ名(生成するPOAの名前)を指定します。アダプタ名は、任意の文字列(Stringオブジェクト)を指定します。“5.14 サーバアプリケーションと環境設定との関連付け”で示したように、インプリメンテーションリポジトリIDやインタフェースリポジトリIDを指定することも可能です。
create_POA()の第3パラメタのポリシには、アプリケーション形態にあったポリシリストを指定します。
この例では、USE_DEFAULT_SERVANTを使用するPOAを作成しています。

```
try {
    // ポリシリスト作成
    org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[4];
    policies[0] = rootPOA.create_servant_retention_policy(
           org.omg.PortableServer.ServantRetentionPolicyValue.NON_RETAIN );
    policies[1] = rootPOA.create_request_processing_policy(
           org.omg.PortableServer.RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
    policies[2] = rootPOA.create_id_assignment_policy(
           org.omg.PortableServer.IdAssignmentPolicyValue.SYSTEM_ID );
    policies[3] = rootPOA.create_id_uniqueness_policy(
           org.omg.PortableServer.IdUniquenessPolicyValue.MULTIPLE_ID );

    // インタフェースリポジトリIDに対応するPOAオブジェクト獲得
    org.omg.PortableServer.POA  POA1 =
            rootPOA.create_POA( "childPOA", RootPOA.the_POAManager(), policies );
}
catch( java.lang.Exception e ) {
          ... // 例外処理
}
```

孫以下のPOAを作成する場合も上記と手順は同じです。子POAのインスタンス(上記の例では、POA1)上でcreate_POA()メソッドを発行することにより、孫POAを生成することが可能です。

注意

POAのアダプタ名（上記例では“childPOA”）には、日本語などのマルチバイト文字を使用できません。

5.15.4 インタフェースの実装

IDLコンパイラで生成されたスケルトン“xxxPOA.java”（xxx：インタフェース名）を継承したServantクラスを作成します。このクラス内でオペレーションの実装を記述します。オペレーションのJavaマッピング後のシグネチャは、xxxOperations.javaファイル（xxx：インタフェース名）内の記述を参照することでわかります。

注意

この実装方法は、継承方式（Inheritance-Based Implementation）に基づくものです。代理方式（Delegation-Based Implementation）を使用する場合は、“5.17 サーバアプリケーションの実装アプローチ”を参照してください。

【IDL定義】

```
module ODsample{
    interface intf{
        long add( in long a, in long b );
    };
};
```

【Servantクラス】

```
class UserServant extends intfPOA{
    public int add( int a, int b ) {
        return( a + b );
    }
}
```

5.15.5 AOMへの登録（活性化）

Servantオブジェクトを生成し、POAのAOMに登録します。以下の例では、RootPOAのAOMにServantオブジェクトを登録しています。servant_to_reference()メソッドを使用することで、Servantオブジェクトを登録すると同時に対応するオブジェクトリファレンスを生成・獲得しています（POAがIMPLICIT_ACTIVATIONポリシを持つ場合）。

【Servantオブジェクトのインスタンス生成とAOMへの登録】

```
try {

    // Servantの生成
    Servant svt = new UserServant();

    // AOMにServantを登録（活性化）、オブジェクトリファレンス取得
    org.omg.CORBA.Object c_obj = rootPOA.servant_to_reference( svt );

}
catch( java.lang.Exception e ) {
    ...  // 例外処理
}
```

5.15.6 ネーミングサービスへの登録

生成したオブジェクトリファレンスをネーミングサービスへ登録します。OD_or_admコマンドでオブジェクトリファレンスの事前生成／登録を行っている場合は、必要ありません。

【ネーミングサービスへの登録】

```
import org.omg.CosNaming.*;
   :
try {
    // ネーミングサービスのオブジェクトリファレンスの取得
    NSObj = Orb.resolve_initial_references( "NameService" );
    // NamingContext型に変換
    NamingContextExt Cos  = NamingContextExtHelper.narrow( NSObj );
    String NCid   = new String( "test" );    // オブジェクト名
    String NCkind = new String( "" );        // オブジェクトタイプ
    NameComponent nc =  new NameComponent( NCid, NCkind );
    NameComponent NCo[] = { nc };
    try {
        Cos.unbind( NCo ); // 同名の古いオブジェクトリファレンスがあれば削除
    }
    catch( java.lang.Exception e ) {}
    Cos.bind( NCo, c_obj ); // オブジェクトリファレンスの登録
}
catch( java.lang.Exception e ) {
    ...  // 例外処理
}
```

5.15.7 オブジェクトリファレンスの事前生成／登録

“5.15.6 ネーミングサービスへの登録”の例では、アプリケーション内でオブジェクトリファレンスの生成とネーミングサービスへの登録を行っていますが、OD_or_admコマンドを使用することで、事前にオブジェクトリファレンスの生成／登録を行うことが可能となります。この場合、アプリケーション内での生成／登録を行う必要がありません。POA構成は、「POA構成方法2」または「POA構成方法4」を使用することになります。

例

OD_or_admコマンド使用例

```
OD_or_adm -c IDL:ODsample/intf:1.0  -a Imple_sample1  -n ODsample::sample1
```

-c IDL:ODsample/intf:1.0

指定されたインタフェースリポジトリIDでオブジェクトリファレンスを登録します。

-a Imple_sample1

指定されたインプリメンテーションリポジトリIDでオブジェクトリファレンスを登録します。

-n ODsample::sample1

ネーミングサービスに登録するオブジェクト名を指定します。

5.15.8 POAManagerの活性化と終了待機

POAに関連付けられたPOAManagerのactivate()を発行することで、クライアントからの受信待ち状態となります。activate()発行直後に、ORBクラスのrun()メソッドを発行します。これにより、サーバアプリケーションは終了せず待機状態となります。

【POAManagerの活性化】

```
try {
    // POAマネージャのactivate
    rootPOA.the_POAManager().activate();
    Orb.run();
}
catch( java.lang.Exception e ) {
    ... // 例外処理
}
```

注意

動作上、activate()発行時に終了待機状態となるため、run()メソッドには制御は移りません。このため、run()メソッドは機能的には何も行いません。しかし、アプリケーションの可搬性の面からrun()メソッドを記述することを推奨します。

注意

Windows32/64

以下の条件のとき、POAManagerのactivate()が復帰せずにサーバアプリケーションが終了します。

- サーバアプリケーションをプロセスモードで作成している場合 かつ
- サーバアプリケーションをCORBAワークユニット運用している場合 かつ
- CORBAワークユニットを停止した場合

POAManagerのactivate()の後に後処理を行う場合は、サーバアプリケーションをスレッドモードで作成するか、またはワークユニットの出口機能を利用してください。ワークユニットの出口機能に関しては、“OLTPサーバ運用ガイド”の“ワークユニットの出口機能”を参照してください。

5.16 サーバアプリケーションの例外処理

(1)例外情報の設定

IDL言語でユーザ例外が定義されていた場合について、サーバアプリケーションプログラムの例を以下に示します。

IDLマッピング

【IDL言語】

```
module ODsample {
    interface  exptest{
        exception testException { string reason; };
        void    op1() raises( testException );
    };
};
```

これをJava言語で記述すると、以下のようになります。

【Java言語】

＜インタフェース＞

```
package ODsample;
public interface exptestOperations{
    public void op1()
        throws ODsample.exptestPackage.testException;
}
```

＜ユーザ例外クラス＞

```
package ODsample.exptestPackage;
public final class testException
    extends org.omg.CORBA.UserException
{
    public java.lang.String reason;
    public testException(){}
    public testException( java.lang.String _reason ){
        reason = _reason;
    }
}
```

サーバアプリケーションでの処理

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

public class expServer {
    public static void main( String args[] ) {
        try {
            // ORBの前処理
                :
            // POAオブジェクトの生成
                :
            // Servantの生成とそのPOAへの登録
                :
            // POAManagerの活性化
                :
        }
        catch ( java.lang.Exception e ) {
            // エラー処理
                :
        }
    }
}

// Servantクラス
// サーバアプリケーションのメソッド
class expServant extends exptestPOA{
    public void op1() throws ODsample.exptestPackage.testException  {
        throw new ODsample.exptestPackage.testException( "Test of UserException" );
    }
}
```

(2)例外情報の獲得

サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“5.10 クライアントアプリケーションの例外処理”を参照してください。

5.17 サーバアプリケーションの実装アプローチ

静的スケルトンインタフェースを使用したサーバアプリケーションの実装上のアプローチとして、以下の2つの方式が存在します。ここでは、これら2つの実装方式とその特徴について説明します。

- 継承方式
- 代理方式

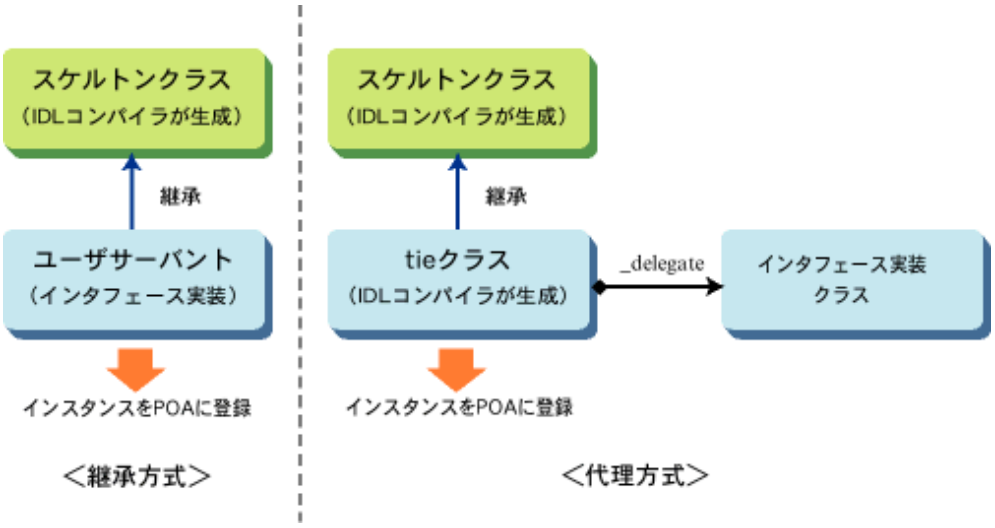
5.17.1 継承方式と代理方式

■継承方式（Inheritance-Based Implementation）

新規にサーバアプリケーションを作成する場合に使用する、一般的な実装方法です。Servant自身がスケルトンを継承する形式で実装します。この結果、オペレーションの起動に必要な機能がServantに継承されます。POAに対しては、Servant自身を登録します。

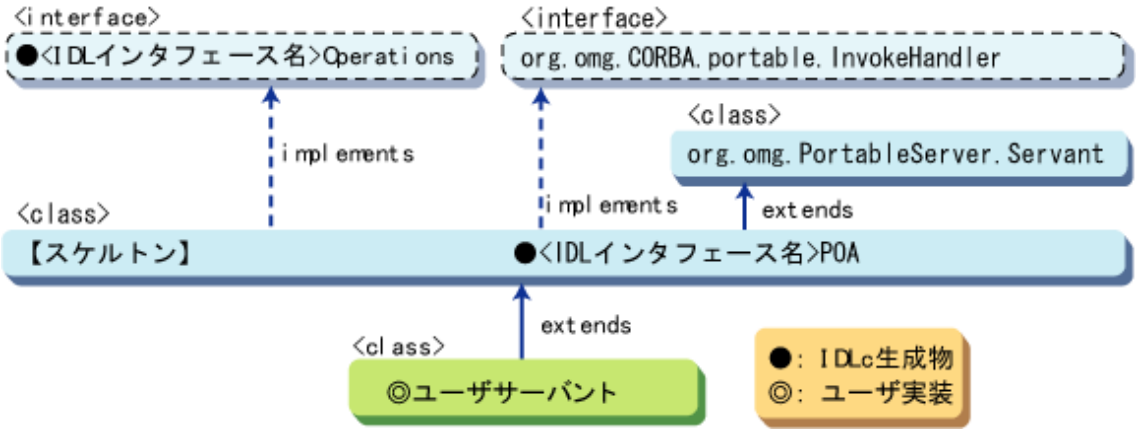
■代理方式（Delegation-Based Implementation）

ユーザが開発した既存のクラスなどをServantとして使用したい場合に有効な方法です。スケルトンを継承し、かつPOAの登録対象となるServantとして**tieクラス**が用意されます。IDLコンパイラにより<interface名>POATie.javaとして生成されます。
ユーザが作成したインタフェース実装クラスのインスタンスは、tieクラス内のメンバオブジェクトとして登録します。オペレーションの起動は、いったんtieクラスを経由して登録したクラスのインスタンスに対して発行されます。



5.17.2 継承方式のServant実装

継承方式によるServantは、以下のような継承関係となります。サーバアプリケーションのメイン部では、このServantをPOAに登録します。



上図のとおり、継承方式ではServantは、以下のようにスケルトンクラスを継承する形式で実装します。

【IDL定義】

```
module ODsample{
    interface   intf{
        long    add(in long a, in long b);
    };
};
```

【Servantの例】

```
class UserServant extends ODsample.intfPOA{
    public int add( int a, int b ) {
        return( a + b );
    }
}
```

メイン処理部では、このServantをPOAに登録します。以下の例では、Default Servantとして登録しています。

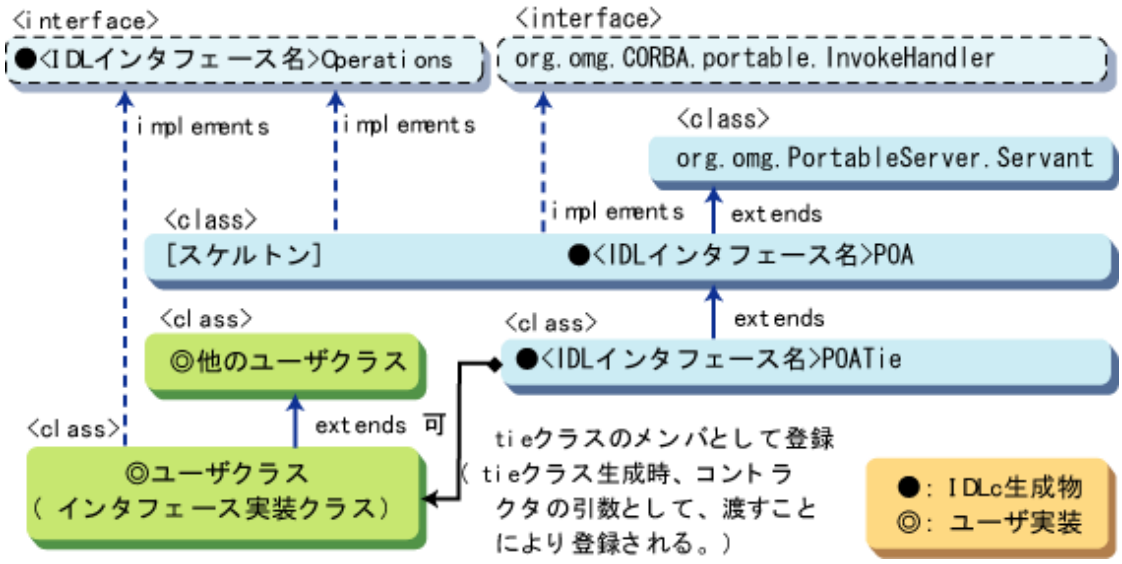
【メイン部の例】

```
// Servantの生成
Servant svt = new UserServant();
// Default Servantに設定
aPOA.set_servant( svt );
```

注）aPOA：POAクラスのインスタンス

5.17.3 代理方式のServant実装

代理方式によるユーザクラス（インタフェース実装クラス）は、以下のような継承関係で実装します。サーバアプリケーションのメイン部では、tieクラス（<インタフェース名>POATie）のインスタンスをPOAに登録します。継承方式を使用した場合のような規定の継承関係から解放されるため、実装の自由度が高くなります。ただし、tieクラスのメンバとして登録するために、インタフェース実装クラスは、<インタフェース名>Operationインタフェースをimplementsする必要があります。
代理方式のアプリケーション例は、“5.19 サーバアプリケーションのプログラミング例”を参照してください。



上図のとおり、代理方式ではユーザクラスは、以下のように<interface名>Operationインタフェースを実装するように記述します。

【IDL定義】

```
module ODsample{
    interface   intf{
        long    add(in long a, in long b);
    };
};
```

【ユーザクラスの例】

```
class UserClass extends OtherUserClass
implements ODsample.intfOperations
{
    public int add( int a, int b ){
        return( a + b );
    }
}
```

メイン処理部では、いったんこのユーザクラスのインスタンスをtieクラスのインスタンスに登録します。tieクラスのコンストラクタの引数としてユーザクラスインスタンスを渡します。
さらに、tieクラスのインスタンスをPOAにServantとして登録します。
以下の例では、Default Servantとして登録しています。

【メイン部の例】

```
// ユーザクラスのインスタンス生成
UserClass uc = new UserClass();
// tieオブジェクト作成とユーザクラスの登録
intfPOATie  tie = new intfPOATie( uc );
// POAのDefault Servantに設定
aPOA.set_servant( tie );
```

注)aPOA:POAクラスのインスタンス

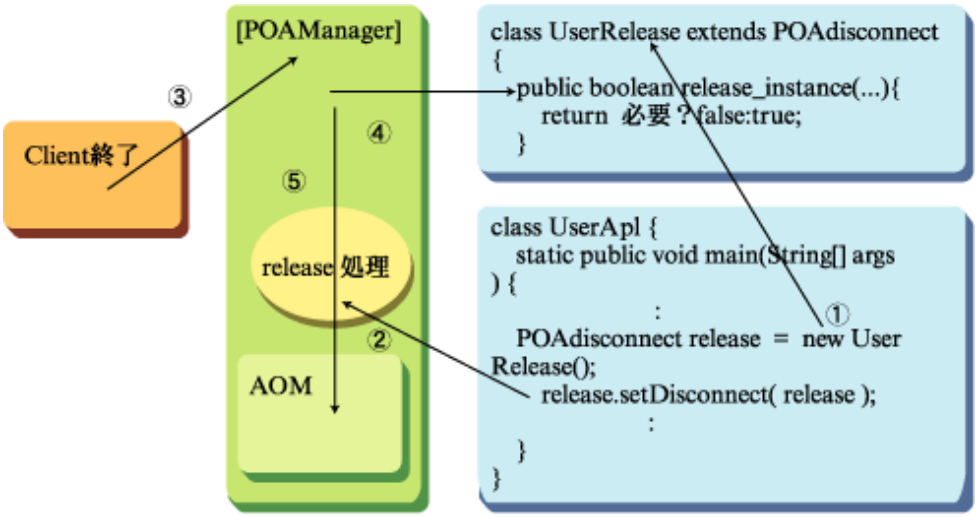
5.17.4 継承方式と代理方式の比較

継承方式、代理方式の特徴を以下の表に示します。
サーバアプリケーションの設計で、どちらの実装方式を使用するかはユーザの選択に任されます。

	継承方式	代理方式
主な用途	新規開発	既存資産の利用
実装の自由度	小さい。 Servantはスケルトンを継承し、他のクラスを継承することができないため。	大きい。 インタフェース実装クラスは、スケルトンを継承する必要がないため、他のクラスを継承できます。
コーディングの複雑さ	比較的簡潔となります。	複雑になりやすい。
性能	オペレーションの起動に必要なスケルトン機能は、Servant自身に継承されているため、比較的よい。	オペレーションの起動は、tieクラスを経由するため、オブジェクトの参照が増加します。これにより、多少のオーバヘッドがあります。

5.18 クライアントとのコネクション切断時のインスタンス解放

クライアントとのコネクションが切断された場合に、サーバアプリケーション側の不要なインスタンスを解放する方法について説明します。
この機能は、インスタンス管理機能を利用し、かつPOAがActiveObjectMapを使用してインスタンス管理を行っている場合にだけ利用できます。



1. Servantインスタンス解放用クラスのインスタンスを生成します。
2. 生成したインスタンスを登録します。
3. クライアントとのコネクション切断が通知されます。
4. サーバントインスタンス解放の是非を問い合わせます。
5. サーバントインスタンスが不要である場合、AOMから削除します。

5.18.1 インスタンス解放処理を行うクラスの作成

クライアントとのコネクション切断時に、インスタンス解放を行うクラスを作成します。このクラスは、com.fujitsu.ObjectDirector.PortableServer.POAdisconnectクラスを継承して作成します。

5.18.2 インスタンス解放処理の実装

クライアントとのコネクションが切断されたときに、作成したクラスのrelease_instance()メソッドが呼び出されます。親クラス（POAdisconnectクラス）のrelease_instance()メソッドは、常にfalse（解放しない）を返します。
release_instance()メソッドが呼び出されたとき、コネクションが切断されたクライアントが使用した、POA、Servantのインスタンスおよびオブジェクトidがパラメタとして通知されます。オーバーライドされたrelease_instance()メソッドでは、これらの情報を元にして、通知されたServantが必要なものであるかどうかを判断し、true（解放する）／false（解放しない）を返します。

5.18.3 インスタンス解放処理を行うクラスインスタンスの登録

POAManagerにユーザが作成したインスタンス解放クラスを登録します。登録には、POAdisconnectクラスのstaticメソッドsetDisconnect()を呼び出します。

5.18.4 インスタンス解放処理を行うクラスインスタンスの変更・削除

すでに登録したインスタンス解放クラスを変更／取消しを行う場合は、POAdisconnectクラスのstaticメソッドresetDisconnect()を呼び出します。

5.19 サーバアプリケーションのプログラミング例

5.19.1 Default Servant使用例（デフォルトインスタンス方式）

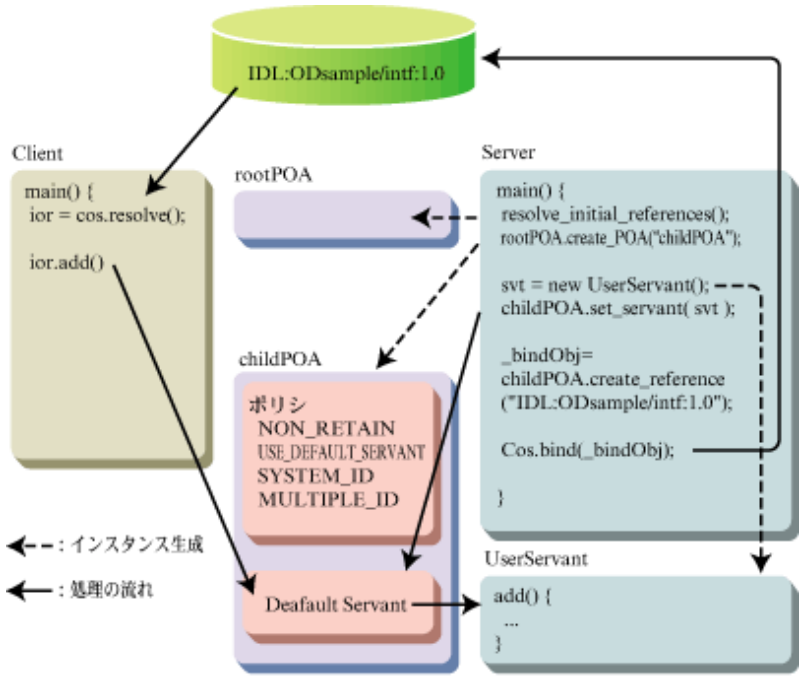
(1)IDL定義

```
module ODsample{
    interface  intf{
        long  add( in  long  a,  in  long  b );
    };
};
```

(2)ネーミングサービスの登録

動的登録

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client {
    public static void main( String args[] ) {

        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // ネーミングサービスのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // ネーミングサービスのresolveメソッドを発行し、
            // サーバアプリケーションのオブジェクトリファレンスの獲得
            String NCid = new String( "ODsample::POAsample1" ); // オブジェクト名
            String NCkind = new String( "" );                   // オブジェクトタイプ
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

            // サーバアプリケーションのオブジェクトリファレンス獲得
            intf target = intfHelper.narrow( Obj );

            int  in1 = 0;     // inパラメタ用変数
            int  in2 = 0;     // inパラメタ用変数
            int  result;      // 復帰値用変数
            String line = null;

            try {
                System.out.print( "in1 => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
```

```
                in1 = Integer.parseInt( line );

                System.out.print( "in2 => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in2 = Integer.parseInt( line );
            }
            catch ( java.lang.NumberFormatException e ) {
                System.exit( 255 );
            }

            // サーバアプリケーションのメソッド呼出し
            result = target.add( in1, in2 );

            // メソッドの結果表示
            System.out.println( in1 + " + " + in2 + " = "+ result );
        }
        catch( org.omg.CORBA.SystemException se ) {
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
             System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}
```

(5)サーバアプリケーション

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.CosNaming.*;
import ODsample.*;

// ユーザアプリケーション：メイン処理クラス
public class Server {
    public static void main( String args[] ) {

        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // RootPOAのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "RootPOA" );
            // RootPOAのPOAオブジェクト獲得
            POA rootPOA = POAHelper.narrow( _tmpObj );

            // インタフェース用のPOA作成
            // ポリシリスト作成
            org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[4];
            policies[0] = rootPOA.create_servant_retention_policy(
                                ServantRetentionPolicyValue.NON_RETAIN );
            policies[1] = rootPOA.create_request_processing_policy(
                                RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
            policies[2] = rootPOA.create_id_assignment_policy(
                                IdAssignmentPolicyValue.SYSTEM_ID );
            policies[3] = rootPOA.create_id_uniqueness_policy(
```

```
                                IdUniquenessPolicyValue.MULTIPLE_ID );

            // Default Servant用のPOAオブジェクトの作成
            POA childPOA = rootPOA.create_POA( "childPOA", null, policies );

            // Servantの生成
            Servant svt = new UserServant();
            // Default Servantに設定
            childPOA.set_servant( svt );

            org.omg.CORBA.Object _bindObj = childPOA.create_reference( "IDL:ODsample/intf:1.0" );

            // ネーミングサービスのオブジェクトリファレンスの取得
            _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // ネーミングサービスのresolveメソッドを発行し、サーバアプリケーション
            // のオブジェクトリファレンスの獲得
            String NCid = new String( "ODsample::POAsample1" ); // オブジェクト名
            String NCkind = new String( "" );                   // オブジェクトタイプ
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            try {
                Cos.unbind( NCo );
            } catch( Exception e ) {
                ;
            }
            Cos.bind( NCo, _bindObj );

            // POAマネージャの獲得
            POAManager poamanager = childPOA.the_POAManager();

            // POAマネージャのactivate
            poamanager.activate();
            Orb.run();
        }
        catch( org.omg.CORBA.SystemException se ) {
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
             System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}

// Servant：メソッド実装クラス（スケルトンを継承）
class UserServant extends intfPOA {
    public int add( int a, int b ) {
        return( a + b );
    }
}
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“5.10 クライアントアプリケーションの例外処理”を参照してください。

5.19.2 Active Object Map（AOM）使用例（Factory-1方式）

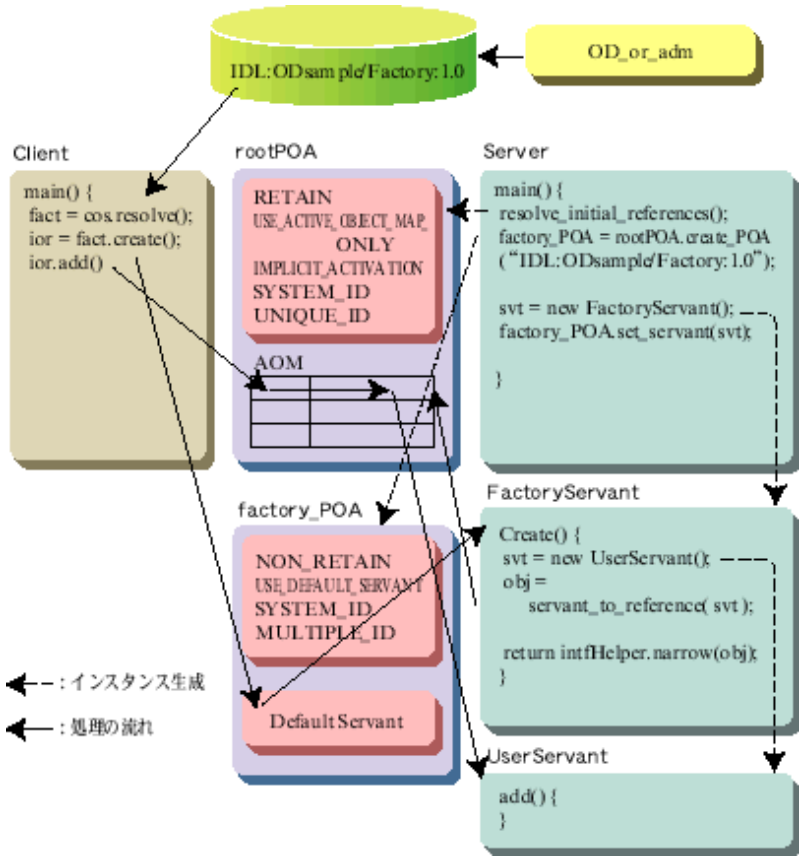
(1)IDL定義

```
module ODsample{
    interface intf{
        readonly  attribute  long  value;
        void  add( in  long  a );
        void  sub( in  long  b );
    };
    interface Factory {
        intf   create();
        void   destroy( in  intf  obj );
    };
};
```

(2)ネーミングサービスの登録

```
OD_or_adm -c IDL:ODsample/Factory:1.0 -n ODsample::POAsample5
```

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client {
    public static void main( String args[] ) {

        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // ネーミングサービスのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // ネーミングサービスのresolveメソッドを発行し、
            // サーバアプリケーションのオブジェクトリファレンスの獲得
            String NCid = new String( "ODsample::POAsample5" ); // オブジェクト名
            String NCkind = new String( "" );                   // オブジェクトタイプ
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

            // Factoryのオブジェクトリファレンス獲得
            Factory target = FactoryHelper.narrow( Obj );

            // Factoryでインタフェース用オブジェクトリファレンス作成
            intf _intf = target.create();

            int    in   = 0;    // inパラメタ用変数
            String line = null;

            try {
                // サーバアプリケーションのメソッド呼出し
                System.out.println( "value = " + _intf.value() );

                System.out.print( "add => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // サーバアプリケーションのメソッド呼出し
                _intf.add( in );
                System.out.println( "value = " + _intf.value() );

                System.out.print( "sub => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // サーバアプリケーションのメソッド呼出し
                _intf.sub( in );
                System.out.println( "value = " + _intf.value() );

                // インスタンスの解放
                target.destroy( _intf );
            }
            catch ( java.lang.NumberFormatException e ){
                System.exit(255);
            }
        }
        catch( org.omg.CORBA.SystemException se ) {
```

```
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
             System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}
```

(5)サーバアプリケーション

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// ユーザアプリケーション：メイン処理クラス
public class Server {
    public static void main( String args[] ) {

        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // RootPOAのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "RootPOA" );
            // RootPOAのPOAオブジェクト獲得
            POA rootPOA  = POAHelper.narrow( _tmpObj );

            // Factoryインタフェース用のPOA作成
            // ポリシリスト作成
            org.omg.CORBA.Policy factory_policies[] = new org.omg.CORBA.Policy[4];
            factory_policies[0] = rootPOA.create_servant_retention_policy(
                                  ServantRetentionPolicyValue.NON_RETAIN );
            factory_policies[1] = rootPOA.create_request_processing_policy(
                                  RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
            factory_policies[2] = rootPOA.create_id_assignment_policy(
                                  IdAssignmentPolicyValue.SYSTEM_ID );
            factory_policies[3] = rootPOA.create_id_uniqueness_policy(
                                  IdUniquenessPolicyValue.MULTIPLE_ID );
            POA factory_POA = rootPOA.create_POA( "IDL:ODsample/Factory:1.0",
                                                  null,
                                                  factory_policies );

            // FactoryServantの生成
            Servant svt = new FactoryServant( rootPOA );
            // Factoryインタフェース用POAのDefault Servantに設定
            factory_POA.set_servant( svt );

            // POAマネージャの獲得
            POAManager poamanager = rootPOA.the_POAManager();

            // POAマネージャのactivate
            poamanager.activate();
            Orb.run();
        }
        catch( org.omg.CORBA.SystemException se ) {
```

```
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
            System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}

// FactoryServant：Factoryメソッド実装クラス（スケルトンを継承）
class FactoryServant extends FactoryPOA {

    private POA poa = null;

    // コンストラクタ
    public FactoryServant( POA poa ) {
        this.poa = poa;
    }

    public intf create() {
        intf       ior;           // UserServantのオブジェクトリファレンス

        try {
            // Servantの生成
            Servant svt = new UserServant();

            // Servantからオブジェクトリファレンスの生成
            // IMPLICIT_ACTIVATIONポリシなので、AOMへ自動登録される
            org.omg.CORBA.Object _tmpObj = this.poa.servant_to_reference( svt );
            ior = intfHelper.narrow( _tmpObj );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "create error: " + e );
            return( null );
        }

        return( ior );
    }

    public void destroy( intf obj ) {
        try {
            // オブジェクトリファレンスからオブジェクトIDを求める
            byte oid[] = this.poa.reference_to_id( obj );
            // Servantをdeactiveにする
            this.poa.deactivate_object( oid );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "destroy error: " + e );
        }
    }
}

// Servant：メソッド実装クラス（スケルトンを継承）
class UserServant extends intfPOA {

    private int    value = 0;
```

```
    public int value() {
        return( this.value );
    }

    public void add( int a ) {
        this.value += a;
    }

    public void sub( int b ) {
        this.value -= b;
    }
}
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“5.10 クライアントアプリケーションの例外処理”を参照してください。

5.19.3 Servant Activator使用例（Factory-2方式）

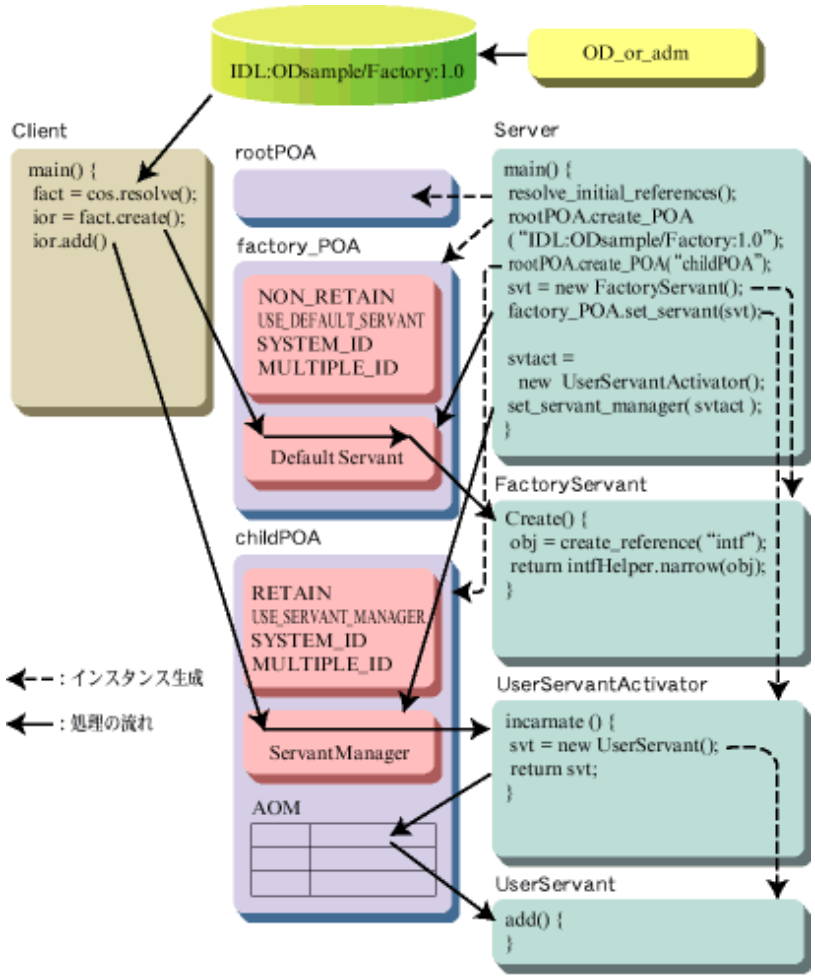
(1)IDL定義

```
module ODsample{
    interface intf{
        readonly  attribute  long  value;
        void  add( in  long  a );
        void  sub( in  long  b );
    };
    interface Factory {
        intf   create();
        void   destroy( in  intf  obj );
    };
};
```

(2)ネーミングサービスの登録

```
OD_or_adm -c IDL:ODsample/Factory:1.0 -n ODsample::POAsample3
```

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client {
    public static void main(String args[]) {

        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // ネーミングサービスのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // ネーミングサービスのresolveメソッドを発行し、
            // サーバアプリケーションのオブジェクトリファレンスの獲得
            String NCid = new String( "ODsample::POAsample3" ); // オブジェクト名
            String NCkind = new String( "" );                   // オブジェクトタイプ
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

            // Factoryのオブジェクトリファレンス獲得
```

```
            Factory target = FactoryHelper.narrow( Obj );

            // Factoryでインタフェース用オブジェクトリファレンスを作成する
            intf _intf = target.create();

            int    in   = 0;    // inパラメタ用変数
            String line = null;

            try {
                // サーバアプリケーションのメソッド呼出し
                System.out.println( "value = " + _intf.value() );

                System.out.print( "add => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // サーバアプリケーションのメソッド呼出し
                _intf.add( in );
                System.out.println( "value = " + _intf.value() );

                System.out.print( "sub => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // サーバアプリケーションのメソッド呼出し
                _intf.sub( in );
                System.out.println( "value = " + _intf.value() );

                // インスタンスの解放
                target.destroy( _intf );
            }
            catch ( java.lang.NumberFormatException e ) {
                System.exit( 255 );
            }
        }
        catch( org.omg.CORBA.SystemException se ) {
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
             System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}
```

(5)サーバアプリケーション

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// ユーザアプリケーション：メイン処理クラス
public class Server {
    public static void main( String args[] ) {

        try {
```

```
        // ORBの生成と初期化
        ORB Orb = ORB.init( args, null );

        // RootPOAのオブジェクトリファレンスの取得
        org.omg.CORBA.Object _tmpObj =
                    Orb.resolve_initial_references( "RootPOA" );
        // RootPOAのPOAオブジェクト獲得
        POA rootPOA = POAHelper.narrow( _tmpObj );

        // Factoryインタフェース用のPOA作成
        // ポリシリスト作成
        org.omg.CORBA.Policy factory_policies[] = new org.omg.CORBA.Policy[4];
        factory_policies[0] = rootPOA.create_servant_retention_policy(
                              ServantRetentionPolicyValue.NON_RETAIN );
        factory_policies[1] = rootPOA.create_request_processing_policy(
                              RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
        factory_policies[2] = rootPOA.create_id_assignment_policy(
                              IdAssignmentPolicyValue.SYSTEM_ID );
        factory_policies[3] = rootPOA.create_id_uniqueness_policy(
                              IdUniquenessPolicyValue.MULTIPLE_ID );
        POA factory_POA = rootPOA.create_POA( "IDL:ODsample/Factory:1.0",
                                              null,
                                              factory_policies );

        // インタフェース用のPOA作成
        // ポリシリスト作成
        org.omg.CORBA.Policy inf_policies[] = new org.omg.CORBA.Policy[4];
        inf_policies[0] = rootPOA.create_servant_retention_policy(
                          ServantRetentionPolicyValue.RETAIN );
        inf_policies[1] = rootPOA.create_request_processing_policy(
                          RequestProcessingPolicyValue.USE_SERVANT_MANAGER );
        inf_policies[2] = rootPOA.create_id_assignment_policy(
                          IdAssignmentPolicyValue.SYSTEM_ID );
        inf_policies[3] = rootPOA.create_id_uniqueness_policy(
                          IdUniquenessPolicyValue.MULTIPLE_ID );
        POA childPOA = rootPOA.create_POA( "childPOA", null, inf_policies );

        // FactoryServantの生成
        Servant svt = new FactoryServant( childPOA );
        // Factoryインタフェース用POAのDefault Servantに設定
        factory_POA.set_servant( svt );

        // ServantActivatorの生成
        ServantActivator svtact = new UserServantActivator();
        // インタフェース用POAのServantManagerとして登録
        childPOA.set_servant_manager( svtact );

        // POAマネージャの獲得
        POAManager poamanager = rootPOA.the_POAManager();

        // POAマネージャのactivate
        poamanager.activate();
        Orb.run();
    }
    catch( org.omg.CORBA.SystemException se ) {
        System.out.println( "ERROR : " + se.getClass().getName()
             + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
         System.exit( 255 );
    }
    catch( org.omg.CORBA.UserException ue ) {
        System.out.println( "ERROR : " + ue.getClass().getName() );
        System.exit( 255 );
    }
```

```
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}

// FactoryServant：Factoryメソッド実装クラス（スケルトンを継承）
class FactoryServant extends FactoryPOA {
    private POA poa = null;

    // コンストラクタ
    public FactoryServant( POA poa ) {
        this.poa = poa;
    }

    public intf create() {
        intf       ior;        // UserServantのオブジェクトリファレンス

        try {
            // オブジェクトリファレンスの生成
            org.omg.CORBA.Object _tmpObj = this.poa.create_reference( "IDL:ODsample/intf:1.0" );
            ior = intfHelper.narrow( _tmpObj );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "create error: " + e );
            return( null );
        }

        return( ior );

    }

    public void destroy( intf obj ) {
        try {
            // オブジェクトリファレンスからオブジェクトIDを求める
            byte oid[] = this.poa.reference_to_id( obj );

            // Servantをdeactiveにする
            this.poa.deactivate_object( oid );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "destroy error: " + e );
        }
    }
}

// Servantアクティベータ：Servant生成クラス（ServantActivatorを継承）
class UserServantActivator extends LocalObject implements ServantActivator {

    public Servant incarnate( byte[] oid, POA adapter ) {

        // Servantの生成
        Servant svt = new UserServant();
        return( svt );
    }

    public void etherealize( byte[] oid,
                             POA adapter,
                             Servant serv,
                             boolean cleanup_in_progress,
                             boolean remaining_activations ) {
```

```
        // 変数の初期化を行う。
        serv = null;

    }
}

// Servant : メソッド実装クラス（スケルトンを継承）
class UserServant extends intfPOA {

    private int    value = 0;

    public int value() {
        return( this.value );
    }

    public void add( int a ) {
        this.value += a;
    }

    public void sub( int b ) {
        this.value -= b;
    }

}
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“5.10 クライアントアプリケーションの例外処理”を参照してください。

5.19.4 Servant Locator使用例（ユーザインスタンス管理方式）

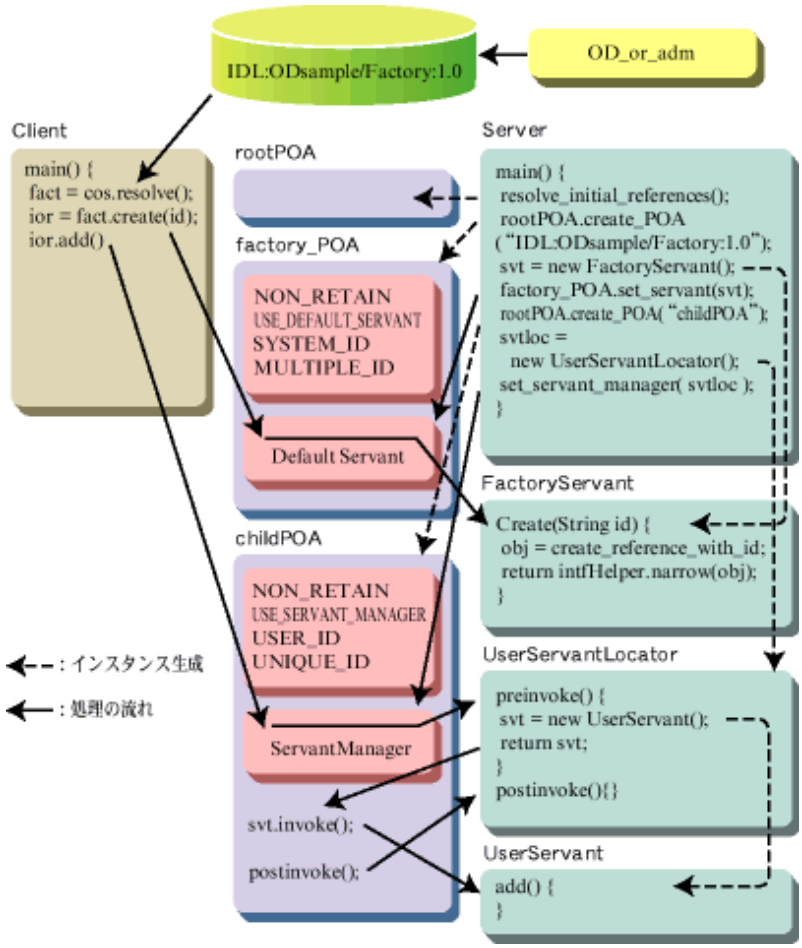
(1)IDL定義

```
module ODsample{
    interface    intf{
        readonly  attribute  long  value;
        void add( in  long  a );
        void sub( in  long  b );
    };
    interface Factory{
        intf   create( in  string  userid );
        void   destroy( in  intf  obj );
    };
};
```

(2)ネーミングサービスの登録

```
OD_or_adm -c IDL:ODsample/Factory:1.0 -n ODsample::POAsample4
```

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client {
    public static void main( String args[] ) {

        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // ネーミングサービスのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // ネーミングサービスのresolveメソッドを発行し、
            // サーバアプリケーションのオブジェクトリファレンスの獲得
            String NCid = new String( "ODsample::POAsample4" ); // オブジェクト名
            String NCkind = new String( "" );                   // オブジェクトタイプ
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

            // サーバアプリケーションのインタフェースクラスの獲得
            Factory target = FactoryHelper.narrow( Obj );
```

```
            int  in;
            String line = null;
            String userid = null;

            try {
                System.out.print( "USERID => " );
                userid = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                if( userid.length() == 0 )
                    System.exit( 255 );

                // Factoryでインタフェースアプリの IORを作成する
                intf _intf = target.create( userid );

                // サーバアプリケーションのメソッド呼出し
                System.out.println( "value = " + _intf.value() );

                System.out.print( "add => " );
                line =  new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // サーバアプリケーションのメソッド呼出し
                _intf.add( in );
                System.out.println( "value = " + _intf.value() );

                System.out.print( "sub => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // サーバアプリケーションのメソッド呼出し
                _intf.sub( in );
                System.out.println( "value = " + _intf.value() );

                target.destroy( _intf );
            }
            catch ( java.lang.NumberFormatException e ) {
                System.exit(255);
            }
        }
        catch( org.omg.CORBA.SystemException se ) {
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
             System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}
```

(5)サーバアプリケーション

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import java.util.*;
import ODsample.*;
```

```
// ユーザアプリケーション：メイン処理クラス
public class Server {
    public static void main( String args[] ) {

        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // RootPOAのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "RootPOA" );
            // RootPOAのPOAオブジェクト獲得
            POA rootPOA = POAHelper.narrow( _tmpObj );

            // Factoryインタフェース用のPOA作成
            // ポリシリスト作成
            org.omg.CORBA.Policy factory_policies[] = new org.omg.CORBA.Policy[4];
            factory_policies[0] = rootPOA.create_servant_retention_policy(
                                  ServantRetentionPolicyValue.NON_RETAIN );
            factory_policies[1] = rootPOA.create_request_processing_policy(
                                  RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
            factory_policies[2] = rootPOA.create_id_assignment_policy(
                                  IdAssignmentPolicyValue.SYSTEM_ID );
            factory_policies[3] = rootPOA.create_id_uniqueness_policy(
                                  IdUniquenessPolicyValue.MULTIPLE_ID );
            POA factory_POA = rootPOA.create_POA( "IDL:ODsample/Factory:1.0",
                                                  null,
                                                  factory_policies );

            // インタフェース用のPOA作成
            // ポリシリスト作成
            org.omg.CORBA.Policy inf_policies[] = new org.omg.CORBA.Policy[4];
            inf_policies[0] = rootPOA.create_servant_retention_policy(
                              ServantRetentionPolicyValue.NON_RETAIN );
            inf_policies[1] = rootPOA.create_request_processing_policy(
                              RequestProcessingPolicyValue.USE_SERVANT_MANAGER );
            inf_policies[2] = rootPOA.create_id_assignment_policy(
                              IdAssignmentPolicyValue.USER_ID );
            inf_policies[3] = rootPOA.create_id_uniqueness_policy(
                              IdUniquenessPolicyValue.UNIQUE_ID );
            String inf_adapter_name = "IDL:ODsample/intf:1.0"; //インタフェースリポジトリID
            POA childPOA = rootPOA.create_POA( "childPOA", null, inf_policies );

            // FactoryServantの生成
            Servant svt = new FactoryServant( childPOA );
            // Factoryインタフェース用POAのDefault Servantに設定
            factory_POA.set_servant( svt );

            // ServantLocatorの生成
            ServantLocator svtloc = new UserServantLocator();
            // インタフェース用POAにServantManagerとして登録
            childPOA.set_servant_manager( svtloc );

            // POAマネージャの獲得
            POAManager poamanager = rootPOA.the_POAManager();

            // POAマネージャのactivate
            poamanager.activate();
            Orb.run();
        }
        catch( org.omg.CORBA.SystemException se ) {
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
             System.exit( 255 );
```

```
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}

// FactoryServant：Factoryメソッド実装クラス（スケルトンを継承）
class FactoryServant extends FactoryPOA {

    static java.util.Hashtable tbl = new java.util.Hashtable();  // 管理テーブル
    private POA poa = null;

    // コンストラクタ
    public FactoryServant( POA poa ) {
        this.poa = poa;
    }

    public intf create( java.lang.String userid ) {
        intf       ior;   // UserServantのオブジェクトリファレンス

        // オブジェクトリファレンスの生成
        org.omg.CORBA.Object _tmpObj = this.poa.create_reference_with_id(
              userid.getBytes(),
              "IDL:ODsample/intf:1.0" );
        ior = intfHelper.narrow( _tmpObj );

        return( ior );

    }

    public void destroy( intf obj )
    {
        try{
            // オブジェクトリファレンスからオブジェクトIDを求める
            byte oid[] = this.poa.reference_to_id( obj );

            // Servantを管理テーブルから削除
            FactoryServant.tbl.remove(  new ObjKey( oid )  );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "destroy error: " + e );
        }
    }
}

// Servantロケータ：Servant生成クラス（ServantLocatorを継承）
class UserServantLocator extends LocalObject implements ServantLocator {

    public Servant preinvoke(
               byte[] oid,
               org.omg.PortableServer.POA adapter,
               java.lang.String operation,
               org.omg.PortableServer.ServantLocatorPackage.CookieHolder cookie ) {

        // Servantを検索する。みつからなければnewで生成する
        Servant svt =  (Servant)FactoryServant.tbl.get( new ObjKey( oid ) );
        if ( svt == null ) {
```

```
            svt = new UserServant();
            FactoryServant.tbl.put( new ObjKey(oid), svt );
        }

        // Cookieの作成
        cookie.value = new ObjKey(oid);

        return( svt );
    }

    public void postinvoke( byte[] oid,
                            org.omg.PortableServer.POA adapter,
                            java.lang.String operation,
                            java.lang.Object cookie,
                            Servant the_servant ) {

        // 変数の初期化を行う。
        the_servant = null;
    }
}

// ObjKey
class ObjKey{
    private byte[]  key;

    public ObjKey( byte[] _key ) {
        key = _key;
    }

    public int hashCode() {
        return (int)key[0];
    }

    public boolean equals( java.lang.Object comp ) {
        int i = key.length;
        if ( i != ((ObjKey)comp).key.length ) {
            return false;
        }
        for ( i--; i>=0; i-- ) {
            if ( key[i] != ((ObjKey)comp).key[i] ) {
                return false;
            }
        }

        return true;
    }
}

// Servant：メソッド実装クラス（スケルトンを継承）
class UserServant extends intfPOA {

    private int    value = 0;

    public int value() {
        return( this.value );
    }

    public void add( int a ) {
        this.value += a;
    }

    public void sub( int b ) {
        this.value -= b;
```

```
    }
}
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“5.10 クライアントアプリケーションの例外処理”を参照してください。

5.19.5 AdapterActivator使用例（find_POA時）

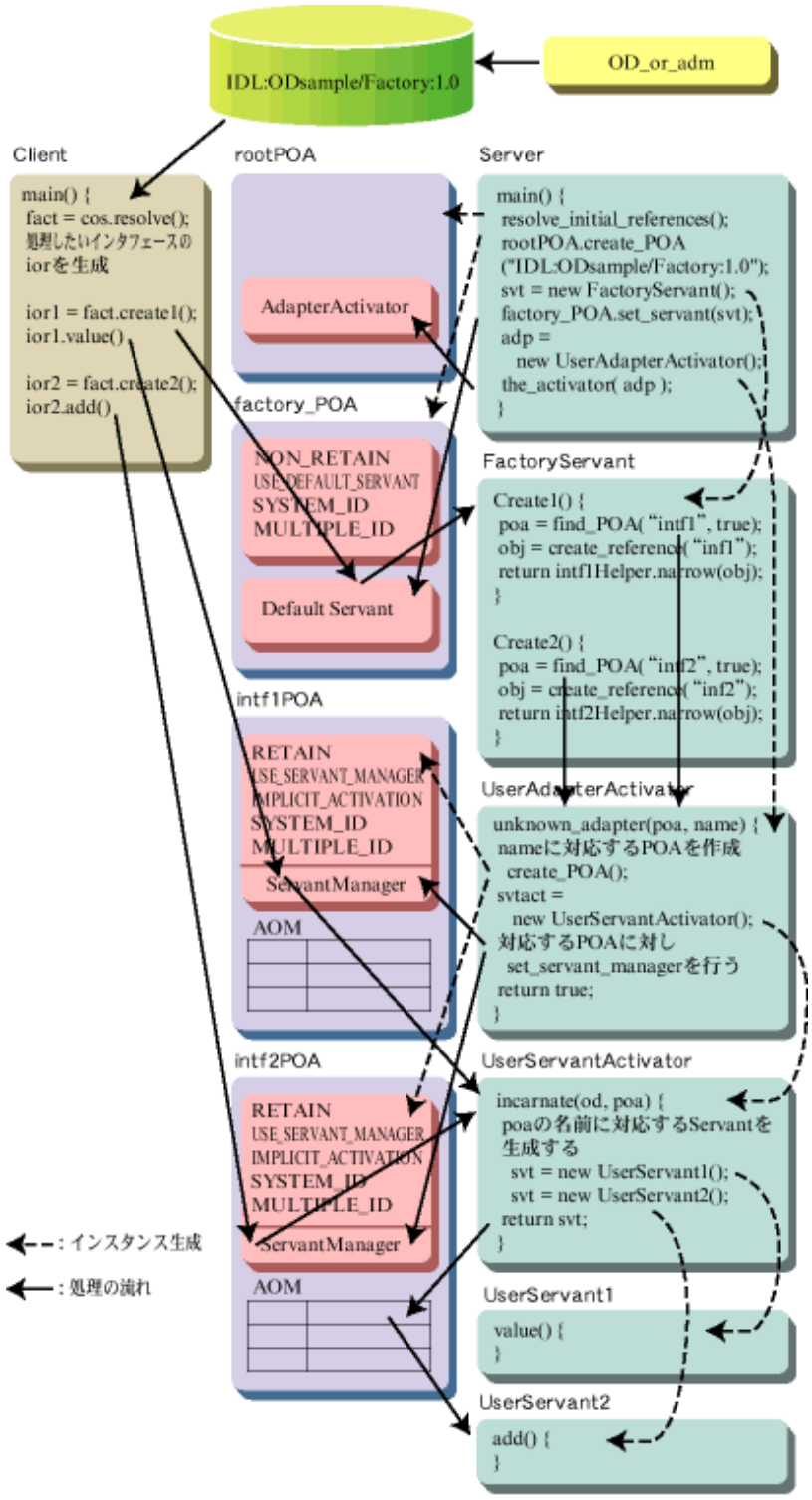
(1)IDL定義

```
module ODsample{
    interface intf1{
        attribute string value;
    };
    interface intf2{
        readonly attribute long value;
        void add( in long a );
        void sub( in long b );
    };
    interface Factory {
        intf1  create1 ();
        intf2  create2 ();
        void   destroy1( in intf1 obj );
        void   destroy2( in intf2 obj );
    };
};
```

(2)ネーミングサービスの登録

```
OD_or_adm -c IDL:ODsample/Factory:1.0 -n ODsample::POAsample9
```

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client {
    public static void main( String args[] ) {
```

```
try {
    // ORBの生成と初期化
    ORB Orb = ORB.init( args, null );

    // ネーミングサービスのオブジェクトリファレンスの取得
    org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
    NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

    // ネーミングサービスのresolveメソッドを発行し、
    // サーバアプリケーションのオブジェクトリファレンスの獲得
    String NCid = new String( "ODsample::POAsample9" ); // オブジェクト名
    String NCkind = new String( "" );                   // オブジェクトタイプ
    NameComponent nc =  new NameComponent( NCid, NCkind );
    NameComponent NCo[] = { nc };
    org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

    int    in   = 0;    // inパラメタ用変数
    String line = null;

    // Factoryのオブジェクトリファレンス獲得
    Factory target = FactoryHelper.narrow( Obj );

    // Factoryでインタフェース1用オブジェクトリファレンスを作成する
    intf1 _intf1 = target.create1();

    // サーバアプリケーションのメソッド呼出し
    System.out.println( "String = " + _intf1.value() );

    System.out.print( "String => " );
    line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();

    // サーバアプリケーションのメソッド呼出し
    _intf1.value( line );
    System.out.println( "String = " + _intf1.value() );

    // インスタンスの解放
    target.destroy1( _intf1 );

    // Factoryでインタフェース2用オブジェクトリファレンスを作成する
    intf2 _intf2 = target.create2();

    try {
        // サーバアプリケーションのメソッド呼出し
        System.out.println( "value = " + _intf2.value() );

        System.out.print( "add => " );
        line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
        in = Integer.parseInt( line );

        // サーバアプリケーションのメソッド呼出し
        _intf2.add( in );
        System.out.println( "value = " + _intf2.value() );

        System.out.print( "sub => " );
        line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
        in = Integer.parseInt( line );

        // サーバアプリケーションのメソッド呼出し
        _intf2.sub( in );
        System.out.println( "value = " + _intf2.value() );

        // インスタンスの解放
```

```
                target.destroy2( _intf2 );
            }
            catch ( java.lang.NumberFormatException e ){
                System.exit( 255 );
            }
        }
        catch( org.omg.CORBA.SystemException se ) {
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
             System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}
```

(5)サーバアプリケーション

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// ユーザアプリケーション：メイン処理クラス
public class Server {
    public static void main( String args[] ) {

        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // RootPOAのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "RootPOA" );
            // RootPOAのPOAオブジェクト獲得
            POA rootPOA = POAHelper.narrow( _tmpObj );

            // Factoryインタフェース用のPOA作成
            // ポリシリスト作成
            org.omg.CORBA.Policy factory_policies[] = new org.omg.CORBA.Policy[4];
            factory_policies[0] = rootPOA.create_servant_retention_policy(
                                  ServantRetentionPolicyValue.NON_RETAIN );
            factory_policies[1] = rootPOA.create_request_processing_policy(
                                  RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
            factory_policies[2] = rootPOA.create_id_assignment_policy(
                                  IdAssignmentPolicyValue.SYSTEM_ID );
            factory_policies[3] = rootPOA.create_id_uniqueness_policy(
                                  IdUniquenessPolicyValue.MULTIPLE_ID );
            POA factory_POA = rootPOA.create_POA( "IDL:ODsample/Factory:1.0",
                                                  null,
                                                  factory_policies);

            // FactoryServantの生成（コンストラクタにインプリ用POAを渡す）
            Servant svt = new FactoryServant( rootPOA );
            // Factoryインタフェース用POAのDefault Servantに設定
            factory_POA.set_servant( svt );

            // AdapterActivatorの生成
```

```
            AdapterActivator adp = new UserAdapterActivator();
            // インプリ用POAのAdapterActivatorに設定
            rootPOA.the_activator( adp );

            // POAマネージャの獲得
            POAManager poamanager = rootPOA.the_POAManager();

            // POAマネージャのactivate
            poamanager.activate();
            Orb.run();
        }
        catch( org.omg.CORBA.SystemException se ) {
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
             System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}

// FactoryServant：Factoryメソッド実装クラス（スケルトンを継承）
class FactoryServant extends FactoryPOA {

    private POA poa = null;

    // コンストラクタ
    public FactoryServant( POA poa ) {
        this.poa = poa;
    }

    public intf1 create1() {
        intf1      ior;      // UserServantのオブジェクトリファレンス

        try {
            // インタフェースリポジトリIDに対応するPOAオブジェクト獲得
            POA intf1POA = this.poa.find_POA( "intf1", true );

            // オブジェクトリファレンスの生成
            org.omg.CORBA.Object _tmpObj = intf1POA.create_reference( "IDL:ODsample/intf1:1.0" );
            ior = intf1Helper.narrow( _tmpObj );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "create error: " + e );
            return( null );
        }

        return( ior );

    }

    public intf2 create2() {
        intf2      ior;                  // UserServantのオブジェクトリファレンス

        try {
            // インタフェースリポジトリIDに対応するPOAオブジェクト獲得
            POA intf2POA = this.poa.find_POA( "intf2", true );
```

```
            // オブジェクトリファレンスの生成
            org.omg.CORBA.Object _tmpObj = intf2POA.create_reference( "IDL:ODsample/intf2:1.0" );
            ior = intf2Helper.narrow( _tmpObj );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "create error: " + e );
            return( null );
        }

       return( ior );

    }

    public void destroy1( intf1 obj ) {

        try {
            // インタフェースIDに対応するPOAオブジェクト獲得
            POA intf1POA = this.poa.find_POA( "intf1", false );

            // オブジェクトリファレンスからオブジェクトIDを求める
            byte oid[] = intf1POA.reference_to_id( obj );

            // Servantをdeactiveにする
            intf1POA.deactivate_object( oid );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "destroy error: " + e );
        }
    }

    public void destroy2( intf2 obj ) {

        try{
            // インタフェースIDに対応するPOAオブジェクト獲得
            POA intf2POA = this.poa.find_POA( "intf2", false );

            // オブジェクトリファレンスからオブジェクトIDを求める
            byte oid[] = intf2POA.reference_to_id( obj );

            // Servantをdeactiveにする
            intf2POA.deactivate_object( oid );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "destroy error: " + e );
        }
    }
}

// アダプタアクティベータ：POA生成クラス（AdapterActivatorを継承）
class UserAdapterActivator extends LocalObject implements AdapterActivator {

    public boolean unknown_adapter( POA parent, String name ) {

        // ポリシリスト作成
        org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[4];
        policies[0] = parent.create_servant_retention_policy(
                                ServantRetentionPolicyValue.RETAIN );
        policies[1] = parent.create_request_processing_policy(
                                RequestProcessingPolicyValue.USE_SERVANT_MANAGER );
        policies[2] = parent.create_id_assignment_policy(
                                IdAssignmentPolicyValue.SYSTEM_ID );
        policies[3] = parent.create_id_uniqueness_policy(
```

```
                                IdUniquenessPolicyValue.MULTIPLE_ID );

        // ServantActivatorの生成
        ServantActivator svtact = new UserServantActivator();

        try {
            if( name.equals( "intf1" ) ) {
                // intf1用のPOA作成
                POA intf1POA = parent.create_POA( name, null, policies );

                // ServantManager登録
                intf1POA.set_servant_manager( svtact );

                return( true );
            }
            else if( name.equals( "intf2") ) {
                // intf2用のPOA作成
                POA intf2POA = parent.create_POA( name, null, policies );

                // ServantManager登録
                intf2POA.set_servant_manager( svtact );

                return( true );
            }
        }
        catch( Exception e ) {
            System.err.println( "unknown_adapter error: " + e );
            return( false );
        }

        return( true );
    }
}

// Servantアクティベータ：Servant生成クラス（ServantActivatorを継承）
class UserServantActivator extends LocalObject implements ServantActivator {
    public Servant incarnate( byte[] oid, POA adapter ) {

        Servant svt = null;

        if ( adapter.the_name().equals( "intf1") )
            svt = new UserServant1();
        else if ( adapter.the_name().equals( "intf2") )
            svt = new UserServant2();

        return( svt );
    }

    public void etherealize( byte[] oid,
                             POA adapter,
                             Servant serv,
                             boolean cleanup_in_progress,
                             boolean remaining_activations ) {

        // 変数の初期化を行う。
        serv = null;
    }
}

// Servant：メソッド実装クラス（スケルトンを継承）
class UserServant1 extends intf1POA {

    private String  value = "";
```

```
    public java.lang.String value() {
        return( this.value );
    }

    public void value( java.lang.String value ) {
        this.value = value;
    }
}

class UserServant2 extends intf2POA {

    private int    value = 0;

    public int value() {
        return( this.value );
    }

    public void add( int a ) {
        this.value += a;
    }

    public void sub( int b ) {
        this.value -= b;
    }
}
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“5.10 クライアントアプリケーションの例外処理”を参照してください。

5.19.6 AdapterActivator使用例（リクエスト受信時）

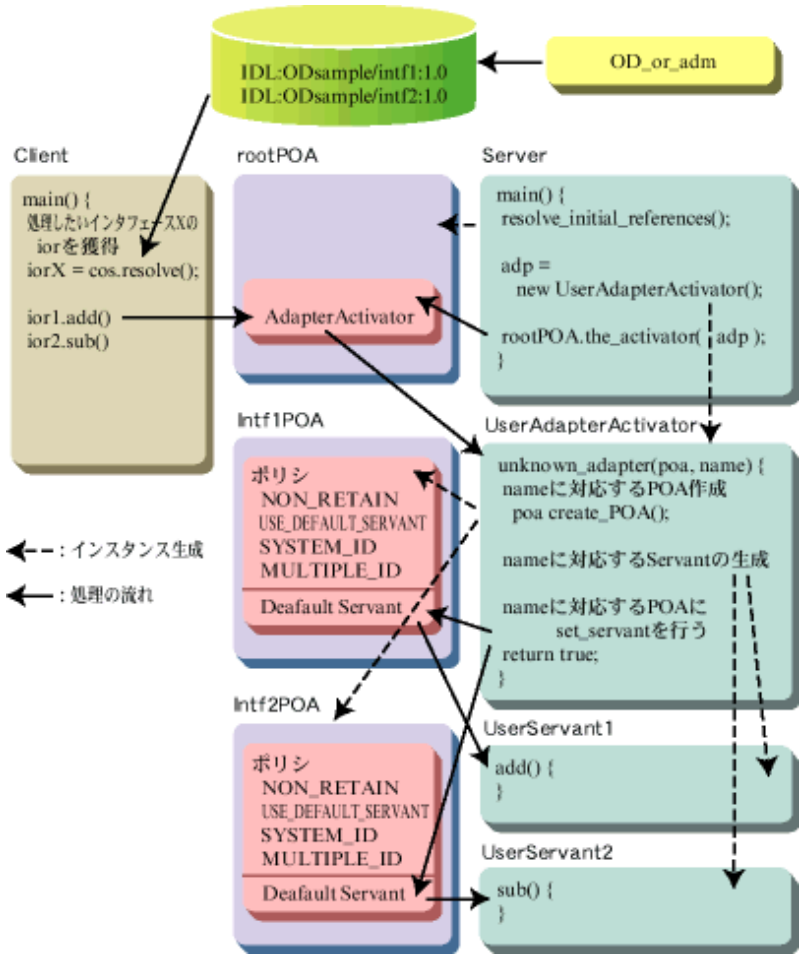
(1)IDL定義

```
module ODsample{
    interface    intf1{
        long  add( in long a, in long b );
    };
    interface  intf2{
        long  sub( in long a, in long b );
    };
};
```

(2)ネーミングサービスの登録

```
OD_or_adm -c IDL:ODsample/intf1:1.0  -a Imple_POAsample10 -n ODsample::POAsample10-1
OD_or_adm -c IDL:ODsample/intf2:1.0  -a Imple_POAsample10 -n ODsample::POAsample10-2
```

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client {
    public static void main( String args[] ) {

        int  in1 = 0;       // inパラメタ用変数
        int  in2 = 0;       // inパラメタ用変数
        String line = null;

        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // ネーミングサービスのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // ネーミングサービスのresolveメソッドを発行し、
            // サーバアプリケーションのオブジェクトリファレンスの獲得
            String NCid = new String( "ODsample::POAsample10-1" ); // オブジェクト名
            String NCkind = new String( "" );                      // オブジェクトタイプ
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
```

```
    org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

    // サーバアプリケーションのオブジェクトリファレンス獲得
    intf1 _intf1 = intf1Helper.narrow( Obj );

    try {
        System.out.print( "in1 => " );
        line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
        in1 = Integer.parseInt( line );

        System.out.print( "in2 => " );
        line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
        in2 = Integer.parseInt( line );
    }
    catch ( java.lang.NumberFormatException e ) {
        System.exit( 255 );
    }

    // サーバアプリケーションのメソッド呼出し
    int result = _intf1.add( in1, in2 );

    // メソッドの結果表示
    System.out.println( in1 + " + " + in2 + " = "+ result );

    // ネーミングサービスのresolveメソッドを発行し、
    // サーバアプリケーションのオブジェクトリファレンスの獲得
    String NCid2 = new String( "ODsample::POAsample1O-2" ); // オブジェクト名
    String NCkind2 = new String( "" );                      // オブジェクトタイプ
    NameComponent nc2 =  new NameComponent( NCid2, NCkind2 );
    NameComponent NCo2[] = { nc2 };
    org.omg.CORBA.Object  Obj2 = Cos.resolve( NCo2 );

    // サーバアプリケーションのオブジェクトリファレンス獲得
    intf2 _intf2 = intf2Helper.narrow( Obj2 );

    try {
        System.out.print( "in1 => " );
        line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
        in1 = Integer.parseInt( line );

        System.out.print( "in2 => " );
        line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
        in2 = Integer.parseInt( line );
    }
    catch ( java.lang.NumberFormatException e ) {
        System.exit( 255 );
    }

    // サーバアプリケーションのメソッド呼出し
    result = _intf2.sub( in1, in2 );

    // メソッドの結果表示
    System.out.println( in1 + " - " + in2 + " = "+ result );
}
catch( org.omg.CORBA.SystemException se ) {
    System.out.println( "ERROR : " + se.getClass().getName()
         + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
     System.exit( 255 );
}
catch( org.omg.CORBA.UserException ue ) {
    System.out.println( "ERROR : " + ue.getClass().getName() );
    System.exit( 255 );
}
```

```
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}
```

(5)サーバアプリケーション

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// ユーザアプリケーション：メイン処理クラス
public class Server {
    public static void main(String args[]) {

        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // RootPOAのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "RootPOA" );
            // RootPOAのPOAオブジェクト獲得
            POA rootPOA = POAHelper.narrow( _tmpObj );

            // AdapterActivatorの生成
            AdapterActivator adp = new UserAdapterActivator();
            // インプリ用POAのAdapterActivatorに設定
            rootPOA.the_activator( adp );
            // POAマネージャの獲得
            POAManager poamanager = rootPOA.the_POAManager();
            // POAマネージャのactivate
            poamanager.activate();
            Orb.run();
        }
        catch( org.omg.CORBA.SystemException se ) {
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
             System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}

// アダプタアクティベータ：POA生成クラス（AdapterActivatorを継承）
class UserAdapterActivator extends LocalObject implements AdapterActivator {
    public boolean unknown_adapter( POA parent, String name ) {

        POA        inf_POA;            // インタフェース用POA
        // ポリシリスト作成
        org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[4];
        policies[0] = parent.create_servant_retention_policy(
                            ServantRetentionPolicyValue.NON_RETAIN );
        policies[1] = parent.create_request_processing_policy(
```

```
                            RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
        policies[2] = parent.create_id_assignment_policy(
                            IdAssignmentPolicyValue.SYSTEM_ID );
        policies[3] = parent.create_id_uniqueness_policy(
                            IdUniquenessPolicyValue.MULTIPLE_ID );
        try {
            if( name.equals( "IDL:ODsample/intf1:1.0" ) ) {
                // intf1用のPOA作成
                inf_POA = parent.create_POA( name, null, policies );

                // Servantの生成
                Servant svt = new UserServant1();
                // Default Servantに設定
                inf_POA.set_servant( svt );

                return( true );
            }
            else if( name.equals( "IDL:ODsample/intf2:1.0" ) ) {
                // intf2用のPOA作成
                inf_POA = parent.create_POA( name, null, policies );

                // Servantの生成
                Servant svt = new UserServant2();
                // Default Servantに設定
                inf_POA.set_servant( svt );

                return( true );
            }
        }
        catch( Exception e ) {
            System.err.println( "unknown_adapter error: " + e );
            return( false );
        }

        return( true );
    }
}

// Servant：メソッド実装クラス（スケルトンを継承）
class UserServant1 extends intf1POA {
    public int add( int a, int b ) {
        return( a + b );
    }
}

class UserServant2 extends intf2POA {
    public int sub( int a, int b ) {
        return( a - b );
    }
}
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“5.10 クライアントアプリケーションの例外処理”を参照してください。

5.19.7 代理方式の実装例（デフォルトインスタンス方式の例）

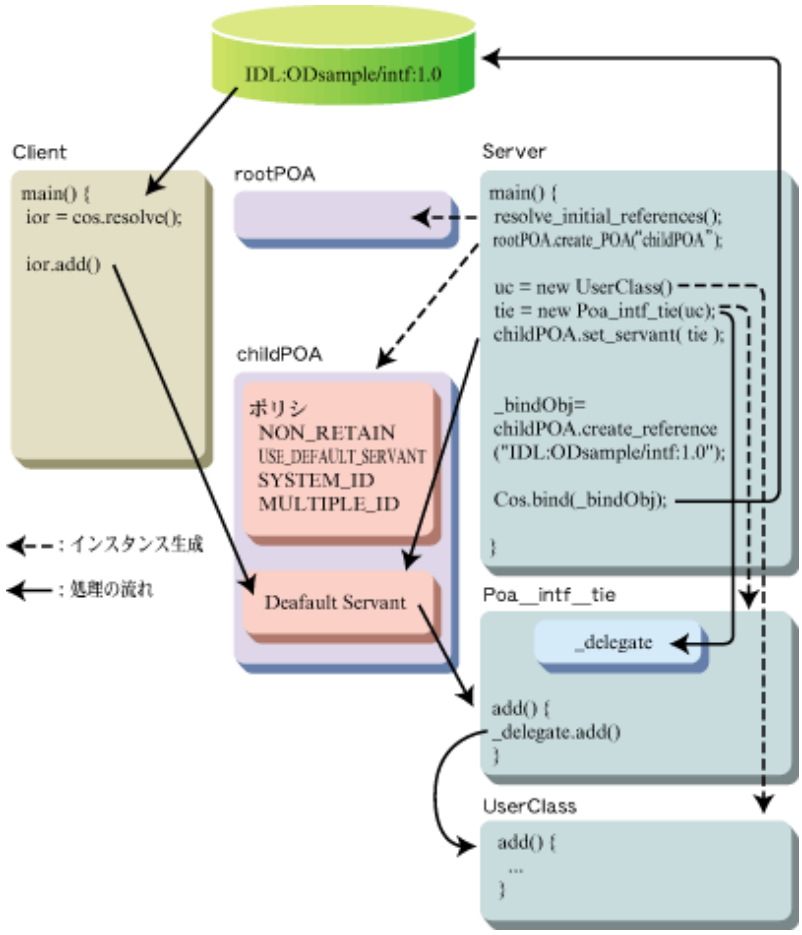
(1)IDL定義

```
module ODsample{
    interface  intf{
        long  add( in  long  a,  in  long  b );
    };
};
```

(2)ネーミングサービスの登録

動的登録

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client {
    public static void main( String args[] ) {

        try {
```

```
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // ネーミングサービスのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // ネーミングサービスのresolveメソッドを発行し、
            // サーバアプリケーションのオブジェクトリファレンスの獲得
            String NCid = new String( "ODsample::POAsample1" ); // オブジェクト名
            String NCkind = new String( "" );                   // オブジェクトタイプ
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

            // サーバアプリケーションのオブジェクトリファレンス獲得
            intf target = intfHelper.narrow( Obj );

            int  in1 = 0;     // inパラメタ用変数
            int  in2 = 0;     // inパラメタ用変数
            int  result;      // 復帰値用変数
            String line = null;

            try {
                System.out.print( "in1 => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in1 = Integer.parseInt( line );

                System.out.print( "in2 => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in2 = Integer.parseInt( line );
            }
            catch ( java.lang.NumberFormatException e ) {
                System.exit( 255 );
            }

            // サーバアプリケーションのメソッド呼出し
            result = target.add( in1, in2 );

            // メソッドの結果表示
            System.out.println( in1 + " + " + in2 + " = "+ result );
        }
        catch( org.omg.CORBA.SystemException se ) {
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
            System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}
```

(5)サーバアプリケーション

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
```

```
import org.omg.CosNaming.*;
import ODsample.*;

// ユーザアプリケーション：メイン処理クラス
public class Server {
    public static void main( String args[] ) {

        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // RootPOAのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj =
            Orb.resolve_initial_references( "RootPOA" );
            // RootPOAのPOAオブジェクト獲得
            POA rootPOA = POAHelper.narrow( _tmpObj );

            // インタフェース用のPOA作成
            // ポリシリスト作成
            org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[4];
            policies[0] = rootPOA.create_servant_retention_policy(
                          ServantRetentionPolicyValue.NON_RETAIN );
            policies[1] = rootPOA.create_request_processing_policy(
                          RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
            policies[2] = rootPOA.create_id_assignment_policy(
                          IdAssignmentPolicyValue.SYSTEM_ID );
            policies[3] = rootPOA.create_id_uniqueness_policy(
                          IdUniquenessPolicyValue.MULTIPLE_ID );

            // Default Servant用のPOAオブジェクトの作成
            POA childPOA = rootPOA.create_POA( "childPOA", null, policies );
            // ユーザクラスのインスタンス生成
            UserClass uc = new UserClass();
            // tieオブジェクト作成とユーザクラスの登録
            intfPOATie  tie = new intfPOATie( uc );
            // POAのDefault Servantに設定
            childPOA.set_servant( tie );

            org.omg.CORBA.Object _bindObj = childPOA.create_reference( "IDL:ODsample/intf:1.0" );

            // ネーミングサービスのオブジェクトリファレンスの取得
            _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // サーバアプリケーションのオブジェクトリファレンスの獲得
            String NCid   = new String( "ODsample::POAsample1" );
            String NCkind = new String( "" );
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            try {
                Cos.unbind( NCo );
            } catch( Exception e ) {
                ;
            }
            Cos.bind( NCo, _bindObj );

            // POAマネージャの獲得
            POAManager poamanager = childPOA.the_POAManager();

            // POAマネージャのactivate
            poamanager.activate();
            Orb.run();
        }
```

```
        catch( org.omg.CORBA.SystemException se ) {
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
            System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}

//ユーザクラス（インタフェース実装クラス）
class UserClass
extends OtherUserClass
implements ODsample.intfOperations
{
    public int add( int a, int b ){
        return( a + b );
    }
}

//他のユーザクラス
class OtherUserClass
{
    public int sub( int a, int b ){
        return( a - b );
    }
}
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“5.10 クライアントアプリケーションの例外処理”を参照してください。

5.19.8 Active Object Map（AOM）使用例（Factory-1方式＋インスタンス解放）

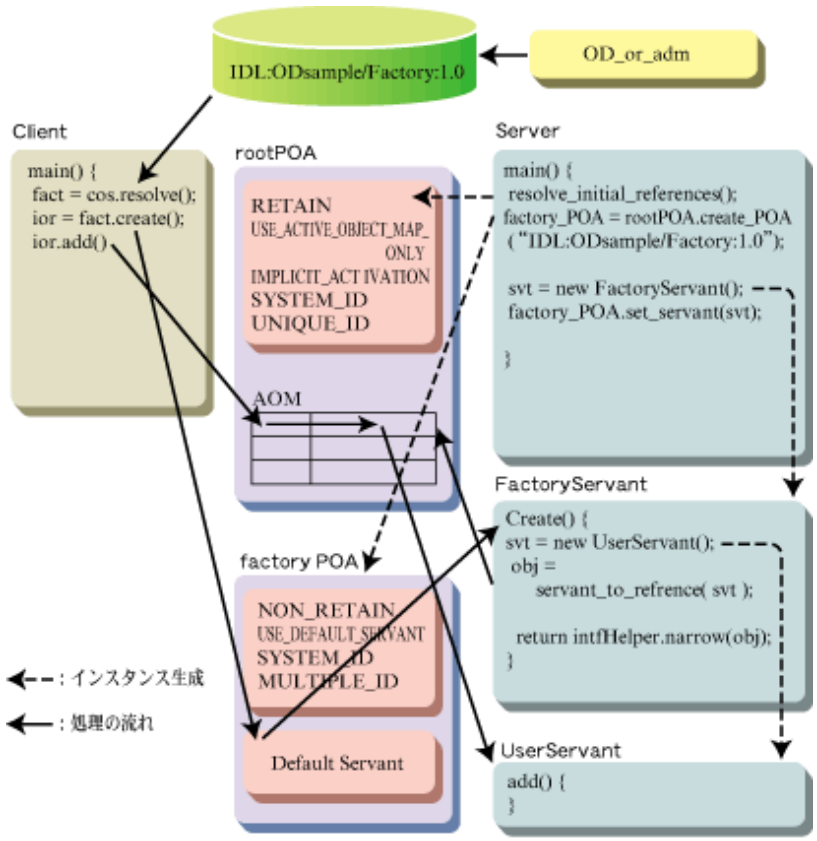
(1)IDL定義

```
module ODsample{
    interface intf{
        readonly  attribute  long  value;
        void  add( in  long  a );
        void  sub( in  long  b );
    };
    interface Factory {
        intf   create( in string username );
        void   destroy( in  intf  obj );
    };
};
```

(2)ネーミングサービスの登録

```
OD_or_adm -c IDL:ODsample/Factory:1.0 -n ODsample::POAsample5
```

(3)アプリケーション構成概要図



(4)クライアントアプリケーション

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client
{
    public static void main(String args[])
    {
        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // ネーミングサービスのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // ネーミングサービスのresolveメソッドを発行し、
            // サーバアプリケーションのオブジェクトリファレンスの獲得
            String NCid   = new String( "ODsample::POAsample5" );
            String NCkind = new String( "" );
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

            // Factoryのオブジェクトリファレンスの獲得
```

```
            Factory target = FactoryHelper.narrow( Obj );

            int  in;
            String line = null;

            try {
                // ユーザIDの入力
                System.out.print( "Input your Name => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                if( line.length() == 0 ) {
                    line = new String( "guest" );
                }

                // Factoryでインタフェース用オブジェクトリファレンス作成
                intf _intf = target.create( line );

                // サーバアプリケーションのメソッド呼出し
                System.out.println( "value = " + _intf.value() );

                System.out.print( "add => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // サーバアプリケーションのメソッド呼出し
                _intf.add( in );
                System.out.println( "value = " + _intf.value() );

                System.out.print( "sub => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // サーバアプリケーションのメソッド呼出し
                _intf.sub( in );
                System.out.println( "value = " + _intf.value() );
            }
            catch (java.lang.NumberFormatException e){
                System.exit(255);
            }
        }
        catch( org.omg.CORBA.SystemException se ) {
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
             System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}
```

(5)サーバアプリケーション

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// ユーザアプリケーション：メイン処理クラス
```

```
public class Server
{
    public static void main(String args[])
    {

        try {
            // ORBの生成と初期化
            ORB Orb = ORB.init( args, null );

            // RootPOAのオブジェクトリファレンスの取得
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "RootPOA" );

            // RootPOAのPOAオブジェクト獲得
            POA rootPOA  = POAHelper.narrow( _tmpObj );

            // Factoryインタフェース用のPOA作成
            // ポリシリスト作成
            org.omg.CORBA.Policy factory_policies[] = new org.omg.CORBA.Policy[4];

            factory_policies[0] = rootPOA.create_servant_retention_policy(
                                  ServantRetentionPolicyValue.NON_RETAIN );
            factory_policies[1] = rootPOA.create_request_processing_policy(
                                  RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
            factory_policies[2] = rootPOA.create_id_assignment_policy(
                                  IdAssignmentPolicyValue.SYSTEM_ID );
            factory_policies[3] = rootPOA.create_id_uniqueness_policy(
                                  IdUniquenessPolicyValue.MULTIPLE_ID );
            POA factory_POA = rootPOA.create_POA( "IDL:ODsample/Factory:1.0",
                                                  null,
                                                  factory_policies );

            // FactoryServantの生成
            Servant svt = new FactoryServant( rootPOA );

            // Factoryインタフェース用POAをDefault Servantに設定
            factory_POA.set_servant( svt );

            // POAマネージャの獲得
            POAManager poamanager = rootPOA.the_POAManager();

            // インスタンス解放処理の登録
            com.fujitsu.ObjectDirector.PortableServer.POAdisconnect clrel = new instrelease();
            clrel.setDisconnect( clrel );

            // POAマネージャのactivate
            poamanager.activate();
            Orb.run();
        }
        catch( org.omg.CORBA.SystemException se ) {
            System.out.println( "ERROR : " + se.getClass().getName()
                 + " : minor = 0x" + java.lang.Integer.toHexString(se.minor) );
             System.exit( 255 );
        }
        catch( org.omg.CORBA.UserException ue ) {
            System.out.println( "ERROR : " + ue.getClass().getName() );
            System.exit( 255 );
        }
        catch ( Exception e ) {
            System.err.println( "ERROR : " + e );
            System.exit( 255 );
        }
    }
}
```

```
// FactoryServant：Factoryメソッド実装クラス（スケルトンを継承）
class FactoryServant extends FactoryPOA
{
    private POA poa = null;
    static public java.util.Hashtable table;

    // コンストラクタ
    public FactoryServant() {
        table = new java.util.Hashtable();
    }
    public FactoryServant( POA param_poa ) {
        this();
        poa = param_poa;
    }

    public intf create( String UserName )
    {
        intf ior;

        try {
            // Servantの検索
            Servant svt = (Servant)table.get( UserName );

            // みつからない場合、Servantを作成（guestは毎回作成）
            if( null == svt ) {
                svt = new UserServant( UserName );
                table.put( UserName, svt );

                // インスタンス数が100を超えるようならば無条件解放に変更する
                if( table.size() > 100 ) {
                    com.fujitsu.ObjectDirector.PortableServer.POAdisconnect clrel =
                        new ALLinstrelease();
                    clrel.resetDisconnect( clrel );
                }
            }

            // Servantからオブジェクトリファレンスの生成
            // IMPLICIT_ACTIVATIONポリシを指定しているので、AOMへ自動登録される
            org.omg.CORBA.Object _tmpObj = this.poa.servant_to_reference( svt );
            ior = intfHelper.narrow( _tmpObj );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.out.println( "create error: " + e );
            e.printStackTrace(System.err);
            throw new org.omg.CORBA.COMM_FAILURE();
        }

        return( ior );
    }

    public void destroy( intf obj )
    {
        try{
            // オブジェクトリファレンスからオブジェクトIDを求める
            byte oid[] = this.poa.reference_to_id( obj );

            // Servantをdeactiveにする
            this.poa.deactivate_object( oid );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.out.println( "destroy error: " + e );
        }
```

```
    }
}

// Servant : メソッド実装クラス（スケルトンを継承）
class UserServant extends intfPOA
{
    private int    value = 0;
    public String UserName = null;

    public UserServant( String UserName ) {
        this.UserName = UserName;
    }

    public int value() {
        return( this.value );
    }

    public void add( int a ) {
        this.value += a;
    }

    public void sub( int b ) {
        this.value -= b;
    }

}

// 条件付きインスタンス解放クラス（POAdisconnectを継承）
class instrelease
    extends com.fujitsu.ObjectDirector.PortableServer.POAdisconnect {

    public boolean release_instance( org.omg.PortableServer.POA        POA,
                                     org.omg.PortableServer.Servant servant,
                                     byte[]                         oid ) {

        UserServant svt = (UserServant)servant;
        if( svt.UserName.equals( "guest" ) ) {
            FactoryServant.table.remove( svt.UserName );
            return true;
        }

        return false;
    }
}

// 無条件インスタンス解放クラス（POAdisconnectを継承）
class ALLinstrelease
    extends com.fujitsu.ObjectDirector.PortableServer.POAdisconnect {

    public boolean release_instance( org.omg.PortableServer.POA        POA,
                                     org.omg.PortableServer.Servant servant,
                                     byte[]                         oid ) {

        UserServant svt = (UserServant)servant;

        FactoryServant.table.remove( svt.UserName );

        return true;
    }
}
```

(6)例外情報の獲得

サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“5.10 クライアントアプリケーションの例外処理”を参照してください。

5.20 データ型に対するマッピング

5.20.1 基本データ型

ここでは、クライアント／サーバアプリケーションをJava言語で作成する場合に使用するデータ型について説明します。CORBAで定義されている基本データ型をJavaプログラムで使用する場合、対応するデータ型は以下のように定義されます。

CORBAデータ型		Java	備考
整数型	long	int	
	unsigned long	int	
	short	short	
	unsigned short	short	
	long long	long	
浮動小数点型	float	float	
	double	double	
文字型	char	char (注)	
	wchar		
オクテッド型	octet	byte	
ブーリアン型	boolean	boolean	
文字列型	string	java.lang.String	“5.20.2 文字列型”参照
	wstring		“5.20.3 ワイド文字列型”参照
列挙型	enum	<enum名>クラス	“5.20.4 列挙型”参照
any型	any	org.omg.CORBA.Any	“5.20.5 any型”参照
オブジェクトリファレンス	Object	org.omg.CORBA.Object	
タイプコード	TypeCode	org.omg.CORBA.TypeCode	

注）IDL定義でcharを使用した場合、日本語文字（マルチバイト文字）は扱えません。日本語文字を使用する場合は、wcharを使用してください。

5.20.2 文字列型

(1)IDLマッピング

IDL言語で文字列型stringを指定した場合は、Java言語で記述すると、java.lang.Stringクラスとorg.omg.CORBA.StringHolderクラスになります。
java.lang.Stringクラスは、inパラメタおよび復帰値を扱う場合に使用します。
org.omg.CORBA.StringHolderクラスは、以下のようになります。本クラスは、out／inoutパラメタを受け渡す場合に使用します。

＜org.omg.CORBA.StringHolderクラス＞

```
final public class StringHolder {
      public java.lang.String value;
      public StringHolder() { }
      public StringHolder( java.lang.String initial ) {...}
}
```

org.omg.CORBA.StringHolderクラス中で定義されているメンバの意味を以下に示します。

org.omg.CORBA.StringHolderクラスのメンバ	意味
value	要素の値。
デフォルトコンストラクタ	valueメンバへの設定なしで、インスタンスを生成するために使用します。
コンストラクタ	valueメンバにパラメタで指定された要素の値を設定します。

IDL言語での定義が以下のように定義されていた場合について、アプリケーションプログラム例を示します。

【IDL言語】

```
module ODsample{
    interface  stringtest{
        string op1( in string str1, out string str2, inout string str3 );
    };
};
```

これをJava言語で記述すると、以下のようになります。

【Java言語】

＜インタフェース＞

```
package ODsample;
public interface stringtestOperations {
    public java.lang.String op1(java.lang.String str1,
                                org.omg.CORBA.StringHolder str2,
                                org.omg.CORBA.StringHolder str3 );
}
```

(2)クライアントアプリケーションでの処理

```
import org.omg.CORBA.*;
import ODsample.*;

public class stringClient {
    public static void main( String args[] ) {
        // ORBの前処理
        // オブジェクトリファレンスの獲得
              :
        try{
            // inパラメタ用変数
            String  in              = "in_str:data";
            // outパラメタ用変数
            StringHolder outHolder   = new StringHolder();
```

```
            // inoutパラメタ用変数
            StringHolder inoutHolder = new StringHolder( "inout_str:data" );
            // 復帰値用変数
            String       result;

           // サーバアプリケーションのメソッド呼出し
           result = target.op1( in, outHolder, inoutHolder );

           // メソッドの結果表示
           System.out.println( result );
           System.out.println( outHolder.value );
           System.out.println( inoutHolder.value );
        }
        catch ( java.lang.Exception e ) {
            // エラー処理
                :
        }
    }
}
```

Java言語でinパラメタをサーバアプリケーションに受け渡す場合は、直接java.lang.Stringクラスのインスタンスにデータをセットします。また、out／inoutパラメタとしてサーバアプリケーションにパラメタを受け渡す場合は、new演算子を使用して各データ型のorg.omg.CORBA.StringHolderクラスのインスタンスを生成します。inoutパラメタには、java.lang.Stringクラスのデータを設定します。設定には、以下の2つの方法があります。

【デフォルトコンストラクタを使用した方法】

```
StringHolder a = new StringHolder();   //デフォルトコンストラクタ
a.value = "in_str:data";               //データの代入
```

【コンストラクタを使用した方法】

```
StringHolder b = new StringHolder( "in_str:data" );  //コンストラクタ
```

なお、サーバの処理結果は、out／inoutパラメタの各インスタンス内のvalueに値が設定されます。復帰値は、java.lang.Stringクラスのインスタンスに直接設定されます。

(3)サーバアプリケーションでの処理

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servantクラス
// サーバアプリケーションのメソッド
class stringServant extends stringtestPOA {

    public String op1( String in, StringHolder outHolder, StringHolder inoutHolder ) {
        // 入力パラメタの表示
        System.out.println( in );
        System.out.println( inoutHolder.value );

        // 出力パラメタの設定
        outHolder.value = "out_str:data";
        inoutHolder.value = "inout-res_str:data";
```

```
        // 復帰
        return( "result_str:data" );
    }
}

public class stringServer {
    public static void main( String args[] ) {
        try {
           // ORBの前処理
                :
           // POAオブジェクトの生成
                :
           // Servantの生成とそのPOAへの登録
                :
           // POAManagerの活性化
                :
        }
        catch ( java.lang.Exception e ) {
            // エラー処理
                :
        }
    }
}
```

Java言語でinパラメタを参照する場合は、inパラメタのインスタンスを参照します。inoutパラメタを参照するには、org.omg.CORBA.StringHolderクラスからjava.lang.Stringクラスのインスタンスを取り出し、このインスタンスを参照します。また、out／inoutパラメタとしてクライアントアプリケーションにパラメタを返す場合は、org.omg.CORBA.StringHolderクラスからjava.lang.Stringクラスのインスタンスを取り出し、このインスタンスに直接設定します。

5.20.3 ワイド文字列型

(1)IDLマッピング

IDL言語で文字列型wstringを指定した場合は、Java言語で記述すると、java.lang.Stringクラスとorg.omg.CORBA.StringHolderクラスになります。
java.lang.Stringクラスは、inパラメタおよび復帰値を扱う場合に使用します。
org.omg.CORBA.StringHolderクラスは、以下のようになります。本クラスは、out／inoutパラメタを受け渡す場合に使用します。

＜org.omg.CORBA.StringHolderクラス＞

```
final public class StringHolder {
      public java.lang.String value;
      public StringHolder() { }
      public StringHolder(java.lang.String initial) {...}
}
```

org.omg.CORBA.StringHolderクラス中で定義されているメンバの意味を以下に示します。

org.omg.CORBA.StringHolderクラスのメンバ	意味
value	要素の値。
デフォルトコンストラクタ	valueメンバへの設定なしで、インスタンスを生成するために使用します。
コンストラクタ	valueメンバにパラメタで指定された要素の値を設定します。

IDL言語での定義が以下のように定義されていた場合について、アプリケーションプログラムの例を示します。

【IDL言語】

```
module ODsample{
    interface  wstringtest{
        wstring op1(in wstring str1, out wstring str2, inout wstring str3);
   };
};
```

これをJava言語で記述すると、以下のようになります。

【Java言語】

＜インタフェース＞

```
package ODsample;
public interface wstringtestOperations {
    public java.lang.String op1(java.lang.String str1,
    org.omg.CORBA.StringHolder str2,
    org.omg.CORBA.StringHolder str3 );
}
```

(2)クライアントアプリケーションでの処理

```
import org.omg.CORBA.*;
import ODsample.*;

public class stringClient {
    public static void main(String args[]) {
        // ORBの前処理
        // オブジェクトリファレンスの獲得
            :
        try{
            // inパラメタ用変数
            String  in              = "in_str:data";
            // outパラメタ用変数
            StringHolder outHolder   = new StringHolder();
            // inoutパラメタ用変数
            StringHolder inoutHolder = new StringHolder( "inout_str:data" );
            // 復帰値用変数
            String       result;

            // サーバアプリケーションのメソッド呼出し
            result = target.op1(in, outHolder, inoutHolder);

           // メソッドの結果表示
           System.out.println( result );
           System.out.println( outHolder.value );
           System.out.println( inoutHolder.value );
        }
        catch (Exception e) {
            // エラー処理
                :
        }
    }
}
```

Java言語でinパラメタをサーバアプリケーションに受け渡す場合は、直接java.lang.Stringクラスのインスタンスにデータをセットします。
また、out／inoutパラメタとしてサーバアプリケーションにパラメタを受け渡す場合は、new演算子を使用して各データ型のorg.omg.CORBA.StringHolderクラスのインスタンスを生成します。inoutパラメタには、java.lang.Stringクラスのデータを設定します。
設定には、以下の2つの方法があります。

【デフォルトコンストラクタを使用した方法】

```
StringHolder a = new StringHolder();   //デフォルトコンストラクタ
a.value = "in_str:data";               //データの代入
```

【コンストラクタを使用した方法】

```
StringHolder b = new StringHolder("in_str:data");  //コンストラクタ
```

なお、サーバの処理結果は、out、inoutパラメタの各インスタンスのvalueに値が設定されます。復帰値は、java.lang.Stringクラスのインスタンスに直接設定されます。

(3)サーバアプリケーションでの処理

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servantクラス
// サーバアプリケーションのメソッド
class stringServant extends stringtestPOA {
    public stringServant(String[] args, java.util.Properties props) {
        super( args, props );
    }

    public String op1( String in, StringHolder outHolder, StringHolder inoutHolder ) {
        // 入力パラメタの表示
        System.out.println( in );
        System.out.println( inoutHolder.value );

        // 出力パラメタの設定
        outHolder.value = "out_str:data";
        inoutHolder.value = "inout-res_str:data";

        // 復帰
        return( "result_str:data" );
    }
}

public class stringServer {
    public static void main(String args[]) {
        try {
            // ORBの前処理
                :
            // Servantの生成とそのORBへの登録
                :
            // オブジェクトリファレンスの登録
                :
            // クライアントからの要求待合せ
                :
        }
        catch (Exception e) {
            // エラー処理
```

```
                :
        }
    }
}
```

Java言語でinパラメタを参照する場合は、inパラメタのインスタンスを参照します。inoutパラメタを参照するには、org.omg.CORBA.StringHolderクラスからjava.lang.Stringクラスのインスタンスを取り出し、このインスタンスを参照します。また、out／inoutパラメタとしてクライアントアプリケーションにパラメタを返す場合、org.omg.CORBA.StringHolderクラスからjava.lang.Stringクラスのインスタンスを取り出し、このインスタンスに直接設定します。

5.20.4 列挙型

(1)IDLマッピング

IDL言語で列挙型enumを指定した場合、Java言語では、enumクラス／enumHolderクラス／enumHelperクラスになります。
enumクラスには、以下のように要素を識別するためのメンバと、メンバにアクセスするためのメソッドが定義されています。本クラスは、inパラメタおよび復帰値を扱う場合に使用します。enumクラス名は、以下のようにIDLで定義したenum名（enumName）になります。

＜enumクラス＞

```
[package <packageName>]
public final class <enumName>{
    public static final int _<label_1> = <labelValue>;
    public static final <enumName> <label_1> = new <enumName>(_<label_1>);
          :
    public static final int _<label_n> = <label Value>;
    public static final <enumName> <label_n> = new <enumName>(_<label_n>);

    public int value() {...}
    public static <enumName> from_int(int value) {...}
    private <enumName>(int value) { }
}
```

enumクラス中で定義されているメンバの意味を以下の表に示します。

enumクラスのメンバ	意味
value()	要素の値を取り出します。
from_int()	要素の値を指定して、要素名を取り出します。

enumHolderクラスは、以下のようにIDLで定義したenum名+Holderとなります。本クラスは、outやinoutパラメタを受け渡す場合に使用します。

＜enumHolderクラス＞

```
[package <packageName>;]
final public class <enumName>Holder
implements org.omg.CORBA.portable.Streamable  {
    public [<packageName>.]<enumName> value;
    public <enumName>Holder() { }
    public <enumName>Holder([<packageName>.]<enumName> value){...}
    public void _read (org.omg.CORBA.portable.InputStream i){...}
    public void _write(org.omg.CORBA.portable.OutputStream o){...}
```

```
    public org.omg.CORBA.TypeCode _type(){...}
}
```

enumHolderクラス中で定義されているメンバの意味を以下の表に示します。

enumHolderクラスのメンバ	意味
value	要素の値。
デフォルトコンストラクタ	valueメンバへの設定なしで、インスタンスを生成するために使用します。
コンストラクタ	valueメンバにパラメタで指定された要素の値を設定します。
_read()	要素の値を読み込みます。
_write()	要素の値を書き込みます。
_type()	TypeCodeを取り出します。

enumHelperクラスは、以下のようにIDLで定義したenum名+Helperになります。
Helperクラスは、ユーザ定義型のデータのデータストリームへの書き込み、データストリームからのユーザ定義型のデータの取り出し、ユーザ定義型のTypeCode、リポジトリIDの参照などの機能を提供します。

＜enumHelperクラス＞

```
[package <packageName>;]
public class <enumName>Helper {
    public static void
        insert (org.omg.CORBA.Any a,[<packageName>.]<enumName> value){...}
    public static [<packageName>.]<enumName>
        extract (org.omg.CORBA.Any a){...}
    public static org.omg.CORBA.TypeCode
        type(){...}
    public static [<packageName>.]<enumName>
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, [<packageName>.]<enumName> value){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
 }
```

enumHelperクラス中で定義されているメンバの意味を以下の表に示します。

enumHelperクラスのメンバ	意味
insert()	要素の値を挿入します。
extract()	要素の値を取り出します。
type()	TypeCodeを取り出します。

enumHelperクラスのメンバ	意味
id()	リポジトリIDを返します。

IDL言語での定義が以下のように定義されていた場合について、アプリケーションプログラム例を示します。

【IDL言語】

```
module ODsample{
    enum EnumType { a, b, c };

    interface  enumtest{
        EnumType op( in EnumType in_p, out EnumType out_p, inout EnumType io_p );
    };
};
```

これをJava言語にマッピングすると、以下のようになります。

【Java言語】

＜インタフェース＞

```
package ODsample;
public interface enumtestOperations {
    public ODsample.EnumType op(ODsample.EnumType in_p,
                                ODsample.EnumTypeHolder out_p,
                                ODsample.EnumTypeHolder io_p );
}
```

＜enumクラス＞

```
package ODsample;
public final class EnumType
implements org.omg.CORBA.portable.IDLEntity
{
    public static final int _a = (int)0;
    public static final EnumType a = new EnumType(_a);
    public static final int _b = (int)1;
    public static final EnumType b = new EnumType(_b);
    public static final int _c = (int)2;
    public static final EnumType c = new EnumType(_c);
    public static final EnumType MAX_VALUE = new EnumType(Integer.MAX_VALUE);
    public int value(){...}
    public static EnumType from_int(int i){...}
    private EnumType(int value){ }
}
```

＜enumHolderクラス＞

```
package ODsample;
public class EnumTypeHolder
implements org.omg.CORBA.portable.Streamable {
    public ODsample.EnumType value;
    public EnumTypeHolder() { }
    public EnumTypeHolder(ODsample.EnumType value){...}
    public void _read(org.omg.CORBA.portable.InputStream i){...}
```

```
    public void _write(org.omg.CORBA.portable.OutputStream o){...}
    public org.omg.CORBA.TypeCode _type(){...}
}
```

＜enumHelperクラス＞

```
package ODsample;
public class EnumTypeHelper
{
    public static void
       insert(org.omg.CORBA.Any a, ODsample.EnumType value){...}
    public static ODsample.EnumType
       extract(org.omg.CORBA.Any a){...}
    public static String
       id(){...}
    public java.lang.Object
       read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
       write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.EnumType
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.EnumType value){...}
}
```

(2)クライアントアプリケーションでの処理

```
import org.omg.CORBA.*;
import ODsample.*;

public class enumClient {
    public static void main(String args[]) {
        // ORBの前処理
        // オブジェクトリファレンスの獲得
              :
        try{
            // inパラメタ用変数
            EnumType  in   = EnumType.a;
            //outパラメタ用変数
            EnumTypeHolder outHolder = new EnumTypeHolder();
            // inoutパラメタ用変数
            EnumTypeHolder inoutHolder = new EnumTypeHolder( EnumType.b );
            // 復帰値用変数
            EnumType       result;

            // サーバアプリケーションのメソッド呼出し
            result = target.op( in, outHolder, inoutHolder );

            // メソッドの結果表示
            System.out.println( result.value() );
            System.out.println( outHolder.value.value() );
            System.out.println( inoutHolder.value.value() );
        }
```

```
        catch ( java.lang.Exception e ) {
            // エラー処理
                :
        }
    }
}
```

Java言語でinパラメタをサーバアプリケーションに受け渡す場合は、直接enumクラスのインスタンスにデータを設定します。また、out／inoutパラメタとしてサーバアプリケーションにパラメタを受け渡す場合は、new演算子を使用して各データ型のenumHolderクラスのインスタンスを生成します。inoutパラメタには、enumクラスのデータを設定します。
設定には、以下の2つの方法があります。

【デフォルトコンストラクタを使用した方法】

```
EnumTypeHolder a = new EnumTypeHolder();  // デフォルトコンストラクタ
a.value = EnumType.c;                     // データの代入
```

【コンストラクタを使用した方法】

```
EnumTypeHolder b = new EnumTypeHolder(EnumType.c);  // コンストラクタ
```

なお、サーバの処理結果は、out／inoutパラメタ、および復帰値の各インスタンスのvalueに値が設定されます。

(3)サーバアプリケーションでの処理

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servantクラス
// サーバアプリケーションのメソッド
class enumServant extends enumtestPOA {

    public EnumType op( EnumType in,
                        EnumTypeHolder outHolder,
                        EnumTypeHolder inoutHolder ) {
        // 入力パラメタの表示
        System.out.println( in.value() );
        System.out.println( inoutHolder.value.value() );

        // 出力パラメタの設定
        outHolder.value = EnumType.b;
        inoutHolder.value = EnumType.c;

        // 復帰
        return( EnumType.a );
    }
}

public class enumServer {
    public static void main(String args[]) {
        try {
            // ORBの前処理
                :
            // POAオブジェクトの生成
                :
```

```
            // Servantの生成とそのPOAへの登録
                :
            // POAManagerの活性化
                :
        }
        catch (java.lang.Exception e) {
            // エラー処理
                :
        }
    }
}
```

Java言語でinパラメタを参照する場合は、inパラメタのインスタンスのvalueで参照します。inoutパラメタを参照するには、enumHolderクラスからenumクラスのインスタンスを取り出し、このインスタンスのvalueで参照します。また、out／inoutパラメタとしてクライアントアプリケーションにパラメタを返す場合は、enumHolderクラスからenumクラスのインスタンスを取り出し、このインスタンスに直接設定します。

5.20.5 any型

(1)IDLマッピング

IDL言語でany型anyを指定した場合、Java言語ではorg.omg.CORBA.Anyクラスとorg.omg.CORBA.AnyHolderクラスになります。
org.omg.CORBA.Anyクラスは、any型を操作するためのメソッド、およびインスタンスメンバが定義されています。本クラスは、inパラメタおよび復帰値を扱う場合に使用します。

org.omg.CORBA.Anyクラス中で定義されているメンバの意味を以下の表に示します。

org.omg.CORBA.Anyクラスのメンバ	意味
insert_xxx()	xxx型の要素の値を挿入します。
extract_xxx()	xxx型の要素の値を取り出します。
type()	TypeCodeの参照／設定。

org.omg.CORBA.AnyHolderクラスは、以下のようにインスタンスメンバ、デフォルトコンストラクタおよびコンストラクタが定義されています。本クラスは、out／inoutパラメタを受け渡す場合に使用します。

＜ org.omg.CORBA.AnyHolderクラス＞

```
final public class AnyHolder {
      public Any value;
      public AnyHolder() { }
      public AnyHolder(Any initial){...}
 }
```

org.omg.CORBA.AnyHolderクラス中で定義されているメンバの意味を以下の表に示します。

org.omg.CORBA.AnyHolderクラスのメンバ	意味
value	要素の値。
デフォルトコンストラクタ	valueメンバへの設定なしで、インスタンスを生成するために使用します。
コンストラクタ	パラメタで指定された値をvalueメンバに設定します。

IDL言語での定義が以下のように定義されていた場合について、アプリケーションプログラムの例を示します。

【IDL言語】

```
module ODsample{
    struct sample1 {
        long    para1;
        string  para2;
    };
    struct sample2 {
        char    para1;
        float   para2;
    };
    struct sample3 {
        char    para1;
        double  para2;
    };
    interface   anytest{
        any op1(in any any1, out any any2, inout any any3 );
    };
};
```

これをJava言語で記述すると、以下のようになります。

【Java言語】

＜インタフェース＞

```
package ODsample;
public interface anytestOperations {
    public org.omg.CORBA.Any op1(org.omg.CORBA.Any any1,
                                 org.omg.CORBA.AnyHolder any2,
                                 org.omg.CORBA.AnyHolder any3 );
}
```

＜ユーザ定義型クラス＞

```
package ODsample;
final public class sample1
    implements org.omg.CORBA.portable.IDLEntity
{
    public int para1;
    public java.lang.String para2;
    public sample1() {}
    public sample1(int para1, java.lang.String para2){...}
}

package ODsample;
final public class sample2
    implements org.omg.CORBA.portable.IDLEntity
{
    public char para1;
    public float para2;
    public sample2() {}
    public sample2(char para1, float para2) {...}
}

package ODsample;
final public class sample3
```

```
      implements org.omg.CORBA.portable.IDLEntity
{
    public char para1;
    public double para2;
    public sample3() {}
    public sample3(char para1, double para2){...}
}
```

＜ユーザ定義型Holderクラス＞

```
package ODsample;
public class sample1Holder  implements org.omg.CORBA.portable.Streamable {
       public ODsample.sample1 value;
       public sample1Holder(){}
       public sample1Holder(ODsample.sample1 value) {...}
       public void _read(org.omg.CORBA.portable.InputStream i) {...}
       public void _write(org.omg.CORBA.portable.OutputStream o) {...}
       public org.omg.CORBA.TypeCode _type() {...}
}

package ODsample;
public class sample2Holder  implements org.omg.CORBA.portable.Streamable {
       public ODsample.sample2 value;
       public sample2Holder() {}
       public sample2Holder(ODsample.sample2 value) {...}
       public void _read(org.omg.CORBA.portable.InputStream i) {...}
       public void _write(org.omg.CORBA.portable.OutputStream o) {...}
       public org.omg.CORBA.TypeCode _type() {...}
}

package ODsample;
public class sample3Holder  implements org.omg.CORBA.portable.Streamable {
       public ODsample.sample3 value;
       public sample3Holder() {}
       public sample3Holder(ODsample.sample3 value) {...}
       public void _read(org.omg.CORBA.portable.InputStream i) {...}
       public void _write(org.omg.CORBA.portable.OutputStream o) {...}
       public org.omg.CORBA.TypeCode _type() {...}
}
```

＜ユーザ定義型Helperクラス＞

```
package ODsample;
public class sample1Helper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.sample1 value){...}
    public static ODsample.sample1
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id(){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
```

```
        type(){...}
    public static ODsample.sample1
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.sample1 value){...}
}

package ODsample;
public class sample2Helper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.sample2 value){...}
    public static ODsample.sample2
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.sample2
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.sample2 value){...}
}

package ODsample;
public class sample3Helper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.sample3 value){...}
    public static ODsample.sample3
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.sample3
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.sample3 value){...}
}
```

(2)クライアントアプリケーションでの処理

inパラメタのany型のデータ領域は、org.omg.CORBA.ORB.create_any()メソッドで初期化して獲得します。なお、inout／outパラメタの場合は、org.omg.CORBA.AnyHolderクラスを使用してデータ領域を割り当てます。
any型にデータを代入するには、Long、Shortなどの基本データ型の場合は、org.omg.CORBA.Anyクラスの基本データ型のinsertメソッドを使用します。その他のユーザ定義型の場合は、ユーザ定義型Helperクラスのinsertメソッドを使用します。これにより、値とTypeCodeの両方を同時に設定できます。
any型からデータを取り出すには、Long、Shortなどの基本データ型の場合、org.omg.CORBA.Anyクラスの基本データ型のextractメソッドを使用します。その他のユーザ定義型の場合は、ユーザ定義型Helperクラスのextractメソッドで取り出したユーザ定義型のインスタンスを使用します。

```
import org.omg.CORBA.*;
import ODsample.*;

public class anyClient {
    public static void main(String args[]) {
        // ORBの前処理
        // オブジェクトリファレンスの獲得
              :
        try{
            int       int_tmp =  1;
            String    str_tmp = "in_str:data";
            // 構造体ODsample.sample1インスタンスの作成
            sample1   smp1    = new sample1( int_tmp, str_tmp );
            // inパラメタ用変数
            org.omg.CORBA.Any in  = org.omg.CORBA.ORB.init().create_any();
            // org.omg.CORBA.Anyクラスにsample1の値、typeを設定
            sample1Helper.insert( in, smp1 );

            // outパラメタ用変数
            org.omg.CORBA.AnyHolder outHolder = new org.omg.CORBA.AnyHolder();

            char      char_tmp  = 'x';
            float     float_tmp = 0.003f;
            // 構造体ODsample.sample2インスタンスの作成
            sample2   smp2      = new sample2( char_tmp, float_tmp );
            org.omg.CORBA.Any any_tmp  = org.omg.CORBA.ORB.init().create_any();
            // org.omg.CORBA.Anyクラスにsample2の値、typeを設定
            sample2Helper.insert( any_tmp, smp2 );
            // inoutパラメタ用変数
            org.omg.CORBA.AnyHolder inoutHolder = new org.omg.CORBA.AnyHolder( any_tmp );

            // 復帰値用変数
            org.omg.CORBA.Any result;

            // サーバアプリケーションのメソッド呼出し
            result = target.op1( in, outHolder, inoutHolder );

            // メソッドの結果表示
            // org.omg.CORBA.Anyクラスからsample3の値を取り出す
            System.out.println( sample3Helper.extract(result).para1 );
            System.out.println( sample3Helper.extract(result).para2 );
            // org.omg.CORBA.AnyHolderクラスからsample1の値を取り出す
            System.out.println( sample1Helper.extract(outHolder.value).para1 );
            System.out.println( sample1Helper.extract(outHolder.value).para2 );
            // org.omg.CORBA.AnyHolderクラスからsample2の値を取り出す
            System.out.println( sample2Helper.extract(inoutHolder.value).para1 );
            System.out.println( sample2Helper.extract(inoutHolder.value).para2 );
        }
        catch (java.lang.Exception e) {
            // エラー処理
```

```
                :
        }
    }
}
```

(3)サーバアプリケーションでの処理

inout／outパラメタを参照するには、Long、Shortなどの基本データ型の場合は、org.omg.CORBA.Anyクラスの基本データ型のextractメソッドを使用します。その他のユーザ定義型の場合は、ユーザ定義型Helperクラスのextractメソッドで取り出したユーザ定義型のインスタンスを使用します。
inout／outパラメタにデータを代入するには、Long、Shortなどの基本データ型の場合は、org.omg.CORBA.Anyクラスの基本データ型のinsertメソッドを使用します。その他のユーザ定義型の場合は、inout／outパラメタのHolderクラスのvalueに、ユーザ定義型Helperクラスのinsertメソッドを使用して設定したany型クラスを設定します。

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servantクラス
// サーバアプリケーションのメソッド
class anyServant extends anytestPOA {

    public org.omg.CORBA.Any op1( org.omg.CORBA.Any in,
                                  org.omg.CORBA.AnyHolder outHolder,
                                  org.omg.CORBA.AnyHolder inoutHolder ) {
        // 入力パラメタの表示
        // org.omg.CORBA.Anyクラスからsample1の値を取り出す
        System.out.println( sample1Helper.extract(in).para1 );
        System.out.println( sample1Helper.extract(in).para2 );

        // org.omg.CORBA.AnyHolderクラスからsample2の値を取り出す
        System.out.println( sample2Helper.extract(inoutHolder.value).para1 );
        System.out.println( sample2Helper.extract(inoutHolder.value).para2 );

        // 出力パラメタの設定
        int     int_tmp =  12345;
        String  str_tmp = "out_str:data";
        // 構造体ODsample.sample1インスタンスの作成
        sample1 smp1    = new sample1( int_tmp, str_tmp );
        org.omg.CORBA.Any any_tmp1  = org.omg.CORBA.ORB.init().create_any();
        // org.omg.CORBA.Anyクラスにsample1の値、typeを設定
        sample1Helper.insert( any_tmp1, smp1 );
        // outパラメタの設定
        outHolder.value = any_tmp1;

        // inoutパラメタを取り出し、構造体ODsample.sample2インスタンスの作成
        sample2 smp2 = sample2Helper.extract(inoutHolder.value);
        smp2.para1   = 'z';                    // inoutパラメタ値を変更
        smp2.para2   = smp2.para2 + 0.007f;    // inoutパラメタ値を変更
        org.omg.CORBA.Any any_tmp2  = org.omg.CORBA.ORB.init().create_any();
        // org.omg.CORBA.Anyクラスにsample2の値、typeを設定
        sample2Helper.insert( any_tmp2, smp2 );
        // inoutパラメタ値の設定
        inoutHolder.value = any_tmp2;

        // 復帰の設定
        char      char_tmp   = 'R';
        double    double_tmp = 9E38;
        // 構造体ODsample.sample3インスタンスの作成
        sample3   smp3      = new sample3( char_tmp, double_tmp );
```

```
        org.omg.CORBA.Any result  = org.omg.CORBA.ORB.init().create_any();
        // org.omg.CORBA.Anyクラスにsample3の値、typeを設定
        sample3Helper.insert( result, smp3 );

        // 復帰
        return( result );
    }
}

public class anyServer {
    public static void main( String args[] ) {
        try {
            // ORBの前処理
                :
            // POAオブジェクトの生成
                :
            // Servantの生成とそのPOAへの登録
                :
            // POAManagerの活性化
                :
        }
        catch ( java.lang.Exception e ) {
            // エラー処理
                :
        }
    }
}
```

5.20.6 シーケンス型

(1)IDLマッピング

IDL言語でシーケンス型sequenceを指定した場合、Java言語では、sequenceの要素の型に対応するデータ型の配列(データ型[ ])、sequenceHolderクラス、およびsequenceHelperクラスになります。
データ型の配列は、inパラメタおよび復帰値を扱う場合に使用します。
sequenceHolderクラス名は、以下のようにIDLで定義したシーケンス型名+Holderになります。本クラスは、out／inoutパラメタを扱う場合に使用します。

＜sequenceHolderクラス＞

```
[package <packageName>;]
final public class <sequence_class>Holder
       implements org.omg.CORBA.portable.Streamable  {
    public <sequence_element_type>[] value;
    public <sequence_class>Holder() {};
    public <sequence_class>Holder(<sequence_element_type>[] value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i){...}
    public void _write(org.omg.CORBA.portable.OutputStream o){...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

sequenceHolderクラスで定義されているメンバの意味を以下の表に示します。

sequenceHolderクラスのメンバ	意味
デフォルトコンストラクタ	valueメンバへの設定なしで、インスタンスを生成するために使用します。
コンストラクタ	valueメンバにパラメタで指定された値(sequence型データ)を設定します。

sequenceHolderクラスのメンバ	意味
_read()	要素の値を読み込みます。
_write()	要素の値を書き込みます。
_type()	TypeCodeを取り出します。

sequenceHeperクラス名は、以下のようにIDLで定義したシーケンス型名+Helperになります。
Helperクラスはユーザ定義型のデータのデータストリームへの書き込み、データストリームからのユーザ定義型のデータの取り出し、ユーザ定義型のTypeCode、リポジトリIDの参照などの機能を提供します。

＜sequenceHelperクラス＞

```
[package <packageName>;]
public class <sequence_class>Helper {
    public static void
        insert(org.omg.CORBA.Any a, <sequence_element_type>[] value){...}
    public static <sequence_element_type>[]
        extract (org.omg.CORBA.Any a){...}
    public static org.omg.CORBA.TypeCode
        type(){...}
    public static String
        id(){...}
    public static int[]
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, int[] value){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
}
```

seqenceHelperクラス中で定義されているメンバの意味を以下の表に示します。

sequenceHelperクラスのメソッド	意味
insert()	要素の値を挿入します。
extract()	要素の値を取り出します。
type()	TypeCodeを取り出します。
id()	リポジトリIDを返します。

IDL言語での定義が以下のように定義されていた場合について、アプリケーションプログラムの例を示します。

【IDL言語】

```
module ODsample{
    interface   seqtest{
        typedef sequence<long>  sampleseq;
```

```
        sampleseq op1(in sampleseq seq1,
                      out sampleseq seq2,
                      inout sampleseq seq3 );
    };
};
```

これをJava言語で記述すると、以下のようになります。

【Java言語】

＜インタフェースクラス＞

```
package ODsample;
public interface seqtestOperations {
    public int[] op1(int[] seq1,
                     ODsample.seqtestPackage.sampleseqHolder seq2,
                     ODsample.seqtestPackage.sampleseqHolder seq3 );
}
```

＜sequenceHolderクラス＞

```
package ODsample.seqtestPackage;
public class sampleseqHolder  implements org.omg.CORBA.portable.Streamable {
    public int[] value;
    public sampleseqHolder() {}
    public sampleseqHolder(int[] value) {...}
    public void _read ( org.omg.CORBA.portable.InputStream _i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream _o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

＜sequenceHelperクラス＞

```
package ODsample.seqtestPackage;
public class sampleseqHelper
{
    public static void
        insert(org.omg.CORBA.Any a, int[] value){...}
    public static int[]
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    public static org.omg.CORBA.TypeCode
        type(){...}
    public static int[]
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, int[] value){...}
}
```

(2)クライアントアプリケーションでの処理

inパラメタのsequence型のデータ領域は、newを使用して獲得します。inout／outパラメタの場合は、sequenceHolderクラスを使用してデータ領域を割り当てます。inoutパラメタのsequenceにデータを割り当てるには、コンストラクタのパラメタで必要な個数分指定することにより、sequenceのインスタンスデータvalueに領域が割り当てられます。そのvalueにデータを設定します。sequenceデータの参照は、インスタンスデータvalueを参照します。

```
import org.omg.CORBA.*;
import ODsample.*;

public class seqClient {
    public static void main(String args[]) {
        // ORBの前処理
        // オブジェクトリファレンスの獲得
              :
        try{
            int i;

            // inパラメタ用変数
            int in[] = new int[10];
            for( i = 0; i < in.length; i++ )
                in[i] = i;

            // outパラメタ用変数
            ODsample.seqtestPackage.sampleseqHolder outHolder =
                        new ODsample.seqtestPackage.sampleseqHolder();
            // inoutパラメタ用変数
            ODsample.seqtestPackage.sampleseqHolder inoutHolder =
                        new ODsample.seqtestPackage.sampleseqHolder( new int[20] );
            for( i = 0; i < inoutHolder.value.length; i++ )
                inoutHolder.value[i] = i * 2;

            // 復帰値用変数
            int  result[];

            // サーバアプリケーションのメソッド呼出し
            result = target.op1( in, outHolder, inoutHolder );

            // メソッドの結果表示
            // 復帰値の表示
            for( i = 0; i < result.length; i++ )
                System.out.println( "result[" + i + "]=" + result[i] );

            // outパラメタの表示
            for( i = 0; i < outHolder.value.length; i++ )
                System.out.println( "outHolder.value[" + i + "]=" + outHolder.value[i] );

            // inoutパラメタの表示
            for( i = 0; i < inoutHolder.value.length; i++ )
                System.out.println( "inoutHolder.value[" + i + "]=" + inoutHolder.value[i] );
        }
        catch (java.lang.Exception e) {
            // エラー処理
                :
        }
    }
}
```

(3)サーバアプリケーションでの処理

inout／outパラメタにデータを代入するには、inout／outのHolderクラスのvalueに設定します。

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servantクラス
// サーバアプリケーションのメソッド
class seqServant extends seqtestPOA {

    public int[] op1( int[] in,
                      ODsample.seqtestPackage.sampleseqHolder outHolder,
                      ODsample.seqtestPackage.sampleseqHolder inoutHolder ) {
        int i;

        // 入力パラメタの表示
        // inパラメタの表示
        for( i = 0; i < in.length; i++ )
            System.out.println( "in[" + i + "]=" + in[i] );

        // inoutパラメタの表示
        for( i = 0; i < inoutHolder.value.length; i++ )
            System.out.println( "inoutHolder.value[" + i + "]=" + inoutHolder.value[i] );

        // 出力パラメタの設定
        // outパラメタの設定
        outHolder.value = new int[5];
        for( i = 0; i < outHolder.value.length; i++ )
            outHolder.value[i] = in[i] * 100;

        // inoutパラメタの設定
        for( i = 0; i < inoutHolder.value.length; i++ )
            inoutHolder.value[i] = inoutHolder.value[i] * 2;

        // 復帰の設定
        int result[] = new int[5];
        for( i = 0; i < result.length; i++ )
            result[i] = in[i] + 100;

        // 復帰
        return( result );
    }
}

public class seqServer {
    public static void main(String args[]) {
        try {
             // ORBの前処理
                :
             // POAオブジェクトの生成
                :
             // Servantの生成とそのPOAへの登録
                :
             // POAManagerの活性化
                :
        }
        catch (java.lang.Exception e) {
            // エラー処理
                :
        }
```

```
    }
}
```

5.20.7 構造体

(1)IDLマッピング

IDL言語で構造体structを指定した場合、Java言語ではstructクラス、structHolderクラス、およびstructHelperクラスになります。
structクラス名は、IDLで定義した構造体名になります。本クラスは、inパラメタおよび復帰値を扱う場合に使用します。
structクラスで定義されているメンバの意味を以下の表に示します。

structクラスのメンバ	意味
デフォルトコンストラクタ	何もしません。
コンストラクタ	指定されたパラメタをインスタンスメンバに設定します。

structHolderクラス名は、以下のようにIDLで定義した構造体名+Holderになります。本クラスは、out／inoutパラメタを扱う場合に使用します。

＜structHolderクラス＞

```
[package <packageName>;]
final public class <struct_class>Holder implements
                        org.omg.CORBA.portable.Streamable {
    public [<packageName>.]<struct_class>  value;
    public <struct_class>Holder() {}
    public <struct_class>Holder([<packageName>.]<struct_class> value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o){...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

structHolderクラスで定義されているメンバの意味を以下の表に示します。

structHolderクラスのメンバ	意味
デフォルトコンストラクタ	valueメンバへの設定なしで、インスタンスを生成するために使用します。
コンストラクタ	valueメンバに指定されたパラメタを設定します。
_read()	要素の値を読み込みます。
_write()	要素の値を書き込みます。
_type()	TypeCodeを取り出します。

structHelperクラス名は、以下のようにIDLで定義した構造体名+Helperになります。
Helperクラスは、ユーザ定義型のデータのデータストリームへの書き込み、データストリームからのユーザ定義型のデータの取り出し、ユーザ定義型のTypeCode, リポジトリIDの参照などの機能を提供します。

＜structHelperクラス＞

```
[package <packageName>;]
public class <struct_class>Helper {
```

```
    public static void
        insert(org.omg.CORBA.Any a, <packageName>.<struct_class>  value){...}
    public static <packageName>.<struct_class>
        extract(org.omg.CORBA.Any a){...}
    public static org.omg.CORBA.TypeCode
        type(){...}
    public static String
        id(){...}
    public static <packageName>.<struct_class>
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream,
              <packageName>.<struct_class>  value){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
}
```

structHelperクラス中で定義されているメンバの意味を以下の表に示します。

structHelperクラスのメンバ	意味
insert()	要素の値を挿入します。
extract()	要素の値を取り出します。
type()	TypeCodeを取り出します。
id()	リポジトリIDを返します。

IDL言語での定義が以下のように定義されていた場合について、アプリケーションプログラムの例を示します。

【IDL言語】

```
module ODsample{
    struct samplefix {      // 構造体(固定長)
        long    para1;
        long    para2;
    };
    struct samplevar {      // 構造体(可変長)
        long    para1;
        string  para2;
    };
    interface   structtest{
        samplefix   op2(
            in samplefix str1,
            out samplefix str2,
            inout samplefix str3 );
        samplevar   op1(
            in samplevar str1,
            out samplevar str2,
            inout samplevar str3 );
    };
};
```

これをJava言語で記述すると、以下のようになります。

【Java言語】

＜インタフェース＞

```
package ODsample;
public interface structtestOperations {
    public ODsample.samplefix op2(ODsample.samplefix str1,
                                   ODsample.samplefixHolder str2,
                                   ODsample.samplefixHolder str3 );
    public ODsample.samplevar op1(ODsample.samplevar str1,
                                   ODsample.samplevarHolder str2,
                                   ODsample.samplevarHolder str3 );
}
```

＜structクラス＞

```
package ODsample;
final public class samplefix
    implements org.omg.CORBA.portable.IDLEntity
{
    public int para1;
    public int para2;
    public samplefix() {}
    public samplefix(int para1, int para2) {...}
}

package ODsample;
final public class samplevar
    implements org.omg.CORBA.portable.IDLEntity
{
    public int para1;
    public java.lang.String para2;
    public samplevar() {}
    public samplevar( int para1, java.lang.String para2) {...}
}
```

＜structHolderクラス＞

```
package ODsample;
public class samplefixHolder
      implements org.omg.CORBA.portable.Streamable {
    public ODsample.samplefix value;
    public samplefixHolder()  {}
    public samplefixHolder(ODsample.samplefix value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

package ODsample;
public class samplevarHolder
        implements org.omg.CORBA.portable.Streamable {
    public ODsample.samplevar value;
    public samplevarHolder()  {}
    public samplevarHolder(ODsample.samplevar value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
```

```
    public void _write(org.omg.CORBA.portable.OutputStream o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

＜structHelperクラス＞

```
package ODsample;
public class samplefixHelper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.samplefix value){...}
    public static ODsample.samplefix
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
       get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
       type(){...}
    public static ODsample.samplefix
       read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
       write(org.omg.CORBA.portable.OutputStream ostream, ODsample.samplefix value){...}
}

package ODsample;
public class samplevarHelper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.samplevar value){...}
    public static ODsample.samplevar
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.samplevar
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.samplevar value){...}
}
```

(2)クライアントアプリケーションでの処理

inパラメタのstruct型のデータ領域は、newを使用して獲得します。inout／outパラメタの場合は、structHolderクラスを使用してデータ領域を割り当てます。
クライアントアプリケーションは、in／inoutパラメタの入力値として、structクラスのインスタンスのメンバに値を設定します。inoutパラメタでは、作成したstructクラスのインスタンスをstructHolderクラスのメンバに設定します。
inout／outパラメタの出力値を参照するには、structHolderクラスのインスタンスのvalueを参照します。

```
import org.omg.CORBA.*;
import ODsample.*;

public class structClient {
    public static void main(String args[]) {
        // ORBの前処理
        // オブジェクトリファレンスの獲得
             :
        try {

            // 固定長構造体

            // inパラメタ用変数
            int        in_fix_para1 = 3;
            int        in_fix_para2 = 10;
            samplefix  in_fix       = new samplefix( in_fix_para1, in_fix_para2 );

            // outパラメタ用変数
            samplefixHolder outHolder_fix = new samplefixHolder();

            // inoutパラメタ用変数
            int        inout_fix_para1 = 20;
            int        inout_fix_para2 = 50;
            samplefix  inout_fix_tmp   = new samplefix(inout_fix_para1, inout_fix_para2);
            samplefixHolder inoutHolder_fix = new samplefixHolder( inout_fix_tmp );

            // 復帰値用変数
            samplefix  ret_fix;

            // サーバアプリケーションのメソッド呼出し
            ret_fix = target.op2( in_fix, outHolder_fix, inoutHolder_fix );

            // メソッドの結果表示
            // 復帰値の表示
            System.out.println( ret_fix.para1 );
            System.out.println( ret_fix.para2 );

            // outパラメタの表示
            System.out.println( outHolder_fix.value.para1 );
            System.out.println( outHolder_fix.value.para2 );

            // inoutパラメタの表示
            System.out.println( inoutHolder_fix.value.para1 );
            System.out.println( inoutHolder_fix.value.para2 );

            // 可変長構造体

            // inパラメタ用変数
            int        in_var_para1 = 3;
            String     in_var_para2 = "test";
            samplevar  in_var       = new samplevar( in_var_para1, in_var_para2 );

            // outパラメタ用変数
            samplevarHolder outHolder_var = new samplevarHolder();
```

```
            // inoutパラメタ用変数
            int        inout_var_para1 = 20;
            String     inout_var_para2 = "dummy";
            samplevar  inout_var_tmp   = new samplevar(inout_var_para1, inout_var_para2);

            samplevarHolder inoutHolder_var = new samplevarHolder( inout_var_tmp );

            // 復帰値用変数
            samplevar  ret_var;

            // サーバアプリケーションのメソッド呼出し
            ret_var = target.op1( in_var, outHolder_var, inoutHolder_var );

            // メソッドの結果表示
            // 復帰値の表示
            System.out.println( ret_var.para1 );
            System.out.println( ret_var.para2 );

            // outパラメタの表示
            System.out.println( outHolder_var.value.para1 );
            System.out.println( outHolder_var.value.para2 );

           // inoutパラメタの表示
           System.out.println( inoutHolder_var.value.para1 );
           System.out.println( inoutHolder_var.value.para2 );
        }
        catch (java.lang.Exception e) {
            // エラー処理
                :
        }
    }
}
```

(3)サーバアプリケーションでの処理

inout／outパラメタにデータを代入するには、inout／outのHolderクラスのvalueに設定します。

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servantクラス
// サーバアプリケーションのメソッド
class structServant extends structtestPOA {

    // 固定長構造体
    public samplefix op2( samplefix in_fix,
                          samplefixHolder outHolder_fix,
                          samplefixHolder inoutHolder_fix ) {
        // 入力パラメタの表示
        // inパラメタの表示
        System.out.println( in_fix.para1 );
        System.out.println( in_fix.para2 );

        // inoutパラメタの表示
        System.out.println( inoutHolder_fix.value.para1 );
        System.out.println( inoutHolder_fix.value.para2 );

        // 出力パラメタの設定
```

```
        // outパラメタの設定
        int        out_fix_para1 = 123;
        int        out_fix_para2 = 456;
        samplefix  out_fix_tmp   = new samplefix( out_fix_para1, out_fix_para2 );
        outHolder_fix.value      = out_fix_tmp;

        // inoutパラメタの設定
        inoutHolder_fix.value.para1 = inoutHolder_fix.value.para1 * 10;
        inoutHolder_fix.value.para2 = inoutHolder_fix.value.para2 * 10;

        // 復帰の設定
        int        ret_fix_para1 = 333;
        int        ret_fix_para2 = 777;
        samplefix  ret_fix       = new samplefix( ret_fix_para1, ret_fix_para2 );

        // 復帰
        return( ret_fix );
    }

    // 可変長構造体
    public samplevar op1( samplevar in_var,
                          samplevarHolder outHolder_var,
                          samplevarHolder inoutHolder_var ) {
        // 入力パラメタの表示
        // inパラメタの表示
        System.out.println( in_var.para1 );
        System.out.println( in_var.para2 );

        // inoutパラメタの表示
        System.out.println( inoutHolder_var.value.para1 );
        System.out.println( inoutHolder_var.value.para2 );

        // 出力パラメタの設定
        // outパラメタの設定
        int        out_var_para1 = 11111;
        String     out_var_para2 = "out-str";
        samplevar  out_var_tmp   = new samplevar( out_var_para1, out_var_para2 );
        outHolder_var.value      = out_var_tmp;

        // inoutパラメタの設定
        inoutHolder_var.value.para1 = inoutHolder_var.value.para1 + 2000;
        inoutHolder_var.value.para2 = inoutHolder_var.value.para2 + "-inout-str";

        // 復帰の設定
        int        ret_var_para1 = 0;
        String     ret_var_para2 = "return-str";
        samplevar  ret_var       = new samplevar( ret_var_para1, ret_var_para2 );

        // 復帰
        return( ret_var );
    }
}

public class structServer {
    public static void main(String args[]) {
        try {
            // ORBの前処理
                :
            // POAオブジェクトの生成
                :
            // Servantの生成とそのPOAへの登録
                :
            // POAManagerの活性化
```

```
                :
        }
        catch (java.lang.Exception e) {
            // エラー処理
                :
        }
    }
}
```

5.20.8 共用体

(1)IDLマッピング

IDL言語で共用体unionを指定した場合、Java言語では、unionクラス、unionHolderクラス、およびunionHelperクラスになります。
unionクラス名は、IDLで定義した共用体名になります。本クラスは、inパラメタおよび復帰値を扱う場合に使用します。
unionクラスで定義されているメンバの意味を以下の表に示します。

unionクラスのメンバ	意味
デフォルトコンストラクタ	何もしません。
弁別子へのアクセスメソッド	弁別子へのアクセスメソッド名は、discriminatorになります。
メンバへの設定メソッド	メンバへの設定メソッド名は、メンバ名になります。
メンバの参照メソッド	メンバの参照メソッド名は、メンバ名になります。

unionHolderクラス名は、IDLで定義した共用体名+Holderになります。本クラスは、out／inoutパラメタを扱う場合に使用します。

＜unionHolderクラス＞

```
[package <packageName>;]
final public class <union_class>Holder implements
                    org.omg.CORBA.portable.Streamable {
    public [<packageName>.]< union _class>  value;
    public < union _class>Holder() {}
    public < union _class>Holder([<packageName>.]< union _class> value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o){...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

unionHolderクラスで定義されているメンバの意味を以下の表に示します。

unionHolderクラスのメンバ	意味
デフォルトコンストラクタ	valueメンバへの設定なしで、インスタンスを生成するために使用します。
コンストラクタ	value メンバに指定されたパラメタを設定します。
_read()	要素の値を読み込みます。
_write()	要素の値を書き込みます。
_type()	TypeCodeを取り出します。

unionHelperクラス名は、以下のようにIDLで定義した共用体名+Helperになります。
Helperクラスは、ユーザ定義型のデータのデータストリームへの書き込み、データストリームからのユーザ定義型のデータの取り出し、ユーザ定義型のTypeCode、リポジトリIDの参照などの機能を提供します。

＜unionHelperクラス＞

```
[package <packageName>;]
public class <union_class>Helper {
    public static void
        insert(org.omg.CORBA.Any a, <packageName>.<union_class>  value){...}
    public static <packageName>.<union_class>
        extract(org.omg.CORBA.Any a){...}
    public static org.omg.CORBA.TypeCode
        type(){...}
    public static String
        id(){...}
    public static <packageName>.<union_class>
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, <packageName>.<union_class>  value){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
  }
```

unionHelperクラス中で定義されているメンバの意味を以下の表に示します。

unionHelperクラスのメンバ	意味
insert()	要素の値を挿入します。
extract()	要素の値を取り出します。
type()	TypeCodeを取り出します。
id()	リポジトリIDを返します。

IDL言語での定義が以下のように定義されていた場合について、アプリケーションプログラムの例を示します。

【IDL言語】

```
module ODsample{
    union samplefix switch(long){       // 共用体(固定長)
        case 1: long    para1;
        case 2: long    para2;
    };
    union samplevar switch(long){       // 共用体(可変長)
        case 1: long    para1;
        case 2: string  para2;
    };
    interface uniontest{
        samplefix  op2(
            in samplefix uni1,
```

```
            out samplefix uni2,
            inout samplefix uni3
        );
        samplevar  op1(
            in samplevar uni1,
            out samplevar uni2,
            inout samplevar uni3
        );
    };
};
```

これをJava言語で記述すると、以下のようになります。

【Java言語】

＜インタフェース＞

```
package ODsample;
public interface uniontestOperations {
    public ODsample.samplefix op2(ODsample.samplefix uni1,
                                  ODsample.samplefixHolder uni2,
                                  ODsample.samplefixHolder uni3 );
    public ODsample.samplevar op1(ODsample.samplevar uni1,
                                  ODsample.samplevarHolder uni2,
                                  ODsample.samplevarHolder uni3 );
}
```

＜unionクラス＞

```
package ODsample;
final public class samplefix
    implements org.omg.CORBA.portable.IDLEntity
{
    private int _discriminator;
    private java.lang.Object value;
    public samplefix() {}
    public int discriminator(){...}
    public int para1()
        throws org.omg.CORBA.BAD_OPERATION{...}
    public void para1( int value ){...}
    public int para2()
        throws org.omg.CORBA.BAD_OPERATION{...}
    public void para2( int value ){...}
}

package ODsample;
final public class samplevar
    implements org.omg.CORBA.portable.IDLEntity
{
    private int _discriminator;
    private java.lang.Object value;
    public samplevar() {}
    public int discriminator(){...}
    public int para1()
        throws org.omg.CORBA.BAD_OPERATION{...}
    public void para1( int value ){...}
    public java.lang.String para2()
        throws org.omg.CORBA.BAD_OPERATION{...}
```

```
    public void para2( java.lang.String value ){...}
}
```

＜unionHolderクラス＞

```
package ODsample;
public class samplefixHolder     implements org.omg.CORBA.portable.Streamable {
    public ODsample.samplefix value;
    public samplefixHolder()  {}
    public samplefixHolder(ODsample.samplefix value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

package ODsample;
public class samplevarHolder  implements org.omg.CORBA.portable.Streamable {
    public ODsample.samplevar value;
    public samplevarHolder()  {}
    public samplevarHolder(ODsample.samplevar value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

＜unionHelperクラス＞

```
package ODsample;
public class samplefixHelper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.samplefix value){...}
    public static ODsample.samplefix
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.samplefix
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.samplefix value){...}
}

package ODsample;
public class samplevarHelper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.samplevar value){...}
    public static ODsample.samplevar
        extract(org.omg.CORBA.Any a){...}
    public static String
```

```
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.samplevar
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.samplevar value){...}
}
```

(2)クライアントアプリケーションでの処理

inパラメタのunion型のデータ領域は、newを使用して獲得します。inout／outパラメタの場合は、unionHolderクラスを使用してデータ領域を割り当てます。unionデータの参照／設定は、メンバ参照関数、弁別子アクセス関数を使用します。

```
import org.omg.CORBA.*;
import ODsample.*;

public class unionClient {
    public static void main(String args[]) {
        // ORBの前処理
        // オブジェクトリファレンスの獲得
              :
        try{
            // 固定長構造体

            // inパラメタ用変数
            samplefix  in_fix   = new samplefix();
            in_fix.para1( 100 ); // 共用型変数をpara1の型で初期化

            // outパラメタ用変数
            samplefixHolder outHolder_fix = new samplefixHolder();

            // inoutパラメタ用変数
            samplefix  inout_fix_tmp = new samplefix();
            inout_fix_tmp.para1( 200 ); // 共用型変数をpara2の型で初期化
            samplefixHolder inoutHolder_fix =  new samplefixHolder(inout_fix_tmp);

            // 復帰値用変数
            samplefix ret_fix;

            // サーバアプリケーションのメソッド呼出し
            ret_fix = target.op2( in_fix, outHolder_fix, inoutHolder_fix );

            // メソッドの結果表示
            // 復帰値の表示
            switch( ret_fix.discriminator() ) {
                case 1:
                    System.out.println( ret_fix.para1() );
                    break;
                case 2:
                    System.out.println( ret_fix.para2() );
```

```
        break;
}

// outパラメタの表示
switch( outHolder_fix.value.discriminator() ) {
    case 1:
        System.out.println( outHolder_fix.value.para1() );
        break;
    case 2:
        System.out.println( outHolder_fix.value.para2() );
        break;
}

// inoutパラメタの表示
switch( inoutHolder_fix.value.discriminator() ) {
    case 1:
        System.out.println( inoutHolder_fix.value.para1() );
        break;
    case 2:
        System.out.println( inoutHolder_fix.value.para2() );
        break;
}

// 可変長構造体

// inパラメタ用変数
samplevar  in_var   = new samplevar();
in_var.para1( 500 ); // 共用型変数をpara1の型で初期化

// outパラメタ用変数
samplevarHolder outHolder_var = new samplevarHolder();

// inoutパラメタ用変数
samplevar  inout_var_tmp = new samplevar();
inout_var_tmp.para1( 200 ); // 共用型変数をpara2の型で初期化
samplevarHolder inoutHolder_var =  new samplevarHolder(inout_var_tmp);

// 復帰値用変数
samplevar ret_var;

// サーバアプリケーションのメソッド呼出し
ret_var = target.op1( in_var, outHolder_var, inoutHolder_var );

// メソッドの結果表示
// 復帰値の表示
switch( ret_var.discriminator() ) {
    case 1:
        System.out.println( ret_var.para1() );
        break;
    case 2:
        System.out.println( ret_var.para2() );
        break;
}

// outパラメタの表示
switch( outHolder_var.value.discriminator() ) {
    case 1:
        System.out.println( outHolder_var.value.para1() );
        break;
    case 2:
        System.out.println( outHolder_var.value.para2() );
        break;
}
```

```
            // inoutパラメタの表示
            switch( inoutHolder_var.value.discriminator() ) {
                case 1:
                    System.out.println( inoutHolder_var.value.para1() );
                    break;
                case 2:
                    System.out.println( inoutHolder_var.value.para2() );
                    break;
            }

        }
        catch (java.lang.Exception e) {
            // エラー処理
                :
        }
    }
}
```

(3)サーバアプリケーションでの処理

inout／outパラメタにデータを代入するには、inout／outのHolderクラスのvalueに設定します。

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servantクラス
// サーバアプリケーションのメソッド
class unionServant extends uniontestPOA {

    public samplefix op2( samplefix in_fix,
                          samplefixHolder outHolder_fix,
                          samplefixHolder inoutHolder_fix ) {
        // 入力パラメタの表示
        // inパラメタの表示
        switch( in_fix.discriminator() )
        {
        case 1:
            System.out.println( in_fix.para1() );
            break;
        case 2:
            System.out.println( in_fix.para2() );
            break;
        }

        // inoutパラメタの表示と設定
        switch( inoutHolder_fix.value.discriminator() )
        {
        case 1:
            System.out.println( inoutHolder_fix.value.para1() );

            // 共用型変数をpara1の型からpara2の型に変更
            inoutHolder_fix.value.para2( inoutHolder_fix.value.para1() * 100 );
            break;
        case 2:
            System.out.println( inoutHolder_fix.value.para2() );

            // 共用型変数をpara2の型からpara1の型に変更
            inoutHolder_fix.value.para1( inoutHolder_fix.value.para2() * 100 );
```

```
            break;
        }

        // 出力パラメタの設定
        // outパラメタの設定
        samplefix  out_fix_tmp = new samplefix();
        out_fix_tmp.para2( 999 ); // 共用型変数をpara2の型で初期化
        outHolder_fix.value = out_fix_tmp;

        // 復帰の設定
        samplefix  ret_fix   = new samplefix();
        ret_fix.para1( 0 );   // 共用型変数をpara1の型で初期化

        // 復帰
        return( ret_fix );
    }

    public samplevar op1( samplevar in_var,
                          samplevarHolder outHolder_var,
                          samplevarHolder inoutHolder_var ) {
        // 入力パラメタの表示
        // inパラメタの表示
        switch( in_var.discriminator() )
        {
        case 1:
            System.out.println( in_var.para1() );
            break;
        case 2:
            System.out.println( in_var.para2() );
            break;
        }

        // inoutパラメタの表示と設定
        switch( inoutHolder_var.value.discriminator() )
        {
        case 1:
            System.out.println( inoutHolder_var.value.para1() );

            // 共用型変数をpara1の型からpara2の型に変更
            inoutHolder_var.value.para2( "inout-str-add-ret" );
            break;
        case 2:
            System.out.println( inoutHolder_var.value.para2() );

            // 共用型変数をpara2の型からpara1の型に変更
            inoutHolder_var.value.para1( 5000 );
            break;
        }

        // 出力パラメタの設定
        // outパラメタの設定
        samplevar  out_var_tmp = new samplevar();
        out_var_tmp.para2( "out-str" ); // 共用型変数をpara2の型で初期化
        outHolder_var.value = out_var_tmp;

        // 復帰の設定
        samplevar  ret_var   = new samplevar();
        ret_var.para2( "ret-str" );   // 共用型変数をpara1の型で初期化

        // 復帰
        return( ret_var );
    }
}
```

```
public class unionServer {
    public static void main(String args[]) {
        try {
             // ORBの前処理
                :
             // POAオブジェクトの生成
                :
             // Servantの生成とそのPOAへの登録
                :
             // POAManagerの活性化
                :
        }
        catch ( java.lang.Exception e ) {
           // エラー処理
                :
        }
    }
}
```

5.20.9 配列

(1)IDLマッピング

IDL言語で配列を定義した場合、Java言語では、要素の型に対応するデータ型の配列、および配列名Holderクラスと配列名Helperクラスになります。
データ型の配列は、inパラメタおよび復帰値を扱う場合に使用します。
配列名Holderクラス名は、IDLで定義した配列名+Holderになります。本クラスは、out／inoutパラメタを扱う場合に使用します。

＜配列名Holderクラス＞

```
[package <packageName>;]
final public class <array_class>Holder
      implements org.omg.CORBA.portable.Streamable  {
    public < array _element_type>[] value;
    public < array _class>Holder() {};
    public < array _class>Holder(<array _element_type>[] value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i){...}
    public void _write(org.omg.CORBA.portable.OutputStream o){...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

配列名Holderクラスで定義されているメンバの意味を以下の表に示します。

配列名Holderクラスのメンバ	意味
デフォルトコンストラクタ	valueメンバへの設定なしで、インスタンスを生成するために使用します。
コンストラクタ	valueメンバに指定されたパラメタを設定します。
_read()	要素の値を読み込みます。
_write()	要素の値を書き込みます。
_type()	TypeCodeを取り出します。

配列名Helperクラス名は、以下のようにIDLで定義した配列名+Helperになります。
Helperクラスは、ユーザ定義型のデータのデータストリームへの書き込み、データストリームからのユーザ定義型のデータの取り出し、ユーザ定義型のTypeCode、リポジトリIDの参照などの機能を提供します。

＜配列名Helperクラス＞

```
[package <packageName>;]
public class <array_class>Helper
{
    public static void
        insert(org.omg.CORBA.Any a, <array _element_type>[] value){...}
    public static <array _element_type>[]
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static <array _element_type>[]
        read( org.omg.CORBA.portable.InputStream istream ){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream,
              <array _element_type>[] value)
              throws  org.omg.CORBA.MARSHAL{...}
}
```

配列名Helperクラス中で定義されているメンバの意味を以下の表に示します。

配列名Helperクラスのメンバ	意味
insert()	要素の値を挿入します。
extract()	要素の値を取り出します。
type()	TypeCodeを取り出します。
id()	リポジトリIDを返します。

IDL言語での定義が以下のように定義されていた場合について、アプリケーションプログラムの例を示します。

【IDL言語】

```
module ODsample{
    interface arraytest{
        typedef long fix[4][3][2];
        typedef string str[2][3][4];
        fix op1( in fix para1, out fix para2, inout fix para3 );
        str op2( in str para1, out str para2, inout str para3 );
    };
};
```

これをJava言語で記述すると、以下のようになります。

【Java言語】

＜インタフェース＞

```
package ODsample;
public interface arraytestOperations {
    public int[][][] op1(int[][][] para1,
                         ODsample.arraytestPackage.fixHolder para2,
                         ODsample.arraytestPackage.fixHolder para3 );
    public java.lang.String[][][] op2(java.lang.String[][][] para1,
                         ODsample.arraytestPackage.strHolder para2,
                         ODsample.arraytestPackage.strHolder para3 );
}
```

＜配列名Holderクラス＞

```
package ODsample.arraytestPackage;
public final class fixHolder  implements org.omg.CORBA.portable.Streamable {
    public int[][][] value;
    public fixHolder()  {}
    public fixHolder(int[][][] value) {...}
    public void _read (org.omg.CORBA.portable.InputStream istream) {...}
    public void _write (org.omg.CORBA.portable.OutputStream ostream) {...}
    public org.omg.CORBA.TypeCode _type () {...}
}

package ODsample.arraytestPackage;
public final class strHolder implements org.omg.CORBA.portable.Streamable {
    public java.lang.String[][][] value;
    public strHolder()  {}
    public strHolder(java.lang.String[][][] value) {...}
    public void _read (org.omg.CORBA.portable.InputStream istream) {...}
    public void _write (org.omg.CORBA.portable.OutputStream ostream) {...}
    public org.omg.CORBA.TypeCode _type () {...}
}
```

＜配列名Helperクラス＞

```
package ODsample.arraytestPackage;
public class fixHelper
{
    public static void
        insert(org.omg.CORBA.Any a, int[][][] value){...}
    public static int[][][]
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
```

```
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static int[][][]
        read( org.omg.CORBA.portable.InputStream istream ){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream,
              int[][][] value) throws  org.omg.CORBA.MARSHAL{...}
}

package ODsample.arraytestPackage;
public class strHelper
{
    public static void
        insert(org.omg.CORBA.Any a, java.lang.String[][][] value){...}
    public static java.lang.String[][][]
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static java.lang.String[][][]
        read( org.omg.CORBA.portable.InputStream istream ){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream,
              java.lang.String[][][] value)
              throws  org.omg.CORBA.MARSHAL{...}
}
```

(2)クライアントアプリケーションでの処理

inパラメタの配列型のデータ領域は、newを使用して獲得します。inout／outパラメタの場合は、配列名Holderクラスを使用してデータ領域を割り当てます。inout／outパラメタを参照するには、インスタンスデータのvalueを参照します。

```
import org.omg.CORBA.*;
import ODsample.*;

public class arrayClient {
    public static void main(String args[]) {
        // ORBの前処理
        // オブジェクトリファレンスの獲得
            :
        try{

            int i, j, k;

            // int型配列

            // inパラメタ用変数
            int[][][] in_fix = new int[4][3][2];
            for( i = 0; i < in_fix.length; i++ )
                for( j = 0; j < in_fix[i].length; j++ )
```

```
        for( k = 0; k < in_fix[i][j].length; k++ )
            in_fix[i][j][k]  =  i + j + k;

// outパラメタ用変数
ODsample.arraytestPackage.fixHolder outHolder_fix =
               new ODsample.arraytestPackage.fixHolder();

// inoutパラメタ用変数
int[][][] inout_fix_tmp = new int[4][3][2];
for( i = 0; i < inout_fix_tmp.length; i++ )
    for( j = 0; j < inout_fix_tmp[i].length; j++ )
        for( k = 0; k < inout_fix_tmp[i][j].length; k++ )
            inout_fix_tmp[i][j][k]  =  i + j + k + 1;
ODsample.arraytestPackage.fixHolder inoutHolder_fix =
      new ODsample.arraytestPackage.fixHolder( inout_fix_tmp );

// 復帰値用変数
int[][][] ret_fix;

// サーバアプリケーションのメソッド呼出し
ret_fix = target.op1( in_fix, outHolder_fix, inoutHolder_fix );

// メソッドの結果表示
// 復帰値の表示
for( i = 0; i < ret_fix.length; i++ )
    for( j = 0; j < ret_fix[i].length; j++ )
        for( k = 0; k < ret_fix[i][j].length; k++ )
            System.out.println( "ret_fix[" + i + "][" + j + "][" + k + "]="
                                + ret_fix[i][j][k] );

// outパラメタの表示
for( i = 0; i < outHolder_fix.value.length; i++ )
    for( j = 0; j < outHolder_fix.value[i].length; j++ )
        for( k = 0; k < outHolder_fix.value[i][j].length; k++ )
            System.out.println(
                "outHolder_fix.value[" + i + "][" + j + "][" + k + "]="
                          +  outHolder_fix.value[i][j][k] );

// inoutパラメタの表示
for( i = 0; i < inoutHolder_fix.value.length; i++ )
    for( j = 0; j < inoutHolder_fix.value[i].length; j++ )
        for( k = 0; k < inoutHolder_fix.value[i][j].length; k++ )
            System.out.println(
                "inoutHolder_fix.value[" + i + "][" + j + "][" + k + "]="
                          + inoutHolder_fix.value[i][j][k] );

// String型配列

// inパラメタ用変数
String[][][] in_str = new String[2][3][4];
for( i = 0; i < in_str.length; i++ )
    for( j = 0; j < in_str[i].length; j++ )
        for( k = 0; k < in_str[i][j].length; k++ )
            in_str[i][j][k]  =  "indata" + i + j + k;

// outパラメタ用変数
ODsample.arraytestPackage.strHolder outHolder_str =
               new ODsample.arraytestPackage.strHolder();

// inoutパラメタ用変数
String[][][] inout_str_tmp = new String[2][3][4];
for( i = 0; i < inout_str_tmp.length; i++ )
    for( j = 0; j < inout_str_tmp[i].length; j++ )
```

```
                for( k = 0; k < inout_str_tmp[i][j].length; k++ )
                   inout_str_tmp[i][j][k]  =  "inoutdata" + i + j + k + "snd";
            ODsample.arraytestPackage.strHolder inoutHolder_str =
                new ODsample.arraytestPackage.strHolder( inout_str_tmp );

            // 復帰値用変数
            String[][][] ret_str;

            // サーバアプリケーションのメソッド呼出し
            ret_str = target.op2( in_str, outHolder_str, inoutHolder_str );

            // メソッドの結果表示
            // 復帰値の表示
           for( i = 0; i < ret_str.length; i++ )
               for( j = 0; j < ret_str[i].length; j++ )
                   for( k = 0; k < ret_str[i][j].length; k++ )
                       System.out.println(
                          "ret_str[" + i + "][" + j + "][" + k + "]=" + ret_str[i][j][k] );

            // outパラメタの表示
            for( i = 0; i < outHolder_str.value.length; i++ )
                for( j = 0; j < outHolder_str.value[i].length; j++ )
                    for( k = 0; k < outHolder_str.value[i][j].length; k++ )
                        System.out.println(
                            "outHolder_str.value[" + i + "][" + j + "][" + k + "]="
                                  + outHolder_str.value[i][j][k] );

            // inoutパラメタの表示
            for( i = 0; i < inoutHolder_str.value.length; i++ )
                for( j = 0; j < inoutHolder_str.value[i].length; j++ )
                    for( k = 0; k < inoutHolder_str.value[i][j].length; k++ )
                        System.out.println(
                            "inoutHolder_str.value[" + i + "][" + j + "][" + k + "]="
                                + inoutHolder_str.value[i][j][k] );
        }
        catch ( java.lang.Exception e ) {
            // エラー処理
                :
        }
    }
}
```

(3)サーバアプリケーションでの処理

inout／outパラメタにデータを代入する場合、inout／outのHolderクラスのvalueに設定します。

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servantクラス
// サーバアプリケーションのメソッド
class arrayServant extends arraytestPOA {

    public int[][][] op1( int[][][] in_fix,
                          ODsample.arraytestPackage.fixHolder outHolder_fix,
                          ODsample.arraytestPackage.fixHolder inoutHolder_fix )
    {
        int i, j, k;
```

```
    // 入力パラメタの表示
    // inパラメタの表示
    for( i = 0; i < in_fix.length; i++ )
        for( j = 0; j < in_fix[i].length; j++ )
            for( k = 0; k < in_fix[i][j].length; k++ )
                System.out.println(
                  "in_fix[" + i + "][" + j + "][" + k + "]=" + in_fix[i][j][k] );

    // inoutパラメタの表示
    for( i = 0; i < inoutHolder_fix.value.length; i++ )
        for( j = 0; j < inoutHolder_fix.value[i].length; j++ )
            for( k = 0; k < inoutHolder_fix.value[i][j].length; k++ )
                System.out.println(
                "inoutHolder_fix.value[" + i + "][" + j + "][" + k + "]="
                                + inoutHolder_fix.value[i][j][k] );

    // 出力パラメタの設定
    // outパラメタの設定
    int[][][] out_fix_tmp = new int[4][3][2];
    for( i = 0; i < out_fix_tmp.length; i++ )
        for( j = 0; j < out_fix_tmp[i].length; j++ )
            for( k = 0; k < out_fix_tmp[i][j].length; k++ )
                out_fix_tmp[i][j][k]  =  ( i + j + k ) * 10;
    outHolder_fix.value = out_fix_tmp;

    // inoutパラメタの設定
    for( i = 0; i < inoutHolder_fix.value.length; i++ )
        for( j = 0; j < inoutHolder_fix.value[i].length; j++ )
            for( k = 0; k < inoutHolder_fix.value[i][j].length; k++ )
                inoutHolder_fix.value[i][j][k] = inoutHolder_fix.value[i][j][k] * 20;

    // 復帰の設定
    int[][][] ret_fix = new int[4][3][2];
    for( i = 0; i < ret_fix.length; i++ )
        for( j = 0; j < ret_fix[i].length; j++ )
            for( k = 0; k < ret_fix[i][j].length; k++ )
                ret_fix[i][j][k]  =  ( i + j + k ) * 100;

    // 復帰
    return( ret_fix );
}

public String[][][] op2( String[][][] in_str,
                         ODsample.arraytestPackage.strHolder outHolder_str,
                         ODsample.arraytestPackage.strHolder inoutHolder_str )
{
    int i, j, k;

    // 入力パラメタの表示
    // inパラメタの表示
    for( i = 0; i < in_str.length; i++ )
        for( j = 0; j < in_str[i].length; j++ )
            for( k = 0; k < in_str[i][j].length; k++ )
                System.out.println(
                  "in_str[" + i + "][" + j + "][" + k + "]=" + in_str[i][j][k] );

    // inoutパラメタの表示
    for( i = 0; i < inoutHolder_str.value.length; i++ )
        for( j = 0; j < inoutHolder_str.value[i].length; j++ )
            for( k = 0; k < inoutHolder_str.value[i][j].length; k++ )
                System.out.println(
                    "inoutHolder_str.value[" + i + "][" + j + "][" + k + "]="
                                + inoutHolder_str.value[i][j][k] );
```

```
        // 出力パラメタの設定
        // outパラメタの設定
        String[][][] out_str_tmp = new String[2][3][4];
        for( i = 0; i < out_str_tmp.length; i++ )
            for( j = 0; j < out_str_tmp[i].length; j++ )
                for( k = 0; k < out_str_tmp[i][j].length; k++ )
                    out_str_tmp[i][j][k]  =  "outdata" + i + j + k;
        outHolder_str.value = out_str_tmp;

        // inoutパラメタの設定
        for( i = 0; i < inoutHolder_str.value.length; i++ )
            for( j = 0; j < inoutHolder_str.value[i].length; j++ )
                for( k = 0; k < inoutHolder_str.value[i][j].length; k++ )
                    inoutHolder_str.value[i][j][k] = inoutHolder_str.value[i][j][k] + "-rcv";

        // 復帰の設定
        String[][][] ret_str = new String[2][3][4];
        for( i = 0; i < ret_str.length; i++ )
            for( j = 0; j < ret_str[i].length; j++ )
                for( k = 0; k < ret_str[i][j].length; k++ )
                    ret_str[i][j][k]  =  "retdata" + i + j + k;

        // 復帰
        return( ret_str );
    }
}

public class arrayServer {
    public static void main(String args[]) {
        try {
             // ORBの前処理
                :
             // POAオブジェクトの生成
                :
             // Servantの生成とそのPOAへの登録
                :
             // POAManagerの活性化
                :
        }
        catch (java.lang.Exception e) {
           // エラー処理
                :
        }
    }
}
```

5.20.10 属性宣言(attribute)のマッピング

(1)IDLマッピング

IDL言語でattributeを指定した場合、Java言語では、IDLで定義された変数名と同じ名前のメンバ変数、およびその変数に対するデータ設定／取得用のメソッド(変数名と同じ名前)にマッピングされます。
IDL言語での定義が以下のように定義されていた場合について、アプリケーションプログラムの例を示します。

【IDL言語】

```
module ODsample{
    interface attrtest{
        attribute long para1;
        attribute string para2;
```

```
        readonly attribute long para3;
    };
};
```

これをJava言語で記述すると、以下のようになります。

【Java言語】

＜インタフェース＞

```
package ODsample;
public interface attrtestOperations
{
    public int para1();
    public void para1( int value );
    public java.lang.String para2();
    public void para2( java.lang.String value );
    public int para3();
}
```

＜Holderクラス＞

```
package ODsample;
public final class attrtestHolder  implements org.omg.CORBA.portable.Streamable {
    public ODsample.attrtest value;
    public attrtestHolder() {...}
    public attrtestHolder(ODsample.attrtest __arg) {...}
    public void _write(org.omg.CORBA.portable.OutputStream out) {...}
    public void _read(org.omg.CORBA.portable.InputStream in) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

＜Helperクラス＞

```
package ODsample;
public class attrtestHelper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.attrtest value){...}
    public static ODsample.attrtest
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.attrtest
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.attrtest value){...}
```

```
    public static ODsample.attrtest
        narrow( org.omg.CORBA.Object obj)
               throws org.omg.CORBA.BAD_PARAM{...}
    private static ODsample.attrtest
        narrow( org.omg.CORBA.Object obj, boolean ignore_check)
               throws org.omg.CORBA.BAD_PARAM{...}
}
```

(2)クライアントアプリケーションでの処理

サーバオブジェクトのデータの直接設定および取出しを行います。

```
import org.omg.CORBA.*;
import ODsample.*;

public class attributeClient {
    public static void main(String args[]) {
        // ORBの前処理
        // オブジェクトリファレンスの獲得
              :
       try{
           // サーバアプリケーション変数の設定
           java.util.Date  dt = new java.util.Date();
           target.para1( (int)dt.getTime() );
           target.para2( dt.toString() );

           // サーバアプリケーション変数の参照
           System.out.println( target.para1() );
           System.out.println( target.para2() );
           System.out.println( target.para3() );

        }
        catch ( Exception e ) {
            // エラー処理
                :
        }
    }
}
```

(3)サーバアプリケーションでの処理

サーバアプリケーションのServantクラスでは、インタフェースクラス変数の参照／設定処理を実装します。

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servantクラス
// サーバアプリケーションのメソッド
class attributeServant extends attrtestPOA {

    // 変数の初期化
    private int    value1 = 0;
    private String value2 = "";
    private int    value3 = 999;

    public int para1(){
        return( value1 );
```

```
    }
    public void para1(int arg){
        value1 = arg;
    }
    public String para2(){
        return( value2 );
    }
    public void para2( String arg ){
        value2 = arg;
    }
    public int para3(){
        return( value3 );
    }
}

public class attributeServer {
    public static void main(String args[]) {
        try {
             // ORBの前処理
                :
             // POAオブジェクトの生成
                :
             // Servantの生成とそのPOAへの登録
                :
             // POAManagerの活性化
                :
        }
        catch (java.lang.Exception e) {
            // エラー処理
                :
        }
    }
}
```

5.20.11 定数宣言(const)のマッピング

(1)IDLマッピング

IDL言語で定数(const)をinterface宣言の中で指定した場合、Java言語では、インタフェース名と同じ名前のpublic interface内で、定数名と同じ名前のpublic static finalメンバ(定数メンバ)としてマッピングされます。

```
[package <packageName>];
public interface <interfaceName> {
    public static final <type> <constName> = (<type>)(<value>);
}
```

IDL言語での定義が以下のように定義されていた場合について、Java言語のマッピング例を示します。

【IDL言語】

```
module Example{
    interface Face{
       const long aLongOne = -321;
    };
};
```

これをJava言語にマッピングすると、以下のようになります。

【Java言語】

```
package Example;
public interface FaceOperations {
    public final int aLongOne = (int) -321;
}
```

また、interface宣言の外で指定した場合、定数名と同じ名前のpublic interfaceとその内部に、valueという名前でpublic static finalメンバが定義されます。

```
[package <moduleName>];
public interface <constName> {
       public static final <type> value = (<type>)(<value>);
}
```

IDLを以下のように定義した場合について、Java言語のマッピングの例を示します。

【IDL言語】

```
module Example{
    const long aLongOne = -321;
};
```

これをJava言語でマッピングすると、以下のようになります。

【Java言語】

```
package Example;
public interface aLongOne {
    final public static int value = (int) -321;
}
```

5.20.12 パラメタ受渡しで使用するデータ型

クライアント／サーバアプリケーションでパラメタ受渡しに使用するデータ型を以下に示します。

CORBAデータ型	in	out	inout	return
long	int	org.omg.CORBA. IntHolder	org.omg.CORBA. IntHolder	int
unsigned long	int	org.omg.CORBA. IntHolder	org.omg.CORBA. IntHolder	int
short	short	org.omg.CORBA. ShortHolder	org.omg.CORBA. ShortHolder	short
unsigned short	short	org.omg.CORBA. ShortHolder	org.omg.CORBA. ShortHolder	short

CORBAデータ型	in	out	inout	return
long long	long	org.omg.CORBA. LongHolder	org.omg.CORBA. LongHolder	long
float	float	org.omg.CORBA. FloatHolder	org.omg.CORBA. FloatHolder	float
double	double	org.omg.CORBA. DoubleHolder	org.omg.CORBA. DoubleHolder	double
boolean	boolean	org.omg.CORBA. BooleanHolder	org.omg.CORBA. BooleanHolder	boolean
char	char	org.omg.CORBA. CharHolder	org.omg.CORBA. CharHolder	char
wchar	char	org.omg.CORBA. CharHolder	org.omg.CORBA. CharHolder	char
octet	byte	org.omg.CORBA. ByteHolder	org.omg.CORBA. ByteHolder	byte
enum	データ名 class	データ名 Holderclass	データ名 Holderclass	データ名 class
struct	データ名 class	データ名 Holderclass	データ名 Holderclass	データ名 class
union	データ名 class	データ名 Holderclass	データ名 Holderclass	データ名 class
string	java.lang. String	org.omg.CORBA. StringHolder	org.omg.CORBA. StringHolder	java.lang. String
wstring	java.lang. String	org.omg.CORBA. StringHolder	org.omg.CORBA. StringHolder	java.lang. String
any	org.omg.CORBA. Any	org.omg.CORBA. AnyHolder	org.omg.CORBA. AnyHolder	org.omg.CORBA. Any
sequence	要素の型[ ]	データ名 Holderclass	データ名 Holderclass	要素の型[ ]
配列	要素の型の配列	データ名 Holderclass	データ名 Holderclass	要素の型の配列
array型	データ型配列	データ名 Holder	データ名 Holder	データ型配列
Object	org.omg.CORBA. Object	org.omg.CORBA. ObjectHolder	org.omg.CORBA. ObjectHolder	org.omg.CORBA. CORBA.Object
TypeCode	org.omg.CORBA. TypeCode	org.omg.CORBA. TypeCodeHolder	org.omg.CORBA. TypeCodeHolder	org.omg.CORBA. TypeCode

注意

(1)string型の扱いについて

Javaでstring型を扱う場合、文字列としてnullを設定する際、オブジェクトの値としてのnullではなく、「null文字列」を設定する必要があります。
以下に設定例を示します。

```
java.lang.String str = null;                        // 誤り
java.lang.String str2 = new java.lang.String("");   // 正しい
```

(2)整数の扱いについて

Javaで整数型を扱う場合、符号なしを意味するunsigned型がないため、IDL定義でunsigned型とした変数を扱う場合でも、代入する値は以下の範囲となるよう注意が必要です。

- int:-2147483648 ～ +2147483647
- short:-32768 ～ +32767
- long:-9223372036854775808 ～ +9223372036854775807

(3)char型の扱いについて

Javaでのchar型は、Unicode(16bit)となります。

(4)日本語の扱いについて

IDL定義でchar／stringを使用した場合は、1バイト系文字／文字列だけの受渡しが可能となります。日本語文字／文字列を受け渡す場合は、IDL定義でwchar／wstringを使用してください。
JIS2004で追加された文字は、wstringだけで使用可能です。

(5)nullオブジェクトの設定について

クライアントアプリケーションのin／inoutパラメタ、およびサーバアプリケーションのout／inoutパラメタ、復帰値では、文字列型、シーケンス型、構造体、共用体、配列、Any型に対して“nullオブジェクト”を設定できません。

(6)Holderクラスの扱いについて

クライアントアプリケーションのinoutパラメタ、およびサーバアプリケーションのoutパラメタで、Holderクラスを使用してパラメタを受け渡す場合は、必ずパラメタを受渡す前にHolderクラスのインスタンス内のvalue変数に値を設定してください。

5.21 注意事項

5.21.1 アプリケーション作成時の注意事項

アプリケーションを作成する場合の注意事項については、“第2章 CORBAアプリケーション開発時の留意事項”を参照してください。

5.21.2 クライアント環境でのアプレット運用時の注意事項

- Windows Server(R) 2008、Windows Server(R) 2012、Windows Vista(R)、Windows(R) 7、Windows(R) 8、またはWindows(R) 8.1上のInternet Explorer 7以降を使用している場合、クライアントにログ情報を格納するには、保護モードを「無効」にする必要があります。
- JBKプラグインを使用する場合は、HTTPSプロトコルが使用可能です。

- Portable-ORBを使用し、いったん起動したJava VMを終了しないで同一アプレット（同一URLからダウンロードされるアプレット）の起動/終了を繰り返して運用する場合は、org.omg.CORBA.ORB.destroy()を使用してORBインスタンスが保持している資源を明示的に解放してください。資源を解放しない場合、サーバアプリケーションとの通信時に例外が発生する可能性があります。
JavaScriptを使用してブラウザの起動／終了を行う場合、またはアクティブデスクトップがインストールされた状態でInternet Explorerを使用する場合、ブラウザの終了によりJava VMが終了されないため、必ず通信資源を解放してください。
- アプレットを使用する場合は、デジタル署名を行う必要があります。デジタル署名については、“5.8 アプレットのデジタル署名”を参照してください。
- Portable-ORBのアプレット運用を行う場合は、以下の点に注意してください。
  - ユーザアプリケーションでorg.omg.CORBA.ORB.create_any()によりAny型を生成する際は、必ず引数を持つorg.omg.CORBA.ORB.init()で作成したオブジェクトリファレンスを使用してください。
  - union型を使用する場合は、IDL定義にdefaultメンバを指定できません。
  - enum型は、union型の弁別子の型として指定することはできません。enum型以外の型を指定してください。

5.21.3 その他の注意事項

- 実行環境とバージョンの異なるJDKでコンパイル(javac)したアプリケーションのクラスは、実行環境のJDK/JREでは正しく動作しない場合があります。この場合、実行環境と同一バージョンのJDK環境で再度コンパイル(javac)してから実行してください。

Windows32/64

- 本製品のインストール時に環境変数CLASSPATHに設定されたODjava4.jarを別名に変更した場合、アンインストールしても設定は削除されません。必要がない場合は、環境変数CLASSPATHの設定から削除してください。
- -Xrsオプション指定を省略してアプリケーションを起動した場合、システムのログオフ時にアプリケーションが終了します。CORBAワークユニット上のCORBA-JavaアプリケーションおよびEJBアプリケーションをログオフ後も動作させるには、-Xrsオプションを指定して起動してください。

第6章 アプリケーションの開発（COBOL）

6.1 アプリケーションの開発手順

6.1.1 アプリケーションのインタフェース定義(IDL)の記述

サーバアプリケーションが提供するインタフェースを定義するために、IDLファイルを作成します。IDLファイルの書式については、“IDL定義”を参照してください。

例

```
module ODdemo{
  interface calculator{
    exception ZEROPARAM {};
    struct result {
      long    add_result;
      long    subtract_result;
      long    multiple_result;
      float   divide_result;
    };

    result calculate( in long a, in long b )
        raises( ZEROPARAM );
  };
};
```

6.1.2 IDLファイルのコンパイル

IDLcコマンドでIDLファイルをコンパイルします。この結果、IDL定義をCOBOL言語にマッピングした、複数の*.cblファイルが出力されます。IDLcコマンドの詳細については、“リファレンスマニュアル(コマンド編)”の“IDLc”を参照してください。

例

```
IDLc -cobol simple.idl
```

“6.1.1 アプリケーションのインタフェース定義(IDL)の記述”で例示した内容のsimple.idlというIDLファイルを“-cobol”オプションを指定してIDLcコマンドでコンパイルすると、以下のファイルが出力されます。

ファイル名	ファイル種別
simple_cdr.cbl	CDRソースファイル
simple_stub.cbl	スタブファイル
simple_ODdemo_calculator_skel.cbl	スケルトンファイル
simple_skel.cbl	領域獲得用スケルトン(注1)
simple_h.cbl	データ型定義ファイル(注2)

注1）基本データ型以外のIDL定義に対する領域獲得用関数が組み込まれます。基本データ型だけでは、作成されません。

注2）スタブ・スケルトンで使用されるデータ型を定義したファイルです。

6.1.3 サーバアプリケーションのプログラミング

サーバアプリケーションのコーディングを行います。サーバアプリケーションは、「初期化処理部」と「インタフェース実装部」に分けられます。詳細については、“6.2 サーバアプリケーションのプログラミング（静的スケルトンインタフェース）”を参照してください。
また、アプリケーション開発時の留意事項として“6.10 COBOLアプリケーション使用時の注意事項”と“2.1 コーディング”を参照してください。

6.1.4 クライアントアプリケーションのプログラミング

クライアントアプリケーションのコーディングを行います。クライアントアプリケーションは、「初期化処理部」と「メソッド呼出し部」に分けられます。詳細については、“6.5 クライアントアプリケーションのプログラミング（静的起動インタフェース）”を参照してください。
また、アプリケーション開発時の留意事項として“6.10 COBOLアプリケーション使用時の注意事項”と“2.1 コーディング”を参照してください。

6.1.5 アプリケーションのコンパイルとリンク

開発環境を使用してアプリケーションのコンパイルとリンクを行います。
コンパイル・リンク時の注意事項については、“6.10 COBOLアプリケーション使用時の注意事項”と“2.2 コンパイル・リンク”を参照してください。

6.1.6 サーバアプリケーションの登録

OD_impl_instコマンドを使用して、インプリメンテーションリポジトリにサーバアプリケーション情報を登録します。
また、ネーミングサービスにサーバアプリケーションのオブジェクトリファレンスを登録する場合は、OD_or_admコマンドを使用します。

例

OD_impl_instコマンド実行例

```
OD_impl_inst -ax def
```

defファイルの内容

```
rep_id                      = IDL:ODdemo/calculator:1.0
type                        = persistent
IDL:ODdemo/calculator:1.0 = ./libODDEMO-CALCULATOR.so
mode                        = SYNC_END
```

OD_or_admコマンド実行例

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -n ODdemo::calculator
```

6.2 サーバアプリケーションのプログラミング（静的スケルトンインタフェース）

サーバアプリケーションは、初期化処理部とインタフェースの実装部分から構成されます。初期化処理部では、以下に示す処理を行います。

6.2.1 初期化

CORBAの初期化メソッドCORBA-ORB-INITを呼び出して、初期化処理を行います。当関数の結果として、ORBのオブジェクトリファレンスが通知されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
 IDENTIFICATION DIVISION.
    PROGRAM-ID. "SERVER-MAIN".
    AUTHOR. OD/IDLCOMPILER VER.2.0.
    INSTALLATION. IDL FILE NAME IS COBSAMPLE.IDL.
    SECURITY. THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
              WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
              GURANTEED IS NOT DONE.
    DATE-WRITTEN. TUE MAY  6 11:03:40 1997
 *
    ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
    SPECIAL-NAMES.
        ARGUMENT-NUMBER IS ARG-C
        ARGUMENT-VALUE  IS ARG-V
        SYMBOLIC CONSTANT
        COPY SYMBOL-CONST IN CORBA.
        .
 *
    DATA DIVISION.
 *
    WORKING-STORAGE SECTION.
    COPY CONST IN CORBA.
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
    01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
    01 COPY BOA  IN CORBA REPLACING CORBA-BOA BY BOA.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
    01 COPY REPOSITORYID IN CORBA REPLACING CORBA-REPOSITORYID BY INTF-REP.
    01 COPY FJ-IMPLEMENTATIONDEF IN CORBA REPLACING FJ-IMPLEMENTATIONDEF BY IMPL-REP.
    01 COPY INTERFACEDEF IN CORBA REPLACING CORBA-INTERFACEDEF BY INTF.
    01 COPY IMPLEMENTATIONDEF IN CORBA REPLACING CORBA-IMPLEMENTATIONDEF BY IMPL.
    01 COPY REFERENCEDATA IN CORBA REPLACING CORBA-REFERENCEDATA BY REF.
    01 REF-P USAGE POINTER.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
    01 NAME-A USAGE POINTER.
    01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING
           COSNAMING-NAMINGCONTEXT BY COS-NAMING.
    01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
    01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING
           COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
    01 NAME-COMPONENT-A USAGE POINTER.
    01 STR-BUF PIC X(30).
    01 INTF-INTF-A PIC X(25) VALUE "IDL:ODdemo/calculator:1.0".
    01 IMPL-INTF-A PIC X(25) VALUE "IDL:ODdemo/calculator:1.0".
    01 TEMP-BUF USAGE POINTER.
    01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
    01 MESS PIC X(30).
 *########## ORB SETTING PARAMETER #########
    01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
```

```
   01 CURRENT-ARG-V.
      02 FILLER OCCURS 6.
         03 CURRENT-ARG-V-VALUE USAGE POINTER.
   01 APLI-NAME PIC X(8) VALUE "simple_s".
   01 TMP-STRING-BUF PIC X(20).
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
*
   LINKAGE SECTION.
*
   PROCEDURE DIVISION.
*
   MAIN.
* ObjectDirectorのObjectIDを設定
* argument set : CURRENT-ARG-V-VALUE
       ACCEPT CURRENT-ARG-C FROM ARG-C.
       COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.

       PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
          IF ARG-COUNT = 1
             MOVE APLI-NAME TO TMP-STRING-BUF
          ELSE
             ACCEPT TMP-STRING-BUF FROM ARG-V
          END-IF

          MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
          CALL "CORBA-STRING-SET" USING
                            CURRENT-ARG-V-VALUE (ARG-COUNT)
                            STRING-LENGTH
                            TMP-STRING-BUF
       END-PERFORM.
       SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.
* ObjectDirector初期化
       MOVE FUNCTION LENG (FJ-OM-ORB-ID) TO STRING-LENGTH.
       CALL "CORBA-STRING-SET" USING
                TEMP-BUF
                STRING-LENGTH
                FJ-OM-ORB-ID.
       CALL "CORBA-ORB-INIT" USING
                CURRENT-ARG-C
                CURRENT-ARG-V
                TEMP-BUF
                ENV
                ORB.
       CALL "CORBA-FREE" USING TEMP-BUF.
       MOVE "CORBA-ORB-INIT" TO MESS.
       PERFORM ENV-CHECK
```

また、基本オブジェクトアダプタの初期化処理を行います。

```
 MOVE FUNCTION LENG (CORBA-BOA-OA-ID) TO STRING-LENGTH.
   CALL "CORBA-STRING-SET" USING
            TEMP-BUF
            STRING-LENGTH
            CORBA-BOA-OA-ID.
   CALL "CORBA-ORB-BOA-INIT" USING
            ORB
            CURRENT-ARG-C
            CURRENT-ARG-V
            TEMP-BUF
            ENV
            BOA.
   CALL "CORBA-FREE" USING TEMP-BUF.
```

```
    MOVE "CORBA-ORB-BOA-INIT" TO MESS.
    PERFORM ENV-CHECK.
```

なお、必要であれば、サーバアプリケーションの初期化も行います。

6.2.2 サーバの活性化

サーバアプリケーションの初期化が完了すると、ORBに対してその旨を通知します。ORBは、この命令が発行された時点で、クライアントからの要求をサーバアプリケーションに伝えます。活性化の方法は、サーバのタイプにより以下のように異なります。

サーバのタイプ	メソッド
shared server	CORBA-BOA-IMPL-IS-READY
unshared server	CORBA-BOA-OBJ-IS-READY
persistent server	CORBA-BOA-IMPL-IS-READY

(1)インプリメンテーションリポジトリのオブジェクトリファレンスの獲得

CORBA-ORB-RESOLVE-INITIAL-REFERENCESメソッドを使用して、インプリメンテーションリポジトリのオブジェクトリファレンスを獲得します。パラメタには、CORBA-ORB-OBJECTID-IMPLREPを指定します。

(2)IMPLEMENTATIONDEFオブジェクトのオブジェクトリファレンスの検索

FJ-IMPLEMENTATIONREP-LOOKUP-IDを使用して、IMPLEMENTATIONDEFオブジェクトのオブジェクトリファレンスを求めます。パラメタには、サーバアプリケーションのIMPLEMENTATIONDEFオブジェクトを指定します。

(3)サーバの活性化

CORBA-BOA-IMPL-IS-READY／CORBA-BOA-OBJ-IS-READYを使用して、サーバの活性化を行います。

注意

メソッドの復帰後は、初期化時にオープンしたファイルのクローズ処理／領域の解放処理などの非活性化処理を行う必要があります。詳細については、“6.2.4 サーバの非活性化”を参照してください。

注意

Windows32/64

以下の条件のとき、CORBA-BOA-IMPL-IS-READY／CORBA-BOA-OBJ-IS-READYが復帰せずにサーバアプリケーションが終了します。

- サーバアプリケーションをプロセスモードで作成している場合 かつ
- サーバアプリケーションをCORBAワークユニット運用している場合 かつ
- CORBAワークユニットを停止した場合

CORBA-BOA-IMPL-IS-READY／CORBA-BOA-OBJ-IS-READYの後に後処理を行う場合は、サーバアプリケーションをスレッドモードで作成するか、またはワークユニットの出口機能を利用してください。ワークユニットの出口機能に関しては、“OLTPサーバ運用ガイド”の“ワークユニットの出口機能”を参照してください。

```
* インプリメンテーション情報のオブジェクトリファレンス検索
    MOVE FUNCTION LENG (CORBA-ORB-OBJECTID-IMPLREP) TO STRING-LENGTH.
    CALL "CORBA-STRING-SET" USING
```

```
            TEMP-BUF
            STRING-LENGTH
            CORBA-ORB-OBJECTID-IMPLREP.
   CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
            ORB
            TEMP-BUF
            ENV
            IMPL-REP.
   CALL "CORBA-FREE" USING TEMP-BUF.
   MOVE "CORBA-ORB-RESOLVE-INITIAL-REFERENCES-2" TO MESS.
   PERFORM ENV-CHECK
* ImplementationRepオブジェクトリファレンス検索
   MOVE FUNCTION LENG (IMPL-INTF-A) TO STRING-LENGTH.
   CALL "CORBA-STRING-SET" USING
            TEMP-BUF
            STRING-LENGTH
            IMPL-INTF-A.
   CALL "FJ-IMPLEMENTATIONREP-LOOKUP-ID" USING
            IMPL-REP
            TEMP-BUF
            ENV
            IMPL.
   CALL "CORBA-FREE" USING TEMP-BUF.
   MOVE "FJ-IMPLEMENTATIONREP-LOOKUP-ID" TO MESS.
   PERFORM ENV-CHECK
* サーバの活性化
   CALL "CORBA-BOA-IMPL-IS-READY" USING
         BOA
         IMPL
         ENV.
   MOVE "CORBA-BOA-IMPL-IS-READY" TO MESS.
   PERFORM ENV-CHECK.
```

6.2.3 インタフェース実装関数

初期化処理の後、サーバアプリケーションで実装するインタフェースの処理を記述します。インタフェース実装関数のパラメタには、オブジェクトリファレンス、IDLで定義したパラメタ、CORBA_Environmen構造体へのポインタが渡されます。
なお、アプリケーションで使用可能なCORBAのデータ型については、“6.8 データ型に対するマッピング”を参照してください。

```
 IDENTIFICATION DIVISION.
* calculateメソッドの実装関数
 PROGRAM-ID. "ODDEMO-CALCULATOR-CALCULATE".
 AUTHOR. OD/IDLCOMPILER VER.2.0.
 INSTALLATION. IDL FILE NAME IS COBSAMPLE.IDL.
 SECURITY. THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
           WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
           GURANTEED IS NOT DONE.
 DATE-WRITTEN. TUE MAY  6 11:03:40 1997
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
     ARGUMENT-NUMBER IS ARG-C
     ARGUMENT-VALUE  IS ARG-V
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .
*
 INPUT-OUTPUT SECTION.
*
 DATA DIVISION.
*
```

```
 WORKING-STORAGE SECTION.
 COPY CONST IN CORBA.
 01 MESS PIC X(30).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 COPY ORB IN CORBA REPLACING CORBA-ORB  BY ORB.
 01 COPY BOA IN CORBA REPLACING CORBA-BOA  BY BOA.
 01 STRING-TMP  PIC X(10).
 01 COPY ULONG  IN CORBA REPLACING CORBA-UNSIGNED-LONG BY FLAGS.
 01 TEMP-BUF USAGE POINTER.
 01 EXCEP.
    03  COPY LONG IN CORBA REPLACING CORBA-LONG BY EXCEP-L.
 01 EXCEP-A  USAGE POINTER.
 01 EX-ODDEMO-CALCULATOR-ZEROPARAM PIC X(35) VALUE "IDL:ODdemo/calculator/ZEROPARAM:1.0".
 01 COPY EXCEPTION-TYPE IN CORBA REPLACING CORBA-EXCEPTION-TYPE BY EX-STATUS.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY C-ARG-C.
 01 C-ARG-V.
    02 FILLER OCCURS 2.
       03 ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_s".

 LINKAGE SECTION.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARAM1.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARAM2.
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 01 A-RESULT.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY ADD-RESULT.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SUBTRACT-RESULT.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY MULTIPLE-RESULT.
    02 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY DEVIDE-RESULT.
*
 PROCEDURE DIVISION USING
* サーバのオブジェクトリファレンス
         OBJ
* 入力値
         PARAM1
* 入力値
         PARAM2
* エラー値
         ENV
* 演算結果
         A-RESULT.
*
 MAIN.

     COMPUTE C-ARG-C = 1.
     MOVE FUNCTION LENG (APLI-NAME) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           ARG-V-VALUE (1)
           STRING-LENGTH
           APLI-NAME.
     SET ARG-V-VALUE(2) TO NULL.

     MOVE    12 TO STRING-LENGTH.
     CALL    "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           C-ARG-C
           C-ARG-V
           TEMP-BUF
           ENV
```

```
           ORB.
      MOVE "CORBA-ORB-INIT" TO MESS.
      PERFORM ENV-CHECK

      CALL "CORBA-FREE" USING
           TEMP-BUF.

      MOVE 15 TO STRING-LENGTH.
      CALL       "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-BOA-OA-ID.
      CALL "CORBA-ORB-BOA-INIT" USING
           ORB
           C-ARG-C
           C-ARG-V
           TEMP-BUF
           ENV
           BOA.
      MOVE "CORBA-ORB-BOA-INIT" TO MESS.
      PERFORM ENV-CHECK.
 * 0除算チェック
      IF PARAM2 = 0
          MOVE 1 TO FLAGS
          MOVE FUNCTION LENG(EX-ODDEMO-CALCULATOR-ZEROPARAM) TO STRING-LENGTH
          CALL "CORBA-STRING-SET" USING
               TEMP-BUF
               STRING-LENGTH
               EX-ODDEMO-CALCULATOR-ZEROPARAM
          MOVE  0 TO  EXCEP-L OF EXCEP
          MOVE  FUNCTION ADDR (EXCEP) TO EXCEP-A

          CALL "CORBA-BOA-SET-EXCEPTION" USING
               BOA
               FLAGS
               TEMP-BUF
               EXCEP-A
               ENV
      ELSE
 * Calculate
          COMPUTE  ADD-RESULT      OF A-RESULT = PARAM1 + PARAM2
          COMPUTE  SUBTRACT-RESULT OF A-RESULT = PARAM1 - PARAM2
          COMPUTE  MULTIPLE-RESULT OF A-RESULT = PARAM1 * PARAM2
          COMPUTE  DEVIDE-RESULT   OF A-RESULT = PARAM1 / PARAM2
      END-IF.
      EXIT PROGRAM.
  MAIN-END.
```

6.2.4 サーバの非活性化

サーバアプリケーションは、利用者からの停止要求を受けた場合、クライアントからの要求を以降受け付けない旨をORBに対して返します。サーバアプリケーションの利用者からの停止要求の方法およびアプリケーションでの停止要求の受信方法などは各ORBベンダーにより異なります。この命令が発行された時点で、ORBは、クライアントから要求されてもサーバアプリケーションに伝えず、クライアントへは例外が返ります。非活性化の方法は、サーバのタイプにより以下のように異なります。

サーバのタイプ	メソッド
shared server	CORBA-BOA-DEACTIVATE-IMPL
unshared server	CORBA-BOA-DEACTIVATE-OBJ
persistent server	CORBA-BOA-DEACTIVATE-IMPL

CORBA-BOA-DEACTIVATE-IMPLでは、サーバアプリケーションのIMPLEMENTATIONREPオブジェクトをパラメタとして指定します。CORBA-BOA-DEACTIVATE-OBJでは、オブジェクトリファレンスをパラメタとして指定します。

```
 * サーバの非活性化
    CALL "CORBA-BOA-DEACTIVATE-IMPL" USING
         BOA
         IMPL
         ENV
         RET-VAL.
```

なお、ワークユニット運用している場合は、isstopwuコマンド／Interstage管理コンソールから停止操作が実行されることにより、クライアントからの要求を以降受け付けない旨をORBに対して通知します。そのため、アプリケーション側で非活性化用のメソッドを発行する必要はありません。
ワークユニット運用ではないCORBAアプリケーションの場合も、odcntlqueコマンドでサーバアプリケーションを終了する場合、アプリケーション側で非活性化用のメソッドを発行する必要はありません。

6.3 サーバアプリケーションの例外処理

(1)例外情報の設定

ユーザ例外を設定する場合は、CORBA-STRING-SET関数でユーザ例外を識別する文字列を設定する領域を獲得し、この領域に文字列をコピーします。
次に、IDLコンパイラで生成されたALLOC関数でユーザ例外を設定する領域を獲得します。ALLOC関数名は、IDLで指定したインタフェース名、例外の識別名、およびALLOCをハイフン(-)でつなげた形式で指定します。IDLで定義した例外情報はCOBOLでは構造体となるので、構造体のメンバ変数に値を設定します。
最後に、CORBA-BOA-SET-EXCEPTION関数でCORBA-ENVIRONMENT構造体に例外情報を設定します。

例外情報の設定例として、以下のIDL定義を使用します。

```
module ODdemo{
    interface       calculator{
            exception ZEROPARAM { long err_code; };
            struct result {
                    long    add_result;
                    long    subtract_result;
                    long    multiple_result;
                    float   divide_result;
            };
            result calculate( in long a, in long b )
                    raises( ZEROPARAM );
    };
};
```

例外情報の設定例を以下に示します。

```
  IDENTIFICATION DIVISION.
  PROGRAM-ID. "ODDEMO-CALCULATOR-CALCULATE".
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SYMBOLIC CONSTANT
  COPY SYMBOL-CONST IN CORBA.

  INPUT-OUTPUT SECTION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
     01 COPY FLOAT IN CORBA    REPLACING CORBA-FLOAT FY FA.
```

```
    01 COPY FLOAT IN CORBA    REPLACING CORBA-FLOAT BY FB.
    01 ZEROPARAM-P USAGE POINTER.
    01 COPY EXCEPTION-TYPE IN CORBA REPLACING CORBA-EXCEPTION-TYPE BY FLAGS.
    01 USER-ID USAGE POINTER.
    01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY USER-ID-LENGTH.
    01 EX-ODDEMO-CALCULATOR-ZEROPARAM PIC X(35) VALUE "IDL:ODdemo/calculator/ZEROPARAM:1.0".

 LINKAGE SECTION.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY A.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY B.
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
    01 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY FRET.
    01 ZEROPARAM-V.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY ERROR-CODE.
 *
  PROCEDURE DIVISION USING
           OBJ
           A
           B
           ENV
           FRET.
 *
  MAIN.
       MOVE A TO FA.
       MOVE B TO FB.
 * エラーZEROPARAMをenvに設定します
       IF B = 0

          SET CORBA-USER-EXCEPTION OF FLAGS TO TRUE

          MOVE FUNCTION LENG (EX-ODDEMO-CALCULATOR-ZEROPARAM) TO USER-ID-LENGTH
          CALL "CORBA-STRING-SET" USING
                USER-ID
                USER-ID-LENGTH
                EX-ODDEMO-CALCULATOR-ZEROPARAM

          CALL "ODDEMO-CALCULATOR-ZEROPARAM-ALLOC" USING ZEROPARAM-P
          SET ADDRESS OF ZEROPARAM-V TO ZEROPARAM-P
          MOVE 0 TO ERROR-CODE OF ZEROPARAM-V

          CALL "CORBA-BOA-SET-EXCEPTION" USING
                BOA
                FLAGS
                USER-ID
                ZEROPARAM-P
                ENV
                RET
          MOVE 0 TO FRET
          GO TO MAIN-END

       END-IF
       COMPUTE FRET = FA / FB.
       EXIT PROGRAM.
  MAIN-END.
  END PROGRAM  "ODDEMO-CALCULATOR-CALCULATE"
```

注意

サーバメソッドの引数に渡されるCORBA-ENVIRONMENT構造体を、サーバメソッド内部から他のメソッドを呼び出す場合の引数として使用しないでください。サーバメソッドの例外情報に、他のメソッドで発生したサーバメソッド内部の例外情報が設定されることになり

ます。サーバメソッド内部から他のメソッドを呼び出す場合は、別のCORBA-ENVIRONMENT構造体を使用してください。
正しいプログラム例／誤ったプログラム例を以下に示します。

[正しいプログラム例]

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "ODDEMO-CALCULATOR-CALCULATE".
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SYMBOLIC CONSTANT
COPY SYMBOL-CONST IN CORBA.

INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ-A.
   01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING COSNAMING-NAMINGCONTEXT BY COS-NAMING.
   01 COPY COSNAMING-NAME          IN CORBA REPLACING COSNAMING-NAM BY NAME.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY LOCAL-ENV.

LINKAGE SECTION.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY A.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY B.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY FRET.
*
 PROCEDURE DIVISION USING
         OBJ
         A
         B
         ENV
         FRET.
*
 MAIN.
      :

*
*  COSNAMING-NAMINGCONTEXT-BINDの例外情報は、LOCAL_ENVに設定されるため、
*  ODDEMO-CALCULATOR-CALCULATEの例外情報として扱われることはない
*
    CALL "COSNAMING-NAMINGCONTEXT-BIND" USING
            COS-NAMING
            NAME
            OBJ-A
            LOCAL-ENV.
        :
```

[誤ったプログラム例]

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "ODDEMO-CALCULATOR-CALCULATE".
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SYMBOLIC CONSTANT
COPY SYMBOL-CONST IN CORBA.

INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ-A.
```

```
    01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING COSNAMING-NAMINGCONTEXT BY COS-NAMING.
    01 COPY COSNAMING-NAME          IN CORBA REPLACING COSNAMING-NAM BY NAME.

LINKAGE SECTION.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY A.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY B.
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
    01 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY FRET.
*
 PROCEDURE DIVISION USING
          OBJ
          A
          B
          ENV
          FRET.
*
 MAIN.
      :

*
* COSNAMING-NAMINGCONTEXT-BINDが例外復帰した場合、COSNAMING-NAMINGCONTEXT-BIND
* の例外情報がODDEMO-CALCULATOR-CALCULATEの例外情報として扱われてしまう
*
     CALL "COSNAMING-NAMINGCONTEXT-BIND" USING
             COS-NAMING
             NAME
             OBJ-A
             ENV.
     :
```

(2)例外情報の獲得

サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“6.6 クライアントアプリケーションの例外処理”を参照してください。

6.4 サーバアプリケーションの登録

作成したアプリケーションをインプリメンテーションリポジトリ、ネーミングサービスに登録します。

6.4.1 インプリメンテーションリポジトリへの登録

OD_impl_instコマンドを使用してインプリメンテーションリポジトリにサーバアプリケーション情報を登録します。
OD_impl_instコマンドでの登録処理の例と指定する情報について以下に示します。

例

インプリメンテーションリポジトリへの登録

```
OD_impl_inst -ax def
```

-ax def

定義ファイルで指定した定義情報でサーバアプリケーション情報を登録することを示します。

Windows32/64

defファイルの内容

```
rep_id                    = IDL:ODdemo/calculator:1.0
type                      = shared
binary                    = D:¥server¥simple_s.exe
IDL:ODdemo/calculator:1.0 = ODDEMO-CALCULATOR.dll,DSI
ior                       = 1.1
locale                    = SJIS
```

rep_id = IDL:ODdemo/calculator:1.0

インプリメンテーションリポジトリIDを指定します。

type = shared

サーバタイプを指定します。指定できるタイプには、persistent、shared、unsharedがあります。

binary = D:¥server¥simple_s.exe

サーバアプリケーションのパス名を指定します。

IDL:ODdemo/calculator:1.0 = ODDEMO-CALCULATOR.dll,DSI

サーバアプリケーションのライブラリ名を指定します。動的スケルトンインタフェースの場合は、ライブラリパスの後に“,DSI”を定義する必要があります。
ライブラリは、サーバアプリケーションと同一ディレクトリ、または環境変数PATHに指定されているディレクトリに格納する必要があります。

ior = 1.1

連携するIORのバージョンを指定します。指定できるバージョンには、1.0、1.1があります。

locale = SJIS

サーバアプリケーションのコード系を指定します。
開発環境として“NetCOBOL”を使用してUNICODEのアプリケーションを開発する場合は、“UNICODE”を指定してください。

Solaris32/64

defファイルの内容

```
rep_id                    = IDL:ODdemo/calculator:1.0
type                      = shared
binary                    = /home/guest/simple_s
IDL:ODdemo/calculator:1.0 = libODDEMO-CALCULATOR.so,DSI
uid                       = user
gid                       = group
ior                       = 1.1
locale                    = EUC
env                       = LD_LIBRARY_PATH=/opt/FJSVcbl/lib:/opt/FSUNod/lib;LANG=ja
```

rep_id = IDL:ODdemo/calculator:1.0

インプリメンテーションリポジトリIDを指定します。

type = shared

サーバタイプを指定します。指定できるタイプには、persistent、shared、unsharedがあります。

binary = /home/guest/simple_s

サーバアプリケーションのパス名を指定します。

IDL:ODdemo/calculator:1.0 = libODDEMO-CALCULATOR.so,DSI

サーバアプリケーションのライブラリ名を指定します。動的スケルトンインタフェースの場合は、ライブラリパスの後に“,DSI”を定義する必要があります。
ライブラリは、環境変数LD_LIBRARY_PATHに指定されたディレクトリに格納する必要があります。

uid = user

サーバアプリケーション実行時のユーザIDを指定します。

gid = group

サーバアプリケーション実行時のグループIDを指定します。

ior = 1.1

連携するIORのバージョンを指定します。指定できるバージョンには、1.0、1.1があります。

locale = EUC

サーバアプリケーションのコード系を指定します。
開発環境として“NetCOBOL”を使用してUNICODEのアプリケーションを開発する場合は、“UNICODE”を指定してください。

env = LD_LIBRARY_PATH=/opt/FJSVcbl/lib:/opt/FSUNod/lib;LANG=ja

サーバアプリケーション実行時の環境変数を指定します。
アプリケーションの実行に必要なライブラリパスを指定してください。

Linux32/64

defファイルの内容

```
rep_id                    = IDL:ODdemo/calculator:1.0
type                      = shared
binary                    = /home/guest/simple_s
IDL:ODdemo/calculator:1.0 = libODDEMO-CALCULATOR.so,DSI
uid                       = user
gid                       = group
ior                       = 1.1
locale                    = EUC
env                       = LD_LIBRARY_PATH=/opt/FJSVcbl/lib:/opt/FJSVod/lib:LANG=ja
```

rep_id = IDL:ODdemo/calculator:1.0

インプリメンテーションリポジトリIDを指定します。

type = shared

サーバタイプを指定します。指定できるタイプには、persistent、shared、unsharedがあります。

binary = /home/guest/simple_s

サーバアプリケーションのパス名を指定します。

IDL:ODdemo/calculator:1.0 = libODDEMO-CALCULATOR.so,DSI

サーバアプリケーションのライブラリ名を指定します。動的スケルトンインタフェースの場合は、ライブラリパスの後に“,DSI”を定義する必要があります。
ライブラリは、環境変数LD_LIBRARY_PATHに指定されたディレクトリに格納する必要があります。

uid = user

サーバアプリケーション実行時のユーザIDを指定します。

gid = group

サーバアプリケーション実行時のグループIDを指定します。

ior = 1.1

連携するIORのバージョンを指定します。指定できるバージョンには、1.0、1.1があります。

locale = EUC

サーバアプリケーションのコード系を指定します。
開発環境として“NetCOBOL”を使用してUNICODEのアプリケーションを開発する場合は、“UNICODE”を指定してください。

env = LD_LIBRARY_PATH=/opt/FJSVcbl/lib:/opt/FJSVod/lib;LANG=ja

サーバアプリケーション実行時の環境変数を指定します。
アプリケーションの実行に必要なライブラリパスを指定してください。

6.4.2 ネーミングサービスへの登録

作成したサーバアプリケーションを他のアプリケーションからオブジェクトとしてアクセスできるようにするために、そのオブジェクトを識別するためのオブジェクトリファレンスを作成します。同時に作成したオブジェクトリファレンスをネーミングサービスに登録します。
オブジェクトを作成する方法として以下の方法があります。

- OD_or_admコマンドを使用します。
- サーバアプリケーションでBOAの機能を使用してオブジェクトリファレンスを作成し、ネーミングサービスに登録します。

■ OD_or_admコマンドを使用する方法

OD_or_admコマンドは、オブジェクトリファレンス作成後、ネーミングサービスに登録します。
OD_or_admコマンドでの登録例と指定する情報を以下に示します。

例

Windows32/64

ネーミングサービスへの登録

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -L SJIS -n ODdemo::calculator
```

-c IDL:ODdemo/calculator:1.0

指定されたインタフェースリポジトリIDでオブジェクトリファレンスを登録します

-L SJIS

ネーミングサービスに登録するオブジェクトに対してコード系を指定します。

-n ODdemo::calculator

ネーミングサービスに登録するオブジェクト名を指定します。

Solaris32/64 Linux32/64

ネーミングサービスへの登録

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -L EUC -n ODdemo::calculator
```

-c IDL:ODdemo/calculator:1.0

指定されたインタフェースリポジトリIDでオブジェクトリファレンスを登録します

-L EUC

ネーミングサービスに登録するオブジェクトに対してコード系を指定します。

-n ODdemo::calculator

ネーミングサービスに登録するオブジェクト名を指定します。

■ サーバアプリケーションで作成する方法

以下の流れで処理を行います。

プログラミングの例を以下に示します。

```
*ORB-INITで取得したObject
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
*ORB-BOA-INITで取得したObject
 01 COPY BOA IN CORBA REPLACING CORBA-BOA BY BOA.
*インタフェースリポジトリのオブジェクトリファレンス
 01 COPY REPOSITORYID IN CORBA REPLACING CORBA-REPOSITORYID BY INTF-REP.
*インプリメンテーションリポジトリのオブジェクトリファレンス
 01 COPY FJ-IMPLEMENTATIONDEF IN CORBA REPLACING FJ-IMPLEMENTATIONDEF BY IMPL-REP.
*インタフェースリポジトリのInterfaceDef
 01 COPY INTERFACEDEF IN CORBA REPLACING CORBA-INTERFACEDEF BY INTF.
*インプリメンテーションリポジトリのImplementationDef
 01 COPY IMPLEMENTATIONDEF IN CORBA REPLACING CORBA-IMPLEMENTATIONDEF BY IMPL.
*ReferenceData格納域
 01 COPY REFERENCEDATA IN CORBA REPLACING CORBA-REFERENCEDATA BY R-ID.
*その他の変数
 01 TEMP-BUF USAGE POINTER.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 INTF-INTF-A PIC X(25) VALUE "IDL:ODdemo/calculator:1.0".
 01 IMPL-INTF-A PIC X(25) VALUE "IDL:ODdemo/calculator:1.0".
 01 R-ID-P USAGE POINTER.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY NEW-OBJ.
```

```
 01 NAMESERVICEID USAGE POINTER.
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING
            COSNAMING-NAMINGCONTEXT BY COS-NAMING.
 01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
 01 NAME-A USAGE POINTER.
 01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING
       COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
 01 NAME-COMPONENT-A USAGE POINTER.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY NUM.
 01 STR-BUF PIC X(64).

*ObjectDiretorの初期化（省略）
            :

*インタフェースリポジトリのオブジェクトリファレンスの獲得
     MOVE FUNCTION LENG (CORBA-ORB-OBJECTID-LIGHTINTFR) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-ORB-OBJECTID-LIGHTINTFR.
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
           ORB
           TEMP-BUF
           ENV
           INTF-REP.
     CALL "CORBA-FREE" USING TEMP-BUF.

*InterfaceDefオブジェクトリファレンスの獲得
*INTF-INTF-A : インプリメンテーションリポジトリID
     MOVE FUNCTION LENG (INTF-INTF-A) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           INTF-INTF-A.
     CALL "CORBA-REPOSITORY-LOOKUP-ID" USING
           INTF-REP
           TEMP-BUF
           ENV
           INTF.
     CALL "CORBA-FREE" USING TEMP-BUF.

*インプリメンテーションリポジトリのオブジェクトリファレンスの獲得
     MOVE FUNCTION LENG (CORBA-ORB-OBJECTID-IMPLREP) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-ORB-OBJECTID-IMPLREP.
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
           ORB
           TEMP-BUF
           ENV
           IMPL-REP.
     CALL "CORBA-FREE" USING TEMP-BUF.

*ImplementationDefオブジェクトのオブジェクトリファレンスの獲得
     MOVE FUNCTION LENG (IMPL-INTF-A) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           IMPL-INTF-A.
     CALL "FJ-IMPLEMENTATIONREP-LOOKUP-ID" USING
           IMPL-REP
           TEMP-BUF
```

```
           ENV
           IMPL.
     CALL "CORBA-FREE" USING TEMP-BUF.

* オブジェクトリファレンスの作成
    MOVE FUNCTION ADDR(R-ID) TO R-ID-P.
    CALL "CORBA-BOA-CREATE" USING
          BOA
          R-ID-P
          INTF
          IMPL
          ENV
          NEW-OBJ.

* NamingServiceのObjectIDの設定
    MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
    CALL  "CORBA-STRING-SET" USING
          NAMESERVICEID
          STRING-LENGTH
          CORBA-ORB-OBJECTID-NAMESERVICE.

* NamingServiceのオブジェクトリファレンス獲得
    CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
          ORB
          NAMESERVICEID
          ENV
          COS-NAMING.

* NAMESERVICEIDに格納した領域の解放
    CALL "CORBA-FREE" USING NAMESERVICEID.

* オブジェクト名の設定
    MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
    MOVE "ODdemo::calculator" TO STR-BUF.
    CALL "CORBA-STRING-SET" USING
          IDL-ID OF NAME-COMPONENT
          STRING-LENGTH
          STR-BUF.

* オブジェクトのタイプの設定
    MOVE " " TO STR-BUF.
    CALL "CORBA-STRING-SET" USING
          KIND OF NAME-COMPONENT
          STRING-LENGTH
          STR-BUF.

* オブジェクト名の数
    MOVE 1 TO SEQ-LENGTH OF NAME.
    MOVE 1 TO SEQ-MAXIMUM OF NAME.
    MOVE 1 TO NUM.

* オブジェクト名格納域の獲得と設定
    CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
          SEQ-MAXIMUM OF NAME
          SEQ-BUFFER OF NAME.
    MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
    MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
    CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
          NAME-A
          NUM
          NAME-COMPONENT-A.

* サーバアプリケーションのオブジェクト
```

```
    CALL "COSNAMING-NAMINGCONTEXT-BIND" USING
         COS-NAMING
         NAME
         NEW-OBJ
         ENV.
```

注意

CORBA-BOA-CREATE関数呼出し時に、インプリメンテーションリポジトリに定義されているデフォルトコード系でオブジェクトリファレンスを作成します。デフォルトコード系は、OD_impl_inst／OD_set_envコマンドで設定します。

6.5 クライアントアプリケーションのプログラミング（静的起動インタフェース）

静的起動インタフェースを使用する場合の、クライアントアプリケーションの処理の流れを以下に示します。

6.5.1 初期化

CORBAの初期化メソッドCORBA-ORB-INITを呼び出して、初期化処理を行います。当メソッドの結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。環境部(ENVIRONMENT DIVISION)およびデータ部(DATA DIVISION)を定義します。

```
 IDENTIFICATION DIVISION.
    PROGRAM-ID. "CLIENT-MAIN".
    AUTHOR. OD/IDLCOMPILER VER.2.0.
    INSTALLATION. IDL FILE NAME IS COBSAMPLE.IDL.
    SECURITY. THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
            WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
            GURANTEED IS NOT DONE.
    DATE-WRITTEN. TUE MAY  6 11:03:40 1997
 *
    ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
    SPECIAL-NAMES.
        ARGUMENT-NUMBER    IS ARG-C
        ARGUMENT-VALUE     IS ARG-V
        SYMBOLIC CONSTANT
        COPY SYMBOL-CONST IN  CORBA.
        .
 *
```

```
   DATA DIVISION.
*
   WORKING-STORAGE SECTION.
   COPY CONST IN CORBA.
   01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
   01 CURRENT-ARG-V.
      02 FILLER OCCURS 6.
         03 CURRENT-ARG-V-VALUE USAGE POINTER.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
   01 APLI-NAME PIC X(8) VALUE "simple_c".
   01 TMP-STRING-BUF PIC X(20).
   01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
   01 TEMP-BUF USAGE POINTER.
   01 MESS PIC X(30).
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
   01 COPY BOA IN CORBA REPLACING CORBA-BOA BY BOA.
*
   PROCEDURE DIVISION.
*
   MAIN.
* ObjectDirectorのObjectIDを設定
       DISPLAY "CLIENT START!!".
* argument set : CURRENT-ARG-V-VALUE
       ACCEPT CURRENT-ARG-C FROM ARG-C.
       COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
       PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
          IF ARG-COUNT = 1
             MOVE APLI-NAME TO TMP-STRING-BUF
          ELSE
             ACCEPT TMP-STRING-BUF FROM ARG-V
          END-IF
          MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
          CALL "CORBA-STRING-SET" USING
                CURRENT-ARG-V-VALUE (ARG-COUNT)
                STRING-LENGTH
                TMP-STRING-BUF
       END-PERFORM.
       SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.
* ObjectDirector初期化
       MOVE        12 TO STRING-LENGTH.
       CALL        "CORBA-STRING-SET" USING
             TEMP-BUF
             STRING-LENGTH
             FJ-OM-ORB-ID.
       CALL "CORBA-ORB-INIT" USING
             CURRENT-ARG-C
             CURRENT-ARG-V
             TEMP-BUF
             ENV
             ORB.
       CALL "CORBA-FREE" USING TEMP-BUF.
       MOVE "CORBA-ORB-INIT" TO MESS.
       PERFORM ENV-CHECK

       MOVE 15 TO STRING-LENGTH.
       CALL        "CORBA-STRING-SET" USING
             TEMP-BUF
             STRING-LENGTH
             CORBA-BOA-OA-ID.
       CALL "CORBA-ORB-BOA-INIT" USING
             ORB
             CURRENT-ARG-C
```

```
            CURRENT-ARG-V
            TEMP-BUF
            ENV
            BOA.
* ORBIDに格納した領域を解放する
       CALL "CORBA-FREE" USING TEMP-BUF.
       MOVE "CORBA-ORB-BOA-INIT" TO MESS.
       PERFORM ENV-CHECK.
```

例外処理については、“6.6 クライアントアプリケーションの例外処理”を参照してください。

6.5.2 ネーミングサービスのオブジェクトリファレンスの獲得

実行するオブジェクトをネーミングサービスから検索するために、ネーミングサービスのオブジェクトリファレンスが必要となります。CORBAインタフェースのオブジェクトリファレンスの取出しメソッドCORBA-ORB-RESOLVE-INITIAL-REFERENCESによりネーミングサービスのオブジェクトリファレンスを取り出します。このとき、CORBA-ORB-OBJECTID-NAMESERVICEをメソッドのパラメタとして指定します。なお、この定数は、各ORBベンダーが提供しているインクルードファイルなどで定義されています。

```
  DATA DIVISION.
  WORKING-STORAGE SECTION.
   01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING
           COSNAMING-NAMINGCONTEXT BY COS-NAMING.

  PROCEDURE DIVISION.
* NamingServiceのObjectIDの設定
     MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-ORB-OBJECTID-NAMESERVICE.
* NamingServiceのオブジェクトリファレンス獲得
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
           ORB
           TEMP-BUF
           ENV
           COS-NAMING.
* NAMESERVICEIDに格納した領域を解放します
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" TO MESS.
     PERFORM ENV-CHECK.
```

6.5.3 サーバアプリケーションのオブジェクトリファレンスの獲得

ネーミングサービスのメソッドCOSNAMING-NAMINGCONTEXT-RESOLVEにより、これから実行したいサーバアプリケーションのオブジェクトリファレンスを取り出します。検索したいオブジェクト名を当メソッドのパラメタとして指定します。

```
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 STR-BUF PIC X(30).
   01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
   01 NAME-A USAGE POINTER.
   01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING
           COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
   01 NAME-COMPONENT-A USAGE POINTER.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.

  PROCEDURE DIVISION.
* オブジェクト名
     MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
     MOVE "ODdemo::calculator" TO STR-BUF.
     CALL "CORBA-STRING-SET" USING
           IDL-ID OF NAME-COMPONENT
```

```
           STRING-LENGTH
           STR-BUF.
* オブジェクトのタイプ
     MOVE " " TO STR-BUF.
     CALL "CORBA-STRING-SET" USING
           KIND OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
* オブジェクト名の数
     MOVE 1 TO SEQ-LENGTH OF NAME.
     MOVE 1 TO SEQ-MAXIMUM OF NAME.
* Sequenceの要素数
     MOVE 1 TO NUM.
*  CosNamingのアクセス情報格納用のSequence領域獲得
     CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
           SEQ-MAXIMUM OF NAME
           SEQ-BUFFER OF NAME.
     MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
     MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
* CosNamingの要素設定
     CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
           NAME-A
           NUM
           NAME-COMPONENT-A.
*  サーバアプリケーションのオブジェクトリファレンスを獲得
     CALL "COSNAMING-NAMINGCONTEXT-RESOLVE" USING
              COS-NAMING
              NAME
              ENV
              OBJ.
     MOVE "COSNAMING-NAMINGCONTEXT-RESOLVE" TO MESS.
     PERFORM ENV-CHECK.
```

6.5.4 メソッドの呼び出し

サーバプログラムのメソッドを呼び出します。メソッド名は、IDLで指定したモジュール名、インタフェース名、およびメソッド名を、ハイフン(-)でつなげた形式で指定します。この例では、ODDEMO、CALCULATOR、CALCULATEがそれにあたります。メソッド呼出し時にネーミングサービスで求めたサーバアプリケーションのオブジェクトリファレンスとサーバプログラムで例外が発生した場合に、例外情報を設定してもらうためにCORBA-ENVIRONMENT構造体を指定します。
なお、アプリケーションで使用可能なCORBAのデータ型については、“6.8 データ型に対するマッピング”を参照してください。

```
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARAM1.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARAM2.
   01 A-RESULT.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY ADD-RESULT.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SUBTRACT-RESULT.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY MULTIPLE-RESULT.
      02 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY DEVIDE-RESULT.

  PROCEDURE DIVISION.
   MOVE 100 TO PARAM1.
   MOVE  20 TO PARAM2.

   CALL "ODDEMO-CALCULATOR-CALCULATE" USING
         OBJ
         PARAM1
         PARAM2
         ENV
         A-RESULT.
```

```
MOVE "ODdemo_calculator_calculate" TO MESS.
PERFORM ENV-CHECK.
```

6.6 クライアントアプリケーションの例外処理

クライアントアプリケーションは、サーバアプリケーションの処理が正常終了したか、異常終了したかを知ることができます。また、異常終了した場合は、システムで異常終了したか、サーバアプリケーションで異常終了したかを知ることができます。前者をシステム例外、後者をユーザ例外と呼びます。
メソッド呼出し時に指定したCORBA-ENVIRONMENT構造体にエラー情報が格納されます。CORBA-ENVIRONMENT構造体は、COBOL登録集ENVIRONMENT.cblを参照してください。
MAJORには、以下の値が設定されます。

- CORBA-NO-EXCEPTION:正常終了
- CORBA-SYSTEM-EXCEPTION:システム例外
- CORBA-USER-EXCEPTION:ユーザ例外

(1)システム例外

次に示すようなシステム例外の値が設定されます。システム例外で通知されるデータ型は文字列であり、文字列をCORBA-EXCEPTION-ID関数でCORBA-ENVIRONMENT構造体から取り出します。
システム例外を以下に示します。例外の意味については、“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。例外の文字列は、COBOL登録集(EX-で始まる原文)を参照してください。

例外情報	例外コード
BAD_CONTEXT	EX-CORBA-STEXCEP-BAD-CONTEXT
BAD_INV_ORDER	EX-CORBA-STEXCEP-BAD-INV-ORDER
BAD_OPERATION	EX-CORBA-STEXCEP-BAD-OPERATION
BAD_PARAM	EX-CORBA-STEXCEP-BAD-PARAM
BAD_QOS	EX-CORBA-STEXCEP-BAD-QOS
BAD_TYPECODE	EX-CORBA-STEXCEP-BAD-TYPECODE
CODESET_INCOMPATIBLE	EX-CORBA-STEXCEP-CODESET-INCOM(注)
COMM_FAILURE	EX-CORBA-STEXCEP-COMM-FAILURE
DATA_CONVERSION	EX-CORBA-STEXCEP-DATA-CONVERSI(注)
FREE_MEM	EX-CORBA-STEXCEP-FREE-MEM
IMP_LIMIT	EX-CORBA-STEXCEP-IMP-LIMIT
INITIALIZE	EX-CORBA-STEXCEP-INITIALIZE
INTERNAL	EX-CORBA-STEXCEP-INTERNAL
INTF_REPOS	EX-CORBA-STEXCEP-INTF-REPOS
INV_FLAG	EX-CORBA-STEXCEP-INV-FLAG
INV_IDENT	EX-CORBA-STEXCEP-INV-IDENT
INV_OBJREF	EX-CORBA-STEXCEP-INV-OBJREF
INV_POLICY	EX-CORBA-STEXCEP-INV-POLICY
MARSHAL	EX-CORBA-STEXCEP-MARSHAL
NO_IMPLEMENT	EX-CORBA-STEXCEP-NO-IMPLEMENT
NO_MEMORY	EX-CORBA-STEXCEP-NO-MEMORY
NO_PERMISSION	EX-CORBA-STEXCEP-NO-PERMISSION

例外情報	例外コード
NO_RESOURCES	EX-CORBA-STEXCEP-NO-RESOURCES
NO_RESPONSE	EX-CORBA-STEXCEP-NO-RESPONSE
OBJ_ADAPTER	EX-CORBA-STEXCEP-OBJ-ADAPTER
PERSIST_STORE	EX-CORBA-STEXCEP-PERSIST-STORE
REBIND	EX-CORBA-STEXCEP-REBIND
TIMEOUT	EX-CORBA-STEXCEP-TIMEOUT
TRANSIENT	EX-CORBA-STEXCEP-TRANSIENT
UNKNOWN	EX-CORBA-STEXCEP-UNKNOWN
INVALID_TRANSACTION	EX-CORBA-STEXCEP-INVALID-TRANS(注)
TRANSACTION_MODE	EX-CORBA-STEXCEP-TRANSACTION-M(注)
TRANSACTION_REQUIRED	EX-CORBA-STEXCEP-TRANSACTION-R(注)
TRANSACTION_ROLLEDBACK	EX-CORBA-STEXCEP-TRANSACTION-R(注)
TRANSACTION_UNAVAILABLE	EX-CORBA-STEXCEP-TRANSACTION-U(注)

注)COBOLの30文字制限のため、短縮名となります。

システム例外発生時は、CORBA-ENVIRONMENT構造体のMINORにマイナーコードが設定されます。マイナーコードの値については、“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。

マイナーコードを表示する場合は、CORBA-ENVIRONMENT構造体のMINORを作業領域に転送してから表示させてください。

(2)ユーザ例外

ユーザ例外でも文字列でエラー情報が通知されるため、CORBA-EXCEPTION-ID関数でCORBA-ENVIRONMENT構造体から取り出します。また、IDLで定義したユーザ定義情報の詳細については、CORBA-EXCEPTION-VALUEでCORBA-ENVIRONMENT構造体から取り出します。

(3)例外情報の獲得

例外情報を獲得する例を以下に示します。

```
WORKING-STORAGE SECTION.
01 WK-ENV PIC 9(10) COMP-5.
LINKAGE SECTION.
01 COPY ENVIRONMENT  IN  CORBA  REPLACING  CORBA-ENVIRONMENT  BY  ENV.

* メソッドの呼び出し
     MOVE 100 TO PARAM-1.
     MOVE 20 TO PARAM-2.
     CALL "ODDEMO-CALCULATE-CALCULATE" USING
           OBJ
           PARAM-1
           PARAM-2
           ENV
           RESULT.
     EVALUATE TRUE
           WHEN CORBA-NO-EXCEPTION OF MAJOR OF ENV
                CONTINUE
* システム例外のエラー処理
           WHEN CORBA-SYSTEM-EXCEPTION OF MAJOR OF ENV
                MOVE FUNCTION LENG (ID) TO STRING-LENGTH
```

```
                CALL "CORBA-STRING-GET" USING
                      IDL-ID OF ENV
                      STRING-LENGTH
                      ID
* マイナーコード取得処理
                      MOVE MINOR OF ENV TO WK-ENV
                      DISPLAY "MINOR OF ENV :" MINOR OF WK-ENV.
                     ...
* ユーザ例外のエラー処理
           WHEN CORBA-USER-EXCEPTION OF MAJOR OF ENV
                MOVE FUNCTION LENG (UEXC) TO STRING-LENGTH
                CALL "CORBA-STRING-GET" USING
                      IDL-ID OF ENV
                      STRING-LENGTH
                      UEXC
      END-EVALUATE.
```

6.7 アプリケーションのテスト

作成したアプリケーションのテスト方法を、以下の内容で説明します。

- サーバアプリケーションのテスト方法
- クライアントアプリケーションのテスト方法

6.7.1 サーバアプリケーションのテスト方法

サーバアプリケーションのテストを行う場合、実際にクライアントアプリケーションと結合して行います。このとき、サーバアプリケーションをデバッガ配下で動作させることで、サーバアプリケーションが正しく作成されているかを確認できます。
COBOLデバッガと連携するときの動作の概要と、サーバアプリケーションをデバッガ配下でテストする場合の手順を、以下に示します。

テスト用モジュールの作成

Windows32/64

テストを行うサーバアプリケーションを、TESTオプションを指定してコンパイルします。リンクする場合は、リンクオプション“/DEBUG”、および“/DEBUGTYPE:COFF”を指定します。コンパイル方法の詳細については、COBOLのマニュアルを参照してください。

Solaris32/64

テストを行うサーバアプリケーションを、TESTオプションを指定してコンパイルします。コンパイル方法の詳細については、COBOLのマニュアルを参照してください。
テスト用モジュールの作成例を以下に示します。

```
CORBA=/opt/FSUNod/include/COBOL
export CORBA
cobol -WC,"TEST" -M -c simple_s.cbl
cobol -L/opt/FSUNod/lib -lOMcbl -o simple_s simple_s.o
cobol -WC,"TEST" -c simple_sa.cbl simple_ODdemo_calculator_skel.cbl simple_skel.cbl simple_cdr.cbl
cobol -G -dy -o libODDEMO-CALCULATOR.so -L/opt/FSUNod/lib -lOMcbl
      simple_sa.o simple_ODdemo_calculator_skel.o simple_skel.o simple_cdr.o
```

Linux32/64

テストを行うサーバアプリケーションを、TESTオプションを指定してコンパイルします。コンパイル方法の詳細については、COBOLのマニュアルを参照してください。
テスト用モジュールの作成例を以下に示します。

```
CORBA=/opt/FJSVod/include/COBOL
export CORBA
cobol -WC,"TEST" -M -c simple_s.cbl
```

```
cobol -L/opt/FJSVod/lib -lOMcbl -o simple_s simple_s.o
cobol -WC,"TEST" -c simple_sa.cbl simple_ODdemo_calculator_skel.cbl simple_skel.cbl simple_cdr.cbl
cobol -G -dy -o libODDEMO-CALCULATOR.so -L/opt/FJSVod/lib -lOMcbl
      simple_sa.o simple_ODdemo_calculator_skel.o simple_skel.o simple_cdr.o
```

COBOLデバッガの起動 Windows32/64

コマンドプロンプトからwinsvdコマンドでCOBOLデバッガを起動し、サーバアプリケーションをCOBOLデバッガ配下で起動します。サーバアプリケーションをCOBOLデバッガ配下で起動する際のwinsvdコマンドのパラメタのうち、基本的なパラメタの指定方法を以下に示します。

```
winsvd  /G開始プログラム名  /Sソースファイル格納フォルダ名 /Dデバッグ情報格納フォルダ名 サーバアプリケーション名
```

/G開始プログラム名

デバッグを開始する外部プログラム名を指定します。

/Sソースファイル格納ディレクトリ名

COBOLソースプログラムの格納フォルダを指定します。

/Dデバッグ情報格納ディレクトリ名

デバッグ情報ファイルの格納フォルダを指定します。デバッグ情報ファイルは、コンパイル時に生成される拡張子が“.svd”のファイルです。

サーバアプリケーション名

アプリケーション名を指定します。

COBOLデバッガの起動 Solaris32/64 Linux32/64

端末から、テスト対象のCOBOLプログラムを指定したsvdコマンドにより、COBOLデバッガを起動してサーバアプリケーションをそのデバッガ配下で起動します。サーバアプリケーションを起動する際のsvdコマンドのパラメタのうち、基本的なパラメタの指定方法を以下に示します。

```
svd  -p 開始プログラム名  -s ソースファイル格納ディレクトリ名 -k デバッグ情報格納ディレクトリ名 サーバアプリケーション名
```

-p 開始プログラム名

デバッグを開始する外部プログラム名を指定します。

-s ソースファイル格納ディレクトリ名

COBOLソースプログラムの格納ディレクトリを指定します。

-k デバッグ情報格納ディレクトリ名

デバッグ情報ファイルの格納ディレクトリを指定します。デバッグ情報ファイルは、コンパイル時に生成される拡張子が“.svd”のファイルです。

サーバアプリケーション名

サーバアプリケーションを指定します。

サーバアプリケーションの起動例を以下に示します。

Solaris32/64

```
% PATH=/opt/FJSVCOBop/bin:${PATH}; export PATH
% LD_LIBRARY_PATH=/opt/FJSVCOBop/lib:/usr/dt/lib:/usr/openwin/lib:${LD_LIBRARY
  _PATH};export LD_LIBRARY_PATH
```

```
% MANPATH=/opt/FJSVCOBop/man/%L:/opt/FJSVCOBop/man:${MANPATH}; export MANPATH
% NLSPATH=/opt/FJSVCOBop/lib/nls/%1/%c/%N.cat:/opt/FJSVCOBop/lib/nls/C/%N.cat; export NLSPATH
% XUSERFILESEARCHPATH=/opt/FJSVCOBop/lib/app-defaults/%L/%N:/opt/FJSVCOBop/lib
  /app-defaults/%N; export XUSERFILESEARCHPATH
% svd  -p SERVER-MAIN  -s /tmp/samples/CalcSample/COBOL -k /tmp/samples/CalcSample/COBOL simple_s
```

Linux32/64

```
% PATH=/opt/FJSVcbl/bin:${PATH}; export PATH
% LD_LIBRARY_PATH=/opt/FJSVcbl/lib:/usr/dt/lib:/usr/openwin/lib:${LD_LIBRARY
  _PATH};export LD_LIBRARY_PATH
% MANPATH=/opt/FJSVcbl/man/%L:/opt/FJSVcbl/man:${MANPATH}; export MANPATH
% NLSPATH=/opt/FJSVcbl/lib/nls/%1/%c/%N.cat:/opt/FJSVcbl/lib/nls/C/%N.cat; export NLSPATH
% XUSERFILESEARCHPATH=/opt/FJSVcbl/lib/app-defaults/%L/%N:/opt/FJSVcbl/lib
  /app-defaults/%N; export XUSERFILESEARCHPATH
% svd  -p SERVER-MAIN  -s /tmp/samples/CalcSample/COBOL -k /tmp/samples/CalcSample/COBOL simple_s
```

COBOLデバッガの詳細については、COBOLのマニュアルを参照してください。

テストの実施

COBOLデバッガを起動すると、COBOLデバッガ画面が表示されます。画面の表示後、クライアントアプリケーションからサーバアプリケーションを呼び出して、処理を実行することで、サーバアプリケーションの動作状態をデバッガから確認できます。これによりアプリケーションを実行でき、ステップ単位でデバッグできます。なお、COBOLプログラムのデバッグ方法の詳細については、COBOLのマニュアルを参照してください。

Windows32/64

インタフェース実装関数をデバッグする場合は、winsvdコマンドの/Gオプションにインタフェース実装関数のプログラム名を指定してください。

Solaris32/64 Linux32/64

インタフェース実装関数をデバッグする場合は、svdコマンドの-pオプションにインタフェース実装関数のプログラム名を指定してください。

COBOLデバッガにより、サーバアプリケーションをデバッグする場合の注意事項について示します。

- サーバアプリケーションのデバッグ中にアプリケーションを停止しないでください。必ずサーバアプリケーションのEXIT PROGRAMを実行し、デバッガに“実行中”が表示されてからサーバアプリケーションを停止してください。
- 1つのサーバアプリケーションに対して、デバッガは1つだけ起動するようにしてください。
- デバッガを終了する場合は、必ずデバッガによる実行状態が完了している状態で行ってください。

6.7.2 クライアントアプリケーションのテスト方法

クライアントアプリケーションのテスト方法は、クライアントアプリケーションを動作させるオペレーティングシステムやミドルウェアによって異なります。使用しているオペレーティングシステムやミドルウェアごとに推奨される方法でテストを行ってください。

6.8 データ型に対するマッピング

COBOLでは、CORBAのデータ型を扱うために、以下の登録集を提供します。

- 文字定数用:CONST.cbl
- SYMBOL IC CONSTANT用:SYMBOL-CONST.cbl
- その他データ型:データ型名.cbl

データ型のCOBOLでの定義を以下に示します。

CORBAデータ型			COBOLでの定義		備考
基本データ型	整数型	long	CORBA-LONG	PIC S9(9) COMP-5.	
		short	CORBA-SHORT	PIC S9(4) COMP-5.	
		unsigned long	CORBA-UNSIGNED-LONG	PIC 9(9) COMP-5.	
		unsigned short	CORBA-UNSIGNED-SHORT	PIC 9(4) COMP-5.	
		long long	CORBA-LONG-LONG	PIC S9(18) COMP-5.	
	浮動小数点型	float	CORBA-FLOAT	COMP-1.	
		double	CORBA-DOUBLE	COMP-2.	
	文字型	char wchar	CORBA-CHAR CORBA-WCHAR	PIC X(1). PIC N(1).	
	オクテッド	octet	CORBA-OCTET	PIC 9(4) COMP-5.	
	ブーリアン	boolean	CORBA-BOOLEAN	PIC 9(9) COMP-5.	
	文字列型	string		PIC X(n).	“6.8.1 文字列型”参照
		wstring		PIC N(n).	“6.8.2 ワイド文字列型”参照
	列挙型	enum	CORBA-ENUM	PIC 9(9) COMP-5.	
	any型	any		集団項目	“6.8.3 any型”参照
シーケンス型		sequence		OCCURS句	“6.8.4 シーケンス型”参照
構造体		struct		集団項目	“6.8.5 構造体”参照
共用体		union		REDEFINES文節	“6.8.6 共用体”参照
固定小数点型		fixed		PIC xx(n) PACKED-DECIMAL.	“6.8.7 固定小数点型”参照
配列		array		OCCURS句	“6.8.8 配列”参照
オブジェクトリファレンス		Object	CORBA-OBJECT	USAGE POINTER	
タイプコード		TypeCode	CORBA-TYPECODE	USAGE POINTER	

注意

- **整数型を扱う場合**
  CORBA-SHORT、CORBA-UNSIGNED-SHORTで扱う値は2バイトで管理されますが、そのまま表示すると4桁以上の値は表示されません。符号を含めて4桁以上の値を表示する場合は、いったんCORBA-LONGなどの変数に代入してから扱ってください。
  また、5桁以上の値が代入されていると、コンパイル時に桁あふれの警告が表示されます。5桁以上の値を使用する場合は、CORBA-LONG型を使用してください。
  同様に、CORBA-LONG、CORBA-UNSIGNED-LONGに対して9桁以上の値を代入する場合は、桁あふれになるので注意してください。

- **共用体の翻訳時の注意事項**
  共用体のメンバにデータの長さの異なるものが存在した場合、COBOL翻訳時に、JMN2232I-Wのメッセージが表示されます。

CORBAサービスでは、共用体のサンプルとしてlong型とstring型の長さの異なるサンプルを提供しています。このサンプルの翻訳時に、JMN2232I-Wのメッセージが出力されます。

6.8.1 文字列型

(1)IDLマッピング

IDL言語で文字列型stringを指定した場合、COBOLでは英数字項目、またはPOINTERでデータ宣言します。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
typedef string<10> t_bound_string;
module ODsample{ //文字列（固定長）
    interface  stringtest{
        t_bound_string op1(in t_bound_string str1, out t_bound_string str2, inout t_bound_string str3);
    };
};

module ODsample{ //文字列（可変長）
    interface  stringtest{
        string op1(in string str1, out string str2, inout string str3);
    };
};
```

COBOL

```
 * 文字列(固定長)
  IDENTIFICATION DIVISION.
  PROGRAM-ID. "ODSAMPLE-STRINGTEST-OP1".
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
 *オブジェクトリファレンス
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 * inパラメタ
   01 STR1 PIC X(10).
 * outパラメタ
   01 STR2 PIC X(10).
 * inoutパラメタ
   01 STR3 PIC X(10).
 *例外情報
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 *復帰値
   01 RET PIC X(10).

  PROCEDURE DIVISION.

   CALL "ODSAMPLE-STRINGTEST-OP1" USING
         OBJ
         STR1
         STR2
         STR3
         ENV
         RET.

 * 文字列(可変長)
  IDENTIFICATION DIVISION.
```

```
 PROGRAM-ID. "ODSAMPLE-STRINGTEST-OP1".
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 COPY CONST IN CORBA.
*オブジェクトリファレンス
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
* inパラメタ
  01 STR1 USAGE POINTER.
* outパラメタ
  01 STR2 USAGE POINTER.
* inoutパラメタ
  01 STR3 USAGE POINTER.
*例外情報
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
*復帰値
  01 RET  USAGE POINTER.

 PROCEDURE DIVISION.

  CALL "ODSAMPLE-STRINGTEST-OP1" USING
        OBJ
        STR1
        STR2
        STR3
        ENV
        RET.
```

(2)クライアントアプリケーションで扱うパラメタ(固定長)

文字列(固定長)のパラメタ(in、out、inout)を扱う場合、領域の獲得／解放を行う必要はありません。データ域にデータを設定します。

クライアントアプリケーションでのDATA DIVISIONおよびPROCEDURE DIVISIONの処理例を以下に示します。

```
DATA DIVISION.
   WORKING-STORAGE SECTION.
   COPY CONST IN CORBA.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
    01 STR1 PIC X(10).
    01 STR2 PIC X(10).
    01 STR3 PIC X(10).
    01 RET PIC X(10).
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.

 PROCEDURE DIVISION.
* inパラメタの設定
    MOVE "IN" TO STR1.
* inoutパラメタの設定
    MOVE "INOUT:1"  TO STR3.
    CALL "ODSAMPLE-STRINGTEST-OP1" USING
          OBJ
          STR1
          STR2
          STR3
          ENV
          RET.
```

(3)クライアントアプリケーションで扱うパラメタ(可変長)

クライアントアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	CORBA-STRING-SET関数を使用して、文字列+終端文字'¥0'分の領域獲得と文字列設定を行います。	－
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。
out 復帰値	－	(inoutパラメタと同じ)

注意

クライアントおよびスタブで獲得した領域は、不要になった時点でCORBA-FREE関数で解放する必要があります。

クライアントアプリケーションでのDATA DIVISIONおよびPROCEDURE DIVISIONの処理例を以下に示します。

```
 DATA DIVISION.
    WORKING-STORAGE SECTION.
    COPY CONST IN CORBA.
     01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
     01 STR1 USAGE POINTER.
     01 STR2 USAGE POINTER.
     01 STR3 USAGE POINTER.

     01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
     01 RET  USAGE POINTER.
     01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
     01 STR-WORK PIC X(30).

  PROCEDURE DIVISION.
* inパラメタの設定
     MOVE "IN" TO STR-WORK.
     MOVE 3 TO LSIZE.
     CALL "CORBA-STRING-SET" USING
           STR1
           LSIZE
           STR-WORK.
* inoutパラメタの設定
     MOVE "INOUT:1"  TO STR-WORK.
     MOVE 8 TO LSIZE.
     CALL "CORBA-STRING-SET" USING
           STR3
           LSIZE
           STR-WORK.
     CALL "ODSAMPLE-STRINGTEST-OP1" USING
           OBJ
           STR1
           STR2
           STR3
           ENV
           RET.

* 復帰値用領域の解放

     CALL "CORBA-FREE" USING
           RET.
* inパラメタ用領域の解放
     CALL "CORBA-FREE" USING
          STR1.
```

```
* outパラメタ用領域の解放
    CALL "CORBA-FREE" USING
         STR2.
* inoutパラメタ用領域の解放
    CALL "CORBA-FREE" USING
         STR3.
```

(4)サーバアプリケーションで扱うパラメタ(固定長)

文字列(固定長)のパラメタ(in、out、inout)を扱う場合、領域の獲得／解放を行う必要はありません。データ域にデータを設定します。

サーバアプリケーションでの処理例を以下に示します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-STRINGTEST-OP1".
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
    SYMBOLIC CONSTANT
    COPY SYMBOL-CONST IN CORBA.
    .
 DATA DIVISION.
 WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR-WORK PIC X(30).
 LINKAGE SECTION.
* オブジェクトリファレンス
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
* inパラメタ
  01 STR1 PIC X(10).
* outパラメタ
  01 STR2 PIC X(10).
* inoutパラメタ
  01 STR3 PIC X(10).
  01 STR PIC X(10).
* 例外情報
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.

 PROCEDURE DIVISION USING OBJ STR1 STR2 STR3 ENV STR.

 MAIN.

* outパラメタの処理
  MOVE "OUT" TO STR2.

* inoutパラメタの処理
  MOVE "INOUT:2" TO STR3.

* 復帰値の処理
  MOVE "RETURN" TO STR.

  MAIN-END.
 END PROGRAM "ODSAMPLE-STRINGTEST-OP1".
```

(5)サーバアプリケーションで扱うパラメタ(可変長)

サーバアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
in	文字列域は、スケルトンで自動的に獲得/解放されます。	―
inout	文字列域は、スケルトンで自動的に獲得されます。	・ 渡されたパラメタより短い文字列を返す場合: 渡された文字列域に文字列を設定します。 ・ 渡されたパラメタより長い文字列を返す場合: 渡された文字列域をCORBA-FREE関数で一度解放し、CORBA-STRING-SET関数で領域獲得/文字列設定します。 文字列域は、スケルトンで自動的に解放されます。
out 復帰値	―	CORBA-STRING-SET関数で領域獲得/文字列設定します。 文字列域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-STRINGTEST-OP1".
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
    SYMBOLIC CONSTANT
    COPY SYMBOL-CONST IN CORBA.
    .
 DATA DIVISION.
 WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR-WORK PIC X(30).
 LINKAGE SECTION.
* オブジェクトリファレンス
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
* inパラメタ
  01 STR1 USAGE IS POINTER.
* outパラメタ
  01 STR2 USAGE IS POINTER.
* inoutパラメタ
  01 STR3 USAGE IS POINTER.
  01 STR USAGE IS POINTER.
* 例外情報
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.

 PROCEDURE DIVISION USING OBJ STR1 STR2 STR3 ENV STR.

 MAIN.

* outパラメタの処理
  MOVE "OUT" TO STR-WORK.
  MOVE 4 TO LSIZE.
* outパラメタの設定
  CALL "CORBA-STRING-SET" USING
        STR2
        LSIZE
        STR-WORK.

* inoutパラメタの処理
* クライアントから渡された領域の解放
  CALL "CORBA-FREE" USING
         STR3.
```

```
  MOVE 8 TO LSIZE.
* 出力パラメタの設定
  MOVE "INOUT:2" TO STR-WORK.
  CALL "CORBA-STRING-SET" USING
        STR3
        LSIZE
        STR-WORK.

* 復帰値の処理
  MOVE 7 TO LSIZE.
* 復帰値の設定
  MOVE "RETURN" TO STR-WORK.
  CALL "CORBA-STRING-SET" USING
        STR
        LSIZE
        STR-WORK.

  MAIN-END.
 END PROGRAM "ODSAMPLE-STRINGTEST-OP1".
```

6.8.2 ワイド文字列型

(1)IDLマッピング

IDL言語でワイド文字列型wstringを指定した場合、COBOLでは日本語項目、またはPOINTERでデータ宣言します。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
typedef wstring<10> t_bound_wstring;
module ODsample{ //ワイド文字列（固定長）
    interface  wstringtest{
        t_bound_wstring op1(in t_bound_wstring str1, out t_bound_wstring str2, inout t_bound_wstring str3);
    };
};

module ODsample{
    interface  wstringtest{
        wstring op1(in wstring str1, out wstring str2, inout wstring str3);
    };
};
```

COBOL

```
  * 文字列(固定長)
   IDENTIFICATION DIVISION.
   PROGRAM-ID. "ODSAMPLE-WSTRINGTEST-OP1".
   ENVIRONMENT DIVISION.
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   COPY CONST IN CORBA.
  *オブジェクトリファレンス
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  * inパラメタ
    01 STR1 PIC N(10).
  * outパラメタ
    01 STR2 PIC N(10).
  * inoutパラメタ
    01 STR3 PIC N(10).
  *例外情報
```

```
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 *復帰値
   01 RET  PIC N(10).

  PROCEDURE DIVISION.

   CALL "ODSAMPLE-WSTRINGTEST-OP1" USING
         OBJ
         STR1
         STR2
         STR3
         ENV
         RET.

 * 文字列(可変長)
  IDENTIFICATION DIVISION.
  PROGRAM-ID. "ODSAMPLE-WSTRINGTEST-OP1".
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
 *オブジェクトリファレンス
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 * inパラメタ
   01 STR1 USAGE POINTER.
 * outパラメタ
   01 STR2 USAGE POINTER.
 * inoutパラメタ
   01 STR3 USAGE POINTER.
 *例外情報
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 *復帰値
   01 RET  USAGE POINTER.

  PROCEDURE DIVISION.

   CALL "ODSAMPLE-WSTRINGTEST-OP1" USING
         OBJ
         STR1
         STR2
         STR3
         ENV
         RET.
```

(2)クライアントアプリケーションで扱うパラメタ(固定長)

文字列(固定長)のパラメタ(in、out、inout)を扱う場合、領域の獲得／解放を行う必要はありません。データ域にデータを設定します。

クライアントアプリケーションでのDATA DIVISIONおよびPROCEDURE DIVISIONの処理例を以下に示します。

```
DATA DIVISION.
   WORKING-STORAGE SECTION.
   COPY CONST IN CORBA.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
    01 STR1 PIC N(10).
    01 STR2 PIC N(10).
    01 STR3 PIC N(10).
    01 RET PIC N(10).
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.

   PROCEDURE DIVISION.
```

```
* inパラメタの設定
    MOVE N"イン" TO STR1.
* inoutパラメタの設定
    MOVE N"インアウト" TO STR3.
    CALL "ODSAMPLE-WSTRINGTEST-OP1" USING
          ORB
          STR1
          STR2
          STR3
          ENV
          RET.
```

(3)クライアントアプリケーションで扱うパラメタ(可変長)

クライアントアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	CORBA-WSTRING-SET関数を使用して、文字列+終端文字'¥0¥0'分の領域獲得と文字列設定を行います。	―
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。
out 復帰値	―	(inoutパラメタと同じ)

注意

クライアントおよびスタブで獲得した領域は、不要になった時点でCORBA-FREE関数で解放する必要があります。

クライアントアプリケーションでのDATA DIVISIONおよびPROCEDURE DIVISIONの処理例を以下に示します。

```
DATA DIVISION.
   WORKING-STORAGE SECTION.
   COPY CONST IN CORBA.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
    01 STR1 USAGE POINTER.
    01 STR2 USAGE POINTER.
    01 STR3 USAGE POINTER.

    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
    01 RET  USAGE POINTER.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
    01 STR-WORK PIC N(30).

   PROCEDURE DIVISION.
* inパラメタの設定
    MOVE N"イン" TO STR-WORK.
    COMPUTE LSIZE = FUNCTION LENGTH(STR-WORK).
    CALL "CORBA-WSTRING-SET" USING
          STR1
          LSIZE
          STR-WORK.
* inoutパラメタの設定
    MOVE N"インアウト" TO STR-WORK.
    COMPUTE LSIZE = FUNCTION LENGTH(STR-WORK).
    CALL "CORBA-WSTRING-SET" USING
          STR3
          LSIZE
          STR-WORK.
```

```
    CALL "ODSAMPLE-WSTRINGTEST-OP1" USING
          OBJ
          STR1
          STR2
          STR3
          ENV
          RET.

* 復帰値用領域の解放

    CALL "CORBA-FREE" USING
          RET.
* inパラメタ用領域の解放
    CALL "CORBA-FREE" USING
          STR1.
* outパラメタ用領域の解放
    CALL "CORBA-FREE" USING
          STR2.
* inoutパラメタ用領域の解放
    CALL "CORBA-FREE" USING
          STR3.
```

(4)サーバアプリケーションで扱うパラメタ(固定長)

文字列(固定長)のパラメタ(in、out、inout)を扱う場合、領域の獲得／解放を行う必要はありません。データ域にデータを設定します。

サーバアプリケーションでの処理例を以下に示します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-WSTRINGTEST-OP1".
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
   SYMBOLIC CONSTANT
    COPY SYMBOL-CONST IN CORBA.
    .
 DATA DIVISION.
 WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR-WORK PIC N(30).
 LINKAGE SECTION.
* オブジェクトリファレンス
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
* inパラメタ
  01 STR1 PIC N(10).
* outパラメタ
  01 STR2 PIC N(10).
* inoutパラメタ
  01 STR3 PIC N(10).
  01 STR PIC N(10).
* 例外情報
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.

 PROCEDURE DIVISION USING OBJ STR1 STR2 STR3 ENV STR.

 MAIN.

* outパラメタの処理
  MOVE N"アウト" TO STR2.

* inoutパラメタの処理
```

```
  MOVE N"インアウト" TO STR3.

* 復帰値の処理
  MOVE N"リターン" TO STR.

  MAIN-END.
 END PROGRAM "ODSAMPLE-WSTRINGTEST-OP1".
```

(5)サーバアプリケーションで扱うパラメタ

サーバアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
in	文字列域は、スケルトンで自動的に獲得/解放されます。	－
inout	文字列域は、スケルトンで自動的に獲得されます。	・ 渡されたパラメタより短い文字列を返す場合: 渡された文字列域に文字列を設定します。 ・ 渡されたパラメタより長い文字列を返す場合: 渡された文字列域をCORBA-FREE関数で一度解放し、CORBA-WSTRING-SET関数で領域獲得/文字列設定します。 文字列域は、スケルトンで自動的に解放されます。
out 復帰値	－	CORBA-WSTRING-SET関数で領域獲得/文字列設定します。 文字列域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-WSTRINGTEST-OP1".
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
   SYMBOLIC CONSTANT
    COPY SYMBOL-CONST IN CORBA.
    .
 DATA DIVISION.
 WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR-WORK PIC N(30).
 LINKAGE SECTION.
* オブジェクトリファレンス
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
* inパラメタ
  01 STR1 USAGE IS POINTER.
* outパラメタ
  01 STR2 USAGE IS POINTER.
* inoutパラメタ
  01 STR3 USAGE IS POINTER.
  01 STR USAGE IS POINTER.
* 例外情報
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.

 PROCEDURE DIVISION USING OBJ STR1 STR2 STR3 ENV STR.

 MAIN.
```

```
* outパラメタの処理
  MOVE N"アウト" TO STR-WORK.
  COMPUTE LSIZE = FUNCTION LENGTH(STR-WORK).
* outパラメタの設定
  CALL "CORBA-WSTRING-SET" USING
        STR2
        LSIZE
        STR-WORK.

* inoutパラメタの処理
* クライアントから渡された領域の解放
  CALL "CORBA-FREE" USING
         STR3.
* 出力パラメタの設定
  MOVE N"インアウト" TO STR-WORK.
  COMPUTE LSIZE = FUNCTION LENGTH(STR-WORK).
  CALL "CORBA-WSTRING-SET" USING
        STR3
        LSIZE
        STR-WORK.

* 復帰値の処理
  MOVE N"リターン" TO STR-WORK.
  COMPUTE LSIZE = FUNCTION LENGTH(STR-WORK).
* 復帰値の設定
  CALL "CORBA-WSTRING-SET" USING
        STR
        LSIZE
        STR-WORK.

  MAIN-END.
 END PROGRAM "ODSAMPLE-WSTRINGTEST-OP1".
```

(6) エンディアンについて Linux64

コード系がUCS2のデータをワイド文字列の変数に格納した場合、デフォルトではCPUのエンディアンに従ってメモリ上にデータが格納されます。CORBA通信を行うだけの場合、メモリ上のデータがビッグエンディアンとリトルエンディアンのどちらの形式で格納されているかを意識する必要はありません。しかし、メモリ上のデータを独自通信のデータとして扱う場合など、CPUのエンディアンとは異なるエンディアンでメモリ上にデータを格納したい場合があります。このとき、COBOLコンパイラのRCSオプションを指定することにより、エンディアンの指定が可能となります。
なお、CPUのエンディアンとは異なるエンディアンを扱う場合は、そのエンディアン用のライブラリをリンクする必要があります。リンクするライブラリについては、“B.1.2 ライブラリ”を参照してください。また、翻訳／リンク手順の例については、“6.10 COBOLアプリケーション使用時の注意事項”を参照してください。

6.8.3 any型

(1)IDLマッピング

IDL言語でany型を指定した場合、COBOLでは集団項目として以下の要素（ANY-TYPE, ANY-VALUE）をPOINTERで宣言します。また、データ域(ANY-VALUE)の獲得関数（関数名：“モジュール名-インタフェース名-構造体名-ALLOC”。以降XX-ALLOC関数と呼ぶ）がIDLコンパイラで生成されます。

```
  CORBA-ANY.
* データ型を識別するTypeCode情報
 49  ANY-TYPE   USAGE POINTER.
* データ域
 49  ANY-VALUE  USAGE POINTER.
```

以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    struct sample1 {
        long    para1;
        string  para2;
    };
    struct sample2 {
        char    para1;
        float   para2;
    };
    struct sample3 {
        char    para1;
        double  para2;
    };
    interface  anytest{
        any     op1(in any any1, out any any2, inout any any3 );
    };
};
```

(2)クライアントアプリケーションで扱うパラメタ

クライアントアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	ANY-TYPEにTypeCode情報(“TC-モジュール名-インタフェース名-データ名-IMPL-SEQ”、IDLファイル名_h.cblで定義)を、ANY-VALUEにデータ域(XX-ALLOC関数で獲得)のポインタを設定します。 領域(CORBA-ANY/ANY-VALUE域)を動的に獲得する場合は、CORBA-ANY-ALLOC関数/XX-ALLOC関数を使用します。	－
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。
out 復帰値	－	(inoutパラメタと同じ)

注意

クライアントおよびスタブで獲得した領域は、不要になった時点でCORBA-FREE関数で解放する必要があります。このとき、ANY-VALUEの指す領域を解放するかどうかを、リリースフラグで指定できます。

リリースフラグの参照／設定は、以下の関数／フラグを使用します。

[関数]

```
CALL "CORBA-ANY-GET-RELEASE" USING
      BUFFER
      RET-VAL.
CALL "CORBA-ANY-SET-RELEASE" USING
      BUFFER
      CORBA-TRUE-VALUE.
```

[フラグ]

- CORBA-TRUE-VALUE:CORBA-FREE関数発行時、ANY-VALUEの指す領域も解放されます。
- CORBA-FALSE-VALUE:CORBA-FREE関数発行時、ANY-VALUEの指す領域は解放されません(デフォルト)。

なお、スタブで獲得されたoutパラメタ、復帰値のリリースフラグは、CORBA-TRUE-VALUEに設定されます。

クライアントアプリケーションの処理例を以下に示します。

```
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
   01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
   01 COPY ANY IN CORBA REPLACING CORBA-ANY BY SMP1.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY T-LONG.
   01 TYPE-P USAGE POINTER.
   01 TC-ODSAMPLE-SAMPLE2-IMPL-SEQ.
      03MEMBER1 PIC X(100) VALUE "69,69,0,0,0,0,15,4,0,8,0,25,73,68,76,58,79,68,
                           115,97,109,112,108,101,47,115,97,109,112,108,101,50,58".
      03 MEMBER2 PIC X(100) VALUE ",49,46,48,0,0,8,115,97,109,112,108,101,50,0,0,2,
                           0,6,112,97,114,97,49,0,0,1,9,0,6,112,97,114,97,50,0,".
      03 MEMBER3 PIC X(100) VALUE "0,1,6".
   01 TC-ODSAMPLE-SAMPLE3-IMPL-SEQ.
      03 MEMBER1 PIC X(100) VALUE "69,69,0,0,0,0,15,8,0,16,0,25,73,68,76,58,79,68,
                            115,97,109,112,108,101,47,115,97,109,112,108,101,51,5".
      03 MEMBER2 PIC X(100) VALUE "8,49,46,48,0,0,8,115,97,109,112,108,101,51,0,0,
                            2,0,6,112,97,114,97,49,0,0,1,9,0,6,112,97,114,97,50,0".
      03 MEMBER3 PIC X(100) VALUE ",0,1,7".
   01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY1.
   01 ANY2 USAGE POINTER.
   01 ANY3 USAGE POINTER.
   01 ANYO USAGE POINTER.
  LINKAGE SECTION.
   01 SMP2.
      02 COPY CHAR IN CORBA REPLACING CORBA-CHAR BY PARA1.
      02 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY PARA2.
   01 SMP3.
      02 COPY CHAR IN CORBA REPLACING CORBA-CHAR BY PARA1.
      02 COPY DOUBLE IN CORBA REPLACING CORBA-DOUBLE BY PARA2.
   01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY2-P.
   01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY3-P.
   01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANYO-P.

 PROCEDURE DIVISION.

 *TypeCodeの設定
 MOVE FUNCTION LENG (TC-ODSAMPLE-SAMPLE3-IMPL-SEQ) TO T-LONG.
   CALL "CORBA-STRING-SET" USING
        TYPE-P
        T-LONG
        TC-ODSAMPLE-SAMPLE3-IMPL-SEQ.
   CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
        TYPE-P
        ANY-TYPE OF ANY1.
   CALL "CORBA-FREE" USING
        TYPE-P.

   CALL "ODSAMPLE-SAMPLE3-ALLOC" USING
        ANY-VALUE OF ANY1.
   SET ADDRESS OF SMP3 TO ANY-VALUE OF ANY1.
 *入力用パラメタの設定
   MOVE "a" TO PARA1 OF SMP3.
 *入力用パラメタの設定
   MOVE 0.00001 TO PARA2 OF SMP3.
```

```
*入出力パラメタ用領域の獲得
  CALL "CORBA-ANY-ALLOC" USING
       ANY3.
  SET ADDRESS OF ANY3-P TO ANY3.
*TypeCodeの設定
  MOVE FUNCTION LENG (TC-ODSAMPLE-SAMPLE2-IMPL-SEQ) TO T-LONG.
  CALL "CORBA-STRING-SET" USING
       TYPE-P
       T-LONG
       TC-ODSAMPLE-SAMPLE2-IMPL-SEQ.
  CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
       TYPE-P
       ANY-TYPE OF ANY3-P.
  CALL "CORBA-FREE" USING
       TYPE-P.

  CALL "ODSAMPLE-SAMPLE2-ALLOC" USING
       ANY-VALUE OF ANY3-P.
  SET ADDRESS OF SMP2 TO ANY-VALUE OF ANY3-P.
*入出力用パラメタの設定
  MOVE "c" TO PARA1 OF SMP2.
*入出力用パラメタの設定
  MOVE 0.0001 TO PARA2 OF SMP2.

  CALL "ODSAMPLE-ANYTEST-OP1" USING
       OBJ
       ANY1
       ANY2
       ANY3
       ENV
       ANY0.
*復帰値の領域解放
  CALL "CORBA-FREE" USING
       ANY0.
*入力パラメタの領域解放
  CALL "CORBA-FREE" USING
       ANY-VALUE OF ANY1.
*出力パラメタの領域解放
  CALL "CORBA-FREE" USING
       ANY2.
*入出力パラメタの領域解放
  CALL "CORBA-FREE" USING
       ANY3.
```

(3)サーバアプリケーションで扱うパラメタ

サーバアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
in	各領域(CORBA-ANY/ANY-VALUE域)は、スケルトンで自動的に獲得/解放されます。	－
inout	各領域(CORBA-ANY/ANY-VALUE域)は、スケルトンで自動的に獲得されます。	ANY-VALUE域をCORBA-FREE関数で一度解放し、XX-ALLOC関数で再獲得します。 領域は、スケルトンで自動的に解放されます。
out 復帰値	－	各領域(CORBA-ANY/ANY-VALUE域)をCORBA-ANY-ALLOC関数/XX-ALLOC関数で獲得します。 領域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-ANYTEST-OP1".
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 COPY CONST IN CORBA.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY T-LONG.
  01 TYPE-P USAGE POINTER.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR-WORK PIC X(20).
  01 TC-ODSAMPLE-SAMPLE1-IMPL-SEQ.
     03 MEMBER1 PIC X(100) VALUE "71,71,0,0,0,0,15,4,0,8,0,25,73,68,76,58,79,68,115,97,
        109,112,108,101,47,115,97,109,112,108,101,49,58".
     03 MEMBER2 PIC X(100) VALUE ",49,46,48,0,0,8,115,97,109,112,108,101,49,0,0,2,
        0,6,112,97,114,97,49,0,0,1,3,0,6,112,97,114,97,50,0,".
     03 MEMBER3 PIC X(100) VALUE "0,3,18,0,0".
  01 TC-ODSAMPLE-SAMPLE2-IMPL-SEQ.
     03MEMBER1 PIC X(100) VALUE "69,69,0,0,0,0,15,4,0,8,0,25,73,68,76,58,79,68,
                           115,97,109,112,108,101,47,115,97,109,112,108,101,50,58".
     03 MEMBER2 PIC X(100) VALUE ",49,46,48,0,0,8,115,97,109,112,108,101,50,0,0,2,
                           0,6,112,97,114,97,49,0,0,1,9,0,6,112,97,114,97,50,0,".
     03 MEMBER3 PIC X(100) VALUE "0,1,6".
  01 TC-ODSAMPLE-SAMPLE3-IMPL-SEQ.
     03 MEMBER1 PIC X(100) VALUE "69,69,0,0,0,0,15,8,0,16,0,25,73,68,76,58,79,68,
                            115,97,109,112,108,101,47,115,97,109,112,108,101,51,5".
     03 MEMBER2 PIC X(100) VALUE "8,49,46,48,0,0,8,115,97,109,112,108,101,51,0,0,
                            2,0,6,112,97,114,97,49,0,0,1,9,0,6,112,97,114,97,50,0".
     03 MEMBER3 PIC X(100) VALUE ",0,1,7".

 LINKAGE SECTION.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 SMP1.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 PARA2 USAGE IS POINTER.
  01 SMP2.
     02 COPY CHAR IN CORBA REPLACING CORBA-CHAR BY PARA1.
     02 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY PARA2.
  01 SMP3.
     02 COPY CHAR IN CORBA REPLACING CORBA-CHAR BY PARA1.
     02 COPY DOUBLE IN CORBA REPLACING CORBA-DOUBLE BY PARA2.
  01 SMP2 USAGE POINTER.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY1.
  01 ANY2 USAGE POINTER.
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY2-P.
  01 ANY3 USAGE POINTER.
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY3-P.
  01 ANYO USAGE POINTER.
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANYO-P.

PROCEDURE DIVISION OBJ ANY1 ANY2 ANY3 ENV ANYO.

MAIN.

*outパラメタ用領域の獲得
  CALL "CORBA-ANY-ALLOC"  USING
       ANY2.
  SET ADDRESS OF ANY2-P TO ANY2.
*TypeCodeの設定
  MOVE FUNCTION LENG (TC-ODSAMPLE-SAMPLE2-IMPL-SEQ) TO T-LONG.
```

```
  CALL "CORBA-STRING-SET" USING
      TYPE-P
      T-LONG
      TC-ODSAMPLE-SAMPLE2-IMPL-SEQ.
  CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
      TYPE-P
      ANY-TYPE OF ANY2-P.
  CALL "CORBA-FREE" USING
      TYPE-P.
*出力用パラメタデータ領域の獲得
  CALL "ODSAMPLE-SAMPLE2-ALLOC" USING
      ANY-VALUE OF ANY2-P.
  SET ADDRESS OF SMP2 TO ANY-VALUE OF ANY2-P.
*出力パラメタの設定
  MOVE "x" TO PARA1 OF SMP2.
*出力パラメタの設定
  MOVE 0.001 TO PARA2 OF SMP2.
  CALL "CORBA-ANY-SET-RELEASE" USING
      ANY2
      CORBA-TRUE-VALUE.

*inoutパラメタの処理
  SET ADDRESS OF ANY3-P TO ANY3.
  CALL "CORBA-FREE" USING ANY-VALUE OF ANY3-P.
  CALL "CORBA-ANY-SET-RELEASE" USING
      ANY3
      CORBA-TRUE-VALUE.
*TypeCodeの設定
  MOVE FUNCTION LENG (TC-ODSAMPLE-SAMPLE3-IMPL-SEQ) TO T-LONG.
  CALL "CORBA-STRING-SET" USING
      TYPE-P
      T-LONG
      TC-ODSAMPLE-SAMPLE3-IMPL-SEQ.
  CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
      TYPE-P
      ANY-TYPE OF ANY3-P.
  CALL "CORBA-FREE" USING
      TYPE-P.
  CALL "ODSAMPLE-SAMPLE3-ALLOC" USING
      ANY-VALUE OF ANY3-P.
  SET ADDRESS OF SMP3 TO ANY-VALUE OF ANY3-P.
  MOVE "y" TO PARA1 OF SMP3.
  MOVE 0.0001 TO PARA2 OF SMP3.

*復帰値の処理
  CALL "CORBA-ANY-ALLOC" USING ANY0.
  SET ADDRESS OF ANY0-P TO ANY0.

  MOVE FUNCTION LENG (TC-ODSAMPLE-SAMPLE1-IMPL-SEQ) TO T-LONG.
  CALL "CORBA-STRING-SET" USING
      TYPE-P
      T-LONG
      TC-ODSAMPLE-SAMPLE1-IMPL-SEQ.
  CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
      TYPE-P
      ANY-TYPE OF ANY0-P.
  CALL "CORBA-FREE" USING
      TYPE-P.

  CALL "ODSAMPLE-SAMPLE1-ALLOC" USING
      ANY-VALUE OF ANY0-P.
  SET ADDRESS OF SMP1 TO ANY-VALUE OF ANY0-P.
  MOVE 300 TO PARA1 OF SMP1.
```

```
    MOVE 5 TO LSIZE.
    MOVE "test" TO STR-WORK.
    CALL "CORBA-STRING-SET" USING
        PARA2 OF SMP1
        LSIZE
        STR-WORK.
    CALL "CORBA-ANY-SET-RELEASE" USING
        ANY0
        CORBA-TRUE-VALUE.

  MAIN-END.
 END PROGRAM "ODSAMPLE-ANYTEST-OP1".
```

6.8.4 シーケンス型

(1)IDLマッピング

IDL言語でシーケンス型sequenceを指定した場合、COBOLでは以下の構造体(シーケンス構造体)をデータ宣言します。また、シーケンス構造体の領域獲得関数(関数名:"CORBA-SEQUENCE-データ型-ALLOC"、以降XX-ALLOC関数と呼ぶ)と、シーケンスデータ域の獲得関数(関数名:"CORBA-SEQUENCE-データ型-ALLOCBUF"、以降XX-ALLOCBUF関数と呼ぶ)がIDLコンパイラで生成されます。

```
    01 CORBA-SEQUENCE.
 * シーケンス配列の最大個数
    03  SEQ-MAXIMUM    PIC  S9(9) COMP-5.
 * シーケンス配列の使用個数
    03  SEQ-LENGTH     PIC  S9(9) COMP-5.
 * シーケンスのデータ域へのポインタ
    03  SEQ-BUFFER     USAGE POINTER.
```

以降では、以下のIDL定義例をもとに説明します。

IDL言語

シーケンス(基本型)

```
module ODsample{
    interface  seqtest{
        typedef sequence<long>  sampleseq;
        sampleseq op1(in sampleseq seq1, out sampleseq seq2,
                      inout sampleseq seq3 );
    };
};
```

シーケンス(構造体型)

```
module ODsample{
    interface  seqtest{
        struct structV { string st; };
        typedef sequence<structV>  sampleseq;
        sampleseq op1(in sampleseq seq1, out sampleseq seq2,
                      inout sampleseq seq3 );
    };
};
```

COBOL

```
   01 ODSAMPLE-SEQTEST-SAMPLESEQ.
* シーケンスの最大長
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
* シーケンスの長さ
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
* シーケンスのデータ
    02 SEQ-BUFFER USAGE IS POINTER.
```

(2)クライアントアプリケーションで扱うパラメタ

クライアントアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	シーケンス構造体のメンバとして、配列の最大個数/使用個数/データ域のアドレスを設定します。 シーケンス構造体/SEQ-BUFFERの領域獲得する場合は、XX-ALLOC関数/XX-ALLOCBUF関数を使用します。	―
inout	シーケンス構造体のメンバとして、配列の最大個数/使用個数/データ域のアドレスを設定します。 シーケンス構造体/SEQ-BUFFERの領域獲得する場合は、XX-ALLOC関数/XX-ALLOCBUF関数を使用します。 リリースフラグにCORBA-TRUE-VALUEを設定する場合、データ部の領域はスタブにより解放されます。 リリースフラグにCORBA-FALSE-VALUEを設定する、またはリリースフラグを設定しない場合、スタブではデータ域を解放しません。この場合、アプリケーション内で領域を保持し、不要になった段階で解放処理を行ってください。	領域は、スタブで自動的に獲得されます(各メンバも設定されます)。 このとき、獲得した領域にはリリースフラグにCORBA-TRUE-VALUEが設定されています。
out 復帰値	―	(inoutパラメタと同じ)

注意

クライアントおよびスタブで獲得した領域は、不要になった時点でCORBA-FREE関数で解放する必要があります。このとき、データ域の領域(XX-ALLOCBUF関数で獲得)を解放するかどうかを、リリースフラグで指定します。

リリースフラグの参照／設定は、以下の関数／フラグを使用します。

[関数]

```
CALL "CORBA-SEQUENCE-GET-RELEASE" USING
     BUFFER
     RET-VAL.
CALL "CORBA-SEQUENCE-SET-RELEASE" USING
     BUFFER
     CORBA-TRUE-VALUE.
```

[フラグ]

- CORBA-TRUE-VALUE:CORBA-FREE関数発行時、SEQ-BUFEERの指す領域も解放されます。

・ CORBA-FALSE-VALUE:CORBA-FREE関数発行時、SEQ-BUFEERの指す領域は解放されません(デフォルト)。

なお、スタブで獲得されたoutパラメタ、復帰値のリリースフラグは、CORBA-TRUE-VALUEに設定されます。

クライアントアプリケーションでの処理例を以下に示します。

シーケンス(基本型)

```
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 SEQ1.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
      02 SEQ-BUFFER USAGE IS POINTER.
   01 SEQ2 USAGE POINTER.
   01 SEQ3-P USAGE POINTER.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LONG-V.
   01 BUF-IN-P USAGE POINTER.
   01 LONG-IN-P USAGE POINTER.
   01 BUF-IO-P USAGE POINTER.
   01 LONG-IO-P USAGE POINTER.
 LINKAGE SECTION.
   01 SEQ3.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
       02 SEQ-BUFFER USAGE IS POINTER.

  PROCEDURE DIVISION.

 * inパラメタの処理
   MOVE 2 TO SEQ-MAXIMUM OF SEQ1.
   MOVE 2 TO SEQ-LENGTH OF SEQ1.
   CALL "CORBA-SEQUENCE-LONG-ALLOCBUF" USING
         SEQ-LENGTH OF SEQ1
         SEQ-BUFFER OF SEQ1.
   MOVE 10 TO LONG-V.
   MOVE FUNCTION ADDR (SEQ1) TO BUF-IN-P.
   PERFORM VARYING I FROM 1 BY 1 UNTIL I > SEQ-LENGTH OF SEQ1
         COMPUTE LONG-V = 20 + LONG-V
         MOVE FUNCTION ADDR(LONG-V) TO LONG-IN-P
         CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
                                    BUF-IN-P
                                    I
                                    LONG-IN-P
   END-PERFORM
 * inoutパラメタの処理
   CALL "CORBA-SEQUENCE-LONG-ALLOC" USING
       SEQ3-P.
   SET ADDRESS OF SEQ3 TO SEQ3-P.
   MOVE 3 TO SEQ-MAXIMUM OF SEQ3.
   MOVE 3 TO SEQ-LENGTH OF SEQ3.
   CALL "CORBA-SEQUENCE-LONG-ALLOCBUF" USING
         SEQ-LENGTH OF SEQ3
         SEQ-BUFFER OF SEQ3.
   MOVE 5 TO LONG-V.
```

```
  MOVE FUNCTION ADDR (SEQ3) TO BUF-IO-P.
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > SEQ-LENGTH OF SEQ3
        COMPUTE LONG-V = 30 + LONG-V
        MOVE FUNCTION ADDR(LONG-V) TO LONG-IO-P
        CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
                                    BUF-IO-P
                                    I
                                    LONG-IO-P
  END-PERFORM
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING
        SEQ3-P
        CORBA-TRUE-VALUE.
  CALL  "ODSAMPLE-SEQTEST-OP1"  USING
      OBJ
      SEQ1
      SEQ2
      SEQ3-P
      ENV
      SEQ0.
  CALL "CORBA-FREE" USING
      SEQ0.
  CALL "CORBA-FREE" USING
      SEQ-BUFFER OF SEQ1.
  CALL "CORBA-FREE" USING
      SEQ2.
  CALL "CORBA-FREE" USING
      SEQ3-P.
```

シーケンス(構造体型)

```
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01OP1-IN-P.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
      02 SEQ-BUFFER USAGE IS POINTER.
   01 OP1-OUT-P USAGE POINTER.
   01 OP1-IO-P USAGE POINTER.
   01 OP1-RESULT USAGE IS POINTER.
   01 TMP-ODSAMPLE-SEQTEST-STRUCTV..
     02 ST USAGE IS POINTER.
   01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY WS-NUM.
   01 TMP-POINTER USAGE IS POINTER.
   01 TMP-SEQ-ADDR USAGE IS POINTER.
   01 TMP-BUF    PIC X(20).
   01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY TMP-LEN.
 LINKAGE SECTION.
   01 TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
       02 SEQ-BUFFER USAGE IS POINTER.

  PROCEDURE DIVISION.

 * inパラメタの処理
   MOVE 10 TO SEQ-MAXIMUM OF OP1-IN-P.
   MOVE 10 TO SEQ-LENGTH OF OP1-IN-P.
   CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOCBUF" USING
```

```
        SEQ-MAXIMUM OF OP1-IN-P
        SEQ-BUFFER OF OP1-IN-P.
    MOVE 10 TO LONG-V.
    MOVE FUNCTION ADDR( OP1-IN-P ) TO TMP-SEQ-ADDR.
    PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH OF OP1-IN-P
        MOVE "ABC" TO TMP-BUF
        MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
        CALL "CORBA-STRING-SET" USING
            ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV
            TMP-LEN
            TMP-BUF
        MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-STRUCTV ) TO TMP-POINTER
        CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            TMP-SEQ-ADDR
            WS-NUM
            TMP-POINTER
    END-PERFORM.
    MOVE FUNCTION ADDR( OP1-IN-P ) TO TMP-SEQ-ADDR
    PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH OF OP1-IN-P
        CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
            TMP-SEQ-ADDR
            WS-NUM
            TMP-POINTER
        SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L TO TMP-POINTER
        MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
        CALL "CORBA-STRING-GET" USING
            ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L
            TMP-LEN
            TMP-BUF
    END-PERFORM.

  * inoutパラメタの処理
    CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOC" USING
        OP1-IO-P.
    SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-IO-P.
    MOVE 10 TO SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
    MOVE 10 TO SEQ-LENGTH OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
    CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOCBUF" USING
            SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
            SEQ-BUFFER OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
    MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
    PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
        OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
        MOVE "DEF" TO TMP-BUF
        MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
        CALL "CORBA-STRING-SET" USING
            ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV
            TMP-LEN
            TMP-BUF
        MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-STRUCTV ) TO TMP-POINTER
        CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            TMP-SEQ-ADDR
            WS-NUM
            TMP-POINTER
    END-PERFORM.
    SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-IO-P.
    MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
    PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
        OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
        CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
           TMP-SEQ-ADDR
            WS-NUM
          TMP-POINTER
```

```
        SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L TO TMP-POINTER
        MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
        CALL "CORBA-STRING-GET" USING
            ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L
            TMP-LEN
            TMP-BUF
    END-PERFORM.
    CALL "CORBA-SEQUENCE-SET-RELEASE" USING
          OP1-IO-P
          CORBA-TRUE-VALUE.
    CALL "ODSAMPLE-SEQTEST-OP1" USING
        OBJ
        OP1-IN-P
        OP1-OUT-P
        OP1-IO-P
        ENV
        OP1-RESULT.
    CALL "CORBA-FREE" USING
        OP1-RESULT.
    CALL "CORBA-FREE" USING
        SEQ-BUFFER OF OP1-IN-P.
    CALL "CORBA-FREE" USING
       OP1-OUT-P.
    CALL "CORBA-FREE" USING
       OP1-IO-P.
```

(3)サーバアプリケーションで扱うパラメタ

シーケンス構造体のメンバとして、配列の最大個数/使用個数/データ域のアドレスを設定します。
シーケンス構造体/SEQ-BUFFERの領域獲得する場合、XX-ALLOC関数/XX-ALLOCBUF関数を使用します。

サーバアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
in	各領域(シーケンス構造体/データ域)は、スケルトンで自動的に獲得/解放されます。	－
inout	各領域(シーケンス構造体/データ域)は、スケルトンで自動的に獲得されます。 このとき、リリースフラグには、CORBA-TRUE-VALUEが設定されています。	inoutデータの領域を更新する場合は、以下の処理を実施してください。 SEQ-BUFFER域をCORBA-FREE関数で一度解放し、XX-ALLOCBUF関数で再獲得します。 リリースフラグにCORBA-TRUE-VALUEを設定する、またはリリースフラグを設定しない場合、領域はスケルトンで自動的に解放されます。 リリースフラグにCORBA-FALSE-VALUEを設定する場合、スケルトンでは領域を解放しません。この場合、C言語などの他言語と連携させて領域を保持し、不要になった段階で解放処理を行ってください。
out 復帰値	－	シーケンス構造体/SEQ-BUFFER域をXX-ALLOC関数/XX-ALLOCBUF関数で獲得します。 リリースフラグにCORBA-TRUE-VALUEを設定する場合、領域はスケルトンで自動的に解放されます。 リリースフラグにCORBA-FALSE-VALUEを設定する、またはリリースフラグを設定しない場合、スケルトンでは領域を解放しません。この場合、C

パラメタ	クライアントから渡されたパラメタ	クライアントへ渡すパラメタ
		言語などの他言語と連携させて領域を保持し、 不要になった段階で解放処理を行ってください。

サーバアプリケーションでの処理例を以下に示します。

シーケンス(基本型)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-SEQTEST-OP1".
 ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .
  DATA DIVISION.
  WORKING-STORAGE SECTION.
   COPY CONST IN CORBA.
   01 STR  USAGE POINTER.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
   01 STR-WORK PIC X(30).
    03 IDL-ID USAGE POINTER.
    03 MINOR PIC 9(9) COMP-5.
    03 IDL-STATUS PIC 9(9) COMP-5.
    03 PARAM USAGE POINTER.
    03 MAGIC PIC 9(9) COMP-5.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
   01 LONG-OUT-P USAGE POINTER.
   01 LONG-IO-P  USAGE POINTER.
   01 LONG-RET-P USAGE POINTER.
   01 BUF-OUT-P  USAGE POINTER.
   01 BUF-IO-P   USAGE POINTER.
   01 BUF-RET-P  USAGE POINTER.
  LINKAGE SECTION.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 SEQ.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
     02 SEQ-BUFFER USAGE POINTER.
   01 SEQ1.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
     02 SEQ-BUFFER USAGE POINTER.
   01 SEQ2.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
     02 SEQ-BUFFER USAGE POINTER.
   01 SEQ3.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
     02 SEQ-BUFFER USAGE POINTER.
   01 SEQ1-P USAGE POINTER.
   01 SEQ2-P USAGE POINTER.
   01 SEQ3-P USAGE POINTER.
   01 ENV.
    03 MAJOR PIC 9(9) COMP-5.
      88 CORBA-NO-EXCEPTION VALUE 0.
      88 CORBA-USER-EXCEPTION VALUE 1.
      88 CORBA-SYSTEM-EXCEPTION VALUE 2.
```

```
  01 SEQ-P USAGE POINTER.
PROCEDURE DIVISION USING
      OBJ
      SEQ1
      SEQ2-P
      SEQ3-P
      ENV
      SEQ-P.

MAIN.
* outパラメタの処理
  CALL "CORBA-SEQUENCE-LONG-ALLOC" USING
      SEQ2-P
  SET ADDRESS OF SEQ2 TO SEQ2-P.
  MOVE 4 TO SEQ-MAXIMUM OF SEQ2.
  MOVE 4 TO SEQ-LENGTH OF SEQ2.
  CALL "CORBA-SEQUENCE-LONG-ALLOCBUF" USING
       SEQ-LENGTH OF SEQ2
       SEQ-BUFFER OF SEQ2.
  MOVE FUNCTION ADDR (SEQ2) TO BUF-OUT-P.
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > SEQ-LENGTH OF SEQ2
       COMPUTE NUM = I * 100
       MOVE FUNCTION ADDR(NUM) TO LONG-OUT-P
       CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            BUF-OUT-P
            I
            LONG-OUT-P
  END-PERFORM
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING
      SEQ2-P
       CORBA-TRUE-VALUE.

* inoutパラメタの処理
  SET ADDRESS OF SEQ3 TO SEQ3-P.
  CALL "CORBA-FREE" USING
       SEQ-BUFFER OF SEQ3.
  MOVE 5 TO SEQ-MAXIMUM OF SEQ3.
  MOVE 5 TO SEQ-LENGTH OF SEQ3.
  CALL "CORBA-SEQUENCE-LONG-ALLOCBUF" USING
       SEQ-LENGTH OF SEQ3
       SEQ-BUFFER OF SEQ3.
  MOVE FUNCTION ADDR (SEQ3) TO BUF-IO-P.
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > SEQ-LENGTH OF SEQ3
       COMPUTE NUM = I * 1000
       MOVE FUNCTION ADDR(NUM) TO LONG-IO-P
       CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
           BUF-IO-P
           I
           LONG-IO-P
  END-PERFORM
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING
      SEQ3-P
       CORBA-TRUE-VALUE.

* 復帰値の処理
  CALL "CORBA-SEQUENCE-LONG-ALLOC" USING
      SEQ-P.
  SET ADDRESS OF SEQ TO SEQ-P.
  MOVE 6 TO SEQ-MAXIMUM OF SEQ.
  MOVE 6 TO SEQ-LENGTH OF SEQ.
  MOVE FUNCTION ADDR(SEQ) TO BUF-RET-P.
  CALL "CORBA-SEQUENCE-LONG-ALLOCBUF" USING
       SEQ-LENGTH OF SEQ
```

```
        SEQ-BUFFER OF SEQ.
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > SEQ-LENGTH OF SEQ
        COMPUTE NUM = I * 10000
        MOVE FUNCTION ADDR(NUM) TO LONG-RET-P
        CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            BUF-RET-P
            I
            LONG-RET-P
  END-PERFORM
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING
      SEQ-P
        CORBA-TRUE-VALUE.
  MAIN-END.
 END PROGRAM "ODSAMPLE-SEQTEST-OP1".
```

シーケンス(構造体型)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-SEQTEST-OP1".
 ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .
  DATA DIVISION.
  WORKING-STORAGE SECTION.
   COPY CONST IN CORBA.
   01 TMP-POINTER USAGE IS POINTER.
   01 TMP-SEQ-ADDR USAGE IS POINTER.
   01 TMP-BUF    PIC X(20).
   01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY TMP-LEN.
   01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY WS-NUM.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY WS-LONG.
   01 TMP-ODSAMPLE-SEQTEST-STRUCTV.
      02 ST USAGE IS POINTER.
  LINKAGE SECTION.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 OP1-IN-P.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
     02 SEQ-BUFFER USAGE IS POINTER.
   01 OP1-OUT-P USAGE IS POINTER.
   01 OP1-IO-P     USAGE IS POINTER.
   01 OP1-RESULT USAGE IS POINTER.
   01 TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
     02 SEQ-BUFFER USAGE IS POINTER.
   01 TMP-ODSAMPLE-SEQTEST-STRUCTV-L.
     02 ST USAGE IS POINTER.

 PROCEDURE DIVISION USING
       OBJ
       OP1-IN-P
       OP1-OUT-P
       OP1-IO-P
       ENV
       OP1-RESULT.

 MAIN.
```

```
* outパラメタの処理
  CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOC" USING OP1-OUT-P.
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING OP1-OUT-P CORBA-TRUE-VALUE.
  SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-OUT-P.
  MOVE 10 TO SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
  MOVE 10 TO SEQ-LENGTH OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
  CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOCBUF" USING
          SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
          SEQ-BUFFER OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
  MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
  PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
      OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
      MOVE "JKL" TO TMP-BUF
      MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
      CALL "CORBA-STRING-SET" USING
          ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV
          TMP-LEN
          TMP-BUF
      MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-STRUCTV ) TO TMP-POINTER
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
          TMP-SEQ-ADDR
          WS-NUM
          TMP-POINTER
  END-PERFORM.
  SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-OUT-P.
  MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
  PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
      OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
      CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
          TMP-SEQ-ADDR
          WS-NUM
          TMP-POINTER
      SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L TO TMP-POINTER
      MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
      CALL "CORBA-STRING-GET" USING
          ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L
          TMP-LEN
          TMP-BUF
  END-PERFORM.
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING
     OP1-OUT-P
     CORBA-TRUE-VALUE.

* inoutパラメタの処理
  CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOC" USING OP1-IO-P.
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING OP1-IO-P CORBA-TRUE-VALUE.
  SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-IO-P.
  MOVE 10 TO SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
  MOVE 10 TO SEQ-LENGTH OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
  CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOCBUF" USING
          SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
          SEQ-BUFFER OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
  MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
  PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
      OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
      MOVE "GHI" TO TMP-BUF
      MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
      CALL "CORBA-STRING-SET" USING
          ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV
          TMP-LEN
          TMP-BUF
      MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-STRUCTV ) TO TMP-POINTER
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
```

```
        TMP-SEQ-ADDR
         WS-NUM
        TMP-POINTER
  END-PERFORM.
  SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-IO-P.
  MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
  PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
      OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
      CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
          TMP-SEQ-ADDR
          WS-NUM
          TMP-POINTER
      SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L TO TMP-POINTER
      MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
      CALL "CORBA-STRING-GET" USING
          ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L
          TMP-LEN
          TMP-BUF
  END-PERFORM.
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING
     OP1-IO-P
     CORBA-TRUE-VALUE.

* 復帰値の処理
  CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOC" USING OP1-RESULT.
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING OP1-RESULT CORBA-TRUE-VALUE.
  SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-RESULT.
  MOVE 10 TO SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
  MOVE 10 TO SEQ-LENGTH OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
  CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOCBUF" USING
          SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
          SEQ-BUFFER OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
  MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
  PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
      OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
      MOVE "MNO" TO TMP-BUF
      MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
      CALL "CORBA-STRING-SET" USING
          ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV
          TMP-LEN
          TMP-BUF
      MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-STRUCTV ) TO TMP-POINTER
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
          TMP-SEQ-ADDR
          WS-NUM
          TMP-POINTER
  END-PERFORM.
  SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-RESULT.
  MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
  PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
      OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
      CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
          TMP-SEQ-ADDR
          WS-NUM
          TMP-POINTER
      SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L TO TMP-POINTER
      MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
      CALL "CORBA-STRING-GET" USING
          ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L
          TMP-LEN
          TMP-BUF
  END-PERFORM.
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING
```

```
      OP1-RESULT
      CORBA-TRUE-VALUE.
  MAIN-END.
END PROGRAM "ODSAMPLE-SEQTEST-OP1".
```

6.8.5 構造体

(1)IDLマッピング

IDL言語で構造体structを指定した場合、COBOLでも構造体でデータ宣言します。
可変長構造体の場合、領域獲得関数(関数名:“モジュール名-インタフェース名-構造体名-ALLOC”、以降XX-ALLOC関数と呼ぶ)がIDLコンパイラで生成されます。
なお、アプリケーションで使用可能なCORBAのデータ型については、“6.8 データ型に対するマッピング”を参照してください。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    struct samplefix {    // 構造体(固定長)
        long   para1;
        long   para2;
        string<10> para3;
    };
    struct samplevar {    // 構造体(可変長)
        long   para1;
        string para2;
    };
    interface  structtest{
        samplefix  op2(
           in samplefix str1,
           out samplefix str2,
           inout samplefix str3
        );
        samplevar  op1(
            in samplevar str1,
            out samplevar str2,
            inout samplevar str3
        );
    };
};
```

COBOL

```
 * 構造体(固定長)
     01 ODSAMPLE-SAMPLEFIX.
        02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
        02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
        02 PARA3 PIC X(10).
 * 構造体(可変長)
     01 ODSAMPLE-SAMPLEVAR.
        02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
        02 PARA2 USAGE IS POINTER.
```

(2)クライアントアプリケーションで扱うパラメタ(固定長)

構造体(固定長)のパラメタ(in、out、inout)を扱う場合、構造体のアドレスをパラメタに指定します。領域の獲得／解放を行う必要はありません。

```
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
```

```
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
    01 FIX1.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
       02 PARA3 PIC X(10).
    01 FIX2.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
       02 PARA3 PIC X(10).
    01 FIX3.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
       02 PARA3 PIC X(10).
    01 FIX0.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
       02 PARA3 PIC X(10).

  PROCEDURE DIVISION.

* inパラメタの設定
  MOVE 10 TO PARA1 OF FIX1.
* inパラメタの設定
  MOVE 11 TO PARA2 OF FIX1.
* inパラメタの設定
  MOVE "IN" TO PARA3 OF FIX1.
* inoutパラメタの設定
  MOVE 20 TO PARA1 OF FIX3.
* inoutパラメタの設定
  MOVE 21 TO PARA2 OF FIX3.
* inoutパラメタの設定
  MOVE "INOUT:1" TO PARA3 OF FIX3.

  CALL "ODSAMPLE-STRUCTTEST-OP2" USING
       OBJ
       FIX1
       FIX2
       FIX3
       ENV
       FIX0.
```

(3)クライアントアプリケーションで扱うパラメタ(可変長)

クライアントアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	構造体領域/可変長データ域を獲得する場合、XX-ALLOC関数/データ域獲得関数を使用します。	―
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。
out 復帰値	―	(inoutパラメタと同じ)

クライアントおよびスタブで動的に獲得した領域は、不要になった時点でCORBA-FREE関数で解放する必要があります。CORBA-FREE関数を発行することで可変長データ域も解放されます。

クライアントアプリケーションでの処理例を以下に示します。

```
 ENVIRONMENT DIVISION.
 DATA DIVISION.
  WORKING-STORAGE SECTION.
   COPY CONST IN CORBA.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 VAR1.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
      02 PARA2 USAGE IS POINTER.
   01 VAR2 USAGE POINTER.
   01 VAR3-P USAGE POINTER.
   01 VAR0 USAGE POINTER.
   01 STR-WORK PIC X(30).
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  LINKAGE SECTION.
   01 VAR3.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
      02 PARA2 USAGE IS POINTER.

 PROCEDURE DIVISION.

*inパラメタの設定
   MOVE 5 TO PARA1 OF VAR1.
   MOVE "test" TO STR-WORK.
   MOVE FUNCTION LENG(STR-WORK) TO LSIZE
   CALL "CORBA-STRING-SET" USING
        PARA2 OF VAR1
        LSIZE
        STR-WORK.

*inoutパラメタの領域獲得
   CALL "ODSAMPLE-SAMPLEVAR-ALLOC" USING
         VAR3-P.
   SET ADDRESS OF VAR3 TO VAR3-P.
*inoutパラメタの設定
   MOVE 4 TO PARA1 OF VAR3.
   MOVE "pro" TO STR-WORK.
   MOVE FUNCTION LENG(STR-WORK) TO LSIZE.
   CALL "CORBA-STRING-SET" USING
        PARA2 OF VAR3
        LSIZE
        STR-WORK.

   CALL "ODSAMPLE-STRUCTTEST-OP1" USING
       OBJ
       VAR1
       VAR2
       VAR3-P
       ENV
       VAR0.
   CALL "CORBA-FREE" USING PARA2 OF VAR1.
   CALL "CORBA-FREE" USING VAR2.
   CALL "CORBA-FREE" USING VAR3-P.
   CALL "CORBA-FREE" USING VAR0.
```

(4)サーバアプリケーションで扱うパラメタ(固定長)

構造体(固定長)のパラメタ(in、out、inout)を扱う場合、構造体領域はスケルトンで自動的に獲得／解放されます。out／inoutパラメタをクライアントアプリケーションに渡す場合、処理結果を構造体メンバに設定します。
サーバアプリケーションでの処理例を以下に示します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "ODSAMPLE-STRUCTTEST-OP2".
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
    SYMBOLIC CONSTANT
    COPY SYMBOL-CONST IN CORBA.
    .

DATA DIVISION.
WORKING-STORAGE SECTION.
 COPY CONST IN CORBA.
 01 STR-WORK PIC X(30).
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
LINKAGE SECTION.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 01 STR1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
    02 PARA3 PIC X(10).
 01 STR2.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
    02 PARA3 PIC X(10).
 01 STR3.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
    02 PARA3 PIC X(10).
 01 ENV.
    03 MAJOR PIC 9(9) COMP-5.
       88 CORBA-NO-EXCEPTION VALUE 0.
       88 CORBA-USER-EXCEPTION VALUE 1.
       88 CORBA-SYSTEM-EXCEPTION VALUE 2.
    03 IDL-ID USAGE POINTER.
    03 MINOR PIC 9(9) COMP-5.
    03 IDL-STATUS PIC 9(9) COMP-5.
    03 PARAM USAGE POINTER.
    03 MAGIC PIC 9(9) COMP-5.
 01 FIX.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
    02 PARA3 PIC X(10).

PROCEDURE DIVISION USING
    OBJ
    STR1
    STR2
    STR3
    ENV
    FIX.

MAIN SECTION.

* outパラメタの設定
   MOVE 0 TO PARA1 OF STR2.
   MOVE 1 TO PARA2 OF STR2.
   MOVE "OUT" TO PARA3 OF STR2.
* inoutパラメタの設定
   MOVE 2 TO PARA1 OF STR3.
   MOVE 3 TO PARA2 OF STR3.
   MOVE "INOUT:2" TO PARA3 OF STR3.
* 復帰値の処理
   MOVE 4 TO PARA1 OF FIX.
```

```
    MOVE 5 TO PARA2 OF FIX.
    MOVE "RTN" TO PARA3 OF FIX.

  MAIN-END.
END PROGRAM "ODSAMPLE-STRUCTTEST-OP2".
```

(5)サーバアプリケーションで扱うパラメタ(可変長)

サーバアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	クライアントからのパラメタ領域	クライアントへのパラメタ領域
in	構造体領域/可変長データ域は、スケルトンで自動的に獲得/解放されます。	－
inout	構造体領域/可変長データ域は、スケルトンで自動的に獲得されます。	可変長データ域をCORBA-FREE関数で一度解放し、データ域獲得関数で再獲得します。 領域は、スケルトンで自動的に解放されます。
out 復帰値	－	構造体/可変長データ域をXX-ALLOC関数/データ域獲得関数で獲得します。 領域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-STRUCTTEST-OP1".
 ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .
 DATA DIVISION.
 WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
  01 STR-WORK PIC X(30).
 LINKAGE SECTION.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 STR1.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 PARA2 USAGE IS POINTER.
  01 STR2 USAGE IS POINTER.
  01 STR2-P.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 PARA2 USAGE IS POINTER.
  01 STR3 USAGE IS POINTER.
  01 STR3-P.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 PARA2 USAGE IS POINTER.
  01 STRO-P.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 PARA2 USAGE IS POINTER.
  01 STRO USAGE IS POINTER.
  01 ENV.
     03 MAJOR PIC 9(9) COMP-5.
        88 CORBA-NO-EXCEPTION VALUE 0.
        88 CORBA-USER-EXCEPTION VALUE 1.
        88 CORBA-SYSTEM-EXCEPTION VALUE 2.
     03 IDL-ID USAGE POINTER.
     03 MINOR PIC 9(9) COMP-5.
```

```
    03 IDL-STATUS PIC 9(9) COMP-5.
    03 PARAM USAGE POINTER.
    03 MAGIC PIC 9(9) COMP-5.

PROCEDURE DIVISION USING
    OBJ
    STR1
    STR2
    STR3
    ENV
    STR0.

MAIN SECTION.

*outパラメタの処理
  CALL "ODSAMPLE-SAMPLEVAR-ALLOC" USING
         STR2.
  SET ADDRESS OF STR2-P TO STR2.
  MOVE 12 TO PARA1 OF STR2-P.
MOVE "(*str2)->para2" TO STR-WORK.
MOVE FUNCTION LENG(STR-WORK) TO LSIZE
  CALL "CORBA-STRING-SET" USING
        PARA2 OF STR2-P
        LSIZE
        STR-WORK.

*inoutパラメタの処理
  SET ADDRESS OF STR3-P TO STR3.
  CALL  "CORBA-FREE"  USING
         PARA2 OF STR3-P.
   MOVE 12 TO PARA1 OF STR3-P.
   MOVE 16 TO LSIZE.
   MOVE "(*str3)->para2" TO STR-WORK.
   MOVE FUNCTION LENG(STR-WORK) TO LSIZE
   CALL "CORBA-STRING-SET" USING
        PARA2 OF STR3-P
        LSIZE
        STR-WORK.

*復帰値の処理
  CALL "ODSAMPLE-SAMPLEVAR-ALLOC" USING
         STR0.
  SET ADDRESS OF STR0-P TO STR0.
  MOVE 11 TO PARA1 OF STR0-P.
  MOVE 16 TO LSIZE.
  MOVE "(*str2)->para2" TO STR-WORK.
  MOVE FUNCTION LENG(STR-WORK) TO LSIZE
  CALL "CORBA-STRING-SET" USING
        PARA2 OF STR0-P
        LSIZE
        STR-WORK.
 MAIN-END.
END PROGRAM "ODSAMPLE-STRUCTTEST-OP1".
```

6.8.6 共用体

(1)IDLマッピング

IDL言語で共用体unionを指定した場合、COBOLではデータ型を識別する弁別情報Dと共用体Uから構成される構造体をデータ宣言します。また、構造体の領域獲得関数(関数名:“モジュール名-インタフェース名-構造体名-ALLOC”、以降XX-ALLOC関数と呼ぶ)がIDLコンパイラで生成されます。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{

    union samplefix switch(long){    // 共用体(固定長)
        case 1: long    para1;
        case 2: long    para2;
    };
    union samplevar switch(long){    // 共用体(可変長)
        case 1: long    para1;
        case 2: string  para2;
    };
    interface  uniontest{
        samplefix  op2(
            in samplefix uni1,
            out samplefix uni2,
            inout samplefix uni3
        );
        samplevar  op1(
            in samplevar uni1,
            out samplevar uni2,
            inout samplevar uni3
        );
    };
};
```

COBOL

```
 * 共用体(固定長)
  01 ODSAMPLE-SAMPLEFIX.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
    02 U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 FILLER REDEFINES U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.

 * 共用体(可変長)
  01 ODSAMPLE-SAMPLEVAR.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
    02 U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 FILLER REDEFINES U.
      03 PARA2 USAGE IS POINTER.
```

(2)クライアントアプリケーションで扱うパラメタ(固定長)

共用体(固定長)のパラメタ(in、out、inout)を扱う場合、領域の獲得／解放を行う必要はありません。弁別情報にデータ型識別値、データ域にデータを設定します。
クライアントアプリケーションでの処理例を以下に示します。

```
ENVIRONMENT DIVISION.
DATA DIVISION.
 WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 UNI1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
    02 U.
```

```
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
      02 FILLER REDEFINES U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    01 UNI2.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
      02 U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
      02 FILLER REDEFINES U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    01 UNI3.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
      02 U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
      02 FILLER REDEFINES U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    01 UNI0.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
      02 U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
      02 FILLER REDEFINES U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY L-SIZE.
    01 STR-WORK PIC X(10).

  PROCEDURE DIVISION.

* inパラメタの弁別子の設定
    MOVE 2 TO D OF UNI1.
* inパラメタの値の設定
    MOVE 100 TO PARA2 OF UNI1.
* inoutパラメタの弁別子の設定
    MOVE 2 TO D OF UNI3.
* inoutパラメタの値の設定
    MOVE 200 TO PARA2 OF UNI3.
    CALL "ODSAMPLE-UNIONTEST-OP2" USING
          OBJ
          UNI1
          UNI2
          UNI3
          ENV
          UNI0.
```

(3)クライアントアプリケーションで扱うパラメタ(可変長)

クライアントアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	サーバへ渡すパラメタ	サーバから渡されたパラメタ
in	構造体領域/可変長データ域を獲得する場合、XX-ALLOC関数/データ領域獲得関数を使用します。 データ型識別値とデータを設定します。	−
inout	(inパラメタと同じ)	領域は、スタブで自動的に獲得されます。
out 復帰値	−	(inoutパラメタと同じ)

注意

クライアントおよびスタブで動的に獲得した領域は、不要になった時点でCORBA-FREE関数で解放する必要があります。CORBA-FREE関数を発行することで可変長データ域も解放されます。

クライアントアプリケーションでの処理例を以下に示します。

```
  ENVIRONMENT DIVISION.
  DATA DIVISION.
   WORKING-STORAGE SECTION.
    COPY CONST IN CORBA.
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
    01 UNI1.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
      02 U.
        03 PARA2 USAGE IS POINTER.
      02 FILLER REDEFINES U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    01 UNI2 USAGE POINTER.
    01 UNI3 USAGE POINTER.
    01 UNI0 USAGE POINTER.
    01 STR-WORK PIC X(30).
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.

   LINKAGE SECTION.
    01 UNI-P.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
      02 U.
        03 PARA2 USAGE IS POINTER.
      02 FILLER REDEFINES U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.

  PROCEDURE DIVISION.

 * inパラメタの弁別子の設定
    MOVE 1 TO D OF UNI1.
 * inパラメタの値の設定
    MOVE 10 TO PARA1 OF UNI1.

  *inoutパラメタの領域獲得
    CALL "ODSAMPLE-SAMPLEVAR-ALLOC" USING
          UNI3.
    SET ADDRESS OF UNI-P TO UNI3.
 * inoutパラメタの弁別子の設定
    MOVE 2 TO D OF UNI1-P.
 * inoutパラメタの値の設定
    MOVE 12 TO LSIZE.
    MOVE "INOUT:para2" TO STR-WORK.
    MOVE FUNCTION LENG(STR-WORK) TO LSIZE
    CALL "CORBA-STRING-SET" USING
         PARA2 OF UNI-P
         LSIZE
         STR-WORK.

    CALL "ODSAMPLE-UNIONTEST-OP1" USING
        OBJ
        UNI1
        UNI2
        UNI3
```

```
       ENV
       UNIO.
* outパラメタの領域の解放
 CALL "CORBA-FREE" USING
       UNI2.
* inoutパラメタの領域の解放
 CALL "CORBA-FREE" USING
       UNI3.
* 復帰値の領域の解放
 CALL "CORBA-FREE" USING
       UNIO.
```

(4)サーバアプリケーションで扱うパラメタ(固定長)

共用体(固定長)のパラメタ(in、out、inout)を扱う場合、構造体領域はスケルトンで自動的に獲得／解放されます。out／inoutパラメタをクライアントアプリケーションに渡す場合、処理結果を構造体メンバに設定します。
サーバアプリケーションでの処理例を以下に示します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-UNIONTEST-OP2".
 DATA DIVISION.
  WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
 LINKAGE SECTION.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 UNI1.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
      02 U.
         03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
      02 FILLER REDEFINES U.
         03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
     01 UNI2.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
      02 U.
         03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
      02 FILLER REDEFINES U.
         03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
   01 UNI3.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
      02 U.
         03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
      02 FILLER REDEFINES U.
         03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
   01 ENV.
      03 MAJOR PIC 9(9) COMP-5.
         88 CORBA-NO-EXCEPTION VALUE 0.
         88 CORBA-USER-EXCEPTION VALUE 1.
         88 CORBA-SYSTEM-EXCEPTION VALUE 2.
      03 IDL-ID USAGE POINTER.
      03 MINOR PIC 9(9) COMP-5.
      03 IDL-STATUS PIC 9(9) COMP-5.
      03 PARAM USAGE POINTER.
      03 MAGIC PIC 9(9) COMP-5.
   01 UNI.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
      02 U.
         03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
      02 FILLER REDEFINES U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.

 PROCEDURE DIVISION USING
       OBJ
```

```
       UNI1
       UNI2
       UNI3
       ENV
       UNI.

MAIN SECTION.

* outパラメタの処理
   MOVE 1 TO D OF UNI2.
   MOVE 10 TO PARA1 OF UNI2.
* inoutパラメタの処理
   MOVE 1 TO D OF UNI3.
   MOVE 30 TO PARA1 OF UNI3.
* 復帰値の処理
   MOVE 1 TO D OF UNI.
   MOVE 100 TO PARA1 OF UNI.
MAIN-END.
END PROGRAM "ODSAMPLE-UNIONTEST-OP2".
```

(5)サーバアプリケーションで扱うパラメタ(可変長)

サーバアプリケーションのパラメタの扱いについて、以下に示します。

パラメタ	クライアントからのパラメタ領域	クライアントへのパラメタ領域
in	構造体領域/可変長データ域は、スケルトンで自動的に獲得/解放されます。	―
inout	構造体領域/可変長データ域は、スケルトンで自動的に獲得されます。	可変長データ域をCORBA-FREE関数で一度解放し、データ域獲得関数で再獲得します。 領域は、スケルトンで自動的に解放されます。
out 復帰値	―	構造体/可変長データ域をXX-ALLOC関数/データ域獲得関数で獲得します。 領域は、スケルトンで自動的に解放されます。

サーバアプリケーションでの処理例を以下に示します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-UNIONTEST-OP1".
 DATA DIVISION.
  WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.
   01 STR-WORK PIC X(30).
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 LINKAGE SECTION.
   01 UNI1.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
     02 U.
       03 PARA2 USAGE IS POINTER.
     02 FILLER REDEFINES U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
   01 UNI2-P.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
     02 U.
       03 PARA2 USAGE IS POINTER.
     02 FILLER REDEFINES U.
       03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
   01 UNI2 USAGE IS POINTER.
   01 UNI3-P.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
```

```
    02 U.
      03 PARA2 USAGE IS POINTER.
    02 FILLER REDEFINES U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
  01 UNI3 USAGE IS POINTER.

  01 ENV.
    03 MAJOR PIC 9(9) COMP-5.
      88 CORBA-NO-EXCEPTION VALUE 0.
      88 CORBA-USER-EXCEPTION VALUE 1.
      88 CORBA-SYSTEM-EXCEPTION VALUE 2.
    03 IDL-ID USAGE POINTER.
    03 MINOR PIC 9(9) COMP-5.
    03 IDL-STATUS PIC 9(9) COMP-5.
    03 PARAM USAGE POINTER.
    03 MAGIC PIC 9(9) COMP-5.
  01 UNI-P.
   02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
   02 U.
    03 PARA2 USAGE IS POINTER.
   02 FILLER REDEFINES U.
    03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
  01 UNI USAGE POINTER.

PROCEDURE DIVISION USING
      OBJ
      UNI1
      UNI2
      UNI3
      ENV
      UNI.

MAIN SECTION.

* outパラメタの処理
  CALL "ODSAMPLE-SAMPLEVAR-ALLOC" USING
      UNI2.
  SET ADDRESS OF UNI2-P TO UNI2.
  MOVE 2 TO D OF UNI2-P.
  MOVE 10 TO LSIZE.
  MOVE "OUT:param" TO STR-WORK.
  MOVE FUNCTION LENG(STR-WORK) TO LSIZE.
  CALL "CORBA-STRING-SET" USING
       PARA2 OF UNI2-P
       LSIZE
       STR-WORK.

* inout復帰値の処理
   SET ADDRESS OF UNI3-P TO UNI3.
   EVALUATE D OF UNI3-P
      WHEN 2
         CALL "CORBA-FREE" USING
              PARA2 OF UNI3-P
   END-EVALUATE.
   MOVE 1 TO D OF UNI3-P.
   MOVE 10 TO PARA1 OF UNI3-P.

* 復帰値の処理
   CALL "ODSAMPLE-SAMPLEVAR-ALLOC" USING
      UNI.
   SET ADDRESS OF UNI-P TO UNI.
   MOVE 1 TO D OF UNI-P.
   MOVE 30 TO PARA1 OF UNI-P.
```

```
MAIN-END.
END PROGRAM "ODSAMPLE-UNIONTEST-OP1".
```

6.8.7 固定小数点型

(1)IDLマッピング

IDL言語で固定小数点型fixedを指定した場合、COBOLでは有効桁数と小数点位置により以下のようにデータ宣言します。

IDL言語

```
// fixed <digits, scale> とした場合 (digits=有効桁数,scale=小数点位置)
typedef fixed <10,2>  fixed1;    // digits > scale > 0 のとき
typedef fixed <10,-2> fixed2;    // digits-18 <= scale < 0 のとき
typedef fixed <2,10>  fixed3;    // digits < scale <= 18 のとき
typedef fixed <10,10> fixed4;    // digits = scale のとき
typedef fixed <10,0>  fixed5;    // scale = 0 のとき
```

COBOL

```
   01 FIXED1 PIC S9(8)V9(2)  PACKED-DECIMAL.
   01 FIXED2 PIC S9(10)P(2)  PACKED-DECIMAL.
   01 FIXED3 PIC SVP(8)9(2)  PACKED-DECIMAL.
   01 FIXED4 PIC SV9(10)     PACKED-DECIMAL.
   01 FIXED5 PIC S9(10)      PACKED-DECIMAL.
```

以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample {
    interface fixedtest {
        fixed <10,3>  op1(
            in       fixed <5,10>    fixed1,
            out      fixed <10,-3>   fixed2,
            inout    fixed <10,10>   fixed3 );
    };
};
```

COBOL

```
   IDENTIFICATION DIVISION.
   PROGRAM-ID. "EXEC-METHOD".
   AUTHOR. OD/IDLCOMPILER VER.2.0.
   INSTALLATION. IDL FILE NAME IS simple.idl.
   SECURITY. THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
             WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
             GURANTEED IS NOT DONE.
   DATE-WRITTEN. TUE MAY  6 11:03:40 1997
  * ALL RIGHTS RESERVED, COPYRIGHT (C) FUJITSU LIMITED 1998
   ENVIRONMENT DIVISION.
   CONFIGURATION SECTION.
   SPECIAL-NAMES.
       SYMBOLIC CONSTANT
       COPY SYMBOL-CONST IN CORBA.
       .
```

```
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT OBJFILE ASSIGN "obj".
*
 DATA DIVISION.
 FILE SECTION.
 FD  OBJFILE.
 01  FILE-BUF PIC X(1024).
*
 WORKING-STORAGE SECTION.
 COPY CONST IN CORBA.
 01 MESS PIC X(40).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 01 FIXED1 PIC SVP(5)9(5) PACKED-DECIMAL.
 01 FIXED2 PIC S9(10)P(3) PACKED-DECIMAL.
 01 FIXED3 PIC SV9(10)    PACKED-DECIMAL.
 01 FIXEDO PIC S9(7)V9(3) PACKED-DECIMAL.
*
 LINKAGE SECTION.

 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
 01 COPY BOOLEAN IN CORBA REPLACING CORBA-BOOLEAN BY RESULT.

*
 PROCEDURE DIVISION USING
         OBJ
         NUM
         RESULT.
*
     MOVE  0.22   TO FIXED1.
     MOVE  0.55555   TO FIXED3.

     CALL "ODSAMPLE-FIXEDTEST-OP1" USING
           OBJ
           FIXED1
           FIXED2
           FIXED3
           ENV
           FIXEDO.
     MOVE "ODSAMPLE-FIXEDTEST-OP1" TO MESS.
     PERFORM ENV-CHECK
*
     EXIT PROGRAM.
 MAIN-END.

 ENV-CHECK SECTION.
     EVALUATE TRUE
         WHEN CORBA-NO-EXCEPTION OF MAJOR OF ENV
              CONTINUE
         WHEN CORBA-USER-EXCEPTION OF MAJOR OF ENV
              DISPLAY "USER-EXCEPTION   : " MESS
              MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
              CALL "CORBA-STRING-GET" USING
                                  IDL-ID OF ENV
                                  STRING-LENGTH
                                  MESS
              DISPLAY "ID : " MESS
              EXIT PROGRAM
         WHEN CORBA-SYSTEM-EXCEPTION OF MAJOR OF ENV
              DISPLAY "SYSTEM-EXCEPTION : " MESS
              MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
```

```
               CALL "CORBA-STRING-GET" USING
                                   IDL-ID OF ENV
                                   STRING-LENGTH
                                   MESS
               DISPLAY "ID : " MESS
               EXIT PROGRAM
           END-EVALUATE.
       ENV-CHECK-END.
           EXIT.
      *
       END PROGRAM "EXEC-METHOD".
```

(2)クライアントアプリケーションで扱うパラメタ

クライアントアプリケーションでの処理例を以下に示します。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. "CLIENT-MAIN".
       AUTHOR. OD/IDLCOMPILER VER.2.0.
       INSTALLATION. IDL FILE NAME IS simple.idl.
       SECURITY. THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
                 WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
                 GURANTEED IS NOT DONE.
       DATE-WRITTEN. TUE MAY  6 11:03:40 1997
      * ALL RIGHTS RESERVED, COPYRIGHT (C) FUJITSU LIMITED 1998
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           ARGUMENT-NUMBER   IS ARG-C
           ARGUMENT-VALUE    IS ARG-V
           SYMBOLIC CONSTANT
           COPY SYMBOL-CONST IN CORBA.
           .
      *
       DATA DIVISION.
      *
       WORKING-STORAGE SECTION.
       COPY CONST IN CORBA.
       01 TEMP-BUF USAGE POINTER.
       01 MESS PIC X(30).
       01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
       01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
       01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
       01 COPY BOA IN CORBA REPLACING CORBA-BOA BY BOA.
       01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
       01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
       01 COPY BOOLEAN IN CORBA    REPLACING CORBA-BOOLEAN BY RET.
       01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
       01 BUFFER USAGE POINTER.
       01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY BUF-LENGTH.
       01 STR-BUF PIC X(30).
       01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING
              COSNAMING-NAMINGCONTEXT BY COS-NAMING.
       01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
       01 NAME-A USAGE POINTER.
       01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING
              COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
       01 NAME-COMPONENT-A USAGE POINTER.
      *########## ORB SETTING PARAMETER #########
       01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
       01 CURRENT-ARG-V.
          02 FILLER OCCURS 6.
```

```
       03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_c".
 01 TMP-STRING-BUF PIC X(20).
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
*
 PROCEDURE DIVISION.
*
 MAIN.

     DISPLAY "CLIENT START!!".
* argument set : CURRENT-ARG-V-VALUE
     ACCEPT CURRENT-ARG-C FROM ARG-C.
     COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
     PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
        IF ARG-COUNT = 1
           MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
        END-IF
        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        CALL "CORBA-STRING-SET" USING
             CURRENT-ARG-V-VALUE (ARG-COUNT)
             STRING-LENGTH
             TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.
* ObjectDirector Initialize
     MOVE  12 TO STRING-LENGTH.
     CALL  "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           CURRENT-ARG-C
           CURRENT-ARG-V
           TEMP-BUF
           ENV
           ORB.
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-INIT" TO MESS.
     PERFORM ENV-CHECK

     MOVE 15 TO STRING-LENGTH.
     CALL  "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-BOA-OA-ID.
     CALL "CORBA-ORB-BOA-INIT" USING
           ORB
           CURRENT-ARG-C
           CURRENT-ARG-V
           TEMP-BUF
           ENV
           BOA.
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-BOA-INIT" TO MESS.
     PERFORM ENV-CHECK.
* NamingService repository get
     MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-ORB-OBJECTID-NAMESERVICE.
```

```
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
           ORB
           TEMP-BUF
           ENV
           COS-NAMING.
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" TO MESS.
     PERFORM ENV-CHECK.
* ODdemo::calculator repository get
     MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
     MOVE "ODsample::fixedtest" TO STR-BUF.
     CALL "CORBA-STRING-SET" USING
           IDL-ID OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
     MOVE " " TO STR-BUF.
     CALL "CORBA-STRING-SET" USING
           KIND OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
     MOVE 1 TO SEQ-LENGTH OF NAME.
     MOVE 1 TO SEQ-MAXIMUM OF NAME.
     MOVE 1 TO NUM.
     CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
           SEQ-MAXIMUM OF NAME
           SEQ-BUFFER OF NAME.
     MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
     MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
     CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
           NAME-A
           NUM
           NAME-COMPONENT-A.
     CALL "COSNAMING-NAMINGCONTEXT-RESOLVE" USING
              COS-NAMING
              NAME
              ENV
              OBJ.
     MOVE "COSNAMING-NAMINGCONTEXT-RESOLVE" TO MESS.
     PERFORM ENV-CHECK.
* CALL fixed program
     CALL "EXEC-METHOD" USING
                 OBJ
                 NUM
                 RET.
     CALL "CORBA-OBJECT-RELEASE" USING OBJ ENV.
     CALL "CORBA-OBJECT-RELEASE" USING COS-NAMING ENV.
     MOVE "CORBA-OBJECT-RELEASE" TO MESS.
     PERFORM ENV-CHECK.
     CALL "CORBA-FREE" USING SEQ-BUFFER OF NAME.

     STOP RUN.
 MAIN-END.

 ENV-CHECK SECTION.
     EVALUATE TRUE
         WHEN CORBA-NO-EXCEPTION OF MAJOR OF ENV
              CONTINUE
         WHEN CORBA-USER-EXCEPTION OF MAJOR OF ENV
              DISPLAY "USER-EXCEPTION   : " MESS
              MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
              CALL "CORBA-STRING-GET" USING
                                   IDL-ID OF ENV
                                   STRING-LENGTH
```

```
                                MESS
            DISPLAY "ID : " MESS
            EXIT PROGRAM
        WHEN CORBA-SYSTEM-EXCEPTION OF MAJOR OF ENV
            DISPLAY "SYSTEM-EXCEPTION : " MESS
            MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
            CALL "CORBA-STRING-GET" USING
                                IDL-ID OF ENV
                                STRING-LENGTH
                                MESS
            DISPLAY "ID : " MESS
            EXIT PROGRAM
    END-EVALUATE.
 ENV-CHECK-END.
    EXIT.
*
 END PROGRAM "CLIENT-MAIN".
```

(3)サーバアプリケーションで扱うパラメタ

サーバアプリケーションでの処理例を以下に示します。

```
  IDENTIFICATION DIVISION.
  PROGRAM-ID. "ODSAMPLE-FIXEDTEST-OP1".
  AUTHOR. OD/IDLcompiler Ver.2.0.
  INSTALLATION. IDL FILE NAME IS simple.idl.
  SECURITY. THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
            WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
            GURANTEED IS NOT DONE.
  DATE-WRITTEN. TUE MAY  6 11:03:40 1997
* ALL RIGHTS RESERVED, COPYRIGHT (C) FUJITSU LIMITED 1998
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
*
  INPUT-OUTPUT SECTION.
*
  DATA DIVISION.
*
  WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.

  LINKAGE SECTION.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 FIXED1 PIC SVP(5)9(5) PACKED-DECIMAL.
  01 FIXED2 PIC S9(10)P(3) PACKED-DECIMAL.
  01 FIXED3 PIC SV9(10)    PACKED-DECIMAL.
  01 FIXED0 PIC S9(7)V9(3) PACKED-DECIMAL.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.

*
  PROCEDURE DIVISION USING
          OBJ
          FIXED1
          FIXED2
          FIXED3
          ENV
          FIXED0.
*
  MAIN.
*
* SET OUT PARAMETER
      MOVE   00 TO FIXED2.
```

```
*
* SET INOUT PARAMETER.
     MOVE 0.88888  TO FIXED3.
*
* SET RETURN.
     MOVE 12345.345 TO FIXED0.
*
 MAIN-END.
*
 END PROGRAM  "ODSAMPLE-FIXEDTEST-OP1".
```

6.8.8 配列

(1)IDLマッピング

IDL言語で配列を指定した場合、COBOLのOCCURS句でデータ宣言します。配列名は“モジュール名-インタフェース名-配列名-V”となり、用途に応じてCOPY文で定義します。配列のインデックスは、1から始まり<配列の大きさ>までとなります。また、配列の領域獲得関数(関数名:“モジュール名-インタフェース名-配列名-ALLOC”、以降XX-ALLOC関数と呼ぶ)がIDLコンパイラで生成されます。以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface  arraytest{
        typedef long    fix[4][3][2];      // 配列(固定長)
        typedef string  str[2][3][4];      // 配列(可変長)
        fix    op1(in fix para1, out fix para2, inout fix para3 );
        str    op2(in str para1, out str para2, inout str para3 );
    };
};
```

COBOL

```
 01 ODSAMPLE-ARRAYTEST-FIX.
   02 FILLER OCCURS 4.
     03 FILLER OCCURS 3.
       04 FILLER OCCURS 2.
         05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-FIX-V.
 01 ODSAMPLE-ARRAYTEST-STR.
   02 FILLER OCCURS 2.
     03 FILLER OCCURS 3.
       04 FILLER OCCURS 4.
         05 ODSAMPLE-ARRAYTEST-STR-V USAGE IS POINTER.
```

(2)クライアントアプリケーションで扱うパラメタ(固定長)

配列(固定長)のin、out、inoutパラメタを扱う場合、領域の獲得／解放を行う必要はありません。復帰値はスタブで自動的に配列領域が獲得されるため、アプリケーションで不要になった時点でCORBA-FREE関数で解放する必要があります。
クライアントアプリケーションでの処理例を以下に示します。

```
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
   COPY CONST IN CORBA.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 PARA1.
     02 FILLER OCCURS 4.
```

```
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 2.
           05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-PARA1-V.
   01 PARA2.
     02 FILLER OCCURS 4.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 2.
           05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-PARA2-V.
   01 PARA3.
     02 FILLER OCCURS 4.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 2.
           05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-PARA3-V.
   01 FIX0.
     02 FILLER OCCURS 4.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 2.
           05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-FIX0-V.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY J.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY K.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.

 PROCEDURE DIVISION.

  PERFORM VARYING I FROM 1 BY 1 UNTIL I > 4
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
            PERFORM VARYING K FROM 1 BY 1 UNTIL K > 2
                MOVE I TO ODSAMPLE-ARRAYTEST-PARA1-V( I J K )
                COMPUTE NUM = I * 10
                MOVE NUM TO ODSAMPLE-ARRAYTEST-PARA3-V( I J K )
            END-PERFORM
       END-PERFORM
  END-PERFORM.

 CALL "ODSAMPLE-ARRAYTEST-OP1" USING
       OBJ
       PARA1
       PARA2
       PARA3
       ENV
       FIX0.
```

(3)クライアントアプリケーションで扱うパラメタ(可変長)

配列(可変長)のinoutパラメタ／outパラメタ／復帰値は、スタブでXX-ALLOC関数により配列領域が獲得されるため、アプリケーションで不要になった時点でCORBA-FREE関数で解放する必要があります。
クライアントアプリケーションでの処理例を以下に示します。

```
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
    COPY CONST IN CORBA.
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
    01 STR1.
      02 FILLER OCCURS 2.
        03 FILLER OCCURS 3.
          04 FILLER OCCURS 4.
            05 ODSAMPLE-ARRAYTEST-STR1-V USAGE IS POINTER.
    01 STR2 USAGE POINTER.
    01 STR3 USAGE POINTER.
```

```
   01 STR0 USAGE POINTER.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY J.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY K.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
   01 STR1 X(20).
   01 STR2 X(20).
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
   LINKAGE SECTION.
   01 STR3-P.
     02 FILLER OCCURS 2.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 4.
           05 ODSAMPLE-ARRAYTEST-STR3-V USAGE IS POINTER.

 PROCEDURE DIVISION.

CALL "ODSAMPLE-ARRAYTEST-STR-ALLOC" USING  STR3.
SET ADDRESS OF STR3-P TO STR3.

PERFORM VARYING I FROM 1 BY 1 UNTIL I > 2
      PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
            PERFORM VARYING K FROM 1 BY 1 UNTIL K > 4
                  MOVE "str1[I][J][K]" TO STR1-D
                  MOVE "str3[I][J][K]" TO STR2-D

                  EVALUATE I
                    WHEN 1
                      INSPECT STR1-D REPLACING ALL "I" BY "0"
                      INSPECT STR2-D REPLACING ALL "I" BY "0"
                    WHEN 2
                      INSPECT STR1-D REPLACING ALL "I" BY "1"
                      INSPECT STR2-D REPLACING ALL "I" BY "1"
                  END-EVALUATE

                  EVALUATE J
                    WHEN 1
                      INSPECT STR1-D REPLACING ALL "J" BY "0"
                      INSPECT STR2-D REPLACING ALL "J" BY "0"
                    WHEN  2
                      INSPECT STR1-D REPLACING ALL "J" BY "1"
                      INSPECT STR2-D REPLACING ALL "J" BY "1"
                    WHEN  3
                      INSPECT STR1-D REPLACING ALL "J" BY "2"
                      INSPECT STR2-D REPLACING ALL "J" BY "2"
                  END-EVALUATE

                  EVALUATE K
                    WHEN 1
                      INSPECT STR1-D REPLACING ALL "K" BY "0"
                      INSPECT STR2-D REPLACING ALL "K" BY "0"
                    WHEN 2
                      INSPECT STR1-D REPLACING ALL "K" BY "1"
                      INSPECT STR2-D REPLACING ALL "K" BY "1"
                    WHEN 3
                      INSPECT STR1-D REPLACING ALL "K" BY "2"
                      INSPECT STR2-D REPLACING ALL "K" BY "2"
                    WHEN 4
                      INSPECT STR1-D REPLACING ALL "K" BY "3"
                      INSPECT STR2-D REPLACING ALL "K" BY "3"
                  END-EVALUATE

                  MOVE FUNCTION LENG(STR1-D) TO LSIZE
```

```
                CALL "CORBA-STRING-SET" USING
                   ODSAMPLE-ARRAYTEST-STR1-V( I J K )
                   LSIZE
                   STR1-D
                MOVE FUNCTION LENG(STR2-D) TO LSIZE
                CALL "CORBA-STRING-SET" USING
                   ODSAMPLE-ARRAYTEST-STR3-V( I J K )
                   LSIZE
                   STR2-D
           END-PERFORM
       END-PERFORM
  END-PERFORM.

  CALL "ODSAMPLE-ARRAYTEST-OP2" USING
    OBJ
    STR1
    STR2
    STR3
    ENV
    STR0.
* inパラメタは、データ値の設定時に獲得した領域のみを解放してください。
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > 2
       PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
            PERFORM VARYING K FROM 1 BY 1 UNTIL K > 4
                 CALL "CORBA-FREE" USING  ODSAMPLE-ARRAYTEST-STR1-V( I J K )
            END-PERFORM
     END-PERFORM
  END-PERFORM.

  CALL "CORBA-FREE"  USING
    STR0.
  CALL "CORBA-FREE"  USING
    STR2.
  CALL "CORBA-FREE"  USING
    STR3.
```

(4)サーバアプリケーションで扱うパラメタ(固定長)

サーバアプリケーションの処理例を以下に示します。

```
   PROGRAM-ID. "ODSAMPLE-ARRAYTEST-OP1"
   ENVIRONMENT DIVISION.
   DATA DIVISION.
   WORKING-STORAGE SECTION.
     COPY CONST IN CORBA.
    LINKAGE SECTION.
     01 ENV.
       03 MAJOR PIC 9(9) COMP-5.
         88 CORBA-NO-EXCEPTION VALUE 0.
         88 CORBA-USER-EXCEPTION VALUE 1.
         88 CORBA-SYSTEM-EXCEPTION VALUE 2.
       03 IDL-ID USAGE POINTER.
       03 MINOR PIC 9(9) COMP-5.
       03 IDL-STATUS PIC 9(9) COMP-5.
       03 PARAM USAGE POINTER.
       03 MAGIC PIC 9(9) COMP-5.
     01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
     01 PARA1.
       02 FILLER OCCURS 4.
         03 FILLER OCCURS 3.
           04 FILLER OCCURS 2.
             05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-PARA1-V.
```

```
   01 PARA2.
     02 FILLER OCCURS 4.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 2.
           05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-PARA2-V.
   01 PARA3.
     02 FILLER OCCURS 4.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 2.
            05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-PARA3-V.
   01 FIX.
     02 FILLER OCCURS 4.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 2.
           05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-FIX-V.
           05 ODSAMPLE-ARRAYTEST-FIX-V USAGE IS POINTER.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY J.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY K.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.

 PROCEDURE DIVISION USING OBJ PARA1 PARA2 PARA3 ENV FIX.

*outパラメタの処理
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > 4
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
              PERFORM VARYING K FROM 1 BY 1 UNTIL K > 2
                    COMPUTE NUM = I + J + K
                    MOVE NUM TO ODSAMPLE-ARRAYTEST-PARA2-V( I J K )
              END-PERFORM
        END-PERFORM
  END-PERFORM.
*inoutパラメタの処理
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > 4
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
              PERFORM VARYING K FROM 1 BY 1 UNTIL K > 2
                    COMPUTE NUM = (I + J + K) * 10
                    MOVE NUM TO ODSAMPLE-ARRAYTEST-PARA3-V( I J K )
              END-PERFORM
        END-PERFORM
  END-PERFORM.
*復帰値の処理
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > 4
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
              PERFORM VARYING K FROM 1 BY 1 UNTIL K > 2
                    COMPUTE NUM = I * J * K
                    MOVE NUM TO ODSAMPLE-ARRAYTEST-FIX-V( I J K )
              END-PERFORM
        END-PERFORM
  END-PERFORM.
```

(5)サーバアプリケーションで扱うパラメタ(可変長)

配列(可変長)のin／inoutパラメタを扱う場合、配列領域はスケルトンで自動的に獲得／解放されます。out／復帰値には、XX-ALLOC関数で配列領域を獲得してデータを設定します。inoutパラメタで領域を拡張する場合、一度CORBA-FREE関数でデータ域を解放し、再度領域を獲得します。ここで獲得した領域はスケルトンで自動的に解放されます。
サーバアプリケーションでの処理例を以下に示します。

```
 PROGRAM-ID. "ODSAMPLE-ARRAYTEST-OP2"
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
```

```
   COPY CONST IN CORBA.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY J.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY K.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
   01 STR-OUT PIC X(20).
   01 STR-IO PIC X(20).
   01 STR-RET PIC X(20).
  LINKAGE SECTION.
   01 ENV.
     03 MAJOR PIC 9(9) COMP-5.
       88 CORBA-NO-EXCEPTION VALUE 0.
       88 CORBA-USER-EXCEPTION VALUE 1.
       88 CORBA-SYSTEM-EXCEPTION VALUE 2.
     03 IDL-ID USAGE POINTER.
     03 MINOR PIC 9(9) COMP-5.
     03 IDL-STATUS PIC 9(9) COMP-5.
     03 PARAM USAGE POINTER.
     03 MAGIC PIC 9(9) COMP-5.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 PARAM1.
     02 FILLER OCCURS 2.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 4.
           05 ODSAMPLE-ARRAYTEST-PARAM1-V USAGE IS POINTER.

   01 PARAM2-P.
     02 FILLER OCCURS 2.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 4.
           05 ODSAMPLE-ARRAYTEST-PARAM2-V USAGE IS POINTER.
   01 PARAM2 USAGE POINTER.
   01 PARAM3-P.
     02 FILLER OCCURS 2.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 4.
           05 ODSAMPLE-ARRAYTEST-PARAM3-V USAGE IS POINTER.
   01 PARAM3 USAGE POINTER.
   01 PARAMO-P.
     02 FILLER OCCURS 2.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 4.
           05 ODSAMPLE-ARRAYTEST-PARAMO-V USAGE IS POINTER.
   01 PARAMO USAGE POINTER.

PROCEDURE DIVISION USING OBJ PARAM1 PARAM2 PARAM3 ENV PARAMO.

* outパラメタの処理
CALL "ODSAMPLE-ARRAYTEST-STR-ALLOC" USING
     PARAM2.

SET ADDRESS  OF PARAM2-P TO PARAM2.
PERFORM VARYING I FROM 1 BY 1 UNTIL I > 2
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
              PERFORM VARYING K FROM 1 BY 1 UNTIL K > 4

                   MOVE "str2[I][J][K]" TO STR-OUT

                   EVALUATE I
                     WHEN 1
                       INSPECT STR-OUT REPLACING ALL "I" BY "0"
                     WHEN 2
                       INSPECT STR-OUT REPLACING ALL "I" BY "1"
```

```
                END-EVALUATE

                EVALUATE J
                  WHEN 1
                    INSPECT STR-OUT REPLACING ALL "J" BY "0"
                  WHEN  2
                    INSPECT STR-OUT REPLACING ALL "J" BY "1"
                  WHEN  3
                    INSPECT STR-OUT REPLACING ALL "J" BY "2"
                END-EVALUATE

                EVALUATE K
                  WHEN 1
                    INSPECT STR-OUT REPLACING ALL "K" BY "0"
                  WHEN 2
                    INSPECT STR-OUT REPLACING ALL "K" BY "1"
                  WHEN 3
                    INSPECT STR-OUT REPLACING ALL "K" BY "2"
                  WHEN 4
                    INSPECT STR-OUT REPLACING ALL "K" BY "3"
                END-EVALUATE

              MOVE FUNCTION LENG(STR-OUT) TO LSIZE

              CALL "CORBA-STRING-SET" USING
                    ODSAMPLE-ARRAYTEST-PARAM2-V( I J K )
                    LSIZE
                    STR-OUT

            END-PERFORM
       END-PERFORM
END-PERFORM.

* inoutパラメタの処理
SET ADDRESS  OF PARAM3-P TO PARAM3.
PERFORM VARYING I FROM 1 BY 1 UNTIL I > 2
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
              PERFORM VARYING K FROM 1 BY 1 UNTIL K > 4
                  CALL "CORBA-FREE" USING ODSAMPLE-ARRAYTEST-PARAM3-V( I J K )

                  MOVE "str3[I][J][K]" TO STR-IO

                  EVALUATE I
                    WHEN 1
                      INSPECT STR-IO REPLACING ALL "I" BY "0"
                    WHEN 2
                      INSPECT STR-IO REPLACING ALL "I" BY "1"
                  END-EVALUATE

                  EVALUATE J
                    WHEN 1
                      INSPECT STR-IO REPLACING ALL "J" BY "0"
                    WHEN  2
                      INSPECT STR-IO REPLACING ALL "J" BY "1"
                    WHEN  3
                      INSPECT STR-IO REPLACING ALL "J" BY "2"
                  END-EVALUATE

                  EVALUATE K
                    WHEN 1
                      INSPECT STR-IO REPLACING ALL "K" BY "0"
                    WHEN 2
                      INSPECT STR-IO REPLACING ALL "K" BY "1"
```

```
                    WHEN 3
                      INSPECT STR-IO REPLACING ALL "K" BY "2"
                    WHEN 4
                      INSPECT STR-IO REPLACING ALL "K" BY "3"
                  END-EVALUATE

                MOVE FUNCTION LENG(STR-IO) TO LSIZE

                CALL "CORBA-STRING-SET" USING
                      ODSAMPLE-ARRAYTEST-PARAM3-V( I J K )
                      LSIZE
                      STR-IO

              END-PERFORM
        END-PERFORM
  END-PERFORM.

* 復帰値の処理
  CALL "ODSAMPLE-ARRAYTEST-STR-ALLOC" USING
       PARAMO.

  SET ADDRESS  OF PARAMO-P TO PARAMO.
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > 2
          PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
                PERFORM VARYING K FROM 1 BY 1 UNTIL K > 4

                   MOVE "str0[I][J][K]" TO STR-RET

                   EVALUATE I
                     WHEN 1
                       INSPECT STR-RET REPLACING ALL "I" BY "0"
                     WHEN 2
                       INSPECT STR-RET REPLACING ALL "I" BY "1"
                   END-EVALUATE

                   EVALUATE J
                     WHEN 1
                       INSPECT STR-RET REPLACING ALL "J" BY "0"
                     WHEN  2
                       INSPECT STR-RET REPLACING ALL "J" BY "1"
                     WHEN  3
                       INSPECT STR-RET REPLACING ALL "J" BY "2"
                   END-EVALUATE

                   EVALUATE K
                     WHEN 1
                       INSPECT STR-RET REPLACING ALL "K" BY "0"
                     WHEN 2
                       INSPECT STR-RET REPLACING ALL "K" BY "1"
                     WHEN 3
                       INSPECT STR-RET REPLACING ALL "K" BY "2"
                     WHEN 4
                       INSPECT STR-RET REPLACING ALL "K" BY "3"
                   END-EVALUATE

                MOVE FUNCTION LENG(STR-RET) TO LSIZE

                CALL "CORBA-STRING-SET" USING
                      ODSAMPLE-ARRAYTEST-PARAMO-V( I J K )
                      LSIZE
                      STR-RET

                END-PERFORM
```

```
            END-PERFORM
        END-PERFORM.
    EXIT PROGRAM.
END PROGRAM "ODSAMPLE-ARRAYTEST-OP2".
```

6.8.9 属性宣言(attribute)のマッピング

(1)IDLマッピング

IDL言語で属性宣言(attribute)を指定した場合、オブジェクトのデータ設定／取得を行う関数（関数名:“モジュール名-インタフェース名--set(get)-変数名”。以降、データ設定関数／データ設定関数と呼ぶ）がIDLコンパイラで生成されます。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
        interface       attrtest{
                attribute long  para1;
                attribute string para2;
                readonly attribute long para3;
        };
};
```

COBOL

```
   IDENTIFICATION DIVISION.
   PROGRAM-ID. "ODSAMPLE-ATTRTEST--SET-PARA1".
   ENVIRONMENT DIVISION.
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   COPY CONST IN CORBA.
  *オブジェクトリファレンス
    01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  * 設定値
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM
  *例外情報
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
```

(2)クライアントアプリケーションで扱うパラメタ

データ設定関数でデータ設定に使用するデータ域は、inパラメタと同様に扱われます。また、データ取得関数でデータ取得に使用するデータ域は復帰値と同様に扱われます。このため、可変長データでは、その領域が不要になった時点でCORBA-FREE関数で領域を解放する必要があります。

```
.IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-ATTRTEST".
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 COPY CONST IN CORBA.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY RET-NUM.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR1 USAGE POINTER.
  01 STR-WORK PIC X(30).
  01 RET-STR1 USAGE POINTER.
  01 ERR-MSG PIC X(30).
```

```
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.

PROCEDURE DIVISION.

 MAIN.

  MOVE  2 TO NUM.
  CALL "ODSAMPLE-ATTRTEST--SET-PARA1" USING
        OBJ
        NUM
        ENV.
  CALL "ODSAMPLE-ATTRTEST--GET-PARA1" USING
        OBJ
        ENV
        RET-NUM.
  DISPLAY "ODsample_attrtest_get_para1 returns " RET-NUM.

  MOVE "test" TO STR-WORK.
  MOVE 5 TO LSIZE.
  CALL "CORBA-STRING-SET" USING
        STR1
        LSIZE
        STR-WORK.
  CALL "ODSAMPLE-ATTRTEST--SET-PARA2" USING
        OBJ
        STR1
        ENV.
  CALL "ODSAMPLE-ATTRTEST--GET-PARA2" USING
        OBJ
        ENV
        RET-STR1.

  MOVE "ODsample_attrtest__get_para2" TO ERR-MSG.
  PERFORM ENV-CHECK.

  DISPLAY "ODsample_attrtest_get_para2 returns " RET-STR1.

  CALL "CORBA-FREE" USING
        RET-STR1.
  CALL "ODSAMPLE-ATTRTEST--GET-PARA3" USING
        OBJ
        ENV
        RET-NUM.

  DISPLAY "ODsample_attrtest_get_para3 returns " RET-NUM.
  MAIN-END.

  ENV-CHECK SECTION.
      EVALUATE TRUE
          WHEN CORBA-NO-EXCEPTION OF MAJOR OF ENV
               CONTINUE
          WHEN CORBA-USER-EXCEPTION OF MAJOR OF ENV
  DISPLAY "USER-EXCEPTION   : " MESS
  MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
 CALL "CORBA-STRING-GET" USING
        IDL-ID OF ENV
        STRING-LENGTH
        MESS.
  DISPLAY "ID : " MESS
  EXIT PROGRAM
  WHEN CORBA-SYSTEM-EXCEPTION OF MAJOR OF ENV
       DISPLAY "SYSTEM-EXCEPTION : " MESS
       MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
```

```
        CALL "CORBA-STRING-GET" USING
              IDL-ID OF ENV
              STRING-LENGTH
              MESS
        DISPLAY "ID : " MESS
        EXIT PROGRAM
    END-EVALUATE.
    ENV-CHECK-END.
    EXIT.
```

(3)サーバアプリケーションで扱うパラメタ

データ設定関数でデータ設定に使用するデータ域はinパラメタと同様に扱われ、スケルトンで自動的に獲得／解放されます。データ取得関数でデータ取得に使用するデータ域は、復帰値と同様に扱われます。可変長データでは、データ域獲得関数で領域獲得する必要があります。獲得した領域は、スケルトンで自動的に解放されます。
関数間でデータを受け渡すには、GLOBAL句で変数を定義します。
以降の例では、変数GNUM／GSTRを使用します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-ATTRTEST-GET-PARA1".
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
  COPY CONST IN CORBA.

 LINKAGE SECTION.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 ENV.
     03 MAJOR PIC 9(9) COMP-5.
        88 CORBA-NO-EXCEPTION VALUE 0.
        88 CORBA-USER-EXCEPTION VALUE 1.
        88 CORBA-SYSTEM-EXCEPTION VALUE 2.
     03 IDL-ID USAGE POINTER.
     03 MINOR PIC 9(9) COMP-5.
     03 IDL-STATUS PIC 9(9) COMP-5.
     03 PARAM USAGE POINTER.
     03 MAGIC PIC 9(9) COMP-5.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY RET-NUM.

  PROCEDURE DIVISION DIVISION USING
      OBJ
      ENV.

 MAIN.

  MOVE  GNUM TO RET-NUM.

 MAIN-END.
 END PROGRAM "ODSAMPLE-ATTRTEST--GET-PARA1".

 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODSAMPLE-ATTRTEST--SET-PARA1".
 ENVIRONMENT DIVISION.
 DATA DIVISION.
WORKING-STORAGE SECTION.
 COPY CONST IN CORBA.

 LINKAGE SECTION.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
  01 ENV.
     03 MAJOR PIC 9(9) COMP-5.
```

```
        88 CORBA-NO-EXCEPTION VALUE 0.
        88 CORBA-USER-EXCEPTION VALUE 1.
        88 CORBA-SYSTEM-EXCEPTION VALUE 2.
    03 IDL-ID USAGE POINTER.
    03 MINOR PIC 9(9) COMP-5.
    03 IDL-STATUS PIC 9(9) COMP-5.
    03 PARAM USAGE POINTER.
    03 MAGIC PIC 9(9) COMP-5.

PROCEDURE DIVISION DIVISION USING
     OBJ
     NUM
     ENV.

MAIN.

     MOVE  NUM TO GNUM.

MAIN-END.
END PROGRAM "ODSAMPLE-ATTRTEST--SET-PARA1".

IDENTIFICATION DIVISION.
PROGRAM-ID. "ODSAMPLE-ATTRTEST--GET-PARA2".
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY CONST IN CORBA.
 01 WSTR USAGE IS POINTER.
LINKAGE SECTION.
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
 01 ENV.
    03 MAJOR PIC 9(9) COMP-5.
       88 CORBA-NO-EXCEPTION VALUE 0.
       88 CORBA-USER-EXCEPTION VALUE 1.
       88 CORBA-SYSTEM-EXCEPTION VALUE 2.
    03 IDL-ID USAGE POINTER.
    03 MINOR PIC 9(9) COMP-5.
    03 IDL-STATUS PIC 9(9) COMP-5.
    03 PARAM USAGE POINTER.
    03 MAGIC PIC 9(9) COMP-5.
 01 STR1 USAGE IS POINTER.

PROCEDURE DIVISION DIVISION USING
     OBJ
     ENV
     STR1.

MAIN.

    MOVE GSTR TO STR1.

MAIN-END.
END PROGRAM "ODSAMPLE-ATTRTEST--GET-PARA2".

IDENTIFICATION DIVISION.
PROGRAM-ID. "ODSAMPLE-ATTRTEST--SET-PARA2".
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY CONST IN CORBA.
LINKAGE SECTION.
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
 01 STR1 USAGE IS POINTER.
```

```
 01 ENV.
    03 MAJOR PIC 9(9) COMP-5.
       88 CORBA-NO-EXCEPTION VALUE 0.
       88 CORBA-USER-EXCEPTION VALUE 1.
       88 CORBA-SYSTEM-EXCEPTION VALUE 2.
    03 IDL-ID USAGE POINTER.
    03 MINOR PIC 9(9) COMP-5.
    03 IDL-STATUS PIC 9(9) COMP-5.
    03 PARAM USAGE POINTER.
    03 MAGIC PIC 9(9) COMP-5.

PROCEDURE DIVISION DIVISION USING
     OBJ
     STR1
     ENV.

MOVE STR1 TO GSTR

MAIN-END.
END PROGRAM "ODSAMPLE-ATTRTEST--SET-PARA2".

IDENTIFICATION DIVISION.
PROGRAM-ID. "ODSAMPLE-ATTRTEST--GET-PARA3".
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY CONST IN CORBA.
LINKAGE SECTION.
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
 01 ENV.
    03 MAJOR PIC 9(9) COMP-5.
       88 CORBA-NO-EXCEPTION VALUE 0.
       88 CORBA-USER-EXCEPTION VALUE 1.
       88 CORBA-SYSTEM-EXCEPTION VALUE 2.
    03 IDL-ID USAGE POINTER.
    03 MINOR PIC 9(9) COMP-5.
    03 IDL-STATUS PIC 9(9) COMP-5.
    03 PARAM USAGE POINTER.
    03 MAGIC PIC 9(9) COMP-5.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY RET-NUM.

PROCEDURE DIVISION DIVISION USING
     OBJ
     ENV
     RET-NUM.

 MAIN.

 MOVE GNUM TO RET-NUM.

MAIN-END.
END PROGRAM "ODSAMPLE-ATTRTEST--GET-PARA3".
```

6.8.10 パラメタ受渡しで使用するデータ型

クライアント／サーバアプリケーションでパラメタ受渡しに使用するデータ型を以下に示します。

CORBAデータ型	in	out	inout	復帰値
long	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE

CORBAデータ型		in	out	inout	復帰値
short		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
unsigned long		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
unsigned short		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
long long		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
float		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
double		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
char		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
wchar		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
octet		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
boolean		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
enum		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
string	bound	TEXT	TEXT	TEXT	TEXT
	unbound	STRING	STRING	STRING	STRING
wstring	bound	TEXT	TEXT	TEXT	TEXT
	unbound	STRING	STRING	STRING	STRING
any		BY REFERENCE	POINTER	POINTER	POINTER
sequence		BY REFERENCE	POINTER	POINTER	POINTER
struct	固定長	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
	可変長	BY REFERENCE	POINTER	POINTER	POINTER
union	固定長	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
	可変長	BY REFERENCE	POINTER	POINTER	POINTER
array(配列)	固定長	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
	可変長	BY REFERENCE	POINTER	POINTER	POINTER
Object		BY REFERENCE	POINTER	POINTER	POINTER
TypeCode		BY REFERENCE	POINTER	POINTER	POINTER

注）POINTERは、POINTERがBY REFERENCEで渡されます。
TEXTは、COBOLの固定長テキストでNULL文字を含みません。
STRINGは、NULLで終わる可変長文字列のポインタです。

注意

(1)NULLポインタの設定について

サーバアプリケーションでのout／inoutパラメタ、復帰値、およびクライアントアプリケーションでのin／inoutパラメタでは、文字列型、シーケンス型、構造体、共用体、配列データ型でNULLポインタを設定できません。

(2)データ型の扱いについて

- 固定長sequenceの場合、lengthにmaximumを超える値を設定した場合、システム例外が通知されます。また、固定長のsequenceの変数は以下のようにマッピングされます。

  IDLファイル

```
module mod {
    typedef sequence<long,10> seq_fix;
};
```

  COBOL

```
   DATA DIVISION.
   WORKING-STORAGE SECTION.
     01 MOD-SEQ-FIX.
        02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
        02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
        02 SEQ-BUFFER USAGE IS POINTER.
        02 FILLER OCCURS 10.
           03 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-VALUE.
```

- 可変長sequenceの場合、sequenceの変数は以下のようにマッピングされます。各要素にアクセスする場合、CORBA-SEQUENCE-ELEMENT-SET関数／CORBA-SEQUENCE-ELEMENT-GET関数を使用します。

  IDLファイル

```
module mod {
    typedef sequence<long> seq_val;
};
```

  COBOL

```
   DATA DIVISION.
   WORKING-STORAGE SECTION.
     01 MOD-SEQ-VAL.
        02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
        02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
        02 SEQ-BUFFER USAGE IS POINTER.
```

- arrayの場合、arrayの各要素は以下のようにマッピングされます。

  IDLファイル

```
module mod {
    typedef long S_array[2][3][4];
};
```

  COBOL

```
   DATA DIVISION.
   WORKING-STORAGE SECTION.
     01 MOD-S-ARRAY.
        02 FILLER OCCURS 2.
           03 FILLER OCCURS 3.
              04 FILLER OCCURS 4.
                 05 COPY LONG IN CORBA REPLACING CORBA-LONG BY MOD-S-ARRAY-V.
```

ー arrayの場合、例外処理は以下のようにマッピングされます。

IDLファイル

```
module mod {
    exception foo {
        long ex_val;
    };
};
```

COBOL

```
   DATA DIVISION.
   WORKING-STORAGE SECTION.
     01 EX-MOD-FOO PIC X(15) VALUE "IDL:mod/foo:1.0".
     01 MOD-FOO.
        02 COPY LONG IN CORBA REPLACING CORBA-LONG BY EX-VAL.
```

(3)COBOLとの整合性について

ー floatは、以下の構文で対応します。

```
[USAGE IS] [COMP-1]        : 4 Byte
```

ー doubleは、以下の構文で対応します。

```
[USAGE IS] [COMP-2]        : 8 Byte
```

ー constは、以下の構文で対応します。
OMG仕様

```
>> CONSTANT constant-name IS literal
```

COBOL仕様

```
SYMBOLIC CONSTANT
    { 記号定数-1 IS定数-1 }
```

ー typedefは、以下の構文で対応します(基本型のみ)。
CORBA登録集として以下の定義を用意します(longの場合)。

```
long-type  usage(local long type)/
```

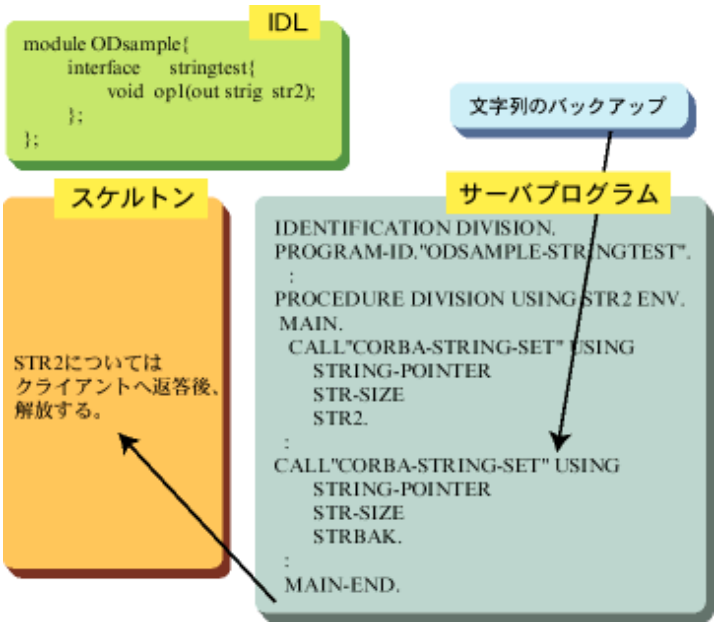
ユーザは、COPY句でtypedefを使用します。

```
    WORKING STORAGE SECTION.
        :
    01 COPY LONG IN CORBA
        REPLACING long-type WITH ws--long-1
    01 COPY LONG IN CORBA
        REPLACING long-type WITH ws--long-2
```

ー 構造体などのtypedefは、COBOLで未サポートのため、使用できません。

6.9 any型、sequence型のリリースフラグ

可変長データは、データをクライアントへ返却後、スケルトン中で自動的に解放します。
そのため、解放したくないデータについては、そのデータをどこか別の領域に複写しておく必要があります。
文字列(string)を例に説明します。



しかし、any、sequenceのような複雑な型の場合は、コピーする処理が複雑なため、リリースフラグを設定し、解放する／しないを設定できるようになっています。
リリースフラグの設定方法について述べます。

変数型	設定方法
any	CORBA-ANY-SET-RELEASEを使用します。デフォルトは、CORBA-FALSE-VALUEです。(注) 例） CALL "CORBA-ANY-ALLOC" USING ANY-VAL. CALL "CORBA-ANY-SET-RELEASE" USING ANY-POINTER REL-FLAG.
sequence	CORBA-SEQUENCE-SET-RELEASEを使用します。デフォルトは、CORBA-FALSE-VALUEです。(注) 例） struct sequence<long> data; －＞ CALL "CORBA-DATA-ALLOC" USING SEQ-POINTER. CALL "CORBA-SEQUENCE-SET-RELEASE" USING SEQ-POINTER REL-FLAG.

注) CORBA-TRUE-VALUEのとき解放されます。CORBA-FALSE-VALUEのときは、解放されません。

6.10 COBOLアプリケーション使用時の注意事項

参照

“第2章 CORBAアプリケーション開発時の留意事項”についても参照してください。

メッセージについて

- COBOLアプリケーション実行時に、COBOLランタイムライブラリからメッセージが表示される場合があります。JMPではじまるメッセージが出力されていた場合は、「COBOL使用手引書」(実行時のメッセージ)を参照してください。

関数名／変数名の扱い

- 関数名の長さは、60文字までです。60文字を超える場合、COBOLコンパイラにてコンパイルエラーとなるため、注意してください。IDLファイルを定義する場合、module名、interface名、オペレーション名を合わせた長さが55文字以上になると、各生成するファイルの関数名が60文字を超える場合があるため、注意してください。
- 変数名の長さは、30文字までです。30文字を超える場合、30文字以降を切り捨てます。
- 変数名の文字長が30文字を超える場合で、同一名になる場合、27文字目以降を数値に置き換えます。
- アンダースコア(_)は、COBOLで使用できないため、ハイフン(-)に置換します。ただし、オペレーション名、インタフェース名で使用されているアンダースコア(_)は、そのまま使用します。
- IDLファイルからクラス名と変数名の関係で、同じ名前に変換される場合は、以下のケースで変数の型などが不明となるため、使用しないでください。

```
typedef long foo_bar;
interface foo { typedef short bar; };
```

予約語の扱い

- COBOLの予約語の文字列がIDLファイルに記述されている場合、対応可能な予約語には文字列の前に“IDL-”を付加します。対応していない予約語の文字列は、IDLファイルに記述できません。予約語については、COBOLのマニュアルを参照してください。対応可能な予約語については、対応可能な予約語の一覧を参照してください。
- const宣言は、COBOLの“SYMBOLIC CONSTANT”となります。OMG規定では“>>CONSTANT”文節に対応しますが、COBOLで未サポートであるため、“SYMBOLIC CONSTANT”となります。

NVList使用方法

- COBOLでのNVListは、NamedValue構造体(登録集NAMEDVALUE.cbl)の配列を指すアドレスを格納するポインタです。
- パラメタは、CORBA-NVLIST-ADD-ITEMで設定します。このとき、以下の点に注意してください。

  LIST

  初期化(CORBA-ORB-CREATE-LIST使用)を行う必要があります。

  ITEMーNAME

  CORBA-STRING-SETで値を設定されたポインタを指定します。省略時は、NULLを指定してください。

  ITEMーTYPE

  CORBA-TYPECODE-FROM-CGEN-TCで作成されたTYPECODEを指定します。省略不可です。

  注意

  CORBA-ORB-TYPECODE-FROM-CGEN-TCに指定可能なパラメタは、登録集CONST.cblに記載されている基本型のTC-XXX、登録集で頭にTCのつくもの、またはIDLコンパイルで作成されるIDLファイル名_h.cblに定義されている、TC-XXX-IMPL-SEQを使用します。

VALUE

パラメタの値のアドレスを設定します。STRINGやOBJECTのようなポインタで扱われるデータも、FUNCTION ADDRでアドレスを取り出して設定します。ITEM-FLAGにCORBA-ARG-OUTなどを指定する際は、NULLを設定します。

VALUE-LEN

VALUEの領域の長さを指定します。省略時は、0を指定します。

ITEM-FLAGS

登録集FLAGS.cblを使用します。省略不可です。

- 領域の解放
NVList自体の解放時はCORBA-NVLIST-FREE、NVListの設定値だけの解放時はCORBA-NVLIST-FREE-MEMBERを使用します。

文字列の設定について

- 関数で文字列をパラメタに持つ場合、CORBA-STRING-SET関数に文字列を設定する必要があります。

例

```
*文字列の長さを求めます
 MOVE FUNCTION LENG (CORBA-ORB-OBJECTID-IMPLREP) TO STRING-LENGTH.
*文字列を作業用のTEMP-BUFに格納します
 CALL "CORBA-STRING-SET" USING
       TEMP-BUF
       STRING-LENGTH
       CORBA-ORB-OBJECTID-IMPLREP.
*関数を呼び出します
 CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
       ORB
       TEMP-BUF
       ENV
       IMPL-REP.
```

- PIC X(m)で宣言されたin/out/inoutの各文字型への代入は、MOVE命令で行います。ただし、COBOL以外にマッピングされたアプリケーションで扱える文字列長は、'¥0'を含みます。
- 日本語文字列を扱う場合、COBOLの日本語項目N(またはNC)から文字項目Xに直接代入できません。CORBA-STRING-SET関数／CORBA-WSTRING-SET関数にNULL文字を設定したあとに、使用してください。なお、文字列の取出しは、CORBA-STRING-GET関数／CORBA-WSTRING-GET関数を使用します。

Windows(R)アプリケーション作成時／運用時の注意事項 Windows32/64

- サーバアプリケーションは、動的リンク構造(主プログラムのロード時に、呼び出される関数すべてをロードする)で作成する必要があります。
動的プログラム構造(ロードが関数の呼出し時に行われる)では作成できないため、アプリケーションの翻訳時は、翻訳オプションにDLOADオプションを指定しないでください。
- サーバアプリケーションのメソッド部では、DISPLAY文を使用しないでください。
- サーバアプリケーションで各インタフェース使用時にオブジェクトへのポインタが渡された場合、ユーザ例外を返すことがあります(例外コード:“BAD PARAM”)。ENVIRONMENTをパラメタに持つ場合は、例外情報を参照してください。

- サーバアプリケーションをshared／unsharedタイプとしてインプリメンテーションリポジトリに登録する場合は、以下の設定を行ってください。また、インプリメンテーションリポジトリの登録では、設定項目envを使用しないでください。

  ー システムの環境変数に、以下の設定を追加します。

  ```
  @EnvSetWindow=UNUSE
  ```

  ー システムの環境変数PATHに、アプリケーションのメソッド実装部であるDLLファイルのパスを追加します。

- CORBAワークユニット運用で翻訳オプション“SSOUT(環境変数情報名)”および環境変数情報“@MessOutFile”を指定せずに、標準出力、標準エラー出力にデータを出力する場合、主プログラムの翻訳オプションに“MAIN(MAIN)”を指定してください。
- クライアントアプリケーションでは、BOA／ServerRequestインタフェースを使用できません。
- アプリケーションの運用前に、以下の環境変数を設定する必要があります。

```
@WinCloseMsg=OFF
```

- クライアントアプリケーションとサーバアプリケーションを同一フォルダに作成しないでください。クライアントアプリケーションとサーバアプリケーションとで同名のオブジェクトファイルが生成され、アプリケーションが異常な動作をする可能性があります。
- COBOLサーバアプリケーションで使用するダイナミックリンクライブラリの名称は、サーバアプリケーションのインタフェース宣言と合わせる必要があります。
  例を以下に示します。

  ー モジュール名：oddemo

  ー インタフェース名：calculator

  ー ダイナミックリンクライブラリ名：ODDEMO-CALCULATOR.DLL

- Windows64
  UNION型のマッピングが32bit版から変更されます。再度IDLコンパイラを実行し、ソースファイルを再作成してください。

Solaris／Linuxアプリケーション作成時／運用時の注意事項 Solaris32/64 Linux32/64

- Shift JIS環境でアプリケーションを運用する場合は、CBR_CODE_CHECK=noを設定してください。
- クライアントアプリケーションでは、BOA／ServerRequestインタフェースを使用できません。
- サーバアプリケーションは、動的リンク構造（主プログラムのロード時に、呼び出される関数すべてをロードする）で作成する必要があります。
  動的プログラム構造（ロードが関数の呼出し時に行われる）では、作成できないため、アプリケーションの翻訳時は、翻訳オプションにDLOADオプションを指定しないでください。
- COBOLサーバアプリケーションで使用するライブラリの名称は、サーバアプリケーションのインタフェース宣言と合わせる必要があります。

例

ー モジュール名：oddemo

ー インタフェース名：calculator

ー ライブラリ名：libODDEMO-CALCULATOR.so

Windows(R)アプリケーションの作成 Windows32

Windows(R)でCOBOLアプリケーションを作成する場合の注意点を以下に示します。

参照

- 詳細な翻訳・リンク方法については、NetCOBOLのマニュアルを参照してください。
- CORBAサービス(クライアント含む)が提供するライブラリについては、“B.1.2 ライブラリ”を参照してください。

ポイント

CORBAサービスのインストールフォルダを“C:¥Interstage¥ODWIN”として説明します。

クライアントアプリケーションの作成

クライアントアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。

翻訳・リンク手順

1. IDLコンパイルを実行します。

```
IDLc -cobol IDLファイル名
```

2. 1.で生成されたスタブ、CDRソースをCOBOLソースファイルに登録し、主プログラムと合わせて翻訳します。翻訳時に、「翻訳オプション」として以下のフォルダ配下の登録集を指定します。

```
CORBA=C:¥Interstage¥ODWIN¥INCLUDE¥COBOL
```

3. リンクするライブラリには、以下のファイルを指定します。
   [プロセスモード(UNICODE、UTF8用以外)の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODCOBCBL.LIB
   [スレッドモード(UNICODE、UTF8用以外)の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODCOBCBLMT.LIB
   [プロセスモード/スレッドモード(UNICODE、UTF8用)の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODCOBCBLUC.LIB

翻訳対象ファイル

```
IDLファイル名_STUB.CBL
IDLファイル名_CDR.CBL
```

翻訳オプション

[共通]

- NAMEオプションを指定してください。

[スレッドモード(UNICODE、UTF8用以外)の場合]

- THREAD(MULTI)オプションを指定してください。

[プロセスモード/スレッドモード(UNICODE、UTF8用)の場合]

- RCSオプション(UCS2-Unicode(UCS2)を使用する)を指定してください。
- スレッドモードの場合、THREAD(MULTI)オプションを指定してください。

翻訳時の「翻訳オプション」については、NetCOBOLのマニュアルを参照してください。

登録集の指定

以下の環境変数を設定してください。

```
CORBA=C:¥Interstage¥ODWIN¥INCLUDE¥COBOL
```

生成ファイル

スタブ、CDRソースで定義された目的オブジェクト

サーバアプリケーションの翻訳・リンク

サーバアプリケーション作成時の翻訳・リンク手順と指定するファイル・オプションを以下に示します。なお、詳細な翻訳・リンク方法については、NetCOBOLのマニュアルを参照してください。

翻訳・リンク手順

1. IDLコンパイルを実行します。

   ```
   IDLc -cobol  IDLファイル名
   ```

2. 1.で生成されたスケルトン、CDRソースを翻訳し、オブジェクトファイル(*.obj)を作成します。プロジェクトマネージャを起動し、「ツール」の「翻訳」を選択して「翻訳時の指定」のファイル・オプションを指定します。

3. サーバアプリケーションのメソッド部(サーバの実処理)を、同様の手順で翻訳します。詳細については、“Windows(R)アプリケーション作成時／運用時の注意事項”を参照してください。

4. 翻訳したオブジェクトを結合し、DLLファイルを作成します。DLLファイル名は、モジュール名-インタフェース名.DLLです。「ツール」の「リンク」を選択し、以下のファイルと共に指定します。
   [プロセスモード(UNICODE、UTF8用以外)の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODCOBCBLSV.LIB
   [スレッドモード(UNICODE、UTF8用以外)の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODCOBCBLMTSV.LIB
   [プロセスモード／スレッドモード(UNICODE、UTF8用)の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODCOBCBLSVUC.LIB

5. サーバアプリケーションのメイン処理を主プログラムとして、実行プログラムを作成します。このとき、以下のファイルと共に指定します。
   [プロセスモード(UNICODE、UTF8用以外)の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODCOBCBLSV.LIB
   [スレッドモード(UNICODE、UTF8用以外)の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODCOBCBLMTSV.LIB
   [プロセスモード／スレッドモード(UNICODE、UTF8用)の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODCOBCBLSVUC.LIB

6. OD_impl_instコマンドでサーバアプリケーションを登録します。実行時に指定する定義ファイルは、サンプルプログラムの以下のファイルを参照してください。
   C:¥Interstage¥ODWIN¥SRC¥SAMPLE¥COMPLEX¥SAMPLELIST.COBOL¥DATA¥*¥*.DEF
   注)modeには、SYNC_ENDを設定してください。
   DLLはDLL名だけを設定し、環境変数PATHにDLLのパスを追加してください。

翻訳対象ファイル

```
IDLファイル名_インタフェース名_SKEL.CBL
IDLファイル名_CDR.CBL
IDLファイル名_SKEL.CBL
```

注意

IDL定義により生成されない場合があります。

翻訳オプション

［共通］

－ NAMEオプションを指定してください。

－ DLOADオプションは指定しないでください。

［スレッドモード（UNICODE、UTF8用以外）の場合］

－ THREAD(MULTI)オプションを指定してください。

［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］

－ RCSオプション（UCS2-Unicode(UCS2)を使用する）を指定してください。

－ スレッドモードの場合、THREAD(MULTI)オプションを指定してください。

翻訳時の「翻訳オプション」については、NetCOBOLのマニュアルを参照してください。

登録集の指定

以下の環境変数を設定してください。

```
CORBA=C:¥Interstage¥ODWIN¥INCLUDE¥COBOL
```

生成ファイル

```
モジュール名-インタフェース名-オペレーション名.OBJ
スケルトン、CDRソースで定義された目的オブジェクト
```

Windows(R)アプリケーションの作成 Windows64

Windows(R)でCOBOLアプリケーションを作成する場合の注意点を以下に示します。

参照

- 詳細な翻訳・リンク方法については、NetCOBOLのマニュアルを参照してください。
- CORBAサービス（クライアント含む）が提供するライブラリについては、“B.1.2 ライブラリ”を参照してください。

ポイント

CORBAサービスのインストールフォルダを“C:¥Interstage¥ODWIN”として説明します。

クライアントアプリケーションの作成

クライアントアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。

翻訳・リンク手順(32bitアプリケーション)

1. IDLコンパイルを実行します。

```
IDLc -cobol IDLファイル名
```

2. 1.で生成されたスタブ、CDRソースをCOBOLソースファイルに登録し、主プログラムと合わせて翻訳します。翻訳時に「翻訳オプション」と環境変数CORBAに登録集のディレクトリを指定してください。

3. リンクするライブラリには、以下のファイルを指定します。
   [プロセスモード(UNICODE、UTF8用以外)の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODCOBCBL.LIB
   [スレッドモード(UNICODE、UTF8用以外)の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODCOBCBLMT.LIB
   [プロセスモード／スレッドモード(UNICODE、UTF8用)の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODCOBCBLUC.LIB

翻訳・リンク手順(64bitアプリケーション)

1. IDLコンパイルを実行します。

```
IDLc -cobol IDLファイル名
```

2. 1.で生成されたスタブ、CDRソースを、主プログラムと合わせて翻訳します。翻訳時に「翻訳オプション」と環境変数COB_CORBAに登録集のディレクトリを指定してください(XXX_C.cbl: 主プログラムファイル名)。

```
set COB_CORBA=C:¥Interstage¥ODWIN¥INCLUDE¥COBOL
COBOL.EXE  -NM "IDLファイル名_STUB.cbl"
COBOL.EXE  -NM "IDLファイル名_CDR.cbl"
COBOL.EXE  -M "XXX_C.cbl"
```

3. 翻訳したオブジェクトを結合し、クライアントアプリケーションを作成します。「リンクオプション」には、以下のファイルを指定します。
   [共通]
   F4AGCIMP.LIB
   libcmt.lib
   kernel32.lib
   [プロセスモード(UNICODE、UTF8用以外)の場合]
   C:¥Interstage¥ODWIN¥LIB¥x64¥ODCOBCBL.LIB
   [スレッドモード(UNICODE、UTF8用以外)の場合]
   C:¥Interstage¥ODWIN¥LIB¥x64¥ODCOBCBLMT.LIB
   [プロセスモード／スレッドモード(UNICODE、UTF8用)の場合]
   C:¥Interstage¥ODWIN¥LIB¥x64¥ODCOBCBLUC.LIB

翻訳対象ファイル

```
IDLファイル名_STUB.CBL
IDLファイル名_CDR.CBL
```

翻訳オプション

[共通]

－ NAMEオプションを指定してください。

[スレッドモード(UNICODE、UTF8用以外)の場合]

－ THREAD(MULTI)オプションを指定してください。

[プロセスモード／スレッドモード(UNICODE、UTF8用)の場合]

－ RCSオプション(UCS2-Unicode(UCS2)を使用する)を指定してください。

－ スレッドモードの場合、THREAD(MULTI)オプションを指定してください。

翻訳時の「翻訳オプション」については、NetCOBOLのマニュアルを参照してください。

登録集の指定

以下の環境変数を設定してください。

－ 64bitアプリケーションの場合

```
COB_CORBA=C:¥Interstage¥ODWIN¥INCLUDE¥COBOL
```

－ 32bitアプリケーションの場合

```
CORBA=C:¥Interstage¥ODWIN¥INCLUDE¥COBOL
```

生成ファイル

スタブ、CDRソースで定義された目的オブジェクト

サーバアプリケーションの翻訳・リンク

サーバアプリケーション作成時の翻訳・リンク手順と指定するファイル・オプションを以下に示します。なお、詳細な翻訳・リンク方法については、NetCOBOLのマニュアルを参照してください。

翻訳・リンク手順

1. IDLコンパイルを実行します。

```
IDLc -cobol  IDLファイル名
```

2. 1.で生成されたスケルトン、CDRソースを翻訳し、サーバアプリケーション(インタフェース実装部分)を翻訳し、オブジェクトファイル(*.obj)を作成します。翻訳時に「翻訳オプション」と、環境変数COB_CORBAに登録集のディレクトリを指定してください（XXX_SA.cbl: インタフェース実装プログラムファイル名）。

```
COBOL.EXE  -NM "IDLファイル名_インタフェース名_SKEL.CBL"
COBOL.EXE  -NM "IDLファイル名_SKEL.cbl"
COBOL.EXE  -NM "IDLファイル名_CDR.cbl"
COBOL.EXE  -NM "XXX_SA.cbl"
```

3. サーバアプリケーション（初期化処理部）を、同様の手順で翻訳します（XXX_S.cbl: 主プログラムファイル名）。詳細については、“Windows(R)アプリケーション作成時／運用時の注意事項 ”を参照してください。

```
COBOL.EXE  -M "XXX_S.cbl
```

4. 翻訳したオブジェクトを結合し、DLLファイルを作成します。DLLファイル名は、モジュール名-インタフェース名.DLLです。「リンクオプション」には、以下のファイルを指定します。
   ［共通］
   F4AGCIMP.LIB
   libcmt.lib
   kernel32.lib
   ［プロセスモード（UNICODE、UTF8用以外）の場合］
   C:¥Interstage¥ODWIN¥LIB¥x64¥ODCOBCBLSV.LIB
   ［スレッドモード（UNICODE、UTF8用以外）の場合］
   C:¥Interstage¥ODWIN¥LIB¥x64¥ODCOBCBLMTSV.LIB
   ［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］
   C:¥Interstage¥ODWIN¥LIB¥x64¥ODCOBCBLSVUC.LIB

5. サーバアプリケーションのメイン処理を主プログラムとして、実行プログラムを作成します。このとき、「リンクオプション」には以下のファイルを指定します。

［共通］
F4AGCIMP.LIB
libcmt.lib
kernel32.lib
［プロセスモード（UNICODE、UTF8用以外）の場合］
C:¥Interstage¥ODWIN¥LIB¥x64¥ODCOBCBLSV.LIB
［スレッドモード（UNICODE、UTF8用以外）の場合］
C:¥Interstage¥ODWIN¥LIB¥x64¥ODCOBCBLMTSV.LIB
［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］
C:¥Interstage¥ODWIN¥LIB¥x64¥ODCOBCBLSVUC.LIB

6. OD_impl_instコマンドでサーバアプリケーションを登録します。実行時に指定する定義ファイルは、サンプルプログラムの以下のファイルを参照してください。
C:¥Interstage¥ODWIN¥SRC¥SAMPLE¥COMPLEX¥SAMPLELIST.COBOL¥DATA¥*¥*.DEF
注）modeには、SYNC_ENDを設定してください。
DLLはDLL名だけを設定し、環境変数PATHにDLLのパスを追加してください。

翻訳対象ファイル

```
IDLファイル名_インタフェース名_SKEL.CBL
IDLファイル名_CDR.CBL
IDLファイル名_SKEL.CBL
```

注意

IDL定義により生成されない場合があります。

翻訳オプション

［共通］

- NAMEオプションを指定してください。
- DLOADオプションは指定しないでください。

［スレッドモード（UNICODE、UTF8用以外）の場合］

- THREAD(MULTI)オプションを指定してください。

［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］

- RCSオプション（UCS2-Unicode(UCS2)を使用する）を指定してください。
- スレッドモードの場合、THREAD(MULTI)オプションを指定してください。

翻訳時の「翻訳オプション」については、NetCOBOLのマニュアルを参照してください。

登録集の指定

以下の環境変数を設定してください。

```
COB_CORBA=C:¥Interstage¥ODWIN¥INCLUDE¥COBOL
```

生成ファイル

```
モジュール名-インタフェース名-オペレーション名.OBJ
スケルトン、CDRソースで定義された目的オブジェクト
```

Solarisアプリケーションの作成 Solaris32/64

SolarisでCOBOLアプリケーションを作成する場合の注意点を以下に示します。

注意

- 開発環境として“NetCOBOL”を使用してUNICODEのアプリケーションを開発する場合、ソースファイルおよび登録集は、UNICODE（UTF-8）で作成してください。
- 翻訳／リンク／実行を行う場合、コード系に“UNICODE”を設定してください。
- プロセスモード（UNICODE、UTF8用）の場合は、副プログラムの翻訳／リンク時に“-Tm”を指定してください。

参照

CORBAサービス（クライアント含む）が提供するライブラリについては、“B.1.2 ライブラリ”を参照してください。

ポイント

CORBAサービスのインストールディレクトリを“/opt/FSUNod”として説明します。

クライアントアプリケーションの作成

クライアントアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。

翻訳・リンク手順

1. IDLコンパイルを実行します。

```
IDLc -cobol IDLファイル名
```

2. 登録集を指定します。環境変数CORBAに、以下のディレクトリを指定してください。

```
CORBA=/opt/FSUNod/include/COBOL
```

3. 主プログラムを翻訳します。コンパイル方法を以下に示します（XXX_c.cbl: 主プログラムファイル名）。
   ［プロセスモード（UNICODE、UTF8用以外）の場合］
   cobol -M -c XXX_c.cbl
   ［スレッドモード（UNICODE、UTF8用以外）の場合］
   cobol -Tm -M -c XXX_c.cbl
   ［プロセスモード（UNICODE、UTF8用）の場合］
   cobol -M -c XXX_c.cbl
   ［スレッドモード（UNICODE、UTF8用）の場合］
   cobol -Tm -M -c XXX_c.cbl

4. 1.で生成されたスタブ、CDRソースを翻訳します。コンパイル方法を以下に示します（A_cdr.cbl: CDRソースファイル名、B_stub.cbl: スタブファイル名）。
   ［プロセスモード（UNICODE、UTF8用以外）の場合］
   cobol -G -o libA_cdr.so A_cdr.cbl
   cobol -G -o libB_stub.so -lA_cdr B_stub.cbl
   ［スレッドモード（UNICODE、UTF8用以外）の場合］
   cobol -G -Tm -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -o libB_stub.so -lA_cdr B_stub.cbl
   ［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］
   cobol -G -Tm -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -o libB_stub.so -lA_cdr B_stub.cbl

5. COBOLライブラリをリンクしてクライアントアプリケーションを作成します（CAP_c:作成するクライアントアプリケーション名）。
   ［プロセスモード（UNICODE、UTF8用以外）の場合］
   /opt/FSUNod/lib/libOMcbl.soをリンクします。
   cobol -L/opt/FSUNod/lib -lOMcbl -o CAP_c -lB_stub -lA_cdr XXX_c.o
   ［スレッドモード（UNICODE、UTF8用以外）の場合］
   /opt/FSUNod/lib/libOMcblMT.soをリンクします。
   cobol -L/opt/FSUNod/lib -lOMcblMT -Tm -o CAP_c -lB_stub -lA_cdr XXX_c.o
   ［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］
   /opt/FSUNod/lib/libOMcblUC.soをリンクします。
   cobol -L/opt/FSUNod/lib -lOMcblUC -Tm -o CAP_c -lB_stub -lA_cdr XXX_c.o

サーバアプリケーションの作成

サーバアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。

翻訳・リンク手順

1. IDLコンパイルを実行します。

```
IDLc -cobol IDLファイル名
```

2. 登録集を指定します。環境変数CORBAに、以下のディレクトリを指定してください。

```
CORBA=/opt/FSUNod/include/COBOL
```

3. 主プログラムを翻訳します。コンパイル方法を以下に示します（XXX_s.cbl: 主プログラムファイル名）。
   ［プロセスモード（UNICODE、UTF8用以外）の場合］
   cobol -M -c XXX_s.cbl
   ［スレッドモード（UNICODE、UTF8用以外）の場合］
   cobol -Tm -M -c XXX_s.cbl
   ［プロセスモード（UNICODE、UTF8用）の場合］
   cobol -M -c XXX_s.cbl
   ［スレッドモード（UNICODE、UTF8用）の場合］
   cobol -Tm -M -c XXX_s.cbl

4. サーバアプリケーション（初期化処理部）を作成します（SAP_s: サーバアプリケーション名）。
   ［プロセスモード（UNICODE、UTF8用以外）の場合］
   /opt/FSUNod/lib/libOMcbl.soをリンクします。
   cobol -lOMcbl -o SAP_s XXX_s.o
   ［スレッドモード（UNICODE、UTF8用以外）の場合］
   /opt/FSUNod/lib/libOMcblMT.soをリンクします。
   cobol -lOMcblMT -Tm -o SAP_s XXX_s.o
   ［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］
   /opt/FSUNod/lib/libOMcblUC.soをリンクします。
   cobol -lOMcblUC -Tm -o SAP_s XXX_s.o

5. 1.で生成されたスケルトン、CDRソースを翻訳します。コンパイル方法を以下に示します（A_cdr.cbl: CDRソースファイル名、B_skel.cbl: スケルトンファイル名）。
   なお、IDL定義に基本データ型以外が含まれる場合は、領域獲得用スケルトンファイルのコンパイルおよびリンクを行う必要があります。
   ［プロセスモード（UNICODE、UTF8用以外）の場合］
   cobol -G -o libA_cdr.so A_cdr.cbl
   cobol -G -o libB_skel.so B_skel.cbl
   ［スレッドモード（UNICODE、UTF8用以外）の場合］
   cobol -G -Tm -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -o libB_skel.so B_skel.cbl
   ［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］
   cobol -G -Tm -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -o libB_skel.so B_skel.cbl

6. COBOLライブラリをリンクしてサーバアプリケーション(インタフェース実装部分)を作成します(D_sa.cbl: COBOLファイル名、libD.so: ライブラリファイル名)。
   [プロセスモード(UNICODE、UTF8用以外)の場合]
   /opt/FSUNod/lib/libOMcbl.soをリンクします。
   cobol -G -lcobol -lOMcbl -o libD.so -lA_cdr -lB_skel D_sa.cbl
   [スレッドモード(UNICODE、UTF8用以外)の場合]
   /opt/FSUNod/lib/libOMcblMT.soをリンクします。
   cobol -G -Tm -lrcobol -lOMcblMT -o libD.so -lA_cdr -lB_skel D_sa.cbl
   [プロセスモード/スレッドモード(UNICODE、UTF8用)の場合]
   /opt/FSUNod/lib/libOMcblUC.soをリンクします。
   cobol -G -Tm -lrcobol -lOMcblUC -o libD.so -lA_cdr -lB_skel D_sa.cbl

Linuxアプリケーションの作成 Linux32

LinuxでCOBOLアプリケーションを作成する場合の注意点を以下に示します。

注意

- UNICODEのアプリケーションを開発する場合、ソースファイルおよび登録集は、UNICODE(UTF-8)で作成してください。
- 翻訳/リンク/実行を行う場合、コード系に“UNICODE”を設定してください。
- プロセスモード(UNICODE、UTF8用)の場合は、副プログラムの翻訳/リンク時に“-Tm”を指定してください。

参照

CORBAサービス(クライアント含む)が提供するライブラリについては、“B.1.2 ライブラリ”を参照してください。

ポイント

CORBAサービスのインストールディレクトリを“/opt/FJSVod”として説明します。

クライアントアプリケーションの作成

クライアントアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。

翻訳・リンク手順

1. IDLコンパイルを実行します。

```
IDLc -cobol IDLファイル名
```

2. 登録集を指定します。環境変数CORBAに、以下のディレクトリを指定してください。

```
CORBA=/opt/FJSVod/include/COBOL
```

3. 主プログラムを翻訳します。コンパイル方法を以下に示します(XXX_c.cbl: 主プログラムファイル名)。
   [プロセスモード(UNICODE、UTF8用以外)の場合]
   cobol -M -c XXX_c.cbl
   [スレッドモード(UNICODE、UTF8用以外)の場合]
   cobol -Tm -M -c XXX_c.cbl
   [プロセスモード(UNICODE、UTF8用)の場合]
   cobol -M -c XXX_c.cbl

［スレッドモード（UNICODE、UTF8用）の場合］
cobol -Tm -M -c XXX_c.cbl

4. 1で生成されたスタブ、CDRソースを翻訳します。コンパイル方法を以下に示します（A_cdr.cbl: CDRソースファイル名、B_stub.cbl: スタブファイル名）。
［プロセスモード（UNICODE、UTF8用以外）の場合］
cobol -G -o libA_cdr.so A_cdr.cbl
cobol -G -o libB_stub.so -L. -lA_cdr B_stub.cbl
［スレッドモード（UNICODE、UTF8用以外）の場合］
cobol -G -Tm -o libA_cdr.so A_cdr.cbl
cobol -G -Tm -o libB_stub.so -L. -lA_cdr B_stub.cbl
［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］
cobol -G -Tm -o libA_cdr.so A_cdr.cbl
cobol -G -Tm -o libB_stub.so -L. -lA_cdr B_stub.cbl

5. COBOLライブラリをリンクしてクライアントアプリケーションを作成します（CAP_c:作成するクライアントアプリケーション名）。
［プロセスモード（UNICODE、UTF8用以外）の場合］
/opt/FJSVod/lib/libOMcbl.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcbl -o CAP_c -L. -lB_stub -lA_cdr XXX_c.o
［スレッドモード（UNICODE、UTF8用以外）の場合］
/opt/FJSVod/lib/libOMcblMT.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcblMT -Tm -o CAP_c -L. -lB_stub -lA_cdr XXX_c.o
［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］
/opt/FJSVod/lib/libOMcblUC.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcblUC -Tm -o CAP_c -L. -lB_stub -lA_cdr XXX_c.o

サーバアプリケーションの作成

サーバアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。

翻訳・リンク手順

1. IDLコンパイルを実行します。

```
IDLc -cobol IDLファイル名
```

2. 登録集を指定します。環境変数CORBAに、以下のディレクトリを指定してください。

```
CORBA=/opt/FJSVod/include/COBOL
```

3. 主プログラムを翻訳します。コンパイル方法を以下に示します（XXX_s.cbl: 主プログラムファイル名）。
［プロセスモード（UNICODE、UTF8用以外）の場合］
cobol -M -c XXX_s.cbl
［スレッドモード（UNICODE、UTF8用以外）の場合］
cobol -Tm -M -c XXX_s.cbl
［プロセスモード（UNICODE、UTF8用）の場合］
cobol -M -c XXX_s.cbl
［スレッドモード（UNICODE、UTF8用）の場合］
cobol -Tm -M -c XXX_s.cbl

4. サーバアプリケーション（初期化処理部）を作成します（SAP_s: サーバアプリケーション名）。
［プロセスモード（UNICODE、UTF8用以外）の場合］
/opt/FJSVod/lib/libOMcbl.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcbl -o SAP_s XXX_s.o
［スレッドモード（UNICODE、UTF8用以外）の場合］
/opt/FJSVod/lib/libOMcblMT.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcblMT -Tm -o SAP_s XXX_s.o
［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］

/opt/FJSVod/lib/libOMcblUC.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcblUC -Tm -o SAP_s XXX_s.o

5. 1で生成されたスケルトン、CDRソースを翻訳します。コンパイル方法を以下に示します(A_cdr.cbl: CDRソースファイル名、B_skel.cbl: スケルトンファイル名)。
なお、IDL定義に基本データ型以外が含まれる場合は、領域獲得用スケルトンファイルのコンパイルおよびリンクも行う必要があります。
[プロセスモード(UNICODE、UTF8用以外)の場合]
cobol -G -o libA_cdr.so A_cdr.cbl
cobol -G -o libB_skel.so B_skel.cbl
[スレッドモード(UNICODE、UTF8用以外)の場合]
cobol -G -Tm -o libA_cdr.so A_cdr.cbl
cobol -G -Tm -o libB_skel.so B_skel.cbl
[プロセスモード/スレッドモード(UNICODE、UTF8用)の場合]
cobol -G -Tm -o libA_cdr.so A_cdr.cbl
cobol -G -Tm -o libB_skel.so B_skel.cbl

6. COBOLライブラリをリンクしてサーバアプリケーション(インタフェース実装部分)を作成します(D_sa.cbl: COBOLファイル名、libD.so: ライブラリファイル名)。
[プロセスモード(UNICODE、UTF8用以外)の場合]
/opt/FJSVod/lib/libOMcbl.soをリンクします。
cobol -G -L/opt/FJSVod/lib -lOMcbl -o libD.so -L. -lA_cdr -lB_skel D_sa.cbl
[スレッドモード(UNICODE、UTF8用以外)の場合]
/opt/FJSVod/lib/libOMcblMT.soをリンクします。
cobol -G -Tm -L/opt/FJSVod/lib -lOMcblMT -o libD.so -L. -lA_cdr -lB_skel D_sa.cbl
[プロセスモード/スレッドモード(UNICODE、UTF8用)の場合]
/opt/FJSVod/lib/libOMcblUC.soをリンクします。
cobol -G -Tm -L/opt/FJSVod/lib -lOMcblUC -o libD.so -L. -lA_cdr -lB_skel D_sa.cbl

Linuxアプリケーションの作成 Linux64

LinuxでCOBOLアプリケーションを作成する場合の注意点を以下に示します。

注意

- UNICODEのアプリケーションを開発する場合、ソースファイルおよび登録集は、UNICODE(UTF-8)で作成してください。
- 翻訳/リンク/実行を行う場合、コード系には“UNICODE”を設定してください。
- UNICODEでCOBOL翻訳を行う場合、RCSオプションで以下のエンディアンが指定可能です。
  - ビッグエンディアン(BE)
  - リトルエンディアン(LE)
- プロセスモード(UNICODE、UTF8用/ビッグエンディアンのUNICODE、UTF8用/リトルエンディアンのUNICODE、UTF8用)の場合は、副プログラムの翻訳/リンク時に“-Tm”を指定してください。

参照

CORBAサービス(クライアント含む)が提供するライブラリについては、“B.1.2 ライブラリ”を参照してください。

ポイント

CORBAサービスのインストールディレクトリを“/opt/FJSVod”として説明します。

クライアントアプリケーションの作成

クライアントアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。

翻訳・リンク手順

1. IDLコンパイルを実行します。

```
IDLc -cobol IDLファイル名
```

2. 登録集を指定します。環境変数CORBAに、以下のディレクトリを指定してください。

```
CORBA=/opt/FJSVod/include/COBOL
```

3. 主プログラムを翻訳します。コンパイル方法を以下に示します（XXX_c.cbl: 主プログラムファイル名）。
   ［プロセスモード（UNICODE、UTF8用以外）の場合］
   cobol -M -c XXX_c.cbl
   ［スレッドモード（UNICODE、UTF8用以外）の場合］
   cobol -Tm -M -c XXX_c.cbl
   ［プロセスモード（UNICODE、UTF8用）の場合］
   cobol -M -c XXX_c.cbl
   ［スレッドモード（UNICODE、UTF8用）の場合］
   cobol -Tm -M -c XXX_c.cbl
   ［プロセスモード（ビッグエンディアンのUNICODE、UTF8用）の場合］
   cobol -WC,"RCS(UCS2,BE)" -M -c XXX_c.cbl
   ［スレッドモード（ビッグエンディアンのUNICODE、UTF8用）の場合］
   cobol -Tm -WC,"RCS(UCS2,BE)" -M -c XXX_c.cbl
   ［プロセスモード（リトルエンディアンのUNICODE、UTF8用）の場合］
   cobol -WC,"RCS(UCS2,LE)" -M -c XXX_c.cbl
   ［スレッドモード（リトルエンディアンのUNICODE、UTF8用）の場合］
   cobol -Tm -WC,"RCS(UCS2,LE)" -M -c XXX_c.cbl

4. 1で生成されたスタブ、CDRソースを翻訳します。コンパイル方法を以下に示します（A_cdr.cbl: CDRソースファイル名、B_stub.cbl: スタブファイル名）。
   ［プロセスモード（UNICODE、UTF8用以外）の場合］
   cobol -G -o libA_cdr.so A_cdr.cbl
   cobol -G -o libB_stub.so -L. -lA_cdr B_stub.cbl
   ［スレッドモード（UNICODE、UTF8用以外）の場合］
   cobol -G -Tm -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -o libB_stub.so -L. -lA_cdr B_stub.cbl
   ［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］
   cobol -G -Tm -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -o libB_stub.so -L. -lA_cdr B_stub.cbl
   ［プロセスモード／スレッドモード（ビッグエンディアンのUNICODE、UTF8用）の場合］
   cobol -G -Tm -WC,"RCS(UCS2,BE)" -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -WC,"RCS(UCS2,BE)" -o libB_stub.so -L. -lA_cdr B_stub.cbl
   ［プロセスモード／スレッドモード（リトルエンディアンのUNICODE、UTF8用）の場合］
   cobol -G -Tm -WC,"RCS(UCS2,LE)" -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -WC,"RCS(UCS2,LE)" -o libB_stub.so -L. -lA_cdr B_stub.cbl

5. COBOLライブラリをリンクしてクライアントアプリケーションを作成します（CAP_c:作成するクライアントアプリケーション名）。
   ［プロセスモード（UNICODE、UTF8用以外）の場合］
   /opt/FJSVod/lib/libOMcbl.soをリンクします。
   cobol -L/opt/FJSVod/lib -lOMcbl -o CAP_c -L. -lB_stub -lA_cdr XXX_c.o
   ［スレッドモード（UNICODE、UTF8用以外）の場合］
   /opt/FJSVod/lib/libOMcblMT.soをリンクします。
   cobol -L/opt/FJSVod/lib -lOMcblMT -Tm -o CAP_c -L. -lB_stub -lA_cdr XXX_c.o
   ［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］

/opt/FJSVod/lib/libOMcblUC.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcblUC -Tm -o CAP_c -L. -lB_stub -lA_cdr XXX_c.o
［プロセスモード／スレッドモード（ビッグエンディアンのUNICODE、UTF8用）の場合］
/opt/FJSVod/lib/libOMcblUCBE.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcblUCBE -Tm -o CAP_c -L. -lB_stub -lA_cdr XXX_c.o
［プロセスモード／スレッドモード（リトルエンディアンのUNICODE、UTF8用）の場合］
/opt/FJSVod/lib/libOMcblUCLE.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcblUCLE -Tm -o CAP_c -L. -lB_stub -lA_cdr XXX_c.o

サーバアプリケーションの作成

サーバアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。

翻訳・リンク手順

1. IDLコンパイルを実行します。

```
IDLc -cobol IDLファイル名
```

2. 登録集を指定します。環境変数CORBAに、以下のディレクトリを指定してください。

```
CORBA=/opt/FJSVod/include/COBOL
```

3. 主プログラムを翻訳します。コンパイル方法を以下に示します（XXX_s.cbl: 主プログラムファイル名）。
［プロセスモード（UNICODE、UTF8用以外）の場合］
cobol -M -c XXX_s.cbl
［スレッドモード（UNICODE、UTF8用以外）の場合］
cobol -Tm -M -c XXX_s.cbl
［プロセスモード（UNICODE、UTF8用）の場合］
cobol -M -c XXX_s.cbl
［スレッドモード（UNICODE、UTF8用）の場合］
cobol -Tm -M -c XXX_s.cbl
［プロセスモード（ビッグエンディアンのUNICODE、UTF8用）の場合］
cobol -WC,"RCS(UCS2,BE)" -M -c XXX_s.cbl
［スレッドモード（ビッグエンディアンのUNICODE、UTF8用）の場合］
cobol -Tm -WC,"RCS(UCS2,BE)" -M -c XXX_s.cbl
［プロセスモード（リトルエンディアンのUNICODE、UTF8用）の場合］
cobol -WC,"RCS(UCS2,LE)" -M -c XXX_s.cbl
［スレッドモード（リトルエンディアンのUNICODE、UTF8用）の場合］
cobol -Tm -WC,"RCS(UCS2,LE)" -M -c XXX_s.cbl

4. サーバアプリケーション（初期化処理部）を作成します（SAP_s: サーバアプリケーション名）。
［プロセスモード（UNICODE、UTF8用以外）の場合］
/opt/FJSVod/lib/libOMcbl.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcbl -o SAP_s XXX_s.o
［スレッドモード（UNICODE、UTF8用以外）の場合］
/opt/FJSVod/lib/libOMcblMT.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcblMT -Tm -o SAP_s XXX_s.o
［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］
/opt/FJSVod/lib/libOMcblUC.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcblUC -Tm -o SAP_s XXX_s.o
［プロセスモード／スレッドモード（ビッグエンディアンのUNICODE、UTF8用）の場合］
/opt/FJSVod/lib/libOMcblUCBE.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcblUCBE -Tm -o SAP_s XXX_s.o
［プロセスモード／スレッドモード（リトルエンディアンのUNICODE、UTF8用）の場合］
/opt/FJSVod/lib/libOMcblUCLE.soをリンクします。
cobol -L/opt/FJSVod/lib -lOMcblUCLE -Tm -o SAP_s XXX_s.o

5. 1で生成されたスケルトン、CDRソースを翻訳します。コンパイル方法を以下に示します（A_cdr.cbl: CDRソースファイル名、B_skel.cbl: スケルトンファイル名）。
   なお、IDL定義に基本データ型以外が含まれる場合は、領域獲得用スケルトンファイルのコンパイルおよびリンクも行う必要があります。
   ［プロセスモード（UNICODE、UTF8用以外）の場合］
   cobol -G -o libA_cdr.so A_cdr.cbl
   cobol -G -o libB_skel.so B_skel.cbl
   ［スレッドモード（UNICODE、UTF8用以外）の場合］
   cobol -G -Tm -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -o libB_skel.so B_skel.cbl
   ［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］
   cobol -G -Tm -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -o libB_skel.so B_skel.cbl
   ［プロセスモード／スレッドモード（ビッグエンディアンのUNICODE、UTF8用）の場合］
   cobol -G -Tm -WC,"RCS(UCS2,BE)" -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -WC,"RCS(UCS2,BE)" -o libB_skel.so B_skel.cbl
   ［プロセスモード／スレッドモード（リトルエンディアンのUNICODE、UTF8用）の場合］
   cobol -G -Tm -WC,"RCS(UCS2,LE)" -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -WC,"RCS(UCS2,LE)" -o libB_skel.so B_skel.cbl

6. COBOLライブラリをリンクしてサーバアプリケーション(インタフェース実装部分)を作成します（D_sa.cbl: COBOLファイル名、libD.so: ライブラリファイル名）。
   ［プロセスモード（UNICODE、UTF8用以外）の場合］
   /opt/FJSVod/lib/libOMcbl.soをリンクします。
   cobol -G -L/opt/FJSVod/lib -lOMcbl -o libD.so -L. -lA_cdr -lB_skel D_sa.cbl
   ［スレッドモード（UNICODE、UTF8用以外）の場合］
   /opt/FJSVod/lib/libOMcblMT.soをリンクします。
   cobol -G -Tm -L/opt/FJSVod/lib -lOMcblMT -o libD.so -L. -lA_cdr -lB_skel D_sa.cbl
   ［プロセスモード／スレッドモード（UNICODE、UTF8用）の場合］
   /opt/FJSVod/lib/libOMcblUC.soをリンクします。
   cobol -G -Tm -L/opt/FJSVod/lib -lOMcblUC -o libD.so -L. -lA_cdr -lB_skel D_sa.cbl
   ［プロセスモード／スレッドモード（ビッグエンディアンのUNICODE、UTF8用）の場合］
   /opt/FJSVod/lib/libOMcblUCBE.soをリンクします。
   cobol -G -Tm -WC,"RCS(UCS2,BE)" -L/opt/FJSVod/lib -lOMcblUCBE -o libD.so -L. -lA_cdr -lB_skel D_sa.cbl
   ［プロセスモード／スレッドモード（リトルエンディアンのUNICODE、UTF8用）の場合］
   /opt/FJSVod/lib/libOMcblUCLE.soをリンクします。
   cobol -G -Tm -WC,"RCS(UCS2,LE)" -L/opt/FJSVod/lib -lOMcblUCLE -o libD.so -L. -lA_cdr -lB_skel D_sa.cbl

対応可能な予約語の一覧

以下に対応可能な予約語を示します。

"ACCEPT"、 "ACCESS"、 "ACTUAL"、 "ADD"、 "ADDRESS"、 "ADVANCING"、 "AFTER"、 "ALL"、 "ALPHABET"、 "ALPHABETIC"、 "ALPHANUMERIC"、 "ALSO"、 "ALTER"、 "ALTERNATE"、 "AND"、 "ANY"、 "APPLY"、 "ARE"、 "AREA"、"AREAS"、"ARITHMETIC"、"AS"、"ASCENDING"、"ASSIGN"、"AT"、"AUTHOR"、"AUTO"、"AUTOMATIC"、 "BASED"、 "BEFORE"、 "BEGINNING"、 "BELL"、 "BINARY"、 "BIT"、 "BITS"、 "BLANK"、 "BLINK"、 "BLOCK"、 "BOTTOM"、 "BY"、 "CALL"、 "CANCEL"、 "CBL"、 "CD"、 "CF"、 "CH"、 "CHANGED"、 "CHARACTER"、 "CHARACTERS"、"CLASS"、"CLOSE"、"COBOL"、"CODE"、"COLLATING"、"COLUMN"、"COMMA"、"COMMAND"、 "COMMIT"、 "COMMON"、 "COMMUNICATION"、 "COMP"、 "COMPLEX"、 "COMPUTATIONAL"、 "COMPUTE"、 "CONFIGURATION"、 "CONNECT"、 "CONSTANT"、 "CONTAINED"、 "CONTAINS"、 "CONTENT"、 "CONTINUE"、 "CONTROL"、"CONTROLS"、"CONVERTING"、"COPY"、"CORR"、"CORRESPONDING"、"COUNT"、"CRP"、"CRT"、 "CURRENCY"、"CURRENT"、"CURSOR"、"DATA"、"DATE"、"DAY"、"DB"、"DE"、"DEBUGGING"、"DEFAULT"、 "DELETE"、"DELIMITED"、"DELIMITER"、"DEPENDING"、"DESCENDING"、"DESTINATION"、"DETAIL"、"DEVICE"、 "DIRECT"、 "DISABLE"、 "DISCONNECT"、 "DISJOINING"、 "DISPLAY"、 "DIVIDE"、 "DIVISION"、 "DOWN"、 "DUPLICATE"、"DUPLICATES"、"DYNAMIC"、"EGCS"、"EGI"、"EJECT"、"ELSE"、"EMI"、"EMPTY"、"ENABLE"、 "END"、 "ENDCOBOL"、 "ENDING"、 "ENTER"、 "ENTRY"、 "ENVIRONMENT"、 "EOL"、 "EOP"、 "EOS"、 "EQUAL"、 "EQUALS"、 "ERASE"、 "ERROR"、 "ESI"、 "EVALUATE"、 "EVERY"、 "EXACT"、 "EXAMINE"、 "EXCEEDS"、

"EXCEPTION"、"EXCLUSIVE"、"EXEC"、"EXIT"、"EXOR"、"EXTEND"、"EXTERNAL"、"FD"、"FETCH"、"FILE"、"FILES"、"FILLER"、"FINAL"、"FIND"、"FINISH"、"FIRST"、"FLADD"、"FOOTING"、"FOR"、"FORM"、"FORMAT"、"FORMATTED"、"FREE"、"FROM"、"FULL"、"FUNCTION"、"GENERATE"、"GET"、"GIVING"、"GLOBAL"、"GO"、"GOBACK"、"GREATER"、"GRID"、"GROUP"、"HEADING"、"HIGHLIGHT"、"ID"、"IDENTIFICATION"、"IF"、"IN"、"INCLUDE"、"INDEX"、"INDEXED"、"INDICATE"、"INITIAL"、"INITIALIZE"、"INITIATE"、"INPUT"、"INSPECT"、"INSTALLATION"、"INTO"、"INVALID"、"IS"、"JAPANESE"、"JOB"、"JOINING"、"JUST"、"JUSTIFIED"、"KANJI"、"KEEP"、"KEY"、"LABEL"、"LAST"、"LD"、"LEADING"、"LEFT"、"LEFTLINE"、"LENGTH"、"LESS"、"LIMIT"、"LIMITED"、"LIMITS"、"LINAGE"、"LINE"、"LINES"、"LINKAGE"、"LOCALLY"、"LOCK"、"LOWLIGHT"、"MANUAL"、"MEMBER"、"MEMORY"、"MERGE"、"MESSAGE"、"MODE"、"MODIFY"、"MODULES"、"MOVE"、"MULTICON"、"MULTIPLE"、"MULTIPLY"、"NAMED"、"NATIONAL"、"NATIVE"、"NEGATIVE"、"NEXT"、"NO"、"NOMINAL"、"NONE"、"NOT"、"NOTE"、"NULL"、"NULLS"、"NUMBER"、"NUMERIC"、"OCCURS"、"OF"、"OFF"、"OMITTED"、"ON"、"ONLY"、"OPEN"、"OPTIONAL"、"OR"、"ORDER"、"ORGANIZATION"、"OTHER"、"OTHERWISE"、"OUTPUT"、"OVERFLOW"、"OVERLINE"、"OWNER"、"PADDING"、"PAGE"、"PASSWORD"、"PERFORM"、"PF"、"PH"、"PIC"、"PICTURE"、"PLUS"、"POINTER"、"POSITION"、"POSITIONING"、"POSITIVE"、"PREFIX"、"PRESENT"、"PREVIOUS"、"PRINTING"、"PRIOR"、"PROCEDURE"、"PROCEDURES"、"PROCEED"、"PROCESSING"、"PROGRAM"、"PROMPT"、"PROTECTED"、"PURGE"、"QUEUE"、"QUOTE"、"QUOTES"、"RANDOM"、"RANGE"、"RD"、"READ"、"READY"、"REALM"、"RECEIVE"、"RECONNECT"、"RECORD"、"RECORDING"、"RECORDS"、"REDEFINES"、"REEL"、"REFERENCE"、"REFERENCES"、"RELATION"、"RELATIVE"、"RELEASE"、"RELOAD"、"REMAINDER"、"REMARKS"、"REMOVAL"、"RENAMES"、"REPEATED"、"REPLACE"、"REPLACING"、"REPORT"、"REPORTING"、"REPORTS"、"RERUN"、"REQUIRED"、"RESERVE"、"RESET"、"RETAINING"、"RETRIEVAL"、"RETURN"、"REVERSED"、"REWIND"、"REWRITE"、"RF"、"RH"、"RIGHT"、"ROLLBACK"、"ROUNDED"、"RUN"、"SA"、"SAME"、"SCREEN"、"SD"、"SEARCH"、"SECTION"、"SECURITY"、"SECURE"、"SEEK"、"SEGMENT"、"SELECT"、"SELECTED"、"SELECTIVE"、"SEND"、"SENTENCE"、"SEPARATE"、"SEQUENCE"、"SEQUENTIAL"、"SERVICE"、"SESSION"、"SET"、"SHARED"、"SIGN"、"SIMPLE"、"SINGLE"、"SIZE"、"SKIP1"、"SKIP2"、"SKIP3"、"SORT"、"SOURCE"、"SPACE"、"SPACES"、"STANDERD"、"START"、"STATION"、"STATIONS"、"STATUS"、"STOP"、"STORE"、"STRING"、"SUBRANGE"、"SUBTRACT"、"SUCCESSIVE"、"SUFFIX"、"SUM"、"SUPPRESS"、"SYMBOLIC"、"SYNC"、"SYNCHRONIZED"、"TABLE"、"TALLY"、"TALLYING"、"TAPE"、"TENANT"、"TERMINAL"、"TERMINATE"、"TEST"、"TEXT"、"THAN"、"THEN"、"THROUGH"、"THRU"、"TIME"、"TIMES"、"TITLE"、"TO"、"TOP"、"TRACE"、"TRACK"、"TRACKS"、"TRAILING"、"TRANSACTION"、"TYPE"、"UNDERLINE"、"UNEQUAL"、"UNIT"、"UNLOCK"、"UNSTRING"、"UNTIL"、"UP"、"UPDATE"、"UPON"、"USAGE"、"USE"、"USING"、"VALID"、"VALIDATE"、"VALUE"、"VALUES"、"VARYING"、"WAIT"、"WHEN"、"WITH"、"WITHIN"、"WORDS"、"WRITE"、"ZERO"、"ZEROES"、"ZEROS"

6.11 COBOL用登録集について

CORBAサービスが提供する関数、データ宣言などに関する登録集原文について、以下に示します。

格納パス(インストールパスはデフォルト)

Windows32/64

```
C:¥Interstage¥ODWIN¥INCLUDE¥COBOL
```

Solaris32/64

```
/opt/FSUNod/include/COBOL
```

Linux32/64

```
/opt/FJSVod/include/COBOL
```

6.11.1 使用例

(1)環境部の構成節での宣言

“COPY SYMBOL-CONST IN CORBA.”宣言をSPECIAL-NAMES内でSYMBOLIC CONSTANT句により行います。

```
000013  ENVIRONMENT DIVISION.
000014  CONFIGURATION SECTION.
000015  SPECIAL-NAMES.
000016      SYMBOLIC CONSTANT
000017      COPY SYMBOL-CONST IN CORBA.
000018      .
```

(2)継承されて使用できる関数の宣言

インタフェースリポジトリで継承されて使用できる関数宣言は登録集のREPLACEソースから必要な関数を抽出して使用する前に宣言を行います。

```
000020  REPLACE
000030  == "CORBA-CONTAINED--GET-DEF-KIND" == BY
000040  == "CORBA-IROBJECT--GET-DEF-KIND" ==
000050     .
000060  IDENTIFICATION DIVISION.
```

(3)データ部の作業場所節での宣言(その1)

“COPY CONST IN CORBA.”宣言をWORKING-STORAGE SECTIONの真下に宣言を行います(レベル番号は設定しません)。

```
000022  WORKING-STORAGE SECTION.
000023  COPY CONST IN CORBA.
```

(4)データ部の作業場所節での宣言(その2)

各種型宣言は、COPY文により領域を宣言します。

記述例1

```
000290  01 COPY ULONG IN CORBA REPLACING
                 1.       2.
000300  CORBA-UNSIGNED-LONG BY STRING-LENGTH.
              3.                4.
```

1. 登録集のソース名を記述します。
2. 環境変数名を記述します(CORBAを使用)。
3. 登録集のソース内のデータ名を記述します。
4. ユーザが定義する変数名を記述します。

記述例2

```
000310 01 COPY ENVIRONMENT    IN CORBA         REPLACING
000320                 CORBA-ENVIRONMENT       BY ENV.
000330 01 COPY ORB            IN CORBA         REPLACING
000340                 CORBA-ORB               BY ORB.
000350 01 COPY BOA            IN CORBA         REPLACING
000360                 CORBA-BOA               BY BOA.
000370 01 COPY OBJECT         IN CORBA         REPLACING
```

6.11.2 登録集原文

(1)データ型宣言

ANY.cbl	CORBA-ANY型構造体
BOOLEAN.cbl	CORBA-BOOLEAN型
CHAR.cbl	CORBA-CHAR型
DOUBLE.cbl	CORBA-DOUBLE型
ENUM.cbl	CORBA-ENUM型
FIXED-D-S.cbl	CORBA-FIXED-D-S型
FLOAT.cbl	CORBA-FLOAT型
LONG.cbl	CORBA-LONG型
OBJECT.cbl	CORBA-OBJECT型
OCTET.cbl	CORBA-OCTET型
SHORT.cbl	CORBA-SHORT型
TYPECODE.cbl	CORBA-TYPECODE型
ULONG.cbl	CORBA-UNSIGNED-LONG型
ULLONG.cbl	CORBA-UNSIGNED-LONG-LONG型
USHORT.cbl	CORBA-UNSIGNED-SHORT型
LLONG.cbl	CORBA-LONG-LONG型
WCHAR.cbl	CORBA-WCHAR型

(2)定数型宣言

COMPLETION-STATUS.cbl	envのメンバ通信の状態
CONST.cbl	固定文字列情報
FJ-IMPLEMENTATIONDEF.cbl	インプリメンテーションオブジェクトリファレンス
FLAGS.cbl	flag
IMPL-BOA.cbl	インプリメントリポジトリID
IMPL-CONTEXT.cbl	インプリメントリポジトリID
IMPL-IDLTYPE.cbl	インプリメントリポジトリID
IMPL-IMPLEMENTATIONDEF.cbl	インプリメントリポジトリID
IMPL-INTERFACEDEF.cbl	インプリメントリポジトリID
IMPL-NVLIST.cbl	インプリメントリポジトリID
IMPL-OBJECT.cbl	インプリメントリポジトリID

IMPL-OPERATIONDEF.cbl	インプリメントリポジトリID
IMPL-ORB.cbl	インプリメントリポジトリID
IMPL-PRINCIPAL.cbl	インプリメントリポジトリID
IMPL-REQUEST.cbl	インプリメントリポジトリID
IMPL-SERVERREQUEST.cbl	インプリメントリポジトリID
IMPL-TYPECODE.cbl	インプリメントリポジトリID
ORBSTATUS.cbl	ORBの情報種別
STATUS.cbl	動的復帰時の確認用フラグ
SYMBOL-CONST.cbl	定数情報

(3)エラー例外用宣言

BAD-CONTEXT.cbl	システム例外定義
BAD-INV-ORDER.cbl	システム例外定義
BAD-OPERATION.cbl	システム例外定義
BAD-PARAM.cbl	システム例外定義
BAD-QOS.cbl	システム例外定義
BAD-TYPECODE.cbl	システム例外定義
CODESET-INCOMPATIBLE.cbl	システム例外定義
COMM-FAILURE.cbl	システム例外
DATA-CONVERSION.cbl	例外情報
ENVIRONMENT.cbl	例外情報構造体
EX-ORB-INVALIDNAME.cbl	例外文字列
EX-STEXCEP-BAD-INV-ORDER.cbl	例外文字列
EX-STEXCEP-BAD-OPERATION.cbl	例外文字列
EX-STEXCEP-BAD-PARAM.cbl	例外文字列
EX-STEXCEP-BAD-QOS.cbl	例外文字列
EX-STEXCEP-BAD-TYPECODE.cbl	例外文字列
EX-STEXCEP-CODESET-INCOM.cbl	例外文字列
EX-STEXCEP-COMM-FAILURE.cbl	例外文字列
EX-STEXCEP-CONTEXT.cbl	例外文字列
EX-STEXCEP-DATA-CONVERSI.cbl	例外文字列
EX-STEXCEP-FREE-MEM.cbl	例外文字列

EX-STEXCEP-IMP-LIMIT.cbl	例外文字列
EX-STEXCEP-INITIALIZE.cbl	例外文字列
EX-STEXCEP-INTERNAL.cbl	例外文字列
EX-STEXCEP-INTF-REPOS.cbl	例外文字列
EX-STEXCEP-INV-FLAG.cbl	例外文字列
EX-STEXCEP-INV-IDENT.cbl	例外文字列
EX-STEXCEP-INV-OBJREF.cbl	例外文字列
EX-STEXCEP-INV-POLICY.cbl	例外文字列
EX-STEXCEP-MARSHAL.cbl	例外文字列
EX-STEXCEP-NO-IMPLEMENT.cbl	例外文字列
EX-STEXCEP-NO-MEMORY.cbl	例外文字列
EX-STEXCEP-NO-PERMISSION.cbl	例外文字列
EX-STEXCEP-NO-RESOURCES.cbl	例外文字列
EX-STEXCEP-NO-RESPONSE.cbl	例外文字列
EX-STEXCEP-OBJ-ADAPTER.cbl	例外文字列
EX-STEXCEP-PERSIST-STORE.cbl	例外文字列
EX-STEXCEP-REBIND.cbl	例外文字列
EX-STEXCEP-TIMEOUT.cbl	例外文字列
EX-STEXCEP-TRANSACTION-M.cbl	例外文字列
EX-STEXCEP-TRANSACTION-U.cbl	例外文字列
EX-STEXCEP-TRANSIENT.cbl	例外文字列
EX-STEXCEP-UNKNOWN.cbl	例外文字列
EX-TYPECODE-BADKIND.cbl	例外文字列
EX-TYPECODE-BOUNDS.cbl	例外文字列
EXCDESCRIPTIONSEQ.cbl	ExceptionDef構造体のsequence
EXCEPTION-TYPE.cbl	ExceptionDef種別
EXCEPTIONDEF.cbl	ExceptionDefオブジェクトリファレンス
EXCEPTIONDEFSEQ.cbl	ExceptionDefオブジェクトリファレンスのsequence
EXCEPTIONDESCRIPTION.cbl	ExceptionDef構造体
FREE-MEM.cbl	例外
IMP-LIMIT.cbl	例外
INITIALIZE.cbl	例外

INTERNAL.cbl	例外
INTF-REPOS.cbl	例外
INV-FLAGS.cbl	例外
INV-IDENT.cbl	例外
INV-OBJREF.cbl	例外
INV-POLICY.cbl	例外
NO-IMPLEMENT.cbl	例外
NO-MEMORY.cbl	例外
NO-PERMISSION.cbl	例外
NO-RESOURCES.cbl	例外
NO-RESPONSE.cbl	例外
NOT-EXIST.cbl	例外
OBJ-ADAPTER.cbl	例外
PARSIST-STORE.cbl	例外
REBIND.cbl	例外
TIMEOUT.cbl	例外
TRANSACTION-MODE.cbl	例外
TRANSACTION-UNAVAILABLE.cbl	例外
TRANSIENT.cbl	例外
UNKNOWN.cbl	例外

(4)ネーミングサービス用宣言

COSNAMING-BINDING-LIST.cbl	ネーミングコンテキストのバインディングリスト構造体
COSNAMING-BINDING.cbl	ネーミングコンテキストのバインディング
COSNAMING-BINDINGITERATOR.cbl	ネーミングコンテキストのバインディングアイテレータ
COSNAMING-BINDINGTYPE.cbl	ネーミングコンテキストのバインディングタイプ
COSNAMING-ISTRING.cbl	ネーミングコンテキストのストリング型
COSNAMING-NAME.cbl	ネーミングコンテキスト名
COSNAMING-NAMECOMPONENT.cbl	ネームコンポーネント
COSNAMING-NAMINGCONTEXT.cbl	ネーミングコンテキストオブジェクトリファレンス
COSNAMING-NAMINGCONTEXT-CANNOT.cbl	ネーミングコンテキストのCannotProceed例外
COSNAMING-NAMINGCONTEXTEXT.cbl	拡張ネーミングコンテキストオブジェクトリファレンス

COSNAMING-NAMINGCONTEXTEXT-ADD.cbl	拡張ネーミングコンテキストのアドレス型
COSNAMING-NAMINGCONTEXTEXT-STR.cbl	拡張ネーミングコンテキストの文字列型ネーミングコンテキスト名
COSNAMING-NAMINGCONTEXTEXT-URL.cbl	拡張ネーミングコンテキストのURLアドレス型
COSNAMING-NAMINGCONTEXT-NOTOO1.cbl	ネーミングコンテキストのNotFound例外
COSNAMING-NAMINGCONTEXT-NOTFOU.cbl	ネーミングコンテキストのNotFound例外の理由

(5)ロードバランスオプション用宣言 Windows32 Solaris32 Linux32

CONST-LBO.cbl	ロードバランスオプションのオブジェクトID
ISOD-LBG.cbl	ロードバランスオブジェクトグループのオブジェクトリファレンス
ISOD-LBG-LOADBALANCETYPE.cbl	ロードバランスタイプ
ISOD-LBG-OBJECTLIST.cbl	オブジェクトリファレンスリストの構造体
ISOD-LBO.cbl	ロードバランスオブジェクトのオブジェクトリファレンス
ISOD-LBO-BINDINGLBG.cbl	ロードバランスオブジェクトグループのバインディング
ISOD-LBO-DOWNSERVERLIST.cbl	ダウンサーバリストの構造体
ISOD-LBO-DUPLICATETYPE.cbl	ロードバランスオプションのAlreadyExist例外の詳細
ISOD-LBO-LBGLIST.cbl	ロードバランスオブジェクトグループのバインディングリスト構造体
ISOD-LBO-LOADBALANCETYPE.cbl	ロードバランスタイプ
ISOD-LBO-NOTFOUNDREASON.cbl	ロードバランスオプションNotFound例外

(6)インタフェースリポジトリ用宣言

INTERFACEDEF-FULLINTERFA.cbl	インタフェース情報クラス
INTERFACEDEF.cbl	InterfaceDefオブジェクトリファレンス
INTERFACEDEFSEQ.cbl	InterfaceDefオブジェクトリファレンスのsequence
INTERFACEDESCRIPTION.cbl	InterfaceDef情報構造体
INTERFACEREP.cbl	InterfaceDefオブジェクトリファレンス
INTF-BOA.cbl	インタフェースリポジトリID
INTF-CONTEXT.cbl	インタフェースリポジトリID
INTF-IDLTYPE.cbl	インタフェースリポジトリID
INTF-IMPLEMENTATIONDEF.cbl	インタフェースリポジトリID
INTF-INTERFACEDEF.cbl	インタフェースリポジトリID
INTF-NVLIST.cbl	インタフェースリポジトリID
INTF-OBJECT.cbl	インタフェースリポジトリID

INTF-OPERATIONDEF.cbl	インタフェースリポジトリID
INTF-ORB.cbl	インタフェースリポジトリID
INTF-PRINCIPAL.cbl	インタフェースリポジトリID
INTF-REQUEST.cbl	インタフェースリポジトリID
INTF-SERVERREQUEST.cbl	インタフェースリポジトリID
INTF-TYPECODE.cbl	インタフェースリポジトリID
IROBJECT.cbl	IRオブジェクトリファレンス

(7)TypeCode関連

TC-ALIAS.cbl	オブジェクト標準のタイプコード
TC-ANY.cbl	オブジェクト標準のタイプコード
TC-ARRAY.cbl	オブジェクト標準のタイプコード
TC-BOA.cbl	オブジェクト標準のタイプコード
TC-BOOLEAN.cbl	オブジェクト標準のタイプコード
TC-CHAR-SEQ.cbl	オブジェクト標準のタイプコード
TC-CHAR.cbl	オブジェクト標準のタイプコード
TC-COMPLETION-STATUS.cbl	オブジェクト標準のタイプコード
TC-CONTEXT.cbl	オブジェクト標準のタイプコード
TC-DOUBLE.cbl	オブジェクト標準のタイプコード
TC-ENUM.cbl	オブジェクト標準のタイプコード
TC-ENUMMEMBERSEQ.cbl	オブジェクト標準のタイプコード
TC-EXCEPT.cbl	オブジェクト標準のタイプコード
TC-EXCEPTION-TYPE.cbl	オブジェクト標準のタイプコード
TC-FLAGS.cbl	オブジェクト標準のタイプコード
TC-FLOAT.cbl	オブジェクト標準のタイプコード
TC-IDENTIFIER.cbl	オブジェクト標準のタイプコード
TC-IDLTYPE.cbl	オブジェクト標準のタイプコード
TC-IMPLEMENTATIONDEF.cbl	オブジェクト標準のタイプコード
TC-INTERFACEDEF.cbl	オブジェクト標準のタイプコード
TC-LONG.cbl	オブジェクト標準のタイプコード
TC-NAMEDVALUE.cbl	オブジェクト標準のタイプコード
TC-NULL.cbl	オブジェクト標準のタイプコード

TC-NVLIST.cbl	オブジェクト標準のタイプコード
TC-OBJECT.cbl	オブジェクト標準のタイプコード
TC-OCTET-SEQ.cbl	オブジェクト標準のタイプコード
TC-OCTET.cbl	オブジェクト標準のタイプコード
TC-OPERATIONDEF.cbl	オブジェクト標準のタイプコード
TC-ORB-INVALIDNAME.cbl	オブジェクト標準のタイプコード
TC-ORB-OBJECTID.cbl	オブジェクト標準のタイプコード
TC-ORB-OBJECTIDLIST.cbl	オブジェクト標準のタイプコード
TC-ORB.cbl	オブジェクト標準のタイプコード
TC-ORBSTATUS.cbl	オブジェクト標準のタイプコード
TC-PRINCIPAL.cbl	オブジェクト標準のタイプコード
TC-REFERENCEDATA.cbl	オブジェクト標準のタイプコード
TC-REPOSITORYID-SEQ.cbl	オブジェクト標準のタイプコード
TC-REPOSITORYID.cbl	オブジェクト標準のタイプコード
TC-REQUEST-SEQ.cbl	オブジェクト標準のタイプコード
TC-REQUEST.cbl	オブジェクト標準のタイプコード
TC-SEQUENCE-CHAR.cbl	オブジェクト標準のタイプコード
TC-SEQUENCE-OCTET.cbl	オブジェクト標準のタイプコード
TC-SEQUENCE-REQUEST.cbl	オブジェクト標準のタイプコード
TC-SEQUENCE-STRING.cbl	オブジェクト標準のタイプコード
TC-SEQUENCE-STRUCTMEMBER.cbl	オブジェクト標準のタイプコード
TC-SEQUENCE-UNIONMEMBER.cbl	オブジェクト標準のタイプコード
TC-SEQUENCE.cbl	オブジェクト標準のタイプコード
TC-SERVERREQUEST.cbl	オブジェクト標準のタイプコード
TC-SHORT.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-BAD-INV-ORDER.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-BAD-OPERATION.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-BAD-PARAM.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-BAD-TYPECODE.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-CODESET-INCOM.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-COMM-FAILURE.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-CONTEXT.cbl	オブジェクト標準のタイプコード

TC-STEXCEP-DATA-CONVERSION.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-FREE-MEM.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-IMP-LIMIT.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-INITIALIZE.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-INTERNAL.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-INTF-REPOS.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-INV-FLAG.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-INV-IDENT.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-INV-OBJREF.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-INV-POLICY.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-MARSHAL.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-NO-IMPLEMENT.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-NO-MEMORY.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-NO-PERMISSION.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-NO-RESOURCES.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-NO-RESPONSE.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-OBJ-ADAPTER.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-PERSIST-STORE.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-REBIND.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-TIMEOUT.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-TRANSACTION-M.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-TRANSACTION-U.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-TRANSIENT.cbl	オブジェクト標準のタイプコード
TC-STEXCEP-UNKNOWN.cbl	オブジェクト標準のタイプコード
TC-STRING.cbl	オブジェクト標準のタイプコード
TC-STRUCT.cbl	オブジェクト標準のタイプコード
TC-STRUCTMEMBER.cbl	オブジェクト標準のタイプコード
TC-STRUCTMEMBERSEQ.cbl	オブジェクト標準のタイプコード
TC-TCKIND.cbl	オブジェクト標準のタイプコード
TC-TYPECODE-BADKIND.cbl	オブジェクト標準のタイプコード
TC-TYPECODE-BOUNDS.cbl	オブジェクト標準のタイプコード
TC-TYPECODE.cbl	オブジェクト標準のタイプコード

TC-ULONG.cbl	オブジェクト標準のタイプコード
TC-UNION.cbl	オブジェクト標準のタイプコード
TC-UNIONMEMBER.cbl	オブジェクト標準のタイプコード
TC-UNIONMEMBERSEQ.cbl	オブジェクト標準のタイプコード
TC-USHORT.cbl	オブジェクト標準のタイプコード
TC-VOID.cbl	オブジェクト標準のタイプコード

(8)その他

ALIASDEF.cbl	AliasDefオブジェクトリファレンス
ARRAYDEF.cbl	AliasDefオブジェクトリファレンス
ATTRDESCRIPTIONSEQ.cbl	AttributeDef構造体のsequence
ATTRIBUTEDEF.cbl	AttributeDefオブジェクトリファレンス
ATTRIBUTEDESCRIPTION.cbl	AttributeDef情報構造体
ATTRIBUTEMODE.cbl	AttributeDef情報構造体の属性
BOA.cbl	BOAオブジェクトリファレンス
CDR.cbl	CDR情報構造体
CHAR-SEQ.cbl	Char型のsequence
CONSTANTDEF.cbl	ConstantDefオブジェクトリファレンス
CONSTANTDESCRIPTION.cbl	ConstantDef情報構造体
CONTAINED-DESCRIPTION.cbl	Containedオブジェクト情報構造体
CONTAINED.cbl	Containedオブジェクト
CONTAINEDSEQ.cbl	Containedオブジェクトのsequence
CONTAINER-DESCRIPTION.cbl	Containedオブジェクト情報構造体
CONTAINER-DESCRIPTIONSEQ.cbl	Containedオブジェクト情報構造体のsequence
CONTAINER.cbl	Containedオブジェクトリファレンス
CONTEXT.cbl	Contextオブジェクトリファレンス
CONTEXTIDENTIFIER.cbl	Context識別名
CONTEXTIDSEQ.cbl	Context識別名のsequence
DEFINITIONKIND.cbl	オブジェクト種別Enum
ENUMDEF.cbl	EnumDefオブジェクトリファレンス
ENUMMEMBERSEQ.cbl	Enumメンバのsequence
FIXEDDEF.cbl	FixedDefオブジェクトリファレンス

IDENTIFIER.cbl	識別名
IDLTYPE.cbl	IDLTypeオブジェクトリファレンス
IMPLEMENTATIONDEF.cbl	ImplementationDefオブジェクトリファレンス
MARSHAL.cbl	例外
MODULEDEF.cbl	ModuleDefオブジェクトリファレンス
MODULEDESCRIPTION.cbl	ModuleDef情報構造体
NAMEDVALUE.cbl	NamedValue構造体
NVLIST.cbl	NVList型
OBJECTID.cbl	オブジェクトを示すID
OBJECTIDLIST.cbl	オブジェクトを示すIDのList
OBJECT-NIL.cbl	CORBA-OBJECT-NILオブジェクト
OCTET-SEQ.cbl	CORBA_Octet型のsequence
OPDESCRIPTIONSEQ.cbl	オペレーション情報構造体のsequence
OPERATIONDEF.cbl	オペレーションオブジェクトリファレンス
OPERATIONDESCRIPTION.cbl	オペレーション情報構造体
OPERATIONMODE.cbl	オペレーションの属性種別
ORB.cbl	ORBオブジェクトリファレンス
PARAMETERDESCRIPTION.cbl	パラメタ情報構造体
PARAMETERMODE.cbl	パラメタの属性種別
PARDESCRIPTIONSEQ.cbl	パラメタ情報構造体のsequence
PRIMITIVEDEF.cbl	PrimitiveDefオブジェクトリファレンス
PRIMITIVEKIND.cbl	Primitiveオブジェクトリファレンスの種別情報
PRINCIPAL.cbl	Principalオブジェクトリファレンス
REFERENCEDATA.cbl	リファレンスデータ
REPLACE.cbl	継承時に使用する
REPOSITORY.cbl	リポジトリオブジェクトリファレンス
REPOSITORYID.cbl	リポジトリオブジェクトを示すID
REPOSITORYIDSEQ.cbl	リポジトリオブジェクトを示すIDのsequence
REQUEST-SEQ.cbl	リクエストオブジェクトのsequence
REQUEST.cbl	リクエストオブジェクトリファレンス
SCOPEDNAME.cbl	スコープ名
SEQUENCE.cbl	CORBA-sequence型

SEQUENCEDEF.cbl	SequenceDefオブジェクトリファレンス
SERVERREQUEST.cbl	サーバリクエストオブジェクトリファレンス
STRINGDEF.cbl	StringDefオブジェクトリファレンス
STRUCTDEF.cbl	StructDefオブジェクトリファレンス
STRUCTMEMBER.cbl	Structメンバ情報構造体
STRUCTMEMBERSEQ.cbl	Structメンバ情報構造体のsequence
TCKIND.cbl	タイプコード種別enum
TYPEDEFDEF.cbl	TypeDefオジェクトリファレンス
TYPEDESCRIPTION.cbl	TypeDef情報構造体
UNIONDEF.cbl	UnionDefオジェクトリファレンス
UNIONMEMBER.cbl	Unionオブジェクトのメンバ情報構造体
UNIONMEMBERSEQ.cbl	Unionオブジェクトのメンバ情報構造体のsequence
VERSIONSPEC.cbl	バージョン情報
WSTRINGDEF.cbl	WStringDefオブジェクトリファレンス

第7章 CORBAアプリケーションの基本的な機能

7.1 クライアントタイムアウト機能

クライアントアプリケーションが静的起動インタフェースでサーバメソッドを呼び出す場合、スタブ内の関数を発行します。サーバにリクエストを送信し、サーバからリプライ応答を受信すると、スタブ内の関数は復帰します。そのため、サーバアプリケーションのハングアップやネットワーク異常でリプライ応答を受信できない場合、スタブ内の関数が復帰せず、クライアントアプリケーションはハングアップすることになります。

クライアントタイムアウト機能を使用すると、リクエストを送信してから一定時間（クライアントタイムアウト時間）内にリプライ応答を受信できなかった場合に、スタブ内の関数から復帰することが可能です。このとき、アプリケーションには、マイナーコード“0x464a0101”のCOMM_FAILURE例外が通知されます。
なお、クライアントタイムアウトが発生しても、サーバアプリケーション側のリクエスト処理がキャンセルされるわけではないので、注意してください。

システム全体のクライアントタイムアウト時間の設定方法

CORBAサービスの動作環境ファイル(config)のperiod_receive_timeoutパラメタ、またはInterstage管理コンソールの[CORBAサービスの詳細設定]の“クライアントタイムアウト時間”パラメタで、そのシステムにおけるCORBAアプリケーションのクライアントタイムアウト時間を設定します。詳細については、“チューニングガイド”または“Interstage管理コンソール ヘルプ”を参照してください。
Portable-ORBの場合は、porbeditenvコマンドの［タイムアウト時間］で設定します。詳細については、“リファレンスマニュアル(コマンド編)”を参照してください。

スレッド単位のクライアントタイムアウト時間の設定方法

CORBAアプリケーションのメソッドごとにクライアントタイムアウト時間を変更したい場合、クライアントタイムアウト時間設定用のAPIを発行することにより、スレッド単位でクライアントタイムアウト時間を設定できます。
各開発言語のクライアントタイムアウト時間設定用APIを以下に示します。詳細については、“リファレンスマニュアル（API編）”を参照してください。

開発言語	API名
C言語	CORBA_ORB_set_client_request_timer
C++言語	CORBA::ORB::set_client_request_timer
Java	com.fujitsu.ObjectDirector.CORBA.ORB.set_client_request_timer
COBOL	CORBA-ORB-SET-CLIENT-REQUEST-TIMER

C言語／C++言語／Javaでは、上記APIを使用したサンプルプログラムを提供しています。サンプルプログラムの作成および実行方法については、“付録D サンプルプログラム”を参照してください。

7.2 CORBAワークユニット

本製品では、業務アプリケーションの運用性向上のため、運用状態を監視するアプリケーション管理機能を提供しています。

アプリケーション管理機能は、アプリケーションを業務単位ごとに運用するものです。この業務単位の枠組みをワークユニットと呼びます。ワークユニットによる運用では、以下のような機能により、業務システムの運用形態に応じた環境構築と運用監視を行うことができます。

- 業務単位での起動／停止
- アプリケーションの性能監視
- プロセス多重度の指定
- アプリケーション異常終了時の自動再起動
- アプリケーションのタイマ監視

CORBAアプリケーションの運用にあたり、ワークユニット運用に適用したものをCORBAワークユニットといいます。

参照

ワークユニットに関する詳細情報については、“OLTPサーバ運用ガイド”を参照してください。

第2部 アプリケーション開発(応用編)

第8章 動的起動インタフェース.....547

第9章 ネーミングサービスのプログラミング.....580

第10章 インタフェースリポジトリサービスのプログラミング.....618

第11章 CORBAアプリケーションの高度な使い方.....645

第12章 CORBAインタフェース.....681

第13章 ネーミングサービスのイニシャルリファレンスの獲得.....692

第8章 動的起動インタフェース

8.1 動的起動インタフェースについて

動的起動インタフェースを使用することにより、実行時に動的にパラメタを組み立ててサーバアプリケーションのメソッドを呼び出すことが可能です。

静的起動インタフェースでは、クライアントアプリケーションの開発時にスタブを使用するため、呼び出すサーバアプリケーションのメソッドの型（インタフェース）が明らかとなっている必要があります。動的起動インタフェースでは、開発の時点に、メソッドの型（インタフェース）が明らかとなっている必要はありません。また、十分に考慮されていれば、クライアントアプリケーションを修正せずに、サーバのメソッドを追加できる可能性があります。
ただし、動的起動インタフェースを使用した場合、サーバアプリケーションのインタフェース情報を取得するために、インタフェースリポジトリと通信する分だけ、静的起動インタフェースに比べて性能は不利となります。また、静的起動インタフェースを使用する場合よりもコーディングが複雑になりやすいため、静的起動インタフェースが使用可能である場合は、静的起動インタフェースを使用することを推奨します。

8.2 C言語の開発

動的起動インタフェース使用する場合のクライアントアプリケーションの流れを以下に示します。

8.2.1 初期化

CORBAの初期化メソッドCORBA_ORB_init()を呼び出して、初期化処理を行います。結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
main( argc, argv )
    int     argc;
    char    *argv[];
{
    int                 current_argc = argc;
    CORBA_ORB           orb;    /* ORBのオブジェクトリファレンス */
    CORBA_Environment   env;    /* 例外情報 */

    orb = CORBA_ORB_init( &current_argc, argv, FJ_OM_ORBid, &env );
```

8.2.2 ネーミングサービスのオブジェクトリファレンスの獲得

実行するオブジェクトをネーミングサービスから検索するために、ネーミングサービスのオブジェクトリファレンスが必要となります。CORBAインタフェースのオブジェクトリファレンスの取出しメソッドCORBA_ORB_resolve_initial_references()によりネーミングサービスのオブジェクトリファレンスを取り出します。このとき、CORBA_ORB_ObjectId_NameServiceをメソッドのパラメタとして指定します。

```
CosNaming_NamingContext  cos_naming;  /* NamingServiceのオブジェクトリファレンス */
CosNaming_Name           name;        /* 検索したい情報格納域 */

/* NamingServiceのオブジェクトリファレンスを獲得 */
cos_naming = CORBA_ORB_resolve_initial_references (
               orb,
               CORBA_ORB_ObjectId_NameService,
               &env);
```

8.2.3 インタフェースリポジトリからサーバアプリケーション情報の獲得

インタフェースリポジトリには、IDLで定義されたモジュール名、インタフェース名、オペレーション名、パラメタが階層構造で格納されます。サーバアプリケーションの情報を獲得するため、以下の手順で検索します。

(1)InterfaceDefオブジェクトリファレンスの獲得

インタフェースリポジトリからサーバアプリケーションの情報を獲得するためには、InterfaceDefオブジェクトのオブジェクトリファレンスが必要となります。このため、サーバアプリケーションのオブジェクト名をパラメタとして、ネーミングサービスからサーバアプリケーションのオブジェクトリファレンスを求め、その後、CORBA_Object_get_interface()により、InterfaceDefオブジェクトリファレンスを獲得します。

```
CORBA_Object          obj;   /* サーバアプリケーションのオブジェクトリファレンス */
CosNaming_Name        name;  /* 検索したい情報格納域 */
CORBA_InterfaceDef    intf;  /* InterfaceDefオブジェクトリファレンス */

name._length = name._maximum = 1;          /* オブジェクト名の数 */
name._buffer = &name_component;            /* オブジェクト名格納域 */
name_component.id = "ODdemo::calculator";  /* オブジェクト名 */
name_component.kind = "";                  /* オブジェクトのタイプ */

/* サーバアプリケーションのオブジェクトリファレンスを獲得 */
obj = CosNaming_NamingContext_resolve (
        cos_naming,
        &name,
        &env );

/* InterfaceDefオブジェクトリファレンス取得 */
intf = CORBA_Object_get_interface(obj, &env);
```

(2)OperationDefオブジェクトリファレンスの獲得

CORBA_InterfaceDef_lookup_name()により、指定されたメソッドをインタフェースリポジトリから検索します。パラメタとしてサーバアプリケーションのメソッド名を指定します。結果として、指定したメソッド情報が格納されているOperationDefオブジェクトのオブジェクトリファレンスが返されます。

```
CORBA_ContainedSeq    *intf_opr;

/* メソッドのOperationDefオブジェクトリファレンス取得 */
intf_opr = CORBA_InterfaceDef_lookup_name(
            intf,
            "calculate",
            -1,
            CORBA_dk_Operation,     /* Operation情報の取得 */
            CORBA_FALSE,
            &env );
```

(3)パラメタ情報の獲得

CORBA_OperationDef_describe()により、サーバアプリケーションがもっているメソッドのパラメタの情報(パラメタの名前、個数、パラメタの型など)をインタフェースリポジトリから検索します。パラメタとして、OperationDefオブジェクトリファレンスを指定します。

```
/* OperationDefオブジェクトが保持しているパラメタ情報の構造体 */
CORBA_Contained_Description    *description;

/* パラメタ情報の構造体の取得 */
description = CORBA_OperationDef_describe( intf_opr->_buffer[0], &env );
```

8.2.4 パラメタの組立て

(1)パラメタリストの生成

CORBA_ORB_create_list()により、サーバアプリケーションに渡すパラメタを格納するための領域を保持するリストオブジェクト(NVListオブジェクト)を生成します。このとき、何個のパラメタを格納するかを指定します。結果として、リストオブジェクトのオブジェクトリファレンスが返されます。NVListオブジェクトインタフェースの詳細については、“12.2 NVListオブジェクト”を参照してください。

```
CORBA_OperationDescription    *opr_description; /* パラメタ情報の構造体 */
CORBA_any                     *tmp_any;         /* ワーク用 */
CORBA_ParDescriptionSeq       *params;          /* パラメタ情報 */
CORBA_NVList                  arg_list;         /* リストオブジェクト */

/* Contained_Description構造体から構造体の抽出 */
tmp_any = &description->value;
opr_description = (CORBA_OperationDescription *)tmp_any->_value;

/* パラメタ情報の抽出 */
params = &opr_description->parameters;

/* リストオブジェクトの生成 */
CORBA_ORB_create_list( orb,
        params->_length,
        &arg_list,
        &env );
```

(2)パラメタリストの設定

CORBA_NVList_add_item()により、サーバアプリケーションに渡すパラメタをリストオブジェクトに設定します。パラメタとして、CORBA_NVListオブジェクトリファレンス、サーバアプリケーションのパラメタの名前、型、値、および長さを設定します。
CORBA_NVList_add_item()によりパラメタ情報をリストオブジェクトに設定する順番は、IDL定義に従う必要があります。IDL定義において、メソッドの第1引数がinパラメタ、第2引数がoutパラメタ、第3引数がinoutパラメタの場合、inパラメタ、outパラメタ、inoutパラメタの

順番でリストオブジェクトに設定しなければなりません。
パラメタの設定方法の詳細については、“データ型に対するマッピング”の“8.2.8 動的インタフェースでのパラメタの獲得／解放”を参照してください。

```
CORBA_long    a,b;

/* パラメタの設定 */
CORBA_NVList_add_item(
    arg_list,                   /* リストオブジェクト */
    params->_buffer[0].name,    /* パラメタ名 */
    TC_long,                    /* パラメタの型 */
    &a,                         /* パラメタの値 */
    sizeof( CORBA_long ),       /* パラメタの長さ */
    CORBA_ARG_IN,               /* パラメタのタイプ(in、out、inout) */
    &env);

CORBA_NVList_add_item(
    arg_list,                   /* リストオブジェクト */
    params->_buffer[1].name,    /* パラメタ名 */
    TC_long,                    /* パラメタの型 */
    &b,                         /* パラメタの値 */
    sizeof( CORBA_long ),       /* パラメタの長さ */
    CORBA_ARG_IN,               /* パラメタのタイプ(in、out、inout) */
    &env);
```

8.2.5 リクエストの作成

CORBA_Object_create_request()により、リクエストオブジェクトを作成します。リクエストオブジェクトに対して、サーバオブジェクトのオブジェクトリファレンス、NVListオブジェクトリファレンス、後述するNamedValueと呼ぶサーバの処理結果を格納する領域を指定します。
結果として、リクエストオブジェクトのオブジェクトリファレンスが返されます。
ここで獲得したリクエストオブジェクトは、不要になった時点で削除する必要があります。オブジェクトの削除については、“8.2.7 リクエストの削除”を参照してください。

```
CORBA_Request       request;                        /* リクエストオブジェクト */
CORBA_NamedValue    result;                         /* 復帰 */

result.argument._type = TC_ODdemo_calculator_result;  /* 復帰パラメタの型 */
result.len = sizeof( ODdemo_calculator_result );      /* 復帰パラメタの長さ */
result.argument._value = NULL;

/*リクエストオブジェクトの生成 */
CORBA_Object_create_request(
    obj,                /* サーバアプリケーションのオブジェクトリファレンス */
    CORBA_OBJECT_NIL,   /* context */
    "calculate",        /* メソッド名 */
    arg_list,           /* 入力パラメタ */
    &result,            /* 復帰値 */
    &request,
    CORBA_OUT_LIST_MEMORY,
    &env );
```

NamedValueは、サーバの処理結果を格納するための領域であり、以下のように定義されています。
IDLファイルの定義例を以下に示します。

```
module CORBA
{
    struct NamedValue
    {
        identifier    name;        /* パラメタの名前 */
```

```
        any             argument;    /* パラメタの値 */
        long            len;         /* パラメタの長さ */
        Flags           arg_modes;   /* パラメタの引き渡し方法(in,out, inout) */
    };
};
```

C言語でのNamedValueの定義を以下に示します。

```
struct  CORBA_NamedValue
{
    CORBA_identifier    name;        /* パラメタの名前 */
    CORBA_any           argument;    /* パラメタの値 */
    CORBA_long          len;         /* パラメタの長さ */
    CORBA_Flags         arg_modes;   /* パラメタの引き渡し方法(in,out,inout) */
};
```

8.2.6 リクエストの送信

サーバアプリケーションに対してリクエストを出します。リクエストの方法として、同期送信と非同期送信の2つの方法があります。

(1)同期送信

CORBA_Request_invoke()により、サーバアプリケーションを呼び出します。パラメタとして、リクエストオブジェクトのオブジェクトリファレンスを指定します。

```
CORBA_Request_invoke( request, NULL, &env );
```

サーバアプリケーションの処理結果は、CORBA_NVList_add_item()で設定したCORBA_NVListオブジェクト(本例:arg_list)に格納されます。

(2)非同期送信

CORBA_Request_send()により、サーバアプリケーションを呼び出します。パラメタとして、リクエストオブジェクトのオブジェクトリファレンスを指定します。サーバアプリケーションの処理結果は、CORBA_Request_get_response()により受け取ります。パラメタとして、リクエストオブジェクトのオブジェクトリファレンスを指定します。

```
CORBA_Request_send( request,             /* 処理の要求 */
    (CORBA_Flags) NULL, &env );
   ...
if ( CORBA_Request_get_response(         /* 処理結果の受け取り */
    request,
    CORBA_RESP_NO_WAIT,
    &env ) == CORBA_OK ) {               /* ユーザ指定の処理 */
        .....
}
```

CORBA_Request_get_response()の結果により、サーバアプリケーションからリクエストが完了していないことが判明した場合は、再度CORBA_Request_get_response()を呼び出します。
サーバアプリケーションの処理結果は、CORBA_NVList_add_item()で設定したCORBA_NVListオブジェクト(本例:arg_list)に格納されます。

8.2.7 リクエストの削除

CORBA_Request_delete()により、リクエストオブジェクトを削除します。パラメタとして、リクエストオブジェクトのオブジェクトリファレンスを指定します。また、CORBA_NVlist_free()により、NVListオブジェクトを削除します。パラメタとして、NVListオブジェクトのオブジェクトリ

ファレンスを指定します。
NVListオブジェクトを削除すると、サーバアプリケーション側で獲得されたoutパラメタ領域／inoutパラメタ領域は自動的に解放されます。

```
CORBA_Request_delete(request,&env);     /* リクエストオブジェクトの削除 */
CORBA_NVList_free( arg_list, &env );    /* パラメタ用NVListオブジェクトの削除 */
```

8.2.8 動的インタフェースでのパラメタの獲得／解放

動的起動インタフェース(DII)を使用して動的にパラメタを作成する方法について説明します。
パラメタは、CORBA_NVList_add_item()で設定します。

inパラメタ

サーバアプリケーションにinパラメタを渡す場合は、クライアントアプリケーションでパラメタ領域を獲得し、そのポインタをCORBA_NVList_add_item()の第4パラメタに指定します。

```
CORBA_NVList_add_item(
      arg_list,
      name,                         /* IDLで指定したパラメタの名前を設定します。 */
      type,                         /* パラメタのTypeCodeを設定します。 */
      &param,                       /* パラメタ領域のポインタを設定します */
      sizeof( CORBA_long ),         /* パラメタのサイズを設定します */
      CORBA_ARG_IN,                 /* CORBA_ARG_INを設定します */
      &env );
```

クライアントアプリケーションで獲得した領域は、使用しなくなった時点で解放します。

outパラメタ

サーバアプリケーションの処理結果をoutパラメタで受け取る場合、クライアントアプリケーションで領域を獲得する必要はありません。
CORBA_NVList_add_item()は、以下のように指定します。

```
CORBA_NVList_add_item(
      arg_list,
      name,                         /* IDLで指定したパラメタの名前を設定します。 */
      type,                         /* パラメタのTypeCodeを設定します。 */
      NULL,                         /* NULLを設定します */
      0,                            /* 0を設定します */
      CORBA_ARG_OUT,                /* CORBA_ARG_OUTを設定します */
      &env );
```

サーバアプリケーションから渡されたデータの領域は、CORBA_NVList_free()でNVListクラスの領域を廃棄する際に自動的に解放されます。このため、NVListクラスの領域を解放した後、サーバアプリケーションから渡されたoutパラメタ領域を参照できません。NVListオブジェクトの解放については、“8.2.7 リクエストの削除”を参照してください。

inoutパラメタ

サーバアプリケーションにinoutパラメタを渡す場合は、クライアントアプリケーションでパラメタ領域を獲得し、そのポインタをCORBA_NVList_add_item()の第4パラメタに指定します。

```
CORBA_NVList_add_item(
      arg_list,
      name,                         /* IDLで指定したパラメタの名前を設定します。 */
      type,                         /* パラメタのTypeCodeを設定します。 */
      &param,                       /* パラメタ領域のポインタを設定します */
```

```
    sizeof( CORBA_long ),       /* パラメタのサイズを設定します */
    CORBA_ARG_INOUT,            /* CORBA_ARG_INOUTを設定します */
    &env );
```

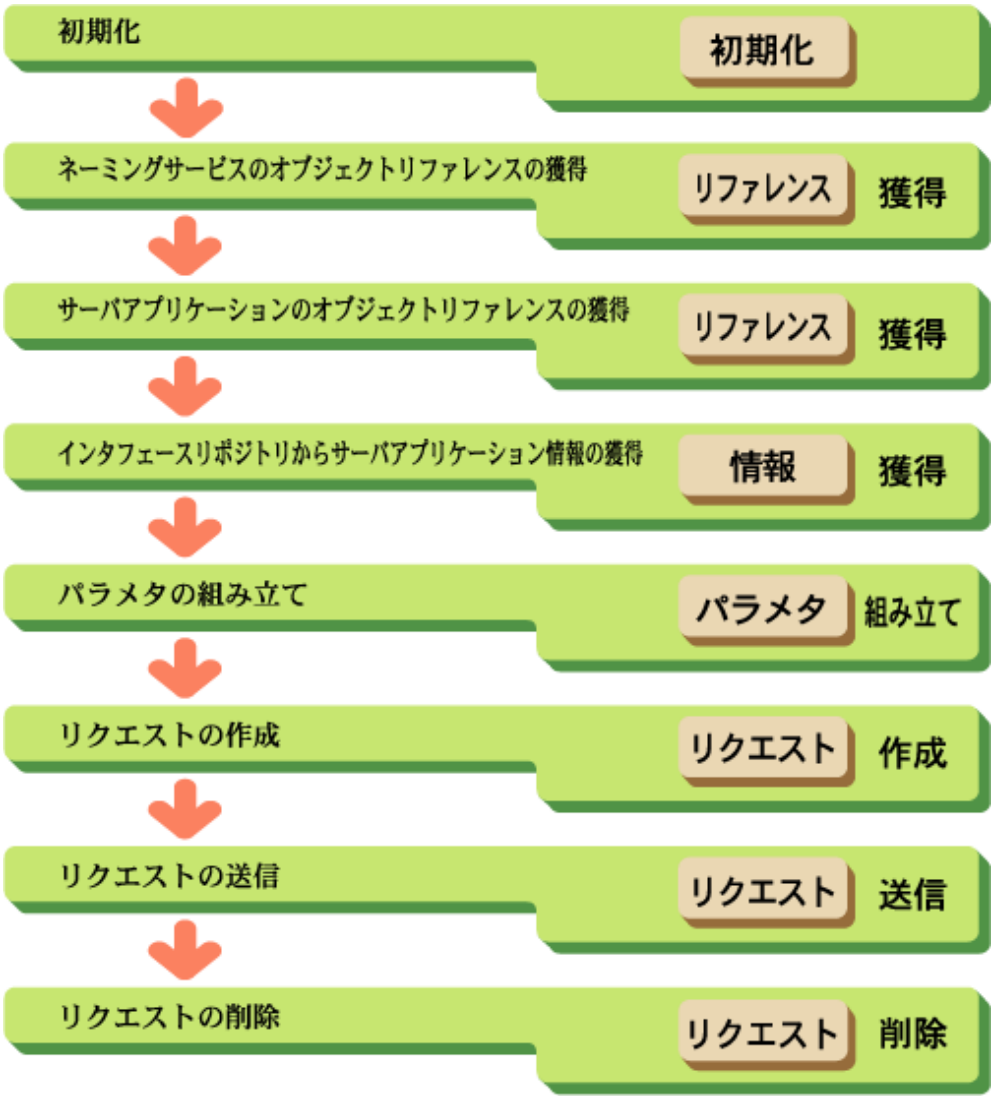
クライアントアプリケーションで獲得したパラメタ領域は、リクエスト発行後、使用しなくなった時点で解放してください。
サーバアプリケーションから渡されたデータ領域は、CORBA_NVList_free()でNVListクラスの領域を廃棄する際に自動的に解放されます。このため、NVListクラスの領域を解放した後、サーバアプリケーションから渡されたinoutパラメタ領域を参照できません。NVListオブジェクトの解放については、“8.2.7 リクエストの削除”を参照してください。

復帰パラメタ

サーバアプリケーションの処理結果を復帰パラメタで受け取る場合、クライアントアプリケーションで領域を獲得する必要はありません。
サーバアプリケーションから渡されたデータ領域は、CORBA_Request_delete()でリクエストオブジェクトを削除する際に、自動的に解放されます。

8.3 C++言語の開発

動的起動インタフェースを使用する場合のクライアントアプリケーションの流れを以下に示します。



8.3.1 初期化

CORBAの初期化メソッドCORBA::ORB_init()を呼び出し、初期化処理を行います。結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
main( argc, argv )
      int    argc;
      char   *argv[];
{
    int                 current_argc = argc;
    CORBA::Environment  env;            // 例外情報
    CORBA::ORB          *orb;           // ORBのオブジェクトリファレンス

    orb = CORBA::ORB_init(&current_argc, argv, FJ_OM_ORBid, env );
```

8.3.2 ネーミングサービスのオブジェクトリファレンスの獲得

実行するオブジェクトをネーミングサービスから検索するために、ネーミングサービスのオブジェクトリファレンスが必要となります。CORBAインタフェースのオブジェクトリファレンスの取出しメソッドCORBA::ORB::resolve_initial_references()によりネーミングサービスのオブジェクトリファレンスを取り出します。このとき、CORBA_ORB_ObjectId_NameServiceをメソッドのパラメタとして指定します。

```
CORBA::Object_ptr    obj;    // NamingServiceのオブジェクトリファレンス

// NamingServiceのオブジェクトリファレンスを獲得
obj = orb->resolve_initial_references(CORBA_ORB_ObjectId_NameService, env );

// NamingContextクラスへの変換
CosNaming::NamingContext_ptr
cos_naming = CosNaming::NamingContext::_narrow( obj );
```

8.3.3 サーバアプリケーションのオブジェクトリファレンスの獲得

ネーミングサービスのメソッドCosNaming::NamingContext::resolve()により、これから実行したいサーバアプリケーションのオブジェクトリファレンスを取り出します。検索したいオブジェクト名を当メソッドのパラメタとして指定します。

```
CosNaming::Name      name;    // CosNaming::Nameのインスタンス
CORBA::Object_ptr    obj;     // サーバアプリケーションのオブジェクトリファレンス

name.length(1);
// オブジェクト名
name[0]->id = (const CORBA::Char *)"ODdemo::calculator";
// オブジェクトのタイプ
name[0]->kind = (const CORBA::Char *)"";

// サーバアプリケーションのオブジェクトリファレンスを獲得
obj = cos_naming->resolve( name, env );
```

8.3.4 インタフェースリポジトリからサーバアプリケーション情報の獲得

インタフェースリポジトリには、IDLで定義されたモジュール名、インタフェース名、オペレーション名、およびパラメタが階層構造で格納されます。サーバアプリケーションの情報を獲得するため、以下の手順で検索します。

(1)InterfaceDefオブジェクトリファレンスの獲得

インタフェースリポジトリからサーバアプリケーションの情報を獲得するためには、InterfaceDefオブジェクトのオブジェクトリファレンスが必要です。そのため、サーバアプリケーションのオブジェクト名をパラメタとして、ネーミングサービスからサーバアプリケーションのオブジェクトリファレンスを求め、その後、CORBA::Object::_get_interface()によりInterfaceDefオブジェクトリファレンスを獲得します。

```
// InterfaceDefオブジェクトリファレンス取得
CORBA::InterfaceDef_ptr  intf = obj->_get_interface( env );
```

(2)OperationDefオブジェクトリファレンスの獲得

CORBA::InterfaceDef::lookup_name()により、インタフェースリポジトリから指定されたメソッドを検索します。パラメタとしてサーバアプリケーションのメソッド名を指定します。結果として、指定したメソッド情報が格納されたOperationDefオブジェクトのオブジェクトリファレンスが返されます。

```
// メソッドのOperationDefオブジェクトリファレンス取得
CORBA::ContainedSeq_ptr intf_opr =  intf->lookup_name(
    "calculate",            // メソッド名
    -1,
    CORBA::dk_Operation,  // Operation情報の取得
    CORBA_FALSE,
    env );

// OperationDefクラスへの変換
CORBA::OperationDef_ptr OperationDef_obj
    = CORBA::OperationDef::_narrow((*intf_opr)[0]);
```

(3)パラメタ情報の獲得

CORBA::OperationDef::describe()により、サーバアプリケーションがもっているメソッドのパラメタの情報(パラメタの名前、個数、パラメタの型など)をインタフェースリポジトリから検索します。パラメタとして、OperationDefオブジェクトリファレンスを指定します。

```
// 指定されたメソッドのパラメタ情報を検索
CORBA::Contained::Description  *description
    = OperationDef_obj->describe( env );
```

8.3.5 パラメタの組立て

(1)パラメタリストの生成

CORBA::ORB::create_list()により、サーバアプリケーションに渡すパラメタを格納するための領域を保持するリストオブジェクト(NVListオブジェクト)を生成します。このとき、何個のパラメタを格納するかを指定します。結果として、リストオブジェクトのオブジェクトリファレンスが返されます。NVListオブジェクトの詳細については、“12.2 NVListオブジェクト”を参照してください。

```
CORBA::Any tmp_any =  *description->value;    // Contained_Description構造体から
                                              // パラメタ情報構造体の抽出
CORBA::OperationDescription *opr_description =
        (CORBA::OperationDescription *)tmp_any.value();

// パラメタ情報の抽出
CORBA::ParDescriptionSeq    *params = &opr_description->parameters;

CORBA::NVList        *arg_list;             // NVList

orb->create_list( params->length(), arg_list, env );  // リストオブジェクトの生成
```

(2)パラメタリストの設定

CORBA::NVList::add_value()により、サーバアプリケーションに渡すパラメタをリストオブジェクトに設定します。パラメタとして、CORBA::NVListオブジェクトリファレンス、サーバアプリケーションのパラメタの名前、型、値、および長さを設定します。
add_value()によりパラメタ情報をリストオブジェクトに設定する順番は、IDL定義に従う必要があります。IDL定義において、メソッドの第1引数がinパラメタ、第2引数がoutパラメタ、第3引数がinoutパラメタの場合、inパラメタ、outパラメタ、inoutパラメタの順番でリストオブジェクトに設定する必要があります。

パラメタの設定方法の詳細については、“データ型に対するマッピング”の“8.3.9 動的インタフェースでのパラメタの獲得／解放”を参照してください。また、any型の詳細については、“データ型に対するマッピング”の “4.6.4 any型”を参照してください。

```
CORBA::Any    p1, p2;
CORBA::Long   x = 10, y = 20;
p1 <<= x;
p2 <<= y;

// パラメタの設定
arg_list->add_value(
    ((*params)[0])->name,     // パラメタ名
    p1,                       // パラメタの型および値
    CORBA::ARG_IN,            // パラメタのタイプ(in,out, inout)
    env );

// パラメタの設定
arg_list->add_value(
    ((*params)[1])->name,    // パラメタ名
    p2,                      // パラメタの型および値
    CORBA::ARG_IN,           // パラメタのタイプ(in,out, inout)
    env );
```

8.3.6 リクエストの作成

CORBA::Object::_create_request()により、リクエストオブジェクトを作成します。リクエストオブジェクトに対して、サーバオブジェクトのオブジェクトリファレンス、NVListオブジェクトリファレンス、後述するNamedValueと呼ぶサーバの処理結果を格納する領域を指定します。結果として、リクエストオブジェクトのオブジェクトリファレンスが返されます。
ここで獲得したリクエストオブジェクトは、不要になった時点で削除する必要があります。オブジェクトの削除については、“8.3.8 リクエストの削除”を参照してください。

```
CORBA::NVList_ptr arg_tmp;

// 復帰パラメタ用リストオブジェクトの生成
orb->create_list(1, arg_tmp, env);

ODdemo::calculator::result *res = NULL;

CORBA::Any any_tmp( _tc_ODdemo_calculator_result, res, CORBA_FALSE );

// 結果受け取り用変数(CORBA::NamedValue型変数)
CORBA::NamedValue_ptr
result = arg_tmp->add_value(
              NULL,             // resultの設定
              any_tmp,
              (CORBA::Flags)0,
              env);

CORBA::Request_ptr  request;

// リクエストオブジェクトの生成
obj->_create_request(
      CORBA_OBJECT_NIL,       // context
      "calculate",            // メソッド名
      arg_list,               // 入力パラメタ
      result,                 // 復帰値
      request,                // 出力パラメタの格納域
      CORBA::OUT_LIST_MEMORY,
      *env );
```

NamedValueは、サーバの処理結果を格納するための領域であり、以下のように定義されています。
IDLファイルでの定義を以下に示します。

```
module CORBA
{
    struct NamedValue
    {
        identifier     name;          // パラメタの名前
        any            argument;      // パラメタの値
        long           len;           // パラメタの長さ
        Flags          arg_modes;     // パラメタの引き渡し方法(in,out,inout)
    };
};
```

C++言語でのNamedValueの定義を以下に示します。

```
class CORBA
{
    class NamedValue
    {
        public:
        const char *name(CORBA::Environment &) const;  // パラメタの名前
        Any *value(CORBA::Environment &) const;        // パラメタの値
        Flags flags(CORBA::Environment &) const;       // パラメタの引き渡し方法(in,out,inout)
    };
};
```

8.3.7 リクエストの送信

サーバアプリケーションに対してリクエストを出します。リクエストの方法として、以下の2つの方法があります。

- 同期送信
- 非同期送信

(1)同期送信

CORBA::Request::invoke()により、サーバアプリケーションを呼び出します。パラメタとして、リクエストオブジェクトのオブジェクトリファレンスを指定します。

```
request->invoke( env );
```

(2)非同期送信

CORBA::Request::send_deferred()により、サーバアプリケーションを呼び出します。パラメタとして、リクエストオブジェクトのオブジェクトリファレンスを指定します。サーバアプリケーションの処理結果は、CORBA::Request::get_response()により受け取ります。パラメタとして、リクエストオブジェクトのオブジェクトリファレンスを指定します。

```
request->send_deferred( env ); // 処理の要求
request->get_response( env );  // 処理結果の受け取り
```

CORBA::Request::get_response()サーバアプリケーションからリクエストが完了していないことが判明した場合は、再度CORBA::Request::get_response()を呼び出します。

処理結果の取り出し

処理結果を受け取った後、CORBA::NVList::item()により、サーバアプリケーションから受け取ったパラメタ情報を取り出します。パラメタ情報の取出しの詳細については、“データ型に対するマッピング”の“8.3.9 動的インタフェースでのパラメタの獲得／解放”を参照してください。

```
CORBA::NamedValue_ptr nv;

// 処理結果の取り出し
nv = arg_list->item(1, env);  // 1番目のパラメタの取り出し
CORBA::Any *ap = nv->value( env );
CORBA::Long *p = (CORBA::Long *)ap->value();

cout << *(CORBA::Long *)p << endl;    // 結果の出力
```

8.3.8 リクエストの削除

CORBA::release()によりリクエストオブジェクトを削除します。パラメタとして、リクエストオブジェクトのオブジェクトリファレンスを指定します。また、同様に、CORBA::release()によりNVListオブジェクトを削除します。パラメタとして、NVListオブジェクトのオブジェクトリファレンスを指定します。
NVListオブジェクトを削除すると、サーバアプリケーション側で獲得されたoutパラメタ領域／inoutパラメタ領域／復帰パラメタ領域は、自動的に解放されます。

```
CORBA::release( request );    // リクエストオブジェクトの削除
CORBA::release( arg_list );   // パラメタ用NVListオブジェクトの削除
CORBA::release( arg_tmp );    // 復帰パラメタ用NVListオブジェクトの削除
```

8.3.9 動的インタフェースでのパラメタの獲得／解放

動的起動インタフェース(DII)を使用して動的にパラメタを作成する方法について説明します。また、作成したパラメタから値を取り出す方法についても説明します。
パラメタは、CORBA::NVList::add_value()を使用して設定します。値の取出しには、CORBA::NVList::item()を使用します。

ここでは、データ型ごとに例を挙げて説明します。例では、以下のIDL定義を使用します。

(1)基本データ型

```
module ODsample{
    interface  dyn1{
        long  add( in    long  a,    // 0番目のパラメタ"a"
                   out   long  b,    // 1番目のパラメタ"b"
                   inout long  c );  // 2番目のパラメタ"c"
    };
};
```

(2)文字列型、ワイド文字列型

```
module ODsample{
    interface  diistring{
        string  stringtest( in    string  a,    // 0番目のパラメタ"a"
                            out   string  b,    // 1番目のパラメタ"b"
                            inout string  c );  // 2番目のパラメタ"c"
    };
};
```

(3)構造体

```
module ODsample{
    interface  diistruct{
        struct  data{
              long  para1;
              char  para2;
        };
        data  structtest( in    data  a,     // 0番目のパラメタ"a"
                          out   data  b,     // 1番目のパラメタ"b"
                          inout data  c );  // 2番目のパラメタ"c"
    };
};
```

(4)シーケンス型

```
module ODsample{
    interface  diiseq{
        typedef sequence<long>  sampleseq;
        sampleseq  seqtest( in    sampleseq  a,     // 0番目のパラメタ"a"
                            out   sampleseq  b,     // 1番目のパラメタ"b"
                            inout sampleseq  c );  // 2番目のパラメタ"c"
    };
};
```

inパラメタ

サーバアプリケーションにinパラメタを渡す場合、クライアントアプリケーションでパラメタ領域を獲得し、add_value()の第2パラメタ(Any型)に指定します。Any型に値を設定する方法については、“4.6.4 any型”を参照してください。
動的に獲得した領域は、リクエスト発行後、使用しなくなった時点で解放してください。
inパラメタの作成例を以下に示します。

(1)基本データ型

```
CORBA::NVList_ptr       arg_list;
CORBA::Environment_ptr  env = new CORBA::Environment;

// リストオブジェクトの生成
orb->create_list( 3, arg_list, *env );

// パラメタ領域の獲得
CORBA::Long         x = 100;
// Any型にパラメタ領域を設定
CORBA::Any          p0;
p0 <<= x;

// inパラメタの作成
arg_list->add_value(
      "a",                  /* IDLで指定したパラメタの名前を設定します */
      p0,                   /* inで使用する値を設定したAny型を設定します */
      CORBA::ARG_IN,        /* CORBA::ARG_INを設定します */
      *env );

// パラメタ領域の解放
// p0のリリースフラグがCORBA_TRUEに設定されているため、デストラクタの延長で
// 自動的に解放されます
```

(2)文字列型、ワイド文字列型

```
CORBA::NVList_ptr       arg_list;
CORBA::Environment_ptr  env = new CORBA::Environment;

// リストオブジェクトの生成
orb->create_list( 3, arg_list, *env );

// パラメタ領域の獲得
CORBA::Char         *str0 = CORBA::string_alloc(3);
strcpy( str0, "IN" );
// Any型にパラメタ領域を設定
CORBA::Any          p0( _tc_string, str0 );

// inパラメタの作成
arg_list->add_value(
      "a",                  /* IDLで指定したパラメタの名前を設定します */
      p0,                   /* inで使用する値を設定したAny型を設定します */
      CORBA::ARG_IN,        /* CORBA::ARG_INを設定します */
      *env );

// パラメタ領域の解放
//  リクエスト発行後、str0が必要なくなった時点で解放します
CORBA::string_free( str0 );
```

(3)構造体

```
CORBA::NVList_ptr       arg_list;
CORBA::Environment_ptr  env = new CORBA::Environment;

// リストオブジェクトの生成
orb->create_list( 3, arg_list, *env );

// パラメタ領域の獲得
ODsample::diistruct::data  *data0 = new ODsample::diistruct::data;
data0->para1 = 10;
data0->para2 = 'x';
// Any型にパラメタ領域を設定
CORBA::Any          p0( _tc_ODsample_diistruct_data, data0 );
                        /* _tc_ODsample_diistruct_dataは、IDLcによって */
                        /* 生成されるヘッダファイルに定義されます */

// inパラメタの作成
arg_list->add_value(
      "a",                  /* IDLで指定したパラメタの名前を設定します */
      p0,                   /* inで使用する値を設定したAny型を設定します */
      CORBA::ARG_IN,        /* CORBA::ARG_INを設定します */
      *env );

// パラメタ領域の解放
//  リクエスト発行後、data0が必要なくなった時点で解放します
delete data0;
```

(4)シーケンス型

```
CORBA::NVList_ptr       arg_list;
CORBA::Environment_ptr  env = new CORBA::Environment;

// リストオブジェクトの生成
orb->create_list( 3, arg_list, *env );

// パラメタ領域の獲得
```

```
CORBA::Long         *p = ODsample::diiseq::sampleseq::allocbuf(3);
for( int i = 0; i < 3; i++ )
    p[i] = i*10;
ODsample::diiseq::sampleseq  *seq0 =
                    new ODsample::diiseq::sampleseq( 3, 3, p, CORBA_TRUE );
// Any型にパラメタ領域を設定
CORBA::Any          p0( _tc_ODsample_diiseq_sampleseq, seq0 );
                        /* _tc_ODsample_diiseq_sampleseqは、IDLcによって */
                        /* 生成されるヘッダファイルに定義されます */

// inパラメタの作成
arg_list->add_value(
      "a",              /* IDLで指定したパラメタの名前を設定します */
      p0,               /* inで使用する値を設定したAny型を設定します */
      CORBA::ARG_IN,    /* CORBA::ARG_INを設定します */
      *env );

// パラメタ領域の解放
//  リクエスト発行後、seq0が必要なくなった時点で解放します
delete seq0;            /* リリースフラグがCORBA_TRUEに設定されているため、*/
                        /* allocbufで獲得した領域（p）も解放されます */
```

outパラメタ

サーバアプリケーションの処理結果をoutパラメタで受け取る場合、クライアントアプリケーションで領域を獲得する必要はありません。
add_value()の第2パラメタには、TypeCodeを指定したAny型を設定します。
サーバアプリケーションから渡されたデータ用に獲得された領域は、NVListクラスの領域を廃棄する際に自動的に解放されます。このため、NVListクラスの領域を解放したあと、サーバアプリケーションから渡されたoutパラメタ領域を参照できません。NVListオブジェクトの解放については、“8.3.8 リクエストの削除”を参照してください。
outパラメタの作成例と処理結果の取出しの例を以下に示します。

(1)基本データ型

```
// Any型にTypeCodeを設定
CORBA::Any          p1( _tc_long, NULL );

// outパラメタの作成
arg_list->add_value(
      "b",              /* IDLで指定したパラメタの名前を設定します */
      p1,               /* TypeCodeを指定したAny型を設定します */
      CORBA::ARG_OUT,   /* CORBA::ARG_OUTを設定します */
      *env );

// 処理結果の取り出し
CORBA::NamedValue_ptr  nv1 = arg_list->item(1, *env);      /* 1番目のパラメタを取り出します */
CORBA::Any             *ap1 = nv1->value( *env );          /* パラメタからAnyを取り出します */
CORBA::Long            *lp1 = (CORBA::Long *)ap1->value(); /* Anyから値を取り出します */

// パラメタ領域の解放
//  lp1の領域はCORBA::release(arg_list)によって解放されます
```

(2)文字列型、ワイド文字列型

```
// Any型にTypeCodeを設定
CORBA::Any          p1( _tc_string, NULL );

// outパラメタの作成
arg_list->add_value(
      "b",              /* IDLで指定したパラメタの名前を設定します */
```

```
      p1,                   /* TypeCodeを指定したAny型を設定します */
      CORBA::ARG_OUT,       /* CORBA::ARG_OUTを設定します */
      *env );

// 処理結果の取り出し
CORBA::NamedValue_ptr  nv1 = arg_list->item(1, *env);      /* 1番目のパラメタを取り出します */
CORBA::Any             *ap1 = nv1->value( *env );          /* パラメタからAnyを取り出します */
CORBA::Char            *cp1 = (CORBA::Char *)ap1->value(); /* Anyから値を取り出します */

// パラメタ領域の解放
//  cp1の領域はCORBA::release(arg_list)によって解放されます
```

(3)構造体

```
// Any型にTypeCodeを設定
CORBA::Any          p1( _tc_ODsample_diistruct_data, NULL );
                        /* _tc_ODsample_diistruct_dataは、IDLcによって */
                        /* 生成されるヘッダファイルに定義されます */

// outパラメタの作成
arg_list->add_value(
      "b",              /* IDLで指定したパラメタの名前を設定します */
      p1,               /* TypeCodeを指定したAny型を設定します */
      CORBA::ARG_OUT,   /* CORBA::ARG_OUTを設定します */
      *env );

// 処理結果の取り出し
CORBA::NamedValue_ptr  nv1 = arg_list->item(1, *env);      /* 1番目のパラメタを取り出します */
CORBA::Any             *ap1 = nv1->value( *env );          /* パラメタからAnyを取り出します */
ODsample::diistruct::data  *dp1 =
          (ODsample::diistruct::data *)(ap1->value());    /* Anyから値を取り出します */

// パラメタ領域の解放
//  dp1の領域はCORBA::release(arg_list)によって解放されます
```

(4)シーケンス型

```
// Any型にTypeCodeを設定
CORBA::Any          p1( _tc_ODsample_diiseq_sampleseq, NULL );
                        /* _tc_ODsample_diiseq_sampleseqは、IDLcによって */
                        /* 生成されるヘッダファイルに定義されます */

// outパラメタの作成
arg_list->add_value(
      "b",              /* IDLで指定したパラメタの名前を設定します */
      p1,               /* TypeCodeを指定したAny型を設定します */
      CORBA::ARG_OUT,   /* CORBA::ARG_OUTを設定します */
      *env );

// 処理結果の取り出し
CORBA::NamedValue_ptr  nv1 = arg_list->item(1, *env);     /* 1番目のパラメタを取り出します */
CORBA::Any             *ap1 = nv1->value( *env );         /* パラメタからAnyを取り出します */
ODsample::diiseq::sampleseq  *sp1 =
          (ODsample::diiseq::sampleseq *)(ap1->value()); /* Anyから値を取り出します */

// パラメタ領域の解放
//  sp1の領域はCORBA::release(arg_list)によって解放されます
//  sp1のリリースフラグはCORBA_TRUEに設定されるため、_bufferの領域も解放されます
```

inoutパラメタ

サーバアプリケーションにinoutパラメタを渡す場合は、クライアントアプリケーションでパラメタ領域を獲得し、add_value()の第2パラメタ(Any型)に指定します。Any型に値を設定する方法については、“4.6.4 any型”を参照してください。
クライアントアプリケーションで獲得したパラメタ領域は、リクエスト発行後、使用しなくなった時点で解放してください。サーバアプリケーションから渡されたデータ用に獲得された領域は、NVListクラスの領域を廃棄する際に自動的に解放されます。このため、NVListクラスの領域を解放したあと、サーバアプリケーションから渡されたinoutパラメタ領域を参照できません。NVListオブジェクトの解放については、“8.3.8 リクエストの削除”を参照してください。
inoutパラメタの作成例と処理結果の取出しの例を以下に示します。

(1)基本データ型

```
// パラメタ領域の獲得
CORBA::Long             z = 200;
// Any型にパラメタ領域を設定
CORBA::Any              p2;
p2 <<= z;

// inoutパラメタの作成
arg_list->add_value(
      "c",                  /* IDLで指定したパラメタの名前を設定します */
      p2,                   /* inoutで使用する値を設定したAny型を設定します */
      CORBA::ARG_INOUT,     /* CORBA::ARG_INOUTを設定します */
      *env );

// 処理結果の取り出し
CORBA::NamedValue_ptr  nv2 = arg_list->item(2, *env);       /* 2番目のパラメタを取り出します */
CORBA::Any             *ap2 = nv2->value( *env );           /* パラメタからAnyを取り出します */
CORBA::Long            *lp2 = (CORBA::Long *)ap2->value(); /* Anyから値を取り出します */

// パラメタ領域の解放
//  p2のリリースフラグがCORBA_TRUEに設定されているため、デストラクタの延長で
//  自動的に解放されます
//  lp2の領域はCORBA::release(arg_list)によって解放されます
```

(2)文字列型、ワイド文字列型

```
// パラメタ領域の獲得
CORBA::Char         *str2 = CORBA::string_alloc(6);
strcpy( str2, "INOUT" );
// Any型にパラメタ領域を設定
CORBA::Any          p2( _tc_string, str2 );

// inoutパラメタの作成
arg_list->add_value(
      "c",                  /* IDLで指定したパラメタの名前を設定します */
      p2,                   /* inoutで使用する値を設定したAny型を設定します */
      CORBA::ARG_INOUT,     /* CORBA::ARG_INOUTを設定します */
      *env );

// 処理結果の取り出し
CORBA::NamedValue_ptr  nv2 = arg_list->item(2, *env);       /* 2番目のパラメタを取り出します */
CORBA::Any             *ap2 = nv2->value( *env );           /* パラメタからAnyを取り出します */
CORBA::Char            *cp2 = (CORBA::Char *)ap2->value(); /* Anyから値を取り出します */

// パラメタ領域の解放
//  リクエスト発行後、str2が必要なくなった時点で解放します
//  cp2の領域はCORBA::release(arg_list)によって解放されます
CORBA::string_free( str2 );
```

(3)構造体

```
// パラメタ領域の獲得
ODsample::diistruct::data  *data2 = new ODsample::diistruct::data;
data2->para1 = 20;
data2->para2 = 'y';
// Any型にパラメタ領域を設定
CORBA::Any          p2( _tc_ODsample_diistruct_data, data2 );
                        /* _tc_ODsample_diistruct_dataは、IDLcによって */
                        /* 生成されるヘッダファイルに定義されます */

// inoutパラメタの作成
arg_list->add_value(
      "c",              /* IDLで指定したパラメタの名前を設定します */
      p2,               /* inoutで使用する値を設定したAny型を設定します */
      CORBA::ARG_INOUT,     /* CORBA::ARG_INOUTを設定します */
      *env );

// 処理結果の取り出し
CORBA::NamedValue_ptr  nv2 = arg_list->item(2, *env);      /* 2番目のパラメタを取り出します */
CORBA::Any             *ap2 = nv2->value( *env );          /* パラメタからAnyを取り出します */
ODsample::diistruct::data  *dp2 =
            (ODsample::diistruct::data *)(ap2->value());   /* Anyから値を取り出します */

// パラメタ領域の解放
//  リクエスト発行後、data2が必要なくなった時点で解放します
//  dp2の領域はCORBA::release(arg_list)によって解放されます
delete data2;
```

(4)シーケンス型

```
// パラメタ領域の獲得
CORBA::Long         *q = ODsample::diiseq::sampleseq::allocbuf(5);
for( int i = 0; i < 5; i++ )
    q[i] = i*100;
ODsample::diiseq::sampleseq  *seq2 =
                    new ODsample::diiseq::sampleseq( 5, 5, q, CORBA_TRUE );
// Any型にパラメタ領域を設定
CORBA::Any          p2( _tc_ODsample_diiseq_sampleseq, seq2 );
                        /* _tc_ODsample_diiseq_sampleseqは、IDLcによって */
                        /* 生成されるヘッダファイルに定義されます */

// inoutパラメタの作成
arg_list->add_value(
      "c",              /* IDLで指定したパラメタの名前を設定します */
      p2,               /* inoutで使用する値を設定したAny型を設定します */
      CORBA::ARG_INOUT,     /* CORBA::ARG_INOUTを設定します */
      *env );

// 処理結果の取り出し
CORBA::NamedValue_ptr  nv2 = arg_list->item(2, *env);      /* 2番目のパラメタを取り出します */
CORBA::Any             *ap2 = nv2->value( *env );          /* パラメタからAnyを取り出します */
ODsample::diiseq::sampleseq  *sp2 =
           (ODsample::diiseq::sampleseq *)(ap2->value()); /* Anyから値を取り出します */

// パラメタ領域の解放
//  リクエスト発行後、seq2が必要なくなった時点で解放します
//  sp2の領域はCORBA::release(arg_list)によって解放されます
//  sp2のリリースフラグはCORBA_TRUEに設定されるため、_bufferの領域も解放されます
delete seq2;                /* リリースフラグがCORBA_TRUEに設定されているため、*/
                            /* allocbufで獲得した領域 (q) も解放されます */
```

復帰パラメタ

サーバアプリケーションの処理結果を復帰パラメタで受け取る場合、クライアントアプリケーションで領域を獲得する必要はありません。
add_value()の第2パラメタには、TypeCodeを指定したAny型を設定します。
サーバアプリケーションから渡されたデータ用に獲得された領域は、NVListクラスの領域を廃棄する際に自動的に解放されます。このため、NVListクラスの領域を解放したあと、サーバアプリケーションから渡された復帰パラメタ領域を参照できません。NVListオブジェクトの解放については、“8.3.8 リクエストの削除”を参照してください。
復帰パラメタの作成例と処理結果の取出しの例を以下に示します。

(1)基本データ型

```
CORBA::NVList_ptr   arg_tmp;

// リストオブジェクトの生成
orb->create_list( 1, arg_tmp, *env );

// Any型にTypeCodeを設定
CORBA::Any          tmp( _tc_long, NULL );

// 復帰パラメタの作成
CORBA::NamedValue_ptr  result =
      arg_tmp->add_value(
            NULL,                 /* NULLを設定します */
            tmp,                  /* TypeCodeを指定したAny型を設定します */
            (CORBA::Flags)0,      /* 0を設定します */
            *env);

// 処理結果の取り出し
CORBA::Any            *ap3 = result->value( *env );       /* パラメタからAnyを取り出します */
CORBA::Long           *lp3 = (CORBA::Long *)ap3->value(); /* Anyから値を取り出します */

// パラメタ領域の解放
// lp3の領域はCORBA::release(arg_tmp)によって解放されます
```

(2)文字列型、ワイド文字列型

```
CORBA::NVList_ptr   arg_tmp;

// リストオブジェクトの生成
orb->create_list( 1, arg_tmp, *env );

// Any型にTypeCodeを設定
CORBA::Any          tmp( _tc_string, NULL );

// 復帰パラメタの作成
CORBA::NamedValue_ptr  result =
      arg_tmp->add_value(
            NULL,                 /* NULLを設定します */
            tmp,                  /* TypeCodeを指定したAny型を設定します */
            (CORBA::Flags)0,      /* 0を設定します */
            *env);

// 処理結果の取り出し
CORBA::Any            *ap3 = result->value( *env );       /* パラメタからAnyを取り出します */
CORBA::Char           *cp3 = (CORBA::Char *)ap3->value(); /* Anyから値を取り出します */

// パラメタ領域の解放
// cp3の領域はCORBA::release(arg_tmp)によって解放されます
```

(3)構造体

```
CORBA::NVList_ptr   arg_tmp;

// リストオブジェクトの生成
orb->create_list( 1, arg_tmp, *env );

// Any型にTypeCodeを設定
CORBA::Any          tmp( _tc_ODsample_diistruct_data, NULL );
                                /* _tc_ODsample_diistruct_dataは、IDLcによって */
                                /* 生成されるヘッダファイルに定義されます */

// 復帰パラメタの作成
CORBA::NamedValue_ptr  result =
      arg_tmp->add_value(
             NULL,              /* NULLを設定します */
             tmp,               /* TypeCodeを指定したAny型を設定します */
             (CORBA::Flags)0,   /* 0を設定します */
             *env);

// 処理結果の取り出し
CORBA::Any             *ap3 = result->value( *env );       /* パラメタからAnyを取り出します */
ODsample::diistruct::data  *dp3 =
           (ODsample::diistruct::data *)(ap3->value());    /* Anyから値を取り出します */

// パラメタ領域の解放
// dp3の領域はCORBA::release(arg_tmp)によって解放されます
```

(4)シーケンス型

```
CORBA::NVList_ptr   arg_tmp;

// リストオブジェクトの生成
orb->create_list( 1, arg_tmp, *env );

// Any型にTypeCodeを設定
CORBA::Any          tmp( _tc_ODsample_diiseq_sampleseq, NULL );
                                /* _tc_ODsample_diiseq_sampleseqは、IDLcによって */
                                /* 生成されるヘッダファイルに定義されます */

// 復帰パラメタの作成
CORBA::NamedValue_ptr  result =
      arg_tmp->add_value(
             NULL,              /* NULLを設定します */
             tmp,               /* TypeCodeを指定したAny型を設定します */
             (CORBA::Flags)0,   /* 0を設定します */
             *env);

// 処理結果の取り出し
CORBA::Any             *ap3 = result->value( *env );      /* パラメタからAnyを取り出します */
ODsample::diiseq::sampleseq  *sp3 =
           (ODsample::diiseq::sampleseq *)(ap3->value()); /* Anyから値を取り出します */

// パラメタ領域の解放
// sp3の領域はCORBA::release(arg_tmp)によって解放されます
// sp3のリリースフラグはCORBA_TRUEに設定されるため、_bufferの領域も解放されます
```

8.4 Java言語の開発

動的起動インタフェースを使用する場合の、クライアントアプリケーションの処理の流れを以下に示します。

8.4.1 初期化

CORBAの初期化メソッドorg.omg.CORBA.ORB.init()を呼び出し、初期化処理を行います。本メソッドの結果として、ORBのオブジェクトリファレンスが通知されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

【アプリケーションの場合の例】

```
public class Sample { //クラスの宣言
    public static void main( String args[] ) {
        org.omg.CORBA.ORB    Orb;      // ORBオブジェクト

        try {
            // ORBの生成と初期化
            Orb = org.omg.CORBA.ORB.init( args, null );
                ...
        }
        catch( java.lang.Exception e ) {
            ...//例外処理
        }
    }
}
```

例外処理try-catchについては、“5.10 クライアントアプリケーションの例外処理”を参照してください。

8.4.2 ネーミングサービスのオブジェクトリファレンスの獲得

実行するオブジェクトをネーミングサービスから検索するために、ネーミングサービスのオブジェクトリファレンスが必要となります。CORBAインタフェースのオブジェクトリファレンスの取出しメソッドorg.omg.CORBA.ORB.resolve_initial_references()によりネーミングサービスのオブジェクトリファレンスを取り出します。このとき、“NameService”をメソッドのパラメタとして指定します。

```
org.omg.CosNaming.NamingContextExt Cos; //NamingService用オブジェクトリファレンス
try {
    //NamingServiceのオブジェクトリファレンスの取得
    org.omg.CORBA.Object _tmpObj =
              Orb.resolve_initial_references( "NameService" );

    // NamingContext型に変換
    Cos = org.omg.CosNaming.NamingContextExtHelper.narrow( _tmpObj );
    if( Cos == null ){
        ...;
    }
}
catch( java.lang.Exception e ){
    ...//例外処理
}
```

8.4.3 サーバアプリケーションのオブジェクトリファレンスの獲得

ネーミングサービスのメソッドCosNaming.NamingContext.resolve()により、これから実行したいサーバアプリケーションのオブジェクトリファレンスを取り出します。検索したいオブジェクト名を本メソッドのパラメタとして指定します。

```
try {
    String NCid = new String( "ODdemo::calculator" );  //オブジェクト名
    String NCkind = new String( "" );                  //オブジェクトのタイプ
    //オブジェクト名、タイプの設定
    org.omg.CosNaming.NameComponent nc =
                    new org.omg.CosNaming.NameComponent( NCid, NCkind );
    org.omg.CosNaming.NameComponent NCo[] = { nc };
    //NamingServiceのresolveメソッドを発行
    org.omg.CORBA.Object target = Cos.resolve( NCo );
}
catch( java.lang.Exception e ) {
    ...//例外処理
}
```

8.4.4 インタフェースリポジトリからサーバアプリケーション情報の獲得

インタフェースリポジトリには、IDLで定義されたモジュール名、インタフェース名、オペレーション名、およびパラメタが階層構造で格納されます。サーバアプリケーションの情報を獲得するため、以下の手順で検索します。

(1)org.omg.CORBA.InterfaceDefオブジェクトリファレンスの獲得

インタフェースリポジトリからサーバアプリケーションの情報を獲得するためには、org.omg.CORBA.InterfaceDefオブジェクトのオブジェクトリファレンスが必要です。org.omg.CORBA.Object._get_interface_def()メソッドを使用します。

```
//org.omg.CORBA.InterfaceDefオブジェクトリファレンス取得
org.omg.CORBA.Object intf_obj = target._get_interface_def();
org.omg.CORBA.InterfaceDef intf = org.omg.CORBA.InterfaceDefHelper.narrow( intf_obj );
```

(2)org.omg.CORBA.OperationDefオブジェクトリファレンスの獲得

org.omg.CORBA.InterfaceDef.lookup_name()メソッドにより、インタフェースリポジトリから指定されたメソッドを検索します。パラメタとしてサーバアプリケーションのメソッド名を指定します。結果として、指定したメソッド情報が格納されたorg.omg.CORBA.OperationDefオブジェクトのオブジェクトリファレンスが通知されます。

```
//メソッドのorg.omg.CORBA.OperationDefオブジェクトリファレンス取得
String name = new String( "calcurate" );
org.omg.CORBA.Contained[] intf_opr =
intf.lookup_name( name, -1, org.omg.CORBA.DefinitionKind.dk_Operation, false );
```

(3)パラメタ情報の獲得

org.omg.CORBA.Contained.describe()メソッドにより、サーバアプリケーションがもっているメソッドのパラメタの情報(パラメタの名前、個数、パラメタの型など)をインタフェースリポジトリから検索します。パラメタとして、org.omg.CORBA.OperationDefオブジェクトリファレンスを指定します。

```
org.omg.CORBA.ContainedPackage.Description desc = intf_opr[0].describe();
//指定されたメソッドのパラメタ情報を検索
```

8.4.5 パラメタの組立て

(1)パラメタリストの生成

org.omg.CORBA.ORB.create_list()メソッドにより、サーバアプリケーションに渡すパラメタを格納するための領域を保持するリストオブジェクトを生成します。このとき、何個のパラメタを格納するかを指定します。結果として、NVListオブジェクトリファレンスが通知されます。NVListオブジェクトインタフェースの詳細については、“12.2 NVListオブジェクト”を参照してください。

```
org.omg.CORBA.NVList Nvl;  //NVList
//org.omg.CORBA.ContainedPackage.Description構造体からパラメタ情報構造体の抽出
org.omg.CORBA.Any opeany = desc.value;
org.omg.CORBA.OperationDescription c
                  = org.omg.OperationDescriptionHelper.extract( opeany );
org.omg.CORBA.ParameterDescription[] paraseq = c.parameters;
//パラメタ情報の抽出
int ParamCount = paraseq.length;
//リストオブジェクトの生成
Nvl = Orb.create_list( ParamCount );
```

(2)パラメタリストの設定

org.omg.CORBA.NVList.add_value()メソッドにより、サーバアプリケーションに渡すパラメタをリストオブジェクトに設定します。パラメタとして、サーバアプリケーションのパラメタの名前、値(Any型)、およびパラメタの形式を設定します。

```
org.omg.CORBA.NamedValue tmpNam;
org.omg.CORBA.Any  Anya = Orb.create_any();
Anya.insert_long(a);
tmpNam = Nvl.add_value( paraseq[0].name, Anya, org.omg.CORBA.ARG_IN.value );
org.omg.CORBA.Any  Anyb = Orb.create_any();
Anyb.insert_long(b);
tmpNam =
    Nvl.value.add_value( paraseq[1].name, Anyb, org.omg.CORBA.ARG_IN.value );
```

8.4.6 リクエストの作成

org.omg.CORBA.Object._create_request()メソッドにより、リクエストオブジェクトを作成します。リクエストオブジェクトに対して、サーバオブジェクトのオブジェクトリファレンス、org.omg.CORBA.NVListオブジェクトリファレンス、後述するorg.omg.CORBA.NamedValueと呼ぶサーバの処理結果を格納する領域を指定します。結果として、リクエストオブジェクトが返されます。

```
//復帰パラメタ
org.omg.CORBA.Any AnyResult = Orb.create_any();
AnyResult.type(c.result);
//復帰パラメタ格納域
org.omg.CORBA.NamedValue Result =
       Orb.create_named_value( null, AnyResult, org.omg.CORBA.ARG_OUT.value);
String OpName = new String(c.name);

//リクエストオブジェクトの生成
org.omg.CORBA.Request Req = target._create_request(
    null,       //context
    OpName,    //メソッド名
    Nvl,       //入力パラメタ
    Result);  //復帰値
```

8.4.7 リクエストの送信

サーバアプリケーションに対してリクエストを出します。リクエストの方法として、以下の2つの方法があります。

- 同期送信
- 非同期送信

(1)同期送信

org.omg.CORBA.Request.invoke()メソッドにより、サーバアプリケーションを呼び出します。

```
Req.invoke();
```

(2)非同期送信

org.omg.CORBA.Request.send_deferred()メソッドにより、サーバアプリケーションを呼び出します。サーバアプリケーションの処理結果は、org.omg.CORBA.Request.get_response()メソッドにより受け取ります。

```
Req.send_deferred();    //処理の要求
Req.get_response();     //処理結果の受取り
```

org.omg.CORBA.Request.get_response()の結果により、サーバアプリケーションからリクエストが完了していないことが判明した場合、再度org.omg.CORBA.Request.get_response()メソッドを呼び出します。

8.5 COBOL言語の開発

動的起動インタフェースを使用する場合の、クライアントアプリケーションの流れを以下に示します。

8.5.1 初期化

CORBAの初期化メソッドCORBA-ORB-INITを呼び出して、初期化処理を行います。当メソッドの結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
 IDENTIFICATION DIVISION.
    PROGRAM-ID. "CLIENT-MAIN".
    AUTHOR. OD/IDLCOMPILER VER.2.0.
    INSTALLATION. IDL FILE NAME IS COBSAMPLE.IDL.
    SECURITY. THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
              WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
              GURANTEED IS NOT DONE.
    DATE-WRITTEN. TUE MAY  6 11:03:40 1997
 *
    ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
    SPECIAL-NAMES.
        ARGUMENT-NUMBER   IS ARG-C
        ARGUMENT-VALUE    IS ARG-V
        SYMBOLIC CONSTANT
        COPY SYMBOL-CONST IN        CORBA.
        .
 *
    DATA DIVISION.
 *
    WORKING-STORAGE SECTION.
    COPY CONST IN CORBA.
    01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
    01 CURRENT-ARG-V.
```

```
       02 FILLER OCCURS 6.
          03 CURRENT-ARG-V-VALUE USAGE POINTER.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
    01 APLI-NAME PIC X(8) VALUE "simple_c".
    01 TMP-STRING-BUF PIC X(20).
    01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
    01 TEMP-BUF USAGE POINTER.
    01 MESS PIC X(30).
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
    01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
    01 COPY BOA IN CORBA REPLACING CORBA-BOA BY BOA.
*
    PROCEDURE DIVISION.
    MAIN.
  * ObjectDirectorのObjectIDを設定
        DISPLAY "CLIENT START!!".
  * argument set : CURRENT-ARG-V-VALUE
        ACCEPT CURRENT-ARG-C FROM ARG-C.
        COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
        PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
           IF ARG-COUNT = 1
              MOVE APLI-NAME TO TMP-STRING-BUF
           ELSE
              ACCEPT TMP-STRING-BUF FROM ARG-V
           END-IF
           MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
           CALL "CORBA-STRING-SET" USING
                 CURRENT-ARG-V-VALUE (ARG-COUNT)
                 STRING-LENGTH
                 TMP-STRING-BUF
        END-PERFORM.
        SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.
  * ObjectDirector初期化
        MOVE  12 TO STRING-LENGTH.
        CALL  "CORBA-STRING-SET" USING
              TEMP-BUF
              STRING-LENGTH
              FJ-OM-ORB-ID.
        CALL "CORBA-ORB-INIT" USING
              CURRENT-ARG-C
              CURRENT-ARG-V
              TEMP-BUF
              ENV
              ORB.
  * ORBIDに格納した領域を解放する
        CALL "CORBA-FREE" USING TEMP-BUF.
        MOVE "CORBA-ORB-INIT" TO MESS.
        PERFORM ENV-CHECK
```

例外処理については、“6.6 クライアントアプリケーションの例外処理”を参照してください。

8.5.2 ネーミングサービスのオブジェクトリファレンスの獲得

実行するオブジェクトをネーミングサービスから検索するために、ネーミングサービスのオブジェクトリファレンスが必要となります。CORBAインタフェースのオブジェクトリファレンスの取出しメソッドCORBA-ORB-RESOLVE-INITIAL-REFERENCESによりネーミングサービスのオブジェクトリファレンスを取り出します。このとき、CORBA-ORB-OBJECTID-NAMESERVICEをメソッドのパラメタとして指定します。

```
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING
           COSNAMING-NAMINGCONTEXT BY COS-NAMING.

   PROCEDURE DIVISION.
```

```
* NamingServiceのObjectIDの設定
      MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
            TEMP-BUF
            STRING-LENGTH
            CORBA-ORB-OBJECTID-NAMESERVICE.
 * NamingServiceのオブジェクトリファレンス獲得
      CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
            ORB
            TEMP-BUF
            ENV
            COS-NAMING.
 * NAMESERVICEIDに格納した領域を解放します
      CALL "CORBA-FREE" USING TEMP-BUF.
      MOVE "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" TO MESS.
      PERFORM ENV-CHECK.
```

8.5.3 インタフェースリポジトリからサーバアプリケーション情報の獲得

インタフェースリポジトリには、ドメイン全体のサーバアプリケーションの情報が格納されています。また、IDLで定義されたモジュール名、インタフェース名、メソッド名、およびパラメタが階層構造で格納されます。
サーバアプリケーションの情報を獲得するために、以下のような(1)から(3)の手順で検索します。

(1)InterfaceDefオブジェクトリファレンスの獲得

インタフェースリポジトリからサーバアプリケーションの情報を獲得するためには、INTERFACEDEFオブジェクトのオブジェクトリファレンスが必要となります。このため、サーバアプリケーションのオブジェクト名をパラメタとして、ネーミングサービスからサーバアプリケーションのオブジェクトリファレンスを求め、その後、CORBA-OBJECT-GET-INTERFACEメソッドによりInterfaceDefオブジェクトリファレンスを獲得します。

```
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 STR-BUF PIC X(30).
   01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING
         COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
   01 NAME-COMPONENT-A USAGE POINTER.
   01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
   01 NAME-A USAGE POINTER.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY INTF-INTF.

  PROCEDURE DIVISION.
 * オブジェクト名
      MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
      MOVE "ODdemo::calculator" TO STR-BUF.
      CALL "CORBA-STRING-SET" USING
            IDL-ID OF NAME-COMPONENT
            STRING-LENGTH
            STR-BUF.
 * オブジェクトのタイプ
      MOVE " " TO STR-BUF.
      CALL "CORBA-STRING-SET" USING
            KIND OF NAME-COMPONENT
            STRING-LENGTH
            STR-BUF.
 * オブジェクト名の数
      MOVE 1 TO SEQ-LENGTH OF NAME.
      MOVE 1 TO SEQ-MAXIMUM OF NAME.
 * Sequenceの要素数
      MOVE 1 TO NUM.
 *  CosNamingのアクセス情報格納用のSequence領域獲得
```

```
     CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
           SEQ-MAXIMUM OF NAME
           SEQ-BUFFER OF NAME.
     MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
     MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
* CosNamingの要素設定
     CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
           NAME-A
           NUM
           NAME-COMPONENT-A.
* サーバアプリケーションのオブジェクトリファレンスを獲得
     CALL "COSNAMING-NAMINGCONTEXT-RESOLVE" USING
              COS-NAMING
              NAME
              ENV
              OBJ.
     MOVE "COSNAMING-NAMINGCONTEXT-RESOLVE" TO MESS.
     PERFORM ENV-CHECK.
* サーバアプリケーションのInterfaceDefを獲得 *
   CALL "CORBA-OBJECT-GET-INTERFACE" USING
         OBJ
         ENV
         INTF-INTF.
```

(2)OperationDef オブジェクトリファレンスの獲得

CORBA-INTERFACEDEF-LOOKUP-NAMEメソッドにより、サーバアプリケーションが複数もっているメソッドから指定されたメソッドをインタフェースリポジトリから検索します。パラメタとしてサーバアプリケーションのメソッド名を指定します。結果として、指定したメソッド情報が格納されているOPERATIONDEFオブジェクトのオブジェクトリファレンスが返されます。

```
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01 BUF USAGE POINTER.
    01 COPY DEFINITIONKIND IN CORBA REPLACING CORBA-DEFINITIONKIND BY DK-FLAG.
    01 COPY BOOLEAN IN CORBA REPLACING CORBA-BOOLEAN BY BL.
    01 INTF-OPR USAGE POINTER.

   PROCEDURE DIVISION.
* メソッド名の入力
    DISPLAY " Please input method name==>".
    ACCEPT BUF.
* オブジェクトリファレンスの取得 : Operation情報の取得
    SET CORBA-DK-OPERATION OF DK-FLAG TO TRUE.
    MOVE -1 TO NUM.
    MOVE CORBA-FALSE TO BL.
    CALL "CORBA-INTERFACEDEF-LOOKUP-NAME" USING
         INTF-INTF
         BUF
         NUM
         DK-FLAG
         BL
         ENV
         INTF-OPR.
```

(3)パラメタ情報の獲得

CORBA-OPERATIONDEF-DESCRIBEメソッドにより、サーバアプリケーションがもっているメソッドのパラメタの情報(パラメタの名前、個数、パラメタの型など)をインタフェースリポジトリから検索します。パラメタとして、OPERATIONDEFオブジェクトリファレンスを指定します。

```
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 WORK-POINTER USAGE POINTER.
   01 DESCRIPTION USAGE POINTER.

   LINKAGE SECTION.
   01 COPY CONTAINED IN CORBA REPLACING CORBA-CONTAINED BY TMP-CONT.

  PROCEDURE DIVISION.
* パラメタ情報の構造体の取得
      MOVE 1 TO NUM.
      CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
            INTF-OPR
            NUM
            WORK-POINTER.
      SET ADDRESS OF TMP-CONT TO WORK-POINTER.
      CALL "CORBA-OPERATIONDEF-DESCRIBE" USING
            TMP-CONT
            ENV
            DESCRIPTION.
```

8.5.4 パラメタの組み立て

(1)パラメタリストの生成

CORBA-ORB-CREATE-LISTメソッドにより、サーバアプリケーションに渡すパラメタを格納するための領域を保持するリストオブジェクトを生成します。このとき、パラメタとして、何個のパラメタを格納するかを指定します。その結果、NVListオブジェクトリファレンスが返されます。NVListオブジェクトインタフェースの詳細については、“12.2 NVListオブジェクト”を参照してください。

```
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 COPY ANY IN CORBA REPLACING CORBA-ANY BY PARAM-ANY.
   01 COPY PARDESCRIPTIONSEQ IN CORBA REPLACING
          CORBA-PARDESCRIPTIONSEQ BY PARAMS.
   01 COPY NVLIST IN CORBA REPLACING CORBA-NVLIST BY ARG-LIST.

   LINKAGE SECTION.
   01 COPY CONTAINED-DESCRIPTION IN CORBA REPLACING
          CORBA-CONTAINED-DESCRIPTION BY TMP-DESC.
   01 COPY OPERATIONDESCRIPTION IN CORBA REPLACING
          CORBA-OPERATIONDESCRIPTION BY TMP-OPR-DESC.

  PROCEDURE DIVISION.
* Contained-Description構造体からパラメタ構造体の抽出
   SET ADDRESS OF TMP-DESC TO DESCRIPTION.
   MOVE IDL-VALUE OF TMP-DESC TO PARAM-ANY.
* パラメタ情報の抽出
   SET ADDRESS OF TMP-OPR-DESC TO ANY-VALUE OF PARAM-ANY.
   MOVE PARAMETERS OF TMP-OPR-DESC TO PARAMS.
   MOVE LENGTH OF PARAMS TO NUM.

* リストオブジェクトの生成
   CALL "CORBA-ORB-CREATE-LIST" USING
        ORB
        NUM
        ARG-LIST
        ENV
        RET-VAL.
```

(2)パラメタリストの設定

CORBA-NVLIST-ADD-ITEMメソッドにより、サーバアプリケーションに渡すパラメタをリストオブジェクトに設定します。パラメタとして、CORBA-NVLISTオブジェクトリファレンス、サーバアプリケーションのパラメタの名前、型、値、および長さを設定します。

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 01 STRING-POINTER USAGE POINTER.
 01 COPY TYPECODE IN CORBA REPLACING CORBA-TYPECODE BY TMP-TYPE.
 01 01 PARAM-POINTER USAGE POINTER.
 01 COPY FLAGS IN CORBA REPLACING CORBA-FLAGS BY FLAG.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY RET-VAL.

PROCEDURE DIVISION.
  MOVE FUNCTION ADDR(PARAMS) TO SEQ-POINTER.
  MOVE "in_p" TO STR-BUF.
  MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
  CALL "CORBA-STRING-SET" USING
             STRING-POINTER
             STRING-LENGTH
             STR-BUF.
  MOVE FUNCTION LENG (TC-LONG) TO STRING-LENGTH.
  CALL "CORBA-STRING-SET" USING
             TEMP-BUF
             STRING-LENGTH
             TC-LONG.
  CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
             TEMP-BUF
             TMP-TYPE.
  CALL "CORBA-FREE" USING
             TEMP-BUF.
  MOVE 1234 TO IN-P.
  MOVE FUNCTION ADDR( IN-P ) TO PARAM-POINTER.
  MOVE LSIZE OF TMPDATA TO NUM.
  SET CORBA-ARG-IN OF FLAG TO TRUE.
  CALL "CORBA-NVLIST-ADD-ITEM" USING
             ARG-LIST
             STRING-POINTER
             TMP-TYPE
             PARAM-POINTER
             NUM
             FLAG
             ENV
             RET-VAL.
```

8.5.5 リクエストの作成

CORBA-OBJECT-CREATE-REQUESTメソッドにより、リクエストオブジェクトを作成します。リクエストオブジェクトに対して、サーバオブジェクトのオブジェクトリファレンス、NVListオブジェクトリファレンス、後述するNamedValueと呼ぶサーバの処理結果を格納する領域を指定します。この結果として、リクエストオブジェクトのオブジェクトリファレンスが返されます。

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 COPY CONTEXT IN CORBA REPLACING CORBA-CONTEXT BY CTX.
 01 COPY REQUEST IN CORBA REPLACING CORBA-REQUEST BY REQ.
 01 COPY NAMEDVALUE IN CORBA REPLACING CORBA-NAMEDVALUE BY RESULT.

PROCEDURE DIVISION.
  MOVE FUNCTION ADDR(RESULT) TO PARAM-POINTER.
  MOVE "calculate" TO STR-BUF.
  MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
  CALL "CORBA-STRING-SET" USING
          STRING-POINTER
```

```
            STRING-LENGTH
      STR-BUF.

      SET CORBA-OUT-LIST-MEMORY OF FLAG TO TRUE.

      CALL "CORBA-OBJECT-CREATE-REQUEST" USING
            OBJ
            CTX
            STRING-POINTER
            ARG-LIST
            PARAM-POINTER
            REQ
            FLAG
            ENV
            RET-VAL.
```

NamedValueは、サーバの処理結果を格納するための領域であり、以下のように定義されています。
IDLファイルでの定義を以下に示します。

```
module CORBA   {
        typedef string                  Identifier;
        enum Flags{
                ARG_IN,
                ARG_OUT,
                ARG_INOUT,
                OUT_LIST_MEMORY,
                IN_COPY_VALUE,
                INV_NO_RESPONSE,
                INV_TERM_ON_ERR,
                RESP_NO_WAIT,
                DEPENDENT_LIST,
                CTX_RESTRICT_SCOPE,
                CTX_DELETE_DESCENDENTS
        };
        struct NamedValue       {
            Identifier    name;       // パラメタの名前
            any           argument;   // パラメタの値
            long          len;        // パラメタの長さ
            Flags         arg_modes;  // パラメタの引き渡し方法(in,out,inout)
        };
};
```

COBOL登録集を以下に示します。

```
 * NamedValue構造体
      CORBA-NAMED-VALUE.
 * パラメタの名前
      07  NAME     USAGE   POINTER.
 * パラメタの値
      07  ARGUMENT.
          09 ANY-TYPE      USAGE POINTER.
          09 ANY-VALUE     USAGE POINTER.
 * パラメタの長さ
      07  LEN      PIC     S9(9) COMP-5.
 * パラメタの引き渡し方法(in,out,inout)
      07  ARG-MODES        PIC     9(9) COMP-5.
          88  CORBA-ARG-IN                VALUE   1.
```

```
        88  CORBA-ARG-OUT                VALUE   2.
        88  CORBA-ARG-INOUT              VALUE   4.
        88  CORBA-OUT-LIST-MEMORY        VALUE   8.
        88  CORBA-IN-COPY-VALUE          VALUE   16.
        88  CORBA-INV-NO-RESPONSE        VALUE   32.
        88  CORBA-INV-TERM-ON-ERR        VALUE   64.
        88  CORBA-RESP-NO-WAIT           VALUE   128.
        88  CORBA-DEPENDENT-LIST         VALUE   256.
        88  CORBA-CTX-RESTRICT-SCOPE     VALUE   512.
        88  CORBA-CTX-DELETE-DESCENDENTS VALUE   1024.
```

8.5.6 リクエストの送信

サーバアプリケーションに対してリクエストを出します。リクエストの方法として、以下の2つの方法があります。

- 同期送信
- 非同期送信

(1)同期送信

CORBA-REQUEST-INVOKEメソッドにより、サーバアプリケーションを呼び出します。復帰値としてサーバアプリケーションの処理結果が返ります。パラメタとして、リクエストオブジェクトのオブジェクトリファレンスを指定します。

```
   MOVE 0 TO  FLAG.
   CALL "CORBA-REQUEST-INVOKE" USING
        REQ
        FLAG
        ENV
        RET-VAL.
```

(2)非同期送信

CORBA-REQUEST-SENDメソッドにより、サーバアプリケーションを呼び出します。パラメタとして、リクエストオブジェクトのオブジェクトリファレンスを指定します。サーバアプリケーションの処理結果は、CORBA-REQUEST-GET-RESPONSEメソッドにより受け取ります。パラメタとして、リクエストオブジェクトのオブジェクトリファレンスを指定します。

```
* 処理の要求
    SET CORBA-INV-NO-RESPONSE OF FLAG TO TRUE.
    CALL "CORBA-REQUEST-SEND" USING
         REQ
         FLAG
         ENV
         RET-VAL.
* 処理結果の受け取り
    MOVE CORBA-RESP-NO-WAIT TO  FLAG.
    CALL "CORBA-REQUEST-GET-RESPONSE" USING
         REQ
         FLAG
         ENV
         RET-VAL.
    IF RET-VAL = CORBA-TRUE
* ユーザの処理
    END-IF
```

CORBA-REQUEST-GET-RESPONSEでサーバアプリケーションからリクエストが完了していないことが判明した場合は、再度CORBA-REQUEST-GET-RESPONSEメソッドを呼び出します。

8.5.7 リクエストの削除

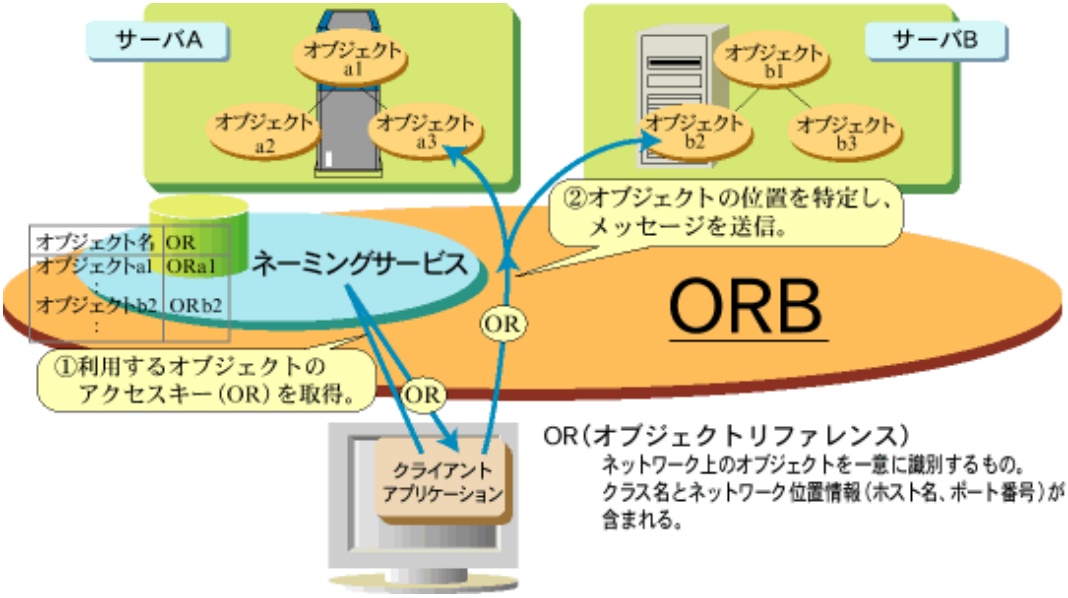
CORBA-REQUEST-DELETEメソッドにより、リクエストオブジェクトを削除します。パラメタとして、リクエストオブジェクトのオブジェクトリファレンスを指定します。

```
*リクエストオブジェクトの削除
    CALL "CORBA-REQUEST-DELETE" USING
         REQ
         ENV
         RET-VAL.
```

第9章 ネーミングサービスのプログラミング

9.1 ネーミングサービスの概要

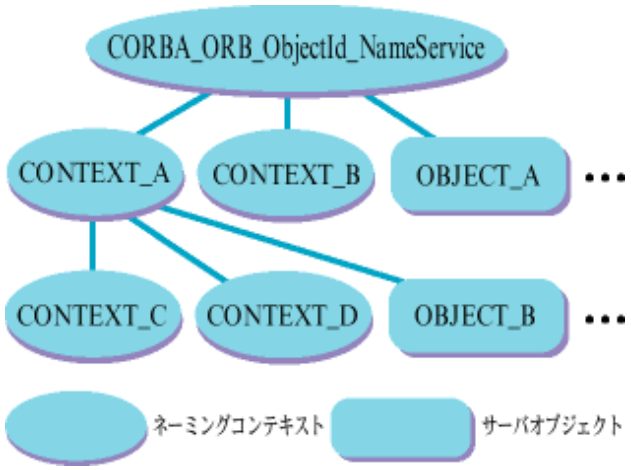
ネーミングサービスは、オブジェクトリファレンスをバインディング名と呼ばれる論理的な名前と関連付けて管理するサービスです。クライアントアプリケーションは、サーバアプリケーションのオブジェクトリファレンスの内容をプログラム中に抱える必要がなく、論理的な名前を使って最新のサーバアプリケーションのオブジェクトリファレンスを獲得することが可能となります。



ネーミングサービスは、以下の3種類のオブジェクトリファレンスを管理しています。

- サーバアプリケーションなどを示す一般的なオブジェクトリファレンス
- ネーミングコンテキスト
  オブジェクトリファレンスや、他のネーミングコンテキストを管理するための入れ物で、ファイルシステムのディレクトリ、フォルダに相当
- Windows32 Solaris32 Linux32
  ロードバランスオブジェクトグループ
  ロードバランスオブジェクトグループについては、“付録F 旧バージョンからの移行上の注意”を参照してください。

ネーミングサービスの例を以下に示します。



ネーミングサービスは、以下のインタフェースを提供しています。

```
module CosNaming {
   typedef string    Istring;
   struct  NameComponent{
               Istring  id;
               Istring  kind;
   };

   typedef    sequence <NameComponent> Name;

   enum    BindingType{
       nobject,
       ncontext
   };

   struct Binding{
       Name         binding_name;
       BindingType  binding_type;
   };

   typedef    sequence <Binding> BindingList;

   interface BindingIterator;

   interface NamingContext{

       enum NotFoundReason {
           missing_node, not_context,
           not_object
       };

       exception NotFound{
           NotFoundReason  why;
           Name            rest_of_name;
       };

       exception CannotProceed{
           NamingContext    cxt;
           Name             rest_of_name;
       };
       exception InvalidName{};
       exception AlreadyBound{};
       exception NotEmpty{};
```

```
    void bind( in Name   n,
               in Object obj )
         raises( NotFound, CannotProceed, InvalidName,
                 AlreadyBound );

    void rebind( in Name   n,
                 in Object obj )
         raises( NotFound, CannotProceed, InvalidName );

    void bind_context(
                 in Name          n,
                 in NamingContext nc )
         raises( NotFound, CannotProceed, InvalidName,
                 AlreadyBound );

    void rebind_context(
                 in Name          n,
                 in NamingContext nc )
         raises( NotFound, CannotProceed, InvalidName );

    Object resolve(
                 in Name   n )
           raises( NotFound, CannotProceed, InvalidName );

    void unbind( in Name   n )
         raises( NotFound, CannotProceed, InvalidName );

    NamingContext new_context();

    NamingContext bind_new_context(
                          in Name   n )
                  raises( NotFound, CannotProceed, InvalidName,
                          AlreadyBound );

    void destroy()
         raises( NotEmpty );

    void list(  in   unsigned long   how_many,
                out  BindingList     bl,
                out  BindingIterator bi );
};

interface BindingIterator{
    boolean next_one(
        out Binding b );

    boolean next_n( in   unsigned long  how_many,
                    out  BindingList    bl );

    void destroy();
};

interface NamingContextExt:NamingContext{

    typedef string  StringName;
    typedef string  Address;
    typedef string  URLString;

    StringName    to_string(
                  in  Name  n )

            raises( InvalidName );
```

```
        Name    to_name(
                        in  StringName  sn )

                raises( InvalidName );

        exception InvalidAddress{};

        URLString    to_url(
                        in  Address     addr,
                        in  StringName  sn )

                raises( InvalidAddress, InvalidName );

        Object  resolve_str(
                        in  StringName  sn )
                raises( NotFound, CannotProceed, InvalidName );

    };
};
```

ネーミングサービスで提供しているインタフェースの機能説明を以下に示します。

インタフェース名	メソッド名	機能説明
NamingContext	bind	オブジェクトリファレンスを、指定されたバインディング名でネーミングサービスに登録します。この関連付けをバインディングと呼びます。
	rebind	オブジェクトリファレンスを、指定されたバインディング名でネーミングサービスに登録します。 すでに同じ名前のバインディングが存在する場合は、古いバインディングは削除され、新しいバインディングが登録されます。
	bind_context	ネーミングコンテキストのオブジェクトリファレンスを、指定されたバインディング名でネーミングサービスに登録します。
	rebind_context	ネーミングコンテキストのオブジェクトリファレンスを、指定されたバインディング名でネーミングサービスに登録します。 すでに同じ名前のバインディングが存在する場合は、古いバインディングは削除され、新しいバインディングが登録されます。
	resolve	指定されたバインディング名に関係付けられたオブジェクトリファレンスを獲得します。 バインディング名として構造体型のバインディング名を指定します。
	unbind	指定されたバインディング名のバインディングをネーミングサービスから削除します。
	new_context	新しいネーミングコンテキストを作成します。作成されたネーミングコンテキストのオブジェクトリファレンスはどのネーミングコンテキストにも登録されていません。
	bind_new_context	新規にネーミングコンテキストを作成し、このネーミングコンテキストのオブジェクトリファレンスを指定されたバインディング名でネーミングサービスに登録します。(new_context+bind_contextと同様)
	destroy	ネーミングコンテキストを削除します。
	list	ネーミングコンテキストに登録されているバインディングの一覧を獲得します。
BindingIterator	next_one	ネーミングコンテキスト内の次のバインディングを獲得します。
	next_n	ネーミングコンテキスト内の次のn個のバインディングを獲得します。
	destroy	BindingIteratorオブジェクトを破壊します。
NamingContextExt	to_string	構造体型のバインディング名を文字列表記のバインディング名に変換します。

インタフェース名	メソッド名	機能説明
	to_name	文字列表記のバインディング名から構造体型のバインディング名に変換します。
	to_url	与えられたアドレスと、文字列表記のバインディング名からURLアドレスを作成します。
	resolve_str	指定されたバインディング名に関係付けられたオブジェクトリファレンスを獲得します。 バインディング名として文字列表記のバインディング名が指定されます。

9.2 ネーミングコンテキストインタフェース

ネーミングコンテキストインタフェースが扱うデータ型とインタフェースの詳細について説明します。

9.2.1 ネーミングコンテキストインタフェースで扱うデータ型

(1) ネームコンポーネント

ネームコンポーネントのIDLを以下に示します。

```
struct NameComponent {
    Istring id;
    Istring kind;
};
```

ネームコンポーネントは、ネーミングコンテキストにオブジェクトリファレンスを登録する際に、オブジェクトリファレンスと関連付けされてネーミングコンテキストに登録されます。

ネームコンポーネントは、idフィールドとkindフィールドで構成されます。それぞれのフィールドは文字列で表され、長さが0("")も許されます。idフィールドとkindフィールドの使い分けに関する規則は、特にありません。利用者が自由に使い分けることができます。

2つのネームコンポーネントが同一であるためには、idとkindの両方のフィールドが一致している必要があります。

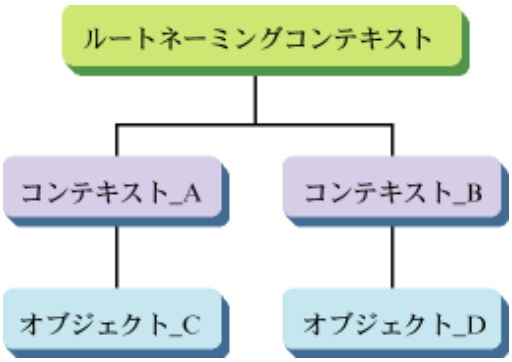
(2) 構造体型のバインディング名

構造体型のバインディング名のIDLを以下に示します。

```
typedef sequence <NameComponent> Name;
```

構造体型のバインディング名は、ネームコンポーネントのシーケンス型です。
階層構造の複数のネーミングコンテキストにまたがったオペレーションをネーミングサービスに要求するために、ネーミングサービスのオペレーションの呼出しには、バインディング名を指定して行います。

1つだけのネームコンポーネントで構成されているバインディング名を単純名、複数のネームコンポーネントで構成されているバインディング名をコンパウンド名と呼びます。



上記の例で、ルートネーミングコンテキストに対して要求を行う際に、オブジェクト_Cを示す構造体型バインディング名は以下のようになります。

index	id	kind	備考
1	"コンテキスト_A"	""	ネームコンポーネント
2	"オブジェクト_C"	""	ネームコンポーネント

(3) バインディング、バインディングリスト

ネーミングコンテキストに登録されるバインディングは、コンテキストタイプ／オブジェクトタイプのバインディングタイプを持ちます。コンパウンド名で指定される最後のネームコンポーネント以外のネームコンポーネントは、コンテキストタイプのバインディングである必要があります。

bind、rebindで作成されたバインディングのバインディングタイプは、オブジェクトタイプです。bind_context、rebind_context、bind_new_contextで作成されたバインディングのバインディングタイプは、コンテキストタイプとなります。

バインディング、バインディングリストのIDLを以下に示します。

```
enum BindingType {nobject, ncontext};

struct Binding {
    Name binding_name;
    BindingType binding_type;
};

typedef sequence <Binding> BindingList;
```

バインディングとバインディングリストは、ネーミングコンテキストに登録されているバインディングの一覧を獲得する場合に返されるバインディング情報を格納するデータ型です。

バインディングはバインディング名とバインディングタイプから構成され、1つのバインディング情報が格納されます。バインディングは、BindingIterator::next_oneオペレーションから返されます。

バインディングリストは、複数のバインディング情報を格納するためのデータ型で、バインディングのシーケンス型です。バインディングリストは、NamingContext::list、およびBindingIterator::next_nオペレーションから返されます。

9.2.2 ネーミングコンテキストインタフェースの呼び出しで発生する例外

ネーミングサービスは、標準的なシステム例外の他に、以下に示すユーザ例外を発生させます。

(1) NotFound

```
enum NotFoundReason{
    missing_node, not_context, not_object
```

```
};

exception NotFound {
    NotFoundReason why;
    Name rest_of_name;
};
```

この例外は、バインディング名で指定された各ネームコンポーネントに相当するバインディングが存在しない場合や、バインディングは存在するが要求されたバインディングのタイプと異なる場合に発生します。

whyには、例外の発生原因により以下の値が設定されます。

missing_node

要求されたバインディングがネーミングコンテキストに登録されていません。

not_context

コンテキストタイプのバインディングが期待される処理で、オブジェクトタイプのバインディングが登録されています。

not_object

オブジェクトタイプのバインディングが期待される処理で、コンテキストタイプのバインディングが登録されています。

rest_of_nameメンバには、処理が行われなかった残りのバインディング名が設定されます。rest_of_nameの最初のネームコンポーネントが、問題があったネームコンポーネントとなります。

(2) CannotProceed

```
exception CannotProceed {
    NamingContext cxt;
    Name rest_of_name;
};
```

この例外は、ネーミングサービスが何らかの異常により、処理を途中で終了した場合に発生します。

cxtメンバには、すでに処理が正常に行われたネーミングコンテキストが設定されます。
rest_of_nameメンバには、処理されなかった残りのバインディング名が設定されます。
エラーの内容によっては、返されたctxとrest_of_nameを使用して再度処理を行うことができます。

(3) InvalidName

```
exception InvalidName {};
```

この例外は、正しくないバインディング名が指定された場合に発生します。

ネームコンポーネントを1つも含まない、長さ0のバインディング名が指定された場合、本例外が発生します。

(4) AlreadyBound

```
exception AlreadyBound {};
```

オブジェクトが指定されたバインディング名で、すでにネーミングコンテキストに登録されていることを意味します。

(5) NotEmpty

```
exception NotEmpty {};
```

この例外は、削除しようとしたネーミングコンテキストに1つ以上のバインディングが存在する場合に発生します。ネーミングコンテキストを削除するためには、ネーミングコンテキストが空である必要があります。

9.2.3 バインディングの作成

バインディングオペレーションはオブジェクトをバインディング名と関連付け、ネーミングコンテキストに登録します。バインディング名に関連付けられたバインディングは、resolveオペレーションを使用して獲得することができます。ネーミングサービスは、バインディングを作成するbind、rebind、bind_context、rebind_context、bind_new_contextの５つのオペレーションをサポートします。このうち、bind_new_contextオペレーションについては、“9.2.6 ネーミングコンテキストの作成”を参照してください。

```
void bind(in Name n, in Object obj)
     raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
void rebind(in Name n, in Object obj)
     raises(NotFound, CannotProceed, InvalidName);
void bind_context(in Name n, in NamingContext nc)
     raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
void rebind_context(in Name n, in NamingContext nc)
     raises(NotFound, CannotProceed, InvalidName);
```

bind

ネーミングサービスに、指定されたバインディング名でオブジェクトリファレンスを登録します。登録されるバインディングは、オブジェクトタイプバインディングです。
指定された名前のバインディングが存在する場合は、AlreadyBound例外が発生します。

rebind

ネーミングサービスに、指定されたバインディング名でオブジェクトリファレンスを登録します。すでに同じ名前のバインディングが存在する場合は、古いバインディングは削除され、新しいバインディングが登録されます。登録されるバインディングは、オブジェクトタイプバインディングです。
すでに同一のバインディング名で登録されている場合、以前のバインディングはタイプがオブジェクトタイプである必要があります。それ以外の場合は、NotFound例外が発生し、whyにnot_objectが設定されます。

bind_context

ネーミングサービスに、指定されたバインディング名でネーミングコンテキストのオブジェクトリファレンスを登録します。登録されるバインディングは、コンテキストタイプのバインディングです。登録するネーミングコンテキストとしてnilリファレンスが指定された場合は、BAD_PARAMシステム例外が発生します。
指定された名前のバインディングが存在する場合は、AlreadyBound例外が発生します。

rebind_context

ネーミングサービスに、指定されたバインディング名でネーミングコンテキストのオブジェクトリファレンスを登録します。すでに同じ名前のバインディングが存在する場合、古いバインディングは削除され、新しいバインディングが登録されます。登録されるバインディングは、コンテキストタイプのバインディングです。

登録するネーミングコンテキストとしてnilリファレンスを指定された場合、BAD_PARAMシステム例外が発生します。
すでにバインディングが存在する場合、以前のバインディングのタイプはコンテキストタイプである必要があります。それ以外の場合、NotFound例外が発生し、whyにnot_contextが設定されます。

rebind_contextを使用する場合、以前に登録されていたネーミングコンテキストのバインディングが削除されます。したがって、このネーミングコンテキストが他のネーミングコンテキストに登録されていなければ、このネーミングコンテキストはどのネーミングコンテキストにも登録されていない状態となります。不要なネーミングコンテキストの場合は、rebind_contextを実行する前にdestroyオペレーションでネーミングコンテキストを削除してください。

9.2.4 バインディングの検索

resolveオペレーションは、与えられたネーミングコンテキスト中のバインディング名に関連付けられたオブジェクトリファレンスを獲得する処理です。与えられたバインディング名は、ネーミングコンテキストに登録されているバインディング名と確実に一致している必要があります。resolveオペレーションではオブジェクトリファレンスだけが返され、バインディングのタイプは返しません。
返されたオブジェクトリファレンスはCORBA::Object型のため、クライアントは、オブジェクトリファレンスをふさわしいタイプにnarrow、またはキャストして使用する必要があります。

```
Object resolve(in Name n )
    raises(NotFound, CannotProceed, InvalidName);
```

バインディング名は、複数のコンポーネントを持つことが可能です。詳細については、“9.2.1 ネーミングコンテキストインタフェースで扱うデータ型”の(2)を参照してください。バインディング名は、複数のネーミングコンテキストをまたがって処理されます。これらのネーミングコンテキストは、1つのネーミングサービスインスタンスに存在することが可能であり、また異なるネーミングサービスインスタンスの間で連携することも可能です。

9.2.5 バインディングの削除

unbindオペレーションは、ネーミングコンテキストから指定されたバインディングを削除します。

```
void unbind(in Name n)
     raises(NotFound, CannotProceed, InvalidName);
```

コンテキストタイプのバインディングを削除する場合、そのネーミングコンテキストが他のネーミングコンテキストに登録されていなければ、このネーミングコンテキストはどのネーミングコンテキストにも登録されていない状態となります。不要なネーミングコンテキストを削除する場合は、destroyオペレーションでネーミングコンテキストを削除後、unbindオペレーションによりバインディングを削除してください。

9.2.6 ネーミングコンテキストの作成

ネーミングサービスは、新しいコンテキストを作成するために、以下の2つのオペレーションをサポートします。

- new_context
- bind_new_context

```
NamingContext new_context();
NamingContext bind_new_context(in Name n)
    raises(NotFound, AlreadyBound, CannotProceed, InvalidName);
```

new_context

新しいネーミングコンテキストを作成します。作成されたネーミングコンテキストのオブジェクトリファレンスは、どのネーミングコンテキストにも登録されていません。
作成したネーミングコンテキストは、不要となった時点で、destroyオペレーションで削除する必要があります。

bind_new_context

新しいネーミングコンテキストを作成し、このネーミングコンテキストのオブジェクトリファレンスを指定されたバインディング名でネーミングサービスに登録します。
本オペレーションは、new_contextオペレーションとbind_contextオペレーションを組み合わせた処理と同等です。登録されるバインディングは、コンテキストタイプのバインディングです。

9.2.7 ネーミングコンテキストの削除

destroyオペレーションは、ネーミングコンテキストを削除します。

```
void destroy()
     raises(NotEmpty);
```

このオペレーションは、このネーミングコンテキストを削除します。削除されるコンテキストにバインディングが存在する場合、NotEmptyの例外が発生し、ネーミングコンテキストは削除されません。

destroyオペレーションを実行しても、このネーミングサービスが登録されているバインディングは、ネーミングコンテキストから削除されません。不要なネーミングコンテキストの場合は、unbindオペレーションによりバインディングも削除してください。

9.2.8 バインディングの一覧獲得

listオペレーションは、ネーミングコンテキストに登録されているバインディングの一覧を獲得します。

```
void list(in unsigned long how_many,
     out BindingList bl, BindingIterator Bi);
```

listは、ネーミングコンテキスト中に含まれていたバインディングをバインディングリストbl中に返します。バインディングリストblは、1つのネームコンポーネントを示す、長さ1のバインディング名を含むバインディングのシーケンスです。

ネーミングサービスは、クライアントから要求されたhow_manyパラメタの個数を上限としたバインディングを返します。how_manyで指定された値が、nsconfigファイルのbl_how_manyパラメタで設定されている最大バインディング数よりも大きい場合は、ネーミングサービスはbl_how_manyパラメタで指定された値のバインディングを返します。

how_manyに0が設定されている場合は、クライアントはバインディングにアクセスするためのbiと、listが長さ0のシーケンスのblを返します。

biパラメタがnilリファレンス以外を返した場合、listの呼出しによりコンテキスト中のすべてのバインディングが返されず、残りのバインディングがアイテレータを使用して回収される必要があることを示しています。biパラメタがnilリファレンスを返した場合は、blパラメタがネーミングコンテキスト中のすべてのバインディングを返したことを意味します。これは、how_manyにどのような値を指定した場合も同様です。

9.3 バインディングアイテレータインタフェース

バインディングアイテレータインタフェースは、listオペレーションで処理されなかったバインディングの一覧を獲得するためのインタフェースを提供します。

```
interface BindingIterator {
    boolean next_one(out Binding b);
    boolean next_n(in unsigned long how_many,
          out BindingList bl);
    void destroy();
}
```

next_one

next_oneオペレーションは、listや以前に呼び出されたnext_n、next_oneによりまだ返却されていない次のバインディングを返します。オペレーションがバインディングを返している場合は、復帰値としてtrueを返します。すべてのバインディングが処理された後のnext_oneオペレーションでは、falseを返します。next_oneがfalseを返した場合、バインディングの値は不定となるため、参照しないでください。
falseを返した後に、さらにnext_oneを呼び出した場合は、OBJECT_NOT_EXISTシステム例外が発生します。

next_n

next_nは、listや以前に呼び出されたnext_n, next_oneによりまだ返却されていないバインディングをパラメタbl中に返します。blが0でない長さのシーケンスの場合、trueが返ります。next_nは、すべてのバインディングが回収された時に一度だけ長さ0のパラメタblと復帰値としてfalseを返します。
falseを返した後に、さらにnext_oneを呼び出した場合、OBJECT_NOT_EXISTシステム例外が発生します。
ネーミングサービスは、クライアントから要求されたhow_manyパラメタの個数を上限としたバインディングを返します。how_manyで指定された値が、nsconfigファイルのbl_how_manyパラメタで設定されている最大バインディング数よりも大きい場合は、ネーミングサービスはbl_how_manyパラメタで指定された値のバインディングを返します。
how_manyに0が指定された場合は、BAD_PARAMシステム例外が発生します。

destroy

destroyオペレーションは、このバインディングアイテレータを破壊します。クライアントがdestroyを呼び出した後に何らかのオペレーションを呼び出した場合、オペレーションはOBJECT_NOT_EXISTシステム例外を返します。
listオペレーションによりバインディングアイテレータを作成し、destroyオペレーションを呼び出さないクライアントは、資源を継続して使用することになります。バインディングアイテレータが返されたクライアントは、責任をもってこのバインディングアイテレータをlistオペレーションにより削除する必要があります。

ネーミングサービスの標準規約では、destroyされないバインディングアイテレータのために資源が継続して使用されることを防ぐために、ネーミングサービスは警告なしで、いつでもアイテレータオブジェクトを破壊することが可能です。ポータビリティが高いクライアントアプリケーションを作成する場合は、バインディングアイテレータが破壊され、OBJECT_NOT_EXISTシステム例外が発生することを想定する必要があります。

9.4 文字列表記のバインディング名

構造体型のバインディング名は、ネームコンポーネントのシーケンスです。そのため、設定および参照ともアプリケーションの処理を複雑にしています。ネーミングコンテキスト拡張インタフェース(NamingContextExtインタフェース)では、文字列表記のバインディング名のシンタックスと、文字列表記のバインディング名と構造体型のバインディング名を相互に変換するオペレーションを提供します。詳細については、“9.6 バインディング名、URL、IOR間の変換”を参照してください。

文字列表記のバインディング名のシンタックスでは、文字'/', '.', '¥'は特殊文字として以下のように扱われます。詳細については、“9.4.2 文字列表記バインディング名のエスケープ機構”を参照してください。

- スラッシュ(/):ネームコンポーネントの区切り文字
- ドット(.):idとkindフィールドの区切り文字
- バックスラッシュ(¥):エスケープキャラクタ

9.4.1 文字列表記バインディング名の基本表記

文字列表記のバインディング名は、'/'文字で区切られたネームコンポーネントから構成されます。

例

“a”、“b”、“c”のネームコンポーネントで構成されるバインディング名は、“a/b/c”と表されます。

'.'文字はネームコンポーネント中のidとkindフィールドを区切るために使用します。

例

文字列表記のバインディング名“a.b/c.d/.”は、以下の構造体型のバインディング名を表します。

index	id	kind
0	a	b
1	c	d
2	<empty>	<empty>

ネームコンポーネント“.”は、空のidと空のkindフィールドのネームコンポーネントを表します。

文字列表記のバインディング名内のネームコンポーネントが“.”文字を含まない場合は、文字列全体がidフィールドとして処理され、kindフィールドは空とみなされます。

例

“a/./c.d/.e”と対応する構造体型バインディング名は、以下のように表されます。

index	id	kind
0	a	<empty>
1	<empty>	<empty>
2	c	d

index	id	kind
3	<empty>	e

ネームコンポーネントが空でないidフィールドと、空のkindフィールドで構成される場合、文字列表記はidフィールドだけである必要があります。後ろに付く'.'文字は、使用できません。

9.4.2 文字列表記バインディング名のエスケープ機構

バックスラッシュ(¥)は、文字列表記のバインディング名内の予約された意味を持つ'/', '.', '¥'をエスケープします。

ネームコンポーネント区切り文字

ネームコンポーネントにスラッシュ'/'を含める場合、バックスラッシュ(¥)をエスケープとして使用します。

例

文字列表記のバインディング名“a/x¥/y¥/z/b”は、“a”、“x/y/z”、“b”のネームコンポーネントで構成されていることを表します。

idとkindフィールド

バックスラッシュエスケープ機構は、idとkindフィールドがドット(.)を含めるようにするためにも使用されます。

例

文字列表記のバインディング名“a¥.b.c¥.d/e.f”は、以下の構造体型のバインディング名を表します。

index	id	kind
0	a.b	c.d
1	e	f

エスケープ文字

エスケープ文字'¥'をネームコンポーネント中に含めたい場合にも、バックスラッシュ(¥)をエスケープ文字として使用します。

例

文字列表記のバインディング名“a/b¥¥/c”は、コンポーネント“a”、“b¥”、“c”を含むバインディング名を表します。

9.5 URLスキーマ

イニシャルサービス／ネーミングコンテキストに登録されているオブジェクトリファレンスを表記するUniform Resource Locater(URL)スキーマについて説明します。

9.5.1 IOR URLスキーマ

IOR URLは、CORBA::Object型の文字列表記(IOR:<hex_octets>)です。このURLフォーマットは、ネーミングサービスとは独立しています。

9.5.2 corbaloc URLスキーマ

IOR形式のURLスキーマをそのまま認識することは、長さやテキストが符号化されているために困難です。corbaloc URLスキーマは、ftp／http URLのように、わかりやすいURLを提供します。このURLも、ネーミングサービスとは独立しています。

corbaloc URLの表記方法については、“13.4 corbaloc URLスキーマ”を参照してください。

9.5.3 corbaname URLスキーマ

corbaname URLは、ネーミングコンテキスト中のバインディング名を含むこと以外、corbaloc URLと同様です。

“corbaname::nshost/NameService#NC”は、nshost上のイニシャルサービス“NameService”で表されるネーミングコンテキストに、バインディング名“NC”で登録されているネーミングコンテキストのオブジェクトリファレンスを表します。

“corbaname::nshost”のようにオブジェクトキーが指定されていない場合は、オブジェクトキーとして“NameService”が指定されたものとみなされます。

“corbaname:rir:#NC”は、オーバライド設定がされない初期設定のネーミングコンテキストに、文字列表記のバインディング名“NC”で登録されているネーミングコンテキストのオブジェクトリファレンスを表します。

“corbaname:rir:”、“corbaname:rir:/NameService”は、“corbaloc:rir:”と同等で、ORBInitRef引数、およびORBDefaultInitRef引数によるイニシャルリファレンスのオーバライドが設定されていない初期設定のネーミングコンテキストのオブジェクトリファレンスを表します。ORBInitRef引数、およびORBDefaultInitRef引数の詳細については、“第13章 ネーミングサービスのイニシャルリファレンスの獲得”を参照してください。

corbanameのシンタックスは、以下のとおりです。

```
<corbaname>     = "corbaname:"<corbaloc_obj>["#"<string_name>]
<corbaloc_obj>  = <obj_addr_list> ["/"<key_string>]
<obj_addr_list> = “13.4 corbaloc URLスキーマ” 参照。
<key_string>    = “13.4 corbaloc URLスキーマ” 参照。
<string_name>   = stringfied Name | empty_string

corbaloc_obj  : corbaloc URLと同じシンタックスで表され、ネーミングコンテキストを指定します。
obj_addr_list : “13.4 corbaloc URLスキーマ” 参照。
key_string    : “13.4 corbaloc URLスキーマ” 参照。
string_name   : 以下に規定されているURLエスケープ付きの文字列表記のバインディング名です。
```

corbanameキャラクタエスケープ

US-ASCII英数字、および以下に示された文字を除く文字がエスケープされます。

```
";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" |
"," | "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
```

corbanameエスケープメカニズム

パーセント(%)がエスケープとして使われます。ネームコンポーネント中のエスケープが必要である文字は、パーセント(%)に続く2桁の16進数で表されます。最初の16進文字は上位のハーフバイトを表し、2つ目の16進文字は下位のハーフバイトを表します。パーセント(%)が2桁の16進数を後に持たない場合、文字列表記のバインディング名の構文は不当になります。

例

エスケープメカニズムの例を以下に示します。

文字列化された名前	URLエスケープ後	備考
a.b/c.d	a.b/c.d	同一のURL形式となる
<a>.b/c.d	%3ca%3e.b/c.d	“<”と“>”がエスケープされる
a.b/ c.d	a.b/%20%20c.d	2つの空白がエスケープされる
a%b/c%d	a%25b/c%25d	2つの%がエスケープされる
a¥¥b/c.d	a%5c%5cb/c.d	“¥”がエスケープされる、文字列化された名前ですでにエスケープされる

9.6 バインディング名、URL、IOR間の変換

NamingContextインタフェースを引き継いだNamingContextExtインタフェースは、URLと文字列表記のバインディング名を使用するためのオペレーションを提供します。

to_string

構造体型のバインディング名を文字列表記のバインディング名に変換します。
指定された構造体型のバインディング名が不当である場合は、InvalidName例外が発生します。

to_name

文字列表記のバインディング名から構造体型のバインディング名に変換します。
指定された文字列表記のバインディング名が不当である場合は、InvalidName例外が発生します。

resolve_str

指定された文字列表記のバインディング名に関係付けられたオブジェクトリファレンスを獲得します。

to_url

与えられたアドレスと、文字列表記のバインディング名からURLスキーマを作成します。
本オペレーションは、“13.4 corbaloc URLスキーマ”で定義されるcorbalocアドレスとキーパラメタ、文字列表記のバインディング名から、パラメタに必要なすべてのエスケープを処理し、完全な形式のURL文字列を返します。corbalocアドレス、キーパラメタまたは文字列表記のバインディング名が異常である場合、例外が発生します。
空の文字列表記のバインディング名""は有効ですが、空のアドレスは異常となります。空のアドレスが指定された場合、InvalidAddress例外が発生します。

CORBA::ORB::string_to_object

本オペレーションは、“9.5 URLスキーマ”で述べたURLをオブジェクトリファレンスに変換します。URLスキーマのシンタックスが不当である場合、BAD_PARAMシステム例外が発生します。

9.7 ネーミングサービスのプログラミング例

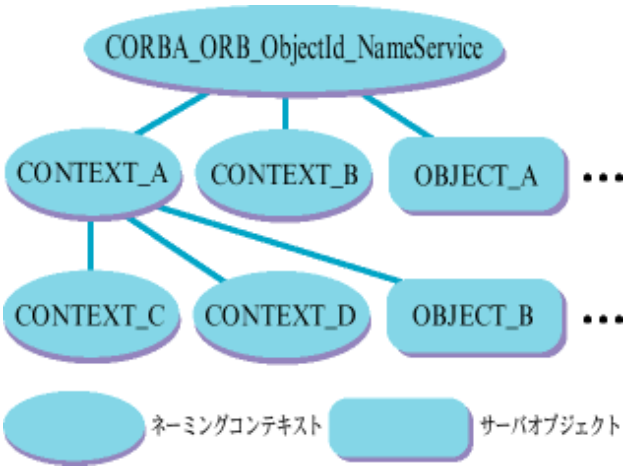
注意

Windows64 Solaris64 Linux64

OOCOBOLは、Windows(R)クライアントアプリケーションで使用できます。

9.7.1 コンテキスト配下のオブジェクトの検索

ネーミングサービスで、以下のようにネーミングコンテキスト配下にオブジェクトが格納されている場合に、オブジェクトリファレンスをどのように検索するかについて説明します。



OBJECT_Bを検索するプログラムの例を以下に示します。

(1)C言語のプログラミング例

1. ORBインタフェースのオブジェクトリファレンスを取り出すCORBA_ORB_resolve_initial_references()により、ネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとしてCORBA_ORB_ObjectId_NameServiceを指定します。
2. シーケンスCosNaming_Nameのバッファポインタに CosNaming_NameComponentの配列のポインタを設定し、次に取り出したいオブジェクトリファレンスの名前を指定します。“OBJECT_B”は“CONTEXT_A”の下に配置されているため、名前の[0]番目には“CONTEXT_A”、[1]番目には“OBJECT_B”を指定します。
3. ネーミングサービスのCosNaming_NamingContext_resolve()を使用して、ネーミングサービスからオブジェクトリファレンスを取得します。

```
CosNaming_NamingContext cos_naming;
CosNaming_Name          name;
CosNaming_NameComponent name_component[2];
CORBA_ORB               orb;
CORBA_Environment       env;
CORBA_Object            obj;

/* ルート(CORBA_ORB_ObjectId_NameService)オブジェクトの取得 */
orb = CORBA_ORB_init(&current_argc, argv, FJ_OM_ORBid, &env );cos_naming =
        CORBA_ORB_resolve_initial_references(
            orb,
            CORBA_ORB_ObjectId_NameService,
            &env );

/* ルートからOBJECT_Bを取得 */
name._length = name._maximum = 2;
name._buffer = name_component;
name._buffer[0].id = "CONTEXT_A";
name._buffer[0].kind = "";
name._buffer[1].id = "OBJECT_B";
name._buffer[1].kind = "";

obj = CosNaming_NamingContext_resolve(
        cos_naming,
        &name,
        &env );
```

(2)C++言語のプログラミング例

1. ORBインタフェースのオブジェクトリファレンスを取り出すCORBA::ORB::resolve_initial_references()によりネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとしてCORBA_ORB_ObjectId_NameServiceを指定します。
2. シーケンスCosNaming::Nameのバッファポインタ(CosNaming::NameComponent_varの配列)の領域をCosNaming::Name::allocbuf()を使って獲得し、次に取り出したいオブジェクトリファレンスの名前を指定します。“OBJECT_B”は“CONTEXT_A”の下に配置されているため、獲得する配列の個数を2、名前の[0]番目に“CONTEXT_A”、[1]番目に“OBJECT_B”を指定します。その後、上記ポインタをパラメタに指定し、演算子newを使ってシーケンスを作成します。
3. ネーミングサービスのCosNaming::NamingContext::resolve()を使用して、ネーミングサービスからオブジェクトリファレンスを取得します。

```
CORBA::ORB_ptr                orb;
CosNaming::Name_ptr           name;
CosNaming::NameComponent_var  *name_component;
CORBA::Object_ptr             obj;
CORBA::Environment            env;

try {
    // ルート (CORBA_ORB_ObjectId_NameService) オブジェクトの取得
    orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, env );
    obj = orb->resolve_initial_references(
             CORBA_ORB_ObjectId_NameService,env );
    CosNaming::NamingContext_ptr
         NamingContext_obj = CosNaming::NamingContext::_narrow( obj );

    CORBA::release( obj );

    // ルートからOBJECT_Bを取得
    name_component_var = CosNaming::Name::allocbuf(2);
    name_component[0]->id = (const CORBA::Char *)"CONTEXT_A";
    name_component[0]->kind = (const CORBA::Char *)"";
    name_component[1]->id = (const CORBA::Char *)"OBJECT_B";
    name_component[1]->kind = (const CORBA::Char *)"";
    name = new CosNaming::Name(2,2,name_component_var,CORBA_TRUE);

    OPNAME("NamingContext::resolve");
    obj = NamingContext_obj->resolve( *name, env );
    CORBA::release( NamingContext_obj );
    delete name;
}
catch ( CORBA::Exception &e ){
     ...  // エラー処理
}
```

(3)Java言語でのプログラミング例

1. ORBインタフェースのオブジェクトリファレンスを取り出すorg.omg.CORBA.ORB.resolve_initial_references()によりネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとして、“NameService”を指定します。
2. org.omg.CosNaming.NameComponent[]の領域を、newを使用して獲得します。
3. 名前を指定します。“OBJECT_B”は“CONTEXT_A”の下に配置されているため、[0]番目に“CONTEXT_A”、[1]番目に“OBJECT_B”を指定します。
4. ネーミングサービスのorg.omg.CosNaming.NamingContext.resolve()を使用して、ネーミングサービスからオブジェクトリファレンスを取得します。

```
public class Resolve {
        public static void main(String args[]){
                try {
                        int     i, j;
```

```
                    org.omg.CORBA.ORB orb =  org.omg.CORBA.ORB.init(arg,null);

                    // ルート (NameService) オブジェクトの取得
                    org.omg.CORBA.Object obj =
                                orb.resolve_initial_references("NameService");
                    org.omg.CosNaming.NamingContextExt cos_naming =
                        org.omg.CosNaming.NamingContextExtHelper.narrow(obj);

                    // ルートからOBJECT_Bを取得
                    org.omg.CosNaming.NameComponent [] name =
                              new org.omg.CosNaming.NameComponent[2];
                    name[0] = new org.omg.CosNaming.NameComponent( "CONTEXT_A", "");
                    name[1] = new org.omg.CosNaming.NameComponent( "OBJECT_B", "");
                    obj = cos_naming.resolve( name );

            }
            catch( Exception e ){
                    ... // エラー処理
            }
      }
}
```

(4)COBOLでのプログラミング例

1. ORBインタフェースのオブジェクトリファレンスを取り出すCORBA-ORB-RESOLVE-INITIAL-REFERENCESによりネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとして、“CORBA-ORB-OBJECTID-NAMESERVICE”を指定します。
2. COSNAMING-NAMEおよびCOSNAMING-NAMECOMPONENTに取り出したいオブジェクトリファレンスの名前を以下のプログラムのように指定します。“OBJECT_B”は“CONTEXT_A”の下に配置されているため、1番目に“CONTEXT_A”、2番目に“OBJECT_B”を指定します。
3. ネーミングサービスのCOSNAMING-NAMINGCONTEXT-RESOLVEを使用して、ネーミングサービスからオブジェクトリファレンスを取得します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "RESOLVE-TEST".
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
      ARGUMENT-NUMBER      IS  ARG-C
      ARGUMENT-VALUE       IS  ARG-V
      SYMBOLIC CONSTANT
      COPY SYMBOL-CONST IN CORBA.
      .
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 COPY CONST  IN CORBA.
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.

*########## NAMING SETTING PARAMETER #########
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING COSNAMING-NAMINGCONTEXT
                                                          BY COS-NAMING.
 01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
 01 NAME-A USAGE POINTER.
 01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING COSNAMING-NAMECOMPONENT
                                                          BY NAME-COMPONENT.
 01 NAME-COMPONENT-A USAGE POINTER.
```

```
*########## ORB SETTING PARAMETER #########
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
 01 CURRENT-ARG-V.
    02 FILLER OCCURS 6.
       03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_c".
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.

 01 TMP-STRING       USAGE POINTER.
 01 TMP-STRING-BUF  PIC X(50).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY ELEMENT-NUMBER.
*
 PROCEDURE DIVISION.

* ObjectDirector初期化
     ACCEPT CURRENT-ARG-C FROM ARG-C.
     COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
     PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
        IF ARG-COUNT = 1
           MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
        END-IF
        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        CALL "CORBA-STRING-SET" USING
              CURRENT-ARG-V-VALUE (ARG-COUNT)
              STRING-LENGTH
              TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.

     MOVE FUNCTION LENG(FJ-OM-ORB-ID) TO STRING-LENGTH.
     CALL  "CORBA-STRING-SET" USING
           TMP-STRING
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           CURRENT-ARG-C
           CURRENT-ARG-V
           TMP-STRING
           ENV
           ORB.
     CALL "CORBA-FREE" USING TMP-STRING.

* ルート(CORBA-ORB-OBJECTID-NAMESERVICE)オブジェクトの取得
     MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TMP-STRING
           STRING-LENGTH
           CORBA-ORB-OBJECTID-NAMESERVICE.
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
           ORB
           TMP-STRING
           ENV
           COS-NAMING.
     CALL "CORBA-FREE" USING TMP-STRING.

* ルートからOBJECT_Bを取得
     MOVE 2 TO SEQ-LENGTH OF NAME.
     MOVE 2 TO SEQ-MAXIMUM OF NAME.
     CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
```

```
            SEQ-MAXIMUM OF NAME
            SEQ-BUFFER OF NAME.

      MOVE "CONTEXT_A" TO TMP-STRING-BUF.
      MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
            IDL-ID OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

      MOVE " " TO TMP-STRING-BUF.
      CALL "CORBA-STRING-SET" USING
            KIND OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

      MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
      MOVE 1 TO ELEMENT-NUMBER.
      MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            NAME-A
            ELEMENT-NUMBER
            NAME-COMPONENT-A.

      MOVE "OBJECT_B" TO TMP-STRING-BUF.
      MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
            IDL-ID OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

      MOVE " " TO TMP-STRING-BUF.
      CALL "CORBA-STRING-SET" USING
            KIND OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

      MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
      MOVE 2 TO ELEMENT-NUMBER.
      MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            NAME-A
            ELEMENT-NUMBER
            NAME-COMPONENT-A.

      CALL "COSNAMING-NAMINGCONTEXT-RESOLVE" USING
            COS-NAMING
            NAME
            ENV
            OBJ.
      IF OBJ = NULL
            CALL "CORBA-STRING-GET" USING
                        IDL-ID OF ENV
                        STRING-LENGTH
                        TMP-STRING-BUF
            DISPLAY "ENV-ID : " TMP-STRING-BUF
      END-IF
      EXIT PROGRAM.
  END PROGRAM "RESOLVE-TEST".
```

(5)OOCOBOLでのプログラミング例

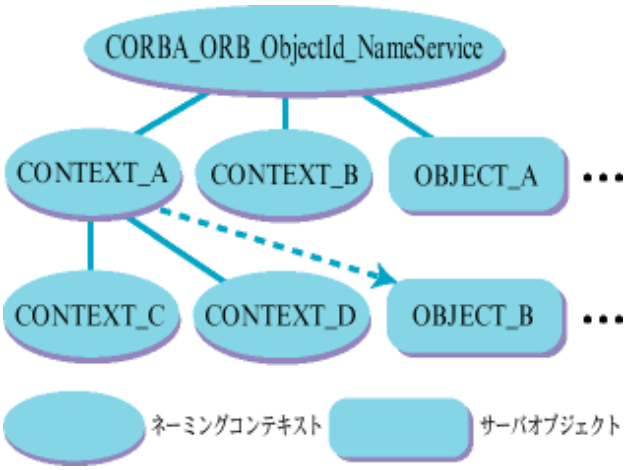
1. ORBインタフェースのオブジェクトリファレンスを取り出すCORBA-ORB-RESOLVE_INITIAL_REFERENCESによりネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとして、“CORBA-ORB-OBJECTID_NAMESERVICE”を指定します。
2. COSNAMING-NAMEおよびCOSNAMING-NAMECOMPONENTに取り出したいオブジェクトリファレンスの名前を以下のプログラムのように指定します。“OBJECT_B”は“CONTEXT_A”の下に配置されているため、1番目に“CONTEXT_A”、2番目に“OBJECT_B”を指定します。
3. ネーミングサービスのCOSNAMING-NAMINGCONTEXT-RESOLVEを使用して、ネーミングサービスからオブジェクトリファレンスを取得します。

```
 IDENTIFICATION          DIVISION.
 PROGRAM-ID.             "CLIENT-MAIN".
*
 ENVIRONMENT             DIVISION.
 CONFIGURATION           SECTION.
 REPOSITORY.
     COPY                CORBA--REP.
     COPY                COSNAMING--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY                CORBA--CONST.
     COPY                COSNAMING--CONST.
     .
*
 DATA                    DIVISION.
 WORKING-STORAGE         SECTION.
 COPY                    CORBA--COPY.
 COPY                    COSNAMING--COPY.
*
 01 ORB                  USAGE OBJECT REFERENCE CORBA-ORB.
 01 BOA                  USAGE OBJECT REFERENCE CORBA-BOA.
 01 IMPL-REP             USAGE OBJECT REFERENCE FJ-IMPLEMENTATIONREP.
 01 IMPL                 USAGE OBJECT REFERENCE CORBA-IMPLEMENTATIONDEF.
 01 INTF-REP             USAGE OBJECT REFERENCE CORBA-REPOSITORY.
 01 INTF                 USAGE OBJECT REFERENCE CORBA-INTERFACEDEF.
 01 OBJ                  USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 NAMING-CONTEXT       USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                 TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT       USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NAME-ID              USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND            USAGE OBJECT REFERENCE CORBA-STRING.
*
 01 EXCEPTION-ID         USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE   PIC   X(50).
 01 API-NAME             PIC   X(50).
*
 01 I                    TYPE                   CORBA-UNSIGNED-LONG.
*
 PROCEDURE               DIVISION.
*
 DECLARATIVES.
*
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::Exception: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
```

```
*
 END DECLARATIVES.
*
 MAIN SECTION.
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
                         USING     "Naming"
                                   FJ-OM_ORBID
                         RETURNING ORB.
*
     MOVE "CORBA::ORB::BOA_init" TO API-NAME.
     INVOKE ORB "BOA_INIT"
                         USING     "Naming"
                                   CORBA-BOA_OAID
                         RETURNING BOA.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                         USING     CORBA-ORB-OBJECTID_NAMESERVICE
                         RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                         USING     OBJ
                         RETURNING NAMING-CONTEXT.
*
     INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH" USING 2 RETURNING NAME.
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING "CONTEXT_A" RETURNING NAME-ID
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING " " RETURNING NAME-KIND
     INVOKE COSNAMING-NAMECOMPONENT "NEW" RETURNING NAME-COMPONENT
     SET IDL-ID OF NAME-COMPONENT TO NAME-ID
     SET KIND OF NAME-COMPONENT TO NAME-KIND
     INVOKE NAME "SET-VALUE" USING 1 NAME-COMPONENT
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING "OBJECT_B" RETURNING NAME-ID
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING " " RETURNING NAME-KIND
     INVOKE COSNAMING-NAMECOMPONENT "NEW" RETURNING NAME-COMPONENT
     SET IDL-ID OF NAME-COMPONENT TO NAME-ID
     SET KIND OF NAME-COMPONENT TO NAME-KIND
     INVOKE NAME "SET-VALUE" USING 2 NAME-COMPONENT
*
     MOVE "CosNaming::NamingContext::resolve" TO API-NAME.
     INVOKE NAMING-CONTEXT "RESOLVE"
                         USING     NAME
                         RETURNING OBJ.
*
     STOP RUN.
*
 END-MAIN.
*
 END PROGRAM "CLIENT-MAIN".
```

9.7.2 コンテキスト配下のオブジェクトの登録

ネーミングサービスで、以下のようにネーミングコンテキスト配下にオブジェクトが格納されている場合に、オブジェクトリファレンスをどのように登録するかについて説明します。



OBJECT_Bを登録するプログラムの例を以下に示します。

(1)C言語のプログラミング例

1. ORBインタフェースのオブジェクトリファレンスを取り出すCORBA_ORB_resolve_initial_references()により、ネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとして、CORBA_ORB_ObjectId_NameServiceを指定します。
2. シーケンスCosNaming_Nameのバッファポインタに CosNaming_NameComponentの配列のポインタを設定し、次に取り出したいオブジェクトリファレンスの名前を指定します。“OBJECT_B”は“CONTEXT_A”の下に配置されるため、獲得する配列の個数を2、名前の[0]番目に“CONTEXT_A”、[1]番目に“OBJECT_B”を指定します。
3. ネーミングサービスのCosNaming_NamingContext_bind()を使用して、ネーミングサービスにオブジェクトリファレンスを登録します。

```
CORBA_ORB                       orb;
CosNaming_NamingContext         cos_naming;
CosNaming_Name                  name;
CosNaming_NameComponent         name_component[2];
CORBA_Environment               env;

CORBA_Object                    obj;

/* ルート(CORBA_ORB_ObjectId_NameService)オブジェクトの取得 */
orb = CORBA_ORB_init(&current_argc, argv, FJ_OM_ORBid, &env );
cos_naming = CORBA_ORB_resolve_initial_references(
                  orb,
                  CORBA_ORB_ObjectId_NameService,
                  &env );

obj = ...; /* 何らかの方法でObjectを作成し、取得します。*/

/* OBJECT_Bのシーケンス領域の設定 */
name._length = name._maximum = 2;
name._buffer = name_component;
name_component[0].id = "CONTEXT_A";
name_component[0].kind = "";
name_component[1].id = "OBJECT_B";
name_component[1].kind = "";

/* ネーミングサービスにオブジェクトリファレンスを登録 */
CosNaming_NamingContext_bind(
      cos_naming,
      &name,
```

```
    obj,
    &env );
```

(2)C++言語のプログラミング例

1. ORBインタフェースのオブジェクトリファレンスを取り出すCORBA::ORB::resolve_initial_references()によりネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとしてCORBA_ORB_ObjectId_NameServiceを指定します。
2. シーケンスCosNaming::Nameのバッファポインタ(CosNaming::NameComponent_varの配列)の領域をCosNaming::Name::allocbuf()を使って獲得し、次に取り出したいオブジェクトリファレンスの名前を指定します。“OBJECT_B”は“CONTEXT_A”の下に配置されるため、獲得する配列の個数を2、名前の[0]番目に“CONTEXT_A”、[1]番目に“OBJECT_B”を指定します。
3. 2.のポインタをパラメタに指定し、演算子newを使ってシーケンスを作成します。
4. ネーミングサービスのCosNaming::NamingContext::bind()を使用して、ネーミングサービスにオブジェクトリファレンスを登録します。

```
CORBA::ORB_ptr                orb;
CosNaming::Name_ptr           name;
CosNaming::NameComponent_var  *name_component;
CORBA::Environment            env;

CORBA::Object_ptr             obj;

try {
    // ルート (CORBA_ORB_ObjectId_NameService) オブジェクトの取得
    orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

    OPNAME("resolve_initial_references");
    CORBA::Object_ptr
    obj = orb->resolve_initial_references(
              CORBA_ORB_ObjectId_NameService, env );

    CosNaming::NamingContext_ptr
    NamingContext_obj = CosNaming::NamingContext::_narrow( obj );
    CORBA::release( obj );

    obj = ...;  // 何らかの方法でObjectを作成し、取得します。

    // OBJECT_Bのシーケンス領域の設定
    name_component = CosNaming::Name::allocbuf(2);
    name_component[0]->id = (const CORBA::Char *)"CONTEXT_A";
    name_component[0]->kind = (const CORBA::Char *)"";
    name_component[1]->id = (const CORBA::Char *)"OBJECT_B";
    name_component[1]->kind = (const CORBA::Char *)"";
    name = new CosNaming::Name(2, 2, name_component,CORBA_TRUE);

    // ネーミングサービスにオブジェクトリファレンスを登録
    OPNAME("NamingContext::bind");
    NamingContext_obj->bind( *name, obj, env );
    delete name;

    CORBA::release( NamingContext_obj );
}
catch ( CORBA::Exception &e ){
    ...  // エラー処理
}
```

(3)Java言語でのプログラミング例

1. ORBインタフェースのオブジェクトリファレンスを取り出すorg.omg.CORBA.ORB.resolve_initial_references()によりネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとして、“NameService”を指定します。
2. org.omg.CosNaming.NameComponent[]の領域を、newを使用して獲得します。
3. 名前を指定します。“OBJECT_B”は“CONTEXT_A”の下に配置されているため、[0]番目に“CONTEXT_A”、[1]番目に“OBJECT_B”を指定します。
4. ネーミングサービスのorg.omg.CosNaming.NamingContext.bind()を使用して、ネーミングサービスにオブジェクトリファレンスを登録します。

```
public class Bind {
      public static void main(String args[]){
              try {
                       // ルート (NameService) オブジェクトの取得
                       org.omg.CORBA.ORB orb =  org.omg.CORBA.ORB.init(arg,null);

                       org.omg.CORBA.Object obj =
                                orb.resolve_initial_references("NameService");
                       org.omg.CosNaming.NamingContextExt cos_naming =
                                org.omg.CosNaming.NamingContextExtHelper.narrow(obj);

                       obj = ...; // 何らかの方法でObjectを作成し、取得します。

                       // OBJECT_Bのシーケンス領域の設定
                        org.omg.CosNaming.NameComponent [] name =
                                    new org.omg.CosNaming.NameComponent[2];
                        name[0] = new org.omg.CosNaming.NameComponent( "CONTEXT_A", "");
                        name[1] = new org.omg.CosNaming.NameComponent( "OBJECT_B", "");

                        // ネーミングサービスにオブジェクトリファレンスを登録
                       cos_naming.bind( name, obj );

              }
              catch( Exception e ){
                     ...  // エラー処理
              }
      }
}
```

(4)COBOLでのプログラミング例

1. ORBインタフェースのオブジェクトリファレンスを取り出すCORBA-ORB-RESOLVE-INITIAL-REFERENCESによりネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとして、“CORBA-ORB-OBJECTID-NAMESERVICE”を指定します。
2. COSNAMING-NAMEおよびCOSNAMING-NAMECOMPONENTに取り出したいオブジェクトリファレンスの名前を以下のプログラムのように指定します。“OBJECT_B”は“CONTEXT_A”の下に配置されているため、1番目に“CONTEXT_A”、2番目に“OBJECT_B”を指定します。
3. ネーミングサービスのCOSNAMING-NAMINGCONTEXT-BINDを使用して、ネーミングサービスにオブジェクトリファレンスを登録します。

```
  IDENTIFICATION DIVISION.
  PROGRAM-ID. "BIND-TEST".
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SPECIAL-NAMES.
      ARGUMENT-NUMBER      IS  ARG-C
      ARGUMENT-VALUE       IS  ARG-V
      SYMBOLIC CONSTANT
```

```
      COPY SYMBOL-CONST IN CORBA.
      .
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 COPY CONST  IN CORBA.
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.

*########## NAMING SETTING PARAMETER #########
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING COSNAMING-NAMINGCONTEXT
                                                              BY COS-NAMING.
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING COSNAMING-NAMINGCONTEXT
                                                              BY COS-NAMING2.
 01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
 01 NAME-A USAGE POINTER.
 01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING COSNAMING-NAMECOMPONENT
                                                              BY NAME-COMPONENT.
 01 NAME-COMPONENT-A USAGE POINTER.

*########## ORB SETTING PARAMETER #########
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
 01 CURRENT-ARG-V.
    02 FILLER OCCURS 6.
       03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_c".
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.

 01 TMP-STRING       USAGE POINTER.
 01 TMP-STRING-BUF  PIC X(50).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY ELEMENT-NUMBER.
*
 PROCEDURE DIVISION.

* ObjectDirector初期化
     ACCEPT CURRENT-ARG-C FROM ARG-C.
     COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
     PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
        IF ARG-COUNT = 1
           MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
        END-IF
        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        CALL "CORBA-STRING-SET" USING
              CURRENT-ARG-V-VALUE (ARG-COUNT)
              STRING-LENGTH
              TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.

     MOVE FUNCTION LENG(FJ-OM-ORB-ID) TO STRING-LENGTH.
     CALL  "CORBA-STRING-SET" USING
           TMP-STRING
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           CURRENT-ARG-C
           CURRENT-ARG-V
           TMP-STRING
           ENV
```

```
            ORB.
      CALL "CORBA-FREE" USING TMP-STRING.

* ルート(CORBA-ORB-OBJECTID-NAMESERVICE)オブジェクトの取得
      MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
            TMP-STRING
            STRING-LENGTH
            CORBA-ORB-OBJECTID-NAMESERVICE.
      CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
            ORB
            TMP-STRING
            ENV
            COS-NAMING.
      CALL "CORBA-FREE" USING TMP-STRING.

* 何らかの方法でObjectを作成し、取得します。
      MOVE  COS-NAMING TO OBJ.


* OBJECT_Bのシーケンス領域の設定
      MOVE 2 TO SEQ-LENGTH OF NAME.
      MOVE 2 TO SEQ-MAXIMUM OF NAME.
      CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
            SEQ-MAXIMUM OF NAME
            SEQ-BUFFER OF NAME.

      MOVE "CONTEXT_A" TO TMP-STRING-BUF.
      MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
            IDL-ID OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF
            .
      MOVE " " TO TMP-STRING-BUF.
      CALL "CORBA-STRING-SET" USING
            KIND OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.
      MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
      MOVE 1 TO ELEMENT-NUMBER.
      MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            NAME-A
            ELEMENT-NUMBER
            NAME-COMPONENT-A.

      MOVE "OBJECT_B" TO TMP-STRING-BUF.
      MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
            IDL-ID OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

      MOVE " " TO TMP-STRING-BUF.
      CALL "CORBA-STRING-SET" USING
            KIND OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

      MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
      MOVE 2 TO ELEMENT-NUMBER.
      MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
```

```
     CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
          NAME-A
          ELEMENT-NUMBER
          NAME-COMPONENT-A.
*ネーミングサービスにオブジェクトリファレンスを登録
     CALL "COSNAMING-NAMINGCONTEXT-BIND" USING
          COS-NAMING
          NAME
          OBJ
          ENV.
     EXIT PROGRAM.
 END PROGRAM "BIND-TEST".
```

(5)OOCOBOLでのプログラミング例

1. ORBインタフェースのオブジェクトリファレンスを取り出すCORBA-ORB-RESOLVE_INITIAL_REFERENCESによりネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとして、“CORBA-ORB-OBJECTID_NAMESERVICE”を指定します。
2. COSNAMING-NAMEおよびCOSNAMING-NAMECOMPONENTに取り出したいオブジェクトリファレンスの名前を以下のプログラムのように指定します。“OBJECT_B”は“CONTEXT_A”の下に配置されているため、1番目に“CONTEXT_A”、2番目に“OBJECT_B”を指定します。
3. ネーミングサービスのCOSNAMING-NAMINGCONTEXT-BINDを使用して、ネーミングサービスにオブジェクトリファレンスを登録します。

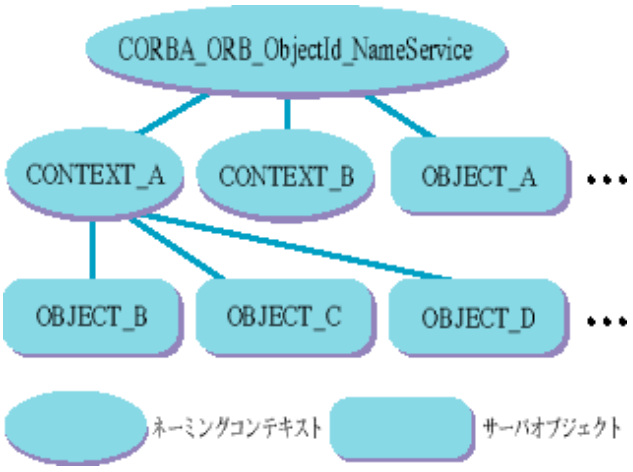
```
 IDENTIFICATION         DIVISION.
 PROGRAM-ID.            "CLIENT-MAIN".
*
 ENVIRONMENT            DIVISION.
 CONFIGURATION          SECTION.
 REPOSITORY.
     COPY               CORBA--REP.
     COPY               COSNAMING--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY               CORBA--CONST.
     COPY               COSNAMING--CONST.
     .
*
 DATA                   DIVISION.
 WORKING-STORAGE        SECTION.
 COPY                   CORBA--COPY.
 COPY                   COSNAMING--COPY.
*
 01 ORB                 USAGE OBJECT REFERENCE CORBA-ORB.
 01 BOA                 USAGE OBJECT REFERENCE CORBA-BOA.
 01 IMPL-REP            USAGE OBJECT REFERENCE FJ-IMPLEMENTATIONREP.
 01 IMPL                USAGE OBJECT REFERENCE CORBA-IMPLEMENTATIONDEF.
 01 INTF-REP            USAGE OBJECT REFERENCE CORBA-REPOSITORY.
 01 INTF                USAGE OBJECT REFERENCE CORBA-INTERFACEDEF.
 01 REF-ID              TYPE                   CORBA-REFERENCEDATA.
 01 OBJ                 USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 NAMING-CONTEXT      USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT      USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NAME-ID             USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND           USAGE OBJECT REFERENCE CORBA-STRING.
*
 01 EXCEPTION-ID        USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE  PIC   X(50).
```

```
 01 API-NAME            PIC   X(50).
*
 01 I                   TYPE                  CORBA-UNSIGNED-LONG.
*
 PROCEDURE              DIVISION.
*
 DECLARATIVES.
*
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::Exception: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
 MAIN SECTION.
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
                        USING     "Naming"
                                  FJ-OM_ORBID
                        RETURNING ORB.
*
     MOVE "CORBA::ORB::BOA_init" TO API-NAME.
     INVOKE ORB "BOA_INIT"
                        USING     "Naming"
                                  CORBA-BOA_OAID
                        RETURNING BOA.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                        USING     CORBA-ORB-OBJECTID_NAMESERVICE
                        RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                        USING     OBJ
                        RETURNING NAMING-CONTEXT.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                        USING     CORBA-OBJECTID_LIGHTINTERF-001
                        RETURNING OBJ.
     INVOKE CORBA-REPOSITORY "NARROW" USING OBJ RETURNING INTF-REP.
*
     MOVE "CORBA::Repository::lookup_id" TO API-NAME.
     INVOKE INTF-REP "LOOKUP_ID" USING "IDL:OBJECT_B:1.0" RETURNING OBJ.
     INVOKE CORBA-INTERFACEDEF "NARROW" USING OBJ RETURNING INTF.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                        USING     CORBA-OBJECTID_IMPLEMENTAT-001
                        RETURNING OBJ.
     INVOKE FJ-IMPLEMENTATIONREP "NARROW" USING OBJ RETURNING IMPL-REP.
*
     MOVE "FJ::ImplementationRep::lookup_id" TO API-NAME.
     INVOKE IMPL-REP "LOOKUP_ID" USING "IDL:OBJECT_B:1.0" RETURNING OBJ.
     INVOKE CORBA-IMPLEMENTATIONDEF "NARROW" USING OBJ RETURNING IMPL.
*
     MOVE "CORBA::BOA::create" TO API-NAME.
```

```
      INVOKE SEQUENCE-OCTET "NEW" RETURNING REF-ID.
      INVOKE BOA "IDL-CREATE" USING REF-ID INTF IMPL RETURNING OBJ.
*
      INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH" USING 2 RETURNING NAME.
      INVOKE CORBA-STRING "NEW-WITH-VALUE" USING "CONTEXT_A" RETURNING NAME-ID
      INVOKE CORBA-STRING "NEW-WITH-VALUE" USING " " RETURNING NAME-KIND
      INVOKE COSNAMING-NAMECOMPONENT "NEW" RETURNING NAME-COMPONENT
      SET IDL-ID OF NAME-COMPONENT TO NAME-ID
      SET KIND OF NAME-COMPONENT TO NAME-KIND
      INVOKE NAME "SET-VALUE" USING 1 NAME-COMPONENT
*
      INVOKE CORBA-STRING "NEW-WITH-VALUE" USING "OBJECT_B" RETURNING NAME-ID
      INVOKE CORBA-STRING "NEW-WITH-VALUE" USING " " RETURNING NAME-KIND
      INVOKE COSNAMING-NAMECOMPONENT "NEW" RETURNING NAME-COMPONENT
      SET IDL-ID OF NAME-COMPONENT TO NAME-ID
      SET KIND OF NAME-COMPONENT TO NAME-KIND
      INVOKE NAME "SET-VALUE" USING 2 NAME-COMPONENT
*
      MOVE "CosNaming::NamingContext::bind" TO API-NAME.
      INVOKE NAMING-CONTEXT "BIND"
                          USING     NAME
                                    OBJ.
*
      STOP RUN.
*
 END-MAIN.
*
 END PROGRAM "CLIENT-MAIN".
```

9.7.3 コンテキスト配下のオブジェクトのリストの取得

ネーミングサービスで、以下のようにネーミングコンテキスト配下にオブジェクトが格納されている場合に、ネーミングコンテキスト配下のオブジェクトリファレンスのリストをどのように取得するかについて説明します。



CONTEXT_Aのリストを取得するプログラムの例を以下に示します。

(1)C言語のプログラミング例

1. オブジェクトリファレンスの検索と同じ方法で、リストを取得したいネーミングコンテキストのオブジェクトリファレンスを取得します。
2. CosNaming_NamingContext_list()によりリストを取得します。
   結果は、第3パラメタCosNaming_BindingListに第2パラメタで指定して個数設定されます。もし、第2パラメタで指定した個数より

含まれるリストの数が多い場合は、残りの検索のためのオブジェクトCosNaming_BindingIteratorにオブジェクトリファレンスが設定されます。第2パラメタで指定した数より含まれるリストの数が少ない場合は、CosNaming_BindingIteratorにCORBA_OBJECT_NILが設定され、CosNaming_BindingListにそのリストの個数分のデータが設定されます。
CosNaming_BindingIterator により残りのリストを取得するには、CosNaming_BindingIterator_next_one() ／ CosNaming_BindingIterator_next_n()を使用します。
本プログラムでは、以下のように処理しています。

1. CosNaming_BindingIterator_next_n()により最初の40個のオブジェクトリファレンスを検索します。
2. _length 0のシーケンスが返るまで、CosNaming_BindingIterator_next_one()で残りのオブジェクトリファレンスのリストを検索します。

3. 検索終了後、CosNaming_BindingIterator_destroy()を呼び出してサーバ側の領域を解放する必要があります。クライアント側の領域は、CORBA_Object_release()を使用します。

```
CosNaming_NamingContext        cos_naming, context_obj;
CosNaming_Name                 name;
CosNaming_NameComponent        name_component;
CORBA_ORB                      orb;
CORBA_Environment              env;
CORBA_Object                   obj;
CosNaming_BindingList          *bl, *bl2;
CosNaming_BindingIterator      bi;
CORBA_long                     how_many;

/* ルート(CORBA_ORB_ObjectId_NameService)オブジェクトの取得 */
cos_naming = CORBA_ORB_resolve_initial_references(
              orb,
              CORBA_ORB_ObjectId_NameService,
              &env );

/* "CONTEXT_A"のネーミングコンテキストの取り出し */
name._length = name._maximum = 1;
name._buffer = &name_component;
name._buffer[0].id = "CONTEXT_A";
name._buffer[0].kind = "";

context_obj = CosNaming_NamingContext_resolve(
              cos_naming,
              &name,
              &env );

/* CONTEXT_A下のオブジェクトリファレンスのリストの取り出し */
name._length = name._maximum = 1;
name._buffer = &name_component;
name._buffer[0].id = "CONTEXT_A";
name._buffer[0].kind = "";

how_many = 40;
CosNaming_NamingContext_list(
      cos_naming,
      how_many,
      &bl,
      &bi,
      &env );

/* CONTEXT_A下の残りのオブジェクトリファレンスの取り出し */
      if( bi != CORBA_OBJECT_NIL ){
              CORBA_free(bl);
              CosNaming_BindingIterator_next_n( bi, how_many, &bl2, &env );
              do {
                      CosNaming_BindingIterator_next_one(
                          bi, &b, &env );
```

```
                ... /* 結果を処理する処理 */
                CORBA_free(b);
        }
        while( env._major != CORBA_NO_EXCEPTION &&
                b->binding_name._length != 0 );
        CosNaming_BindingIterator_destroy( bi, &env );
    }
```

(2)C++言語のプログラミング例

1. オブジェクトリファレンスの検索と同じ方法で、リストを取得したいネーミングコンテキストのオブジェクトリファレンスを取得します。
2. CosNaming::NamingContext::list()を使ってリストを取得します。
   結果は、第2パラメタCosNaming::BindingListに第1パラメタで指定して個数設定されます。もし、第1パラメタで指定した個数より含まれるリストの数が多い場合は、残りの検索のためのオブジェクトCosNaming::BindingIteratorにオブジェクトリファレンスが設定されます。第2パラメタで指定した数より含まれるリストの数が少ない場合は、CosNaming::BindingIteratorにNILオブジェクトが設定され、CosNaming::BindingListにそのリストの個数分のデータが設定されます。
   CosNaming::BindingIteratorを使って残りのリストを取得するには、CosNaming::BindingIterator::next_one()／CosNaming::BindingIterator::next_n()を使用します。
   本プログラムでは、以下のように処理しています。
   1. CosNaming::BindingIterator::next_n()により最初の40個のオブジェクトリファレンスを検索します。
   2. _length 0のシーケンスが返るまで、CosNaming::BindingIterator::next_one()で残りのオブジェクトリファレンスのリストを検索します。
3. 検索終了後、CosNaming::BindingIterator::destroy()を呼び出してサーバ側の領域を解放する必要があります。クライアント側の領域は、CORBA::release()を使用します。

```
CORBA::ORB_ptr                  orb;
CosNaming::NamingContext_ptr    cos_naming;
CosNaming::Name_ptr             name;
CosNaming::NameComponent_var    *name_component;
CosNaming::BindingList_ptr      bl;
CosNaming::BindingIterator      bi;
CosNaming::Binding_ptr          b;
CORBA::Object_ptr               obj;
CosNaming::NamingContext_ptr    cos_naming, new_context;
CORBA::Long                     how_many;
CORBA::Environment              env;
CORBA::Long                     len;

try {
      // ルート (CORBA_ORB_ObjectId_NameService) オブジェクトの取得
      orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, env );
      obj = orb->resolve_initial_references(
          CORBA_ORB_ObjectId_NameService,
          env );
      cos_naming = CosNaming::NamingContext::_narrow(obj);
      CORBA::release(obj);

      // ルートからCONTEXT_Aを取得
      CosNaming::NameComponent_var *name_component_var =
             CosNaming::Name::allocbuf(1);
      name_component[0]->id = (const CORBA::Char *)"CONTEXT_A";
      name_component[0]->kind = (const CORBA::Char *)"";
      name = new CosNaming::Name(1,1,name_component_var,CORBA_TRUE);
      obj = cos_naming->resolve( *name, env );
      delete name;
      new_context = CosNaming::NamingContext::_narrow(obj);
      CORBA::release(obj);
```

```
        // CONTEXT_A下のオブジェクトリファレンスのリストの取り出し
        how_many = 40;
        cos_naming->list( how_many, bl, bi, env );
        ...  //  結果を処理する処理
        delete bl;

        // CONTEXT_A下の残りのオブジェクトリファレンスの取り出し
        if( bi->is_nil(env) != CORBA_TRUE ){
                bi->next_n( how_many, bl, env );
                ...  // 結果を処理する処理

                do {
                    bi->next_one(b,env);
                    // 結果を処理する処理
                    len = b->binding_name->length();
                    delete b;
                }
                while( len != 0 );
        }
        bi->destroy(env);
        CORBA::release(bi);
}
catch ( CORBA::Exception &e ){
    ...  // エラー処理
}
```

(3)Java言語のプログラミング例

1. オブジェクトリファレンスの検索と同じ方法で、リストを取得したいネーミングコンテキストのオブジェクトリファレンスを取得します。

2. org.omg.CosNaming.NamingContext.list()を使ってリストを取得します。
   結果は、第2パラメタorg.omg.CosNaming.BindingListHolder.valueに第1パラメタで指定した個数だけorg.omg.CosNaming.Bindingの配列が設定されます。もし、第1パラメタで指定した個数より含まれるリストの数が多い場合は、残りの検索のためのオブジェクトorg.omg.CosNaming.BindingIteratorHolder.valueにorg.omg.CosNaming.BindingIteratorのオブジェクトリファレンスが設定されます。第2パラメタで指定した数より含まれるリストの数が少ない場合は、org.omg.CosNaming.BindingIterator.valueにnullが、org.omg.CosNaming.BindingListHolder.valueにそのリストの個数分のデータが設定されます。
   org.omg.CosNaming.BindingIteratorを使って残りのリストを取得するには、org.omg.CosNaming.BindingIterator.next_one()／org.omg.CosNaming.BindingIterator.next_n()を使用します。
   本プログラムでは、以下のように処理しています。

   1. org.omg.CosNaming.BindingIterator.next_n()を使って最初の40個のオブジェクトリファレンスを検索します。

   2. length 0のシーケンスが返るまで、org.omg.CosNaming.BindingIterator.next_one()で残りのオブジェクトリファレンスのリストを検索します。

```
public class List {
    public static void main(String args[]){
        try {
            int     i, j;

            // ルート (CORBA_ORB_ObjectId_NameService) オブジェクトの取得
            org.omg.CORBA.ORB orb =  org.omg.CORBA.ORB.init(arg,null);
            org.omg.CORBA.Object obj = orb.resolve_initial_references("NameService");
            org.omg.CosNaming.NamingContextExt con_obj =
                            org.omg.CosNaming.NamingContextExtHelper.narrow(obj);

            // ルートからCONTEXT_Aを取得
            org.omg.CosNaming.NameComponent [] name =
                            new org.omg.CosNaming.NameComponent[1];
            name[0] = new org.omg.CosNaming.NameComponent("CONTEXT_A", "");

            org.omg.CORBA.Object tmp_obj =  con_obj.resolve( name );
```

```
            org.omg.CosNaming.NamingContextExt new_context =
                    org.omg.CosNaming.NamingContextExtHelper.narrow(tmp_obj);

            // CONTEXT_A下のオブジェクトリファレンスのリストの取り出し
            int how_many = 40;
            org.omg.CosNaming.BindingListHolder bl_var =
                    new org.omg.CosNaming.BindingListHolder();
            org.omg.CosNaming.BindingIteratorHolder bi_var =
                    new org.omg.CosNaming.BindingIteratorHolder();
            new_context.list( how_many, bl_var, bi_var );

                ...  // 結果を処理する処理

            // CONTEXT_A下の残りのオブジェクトリファレンスの取り出し
            if( bi_var.value != null)) {
                org.omg.CosNaming.BindingIterator bi_obj = bi_var.value;
                bi_obj.next_n( how_many, bl_var );
                     ...      // 結果を処理する処理

                org.omg.CosNaming.BindingHolder b_var =
                       new org.omg.CosNaming.BindingHolder();
                do {
                    bi_obj.next_one( b_var );
                     ...      // 結果を処理する処理
                }
                while( b_var.value.binding_name.length != 0 );
             }
        }
        catch( Exception e ){
            ...  // エラー処理
        }
    }
}
```

(4)COBOLのプログラミング例

1. オブジェクトリファレンスの検索と同じ方法で、リストを取得したいネーミングコンテキストのオブジェクトリファレンスを取得します。

2. COSNAMING-NAMINGCONTEXT-LISTを使ってリストを取得します。
   結果は、BLで指定されたCOSNAMING-BINDINGLISTにHOW-MANYで指定された個数分設定されます。もし、HOW-MANYで指定された個数より含まれるリストの数が多い場合は、残りの検索のためのオブジェクトCOSNAMING-BINDINGITERATORにオブジェクトリファレンスが設定されます。
   COSNAMING-BINDINGITERATORを使って残りのリストを取得するには、COSNAMING-BINDINGITERATOR-NEXT-ONE／COSNAMING-BINDINGITERATOR-NEXT-Nを使用します。
   本プログラムでは、以下のように処理しています。

   1. COSNAMING-BINDINGITERATOR-NEXT-Nを使って最初の40個のオブジェクトリファレンスを検索します。

   2. COSNAMING-BINDINGITERATOR-NEXT-ONEで残りのオブジェクトリファレンスのリストを検索します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "LIST-TEST".
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
      ARGUMENT-NUMBER      IS  ARG-C
      ARGUMENT-VALUE       IS  ARG-V
      SYMBOLIC CONSTANT
      COPY SYMBOL-CONST IN CORBA.
      .
 DATA DIVISION.
 WORKING-STORAGE SECTION.
```

```
 COPY CONST  IN CORBA.
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.

*########## NAMING SETTING PARAMETER #########
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING COSNAMING-NAMINGCONTEXT
                                                              BY COS-NAMING.
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING COSNAMING-NAMINGCONTEXT
                                                              BY CONTEXT-OBJ.
 01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
 01 NAME-A USAGE POINTER.
 01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING COSNAMING-NAMECOMPONENT
                                                              BY NAME-COMPONENT.
 01 NAME-COMPONENT-A USAGE POINTER.
 01 BL     USAGE POINTER.
 01 BL2    USAGE POINTER.
 01 COPY COSNAMING-BINDINGITERATOR IN CORBA REPLACING COSNAMING-BINDINGITERATOR
                                                              BY BI.
 01 B       USAGE POINTER.

*########## ORB SETTING PARAMETER #########
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
 01 CURRENT-ARG-V.
   02 FILLER OCCURS 6.
      03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_c".
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.

 01 TMP-STRING       USAGE POINTER.
 01 TMP-STRING-BUF  PIC X(20).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY ELEMENT-NUMBER.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY HOW-MANY.
 01 COPY BOOLEAN IN CORBA REPLACING CORBA-BOOLEAN BY ROOP-FLAG.
 01 COPY BOOLEAN IN CORBA REPLACING CORBA-BOOLEAN BY RET.
*
 LINKAGE SECTION.
 01 COPY COSNAMING-BINDING IN CORBA REPLACING COSNAMING-BINDING
                                                               BY B-VALUE.

 PROCEDURE DIVISION.

* ObjectDirector初期化
     ACCEPT CURRENT-ARG-C FROM ARG-C.
     COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
     PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
        IF ARG-COUNT = 1
           MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
        END-IF
        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        CALL "CORBA-STRING-SET" USING
              CURRENT-ARG-V-VALUE (ARG-COUNT)
              STRING-LENGTH
              TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.

     MOVE FUNCTION LENG(FJ-OM-ORB-ID) TO STRING-LENGTH.
     CALL  "CORBA-STRING-SET" USING
```

```
            TMP-STRING
            STRING-LENGTH
            FJ-OM-ORB-ID.
      CALL "CORBA-ORB-INIT" USING
            CURRENT-ARG-C
            CURRENT-ARG-V
            TMP-STRING
            ENV
            ORB.
      CALL "CORBA-FREE" USING TMP-STRING.

* ルート(CORBA-ORB-OBJECTID-NAMESERVICE)オブジェクトの取得
      MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
            TMP-STRING
            STRING-LENGTH
            CORBA-ORB-OBJECTID-NAMESERVICE.
      CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
            ORB
            TMP-STRING
            ENV
            COS-NAMING.
      CALL "CORBA-FREE" USING TMP-STRING.

* "CONTEXT_A"のネーミングコンテキストの取り出し
      MOVE 1 TO SEQ-LENGTH OF NAME.
      MOVE 1 TO SEQ-MAXIMUM OF NAME.
      CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
            SEQ-MAXIMUM OF NAME
            SEQ-BUFFER OF NAME.

      MOVE "CONTEXT_A" TO TMP-STRING-BUF.
      MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
            IDL-ID OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

      MOVE " " TO TMP-STRING-BUF.
      CALL "CORBA-STRING-SET" USING
            KIND OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

      MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
      MOVE 1 TO ELEMENT-NUMBER.
      MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            NAME-A
            ELEMENT-NUMBER
            NAME-COMPONENT-A.

      CALL "COSNAMING-NAMINGCONTEXT-RESOLVE" USING
            COS-NAMING
            NAME
            ENV
            CONTEXT-OBJ.

* CONTEXT_A下のオブジェクトリファレンスのリストの取り出し
      MOVE  40 TO  HOW-MANY.
      CALL "COSNAMING-NAMINGCONTEXT-LIST" USING
            CONTEXT-OBJ
            HOW-MANY
```

```
           BL
           BI
           ENV.
* CONTEXT_A下の残りのオブジェクトリファレンスの取り出し
     IF NOT BI = CORBA-OBJECT-NIL THEN
            CALL "CORBA-FREE" USING  BL

            CALL "COSNAMING-BINDINGITERATOR-NEXT-N" USING
                   BI
                   HOW-MANY
                   BL2
                   ENV
                   RET

            MOVE CORBA-TRUE TO ROOP-FLAG

            PERFORM UNTIL ROOP-FLAG = CORBA-FALSE
                   CALL "COSNAMING-BINDINGITERATOR-NEXT-ONE" USING
                          BI
                          B
                          ENV
                          RET

* 結果を処理する処理

                   EVALUATE TRUE
                   WHEN CORBA-NO-EXCEPTION OF ENV
                          CONTINUE
                   WHEN CORBA-USER-EXCEPTION OF ENV
                   WHEN CORBA-USER-EXCEPTION OF ENV
                          EXIT PROGRAM
                   END-EVALUATE

                   SET ADDRESS OF B-VALUE TO B
                   IF SEQ-LENGTH OF B-VALUE = 0
                          MOVE CORBA-FALSE TO ROOP-FLAG
                   ELSE
                          CALL "CORBA-FREE" USING B
                   END-IF
            END-PERFORM

            CALL "COSNAMING-BINDINGITERATOR-DESTROY" USING
                   BI
                   ENV
                   RET
            CALL "CORBA-OBJECT-RELEASE" USING
                   BI
                   ENV
       END-IF.
       EXIT PROGRAM.
 END PROGRAM "LIST-TEST".
```

(5)OOCOBOLのプログラミング例

1. オブジェクトリファレンスの検索と同じ方法で、リストを取得したいネーミングコンテキストのオブジェクトリファレンスを取得します。

2. COSNAMING-NAMINGCONTEXT-LISTを使ってリストを取得します。
   結果は、BLで指定されたCOSNAMING-BINDINGLISTにHOW-MANYで指定された個数分設定されます。もし、HOW-MANYで指定された個数より含まれるリストの数が多い場合は、残りの検索のためのオブジェクトCOSNAMING-BINDINGITERATORにオブジェクトリファレンスが設定されます。
   COSNAMING-BINDINGITERATORを使って残りのリストを取得するには、COSNAMING-BINDINGITERATOR-NEXT_ONE

／COSNAMING-BINDINGITERATOR-NEXT_Nを使用します。
本プログラムでは、以下のように処理しています。

1. COSNAMING-BINDINGITERATOR-NEXT_Nを使って最初の40個のオブジェクトリファレンスを検索します。
2. COSNAMING-BINDINGITERATOR-NEXT_ONEで残りのオブジェクトリファレンスのリストを検索します。

```
 IDENTIFICATION          DIVISION.
 PROGRAM-ID.             "CLIENT-MAIN".
*
 ENVIRONMENT             DIVISION.
 CONFIGURATION           SECTION.
 REPOSITORY.
     COPY                CORBA--REP.
     COPY                COSNAMING--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY                CORBA--CONST.
     COPY                COSNAMING--CONST.
     .
*
 DATA                    DIVISION.
 WORKING-STORAGE         SECTION.
 COPY                    CORBA--COPY.
 COPY                    COSNAMING--COPY.
*
 01 ORB                  USAGE OBJECT REFERENCE CORBA-ORB.
 01 BOA                  USAGE OBJECT REFERENCE CORBA-BOA.
 01 OBJ                  USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 NAMING-CONTEXT       USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                 TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT       USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NEW-CONTEXT          USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 BL                   TYPE                   COSNAMING-BINDINGLIST.
 01 BI                   USAGE OBJECT REFERENCE COSNAMING-BINDINGITERATOR.
 01 NAME-ID              USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND            USAGE OBJECT REFERENCE CORBA-STRING.
 01 HOW-MANY             TYPE                   CORBA-UNSIGNED-LONG.
 01 B                    USAGE OBJECT REFERENCE COSNAMING-BINDING.
 01 LEN                  TYPE                   CORBA-UNSIGNED-LONG.
*
 01 EXCEPTION-ID         USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE   PIC   X(50).
 01 API-NAME             PIC   X(50).
*
 01 I                    TYPE                   CORBA-UNSIGNED-LONG.
*
 PROCEDURE               DIVISION.
*
 DECLARATIVES.
*
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::Exception: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
 MAIN SECTION.
```

```
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
                         USING     "Naming"
                                   FJ-OM_ORBID
                         RETURNING ORB.
*
     MOVE "CORBA::ORB::BOA_init" TO API-NAME.
     INVOKE ORB "BOA_INIT"
                         USING     "Naming"
                                   CORBA-BOA_OAID
                         RETURNING BOA.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                         USING     CORBA-ORB-OBJECTID_NAMESERVICE
                         RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                         USING     OBJ
                         RETURNING NAMING-CONTEXT.
*
*
     INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH" USING 1 RETURNING NAME.
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING "CONTEXT_A" RETURNING NAME-ID
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING " " RETURNING NAME-KIND
     INVOKE COSNAMING-NAMECOMPONENT "NEW" RETURNING NAME-COMPONENT
     SET IDL-ID OF NAME-COMPONENT TO NAME-ID
     SET KIND OF NAME-COMPONENT TO NAME-KIND
     INVOKE NAME "SET-VALUE" USING 1 NAME-COMPONENT.
*
     MOVE "CosNaming::NamingContext::resolve" TO API-NAME.
     INVOKE NAMING-CONTEXT "RESOLVE"
                        USING     NAME
                         RETURNING OBJ.
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW" USING OBJ RETURNING NEW-CONTEXT.
*
     MOVE 40 TO HOW-MANY.
     MOVE "CosNaming::NamingContext::list" TO API-NAME.
     INVOKE NEW-CONTEXT "LIST" USING HOW-MANY BL BI.
*
     IF BI NOT = NULL
       MOVE "CosNaming::BindingIterator::next_n" TO API-NAME
       INVOKE BI "NEXT_N" USING HOW-MANY BL
       MOVE "CosNaming::BindingIterator::next_one" TO API-NAME
       SET NAME TO BINDING_NAME OF B
       MOVE SEQ-LENGTH OF NAME TO LEN
       PERFORM UNTIL LEN = 0
         MOVE "CosNaming::BindingIterator::next_one" TO API-NAME
         SET NAME TO BINDING_NAME OF B
         MOVE SEQ-LENGTH OF NAME TO LEN
       END-PERFORM
       MOVE "CosNaming::BindingIterator::destroy" TO API-NAME
       INVOKE BI "DESTROY"
     END-IF.
*
     STOP RUN.
*
 END-MAIN.
*
 END PROGRAM "CLIENT-MAIN".
```

第10章　インタフェースリポジトリサービスのプログラミング

10.1 インタフェースリポジトリサービスで管理するオブジェクトの種類

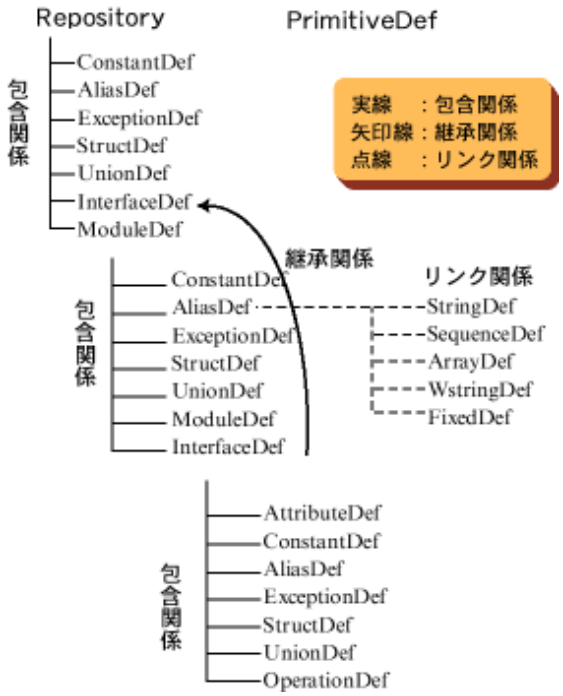
インタフェースリポジトリサービスは、IDLで定義されたモジュール宣言、インタフェース宣言、オペレーション宣言、定数宣言、型宣言などの宣言に対応してModuleDefオブジェクト、InterfaceDefオブジェクト、OperationDefオブジェクト、ConstantDefオブジェクト、AliasDefオブジェクトなどのオブジェクトをインタフェースリポジトリサービス内に管理・格納するサービスです。
以下にIDL宣言に対応するオブジェクトを示します。
なお、IDL宣言とは別に、インタフェースリポジトリサービスのルートオブジェクトとしてRepositoryオブジェクト、long,shortなどの基本的なデータ型を表現するオブジェクトとしてPrimitiveDefオブジェクトがあります。

IDL宣言	オブジェクト
――――	Repositoryオブジェクト
――――	PrimitiveDefオブジェクト
モジュール宣言	ModuleDefオブジェクト
インタフェース宣言	InterfaceDefオブジェクト
オペレーション宣言	OperationDefオブジェクト
属性宣言	AttributeDefオブジェクト
定数宣言	ConstantDefオブジェクト
Typedef宣言	AliasDefオブジェクト
文字列型宣言	StringDefオブジェクト
ワイド文字列型宣言	WstringDefオブジェクト
列挙型宣言	EnumDefオブジェクト
シーケンス型宣言	SequenceDefオブジェクト
構造体宣言	StructDefオブジェクト
共用体宣言	UnionDefオブジェクト
固定小数点型宣言(注)	FixedDefオブジェクト
配列宣言	ArrayDefオブジェクト
例外宣言	ExceptionDefオブジェクト

注) 固定小数点型宣言は、COBOLマッピングだけで使用可能です。

10.2 インタフェースリポジトリオブジェクトの包含／継承関係

インタフェースリポジトリサービスで管理されるオブジェクトは、IDL定義と対応して包含関係が形成されます。他のリポジトリオブジェクトを包含できるリポジトリオブジェクトは、Repositoryオブジェクト、ModuleDefオブジェクト、InterfaceDefオブジェクトです。InterfaceDefオブジェクトは他のInterfaceDefオブジェクトを継承できます。
また、StringDefオブジェクト、ArrayDefオブジェクト、SequenceDefオブジェクト、WstringDefオブジェクト、FixedDefオブジェクトは、AliasDefオブジェクトからリンクされることにより、包含関係が形成されます。
包含関係を以下の図に示します。



10.3 インタフェースリポジトリサービスインタフェース

インタフェースリポジトリサービスは、以下のインタフェースを提供しています。

10.3.1 インタフェースリポジトリサービスが提供するインタフェース

```
module CORBA{
      interface IRObject;
      interface Contained    : IRObject{};
      interface Container    : IRObject{};
      interface IDLType      : IRObject{};
      interface Repository   : Container{};
      interface ModuleDef    : Container, Contained{};
      interface ConstantDef  : Contained{};
      interface TypedefDef   : Contained, IDLType{};
      interface StructDef    : TypedefDef{};
      interface UnionDef     : TypedefDef{};
      interface EnumDef      : TypedefDef{};
      interface AliasDef     : TypedefDef{};
      interface PrimitiveDef : IDLType{};
      interface StringDef    : IDLType{};
      interface WstringDef   : IDLType{};
      interface SequenceDef  : IDLType{};
      interface ArrayDef     : IDLType{};
      interface FixedDef     : IDLType{};
      interface ExceptionDef : Contained{};
      interface AttributeDef : Contained{};
      interface OperationDef : Contained{};
      interface InterfaceDef : Contained, Container, IDLType{};
   // interface TypeCode;

      typedef string    Identifier;
      typedef string    ScopedName;
      typedef string    RepositoryId;

      enum DefinitionKind{
          dk_none,dk_all,dk_Attribute,dk_Constant,dk_Exception,
```

```
        dk_Interface,dk_Module,dk_Operation,dk_Typedef,
        dk_Alias,dk_Struct,dk_Union,dk_Enum,dk_Primitive,
        dk_String,dk_Sequence,dk_Array,dk_Repository,
      dk_Wstring,dk_Fixed
    };

    interface IRObject{
        // read interface
        readonly attribute DefinitionKind   def_kind;

        // write interface
        void destroy();
    };

    typedef string    VersionSpec;

    interface Contained;
    interface Repository;
    interface Container;
  // interface TypeCode;

    interface Contained:IRObject{
        // read/write interface
        attribute RepositoryId      id;
        attribute Identifier        name;
        attribute VersionSpec       version;

        // read interface
        readonly attribute Container         defined_in;
        readonly attribute ScopedName        absolute_name;
        readonly attribute Repository        containing_repository;

        struct Description{
            DefinitionKind    kind;
            any               value;
        };

        Description describe();

        // write interface
        void move(
            in Container       new_container,
            in Identifier      new_name,
            in VersionSpec     new_version
        );
    };

    interface ModuleDef;
    interface ConstantDef;
    interface IDLType;
    interface StructDef;
    interface UnionDef;
    interface EnumDef;
    interface AliasDef;
    interface InterfaceDef;
    interface ExceptionDef;

    typedef sequence<InterfaceDef>InterfaceDefSeq;

    typedef sequence<Contained>ContainedSeq;

    struct StructMember{
        Identifier name;
```

```
    TypeCode   type;
    IDLType    type_def;
};

typedef sequence<StructMember>StructMemberSeq;

struct UnionMember{
    Identifier name;
    any        label;
    TypeCode   type;
    IDLType    type_def;
};

typedef sequence<UnionMember>UnionMemberSeq;

typedef sequence<Identifier>EnumMemberSeq;

interface Container:IRObject{
    // read interface

    Contained lookup(in ScopedName search_name);

    ContainedSeq contents(
        in DefinitionKind   limit_type,
        in boolean          exclude_inherited
    );

    ContainedSeq lookup_name(
        in Identifier       search_name,
        in long             levels_to_search,
        in DefinitionKind   limit_type,
        in boolean          exclude_inherited
    );

    struct Description{
        Contained           contained_object;
        DefinitionKind      kind;
        any                 value;
   };

    typedef sequence<Description>DescriptionSeq;

    DescriptionSeq describe_contents(
        in DefinitionKind   limit_type,
        in boolean          exclude_inherited,
        in long             max_returned_objs
    );

    // write interface

    ModuleDef create_module(
        in RepositoryId     id,
        in Identifier       name,
        in VersionSpec      version
    );

    ConstantDef create_constant(
        in RepositoryId     id,
        in Identifier       name,
        in VersionSpec      version,
        in IDLType          type,
        in any              value
    );
```

```
    ExceptionDef create_exception(
        in RepositoryId     id,
        in Identifier       name,
        in VersionSpec      version,
        in StructMemberSeq  members
    );

    StructDef create_struct(
        in RepositoryId     id,
        in Identifier       name,
        in VersionSpec      version,
        in StructMemberSeq  members
    );

    UnionDef create_union(
        in RepositoryId     id,
        in Identifier       name,
        in VersionSpec      version,
        in IDLType          discriminator_type,
        in UnionMemberSeq   members
    );

    EnumDef create_enum(
        in RepositoryId     id,
        in Identifier       name,
        in VersionSpec      version,
        in EnumMemberSeq    members
    );

    AliasDef create_alias(
        in RepositoryId     id,
        in Identifier       name,
        in VersionSpec      version,
        in IDLType          original_type
    );

    InterfaceDef create_interface(
        in RepositoryId     id,
        in Identifier       name,
        in VersionSpec      version,
        in InterfaceDefSeq  base_interfaces
    );
};

interface IDLType:IRObject{
    readonly attribute TypeCode type;
};

enum PrimitiveKind{
    pk_null,pk_void,pk_short,pk_long,pk_ushort,pk_ulong,
    pk_float,pk_double,pk_boolean,pk_char,pk_octet,
    pk_any,pk_TypeCode,pk_Principal,pk_string,pk_objref,
    pk_longlong,pk_ulonglong,pk_longdouble,pk_wchar,pk_wstring
};

interface PrimitiveDef;
interface StringDef;
interface WstringDef;
interface SequenceDef;
interface ArrayDef;
interface FixedDef;
interface Repository:Container{
```

```
    // read interface
    Contained lookup_id(in RepositoryId search_id);
    PrimitiveDef get_primitive(in PrimitiveKind kind);

    // write interface
    StringDef create_string(in unsigned long bound);

    WstringDef create_wstring(in unsigned long bound);

    SequenceDef create_sequence(
        in unsigned long bound,
        in IDLType element_type
    );

    ArrayDef create_array(
        in unsigned long length,
        in IDLType element_type
    );

    FixedDef create_fixed(
        in unsigned short digits,
        in short scale
    );

    void load();
    void destroy_rep();
};

interface ModuleDef:Container,Contained{
};

struct ModuleDescription{
    Identifier         name;
    RepositoryId       id;
    RepositoryId       defined_in;
    VersionSpec        version;
};

interface ConstantDef:Contained{
    readonly attribute TypeCode type;
             attribute IDLType  type_def;
             attribute any      value;
};

struct ConstantDescription{
    Identifier         name;
    RepositoryId       id;
    RepositoryId       defined_in;
    VersionSpec        version;
    TypeCode           type;
    any                value;
};

interface TypedefDef:Contained,IDLType{
};

struct TypeDescription{
    Identifier         name;
    RepositoryId       id;
    RepositoryId       defined_in;
    VersionSpec        version;
    TypeCode           type;
```

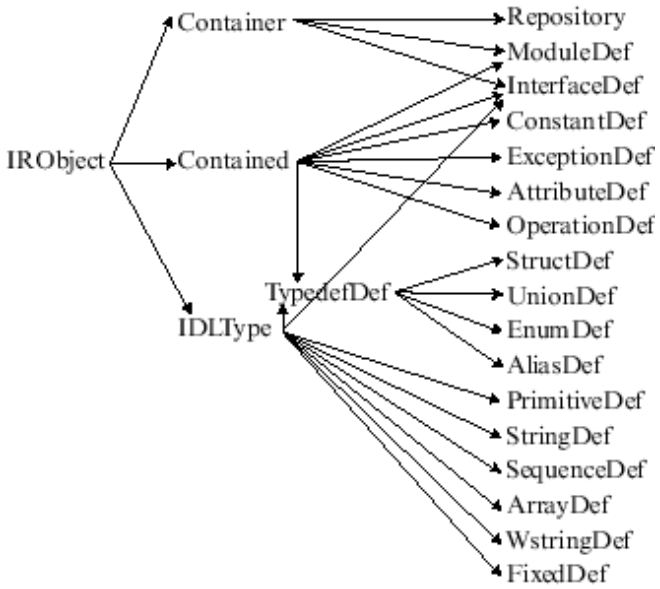
```
 };

interface StructDef:TypedefDef{
            attribute StructMemberSeq members;
};

interface UnionDef:TypedefDef{
    readonly attribute TypeCode discriminator_type;
            attribute IDLType  discriminator_type_def;
            attribute UnionMemberSeq members;
};

interface EnumDef:TypedefDef{
            attribute EnumMemberSeq members;
};

interface AliasDef:TypedefDef{
            attribute IDLType original_type_def;
};

interface PrimitiveDef:IDLType{
    readonly attribute PrimitiveKind kind;
};

interface StringDef:IDLType{
            attribute unsigned long bound;
};

interface WstringDef:IDLType{
            attribute unsigned long bound;
};

interface SequenceDef:IDLType{
            attribute unsigned long bound;
    readonly attribute TypeCode element_type;
            attribute IDLType  element_type_def;
};

interface ArrayDef:IDLType{
            attribute unsigned long length;
    readonly attribute TypeCode element_type;
            attribute IDLType  element_type_def;
};

interface ExceptionDef:Contained{
    readonly attribute TypeCode type;
            attribute StructMemberSeq members;
};

struct ExceptionDescription{
    Identifier      name;
    RepositoryId    id;
    RepositoryId    defined_in;
    VersionSpec     version;
    TypeCode        type;
};

enum AttributeMode{ATTR_NORMAL,ATTR_READONLY};

interface AttributeDef:Contained{
    readonly attribute TypeCode type;
            attribute IDLType  type_def;
            attribute AttributeMode mode;
```

```
};

struct AttributeDescription{
    Identifier      name;
    RepositoryId    id;
    RepositoryId    defined_in;
    VersionSpec     version;
    TypeCode        type;
    AttributeMode   mode;
};

enum OperationMode{OP_NORMAL,OP_ONEWAY};

enum ParameterMode{PARAM_IN,PARAM_OUT,PARAM_INOUT};

struct ParameterDescription{
    Identifier      name;
    TypeCode        type;
    IDLType         type_def;
    ParameterMode   mode;
};

typedef sequence<ParameterDescription>ParDescriptionSeq;
typedef Identifier ContextIdentifier;
typedef sequence<ContextIdentifier>ContextIdSeq;

typedef sequence<ExceptionDef>ExceptionDefSeq;
typedef sequence<ExceptionDescription>ExcDescriptionSeq;

interface OperationDef:Contained{
    readonly attribute TypeCode result;
             attribute IDLType  result_def;
             attribute ParDescriptionSeq params;
             attribute OperationMode mode;
             attribute ContextIdSeq contexts;
             attribute ExceptionDefSeq exceptions;
};

struct OperationDescription{
    Identifier      name;
    RepositoryId    id;
    RepositoryId    defined_in;
    VersionSpec     version;
    TypeCode        result;
    OperationMode   mode;
    ContextIdSeq    contexts;
    ParDescriptionSeq parameters;
    ExcDescriptionSeq exceptions;
};

typedef sequence<RepositoryId>RepositoryIdSeq;
typedef sequence<OperationDescription>OpDescriptionSeq;
typedef sequence<AttributeDescription>AttrDescriptionSeq;

interface InterfaceDef:Container,Contained,IDLType{
    // read/write interface
    attribute InterfaceDefSeq base_interfaces;

    // read interface
    boolean is_a(in RepositoryId interface_id);   // OOCOBOLで使用する場合、メソッド名はIDL-IS_Aです。

    struct FullInterfaceDescription{
        Identifier      name;
```

```
            RepositoryId    id;
            RepositoryId    defined_in;
            VersionSpec     version;
            OpDescriptionSeq operations;
            AttrDescriptionSeq attributes;
            RepositoryIdSeq base_interfaces;
            TypeCode        type;
        };
        FullInterfaceDescription describe_interface();

        // write interface

        AttributeDef create_attribute(
            in RepositoryId id,
            in Identifier name,
            in VersionSpec version,
            in IDLType type,
            in AttributeMode mode
        );

        OperationDef create_operation(
            in RepositoryId id,
            in Identifier name,
            in VersionSpec version,
            in IDLType result,
            in OperationMode mode,
            in ParDescriptionSeq params,
            in ExceptionDefSeq exceptions,
            in ContextIdSeq contexts
        );
    };

    struct InterfaceDescription{
        Identifier      name;
        RepositoryId    id;
        RepositoryId    defined_in;
        VersionSpec     version;
        RepositoryIdSeq base_interfaces;
    };

    interface InterfaceRep{
        // write interface
        Repository   create_repository();
        boolean      create_primitives();
    };
};
```

10.3.2 インタフェースの継承関係

インタフェースの継承関係を以下の図に示します。



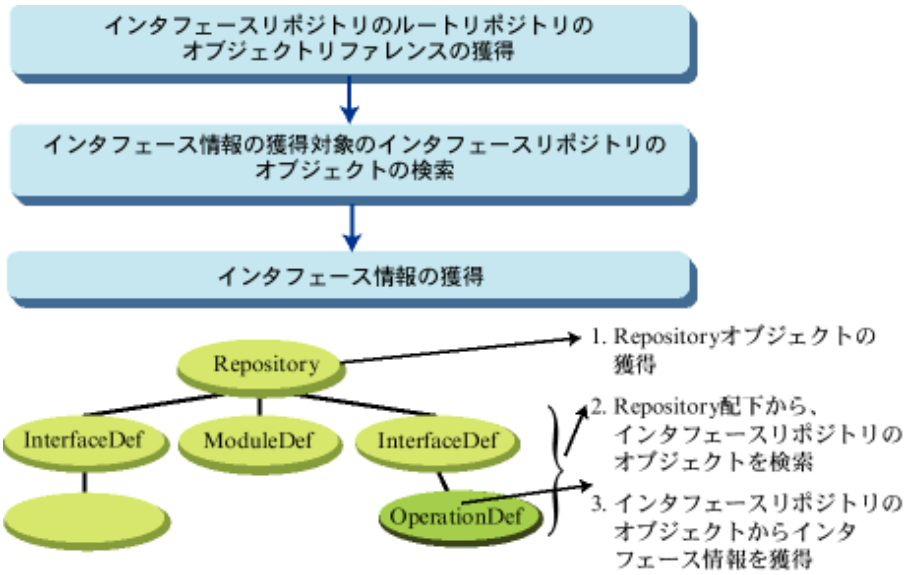
インタフェースリポジトリサービスで提供しているインタフェースの機能説明を以下の表に示します。

インタフェース	オブジェクト	機能説明
IRObject	def_kind	リポジトリオブジェクトの種類を返します。
Contained	id	リポジトリオブジェクトのリポジトリIDを返します。
	name	リポジトリオブジェクトの単純名を返します。
	version	リポジトリオブジェクトのバージョンを返します。
	defined_in	リポジトリオブジェクトを含有するリポジトリオブジェクトを返します。
	absolute_name	リポジトリオブジェクトのスコープ名を返します。
	containing_repository	Repositoryオブジェクトを返します。
	describe	リポジトリオブジェクトのインタフェース情報を返します。
Container	lookup	リポジトリオブジェクトをスコープ名で検索します。
	contents	含有するリポジトリオブジェクトをリスト形式で返します。
	lookup_name	リポジトリオブジェクトを単純名で検索します。
	describe_contents	含有するリポジトリオブジェクトの情報をリスト形式で返します。
IDLType	type	IDLTypeオブジェクトの型のタイプコードを返します。
Repository	lookup_id	リポジトリオブジェクトをリポジトリIDで検索します。
	get_primitive	long,shortなどのプリミティブなデータ型を表すPrimitiveDefオブジェクトのオブジェクトリファレンスを返します。
ConstantDef	type	ConstantDefオブジェクトの型のタイプコードを返します。
	type_def	ConstantDefオブジェクトの型を表現するオブジェクトを返します。
	value	ConstantDefオブジェクトの定数値を返します。
StructDef	members	StructDefオブジェクトのメンバのリストを返します。
UnionDef	discriminator_type	UnionDefオブジェクトの弁別子の型のタイプコードを返します。
	discriminator_type_def	UnionDefオブジェクトの弁別子の型のオブジェクトを返します。

インタフェース	オブジェクト	機能説明
	members	UnionDefオブジェクトのメンバのリストを返します。
EnumDef	members	EnumDefオブジェクトのメンバのリストを返します。
PrimitiveDef	kind	PrimitiveDefオブジェクトの種類を定義した実際の型のオブジェクト
AliasDef	original_type_def	AliasDefオブジェクトのtypedefを定義した実際の型のオブジェクト
StringDef	bound	StringDefオブジェクトの最大文字数を返します。
SequenceDef	bound	SequenceDefオブジェクトの最大文字数を返します。
	type	SequenceDefオブジェクトの要素の型のタイプコードを返します。
	type_def	SequenceDefオブジェクトの要素の型のオブジェクトを返します。
ArrayDef	bound	ArrayDefオブジェクトの配列の要素数を返します。
	type	ArrayDefオブジェクトの配列要素の型のタイプコードを返します。
	type_def	ArrayDefオブジェクトの配列要素の型のオブジェクトを返します。
WstringDef	bound	WstringDefオブジェクトの最大文字数を返します。
FixedDef	digits	FixedDefオブジェクトの桁数を返します。
	scale	FixedDefオブジェクトのスケールを返します。
InterfaceDef	base_interfaces	InterfaceDefオブジェクトが継承するInterfaceDefオブジェクトのリストを返します。
	describe_interface	InterfaceDefオブジェクトの詳細なインタフェース情報を返します。describeオペレーションで返される情報に加えて、オペレーション情報、アトリビュート情報が返されます。
	is_a	InterfaceDefオブジェクトが他のInterfaceDefオブジェクトを継承しているかどうかを判定します。
OperationDef	result	OperationDefオブジェクトの戻り値のタイプコードを返します。
	result_def	OperationDefオブジェクトの戻り値を表現するオブジェクトを返します。
	params	OperationDefオブジェクトのパラメタリストを返します。
	mode	OperationDefオブジェクトがonewayかどうかを示すモードを返します。
	contexts	OperationDefオブジェクトのコンテキストプロパティ名のリストを返します。
	exceptions	OperationDefオブジェクトのExceptionDefオブジェクトのリストを返します。
AttributeDef	type	AttributeDefオブジェクトの型のタイプコードを返します。
	type_def	AttributeDefオブジェクトの型を表現するオブジェクト。
	mode	AttributeDefオブジェクトの属性がreadonlyかどうかを示す属性を返します。
ExceptionDef	type	ExceptionDefオブジェクトの例外定義の型のタイプコードを返します。
	members	ExceptionDefオブジェクトの例外定義の構造体のメンバのリストを返します

10.4 インタフェースリポジトリサービスのプログラミング

インタフェースリポジトリサービスより、サーバアプリケーションの提供する関数のパラメタ、復帰値などのインタフェース情報を獲得する場合のプログラミングの流れを以下に示します。



10.4.1 ルートリポジトリのオブジェクトリファレンスを獲得

インタフェースリポジトリのルートリポジトリのオブジェクトリファレンスを獲得します。
オブジェクトリファレンスを取り出すresolve_initial_referencesメソッドによりルートリポジトリ(Repositoryオブジェクト)のオブジェクトリファレンスを取り出します。

10.4.2 インタフェースリポジトリのオブジェクトの検索

インタフェース情報の獲得対象のインタフェースリポジトリのオブジェクトを検索します。
以下のメソッドにより、インタフェースリポジトリのオブジェクトを検索します。

インタフェース名	メソッド名	キー	機能
Repository	lookup_id	リポジトリID	指定されたリポジトリIDのオブジェクトのオブジェクトリファレンスを返します。
Container	lookup	スコープ名	指定されたスコープ名のオブジェクトのオブジェクトリファレンスを返します。
	lookup_name	単純名	指定された単純名のオブジェクトのオブジェクトリファレンスをリスト形式で返します。
	contents	―	含有するオブジェクトのオブジェクトリファレンスをリスト形式で返します。

10.4.3 インタフェース情報の獲得

インタフェース情報を獲得します。
以下のメソッドにより、インタフェースリポジトリのオブジェクトのインタフェース情報を獲得します。

インタフェース名	メソッド名	機能
Contained	describe	指定されたオブジェクトのインタフェース情報を返します。
Container	describe_contents	指定されたオブジェクトが含有するオブジェクトのインタフェース情報をリスト形式で返します。

“10.4.2 インタフェースリポジトリのオブジェクトの検索”で獲得したオブジェクトリファレンスを指定して、インタフェース情報を獲得します。

10.4.4 インタフェースリポジトリサービスのプログラミング例

注意

Windows64 Solaris64 Linux64

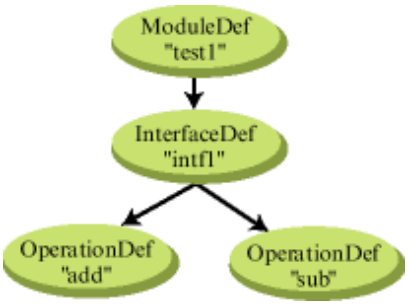
OOCOBOLは、Windows(R)クライアントアプリケーションで使用できます。

例1:OperationDefオブジェクトの情報の獲得

以下のIDL定義に対するOperationDefオブジェクトのパラメタ情報を獲得するプログラミングについて説明します。

```
module test1 {
    interface intf1  {
        long add(in long a,  in long b) ;
        long sub(in long c,  in long d) ;
    };
};
```

上記のIDL定義の場合、以下のオブジェクトがインタフェースリポジトリサービスに格納されます。



(1)インタフェースリポジトリのオブジェクトリファレンスの獲得

システム提供サービスのオブジェクトリファレンスの取出しメソッドresolve_initial_references()により、ルートリポジトリのオブジェクトリファレンスを取り出します。

C言語

```
CORBA_Repository           intf_rep;
/* インタフェースリポジトリのオブジェクトリファレンス獲得 */
intf_rep = CORBA_ORB_resolve_initial_references(
              orb,CORBA_ORB_ObjectId_InterfaceRepository, &env);
```

C++言語

```
CORBA::Object_ptr      obj;
CORBA::Repository_ptr  intf_rep;

// インタフェースリポジトリのオブジェクトリファレンス獲得
obj = orb->resolve_initial_references(
      CORBA_ORB_ObjectId_InterfaceRepository, *env );

// CORBA::Repositoryクラスに変換
intf_rep = CORBA::Repository::_narrow( obj );
```

Java言語

```
org.omg.CORBA.Object       Obj;
org.omg.CORBA.Repository   Rep;
// インタフェースリポジトリのオブジェクトリファレンス獲得
Obj = Orb.resolve_initial_references( "InterfaceRepository" );

// RepositoryHelperクラスに変換
Rep = org.omg.CORBA.RepositoryHelper.narrow( Obj );
```

COBOL

```
*########## インタフェースリポジトリのオブジェクトリファレンス獲得 ##########
   MOVE FUNCTION LENG (CORBA-ORB-OBJECTID-INTFREP) TO STRING-LENGTH.
   CALL "CORBA-STRING-SET" USING
                TEMP-BUF
                STRING-LENGTH
                CORBA-ORB-OBJECTID-INTFREP.
   CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
                ORB
                TEMP-BUF
                ENV
                OBJ.
```

OOCOBOL

```
  01 ORB     OBJECT REFERENCE CORBA-ORB.
  01 OBJ     OBJECT REFERENCE CORBA-OBJECT.
  01 REP     OBJECT REFERENCE CORBA-REPOSITORY.

    INVOKE CORBA "ORB_INIT" USING APL-NAME FJ-OM_ORBID RETURNING ORB.
 *######## インタフェースリポジトリのオブジェクトリファレンス獲得
    INVOKE ORB "RESOLVE_INITIAL_REFERENCES" USING CORBA-OBJECTID_INTERFACERE-001
                                          RETURNING OBJ.
 *######## REPOSITORYクラスに変換
    INVOKE CORBA-REPOSITORY "NARROW" USING OBJ
                                     RETURNING REP.
```

(2)OperationDefオブジェクトのオブジェクトリファレンスの獲得

対象のOperationDefオブジェクトのリポジトリIDである“IDL:test1/intf1/add:1.0”とルートリポジトリのオブジェクトリファレンスを指定してlookup_id()を発行すると、OperationDefオブジェクトのオブジェクトリファレンスが獲得できます。

C言語

```
CORBA_Contained intfobj;
/* オブジェクトの検索 */
intfobj = CORBA_Repository_lookup_id(intf_rep, "IDL:test1/intf1/add:1.0", &env);
```

C++言語

```
CORBA::Object_ptr intfobj;
// オブジェクトの検索
intfobj = intf_rep->lookup_id( "IDL:test1/intf1/add:1.0", *env );
```

Java言語

```
org.omg.CORBA.Object      obj;

// オブジェクトの検索
obj = Rep.lookup_id( "IDL:test1/intf1/add:1.0" );
```

COBOL

```
*########## インタフェースリポジトリのオブジェクト検索 ##########
     MOVE FUNCTION LENG ("IDL:test1/intf1/add:1.0") TO STRING-LENGTH
     MOVE "IDL:test1/intf1/add:1.0" TO IN-BUF
     CALL "CORBA-STRING-SET" USING
               REPOSITORYID
               STRING-LENGTH
               IN-BUF.
     CALL "CORBA-REPOSITORY-LOOKUP-ID" USING
               OBJ
               REPOSITORYID
               ENV
               INTF-INTF.
```

OOCOBOL

```
 01 R-OBJ    OBJECT REFERENCE CORBA-OBJECT.

*######## インタフェースリポジトリのオブジェクト検索
     INVOKE REP "LOOKUP_ID" USING "IDL:test1/intf1/add:1.0"
                            RETURNING R-OBJ.
```

(3)パラメタ情報の獲得

OperationDefオブジェクトのオブジェクトリファレンスを指定してdescribeメソッドを呼び出すことにより、サーバアプリケーションが持っているメソッドの情報をインタフェースリポジトリから検索します。その結果、パラメタの情報(パラメタの名前、個数、パラメタの型など)が返されます。

C言語

```
CORBA_Contained_Description       *ope_des;
CORBA_OperationDescription        *o;
CORBA_TCKind  kind;
CORBA_ParDescriptionSeq           *parameters;
CORBA_ParameterDescription        *prmp;

/* OperationDefオブジェクトのインタフェース情報の獲得  */
ope_des = CORBA_OperationDef_describe(intfobj,&env);
/* OperationDefオブジェクト固有情報構造体の取り出し    */
o = (CORBA_OperationDescription *)ope_des->value._value;

parameters = &(o->parameters);  /* パラメタシーケンス情報を参照  */
/* パラメタ数の参照 */
printf("Parameter number = %d¥n", parameters->_length );
for(k=0,prmp=parameters->_buffer; k<parameters->_length; k++){
      /* パラメタの名前の参照             */
      printf("Parameter Identifier(%d) = %s¥n", k, prmp[k].name);
      /* パラメタの型の参照               */
      kind = CORBA_TypeCode_kind(prmp[k].type,&env);
      printf("TypeCode_kind(%d) = %d¥n",k,kind);
```

```
      /* パラメタのモード(IN/OUT/INOUT)  */
      printf("Parameter_mode(%d) = %d¥n",k, prmp[k].mode);
}
```

C++言語

```
CORBA::OperationDef_ptr opef;
CORBA::Contained::Description  *desc;
CORBA::OperationDescription    *c;

// OperationDefオブジェクトのインタフェース情報の獲得
opef = CORBA::OperationDef::_narrow(intfobj);
desc   = opef->describe(*env);
// OperationDefオブジェクト固有情報構造体の取り出し
CORBA::Any *opeany = (CORBA::Any*)(desc->value);
c = (CORBA::OperationDescription *)opeany->value();
// パラメタシーケンス情報を参照
CORBA::ParDescriptionSeq *parseq = (CORBA::ParDescriptionSeq*)c->parameters;
// パラメタ数の参照
printf("Parameter number = %d¥n", parseq->length() );
for( int n = 0;n < parseq->length(); n++) {
      CORBA::ParameterDescription *parades =
                (CORBA::ParameterDescription*)(*parseq)[m];
      // パラメタの名前の参照
      printf("para name = %s ¥n", (char*)parades->name);
      // パラメタの型の参照
      CORBA::TypeCode
      *typecode2 = (CORBA::TypeCode *)(parades->type);
      kind = typecode2->kind(*env);
      printf("TypeCode_kind = %d¥n",kind);

      // パラメタのモード(IN/OUT/INOUT)
      printf("para mode = %d ¥n", parades->mode);
}
```

Java言語

```
org.omg.CORBA.OperationDef                  opef;
org.omg.CORBA.ContainedPackage.Description  desc;
org.omg.CORBA.OperationDescription          c;
org.omg.CORBA.ParameterDescription          paraseq[];

// OperationDefオブジェクトのインタフェース情報の獲得
opef = org.omg.CORBA.OperationDefHelper.narrow( obj );
desc = opef.describe();
// OperationDefオブジェクト固有情報構造体の取り出し
org.omg.CORBA.Any opeany = desc.value;
c = ( org.omg.CORBA.OperationDescription )
          org.omg.CORBA.OperationDescriptionHelper.extract( opeany );
// パラメタシーケンス情報を参照
paraseq = c.parameters;
// パラメタ数の参照
inoutcc.output( "Parameter number = " + paraseq.length );
for( i = 0; i < paraseq.length; i++ ) {
    parades = paraseq[i];
    // パラメタの名前の参照
    inoutcc.output( "para name = " + parades.name );
    // パラメタの型の参照
    typecode = parades.type;
    kind = typecode.kind().value();
    inoutcc.output( "TypeCode_kind = " + kind );
```

```
    // パラメタのモード
    inoutcc.output( "para mode = " + parades.mode.value() );
}
```

COBOL

```
*########## OperationDefオブジェクトのインタフェース情報の獲得 ##########
   CALL "CORBA-OPERATIONDEF-DESCRIBE" USING
                OBJ
                ENV
                LINK-BUF.
*--- OperationDefオブジェクト固有情報構造体の取り出し ---
   SET ADDRESS OF LINK-CONDES TO LINK-BUF.
   MOVE IDL-VALUE OF LINK-CONDES TO TMP-ANY.
   MOVE ANY-VALUE OF TMP-ANY TO TEMP-BUF.
   SET ADDRESS OF TMP-OPR TO TEMP-BUF.
   MOVE FUNCTION ADDR(PARAMETERS OF TMP-OPR) TO TEMP-BUF.
   SET ADDRESS OF PARDESSEQ TO TEMP-BUF.

*---- パラメタシーケンス情報を参照 ---
*---- パラメタ数の参照 ---
   IF SEQ-LENGTH OF PARDESSEQ = 0
        DISPLAY "PARA-LENGTH = 0"
   END-IF.
   DISPLAY "PARAMETER NUMBER = "  SEQ-LENGTH OF PARDESSEQ.
   PERFORM TEST BEFORE
           VARYING CNT FROM 1 BY 1
               UNTIL CNT > SEQ-LENGTH OF PARDESSEQ
           CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
                 TEMP-BUF
                 CNT
                 ELEMENT-TYPE
           SET  ADDRESS OF TMP-PARA TO ELEMENT-TYPE
*----       パラメタの名前の参照     ----
           MOVE FUNCTION LENG (DSP-BUF) TO STRING-LENGTH
           CALL "CORBA-STRING-GET" USING
               NAME OF TMP-PARA
               STRING-LENGTH
               DSP-BUF
           DISPLAY "NAME=" DSP-BUF
*----       パラメタの型の参照     ----
           CALL "CORBA-TYPECODE-KIND" USING
                IDL-TYPE OF TMP-PARA
                ENV
                TMP-KIND
           DISPLAY "TYPECODE KIND=" TMP-KIND
*----       パラメタのモード(IN/OUT/INOUT)    ----
           DISPLAY "PARA MODE=" IDL-MODE OF TMP-PARA
   END-PERFORM.
```

OOCOBOL

```
   01 OPEF          OBJECT REFERENCE CORBA-OPERATIONDEF.
   01 DESC          OBJECT REFERENCE CORBA-CONTAINED-DESCRIPTION.
   01 OPE-ANY       OBJECT REFERENCE CORBA-ANY.
   01 C             OBJECT REFERENCE CORBA-OPERATIONDESCRIPTION.
   01 PARAMSEQ      TYPE CORBA-PARDESCRIPTIONSEQ.
   01 PARADES       OBJECT REFERENCE CORBA-PARAMETERDESCRIPTION.
   01 STR           OBJECT REFERENCE CORBA-STRING.
   01 TYPECODE      OBJECT REFERENCE CORBA-TYPECODE.
```

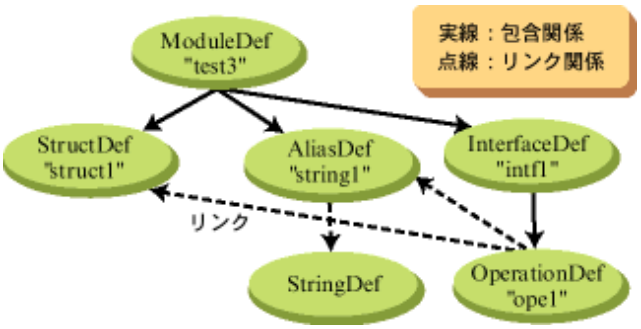
```
   01 PARA-NAME     PIC X(40).
   01 SEQ-LEN       TYPE CORBA-LONG.
   01 N             TYPE CORBA-UNSIGNED-LONG.
   01 TCK           TYPE CORBA-TCKIND.
   01 PARA-MODE     TYPE CORBA-PARAMETERMODE.

*######## OperationDefオブジェクトのインタフェース情報の獲得
     INVOKE CORBA-OPERATIONDEF "NARROW" USING INTF-OBJ
                                        RETURNING OPEF
     INVOKE OPEF "DESCRIBE" RETURNING DESC.
     SET OPE-ANY TO IDL-VALUE OF DESC.
*
     INVOKE CORBA-OPERATIONDESCRIPTION--H "EXTRACT" USING OPE-ANY
                                                    RETURNING C.
*######## パラメタシーケンス情報を参照
     SET PARAMSEQ TO PARAMETERS OF C.
*######## パラメタ数の参照
     MOVE SEQ-LENGTH OF PARAMSEQ TO SEQ-LEN.
     DISPLAY "PARAMETER NUMBER = " SEQ-LEN.
     PERFORM VARYING  N FROM 1 BY 1 UNTIL SEQ-LEN < N
        INVOKE PARAMSEQ "GET-VALUE" USING N
                                    RETURNING  PARADES
*######## パラメタの名前の参照
        SET STR TO NAME OF PARADES
        INVOKE STR "GET-VALUE" RETURNING  PARA-NAME
        DISPLAY " PARA NAME = " PARA-NAME
*######## パラメタの型の参照
        SET TYPECODE TO IDL-TYPE OF PARADES
        INVOKE TYPECODE "KIND" RETURNING TCK
        DISPLAY "TYPECODE_KIND = " TCK
*######## パラメタのモード
        MOVE IDL-MODE OF PARADES TO PARA-MODE
        DISPLAY "PARA MODE = " PARA-MODE
     END-PERFORM.
```

例2:StructDefオブジェクト、AliasDefオブジェクトの情報の獲得

以下のIDL定義に対するStructDefオブジェクト(構造体定義)、AliasDefオブジェクト(typedef定義情報)の情報を獲得するプログラミングについて説明します。

```
module test3 {
    struct  struct1{
        long  para1;
        short para2;
    };
    typedef  string<50>  string1;
    interface intf1  {
        long  ope1(in  struct1  struct1_data, in string1 string1_data) ;
    };
};
```

上記のIDL定義の場合、以下のオブジェクトがインタフェースリポジトリサービスに格納されます。



(1)オブジェクトリファレンスの獲得

OperationDefオブジェクトのパラメタ情報より、StructDefオブジェクト、AliasDefオブジェクトのオブジェクトリファレンスを獲得します。

C言語

```
CORBA_Repository                intf_rep;
CORBA_Contained                 intfobj;
CORBA_Contained_Description     *ope_des;
CORBA_OperationDescription      *o;
CORBA_TCKind                    kind;
CORBA_ParDescriptionSeq         *parameters;
CORBA_ParameterDescription      *prmp;
CORBA_StructDef                 StructObj;
CORBA_AliasDef                  AliasObj;

/* インタフェースリポジトリのオブジェクトリファレンス獲得 */
intf_rep = CORBA_ORB_resolve_initial_references(
              orb,CORBA_ORB_ObjectId_InterfaceRepository, &env);

/* オブジェクトの検索 */
intfobj = CORBA_Repository_lookup_id(intf_rep, "IDL:test1/intf1/ope1:1.0", &env);

/* OperationDefオブジェクトのインタフェース情報の獲得  */
ope_des = CORBA_OperationDef_describe(intfobj,&env);
exception_check("CORBA_OperationDef_describe");
/* OperationDefオブジェクト固有情報構造体の取り出し    */
o = (CORBA_OperationDescription *)ope_des->value._value;

parameters = &(o->parameters);  /* パラメタシーケンス情報を参照  */
for(k=0,prmp=parameters->_buffer; k<parameters->_length; k++){
      /* パラメタの型の参照              */
      kind = CORBA_IRObject__get_def_kind(prmp[k].type_def,&env);
      if(kind == CORBA_dk_Struct)
           /* StructObjにStructDefのオブジェクトリファレンス設定 */
           StructObj = (CORBA_StructDef)prmp[k].type_def;
      else if(kind == CORBA_dk_Alias)
             /* AliasObjにAliasDefのオブジェクトリファレンス設定 */
             AliasObj  = (CORBA_AliasDef)prmp[k].type_def;
}
```

C++言語

```
CORBA::Object_ptr               obj;
CORBA::Repository_ptr           intf_rep;
CORBA::Contained_ptr            intfobj;
CORBA::OperationDef_ptr         opef;
CORBA::Contained::Description   *desc;
```

```
CORBA::OperationDescription    *c;
CORBA::IRObject_ptr            irobj;
CORBA::StructDef_ptr           StructObj;
CORBA::AliasDef_ptr            AliasObj;
CORBA::IDLType_ptr             idltypep;

// インタフェースリポジトリのオブジェクトリファレンス獲得
obj = orb->resolve_initial_references(
        CORBA_ORB_ObjectId_InterfaceRepository, *env );
// CORBA::Repositoryクラスに変換
intf_rep = CORBA::Repository::_narrow( obj );

// オブジェクトの検索
intfobj = intf_rep->lookup_id( "IDL:test3/intf1/ope1:1.0", *env );
if(intfobj == CORBA_OBJECT_NIL){
      /* オブジェクトが存在しない  */
      printf("Not Found Object¥n");
      exit(1);
}

// OperationDefオブジェクトのインタフェース情報の獲得
opef = CORBA::OperationDef::_narrow(intfobj);
desc   = opef->describe(*env);

// OperationDefオブジェクト固有情報構造体の取り出し
CORBA::Any *opeany = (CORBA::Any*)(desc->value);
c = (CORBA::OperationDescription *)opeany->value();

// パラメタシーケンス情報を参照
CORBA::ParDescriptionSeq *parseq = (CORBA::ParDescriptionSeq*)c->parameters;
for( int n = 0;n < parseq->length(); n++){
      CORBA::ParameterDescription *parades =
                  (CORBA::ParameterDescription*)(*parseq)[n];
      // パラメタの型の参照
      objvar = (CORBA::Object_var*)&(parades->type_def);
      idlobj = (CORBA::Object_ptr)(*objvar);
      idltypep = CORBA::IDLType::_narrow( idlobj );
      kind  = idltypep->def_kind(*env);
      if(kind == CORBA::dk_Struct)
            // StructObjにStructDefのオブジェクトリファレンス設定
            StructObj = CORBA::StructDef::_narrow(idltypep);
      else if(kind == CORBA::dk_Alias)
            // AliasObjにAliasDefのオブジェクトリファレンス設定
            AliasObj  = CORBA::AliasDef::_narrow(idltypep);
}
```

Java言語

```
org.omg.CORBA.Object                        Obj;
org.omg.CORBA.Repository                    Rep;
org.omg.CORBA.Object                        obj;
org.omg.CORBA.OperationDef                  opef;
org.omg.CORBA.ContainedPackage.Description  desc;
org.omg.CORBA.OperationDescription          c;
org.omg.CORBA.ParameterDescription          paraseq[];
org.omg.CORBA.ParameterDescription          parades;
int                                         kind;
org.omg.CORBA.IDLType                       idltype;
org.omg.CORBA.AliasDef                      AliasObj;
org.omg.CORBA.StructDef                     StructObj;

// インタフェースリポジトリのオブジェクトリファレンス獲得
```

```
Obj = Orb.resolve_initial_references( "InterfaceRepository" );
// RepositoryHelperクラスに変換
Rep = org.omg.CORBA.RepositoryHelper.narrow( Obj );
// オブジェクトの検索
obj = Rep.lookup_id( "IDL:test3/intf1/ope1:1.0" );
// OperationDefオブジェクトのインタフェース情報の獲得
opef = org.omg.CORBA.OperationDefHelper.narrow( obj );
desc = opef.describe();

// OperationDefオブジェクト固有情報構造体の取り出し
org.omg.CORBA.Any opeany = desc.value;
c = ( org.omg.CORBA.OperationDescription )
    org.omg.CORBA.OperationDescriptionHelper.extract( opeany );
// パラメタシーケンス情報を参照
paraseq = c.parameters;
for( i = 0; i < paraseq.length; i++ ) {
    // パラメタの型の参照
    parades = paraseq[i];
    idltype = org.omg.CORBA.IDLTypeHelper.narrow( parades.type_def );
    kind = idltype.def_kind();
    if( kind == org.omg.CORBA.DefinitionKind.dk_Struct.value() ) {
        // StructObjにStructDefのオブジェクトリファレンス設定
        StructObj = org.omg.CORBA.StructDefHelper.narrow( obj );
    } else if( kind == org.omg.CORBA.DefinitionKind.dk_Alias.value() ) {
        // AliasObjにAliasDefのオブジェクトリファレンス設定
        AliasObj = org.omg.CORBA.AliasDefHelper.narrow(stcf);
    }
}
```

COBOL

```
*########## インタフェースリポジトリのオブジェクト検索 ##########
     MOVE FUNCTION LENG ("IDL:test3/intf1/ope1:1.0") TO STRING-LENGTH.
     MOVE "IDL:test3/intf1/ope1:1.0" TO REP-BUF.
     CALL "CORBA-STRING-SET" USING
                REPOSITORYID
                STRING-LENGTH
                REP-BUF.
     CALL "CORBA-REPOSITORY-LOOKUP-ID" USING
                OBJ
                REPOSITORYID
                ENV
                INTF-INTF.
     CALL "CORBA-FREE" USING REPOSITORYID.
*--- オブジェクトが存在するかのチェック ---
     CALL "CORBA-OBJECT-IS-NIL" USING
                INTF-INTF
                ENV
                TMP-RESULT.
     IF TMP-RESULT = CORBA-TRUE
*--- オブジェクトは存在しない ---
          DISPLAY "Null Object Reference"
          GO TO MAIN-END
     END-IF.

*########## OperationDefオブジェクトのインタフェース情報の獲得 ##########
   MOVE INTF-INTF TO OBJ.
   CALL "CORBA-OPERATIONDEF-DESCRIBE" USING
                OBJ
                ENV
                LINK-BUF.
   MOVE "CORBA-OPERATIONDEF--DESCRIBE" TO MESS.
```

```
   PERFORM ENV-CHECK.
*--- OperationDefオブジェクト固有情報構造体の取り出し ---
   SET ADDRESS OF LINK-CONDES TO LINK-BUF.
   MOVE IDL-VALUE OF LINK-CONDES TO TMP-ANY.
   MOVE ANY-VALUE OF TMP-ANY TO TEMP-BUF.
   SET ADDRESS OF TMP-OPR TO TEMP-BUF.
   MOVE FUNCTION ADDR(PARAMETERS OF TMP-OPR) TO TEMP-BUF.
   SET ADDRESS OF PARDESSEQ TO TEMP-BUF.
*---- パラメタシーケンス情報を参照 ---
   IF SEQ-LENGTH OF PARDESSEQ = 0
*---- パラメタがない ---
        DISPLAY "PARA-LENGTH = 0"
   END-IF.
   PERFORM TEST BEFORE
           VARYING CNT FROM 1 BY 1
               UNTIL CNT > SEQ-LENGTH OF PARDESSEQ
           CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
                 TEMP-BUF
                 CNT
                 ELEMENT-TYPE
           SET  ADDRESS OF TMP-PARA TO ELEMENT-TYPE
           MOVE FUNCTION LENG (DSP-BUF) TO STRING-LENGTH
*---         パラメタの型の参照  ----
           CALL "CORBA-TYPECODE-KIND" USING
                IDL-TYPE OF TMP-PARA
                ENV
                TMP-KIND
           MOVE TMP-KIND TO TYPE-KIND
           IF  CORBA-TK-STRUCT OF  TYPE-KIND
*---              STRUCT-OBJにSTRUCTDEFのオブジェクトリファレンス設定
                MOVE TYPE-DEF OF TMP-PARA TO STRUCT-OBJ
           ELSE
                IF  CORBA-TK-ALIAS OF TYPE-KIND
*---                    ALIAS-OBJにALIASDEFのオブジェクトリファレンス設定
                     MOVE TYPE-DEF OF TMP-PARA TO  ALIAS-OBJ
                END-IF
           END-IF
   END-PERFORM.
```

OOCOBOL

```
   01 ORB           OBJECT REFERENCE CORBA-ORB.
   01 OBJ           OBJECT REFERENCE CORBA-OBJECT.
   01 REP           OBJECT REFERENCE CORBA-REPOSITORY.
   01 CONTAINED-OBJ OBJECT REFERENCE CORBA-CONTAINED.
   01 DESC          OBJECT REFERENCE CORBA-CONTAINED-DESCRIPTION.
   01 OPE-ANY       OBJECT REFERENCE CORBA-ANY.
   01 C             OBJECT REFERENCE CORBA-OPERATIONDESCRIPTION.
   01 PARAMSEQ      TYPE CORBA-PARDESCRIPTIONSEQ.
   01 PARADES       OBJECT REFERENCE CORBA-PARAMETERDESCRIPTION.
   01 STR           OBJECT REFERENCE CORBA-STRING.
   01 IDLTYPE       OBJECT REFERENCE CORBA-IDLTYPE.
   01 PARA-NAME     PIC X(40).
   01 SEQ-LEN       TYPE CORBA-LONG.
   01 N             TYPE CORBA-UNSIGNED-LONG.
   01 TCK           TYPE CORBA-TCKIND.
   01 PARA-MODE     TYPE CORBA-PARAMETERMODE.
   01 KIND          TYPE CORBA-DEFINITIONKIND.
   01 ALIAS-OBJ     OBJECT REFERENCE CORBA-OBJECT.
   01 ALIASDEF-OBJ  OBJECT REFERENCE CORBA-ALIASDEF.
   01 OPEF          OBJECT REFERENCE CORBA-OPERATIONDEF.
   01 TYPECODE      OBJECT REFERENCE CORBA-TYPECODE.
```

```
   01 MEM-NAME      PIC X(20).
   01 STRUCT-OBJ    OBJECT REFERENCE CORBA-OBJECT.
   01 STRUCTDEF-OBJ OBJECT REFERENCE CORBA-STRUCTDEF.
   01 TYPE_DEF      PIC X(40).
   01 ID-NAME       OBJECT REFERENCE CORBA-STRING.

     INVOKE CORBA "ORB_INIT" USING APL-NAME FJ-OM_ORBID RETURNING ORB.
*######## インタフェースリポジトリのオブジェクトリファレンス獲得
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES" USING CORBA-OBJECTID_INTERFACERE-001
                                             RETURNING OBJ.
*######## REPOSITORYクラスに変換
     INVOKE CORBA-REPOSITORY "NARROW" USING OBJ
                                     RETURNING REP.
*######## オブジェクトの検索
     INVOKE REP "LOOKUP_ID" USING "IDL:test3/intf1/ope1:1.0"
                            RETURNING OBJ.
*######## OperationDefオブジェクト固有情報構造体の取り出し
     INVOKE CORBA-OPERATIONDEF "NARROW" USING OBJ
                                       RETURNING OPEF.
     INVOKE OPEF "DESCRIBE" RETURNING DESC.
     SET OPE-ANY TO IDL-VALUE OF DESC.
*
     INVOKE CORBA-OPERATIONDESCRIPTION--H "EXTRACT" USING OPE-ANY
                                                    RETURNING C.
*######## パラメタシーケンス情報を参照
     SET PARAMSEQ TO PARAMETERS OF C.
*######## パラメタ数の参照
     MOVE SEQ-LENGTH OF PARAMSEQ TO SEQ-LEN.
     DISPLAY "PARAMETER NUMBER = " SEQ-LEN.
     PERFORM VARYING  N FROM 1 BY 1 UNTIL SEQ-LEN < N
        INVOKE PARAMSEQ "GET-VALUE" USING N
                                  RETURNING  PARADES
*######## パラメタの型の参照
        SET IDLTYPE TO TYPE_DEF OF PARADES
        MOVE DEF_KIND OF IDLTYPE TO KIND
*######## 型のIDの取得
        SET TYPECODE TO IDL-TYPE OF PARADES
        INVOKE TYPECODE "ID" RETURNING ID-NAME
        INVOKE ID-NAME "GET-VALUE" RETURNING TYPE_DEF
*
        EVALUATE TRUE
        WHEN  CORBA-DK_STRUCT OF KIND
*######## STRUCT-OBJにStructDefのオブジェクトリファレンス設定
          INVOKE REP "LOOKUP_ID" USING TYPE_DEF
                                 RETURNING STRUCT-OBJ
          INVOKE CORBA-STRUCTDEF "NARROW" USING STRUCT-OBJ
                                          RETURNING STRUCTDEF-OBJ
        WHEN CORBA-DK_ALIAS  OF KIND
*######## ALIAS-OBJにAliasDefのオブジェクトリファレンス設定
          INVOKE REP "LOOKUP_ID" USING TYPE_DEF
                                 RETURNING ALIAS-OBJ
          INVOKE CORBA-ALIASDEF "NARROW" USING ALIAS-OBJ
                                         RETURNING ALIASDEF-OBJ
        END-EVALUATE
   END-PERFORM.
```

(2)StructDefオブジェクトのメンバ情報の獲得

StructDefオブジェクトのオブジェクトリファレンスを指定してmembersメソッドを発行すると、構造体のメンバ情報(識別名、タイプコード)がリスト形式で返されます。

C言語

```
CORBA_StructMemberSeq          *members;
CORBA_StructMember             *str;

/* 構造体のメンバ情報がリスト形式で返されます */
members = CORBA_StructDef__get_members(StructObj,&env);

/* 構造体メンバの解析                          */
for(i=0,str=members->_buffer; i<members->_length;i++){
      /* 識別名の参照                         */
      printf("Identifier     : %s¥n",str[i].name);
      /* タイプコードの参照                   */
      printf("TypeCode_kind  : %d¥n",CORBA_TypeCode_kind(str[i].type, &env));
}
```

C++言語

```
CORBA::StructDef_ptr          strobj;
CORBA::StructMemberSeq        *structseq;
CORBA::StructMember           *structmem;
// 構造体のメンバ情報がリスト形式で返されます
strobj = CORBA::StructDef::_narrow(StructObj);
structseq = strobj->members(*env);
for(int y=0; y < structseq->length(); y++) {
      /* 識別名の参照                         */
      structmem = (CORBA::StructMember*)(*structseq)[y];
      printf( "name = %s¥n", (char *)structmem->name );
      /* タイプコードの参照                   */
      CORBA::TypeCode
      *typecode2 = (CORBA::TypeCode *)(structmem->type);
      kind = typecode2->kind(*env);
      printf("TypeCode_kind = %d¥n",kind);
}
```

Java言語

```
org.omg.CORBA.StructDef     strobj;
org.omg.CORBA.StructMember  structseq[];
org.omg.CORBA.StructMember  structmem;
org.omg.CORBA.TypeCode      typecode;
int                         tkind;

// 構造体のメンバ情報がリスト形式で返されます
strobj = org.omg.CORBA.StructDefHelper.narrow( StructObj );
structseq = strobj.members();
for( i = 0; i < structseq.length; i++ ) {
    // 識別名の参照
    structmem = structseq[i];
    inoutcc.output( "name = " + structmem.name );
    // タイプコードの参照
    typecode = structmem.type;
    tkind = typecode.kind().value();
    inoutcc.output( "TypeCode_kind = " + tkind );
}
```

COBOL

```
*##########  構造体のメンバ情報がリスト形式で返されます ##########
   CALL "CORBA-STRUCTDEF--GET-MEMBERS" USING
```

```
               STRUCT-OBJ
               ENV
               LINK-BUF.
   SET ADDRESS OF STRMEMSEQ TO LINK-BUF.
   IF SEQ-LENGTH OF STRMEMSEQ = 0
        DISPLAY "STRMEM-LENGTH = 0"
   END-IF.
   PERFORM TEST BEFORE
           VARYING CNT FROM 1 BY 1
               UNTIL CNT > SEQ-LENGTH OF STRMEMSEQ
           CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
                 LINK-BUF
                 CNT
                 ELEMENT-TYPE

           SET  ADDRESS OF STRMEM TO ELEMENT-TYPE
*---         識別名の参照       ---
           MOVE FUNCTION LENG (DSP-BUF) TO STRING-LENGTH
           CALL "CORBA-STRING-GET" USING
               NAME OF STRMEM
               STRING-LENGTH
               DSP-BUF
           DISPLAY "NAME=" DSP-BUF
*---         タイプコードの参照    ---
           CALL "CORBA-TYPECODE-KIND" USING
                IDL-TYPE OF STRMEM
                ENV
                TMP-KIND
           DISPLAY "TYPECODE KIND=" TMP-KIND
   END-PERFORM.
```

OOCOBOL

```
   01 MEM-STR       OBJECT REFERENCE CORBA-STRING.
   01 TKIND         TYPE CORBA-TCKIND.
   01 STRUCT-MEM    OBJECT REFERENCE CORBA-STRUCTMEMBER.
   01 I             TYPE CORBA-UNSIGNED-LONG.
   01 MEM-NUM       TYPE CORBA-UNSIGNED-LONG.
   01 STRUCT-SEQ    TYPE CORBA-STRUCTMEMBERSEQ.

*######## 構造体のメンバ情報がリスト形式で返されます
        SET STRUCT-SEQ TO MEMBERS OF STRUCTDEF-OBJ
        MOVE SEQ-LENGTH OF STRUCT-SEQ TO MEM-NUM
        PERFORM VARYING I FROM 1 BY 1 UNTIL MEM-NUM < I
*######## 識別名の参照
          INVOKE STRUCT-SEQ "GET-VALUE" USING I
                                      RETURNING STRUCT-MEM
          SET MEM-STR TO NAME OF STRUCT-MEM
          INVOKE MEM-STR "GET-VALUE" RETURNING MEM-NAME
          DISPLAY "NAME = " MEM-NAME
*######## タイプコードの参照
          SET TYPECODE TO IDL-TYPE OF STRUCT-MEM
          INVOKE TYPECODE "KIND" RETURNING TKIND
          DISPLAY "TYPECODE_KIND = " TKIND
        END-PERFORM
```

(3)StringDefオブジェクトのオブジェクトリファレンスの獲得

AliasDefオブジェクトのオブジェクトリファレンスを指定してoriginal_type_defメソッドを発行すると、StringDefオブジェクトのオブジェクトリファレンスが返されます。

C言語

```
CORBA_IDLType  originalobj;
/* StringDefオブジェクトのオブジェクトの獲得 */
originalobj = CORBA_AliasDef__get_original_type_def(AliasObj,&env);
```

C++言語

```
CORBA::IDLType_ptr  originalobj;
// StringDefオブジェクトのオブジェクトの獲得
originalobj = AliasObj->original_type_def(*env);
```

Java言語

```
CORBA.IDLType         originalobj;
// StringDefオブジェクトのオブジェクトの獲得
originalobj = AliasObj.original_type_def();
```

COBOL

```
*########## StringDefオブジェクトのオブジェクトの獲得 ##########
   CALL "CORBA-ALIASDEF--GET-ORIGINAL-TYPE-DEF" USING
                ALIAS-OBJ
                ENV
                IDLTYPE.
```

OOCOBOL

```
 01 ORIGINALOBJ   OBJECT REFERENCE CORBA-IDLTYPE.
 01 TMP-OBJ       OBJECT REFERENCE CORBA-OBJECT.

*######## StringDefオブジェクトのオブジェクトの獲得
       SET ORIGINALOBJ TO ORIGINAL_TYPE_DEF OF ALIASDEF-OBJ
       SET TMP-OBJ TO ORIGINALOBJ
```

(4)StringDefオブジェクトの情報獲得

StringDefオブジェクトのオブジェクトリファレンスを指定してboundメソッドを発行すると、最大文字列長が返されます。

C言語

```
CORBA_unsigned_long bound;
/* StringDefオブジェクトの最大文字列長の獲得 */
bound = CORBA_StringDef__get_bound(originalobj,&env);
```

C++言語

```
CORBA::ULong          len;
CORBA::StringDef_ptr  stringobj;
// StringDefオブジェクトの最大文字列長の獲得
stringobj = CORBA::StringDef::_narrow(originalobj);
len = stringobj->bound(*env);
```

Java言語

```
org.omg.CORBA.StringDef     stringobj;
int                         len;
// StringDefオブジェクトの最大文字列長の獲得
stringobj = org.omg.CORBA.StringDefHelper.narrow( originalobj );
len = stringobj.bound();
```

COBOL

```
*--- StringDefオブジェクトの最大文字列長の獲得 ---
   MOVE IDLTYPE TO TMP-OBJ.
   CALL "CORBA-STRINGDEF--GET-BOUND" USING
                TMP-OBJ
                ENV
                BOUND.
```

OOCOBOL

```
 01 STRINGDEF-OBJ OBJECT REFERENCE CORBA-STRINGDEF.
 01 LEN           TYPE CORBA-UNSIGNED-LONG.

*######## StringDefオブジェクトの最大文字列長の獲得
      INVOKE CORBA-STRINGDEF "NARROW" USING TMP-OBJ
                                    RETURNING STRINGDEF-OBJ
      MOVE BOUND OF STRINGDEF-OBJ TO LEN
```

第11章　CORBAアプリケーションの高度な使い方

11.1 Factory

Factoryインタフェースは、ユーザプログラムのサーバアプリケーションに実装します。IDLに以下のインタフェース定義を追加します。

注意

Factoryインタフェースを用いたアプリケーションは、C++言語／Java言語において使用できます。

ポイント

ここでは、C++言語のFactoryについて説明します。

Java言語のFactoryについては、“5.12 インスタンス管理とアプリケーション形態”を参照してください。

IDL定義

```
interface factory {
    Object create_obj();  /* オブジェクトリファレンスを作成するメソッドを定義 */
};
```

ここでは、以下のIDL定義に従って、Factoryの実装について説明します。

```
module ODsample {
      interface intf1 {
              attribute long      x;
              attribute string    y;
              void destroy();
      };
      interface factory {
              intf1 create_obj();
      };
};
```

11.1.1 サーバアプリケーションのプログラミング

Factoryインタフェースによるオブジェクトリファレンスの作成方法について説明します。

(1) サーバの初期化／活性化

インタフェース情報およびインプリメンテーション情報を獲得し、オブジェクトリファレンスを作成します。

1. 初期化
   CORBA::ORB_init()、CORBA::ORB::BOA_init()によりサーバの初期化を行います。
2. インプリメンテーション情報の獲得
   Factoryインタフェースのインプリメンテーション情報を獲得します。FJ::ImplementationRep::lookup_id()でFactoryインタフェースのインプリメンテーションリポジトリID(“_IMPL_モジュール名_factory”)を指定します。
3. インタフェース情報の獲得
   Factory以外のインタフェース情報を獲得します。

4. サーバの活性化

2.で獲得したインプリメンテーション情報を指定してCORBA::BOA::impl_is_ready()またはCORBA::BOA::obj_is_ready()を発行します。これにより、IDLに定義されたすべてのインタフェースが活性化されます。

```
boa->impl_is_ready( impl, env );
```

ここでは、main関数でインプリメンテーション情報／インタフェース情報を獲得する例を示します。

```
static  CORBA::ORB_ptr                orb;
static  CORBA::BOA_ptr                boa;
static  CORBA::Repository_ptr         intf_rep;
static  FJ::ImplementationRep_ptr     impl_rep;
static  CORBA::InterfaceDef_ptr       intf1;
static  CORBA::ImplementationDef_ptr  impl;
static  CORBA::Object_ptr             o;
static  CORBA::Environment            env;
CORBA::ReferenceData                  id;

int
main( int argc, char *argv[] )
{
      int        current_argc = argc;
      char       buf[128];

      // 初期化処理
      orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );
      boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, env );

      // インプリメンテーション情報の獲得
      o = orb->resolve_initial_references(
              CORBA_ORB_ObjectId_ImplementationRepository, env );
      impl_rep = FJ::ImplementationRep::_narrow( o );
      o = impl_rep->lookup_id( _IMPL_ODsample_factory, env );
      impl = CORBA::ImplementationDef::_narrow( o );

      // インタフェース情報の獲得
      o = orb->resolve_initial_references(
              CORBA_ORB_ObjectId_LightInterfaceRepository, env );
      intf_rep = CORBA::Repository::_narrow( o );
      o = intf_rep->lookup_id( _INTF_ODsample_intf1, env );
      intf1 = CORBA::InterfaceDef::_narrow( o );

      // サーバの活性化
      boa->impl_is_ready( impl, env );
}
```

(2) Factoryインタフェースの実装関数

前項で獲得したインプリメンテーション情報／インタフェース情報を使用して、CORBA::BOA::create()でオブジェクトリファレンスを作成します。

```
// create_obj メソッドの実装
ODsample::intf1_ptr
ODsample_factory_impl::create_obj( CORBA::Environment &env )
      throw( CORBA::Exception )
{
      // BOA::create関数を使用して、オブジェクトリファレンスを作成します。
      CORBA::Object_ptr new_obj = boa->create( id, intf1,  impl, env );
      ODsample::intf1_ptr fac = ODsample::intf1::_narrow( new_obj );
      CORBA::release( new_obj );
      return( fac );
}
```

(3) ネーミングサービスへの登録

OD_or_admコマンドでネーミングサービスにFactoryインタフェースのオブジェクトリファレンスを登録します。-cオプションには、Factoryインタフェースのインタフェースリポジトリを指定します。

```
OD_or_adm -c IDL:ODsample/factory:1.0 -n ODsample::factory
```

11.1.2 クライアントアプリケーションのプログラミング

(1) Factoryインタフェースのオブジェクトリファレンスの獲得

ネーミングサービスより、Factoryインタフェースのオブジェクトリファレンスを獲得します。

```
CosNaming::Name_ptr           name;
CosNaming::NameComponent_var  *name_component;

CORBA::Object_ptr
      obj = orb->resolve_initial_references( CORBA_ORB_ObjectId_NameService, env );

CosNaming::NamingContext_ptr
      NamingContext_obj = CosNaming::NamingContext::_narrow( obj );
CORBA::release( obj );

name_component = CosNaming::Name::allocbuf(1);
name_component[0]->id = (const CORBA::Char *)"ODsample::factory";
name_component[0]->kind = (const CORBA::Char *)"";
name = new CosNaming::Name(1, 1, name_component,CORBA_TRUE);
obj = NamingContext_obj->resolve( *name, env );
CORBA::release( NamingContext_obj );
delete name;
```

(2) Factoryインタフェースに実装したメソッドの呼び出し

ネーミングサービスより獲得したオブジェクトリファレンスを使用してcreate_objメソッドを呼び出します。OD_impl_instコマンドでサーバアプリケーションをインプリメンテーションリポジトリに登録する場合は、定義ファイル(“OD_impl_inst -ax”)で“iswitch=ON”を指定すると、メソッドの初回呼出し時にサーバアプリケーション内にクライアントごとのインスタンスデータが作成されます。

```
// インタフェース"intf1"のオブジェクトリファレンスの獲得
ODsample::factory_ptr ap = ODsample::factory::_narrow( obj );
ODsample::factory_var av = ap;

ODsample::intf1_var target = av->create_obj( env );
```

11.1.3 private領域の実装

private領域を使用する場合のプログラミング例を以下に示します。
IDLでattributeを定義すると、インプリメンテーションクラス用のprivateデータ域にマッピングされます。この領域は、サーバアプリケーションが管理します。attributeの詳細については、“4.6.10 属性宣言(attribute)のマッピング”を参照してください。

(1) サーバアプリケーションの処理

サーバアプリケーションのprivate領域の設定／参照メソッドの実装例を以下に示します。

```
// attribute x の設定（固定長データ）
void
ODsample_intf1_impl::x( CORBA::Long l, CORBA::Environment &env )
      throw( CORBA::Exception )
{
```

```
        __x = l;
}

// attribute x の参照（固定長データ）
CORBA::Long
ODsample_intf1_impl::x( CORBA::Environment &env )
        throw( CORBA::Exception )
{
        return( __x );
}

// attribute y の設定（可変長データ）
void
ODsample_intf1_impl::y( const CORBA::Char* s, CORBA::Environment &env )
        throw( CORBA::Exception )
{
        __y = (const CORBA::Char *)s;
}

// attribute y の参照（可変長データ）
CORBA::Char *
ODsample_intf1_impl::y( CORBA::Environment &env )
        throw( CORBA::Exception )
{
        CORBA::Char *tmp;

        tmp = CORBA::string_alloc(strlen(__y));
        strcpy( tmp, __y );
        return( tmp );
}
```

また、IDLコンパイラの生成するヘッダファイルに直接privateデータを設定することもできます。このとき、文字列などの可変長データを使用する場合は、獲得した領域を解放する必要があります。詳細については、“11.1.5 終了処理のプログラミング”を参照してください。

```
class    ODsample_intf1_impl : public _sk_ODsample_intf1
{
public:
    ODsample_intf1_impl();
    ~ODsample_intf1_impl();

    // Attribute Method ( for _get )
    CORBA::Long    x(
    .......

private :
    CORBA::Long    __x;
    CORBA::String_var    __y;

    // ユーザ追加領域
    long    user_data;
    char    *data;
};
```

注意

private領域を使用した場合は、アプリケーション実行時にクライアントごとにインスタンスを持たせるため、OD_impl_instコマンド実行時（インプリメンテーションリポジトリ登録）に“iswitch=ON”を指定する必要があります。詳細については、“リファレンスマニュアル（コマンド編）”の“OD_impl_inst”を参照してください。

(2) クライアントアプリケーションの処理

クライアントアプリケーションでのattributeの使用例を以下に示します。

```
// attribute x の設定（固定長データ）
target->x( 3, env );

// attribute x の参照（固定長データ）
CORBA::Long result = target->x( env );

// attribute y の設定（可変長データ）
CORBA::Char *dummy = CORBA::string_alloc(7);
strcpy( dummy, "testabc" );
target->y( dummy, env );

// attribute y の参照（可変長データ）
CORBA::Char *data = target->y( env );
cout << "data = " << data << endl;
```

11.1.4 クライアントごとのprivateデータの初期化

Factoryオブジェクトを生成した際に、privateデータを初期化するときの値をクライアントごとに設定する場合の方法例について以下に示します。

IDL定義

Factoryインタフェースのオブジェクトを生成するcreate_objメソッドに、識別子として使用するパラメタを追加します。ここでは、string型パラメタを使用し、クライアントアプリケーションを使用しているユーザ名を渡すこととします。

```
module ODsample {
    interface intf1 {
        attribute long      x;
        attribute string    y;
        void destroy();
    };
    interface factory {
        intf1 create_obj( in string user_name );
    };
};
```

(1) サーバアプリケーションの処理

Factoryインタフェースの実装処理関数

上記IDL定義で示してあるパラメタ“user_name”をオブジェクトの識別情報（ReferenceData）として設定します。

```
// create_obj メソッドの実装
ODsample::intf1_ptr
    ODsample_factory_impl::create_obj(
        CORBA::Char *user_name, CORBA::Environment &env )
throw( CORBA::Exception ) {
    CORBA::ReferenceData  *id;
    CORBA::Octet          *data;
    CORBA::ULong          len;

    len = strlen( user_name ) + 1;
    data = CORBA::ReferenceData::allocbuf( len );
    strcpy( data, user_name );
    id = new CORBA::ReferenceData( len, len, data, CORBA_FALSE );
    CORBA::Object_ptr new_obj = boa->create( id, intf1,  impl, env );
    ODsample::intf1_ptr fac = ODsample::intf1::_narrow( new_obj );
    CORBA::release( new_obj );
```

```
    delete id;
    delete data;
    return( fac );
}
```

スケルトンの変更

コンストラクタは、IDLコンパイラが生成したスケルトン上に以下のように生成されます。ヘッダファイルに直接設定したuser_var／dataを初期化する処理を以下のように追加します。

```
ODsample_intf1_impl::ODsample_intf1_impl()
{
    // privateデータの初期化
    user_var = 0;
    data = NULL;
}
```

メソッド実装関数の処理例

各メソッドが呼び出されたときにprivateデータのdataがNULLの場合は、クライアントごとのprivateデータの初期設定が行われないと判断して、初期設定を行います。
IDL定義にあるxの参照メソッドの実装関数の例を以下に示します。

```
// attribute x の参照（固定長データ）  CORBA::Long
ODsample_intf1_impl::x( CORBA::Environment &env )
    throw( CORBA::Exception )
{
    if (this->data == NULL)
        intf1_initialize( this );
    return( __x );
}
```

オブジェクトの識別情報(ReferenceData)を取り出し、ユーザ名に応じた初期値を設定する処理を以下のように行います。

```
// クライアントごとのprivateデータの初期化処理
void
intf1_initialize( ODsample_intf1_impl obj )
{
    CORBA::Environment     env;
    CORBA::ReferenceData   *id;
    CORBA::ULong           len, i;
    CORBA::Char            *name;

    id = obj->get_id( &env );
    len = id->length();
    name = CORBA::string_alloc( len );
    for (i = 0; i < len; i++)
        name[i] = id[i];
        if (strcmp(name, ...) == 0) {
            // ユーザ名nameに対応した初期化処理
            obj->x = ...;
            obj->y = ...;
            obj->user_var = ...;
            obj->data = name;
        }
           ...
        delete id;
}
```

(2) クライアントアプリケーションの処理

“11.1.2 クライアントアプリケーションのプログラミング”を参照してください。

11.1.5 終了処理のプログラミング

メソッド内でdisposeが呼び出された場合は、スケルトンに実装されているインプリメンテーションクラスのデストラクタが呼び出されます。
IDLコンパイラの作成したスケルトン(xxxx_C++_skel.c)を編集することにより、アプリケーション用の終了処理を実装できます。

(1) サーバアプリケーションの処理

thisポインタに対して、CORBA::BOA::dispose()を発行します。
終了処理メソッドの例を以下に示します。

```
// 終了処理メソッド
void
ODsample_intf1_impl::destroy( CORBA::Environment &env )
      throw( CORBA::Exception )
{
      try {
              boa->dispose( this,  env );
      }
      catch( CORBA::SystemException &se ){
              throw( se );
      }
}
```

(2) スケルトンの変更

デストラクタは、スケルトンに以下のように生成されます。必要に応じて終了処理を追加します。可変長のprivateデータを使用した場合は、解放処理を追加します。

```
ODsample_intf1_impl::~ODsample_intf1_impl()
{

// privateデータの解放
    if (data)
        delete data;

// 終了処理を記述
    .....
}
```

11.1.6 クライアントとの接続切断時の処理関数の登録

クライアントの異常終了などで接続が切断された場合は、スケルトンに実装されているインプリメンテーションクラスのデストラクタが自動的に呼び出されますが、デストラクタが実行される前に何らかの処理を行う場合は、その処理内容を記述した処理関数を登録します。
ここで登録された関数は、デストラクタが呼び出される直前に実行されます。

(1) 処理関数の形式

処理関数の形式を以下に示します。引数はvoidのポインタですが、呼出し時にthisポインタが渡されます。復帰値は、ありません。

```
void exit_func( void *this_pointer );
```

(2) 処理関数の登録

処理関数、CORBA::ORB::reg_exit_function()で登録します。登録は、CORBA::ORB_init()とCORBA::BOA::impl_is_ready()の間で行ってください。
登録例を以下に示します。

```
void exit_func( void *this_pointer )
{
    // 処理内容。例は後で示す。
}

int
main( int argc, char *argv[] )
{
    int    current_argc = argc;
    .....

    // 初期化処理
    orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

    ....

    // 処理関数の登録
    orb->reg_exit_function( orb, _INTF_ODsample_intf1, exit_func, env );

    ....

    // サーバの活性化
    boa->impl_is_ready( impl, env );
}
```

(3) 処理関数の内容例

privateデータの内容と共に簡単な警告を出力します。ただし、privateメンバを参照するためには、exit_func関数をODsample_intf1_implクラスのfriend関数にしておく必要があります。

```
void exit_func( void *this_pointer )
{
    ODsample_intf1_impl *ptr = (ODsample_intf1_impl *)this_pointer;
}
```

CORBA::BOA::dispose()が発行された後にクライアントとの接続が切断された場合、登録された処理関数は呼び出されません。この場合も処理関数を実行する場合は、CORBA::BOA::dispose()の発行前に処理関数を実行してください。

例

```
void
ODsample_intf1_impl::destroy( CORBA::Environment &env )
    throw( CORBA::Exception )
{
    try {
        exit_func( this ); // 登録関数の明示的な呼び出し
        boa->dispose( this, env );
    }
    catch( CORBA::SystemException &se ) {
        throw( se );
    }
}
```

11.2 オブジェクト単位のプロセスバインド

プロセス多重度が2以上のサーバアプリケーションでは、クライアントからのリクエストがどのプロセスに割り当てられるかは不定となります。オブジェクト単位のプロセスバインド機能を使用することにより、常に特定のサーバアプリケーションにリクエストを割り当てることが可能となります。

オブジェクト単位のプロセスバインド機能を使用したサーバアプリケーションは、C++言語において使用可能です。なお、クライアントアプリケーションは任意の開発言語を使用可能です。

プロセスバインドを使用したプログラミングの例については、“付録E プロセスバインド機能を用いたセション管理の例”を参照してください。

11.2.1 定義情報

オブジェクト単位のプロセスバインド機能を使用する場合は、インプリメンテーションリポジトリを登録する際に“iswitch”パラメタに“object”を設定してください。また、11.2.5 セションタイムアウト機能を使用する場合は、“ssn_timeout”パラメタも設定してください。詳細については、“リファレンスマニュアル(コマンド編)”の“OD_impl_inst”を参照してください。

システムに登録可能なプロセスとオブジェクトのバインド関係の数は、CORBAサービスの動作環境ファイル(config)の“max_bind_instances”パラメタで設定します。詳細については、“チューニングガイド”の“config”を参照してください。

11.2.2 使用するAPI

オブジェクト単位のプロセスバインド機能では、以下のAPIを使用します。

CORBA::ORB::bind_object()

オブジェクトとプロセスのバインド関係を登録します。

CORBA::ORB::unbind_object()

オブジェクトとプロセスのバインド関係を解除します。

CORBA::ORB::set_unbinded_object_rejecting()

バインド関係未登録オブジェクトに対して例外を通知するインタフェースを登録します。

CORBA::Object::check_ssn_timeout()

セションタイムアウトが発生したか否かをチェックします。

各APIの詳細については、“リファレンスマニュアル(API編)”を参照してください。

11.2.3 オブジェクトとインスタンスの関係

サーバプロセス内では、CORBAオブジェクトとC++インスタンスの対応関係は、1対1となります。そのため、同じオブジェクトを複数クライアントで共用する場合、サーバ側のC++インスタンスは同一のものが使用されます。

サーバ側のC++インスタンスはオブジェクトを使用した初回アクセス時にメソッド呼出し直前で作成され、以下の条件の時に解放されます。

- CORBA::ORB::unbind_object()を発行してバインド関係を解除した場合。
- セションタイムアウトが発生した場合。

CORBA::ORB::unbind_object()を発行した場合、以下のタイミングでインスタンスは解放されます。

- スケルトンからの復帰後。かつ、
- 該当のインスタンスに対してクライアントからアクセスされていない。

たとえば、CORBA::ORB::unbind_object()が発行された時点で複数クライアントから該当インスタンスにアクセスされた場合、最後のクライアントアクセス後にスケルトンから復帰した直後にインスタンスが解放されます。

注意

セション継続用オブジェクトに対応するインスタンスが解放された後に、クライアントからセション継続用オブジェクトを使用してアクセスされた場合、オブジェクトのインタフェースがCORBA::ORB::set_unbinded_object_rejecting()で登録されているかどうかで動作が異なります。

- **オブジェクトのインタフェースが登録されていない場合**
  新しくC++インスタンスを生成し、サーバメソッドを呼び出します。
- **オブジェクトのインタフェースが登録されている場合**
  クライアントにシステム例外を通知します。

11.2.4 リクエスト振り分け方式

“iswitch”パラメタに“object”を設定されたインプリメンテーションのサーバアプリケーションに対してバインド関係が登録されているオブジェクトを使用している場合、バインド関係が登録されているプロセスにリクエストが振り分けられます。

バインド関係が登録されていないオブジェクトを使用し、サーバアプリケーションの各プロセスに登録されているバインド数がすべて等しい場合は、通常のCORBAサーバアプリケーションと同じ論理でリクエストは振り分けられます。
バインド関係が登録されていないオブジェクトを使用し、バインド数が異なる場合は、バインド数が最も少ないプロセスにリクエストが振り分けられます。このとき、振り分け先のプロセスのリクエスト処理スレッドが空いていない場合、他のプロセスに空きスレッドがあっても、振り分け先のプロセスのリクエスト処理スレッドが空くまでリクエストは滞留します。

例

リクエストが滞留する場合の例

以下のような場合にリクエストが滞留します。

サーバアプリケーションの各プロセスのバインド数が以下の場合とします。

サーバアプリケーションプロセスA:バインド数3

サーバアプリケーションプロセスB:バインド数2

サーバアプリケーションプロセスC:バインド数2

上記のような状態で、クライアントからバインド関係が登録されていないオブジェクトを使用した場合、プロセスBにリクエストが振り分けられますが、バインド数に変化がないため、次のバインド関係が登録されていないオブジェクトを使用したリクエストもプロセスBに振り分けられます。このとき、プロセスBのリクエスト処理スレッドが空いていない場合、リクエストが滞留することになります。

リクエストの滞留が起きないようにするには、バインド関係が登録されていないリクエストを受け付けるサーバアプリケーションを作成し、サーバアプリケーションへのリクエストを別々に行うようにしてください。

11.2.5 セションタイムアウト機能

サーバアプリケーションのCORBAアプリケーション情報定義の “ssn_timeout”パラメタを指定することにより、クライアントからのアクセスが一定時間ない場合に、オブジェクトとプロセスのバインド関係を解除できます。
セションタイムアウトが発生した場合、オブジェクトに対応するインスタンスがすでに生成されている場合、デストラクタが動作しインスタンスを解放します。デストラクタ内でCORBA::Object::check_ssn_timeout()を発行することにより、デストラクタが発行された契機を判定可能です。

なお、セションタイムアウトのタイマ値は、他のタイマ(period_receive_timeoutなど)との大小関係を意識する必要はありません。

11.2.6 プロセスの終了

プロセスが異常終了した場合

オブジェクトとプロセスのバインド関係が登録された状態で該当プロセスが終了した場合、CORBAサービスで管理しているバインド関係の情報は自動的に削除されます。

ワークユニット運用時の停止操作

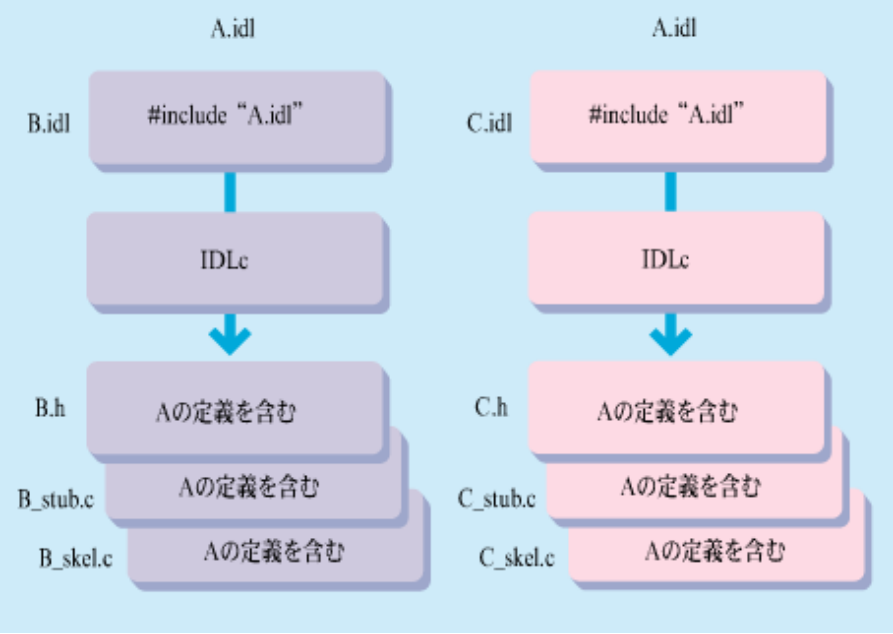
セション継続中に、isstopwuコマンドによりワークユニットの停止要求が行われた際、通常停止の場合、エラーとなります。同期停止／強制停止の場合、ワークユニットは停止します。このとき、セション管理用テーブルに登録されている該当プロセスの情報は、自動的に削除されます。

11.3 複数のIDLファイルのコンパイル方法

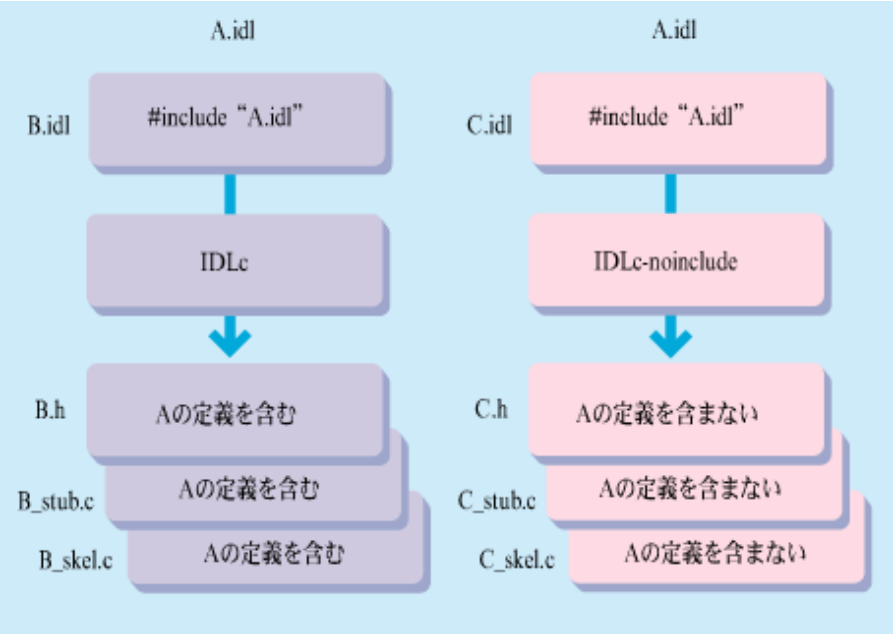
複数のIDLファイルを操作する場合、IDLファイルに#includeを使用する方法があります。ここでは、#includeを使用する際の問題点および対策として使用する-noincludeオプションについて説明します。

11.3.1 #include文

IDLファイルで#include文を使用して他のIDLファイルをインクルードし、IDLコンパイラ(IDLc)でコンパイルすると、生成されたソースにインクルードしたIDLファイルの定義を含むことができます。しかし、複数のIDLファイルから同一のIDLファイルをインクルードしていると、同一の定義がそれぞれのスタブ・スケルトンに生成されるため、リンケージ時にリンクエラーが発生します。
#include文を含むIDLファイルのコンパイルについて、以下の図に示します。



複数のIDLファイルから同一のIDLファイルをインクルードしている際に発生するリンクエラーを防止する対策として、IDLコンパイラ(IDLc)の-noincludeオプションを使用します。これにより、#include文で定義されたIDLファイル（以降、インクルードファイルという）を展開しないため、スタブ・スケルトンの生成時には、ヘッダ定義として“#include XX.h”(IDLファイル名をXX.idlとする)だけが記述されます。なお、インタフェースリポジトリには、IDLコンパイラ実行時に指定されたIDLファイルの情報だけが登録され、インクルードファイルの情報は登録されません。
-noincludeオプションを指定しない場合と指定した場合のマッピング結果を以下の図に示します。



-noinclude機能を使用するケース

-noincludeオプションを指定することにより、以下の多重定義を避けることができます。

(1) マッピングする場合

- インクルードファイルがすでにIDLコンパイル済みであり、IDLファイルの生成物とリンクする場合。
- 複数のIDLファイルで同一のインクルードファイルを定義しており、生成物同士をリンクする場合。

(2) インタフェースリポジトリに登録する場合

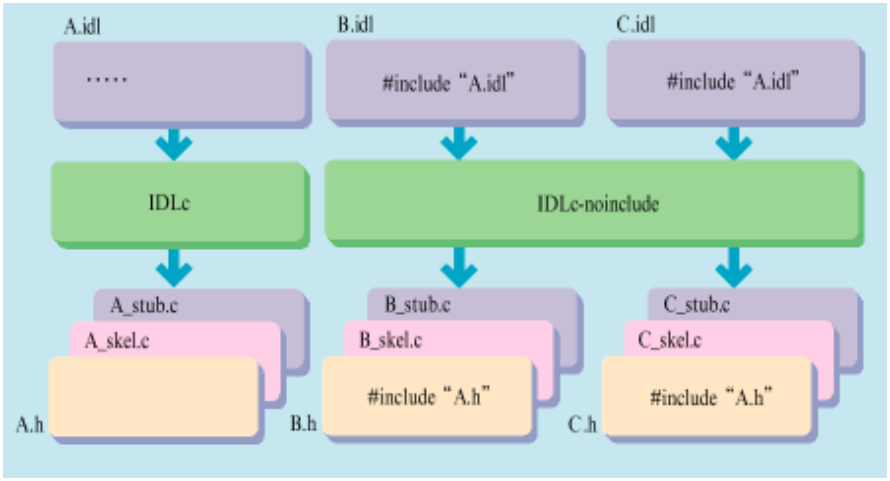
- すでにインタフェースリポジトリにインクルードファイルの情報が登録されている状態で、IDLファイルの情報だけをインタフェースリポジトリに登録する場合。
- 複数のIDLファイルで同一のインクルードファイルを定義している状態で、すべてのIDL定義をインタフェースリポジトリに登録する場合。

11.3.2 -noincludeオプションの使用方法

IDLファイルとインクルードファイルは、以下のように別々にIDLコンパイルを行います。

- インクルードファイルは、単独で-noincludeを指定せずにIDLコンパイルを行います。
- IDLファイルは、-noincludeを指定してIDLコンパイルを行います。

インクルードファイルを含む複数のIDLファイルでのIDLコンパイラの実行結果について、以下の図に示します。



注意

- IDLコンパイラはインクルードファイルの構文解析も行い、構文上のエラーやIDLファイルの記述との衝突チェックも行います。このため、インクルードファイルに構文上の問題がある場合は、前もって修正する必要があります。
- -noincludeオプションを-R／-aオプションと併用（インタフェースリポジトリに登録）する場合、IDLコンパイラはインクルードファイルに記述された識別子がインタフェースリポジトリに登録されているかをチェックします。このため、インクルードファイルはすでにインタフェースリポジトリに登録されている必要があります。
- -noincludeと-update／-deleteは併用できません。このため、IDLファイルの情報を変更する場合は、"-R -delete"で定義情報をインタフェースリポジトリからいったん削除し、再度、変更した情報を-R／-aで登録します。このとき、インクルードファイルの情報も削除されるため、先にインクルードファイルの情報を-R／-aにより登録する必要があります。
- IDLコンパイラで-noincludeオプションを指定して生成したソースファイルを、Visual C++(R)でコンパイルを実行した場合、以下のエラーメッセージが表示されることがあります。

```
xxxx : error LNK2001: 外部シンボル "_xxxxx" は未解決です
```

この場合、IDLコンパイラに-noincludeオプションを指定せずにソースファイルを生成し、リンクオプションに/FORCEオプションを追加して、再度コンパイルを実行してください。このとき、Visual C++(R)でプログラム全体の最適化なしを指定してください。

エラーの原因と対処

-noincludeオプションを指定した場合にIDLコンパイラから表示されるエラーメッセージ、およびその原因／対処を以下に示します。

1	現象	"identifier redeclaration of ..."のエラーが発生します。
	原因	1.IDLファイルの定義とインクルードファイルの定義が衝突しています。 2.-a／-Rと併用した場合、すでにIDLファイルの定義が登録されています。
	対処	1.衝突する定義を避けるため、識別子の名前を変更するなどの修正を行います。 2.IDLファイルの情報を変更する場合は、-noincludeと-delete／-updateは併用できません。-deleteでいったん情報を削除し、インクルードファイルのIDLコンパイルからやり直します。
2	現象	-Rなど指定時に"XXX is not installed."のエラーが発生します。
	原因	インタフェースリポジトリにインクルードファイルの定義が登録されていません。
	対処	前もってインタフェースリポジトリにインクルードファイルを登録しておきます。

3	現象	その他の構文エラーが発生します。
	原因	IDL構文上の問題があります。
	対処	メッセージに該当するIDLの定義を修正します。

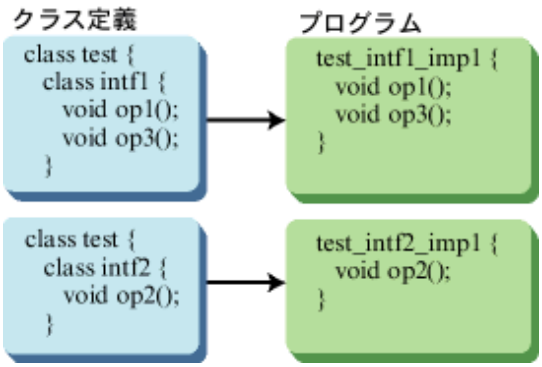
11.4 複数インタフェースの1プロセスへの実装

11.4.1 同一プログラム内の別オブジェクトの呼び出し

同一プログラム内に複数のオブジェクトが実装されている場合のプログラミング方法について説明します。

```
// IDL
module test {
    interface intf1 {
        void op1();
        void op3();
    };
    interface intf2 {
        void op2();
    };
};
```

C++言語では、サーバアプリケーションの実装部は、以下のように各implクラスのメンバ関数としてマッピングされます。



上記IDL定義では、オブジェクトのメソッド呼び出す場合、以下の2とおりの方法があります。

- 同じオブジェクト内のメソッドop3を呼び出す場合、this->op3()のように記述します。
- 同じプログラム内の別オブジェクトのメソッドを呼び出す場合、new演算子を使ってimplクラスのインスタンスを作成し、そのインスタンスを呼び出します。なお、処理終了後は、delete演算子によりインスタンスを削除する必要があります。

```
test_intf1_impl::op1()
{
      this->op3();

      obj = new test_intf2_impl();
      obj->op2();
      delete obj;
}
```

11.4.2 1プロセス－1インタフェースの場合

IDLコンパイラがヘッダファイルにインプリメンテーションおよびインタフェースのリポジトリIDを出力します。

例

```
// IDL
module mod1 {
    interface intf1 {
        ...
    };
};

// *.h
#define _IMPL_mod1_intf1    "IDL:mod1/intf1:1.0"
#define _INTF_mod1_intf1    "IDL:mod1/intf1:1.0"
```

(1) インプリメンテーションの登録／活性化

OD_impl_instコマンドにより、上記インプリメンテーションリポジトリIDで登録します。

```
OD_impl_inst -a -r IDL:mod1/intf1:1.0 ...
```

サーバプログラムで上記インプリメンテーションリポジトリIDの活性化を行います。

```
impl = FJ_ImplementationRep_lookup_id( impl_rep,
      _IMPL_mod1_intf1,
      &env );
CORBA_BOA_impl_is_ready(boa, impl, &env );
```

11.4.3 1プロセス－複数インタフェースの場合

IDLコンパイラがヘッダファイルにインプリメンテーションおよびインタフェースのリポジトリIDを出力します。

例

```
// IDL
module mod1 {
    interface intf1 {
        ...
    };
};
    module mod2 {
    interface intf2 {
        ...
    };
};

// *.h
#define _IMPL_mod1_intf1        "IDL:mod1/intf1:1.0"
#define _INTF_mod1_intf1        "IDL:mod1/intf1:1.0"
#define _IMPL_mod2_intf2        "IDL:mod2/intf2:1.0"
#define _INTF_mod2_intf2        "IDL:mod2/intf2:1.0"
```

上記のIDLの例では、インプリメンテーションリポジトリIDが2つ出力されます。
2つのインタフェースを1プロセスで実装する場合は、そのプロセスを表すインプリメンテーションリポジトリIDを1つ決めます。“IDL:*/*:

1.0”の形式でなくてもかまいません。
このプロセスを表すインプリメンテーションリポジトリIDを“IDL:multi-interfaces:1.0”とした場合の例を以下に示します。

(1) インプリメンテーションの登録／活性化

OD_impl_instコマンドを使用して上記プロセスを表すインプリメンテーションリポジトリIDで登録します。
複数のインタフェースを実装するため、-axオプションの定義ファイルを使用して、実装するすべてのインタフェースリポジトリIDを定義する必要があります。

```
OD_impl_inst -ax def
```

defファイルの内容

```
rep_id              = IDL:multi-interfaces:1.0
type                = ..
                    :
IDL:mod1/intf1:1.0  =
IDL:mod2/intf2:1.0  =
                    :
```

サーバプログラムでこのプロセスを表すインプリメンテーションリポジトリIDの活性化を行います。

```
impl = FJ_ImplementationRep_lookup_id (
               impl_rep,
               "IDL:multi-interfaces:1.0",
               &env );
CORBA_BOA_impl_is_ready(boa, impl, &env );
```

(2) ネーミングサービスへの登録

OD_or_admコマンドを使用して、上記インプリメンテーションの登録時に指定したリポジトリIDでネーミングサービスに登録するオブジェクトリファレンスを作成します。

```
OD_or_adm -a IDL:multi-interfaces:1.0 -c IDL:mod1/intf1:1.0 -n mod1::intf1
OD_or_adm -a IDL:multi-interfaces:1.0 -c IDL:mod2/intf2:1.0 -n mod2::intf2
```

または、サーバプログラムを使用して、上記のOD_or_admコマンドと同様の処理を行います。

```
impl = FJ_ImplementationRep_lookup_id (
      impl_rep,
      "IDL:multi-interfaces:1.0",
      &env );

intf1 = CORBA_Repository_lookup_id(
      intf_rep,
      _INTF_mod1_intf1,
      &env );
intf2 = CORBA_Repository_lookup_id(
      intf_rep,
      _INTF_mod2_intf2,
      &env );

/* mod1/intf1のオブジェクトの作成 */
obj1 = CORBA_BOA_create( boa, &id, intf1, impl, &env );

/* mod1/intf1のネーミングサービスへの登録 */
name1._length = name1._maximum = 1;
name1._buffer = &name1_component;
name_component1.id = "mod1::intf1";
name_component1.kind = "";
```

```
CosNaming_NamingContext_bind(
      cos_naming,
      &name1,
      obj1,
      &env);

/* mod2/intf2のオブジェクトの作成 */
obj2 = CORBA_BOA_create( boa, &id, intf2, impl, &env );

/* mod2/intf2のネーミングサービスへの登録 */
name2._length = name2._maximum = 1;
name2._buffer = &name2_component;
name_component2.id = "mod2::intf2";
name_component2.kind = "";

CosNaming_NamingContext_bind(
      cos_naming,
      &name2,
      obj2,
      &env);
```

(3) クライアントからのアクセス

インタフェース mod1/intf1 へは、名前“mod1::intf1”でネーミングサービスから取り出したオブジェクトを使用します。インタフェース mod2/intf2 へは、名前“mod2::intf2”でネーミングサービスから取り出したオブジェクトを使用してリクエストを発行します。

11.5 ライブラリの作成方法

アプリケーションを作成する場合、オブジェクトを1つのロードモジュールにリンクする方法と、オブジェクト単位にライブラリ(soファイル、DLLファイル)化する方法の2つがあります。ここでは、ライブラリの作成方法について説明します。

11.5.1 ライブラリの作成方法

(1) IDL定義の例

1つのモジュール内に2つのインタフェースが定義されている場合を例に説明します。

[X.idl]

```
module M {
        interface I1 {
                ...
        };
        interface I2 {
                ...
        };
};
```

(2) Cマッピングの場合

1. ライブラリ作成用のオプションを指定して、IDLコンパイルを行います。

```
IDLc -dy IDLファイル名
```

2. IDLファイルのインタフェース定義から生成されたスケルトン(*_skel.c)と、インタフェースのメソッド関数を実装したソースから、インタフェースを実装したライブラリ(*.so／*.dll)を作成します。実装部のコーディングは、ライブラリ化しない場合と同様です。

3. ライブラリは、以下の手順で作成します。

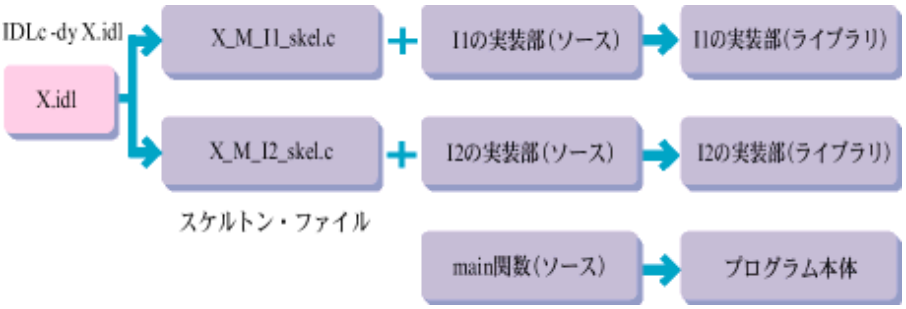
Windows32/64
Visual C++(R)のプロジェクト新規作成で“DLL(ダイナミックリンクライブラリ)作成”を指定します。

Solaris32/64

```
cc -c *.c ....
cc -G -Kpic -o ライブラリ名 オブジェクトファイル(*.o) ....
```

Linux32/64

```
gcc -c *.c ....
gcc -shared -fPIC -o ライブラリ名 オブジェクトファイル(*.o) ....
```



(3) C++マッピングの場合

1. ライブラリ作成用のオプションを指定して、IDLコンパイルを行います。

Windows32/64

```
IDLc -dy -vcpp IDLファイル名
```

Solaris32/64 Linux32/64

```
IDLc -dy -cpp IDLファイル名
```

2. IDLファイルのインタフェース定義から生成されたスケルトン(*_skel_c++.C)と、インタフェースのメソッド関数を実装したソース、およびIDLファイルから生成されたCDRソース(*_cdr.c)、クラスソース(*_c++.C)から、インタフェースを実装したライブラリ(*.so／*.dll)を作成します。実装部のコーディングは、ライブラリ化しない場合と同様です。

3. ライブラリは、以下の手順で作成します。

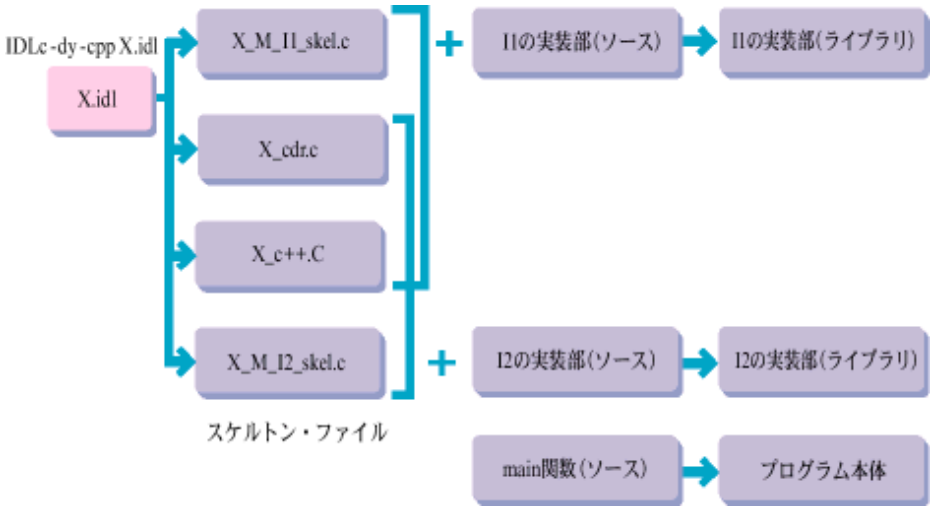
Windows32/64
Visual C++(R)のプロジェクト新規作成で、“DLL(ダイナミックリンクライブラリ)作成”を指定します。

Solaris32/64

```
cc -G -Kpic -o ライブラリ名 オブジェクトファイル(*.o) ....
```

Linux32/64

```
g++ -shared -fPIC -o ライブラリ名 オブジェクトファイル(*.o) ....
```



(4) COBOLマッピングの場合

1. IDLコンパイルを行います。

```
IDLc -cobol IDLファイル名
```

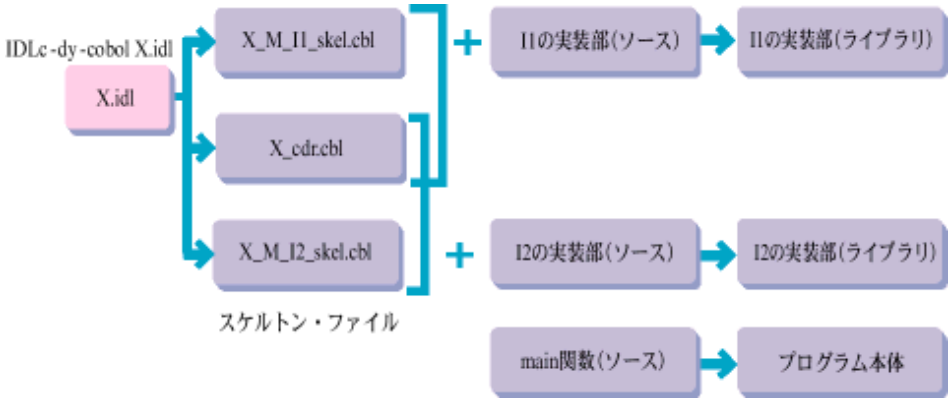
2. IDLファイルのインタフェース定義から生成されたスケルトン(*_skel.cbl)と、インタフェースのメソッド関数を実装したソース、およびIDLファイルから生成されたCDRソース(*_cdr.cbl)から、インタフェースを実装したライブラリ(*.so／*.dll)を作成します。
3. ライブラリは、以下の手順で作成します。

Windows32/64

COBOL開発環境のプロジェクトマネージャで、ライブラリを作成します。

Solaris32/64

```
cobol -c *.cbl ....
cc -G -Kpic -o ライブラリ名 オブジェクトファイル(*.o) ....
```

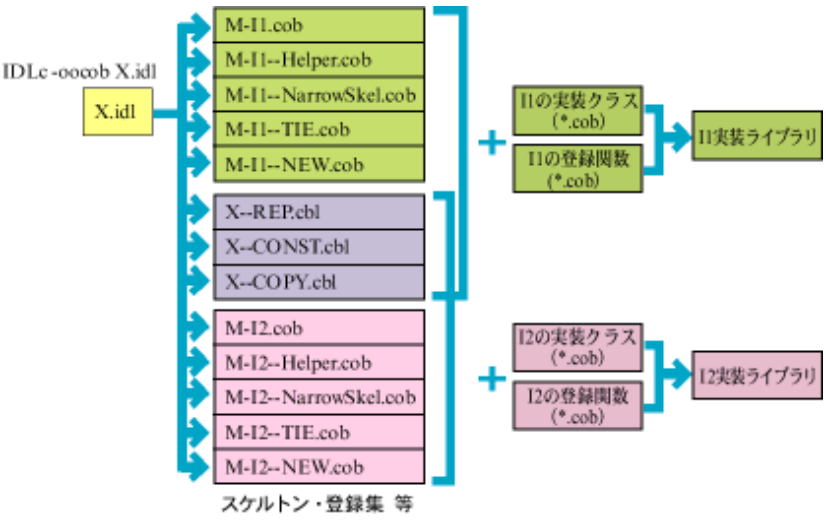


(5) OOCOBOLマッピングの場合 Windows32 Solaris32 Linux32

1. IDLコンパイルを行います。

```
IDLc -oocob IDLファイル名
```

2. IDLファイルのインタフェース定義から生成されたスケルトンなどのクラス(インタフェース名*.cob)と、インタフェースのメソッド関数を実装したソース、登録関数を定義したソース、およびIDLファイルから生成された登録集(*.cbl)から、インタフェースを実装したライブラリ(*.dll)を作成します。



(6) アプリケーションの登録

作成したライブラリをインプリメンテーションリポジトリに登録するため、以下のコマンドを実行します。

```
OD_impl_inst -ax 定義ファイル
```

定義ファイルには、各アプリケーションの構成などを以下のように記述します。

```
rep_id       = インプリメンテーションリポジトリID
type         = サーバタイプ
binary       = プログラム本体のパス
IDL:M/I1:1.0 = I1の実装部ライブラリのパス （注）
IDL:M/I2:1.0 = I2の実装部ライブラリのパス
```

注）“インタフェースリポジトリID=ライブラリパス[,プレフィックス]”の形式で、インタフェース数だけ記述します。プレフィックスは、IDLコンパイラの-Sオプションで指定したプレフィックスを指定します。

11.5.2 インタフェースの継承とライブラリ

インタフェースの継承とライブラリの関係について説明します。

(1) IDL定義の例

次のようにインタフェースAをインタフェースBが継承している場合を考えます。

[Y.idl]

```
interface A {
        ....
};
```

[Z.idl]

```
#include "Y.idl"
interface B : A {
          ....
};
```

(2) プログラムの作成

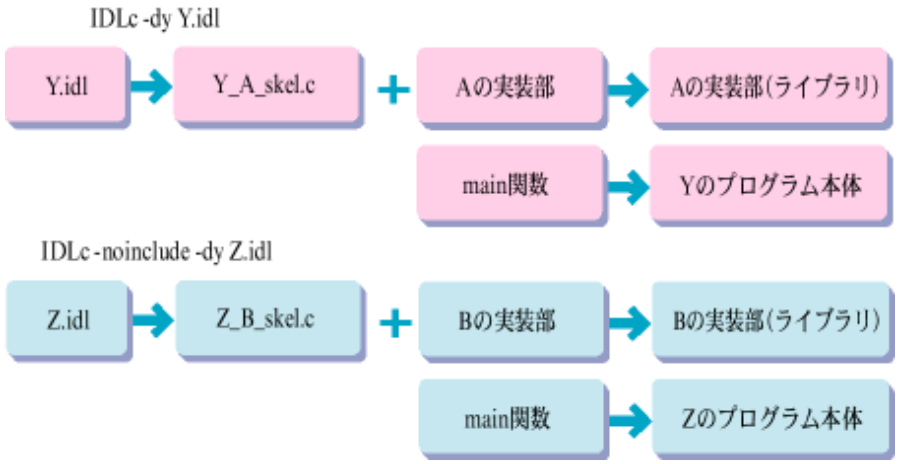
1. 継承元のIDLファイルのIDLコンパイルは、継承がない場合と同様の方法で行います。

```
IDLc -dy [-C|-cpp|-cobol|-oocob] IDLファイル
```

2. 継承先のIDLファイルのIDLコンパイルは、-noincludeオプションを指定して行います。

```
IDLc -dy -noinclude [-C|-cpp|-cobol|-oocob] IDLファイル
```

3. それぞれのインタフェースに対して実装部であるライブラリを作成し、またプログラム本体もそれぞれ作成します。



(3) インプリメンテーションリポジトリへの登録

継承元／継承先のアプリケーションをそれぞれ以下のように登録します。

継承元

```
OD_impl_inst -ax Yの定義ファイル
```

[Yの定義ファイル]

```
rep_id    = YのインプリメンテーションリポジトリID
...
IDL:A:1.0 = Aの実装部ライブラリのパス
...
```

継承先

```
OD_impl_inst  -ax  Zの定義ファイル
```

[Zの定義ファイル]

```
rep_id    = ZのインプリメンテーションリポジトリID
...
IDL:A:1.0 = Aの実装部ライブラリのパス （注1）
```

```
IDL:B:1.0 = Bの実装部ライブラリのパス,,IDL:A:1.0 (注2)
...
```

注1) 継承元インタフェースの定義を記述します。形式は、以下のとおりです。

```
インタフェースリポジトリID=ライブラリパス,[プレフィックス]
```

注2) 継承先インタフェースの定義を記述します。形式は、以下のとおりです。継承元のインタフェースが複数ある場合は、インタフェースリポジトリIDをカンマ(,)で区切って複数記述します。

```
インタフェースリポジトリID=ライブラリパス,[プレフィックス],継承元インタフェースリポジトリID
```

例

Windows32/64

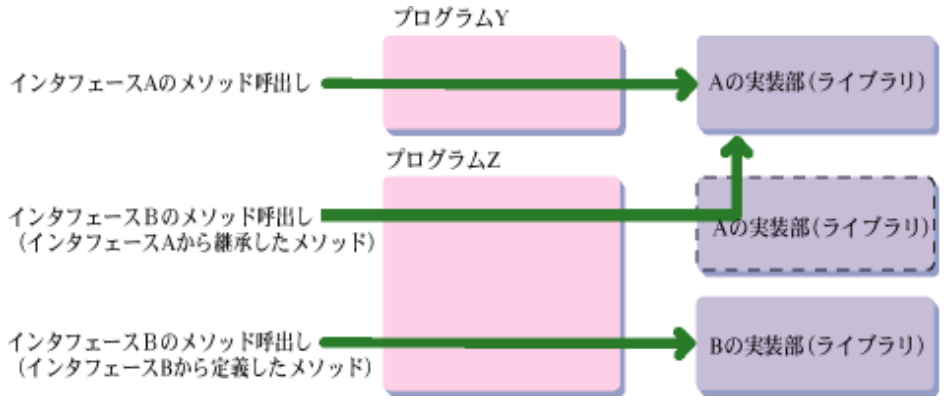
```
IDL:a:1.0 = C:¥server¥libA.dll,,IDL:b:1.0,IDL:c:1.0
```

Solaris32/64 Linux32/64

```
IDL:a:1.0 = /usr/prog/liba.so,,IDL:b:1.0,IDL:c:1.0
```

(4) メソッドの呼び出しイメージ

IDL継承機能を用いて作成したライブラリに対するメソッドの呼び出しイメージを以下に示します。



11.5.3 ライブラリ作成時の注意事項

(1) インタフェースが依存関係にある場合(C++マッピング)

別々のライブラリとして実装するインタフェース間で、一方のインタフェースをもう一方のインタフェースでパラメタやデータ型のメンバとして使用する場合は、ライブラリの作成時に注意する必要があります。

例

例1: IDL定義

```
interface A {
    ...
};
interface B {
    long  op1(  in  A  param  );      /* インタフェースAを使用 */
```

```
    ...
};
```

インタフェースBのライブラリ作成時には、-lオプションでAのライブラリ名を指定する必要があります。

Solaris32/64

```
CC -G -Kpic -o Bのライブラリ名 オブジェクトファイル(*.o) ... -l Aのライブラリ名
```

Linux32/64

```
gcc -shared -fPIC -o Bのライブラリ名 オブジェクトファイル(*.o) ... -l Aのライブラリ名
```

また、アプリケーションの実行時には、Aのライブラリが環境変数LD_LIBRARY_PATHに含まれている必要があります。

例

例2: IDL定義

```
interface C {
        long  op1( in D param );        /* インタフェースDを使用 */
};
interface D {
        long  op2( in C param );        /* インタフェースCを使用 */
};
```

インタフェースをパラメタやでデータ型のメンバとして相互に使用する場合、インタフェースCとDを実装するライブラリは1つのライブラリとして作成する必要があります。

11.5.4 サーバアプリケーションのライブラリ化の例

■C言語／C++言語の場合

サーバのメソッド部をIDLファイルのインタフェース定義単位で分割し、ライブラリ化する場合、以下の点に注意してください。
1つのモジュール内に1つのインタフェースが定義されている場合を例に説明します。

[X.idl]

```
module M {
    interface I1 {
        ...
    };
};
```

(1) IDLファイルのコンパイル

IDLファイルをコンパイルしてスタブ・スケルトンを生成する場合、以下のオプションを指定する必要があります。

C言語の場合

```
IDLc -dy X.idl
```

C++言語の場合

```
IDLc -dy -cpp X.idl
```

-dyオプションを指定することで、IDLファイルに定義されたインタフェースごとにスケルトンを生成します。ライブラリは、生成されたスケルトンファイルごとに作成します。

(2) アプリケーションコンパイル時の手順

IDLファイルから生成されたスケルトンをコンパイルして*.oファイルを作成した後、Cコンパイラのライブラリ作成オプション(-Gなど)を指定することで、ライブラリを作成します。

Solaris32/64

```
cc -I$OD_HOME/include  -D_REENTRANT -DNeedFunctionPrototypes
   -lsocket -lnsl -lthread -L$OD_HOME/lib -lOM -o X_M_I1_skel.o  -c X_M_I1_skel.c
cc -G -Kpic -I$OD_HOME/include  -D_REENTRANT -DNeedFunctionPrototypes
   -lsocket -lnsl -lthread -L$OD_HOME/lib -lOM -o libM_I1.so  X_M_I1_skel.o
```

Linux32/64

```
gcc -I$OD_HOME/include  -D_REENTRANT -DNeedFunctionPrototypes
    -Wl -E -lnsl -lpthread -L$OD_HOME/lib -lOM -o X_M_I1_skel.o  -c X_M_I1_skel.c
gcc -shared -fPIC -I$OD_HOME/include  -D_REENTRANT -DNeedFunctionPrototypes
    -Wl -E -lnsl -lpthread -L$OD_HOME/lib -lOM -o libM_I1.so  X_M_I1_skel.o
```

(3) OD_impl_instでの登録

IDLファイルから作成したライブラリをOD_impl_instコマンドの定義ファイルで指定します。

```
OD_impl_inst -ax X_def
```

X_def:定義ファイル

定義ファイルX_defに“IDL:M/I1:1.0 = /user/libM_I1.so”を指定することにより、実際にクライアントからの要求時に/user/libM_I1.soが読み込まれ、サーバメソッドが呼び出されます。

(4) 継承を使用する場合のOD_impl_instでの登録

継承機能を使用する場合は、OD_impl_instコマンドの定義ファイルに以下を指定します。

例

Solaris32/64 Linux32/64

モジュール名:M、インタフェース名:I1、継承名:I2の場合

```
IDL:M/I1:1.0 = /user/lib/libM_I1.so
IDL:M/I2:1.0 = /user/lib/libM_I2.so,,IDL:M/I1:1.0
```

■COBOLの場合

サーバのメソッド部をIDLファイルのインタフェース定義単位で分割し、ライブラリ化する必要があります。このとき、以下の点に注意してください。
1つのモジュール内に1つのインタフェースが定義されている場合を例に説明します。

[X.idl]

```
module M {
     interface I1 {
          ...
     };
};
```

(1) IDLファイルのコンパイル

IDLファイルをコンパイルしてスタブ／スケルトンを生成する場合、以下のオプションを指定する必要があります。

```
IDLc -cobol IDLファイル名
```

(2) アプリケーションコンパイル時の手順

IDLファイルから生成されたスケルトンをコンパイルして*.oファイルを作成した後、COBOLコンパイラのライブラリ作成オプション(-Gなど)を指定することで、ライブラリを作成します。

```
cobol  -G  -o libX_cdr.so  X_cdr.cbl
cobol  -G  -o libX_skel.so  X_skel.cbl
cobol  -G  -o libX_M_I1_skel.so  X_M_I1_skel.cbl
cobol  -G  -lcobol -L$OD_HOME/lib -lOMcbl -o libM-I1.so -lX_cdr -lX_skel -lX_M_I1_skel M_I1.cbl
```

注意

- 環境変数LD_LIBRARY_PATHに、COBOLのランタイムlibcobol.soの格納ディレクトリおよびアプリケーション作成時のカレントディレクトリを指定する必要があります。
- 本製品の提供する登録集を参照するため、あらかじめ環境変数CORBAに以下を設定すると、COBOLソースのコンパイルが容易になります。

  ```
  CORBA = $OD_HOME/include/COBOL
  ```

- COBOLのサーバアプリケーションは、ライブラリの形態で利用されます。ライブラリの名称は、以下のとおりです。ライブラリファイル名のモジュール名とインタフェース名部分は、IDL定義の内容にかかわらず大文字となります。

  ```
  libモジュール名-インタフェース名.so
  ```

  作成するライブラリを構成するモジュールを以下に示します(IDLファイル:X.idl)。

  ```
  libX_cdr.so
  libX_skel.so
  libX_MM_I1_skel.so
  ```

  また、ユーザ作成のサーバ実装部から他の関数をCALLする場合、同様にライブラリ化し、環境変数LD_LIBRARY_PATHなどに設定する必要があります。詳細については、COBOLのマニュアルを参照してください。

- COBOLソースコンパイル時に“-dy”オプションを指定しないでください。

(3) OD_impl_instでの登録

IDLファイルから作成したライブラリをOD_impl_instコマンドの定義ファイルで指定します。

```
OD_impl_inst  -ax  X_def        (X_def:定義ファイル)
```

定義ファイルX_defに“IDL:M/I1:1.0=/user/libM-I1.so”を指定することにより、実際にクライアントからの要求時に/user/libM-I1.soが読み込まれ、サーバメソッドが呼び出されます。

注意

定義ファイルに指定するライブラリのファイル名には、“アプリケーションコンパイル時の手順”で作成したライブラリのファイル名を指定してください。このとき、大文字小文字も一致させてください。

(4) 継承を使用する場合のOD_impl_instでの登録

継承機能を使用する場合、OD_impl_instコマンドの定義ファイルに以下を指定します。

例

モジュール名:M、インタフェース名:I1、継承名:I2の場合

```
IDL:M/I1:1.0 = /user/lib/libM-I1.so
IDL:M/I2:1.0 = /user/lib/libM-I2.so,,IDL:M/I1:1.0
```

(5) COBOLでの“IDL:BAD_OPERATION:1.0”の発生条件について

以下にシステム例外として“IDL:BAD_OPERATION:1.0”が通知された場合のサーバアプリケーションの見直しのポイントを示します。

アプリケーションの誤り

- 存在しない関数を使用
  ライブラリの作成時に存在しない関数を誤ってCALLしている場合、未決シンボルが残ったままでライブラリができます。通常、他のライブラリにより未決シンボルが解決されますが、アプリケーション動作時に解決がされない場合、サーバアプリケーションのメソッド呼出しに失敗して、BAD_OPERATIONが通知されます。

COBOLコンパイルの指定方法誤り

- コンパイルオプションの指定誤り
  COBOLソースのコンパイル時に“-dy”が指定された場合、“-dy”指定で作成されたアプリケーションからlibOMcbl.so／libOMircbl.soが提供するObjectDirectorの各サービス関数が使用できず、COBOLランタイムから以下のメッセージが表示されてサーバアプリケーションの各関数呼出しに失敗し、BAD_OPERATIONが通知されます。

  例） libXXXX.soが見つかりません。

  （XXXX:ObjectDirectorの関数名）

- OD_impl_instコマンドで指定した定義ファイルに記述したサーバメソッド用ライブラリを作成する際にIDLコンパイラから生成された以下のCOBOLソースから作成したライブラリが結合されていない場合、BAD_OPERATIONが通知されます。

  ー IDLファイル名_インタフェース名_skel.so

  ー IDLファイル名_cdr.so

  ー IDLファイル名_skel.so（struct,unionなどの領域獲得関数用ソース）

環境設定の誤り

- インプリメンテーション情報登録の誤り
  OD_impl_instコマンドで指定する定義ファイルの記述に以下の誤りがあり、サーバアプリケーションのメソッド呼出しに失敗し、BAD_OPERATIONが通知されます。

  ー 定義ファイルに記述するリポジトリID(rep_id)がIDLと一致していません。

  ー persistent以外の場合で、環境変数定義(env)に環境変数LD_LIBRARY_PATHが設定されていません。または、COBOLランタイム／サーバアプリケーションの格納先、サーバアプリケーション／メソッドのライブラリの検索パスが定義されていません。

  ー 継承するライブラリの定義が漏れています。

ー リポジトリIDとライブラリの対応付けに誤りがあります。

```
IDL:module/interface:1.0 = ライブラリ名
```

- アプリケーション起動時の環境変数LD_LIBRARY_PATHの設定誤り

  ー 環境変数LD_LIBRARY_PATHに、サーバアプリケーションの格納先、サーバアプリケーション／メソッドのライブラリが定義されていません。

  ー スレッド用libOM.soが有効となっている場合、サーバアプリケーションの起動／メソッドの呼出しに失敗します。COBOL使用時は、以下のパスを指定する必要があります。

```
$OD_HOME/lib/nt
```

11.6 スレッドモード・プロセスモードのアプリケーション

Solaris32/64 Linux32/64

ここでは、スレッドモード・プロセスモードのアプリケーションについて説明します。

11.6.1 スレッドモード・プロセスモードのアプリケーションの作成

(1) スレッドモードの場合

スレッドモードのアプリケーションを作成する場合は、以下のライブラリと結合する必要があります。

ライブラリ(インストールパスはデフォルト)

Solaris32/64

```
libthread.so
/opt/FSUNod/lib/libOM.so(必須)
```

Linux32/64

```
libthread.so
/opt/FJSVod/lib/libOM.so(必須)
```

ポイント

結合されているライブラリは、lddコマンドで確認します。

C／C++アプリケーションのコンパイル時は、-D_REENTRANT宣言が必要です。詳細については、システムのスレッドプログラミングに関するマニュアルなどを参照してください。

(2) プロセスモードの場合

プロセスモードのアプリケーションを作成する場合は、以下のライブラリと結合する必要があります。

ライブラリ(インストールパスはデフォルト)

Solaris32/64

```
/opt/FSUNod/lib/nt/libOM.so(必須)
```

Linux32/64

```
/opt/FJSVod/lib/nt/libOM.so(必須)
```

注意

作成したアプリケーションを動作させるときには、LD_LIBRARY_PATH=$OD_HOME/lib/ntを設定する必要があります。

ポイント

結合されているライブラリは、lddコマンドで確認します。

(3) スレッドアプリケーションからのスレッド生成

スレッドモードのサーバアプリケーションからさらにスレッドを生成する場合、以下のスレッドライブラリを使用します。

Solaris32/64

スレッドライブラリ	thr_create
指定フラグ	THR_NEW_LWP(新しいスレッドをLWPに固定的に結合します。)
	THR_BOUND(非結合スレッドの並行度を1つ増やします。)

設定していない場合は、CORBA_ORB_init()が例外UNKNOWNでエラー復帰します。

Linux32/64

スレッドライブラリ	pthread_create

(4) プロセスアプリケーションからのプロセス生成

プロセスモードのサーバアプリケーションから、さらにプロセスを生成する場合は、fork(システムコール)を使用します。

(5) プロセスアプリケーションの登録時の注意事項

プロセスモードのアプリケーションをshared/unsharedタイプ(persistent以外)としてインプリメンテーションリポジトリに登録する方法について説明します。
アプリケーションの種類に合わせて定義ファイルを作成し、OD_impl_instコマンドの-axオプションで登録します。

```
OD_impl_inst -ax 定義ファイル
```

以下に定義ファイルの例を示します。以下の設定項目は、最低限必要です。他の設定項目は、必要に応じて設定します。

例

定義ファイル例

Solaris32/64

```
rep_id = インプリメンテーションリポジトリID
type   = サーバタイプ
binary = アプリケーションのパス
env    = LD_LIBRARY_PATH=/opt/FSUNod/lib/nt
```

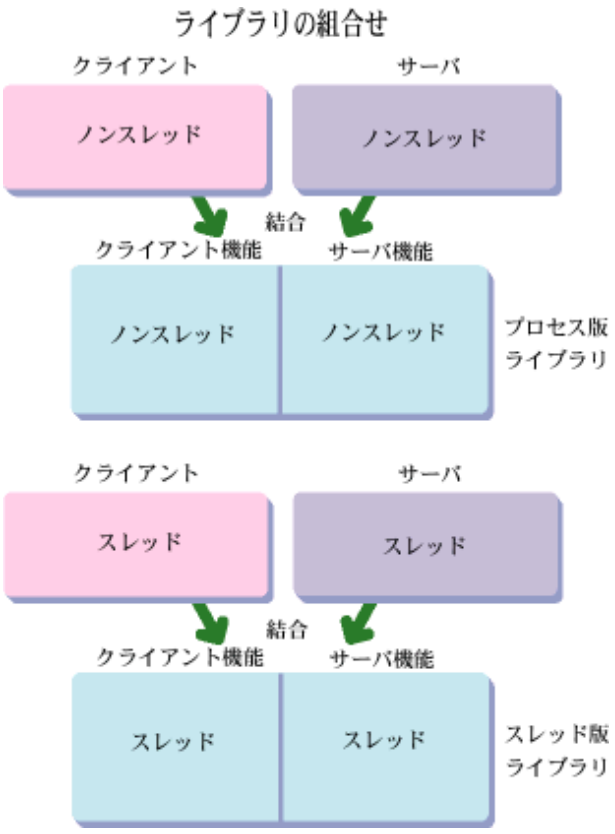
Linux32/64

```
rep_id = インプリメンテーションリポジトリID
type   = サーバタイプ
binary = アプリケーションのパス
env    = LD_LIBRARY_PATH=/opt/FJSVod/lib/nt
```

11.6.2 サーバ・クライアント同時実装時の注意事項

アプリケーションのモードについて

アプリケーションにサーバ機能とクライアント機能を同時に実装する場合、アプリケーションのモードであるスレッドモードとプロセスモードは結合できないため、どちらか一方の形態となります。アプリケーションのモードでの組合せは、以下の図のように2種類あります。



スレッド版ライブラリ	Solaris32/64 /opt/FSUNod/lib/libOM.so	クライアント／サーバ共にスレッドライブラリを使用するCOREライブラリ
	Linux32/64 /opt/FJSVod/lib/libOM.so	
プロセス版ライブラリ	Solaris32/64 /opt/FSUNod/lib/nt/libOM.so	クライアント／サーバ共にスレッドライブラリを使用しないCOREライブラリ
	Linux32/64 /opt/FJSVod/lib/nt/libOM.so	

関数名の重複（スタブ／スケルトン）について

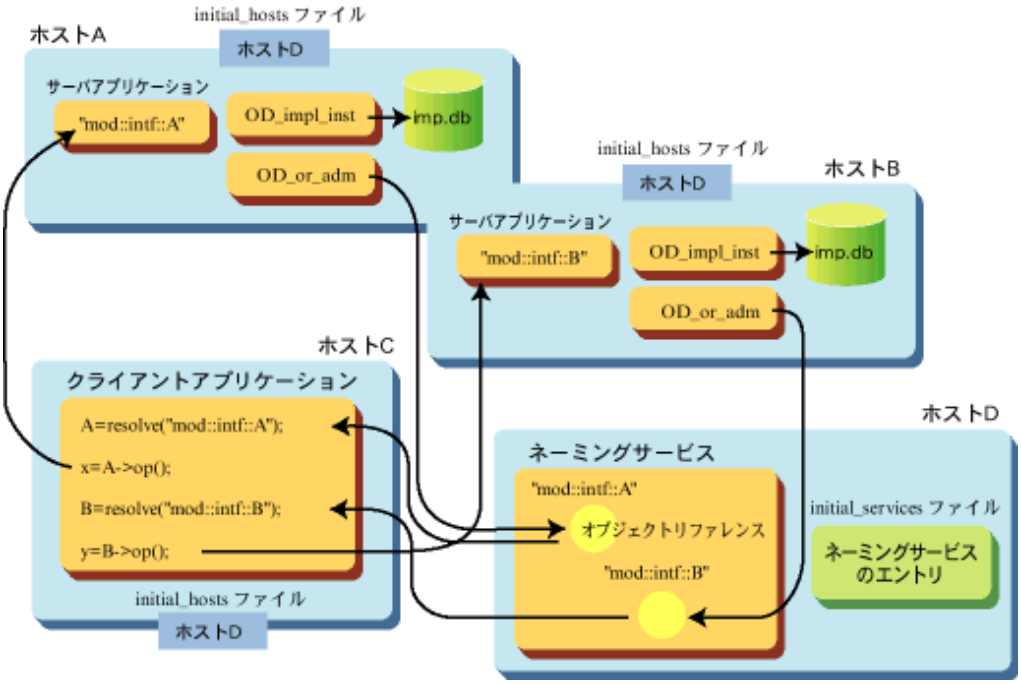
アプリケーションにサーバ機能とクライアント機能を同時に実装する場合、同一の関数名がスタブ／スケルトンに生成されることがあります。これらを結合すると二重定義となるため、IDLファイル（スタブ／スケルトン）の作成は注意してください。

11.7 サーバアプリケーションの複数ホストへの配置

同じサーバアプリケーションを複数のホスト上に分散配置し、クライアントアプリケーションからそれぞれのサーバアプリケーションにアクセスする場合のプログラミング／環境設定について説明します。

11.7.1 環境設定

各ホストでの環境設定について説明します。
OD_or_admコマンドでの登録、resolve関数によるオブジェクトリファレンスの獲得で、同じネーミングサービスへアクセスして情報の整合性を保つため、以下の図に示すように、各ホストのinitial_hostsにネーミングサービスが動作しているホスト名を設定します。



11.7.2 サーバアプリケーションのプログラミング

同じサーバアプリケーションを複数のホスト上に分散配置する場合と配置しない場合とで、サーバアプリケーションのプログラミング方法に違いはなく、複数ホスト上に配置されることを意識する必要はありません。

11.7.3 サーバアプリケーションの登録

作成したサーバアプリケーションをObjectDirectorに登録する方法について説明します。

インプリメンテーションリポジトリへの登録

インプリメンテーションリポジトリへサーバアプリケーションの実装情報を登録します。
OD_impl_instコマンドにより、サーバアプリケーションを配置する各ホスト上で登録します。

sharedタイプの場合

```
OD_impl_inst -a -r  IDL:mod/intf:1.0 -t S -f /home/prog/prog_s
                              1                      2
```

1. リポジトリID
   同じサーバアプリケーションのため、同じリポジトリIDを指定します。

2. アプリケーションのパス
   各ホスト上のサーバアプリケーションのパスを指定します。

ネーミングサービスへの登録

ネーミングサービスへサーバアプリケーションのオブジェクトリファレンスを登録します。
OD_or_admコマンドにより、サーバアプリケーションを配置する各ホスト上で登録します。

```
OD_or_adm -c IDL:mod/intf:1.0 -n name
                  1           2
```

1. リポジトリID
   同じサーバアプリケーションのため、同じリポジトリIDを指定します。

2. オブジェクトリファレンスの名前
   各ホスト上のサーバアプリケーションを表す名前をそれぞれ指定します。この名前により、クライアントアプリケーションは各ホスト上のサーバアプリケーションを区別します。

11.7.4 クライアントアプリケーションのプログラミング

各ホスト上のサーバアプリケーションのオブジェクトリファレンスにそれぞれ名前が付けられて、ネーミングサービスに登録されています。クライアントアプリケーションは、この名前でオブジェクトリファレンスを獲得し、サーバアプリケーションにアクセスします。
プログラミング例を以下に示します。

```
CosNaming_NamingContext  naming;
CosNaming_Name           nameA, nameB;
CosNaming_NameComponent  namecpA, namecpB;

/* オブジェクトリファレンス"mod::intf::A" 獲得 */
nameA._length = nameA._maximum = 1;
nameA._buffer = &namecpA;
namecpA.id    = "mod::intf::A";
namecpA.kind  = "";
objA          = CosNaming_NamingContext_resolve(naming, &nameA, &env );

/* オブジェクトリファレンス"mod::intf::B" 獲得 */
nameB._length = nameB._maximum = 1;
nameB._buffer = &namecpB;
namecpB.id    = "mod::intf::B";
namecpB.kind  = "";
objB          = CosNaming_NamingContext_resolve(naming, &nameB, &env );
```

11.8 ファイルによるオブジェクトリファレンスの配布

ここでは、オブジェクトリファレンスをファイルで配布する方法について説明します。

オブジェクトリファレンスをファイルで配布することにより、クライアントアプリケーションはネーミングサービスと通信せずにオブジェクトリファレンスを取得できます。このため、サーバアプリケーションのオブジェクトリファレンスにSSL情報が付加されている場合など、ネーミングサービスとの通信とサーバアプリケーションとの通信とでそれぞれ別コネクションを使用する場合は、オブジェクトリファレンスをファイルで配布することにより使用するコネクション数を減らすことが可能となります。

ただし、オブジェクトリファレンスをファイルで配布する場合は、すべてのクライアントマシンにファイルを配布する必要があり、ネーミングサービスを使用する場合より管理上のコストが増える可能性があります。たとえば、サーバアプリケーションのオブジェクトリファレンス更新時に、ネーミングサービスを使用していれば、ネーミングサービスに登録されている情報を更新するだけですが、ファイルで配布しているとすべてのクライアントに再配布する必要があります。

11.8.1 オブジェクトリファレンスのファイルへの出力

OD_or_admコマンドでオブジェクトリファレンスを生成する際に“-o”オプションを指定し、文字列形式のオブジェクトリファレンスをファイルに出力します。

また、アプリケーション内で生成／取得を行ったオブジェクトリファレンスを文字列に変換することにより、ファイルに出力できます。
オブジェクトリファレンスを文字列に変換する各開発言語のAPIを以下に示します。詳細については、“リファレンスマニュアル（API編）”を参照してください。

開発言語	API名
C言語	CORBA_ORB_object_to_string
C++言語	CORBA::ORB::object_to_string
Java	org.omg.CORBA.ORB.object_to_string
COBOL	CORBA-ORB-OBJECT-TO-STRING

例

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -o calc_ior
```

-o cal_ior

出力先のファイル名を指定します。

11.8.2 ファイルの配布

FTPなど任意の方法で、クライアントマシンに文字列形式のオブジェクトリファレンスが設定されたファイルを配布します。

11.8.3 クライアントアプリケーションのプログラミング

クライアントアプリケーションでは、ファイルから読み込んだ文字列をオブジェクトリファレンスに変換し、サーバアプリケーションを呼び出します。

文字列をオブジェクトリファレンスに変換する各開発言語のAPIを以下に示します。詳細については、“リファレンスマニュアル（API編）”を参照してください。

開発言語	API名
C言語	CORBA_ORB_string_to_object
C++言語	CORBA::ORB::string_to_object
Java	org.omg.CORBA.ORB.string_to_object
COBOL	CORBA-ORB-STRING-TO-OBJECT

C言語／C++言語／Javaでは、上記APIを使用したサンプルプログラムを提供しています。サンプルプログラムの作成／実行方法については、“付録D サンプルプログラム”を参照してください。

11.9 通信資源の解放

クライアントアプリケーションは、サーバメソッド呼出し時にサーバとのコネクションを接続します。サーバメソッド呼出しごとにコネクション接続処理が動作しなくても済むように、接続されたコネクションは維持されます。しかし、クライアントとサーバのコネクションが維持された状態は通信資源を使用しているため、サーバ側のクライアントから受付け可能なコネクション数超過の要因にもなります。
ここでは、コネクションを切断し、通信資源を解放する機能について説明します。

サーバ側無通信監視機能

サーバ側無通信監視機能により、サーバにおける無通信状態(クライアントからのリクエスト送信なし)を監視できます。監視時間を超過してもクライアントからのリクエスト送信がない場合、クライアントとのコネクションを切断して通信資源を解放します。なお、インプリメン

テーションリポジトリへの登録時にiswitchパラメタに“ON”が設定されたサーバアプリケーション呼出しに使用したコネクションの場合は、監視時間を超過してもコネクションを切断しません。

CORBAサービスの動作環境ファイル(config)のperiod_idle_con_timeoutパラメタ、またはInterstage管理コンソールの[CORBAサービスの詳細設定]の“サーバ側無通信監視時間”パラメタで、そのシステムにおけるサーバ側無通信監視時間を設定します。詳細については、“チューニングガイド”または“Interstage管理コンソール ヘルプ”を参照してください。

クライアント側無通信監視機能

クライアント側無通信監視機能により、クライアントにおける無通信状態(サーバへのリクエスト送信なし)の監視できます。監視時間を超過してもサーバへのリクエスト送信がない場合、サーバとのコネクションを切断し通信資源を解放します（注）。なお、サーバ側無通信監視機能と異なり、インプリメンテーションリポジトリへの登録時にiswitchパラメタに“ON”が設定されたサーバアプリケーション呼出しに使用したコネクションでも、監視時間を超過した場合はコネクションを切断します。

CORBAサービスの動作環境ファイル(config)のperiod_client_idle_con_timeoutパラメタ、またはInterstage管理コンソールの[CORBAサービスの詳細設定]の“クライアント側無通信監視時間”パラメタで、そのシステムにおけるCORBAアプリケーションのクライアント側無通信監視時間を設定します。詳細については、“チューニングガイド”または“Interstage管理コンソール ヘルプ”を参照してください。Portable-ORBの場合は、porbeditenvコマンドの［無通信監視時間］で設定します。詳細については、“リファレンスマニュアル(コマンド編)”を参照してください。

注）プロセスモードの場合、時間超過のタイミングではコネクションを切断せず、次回リクエスト送信時にサーバとのコネクションの切断／再接続を行います。

APIによる通信資源の解放

通信資源解放用のAPIを発行することにより、クライアントアプリケーションは任意のタイミングで通信資源を解放できます。（注）
たとえば、サーバアプリケーションのオブジェクトリファレンスにSSL情報が付加されているなど、ネーミングサービスとの通信とサーバアプリケーションとの通信とでそれぞれ別コネクションを使用する場合は、サーバアプリケーションのオブジェクトリファレンス取得後にネーミングサービスとの通信用の資源を解放することにより、使用する通信資源量を減らすことが可能です。

通信資源解放用の各開発言語のAPIを以下に示します。詳細については、“リファレンスマニュアル（API編）”を参照してください。

開発言語	API名
C言語	CORBA_ORB_net_disconnect
C++言語	CORBA::ORB::net_disconnect
Java	com.fujitsu.ObjectDirector.CORBA.ORB.net_disconnect
COBOL	CORBA-ORB-NET-DISCONNECT

C言語／C++言語／Javaでは、上記APIを使用したサンプルプログラムを提供しています。サンプルプログラムの作成／実行方法については、“付録D サンプルプログラム”を参照してください。

注）マルチスレッドで動作するアプリケーションの場合、別スレッドでサーバと通信しているときに通信資源解放用のAPIを発行すると、サーバと通信しているスレッドにシステム例外が通知される可能性があります。そのため、別スレッドでサーバと通信していないことを確認してから通信資源解放用のAPIを発行してください。

11.10 出口関数

CORBAサービスでは、C言語／C++言語において、出口関数機能を提供しています。この機能では、CORBAサーバアプリケーションにおいてあらかじめ任意の関数を登録しておくことにより、サーバアプリケーションがクライアントにリプライを送信した後（インタフェース実装関数の処理が完了した後）に、その関数を実行できます。この関数を出口関数と呼びます。

11.10.1 出口関数の登録

出口関数は、CORBAサーバアプリケーションにおいて登録用APIを発行することにより登録します。

出口関数を登録するAPIを以下に示します。詳細については、“リファレンスマニュアル（API編）”を参照してください。

開発言語	API名
C言語	CORBA_ORB_register_reply_interceptor
C++言語	CORBA::ORB::register_reply_interceptor

11.10.2 出口関数の最大処理時間監視

出口関数機能では、出口関数の最大処理時間監視機構を提供しています。この機構では、出口関数が最大処理時間を超えても復帰しない場合はタイムアウトとし、出口関数を発行したCORBAサーバアプリケーションを強制終了します。このとき、od10979メッセージが出力されます。
出口関数の最大処理時間は、300秒(デフォルト)です。任意の時間を指定する場合、または最大処理時間監視機構を無効とする場合は、インプリメンテーションリポジトリにサーバアプリケーションを登録する際にreply_interceptor_timeoutパラメタを設定します。詳細については、“リファレンスマニュアル(コマンド編)”の“OD_impl_inst”を参照してください。

11.10.3 出口関数使用時の注意事項

- インタフェース実装関数にonewayが指定されているなど、CORBAサーバアプリケーションがクライアントへリプライを送信しない場合でも、インタフェース実装関数の処理が完了した後に出口関数が実行されます。
- 出口関数は、以下の形式にする必要があります。

```
void 出口関数名()
```

  また、C++言語において出口関数としてメンバ関数を使用する場合は、public指定の静的メンバ関数とする必要があります。

- 登録可能な出口関数の数は、CORBAサーバアプリケーションごとに一つです。
- 出口関数の最大処理時間監視機構が有効な場合、タイムアウトが発生すると該当のCORBAサーバアプリケーションは強制終了されます。このため、出口関数を使用する際は、アプリケーション自動再起動機能が存在するワークユニット配下でのアプリケーション運用を推奨します。

11.10.4 出口関数の例

本節では、C言語／C++言語のそれぞれについて、出口関数を使用する場合のプログラミング例とその出力結果を示します。

■C言語の場合

(1) プログラミング例

C言語を使用した場合のサーバアプリケーションのプログラミング例を以下に示します。

```
/* 出口関数 */
void
exitfunc()
{
    printf( "exitfunc is called.¥n" );

    return;
}

/* インタフェース実装関数 */
CORBA_long
ODsample_intf1_op1(
    ODsample_intf1      obj,
    CORBA_long          l,
    CORBA_Environment   *env )
{
    printf( "ODsample_intf1_op1 is called.¥n" );
```

```
    return 0;
}

/* main */
int
main( int argc, char *argv[] )
{
    CORBA_ORB  orb;
    CORBA_BOA  boa;
    CORBA_ImplementationDef  impl;
    CORBA_Environment        env;

    orb = CORBA_ORB_init( &argc, argv, FJ_OM_ORBid, &env );

    CORBA_ORB_register_reply_interceptor( orb, exitfunc, &env );

    ...  /* 中略 */

    CORBA_BOA_impl_is_ready( boa, impl, &env );

    return 0;
}
```

(2) 出力結果

上記のサーバアプリケーション(インタフェース実装関数)に対してリクエストを送信した場合、サーバアプリケーションの出力結果は以下のようになります。

```
ODsample_intf1_op1 is called.
exitfunc is called.
```

■C++言語の場合

(1)プログラミング例

C++言語を使用した場合のサーバアプリケーションのプログラミング例(出口関数としてメンバ関数を使用する場合)を以下に示します。

```
/* 出口関数 */
class MyClass
{
public:
    static void exitfunc()
    {
        printf( "MyClass::exitfunc is called.¥n" );

        return;
    }
};

/* インタフェース実装関数 */
CORBA::Long
ODsample_intf1_impl::op1(
    CORBA::Long         l,
    CORBA::Environment  &env )
    throw( CORBA::Exception )
{
    printf( "ODsample_intf1_impl::op1 is called.¥n" );

    return 0;
```

```
}

/* main */
int
main( int argc, char *argv[] )
{
    CORBA::ORB_ptr  orb;
    CORBA::BOA_ptr  boa;
    CORBA::ImplementationDef_ptr  impl;
    CORBA::Environment_ptr        env = new CORBA::Environment;

    try {
        orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, *env );

        orb->register_reply_interceptor( MyClass::exitfunc, *env );

        ...  /* 中略 */

        boa->impl_is_ready( impl, *env );
    }
    catch ( CORBA::SystemException &se ) {
        printf( "SystemException raised!¥n" );
        return 1;
    }
    catch ( CORBA::Exception &e ) {
        printf( "Exception raised!¥n" );
        return 1;
    }

    delete env;

    return 0;
}
```

(2) 出力結果

上記のサーバアプリケーション(インタフェース実装関数)に対してリクエストを送信した場合、サーバアプリケーションの出力結果は以下のようになります。

```
ODsample_intf1_impl::op1 is called.
Myclass::exitfunc is called.
```

11.11 Interstage for GSとの連携

Interstage for GSのCORBA機能と連携する場合は、Interstage for GSのマニュアル(「OS IV Interstageマニュアルセット」)を参照して環境構築を行ってください。

第12章　CORBAインタフェース

本章では、動的にデータを操作するためのインタフェースについて説明します。

12.1 TypeCodeオブジェクト

12.1.1 TypeCodeオブジェクトとは

TypeCodeオブジェクトは、クライアント／サーバ間でデータ型が不定の場合に使用し、あらゆるデータ型を表現することが可能です。以下のケースで使用されます。

- any型の記述
- インタフェースリポジトリ(以降IRと呼ぶ)の復帰値および、動的起動インタフェースを使用する場合

TypeCodeオブジェクトで使用できるデータ型を以下に示します。TypeCodeオブジェクトの属性情報には、整数型、文字列型などを表すデータの識別子と、文字列の最大長などの付加情報から構成されています。

TypeCode オブジェクト	種類・データ型	属性情報	
		識別子	付加情報
TC_null	未定義	tk_null	なし
TC_void	void型	tk_void	なし
TC_ushort	整数型	tk_ushort	なし
TC_short		tk_short	
TC_ulong		tk_ulong	
TC_long		tk_long	
TC_longlong		tk_longlong	
TC_float	浮動小数点型	tk_float	なし
TC_double		tk_double	
TC_longdouble Windows32/64 Solaris32/64		tk_longdouble	
TC_char	文字型	tk_char	なし
TC_wchar	ワイド文字型	tk_wchar	なし
TC_octet	オクテッド型	tk_octet	なし
TC_boolean	ブーリアン型	tk_boolean	なし
TC_string	文字列型	tk_string	最大長
TC_wstring	ワイド文字列型	tk_wstring	最大長
TC_enum (注)	列挙型	tk_enum	データ型のリポジトリID、名前、含まれるデータの名前、数、データの型
TC_any	any型	tk_any	なし
TC_sequence (注)	シーケンス型	tk_sequence	sequenceのメンバのデータ型、最大長
TC_struct (注)	構造体	tk_struct	データ型のリポジトリID、名前、含まれるデータの名前、数、データの型
TC_union (注)	共用体	tk_union	データ型のリポジトリID、名前、含まれるデータの名前、数、データの型、メンバのラベル番号、弁別情報のデータ型

TypeCode オブジェクト	種類・データ型	属性情報	
		識別子	付加情報
TC_fixed (注)	固定小数点型	tk_fixed	桁数、小数点の位置
TC_array (注)	配列	tk_array	arrayのメンバのデータ型、最大要素数
TC_alias (注)	typedefで定義されたデータ型	tk_alias	データ型のリポジトリID、名前
TC_TypeCode	TypeCode	tk_TypeCode	なし
TC_Principal	認証用オブジェクト	tk_Principal	なし
TC_objref	オブジェクトリファレンス	tk_objref	データ型のリポジトリID、名前
TC_except (注)	例外	tk_except	データ型のリポジトリID、名前、含まれるデータの名前、数、データの型

注) TypeCodeオブジェクト名は、IDL定義に基づいて決められます。

12.1.2 TypeCodeインタフェース

TypeCodeは、以下のインタフェースを提供しています。

```
module CORBA{
    enum TCKind{
        tk_null, tk_void,
        tk_short, tk_long, tk_ushort, tk_ulong,
        tk_float, tk_double, tk_boolean, tk_char,
        tk_octet, tk_any, tk_TypeCode, tk_Principal, tk_objref,
        tk_struct, tk_union, tk_enum, tk_string,
        tk_sequence, tk_array, tk_alias, tk_except
    };

    interface TypeCode{
        exception Bounds{};
        exception BadKind{};

        boolean         equal( in TypeCode tc );

        TCKind          kind();

        RepositoryId    id()
            raises( BadKind );

        Identifier      name()
            raises( BadKind );

        unsigned long   member_count()
            raises( BadKind, Bounds );

        Identifier      member_name(
            in unsigned long index )
            raises( BadKind, Bounds );

        TypeCode        member_type(
            in unsigned long index )
            raises( BadKind, Bounds );

        any             member_label(
            in unsigned long index )
            raises( BadKind, Bounds );
```

```
        TypeCode        discriminator_type()
            raises( BadKind );

        long            default_index()
            raises( BadKind );

        unsigned long   length()
            raises( BadKind );

        TypeCode        content_type()
            raises( BadKind );

        long            param_count();

        any             parameter( in long index )
            raises( Bounds );
    };
};
```

TypeCodeインタフェースの機能説明を以下に示します。

TypeCodeのメソッド	機能説明
equal	指定されたTypeCodeが等しいかを比較します。
kind	TypeCodeの指すデータ型の数値を返します。
id	TypeCodeの指すデータ型のリポジトリIDを返します。
name	TypeCodeの指すデータ型の名前を返します。
member_count	TypeCodeの指すデータ型に含まれるメンバの数を返します。
member_name	指定したメンバの名前を返します。
member_type	指定したメンバのTypeCodeを返します。
member_label	指定したメンバのラベルを返します。
discriminator_type	弁別情報定義のTypeCodeを返します。
default_index	defaultメンバのインデックスを返します。
length	TypeCodeオブジェクトの長さ(要素数)を返します。
content_type	メンバのTypeCodeを返します。

各データ型で使用できるTypeCodeインタフェースを以下に示します。

	string	enum	sequence	struct	union	array	alias	ObjectRef	exception
id		○		○	○		○	○	○
name	○	○		○	○		○	○	○
member_count		○		○	○				○
member_name		○		○	○				○
member_type		○		○	○				○
member_label				×	○				
discriminator_type				×	○				
default_index					○				

	string	enum	sequence	struct	union	array	alias	ObjectRef	exception
length	○		○			○			
content_type			○			○			

12.2 NVListオブジェクト

12.2.1 NVListオブジェクトとは

NVListオブジェクトは、クライアント／サーバ間でパラメタを受け渡すためのインタフェースであり、以下のケースで使用されます。

- 動的起動インタフェース
- 動的スケルトンインタフェース

12.2.2 NVListインタフェース

NVListは、以下のインタフェースを提供しています。

```
module CORBA{
    typedef string                  Identifier;
    enum ORBStatus { OK, FAILED };
    enum Flags{
        ARG_IN,
        ARG_OUT,
        ARG_INOUT,
        OUT_LIST_MEMORY,
        IN_COPY_VALUE,
        INV_NO_RESPONSE,
        INV_TERM_ON_ERR,
        RESP_NO_WAIT,
        DEPENDENT_LIST,
        CTX_RESTRICT_SCOPE,
        CTX_DELETE_DESCENDENTS
    };
    interface NVList{
        ORBStatus  add_item(
                   in Identifier    item_name,
                   in TypeCode      item_type,
                   in Object        value, /* void */
                   in long          value_len,
                   in Flags         flags );

        ORBStatus  free();

        ORBStatus  free_memory();

        ORBStatus  get_count(
                   out long         count );
    };
};
```

NVListインタフェースの機能説明を以下に示します。

NVListのメソッド	機能説明
add_item	NVListにデータを追加します。
free	NVListおよびNVListの領域を解放します。

NVListのメソッド	機能説明
free_memory	NVListの領域を解放します。
get_count	NVListに格納されているデータの個数を返します。

12.3 Contextオブジェクト

12.3.1 Contextオブジェクトとは

通常のメソッドの引数としてクライアントアプリケーションからサーバアプリケーションに情報を渡すのではなく、あらかじめ、Contextと呼ばれるオブジェクトにサーバアプリケーションに渡す情報を設定することにより、メソッドの起動時に無条件に情報を伝播します。
Contextオブジェクトは、以下の3種類の情報を管理します。

Context名:

複数のプロパティをグループ分けするための名前

プロパティ名:

情報を識別するための名前

プロパティの値:

実際の情報

プリンタの例では、プリンタがContext名であり、プリンタの紙サイズ、プリンタの種類などをプロパティ名に対応させることができます。また、Contextオブジェクトは以下の2種類に大別できます。

種類	内容	プロパティ名
システムデフォルト	システムであらかじめ定義されているContext	ベンダー依存
ユーザ定義 (ChildContextと呼ぶ)	ユーザ定義のContext。ChildContextは、システムデフォルトContextの子供として管理されます。	生成時に指定します。

CORBAでは、システムデフォルトのプロパティ名の例として、_USER(ユーザ)、_GROUP(グループ)、_SYSTEM(システム)を上げていますが、ベンダー依存です。CORBAサービスでは、システムデフォルトのContextを定義していません。環境変数は、ChildContext、システムデフォルトの順に検索されます。

12.3.2 Contextインタフェース

(1)C言語

Contextインタフェースは、C言語では、以下のIDLを元にAPIが提供されています。

```
module CORBA {
       typedef string                  Identifier;
       enum ORBStatus { OK, FAILED };
       enum Flags{
                ARG_IN,
                ARG_OUT,
                ARG_INOUT,
                OUT_LIST_MEMORY,
                IN_COPY_VALUE,
                INV_NO_RESPONSE,
                INV_TERM_ON_ERR,
                RESP_NO_WAIT,
                DEPENDENT_LIST,
                CTX_RESTRICT_SCOPE,
                CTX_DELETE_DESCENDENTS
```

```
        };

        interface ORB {
                ...
                ORBStatus get_default_context( out Context );
                ...
        };
        interface Context {
                ORBStatus set_one_value(
                                in Identifier           prop_name,
                                in string               value );
                ORBStatus set_values(
                                in NVList               values );
                ORBStatus get_values(
                                in Identifier           start_scope,
                                in Flags                op_flags,
                                in Identifier           prop_name,
                                out NVList              values );
                ORBStatus delete_values(
                                in Identifier           prop_name );
                ORBStatus create_child(
                                in Identifier           ctx_name,
                                out Context             child_ctx );
                ORBStatus delete(
                                in Flags                del_flags );
        };
};
```

NVListについては、“12.2 NVListオブジェクト”を参照してください。
Contextインタフェースで提供しているAPI(C言語)の機能説明について、以下に示します。

インタフェース名	メソッド名	機能説明
ORB	get_default_context	デフォルトのコンテキストオブジェクトを取得します。
Context	set_one_value	プロパティ名とそのデータ値(ともに文字列)を指定されたコンテキストに関連付けます。
	set_values	NVListで指定された複数のプロパティ名とそのデータ値(ともに文字列)の組合せをコンテキストに関連付けます。
	get_values	コンテキストに関連付けられた指定されたパラメタに関連した情報のリストを取得します。
	delete_values	プロパティ名、およびそれに関連付けられているデータ値のリストを削除します。
	create_child	childコンテキストを作成します。
	delete	コンテキストオブジェクトを削除します。

(2)C++言語・Java言語

Contextインタフェースは、C++言語／Java言語では、以下のIDLを元にAPIが提供されています。

```
module CORBA {
        typedef string                  Identifier;
        enum ORBStatus { OK, FAILED };
        enum Flags{
                ARG_IN,
                ARG_OUT,
                ARG_INOUT,
                OUT_LIST_MEMORY,
```

```
                IN_COPY_VALUE,
                INV_NO_RESPONSE,
                INV_TERM_ON_ERR,
                RESP_NO_WAIT,
                DEPENDENT_LIST,
                CTX_RESTRICT_SCOPE,
                CTX_DELETE_DESCENDENTS
        };

        interface ORB {
                ...
                Status get_default_context( out Context );
                ...
        };
        interface Context {
                readonly attribute Identifier     context_name;
                readonly attribute context        parent;

        ORBStatus set_one_value(
                in Identifier prop_name,
                in any propvalue );
        ORBStatus set_values( in NVList values );
        ORBStatus get_values(
                in Identifier start_scope,
                in Flags op_flags,
                in Identifier pattern,
                out NVList values );
        ORBStatus delete_values(
                in Identifier start_scope,
                in Flags op_name,
                in Identifier pattern,
                out NVList values );
        ORBStatus create_child(
                in Identifier child_ctx_name,
                out Context child_ctx );
        };
};
```

Contextインタフェースで提供しているAPI(C++言語)／Contextクラスで提供しているAPI(Java言語)の機能について、以下に示します。

インタフェース名／クラス名	メソッド名	機能説明
ORB	get_default_context	デフォルトのコンテキストオブジェクトを取得します。
Context	context_name	コンテキスト名を取得します。
	parent	親のコンテキストオブジェクトを取得します。
	set_one_value	プロパティ名とそのデータ値(ともに文字列)を指定されたコンテキストに関連付けます。
	set_values	NVListで指定された複数のプロパティ名とそのデータ値(ともに文字列)の組合せをコンテキストに関連付けます。
	get_values	コンテキストに関連付けられた指定されたパラメタに関連した情報のリストを取得します。
	delete_values	プロパティ名、およびそれに関連付けられているデータ値のリストを削除します。
	create_child	childコンテキストを作成します。

(3)COBOL

Contextインタフェースで提供しているAPI(COBOL)の機能について、以下に示します。

インタフェース名	メソッド名	機能説明
ORB	GET-DEFAULT-CONTEXT	デフォルトのコンテキストオブジェクトを取得します。
Context	SET-ONE-VALUE	プロパティ名とそのデータ値(ともに文字列)を指定されたコンテキストに関連付けます。
	SET-VALUES	NVListで指定された複数のプロパティ名とそのデータ値(ともに文字列)の組合せをコンテキストに関連付けます。
	GET-VALUES	コンテキストに関連付けられた指定されたパラメタに関連した情報のリストを取得します。
	DELETE-VALUES	プロパティ名、およびそれに関連付けられているデータ値のリストを削除します。
	CREATE-CHILD	childコンテキストを作成します。
	DELETE	コンテキストオブジェクトを削除します。

(4)OOCOBOL

Contextインタフェースで提供しているAPI(OOCOBOL)の機能について、以下に示します。

インタフェース名	メソッド名	機能説明
ORB	GET_DEFAULT_CONTEXT	デフォルトのコンテキストオブジェクトを取得します。
Context	CONTEXT_NAME	コンテキスト名を取得します。
	PARENT	親のコンテキストオブジェクトを取得します。
	SET_ONE_VALUE	プロパティ名とそのデータ値(ともに文字列)を指定されたコンテキストに関連付けます。
	SET_VALUES	NVListで指定された複数のプロパティ名とそのデータ値(ともに文字列)の組合せをコンテキストに関連付けます。
	GET_VALUES	コンテキストに関連付けられた指定されたパラメタに関連した情報のリストを取得します。
	DELETE_VALUES	プロパティ名、およびそれに関連付けられているデータ値のリストを削除します。
	CREATE_CHILD	childコンテキストを作成します。

注意

Windows64 Solaris64 Linux64

OOCOBOLは、Windows(R)クライアントアプリケーションで使用できます。

12.3.3 Contextインタフェースの使用例

Contextインタフェースを使用したデータ値の設定方法／取得方法についてのプログラム例を以下に示します。IDLは、以下のIDLを使用します。

```
module ODsample {
      interface contexttest {
```

```
            long    op( in long x )
                    context( "DATA", "DATA1", "DATA2" );
    };
};
```

注意

Portable-ORBを使用した運用を行う場合、Contextインタフェースは使用できません。

(1)C言語でのプログラミング例

クライアントプログラム

CORBA_ORB_get_default_context()で取得したContextオブジェクトに対して、CORBA_Context_set_one_value()を発行してプロパティ名とそのデータ値をContextオブジェクトに関連付けます。

```
CORBA_ORB               orb;
CORBA_Environment       env;
CORBA_Object            obj;
CORBA_Context_ptr       context;

    /*デフォルト・コンテキストの取得 */
    CORBA_ORB_get_default_context( orb, &context, &env );

    /* データ値の設定 */
    CORBA_ORB_set_one_value( context, "DATA", "data", env );
    CORBA_ORB_set_one_value( context, "DATA1", "data1", env );
    CORBA_ORB_set_one_value( context, "DATA2", "data2", env );

    /* Contextの送信 */
    ODsample_contexttest_op( obj, 3, context, &env );
```

サーバプログラム

サーバでは、CORBA_Context_get_values()を使用して、プロパティ名とそれに関連したデータ値を取得できます。このとき、検索するプロパティ名にワイルドカード(*)を使用できます。

```
void ODsample_contexttest_op(
    ODsample_contexttest    obj,
    CORBA_long              x,
    CORBA_Context           context,
    CORBA_Environment       *env )
{
    CORBA_NVList      nvlist;
    CORBA_long        len;

    /* コンテキストに関連付けられた情報の取得 */
    CORBA_Context_get_values( context, "", 0, "DATA*", &nvlist , env );

    /* nvlistの解析 */
    CORBA_NVList_get_count( nvlist, &len, &env );
    for ( i = 0; i < len; i++ ){
            printf("NVList:[%s]¥n", *(CORBA_string *)nvlist[i].argument._value );
            /* DATA, DATA1, DATA2の各データ値が表示されます。*/
    }
    ...
}
```

(2)C++言語でのプログラミング例

クライアントプログラム

CORBA::ORB::get_default_context()で取得したContextオブジェクトに対して、CORBA::Context::set_one_value()を発行してプロパティ名とそのデータ値をContextオブジェクトに関連付けます。

```
CORBA::ORB_ptr           orb;
CORBA::Environment       env;
CORBA::Object_ptr        obj;
CORBA::Context_ptr       context;
try {
     /*デフォルト・コンテキストの取得 */
     orb->get_default_context( context, env );

     /* データ値の設定 */
     CORBA::Any        p1, p2, p3;
     p1 <<= (const CORBA::Char *)"data";
     context->set_one_value( "DATA", p1, env );
     p2 <<= (const CORBA::Char *)"data1";
     context->set_one_value("DATA1", p2, env );
     p3 <<= (const CORBA::Char *)"data2";
     context_set_one_value( "DATA2", p3, env );

     /* Contextの送信 */
     obj->op( 3, context, env );
}
catch( CORBA::Exception &e ){
     // エラー処理
}
```

サーバプログラム

サーバでは、CORBA::Context::get_values()を使用して、プロパティ名とそれに関連したデータ値を取得できます。このとき、検索するプロパティ名にワイルドカード(*)を使用できます。

```
void ODsample_contexttest_impl::op(
     CORBA::Long              x,
     CORBA::Context_ptr       context,
     CORBA::Environment       &env )
     throw( CORBA::Exception )
{
     CORBA::NVList_ptr        nvlist;
     CORBA::NamedValue_ptr    nv;

     /* コンテキストに関連付けられた情報の取得 */
     context->get_values( NULL, 0, "DATA*", nvlist , env );

     /* nvlistの解析 */
     CORBA::Long        len = nvlist->count( env );
     for( I = 0; I < len; I++ ){
           nv = nvlist->item( I, env );
           CORBA::Any *data = nv->value( env );
           CORBA::Char *str = (CORBA::Char *)data->value();
     }
     ...
}
```

(3)Java言語でのプログラミング例

クライアントプログラム

org.omg.CORBA.ORB.get_default_context()で取得したContextオブジェクトに対して、org.omg.CORBA.Context.set_one_value()を発行してプロパティ名とそのデータ値をContextオブジェクトに関連付けます。

```
org.omg.CORBA.ORB   Orb;
ODsample.contexttestOperations   obj;

try {
                        :
     /*デフォルト・コンテキストの取得 */
     org.omg.CORBA.Context ctx = Orb.get_default_context();

     /* データ値の設定 */
     org.omg.CORBA.Any p1,p2,p3;
     p1 = Orb.create_any();
     p1.insert_string("data");
     ctx.set_one_value("DATA",p1);

     p2 = Orb.create_any();
     p2.insert_string("data1");
     ctx.set_one_value("DATA1",p2);

     p3 = Orb.create_any();
     p3.insert_string("data2");
     ctx.set_one_value("DATA2",p3);

     /* Contextの送信 */
     obj.op( 3, ctx );
}
catch ( Exception e ) {
     System.out.println( "ERROR" );  // エラー処理
}
```

サーバプログラム

サーバでは、org.omg.CORBA.Context.get_values()を使用して、プロパティ名とそれに関連したデータ値を取得できます。このとき、検索するプロパティ名にワイルドカード(*)を使用できます。

```
/* Servant内のオペレーション実装部分 */
public void op( int x, org.omg.CORBA.Context ctx )
{
     /* コンテキストに関連付けられた情報の取得 */
     NVList nvlist = ctx.get_values("",0,"DATA*");

     /* nvlistの解析 */
     int count = nvlist.count();
     org.omg.CORBA.NamedValue nv;
     String str;
     for(int i=0;i<count;i++){
          nv = nvlist.item(i);
          str = nv.value().extract_string();
          System.out.println( "result = " + str );
          /* DATA, DATA1, DATA2の各データ値が表示されます。*/
     }
     return;
}
```

第13章　ネーミングサービスのイニシャルリファレンスの獲得

本章では、ネーミングサービスのイニシャルリファレンスの獲得について説明します。

CORBA_ORB_resolve_initial_references()によりイニシャルサービスのオブジェクトリファレンスを獲得する際、クライアントアプリケーションごとに必要な異なるオブジェクトリファレンスを獲得するための手段を提供します。イニシャルリファレンスのオーバライド設定のために、初期設定によるコンフィギュレーションに加えて、2つのCORBA_ORB_init引数を提供します。

13.1 ORBInitRef

ORBイニシャルリファレンス引数“-ORBInitRef”は、クライアントが必要とするイニシャルサービスをサービス名ごとに指定します。
フォーマットを以下に示します。

```
-ORBInitRef <ObjectID>=<ObjectURL>
```

例

```
-ORBInitRef  NameService=IOR:00230021AB...
-ORBInitRef  NameService=corbaloc::nshost/NameService
-ORBInitRef  NameService=corbaname::nshost/NameService#nshost2
```

<ObjectID>には、イニシャルサービスのサービス名を指定します。

<ObjectURL>には、CORBA_ORB_string_to_objectによってサポートされるURLスキーマのどれでも指定可能です。URLが正しくない場合は、CORBA_ORB_initオペレーションでBAD_PARAM例外が発生します。

参照

Javaアプレットを使用する場合の指定方法については、“リファレンスマニュアル（API編）”の“org.omg.CORBA.ORB.init()”を参照してください。

13.2 ORBDefaultInitRef

ORBデフォルトイニシャルリファレンス引数“-ORBDefaultInitRef”は、“-ORBInitRef”で明示的に指定されないイニシャルリファレンスの解決を援助します。
“-ORBDefaultInitRef”は、CORBA_ORB_resolve_initial_referencesオペレーション実行時、スラッシュ(/)を追加した後、サービス名として指定されたオブジェクトキーを付加して目的とするオブジェクトリファレンスを獲得します。

例

```
-ORBDefaultInitRef corbaloc::inithost
```

上記のCORBA_ORB_initを実施した後、CORBA_ORB_resolve_initial_references("NotificationService")を実行すると、“corbaloc::inithost/NotificationService”のオブジェクトリファレンスが獲得されます。

CORBA::ORB_initオペレーションで“-ORBDefaultInitRef corbaname::inithost,:inithost2”が指定された後に、CORBA_ORB_resolve_initial_references("NameService")が呼び出された場合、“corbaname::inithost/NameService”／“corbaname::inithost2/NameService”のオブジェクトリファレンスが獲得されます。

参照

Javaアプレットを使用する場合の指定方法については、“リファレンスマニュアル(API編)”の“org.omg.CORBA.ORB.init()”を参照してください。

13.3 イニシャルサービスの検索順序

与えられた<ObjectID>をCORBA_ORB_resolve_initial_referencesの呼出し処理のためのデフォルト順は、以下のとおりです。

1. <ObjectID>のための-ORBInitRefで解決します。
2. -ORBDefaultInitRefエントリーで解決します。ただし、<ObjectID>が以下である場合、-ORBDefaultInitRefエントリーは無視されます。

```
ImplementationRepository
FJ_LightInterfaceRepository
FJ_ORB_admin
RootPOA
POACurrent
TransactionCurrent
SecurityCurrent
TransactionIdentification
TransactionServerInit
```

3. initial_servicesファイルの設定で解決します。
4. 環境設定ファイルinitial_hostsの設定で解決します。

13.4 corbaloc URLスキーマ

IOR形式のURLをそのまま認識することは、長さやテキストが符号化されているために困難です。corbaloc URLスキーマは、文字列表記のオブジェクトリファレンスをftpやhttp URLのように、わかりやすい表記で提供します。

13.4.1 corbaloc URLスキーマ

corbaloc URLスキーマは、文字列表記のオブジェクトリファレンスを提供します。

```
corbaloc::inithost/NotificationService
corbaloc:iiop:1.0@inithost/NotificationService
corbaloc::inithost,:inithost2:8002/NameService
corbaloc:rir:/NotificationService
corbaloc:rir:/NameService
```

完全なシンタックスを以下に示します。

```
<corbaloc>= "corbaloc:"<obj_addr_list>["/"<key_string>]
<obj_addr_list>= [<obj_addr> ","]* <obj_addr>
<obj_addr>= <prot_addr>
<prot_addr>= <rir_prot_addr> | <iiop_prot_addr>

<rir_prot_addr>= <rir_prot_token>":"
<rir_prot_token>= "rir"

<iiop_prot_addr>= <iiop_id><iiop_addr>
<iiop_id>= ":" | <iiop_prot_token>":"
<iiop_prot_token>= "iiop"
<iiop_addr>= 本節を参照。

<key_string>= <string> | empty_string
```

詳細

obj_addr_list

アドレス情報がコンマ(,)で区切られたリストです。オブジェクトリファレンスの獲得時に複数のアドレスからオブジェクトを獲得できます。

obj_addr

プロトコル識別子／バージョンタグ／プロトコル特定のアドレスです。コンマ(,)およびスラッシュ(/)は、指定できません。

rir_prot_addr

CORBA_ORB_resolve_initial_referencesプロトコル識別子です。

iiop_prot_addr

iiopプロトコル識別子、バージョンタグ、およびDNS形式ホスト名／IPアドレスを含むアドレスです。
「corbaloc::inithost/NotificationService」のようにiiopプロトコル識別子が省略されている場合は、iiopプロトコル識別子に"iiop"を指定した「corbaloc:iiop:inithost/NotificationService」と同じ意味になります。

key_string

文字列化されたオブジェクトキーです。

US-ASCII英数字、および以下に示された文字を除く文字がエスケープされます。

```
";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" |
"," | "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
```

13.4.2 corbaloc:rir URL

corbaloc:rir URLは、URLを介してORBのイニシャルリファレンスの設定を獲得するために使用されます。
corbaloc:rir URLは、初期設定（ORBInitRef引数およびORBDefaultInitRef引数によるオーバライドが未設定）のイニシャルリファレンスを獲得するために使用されます。ORBInitRef引数、およびORBDefaultInitRef引数の詳細については、“13.1 ORBInitRef”、“13.2 ORBDefaultInitRef”を参照してください。

プロトコルアドレスシンタックスを以下に示します。

```
<rir_prot_addr>= "rir:"
```

詳細

rir_prot_addr

CORBA_ORB_resolve_initial_referencesプロトコル識別子です。“rir:”を指定します。

13.4.3 corbaloc:iiop URL

corbaloc:iiop URLは、TCP/IPとDNS中心環境で使用するために定義されます。

完全なプロトコルアドレスシンタックスを以下に示します。

```
<iiop_prot_addr>= <iiop_id><iiop_addr>
<iiop_id>= <iiop_default> | <iiop_prot_token>":"
<iiop_default>= ":"
<iiop_prot_token>= "iiop"
<iiop_addr>= <version> <host> [":" <port>]
<host>= DNS-style Host Name | ip_address
<version>= <major> "." <minor> "@" | empty_string
<port>= number
<major>= number
<minor>= number
```

詳細

iiop_prot_addr

iiopプロトコル識別子、バージョンタグ、DNS形式ホスト名／IPアドレスを含むアドレスです。

iiop_id

iiopプロトコルcorbalocを表すために認識されるトークンです。

iiop_default

":" iiopプロトコルを示すデフォルトトークンです。

iiop_prot_token

"iiop" iiopプロトコルトークンです。

iiop_addr

単一アドレスです。

host

DNS形式ホスト名／IPアドレスです。

version

メジャーとマイナーバージョン番号が“.”により分割され、さらに“@”が続きます。省略値は、“1.0”です。現在サポートしているバージョンは、“1.0”だけです。

ip_address

IPアドレスです。IPv6アドレスは、指定できません。

port

エージェントがリッスンしているポート番号です。

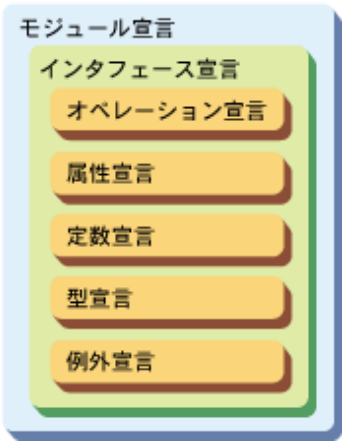
付録A　IDL定義

IDL(Interface Definition Language:インタフェース定義言語)は、オブジェクトのインタフェースを定義する言語です。
IDLにより、サーバアプリケーションが提供するインタフェースを定義できます。インタフェースを定義したIDLファイルをIDLコンパイラでコンパイルすることにより、指定した言語にマッピングされたクライアントアプリケーション／サーバアプリケーション用のソースコードが生成されます。このソースコードを、それぞれスタブ、スケルトンといいます。クライアントアプリケーションからは、スタブ/スケルトンを通してIDLで定義したサーバアプリケーションを呼び出すことができます。

A.1　IDLの書式

オブジェクトのインタフェースや、そのインタフェースで使用される型を宣言する場合は、以下の書式で記述します。

IDL定義＝



各宣言の後ろには、セミコロン(;)を付けます。
IDLで記述したインタフェース例を以下に示します。

```
module Module1 {                          // モジュール宣言
      interface Func1 {                   // インタフェース宣言
          typedef long otype;             // 型宣言
          exception  FuncException {      // 例外宣言
              string  reason;
          };
          otype  Open(in string name)     // オペレーション宣言
                  raises(FuncException);
          readonly attribute long  data;  // 属性宣言
          typedef Object  FuncObject;     // 型宣言
      };
  };
```

A.1.1　コメント

IDL中でのコメントの表記方法は、以下の2種類です。

- /*と*/の間に、コメントを書きます。
- //に続けて、コメントを書きます。この場合、行の終わりまでがコメントとなります。

共にコメントの中でのコメント文字(/*,*/,//)は、特別な意味を持たず、他の文字と同じように扱われます。コメントとして、アルファベット、数字、漢字、かな文字、記号文字、垂直タブ、水平タブ、改ページ、改行、および空白を使用できます。

A.1.2 識別子

識別子は、IDLで記述する各宣言を識別するために使用します。識別子は、先頭が英文字で、英文字、数字、およびアンダースコア(_)で指定します。なお、英文字の大文字と小文字は同じものとして扱われるため、大文字と小文字だけが異なる識別子を使用すると、同名とみなされ、コンパイルエラーとなります。
また、すべてのIDLファイル内に設定した識別子は、一意である必要があります。たとえば、異なるIDLファイルでも定数とインタフェースに同じ識別名を使用すると、コンパイルエラーとなります。

なお、以下に示す識別子はキーワードとして予約されているため、他の目的には使用できません。

any	context	fixed	longdouble	out	struct	union
attribute	default	float	longlong	raises	switch	void
boolean	double	in	module	readonly	TRUE	wchar
case	enum	inout	Object	sequence	TypeCode	wstring
char	exception	interface	octet	short	typedef	
const	FALSE	long	oneway	string	unsigned	

キーワードは識別子であるため、識別子と同じ規則に従い、正確に書く必要があります。たとえば、“interface”は正確ですが、“Interface”は“I”が大文字であるため、コンパイルエラーとなります。

また、特殊文字として以下の文字を使用します。

```
;  {  }  :  ,  =  +  -  (  )  <  >  [  ]  '  "  ¥  |  ^  &  *  /  %  ~
```

注意

Java言語のCORBAアプリケーションを開発する場合は、識別子にDOSデバイス名を指定できません。

A.1.3 定数

IDLでは、以下の定数が使用できます。

- 整数定数
- 文字定数
- 浮動小数点定数
- 文字列定数

(1) 整数定数

整数定数では、以下の3種類の表現が使用できます。

10進数表現	0以外の数字、および＋／－で始まる数字列（128、－45など）
8進数表現	0で始まる数字列（0777、04など）
16進数表現	0x、または0Xで始まる英数字列（0x00、0Xeff9など）

(2) 文字定数

文字定数は、文字をシングルクォート(')で括ります。なお、以下の文字を指定する場合は、エンマーク(¥)と組み合わせてエスケープ文字として使用します。

```
復帰                ¥r
改行                ¥n
改ページ            ¥f
```

水平タブ	¥t
垂直タブ	¥v
シングルクォート	¥'
ダブルクォート	¥"
アラート	¥a
エスケープ文字	¥¥
バックスペース	¥b
クエスチョンマーク	¥?
8進数	¥ooo
16進数	¥xhh

- エンマーク(¥)の後ろに上記以外の文字が指定された場合は、エンマーク(¥)と指定された文字で扱われます。
- ¥oooは、8進数を指定する方法です。oooは、1～3桁で指定します。
- ¥xhhは、16進数を指定する方法です。hhは、2桁以内で指定します。

(3) 浮動小数点定数

浮動小数点定数は、0から9の数字、および'+'／'-'／'.'で指定します(1.2, -0.46など)。また、整数部と指数部に分けて指定することもできます。小数点の代わりに、e／Eを指定することもできます。指数部の符号を付けることもできます。
浮動小数点定数を以下に示します。

```
1e5 = 1 × 10^5 = 100000
1.23E3 = 1.23 × 10^3 = 1230
-3E-2 = -3 × 10^-2 = -0.03
```

(4) 文字列定数

文字列定数は、文字列をダブルクォーテーション(")で括ります。文字列の中にダブルクォーテーション(")を含む場合は、その前にエンマーク(¥)を付けて"¥""と指定する必要があります。また、先頭が英文字で、英文字、数字、ピリオド(.)、アンダースコア(_)からなる文字列リテラルは、文字'¥0'を含んではいけません。

A.1.4 区切り文字

隣り合った識別子、定数などを区別するために、以下の文字が使用可能です。

- 空白
- 水平タブ
- 垂直タブ
- 改行
- 改ページ
- コメント

A.1.5 名前とスコープ

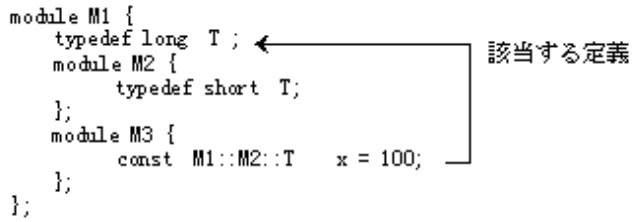
スコープとは、IDL内での名前の一意性を定義するものです。

スコープを指定する方法には、以下の3つの方法があります。

- 先頭が識別子で始まるスコープ名(識別子::識別子など)
- 先頭が“::”で始まるスコープ名
- 識別子だけのスコープ名

(1)先頭が識別子で始まるスコープ名(識別子::識別子など)

先頭が識別子で始まり、各識別子を2つのコロン(::)で接続させたスコープ名をフルスコープと呼びます。フルスコープは、IDLファイルの最上位モジュールからの識別子の並びを指定します。
以下の例では、モジュールM3内の定数xの型は、スコープ名“M1::M2::T”で定義した型(short型)となります。



インタフェースを継承している場合は、そのインタフェースのスコープ名を指定することにより、そのインタフェースが継承しているベースインタフェースの定義を利用できます。
以下の例では、インタフェースI3内の定数xの型は、インタフェースI2のベースインタフェースI1にある定義T、すなわちshort型となります。

```
module M1 {
    interface  I1 {
        typedef short  T ;
    };
    interface  I2:I1 {
    };
    interface  I3 {
        const  M1::I2::T  x = 100 ;
    };
};
```

(2)先頭が“::”で始まるスコープ名

2つのコロン(::)の後ろには、このスコープ名を指定した定義を含むモジュールから下の階層の識別子の並びを指定します。
以下の例では、スコープ名“::T”は、このスコープ名を定義した定数を含むモジュールM1内のTが有効となるため、インタフェースI2内の定数xの型は、long型となります。

```
module  M1 {
    typedef long  T ;
    interface  I1 {
        typedef short  T ;
    };
    interface  I2 {
        const ::T  x = 100 ;
    };
};
```

(3)識別子のみのスコープ名

スコープの中では、識別子だけで他の定義を使うことができます。

```
module  M {
    typedef long  L ;
    const  L x = 100 ;
};
```

名前が同一スコープ内に存在しない場合、自らを含む上位スコープを順に探索するため、上位のスコープの名前は以下のようにスコープ名を指定しないで名前だけで参照できます。
以下の例では、モジュールM2内で使用しているLは、モジュールM2のスコープ内に存在しませんが、その上位スコープのモジュールM1にLの定義があるため、この定義(long型)が有効となります。

```
module  M1 {
    typedef long  L ;
    module  M2 {
         const  L x = 100 ;
    };
};
```

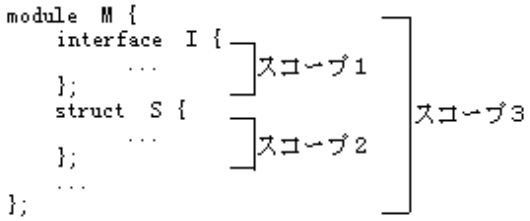
あるスコープで、スコープを指定されていない名前が一度使われると、そのスコープでその名前を再定義できません。このような再定義は、コンパイルエラーを引き起こします。
以下を定義すると、スコープのネスト化が行われます。

- モジュール
- インタフェース
- 構造体
- 共用体
- オペレーション
- 例外

また、以下を定義すると、新たなスコープが形成されます。

- 型
- 定数
- 列挙値
- 例外
- インタフェース
- 属性
- オペレーション

スコープの形成例を以下に示します。この例では、3つのスコープが形成されます。



スコープ内では、各定義の識別名を重複して定義できません。なお、ネストしたスコープの中では、同じ名前の識別子を定義できます。各定義の識別子は大文字と小文字との区別をしていないため、大文字／小文字が異なるだけの識別子は、同一とみなされます。

A.1.6 C++との違い

IDLの文法は、C++をもとにしています。ただし、C++にはない以下の制限があります。

- 関数の復帰値の型を必ず指定する必要があります。
- オペレーション宣言の各パラメタには、名前を定義する必要があります。
- 空のパラメタリストの代わりに、voidを使用できません。
- 構造型（構造体、弁別情報付き共用体、列挙型）には、タグが必要です。
- 整数は、明示的にshort／longを宣言する必要があります。intやunsignedなどを定義できません。
- charに、signed／unsignedのキーワードを付けることはできません。

A.1.7 プリプロセス

IDLのプリプロセスは、ANSI C++のプリプロセスに基づいて、以下の機能を持ちます。

- マクロの置換（#define、#undef）
- 条件付きコンパイル（#if、#ifdef、#ifndef）
- ソースファイルの引用（#include）
- 診断時やシンボリックデバッグのための行番号コントロール（#line）
- 与えられた文字列を持つ診断用メッセージの生成（#err）
- リポジトリIDの置換（#pragma）

#で始まる行は、プリプロセスへの情報を指定します。#の前にスペースがあってもかまいません。この行はシンタックス上その他のIDLと独立しており、IDL上のどこに指定してもかまいません。ソースファイルの中で、改行の直前にあるバックスラッシュ（¥）は次の行に続くことを表します。バックスラッシュをソースファイルの最後の文字とすることはできません。

(1)プラグマ(#pragma)

プラグマは、RepositoryIDを変更する機能です。プラグマには、以下の3パターンが指定できます。

IDプラグマ

形式

#pragma ID <name> "<id>"

機能

<name>で指定されたオブジェクトのRepositoryIDを、<id>と置き換えます。
<name>は、対象とするオブジェクトの識別子であり、スコープ名で指定できます。

PREFIXプラグマ

形式

#pragma prefix "<string>"

機能

宣言されたPREFIXプラグマは、現在のスコープの端に達するか、または別のPREFIXプラグマに遭遇するまで、生成されるRepositoryIDのPREFIXとして<string>を設定します。
<string>は、任意の文字列です。

VERSIONプラグマ

形式

#pragma version <name> <major>.<minor>

機能

<name>で指定されたRepositoryIDのバージョンを、<major>.<minor>のVERSIONに置き換えます。
<name>は、対象とするオブジェクトの識別子であり、スコープ名で指定します。<major>と<minor>は、正数を指定します。

例

プラグマを使用したRepositoryIDの変更例を以下に示します。

```
module  M1{
    typedef long T1;
    typedef long T2;
    #pragma ID T2 "IDL:M1/T2_long:1.0"
};

#pragma prefix "P1"

module  M2{
    module  M3{
        #pragma prefix "P2"
        typedef long T3;
    };
    typedef long T4;
    #pragma version T4 2.4
};
```

スコープされた名前、およびRepositoryIDを以下に示します。

```
::M1::T1            IDL:M1/T1:1.0
::M1::T2            IDL:M1/T2_long:1.0
::M2::M3::T3        IDL:P2/T3:1.0
::M2::T4            IDL:P1/M2/T4:2.4
```

A.2 モジュール宣言

モジュールは、IDLのメソッド名や型名などが他のIDLと重複しないように、オブジェクトのグループ化宣言を行うものです。モジュール宣言では、モジュールとモジュールを束ねて、さらに大きなモジュールを宣言できます。
以下に、モジュール宣言の書式を示します。

```
module モジュール名 {
    IDLの繰り返し
};
```

モジュール宣言の例を以下に示します。

```
module  Module1 {                                   // モジュール宣言
    module  Module2 {                               // モジュール宣言
        typedef Object  otype;                      // 型宣言
        interface  Func1 {                          // インタフェース宣言
            otype  Open(in string name);            // オペレーション宣言
        };
    };
    interface  Func1 {                              // インタフェース宣言
        exception FuncException {
            string  reason;
        };
        Module1::Module2::otype  Open(in string name) // オペレーション宣言
                raises(FuncException);
```

```
    };
};
```

上記の例のように、モジュールModule1のモジュールModule2内、およびモジュールModule1内の両方で、同名のインタフェースFunc1が宣言された場合、Func1は以下のように別々のインタフェースとして扱うことができます。

```
<1>  Module1::Module2::Func1
<2>  Module1::Func1
```

Java言語のCORBAアプリケーションを開発する場合は、モジュール宣言を行う必要があります。モジュール宣言がない場合は、Javaコンパイラでエラーが発生します。

A.3 インタフェース宣言

インタフェース宣言では、インタフェースの実体を定義します。インタフェース宣言は、C++言語のクラス宣言と同様に他のインタフェース宣言を継承して、新しいインタフェース宣言を行うことができます。他のインタフェース宣言を継承すると、その継承したインタフェース宣言のオペレーション宣言や型宣言などを、あたかも自分で宣言したと同じに扱えるようになります。

インタフェースの継承では、新しいインタフェースを宣言する際に、他のインタフェース宣言を使用して、差分だけを定義できます。また、複数のインタフェースを継承することもできます。これを多重継承と呼びます。すべてのインタフェース定義は、明示的に記述しなくてもObjectインタフェースを継承します。このため、オブジェクト自身にインタフェース情報を問い合わせるメソッドget_interfaceDefなどのObjectインタフェースを使用できます。

以下に、インタフェース宣言の書式を示します。

```
interface インタフェース名 ;   // フォワード宣言

interface インタフェース名 [:継承するインタフェース] {  // ヘッダ
    オペレーション宣言 ;       // 本体
    属性宣言 ;
    定数宣言 ;
    型宣言 ;
    例外宣言 ;
};
```

[ ]内は省略可能です。

最初の書式(インタフェース名だけ)は、フォワード宣言です。インタフェースの実体を定義する前に、インタフェース名だけを定義するものです。

フォワード宣言により、相互参照するインタフェースを定義することが可能となります。フォワード宣言されたインタフェースは、それより後方でその実体を定義する必要があります。複数箇所で同じインタフェース名をフォワード宣言することもできます。

2行目以降の書式は、インタフェースの実体を宣言するものです。“継承するインタフェース”を指定すると、他のインタフェースを継承できます。継承するインタフェースの指定方法については、“A.3.1 インタフェースの継承”を参照してください。インタフェースは、1つまたは複数のオペレーション宣言、属性宣言、定数宣言、型宣言から構成され、空のインタフェース(何も宣言しない)とすることもできます。各宣言の詳細については、以降に説明します。
以下にインタフェース宣言の例を示します。この例では、インタフェースFunc2がインタフェースFunc1を継承する際の定義を示します。この場合、クライアントは、インタフェースFunc2のオブジェクトにOpenメソッドの呼出しができます。

```
module Module1 {
    typedef Object  otype;
    interface  Func1 {                          // インタフェース宣言
        exception  FuncException {              // 例外宣言
            string  reason;
        };
        Module1::otype  Open(in string name)    // オペレーション宣言
```

```
                raises(FuncException);
        readonly attribute long  data;              // 属性宣言
        typedef Object  FuncObject;                 // 型宣言
    };

    interface  Func2 : Func1 {                      // インタフェース宣言
        Module1::otype  Reopen(in string name);     // オペレーション宣言
        oneway void Close(in Module1::otype obj);   // オペレーション宣言
    };
};
```

A.3.1 インタフェースの継承

継承とは、あるインタフェースで定義したオペレーションを、別のインタフェースに引き継ぐための定義です。オペレーションなどを引き継ぐ元のインタフェースのことをベースインタフェースと呼びます。継承するインタフェースは、ベースインタフェースのスコープ名をコロン(:)に続いて指定します。

継承の指定例を以下に示します。この例では、インタフェース“A”がベースインタフェースです。ベースインタフェースを複数指定する場合は、カンマ(,)で区切って定義してください。

```
interface A {
    long op1(in long a);
};
interface B:A {
    long op2(in long b);
};
```

ベースインタフェースを継承したインタフェースでは、新しい要素(定数、型、例外、属性、オペレーション)を追加して宣言できます。継承元のインタフェースで定義した同じ名前の型、定数、および例外が再定義できます。2つ以上のインタフェースを継承する場合、継承元の同じ識別子があると継承できません。

```
interface A {
    const long a = 10;
};
interface B:A {
    const long a = 100;
    long op(in long A::a);
};
```

“継承するインタフェース”で指定されたインタフェースを直接のベースと呼びます。また、ベースインタフェースが、さらに別のインタフェースを継承する場合、このインタフェースを間接のベースと呼びます。
以下の例では、インタフェース“C”から見た場合に、インタフェースBが直接のベース、インタフェースAが間接のベースとなります。

```
interface  A   { ... };
interface  B:A { ... };
interface  C:B { ... };
```

インタフェースは、複数の直接のベースから継承できます。この継承の方法を多重継承と呼びます。直接のベースを指定する順番を変えても、継承されるインタフェースは同じものになります。
多重継承の指定例を以下に示します。

```
interface  A   { ... };
interface  B:A { ... };
interface  C:A { ... };
```

インタフェースの継承で、1つのインタフェースが直接のベースとして2回指定されることはありません。ただし、間接のベースとして複数回指定されることがあります。
間接のベースとして複数回指定される例を以下に示します。

```
interface  A     { ... };
interface  B:A   { ... };
interface  C:A   { ... };
interface  D:B,C { ... };
```

これらのインタフェースの関係を以下に示します。

```
B → A
↑   ↑
D → C
```

ベースインタフェースの要素への参照に、あいまいさがあってはなりません。ベースインタフェースの要素に同名の定数、型、例外が存在すると、あいまいさが生じる原因となります。
あいまいさが生じる例を以下に示します。この例では、インタフェースCのXの型にあいまいさが生じます。

```
interface A {
    typedef long X;
};
interface B {
    typedef short X;
};
interface C:A,B {
    const X = 100;
};
```

多重継承の場合、ベースインタフェースで定数、型、例外が再定義されても、継承されるオペレーションや属性のシンタックスは変わりません。
以下に例を示します。

```
module Module1 {
    const long L = 3;
    interface A {
        typedef float s[L];
        void f(in s para);    // Sは3つのfloatからなります
    };
    interface B {
        const long L = 4;
    };
    interface C:B,A { };      // f()のシグネチャは？
};
```

インタフェースAを定義した時点で、定数Lの実体との結びつきが決まるため、インタフェースCにおけるオペレーションfの形式は、以下のようにインタフェースAと同じになります。

```
typedef float  s[3];
void  f(in s  para);
```

この規則のために、導出されたインタフェースの中で定数／型／例外を再定義しても、ベースインタフェースから継承したオペレーション／属性へは影響ありません。

A.4 オペレーション宣言

オペレーション宣言は、メソッド名、復帰値の型、パラメタのデータ型、メッセージび出し時に返される例外、コンテキスト情報（サーバアプリケーションとクライアントアプリケーションの間で共通に持つ環境変数に相当）を含みます。

オペレーション宣言の書式を以下に示します（[ ]内は省略可能）。

```
[ oneway ] 復帰値のデータ型 メソッド名 (                  // メソッド名定義
    [ パラメタタイプ データ型 パラメタ名[, ..] ]          // パラメタ
)
    [ raises  ( 例外構造体名[, ..] ) ]
    [ context ( コンテキスト名[, ..] ) ];
```

各キーワードと指定項目の意味を以下に説明します。

oneway

オペレーション属性を指定します。
onewayを指定すると、サーバは呼び出されたメソッドの実行結果をクライアントに通知しません。このため、以下の制約があります。

- 出力パラメタを持つことができないため、復帰値の型はvoidとします。
- raises式を含むことができません。raises式を含むオペレーションを起動すると、標準例外が発生する可能性があります。

クライアントは、メソッドの実行結果の応答を待たずに処理を進めることができます。ただし、メソッドの実行が異常終了しても、クライアントには通知されません。

onewayを指定しないと、メソッド実行が正常終了した場合はその復帰値が、例外が発生した場合は例外が通知されます。

復帰値のデータ型

IDLで定義できる任意の型を指定します。復帰値がない場合は、voidを指定します。

メソッド名

定義するオペレーションの名前を指定します。

パラメタ

0個または1個以上のパラメタを指定します。パラメタを複数指定する場合は、カンマ(,)で区切ります。

パラメタタイプ

クライアントアプリケーションとサーバアプリケーションとでパラメタが渡される方向を指定します。
パラメタタイプには、以下の3つのタイプがあります。

- in：パラメタは、クライアントからメソッド（サーバ）に渡されます。
- out：パラメタは、メソッド（サーバ）からクライアントに渡されます。
- inout：パラメタは、クライアントーサーバ間で双方向に渡されます。

inタイプは、関数呼出しなどで値を渡す場合に使用されます。
outタイプは、メソッドの実行結果として複数のデータを受け取る場合に使用します。受け取るデータが1つであれば、メソッドの復帰値（後述）として受け取ることもできます。
inoutタイプは、双方向にデータのやりとりするメソッド処理の場合に使用します。

inタイプのパラメタは、サーバ側で変更することができません。メソッド実行の結果、例外が発生した場合、データはoutタイプ／inoutタイプのパラメタ／復帰値に設定されません。

パラメタのデータ型

パラメタのデータ型として、以下が指定できます。

- 基本型
- シーケンス型
- 文字列型
- スコープ名

スコープ名は、上記で示したデータ型で定義されている必要があります。パラメタ名には、単独の識別子／配列が指定できます。

raises

raises式は、メソッドがユーザ例外で異常終了したときの詳細情報を受け取る例外構造体を定義します。通常、メソッドの呼出しは環境構造体(Environment)を付けて行われ、システム例外が発生すると、そのエラーコードが環境構造体に設定されます。raises式を定義すると、メソッドで異常が発生した場合に、メソッド固有のエラー(ユーザ例外)の情報を受け取ることができます。そのエラー情報は、環境構造体に設定されて渡されます。

context

context式は、クライアントアプリケーションとサーバアプリケーションで共有する情報を定義します。たとえば、通常、サーバアプリケーションは呼び出されたメソッドがどのクライアントアプリケーションからのものかを判断できませんが、context式でクライアントアプリケーション情報を指定することにより、サーバアプリケーションはクライアントアプリケーションを見分けることができます。
contextの詳細については、“12.3 Contextオブジェクト”を参照してください。

注意

Portable-ORBを使用した運用を行う場合、Context式は使用できません。

例

オペレーション宣言の例を以下に示します。

```
module Module1 {
    typedef Object otype;
    interface A {
        exception  FuncException {                  // 例外宣言
            string        reason;
        };

        Module1::otype  Open(in string name)        // オペレーション宣言
                raises(FuncException);
        oneway void  Close(in Module1::otype obj);  // オペレーション宣言
    };
};
```

A.5 属性宣言

インタフェースが持つデータを宣言します。
属性宣言の形式を以下に示します([ ]内は省略可能)。

```
[readonly] attribute  データ型  変数名;
```

変数名には、単独の識別子か配列が指定できます。readonlyを指定すると、読取り専用の変数が定義できます。readonlyを省略すると、読書き可能な変数が定義できます。
属性宣言の例を以下に示します。

```
attribute string   Name;
readonly attribute long Id;
```

A.6 定数宣言

定数宣言の書式を以下に示します。

```
const データ型  定数名 = 定数式;
```

定数宣言として使用できるデータ型を以下に示します。

- 整数型（long、short、unsigned long、unsigned short、long long）
- 浮動小数点型（float、double、long double）
- 文字型（char）
- ブーリアン型（boolean）
- 文字列型（string）

定数式は、左辺のデータ型と対応する必要があります。
また、定数式内では、異なるデータ型を混在させることはできません。たとえば、整数型と浮動小数点型を混在させることはできません。
定数式に使用可能な定数を以下に示します。

- 整数定数
- 文字定数
- 浮動小数点定数
- 文字列定数

定数式では、以下に示す演算子が使用できます。

	計算名	演算子	書き方	意味
単項演算子	補数	~	~n	nの補数を作ります。
	正符号	+	+n	正数を示します。
	負符号	-	-n	負数を示します。
2項演算子	加算	+	n1 + n2	n1とn2を加算します。
	減算	-	n1 - n2	n1からn2を減算します。
	乗算	*	n1 * n2	n1にn2を掛けます。
	除算	/	n1 / n2	n1をn2で割ります。
	剰余算	%	n1 % n2	n1をn2で割った余りです。
	左シフト	<<	n1 << n2	n1をn2ビット分左シフトします。 n2は、0以上32未満でなければなりません。
	右シフト	>>	n1 >> n2	n1をn2ビット分右シフトします。 n2は、0以上32未満でなければなりません。
	論理積(AND)	&	n1 & n2	n1とn2の論理積を作ります。

	計算名	演算子	書き方	意味
	論理和(OR)	|	n1 | n2	n1とn2の論理和を作ります。
	排他的論理和 (XOR)	^	n1 ^ n2	n1とn2の排他的論理和を作ります。

- 整数定数式に負の整数定数が含まれている場合は、signed longとして扱われます。負符号／負の整数定数が含まれていない場合は、unsigned longとして扱われます。
- 計算結果の値は、宣言された定数のデータ型に変換されます。計算結果の値がデータ型の範囲を超えた場合、または計算途中でsigned long／unsigned longの範囲を超えた場合は、IDLコンパイル時／生成されたスタブ・スケルトンのコンパイル時にエラーとなります。
- 整数定数式では、上記のすべての演算子が使用できます。
- 浮動小数点定数式は、倍精度(double)に変換されて倍精度で計算されます。計算結果の値は、宣言されたデータ型に変換されます。この変換に失敗した場合、または計算途中の値が倍精度の範囲を超えた場合は、エラーとなります。
- 浮動小数点定数式では、以下の演算子が使用できます。

  ― 単項演算子:+、-

  ― 2項演算子:*、/、+、-
- シフト演算を使用する場合、32ビット以上のデータ型(long long型)に対しても、n2に指定可能な最大値は31です。データ型の範囲以上のn2を指定した場合(short型に31ビットシフトなど)は、エラーとなります。

例

定数宣言の例を以下に示します。

```
const long l1  = 3;
const long l2  = (3 + 2);
const string S = "xxxxx";
typedef  long X;
const  X  l3 = 4;
```

A.7 データ型と型宣言

ここでは、IDLでサポートしているデータ型と型宣言について説明します。

A.7.1 IDLでサポートしているデータ型

IDLがサポートしているデータ型(型宣言で使用できるデータ型)を以下に示します。

データ型		型名	備考
基本データ型	整数型 (注)	short	符号つきshort:-2^{15}～$2^{15}-1$
		unsigned short	符号なしshort:0～$2^{16}-1$
		long	符号つきlong:-2^{31}～$2^{31}-1$
		unsigned long	符号なしlong:0～$2^{32}-1$
		long long	符号つきlong long:-2^{63}～$2^{63}-1$
	浮動小数点型	float	
		double	
		long double	

データ型		型名	備考
	文字型	char(8ビット)	システム間でコード系が異なると変換されます。
		wchar(16ビット)	
	オクテット型	octet(8ビット)	文字型と異なり、システム間でデータ変換されません。
	ブーリアン型	boolean	真(TRUE)か偽(FALSE)のどちらか一方だけを示します。
	文字列型	string	¥0で終了する文字配列
		wstring	¥0¥0で終了する2byte文字の配列
	列挙型	enum	
	any型	any	データ型を複合して使用するためのデータ型。構造体のメンバとして定義できます。
シーケンス型		sequence	
構造体		struct	
共用体		union	
固定小数点型		fixed	COBOLマッピング、OOCOBOLマッピングのみ
オブジェクトリファレンス		Object	
		interface	
タイプコード		TypeCode	

注)COBOLマッピング、OOCOBOLマッピングでは、以下の数値が使用可能です。他の言語とは異なるため、注意してください。

- long:-999,999,999～999,999,999
- unsigned long:0～999,999,999
- short:-9,999～9,999
- unsigned short:0～9,999
- long long:-999,999,999,999,999,999～999,999,999,999,999,999

注意

Windows64 Solaris64 Linux64

OOCOBOLは、Windows(R)クライアントアプリケーションで使用できます。

これらのデータ型には固定長と可変長があり、以下のように分類できます。

固定長データ	可変長データ
• 整数型 • 浮動小数点型 • 文字型 • オクテット型 • ブーリアン型 • 列挙型 • any型 • 文字列型、シーケンス型、固定小数点型を含まない構造体、および共用体	• 文字列型 • シーケンス型 • 固定小数点型 • 文字列型、シーケンス型、固定小数点型を含む構造体、および共用体 • オブジェクトリファレンス

コード変換の対象は、IDLファイルに記述された以下のデータです。

- char
- wchar
- string
- wstring

A.7.2 基本データ型

データ型宣言では、インタフェース内で使用されるデータ型を定義できます。既存のデータ型に別名を付ける場合などにも使用できます。基本データ型のうち、列挙型以外のデータ型は、型宣言にtypedefを使用します。

型宣言(文字列型、列挙型以外)の書式を以下に示します。

```
typedef long             データ型名;
typedef short            データ型名;
typedef unsigned long    データ型名;
typedef unsigned short   データ型名;
typedef long long        データ型名;

typedef float            データ型名;
typedef double           データ型名;
typedef long double      データ型名;

typedef char             データ型名;
typedef wchar            データ型名;

typedef octet            データ型名;

typedef boolean          データ型名;

typedef any              データ型名;
```

文字列型の型宣言

文字列型の型宣言の書式を以下に示します。

```
typedef string<サイズ>   データ型名;
typedef string           データ型名;

typedef wstring<サイズ>  データ型名;
typedef wstring          データ型名;
```

文字列型(string、wstring)には、サイズ指定のある文字列(上の書式)とサイズ指定のない文字列(下の書式)があります。サイズ指定のない文字列のサイズは、実行時に指定されます。

例

文字列型宣言の例を以下に示します。

```
typedef string<100> A;     (サイズ指定あり)
typedef string      A;     (サイズ指定なし)
```

注意

文字列型(string、wstring)は、文字型(char、wchar)の配列よりも性能面で優れています。文字列を扱う場合は、必要に応じて使い分けてください。

列挙型の型宣言

列挙型の型宣言では、typedefを使用しません。
形式を以下に示します。

```
enum　データ型名 { 要素, .. };
```

要素を複数指定する場合は、各要素をカンマ(,)で区切ります。1つの列挙型で指定できる要素の最大数は、2^{32}個です。

例

列挙型の記述例を以下に示します。

```
enum E {
    black, white, blue, red
};
```

注意

列挙型は、unsigned longとして扱われます。

値は、宣言された順に、0,1,2..となります。上記の例の場合は、black=0,white=1,blue=2,red=3となります。

A.7.3 シーケンス型

シーケンス型の宣言の書式を以下に示します。

```
typedef sequence<データ型, サイズ>　データ型名;
typedef sequence<データ型>　　データ型名;
```

シーケンス型は、データ型の1次元配列です。データ型として、基本データ型、シーケンス型(sequence)、構造体(struct)、共用体(union)、配列が指定できます。

シーケンス型には、サイズ指定のあるシーケンス(上の書式)とサイズ指定のない書式(下の書式)があります。サイズ指定のないシーケンスのサイズは、実行時に指定されます。

例

基本データ型long、およびユーザ定義型構造体のシーケンス型の宣言の例を以下に示します。

```
typedef sequence<long,10>　A;　(サイズ指定あり : long)
typedef sequence<long>　　B;　(サイズ指定なし : long)

struct sample {
    long　　x;
    char　　y;
};
```

```
typedef sequence<sample,30>  C;      (サイズ指定あり：ユーザ定義型構造体)
typedef sequence<sample>     D;      (サイズ指定なし：ユーザ定義型構造体)
```

注意

シーケンス型は、typedefで宣言する必要があります。このため、TypeCodeオブジェクトは、TC_sequenceではなく、TC_aliasとなります。TypeCodeオブジェクトについては、“12.1 TypeCodeオブジェクト”を参照してください。

A.7.4 構造体

構造体の宣言の書式を以下に示します。

```
struct データ型名 {
    構造体メンバの宣言
};
```

構造体メンバの宣言の書式を以下に示します。typedefは、使用しません。

```
基本データ型   メンバ名;
シーケンス型   メンバ名;
構造体         メンバ名;
共用体         メンバ名;
固定小数点型   メンバ名;
スコープ名     メンバ名;
```

メンバ名には、単独の識別子／配列を指定します。

構造体メンバは、1つ以上必要です。空の構造体は、許されていません。

例

構造体の記述例を以下に示します。

```
module A {
    typedef long B;
};
struct S {
    string    name;
    short     number;
    long      value;
    ::A::B    s;
};
```

再帰的な型宣言は、シーケンス型を使用したものだけが許されています。

```
struct  aaa {
    long  l;
    sequence<aaa>  bbb;
};
```

構造体名を定義するために、typedefを使用することもできます。

注意

IDLのstruct型は、C言語のstruct型と異なり、構造体の宣言を意味します。このため、構造体のメンバに構造体を宣言する場合は、以下のように記述します。

```
struct A {
    char xx;
};
struct B {
    A yy;
};
```

IDLのstruct型で構造体のメンバに構造体を宣言する場合、以下のように配列型のメンバ名にstruct構造体名と同じ名前を指定できません。

```
struct A {
    char xx;
};
struct B {
    A A[5];
};
```

A.7.5 共用体

共用体の宣言の形式を以下に示します。

```
union  データ型名  switch (データ型) {
    case 定数式 ;
        要素 [, ..]
         :
    default;
        要素 [, ..]
         :
};
```

要素の形式を以下に示します。typedefは、使用しません。

```
基本データ型   要素名;
シーケンス型   要素名;
構造体         要素名;
共用体         要素名;
固定小数点型   要素名;
スコープ名     要素名;
```

要素名には、単独の識別子／配列を指定します。

IDLの共用体は、C言語の共用体とswitch文とを合わせたものです。IDLの共用体の要素をcase文に関連付けて定義する必要があります。

共用体のswitch文の型定義には、どの要素が使用されるかを決定する型を指定する必要があります。

case文の定数式は、switch文の型定義と一貫性を持つ必要があります。定数式は、0～65535の間の値を指定します。

case文のdefaultは、1回だけ使用することが許可されています。

要素内のスコープ名は、定義済みの整数、文字、ブーリアン、列挙型のいずれかである必要があります。

case文の定数式は、switch文の型定義で指定された型と一致する必要があります。

一致の規則を以下に示します。

long:

longの範囲内のすべての整数値

short:

shortの範囲内のすべての整数値

unsigned long:

unsigned longの範囲内のすべての整数値

unsigned short:

unsigned shortの範囲内のすべての整数値

long long:

long longの範囲内のすべての整数値

char:

char

boolean:

真(TRUE)または偽(FALSE)

enum:

enumのすべてのメンバ

1つの共用体内で、要素名は一意である必要があります。

switchで指定した型が列挙型である場合、その識別子は共用体のスコープの中に入れ、要素名と異なっている必要があります。

共用体の型指定がとりうるすべての値をcase文で指定する必要はありません。

共用体の値は、型定義の値と以下のいずれかからなります。

- データ型がcase文に明示されている場合、要素の値はcase文の記述に対応します。
- defaultが指定されている場合、要素の値はdefaultの記述に対応します。
- それ以外の値は、ありません。

例

共用体の記述例を以下に示します。

```
union U switch (long) {
      case 1:
          long a;
      case 2:
          short b;
      default:
          char c;
};
```

A.7.6 固定小数点型

固定小数点型の形式を以下に示します。

```
typedef fixed<有効桁数, 小数点位置>  データ型名;
```

固定小数点型は、COBOLマッピング、OOCOBOLマッピングで使用可能です。
有効桁数には、数字項目(COBOL／OOCOBOL)の桁(有効桁)の長さを指定します。指定可能範囲は、1～18です。
小数点位置は数字項目での小数点位置を示すもので、小数点を有効桁の右端から左方向に何桁分移動するかを指定します。負値の場合は、右方向に移動することになります。指定可能範囲は、(有効桁数-18)～17です。

有効桁数と小数点位置の指定

IDL言語で固定小数点型fixedを指定した場合、COBOL／OOCOBOLで数字項目(PICTURE句)として定義されます。このとき、有効桁数と小数点位置により数字項目の定義形式が決定されます。

有効桁数／小数点位置によるCOBOL／OOCOBLでの桁の表現と数字項目の定義形式を以下に示します。有効桁を“|-|”、小数点位置を“v”、仮想桁を“|0|”とします。

fixed<5,2>の場合 (有効桁数:5桁、小数点位置:右端から左方向へ2桁)

```
|-|-|-|-|-|
      v
```

数字項目の定義形式

```
PIC  S9(3)V9(2)  PACKED-DECIMAL
```

fixed<5,-2>の場合 (有効桁数:5桁、小数点位置:右端から右方向へ2桁)

小数点が有効桁の範囲外に位置するため、範囲外の桁は0で埋められます。この範囲外の桁を仮想桁といいます。

```
|-|-|-|-|-|0|0|
              v
```

数字項目の定義形式

```
PIC  S9(5)P(2)V  PACKED-DECIMAL
```

fixed<2,5>の場合 (有効桁数:2桁、小数点位置:右端から左方向へ5桁)

```
|0|0|0|-|-|
v
```

数字項目の定義形式

```
PIC  SVP(3)9(2)  PACKED-DECIMAL
```

注意

- COBOL／OOCOBOLでは、有効桁＋仮想桁を18桁以下とする必要があります。
- Windows64 Solaris64 Linux64
  OOCOBOLは、Windows(R)クライアントアプリケーションで使用できます。

A.7.7 オブジェクトリファレンス

オブジェクトリファレンスの宣言の書式を以下に示します。

```
typedef Object       データ型名;
```

A.7.8 タイプコード

タイプコードの宣言の書式を以下に示します。

```
typedef TypeCode       データ型名;
```

A.7.9 配列

配列宣言では、多次元固定長の配列を定義します。配列宣言の書式を以下に示します。

```
typedef データ型　識別子 [ 配列サイズ ], ..;
```

配列の次元は5階層までに制限され、各次元で配列サイズを明示する必要があります。

配列サイズは、正の整数定数式で指定します。配列サイズは、コンパイル時に固定されます。

例

配列宣言の例を以下に示します。

```
typedef long A[5][10];
```

配列は、typedefで宣言する必要があります。このため、TypeCodeオブジェクトは、TC_arrayではなく、TC_aliasとなります。TypeCodeオブジェクトについては、“12.1 TypeCodeオブジェクト”を参照してください。

A.8 例外宣言

例外宣言では、オペレーション実行中に例外が発生したときに例外情報を受け渡すための識別子（または例外構造体名）を定義します。オペレーション宣言のraises式を定義する場合は、この構造体名を指定します。

例外宣言の書式を以下に示します。

```
exception 例外識別子(例外構造体名) {
    データ型　メンバ名;  // 構造体メンバを宣言
        :
};
```

例外識別子は、構造体メンバを宣言して構造体（例外構造体）として定義することもできます。構造体メンバを複数定義する場合は、各メンバをセミコロン（;）で区切ります。

メンバとして定義できるデータ型を以下に示します。

- 基本データ型
- シーケンス型
- 構造体
- 共用体
- 固定小数点型
- スコープ名

メソッドの実行結果として例外が返された場合、この例外識別子を参照できます。例外識別子にメンバ宣言がある場合は、そのメンバにアクセスできます。メンバ宣言がない場合は、その例外識別子だけを参照できます。

例

例外宣言の例を以下に示します。

```
exception  FuncException {    // 例外宣言
    string        reason;
};
```

また、例外にはシステムでの異常終了を通知するシステム例外とサーバアプリケーションでの異常終了を通知するユーザ例外があります。ただし、コンポーネントトランザクションサービスでは、システム例外は使用できません。

例外の使用方法については、各言語の“サーバアプリケーションの例外処理”を参照してください。

注意

C++言語の場合、構造体メンバの宣言にメンバ名として「id」を使用できません。

A.9 IDLのフォーマット

IDLの書式のフォーマットを以下に示します。::= は、以下のように定義されます。

```
|       または
<文字>      非終端文字
"文字"    リテラル
*         記号の前にある構文の要素を0回以上繰り返すことができます。
+         記号の前にある構文の要素を1回以上繰り返すことができます。
{}        { }で括られた構文の要素は1つの要素として取り扱われます。
[]        [ ]で括られた構文の要素はオプションで、0回以上現れます。
```

IDL仕様

```
<specification>      ::=<definition>+
```

IDLの定義

```
<definition>         ::=<type_dcl>";"
                     | <const_dcl>";"
                     | <except_dcl>";"
                     | <interface>";"
                     | <module>";"
```

モジュール宣言

```
<module>             ::="module"<identifier>"{"<definition>"+}"
```

インタフェース宣言

```
<interface>          ::=<interface_dcl>
                     | <forward_dcl>
<interface_dcl>      ::=<interface_header>"{"<interface_body>"}"
<forward_dcl>        ::="interface"<identifier>
<interface_header>   ::="interface"<identifier>[<inheritance_spec>]
<interface_body>     ::=<export>*
<export>             ::=<type_dcl>";"
                     | <const_dcl>";"
                     | <except_dcl>";"
                     | <attr_dcl>";"
                     | <op_dcl>";"
```

継承の仕様

```
<inheritance_spec>   ::=":"<scoped name>{","<scoped name>}*
<scoped_name>        ::=<identifier>
                     | "::"<identifier>
                     | <scoped_name>"::"<identifier>
```

定数宣言

```
<const_dcl>          ::="const"<const_type><identifier>"="<const_exp>
<const_type>         ::=<integer_type>
                     | <char_type>
                     | <boolean_type>
                     | <floating_pt_type>
                     | <string_type>
                     | <scoped_name>
```

定数式

```
<const_exp>          ::=<or_expr>
<or_expr>            ::=<xor_expr>
                     | <or_expr>"|"<xor_expr>
<xor_expr>           ::=<and_expr>
                     | <xor_expr>"^"<and_expr>
<and_expr>           ::=<shift_expr>
                     | <add_expr>"&"<shift_expr>
<shift_expr>         ::=<add_expr>
                     | <shift_expr>">>"<add_expr>
                     | <shift_expr>"<<"<add_expr>
<add_expr>           ::=<mult_expr>
                     | <add_expr>"+"<mult_expr>
                     | <add_expr>"-"<mult_expr>
<mult_expr>          ::=<unary_expr>
                     | <mult_expr>"*"<unary_expr>
                     | <mult_expr>"/"<unary_expr>
                     | <mult_expr>"%"<unary_expr>
<unary_expr>         ::=<unary_operator><primary_expr>
                     | <primary_expr>
<unary_operator>     ::="-"
                     | "+"
                     | "～"
<primary_expr>       ::=<scoped_name>
                     | <literal>
                     | "("<const_exp>")"
<literal>            ::=<integer_literal>
                     | <string_literal>
                     | <character_literal>
                     | <floating_literal>
                     | <boolean_literal>
<boolean_literal>    ::="TRUE"
                     | "FALSE"
<positive_int_const> ::=<const_exp>
```

型宣言

```
<type_dcl>           ::="typedef"<type_declarator>
                     | <struct_type>
                     | <union_type>
                     | <enum_type>
<type_declarator>    ::=<type_spec><declarators>
```

```
<type_spec>           ::=<simple_type_spec>
                      | <constr_type_spec>
<simple_type_spec>    ::=<base_type_spec>
                      | <template_type_spec>
                      | <scoped_name>
<base_type_spec>      ::=<floating_pt_type>
                      | <integer_type>
                      | <char_type>
                      | <wide_char_type>
                      | <boolean_type>
                      | <octet_type>
                      | <any_type>
                      | <object_type>
<template_type_spec>  ::=<sequence_type>
                      | <string_type>
                      | <wide_string_type>
                      | <fixed_pt_type>
<constr_type_spec>    ::=<struct_type>
                      | <union_type>
                      | <enum_type>
<declarators>         ::=<declarator>{","<declarator>}*
<declarator>          ::=<simple_declarator>
                      | <complex_declarator>
<simple_declarator>   ::=<identifier>
<complex_declarator>  ::=<array_declarator>
```

データ型

```
<floating_pt_type>    ::="float"
                      | "double"
                      | "long" "double"
<integer_type>        ::=<signed_int>
                      | <unsigned_int>
<signed_int>          ::=<signed_long_int>
                      | <signed_short_int>
                      | <signed_longlong_int>
<signed_long_int>     ::="long"
<signed_short_int>    ::="short"
<signed_longlong_int> ::="long""long"
<unsigned_int>        ::=<unsigned_long_int>
                      | <unsigned_short_int>
<unsigned_long_int>   ::="unsigned""long"
<unsigned_short_int>  ::="unsigned""short"
<char_type>           ::="char"
<wide_char_type>      ::="wchar"
<boolean_type>        ::="boolean"
<octet_type>          ::="octet"
<struct_type>         ::="struct"<identifier>"{"<member_list>"}"
<member_list>         ::=<member>+
<member>              ::=<type_spec><declarator>";"
<union_type>          ::="union"<identifier>"switch"
                      "("<switch_type_spec>")"
                      "{"<switch_body>"}"
<switch_type_spec>    ::=<integer_type>
                      | <char_type>
                      | <boolean_type>
                      | <enum_type>
                      | <scoped_name>
<switch_body> ::=<case>+
<case>                ::=<case_label>+<element_spec>";"
<case_label>          ::="case"<const_exp>";"
                      | "default"";"
```

```
<element_spec>        ::=<type_spec><declarators>
<sequence_type>       ::="sequence""<"<simple_type_spec>",""
                      <positive_int_const>">"
                      | "sequence""<"<simple_type_spec>>",""
<string_type>         ::="string""<"<positive_int_const>">"
                      | "string"
<wide_string_type>    ::="wstring""<"<positive_int_const>">"
                      | "wstring"
<fixed_pt_type>       ::="fixed""<"<positive_int_const>","<integer_literal>">"
<array_declarator>    ::=<indentifier><fixed_array_size>+
<fixed_array_size>    ::="["<positive_int_const>"]"
<attr_dcl>            ::=["readonly"]"attribute"<param_type_spec>
                      <simple_declarator>{","<simple_declarator>}*
```

例外宣言

```
<except_dcl>          ::="exception"<identifier>"{"<member>*"}"
```

オペレーション宣言

```
<op_dcl>              ::=[<op_attribute>]<op_type_spec><identifier>
                      <parameter_dcls>[<raises_expr>][<context_expr>]
<op_attribute>        ::="oneway"
<op_type_spec>        ::=<param_type_spec>
                      | "void"
<parameter_dcls>      ::="("<param_dcl>{","<param_dcl>}*")"
                      | "("")"
<param_dcl>           ::=<param_attribute><param_type_spec> <simple_declarator>
<param_attribute>     ::="in"
                      | "out"
                      | "inout"
<raises_expr>         ::="raises""("<scoped_name>{","<scoped_name>}*")"
<context_expr>        ::="context""("<string_literal>{","<string_literal>}*")"
<param_type_spec>     ::=<base_type_spec>
                      | <string_type>
                      | <wide_string_type>
                      | <fixed_pt_type>
                      | <scoped_name>
```

A.10 コンポーネントトランザクションサービスにおけるIDLの利用範囲

Windows32 Solaris32 Linux32

コンポーネントトランザクションサービスではIDLの文法に準拠していますが、次のように利用範囲を限定しています。

トランザクションアプリケーション向けの利用範囲を、以下に示します。

データ型			サポート可否	備考
基本データ型	整数型(注)	long long long short unsigned long unsigned short	○	符号つきlong long:-2^{63}～$2^{63}-1$ 符号つきlong :-2^{31}～$2^{31}-1$ 符号つきshort :-2^{15}～$2^{15}-1$ 符号なしlong :0～$2^{32}-1$ 符号なしshort :0～$2^{16}-1$
	浮動小数点型	float double long double	△	COBOLでは、long doubleは未サポートです。
	文字型	char wchar	○	

データ型			サポート可否	備考
	オクテット型	octet	○	
	ブーリアン型	boolean	○	
	列挙型	enum	×	
	any型	any	×	
	文字列型	string wstring	○	
シーケンス型		sequence	△	要素のデータ型は、基本データ型とします。
配列		array	△	配列要素のデータ型は、基本データ型および構造体とします。また、配列の次元数は1次元までとします。
構造体		struct	△	構造体メンバのデータ型は、基本データ型だけとします。
共用体		union	×	
固定小数点型		fixed	×	
オブジェクトリファレンス		Object	×	
		interface	×	
タイプコード		TypeCode	×	

○:サポート
△:一部サポート
×:未サポート

注)COBOLマッピングでは、以下の数値が使用可能です。他言語とは異なるため、注意してください。

- long:-999,999,999～999,999,999
- unsigned long:0～999,999,999
- short:-9,999～9,999
- unsigned short:0～9,999
- long long:-999,999,999,999,999,999～999,999,999,999,999,999

また、IDLの文法全体に対して、利用不可の項目を以下に示します。

(1) モジュールのないインタフェース

未サポート

例

```
module A {
   interface B {
       ...
    };
};
interface C {};      → 不可
```

(2) Web連携時におけるモジュール宣言の入れ子

未サポート

(3) 属性

未サポート

(4) 例外

一部サポート
基本データ型、配列以外は、使用できません。
配列要素のデータ型に、基本データ型以外を使用できません。
また、配列の次元数は1次元までとします。

例

```
typedef struct  A {
    long   a1;
} A_str;
    exception  B {
         A_str   b1[4];
};
```

(5) コンテキスト

未サポート

(6) オブジェクト名の文字数が255文字を超えている

未サポート

(7) 復帰値のデータ型がlong、oneway void以外のオペレーション

未サポート

(8) struct の typedef宣言

一部サポート
サーバアプリケーション言語がCOBOL言語の場合は、使用できません。

(9) インタフェースの継承記述以外のスコープ記述

例

```
module M1 {
    interface I1 {
        typedef long L1;
        long ope1();
    };
    interface I2 {
        long ope2(in M1::I1::L1 para1);  → 不可
    };
};
```

注意

以下の予約語は、TDであらかじめ用途が決められた固有の意味を持つ語のため、識別子に記述できません。

- TD_RTNVAL

付録B　提供プログラム

注意

ここでは、格納パスに記載されているディレクトリは、インストールパスをデフォルトで示しています。

B.1　CORBAサービスの提供プログラム

B.1.1　インクルードファイル

アプリケーションが各サービスを使用する場合に必要となるインクルードファイルを以下に示します。

格納パス Windows32/64

```
C:¥Interstage¥ODWIN¥INCLUDE
```

格納パス Solaris32/64

```
/opt/FSUNod/include
```

格納パス Linux32/64

```
/opt/FJSVod/include
```

インクルードファイル Windows32/64

開発言語	サービス	インクルードファイル名
C言語	ORBコア（注1）	orb.h
	インタフェースリポジトリ	INTFREP.h
	ネーミングサービス	COSNAMIN.h
	Windows32 ロードバランス	OM_LBO.h
C++言語	ORBコア（注1）	orb_cplus.h
	Windows32 POAランタイム	OM_CORBA_poa.h（注4） poa.h common_poa.h（注4） servant.h（注4） serverrequest.h
	インタフェースリポジトリ	orb_cplus.h（注1）
	ネーミングサービス	CosNaming_cplus.h
	Windows32 ロードバランス	OM_LBOcpp.h
COBOL（注2）		COBOL¥*
Windows32 OOCOBOL（注3）		OOCOB¥*

注1）ORBコアのインクルードファイルに含まれます。

注2)“6.11 COBOL用登録集について”を参照してください。

注3)“G.3.10 OOCOBOL用登録集について”を参照してください。

注4)C++言語 POAランタイムのインクルードファイルpoa.hに含まれます。

インクルードファイル Solaris32/64 Linux32/64

開発言語	サービス	インクルードファイル名
C言語	ORBコア（注1）	orb.h
	インタフェースリポジトリ	InterfaceRep.h
	ネーミングサービス	CosNaming.h
	Solaris32 Linux32 ロードバランス	OM_LBO.h
C++言語	ORBコア（注1）	orb_cplus.h
	Solaris32 Linux32 POAランタイム	OM_CORBA_poa.h（注4） poa.h common_poa.h（注4） servant.h（注4） serverrequest.h
	インタフェースリポジトリ	orb_cplus.h（注1）
	ネーミングサービス	CosNaming_cplus.h
	Solaris32 Linux32 ロードバランス	OM_LBOcpp.h
COBOL（注2）	―	COBOL/*
OOCOBOL（注3）	―	oocob/*

注1)ORBコアのインクルードファイルに含まれます。

注2)“6.11 COBOL用登録集について”を参照してください。

注3)“G.3.10 OOCOBOL用登録集について”を参照してください。

注4)C++言語 POAランタイムのインクルードファイルpoa.hに含まれます。

インクルードファイル名 Windows32/64

インストールされるインクルードファイルは、一部、マニュアル(“リファレンスマニュアル(API編)”など)に記述されているファイル名(Solaris／Linuxシステムでのファイル名)と異なるものがあります。アプリケーション作成時には、注意してください。

マニュアル上のファイル名	Windows(R)でのファイル名
CosNaming.h	COSNAMIN.H
InterfaceRep.h	INTFREP.H
OM_impl_rep.h	OM_IMR.H
OM_intf_rep.h	OM_IFR.H
OM_ORB.h	OM_O.H
OM_stub_skel.h	OM_STSK.H

B.1.2 ライブラリ

CORBAサービス(クライアントを含む)の提供するサービスとライブラリについて説明します。アプリケーションのリンク(ビルド)時に、ORBコアおよび必要なサービスのライブラリを指定してください。

B.1.2.1 サーバ用ライブラリ

アプリケーションが各サービスを使用する場合に必要となるライブラリを以下に示します。

格納パス(Java言語以外) Windows32

```
C:¥Interstage¥ODWIN¥LIB
```

格納パス(Java言語以外) Windows64

```
C:¥Interstage¥ODWIN¥LIB¥x64
```

格納パス(Java言語) Windows32/64

```
C:¥Interstage¥ODWIN¥ETC¥CLASS
```

ポイント

Windows32/64

サーバ用ライブラリは、スレッドモード、プロセスモードとも、上記に格納されます。

格納パス(Java言語以外) Solaris32/64

```
/opt/FSUNod/lib
```

/usr/lib配下にシンボリックリンクファイルが作成されます。

格納パス(Java言語) Solaris32/64

```
/opt/FSUNod/etc/class
```

格納パス(Java言語以外) Linux32/64

```
/opt/FJSVod/lib
```

/usr/lib配下にシンボリックリンクファイルが作成されます。

格納パス(Java言語) Linux32/64

```
/opt/FJSVod/etc/class
```

ポイント

Solaris32/64 Linux32/64

上記ディレクトリ($OD_HOME/lib)には、スレッドモード用のlibOM.soが格納されます。プロセスモード用のlibOM.soは$OD_HOME/lib/ntに格納されます。なお、プロセスモード用のlibOM.soを使用する場合は、LD_LIBRARY_PATH=$OD_HOME/lib/ntを設定してください。

サーバ用ライブラリ Windows32/64

開発言語	サービス	ライブラリ名
C言語	ORBコア（注1）	ODSV.LIB
	インタフェースリポジトリ	ODIFSV.LIB
	ネーミングサービス	ODCNSV.LIB
	Windows32 ロードバランス	ODLBOSV.LIB
C++言語	ORBコア（注1）	ODSV.LIB ODSVCPP.LIB
	Windows32 POAランタイム	ODSVCPPPOA.LIB
	インタフェースリポジトリ	（注2）
	ネーミングサービス	ODCNSCPP.LIB
	Windows32 ロードバランス	ODLBSCPP.LIB
Java言語		ODjava4.jar
COBOL (スレッドモード)	ORBコア（注1）	ODCOBCBLMTSV.LIB(UNICODE以外) ODCOBCBLSVUC.LIB(UNICODE用)（注3）
	インタフェースリポジトリ	LIBOMIRCBLMTSV.LIB(UNICODE以外) LIBOMIRCBLSVUC.LIB(UNICODE用)（注3）
	ネーミングサービス	（注2）
	Windows32 ロードバランス	ODLBSCBLMT.LIB
COBOL (プロセスモード)	ORBコア（注1）	ODCOBCBLSV.LIB(UNICODE以外) ODCOBCBLSVUC.LIB(UNICODE用)（注3）
	インタフェースリポジトリ	LIBOMIRCBLSV.LIB(UNICODE以外) LIBOMIRCBLSVUC.LIB(UNICODE用)（注3）
	ネーミングサービス	（注2）
	Windows32 ロードバランス	ODLBSCBL.LIB
Windows32 OOCOBOL	ORBコア（注1）	ODOOCOBSV.LIB(UNICODE以外) ODOOCOBSVUC.LIB(UNICODE用)（注3）
	インタフェースリポジトリ	LIBOMIROOCOB.LIB(UNICODE以外) LIBOMIROOCOBUC.LIB(UNICODE用)（注3）
	ネーミングサービス	ODCNSOOCOB.LIB(UNICODE以外) ODCNSOOCOBUC.LIB(UNICODE用)（注3）
	ロードバランス	ODLBSOOCOB.LIB

注1）ORBコア以外のサービスを使用する場合も、ORBコアのライブラリが必要です。

注2）ORBコアのライブラリに含まれます。

注3）UNICODEを使用するアプリケーションで指定してください。

サーバ用ライブラリ Solaris32/64 Linux32/64

開発言語	サービス	ライブラリ名
C言語	ORBコア（注1）	libOM.so
	インタフェースリポジトリ	libOMir.so
	ネーミングサービス	libOMcn.so
	Solaris32 Linux32 ロードバランス	libOMlbo.so
Solaris32/64 C++言語	ORBコア	libOM.so libOMcpp50.so
	Solaris32 POAランタイム	libOMcpppoa50.so
	インタフェースリポジトリ	（注2）
	ネーミングサービス	libOMcncpp50.so
	Solaris32 ロードバランス	libOMlbocpp50.so
Linux32/64 C++言語	ORBコア（注1）	libOM.so libOMcpp.so
	Linux32 POAランタイム	libOMcpppoa.so
	インタフェースリポジトリ	（注2）
	ネーミングサービス	libOMcncpp.so
	Linux32 ロードバランス	libOMlbocpp.so
Java言語		ODjava4.jar
COBOL (スレッドモード)	ORBコア（注1）	libOMcblMT.so(UNICODE以外) libOMcblUC.so(UNICODE用)（注3） Linux64 libOMcblUCBE.so(ビッグエンディアンのUNICODE用)（注4） libOMcblUCLE.so(リトルエンディアンのUNICODE用)（注5）
	インタフェースリポジトリ	libOMircblMT.so(UNICODE以外) libOMircblUC.so(UNICODE用)（注3） Linux64 libOMircblUCBE.so(ビッグエンディアンのUNICODE用)（注4） libOMircblUCLE.so(リトルエンディアンのUNICODE用)（注5）
	ネーミングサービス	（注2）
	Linux32 ロードバランス	libOMlbocblMT.so
COBOL (プロセスモード)	ORBコア（注1）	libOMcbl.so
	インタフェースリポジトリ	libOMircbl.so
	ネーミングサービス	（注2）

開発言語	サービス	ライブラリ名
	Linux32 ロードバランス	libOMlbocbl.so
OOCOBOL	ORBコア（注1）	libOMoocob.so(UNICODE以外) libOMoocobUC.so(UNICODE用)（注3）
	インタフェースリポジトリ	libOMiroocob.so(UNICODE以外) libOMiroocobUC.so(UNICODE用)（注3）
	ネーミングサービス	libOMcnoocob.so(UNICODE以外) libOMcnoocobUC.so(UNICODE用)（注3）
	Linux32 ロードバランス	libOMlbooocob.so

注1）ORBコア以外のサービスを使用する場合も、ORBコアのライブラリが必要です。

注2）ORBコアのライブラリに含まれます。

注3）UNICODEを使用するアプリケーションで指定してください。

注4）UNICODEを使用するアプリケーションで、RCSオプションでビッグエンディアンを選択する場合に指定してください。

注5）UNICODEを使用するアプリケーションで、RCSオプションでリトルエンディアンを選択する場合に指定してください。

B.1.2.2 クライアント用ライブラリ

アプリケーションが各サービスを使用する場合に必要となるライブラリを以下に示します。

格納パス（Java言語以外）

```
C:¥Interstage¥ODWIN¥LIB
```

格納パス（Java言語）

```
C:¥Interstage¥ODWIN¥ETC¥CLASS
```

開発言語	サービス	ライブラリ名
C言語	ORBコア（注1）	ODWIN.LIB
	インタフェースリポジトリ	ODIF.LIB
	ネーミングサービス	ODCN.LIB
C++言語	ORBコア（注1）	ODWIN.LIB ODWINCPP.LIB
	インタフェースリポジトリ	（注2）
	ネーミングサービス	ODCNCPP.LIB
Java言語		ODJAVA4.JAR
COBOL (スレッドモード)	ORBコア（注1）	ODCOBCBLMT.LIB(UNICODE、UTF8以外) ODCOBCBLUC.LIB(UNICODE、UTF8用)（注3）
	インタフェースリポジトリ	LIBOMIRCBLMT.LIB(UNICODE、UTF8以外) LIBOMIRCBLUC.LIB(UNICODE、UTF8用)（注3）
	ネーミングサービス	（注2）

開発言語	サービス	ライブラリ名
COBOL (プロセスモード)	ORBコア（注1）	ODCOBCBL.LIB(UNICODE、UTF8以外) ODCOBCBLUC.LIB(UNICODE、UTF8用)（注3）
	インタフェースリポジトリ	LIBOMIRCBL.LIB(UNICODE、UTF8以外) LIBOMIRCBLUC.LIB(UNICODE、UTF8用)（注3）
	ネーミングサービス	（注2）
OOCOBOL	ORBコア（注1）	ODOOCOB.LIB(UNICODE、UTF8以外) ODOOCOBUC.LIB(UNICODE、UTF8用)（注3）
	インタフェースリポジトリ	LIBOMIROOCOB.LIB(UNICODE、UTF8以外) LIBOMIROOCOBUC.LIB(UNICODE、UTF8用)（注3）
	ネーミングサービス	ODCNOOCOB.LIB(UNICODE、UTF8以外) ODCNOOCOBUC.LIB(UNICODE、UTF8用)（注3）

注1）ORBコア以外のサービスを使用する場合も、ORBコアのライブラリが必要です。

注2）ORBコアのライブラリに含まれます。

注3）UNICODE、UTF8を使用するアプリケーションで指定してください。

B.2 Portable-ORBの提供プログラム

Portable-ORBが提供するライブラリについて説明します。

格納パス Windows32/64

```
C:¥Interstage¥PORB¥LIB
```

格納パス Solaris32/64 Linux32/64

```
/opt/FJSVporb/lib
```

開発言語	運用形態	サービス	ライブラリ名
Java言語	Javaアプリケーション	ORBコア	ODporb4.jar
		インタフェースリポジトリ	InterfaceRep4.jar
		ネーミングサービス	CosNaming4.jar
	Javaアプレット （JBKプラグイン）	ORBコア	ODporb4_plugin.jar
		インタフェースリポジトリ	InterfaceRep4_plugin.jar
		ネーミングサービス	CosNaming4_plugin.jar

付録C　インタフェース定義情報の移出入

CORBAサービス上で開発した分散アプリケーションで、動的起動インタフェースを使用する場合に必要な運用サーバのインタフェースリポジトリへのインタフェース定義情報の登録方法について説明します。

C.1　インタフェース定義情報の登録

CORBAサービス上で開発した分散アプリケーションで動的起動インタフェースを使用する場合、運用サーバのインタフェースリポジトリにはリクエストの組立てに必要なインタフェース定義情報を事前に登録しておく必要があります。

インタフェース定義情報の登録は、以下の方法があります。

- IDLコンパイラを使用して登録する方法
- 移出入コマンドを使用して登録する方法

移出入コマンドを使用したインタフェース定義情報の登録方法について、以下に説明します。

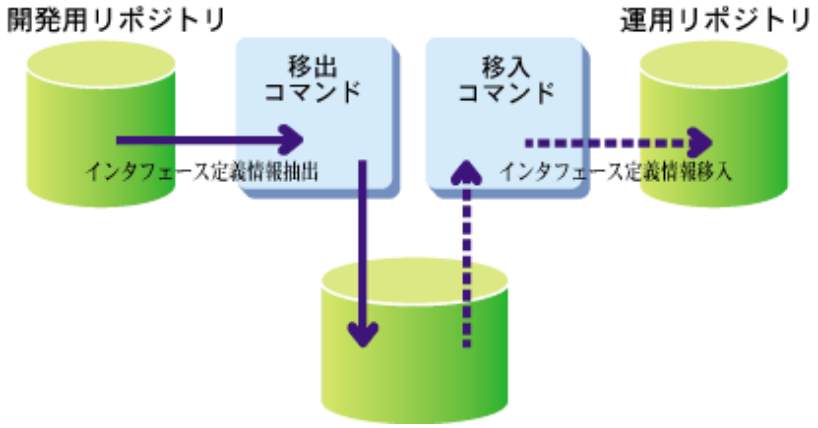
C.2　プログラム開発から運用への手順

登録に必要なインタフェース定義情報は、移出コマンドを使用して開発用のインタフェースリポジトリから移出します。

移出コマンドは、インタフェースリポジトリから指定したインタフェース／包含されるインタフェース定義情報を移出し、ファイルに出力します（以降、このファイルのことをインタフェース情報ファイルと呼びます）。

インタフェース情報ファイルはシステムに依存しないため、分散アプリケーションに添付する形で配布できます。

分散アプリケーションを実装する各サーバでは、移入コマンドを使用してインタフェース情報ファイルよりインタフェース定義情報を登録できます。



C.3　実行例

C.3.1　インタフェース定義情報の移出

odexportirコマンドを使用してインタフェース定義情報を移出します。

例

インタフェースリポジトリからmod1モジュールおよびmod1に包含されるインタフェース定義情報を移出し、intdefという名前のインタフェース情報ファイルを生成します。

```
odexportir -r IDL:mod1:1.0 intdef
```

注意

Windows32/64

移出コマンドを実行する場合は、環境変数TZにタイムゾーンを設定する必要があります。

```
TZ = "JST-9"
```

本設定を行わない場合、作成されたインタフェース情報ファイルの移出時刻情報が正しく設定されないことがあります。

C.3.2 インタフェース定義情報の移入

odimportirコマンドを使用してインタフェース定義情報を登録します。

例

インタフェースリポジトリにインタフェース定義情報ファイル(intdef)の内容を登録します。

```
odimportir intdef
```

注意

移入時に、既存のインタフェース定義情報を上書きする可能性があります。移入コマンドを実行する前に、必ずodbackupsysコマンドでバックアップを行い、資源を退避してください。

付録D　サンプルプログラム

D.1　サンプルプログラムについて

本節では、CORBAサービスおよびPortable-ORBが提供するサンプルプログラムについて、その種別と格納されているフォルダ、ディレクトリ、およびその概要を説明します。
サンプルプログラムの作成・実行の手順は、“D.3 実行手順（Windows(R)）”および“D.4 実行手順（Solaris/Linux）”を参照してください。

D.1.1　サンプルプログラムの種別

サンプルプログラムの種別と格納されているフォルダ、ディレクトリを以下に示します。

Windows32/64

種別		フォルダ(注)	言語
静的インタフェース	各種データ型	%OD_SAMPLES%¥complex¥samplelist.C¥data¥*	C
		%OD_SAMPLES%¥complex¥samplelist.C++¥data¥*	C++
		%OD_SAMPLES%¥complex¥samplelist.Java¥data¥*	Java
		%OD_SAMPLES%¥complex¥samplelist.COBOL¥data¥*	COBOL
	四則演算	%OD_SAMPLES%¥CalcSample¥c_s, c_c	C
		%OD_SAMPLES%¥CalcSample¥c++_s, c++_c	C++
		%OD_SAMPLES%¥CalcSample¥java	Java
		%OD_SAMPLES%¥CalcSample¥COBOL	COBOL
動的起動インタフェース	基本	%OD_SAMPLES%¥complex¥samplelist.C¥dii¥*	C
		OD_SAMPLES%¥complex¥samplelist.C++¥dii¥*	C++
	四則演算	%OD_SAMPLES%¥CalcSample¥c_dii	C
		%OD_SAMPLES%¥CalcSample¥c++_dii	C++
ネーミングサービス		%OD_SAMPLES%¥complex¥samplelist.C¥naming¥*	C
		%OD_SAMPLES%¥complex¥samplelist.C++¥naming¥*	C++
インタフェースリポジトリ		%OD_SAMPLES%¥irsample¥*	C／C++／COBOL
POAインタフェース		Windows32 %OD_SAMPLES%¥POA¥C++	C++
		%OD_SAMPLES%¥POA¥java_1_4	Java
各種API		%OD_SAMPLES%¥complex¥samplelist.C¥api¥*	C
		%OD_SAMPLES%¥complex¥samplelist.C++¥api¥*	C++
		%OD_SAMPLES%¥complex¥samplelist.Java¥api¥*	Java
プロセスバインド		%OD_SAMPLES%¥complex¥samplelist.C++¥ssn¥*	C++
Portable-ORB	各種データ型	%OD_SAMPLES%¥complex¥samplelist.Java¥data¥*	Java
	POAインタフェース	%OD_SAMPLES%¥POA¥java_1_4¥*	Java
	四則演算	%PORB_SAMPLES%¥CalcSample¥java	Java
	各種API	%OD_SAMPLES%¥complex¥samplelist.Java¥api¥*	Java
Windows32 その他		%OD_SAMPLES%¥simple_s, simple_c	C

種別	フォルダ(注)	言語
	%OD_SAMPLES%¥simple_cpp_s, simple_cpp_c	C++
	%OD_SAMPLES%¥java	Java

注)
ここで使用している環境変数は、以下です(Interstageのインストールフォルダ:デフォルト)。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample
PORB_SAMPLES	C:¥Interstage¥Porb¥src¥samples

Solaris32/64 Linux32/64

種別		ディレクトリ(注1)	言語
静的インタフェース	各種データ型(スレッドモード用)	$OD_SAMPLES/complex/samplelist.C/data/*	C
		$OD_SAMPLES/complex/samplelist.C++/data/*	C++
		$OD_SAMPLES/complex/samplelist.Java/data/*	Java
		$OD_SAMPLES/complex/samplelist.C/data_win/*(注2)	C
	各種データ型(プロセスモード用)	$OD_SAMPLES/complex/samplelist.C_nt/data/*	C
		$OD_SAMPLES/complex/samplelist.C++_nt/data/*	C++
		$OD_SAMPLES/complex/samplelist.C++_nt/data/*	C++
	四則演算	$OD_SAMPLES/CalcSample/c	C
		$OD_SAMPLES/CalcSample/c++	C++
		$OD_SAMPLES/CalcSample/java(注2)	Java
		$OD_SAMPLES/CalcSample/COBOL	COBOL
動的起動インタフェース	基本(スレッドモード用)	$OD_SAMPLES/complex/samplelist.C/dii/*	C
		$OD_SAMPLES/complex/samplelist.C++/dii/*	C++
	基本(プロセスモード用)	$OD_SAMPLES/complex/samplelist.C_nt/dii/*	C
		$OD_SAMPLES/complex/samplelist.C++_nt/dii/*	C++
	四則演算	$OD_SAMPLES/CalcSample/c_dii	C
		$OD_SAMPLES/CalcSample/c++_dii	C++
ネーミングサービス		$OD_SAMPLES/complex/samplelist.C/naming/*	C
		$OD_SAMPLES/complex/samplelist.C++/naming/*	C++
インタフェースリポジトリ		$OD_SAMPLES/irsample/*	C／C++／COBOL
POAインタフェース		Solaris32 Linux32 $OD_SAMPLES/POA/C++/*	C++
		$OD_SAMPLES/POA/java_1_4/*	Java
プロセスバインド		$OD_SAMPLES/complex/samplelist.C++/ssn/*	C++
各種API	スレッドモード用	$OD_SAMPLES/complex/samplelist.C/api/*	C
		$OD_SAMPLES/complex/samplelist.C++/api/*	C++
		$OD_SAMPLES/complex/samplelist.Java/api/*	Java
	プロセスモード用	$OD_SAMPLES/complex/samplelist.C_nt/api/*	C

種別		ディレクトリ(注1)	言語
		$OD_SAMPLES/complex/samplelist.C++_nt/api/*	C++
Portable-ORB	各種データ型	$OD_SAMPLES/complex/samplelist.Java/data/*	Java
	POAインタフェース	$OD_SAMPLES/POA/java_1_4/*	Java
	四則演算	$PORB_SAMPLES/CalcSample/java	Java
	各種API	$OD_SAMPLES/complex/samplelist.Java/API/*	Java
その他	スレッドモード用	$OD_SAMPLES/simple	C
		$OD_SAMPLES/simple_cpp	C++
		$OD_SAMPLES/java(注2)	Java
	プロセスモード用	$OD_SAMPLES/complex/samplelist.C_nt/simple	C
		$OD_SAMPLES/complex/samplelist.C++_nt/simple_cpp	C++
	soライブラリ化	$OD_SAMPLES/simple_so	C
		$OD_SAMPLES/simple_cpp_so	C++

注1)
ここで使用している環境変数は、以下です(Interstageのインストールパス:デフォルト)。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64
PORB_SAMPLES	/opt/FJSVporb/src/samples	

注2)
Windows(R)クライアント用のサンプルプログラムです。

D.1.2 サンプルプログラムの概要

サンプルプログラムの概要について以下に説明します。

(1) 静的インタフェース

静的起動・静的スケルトンインタフェースを使用するサンプルプログラムとその種類を以下に示します。

■各種データ型

項	種類	概要	備考
1	any	any型(構造体)のデータを受け渡します。	
2	any2	any型(long、string、sequence)のデータを受け渡します。	C／Java
3	array	array型のデータを受け渡します。	
4	attribute	attributeとしてlong、stringのデータを受け渡します。	
5	context	context型のデータを受け渡します。	
6	exception	ユーザ定義型の例外を受け渡します。	C／C++／Java
7	sequence	sequence型(long)のデータを受け渡します。	
8	sequence2	sequence型(octet)のデータを受け渡します。	C／C++／Java
9	string	string型のデータを受け渡します。	
10	struct	long型とstring型を要素として持つ構造体型のデータを受け渡します。	

項	種類	概要	備考
11	union	共用体型のデータを受け渡します。	
12	Solaris32/64 Linux32/64 その他	基本データ型のデータを受け渡します。	COBOL

■四則演算

項	種類	概要	備考
1	c	ゼロ除算を行い、例外を発生させます。	
2	c++	ゼロ除算を行い、例外を発生させます。	
3	COBOL	ゼロ除算を行い、例外を発生させます。	
4	java	Webブラウザから入力された値の四則演算を行い、その結果を表示します。	

(2) 動的起動インタフェース

動的起動インタフェースを使用するサンプルプログラムとその種類を以下に示します。

■基本

項	種類	概要	備考
1	dyn1	long型のデータを受け渡します。	
2	dyn2	long型とchar型を要素として持つ構造体型のデータを受け渡します。	
3	dyn3	共用体型のデータを受け渡します。	
4	dii_long	long型のデータを受け渡します。	C++
5	dii_string	string型のデータを受け渡します。	C++
6	dii_struct	構造体型のデータを受け渡します。	C++
7	dii_sequence	sequence型のデータを受け渡します。	C++

■四則演算

項	種類	概要	備考
1	c_dii	四則演算を行い、その結果を表示します。	
2	c++_dii	四則演算を行い、その結果を表示します。	

(3) ネーミングサービス

ネーミングサービスを使用するサンプルプログラムとその種類を以下に示します。

項	種類	概要	備考
1	list	ネーミングサービスのlistメソッドを実行します。	
2	bind	ネーミングサービスのbindメソッドを実行します。	
3	resolve	ネーミングサービスのresolve メソッドを実行します。	
4	destroy	ネーミングサービスのunbind/destroyメソッドを実行します。	

(4) インタフェースリポジトリ

インタフェースリポジトリを使用するサンプルプログラムとその種類を以下に示します。

項	種類	概要	備考
1	csample1 cppsample1 cblsample1	OperationDefオブジェクト情報を獲得します。	
2	csample2 cppsample2 cblsample2	StructDef,AliasDefオブジェクト情報を獲得します。	

(5) POAインタフェース

POAインタフェースを使用するサンプルプログラムとその種類を以下に示します。

項	種類	概要	備考
1	DefaultServant	デフォルトインスタンスを使用します。 各クライアントは、同一のServantオブジェクトを使用する方式です。	
2	Factory-1	Factoryオブジェクトを使用してクライアントごとに異なるServantオブジェクトを使用する方式です。Factory内でオブジェクトリファレンスおよびインスタンスを作成します。	
3	Factory-2	Factoryオブジェクトを使用してクライアントごとに異なるServantオブジェクトを使用する方式です。Factory内でオブジェクトリファレンスを作成し、リクエスト受信時にServantObject内でインスタンスを作成します。	
4	ServantLocator	ユーザが作成するServantManagerオブジェクトにインスタンス管理を任せる方式です。	
5	AdapterActivator-1	必要なときにPOAオブジェクトを作成します。複数のPOAオブジェクトを実装するアプリケーションに適しています。find_POAメソッドが呼ばれたときにAdapterActivatorが動作します。	
6	AdapterActivator-2	最初のリクエスト受信時にPOAオブジェクトを作成します。複数のPOAオブジェクトを実装するアプリケーションに適しています。最初のリクエスト受信時に動作します。	
7	Windows32 Solaris32 Linux32 DefaultServant_tie	デフォルトインスタンスの代理方式を使用します。 各クライアントは、同一のServantオブジェクトを使用する方式です。	C++
8	Windows32 Solaris32 Linux32 Factory_1_instance_tie	クライアントとのコネクション切断時にインスタンス解放を行う方式です。 Factoryオブジェクトを使用してクライアントごとに異なるServantオブジェクトを使用する方式の、代理方式での実装です。Factory内でオブジェクトリファレンスおよびインスタンスを作成します。	C++

(6) プロセスバインド

プロセスバインドを使用するサンプルプログラムとその種類を以下に示します。

項	種類	概要	備考
1	ssn_sample	プロセスバインドを行います。	
2	ssn_sample2	プロセスバインドを行い、セションタイムアウトが発生します。	

(7) 各種API

各種APIを使用するサンプルプログラムとその種類を以下に示します。

項	種類	概要	備考
1	net_disconnect	通信資源の解放を行います。	
2	set_client_request_timer	サーバメソッド待機時間を設定します。	
3	string_to_object	ファイルによりオブジェクトリファレンスを配布します。	

(8) Portable-ORB

Portable-ORBのサンプルプログラムとその種類を以下に示します。

■各種データ型

静的起動インタフェースの“各種データ型”を参照してください。

■POAインタフェース

“(5) POAインタフェース”を参照してください。

■四則演算

項	種類	概要	備考
1	java	四則演算を行い、その結果を表示します。	

■各種API

“(7) 各種API”を参照してください。

(9) その他

上記以外のサンプルプログラムを以下に示します。

Windows32

項	種類	概要	備考
1	simple	加算とその結果を返します。	
2	simple_cpp	クライアントから入力された値の累積値を返します。	
3	java	Webブラウザから入力された値の加算結果を返します。	

Solaris32/64 Linux32/64

項	種類	概要	備考
1	simple	加算とその結果を返します。	スレッドモード用／プロセスモード用

項	種類	概要	備考
2	simple_cpp	クライアントから入力された値の累積値を返します。	スレッドモード用／プロセスモード用
3	java	Webブラウザから入力された値の加算結果を返します。	スレッドモード用
4	simple_so	加算とその結果を返します。	soライブラリ化
5	simple_cpp_so	クライアントから入力された値の累積値を返します。	soライブラリ化

D.2 使用時の注意事項

サンプルプログラムを使用するときに注意すべき点を以下に示します。

- サンプルプログラムを使用するときは、アプリケーション開発環境が必要です。Interstageがサポートするアプリケーション開発環境については、“システム設計ガイド”を参照してください。
- Java言語のCORBAアプリケーションをコンパイル／実行するためには、環境設定を行う必要があります。ORBの指定については、“5.4.2 ORB(Object Request Broker)の指定”を参照してください。クライアントの環境設定については、“5.5 クライアントの環境設定（プレインストール型Javaライブラリ）”または“5.6 クライアントの環境設定（Portable-ORB）”を参照してください。サーバの環境設定については、“5.7 サーバの環境設定”を参照してください。なお、環境設定を行わない場合、サンプルプログラムを正常に動作させることはできません。
- サンプルプログラムは、ネーミングサービスとインタフェースリポジトリを使用します。そのため、事前にネーミングサービスとインタフェースリポジトリのキャッシュサーバが起動されている必要があります。
- IDLコンパイル実行時にIDLファイルの拡張子が大文字である場合、小文字にして実行させます。また、IDLファイル名がすべて大文字である場合も同様です。
- サンプルプログラムは、persistent型として作成されています。ネーミングサービスとインタフェースリポジトリサービスは、インタフェースプログラミングです。
- Solaris32/64
  コンパイラのパスがMakefileに記載されている場合、お使いのコンパイラに合わせて修正してください。
- Solaris32/64
  Oracle Solaris Studio 12.2以降を使用してC++アプリケーションを作成する場合は、CCコマンドに“-features=rvalueref”を指定する必要があります。
- Linux64
  Linux for Intel64（32ビット互換）版Interstage Application Serverでアプリケーションを作成する場合は、gcc/g++コマンドに“-m32 -mtune=i386”オプションを指定する必要があります。
- Solaris32/64 Linux32/64
  プロセスモード、およびCOBOLのサンプルプログラムを使用するときは、環境変数LD_LIBRARY_PATHを設定する必要があります。詳細については、“2.2.1 スレッドモードとプロセスモード”を参照してください。
- “D.4 実行手順（Solaris/Linux）”における実行例は、カレントディレクトリにパスが通っていることを前提としています。
- Windows32/64
  サンプルアプリケーションは、Microsoft(R) Visual C++(R) .NET Standard 2003を利用して開発されています。Microsoft(R) Visual Studio(R) 2005以降を利用してソリューション(.sln)ファイルを開く場合、Microsoft(R) Visual Studio(R)起動時に表示される[Visual Studio 変換ウィザード]でソリューション(.sln)ファイルの変換を行ってください。
- Solaris64
  Sun Studio 12以降を使用してC/C++アプリケーションを作成すると以下の警告が出力されますが、動作に影響は無いため対処は不要です。

```
警告: -xarch=generic64 は非推奨です。64 ビットプログラムの作成には、-m64 を使用してください
```

D.3 実行手順（Windows(R)）

本節では、Windows(R)におけるサンプルプログラムの実行手順について説明します。なお、サンプルプログラムを実行する前に、“D.2 使用時の注意事項”を確認してください。

D.3.1 静的インタフェース

(1) 静的インタフェースのサンプルプログラムの実行例を説明します。

種別	フォルダ(注1)	備考
静的インタフェース／各種データ型（C言語）	%OD_SAMPLES%¥complex¥samplelist.C¥data¥*¥*_s, *_c	(注2)
静的インタフェース／各種データ型（C++言語）	%OD_SAMPLES%¥complex¥samplelist.C++¥data¥*¥*_s, *_c	(注2)
静的インタフェース／各種データ型（Java言語）	%OD_SAMPLES%¥complex¥samplelist.Java¥data¥*	
静的インタフェース／四則演算（C言語）	%OD_SAMPLES%¥CalcSample¥c_s, c_c	(注2)
静的インタフェース／四則演算（C++言語）	%OD_SAMPLES%¥CalcSample¥c++_s, c++_c	(注2)
静的インタフェース／四則演算（Java言語）	%OD_SAMPLES%¥CalcSample¥java	

注1)
ここで使用している環境変数は、以下です（Interstageのインストールフォルダ：デフォルト）。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample

注2)
フォルダ名の最後に_sの付くフォルダがサーバアプリケーションフォルダ、_cの付くフォルダがクライアントアプリケーションフォルダです。それぞれフォルダ名が一致する_s、_cフォルダの組合せで使用します。

D.3.1.1 静的インタフェース／各種データ型（C言語）

＜作成／サーバ＞

サーバアプリケーションの作成手順として、Microsoft(R) Visual Studio(R)を使用した場合の実行例を以下に示します。

1. スケルトンの作成

コマンドプロンプトを起動し、サーバアプリケーションフォルダへ現フォルダを移動します。その後、IDLコンパイルを実行して、スケルトンを作成します。

```
IDLc  simple.idl
```

2. サーバアプリケーションのビルド

エクスプローラなどからソリューション(.sln)ファイルをダブルクリックしてMicrosoft(R) Visual Studio(R)を起動し、以下の手順でビルドします。

1. 起動されたMicrosoft(R) Visual Studio(R)のメニューバーから「ビルド」-「構成マネージャ」を選択して、「アクティブ ソリューション構成」で「Release」を選択します。

Windows64
「アクティブ ソリューション プラットフォーム」で「x64」を選択します。

2. Microsoft(R) Visual Studio(R)のメニューバーから「ビルド」-「ソリューションのビルド」を選択してサーバアプリケーションを作成します。
   ビルド時に警告メッセージが出力されることがありますが、動作上は問題ありません。

＜作成／クライアント＞

クライアントアプリケーションの作成手順として、Microsoft(R) Visual Studio(R)を使用した場合の実行例を以下に示します。

1. スタブの作成

コマンドプロンプトにおいて、上記で作成したサーバアプリケーションに対応したクライアントアプリケーションフォルダへ現フォルダを移動します。その後、IDLコンパイラを実行してスタブを作成します。

```
IDLc  simple.idl
```

2. クライアントアプリケーションのビルド

エクスプローラなどからソリューション(.sln)ファイルをダブルクリックしてMicrosoft(R) Visual Studio(R)を起動します。

1. 起動されたVisual C++(R)のメニューバーから「ビルド」-「構成マネージャ」を選択して、「アクティブ ソリューション構成」で「Release」を選択します。
   Windows64
   「アクティブ ソリューション プラットフォーム」で「x64」を選択します。
2. 起動されたMicrosoft(R) Visual Studio(R)のメニューバーから「ビルド」-「ソリューションのビルド」を選択してクライアントアプリケーションを作成します。
   ビルド時に警告メッセージが出力されることがありますが、動作上は問題ありません。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. サーバアプリケーションの登録

コマンドプロンプトにおいて、サーバアプリケーションフォルダへ現フォルダを移動します。その後、registar.batを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.bat
```

2. サーバアプリケーションの起動

exec-SV.batを実行して、サーバアプリケーションを起動します。

```
start  exec-SV.bat
```

3. クライアントアプリケーションの起動

コマンドプロンプトにおいて、クライアントアプリケーションフォルダへ現フォルダを移動します。その後、exec-CL.batを実行して、クライアントアプリケーションを起動し、出力結果を確認します。入力を求められた場合、値を入力します。

```
exec-CL.bat
```

4. サーバアプリケーションの終了

コマンドプロンプトにおいて、サーバアプリケーションフォルダへ現フォルダを移動します。その後、stop-SV.batを実行して、サーバアプリケーションを停止します。

```
stop-SV.bat
```

5. サーバアプリケーションの登録削除

unregister.batを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.bat
```

ポイント

各種データ型のサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合は、“2. サーバアプリケーションの起動”においてexec-SV.batを使用しないで、以下を実行します。

1. ワークユニット定義の修正

simple.wuの以下の項目をインストール環境に合わせます。

```
[Control Option]
Path: インストールフォルダを修正します。
Current Directory: カレントフォルダとして使用するフォルダ名に修正します。
```

2. ワークユニット定義の登録

isaddwudefコマンドを実行して、ワークユニット定義を登録します。

```
isaddwudef  simple.wu
```

3. ワークユニットの起動

isstartwuコマンドを実行して、ワークユニットを起動します。

```
isstartwu  ODSAMPLE
```

また、“4. サーバアプリケーションの終了”においてstop-SV.batを使用しないで、以下を実行します。

1. ワークユニットの停止

isstopwuコマンドを実行して、ワークユニットを停止します。

```
isstopwu  ODSAMPLE
```

2. ワークユニット定義の削除

isdelwudefコマンドを実行して、ワークユニット定義を削除します。

```
isdelwudef  ODSAMPLE
```

注意

exception のサンプルプログラムを実行すると、以下のメッセージが出力されます。これはサンプルプログラムの出力であり、システムや内部で異常が発生したことによるものではありません。

```
ret = [10]
env_check: invoke ODsample_exceptest_getinfo(20) fails
exception-id = IDL:ODsample/exceptest/NOT_FOUND:1.0
:Detail [20] Count [3]
```

D.3.1.2 静的インタフェース／各種データ型（C++言語）

＜作成／サーバ＞

“D.3.1.1 静的インタフェース／各種データ型（C言語）”の＜作成／サーバ＞を参照してください。
ただし、スケルトンの作成では、以下を実行してください。

```
IDLc -vcpp simple.idl
```

＜作成／クライアント＞

“D.3.1.1 静的インタフェース／各種データ型（C言語）”の＜作成／クライアント＞を参照してください。
ただし、スタブの作成では、以下を実行してください。

```
IDLc -vcpp simple.idl
```

＜実行＞

“D.3.1.1 静的インタフェース／各種データ型（C言語）”の＜実行＞を参照してください。

ポイント

各種データ型のサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合は、“D.3.1.1 静的インタフェース／各種データ型（C言語）”のポイントを参照してください。

注意

exception のサンプルプログラムを実行すると、以下のメッセージが出力されます。これはサンプルプログラムの出力であり、システムや内部で異常が発生したことによるものではありません。

```
10
NOT_FOUND exception raised!
exception-id = IDL:ODsample/exceptest/NOTFOUND:1.0
```

D.3.1.3 静的インタフェース／各種データ型（Java言語）

＜作成＞

サーバアプリケーションの作成手順を以下に示します。

1. 環境変数の設定

コマンドプロンプトを起動し、環境変数CLASSPATHを設定します。

```
set CLASSPATH=.;C:¥Interstage¥ODWIN¥etc¥Class¥ODjava4.jar;%CLASSPATH%
```

注意

環境変数CLASSPATHは、インストール時にODjava4.jarが設定されます。

2. アプリケーションのコンパイル

コマンドプロンプトにおいて、サンプルアプリケーションフォルダへ現フォルダを移動します。その後、make.batを実行してアプリケーションをコンパイルします。

```
make.bat
```

＜実行＞

“D.3.1.1 静的インタフェース／各種データ型(C言語)”の＜実行＞を参照してください。このとき、上記の“環境変数の設定”を行ったコマンドプロンプトを使用してください。

ポイント

各種データ型のサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合は、“D.3.1.1 静的インタフェース／各種データ型(C言語)”のポイントを参照してください。
ただし、simple.wuを以下のように修正してください。

```
[Control Option]
Path: 使用するJDK, JREのバージョンに合わせてインストールフォルダを修正します。
Current Directory: カレントフォルダとして使用するフォルダ名に修正します。
[Application Program]
CLASSPATH for Application: インストールフォルダを修正します。
```

D.3.1.4 静的インタフェース／四則演算(C言語)

“D.3.1.1 静的インタフェース／各種データ型(C言語)”を参照してください。

注意

サンプルプログラムを実行すると、以下のメッセージが出力されます。これはサンプルプログラムの出力であり、システムや内部で異常が発生したことによるものではありません。

```
env_check: ODdemo_calculator_calculate fails
exception-id = IDL:ODdemo/calculator/ZEROPARAM:1.0
```

D.3.1.5 静的インタフェース／四則演算(C++言語)

“D.3.1.2 静的インタフェース／各種データ型(C++言語)”を参照してください。

注意

サンプルプログラムを実行すると、以下のメッセージが出力されます。これはサンプルプログラムの出力であり、システムや内部で異常が発生したことによるものではありません。

```
Exception raised!
exception-id = IDL:ODdemo/calculator/ZEROPARAM:1.0
```

D.3.1.6 静的インタフェース／四則演算（Java言語）

クライアントマシン上では、Internet ExplorerにJBKプラグインをインストールしておいてください。
また、四則演算（Java）では、以下のいずれかをサーバアプリケーションとして使用します。

種別	フォルダ(注)
静的インタフェース／四則演算（C言語）	%OD_SAMPLES%¥CalcSample¥c_s
静的インタフェース／四則演算（C++言語）	%OD_SAMPLES%¥CalcSample¥c++_s

注）ここで使用している環境変数は、以下です（Interstageのインストールフォルダ：デフォルト）。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample

＜作成／サーバ＞

静的インタフェース／四則演算（C言語）のアプリケーションを使用する場合は、“D.3.1.1 静的インタフェース／各種データ型（C言語）”の＜作成／サーバ＞を参照してください。静的インタフェース／四則演算（C++言語）のアプリケーションを使用する場合は、“D.3.1.2 静的インタフェース／各種データ型（C++言語）”の＜作成／サーバ＞を参照してください。

＜作成／クライアント＞

クライアントアプリケーションの作成手順を以下に示します。

1. 環境変数の設定

コマンドプロンプトを起動し、環境変数CLASSPATHを設定します。

```
set CLASSPATH=.;C:¥Interstage¥ODWIN¥etc¥Class¥ODjava4.jar;%CLASSPATH%
```

注意

環境変数CLASSPATHは、インストール時にODjava4.jarが設定されます。

2. スタブの作成（サーバマシン）

サーバマシンにおいてコマンドプロンプトを起動し、サンプルアプリケーションフォルダへ現フォルダを移動します。その後、make.batを実行してJavaクライアントのためのスタブなどを作成します。

```
make.bat
```

3. クライアントアプリケーションのコンパイル（サーバマシン）

apl-compile.batを実行してJavaのソースをコンパイルします。

```
apl-compile.bat
```

4. クライアントアプリケーションの配置（クライアントマシン）

サンプルアプリケーションフォルダ配下にあるすべてのファイルをクライアントマシンの作業用フォルダ配下へコピーします。クライアントマシンへは、ファイル共有およびエクスプローラなどを利用してコピーしてください。

5. 権限の設定（Windows(R)クライアント）

Javaライブラリ(ODjava4.jar)、アプリケーション格納フォルダに権限を設定します。
詳細については、“5.5.2 Javaライブラリに対する権限の設定”を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. サーバアプリケーションの登録（サーバマシン）

サーバマシンのコマンドプロンプトにおいて、サーバアプリケーションフォルダへ現フォルダを移動します。その後、registar.batを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.bat
```

2. サーバアプリケーションの起動（サーバマシン）

exec-SV.batを実行して、サーバアプリケーションを起動します。

```
start  exec-SV.bat
```

3. クライアントアプリケーションの起動（クライアントマシン）

クライアントマシンにおいてブラウザを起動します。エクスプローラから作業フォルダ配下のhtmlファイルをダブルクリックして、ブラウザを起動します。その後、入力画面から数値や文字を入力して“OK”ボタンをクリックし、結果を確認します。

4. サーバアプリケーションの終了（サーバマシン）

サーバマシンのコマンドプロンプトにおいてstop-SV.batを実行して、サーバアプリケーションを停止します。

```
stop-SV.bat
```

5. サーバアプリケーションの登録削除（サーバマシン）

unregister.batを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.bat
```

D.3.2　動的起動インタフェース

(2) 動的起動インタフェースのサンプルプログラムの実行例を説明します。

種別	フォルダ(注1)	備考
動的起動インタフェース／基本(C言語)	%OD_SAMPLES%¥complex¥samplelist.C¥dii¥*_s, *_c	(注2)
動的起動インタフェース／基本(C++言語)	%OD_SAMPLES%¥complex¥samplelist.C++¥dii¥*_s, *_c	(注2)
動的起動インタフェース／四則演算(C言語)	%OD_SAMPLES%¥CalcSample¥c_dii	
動的起動インタフェース／四則演算(C++言語)	%OD_SAMPLES%¥CalcSample¥c++_dii	

注1)
ここで使用している環境変数は、以下です(Interstageのインストールフォルダ:デフォルト)。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample

注2)
フォルダ名の最後に_sの付くフォルダがサーバアプリケーションフォルダ、_cの付くフォルダがクライアントアプリケーションフォルダです。それぞれフォルダ名が一致する_s、_cフォルダの組合せで使用します。

D.3.2.1 動的起動インタフェース／基本(C言語)

"D.3.1.1 静的インタフェース／各種データ型(C言語)"を参照してください。

D.3.2.2 動的起動インタフェース／基本(C++言語)

"D.3.1.2 静的インタフェース／各種データ型(C++言語)"を参照してください。

注意

dyn3のサンプルプログラムでは、クライアントアプリケーションの実行時に以下のエラーメッセージが表示されることがあります。これはクライアントアプリケーションの実行時のタイミングによるもので、動作上は問題ありません。

```
error poll-response
```

D.3.2.3 動的起動インタフェース／四則演算(C言語)

四則演算(C言語)では、サーバアプリケーションとして以下を使用します。

種別	フォルダ(注)
静的インタフェース／四則演算(C言語)	%OD_SAMPLES%¥CalcSample¥c_s

注)
ここで使用している環境変数は、以下です(Interstageのインストールフォルダ:デフォルト)。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample

＜作成／サーバ＞

%OD_SAMPLES%¥CalcSample¥c_sをサーバアプリケーションフォルダとして、その配下のアプリケーションをビルドすることでサーバアプリケーションを作成します。
作成方法については、“D.3.1.1 静的インタフェース／各種データ型（C言語）”の＜作成／サーバ＞を参照してください。

＜作成／クライアント＞

%OD_SAMPLES%¥CalcSample¥c_diiをクライアントアプリケーションディレクトリとして、その配下のアプリケーションをビルドすることでクライアントアプリケーションを作成します。
作成方法については、“D.3.1.1 静的インタフェース／各種データ型（C言語）”の＜作成／クライアント＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. サーバアプリケーションの登録

コマンドプロンプトにおいて、クライアントアプリケーションフォルダへ現フォルダを移動します。その後、registar.batを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.bat
```

2. サーバアプリケーションの起動

コマンドプロンプトにおいて、サーバアプリケーションフォルダへ現フォルダを移動します。その後、exec-SV.batを実行して、サーバアプリケーションを起動します。

```
start  exec-SV.bat
```

3. クライアントアプリケーションの起動

コマンドプロンプトにおいて、クライアントアプリケーションフォルダへ現フォルダを移動します。その後、exec-CL.batを実行して、クライアントアプリケーションを起動し、出力結果を確認します。

```
exec-CL.bat
```

4. サーバアプリケーションの終了

コマンドプロンプトにおいて、サーバアプリケーションフォルダへ現フォルダを移動します。その後、stop-SV.batを実行して、サーバアプリケーションを停止します。

```
stop-SV.bat
```

5. サーバアプリケーションの登録削除

コマンドプロンプトにおいて、クライアントアプリケーションフォルダへ現フォルダを移動します。その後、unregister.batを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.bat
```

D.3.2.4 動的起動インタフェース／四則演算（C++言語）

四則演算（C++言語）では、サーバアプリケーションとして以下を使用します。

種別	フォルダ(注)
静的インタフェース／四則演算（C++言語）	%OD_SAMPLES%¥CalcSample¥c++_s

注)
ここで使用している環境変数は、以下です（Interstageのインストールフォルダ：デフォルト）。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample

＜作成／サーバ＞

%OD_SAMPLES%¥CalcSample¥c++_sをサーバアプリケーションフォルダとして、その配下のアプリケーションをビルドすることでサーバアプリケーションを作成します。
作成方法については、“D.3.1.2 静的インタフェース／各種データ型（C++言語）”の＜作成／サーバ＞を参照してください。

＜作成／クライアント＞

%OD_SAMPLES%¥CalcSample¥c++_diiをクライアントアプリケーションディレクトリとして、その配下のアプリケーションをビルドすることでクライアントアプリケーションを作成します。
作成方法については、“D.3.1.2 静的インタフェース／各種データ型（C++言語）”の＜作成／クライアント＞を参照してください。

＜実行＞

“D.3.2.3 動的起動インタフェース／四則演算（C言語）”の＜実行＞を参照してください。

注意

クライアントアプリケーションの実行時に以下のエラーメッセージが表示されることがあります。これは、クライアントアプリケーションの実行時のタイミングによるもので、動作上は問題ありません。

```
error poll-response
```

D.3.3 ネーミングサービス

(3) ネーミングサービスのサンプルプログラムの実行例を説明します。

種別	フォルダ(注)	備考
ネーミングサービス／listメソッド	%OD_SAMPLES%¥complex¥samplelist.C¥naming¥list	C
	%OD_SAMPLES%¥complex¥samplelist.C++¥naming¥list	C++
ネーミングサービス／bind, resolve, destroyメソッド	%OD_SAMPLES%¥complex¥samplelist.C¥naming¥bind, resolve, destroy	C
	%OD_SAMPLES%¥complex¥samplelist.C++¥naming¥bind, resolve, destroy	C++

注)
ここで使用している環境変数は、以下です（Interstageのインストールフォルダ：デフォルト）。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample

D.3.3.1 ネーミングサービス／listメソッド

＜作成＞

アプリケーションの作成手順として、Microsoft(R) Visual Studio(R)を使用した場合の実行例を以下に示します。

1. アプリケーションのビルド

エクスプローラなどからソリューション(.sln)ファイルをダブルクリックしてMicrosoft(R) Visual Studio(R)を起動し、以下の手順でビルドします。

1. 起動されたMicrosoft(R) Visual Studio(R)のメニューバーから「ビルド」-「構成マネージャ」を選択して、「アクティブ ソリューション構成」で「Release」を選択します。

   「アクティブ ソリューション プラットフォーム」で「x64」を選択します。
2. 起動されたVisual C++(R)のメニューバーから「ビルド」-「ソリューションのビルド」を選択してアプリケーションを作成します。ビルド時に警告メッセージが出力されることがありますが、動作上は問題ありません。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. アプリケーションの登録

コマンドプロンプトを起動し、サンプルアプリケーションフォルダへ現フォルダを移動します。その後、registar.batを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.bat
```

2. アプリケーションの起動

exec-CL.batを実行して、アプリケーションを起動し、結果を確認します。

```
exec-CL.bat
```

3. アプリケーションの登録削除

unregister.batを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.bat
```

D.3.3.2 ネーミングサービス／bind, resolve, destroyメソッド

＜作成＞

“D.3.3.1 ネーミングサービス／listメソッド”の＜作成＞を参照してください。
bind、resolve、destroyのそれぞれについてビルドしてください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. アプリケーションの登録

コマンドプロンプトを起動し、bindフォルダへ現フォルダを移動します。その後、registar.batを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.bat
```

2. bindメソッドアプリケーションの起動

exec-CL.batを実行して、bindメソッドアプリケーションを起動します。アプリケーションが正常に動作した場合は、何も出力されません。

```
exec-CL.bat
```

3. resolveメソッドアプリケーションの起動

コマンドプロンプトにおいて、resolveフォルダへ現フォルダを移動します。その後、exec-CL.batを実行して、resolveメソッドアプリケーションを起動します。アプリケーションが正常に動作した場合は、何も出力されません。

```
exec-CL.bat
```

4. destroyメソッドアプリケーションの起動

コマンドプロンプトにおいて、destroyフォルダへ現フォルダを移動します。その後、exec-CL.batを実行して、destroyメソッドアプリケーションを起動します。アプリケーションが正常に動作した場合は、何も出力されません。

```
exec-CL.bat
```

5. アプリケーションの登録削除

コマンドプロンプトにおいて、bindフォルダへ現フォルダを移動します。その後、unregister.batを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.bat
```

D.3.4　インタフェースリポジトリ

(4) インタフェースリポジトリのサンプルプログラムの実行例を説明します。

種別	フォルダ(注)	備考
インタフェースリポジトリ	%OD_SAMPLES%¥irsample¥*	

注)
ここで使用している環境変数は、以下です(Interstageのインストールフォルダ:デフォルト)。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample

D.3.4.1 インタフェースリポジトリ

“D.3.3.1 ネーミングサービス／listメソッド”を参照してください。

D.3.5 POAインタフェース

(5) POAインタフェースのサンプルプログラムの実行例を説明します。

種別	フォルダ(注1)	備考
Windows32 POAインタフェース(C++言語)	%OD_SAMPLES%¥POA¥C++¥*¥*_s, *_c	(注2)
POAインタフェース(Java言語)	%OD_SAMPLES%¥POA¥java_1_4¥*	

注1)
ここで使用している環境変数は、以下です(Interstageのインストールフォルダ:デフォルト)。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample

注2)
フォルダ名の最後に_sの付くフォルダがサーバアプリケーションフォルダ、_cの付くフォルダがクライアントアプリケーションフォルダです。それぞれフォルダ名が一致する_s、_cフォルダの組合せで使用します。

D.3.5.1 POAインタフェース(C++言語)

Windows32

<作成/サーバ>

"D.3.1.1 静的インタフェース/各種データ型(C言語)"の<作成/サーバ>を参照してください。
ただし、スケルトンの作成では、以下を実行してください。

- tieクラス方式の場合

```
IDLc -vcpp -poa -tie simple.idl
```

- 上記以外の場合

```
IDLc -vcpp -poa simple.idl
```

注意

08_factory_1_instance_tieのサーバアプリケーションを使用する場合は、サーバアプリケーションを作成する前にMicrosoft(R) Visual Studio(R)の[プロジェクト]-[プロパティ]-[構成プロパティ]-[C/C++]-[言語]で「ランタイム型情報を有効にする」に「はい (/GR)」を指定してください。

<作成/クライアント>

"D.3.1.1 静的インタフェース/各種データ型(C言語)"の<作成/クライアント>を参照してください。
ただし、スタブの作成では、以下を実行してください。

```
IDLc -vcpp -poa simple.idl
```

<実行>

"D.3.1.1 静的インタフェース/各種データ型(C言語)"の<実行>を参照してください。

D.3.5.2 POAインタフェース(Java言語)

“D.3.1.3 静的インタフェース／各種データ型(Java言語)”を参照してください。

D.3.6 プロセスバインド

(6) プロセスバインドのサンプルプログラムの実行例を説明します。

種別	フォルダ(注1)	備考
プロセスバインド	%OD_SAMPLES%¥complex¥samplelist.C++¥ssn¥*_s, *_c	(注2)

注1)
ここで使用している環境変数は、以下です(Interstageのインストールフォルダ:デフォルト)。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample

注2)
フォルダ名の最後に_sの付くフォルダがサーバアプリケーションフォルダ、_cの付くフォルダがクライアントアプリケーションフォルダです。それぞれフォルダ名が一致する_s、_cフォルダの組合せで使用します。

D.3.6.1 プロセスバインド

“D.3.1.2 静的インタフェース／各種データ型(C++言語)”を参照してください。

ポイント

プロセスバインドのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合は、“D.3.1.1 静的インタフェース／各種データ型(C言語)”のポイントを参照してください。

注意

ssn_sample2のサンプルプログラムを実行すると、以下のメッセージが出力されます。これはサンプルプログラムの出力であり、システムやCORBAサービスで異常が発生したことによるものではありません。

```
SystemException raised!
exception-id = IDL:CORBA/StExcep/INV_OBJREF:1.0
minor = 0x464a0928
```

D.3.7 各種API

(7) 各種APIのサンプルプログラムの実行例を説明します。

種別	フォルダ(注1)	備考
net_disconnect	%OD_SAMPLES%¥complex¥samplelist.C¥api¥net_disconnect_s, net_disconnect_c %OD_SAMPLES%¥complex¥samplelist.C++¥api¥net_disconnect_s, net_disconnect_c %OD_SAMPLES%¥complex¥samplelist.Java¥api¥net_disconnect	(注2)
set_client_request_timer	%OD_SAMPLES%¥complex¥samplelist.C¥api¥set_client_request_timer_s, set_client_request_timer_c %OD_SAMPLES%¥complex¥samplelist.C++¥api¥set_client_request_timer_s,	(注2)

種別	フォルダ(注1)	備考
	set_client_request_timer_c %OD_SAMPLES%¥complex¥samplelist.Java¥api¥set_client_request_timer	
string_to_object	%OD_SAMPLES%¥complex¥samplelist.C¥api¥string_to_object_s, string_to_object_c %OD_SAMPLES%¥complex¥samplelist.C++¥api¥string_to_object_s, string_to_object_c %OD_SAMPLES%¥complex¥samplelist.Java¥api¥string_to_object	(注2)

注1)
ここで使用している環境変数は、以下です(Interstageのインストールフォルダ:デフォルト)

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample

注2)
フォルダ名の最後に_sの付くフォルダがサーバアプリケーションフォルダ、_cの付くフォルダがクライアントアプリケーションフォルダです。それぞれフォルダ名が一致する_s、_cフォルダの組合せで使用します。

D.3.7.1 net_disconnect

＜作成＞

C言語の場合は、“D.3.1.1 静的インタフェース／各種データ型(C言語)”の＜作成／サーバ＞および＜作成／クライアント＞を参照してください。
C++言語の場合は、“D.3.1.2 静的インタフェース／各種データ型(C++言語)”の＜作成／サーバ＞および＜作成／クライアント＞を参照してください。
Java言語の場合は、“D.3.1.3 静的インタフェース／各種データ型(Java言語)”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。
なお、Java言語の場合は＜作成＞の“環境変数の設定”を行ったコマンドプロンプトを使用してください。

1. SSL環境設定

SSLの環境設定を行います。詳細については、“セキュリティシステム運用ガイド”の“CORBAサービスでSSLを利用する方法”を参照してください。

2. サーバアプリケーションの登録

コマンドプロンプトを起動し、サーバアプリケーションフォルダへ現フォルダを移動します。その後、registar.batを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.bat
```

3. サーバアプリケーションの起動

exec-SV.batを実行して、サーバアプリケーションを起動します。

```
start exec-SV.bat
```

4. コネクション数の確認(疎通前)

odprtcurparamコマンドを使用して、疎通前のコネクション数(IIOP_resp_conのvalueの値)を確認します。

```
odprtcurparam
```

5. クライアントアプリケーションの起動

コマンドプロンプトを起動し、クライアントアプリケーションフォルダへ現フォルダを移動します。その後、exec-CL.batを実行して、クライアントアプリケーションを起動します。

```
exec-CL.bat
```

6. コネクション数の確認(疎通後)

サーバ側のコマンドプロンプトにおいて、odprtcurparamコマンドを使用して、疎通後のコネクション数(IIOP_resp_conのvalueの値)が疎通前より1だけ増加していることを確認します。

```
odprtcurparam
```

7. クライアントアプリケーションの終了

クライアント側のコマンドプロンプトにおいてEnterキーを押下し、クライアントアプリケーションを終了します。

8. サーバアプリケーションの終了

コマンドプロンプトにおいて、サーバアプリケーションフォルダへ現フォルダを移動します。その後、stop-SV.batを実行して、サーバアプリケーションを停止します。

```
stop-SV.bat
```

9. サーバアプリケーションの登録削除

unregister.batを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.bat
```

ポイント

- クライアントアプリケーションにおいて、CORBA_ORB_net_disconnect(C言語)／CORBA::ORB::net_disconnect(C++言語)／com.fujitsu.ObjectDirector.CORBA.ORB.net_disconnect(Java言語)をコメントアウトした上で作成／実行することにより、これらのAPIが通信資源(コネクション数)に与える影響を確認できます。
  該当のAPIをコメントアウトしたアプリケーションでは通信資源が解放されないため、“6. コネクション数の確認(疎通後)”においてコネクション数は疎通前より2だけ増加します。
- net_disconnectのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
  サーバアプリケーションをワークユニット上で動作させる場合、C言語／C++言語については“D.3.1.1 静的インタフェース／各種データ型(C言語)”のポイントを参照してください。Java言語については“D.3.1.3 静的インタフェース／各種データ型(Java言語)”のポイントを参照してください。

注意

サーバ側無通信監視機能／クライアント側無通信監視機能の影響により、“6. コネクション数の確認(疎通後)”において期待する結果が得られないことがあります。この場合、再度アプリケーションを実行してください。

D.3.7.2 set_client_request_timer

＜作成＞

C言語の場合は、“D.3.1.1 静的インタフェース／各種データ型（C言語）”の＜作成／サーバ＞および＜作成／クライアント＞を参照してください。
C++言語の場合は、“D.3.1.2 静的インタフェース／各種データ型（C++言語）”の＜作成／サーバ＞および＜作成／クライアント＞を参照してください。
Java言語の場合は、“D.3.1.3 静的インタフェース／各種データ型（Java言語）”の＜作成＞を参照してください。

＜実行＞

“D.3.1.1 静的インタフェース／各種データ型（C言語）”の＜実行＞を参照してください。
なお、Java言語の場合は＜作成＞の“環境変数の設定”を行ったコマンドプロンプトを使用してください。

ポイント

set_client_request_timerのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合、C言語／C++言語については“D.3.1.1 静的インタフェース／各種データ型（C言語）”のポイントを参照してください。Java言語については“D.3.1.3 静的インタフェース／各種データ型（Java言語）”のポイントを参照してください。

D.3.7.3 string_to_object

＜作成＞

C言語の場合は、“D.3.1.1 静的インタフェース／各種データ型（C言語）”の＜作成／サーバ＞および＜作成／クライアント＞を参照してください。
C++言語の場合は、“D.3.1.2 静的インタフェース／各種データ型（C++言語）”の＜作成／サーバ＞および＜作成／クライアント＞を参照してください。
Java言語の場合は、“D.3.1.3 静的インタフェース／各種データ型（Java言語）”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。
なお、Java言語の場合は＜作成＞の“環境変数の設定”を行ったコマンドプロンプトを使用してください。

1. サーバアプリケーションの登録

コマンドプロンプトを起動し、サーバアプリケーションフォルダへ現フォルダを移動します。その後、registar.batを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDの登録、およびオブジェクトリファレンスのファイル出力を実行します。

```
register.bat
```

2. サーバアプリケーションの起動

exec-SV.batを実行して、サーバアプリケーションを起動します。

```
start  exec-SV.bat
```

3. クライアントアプリケーションの起動

コマンドプロンプトにおいて、クライアントアプリケーションフォルダへ現フォルダを移動します。その後、exec-CL.batを実行して、クライアントアプリケーションを起動します。なお、必要に応じて、Interstageのインストール環境にあわせてexec-CL.batを修正してください。

```
exec-CL.bat
```

4. サーバアプリケーションの終了

コマンドプロンプトにおいて、サーバアプリケーションフォルダへ現フォルダを移動します。その後、stop-SV.batを実行して、サーバアプリケーションを停止します。

```
stop-SV.bat
```

5. サーバアプリケーションの登録削除

unregister.batを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.bat
```

ポイント

string_to_objectのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合、C言語／C++言語については“D.3.1.1 静的インタフェース／各種データ型（C言語）”のポイントを参照してください。Java言語については“D.3.1.3 静的インタフェース／各種データ型（Java言語）”のポイントを参照してください。

D.3.8 Portable-ORB

(8) Portable-ORBのJava連携機能で使用できるサンプルプログラムの実行例を説明します。
Java言語のCORBAアプリケーションをコンパイル／実行するためには、環境設定を行う必要があります。ORBの指定については、“5.4.2 ORB(Object Request Broker)の指定”を参照してください。クライアントの環境設定については、“5.6 クライアントの環境設定（Portable-ORB）”を参照してください。サーバの環境設定については、“5.7 サーバの環境設定”を参照してください。また、アプレットを実行する場合は、事前に“5.21.2 クライアント環境でのアプレット運用時の注意事項”を参照してください。
サンプルプログラムは、以下のフォルダ配下に格納されています。

種別	フォルダ(注)
Portable-ORB／各種データ型	%OD_SAMPLES%¥complex¥samplelist.Java¥data¥*
Portable-ORB／POAインタフェース	%OD_SAMPLES%¥POA¥java_1_4¥*
Portable-ORB／四則演算	%PORB_SAMPLES%¥CalcSample¥java
Portable-ORB／各種API（net_disconnect）	%OD_SAMPLES%¥complex¥samplelist.Java¥API¥net_disconnect
Portable-ORB／各種API（set_client_request_timer）	%OD_SAMPLES%¥complex¥samplelist.Java¥API¥set_client_request_timer
Portable-ORB／各種API（string_to_object）	%OD_SAMPLES%¥complex¥samplelist.Java¥API¥string_to_object

注)
ここで使用している環境変数は、以下です（Interstageのインストールフォルダ：デフォルト）。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample
PORB_SAMPLES	C:¥Interstage¥Porb¥src¥samples

D.3.8.1 Portable-ORB／各種データ型

注意

Portable-ORBでは、context型のデータの受け渡しができないため、context型のサンプルは実行できません。

＜作成＞

“D.3.1.3 静的インタフェース／各種データ型(Java言語)”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. 環境変数(サーバ)の設定

コマンドプロンプトを起動し、環境変数CLASSPATHを設定します。

```
set CLASSPATH=.;C:¥Interstage¥ODWIN¥etc¥Class¥ODjava4.jar;%CLASSPATH%
```

注意

環境変数CLASSPATHは、インストール時にODjava4.jarが設定されます。

2. サーバアプリケーションの登録

コマンドプロンプトにおいて、サンプルアプリケーションフォルダへ現フォルダを移動します。その後、registar.batを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.bat
```

3. サーバアプリケーションの起動

exec-SV.batを実行して、サーバアプリケーションを起動します。

```
exec-SV.bat
```

4. 環境変数(クライアント)の設定

コマンドプロンプトを起動し、Portable-ORBを動作させるための環境変数CLASSPATHを指定します(PORB_HOME:Portable-ORBのインストールパス“C:¥Interstage¥PORB”)。

```
set CLASSPATH=.;%PORB_HOME%¥lib¥ODporb4.jar;%PORB_HOME%¥lib¥CosNaming4.jar;%CLASSPATH%
```

5. クライアントアプリケーションの起動

コマンドプロンプトにおいて、サンプルアプリケーションフォルダへ現フォルダを移動します。その後、exec-CL.batを実行して、クライアントアプリケーションを起動し、出力結果を確認します。

```
exec-CL.bat
```

6. サーバアプリケーションの終了

stop-SV.batを実行して、サーバアプリケーションを停止します。

```
stop-SV.bat
```

7. サーバアプリケーションの登録削除

unregister.batを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.bat
```

D.3.8.2 Portable-ORB／POAインタフェース

“D.3.8.1 Portable-ORB／各種データ型”を参照してください。

D.3.8.3 Portable-ORB／四則演算

ここでは、同一システム(サーバ)にCORBAサービスとPortable-ORBをインストールした上で、同一システム内のWebサーバ上にJavaアプレットを格納し、Windows(R)クライアント上にJavaアプレットとPortable-ORBをダウンロードして使用する場合の実行例を示します。なお、Windows(R)クライアント上にJBKプラグインをインストールしておいてください。

また、ここでは、以下のいずれかをサーバアプリケーションとして使用します。

種別	フォルダ(注)
静的インタフェース／四則演算(C言語)	%OD_SAMPLES%¥CalcSample¥c_s
静的インタフェース／四則演算(C++言語)	%OD_SAMPLES%¥CalcSample¥c++_s

注)
ここで使用している環境変数は、以下です(Interstageのインストールフォルダ:デフォルト)。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample

＜作成／サーバ＞

D.3.1.4 静的インタフェース／四則演算(C言語)のアプリケーションを使用する場合は、“D.3.1.1 静的インタフェース／各種データ型(C言語)”の＜作成／サーバ＞を参照してください。D.3.1.5 静的インタフェース／四則演算(C++言語)のアプリケーションを使用する場合は、“D.3.1.2 静的インタフェース／各種データ型(C++言語)”の＜作成／サーバ＞を参照してください。

＜作成／クライアント＞

クライアントアプリケーションの作成手順を以下に示します。
クライアントアプリケーションは、サーバ上で作成します。

1. 環境変数の設定

コマンドプロンプトを起動し、環境変数CLASSPATHを設定します（PORB_HOME：Portable-ORBのインストールパス“C:¥Interstage¥PORB”）。

```
set CLASSPATH=.;%PORB_HOME%¥lib¥ODporb4.jar;%PORB_HOME%¥lib¥CosNaming4.jar;%CLASSPATH%
```

2. スタブの作成

コマンドプロンプトにおいて、サンプルアプリケーションフォルダへ現フォルダを移動します。その後、make.batを実行して、スタブを作成します。

```
make.bat
```

3. クライアントアプリケーションの作成

apl-compile.batを実行して、クライアントアプリケーションのjarアーカイブファイル(client2.jar)を作成します。

```
apl-compile.bat
```

4. デジタル署名の実施

作成したクライアントアプリケーションのjarアーカイブファイルと使用するPortable-ORBのライブラリにデジタル署名を実施し、クライアントアプリケーションを実行するWindows(R)クライアントでデジタル署名を有効とするための環境設定を行います。
デジタル署名の詳細については、“5.8.2 ダウンロードオブジェクトと署名対象オブジェクト”および“5.8.3 デジタル署名手順”を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. クライアントアプリケーションのコピー（サーバマシン）

作成したクライアントアプリケーションをWebサーバのドキュメントルート配下に存在する任意のフォルダにコピーします（APPLETDIR:アプレットを格納するフォルダ）。

```
mkdir %APPLETDIR%
copy %PORB_SAMPLES%¥CalcSample¥java¥client2.jar %APPLETDIR%
copy %PORB_SAMPLES%¥CalcSample¥java¥CalcSample4.html %APPLETDIR%
copy %PORB_SAMPLES%¥CalcSample¥java¥CalcSample4.js %APPLETDIR%
```

2. Portable-ORBファイルのコピー（サーバマシン）

Portable-ORBをダウンロード可能とするために、必要なファイルをコピーします。
なお、アプレットと同一フォルダにコピーしないで使用する場合は、ダウンロード可能なようにHTML内のARCHIVE指定の変更、および動作環境ファイルの格納位置をPORB_HOMEによって指定します。
PORB_HOMEの指定方法については、“5.6.5 Portable-ORB動作環境ファイルの指定”を参照してください。

```
cd %APPLETDIR%
copy %PORB_HOME%¥lib¥ODporb4_plugin.jar  .
copy %PORB_HOME%¥lib¥CosNaming4_plugin.jar  .
copy %PORB_HOME%¥lib¥InterfaceRep4_plugin.jar  .
mkdir etc
copy %PORB_HOME%¥etc¥* etc
```

3. Portable-ORBの動作環境ファイルの設定（サーバマシン）

Portable-ORBの動作環境ファイルを設定します。porbeditenvコマンドを使用して、%APPLETDIR%¥etcに格納される動作環境ファイルのホスト情報を設定します。

4. サーバアプリケーションの登録（サーバマシン）

コマンドプロンプトでサンプルアプリケーションフォルダへ現フォルダを移動します。その後、registar.batを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.bat
```

5. サーバアプリケーションの起動（サーバマシン）

exec-SV.batを実行して、サーバアプリケーションを起動します。

```
start exec-SV.bat
```

6. クライアントの実行（クライアントマシン）

Portable-ORBのORBクラスが動作するようにORBクラスを指定します。指定方法については、“5.4.2 ORB(Object Request Broker)の指定”を参照してください。
その後、ブラウザでCalcSample4.htmlが格納されるURLを指定し、アプレットを起動します（CalcSample4.htmlをダブルクリックした場合（fileプロトコル使用）は、セキュリティチェックの対象となり、正常に動作しないことがあります）。
入力画面から数値や文字を入力して“OK”をクリックし、結果を確認します。

7. サーバアプリケーションの終了（サーバマシン）

stop-SV.batを実行して、サーバアプリケーションを停止します。

```
stop-SV.bat
```

8. サーバアプリケーションの登録削除（サーバマシン）

unregister.batを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.bat
```

D.3.8.4 Portable-ORB／各種API（net_disconnect）

＜作成＞

“D.3.1.3 静的インタフェース／各種データ型（Java言語）”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. 環境変数（サーバ）の設定

コマンドプロンプトを起動し、環境変数CLASSPATHを設定します。

```
set CLASSPATH=.;C:¥Interstage¥ODWIN¥etc¥Class¥ODjava4.jar;%CLASSPATH%
```

注意

環境変数CLASSPATHは、インストール時にODjava4.jarが設定されます。

2. SSL環境(サーバ)設定

サーバにおいてSSLの環境設定を行います。SSLの環境設定については、“セキュリティシステム運用ガイド”の“CORBAサービスでSSLを利用する方法”を参照してください。

3. SSL環境(クライアント)設定

クライアントにおいてSSLの環境設定を行います。SSLの環境設定については、“セキュリティシステム運用ガイド”の“Portable-ORBでSSLを利用する方法”を参照してください。

4. サーバアプリケーションの登録

コマンドプロンプトにおいて、サンプルアプリケーションフォルダへ現フォルダを移動します。その後、registar.batを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.bat
```

5. サーバアプリケーションの起動

exec-SV.batを実行して、サーバアプリケーションを起動します。

```
start exec-SV.bat
```

6. コネクション数の確認(疎通前)

odprtcurparamコマンドを使用して、疎通前のコネクション数(IIOP_resp_conのvalueの値)を確認します。

```
odprtcurparam
```

7. 環境変数(クライアント)の設定

コマンドプロンプトを起動し、Portable-ORBを動作させるための環境変数CLASSPATHを指定します(PORB_HOME:Portable-ORBのインストールパス“C:¥Interstage¥PORB”)。

```
set CLASSPATH=.;%PORB_HOME%¥lib¥ODporb4.jar;%PORB_HOME%¥lib¥CosNaming4.jar;%CLASSPATH%
```

8. クライアントアプリケーションの起動

コマンドプロンプトにおいて、クライアントアプリケーションフォルダへ現フォルダを移動します。その後、exec-CL.batを実行して、クライアントアプリケーションを起動します。

```
exec-CL.bat
```

9. コネクション数の確認(疎通後)

サーバ側のコマンドプロンプトにおいて、odprtcurparamコマンドを使用して、疎通後のコネクション数(IIOP_resp_conのvalueの値)が疎通前より1だけ増加していることを確認します。

```
odprtcurparam
```

10. クライアントアプリケーションの終了

クライアント側のコマンドプロンプトにおいてEnterキーを押下し、クライアントアプリケーションを終了します。

11. サーバアプリケーションの終了

コマンドプロンプトにおいて、サーバアプリケーションフォルダへ現フォルダを移動します。その後、stop-SV.batを実行して、サーバアプリケーションを停止します。

```
stop-SV.bat
```

12. サーバアプリケーションの登録削除

unregister.batを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.bat
```

ポイント

- クライアントアプリケーションにおいて、com.fujitsu.ObjectDirector.CORBA.ORB.net_disconnectをコメントアウトした上で作成／実行することにより、APIが通信資源（コネクション数）に与える影響を確認できます。
  該当のAPIをコメントアウトしたアプリケーションでは通信資源が解放されないため、“9. コネクション数の確認（疎通後）”においてコネクション数は疎通前より2だけ増加します。
- net_disconnectのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
  サーバアプリケーションをワークユニット上で動作させる場合は、“D.3.1.3 静的インタフェース／各種データ型（Java言語）”のポイントを参照してください。

注意

サーバ側無通信監視機能／クライアント側無通信監視機能の影響により、“9. コネクション数の確認（疎通後）”において期待する結果が得られないことがあります。この場合、再度アプリケーションを実行してください。

D.3.8.5 Portable-ORB／各種API（set_client_request_timer）

“D.3.8.1 Portable-ORB／各種データ型”を参照してください。

ポイント

set_client_request_timerのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合は、“D.3.1.3 静的インタフェース／各種データ型（Java言語）”のポイントを参照してください。

D.3.8.6 Portable-ORB／各種API（string_to_object）

＜作成＞

“D.3.1.3 静的インタフェース／各種データ型（Java言語）”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. 環境変数（サーバ）の設定

コマンドプロンプトを起動し、環境変数CLASSPATHを設定します。

```
set CLASSPATH=.;C:¥Interstage¥ODWIN¥etc¥Class¥ODjava4.jar;%CLASSPATH%
```

環境変数CLASSPATHは、インストール時にODjava4.jarが設定されます。

2. サーバアプリケーションの登録

コマンドプロンプトを起動し、サーバアプリケーションフォルダへ現フォルダを移動します。その後、registar.batを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDの登録、およびオブジェクトリファレンスのファイル出力を実行します。

```
register.bat
```

3. サーバアプリケーションの起動

exec-SV.batを実行して、サーバアプリケーションを起動します。

```
start  exec-SV.bat
```

4. 環境変数（クライアント）の設定

コマンドプロンプトを起動し、Portable-ORBを動作させるための環境変数CLASSPATHを指定します（PORB_HOME：Portable-ORBのインストールパス“C:¥Interstage¥PORB”）。

```
set CLASSPATH=.;%PORB_HOME%¥lib¥ODporb4.jar;%PORB_HOME%¥lib¥CosNaming4.jar;%CLASSPATH%
```

5. クライアントアプリケーションの起動

コマンドプロンプトにおいて、クライアントアプリケーションフォルダへ現フォルダを移動します。その後、exec-CL.batを実行して、クライアントアプリケーションを起動します。なお、必要に応じて、Interstageのインストール環境にあわせてexec-CL.batを修正してください。

```
exec-CL.bat
```

6. サーバアプリケーションの終了

コマンドプロンプトにおいて、サーバアプリケーションフォルダへ現フォルダを移動します。その後、stop-SV.batを実行して、サーバアプリケーションを停止します。

```
stop-SV.bat
```

7. サーバアプリケーションの登録削除

unregister.batを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.bat
```

ポイント

string_to_objectのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合は、“D.3.1.3 静的インタフェース／各種データ型（Java言語）”のポイントを参照してください。

D.3.9 その他

Windows32

(9) その他のサンプルプログラムの実行例を説明します。

種別	フォルダ(注1)	備考
その他（C言語）	%OD_SAMPLES%¥simple_s, simple_c	(注2)
その他（C++言語）	%OD_SAMPLES%¥simple_cpp_s, simple_cpp_c	(注2)
その他（Java言語）	%OD_SAMPLES%¥java¥Server, Client	

注1)
ここで使用している環境変数は、以下です（Interstageのインストールフォルダ：デフォルト）。

環境変数	値
OD_SAMPLES	C:¥Interstage¥ODWIN¥src¥sample

注2)
フォルダ名の最後に_sの付くフォルダがサーバアプリケーションフォルダ、_cの付くフォルダがクライアントアプリケーションフォルダです。それぞれフォルダ名が一致する_s、_cフォルダの組合せで使用します。

D.3.9.1 その他（C言語）

“D.3.1.1 静的インタフェース／各種データ型（C言語）”を参照してください。

D.3.9.2 その他（C++言語）

“D.3.1.2 静的インタフェース／各種データ型（C++言語）”を参照してください。

D.3.9.3 その他（Java言語）

クライアントマシン上では、Internet ExplorerにJBKプラグインをインストールしておいてください。

＜作成／サーバ＞

サーバアプリケーションの作成手順として、Microsoft(R) Visual Studio(R)を使用した場合の実行例を以下に示します。

1. スケルトンの作成

コマンドプロンプトを起動し、サーバアプリケーションフォルダへ現フォルダを移動します。その後、IDLコンパイルを実行して、スケルトンを作成します。

```
IDLc sample.idl
```

2. サーバアプリケーションのビルド

エクスプローラなどからソリューション(.sln)ファイルをダブルクリックしてMicrosoft(R) Visual Studio(R)を起動し、以下の手順でビルドします。

1. 起動されたMicrosoft(R) Visual Studio(R)のメニューバーから「ビルド」-「構成マネージャ」を選択して、「アクティブ ソリューション構成」で「Release」を選択します。
2. Visual C++(R)のメニューバーから「ビルド」-「ソリューションのビルド」を選択してサーバアプリケーションを作成します。
   ビルド時に警告メッセージが出力されることがありますが、動作上は問題ありません。

＜作成／クライアント＞

Clientをクライアントアプリケーションフォルダとして、その配下のアプリケーションをコンパイルすることでクライアントアプリケーションを作成します。
クライアントアプリケーションの作成手順を以下に示します。

1. スタブの作成（サーバマシン）

コマンドプロンプトを起動し、クライアントアプリケーションフォルダへ現フォルダを移動します。その後、make.batを実行してJavaクライアントのためのスタブなどを作成します。

```
make.bat
```

2. クライアントアプリケーションのコンパイル（サーバマシン）

apl-compile.batを実行してJavaのソースをコンパイルします。

```
apl-compile.bat
```

3. クライアントアプリケーションの配置（クライアントマシン）

サーバマシンのクライアントアプリケーションフォルダ配下にあるすべてのファイルをクライアントマシンの作業用フォルダ配下へコピーします。クライアントマシンへは、ファイル共有およびエクスプローラなどを利用してコピーしてください。

4. 権限の設定（Windows(R)クライアント）

Javaライブラリ(ODjava4.jar)、アプリケーション格納フォルダに権限を設定します。
詳細については、“5.5.2 Javaライブラリに対する権限の設定”を参照してください。

＜実行＞

“D.3.1.6 静的インタフェース／四則演算（Java言語）”の＜実行＞を参照してください。

D.4 実行手順（Solaris/Linux）

本節では、Solaris/Linuxにおけるサンプルプログラムの実行手順について説明します。なお、サンプルプログラムを実行する前に、“D.2 使用時の注意事項”を確認してください。

D.4.1 静的インタフェース

(1) 静的インタフェースのサンプルプログラムの実行例を説明します。

種別	ディレクトリ(注)	備考
静的インタフェース／各種データ型（C言語）	$OD_SAMPLES/complex/samplelist.C/data/*	スレッドモード用
	$OD_SAMPLES/complex/samplelist.C_nt/data/*	プロセスモード用
静的インタフェース／各種データ型（C++言語）	$OD_SAMPLES/complex/samplelist.C++/data/*	スレッドモード用
	$OD_SAMPLES/complex/samplelist.C++_nt/data/*	プロセスモード用
静的インタフェース／各種データ型（Java言語）	$OD_SAMPLES/complex/samplelist.Java/data/*	
静的インタフェース／各種データ型（COBOL）	$OD_SAMPLES/complex/samplelist.COBOL/data/*	
静的インタフェース／各種データ型（Windows(R)クライアント）	$OD_SAMPLES/complex/samplelist.C/data_win/*	
静的インタフェース／四則演算（C言語）	$OD_SAMPLES/CalcSample/c	
静的インタフェース／四則演算（C++言語）	$OD_SAMPLES/CalcSample/c++	
静的インタフェース／四則演算（Java言語）	$OD_SAMPLES/CalcSample/java	
静的インタフェース／四則演算（COBOL）	$OD_SAMPLES/CalcSample/COBOL	

注)
ここで使用している環境変数は、以下です（Interstageのインストールパス：デフォルト）。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

D.4.1.1 静的インタフェース／各種データ型（C言語）

＜作成＞

アプリケーションの作成手順を以下に示します。

1. アプリケーションのコンパイル

サンプルアプリケーションディレクトリへ移動し、makeコマンドを実行してサーバ、およびクライアントアプリケーションを作成します。
このとき、警告メッセージが出力されることがありますが、動作上は問題ありません。

```
make
```

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. サーバアプリケーションの登録

registar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.sh
```

2. サーバアプリケーションの起動

exec-SV.shを実行して、サーバアプリケーションを起動します。

```
exec-SV.sh &
```

3. クライアントアプリケーションの起動

exec-CL.shを実行してクライアントアプリケーションを起動し、出力結果を確認します。入力を求められた場合、値を入力します。

```
exec-CL.sh
```

4. サーバアプリケーションの終了

stop-SV.shを実行して、サーバアプリケーションを停止します。

```
stop-SV.sh
```

5. サーバアプリケーションの登録削除

unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

ポイント

各種データ型のサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合、“2. サーバアプリケーションの起動”においてexec-SV.shを使用しないで、以下を実行します。

1. ワークユニット定義の登録

isaddwudefコマンドを実行して、ワークユニット定義を登録します。

```
isaddwudef simple.wu
```

2. ワークユニットの起動

isstartwuコマンドを実行して、ワークユニットを起動します。

```
isstartwu ODSAMPLE
```

また、“4. サーバアプリケーションの終了”においてstop-SV.shを使用しないで、以下を実行します。

1. ワークユニットの停止

isstopwuコマンドを実行して、ワークユニットを停止します。

```
isstopwu ODSAMPLE
```

2. ワークユニット定義の削除

isdelwudefコマンドを実行して、ワークユニット定義を削除します。

```
isdelwudef ODSAMPLE
```

注意

exception のサンプルプログラムを実行すると、以下のメッセージが出力されます。これはサンプルプログラムの出力であり、システムや内部で異常が発生したことによるものではありません。

```
ret = [10]
env_check: invoke ODsample_exceptest_getinfo(20) fails
exception-id = IDL:ODsample/exceptest/NOT_FOUND:1.0
:Detail [20] Count [3]
```

D.4.1.2 静的インタフェース／各種データ型（C++言語）

“D.4.1.1 静的インタフェース／各種データ型（C言語）”を参照してください。

ポイント

各種データ型のサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合は、“D.4.1.1 静的インタフェース／各種データ型（C言語）”のポイントを参照してください。

注意

exception のサンプルプログラムを実行すると、以下のメッセージが出力されます。これはサンプルプログラムの出力であり、システムや内部で異常が発生したことによるものではありません。

```
10
NOT_FOUND exception raised!
exception-id = IDL:ODsample/exceptest/NOTFOUND:1.0
```

D.4.1.3 静的インタフェース／各種データ型（Java言語）

＜作成＞

1. 環境変数の設定

環境変数CLASSPATH、LD_LIBRARY_PATH、PATHを設定します。環境変数PATHには、使用するJDK/JREのディレクトリを指定します（OD_HOME：CORBAサービスのインストールパス）。

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export CLASSPATH

LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH

PATH=<JDK/JREのインストールパス>/bin:$PATH
export PATH
```

2. アプリケーションのコンパイル

サンプルアプリケーションディレクトリへ移動し、makeコマンドを実行してサーバ／クライアントアプリケーションを作成します。

```
make
```

＜実行＞

“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜実行＞を参照してください。このとき、上記の“環境変数の設定”を実施した端末で実施してください。

ポイント

各種データ型のサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合は、“D.4.1.1 静的インタフェース／各種データ型（C言語）”のポイントを参照してください。
ただし、simple.wuを動作環境に合わせて修正してください。

```
[Control Option]
Path：使用するJDK/JREのディレクトリに修正します。
```

D.4.1.4 静的インタフェース／各種データ型（COBOL）

＜作成＞

“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. サーバアプリケーションの登録

registarを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register
```

2. サーバアプリケーションの起動

サーバアプリケーションを起動します。
なお、サーバアプリケーション名は、XXX_s(XXX：3桁の数字)／simple_sです。

```
014_s &
```

3. クライアントアプリケーションの起動

クライアントアプリケーションを起動し、出力結果を確認します。
なお、クライアントアプリケーション名は、XXX_c(XXX：3桁の数字)／simple_cです

```
014_c
```

4. サーバアプリケーションの終了

odcntlqueコマンドを実行して、サーバアプリケーションを停止します(OD_HOME:CORBAサービスのインストールパス)。

```
$OD_HOME/bin/odcntlque -s IDL:INTF_A:1.0
```

5. サーバアプリケーションの登録削除

unregisterを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister
```

ポイント

各種データ型のサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合、“1. サーバアプリケーションの登録”において、インタフェース定義のパスを絶対パスに修正する必要があります。register内でOD_impl_instコマンドによりインプリメンテーションリポジトリ情報が登録されていますが、OD_impl_instコマンドのaxオプションの引数に指定されている定義ファイルをエディタで開き、インタフェース定義に指定されているライブラリのパスを絶対パスに変更してください。
また、“2. サーバアプリケーションの起動”においてサーバアプリケーション名を指定して起動するのではなく、以下を実行します。

1. ワークユニット定義の登録

isaddwudefコマンドを実行して、ワークユニット定義を登録します。

- 日本語版のCOBOLの場合

```
isaddwudef  simple.wu
```

- 英語版のCOBOLの場合

```
isaddwudef  simple_e.wu
```

2. ワークユニットの起動

isstartwuコマンドを実行して、ワークユニットを起動します。

```
isstartwu  ODSAMPLE
```

“4. サーバアプリケーションの終了”においてodcntlqueコマンドを使用しないで、以下を実行します。

1. ワークユニットの停止

isstopwuコマンドを実行して、ワークユニットを停止します。

```
isstopwu  ODSAMPLE
```

2. ワークユニット定義の削除

isdelwudefコマンドを実行して、ワークユニット定義を削除します。

```
isdelwudef  ODSAMPLE
```

D.4.1.5 静的インタフェース／各種データ型（Windows(R)クライアント）

各種データ型（Windows(R)クライアント）では、サーバアプリケーションとして以下を使用します。

種別	ディレクトリ(注)
静的インタフェース／各種データ型（C言語）	$OD_SAMPLES/complex/samplelist.C/data/*

注)
ここで使用している環境変数は、以下です（Interstageのインストールパス：デフォルト）。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

クライアントアプリケーションディレクトリと同一名のディレクトリをサーバアプリケーションディレクトリとして使用します。たとえば、$OD_SAMPLES/complex/samplelist.C/data_win/anyをクライアントアプリケーションディレクトリとする場合、サーバアプリケーションディレクトリは$OD_SAMPLES/complex/samplelist.C/data/anyになります。

＜作成／サーバ＞

“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜作成＞を参照してください。

＜作成／クライアント＞

クライアントアプリケーションの作成手順を以下に示します。

1. スタブの作成（サーバ）

IDL コンパイラを実行してスタブを作成します。

```
IDLc simple.idl
```

2. クライアントアプリケーションファイルの転送（Windows(R)クライアント）

クライアントアプリケーションディレクトリ配下にあるすべてのファイルをWindows(R)クライアントへ転送します。
ftpを使用した場合の例を以下に示します。なお、Windows(R)クライアントへファイルを転送するときは、以下のように指定します。

- “プロジェクトワークスペースファイル”：バイナリで転送
- “その他のファイル”：ネットワークASCIIで転送

```
ftp ＜Solaris/Linuxサーバ名＞
:
get simple.h
:
mget *.mak
bin
mget *.mdp
```

3. クライアントアプリケーションのビルド（Windows(R)クライアント）

エクスプローラなどから、転送したプロジェクトワークスペース(.mdp)ファイルをダブルクリックしてMicrosoft(R) Visual Studio(R)を起動します。このとき、「変換しますか？」という問い合わせがありますので、「はい」を選択してプロジェクト形式を変換します。

その後、「ビルド」-「ソリューションのビルド」を選択してクライアントアプリケーションを作成します。ビルド時に警告メッセージが出力されることがありますが、動作上は問題ありません。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. サーバアプリケーションの登録（サーバ）

サーバにおいてregistar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.sh
```

2. サーバアプリケーションの起動（サーバ）

exec-SV.shを実行して、サーバアプリケーションを起動します。

```
exec-SV.sh &
```

3. クライアントアプリケーションの起動（Windows(R)クライアント）

Windows(R)クライアントにおいて、Microsoft(R) Visual Studio(R)のメニューバーから「デバッグ」-「デバッグなしで開始」を選択します。クライアントアプリケーションが起動されるので、結果を確認します。

4. サーバアプリケーションの終了（サーバ）

サーバにおいて、stop-SV.shを実行して、サーバアプリケーションを停止します。

```
stop-SV.sh
```

5. サーバアプリケーションの登録削除（サーバ）

unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

注意

exception のサンプルプログラムを実行すると、以下のメッセージが出力されます。これはサンプルプログラムの出力であり、システムや内部で異常が発生したことによるものではありません。

```
id = [IDL:ODsample/exceptest/NOT_FOUND:1.0]:Detail [20] Count [3]
IDL:ODsample/exceptest/NOT_FOUND:1.0
```

D.4.1.6 静的インタフェース／四則演算（C言語）

“D.4.1.1 静的インタフェース／各種データ型（C言語）”を参照してください。

注意

サンプルプログラムを実行すると、以下のメッセージが出力されます。これはサンプルプログラムの出力であり、システムや内部で異常が発生したことによるものではありません。

```
env_check: ODdemo_calculator_calculate fails
exception-id = IDL:ODdemo/calculator/ZEROPARAM:1.0
```

D.4.1.7 静的インタフェース／四則演算（C++言語）

“D.4.1.1 静的インタフェース／各種データ型（C言語）”を参照してください。

注意

サンプルプログラムを実行すると、以下のメッセージが出力されます。これはサンプルプログラムの出力であり、システムや内部で異常が発生したことによるものではありません。

```
Exception raised!
exception-id = IDL:ODdemo/calculator/ZEROPARAM:1.0
```

D.4.1.8 静的インタフェース／四則演算（Java言語）

Windows(R)クライアントマシン上には、Internet ExplorerにJBKプラグインをインストールしておいてください。

また、四則演算（Java）では、以下のいずれかをサーバアプリケーションとして使用します。

種別	ディレクトリ(注)
静的インタフェース／四則演算（C言語）	$OD_SAMPLES/CalcSample/c
静的インタフェース／四則演算（C++言語）	$OD_SAMPLES/CalcSample/c++

注)
ここで使用している環境変数は、以下です（Interstageのインストールパス：デフォルト）。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

＜作成／サーバ＞

“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜作成＞を参照してください。

＜作成／クライアント＞

クライアントアプリケーションの作成手順を以下に示します。

1. 環境変数の設定

環境変数CLASSPATH、LD_LIBRARY_PATH、PATHを設定します。環境変数PATHには、使用するJDK/JREのディレクトリを指定します（OD_HOME：CORBAサービスのインストールパス）。

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export  CLASSPATH

LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH

PATH=<JDK/JREのインストールパス>/bin:$PATH
export PATH
```

2. スタブの作成(サーバマシン)

サンプルアプリケーションディレクトリへ移動します。その後、makeコマンドを実行してJavaクライアントのためのスタブなどを作成します。

```
make
```

3. クライアントアプリケーションの配置(Windows(R)クライアントマシン)

サンプルアプリケーションディレクトリ配下にあるすべてのファイルをWindows(R)クライアントへ転送します。このとき、Windows(R)クライアントのサブディレクトリ配下の構成、およびディレクトリ名(大文字/小文字)は、Solaris/Linuxサーバと同一にする必要があります。ftpを使用した場合の例を以下に示します。Windows(R)クライアントの格納先のフォルダ配下にODdemoフォルダを、ODdemoフォルダ配下にはcalculatorPackageフォルダをあらかじめ作成しておいてください。なお、Windows(R)クライアントへファイルを転送するときは、すべてのファイルについてネットワークASCIIで転送するように指定します。

```
ftp ＜Solaris/Linuxサーバ名＞
:
cd $SAMPLES/CalcSample/java
mget *
lcd ODdemo
cd ODdemo
mget *
lcd calculatorPackage
cd calculatorPackage
mget *
```

4. 権限の設定(Windows(R)クライアント)

Javaライブラリ(ODjava4.jar)、アプリケーション格納フォルダに権限を設定します。
詳細については、“5.5.2 Javaライブラリに対する権限の設定”を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. サーバアプリケーションの登録(サーバマシン)

サーバにおいて、registar.shを実行してサンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.sh
```

2. サーバアプリケーションの起動(サーバマシン)

exec-SV.shを実行して、サーバアプリケーションを起動します。

```
exec-SV.sh &
```

3. クライアントアプリケーションの起動(Windows(R)クライアントマシン)

Windows(R)クライアントにおいて、ブラウザを起動します。エクスプローラから作業フォルダ配下のhtmlファイルをダブルクリックして、ブラウザを起動します。その後、入力画面から数値や文字を入力して“OK”ボタンをクリックし、結果を確認します。

4. サーバアプリケーションの終了(サーバマシン)

サーバにおいて、stop-SV.shを実行して、サーバアプリケーションを停止します。

```
stop-SV.sh
```

5. サーバアプリケーションの登録削除(サーバマシン)

unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

D.4.1.9 静的インタフェース／四則演算(COBOL)

“D.4.1.4 静的インタフェース／各種データ型(COBOL)”を参照してください。

注意

サンプルプログラムを実行すると、以下のメッセージが出力されます。これはサンプルプログラムの出力であり、システムや内部で異常が発生したことによるものではありません。

```
USER-EXCEPTION: ODdemo_calculator_calculate
ID: IDL:ODdemo/calculator/ZEROPARAM:1.0
```

D.4.2 動的起動インタフェース

(2) 動的起動インタフェースのサンプルプログラムの実行例を説明します。

種別	ディレクトリ(注)	備考
動的起動インタフェース／基本(C言語)	$OD_SAMPLES/complex/samplelist.C/dii/*	スレッドモード用
	$OD_SAMPLES/complex/samplelist.C_nt/dii/*	プロセスモード用
動的起動インタフェース／基本(C++言語)	$OD_SAMPLES/complex/samplelist.C++/dii/*	スレッドモード用
	$OD_SAMPLES/ complex/samplelist.C++_nt/dii/*	プロセスモード用
動的起動インタフェース／四則演算(C言語)	$OD_SAMPLES/CalcSample/c_dii	
動的起動インタフェース／四則演算(C++言語)	$OD_SAMPLES/CalcSample/c++_dii	

注)
ここで使用している環境変数は、以下です(Interstageのインストールパス:デフォルト)。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

D.4.2.1 動的起動インタフェース／基本（C言語）

“D.4.1.1 静的インタフェース／各種データ型（C言語）”を参照してください。

D.4.2.2 動的起動インタフェース／基本（C++言語）

“D.4.1.1 静的インタフェース／各種データ型（C言語）”を参照してください。

注意

dyn3のサンプルプログラムでは、クライアントアプリケーションの実行時に以下のエラーメッセージが表示されることがあります。これはクライアントアプリケーションの実行時のタイミングによるもので、動作上は問題ありません。

```
error poll-response
```

D.4.2.3 動的起動インタフェース／四則演算（C言語）

四則演算（C言語）では、以下をサーバアプリケーションとして使用します。

種別	ディレクトリ(注)
静的インタフェース／四則演算（C言語）	$OD_SAMPLES/CalcSample/c

注)
ここで使用している環境変数は、以下です（Interstageのインストールパス：デフォルト）。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

＜作成／サーバ＞

$OD_SAMPLES/CalcSample/cをサーバアプリケーションディレクトリとして、その配下のアプリケーションをコンパイルすることでサーバアプリケーションを作成します。
作成方法については、“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜作成＞を参照してください。

＜作成／クライアント＞

$OD_SAMPLES/CalcSample/c_diiをクライアントアプリケーションディレクトリとして、その配下のアプリケーションをコンパイルすることでクライアントアプリケーションを作成します。
作成方法については、“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. サーバアプリケーションの登録

クライアントアプリケーションディレクトリに移動します。その後、registar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.sh
```

2. サーバアプリケーションの起動

サーバアプリケーションディレクトリに移動します。その後、exec-SV.shを実行して、サーバアプリケーションを起動します。

```
exec-SV.sh &
```

3. クライアントアプリケーションの起動

クライアントアプリケーションディレクトリに移動します。その後、exec-CL.shを実行してクライアントアプリケーションを起動し、出力結果を確認します。

```
exec-CL.sh
```

4. サーバアプリケーションの終了

サーバアプリケーションディレクトリに移動します。その後、stop-SV.shを実行して、サーバアプリケーションを停止します。

```
stop-SV.sh
```

5. サーバアプリケーションの登録削除

クライアントアプリケーションディレクトリに移動します。その後、unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

D.4.2.4 動的起動インタフェース／四則演算（C++言語）

四則演算（C++言語）では、以下をサーバアプリケーションとして使用します。

種別	ディレクトリ(注)
D.4.1.7 静的インタフェース／四則演算（C++言語）	$OD_SAMPLES/CalcSample/c++

注)
ここで使用している環境変数は、以下です（Interstageのインストールパス：デフォルト）。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

＜作成／サーバ＞

$OD_SAMPLES/CalcSample/c++をサーバアプリケーションディレクトリとして、その配下のアプリケーションをコンパイルすることでサーバアプリケーションを作成します。
作成方法については、“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜作成＞を参照してください。

＜作成／クライアント＞

$OD_SAMPLES/CalcSample/c++_diiをクライアントアプリケーションディレクトリとして、その配下のアプリケーションをコンパイルすることでクライアントアプリケーションを作成します。
作成方法については、“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜作成＞を参照してください。

＜実行＞

“D.4.2.3 動的起動インタフェース／四則演算（C言語）”の＜実行＞を参照してください。

注意

クライアントアプリケーションの実行時に以下のエラーメッセージが表示されることがあります。これはクライアントアプリケーションの実行時のタイミングによるもので、動作上は問題ありません。

```
error poll-response
```

D.4.3 ネーミングサービス

(3) ネーミングサービスのサンプルプログラムの実行例を説明します。

種別	ディレクトリ(注)	備考
ネーミングサービス／listメソッド	$OD_SAMPLES/complex/samplelist.C/naming/list $OD_SAMPLES/complex/samplelist.C++/naming/list	C C++
ネーミングサービス／bind, resolve, destroyメソッド	$OD_SAMPLES/complex/samplelist.C/naming/bind, resolve, destroy $OD_SAMPLES/complex/samplelist.C++/naming/bind, resolve, destroy	C C++

注)
ここで使用している環境変数は、以下です（Interstageのインストールパス：デフォルト）。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

D.4.3.1 ネーミングサービス／listメソッド

＜作成＞

“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. アプリケーションの登録

registar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.sh
```

2. クライアントアプリケーションの起動

exec-CL.shを実行してクライアントアプリケーションを起動し、出力結果を確認します。

```
exec-CL.sh
```

3. アプリケーションの登録削除

unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

D.4.3.2 ネーミングサービス／bind, resolve, destroyメソッド

＜作成＞

“D.4.1.1 静的インタフェース／各種データ型(C言語)”の＜作成＞を参照してください。
bind、resolve、destroyのそれぞれについてコンパイルを行ってください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. アプリケーションの登録

bindディレクトリへ移動します。その後、registar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.sh
```

2. bindメソッドアプリケーションの起動

exec-CL.shを実行して、bindメソッドアプリケーションを起動します。アプリケーションが正常に動作した場合は、何も出力されません。

```
exec-CL.sh
```

3. resolveメソッドアプリケーションの起動

resolveディレクトリへ移動します。その後、exec-CL.shを実行して、resolveメソッドアプリケーションを起動します。アプリケーションが正常に動作した場合は、何も出力されません。

```
exec-CL.sh
```

4. destroyメソッドアプリケーションの起動

destroyディレクトリへ移動します。その後、exec-CL.shを実行して、destroyメソッドアプリケーションを起動します。アプリケーションが正常に動作した場合は、何も出力されません。

```
exec-CL.sh
```

5. アプリケーションの登録削除

bindディレクトリへ移動します。その後、unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

D.4.4 インタフェースリポジトリ

(4) インタフェースリポジトリのサンプルプログラムの実行例を説明します。

種別	ディレクトリ(注)	備考
インタフェースリポジトリ	$OD_SAMPLES/irsample/*	

注)
ここで使用している環境変数は、以下です(Interstageのインストールパス:デフォルト)。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

D.4.4.1 インタフェースリポジトリ

＜作成＞

“D.4.1.1 静的インタフェース／各種データ型(C言語)”の＜作成＞を参照してください。

＜実行＞

アプリケーションの実行手順を以下に示します。

1. アプリケーションの登録

registar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.sh
```

2. アプリケーションの実行

exec-CL.shを実行して、アプリケーションを実行し、結果を確認します。

```
exec-CL.sh
```

3. アプリケーションの登録削除

unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

D.4.5 POAインタフェース

(5) POAインタフェースのサンプルプログラムの実行例を説明します。

種別	ディレクトリ(注)	備考
Solaris32 Linux32 POAインタフェース(C++言語)	$OD_SAMPLES/POA/C++/*	
POAインタフェース(Java言語)	$OD_SAMPLES/POA/java_1_4/*	

注)
ここで使用している環境変数は、以下です(Interstageのインストールパス:デフォルト)。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

D.4.5.1 POAインタフェース(C++言語)

Solaris32 Linux32

"D.4.1.1 静的インタフェース/各種データ型(C言語)"を参照してください。

D.4.5.2 POAインタフェース(Java言語)

"D.4.1.3 静的インタフェース/各種データ型(Java言語)"を参照してください。

D.4.6 プロセスバインド

(6) プロセスバインドのサンプルプログラムの実行例を説明します。

種別	ディレクトリ(注)	備考
プロセスバインド	$OD_SAMPLES/complex/samplelist.C++/ssn/*	

注)
ここで使用している環境変数は、以下です(Interstageのインストールパス:デフォルト)。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

D.4.6.1 プロセスバインド

"D.4.1.1 静的インタフェース/各種データ型(C言語)"を参照してください。

ポイント

プロセスバインドのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合は、"D.4.1.1 静的インタフェース/各種データ型(C言語)"のポイントを参照してください。

注意

ssn_sample2のサンプルプログラムを実行すると、以下のメッセージが出力されます。これはサンプルプログラムの出力であり、システムやCORBAサービスで異常が発生したことによるものではありません。

```
SystemException raised!
exception-id = IDL:CORBA/StExcep/INV_OBJREF:1.0
minor = 0x464a0928
```

D.4.7 各種API

(7) 各種APIのサンプルプログラムの実行例を説明します。

種別	ディレクトリ(注)	備考
net_disconnect	$OD_SAMPLES/complex/samplelist.C/api/net_disconnect $OD_SAMPLES/complex/samplelist.C++/api/net_disconnect $OD_SAMPLES/complex/samplelist.Java/api/net_disconnect	スレッドモード用
	$OD_SAMPLES/complex/samplelist.C_nt/api/net_disconnect $OD_SAMPLES/complex/samplelist.C++_nt/api/net_disconnect	プロセスモード用
set_client_request_timer	$OD_SAMPLES/complex/samplelist.C/api/set_client_request_timer $OD_SAMPLES/complex/samplelist.C++/api/set_client_request_timer $OD_SAMPLES/complex/samplelist.Java/api/set_client_request_timer	スレッドモード用
	$OD_SAMPLES/complex/samplelist.C_nt/api/set_client_request_timer $OD_SAMPLES/complex/samplelist.C++_nt/api/set_client_request_timer	プロセスモード用
string_to_object	$OD_SAMPLES/complex/samplelist.C/api/string_to_object $OD_SAMPLES/complex/samplelist.C++/api/string_to_object $OD_SAMPLES/complex/samplelist.Java/api/string_to_object	スレッドモード用
	$OD_SAMPLES/complex/samplelist.C_nt/api/string_to_object $OD_SAMPLES/complex/samplelist.C++_nt/api/string_to_object	プロセスモード用

注)
ここで使用している環境変数は、以下です(Interstageのインストールパス:デフォルト)。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

D.4.7.1 net_disconnect

＜作成＞

C言語／C++言語の場合は、“D.4.1.1 静的インタフェース／各種データ型(C言語)”の＜作成＞を参照してください。
Java言語の場合は、“D.4.1.3 静的インタフェース／各種データ型(Java言語)”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。
なお、Java言語の場合は、＜作成＞の“環境変数の設定”を行った端末で実施してください。

1. SSL環境設定

SSLの環境設定を行います。詳細については、“セキュリティシステム運用ガイド”の“CORBAサービスでSSLを利用する方法”を参照してください。

2. サーバアプリケーションの登録

registar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.sh
```

3. サーバアプリケーションの起動

exec-SV.shを実行して、サーバアプリケーションを起動します。

```
exec-SV.sh &
```

4. コネクション数の確認（疎通前）

odprtcurparamコマンドを使用して、疎通前のコネクション数（IIOP_resp_conのvalueの値）を確認します（OD_HOME：CORBAサービスのインストールパス）。

```
$OD_HOME/bin/odprtcurparam
```

5. クライアントアプリケーションの起動

exec-CL.shを実行してクライアントアプリケーションを起動します。

```
exec-CL.sh
```

6. コネクション数の確認（疎通後）

サーバ側のコマンドプロンプトにおいてodprtcurparamコマンドを使用して、疎通後のコネクション数（IIOP_resp_conのvalueの値）が疎通前より1だけ増加していることを確認します。

```
$OD_HOME/bin/odprtcurparam
```

7. クライアントアプリケーションの終了

クライアント側のコマンドプロンプトにおいてEnterキーを押下し、クライアントアプリケーションを終了します。

8. サーバアプリケーションの終了

stop-SV.shを実行して、サーバアプリケーションを停止します。

```
stop-SV.sh
```

9. サーバアプリケーションの登録削除

unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

ポイント

- クライアントアプリケーションにおいて、CORBA_ORB_net_disconnect（C言語）／CORBA::ORB::net_disconnect（C++言語）／com.fujitsu.ObjectDirector.CORBA.ORB.net_disconnect（Java言語）をコメントアウトした上で作成／実行することにより、これらのAPIが通信資源（コネクション数）に与える影響を確認できます。
  該当のAPIをコメントアウトしたアプリケーションでは通信資源が解放されないため、“6. コネクション数の確認（疎通後）”においてコネクション数は疎通前より2だけ増加します。
- net_disconnectのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
  サーバアプリケーションをワークユニット上で動作させる場合、C言語／C++言語については“D.4.1.1 静的インタフェース／各種データ型（C言語）”のポイントを参照してください。Java言語については“D.4.1.3 静的インタフェース／各種データ型（Java言語）”のポイントを参照してください。

注意

サーバ側無通信監視機能／クライアント側無通信監視機能の影響により、“6. コネクション数の確認（疎通後）”において期待する結果が得られないことがあります。この場合、再度アプリケーションを実行してください。

D.4.7.2 set_client_request_timer

＜作成＞

C言語／C++言語の場合は、“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜作成＞を参照してください。
Java言語の場合は、“D.4.1.3 静的インタフェース／各種データ型（Java言語）”の＜作成＞を参照してください。

＜実行＞

“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜実行＞を参照してください。
なお、Java言語の場合は＜作成＞の“環境変数の設定”を実施した端末で実施してください。

ポイント

set_client_request_timerのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合、C言語／C++言語については“D.4.1.1 静的インタフェース／各種データ型（C言語）”のポイントを参照してください。Java言語については“D.4.1.3 静的インタフェース／各種データ型（Java言語）”のポイントを参照してください。

D.4.7.3 string_to_object

＜作成＞

C言語／C++言語の場合は、“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜作成＞を参照してください。
Java言語の場合は、“D.4.1.3 静的インタフェース／各種データ型（Java言語）”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。
なお、Java言語の場合は、＜作成＞の“環境変数の設定”を実施した端末で実施してください。

1. サーバアプリケーションの登録

registar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDの登録と、オブジェクトリファレンスのファイル出力を実行します。

```
register.sh
```

2. サーバアプリケーションの起動

exec-SV.shを実行して、サーバアプリケーションを起動します。

```
exec-SV.sh &
```

3. クライアントアプリケーションの起動

exec-CL.shを実行してクライアントアプリケーションを起動し、出力結果を確認します。なお、必要に応じて、Interstageのインストール環境にあわせてexec-CL.shを修正してください。

```
exec-CL.sh
```

4. サーバアプリケーションの終了

stop-SV.shを実行して、サーバアプリケーションを停止します。

```
stop-SV.sh
```

5. サーバアプリケーションの登録削除

unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

ポイント

string_to_objectのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合、C言語／C++言語については“D.4.1.1 静的インタフェース／各種データ型（C言語）”のポイントを参照してください。Java言語については“D.4.1.3 静的インタフェース／各種データ型（Java言語）”のポイントを参照してください。

D.4.8 Portable-ORB

(8) Portable-ORBのJava連携機能で使用できるサンプルプログラムの実行例を説明します。
Java言語のCORBAアプリケーションをコンパイル、実行するためには、環境設定を行う必要があります。ORBの指定については、“5.4.2 ORB(Object Request Broker)の指定”を参照してください。クライアントの環境設定については、“5.6 クライアントの環境設定（Portable-ORB）”を参照してください。サーバの環境設定については、“5.7 サーバの環境設定”を参照してください。また、アプレットを実行する場合は、事前に“5.21.2 クライアント環境でのアプレット運用時の注意事項”を参照してください。
サンプルプログラムは、以下のディレクトリ配下に格納されています。

種別	ディレクトリ(注)	備考
Portable-ORB／各種データ型	$OD_SAMPLES/complex/samplelist.Java/data/*	
Portable-ORB／POAインタフェース	$OD_SAMPLES/POA/java_1_4/*	
Portable-ORB／四則演算	$PORB_SAMPLES/CalcSample/java	
Portable-ORB／各種API（net_disconnect）	$OD_SAMPLES/complex/samplelist.Java/API/net_disconnect	

種別	ディレクトリ(注)	備考
Portable-ORB／各種API（set_client_request_timer）	$OD_SAMPLES/complex/samplelist.Java/API/set_client_request_timer	
Portable-ORB／各種API（string_to_object）	$OD_SAMPLES/complex/samplelist.Java/API/string_to_object	

注)
ここで使用している環境変数は、以下です(Interstageのインストールパス:デフォルト)。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64
PORB_SAMPLES	/opt/FJSVporb/src/samples	

D.4.8.1 Portable-ORB／各種データ型

注意

Portable-ORBでは、context型のデータの受け渡しができないため、context型のサンプルは実行できません。

Solaris32/64
ここでは、同一システムにCORBAサービスとPortable-ORBをインストールします。サーバアプリケーションはJavaアプリケーションで、クライアントアプリケーションがPortable-ORBを使用する場合の実行例を示します。

＜作成＞

“D.4.1.3 静的インタフェース／各種データ型(Java言語)”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. 環境変数(サーバ)の設定

環境変数CLASSPATH、LD_LIBRARY_PATH、PATHを設定します。環境変数PATHには、使用するJDK/JREのディレクトリを指定します(OD_HOME:CORBAサービスのインストールパス)。

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export  CLASSPATH

LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH

PATH=<JDK/JREのインストールパス>/bin:$PATH
export PATH
```

2. サーバアプリケーションの登録

サンプルアプリケーションディレクトリへ移動します。その後、registar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.sh
```

3. サーバアプリケーションの起動

exec-SV.shを実行して、サーバアプリケーションを起動します。

```
exec-SV.sh
```

4. 環境変数(クライアント)の設定

別端末を起動して、Portable-ORBを動作させるための環境変数CLASSPATH、PATHを指定します(PORB_HOME:Portable-ORBのインストールパス"/opt/FJSVporb")。

```
CLASSPATH=.:$PORB_HOME/lib/ODporb4.jar:$PORB_HOME/lib/CosNaming4.jar:$CLASSPATH
export CLASSPATH

PATH=<JDK/JREのインストールパス>/bin:$PATH
export PATH
```

5. クライアントアプリケーションの起動

サンプルアプリケーションディレクトリへ移動します。その後、exec-CL.shを実行して、クライアントアプリケーションを起動し、出力結果を確認します。

```
exec-CL.sh
```

6. サーバアプリケーションの終了

stop-SV.shを実行して、サーバアプリケーションを停止します。

```
stop-SV.sh
```

7. サーバアプリケーションの登録削除

unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

Linux32/64

ここでは、サーバシステムにCORBAサービスを、Windows(R)クライアント上にPortable-ORBをインストールします。サーバアプリケーションはJavaアプリケーションで、クライアントアプリケーションがPortable-ORBを使用する場合の実行例を示します。
なお、Windows(R)クライアントには、Java実行環境をインストールしておいてください。

＜作成＞

"D.4.1.3 静的インタフェース／各種データ型(Java言語)"の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. 環境変数の設定(サーバマシン)

サーバにおいて、環境変数CLASSPATH、LD_LIBRARY_PATH、PATHを設定します。環境変数PATHには、使用するJDK/JREのディレクトリを指定します(OD_HOME:CORBAサービスのインストールパス)。

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export  CLASSPATH

LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH

PATH=<JDK/JREのインストールパス>/bin:$PATH
export PATH
```

2. サーバアプリケーションの登録(サーバマシン)

サンプルアプリケーションディレクトリへ移動します。その後、registar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.sh
```

3. サーバアプリケーションの起動(サーバマシン)

exec-SV.shを実行して、サーバアプリケーションを起動します。

```
exec-SV.sh &
```

4. クライアントアプリケーションの転送(Windows(R)クライアントマシン)

サンプルアプリケーションディレクトリ配下にある(サブディレクトリ含む)すべてのクラスファイル(拡張子がclassであるファイル)をWindows(R)クライアントへ転送します。その際、Windows(R)クライアントのサブディレクトリ配下の構成、およびディレクトリ名(大文字/小文字)は、Solaris/Linuxサーバと同一にする必要があります。
ftpを使用した場合の例を以下に示します。Windows(R)クライアントの格納先のディレクトリ配下にODsampleディレクトリを、ODsampleディレクトリ配下にはarraytestPackageディレクトリをあらかじめ作成しておいてください。

```
ftp <Linuxサーバ名>
:
cd /opt/FJSVod/src/samples/complex/samplelist.Java/data/array
bin
mget *.class
lcd ODsample
cd ODsample
mget *.class
lcd arraytestPackage
cd arraytestPackage
mget *.class
```

5. 環境変数の設定(Windows(R)クライアントマシン)

Windows(R)クライアントにおいてPortable-ORBのORBクラスを動作させるため、ORBクラスを指定します。指定方法については、"5.4.2 ORB(Object Request Broker)の指定"を参照してください。
また、コマンドプロンプトを起動して、Portable-ORBを動作させるための環境変数CLASSPATHを指定します(PORB_HOME:Portable-ORBのインストールパス"C:¥Interstage¥PORB")。

```
set CLASSPATH=.;%PORB_HOME%¥lib¥ODporb4.jar;%PORB_HOME%¥lib¥CosNaming4.jar;%CLASSPATH%
```

6. Portable-ORBの動作環境ファイルの設定（Windows(R)クライアントマシン）

Portable-ORBの動作環境ファイルを設定します。porbeditenvコマンドを使用して、%PORB_HOME%に格納される動作環境ファイルのホスト情報を設定します。

7. クライアントアプリケーションの起動（Windows(R)クライアントマシン）

クラスファイルを転送した任意のディレクトリでクライアントアプリケーションを実行し、結果を確認します。

```
java simple_c
```

8. サーバアプリケーションの終了（サーバマシン）

サーバにおいてstop-SV.shを実行して、サーバアプリケーションを停止します。

```
stop-SV.sh
```

9. サーバアプリケーションの登録削除（サーバマシン）

unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

D.4.8.2 Portable-ORB／POAインタフェース

“D.4.8.1 Portable-ORB／各種データ型”を参照してください。

D.4.8.3 Portable-ORB／四則演算

ここでは、同一システム（サーバ）にCORBAサービスとPortable-ORBをインストールした上で、同一システム内のWebサーバ上にJavaアプレットを格納し、Windows(R)クライアント上にJavaアプレットとPortable-ORBをダウンロードして使用する場合の実行例を示します。なお、Windows(R)クライアント上にJBKプラグインをインストールしておいてください。

また、ここでは以下のいずれかをサーバアプリケーションとして使用します。

種別	ディレクトリ(注)
静的インタフェース／四則演算（C言語）	$OD_SAMPLES/CalcSample/c
静的インタフェース／四則演算（C++言語）	$OD_SAMPLES/CalcSample/c++

注)
ここで使用している環境変数は、以下です（Interstageのインストールパス：デフォルト）。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

＜作成／サーバ＞

“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜作成＞を参照してください。

＜作成／クライアント＞

クライアントアプリケーションの作成手順を以下に示します。
クライアントアプリケーションは、サーバ上で作成します。

1. 環境変数の設定

コマンドプロンプトを起動し、環境変数CLASSPATH、PATHを設定します（PORB_HOME：Portable-ORBのインストールパス“/opt/FJSVporb”）。

```
CLASSPATH=.:$PORB_HOME/lib/ODporb4.jar:$PORB_HOME/lib/CosNaming4.jar:$CLASSPATH
export CLASSPATH

PATH=<JDK/JREのインストールパス>/bin:$PATH
export PATH
```

2. クライアントアプリケーションの作成

makeコマンドおよびcreate-jar.shを実行して、クライアントアプリケーションのjarアーカイブファイル(client2.jar)を作成します。

```
make
create-jar.sh
```

3. デジタル署名の実施

作成したクライアントアプリケーションのjarアーカイブファイルと使用するPortable-ORBのライブラリにデジタル署名を実施し、クライアントアプリケーションを実行するWindows(R)クライアントでデジタル署名を有効とするための環境設定を行います。
デジタル署名の詳細については、“5.8.2 ダウンロードオブジェクトと署名対象オブジェクト”および“5.8.3 デジタル署名手順”を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. クライアントアプリケーションのコピー（サーバマシン）

作成したクライアントアプリケーションをWebサーバのドキュメントルート配下に存在する任意のディレクトリにコピーします（APPLETDIR:アプレットの格納パス）。
なお、client2.jarは、＜作成／クライアント＞の“デジタル署名の実施”でデジタル署名を実施したjarアーカイブファイルです。

```
mkdir $APPLETDIR
cp $PORB_SAMPLES/CalcSample/java/CalcSample4.html $APPLETDIR
cp $PORB_SAMPLES/CalcSample/java/CalcSample4.js $APPLETDIR
cp $PORB_SAMPLES/CalcSample/java/client2.jar $APPLETDIR
```

2. Portable-ORBファイルのリンク（サーバマシン）

Webサーバにおいて、Portable-ORBをダウンロード可能なようにリンクコマンドを使用してリンクを作成します。
なお、ODporb4_plugin.jar、CosNaming4_plugin.jar、InterfaceRep4_plugin.jarは、＜作成／クライアント＞の“デジタル署名の実施”でデジタル署名を実施したjarアーカイブファイルです。

```
cd $APPLETDIR
ln -s $PORB_HOME/lib/ODporb4_plugin.jar ODporb4_plugin.jar
ln -s $PORB_HOME/lib/CosNaming4_plugin.jar CosNaming4_plugin.jar
ln -s $PORB_HOME/lib/InterfaceRep4_plugin.jar InterfaceRep4_plugin.jar
ln -s $PORB_HOME/etc etc
```

3. Portable-ORBの動作環境ファイルの設定（サーバマシン）

Portable-ORBの動作環境ファイルを設定します。porbeditenvコマンドを使用して、$PORB_HOMEに格納される動作環境ファイルのホスト情報を設定します。

4. サーバアプリケーションの登録（サーバマシン）

サーバにおいて、サンプルアプリケーションディレクトリへ移動します。その後、registar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.sh
```

5. サーバアプリケーションの起動（サーバマシン）

exec-SV.shを実行して、サーバアプリケーションを起動します。

```
exec-SV.sh &
```

6. クライアントの実行（Windows(R)クライアントマシン）

Windows(R)クライアントにおいて、Portable-ORBのORBクラスが動作するようにORBクラスを指定します。指定方法については、“5.4.2 ORB(Object Request Broker)の指定”を参照してください。
その後、ブラウザでCalcSample4.htmlが格納されるURLを指定し、アプレットを起動します。CalcSample4.htmlをダブルクリックした場合（fileプロトコル使用）は、セキュリティチェックの対象となり、正常に動作しないことがあります。
入力画面から数値や文字を入力して“OK”をクリックし、結果を確認します。

7. サーバアプリケーションの終了（サーバマシン）

サーバにおいてstop-SV.shを実行して、サーバアプリケーションを停止します。

```
stop-SV.sh
```

8. サーバアプリケーションの登録削除（サーバマシン）

unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

D.4.8.4 Portable-ORB／各種API（net_disconnect）

＜作成＞

“D.4.1.3 静的インタフェース／各種データ型（Java言語）”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. 環境変数（サーバ）の設定

環境変数CLASSPATH、LD_LIBRARY_PATH、PATHを設定します。環境変数PATHには、使用するJDK/JREのディレクトリを指定します（OD_HOME：CORBAサービスのインストールパス）。

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export  CLASSPATH

LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH

PATH=<JDK/JREのインストールパス>/bin:$PATH
export PATH
```

2. SSL環境(サーバ)設定

サーバにおいてSSLの環境設定を行います。SSLの環境設定については、“セキュリティシステム運用ガイド”の“CORBAサービスでSSLを利用する方法”を参照してください。

3. SSL環境(クライアント)設定

クライアントにおいてSSLの環境設定を行います。SSLの環境設定については、“セキュリティシステム運用ガイド”の“Portable-ORBでSSLを利用する方法”を参照してください。

4. サーバアプリケーションの登録

registar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDとオブジェクトリファレンスを登録します。

```
register.sh
```

5. サーバアプリケーションの起動

exec-SV.shを実行して、サーバアプリケーションを起動します。

```
exec-SV.sh &
```

6.コネクション数の確認(疎通前)

odprtcurparamコマンドを使用して、疎通前のコネクション数(IIOP_resp_conのvalueの値)を確認します(OD_HOME:CORBAサービスのインストールパス)。

```
$OD_HOME/bin/odprtcurparam
```

7. 環境変数(クライアント)の設定

別端末を起動して、Portable-ORBを動作させるための環境変数CLASSPATH／PATHを指定します(PORB_HOME:Portable-ORBのインストールパス“/opt/FJSVporb”)。

```
CLASSPATH=.:$PORB_HOME/lib/ODporb4.jar:$PORB_HOME/lib/CosNaming4.jar:$CLASSPATH
export CLASSPATH

PATH=<JDK/JREのインストールパス>/bin:$PATH
export PATH
```

8. クライアントアプリケーションの起動

exec-CL.shを実行してクライアントアプリケーションを起動します。

```
exec-CL.sh
```

9. コネクション数の確認(疎通後)

サーバ側のコマンドプロンプトにおいてodprtcurparamコマンドを使用して、疎通後のコネクション数(IIOP_resp_conのvalueの値)が疎通前より1だけ増加していることを確認します。

```
$OD_HOME/bin/odprtcurparam
```

10. クライアントアプリケーションの終了

クライアント側のコマンドプロンプトにおいてEnterキーを押下し、クライアントアプリケーションを終了します。

11. サーバアプリケーションの終了

stop-SV.shを実行して、サーバアプリケーションを停止します。

```
stop-SV.sh
```

12. サーバアプリケーションの登録削除

unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

ポイント

- クライアントアプリケーションにおいて、com.fujitsu.ObjectDirector.CORBA.ORB.net_disconnectをコメントアウトした上で作成/実行することにより、APIが通信資源(コネクション数)に与える影響を確認できます。
  該当のAPIをコメントアウトしたアプリケーションでは通信資源が解放されないため、"9. コネクション数の確認(疎通後)"においてコネクション数は疎通前より2だけ増加します。
- net_disconnectのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
  サーバアプリケーションをワークユニット上で動作させる場合は、"D.4.1.3 静的インタフェース/各種データ型(Java言語)"のポイントを参照してください。

注意

サーバ側無通信監視機能/クライアント側無通信監視機能の影響により、"9. コネクション数の確認(疎通後)"において期待する結果が得られないことがあります。この場合、再度アプリケーションを実行してください。

D.4.8.5 Portable-ORB/各種API(set_client_request_timer)

"D.4.8.1 Portable-ORB/各種データ型"を参照してください。

ポイント

set_client_request_timerのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合は、"D.4.1.3 静的インタフェース/各種データ型(Java言語)"のポイントを参照してください。

D.4.8.6 Portable-ORB／各種API（string_to_object）

＜作成＞

“D.4.1.3 静的インタフェース／各種データ型（Java言語）”の＜作成＞を参照してください。

＜実行＞

作成したアプリケーションの実行手順を以下に示します。

1. 環境変数（サーバ）の設定

環境変数CLASSPATH、LD_LIBRARY_PATH、PATHを設定します。環境変数PATHには、使用するJDK/JREのディレクトリを指定します（OD_HOME：CORBAサービスのインストールパス）。

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export  CLASSPATH

LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH

PATH=<JDK/JREのインストールパス>/bin:$PATH
export PATH
```

2. サーバアプリケーションの登録

registar.shを実行して、サンプルプログラムを実行するために必要なインプリメンテーションリポジトリIDの登録、およびオブジェクトリファレンスのファイル出力を実行します。

```
register.sh
```

3. サーバアプリケーションの起動

exec-SV.shを実行して、サーバアプリケーションを起動します。

```
exec-SV.sh &
```

4. 環境変数（クライアント）の設定

別端末を起動して、Portable-ORBを動作させるための環境変数CLASSPATH／PATHを指定します（PORB_HOME：Portable-ORBのインストールパス“/opt/FJSVporb”）。

```
CLASSPATH=.:$PORB_HOME/lib/ODporb4.jar:$PORB_HOME/lib/CosNaming4.jar:$CLASSPATH
export CLASSPATH

PATH=<JDK/JREのインストールパス>/bin:$PATH
export PATH
```

5. クライアントアプリケーションの起動

exec-CL.shを実行してクライアントアプリケーションを起動し、出力結果を確認します。なお、必要に応じて、Interstageのインストール環境にあわせてexec-CL.shを修正してください。

```
exec-CL.sh
```

6. サーバアプリケーションの終了

stop-SV.shを実行して、サーバアプリケーションを停止します。

```
stop-SV.sh
```

7. サーバアプリケーションの登録削除

unregister.shを実行して、不要となったサンプルプログラムのインプリメンテーションリポジトリIDとオブジェクトリファレンスを削除します。

```
unregister.sh
```

ポイント

string_to_objectのサーバアプリケーションは、ワークユニット上で動作させることも可能です。
サーバアプリケーションをワークユニット上で動作させる場合は、“D.4.1.3 静的インタフェース／各種データ型（Java言語）”のポイントを参照してください。

D.4.9 その他

(9) その他のサンプルプログラムの実行例を説明します。

種別	ディレクトリ(注)	備考
その他（C言語）	$OD_SAMPLES/simple	スレッドモード用
	$OD_SAMPLES/complex/samplelist.C_nt/simple	プロセスモード用
	$OD_SAMPLES/simple_so	soライブラリ化
その他（C++言語）	$OD_SAMPLES/simple_cpp	スレッドモード用
	$OD_SAMPLES/complex/samplelist.C++_nt/simple_cpp	プロセスモード用
	$OD_SAMPLES/simple_cpp_so	soライブラリ化
その他（Java言語）	$OD_SAMPLES/java	

注)
ここで使用している環境変数は、以下です（Interstageのインストールパス：デフォルト）。

環境変数	値	備考
OD_SAMPLES	/opt/FSUNod/src/samples	Solaris32/64
	/opt/FJSVod/src/samples	Linux32/64

D.4.9.1 その他（C言語）

“D.4.1.1 静的インタフェース／各種データ型（C言語）”を参照してください。

D.4.9.2 その他（C++言語）

“D.4.1.1 静的インタフェース／各種データ型（C言語）”を参照してください。

D.4.9.3 その他（Java言語）

Windows(R)クライアントマシン上では、Internet ExplorerにJBKプラグインをインストールしておいてください。

＜作成／サーバ＞

serverをサーバアプリケーションディレクトリとして、その配下のアプリケーションをコンパイルすることでサーバアプリケーションを作成します。

“D.4.1.1 静的インタフェース／各種データ型（C言語）”の＜作成＞を参照してください。

＜作成／クライアント＞

clientをクライアントアプリケーションディレクトリとして、その配下のアプリケーションをコンパイルすることでクライアントアプリケーションを作成します。

クライアントアプリケーションの作成手順を以下に示します。

1. クライアントアプリケーションのコンパイル（サーバマシン）

クライアントアプリケーションディレクトリへ移動します。その後、makeコマンドを実行してクライアントアプリケーションを作成します。

```
make
```

2. クライアントアプリケーションの配置（Windows(R)クライアントマシン）

クライアントアプリケーションディレクトリ配下にあるすべてのファイルをWindows(R)クライアントへ転送します。
ftpを使用した場合の例を以下に示します。Windows(R)クライアントの格納先のフォルダ配下には、あらかじめAppOpSampleディレクトリを作成しておいてください。なお、Windows(R)クライアントへファイルを転送するときは、すべてのファイルについてネットワークASCIIで転送するように指定します。

```
ftp ＜Solaris/Linuxサーバ名＞
：
cd $SAMPLES/java/client
mget *
lcd AppOpSample
cd AppOpSample
mget *
```

3. 権限の設定（Windows(R)クライアント）

Javaライブラリ(ODjava4.jar) 、およびアプリケーションの格納パスに権限を設定します。
詳細については、“5.5.2 Javaライブラリに対する権限の設定”を参照してください。

＜実行＞

“D.4.1.8 静的インタフェース／四則演算（Java言語）”の＜実行＞を参照してください。

付録E　プロセスバインド機能を用いたセション管理の例

以下のIDL定義に従って、プロセスバインド機能を用いてセション管理を実装した場合の例について説明します。

```
module ODsample
{
    interface ssnintf
    {
        void destroyssn();
        void invoke_ssn(in long a);
    };

    interface testintf
    {
        Object createssn();
    };
};
```

なお、この例では、testintfインタフェースはセション継続を行うオブジェクトリファレンスを取得するために使用され、ssnintfインタフェースは実際のセション継続を行うメソッドを呼び出すために使用されます。

E.1　クライアントアプリケーションのプログラミング

クライアントアプリケーションの処理の流れを以下に示します。

E.1.1　初期化

CORBAの初期化メソッドCORBA::ORB_init()を呼び出し、初期化処理を行います。結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
main( int argc, char *argv[] )
{
    CORBA::ORB_ptr          orb;                    // ORBのオブジェクトリファレンス
    CORBA::Environment_ptr  env;                    // 例外情報
    int                     current_argc = argc;

    env = new CORBA::Environment;                   // 例外情報
    orb = CORBA::ORB_init(current_argc, argv, FJ_OM_ORBid, *env);
```

E.1.2 ネーミングサービスのオブジェクトリファレンスの獲得

実行するオブジェクトをネーミングサービスから検索するために、ネーミングサービスのオブジェクトリファレンスが必要となります。CORBAインタフェースのオブジェクトリファレンスの取出しメソッドCORBA::ORB::resolve_initial_references()によりネーミングサービスのオブジェクトリファレンスを取り出します。このとき、CORBA_ORB_ObjectId_NameServiceをメソッドのパラメタとして指定します。

```
// NamingServiceのオブジェクトリファレンスを獲得
CORBA::Object_ptr
obj = orb->resolve_initial_references( CORBA_ORB_ObjectId_NameService, env );

// NamingContextクラスへの変換
CosNaming::NamingContext_ptr
cos_naming = CosNaming::NamingContext::_narrow( obj );
```

E.1.3 サーバアプリケーションのオブジェクトリファレンスの獲得

ネーミングサービスのメソッドCosNaming::NamingContext::resolve()により、セション継続用のオブジェクトリファレンスを取得するためのサーバアプリケーションのオブジェクトリファレンスを取り出します。検索するオブジェクト名を本メソッドのパラメタとして指定します。

```
CosNaming::Name         name;    // CosNaming::Nameのインスタンス
CORBA::Object_ptr       obj;     // サーバアプリケーションのオブジェクトリファレンス

name.length(1);
// オブジェクト名
name[0]->id = (const CORBA::Char *)"ODsample::ssntest";
// オブジェクトのタイプ
name[0]->kind = (const CORBA::Char *)"";

// サーバアプリケーションのオブジェクトリファレンスを獲得
obj = cos_naming->resolve( name, *env );

// ODsample::testintfクラスへの変換
ODsample::testintf_ptr nossn_ap = ODsample::testintf::_narrow( obj );
```

E.1.4 セション継続用オブジェクトリファレンスの獲得

サーバアプリケーションで実装されているセション継続用オブジェクトリファレンス取得用のメソッドを呼び出すことにより、これから実行するサーバアプリケーションのオブジェクトリファレンスを取得します。

```
// セション継続用のオブジェクトリファレンスを獲得
obj = nossn_ap->createssn( *env );

// ODsample::ssnintfクラスへの変換
ODsample::ssnintf_ptr ssn_ap = ODsample::ssnintf::_narrow( obj );
```

E.1.5 メソッドの呼出し

セション継続用サーバプログラムのメソッドを呼び出します。なお、メソッド呼出し時は、サーバプログラムで例外が発生した場合に例外情報を獲得するために、CORBA::Environment構造体を指定します。

```
CORBA::Long           iVal;  // パラメタ

// メソッドの呼出し
iVal = 10;
ssn_ap->invoke_ssn( iVal, *env );

iVal = 20;
ssn_ap->invoke_ssn( iVal, *env );
```

E.1.6　セション破棄用のメソッドの呼出し

サーバアプリケーションで実装されているセション破棄用のメソッドを呼び出します。

```
// セション破棄用メソッドの呼出し
ssn_ap->destroyssn( *env );
```

E.2　サーバアプリケーションのプログラミング

サーバアプリケーションは、初期化処理部とインタフェースの実装部分から構成されます。サーバアプリケーションの処理の流れを以下に示します。

E.2.1　初期化

CORBAの初期化メソッドCORBA::ORB_init()を呼び出して、初期化処理を行います。結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
main ( int argc, char *argv[] )
{
    CORBA::ORB_ptr          orb;    // ORBのオブジェクトリファレンス
    CORBA::Environment_ptr  env;    // 例外情報
```

```
    int                     current_argc = argc;

    env = new CORBA::Environment;
    orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, *env );
```

また、CORBA::ORB::BOA_init()の呼出しにより、基本オブジェクトアダプタの初期化処理を行います。

```
CORBA::BOA_ptr  boa;                // BOA用オブジェクトリファレンス
boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, *env );
```

また、セション継続用のオブジェクトリファレンスの生成処理のため、CORBA::Repository::lookup_id()の呼出しにより、セション継続用インタフェースのオブジェクトリファレンスを取得します。

```
CORBA::InterfaceDef_ptr      intf;     // インタフェース情報検索用
CORBA::Repository_ptr        intf_rep; // Repositoryオブジェクトリファレンス
CORBA::Object_ptr            o;        // オブジェクトリファレンスのポインタ

// インタフェース情報のオブジェクトリファレンス検索
o = orb->resolve_initial_references( CORBA_ORB_ObjectId_LightInterfaceRepository, *env );

// CORBA::Repositoryクラスへの変換
intf_rep = CORBA::Repository::_narrow(o);

// セション継続用のインタフェースオブジェクトの検索
o = intf_rep->lookup_id( _INTF_ODsample_ssnintf, *env );

// CORBA::InterfaceDefクラスへの変換
intf = CORBA::InterfaceDef::_narrow(o);
```

なお、必要であればサーバアプリケーションの初期化処理も行います。

E.2.2 サーバの活性化

サーバアプリケーションの初期化が完了すると、ORBに対してその旨を通知します。ORBは、この命令が発行された時点で、クライアントからの要求をサーバアプリケーションに伝えます。

(1)インプリメンテーションリポジトリのオブジェクトリファレンスの獲得

インプリメンテーションリポジトリのオブジェクトリファレンスを取り出します。メソッドのパラメタには、CORBA_ORB_ObjectId_ImplementationRepositoryを指定します。

(2)ImplementationDefオブジェクトのオブジェクトリファレンスの検索

FJ::ImplementationRep::lookup_id()を使用して、ImplementationDefオブジェクトのオブジェクトリファレンスを求めます。パラメタには、サーバアプリケーションのImplementationRepオブジェクトを指定します。

(3)サーバの活性化

CORBA::BOA::impl_is_ready()を使用して、サーバの活性化を行います。

```
CORBA::ImplementationDef_ptr impl;     // インプリメンテーション情報検索用
FJ::ImplementationRep_ptr    impl_rep; // ImplementationRepオブジェクトリファレンス

// インプリメンテーション情報のオブジェクトリファレンス検索
o = orb->resolve_initial_references( CORBA_ORB_ObjectId_ImplementationRepository, *env );

// ImplementationRepクラスへの変換
```

```
impl_rep = FJ::ImplementationRep::_narrow(o);

// サーバのImplementationRepオブジェクトの検索
o = impl_rep->lookup_id( "IMPL_SSNSAMPLE", *env );

// ImplementationDefクラスへの変換
impl = CORBA::ImplementationDef::_narrow(o);

// サーバの活性化
boa->impl_is_ready( impl, *env );
```

E.2.3 セション継続用のオブジェクトリファレンス生成

クライアントからセション継続用オブジェクトリファレンス取得用のメソッドが要求された場合、クライアントに通知するためのオブジェクトリファレンスを生成します。

```
CORBA::Object_ptr
ODsample_testintf_impl::createssn(
    CORBA::Environment  &arg_env )
    throw ( CORBA::Exception )
{
    CORBA::Object_ptr           obj;       // オブジェクトリファレンスのポインタ
    CORBA::ReferenceData        ref_data;  // オブジェクトの識別情報
    CORBA::Environment          local_env; // 例外情報

    //オブジェクトリファレンス生成
    obj = boa->create( ref_data, intf, impl, local_env );
```

E.2.4 セション継続用のオブジェクトリファレンスのバインド関係登録

オブジェクトリファレンスの生成が完了した後、ORBに対して自プロセスとのバインド関係の登録を通知します。ORBは、このオブジェクトリファレンスを使用したリクエストを受信した場合に、登録関係にあるプロセスにリクエストを振り分けます。

```
// 自プロセスとオブジェクトリファレンスのバインド関係を登録
orb->bind_object( obj, local_env );

return obj;
```

E.2.5 セション継続メソッドの実行

クライアントからセション継続用インタフェースのメソッドが要求された場合、サーバアプリケーションでは、セション継続を行うために特別な処理を行う必要はありません。

```
void
ODsample_ssnintf_impl::invoke_ssn(
      CORBA::Long         a,
      CORBA::Environment  &arg_env )
      throw ( CORBA::Exception )
{
    // 任意の処理を実行
    cout << "invoke_ssn" << a << endl;

    return;
}
```

E.2.6 セション継続用のオブジェクトリファレンスのバインド関係登録削除

クライアントからセション破棄用のメソッドが要求された場合、オブジェクトリファレンスのバインド関係の登録削除関数を発行します。

```
void
ODsample_ssnintf_impl::destroyssn(
      CORBA::Environment  &arg_env )
      throw ( CORBA::Exception )
{
    CORBA::Environment       local_env; // 例外情報

    // オブジェクトリファレンスのバインド関係の登録削除
    orb->unbind_object( this, local_env );

    return;
}
```

付録F　旧バージョンからの移行上の注意

旧バージョンから移行する際の注意事項について説明します。

F.1　Javaインタフェース(JDK/JRE 1.3以前からの移行)

JDK/JRE 1.3以前のJavaインタフェースを使用したアプリケーションを、本製品で使用しているJDK/JREの環境に移行する場合の注意事項について説明します。

■Javaアプリケーション実行前の注意事項

JDK/JRE 1.3以前のjavaコンパイラでコンパイルしたアプリケーションは、本製品で使用しているJDK/JREのjavaコンパイラで再度コンパイルする必要があります。JDK 1.3以前のjavaコンパイラでコンパイルしたアプリケーションを実行すると、以下のメソッドで例外“java.lang.IncompatibleClassChangeError”が発生します。

- org.omg.PortableServer.POAManager.activate()
- org.omg.PortableServer.POAManager.hold_requests()
- org.omg.PortableServer.POAManager.discard_requests()
- org.omg.PortableServer.POAManager.deactivate()

■Javaアプリケーション実行時の注意事項

org.omg.CORBA.SystemExceptionのサブクラスに対して呼び出したtoString()の結果が、JDK/JRE 1.3以前を使用した場合の結果と異なります。マイナーコードは、すべての桁ではなく、一部だけが表示されるようになります。
マイナーコード“0x53550123”、“0x4f4d0123”、“0x464a0123”の場合を例に、org.omg.CORBA.NO_MEMORYのtoString()の結果からマイナーコードを確認する手順を以下に示します。なお、JDK/JRE 1.3以前を使用した場合の結果と同様に表示する場合は、“Javaアプリケーション作成時の注意事項”の“org.omg.CORBA.SystemException”を参照して対応してください。

マイナーコード“0x53550123”の場合

```
org.omg.CORBA.NO_MEMORY: Memory Error  vmcid: SUN  minor code: 291  completed: No
```

マイナーコードと“0xfffff000”との論理積が“0x53550000”と等しい場合は、以下のように表示されます。

－ “vmcid: ”のあとに“SUN”

－ “minor code: ”のあとに、マイナーコードと“0x00000fff”との論理積の結果(10進数)

マイナーコード“0x4f4d0123”の場合

```
org.omg.CORBA.NO_MEMORY: Memory Error  vmcid: OMG  minor code: 291  completed: No
```

マイナーコードと“0xfffff000”との論理積が“0x4f4d0000”と等しい場合は、以下のように表示されます。

－ “vmcid: ”のあとに“OMG”

－ “minor code: ”のあとに、マイナーコードと“0x00000fff”との論理積の結果(10進数)

マイナーコード“0x464a0123”の場合

```
org.omg.CORBA.NO_MEMORY: Memory Error  vmcid: 0x464a0000  minor code: 291  completed: No
```

マイナーコードと“0xfffff000”との論理積が“0x53550000”／“0x4d4f0000”と異なる場合は、以下のように表示されます。

－ “vmcid: ”のあとに、マイナーコードと0xfffff000との論理積

－ “minor code: ”のあとに、マイナーコードと0x00000fffとの論理積の結果(10進数)

したがって、org.omg.CORBA.SystemExceptionのtoString()の結果において、“vmcid: ”のあとの表示で上位5桁を、“minor code: ”のあとの表示で下位3ケタを確認できます。
たとえば、org.omg.CORBA.SystemExceptionのtoString()の結果に“vmcid: SUN minor code: 564”が表示されている場合は、次の手順でマイナーコードを確認します。

1. “vmcid: ”のあとが“SUN”であることから、上位5桁は“0x53550”となります。
2. “minor code: ”のあとが“564”であることから、564（10進数）を16進数に変換して、下位3桁は“0x234”となります。
3. 1の上位5桁および2の下位3桁から、マイナーコードは“0x53550234”であることが確認できます。

■Javaアプリケーション作成時の注意事項

JDK/JRE 1.3以前のインタフェースと本製品で使用しているJDK/JREのインタフェースには、機能差異があります。
JDK/JRE 1.3以前のJavaインタフェースを使用したアプリケーションを移行する場合は、以下の機能差異を考慮して、それぞれの対処を行ってください。

1. org.omg.CORBA.PortableServer.POA.create_reference_with_id()
2. org.omg.PortableServer.AdapterActivator
3. org.omg.PortableServer.ServantActivator
4. org.omg.PortableServer.ServantLocator
5. IDLcで生成されるスタブ・スケルトン
6. org.omg.CORBA.SystemException
7. org.omg.CORBA.ORB.resolve_initial_references()
8. org.omg.PortableServer.Servant._this_object()
9. org.omg.PortableServer.Servant._orb()

(1) org.omg.CORBA.PortableServer.POA.create_reference_with_id()

機能差異

org.omg.PortableServer.POA.create_reference_with_id()は、org.omg.PortableServer.POAPackage.WrongPolicyをスローしません。

org.omg.PortableServer.POA.create_reference_with_id()を使用しているプログラムは、org.omg.PortableServer.POAPackage.WrongPolicyをcatchしないように変更する必要があります。

対処方法

org.omg.PortableServer.POA.create_reference_with_id()がorg.omg.PortableServer.POAPackage.WrongPolicy（非Runtime例外）をスローすることを期待しているプログラムは、そのtry～catch節を見直してください。

例

[移行前（JDK/JRE 1.3以前）のプログラム処理]

```
try {
    org.omg.CORBA.Object obj = myPoa.create_reference_with_id( objID, intfID );
    } catch( org.omg.PortableServer.POAPackage.WrongPolicy e ) {
        // エラー処理
    }
    ...
```

[移行後のプログラム処理]

```
org.omg.CORBA.Object obj = myPoa.create_reference_with_id( objID, intfID );
...
```

(2) org.omg.PortableServer.AdapterActivator

機能差異

org.omg.PortableServer.AdapterActivatorは、interfaceとして扱われます。

対処方法

以下の対処を行ってください。

- org.omg.CORBA.LocalObjectをextendsします。
- org.omg.PortableServer.AdapterActivatorをextendsしているユーザクラスを、implementsに変更します。

例

[移行前(JDK/JRE 1.3以前)のプログラム処理]

```
class UserAdapterActivator
    extends org.omg.PortableServer.AdapterActivator
    {
        ...
    }
```

[移行後のプログラム処理]

```
class UserAdapterActivator
    extends org.omg.CORBA.LocalObject
    implements org.omg.PortableServer.AdapterActivator
    {
        ...
    }
```

(3) org.omg.PortableServer.ServantActivator

機能差異

org.omg.PortableServer.ServantActivatorは、interfaceとして扱われます。

対処方法

以下の対処を行ってください。

- org.omg.CORBA.LocalObjectをextendsします。
- org.omg.PortableServer.ServantActivatorをextendsしているユーザクラスを、implementsに変更します。

例

[移行前(JDK/JRE 1.3以前)のプログラム処理]

```
class UserServantActivator
    extends org.omg.PortableServer.ServantActivator
    {
        ...
    }
```

[移行後のプログラム処理]

```
class UserServantActivator
    extends org.omg.CORBA.LocalObject
    implements org.omg.PortableServer.ServantActivator
    {
        ...
    }
```

(4) org.omg.PortableServer.ServantLocator

機能差異

org.omg.PortableServer.ServantLocatorは、interfaceとして扱われます。

対処方法

以下の対処を行ってください。

- org.omg.CORBA.LocalObjectをextendsします。
- org.omg.PortableServer.ServantLocatorをextendsしているユーザクラスを、implementsに変更します。

例

[移行前(JDK/JRE 1.3以前)のプログラム処理]

```
class UserServantLocator
    extends org.omg.PortableServer.ServantLocator
    {
        ...
    }
```

[移行後のプログラム処理]

```
class UserServantLocator
    extends org.omg.CORBA.LocalObject
    implements org.omg.PortableServer.ServantLocator
    {
        ...
    }
```

(5) IDLcで生成されるスタブ・スケルトン

機能差異

module宣言のない定義ファイルから生成されたスタブ・スケルトンは、javacで失敗します。

対処方法

必ずmodule宣言の下に内容を記述するようにしてください。

例

[移行前(JDK/JRE 1.3以前)のプログラム処理]

```
interface   intf{
    long    add(in long a, in long b);
};
```

[移行後のプログラム処理]

```
module mod{
    interface   intf{
        long    add(in long a, in long b);
    };
};
```

(6) org.omg.CORBA.SystemException

機能差異

org.omg.CORBA.SystemExceptionのサブクラスに対して呼び出したtoString()の結果が異なります。特にマイナーコードについては、すべての桁が表示されません。

対処方法

必ず以下の形式で例外情報を出力するようにしてください。

例

```
try {
        :
}
catch( org.omg.CORBA.SystemException e ) {
    System.out.println( "ERROR : " + e.getClass().getName()
        + " : minor = 0x" + java.lang.Integer.toHexString(e.minor) );
        :
}
```

(7) org.omg.CORBA.ORB.resolve_initial_references()

機能差異

以下のいずれかの条件で、“RootPOA”を指定してorg.omg.CORBA.ORB.resolve_initial_references()を呼び出した場合、nullが返却されます。

- サーバアプリケーションを登録していない。
- 環境変数OD_IMPLIDを使用したインプリメンテーションの登録方法に誤りがある。

対処方法

必ず復帰値を確認するようにプログラムを変更してください。

(8) org.omg.PortableServer.Servant._this_object()

機能差異

org.omg.PortableServer.Servant._this_object(org.omg.CORBA.ORB orb)形式は、使用できません。
Servantクラスに ORBインスタンスを設定する場合、以下のいずれかのメソッドに、対象となるServantクラスを指定して呼び出す必要があります。

- org.omg.PortableServer.POA.set_servant()
- org.omg.PortableServer.POA.activate_object()
- org.omg.PortableServer.POA.activate_object_with_id()
- org.omg.PortableServer.POA.servant_to_reference()
- org.omg.PortableServer.POA.servant_to_id()

なお、上記のメソッドを使用する場合は、対象となるPOAに必要なポリシが設定されている必要があります。詳細については、“リファレンスマニュアル(API編)”のそれぞれのメソッドを参照してください。

対処方法

org.omg.PortableServer.Servant._this_object(org.omg.CORBA.ORB orb)を使用せずに、上記のメソッドを使用するようにプログラムを変更してください。

例

[移行前(JDK/JRE 1.3以前のインタフェースを使用した場合)のプログラム処理]

```
org.omg.CORBA.ORB Orb;
...
org.omg.PortableServer.Servant svt = new UserServant();
svt._this_object(Orb);
...
```

[移行後のプログラム処理]

```
org.omg.PortableServer.POA childPOA;
...
org.omg.PortableServer.Servant svt = new UserServant();
childPOA.servant_to_reference( svt );
...
```

(9) org.omg.PortableServer.Servant._orb()

機能差異

org.omg.PortableServer.Servant._orb(org.omg.CORBA.ORB orb)形式は、使用できません。
Servantクラスに ORBインスタンスを設定する場合、以下のいずれかのメソッドに、対象となるServantクラスを指定して呼び出す必要があります。

- org.omg.PortableServer.POA.set_servant()
- org.omg.PortableServer.POA.activate_object()
- org.omg.PortableServer.POA.activate_object_with_id()
- org.omg.PortableServer.POA.servant_to_reference()
- org.omg.PortableServer.POA.servant_to_id()

なお、上記のメソッドを使用する場合は、対象となるPOAに必要なポリシが設定されている必要があります。詳細については、“リファレンスマニュアル(API編)”のそれぞれのメソッドを参照してください。

対処方法

org.omg.PortableServer.Servant._this_orb(org.omg.CORBA.ORB orb)を使用せずに、上記のメソッドを使用するようにプログラムを変更してください。

例

[移行前(JDK/JRE 1.3以前のインタフェースを使用した場合)のプログラム処理]

```
org.omg.CORBA.ORB Orb;
...
org.omg.PortableServer.Servant svt = new UserServant();
svt._orb(Orb);
...
```

[移行後のプログラム処理]

```
org.omg.PortableServer.POA childPOA;
...
org.omg.PortableServer.Servant svt = new UserServant();
childPOA.set_servant( svt );
...
```

F.2 ネーミングサービスに関する注意事項(V2以前のバージョンからの移行)

Windows32 Solaris32

旧バージョン(V2以前)のネーミングサービスを使用する場合の注意事項について説明します。

(1) ネーミングサービスに登録するバインディング名の指定について

内容

OD_or_adm、odlistns、odadministerlb、oddisplaylboコマンドで指定するネーミングサービスに登録するバインディング名の指定方法がCORBAの標準にあわせることにより一部が変更されました。

対処方法

コマンド入力時、新しい指定方法でバインディング名を指定してください。

備考

ネーミングサービスの資産には、影響はありません。新しいバインディング名の指定方法は、“9.4 文字列表記のバインディング名”を参照してください。
V2までのバインディング名は、ネーミングコンテキストを含むオブジェクトリファレンスを階層で指定します。この場合、先頭をスラッシュ(/)とするフルパスで指定し、スラッシュ(/)で区切ります。先頭がスラッシュ(/)の場合は、バインディング名がコンパウンド名とみなされるため、“/name”はエラーとなります。

(2) 旧バージョンのINTERSTAGEとの混在運用について

内容

ネーミングサービスの拡張機能のサポートにより、ネーミングサービスのインタフェース名が変更されました。このため、以下の運用形態でネーミングサービスのイニシャルサービスに登録する場合は、従来のインタフェースIDで再度初期化を行う必要があります。従来のインタフェースIDでネーミングサービスを初期化することは、拡張機能を使用しない設定となります。この設定を行った場合は、拡張機能を使用できません。

- V2以前のパッケージがインストールされているマシンのJavaクライアントから、V3以降のネーミングサービスにアクセスする場合。
- V3以降のサーバパッケージがインストールされているマシンから、以下のいずれかのネーミングサービスにアクセスする場合。
  - V2以前のサーバパッケージがインストールされているマシンのネーミングサービス
  - V3以降のサーバパッケージがインストールされているマシンの拡張機能を使用しない設定で初期化されているネーミングサービス

対処方法

以下に示す2つのオペレーションを実行してください。

1. isinit／odadminコマンドでネーミングサービスの設定後、odsetnsコマンドでネーミングサービスを再登録します。odsetnsコマンドについては、「ポイント」を参照してください。
2. nsconfigファイルのパラメタ“ext_intf”に“no”を指定します。

ポイント

ネーミングサービスの拡張機能とは、CosNaming::NamingContextExtインタフェースで提供される以下の4つのオペレーションです。

- to_string
- to_name
- to_url
- resolve_str

odsetnsコマンドについて、以下に説明します。

形式

```
odsetns [ -l | -r -h host -p port] [ -L code ] [ -M system ]
```

オプションとパラメタ

-l

ローカルホストのネーミングサービスを使用します。

-r

リモートホストのネーミングサービスを使用します。

-h host

リモートホストのホスト名を指定します。

-p port

リモートホストのポート番号を指定します。

-L code

日本語を使用する場合、データベースに格納するコード系としてSJIS、EUCのどちらかを指定します。

-M system Solaris32/64

拡張システムを生成した場合に、運用の対象となるシステム名systemを指定します。本オプションを省略すると、デフォルトシステムでの運用が対象となります。

注意事項

- 本コマンド実行時は、CORBAサービスが起動されている必要があります。
- 本コマンド実行時は、ネーミングサービスが停止されている必要があります。
- 本コマンドは、isinit／odadminコマンドを実行するごとに実行する必要があります。

メッセージ

odsetnsコマンドで起動時に表示されるメッセージと意味を以下に示します。以下以外のエラーが発生した場合は、技術員に連絡してください。

```
odsetns : error occured code = 101
odsetns : reason = IDL:FJ/ImplementationBusy:1.0
```

［対処］
ネーミングサービスが起動されています。ネーミングサービスを停止後、再度本コマンドを実行してください。

```
odsetns : error occured code = 1
odsetns : reason = IDL:CORBA/StExcep/UNKNOWN:1.0
```

［対処］
CORBAサービスが起動されていません。CORBAサービスを起動後、再度本コマンドを実行してください。

(3) 旧バージョンのホスト上のネーミングサービスのリンク方法

内容

V2以前のネーミングサービスをV3以降のネーミングサービスにリンクする機能を提供します。

対処方法

OD_or_admコマンドを以下の形式で実行してください。

形式

```
OD_or_adm -c IDL:CosNaming/NamingContext:1.0 -h host -p port -r context.root -n name [-L code]
```

オプションとパラメタ

以下のパラメタ以外は、上記の例のとおり指定してください。

-h host

リンクさせたいネーミングサービスが存在するホスト名を指定します。

-p port

リンクさせたいネーミングサービスが存在するポート番号を指定します。

-n name

登録したいバインディング名を指定します。

-L code

日本語を使用する場合は、データベースに格納するコード系としてSJIS、EUCのどちらかを指定します。

付録G　旧互換機能

旧バージョンに対する互換用の機能について説明します。

G.1　COM/CORBA連携プログラミング

パソコン側のVisual Basicのクライアントアプリケーションが、OLE2インタフェースを使用して、CORBA準拠のサーバアプリケーションと連携する方法について説明します。

注意

- ユーザインタフェースとしてVisual Basicを使用する場合、C(C++)言語で作成したDLLを呼び出す方法を推奨します。
- Windows64
  COM/CORBA連携プログラミングは、使用できません。

G.1.1　OLE-CORBAゲートウェイとは

パソコン側のクライアントアプリケーションを以降OLEクライアントと呼びます。本機能を実現するために、OLE-CORBAゲートウェイ機能がパソコン上に提供されているOLEサーバを利用します。
OLE-CORBAゲートウェイ機能は、OLEクライアントからの要求された情報を元にCORBAのサーバアプリケーションを起動し、その結果をOLEクライアントに通知します。OLE-CORBAゲートウェイ機能は、以下の3つの機能を提供しています。

(1) GetObject関数

GetObject関数は、CORBAのサーバアプリケーションのオブジェクトリファレンスを検索し、その結果をクライアントに通知します。

(2) CreateType関数

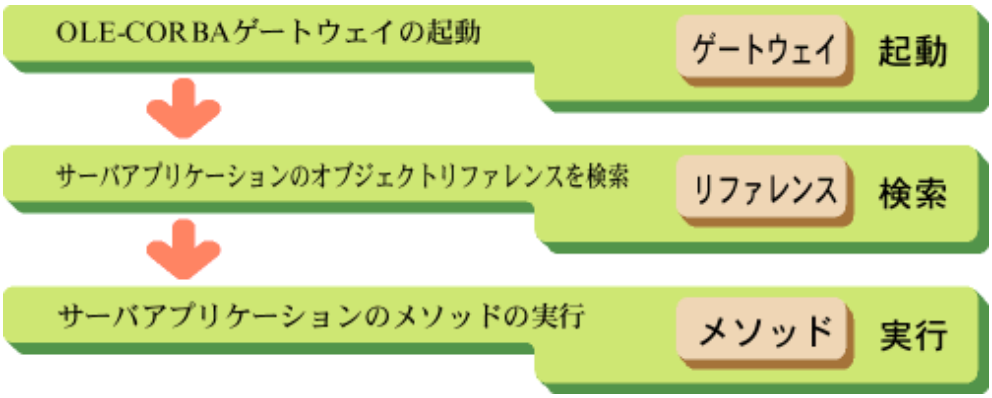
CORBAで規定されているデータ型とVisual Basicで規定されているデータ型は異なるものがあります。Visual BasicとCORBA共通のデータ型については、そのまま受け渡すことができますが、そうでない場合には、OLE-CORBAゲートウェイ機能で提供されているCreateType関数またはCreateTypeById関数を使用して、そのデータ型のオブジェクトリファレンスを検索し、そのオブジェクト内のメンバにアクセスする必要があります。CreateType関数は、サーバアプリケーションのパラメタの名前を元にデータ型のオブジェクトリファレンスを検索する機能です。

(3) CreateTypeById関数

CreateTypeById関数は、サーバアプリケーションのパラメタのRepositoryIDを元にデータ型のオブジェクトリファレンスを検索する機能です。

G.1.2　OLEクライアントの処理

OLEクライアントの処理の流れを以下に示します。



G.1.2.1 OLE-CORBAゲートウェイの起動

OLE-CORBAゲートウェイは、CORBA.Factoryという名前でパソコン上のRegistoryに登録されています。OLE-CORBAゲートウェイを起動するためには、Visual Basicが標準提供しているCreateObject関数を使用し、そのパラメタとしてCORBA.Factoryを指定します。

G.1.2.2 サーバアプリケーションのオブジェクトリファレンスの検索

サーバアプリケーションのオブジェクトリファレンスを検索するために、GetObject関数を使用し、そのパラメタとしてサーバアプリケーションの名前を指定します。
GetObject関数では、パラメタで指定されたサーバアプリケーション名をネーミングサービスから検索後、オブジェクトリファレンスをクライアントに通知します。サーバのアプリケーション名を指定する場合は、ネーミングサービスに登録したオブジェクト名を指定します。なお、サーバオブジェクトのIDL定義がインタフェースリポジトリに登録されている必要があります。

G.1.2.3 メソッドの実行

(1) 実行時バインディング

GetObject関数で検索したオブジェクトのメソッドを呼び出すことにより、サーバアプリケーションが持っているメソッドを呼び出すことができます。以下にクライアントのVisual Basicでのプログラミング例を示します。

```
Dim GW As Object                                  // OLE-CORBAゲートウェイのインスタンス
Dim calculator As Object                          // サーバのオブジェクトリファレンス
Dim a As Long                                     // 入力用変数
Dim b As Long                                     // 入力用変数
Dim exp As Object                                 // エラー設定用
Dim res As Variant                                // 返り値用変数

Set GW = CreateObject("CORBA.Factory")            // OLE-CORBAゲートウェイの起動
Set calculator = GW.GetObject("ODdemo::calculator") // CORBAオブジェクトの生成
a = Text1.Text                                    // 画面に入力されたデータを代入
b = Text2.Text                                    // 画面に入力されたデータを代入
Set res = calculator.calculate(a, b, exp)         // メソッドの呼び出し
```

(2) 事前バインディング

CORBA.Factoryオブジェクトを格納する変数をクラス型で宣言することにより、Visual Basicの事前バインディングが有効になります。事前バインディングは、従来の実行時バインディングと比べVisual Basicプログラム実行時にCORBAクラスを読み込まないため、性能向上が図れます。Visual Basicのアプリケーション開発時に、事前バインディングを使用するには、Visual Basicの“プロジェクト”メニューの“参照設定”を選択し、“ODOLE 1.0 Type Library”を有効にしてください。以下に事前バインディングのプログラミング例を示します。

```
Dim Factory as CORBAFactory

Set Factory = CreateObject("CORBA.Factory")
```

G.1.3 サーバアプリケーションへのパラメタ受渡し

CORBAで規定されているデータ型とVisual Basicで規定されているデータ型は異なるものがあります。Visual BasicとCORBA共通のデータ型については、そのまま受け渡すことができます。そうでない場合には、OLEサーバで提供されているCreateType関数またはCreateTypeById関数を使用して、そのデータ型のオブジェクトリファレンスを検索し、そのオブジェクト内のメンバにアクセスします。

G.1.4 基本データ型の指定方法

以下にCORBAとVisual Basicで提供しているデータ型と、Visual Basicでの指定方法を示します。

CORBAデータ型			Visual Basicでの指定方法
基本データ型	整数型	long	Long
		short	Integer
		unsigned long	Long
		unsigned short	Long
	浮動小数点型	float	Single
		double	Double
	文字型	char	Integer
	オクテッド型	octet	Integer
	ブーリアン型	boolean	Integer
	文字列型	string	“G.1.4.1 文字列型”参照
	列挙型	enum	“G.1.4.2 列挙型”参照
	any型	any	“G.1.4.3 any型”参照
シーケンス型		sequence	“G.1.5.1 シーケンス型”参照
構造体		struct	“G.1.5.2 構造体”参照
共用体		union	“G.1.5.3 共用体”参照
配列		array	“G.1.5.4 配列”参照

注意

データ型の注意事項

- unsigned shortを持つサーバアプリケーションを呼び出す場合、unsigned shortの範囲で指定します。マイナス値や65536以上の値は、指定できません。
- unsigned longを持つサーバアプリケーションを呼び出す場合、マイナス値は指定できません。
- charやoctetを持つサーバアプリケーションを呼び出す場合、charの範囲で指定します。マイナス値や256以上の値は、指定できません。
- booleanを持つサーバアプリケーションを呼び出す場合、0と1以外の値は指定できません。
- any型を持つサーバアプリケーションを呼び出す場合、変数をObject型で宣言し、データ型のインスタンスをCreateType関数／CreateTypeById関数で作成し、valueやtypeはインスタンスのプロパティとして扱います。

・ objref型／TypeCode型／配列型をパラメタとする場合、およびany型／シーケンス型／構造体／共用体のメンバにstring, objref, struct, union, sequence, arrayを持つ場合は、十分なメモリを確保する必要があります。

Visual Basicでのアプリケーション作成時の注意事項

(1) メモリの解放方法

Visual Basicでのオブジェクトおよびout／inoutパラメタで使用した領域の解放方法を以下に示します。

・ CORBAのオブジェクト(CreateTypeしたデータも含む)は、以下のように解放します。

```
Set Ap = Nothing
```

Ap:解放したいオブジェクト

・ 配列の宣言は固定数ではなく可変(動的)で宣言し、Erase文で解放します。

・ 文字列は、NULL文字を設定することで解放します。

```
str = ""
```

(2) 配列(sequence型含む)のパラメタの受渡し

配列(sequence型含む)をパラメタとして受け渡す場合、以下の手順で行います。配列宣言でVisual Basic表現とIDL宣言で要素並びに差異があります。

```
Visual Basic  Dim a(2,3) As Long
IDL定義       long a(4,3)
```

・ inoutで要素数が変更される場合は、動的宣言を行い、クライアントで以下のように再配置してデータを送付します。

```
ReDim abc(n1,n2,n3) As Long
```

・ サーバから結果を受け取った際、データの取り出し前にReDim Preserve abc(m1,m2m3) As Longを発行し、内容を保持した状態で型変換を行い、データにアクセスします。

なお、inパラメタに指定した領域で、値が未設定域である場合は、0が渡されます。

G.1.4.1 文字列型

固定長、可変長の文字列は、それぞれ、長さ指定のあるStringと、長さ指定のないStringを使用します。なお、固定長の文字列を使用する場合は、IDL定義に“使用する文字数+1”の値を指定する必要があります。
文字列のVisual Basicでの使用例を以下に示します。

IDL言語

```
typedef string<11>    sten;    // 固定長(10文字)のString
typedef string        sinf;    // 可変長のString
```

Visual Basic

```
Dim sten As String * 11        // 固定長(10文字)のString
Dim sinf As String             // 可変長のString
```

G.1.4.2 列挙型

列挙型はそれぞれの要素を定数として宣言します。

列挙型のVisual Basicでの使用例を以下に示します。

IDL言語

```
typedef enum EM {red,green,blue}Foo;
```

Visual Basic

```
Global Const red   = 0    // 各要素を定数として宣言
Global Const green = 1    // 各要素を定数として宣言
Global Const blue  = 2    // 各要素を定数として宣言
```

G.1.4.3 any型

any型を持つサーバアプリケーションを呼び出す場合は、変数をObject型で宣言し、データ型のオブジェクトリファレンスをCreateType関数／CreateTypeById関数で検索し、value／typeはそのオブジェクトのプロパティとして扱います。any型データのTypeCodeプロパティに値を設定する場合は、サーバにTypeCodeを返すオペレーションを用意し、返された値を設定します。valueプロパティには、データ型に応じた方法で値を設定します。

(1) any型データにデータを設定する場合

any型データのVisual Basicで設定する使用例を以下に示します。

IDL言語

```
module mod {
    typedef  any  typeany;     // any型
    interface intf {
        TypeCode GetTypeCode();// TypeCodeプロパティに設定するTypeCodeを返す関数
        any op( in any a );    // any型のパラメタを持つop関数
                               // any型の復帰値
    };
};
```

Visual Basic

```
Dim factory As Object                          // OLE-CORBAゲートウェイのインスタンス
Dim obj As Object                              // サーバのオブジェクトリファレンス
Set factory = CreateObject("CORBA.Factory")    // OLE-CORBAゲートウェイの起動
Set obj = factory.GetObject("example")         // オブジェクトの検索

Dim anydata As Object                          // データ型のリファレンス(パラメタ受渡し用)
Dim anydata1 As Object                         // データ型のリファレンス(復帰値用)
Dim obj3 As Object                             // データ型のリファレンス(TypeCode用)

Set anydata =factory.CreateType(obj, "typeany") // データ型のリファレンスの検索
Set obj3= obj.GetTypeCode()                    // TypeCodeオブジェクトの獲得
anydata.TypeCode = obj3                        // TypeCodeの設定
anydata.Value = &H80000000                     // データの設定

anydata1 =obj.op(anydata)                      // op関数の呼び出し
```

(2) any型データからデータを取り出す場合

any型データから値を取り出す場合は、any型データのTypeCodeメンバに対してkind関数を呼び出し、データの型を取り出します。取り出したデータ型に対する値をvalueメンバから取り出します。

any型データのVisual Basicでの使用例を以下に示します。

Visual Basic

```
anydata = obj.op(anydata)              // op関数の呼び出し

kind = anydata.TypeCode.kind           // TypeCodeの取り出し
Select Case kind
Case 2          'short                  // データ型がshortの処理
s = anydata.Value
Case 15         'struct                 // データ型がstructの処理
l = anydata.Value.d1
str = anydata.Value.d2
Case 16  'union                         // データ型がunionの処理
dic = anydata.Value.UNION_d
    Select Case dic
    Case 1
        l = anydata.Value.d1
    Case 2
        str = anydata.Value.d2
    End Select
End Select
```

G.1.5 その他のデータ型の指定方法

その他のデータ型については、CreateType関数またはCreateTypeById関数を使用して構造体等のデータ型のインスタンスを作成し、インスタンス内のメンバにアクセスするようにしなければなりません。なお、CreateType関数では、アクセスしたいデータ型の名前を指定しますが、CreateTypeById関数では、データ型のリポジトリIDを指定します。

G.1.5.1 シーケンス型

シーケンスは1次元の配列として扱います。なお、データ型により、定義が異なります。

(1) シーケンスがany型以外の基本データ型の場合

シーケンスが固定長の場合は、1次元の配列をIDLで指定した個数分、獲得します。

シーケンスがlong型で宣言された場合のVisual Basicでの使用例を以下に示します。

IDL言語

```
typedef sequence<long,10> vec10;       // 10個のlong型
```

Visual Basic

```
Dim vec10(10) As long                  // 10個のlong型を宣言
vec10(0) = 100                         // 各メンバにデータを代入
vec10(1) = 101                         // 各メンバにデータを代入
...
vec10(9) = 109                         // 各メンバにデータを代入
```

(2) シーケンスが構造体、共用体、any型、オブジェクト、またはTypeCodeの場合

シーケンスが構造体、共用体、any型、オブジェクト、またはTypeCodeから構成される場合は、シーケンスのメンバをCreateType関数／CreateTypeById関数でオブジェクトのリファレンスを検索し、その結果をObject型の配列に格納します。

シーケンスがstruct型で宣言された場合のVisual Basicでの使用例を以下に示します。

IDL言語

```
typedef struct STR {                        // aとbを持つstruct
    long a,b;
} Foo;
typedef sequence<Foo,10> FooSeq;            // aとbを持つstruct10個のシーケンス
```

Visual Basic

```
Dim FooSeq(10) As Object                    // 10個のObject型を宣言

FooSeq(0) = factory.CreateType( obj, "Foo" ) // データ型のリファレンスを10個分検索
FooSeq(1) = factory.CreateType( obj, "Foo" )
...
```

G.1.5.2 構造体

構造体に値を設定するには、CreateType関数／CreateTypeById関数でオブジェクトのリファレンスを検索し、その結果をObject型変数に格納します。オブジェクトのプロパティに値を設定することにより、構造体のメンバに値を設定することができます。

構造体のVisual Basicでの使用例を以下に示します。

IDL言語

```
interface example5 {
    typedef struct STR {        // long型aとbを持つstruct
        long a,b;
    }Foo;
    ...
};
```

Visual Basic

```
Dim str1 As Object                          // データ型structのリファレンス

Set str1 = factory.CreateType(obj, "Foo")   // データ型のリファレンスの検索
str1.a = 100                                // aにデータを代入
str1.b = -50                                // bにデータを代入
```

G.1.5.3 共用体

共用体に値を設定するには、CreateType関数／CreateTypeById関数でオブジェクトのリファレンスを検索し、その結果をObject型変数に格納します。オブジェクトのプロパティに設定することにより、共用体のメンバに値を設定することができます。プロパティのUNION_dは、データの種別を格納した弁別情報です。

共用体のVisual Basicでの使用例を以下に示します。

IDL言語

```
interface example6 {
    typedef union UNI switch(long) {   // long、float、charを持つ共用体
        case 1 : long x;
        case 2 : float y;
        default: char z;
    } Foo;
    short set(in Foo udata);           // 共用体のパラメタを持つset関数、
                                       // 復帰値short型
    ...
} ;
```

Visual Basic

```
Dim uni1 As Object                          // データ型のリファレンス
Dim discriminator As Variant                // 弁別情報格納変数
Dim ret As Integer                          // 復帰値格納変数
Dim ans As Variant                          // 共用体データ格納変数

Set uni1 = factory.CreateType(obj, "Foo")   // データ型のリファレンスの検索
uni1.x = 100                                // 共用体にデータを代入
ret = obj.set(uni1)                         // set関数の呼び出し

discriminator = uni1.UNION_d                // 弁別情報の格納
Select Case discriminator                   // 弁別情報によるデータの判定
Case 1                                      // long型の処理
    ans = uni1.x
Case 2
    ans = uni1.y
Case Else                                   // char型の処理
    ans = uni1.z
```

G.1.5.4 配列

配列は、Visual Basicの配列を使用します。ただし、配列が構造体、共用体、any型、オブジェクト、TypeCodeの場合は、CreateType関数／CreateTypeById関数でインスタンスを作成し、その結果をObject型変数に格納します。

配列のVisual Basicでの使用例を以下に示します。

IDL言語

```
typedef long a[4][5];
interface example8 {
    typedef struct STR {
        long a,b;
    } Foo;
    typedef Foo FooArr[3][4];
    ...
} ;
```

Visual Basic

```
Dim a(4,3) As long
Dim FooArr(3,2) As Object
Dim factory As Object
Dim obj As Object
```

```
a(0,0) = 100
a(1,0) = 101
...
a(4,3) = 119

Set factory = CreateObject("CORBA.Factory")
Set obj = factory.GetObject("example8")
FooArr(0,0) = factory.CreateType(obj, "Foo")
FooArr(1,0) = factory.CreateType(obj, "Foo")
...
```

G.1.6 サーバの処理結果

復帰値にサーバの処理結果が通知され、その型が構造体、共用体、any、オブジェクト、TypeCodeの場合、Object型変数で受け取ります。なお、その他の型の場合、復帰値はそれぞれの型の値となります。

IDL言語

```
interface example13 {
    typedef struct STR {
        long a,b;
    } Foo;
    long      ope1();
    string    ope2();
    Foo       ope3();
} ;
```

Visual Basic

```
Dim factory As Object
Dim ex13 As Object
Dim ret1 As long
Dim ret2 As string
Dim ret3 As Object
Dim sa As long

Set factory = CreateObject("CORBA.Factory")
Set ex13 = factory.GetObject("example13")
ret1 = ex13.op1
ret2 = ex13.op2
Set ret3 = ex13.op3
sa = ret3.a
```

G.1.7 属性の設定・参照

属性の値を設定・参照するには、GetObject関数で作成したインスタンスのプロパティに対して値の設定・参照を行います。ただし、readonlyの属性に対しては、値の参照しか行えません。

IDL言語

```
interface example1 {
    attribute long              att1;
    readonly attribute string   att2;
} ;
```

Visual Basic

```
Dim factory As Object
Dim obj As Object
Dim att1 As long
Dim att2 As string

Set factory = CreateObject( "CORBA.Factory" )
Set obj = factory.GetObject( "example1" )
obj.att1 = 100&
att1 = obj.att1
att2 = obj.att2
```

G.1.8 例外

例外を扱う場合は、変数をオブジェクト型で宣言し、この変数をオペレーションの実行時のパラメタの最後に追加します。例外が発生した場合、この変数に例外情報を保持した例外オブジェクトのリファレンスが設定されます。

このオブジェクトのEX_majorメンバには、以下の値が設定されます。

- 2:システム例外
- 1:ユーザ例外
- 0:正常終了

システム例外が発生した場合は、オブジェクトのリポジトリIDメンバにシステム例外の識別情報が文字列として設定されます。システム例外を識別するために一覧に示すシステム例外と文字列を比較します。

例外コードの一覧とその意味を以下に示します。例外の意味については、“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。

例外情報	例外コード
BAD_CONTEXT	IDL:CORBA/StExcep/BAD_CONTEXT:1.0
BAD_INV_ORDER	IDL:CORBA/StExcep/BAD_INV_ORDER:1.0
BAD_OPERATION	IDL:CORBA/StExcep/BAD_OPERATION:1.0
BAD_PARAM	IDL:CORBA/StExcep/BAD_PARAM:1.0
BAD_QOS	IDL:CORBA/StExcep/BAD_QOS:1.0
BAD_TYPECODE	IDL:CORBA/StExcep/BAD_TYPECODE:1.0
CODESET_INCOMPATIBLE	IDL:CORBA/StExcep/CODESET_INCOMPATIBLE:1.0
COMM_FAILURE	IDL:CORBA/StExcep/COMM_FAILURE:1.0
CONTEXT	IDL:CORBA/StExcep/CONTEXT:1.0
DATA_CONVERSION	IDL:CORBA/StExcep/DATA_CONVERSION:1.0
FREE_MEM	IDL:CORBA/StExcep/FREE_MEM:1.0
IMP_LIMIT	IDL:CORBA/StExcep/IMP_LIMIT:1.0
INITIALIZE	IDL:CORBA/StExcep/INITIALIZE:1.0
INTERNAL	IDL:CORBA/StExcep/INTERNAL:1.0
INTF_REPOS	IDL:CORBA/StExcep/INTF_REPOS:1.0
INV_FLAG	IDL:CORBA/StExcep/INV_FLAG:1.0
INV_IDENT	IDL:CORBA/StExcep/INV_IDENT:1.0
INV_OBJREF	IDL:CORBA/StExcep/INV_OBJREF:1.0
INV_POLICY	IDL:CORBA/StExcep/INV_POLICY:1.0

例外情報	例外コード
MARSHAL	IDL:CORBA/StExcep/MARSHAL:1.0
NO_IMPLEMENT	IDL:CORBA/StExcep/NO_IMPLEMENT:1.0
NO_MEMORY	IDL:CORBA/StExcep/NO_MEMORY:1.0
NO_PERMISSION	IDL:CORBA/StExcep/NO_PERMISSION:1.0
NO_RESOURCES	IDL:CORBA/StExcep/NO_RESOURCES:1.0
NO_RESPONSE	IDL:CORBA/StExcep/NO_RESPONSE:1.0
OBJ_ADAPTER	IDL:CORBA/StExcep/OBJ_ADAPTER:1.0
PERSIST_STORE	IDL:CORBA/StExcep/PERSIST_STORE:1.0
REBIND	IDL:CORBA/StExcep/REBIND:1.0
TIMEOUT	IDL:CORBA/StExcep/TIMEOUT:1.0
TRANSIENT	IDL:CORBA/StExcep/TRANSIENT:1.0
UNKNOWN	IDL:CORBA/StExcep/UNKNOWN:1.0
INVALID_TRANSACTION	IDL:CORBA/StExcep/INVALID_TRANSACTION:1.0
TRANSACTION_MODE	IDL:CORBA/StExcep/TRANSACTION_MODE:1.0
TRANSACTION_REQUIRED	IDL:CORBA/StExcep/TRANSACTION_REQUIRED:1.0
TRANSACTION_ROLLEDBACK	IDL:CORBA/StExcep/TRANSACTION_ROLLEDBACK:1.0
TRANSACTION_UNAVAILABLE	IDL:CORBA/StExcep/TRANSACTION_UNAVAILABLE:1.0

システム例外発生時は、EX_minorメンバにマイナーコードが設定されます。マイナーコードの値については、“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。

ユーザ例外が発生した場合は、IDLで定義した例外情報のリポジトリIDが例外オブジェクトのリポジトリIDメンバに設定されます。

リポジトリIDの形式を以下に示します。

```
IDL: 例外の名前:1.0
```

また、例外の詳細情報はIDLで定義した名前でオブジェクトのメンバ変数としてアクセスできます。

例外のVisual Basicでの使用例を以下に示します。

IDL言語

```
exception foo {          // ユーザ例外の定義
    long dummy;
} ;
interface example9 {
    // long型のパラメタを持つope関数、復帰値なし
    void ope(in long arg) raises(foo);
} ;
```

Visual Basic

```
Global Const CORBA_NO_EXCEPTION     = 0  // 正常終了
Global Const CORBA_USER_EXCEPTION   = 1  // ユーザ例外
Global Const CORBA_SYSTEM_EXCEPTION = 2  // システム例外

Dim excp As Object                       // 例外情報を格納する変数
```

```
Dim dummy As long                          // ユーザ例外を格納する変数

obj.ope(1000, excp)                        // ope関数の呼び出し
                                           // システム例外の判定およびその処理
if excp.EX_major = CORBA_SYSTEM_EXCEPTION Then
    if excp.EX_RepositoryID = "IDL:CORBA/StExcep/UNKNOWN:1.0" Then
    // UNKNOWNに対応したエラー処理
    else if excp.EX_RepositoryID = "IDL:CORBA/StExcep/BAD_PARAM:1.0" Then
    // BAD_PARAMに対応したエラー処理
        // マイナーコードが設定されている場合
        if excp.EX_minor = 1 Then    // CORBA_Environmentの_minorメンバ
            // マイナーコードに対応した処理
        endif
    .
    .
// ユーザ例外の判定およびその処理
else if excp.EX_major = CORBA_USER_EXCEPTION Then
    if excp.EX_RepositoryID = "IDL:foo:1.0" Then
    dummy = excp.dummy     // ユーザ例外に対応した処理
    endif
endif
```

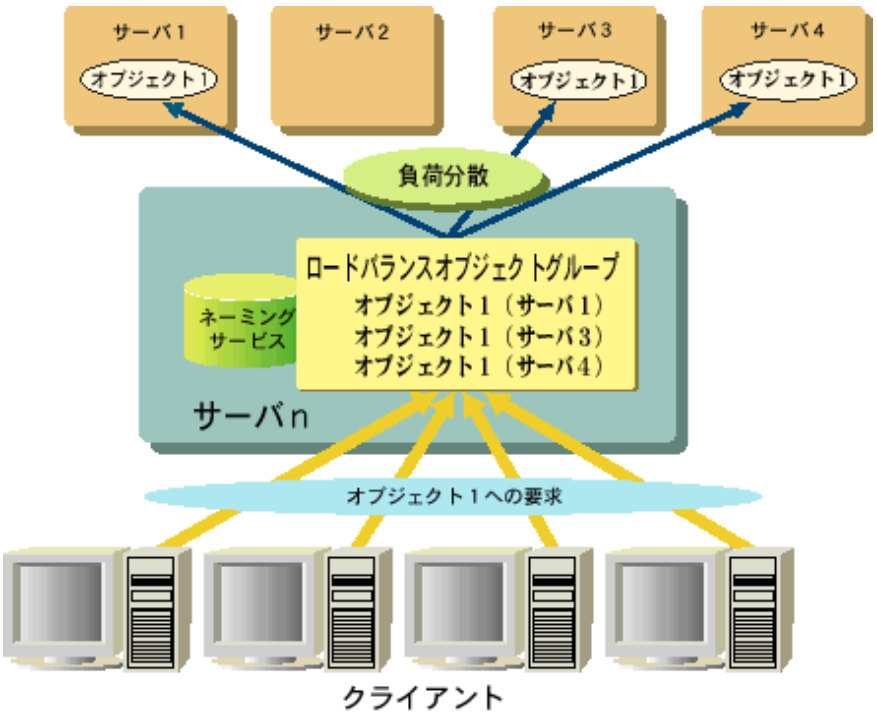
G.2 ロードバランス機能のプログラミング

Windows32 Solaris32 Linux32

ロードバランス機能のプログラミングについて説明します。

G.2.1 ロードバランスの運用

ネーミングサービスと連携して、均一な運用環境での負荷分散を実現する機能を提供しています。
同じサーバオブジェクト(インタフェースが同一で、同じ機能を提供するオブジェクト)を複数のサーバ上に配置し、このオブジェクトに対してクライアントから要求があった場合には、これらのオブジェクトへの負荷が均等になるように調整してオブジェクトリファレンスを返します。



ロードバランスでは、ロードバランス対象のオブジェクトの登録方法により以下の2とおりの運用があります。

固定登録する場合

ロードバランス対象のオブジェクトを、システム環境構築時に1回だけ登録します。ロードバランス運用の対象となるオブジェクトが固定である場合に使用します。

サーバオブジェクトの起動時に毎回登録する場合

ロードバランス対象のオブジェクトを、サーバオブジェクトの起動時に毎回登録します。ロードバランス運用の対象となるオブジェクトが可変である場合に使用します。

G.2.2 ロードバランス機能インタフェース

ロードバランスは、ロードバランスオプションとネーミングサービスに登録されたロードバランスオブジェクトグループの2つのインタフェースを有します。

ロードバランスオプションインタフェース

ネーミングサービスと連携して、ロードバランスオブジェクトグループの登録、削除、検索などを行うための機能を持つインタフェース。

ロードバランスオブジェクトグループインタフェース

ロードバランスオブジェクトグループへのオブジェクトリファレンスの登録、削除、検索などを行うための機能を持つインタフェース

ロードバランスは、以下のインタフェースを提供しています。

```
#include "CosNaming.idl"

module ISOD {
    interface LBG {
        exception NotFound{};
        exception CannotProceed{
            CosNaming::NamingContext    cxt;
            CosNaming::Name             rest_of_name;
        };
        exception CannotProceed2{};
        exception AlreadyBound{};
        exception BadObject{};
        exception OperationBusy{};

        enum LoadBalanceType{ roundrobin };

        typedef sequence<Object>        ObjectList;

        void  bind(
            in Object                   objectref
            ) raises( AlreadyBound, BadObject, CannotProceed2 );

        void  unbind (
            in Object                   objectref
            ) raises( NotFound, BadObject, CannotProceed2 );

        Object  rebind_default (
            in Object                   objectref
            ) raises( CannotProceed, BadObject, CannotProceed2,
                      OperationBusy );

        Object resolve(
            ) raises( NotFound, CannotProceed2 );

        void list(
```

```
        out ObjectList              objectlist
        ) raises( CannotProceed2 );
};

interface LBO {
    exception InvalidName{};
    enum NotFoundReason{ missing_node, not_context, not_object };
    exception NotFound{
        NotFoundReason              why;
        CosNaming::Name             rest_of_name;
    };
    exception CannotProceed{
        CosNaming::NamingContext    cxt;
        CosNaming::Name             rest_of_name;
    };
    exception CannotProceed2{};
    enum DuplicateType{ objectgroup, other };
    exception AlreadyExist{
        DuplicateType               Duplicate_type;
    };
    exception NotEmpty{};
    exception InvalidArgument{};
    exception InvalidType{};
    exception BadObject{};
    exception OperationBusy{};

    enum LoadBalanceType{ roundrobin };
    struct BindingLBG {
        CosNaming::Name             LBG_name;
        LoadBalanceType             LoadBalance_type;
    };
    typedef sequence<BindingLBG>    LBGList;

    LBG create_LBG(
        in CosNaming::NamingContext nc,
        in CosNaming::Name          name,
        in LoadBalanceType          loadbalancetype,
        in Object                   defaultobjref
        ) raises( NotFound, CannotProceed, InvalidName, AlreadyExist,
                  InvalidType, BadObject, OperationBusy );

    LBG resolve_LBG(
        in CosNaming::NamingContext nc,
        in CosNaming::Name          name
        ) raises( NotFound, CannotProceed, InvalidName, OperationBusy );

    Object delete_LBG(
        in CosNaming::NamingContext nc,
        in CosNaming::Name          name
        ) raises( NotFound, CannotProceed, InvalidName, NotEmpty,
                  OperationBusy );

    void list_LBG(
        in CosNaming::NamingContext nc,
        out LBGList                 objectgrouplist
        );

    void  notify_down(
        in string                   HostName
        ) raises( InvalidArgument, CannotProceed2 );

    void  notify_recover(
        in string                   HostName
```

```
            ) raises( InvalidArgument, CannotProceed2 );
    };
};
```

ロードバランスオプションで提供しているインタフェースの機能説明を以下に示します。

インタフェース名	メソッド名	機能説明
LBO	create	オブジェクト名とロードバランスオブジェクトグループのオブジェクトリファレンスを関係付け、ネーミングサービスおよびロードバランスに登録します。
	resolve	オブジェクト名とロードバランスオブジェクトグループのオブジェクトリファレンスを取得します。
	delete	オブジェクト名に関連付けられたロードバランスオブジェクトグループのオブジェクトリファレンスをネーミングサービスおよびロードバランスから削除します。
	list	指定されたネーミングコンテキスト配下に登録されているロードバランスオブジェクトグループのオブジェクトリファレンスの一覧を取得します。
	notify_down	指定されたサーバマシンがダウンしたことを通知します。
	notify_recover	指定されたサーバマシンが復旧したことを通知します。
LBG	bind	オブジェクト名とオブジェクトリファレンスを関連付け、ロードバランスグループに登録します。
	unbind	オブジェクト名とオブジェクトリファレンスをロードバランスオブジェクトグループから削除します。
	rebind_default	ロードバランスオブジェクトグループのデフォルトオブジェクトのオブジェクトリファレンスを変更します。
	list	ロードバランスオブジェクトグループに登録されているオブジェクトのオブジェクトリファレンスの一覧を取得します。

G.2.3 ロードバランスオプションインタフェース

G.2.3.1 ロードバランスオブジェクトグループへのオブジェクトの登録

ロードバランスオブジェクトグループにロードバランスオブジェクトを登録する方法について説明します。

注意

ロードバランスオブジェクトを登録するには、事前にロードバランスオプションの環境を設定する必要があります。設定方法については、“高信頼性システム運用ガイド”の“ロードバランス機能を利用した場合の設計”を参照してください。

(1) C言語のプログラミング例

ORBインタフェースのオブジェクトリファレンスを取り出すCORBA_ORB_resolve_initial_references()により、ロードバランスオプションとネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとして、LBO_ObjectId_LoadBalancingOption, CORBA_ORB_ObjectId_NameServiceを指定します。
次に、シーケンスCosNaming_Nameのバッファポインタに CosNaming_NameComponentの配列のポインタを設定し、次に取り出したいロードバランスオブジェクトグループ名を指定します。今回の例では、ルートネーミングコンテキスト配下に登録されたロードバランスオブジェクトグループ名“LBG_A”にオブジェクトを登録する例を示します。
次に、ロードバランスオプションのISOD_LBO_resolve_LBG()を用いて、ロードバランスオブジェクトグループのオブジェクトリファレンスを取り出します。
最後にロードバランスオブジェクトグループに登録したいオブジェクトリファレンスを指定して、ISOD_LBG_bind()を用いて、ロードバランスオブジェクトグループにオブジェクトリファレンスを登録します。

```
CORBA_ORB                       orb;
CosNaming_NamingContext         cos_naming;
ISOD_LBO                        lbo;
ISOD_LBG                        lbg;
CosNaming_Name                  name;
CosNaming_NameComponent         name_component[1];
CORBA_Environment               env;

CORBA_Object                    obj;

orb = CORBA_ORB_init(&current_argc, argv, FJ_OM_ORBid, &env );

/* ネーミングサービスのオブジェクトリファレンス獲得 */
cos_naming = CORBA_ORB_resolve_initial_references(
                orb,
                CORBA_ORB_ObjectId_NameService,
                &env );

/* ロードバランスオプションのオブジェクトリファレンス獲得 */
lbo = CORBA_ORB_resolve_initial_references(
                orb,
                LBO_ObjectId_LoadBalancingOption,
                &env );

/* LBG_Aのシーケンス領域の設定 */
name._length = name._maximum = 1;
name._buffer = name_component;
name_component[0].id = "LBG_A";
name_component[0].kind = "";

/* ロードバランスオブジェクトグループの取得 */
lbg = ISOD_LBO_resolve_LBG(
      lbo,
      cos_naming,
      &name,
      &env );

obj = ...; /* 何らかの方法でObjectを作成し、取得します。*/

/* ロードバランスオブジェクトグループにオブジェクトリファレンスを登録 */
ISOD_LBG_bind(
      lbg,
      obj,
      &env );
```

(2)C++言語のプログラミング例

ORBインタフェースのオブジェクトリファレンスを取り出すCORBA::ORB::resolve_initial_references()によりロードバランスオプションとネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとして、LBO_ObjectId_LoadBalancingOption, CORBA_ORB_ObjectId_NameServiceを指定します。
次に、シーケンスCosNaming::Nameのバッファポインタ(CosNaming::NameComponent_varの配列)の領域をCosNaming::Name::allocbuf()を使って獲得し、次に取り出したいロードバランスオブジェクトグループ名前指定します。その後、上記ポインタをパラメタに指定し、演算子newを使ってシーケンスを作成します。今回の例では、ルートネーミングコンテキスト配下に登録されたロードバランスオブジェクトグループ名“LBG_A”にオブジェクトを登録する例を示します。
次に、ロードバランスオプションのISOD::LBO::resolve_LBG()を用いて、ロードバランスオブジェクトグループのオブジェクトリファレンスを取り出します。
最後にロードバランスオブジェクトグループに登録したいオブジェクトリファレンスを指定して、ISOD::LBG::bind()を用いて、ロードバランスオブジェクトグループにオブジェクトリファレンスを登録します。

```
CORBA::ORB_ptr                  orb;
CosNaming::Name_ptr             name;
```

```
CosNaming::NameComponent_var   *name_component;
CORBA::Environment             env;

CORBA::Object_ptr              obj;

try {
    orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

    // ネーミングサービスのオブジェクトリファレンス獲得
    CORBA::Object_ptr
    obj = orb->resolve_initial_references(
                CORBA_ORB_ObjectId_NameService, env );

    CosNaming::NamingContext_ptr
    NamingContext_obj = CosNaming::NamingContext::_narrow( obj );
    CORBA::release( obj );

    // ロードバランスオプションのオブジェクトリファレンス獲得
    obj = orb->resolve_initial_references(
                LBO_ObjectId_LoadBalancingOption, env );

    ISOD::LBO_ptr
    LBO_obj = ISOD::LBO::_narrow( obj );
    CORBA::release( obj );

    // LBG_Aのシーケンス領域の設定
    name_component = CosNaming::Name::allocbuf(1);
    name_component[0]->id = (const CORBA::Char *)"LBG_A";
    name_component[0]->kind = (const CORBA::Char *)"";
    name = new CosNaming::Name(1, 1, name_component,CORBA_TRUE);

    // ロードバランスオブジェクトグループの取得
    ISOD::LBG_ptr
    LBG_obj = LBO_obj->resolve_LBG( NamingContext_obj, *name, env );
    delete name;

    obj = ...;  // 何らかの方法でObjectを作成し、取得します。

    // ロードバランスオブジェクトグループにオブジェクトリファレンスを登録
    LBG_obj->bind( obj, env );

    CORBA::release( NamingContext_obj );
    CORBA::release( LBO_obj );
    CORBA::release( LBG_obj );
}
catch ( CORBA::Exception &e ){
    ...  // エラー処理
}
```

(3)Java言語のプログラム例

ORBインタフェースのオブジェクトリファレンスを取り出すorg.omg.CORBA.ORB.resolve_initial_references()によりロードバランスオプションとネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとして、“FJ_LoadBalancingOption”,“NameService”を指定します。
次に、org.omg.CosNaming.NameComponentに取り出したいロードバランスオブジェクトグループ名を指定します。今回の例では、ルートネーミングコンテキスト配下に登録されたロードバランスオブジェクトグループ名“LBG_A”にオブジェクトを登録する例を示します。
次に、ロードバランスオプションのcom.fujitsu.ObjectDirector.ISOD.LBO.resolve_LBG()を用いて、ロードバランスオブジェクトグループのオブジェクトリファレンスを取り出します。
最後にロードバランスオブジェクトグループに登録したいオブジェクトリファレンスを指定して、com.fujitsu.ObjectDirector.ISOD.LBG.bind()を用いて、ロードバランスオブジェクトグループにオブジェクトリファレンスを登録します。

```
try {
    org.omg.CORBA.ORB orb =  org.omg.CORBA.ORB.init(args,null);

    // ネーミングサービスのオブジェクトリファレンス獲得
    org.omg.CORBA.Object obj =
          orb.resolve_initial_references("NameService");
    org.omg.CosNaming.NamingContextExt cos_naming =
          org.omg.CosNaming.NamingContextExtHelper.narrow(obj);

    // ロードバランスオプションのオブジェクトリファレンス獲得
    obj = orb.resolve_initial_references("FJ_LoadBalancingOption");
    com.fujitsu.ObjectDirector.ISOD.LBO lbo =
          com.fujitsu.ObjectDirector.ISOD.LBOHelper.narrow(obj);

    // LBG_Aのシーケンス領域の設定
    org.omg.CosNaming.NameComponent [] name =
          new org.omg.CosNaming.NameComponent[1];
    name[0] = new org.omg.CosNaming.NameComponent("LBG_A", "");

    // ロードバランスオブジェクトグループの取得
    com.fujitsu.ObjectDirector.ISOD.LBG lbg = lbo.resolve_LBG(cos_naming, name);

    obj = ...; // 何らかの方法でObjectを作成し、取得します。

    // ロードバランスオブジェクトグループにオブジェクトリファレンスを登録
    lbg.bind(obj);
}
catch( Exception e ){
    ...  // エラー処理
}
```

(4)COBOL言語のプログラミング例

ORBインタフェースのオブジェクトリファレンスを取り出すCORBA-ORB-RESOLVE-INITIAL-REFERENCESにより、ロードバランスオプションとネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとして、LBO-OBJECTID、CORBA-ORB-OBJECTID-NAMESERVICEを指定します。
次に、COSNAMING-NAMEおよびCOSNAMING-NAMECOMPONENTに取り出したいロードバランスオブジェクトグループ名を指定します。今回の例では、ルートネーミングコンテキスト配下に登録されたロードバランスオブジェクトグループ名“LBG_A”にオブジェクトを登録する例を示します。
次に、ロードバランスオプションのISOD-LBO-RESOLVE-LBGを用いて、ロードバランスオブジェクトグループのオブジェクトリファレンスを取り出します。
最後にロードバランスオブジェクトグループに登録したいオブジェクトリファレンスを指定して、ISOD-LBG-BINDを用いて、ロードバランスオブジェクトグループにオブジェクトリファレンスを登録します。

```
   IDENTIFICATION DIVISION.
   PROGRAM-ID. "BIND".
   ENVIRONMENT DIVISION.
   CONFIGURATION SECTION.
   SPECIAL-NAMES.
       ARGUMENT-NUMBER      IS  ARG-C
       ARGUMENT-VALUE       IS  ARG-V
       SYMBOLIC CONSTANT
       COPY SYMBOL-CONST IN CORBA.
         .
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   COPY CONST  IN CORBA.
   COPY CONST-LBO IN CORBA.
   01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
```

```
*########## NAMING SETTING PARAMETER #########
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING COSNAMING-NAMINGCONTEXT
                                                                  BY COS-NAMING.
 01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
 01 NAME-A USAGE POINTER.
 01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING COSNAMING-NAMECOMPONENT
                                                                  BY NAME-COMPONENT.
 01 NAME-COMPONENT-A USAGE POINTER.
 01 COPY ISOD-LBO IN CORBA REPLACING ISOD-LBO BY LBO.
 01 COPY ISOD-LBG IN CORBA REPLACING ISOD-LBG BY LBG.

*########## ORB SETTING PARAMETER #########
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
 01 CURRENT-ARG-V.
    02 FILLER OCCURS 6.
       03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_c".
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.

 01 TMP-STRING       USAGE POINTER.
 01 TMP-STRING-BUF  PIC X(50).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY ELEMENT-NUMBER.
*
 PROCEDURE DIVISION.

     ACCEPT CURRENT-ARG-C FROM ARG-C.
     COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
     PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
        IF ARG-COUNT = 1
           MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
        END-IF
        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        CALL "CORBA-STRING-SET" USING
              CURRENT-ARG-V-VALUE (ARG-COUNT)
              STRING-LENGTH
              TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.

     MOVE FUNCTION LENG(FJ-OM-ORB-ID) TO STRING-LENGTH.
     CALL  "CORBA-STRING-SET" USING
           TMP-STRING
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           CURRENT-ARG-C
           CURRENT-ARG-V
           TMP-STRING
           ENV
           ORB.
     CALL "CORBA-FREE" USING TMP-STRING.

* ネーミングサービスのオブジェクトリファレンス獲得
     MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TMP-STRING
           STRING-LENGTH
           CORBA-ORB-OBJECTID-NAMESERVICE.
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
```

```
            ORB
            TMP-STRING
            ENV
            COS-NAMING.
      CALL "CORBA-FREE" USING TMP-STRING.

* ロードバランスオプションのオブジェクトリファレンス獲得
      MOVE FUNCTION LENG ( LBO-OBJECTID ) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
            TMP-STRING
            STRING-LENGTH
            LBO-OBJECTID.
      CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
            ORB
            TMP-STRING
            ENV
            LBO.
      CALL "CORBA-FREE" USING TMP-STRING.

* LBG_Aのシーケンス領域の設定
      MOVE 1 TO SEQ-LENGTH OF NAME.
      MOVE 1 TO SEQ-MAXIMUM OF NAME.
      CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
            SEQ-MAXIMUM OF NAME
            SEQ-BUFFER OF NAME.

      MOVE "LBG_A" TO TMP-STRING-BUF.
      MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
            IDL-ID OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF
            .
      MOVE " " TO TMP-STRING-BUF.
      CALL "CORBA-STRING-SET" USING
            KIND OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.
      MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
      MOVE 1 TO ELEMENT-NUMBER.
      MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            NAME-A
            ELEMENT-NUMBER
            NAME-COMPONENT-A.

* ロードバランスオブジェクトグループの取得
      CALL "ISOD-LBO-RESOLVE-LBG" USING
            LBO
            COS-NAMING
            NAME
            ENV
            LBG.

* 何らかの方法でObjectを作成し、取得します。
      MOVE ...  TO OBJ.

* ロードバランスオブジェクトグループにオブジェクトリファレンスを登録
      CALL "ISOD-LBG-BIND" USING
            LBG
            OBJ
            ENV.
```

```
    EXIT PROGRAM.
END PROGRAM "BIND".
```

(5)OOCOBOL言語のプログラミング例

ORBインタフェースのオブジェクトリファレンスを取り出すCORBA-ORB-RESOLVE-INITIAL-REFERENCESにより、ロードバランスオプションとネーミングサービスのオブジェクトリファレンスを取り出します。パラメタとして、“FJ_LoadBalancingOption”、CORBA-ORB-OBJECTID-NAMESERVICEを指定します。
次に、COSNAMING-NAMEおよびCOSNAMING-NAMECOMPONENTに取り出したいロードバランスオブジェクトグループ名を指定します。今回の例では、ルートネーミングコンテキスト配下に登録されたロードバランスオブジェクトグループ名“LBG_A”にオブジェクトを登録する例を示します。
次に、ロードバランスオプションのISOD-LBO-RESOLVE-LBGを用いて、ロードバランスオブジェクトグループのオブジェクトリファレンスを取り出します。
最後にロードバランスオブジェクトグループに登録したいオブジェクトリファレンスを指定して、ISOD-LBG-BINDを用いて、ロードバランスオブジェクトグループにオブジェクトリファレンスを登録します。

```
 IDENTIFICATION         DIVISION.
 PROGRAM-ID.            "BIND".
*
 ENVIRONMENT            DIVISION.
 CONFIGURATION          SECTION.
 REPOSITORY.
     COPY               CORBA--REP.
     COPY               COSNAMING--REP.
     COPY               NSLBO--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY               CORBA--CONST.
     COPY               COSNAMING--CONST.
     COPY               NSLBO--CONST.
     .
*
 DATA                   DIVISION.
 WORKING-STORAGE        SECTION.
 COPY                   CORBA--COPY.
 COPY                   COSNAMING--COPY.
 COPY                   NSLBO--COPY.
*
 01 ORB                 USAGE OBJECT REFERENCE CORBA-ORB.
 01 OBJ                 USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 NAMING-CONTEXT      USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT      USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NAME-ID             USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND           USAGE OBJECT REFERENCE CORBA-STRING.
 01 LBO                 USAGE OBJECT REFERENCE ISOD-LBO.
 01 LBG                 USAGE OBJECT REFERENCE ISOD-LBG.
*
 PROCEDURE              DIVISION.
*
 DECLARATIVES.
*
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
* エラー処理
     ...
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
```

```
 MAIN SECTION.

     INVOKE CORBA "ORB_INIT"
                         USING     "simple_c"
                                   FJ-OM_ORBID
                         RETURNING ORB.

* ネーミングサービスのオブジェクトリファレンス獲得
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                         USING     CORBA-ORB-OBJECTID_NAMESERVICE
                         RETURNING OBJ.
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                         USING     OBJ
                         RETURNING NAMING-CONTEXT.

*ロードバランスオプションのオブジェクトリファレンス獲得
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                         USING     "FJ_LoadBalancingOption"
                         RETURNING OBJ.
     INVOKE ISOD-LBO "NARROW"
                         USING     OBJ
                         RETURNING LBO.

* LBG_Aのシーケンス領域の設定
     INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH" USING 1 RETURNING NAME.
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING "LBG_A" RETURNING NAME-ID
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING " " RETURNING NAME-KIND
     INVOKE COSNAMING-NAMECOMPONENT "NEW" RETURNING NAME-COMPONENT
     SET IDL-ID OF NAME-COMPONENT TO NAME-ID
     SET KIND OF NAME-COMPONENT TO NAME-KIND
     INVOKE NAME "SET-VALUE" USING 1 NAME-COMPONENT

* ロードバランスオブジェクトグループの取得
     INVOKE LBO "RESOLVE_LBG"
                         USING     NAMING-CONTEXT
                                   NAME
                         RETURNING OBJ.
     INVOKE ISOD-LBG "NARROW"
                         USING     OBJ
                         RETURNING LBG.

* 何らかの方法でObjectを作成し、取得します。
     MOVE ... TO OBJ.

* ロードバランスオブジェクトグループにオブジェクトリファレンスを登録
     INVOKE LBG "BIND"
                         USING     OBJ.
*
     STOP RUN.
*
 END-MAIN.
*
 END PROGRAM "BIND".
```

G.3 アプリケーションの開発(OOCOBOL)

OOCOBOL特有の機能を使用する場合を除いて、COBOLやC++などの別言語での開発を推奨します。

注意

Windows64 Solaris64 Linux64

OOCOBOLは、Windows(R)クライアントアプリケーションで使用できます。

G.3.1 クライアントアプリケーションのプログラミング（静的起動インタフェース）

静的起動インタフェースを使用する場合のクライアントアプリケーションの処理の流れを以下に示します。

G.3.1.1 初期化

CORBAの初期化メソッドCORBA-ORB_INITを呼び出し初期化処理を行います。当メソッドの結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。環境部(ENVIRONMENT DIVISION)およびデータ部(DATA DIVISION)を定義します。ここでは、IDLファイル名を“simple.idl”として説明します。IDLコンパイラが出力する登録集ファイル名は、“simple--rep.cbl”, “simple--const.cbl”,“simple--copy.cbl”となります。

```
 IDENTIFICATION          DIVISION.
 PROGRAM-ID.             "SIMPLE_C"
*
 ENVIRONMENT             DIVISION.
 CONFIGURATION           SECTION.
 REPOSITORY.
* ObjectDirectorの標準登録集（リポジトリ宣言用）
     COPY                CORBA--REP.
* ネーミングサービスの標準登録集（リポジトリ宣言用）
     COPY                COSNAMING--REP.
* IDLコンパイラが出力した登録集（リポジトリ宣言用）
     COPY                SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
* ObjectDirectorの標準登録集（定数宣言用）
     COPY                CORBA--CONST.
* ネーミングサービスの標準登録集（定数宣言用）
     COPY                COSNAMING--CONST.
* IDLコンパイラが出力した登録集（定数宣言用）
     COPY                SIMPLE--CONST.
     .
```

```
*
 DATA                    DIVISION.
 WORKING-STORAGE         SECTION.
 01 EXCEPTION-ID         USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE   PIC   X(50).
 01 API-NAME             PIC   X(50).
 01 ORB                  USAGE OBJECT REFERENCE CORBA-ORB.
*
 PROCEDURE              DIVISION.
*
 DECLARATIVES.
*
 ZEROPARAM-ERR SECTION.
     USE EXCEPTION ODDEMO-CALCULATOR-ZEROPARAM.
     DISPLAY "ZEROPARAM Exception: " API-NAME.
     EXIT PROGRAM.
 END-ZEROPARAM-ERR.
*
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::UserException: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
 MAIN SECTION.
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
                         USING     "SIMPLE_C"
                                   FJ-OM_ORBID
                         RETURNING ORB.
```

例外処理については、“G.3.3 クライアントアプリケーションの例外処理”を参照してください。

G.3.1.2 ネーミングサービスのオブジェクトリファレンスの獲得

実行するオブジェクトをネーミングサービスから検索するために、ネーミングサービスのオブジェクトリファレンスが必要となります。CORBAインタフェースのオブジェクトリファレンスの取出しメソッドCORBA-ORB-RESOLVE_INITIAL_REFERENCESによりネーミングサービスのオブジェクトリファレンスを取り出します。このとき、CORBA-ORB-OBJECTID_NAMESERVICEをメソッドのパラメタとして指定します。なお、この定数は、各ORBベンダーが提供しているインクルードファイル等で定義されています。

```
  DATA DIVISION.
  WORKING-STORAGE         SECTION.
  01 OBJ                  USAGE OBJECT REFERENCE CORBA-OBJECT.

 PROCEDURE DIVISION.
    MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
    INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                  USING     CORBA-ORB-OBJECTID_NAMESERVICE
                  RETURNING OBJ.
```

G.3.1.3 サーバアプリケーションのオブジェクトリファレンスの獲得

ネーミングサービスのCOSNAMING-NAMINGCONTEXT-RESOLVEメソッドにより、これから実行したいサーバアプリケーションのオブジェクトリファレンスを取り出します。検索したいオブジェクト名を当メソッドのパラメタとして指定します。

```
 DATA DIVISION.
 WORKING-STORAGE        SECTION.
 01 NAMING-CONTEXT      USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT      USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NAME-ID             USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND           USAGE OBJECT REFERENCE CORBA-STRING.
 01 TARGET-OBJ          USAGE OBJECT REFERENCE ODDEMO-CALCULATOR.

PROCEDURE DIVISION.
   INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                    USING     OBJ
                    RETURNING NAMING-CONTEXT.
*
   INVOKE CORBA-STRING "NEW-WITH-VALUE"
                    USING     "ODdemo::calculator"
                    RETURNING NAME-ID.
*
   INVOKE CORBA-STRING "NEW-WITH-VALUE"
                    USING     " "
                    RETURNING NAME-KIND.
*
   INVOKE COSNAMING-NAMECOMPONENT "NEW"
                    RETURNING NAME-COMPONENT.
*
   SET IDL-ID OF NAME-COMPONENT TO NAME-ID.
   SET KIND   OF NAME-COMPONENT TO NAME-KIND.
*
   INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH"
                    USING     1
                    RETURNING NAME.
*
   INVOKE NAME "SET-VALUE"
                    USING     1
                              NAME-COMPONENT.
*
   MOVE "CosNaming::NamingContext::resolve" TO API-NAME.
   INVOKE NAMING-CONTEXT "RESOLVE"
                    USING     NAME
                    RETURNING OBJ.
*
   INVOKE ODDEMO-CALCULATOR "NARROW"
                    USING     OBJ
                    RETURNING TARGET-OBJ.
```

G.3.1.4 メソッドの呼び出し

サーバプログラムのメソッドを呼び出します。メソッド名はIDLで指定したモジュール名、インタフェース名およびメソッド名を2つのコロン(“::”)でつなげた形式で指定します。この例では、ODdemo,Calculator,calculateがそれにあたります。なお、メソッド呼出し時にネーミングサービスで求めたサーバアプリケーションのオブジェクトリファレンスとサーバプログラムで例外が発生した場合に例外情報を設定してもらうためにRETURNING指定を行います。

```
 DATA DIVISION.
 WORKING-STORAGE        SECTION.
 01 RESULT              TYPE                   ODDEMO-CALCULATOR-RESULT.
 01 PARAM-A             TYPE                   CORBA-LONG.
```

```
 01 PARAM-B              TYPE                    CORBA-LONG.

PROCEDURE DIVISION.
   MOVE 10 TO PARAM-A.
   MOVE 5  TO PARAM-B.
*
   MOVE "ODdemo::Calculator::calculate(10,5)" TO API-NAME.
   INVOKE TARGET-OBJ "CALCULATE"
                     USING     PARAM-A
                               PARAM-B
                     RETURNING RESULT.
*
   PERFORM PRINT-RESULT.
*
   MOVE 100 TO PARAM-A.
   MOVE 0   TO PARAM-B.
*
   MOVE "ODdemo::Calculator::calculate(100,0)" TO API-NAME.
   INVOKE TARGET-OBJ "CALCULATE"
                     USING     PARAM-A
                               PARAM-B
                     RETURNING RESULT.
*
   PERFORM PRINT-RESULT.
*
   STOP RUN.
*
 END-MAIN.
```

G.3.2 クライアントアプリケーションのプログラミング（動的起動インタフェース）

動的起動インタフェースを使用する場合の、クライアントアプリケーションの流れを以下に示します。

G.3.2.1 初期化

CORBAの初期化メソッドCORBA-ORB_INITを呼び出し初期化処理を行います。当メソッドの結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。環境部(ENVIRONMENT DIVISION)およびデータ部(DATA DIVISION)を定義します。
ここでは、IDLファイル名を“simple.idl”として説明します。IDLコンパイラが出力する登録集ファイル名は、“simple--rep.cbl”, “simple--const.cbl”,“simple--copy.cbl”となります。

```
 IDENTIFICATION          DIVISION.
 PROGRAM-ID.             "SIMPLE_C"
*
 ENVIRONMENT             DIVISION.
 CONFIGURATION           SECTION.
 REPOSITORY.
* ObjectDirectorの標準登録集（リポジトリ宣言用）
     COPY                CORBA--REP.
* ネーミングサービスの標準登録集（リポジトリ宣言用）
     COPY                COSNAMING--REP.
* インタフェースリポジトリの標準登録集（リポジトリ宣言用）
     COPY                INTERFACEREP--REP.
* IDLコンパイラが出力した登録集（リポジトリ宣言用）
     COPY                SIMPLE--REP.
    .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
* ObjectDirectorの標準登録集（定数宣言用）
     COPY                CORBA--CONST.
```

```
* ネーミングサービスの標準登録集（定数宣言用）
     COPY               COSNAMING--CONST.
* インタフェースリポジトリの標準登録集（定数宣言用）
     COPY               INTERFACEREP--CONST.
* IDLコンパイラが出力した登録集（定数宣言用）
     COPY               SIMPLE--CONST.
     .
*
 DATA                   DIVISION.
   WORKING-STORAGE        SECTION.
   01 APL-NAME           PIC   X(8)   VALUE     "Simple_c".
   01 EXCEPTION-ID       OBJECT REFERENCE CORBA-STRING.
   01 EXCEPTION-ID-VALUE  PIC    X(50).
   01 API-NAME           PIC    X(50).
   01 ORB                OBJECT REFERENCE CORBA-ORB.
*
 PROCEDURE DIVISION.
*
 DECLARATIVES.
 ERR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY API-NAME " is failed.".
     DISPLAY "Exception-id is " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-ERR.
 END DECLARATIVES.
*
     INVOKE CORBA "ORB_INIT" USING APL-NAME FJ-OM_ORBID RETURNING ORB.
```

例外処理については、“G.3.3 クライアントアプリケーションの例外処理”を参照してください。

G.3.2.2 ネーミングサービスのオブジェクトリファレンスの獲得

実行するオブジェクトをネーミングサービスから検索するために、ネーミングサービスのオブジェクトリファレンスが必要となります。CORBAインタフェースのオブジェクトリファレンスの取出しメソッドCORBA-ORB-RESOLVE_INITIAL_REFERENCESによりネーミングサービスのオブジェクトリファレンスを取り出します。このとき、CORBA-ORB-OBJECTID_NAMESERVICE（値は“NameService”）をメソッドのパラメタとして指定します。

```
   DATA DIVISION.
   WORKING-STORAGE        SECTION.
   01 OBJ                 OBJECT REFERENCE CORBA-OBJECT.
   01 NAMING-CONTEXT      OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.

  PROCEDURE DIVISION.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES" USING     CORBA-ORB-OBJECTID_NAMESERVICE
                                             RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW" USING     OBJ
                                             RETURNING NAMING-CONTEXT.
```

G.3.2.3 サーバアプリケーションのオブジェクトリファレンスの獲得

ネーミングサービスCOSNAMING-NAMINGCONTEXT-RESOLVEメソッドにより、これから実行したいサーバアプリケーションのオブジェクトリファレンスを取り出します。検索したいオブジェクト名を本メソッドのパラメタとして指定します。

```
   DATA DIVISION.
   WORKING-STORAGE        SECTION.
```

```
  01 NAME                TYPE             COSNAMING-NAME.
  01 NAME-COMPONENT      OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
  01 NAME-ID             OBJECT REFERENCE CORBA-STRING.
  01 NAME-KIND           OBJECT REFERENCE CORBA-STRING.
  01 NAME-LEN            TYPE             CORBA-UNSIGNED-LONG.
  01 NAME-ID-VALUE       PIC   X(18)  VALUE     "ODdemo::calculator".
  01 NAME-KIND-VALUE     PIC   X(1)   VALUE     SPACE.

 PROCEDURE DIVISION.
    MOVE 1 TO NAME-LEN.
    INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH" USING     NAME-LEN
                                                        RETURNING NAME.
    INVOKE COSNAMING-NAMECOMPONENT "NEW" RETURNING NAME-COMPONENT.
    INVOKE CORBA-STRING "NEW-WITH-VALUE" USING     NAME-ID-VALUE   *> オブジェクト名
                                         RETURNING NAME-ID.
    INVOKE CORBA-STRING "NEW-WITH-VALUE" USING     NAME-KIND-VALUE  *> オブジェクトタイプ
                                         RETURNING NAME-KIND.
    SET IDL-ID OF NAME-COMPONENT TO NAME-ID.
    SET KIND OF NAME-COMPONENT TO NAME-KIND.
    INVOKE NAME "SET-VALUE" USING NAME-LEN NAME-COMPONENT.
    INVOKE NAME "SET-VALUE" USING NAME-LEN
                                  NAME-COMPONENT.
* NamingServiceのresolveメソッドを発行
     INVOKE NAMING-CONTEXT "RESOLVE" USING NAME RETURNING OBJ.
```

G.3.2.4 インタフェースリポジトリからサーバアプリケーション情報の獲得

インタフェースリポジトリには、IDLで定義されたモジュール名、インタフェース名、オペレーション名およびパラメタが階層構造で格納されます。サーバアプリケーションの情報を獲得するため、以下の手順で検索を行います。

(1)CORBA-INTERFACEDEFオブジェクトリファレンスの獲得

インタフェースリポジトリからサーバアプリケーションの情報を獲得するためには、INTERFACEDEFオブジェクトのオブジェクトリファレンスが必要です。CORBA-OBJECT-GET_INTERFACE_DEFメソッドを使用します。

```
  DATA DIVISION.
  WORKING-STORAGE        SECTION.
  01 INTF-OBJ            OBJECT REFERENCE CORBA-OBJECT.
  01 INTF                OBJECT REFERENCE CORBA-INTERFACEDEF.

 PROCEDURE DIVISION.
* CORBA-INTERFACEDEFオブジェクトリファレンス取得
     INVOKE OBJ "GET_INTERFACE_DEF" RETURNING INTF-OBJ.
     INVOKE CORBA-INTERFACEDEF "NARROW" USING INTF-OBJ
                                       RETURNING INTF.
```

(2)SEQUENCE-CORBA-CONTAINEDオブジェクトリファレンスの獲得

CORBA-INTERFACEDEF-LOOKUP_NAMEメソッドにより、サーバアプリケーションが複数もっているメソッドから指定されたメソッドをインタフェースリポジトリから検索します。パラメタとしてサーバアプリケーションのメソッド名を指定します。結果として、指定したメソッド情報が格納されているSEQUENCE-CORBA-CONTAINEDオブジェクトのオブジェクトリファレンスが返されます。

```
  DATA DIVISION.
  WORKING-STORAGE        SECTION.
  01 DK-TYPE             TYPE             CORBA-DEFINITIONKIND.
  01 CTD-SEQ             OBJECT REFERENCE SEQUENCE-CORBA-CONTAINED.

 PROCEDURE DIVISION.
* SEQUENCE-CORBA-CONTAINEDオブジェクトリファレンス取得
```

```
     SET  CORBA-DK_OPERATION OF DK-TYPE TO TRUE.
     INVOKE INTF "LOOKUP_NAME" USING  "calculate"
                                      -1
                                       DK-TYPE
                                       CORBA-FALSE
                                RETURNING CTD-SEQ.
```

(3)パラメタ情報の獲得

CORBA-CONTAINED-DESCRIBEメソッドにより、サーバアプリケーションがもっているメソッドのパラメタの情報(パラメタの名前、個数、パラメタの型等)をインタフェースリポジトリから検索します。パラメタとして、CORBA-CONTAINED-DESCRIPTIONオブジェクトリファレンスを指定します。

```
   DATA DIVISION.
   WORKING-STORAGE        SECTION.
   01 TMP-CTD             OBJECT REFERENCE CORBA-CONTAINED.
   01 CTD-DES             OBJECT REFERENCE CORBA-CONTAINED-DESCRIPTION.

  PROCEDURE DIVISION.
* 指定されたメソッドのパラメタ情報を検索
     INVOKE CTD-SEQ "GET-VALUE" USING 1
                                RETURNING TMP-CTD.
     INVOKE TMP-CTD "DESCRIBE" RETURNING CTD-DES.
```

G.3.2.5 パラメタの組立て

(1)パラメタリストの生成

CORBA-ORB-CREATE_LISTメソッドにより、サーバアプリケーションに渡すパラメタを格納するための領域を保持するリストオブジェクトを生成します。このとき、何個のパラメタを格納するかを指定します。結果として、NVListオブジェクトリファレンスが通知されます。NVListオブジェクトインタフェースの詳細については、“NVListオブジェクト”を参照してください。

```
   DATA DIVISION.
   WORKING-STORAGE        SECTION.
   01 OPE-ANY             OBJECT REFERENCE CORBA-ANY.
   01 OPE-DES             OBJECT REFERENCE CORBA-OPERATIONDESCRIPTION.
   01 PARA-SEQ            TYPE             CORBA-PARDESCRIPTIONSEQ.
   01 PARAM-COUNT         TYPE             CORBA-LONG.
   01 NVLIST              OBJECT REFERENCE CORBA-NVLIST.

  PROCEDURE DIVISION.
* CORBA-CONTAINED-DESCRIPTION構造体からパラメタ情報構造体の抽出
     SET OPE-ANY TO IDL-VALUE OF CTD-DES.
     INVOKE CORBA-OPERATIONDESCRIPTION--H  "EXTRACT" USING OPE-ANY
                                                    RETURNING OPE-DES.
* パラメタ情報の抽出
     SET PARA-SEQ TO PARAMETERS OF OPE-DES.
     MOVE SEQ-LENGTH OF PARA-SEQ TO PARAM-COUNT.
* リストオブジェクトの生成
     INVOKE ORB "CREATE_LIST" USING  PARAM-COUNT
                                     NVLIST.
```

(2)パラメタリストの設定

CORBA-NVLIST-ADD_VALUEメソッドにより、サーバアプリケーションに渡すパラメタをリストオブジェクトに設定します。パラメタとして、サーバアプリケーションのパラメタの名前、値(Any型)およびパラメタの形式を設定します。

```
   DATA DIVISION.
   WORKING-STORAGE        SECTION.
   01 PARA-DES1           OBJECT REFERENCE CORBA-PARAMETERDESCRIPTION.
   01 PARA-DES2           OBJECT REFERENCE CORBA-PARAMETERDESCRIPTION.
   01 STR-1               OBJECT REFERENCE CORBA-STRING.
   01 NAME1               PIC    X(40).
   01 STR-2               OBJECT REFERENCE CORBA-STRING.
   01 NAME2               PIC    X(40).
   01 ANY-A               OBJECT REFERENCE CORBA-ANY.
   01 ANY-B               OBJECT REFERENCE CORBA-ANY.
   01 TMP-NAMEDVALUE      OBJECT REFERENCE CORBA-NAMEDVALUE.
   01 FLG                 TYPE             CORBA-FLAGS.

  PROCEDURE DIVISION.
* PARAMERTER 1
      INVOKE PARA-SEQ "GET-VALUE" USING 1
                                  RETURNING PARA-DES1
      SET STR-1 TO  NAME OF PARA-DES1.
      INVOKE STR-1 "GET-VALUE" RETURNING NAME1.
      INVOKE CORBA-ANY "NEW" RETURNING  ANY-A.
      INVOKE ANY-A "INSERT_LONG" USING  100.
      SET CORBA-ARG_IN OF FLG TO TRUE.
      INVOKE NVLIST "ADD_VALUE" USING      NAME1
                                           ANY-A
                                           FLG
                                RETURNING  TMP-NAMEDVALUE.
* PARAMETER 2
      INVOKE PARA-SEQ "GET-VALUE" USING 2
                                  RETURNING PARA-DES2
      SET STR-2 TO  NAME OF PARA-DES2.
      INVOKE STR-2 "GET-VALUE" RETURNING NAME2.
      INVOKE CORBA-ANY "NEW" RETURNING  ANY-B.
      INVOKE ANY-B "INSERT_LONG" USING  20.
      SET CORBA-ARG_IN OF FLG TO TRUE.
      INVOKE NVLIST "ADD_VALUE" USING      NAME2
                                           ANY-B
                                           FLG
                                RETURNING  TMP-NAMEDVALUE.
```

G.3.2.6 リクエストの作成

CORBA-OBJECT-CREATE_REQUESTメソッドにより、リクエストオブジェクトを作成します。リクエストオブジェクトに対して、サーバオブジェクトのオブジェクトリファレンス、NVListオブジェクトリファレンス、後述するNamedValueと呼ぶサーバの処理結果を格納する領域を指定します。結果として、リクエストオブジェクトが返されます。

```
   DATA DIVISION.
   WORKING-STORAGE        SECTION.
   01 RESULT-TYPE         OBJECT REFERENCE CORBA-TYPECODE.
   01 ANY-RESULT          OBJECT REFERENCE CORBA-ANY.
   01 NAMEDVALUE          OBJECT REFERENCE CORBA-NAMEDVALUE.
   01 EXCEPTIONLIST       OBJECT REFERENCE CORBA-EXCEPTIONLIST.
   01 TMP-EXCEP           TYPE             CORBA-EXCDESCRIPTIONSEQ.
   01 EX-DES              OBJECT REFERENCE CORBA-EXCEPTIONDESCRIPTION.
   01 EX-TYPE             OBJECT REFERENCE CORBA-TYPECODE.
   01 REQ                 OBJECT REFERENCE CORBA-REQUEST.

  PROCEDURE DIVISION.
* 復帰パラメタ
      SET RESULT-TYPE TO  RESULT OF OPE-DES.
      INVOKE CORBA-ANY "NEW" RETURNING ANY-RESULT
      INVOKE ANY-RESULT "SET-TYPE" USING RESULT-TYPE.
```

```
      SET CORBA-ARG_OUT OF FLG TO TRUE.
* 復帰パラメタ格納域
      INVOKE ORB "CREATE_NAMED_VALUE" USING      " "
                                                 ANY-RESULT
                                                 FLG
                                       RETURNING  NAMEDVALUE.
 * EXCEPTIONLIST
      INVOKE ORB "CREATE_EXCEPTION_LIST" USING EXCEPTIONLIST.
      SET TMP-EXCEP TO EXCEPTIONS OF OPE-DES.
      INVOKE TMP-EXCEP "GET-VALUE" USING 1 RETURNING EX-DES.
      SET EX-TYPE TO IDL-TYPE OF EX-DES.
      INVOKE EXCEPTIONLIST "ADD" USING EX-TYPE.
*  リクエストオブジェクトの生成
      INVOKE OBJ "CREATE_REQUEST" USING  NULL          *>CONTEXT
                                         "calculate"   *>メソッド名
                                         NVLIST        *>入力パラメタ
                                         NAMEDVALUE    *>復帰値
                                         EXCEPTIONLIST *>EXCEPTIONLIST
                                         NULL          *>CONTEXTLIST
                                         REQ.          *>リクエスト
```

G.3.2.7 リクエストの送信

サーバアプリケーションに対してリクエストを出します。リクエストの方法として、同期送信と非同期送信の2つの方法があります。

(1)同期送信

CORBA-REQUEST-INVOKEメソッドにより、サーバアプリケーションを呼び出します。

```
  INVOKE REQ "INVOKE".
```

(2)非同期送信

CORBA-REQUEST-SEND_DEFERREDメソッドにより、サーバアプリケーションを呼び出します。サーバアプリケーションの処理結果は、CORBA-REQUEST-GET_RESPONSEメソッドにより受け取ります。

```
* 処理の要求
  INVOKE REQ "SEND_DEFERRED".
* 処理結果の受取り
  INVOKE REQ "GET_RESPONSE".
```

CORBA-REQUEST-GET_RESPONSEサーバアプリケーションからリクエストが完了していないことが判明した場合、再度CORBA-REQUEST-GET_RESPONSEメソッドを呼び出します。

G.3.3 クライアントアプリケーションの例外処理

クライアントアプリケーションは、サーバアプリケーションの処理が正常終了したか異常終了したかを知ることができます。また、異常終了した場合には、システムで異常終了したかサーバアプリケーションで異常終了したかを知ることができます。前者をシステム例外、後者をユーザ例外と呼びます。システム例外を以下に示します。例外の意味については、“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。

例外情報	例外コード
BADC_ONTEXT	CORBA-BAD_CONTEXT
BAD_INV_ORDER	CORBA-BAD_INV_ORDER
BAD_OPERATION	CORBA-BAD_OPERATION
BAD_PARAM	CORBA-BAD_PARAM

例外情報	例外コード
BAD_QOS	CORBA-BAD_QOS
BAD_TYPECODE	CORBA-BAD_TYPECODE
CODESET_INCOMPATIBLE	CORBA-CODESET_INCOMPATIBLE
COMM_FAILURE	CORBA-COMM_FAILURE
DATA_CONVERSION	CORBA-DATA_CONVERSION
FREE_MEM	CORBA-FREE_MEM
IMP_LIMIT	CORBA-IMP_LIMIT
INITIALIZE	CORBA-INITIALIZE
INTERNAL	CORBA-INTERNAL
INTF_REPOS	CORBA-INTF_REPOS
INV_FLAG	CORBA-INV_FLAG
INV_IDENT	CORBA-INV_IDENT
INV_OBJREF	CORBA-INV_OBJREF
INV_POLICY	CORBA-INV_POLICY
MARSHAL	CORBA-MARSHAL
NO_IMPLEMENT	CORBA-NO_IMPLEMENT
NO_MEMORY	CORBA-NO_MEMORY
NO_PERMISSION	CORBA-NO_PERMISSION
NO_RESOURCES	CORBA-NO_RESOURCES
NO_RESPONSE	CORBA-NO_RESPONSE
OBJ_ADAPTER	CORBA-OBJ_ADAPTER
PERSIST_STORE	CORBA-PERSIST_STORE
REBIND	CORBA-REBIND
TIMEOUT	CORBA-TIMEOUT
TRANSIENT	CORBA-TRANSIENT
UNKNOWN	CORBA-UNKNOWN
INVALID_TRANSACTION	CORBA-INVALID_TRANSACTION
TRANSACTION_MODE	CORBA-TRANSACTION_MODE
TRANSACTION_REQUIRED	CORBA-TRANSACTION_REQUIRED
TRANSACTION_ROLLEDBACK	CORBA-TRANSACTION_ROLLEDBACK
TRANSACTION_UNAVAILABLE	CORBA-TRANSACTION_UNAVAILABLE

USE文にCORBA-EXCEPTIONクラス等の例外クラスを定義することで例外情報を取り出します。取り出したオブジェクトはCORBA-EXCEPTION-ID-GET-VALUEで文字に変換することができます。
また、システム例外発生時には、CORBA-SYSTEMEXCEPTION-MINORを使用してマイナーコードを獲得することができます。マイナーコードの値については、“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。

G.3.3.1 静的起動インタフェース例外

静的起動インタフェース例外情報を獲得する例を以下に示します。

```
 WORKING-STORAGE SECTION.
 01  API-NAME            PIC   X(50)
 01  EXCEPTION-ID        USAGE OBJECT REFERENCE CORBA-STRING.
 01  EXCEPTION-ID-VALUE  PIC   X(50).
 01  OUT-MINOR           TYPE  CORBA-UNSIGNED-LONG.

 DECLARATIVES.
 *
 ZEROPARAM-ERR SECTION.
      USE EXCEPTION ODDEMO-CALCULATOR-ZEROPARAM.
      DISPLAY "ZEROPARAM Exception: " API-NAME.
      EXIT PROGRAM.
 END-ZEROPARAM-ERR.
 *
 OTHER-ERROR SECTION.
      USE EXCEPTION CORBA-EXCEPTION.
      DISPLAY "CORBA::UserException: " API-NAME.
      SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
      INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
      DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
      EXIT PROGRAM.
 END-OTHER-ERR.
 * マイナーコード取得処理
 MINOR-ERROR SECTION.
      USE EXCEPTION CORBA-SYSTEMEXCEPTION.
      MOVE MINOR OF EXCEPTION-OBJECT AS CORBA-SYSTEMEXCEPTION TO OUT-MINOR.
      DISPLAY "   Exception-minor: " OUT-MINOR.
 END-MINOR-ERROR.

 END DECLARATIVES.
```

G.3.3.2 動的起動インタフェース例外

動的起動インタフェース例外情報を獲得する例を以下に示します。

```
* リクエストの送信
    INVOKE REQ "INVOKE".
* 例外処理
     SET TMP-ENV TO ENV OF REQ.
     IF NOT TMP-ENV = NULL
       SET EXCEP TO IDL-EXCEPTION OF TMP-ENV
       INVOKE EXCEP AS CORBA-UNKNOWNUSEREXCEPTION "EXCEPTION" RETURNING ANY-EXC
       SET EXC-STR TO IDL-ID OF EXCEP
       INVOKE EXC-STR "GET-VALUE" RETURNING EXCEP-ID
       DISPLAY "EXCEPTION ID :  "  EXCEP-ID
       EXIT PROGRAM
      END-IF.
```

G.3.4 サーバアプリケーションのプログラミング（静的スケルトンインタフェース）

Windows32 Solaris32 Linux32

サーバアプリケーションは、初期化処理部とインタフェースの実装部分から構成されます。初期化処理部では、以下に示す処理を行います。

G.3.4.1 初期化

CORBAの初期化メソッドCORBA-ORB_INITを呼び出し初期化処理を行います。当関数の結果として、ORBのオブジェクトリファレンスが通知されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
 IDENTIFICATION          DIVISION.
 PROGRAM-ID.             "SIMPLE_S".
*
 ENVIRONMENT             DIVISION.
 CONFIGURATION           SECTION.
 REPOSITORY.
     COPY                CORBA--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY                CORBA--CONST.
     COPY                SIMPLE--CONST.
     .
*
 DATA                    DIVISION.
 WORKING-STORAGE         SECTION.
 COPY                    CORBA--COPY.
*
 01 ORB                  USAGE OBJECT REFERENCE CORBA-ORB.
 01 BOA                  USAGE OBJECT REFERENCE CORBA-BOA.
 01 IMPL-REP             USAGE OBJECT REFERENCE FJ-IMPLEMENTATIONREP.
 01 IMPL                 USAGE OBJECT REFERENCE CORBA-IMPLEMENTATIONDEF.
 01 OBJ                  USAGE OBJECT REFERENCE CORBA-OBJECT.
*
 01 EXCEPTION-ID         USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE   PIC   X(50).
 01 API-NAME             PIC   X(50).
*
*
 PROCEDURE               DIVISION.
*
 DECLARATIVES.
*
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::UserException: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
 MAIN SECTION.
*
```

```
      MOVE "CORBA::ORB_init" TO API-NAME.
      INVOKE CORBA "ORB_INIT"
                          USING     "SIMPLE_S"
                                    FJ-OM_ORBID
                          RETURNING ORB.
*
      MOVE "CORBA::ORB::BOA_init" TO API-NAME.
      INVOKE ORB "BOA_INIT"
                          USING     "SIMPLE_S"
                                    CORBA-BOA_OAID
                          RETURNING BOA.
```

なお、必要であればサーバアプリケーションの初期化も行います。

G.3.4.2 サーバの活性化

サーバアプリケーションの初期化が完了すると、ORBに対してその旨を通知します。ORBは、この命令が発行された時点で、クライアントからの要求をサーバアプリケーションに伝えます。活性化の方法は、サーバのタイプによって以下のように異なります。

サーバのタイプ	メソッド
shared server	CORBA-BOA-IMPL_IS_READY
unshared server	CORBA-BOA-OBJ_IS_READY
persistent server	CORBA-BOA-IMPL_IS_READY

1. インプリメンテーションリポジトリのオブジェクトリファレンスを取り出します。
   CORBA-ORB-RESOLVE_INITIAL_REFERENCESメソッドを用いて、パラメタにCORBA-ORB-OBJECTID_IMPLREPを指定します。
2. IMPLEMENTATIONREPオブジェクトのオブジェクトリファレンスを求めます。
   FJ-IMPLEMENTATIONREP-LOOKUP_IDを用いて、パラメタにサーバアプリケーションのIMPLEMENTATIONREPオブジェクトを指定します。
3. サーバの活性化を行います。
   CORBA-BOA-IMPL_IS_READYまたはCORBA-BOA-OBJ_IS_READYを用います。

注意

メソッドの復帰後は、初期化時にオープンしたファイルのクローズ処理や、領域の解放処理などの非活性化処理(“G.3.4.4 サーバの非活性化”を参照)を行う必要があります。

注意

Windows32

以下の条件のとき、CORBA-BOA-IMPL-IS-READY／CORBA-BOA-OBJ-IS-READYが復帰せずにサーバアプリケーションが終了します。

- サーバアプリケーションをプロセスモードで作成している場合 かつ
- サーバアプリケーションをCORBAワークユニット運用している場合 かつ
- CORBAワークユニットを停止した場合

CORBA-BOA-IMPL-IS-READY／CORBA-BOA-OBJ-IS-READYの後に後処理を行う場合は、サーバアプリケーションをスレッドモードで作成するか、またはワークユニットの出口機能を利用してください。ワークユニットの出口機能に関しては、“OLTPサーバ運用ガイド”の“ワークユニットの出口機能”を参照してください。

```
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                        USING     CORBA-OBJECTID_IMPLEMENTAT-001
                        RETURNING OBJ.
*
     INVOKE FJ-IMPLEMENTATIONREP "NARROW"
                        USING     OBJ
                        RETURNING IMPL-REP.
*
     MOVE "FJ::ImplementationRep::lookup_id" TO API-NAME.
     INVOKE IMPL-REP "LOOKUP_ID"
                        USING     IMPL-ODDEMO-CALCULATOR
                        RETURNING OBJ.
*
     INVOKE CORBA-IMPLEMENTATIONDEF "NARROW"
                        USING     OBJ
                        RETURNING IMPL.
*
     MOVE "CORBA::BOA::impl_is_ready" TO API-NAME.
     INVOKE BOA "IMPL_IS_READY"
                        USING     IMPL.
```

G.3.4.3 インタフェース実装クラス

初期化処理の後、サーバアプリケーションで実装するインタフェースの処理を記述します。OOCOBOLでは、インタフェースの処理を行うインタフェース実装クラスと、インタフェース実装クラスの生成プログラムを用意する必要があります。

インタフェース実装クラス

```
  CLASS-ID.              ODDEMO-CALCULATOR-IMPL
      INHERITS           ODDEMO-CALCULATOR.
 *
  ENVIRONMENT            DIVISION.
  CONFIGURATION          SECTION.
  REPOSITORY.
      COPY               CORBA--REP.
      COPY               SIMPLE--REP.
      .
  SPECIAL-NAMES.
      SYMBOLIC CONSTANT
      COPY               CORBA--CONST.
      COPY               SIMPLE--CONST.
      .
 *
  OBJECT.
   DATA                  DIVISION.
   WORKING-STORAGE       SECTION.
   COPY                  CORBA--COPY.
   COPY                  SIMPLE--COPY.
 *
   PROCEDURE             DIVISION.
 *
    METHOD-ID.           CALCULATE OVERRIDE.
    DATA                 DIVISION.
    WORKING-STORAGE      SECTION.
    01 EXC              USAGE OBJECT REFERENCE ODDEMO-CALCULATOR-ZEROPARAM.
    LINKAGE              SECTION.
    01 A                TYPE                   CORBA-LONG.
    01 B                TYPE                   CORBA-LONG.
    01 RESULT           TYPE                   ODDEMO-CALCULATOR-RESULT.
```

```
*
   PROCEDURE            DIVISION
                        USING     A
                                  B
                        RETURNING RESULT
                        RAISING   CORBA-SYSTEMEXCEPTION
                                  ODDEMO-CALCULATOR-ZEROPARAM.
*
     IF B = 0
       INVOKE ODDEMO-CALCULATOR-ZEROPARAM "NEW" RETURNING EXC
       EXIT METHOD RAISING EXC
     END-IF.
*
     COMPUTE ADD_RESULT OF RESULT = A + B.
     COMPUTE SUBTRACT_RESULT OF RESULT = A - B.
     COMPUTE MULTIPLE_RESULT OF RESULT = A * B.
     COMPUTE DIVIDE_RESULT OF RESULT = A / B.
*
   END METHOD CALCULATE.
*
 END OBJECT.
*
 END CLASS ODDEMO-CALCULATOR-IMPL.
```

インタフェース実装クラス生成プログラム

インタフェース実装クラスは、ORBが生成します。ORBからインタフェース実装クラス生成プログラムが呼ばれて、インタフェース実装クラスが生成されます。インタフェース実装クラス生成プログラム名は、モジュール名、インタフェース名をハイフン(“-”)でつなげた名前に“−INIT”を付加した名前にします。RETURNING句には、インタフェースクラスを指定します。このプログラムではインタフェース実装クラスを生成して返します。

```
  IDENTIFICATION        DIVISION.
  PROGRAM-ID.           ODDEMO-CALCULATOR--INIT.
*
  ENVIRONMENT           DIVISION.
  CONFIGURATION         SECTION.
  REPOSITORY.
      CLASS             ODDEMO-CALCULATOR
      CLASS             ODDEMO-CALCULATOR-IMPL
      .
*
  DATA                  DIVISION.
  LINKAGE               SECTION.
  01 RESULT             USAGE OBJECT REFERENCE ODDEMO-CALCULATOR.
*
  PROCEDURE             DIVISION
                        RETURNING RESULT.
*
      INVOKE ODDEMO-CALCULATOR-IMPL "NEW"
                        RETURNING RESULT.
*
  END PROGRAM "ODDEMO-CALCULATOR--INIT".
```

G.3.4.4 サーバの非活性化

サーバアプリケーションは、利用者からの停止要求を受けた場合、クライアントからの要求を以降受け付けない旨をORBに対して返します。サーバアプリケーションの利用者からの停止要求の方法およびアプリケーションでの停止要求の受信方法等は各ORBベンダーにより異なります。この命令が発行された時点で、ORBは、クライアントから要求されてもサーバアプリケーションに伝えず、クライアントへは例外が返ります。非活性化の方法は、サーバのタイプによって以下のように異なります。

サーバのタイプ	メソッド
shared server	CORBA-BOA-DEACTIVATE_IMPL
unshared server	CORBA-BOA-DEACTIVATE_OBJ
persistent server	CORBA-BOA-DEACTIVATE_IMPL

CORBA-BOA-DEACTIVATE_IMPLでは、サーバアプリケーションのIMPLEMENTATIONREPオブジェクトをパラメタとして指定します。CORBA-BOA-DEACTIVATE_OBJでは、オブジェクトリファレンスをパラメタとして指定します。

```
*
     MOVE "CORBA::BOA::diactivate_impl" TO API-NAME.
     INVOKE BOA "DEACTIVATE_IMPL"
                        USING     IMPL.
```

なお、ワークユニット運用している場合は、isstopwuコマンドやInterstage管理コンソールから停止操作が実行されることにより、クライアントからの要求を以降受け付けない旨をORBに対して通知します。そのため、アプリケーション側で非活性化用のメソッドを発行する必要はありません。
ワークユニット運用でないCORBAアプリケーションの場合も、odcntlqueコマンドでサーバアプリケーションの終了を行う場合は、アプリケーション側で非活性化用のメソッドを発行する必要はありません。

G.3.5 サーバアプリケーションの例外処理

Windows32 Solaris32 Linux32

(1)例外情報の設定

例外情報を設定するためには、まずNEWメソッドで例外情報を設定する領域を獲得します。続いて、EXIT METHOD文にRAISING指定を行うことにより、例外情報を設定します。

```
 CLASS-ID.              ODDEMO-CALCULATOR-IMPL
*
 OBJECT.
*
  PROCEDURE             DIVISION.
*
   METHOD-ID.           CALCULATE OVERRIDE.
   DATA                 DIVISION.
   WORKING-STORAGE      SECTION.
   01 EXC               USAGE OBJECT REFERENCE ODDEMO-CALCULATOR-ZEROPARAM.
   LINKAGE              SECTION.
   01 A                 TYPE                   CORBA-LONG.
   01 B                 TYPE                   CORBA-LONG.
   01 RESULT            TYPE                   ODDEMO-CALCULATOR-RESULT.
*
   PROCEDURE            DIVISION
                        USING     A
                                  B
                        RETURNING RESULT
                        RAISING   CORBA-SYSTEMEXCEPTION
                                  ODDEMO-CALCULATOR-ZEROPARAM.
*
     IF B = 0
       INVOKE ODDEMO-CALCULATOR-ZEROPARAM "NEW" RETURNING EXC
       EXIT METHOD RAISING EXC
     END-IF.
*
     COMPUTE ADD_RESULT OF RESULT = A + B.
     COMPUTE SUBTRACT_RESULT OF RESULT = A - B.
```

```
      COMPUTE MULTIPLE_RESULT OF RESULT = A * B.
      COMPUTE DIVIDE_RESULT OF RESULT = A / B.
*
   END METHOD CALCULATE.
*
 END OBJECT.
*
 END CLASS ODDEMO-CALCULATOR-IMPL.
```

(2)例外情報の獲得

サーバアプリケーションで例外を獲得する方法は、クライアントアプリケーションの例外処理と同様です。詳細については、“G.3.3 クライアントアプリケーションの例外処理”を参照してください。

G.3.6 サーバアプリケーションの登録

Windows32 Solaris32 Linux32

作成したアプリケーションをインプリメンテーションリポジトリ、ネーミングサービスに登録します。

G.3.6.1 インプリメンテーションリポジトリへの登録

インプリメンテーションリポジトリにサーバアプリケーション情報を登録するため、OD_impl_instコマンドを使用します。OD_impl_instコマンドでの登録処理の例と指定する情報について以下に示します。

例

インプリメンテーションリポジトリへの登録

```
OD_impl_inst -ax def
```

-ax def

定義ファイルで指定した定義情報でサーバアプリケーション情報を登録することを示します。

defファイルの内容 Windows32

```
rep_id                    = IDL:ODdemo/calculator:1.0
type                      = shared
binary                    = D:¥server¥simple_s.exe
IDL:ODdemo/calculator:1.0 = ODDEMO-CALCULATOR.dll
mode                      = SYNC_END
ior                       = 1.1
locale                    = SJIS
```

rep_id = IDL:ODdemo/calculator:1.0

インプリメンテーションリポジトリIDを指定します。

type = shared

サーバタイプを指定します。指定できるタイプには、persistent、shared、unsharedがあります。

binary = D:¥server¥simple_s.exe

サーバアプリケーションのパス名を指定します。

IDL:ODdemo/calculator:1.0 = ODDEMO-CALCULATOR.dll

サーバアプリケーションのライブラリ名を指定します。
ライブラリは、サーバアプリケーションと同一ディレクトリ、または環境変数PATHに指定されているディレクトリに格納する必要があります。

mode = SYNC_END

サーバアプリケーションがCORBA-BOA-IMPL_IS_READY関数（または、CORBA-BOA-OBJ_IS_READY関数）発行時の動作モードを指定します。OOCOBOLサーバアプリケーションをスレッドモードで作成した場合は、SYNC_ENDを指定します。

ior = 1.1

連携するIORのバージョンを指定します。指定できるバージョンには、1.0、1.1があります。

locale = SJIS

サーバアプリケーションのコード系を指定します。
開発環境として“NetCOBOL”を使用してUNICODEのアプリケーションを開発する場合は、“UNICODE”を指定してください。

defファイルの内容 Solaris32 Linux32

```
rep_id = IDL:ODdemo/calculator:1.0
type = shared
binary = /home/guest/simple_s
IDL:ODdemo/calculator:1.0 = libODDEMO-CALCULATOR.so
uid = user
gid = group
mode = SYNC_END
ior = 1.1
locale = EUC
```

rep_id = IDL:ODdemo/calculator:1.0

インプリメンテーションリポジトリIDを指定します。

type = shared

サーバタイプを指定します。指定できるタイプには、persistent、shared、unsharedがあります。

binary = /home/guest/simple_s

サーバアプリケーションのパス名を指定します。

IDL:ODdemo/calculator:1.0 = libODDEMO-CALCULATOR.so

サーバアプリケーションのライブラリ名を指定します。
ライブラリは、環境変数LD_LIBRARY_PATHに指定されたディレクトリに格納する必要があります。

uid = user

サーバアプリケーション実行時のユーザIDを指定します。

gid = group

サーバアプリケーション実行時のグループIDを指定します。

mode = SYNC_END

サーバアプリケーションがCORBA-BOA-IMPL_IS_READY関数（または、CORBA-BOA-OBJ_IS_READY関数）発行時の動作モードを指定します。OOCOBOLサーバアプリケーションをスレッドモードで作成した場合は、SYNC_ENDを指定します。

ior = 1.1

連携するIORのバージョンを指定します。指定できるバージョンには、1.0、1.1があります。

locale = EUC

サーバアプリケーションのコード系を指定します。
開発環境として“NetCOBOL”を使用してUNICODEのアプリケーションを開発する場合は、“UNICODE”を指定してください。

G.3.6.2 ネーミングサービスへの登録

作成したサーバアプリケーションを他のアプリケーションからオブジェクトとしてアクセスできるようにするために、そのオブジェクトを識別するためのオブジェクトリファレンスを作成します。同時に作成したオブジェクトリファレンスをネーミングサービスに登録します。
オブジェクトを作成する方法として以下の方法があります。

- OD_or_admコマンドを使用します。
- サーバアプリケーションでBOAの機能を使用してオブジェクトリファレンスを作成し、ネーミングサービスに登録します。

■ OD_or_admコマンドを使用する方法

OD_or_admコマンドは、オブジェクトリファレンス作成後、ネーミングサービスに登録します。OD_or_admコマンドでの登録例と指定する情報を以下に示します。

例

ネーミングサービスへの登録 Windows32/64

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -L SJIS -n ODdemo::calculator
```

-c IDL:ODdemo/calculator:1.0

指定されたインタフェースリポジトリIDでオブジェクトリファレンスを登録します

-L SJIS

ネーミングサービスに登録するオブジェクトに対してコード系を指定します。

-n ODdemo::calculator

ネーミングサービスに登録するオブジェクト名を指定します。

ネーミングサービスへの登録 Solaris32/64 Linux32/64

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -L EUC -n ODdemo::calculator
```

-c IDL:ODdemo/calculator:1.0

指定されたインタフェースリポジトリIDでオブジェクトリファレンスを登録します

-L EUC

ネーミングサービスに登録するオブジェクトに対してコード系を指定します。

-n ODdemo::calculator

ネーミングサービスに登録するオブジェクト名を指定します。

■ サーバアプリケーションで作成する方法

以下の流れで処理を行います。

以下にプログラミングの例を示します。

```
 IDENTIFICATION          DIVISION.
 PROGRAM-ID.             "SIMPLE_S".
*
 ENVIRONMENT             DIVISION.
 CONFIGURATION           SECTION.
 REPOSITORY.
     COPY                CORBA--REP.
     COPY                COSNAMING--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY                CORBA--CONST.
     COPY                SIMPLE--CONST.
     COPY                COSNAMING--CONST.
     .
*
```

```
 DATA                   DIVISION.
 WORKING-STORAGE        SECTION.
 COPY                   CORBA--COPY.
 COPY                   COSNAMING--COPY.
*
 01 ORB                 USAGE OBJECT REFERENCE CORBA-ORB.
 01 BOA                 USAGE OBJECT REFERENCE CORBA-BOA.
 01 IMPL-REP            USAGE OBJECT REFERENCE FJ-IMPLEMENTATIONREP.
 01 INTF-REP            USAGE OBJECT REFERENCE CORBA-REPOSITORY.
 01 IMPL                USAGE OBJECT REFERENCE CORBA-IMPLEMENTATIONDEF.
 01 INTF                USAGE OBJECT REFERENCE CORBA-INTERFACEDEF.
 01 OBJ                 USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 REF-ID              USAGE OBJECT REFERENCE SEQUENCE-OCTET.
 01 NAMING-CONTEXT      USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT      USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NAME-ID             USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND           USAGE OBJECT REFERENCE CORBA-STRING.
 01 NEW-OBJ             USAGE OBJECT REFERENCE CORBA-OBJECT.

*
 01 EXCEPTION-ID        USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE  PIC   X(50).
 01 API-NAME            PIC   X(50).
*
*
 PROCEDURE              DIVISION.
*
 DECLARATIVES.
*
*
 ALREADY-ERR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CosNaming::NamingContext::AlreadyBound: " API-NAME.
 ALREADY-ERR.
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::UserException: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
 MAIN SECTION.
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
                        USING     "simple_s"
                                  FJ-OM_ORBID
                        RETURNING ORB.
*
     MOVE "CORBA::ORB::BOA_init" TO API-NAME.
     INVOKE ORB "BOA_INIT"
                        USING     "simple_s"
                                  CORBA-BOA_OAID
                        RETURNING BOA.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                        USING     CORBA-OBJECTID_LIGHTINTERF-001
```

```
                    RETURNING OBJ.
*
     INVOKE CORBA-REPOSITORY "NARROW"
                    USING     OBJ
                    RETURNING INTF-REP.
*
     MOVE "CORBA::Repository::lookup_id" TO API-NAME.
     INVOKE INTF-REP "LOOKUP_ID" USING     "IDL:ODdemo/calculator:1.0"
                                 RETURNING OBJ.

     INVOKE CORBA-INTERFACEDEF "NARROW" USING     OBJ
                                        RETURNING INTF.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                    USING     CORBA-OBJECTID_IMPLEMENTAT-001
                    RETURNING OBJ.
*
     INVOKE FJ-IMPLEMENTATIONREP "NARROW"
                    USING     OBJ
                    RETURNING IMPL-REP.
*
     MOVE "FJ::ImplementationRep::lookup_id" TO API-NAME.
     INVOKE IMPL-REP "LOOKUP_ID"
                    USING     "IDL:ODdemo/calculator:1.0"
                    RETURNING OBJ.
*
     INVOKE CORBA-IMPLEMENTATIONDEF "NARROW"
                    USING     OBJ
                    RETURNING IMPL.

     INVOKE SEQUENCE-OCTET "NEW" RETURNING REF-ID.
     MOVE "CORBA::BOA::create" TO API-NAME.
     INVOKE BOA "IDL-CREATE"
                    USING     REF-ID
                              INTF
                              IMPL
                    RETURNING NEW-OBJ.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                    USING     CORBA-ORB-OBJECTID_NAMESERVICE
                    RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                    USING     OBJ
                    RETURNING NAMING-CONTEXT.
*
     INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH"
                    USING     1
                    RETURNING NAME.
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE"
                    USING     "ODdemo::calculator"
                    RETURNING NAME-ID.
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE"
                    USING     " "
                    RETURNING NAME-KIND.
*
     INVOKE COSNAMING-NAMECOMPONENT "NEW"
                    RETURNING NAME-COMPONENT.
*
```

```
      SET IDL-ID OF NAME-COMPONENT TO NAME-ID.
      SET KIND OF NAME-COMPONENT TO NAME-KIND.
      INVOKE NAME "SET-VALUE" USING 1 NAME-COMPONENT.
*
      MOVE "CosNaming::NamingContext::unbind" TO API-NAME.
      INVOKE NAMING-CONTEXT "UNBIND"
                          USING     NAME.
*
      MOVE "CosNaming::NamingContext::bind" TO API-NAME.
      INVOKE NAMING-CONTEXT "BIND"
                          USING     NAME
                                    NEW-OBJ.
*
      MOVE "CORBA::BOA::impl_is_ready" TO API-NAME.
      INVOKE BOA "IMPL_IS_READY"
                          USING     IMPL.
*
      STOP RUN.
*
 END-MAIN.
*
 END PROGRAM "SIMPLE_S".
```

CORBA-BOA-CREATE関数呼出し時にインプリメンテーションリポジトリに定義されているデフォルトコード系でオブジェクトリファレンスを作成します。デフォルトコード系の設定はOD_impl_instまたはOD_set_envコマンドで行います。

G.3.7 アプリケーションのテスト

Windows32 Solaris32 Linux32

“アプリケーションの開発(COBOL)”の“6.7 アプリケーションのテスト”を参照してください。

ただし、サーバアプリケーションのテスト用モジュールの作成例は、以下のようになります。

Solaris32

```
 %CORBA=/opt/FSUNod/include/COBOL
 %export CORBA
 %cobol -WC,"TEST" -I/opt/FSUNod/include/oocob -R/opt/FSUNod/rep -M -c simple_s.cob
 %cobol -L/opt/FSUNod/lib -lOMoocob -o simple_s simple_s.o
 %cobol -WC,"TEST" -I/opt/FSUNod/include/oocob
-R/opt/FSUNod/rep -c ODDEMO-CALCULATOR-IMPL.cob
ODDEMO-CALCULATOR--NEW.cob
ODDEMO-CALCULATOR--NarrowSkel.cob
ODDEMO-CALCULATOR--helper.cob
ODDEMO-CALCULATOR--tie.cob
ODDEMO-CALCULATOR-RESULT--helper.cob
ODDEMO-CALCULATOR-ZEROPARAM--helper.cob
ODDEMO-CALCULATOR-ZEROPARAM.cob
ODDEMO-CALCULATOR.cob Simple--INIT.cob
 %cobol -G -dy -o libODDEMO-CALCULATOR.so -L/opt/FSUNod/lib -lOMoocob
ODDEMO-CALCULATOR-IMPL.o
ODDEMO-CALCULATOR--NEW.o
ODDEMO-CALCULATOR--NarrowSkel.o
ODDEMO-CALCULATOR--helper.o
ODDEMO-CALCULATOR--tie.o
ODDEMO-CALCULATOR-RESULT--helper.o
ODDEMO-CALCULATOR-ZEROPARAM--helper.o
ODDEMO-CALCULATOR-ZEROPARAM.o
ODDEMO-CALCULATOR.o Simple--INIT.o
```

Linux32

```
 %CORBA=/opt/FJSVod/include/COBOL
 %export CORBA
 %cobol -WC,"TEST" -I/opt/FJSVod/include/oocob -R/opt/FJSVod/rep -M -c simple_s.cob
 %cobol -L/opt/FJSVod/lib -lOMoocob -o simple_s simple_s.o
 %cobol -WC,"TEST" -I/opt/FJSVod/include/oocob -R/opt/FJSVod/rep -c
ODDEMO-CALCULATOR-IMPL.cob
ODDEMO-CALCULATOR--NEW.cob
ODDEMO-CALCULATOR--NarrowSkel.cob
ODDEMO-CALCULATOR--helper.cob
ODDEMO-CALCULATOR--tie.cob
ODDEMO-CALCULATOR-RESULT--helper.cob
ODDEMO-CALCULATOR-ZEROPARAM--helper.cob
ODDEMO-CALCULATOR-ZEROPARAM.cob
ODDEMO-CALCULATOR.cob Simple--INIT.cob
 %cobol -G -dy -o libODDEMO-CALCULATOR.so -L/opt/FJSVod/lib -lOMoocob
ODDEMO-CALCULATOR-IMPL.o
ODDEMO-CALCULATOR--NEW.o
ODDEMO-CALCULATOR--NarrowSkel.o
ODDEMO-CALCULATOR--helper.o
ODDEMO-CALCULATOR--tie.o
ODDEMO-CALCULATOR-RESULT--helper.o
ODDEMO-CALCULATOR-ZEROPARAM--helper.o
ODDEMO-CALCULATOR-ZEROPARAM.o
ODDEMO-CALCULATOR.o Simple--INIT.o
```

Windows64 Linux64

“アプリケーションの開発(COBOL)”の“6.7 アプリケーションのテスト”を参照してください。

G.3.8 データ型に対するマッピング

CORBAのデータ型を扱うために、OOCOBOLでは以下の登録集を提供します。

- リポジトリ宣言用:CORBA--REP.cbl
- 定数宣言用:CORBA--CONST.cbl
- データ型:CORBA--COPY.cbl

以下にデータ型のOOCOBOLでの定義を示します。

CORBAデータ型			OOCOBOLでの定義		備考
基本データ型	整数型	long	CORBA-LONG	PIC S9(9) COMP-5.	
		short	CORBA-SHORT	PIC S9(4) COMP-5.	
		unsigned long	CORBA-UNSIGNED-LONG	PIC 9(9) COMP-5.	
		unsigned short	CORBA-UNSIGNED-SHORT	PIC 9(4) COMP-5.	
		long long	CORBA-LONG-LONG	PIC S9(18) COMP-5.	
	浮動小数点型	float	CORBA-FLOAT	COMP-1.	
		double	CORBA-DOUBLE	COMP-2.	
	文字型	char	CORBA-CHAR	PIC X(1).	
		wchar	CORBA-WCHAR	PIC N(1).	
	オクテッド	octet	CORBA-OCTET	PIC X(1)	
	ブーリアン	boolean	CORBA-BOOLEAN	PIC 1(1)	

CORBAデータ型			OOCOBOLでの定義		備考
	文字列型	string		PIC X(n). (サイズ指定あり) クラス (サイズ指定なし)	“G.3.8.1 文字列型”参照
	ワイド文字列型	wstring		PIC N(n). (サイズ指定あり) クラス (サイズ指定なし)	“G.3.8.2 ワイド文字列型”参照
	列挙型	enum	CORBA-ENUM	PIC 9(9) COMP-5.	
シーケンス型		sequence		クラス	“G.3.8.3 シーケンス型”参照
構造体		struct		集団項目(固定長) クラス(可変長)	“G.3.8.4 構造体”参照
共用体		union		クラス	“G.3.8.5 共用体”参照
固定小数点型		fixed		PIC xx(n) PACKD-DECIMAL.	“G.3.8.6 固定小数点型”参照
オブジェクトリファレンス		Object	CORBA-OBJECT	OBJECT REFERENCE	
タイプコード		TypeCode	CORBA-TYPECODE	OBJECT REFERENCE	

G.3.8.1 文字列型

(1)IDLマッピング

IDL言語で文字列型stringを指定した場合、以下のデータ型で宣言します。

- サイズ指定あり:PIC X(n)
- サイズ指定なし:CORBA-STRINGクラス

以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface  stringtest{
         string op1(in string str1, out string str2, inout string str3);
    };
};
```

OOCOBOL

```
   IDENTIFICATION DIVISION.
   PROGRAM-ID. "ODSAMPLE-STRINGTEST-OP1".
  *
   ENVIRONMENT DIVISION.
  *
   DATA DIVISION.
```

```
 WORKING-STORAGE SECTION.
 COPY  CORBA--COPY.
 COPY  COSNAMING--COPY.
 COPY  SIMPLE--COPY.
 .
 01 OBJ  USAGE OBJECT REFERENCE CORBA-OBJECT.
*inパラメタ
 01 STR1   PIC X(50).
*outパラメタ
 01 STR2   USAGE OBJECT REFERENCE CORBA-STRING.
*inoutパラメタ
 01 STR3   USAGE OBJECT REFERENCE CORBA-STRING.
*復帰値
 01 STR-RET  USAGE OBJECT REFERENCE CORBA-STRING.
*
 PROCEDURE DIVISION.
 INVOKE OBJ "ODSAMPLE-STRINGTEST-OP1"
        USING  STR1
               STR2
               STR3
        RETURNING  STR-RET.
```

(2)クライアントアプリケーション例

クライアントアプリケーションでは、文字列型のパラメタとして以下のデータ項目を使用します。

- inパラメタ:PIC X(n) (サイズ指定あり)
- outパラメタ:CORBA-STRING(クラス)
- inoutパラメタ:CORBA-STRING(クラス)
- 復帰値:CORBA-STRING(クラス)

クライアントアプリケーション例を以下に示します。

```
 IDENTIFICATION  DIVISION.
 PROGRAM-ID.     "SIMPLE_C".
*
 ENVIRONMENT     DIVISION.
 CONFIGURATION   SECTION.
 REPOSITORY.
     COPY  CORBA--REP.
     COPY  SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY  CORBA--CONST.
     COPY  SIMPLE--CONST.
     .
*
 DATA            DIVISION.
 WORKING-STORAGE  SECTION.
 COPY  CORBA--COPY.
 COPY  COSNAMING--COPY.
 COPY  SIMPLE--COPY.
*inパラメタ
 01 STR1  PIC X(50).
*outパラメタ
 01 STR2  USAGE OBJECT REFERENCE CORBA-STRING.
*inoutパラメタ
 01 STR3  USAGE OBJECT REFERENCE CORBA-STRING.
*復帰値
 01 STR-RET  USAGE OBJECT REFERENCE CORBA-STRING.
```

```
*
 01 OBJ         USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 TARGET-OBJ  USAGE OBJECT REFERENCE ODDEMO-STRINGTEST.
 01 STR-WORK    PIC X(50).
*
 PROCEDURE DIVISION.
*inパラメタの設定
 MOVE    "IN" TO STR1.
* inoutパラメタの設定
 INVOKE  CORBA-STRING  "NEW" RETURNING STR3.
 INVOKE  STR3          "SET-VALUE" USING "INOUT".
*
 INVOKE  ODSAMPLE-STRINGTEST "NARROW"
         USING      OBJ
         RETURNING  TARGET-OBJ.
*
 INVOKE  TARGET-OBJ "OP1"
         USING  STR1
                STR2
                STR3
         RETURNING  STR-RET.
*復帰値の参照
 INVOKE  STR-RET  "GET-VALUE" RETURNING STR-WORK.
 DISPLAY "RETURN : " STR-WORK.
*outパラメタの参照
 INVOKE  STR2  "GET-VALUE" RETURNING STR-WORK.
 DISPLAY "OUT : " STR-WORK.
*inoutパラメタの参照
 INVOKE  STR3  "GET-VALUE" RETURNING STR-WORK.
 DISPLAY "INOUT : " STR-WORK.
END PROGRAM "SIMPLE_C".
```

(3)サーバアプリケーション例 Windows32 Solaris32 Linux32

サーバアプリケーションでは、文字列型のパラメタとして以下のデータ項目を使用します。

- inパラメタ:PIC X ANY LENGTH
- outパラメタ:CORBA-STRING(クラス)
- inoutパラメタ:CORBA-STRING(クラス)
- 復帰値:CORBA-STRING(クラス)

サーバアプリケーション例を以下に示します。

```
 CLASS-ID.     ODSAMPLE-STRINGTEST-IMPL.
     INHERITS  ODSAMPLE-STRINGTEST
*
 ENVIRONMENT   DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     COPY  CORBA--REP.
     COPY  SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY  CORBA--CONST.
     COPY  SIMPLE--CONST.
     .
*
 OBJECT.
 DATA             DIVISION.
 WORKING-STORAGE  SECTION.
```

```
 COPY  CORBA--COPY.
 COPY  SIMPLE--COPY.
*
 PROCEDURE  DIVISION.
 METHOD-ID  OP1 OVERRIDE.
*
 DATA             DIVISION.
 WORKING-STORAGE  SECTION.
 01 OP-NAME    PIC X(50).
 01 STR-WORK1  PIC X(50).
 01 STR-WORK2  PIC X(50).
*
 LINKAGE SECTION.
 01 STR1    PIC X ANY LENGTH.
 01 STR2    USAGE OBJECT REFERENCE   CORBA-STRING.
 01 STR3    USAGE OBJECT REFERENCE   CORBA-STRING.
 01 RESULT  USAGE OBJECT REFERENCE   CORBA-STRING.
*
 PROCEDURE DIVISION.
           USING  STR1
                  STR2
                  STR3
           RETURNING RESULT.
*outパラメタの設定
 INVOKE  CORBA-STRING "NEW"  RETURNING STR2.
 INVOKE  STR2         "SET-VALUE" USING "OP1:OUT".
*inoutパラメタの参照/設定
 MOVE "OP1:" TO OP-NAME.
 INVOKE  STR3  "GET-VALUE" RETURNING STR-WORK1.
 STRING  OP-NAME   DELIMITED BY SIZE
         STR-WORK1 DELIMITED BY SIZE
         INTO      STR-WORK2.
 INVOKE  STR3  "SET-VALUE" USING STR-WORK2.
*復帰値の設定
 INVOKE  CORBA-STRING  "NEW" RETURNING RESULT.
 INVOKE  RESULT        "SET-VALUE" USING "OP1:RESULT OK".
*
 END  METHOD OP1.
 END  OBJECT.
 END  CLASS  ODSAMPLE-STRINGTEST-IMPL.
```

G.3.8.2 ワイド文字列型

(1)IDLマッピング

IDL言語でワイド文字列型wstringを指定した場合、以下のデータ型で宣言します。

- サイズ指定あり:PIC N(n)
- サイズ指定なし:CORBA-WSTRINGクラス

以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
    interface  wstringtest{
         wstring op1(in wstring str1, out wstring str2, inout wstring str3);
```

```
    };
};
```

OOCOBOL

```
  IDENTIFICATION DIVISION.
  PROGRAM-ID. "ODSAMPLE-WSTRINGTEST-OP1".
 *
  ENVIRONMENT DIVISION.
 *
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  COPY  CORBA--COPY.
  COPY  COSNAMING--COPY.
  COPY  SIMPLE--COPY.
  .
  01 OBJ  USAGE OBJECT REFERENCE CORBA-OBJECT.
 *inパラメタ
  01 STR1   PIC N(50).
 *outパラメタ
  01 STR2   USAGE OBJECT REFERENCE CORBA-WSTRING.
 *inoutパラメタ
  01 STR3   USAGE OBJECT REFERENCE CORBA-WSTRING.
 *復帰値
  01 STR-RET  USAGE OBJECT REFERENCE CORBA-WSTRING.
 *
  PROCEDURE DIVISION.
  INVOKE OBJ "ODSAMPLE-WSTRINGTEST-OP1"
         USING  STR1
                STR2
                STR3
         RETURNING  STR-RET.
```

(2)クライアントアプリケーション例

クライアントアプリケーションでは、ワイド文字列型のパラメタとして以下のデータ項目を使用します。

- inパラメタ:PIC N(n) (サイズ指定あり)
- outパラメタ:CORBA-WSTRING(クラス)
- inoutパラメタ:CORBA-WSTRING(クラス)
- 復帰値:CORBA-WSTRING(クラス)

クライアントアプリケーション例を以下に示します。

```
  IDENTIFICATION  DIVISION.
  PROGRAM-ID.     "SIMPLE_C".
 *
  ENVIRONMENT     DIVISION.
  CONFIGURATION   SECTION.
  REPOSITORY.
      COPY  CORBA--REP.
      COPY  SIMPLE--REP.
      .
  SPECIAL-NAMES.
      SYMBOLIC CONSTANT
      COPY  CORBA--CONST.
```

```
      COPY  SIMPLE--CONST.
      .
*
 DATA             DIVISION.
 WORKING-STORAGE  SECTION.
 COPY  CORBA--COPY.
 COPY  SIMPLE--COPY.
*inパラメタ
 01 STR1  PIC N(50).
*outパラメタ
 01 STR2  USAGE OBJECT REFERENCE CORBA-WSTRING.
*inoutパラメタ
 01 STR3  USAGE OBJECT REFERENCE CORBA-WSTRING.
*復帰値
 01 STR-RET  USAGE OBJECT REFERENCE CORBA-WSTRING.
*
 01 OBJ         USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 TARGET-OBJ  USAGE OBJECT REFERENCE ODDEMO-WSTRINGTEST.
 01 STR-WORK    PIC N(50).
*
 PROCEDURE DIVISION.
*inパラメタの設定
 MOVE    N"イン" TO STR1.
* inoutパラメタの設定
 INVOKE  CORBA-WSTRING  "NEW" RETURNING STR3.
 INVOKE  STR3           "SET-VALUE" USING N"インアウト".
*
 INVOKE  ODSAMPLE-WSTRINGTEST "NARROW"
         USING      OBJ
         RETURNING  TARGET-OBJ.
*
 INVOKE  TARGET-OBJ "OP1"
         USING  STR1
                STR2
                STR3
         RETURNING  STR-RET.
*復帰値の参照
 INVOKE  STR-RET  "GET-VALUE" RETURNING STR-WORK.
 DISPLAY "RETURN : " STR-WORK.
*outパラメタの参照
 INVOKE  STR2  "GET-VALUE" RETURNING STR-WORK.
 DISPLAY "OUT : " STR-WORK.
*inoutパラメタの参照
 INVOKE  STR3  "GET-VALUE" RETURNING STR-WORK.
 DISPLAY "INOUT : " STR-WORK.
END PROGRAM "SIMPLE_C".
```

(3)サーバアプリケーション例 Windows32 Solaris32 Linux32

サーバアントアプリケーションでは、ワイド文字列型のパラメタとして以下のデータ項目を使用します。

- inパラメタ:PIC N ANY LENGTH
- outパラメタ:CORBA-WSTRING(クラス)
- inoutパラメタ:CORBA-WSTRING(クラス)
- 復帰値:CORBA-WSTRING(クラス)

サーバアプリケーション例を以下に示します。

```
 CLASS-ID.     ODSAMPLE-WSTRINGTEST-IMPL.
     INHERITS  ODSAMPLE-WSTRINGTEST
*
```

```
 ENVIRONMENT   DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     COPY  CORBA--REP.
     COPY  SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY  CORBA--CONST.
     COPY  SIMBOL--CONST.
     .
*
 OBJECT.
 DATA             DIVISION.
 WORKING-STORAGE  SECTION.
 COPY  CORBA--COPY.
*
 PROCEDURE  DIVISION.
 METHOD-ID  OP1 OVERRIDE.
*
 DATA             DIVISION.
 WORKING-STORAGE  SECTION.
 01 OP-NAME    PIC N(50).
 01 STR-WORK1  PIC N(50).
 01 STR-WORK2  PIC N(50).
*
 LINKAGE SECTION.
 01 STR1    PIC N ANY LENGTH.
 01 STR2    USAGE OBJECT REFERENCE   CORBA-WSTRING.
 01 STR3    USAGE OBJECT REFERENCE   CORBA-WSTRING.
 01 RESULT  USAGE OBJECT REFERENCE   CORBA-WSTRING.
*
 PROCEDURE DIVISION.
           USING  STR1
                  STR2
                  STR3
           RETURNING RESULT.
*outパラメタの設定
 INVOKE  CORBA-WSTRING "NEW"  RETURNING STR2.
 INVOKE  STR2          "SET-VALUE" USING "ＯＰ１：ＯＵＴ".
*inoutパラメタの参照/設定
 MOVE N"メソッドOP1:" TO OP-NAME.
 INVOKE  STR3  "GET-VALUE" RETURNING STR-WORK1.
 STRING  OP-NAME   DELIMITED BY SIZE
         STR-WORK1 DELIMITED BY SIZE
         INTO      STR-WORK2.
 INVOKE  STR3  "SET-VALUE" USING STR-WORK2.
*復帰値の設定
 INVOKE  CORBA-WSTRING  "NEW" RETURNING RESULT.
 INVOKE  RESULT         "SET-VALUE" USING N"メソッドＯＰ１：ＲＥＳＵＬＴ".
*
 END  METHOD OP1.
 END  OBJECT.
 END  CLASS  ODSAMPLE-WSTRINGTEST-IMPL.
```

G.3.8.3 シーケンス型

(1) IDLマッピング

IDL言語でシーケンス型sequenceを指定した場合、クラス(シーケンスクラス)でデータ宣言します。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

［ファイル名: simple.idl］

```
module ODsample{
    interface  seqtest{
          typedef sequence<long>  sampleseq;
          sampleseq op1(in sampleseq seq1, out sampleseq seq2, inout sampleseq seq3 );
          typedef sequence<string> sampleseq2;
          sampleseq2 op2(in sampleseq2 seq1, out sampleseq2 seq2, inout sampleseq2 seq3 );
     };
};
```

OOCOBOL

［ファイル名: simple--copy.cbl］

```
 * データ型宣言(typedef)
  01  ODSAMPLE-SEQTEST-SAMPLESEQ TYPEDEF OBJECT REFERENCE SEQUENCE-LONG.
```

［ファイル名: sequence-long.cob］

```
 * シーケンスクラス(sequence<long>)
  CLASS-ID. sequence-long AS "sequence-long" INHERITS SEQUENCEBASE.
 *
  FACTORY.
   PROCEDURE DIVISION.
 *
 * NEW-WITH-LENGTH METHOD
    METHOD-ID. NEW-WITH-LENGTH AS "NEW-WITH-LENGTH".
    DATA DIVISION.
    LINKAGE SECTION.
    COPY CORBA--COPY.
    01  O--SEQ-LENGTH TYPE CORBA-UNSIGNED-LONG.
    01  O--SEQ-OBJ OBJECT REFERENCE SELF.
    PROCEDURE DIVISION
      USING O--SEQ-LENGTH RETURNING O--SEQ-OBJ
      RAISING CORBA-SYSTEMEXCEPTION.
    END METHOD NEW-WITH-LENGTH.
  END FACTORY.
 *
  OBJECT.
   PROCEDURE DIVISION.
 *
 * SEQ-MAXIMUM PROPERTY : GET
    METHOD-ID. GET PROPERTY SEQ-MAXIMUM.
    DATA DIVISION.
    LINKAGE SECTION.
    01 O--SEQ-MAXIMUM TYPE CORBA-UNSIGNED-LONG.
    PROCEDURE DIVISION RETURNING O--SEQ-MAXIMUM.
      MOVE O--MAXIMUM TO O--SEQ-MAXIMUM.
    END METHOD.
 *
 * SEQ-LENGTH PROPERTY : GET
    METHOD-ID. GET PROPERTY SEQ-LENGTH.
    DATA DIVISION.
    LINKAGE SECTION.
    01  O--LENGTH-RESULT TYPE CORBA-UNSIGNED-LONG.
    PROCEDURE DIVISION RETURNING O--LENGTH-RESULT.
      MOVE O--LENGTH TO O--LENGTH-RESULT.
```

```
    END METHOD.
*
* SEQ-LENGTH PROPERTY : SET
    METHOD-ID. SET PROPERTY SEQ-LENGTH.
    DATA DIVISION.
    LINKAGE SECTION.
    01  O--LENGTH-PARAM TYPE CORBA-UNSIGNED-LONG.
    PROCEDURE DIVISION USING O--LENGTH-PARAM
      RAISING CORBA-SYSTEMEXCEPTION.
    END METHOD.
*
* GET-VALUE METHOD
    METHOD-ID. GET-VALUE AS "GET-VALUE".
    DATA DIVISION.
    LINKAGE SECTION.
    01  O--ELEMS TYPE CORBA-UNSIGNED-LONG.
    01  O--VALUE TYPE CORBA-LONG.
    PROCEDURE DIVISION USING O--ELEMS RETURNING O--VALUE.
    END METHOD GET-VALUE.
*
* SET-VALUE METHOD
    METHOD-ID. SET-VALUE AS "SET-VALUE".
    DATA DIVISION.
    LINKAGE SECTION.
    01  O--ELEMS TYPE CORBA-UNSIGNED-LONG.
    01  O--VALUE TYPE CORBA-LONG.
    PROCEDURE DIVISION USING O--ELEMS O--VALUE.
    END METHOD SET-VALUE.
*
* CLONE METHOD
    METHOD-ID. CLONE AS "CLONE".
    DATA DIVISION.
    LINKAGE SECTION.
    01 O--CLONE-OBJ OBJECT REFERENCE sequence-long.
    PROCEDURE DIVISION RETURNING O--CLONE-OBJ
      RAISING CORBA-SYSTEMEXCEPTION.
    END METHOD CLONE.
*
 END OBJECT.
 END CLASS sequence-long.

* シーケンスクラス(sequence<string>)
 CLASS-ID. sequence-string AS "sequence-string" INHERITS SEQUENCEBASE.
*
 FACTORY.
  PROCEDURE DIVISION.
*
* NEW-WITH-LENGTH METHOD
    METHOD-ID. NEW-WITH-LENGTH AS "NEW-WITH-LENGTH".
    DATA DIVISION.
    LINKAGE SECTION.
    COPY CORBA--COPY.
    01  O--SEQ-LENGTH TYPE CORBA-UNSIGNED-LONG.
    01  O--SEQ-OBJ OBJECT REFERENCE SELF.
    PROCEDURE DIVISION
      USING O--SEQ-LENGTH RETURNING O--SEQ-OBJ
      RAISING CORBA-SYSTEMEXCEPTION.
    END METHOD NEW-WITH-LENGTH.
 END FACTORY.
*
 OBJECT.
  PROCEDURE DIVISION.
*
```

```
* SEQ-MAXIMUM PROPERTY : GET
   METHOD-ID. GET PROPERTY SEQ-MAXIMUM.
   DATA DIVISION.
   LINKAGE SECTION.
   01 O--SEQ-MAXIMUM TYPE CORBA-UNSIGNED-LONG.
   PROCEDURE DIVISION RETURNING O--SEQ-MAXIMUM.
     MOVE O--MAXIMUM TO O--SEQ-MAXIMUM.
   END METHOD.
*
* SEQ-LENGTH PROPERTY : GET
   METHOD-ID. GET PROPERTY SEQ-LENGTH.
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--LENGTH-RESULT TYPE CORBA-UNSIGNED-LONG.
   PROCEDURE DIVISION RETURNING O--LENGTH-RESULT.
     MOVE O--LENGTH TO O--LENGTH-RESULT.
   END METHOD.
*
* SEQ-LENGTH PROPERTY : SET
   METHOD-ID. SET PROPERTY SEQ-LENGTH.
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--LENGTH-PARAM TYPE CORBA-UNSIGNED-LONG.
   PROCEDURE DIVISION USING O--LENGTH-PARAM
     RAISING CORBA-SYSTEMEXCEPTION.
   END METHOD.
*
* GET-VALUE METHOD
   METHOD-ID. GET-VALUE AS "GET-VALUE".
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--ELEMS TYPE CORBA-UNSIGNED-LONG.
   01  O--VALUE USAGE OBJECT REFERENCE CORBA-STRING.
   PROCEDURE DIVISION USING O--ELEMS RETURNING O--VALUE.
   END METHOD GET-VALUE.
*
* SET-VALUE METHOD
   METHOD-ID. SET-VALUE AS "SET-VALUE".
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--ELEMS TYPE CORBA-UNSIGNED-LONG.
   01  O--VALUE USAGE OBJECT REFERENCE CORBA-STRING.
   PROCEDURE DIVISION USING O--ELEMS O--VALUE.
   END METHOD SET-VALUE.
*
* CLONE METHOD
   METHOD-ID. CLONE AS "CLONE".
   DATA DIVISION.
   LINKAGE SECTION.
   01 O--CLONE-OBJ OBJECT REFERENCE sequence-string.
   PROCEDURE DIVISION RETURNING O--CLONE-OBJ
     RAISING CORBA-SYSTEMEXCEPTION.
   END METHOD CLONE.
*
 END OBJECT.
 END CLASS sequence-string.
```

(2)シーケンスクラスのメソッド／プロパティ

シーケンスクラスで定義されているメソッド／プロパティを以下に示します。

メソッド／プロパティ	機能
NEW-WITH-LENGTHメソッド	(ファクトリメソッド)指定の長さでシーケンスクラスを作成します。
GET-VALUEメソッド	(オブジェクトメソッド)指定番号の要素の値を取得します。
SET-VALUEメソッド	(オブジェクトメソッド)指定番号の要素に値を設定します。
CLONEメソッド	(オブジェクトメソッド)シーケンスクラスを複製します。
SEQ-MAXIMUMプロパティ	シーケンス最大長の値を取得します。
SEQ-LENGTHプロパティ	シーケンス長の値の取得／設定を行います。

メソッド／プロパティの使用例を以下に示します。

```
* NEW-WITH-LENGTHメソッド
*   シーケンスクラスをSEQとして作成
 01  SEQ       USAGE OBJECT REFERENCE  SEQUENCE-LONG.
 01  SEQ-LEN   USAGE OBJECT REFERENCE  CORBA-UNSIGNED-LONG.
 INVOKE SEQUENCE-LONG "NEW-WITH-LENGTH"
        USING SEQ-LEN  RETURNING SEQ.

* GET-VALUEメソッド
*   1番目の要素の値をSEQ-GVALに取得
 01  SEQ       USAGE OBJECT REFERENCE SEQUENCE-LONG.
 01  SEQ-GVAL  TYPE  CORBA-LONG.
 INVOKE SEQ "GET-VALUE"  USING 1  RETURNING SEQ-GVAL.

* SET-VALUEメソッド
*   1番目の要素にSEQ-SVALを設定
 01  SEQ       USAGE OBJECT REFERENCE SEQUENCE-LONG.
 01  SEQ-SVAL  TYPE  CORBA-LONG.
 INVOKE SEQ "SET-VALUE"  USING 1 SEQ-SVAL.

* CLONEメソッド
*   シーケンスクラスをSEQ-CLONEとして複製
 01  SEQ       USAGE OBJECT REFERENCE SEQUENCE-LONG.
 01  SEQ-CLON  USAGE OBJECT REFERENCE SEQUENCE-LONG.
 INVOKE SEQ "CLONE"  RETURNING SEQ-CLON.

* SEQ-MAXIMUMプロパティ
*   シーケンス最大長の値をSEQ-MAXに取得
 01  SEQ       USAGE OBJECT REFERENCE SEQUENCE-LONG.
 01  SEQ-MAX   TYPE  CORBA-UNSIGNED-LONG.
 MOVE SEQ-MAXIMUM OF SEQ  TO SEQ-MAX.

* SEQ-LENGTHプロパティ
*   シーケンス長の値をSEQ-LENに取得
 01  SEQ       USAGE OBJECT REFERENCE SEQUENCE-LONG.
 01  SEQ-LEN   TYPE  CORBA-UNSIGNED-LONG.
 MOVE SEQ-LENGTH OF SEQ  TO SEQ-LEN.

*  シーケンス長にSEQ-LENを設定
 01  SEQ       USAGE OBJECT REFERENCE SEQUENCE-LONG.
 01  SEQ-LEN   TYPE  CORBA-UNSIGNED-LONG.
 MOVE SEQ-LEN  TO SEQ-LENGTH OF SEQ.
```

(3)クライアントアプリケーション例

クライアントアプリケーション例を以下に示します。

```
 IDENTIFICATION          DIVISION.
 PROGRAM-ID.             "CLIENT-MAIN".
*
 ENVIRONMENT             DIVISION.
 CONFIGURATION           SECTION.
 REPOSITORY.
     COPY                CORBA--REP.
     COPY                COSNAMING--REP.
     COPY                SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY                CORBA--CONST.
     COPY                COSNAMING--CONST.
     COPY                SIMPLE--CONST.
     .
*
 DATA                    DIVISION.
 WORKING-STORAGE         SECTION.
 COPY                    CORBA--COPY.
 COPY                    COSNAMING--COPY.
 COPY                    SIMPLE--COPY.
*
 01 ORB                  USAGE OBJECT REFERENCE CORBA-ORB.
 01 OBJ                  USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 NAMING-CONTEXT       USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                 TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT       USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NAME-ID              USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND            USAGE OBJECT REFERENCE CORBA-STRING.
*
 01 EXCEPTION-ID         USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE   PIC   X(50).
 01 API-NAME             PIC   X(50).
*
 01 TARGET-OBJ           USAGE OBJECT REFERENCE ODSAMPLE-SEQTEST.
 01 RESULT               TYPE                   ODSAMPLE-SEQTEST-SAMPLESEQ.
 01 SEQ1                 TYPE                   ODSAMPLE-SEQTEST-SAMPLESEQ.
 01 SEQ2                 TYPE                   ODSAMPLE-SEQTEST-SAMPLESEQ.
 01 SEQ3                 TYPE                   ODSAMPLE-SEQTEST-SAMPLESEQ.
 01 SEQ-VALUE            TYPE                   CORBA-LONG.
 01 OP2-RESULT           TYPE                   ODSAMPLE-SEQTEST-SAMPLESEQ2.
 01 OP2-SEQ1             TYPE                   ODSAMPLE-SEQTEST-SAMPLESEQ2.
 01 OP2-SEQ2             TYPE                   ODSAMPLE-SEQTEST-SAMPLESEQ2.
 01 OP2-SEQ3             TYPE                   ODSAMPLE-SEQTEST-SAMPLESEQ2.
 01 OP2-SEQ-VALUE        USAGE OBJECT REFERENCE CORBA-STRING.
 01 I                    TYPE                   CORBA-UNSIGNED-LONG.
*
 PROCEDURE               DIVISION.
*
 DECLARATIVES.
*
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::Exception: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
```

```
 MAIN SECTION.
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
                         USING     "simple_c"
                                   FJ-OM_ORBID
                         RETURNING ORB.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                         USING     CORBA-ORB-OBJECTID_NAMESERVICE
                         RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                         USING     OBJ
                         RETURNING NAMING-CONTEXT.
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE"
                         USING     "ODsample::seqtest"
                         RETURNING NAME-ID.
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE"
                         USING     " "
                         RETURNING NAME-KIND.
*
     INVOKE COSNAMING-NAMECOMPONENT "NEW"
                         RETURNING NAME-COMPONENT.
*
     SET IDL-ID OF NAME-COMPONENT TO NAME-ID.
     SET KIND   OF NAME-COMPONENT TO NAME-KIND.
*
     INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH"
                         USING     1
                         RETURNING NAME.
*
     INVOKE NAME "SET-VALUE"
                         USING     1
                                   NAME-COMPONENT.
*
     MOVE "CosNaming::NamingContext::resolve" TO API-NAME.
     INVOKE NAMING-CONTEXT "RESOLVE"
                         USING     NAME
                         RETURNING OBJ.
*
     INVOKE ODSAMPLE-SEQTEST "NARROW"
                         USING     OBJ
                         RETURNING TARGET-OBJ.
*
*  sequence<long>
*
*  SET IN PARAMETER
     INVOKE SEQUENCE-LONG "NEW" RETURNING SEQ1.
     MOVE 2 TO SEQ-LENGTH OF SEQ1.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 2
       MOVE I TO SEQ-VALUE
       INVOKE SEQ1 "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.
*
*  SET INOUT PARAMETER
     INVOKE SEQUENCE-LONG "NEW" RETURNING SEQ3.
     MOVE 3 TO SEQ-LENGTH OF SEQ3.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 3
       COMPUTE SEQ-VALUE = I * 10
```

```
       INVOKE SEQ3 "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.
*
     MOVE "ODdemo::seqtest::op1" TO API-NAME.
     INVOKE TARGET-OBJ "OP1"
                         USING     SEQ1
                                   SEQ2
                                   SEQ3
                         RETURNING RESULT.
*
*  sequence<string>
*
*  SET IN PARAMETER
     INVOKE SEQUENCE-CORBA-STRING "NEW" RETURNING OP2-SEQ1.
     MOVE 2 TO SEQ-LENGTH OF OP2-SEQ1.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 2
       INVOKE CORBA-STRING "NEW-WITH-VALUE" USING     "TEST"
                                            RETURNING OP2-SEQ-VALUE
       INVOKE OP2-SEQ1 "SET-VALUE" USING I OP2-SEQ-VALUE
     END-PERFORM.
*
*  SET INOUT PARAMETER
     INVOKE SEQUENCE-CORBA-STRING "NEW" RETURNING OP2-SEQ3.
     MOVE 3 TO SEQ-LENGTH OF OP2-SEQ3.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 3
       INVOKE CORBA-STRING "NEW-WITH-VALUE" USING     "TEST"
                                            RETURNING OP2-SEQ-VALUE
       INVOKE OP2-SEQ3 "SET-VALUE" USING I OP2-SEQ-VALUE
     END-PERFORM.
*
     MOVE "ODdemo::seqtest::op2" TO API-NAME.
     INVOKE TARGET-OBJ "OP2"
                         USING     OP2-SEQ1
                                   OP2-SEQ2
                                   OP2-SEQ3
                         RETURNING OP2-RESULT.
*
     STOP RUN.
*
 END-MAIN.
*
 END PROGRAM "CLIENT-MAIN".
```

(4)サーバアプリケーション例 Windows32 Solaris32 Linux32

サーバアプリケーション例を以下に示します。

```
 CLASS-ID.             ODSAMPLE-SEQTEST-IMPL
     INHERITS          ODSAMPLE-SEQTEST.
*
 ENVIRONMENT           DIVISION.
 CONFIGURATION         SECTION.
 REPOSITORY.
     COPY              CORBA--REP.
     COPY              SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY              CORBA--CONST.
     COPY              SIMPLE--CONST.
     .
*
```

```
 OBJECT.
  DATA                  DIVISION.
  WORKING-STORAGE       SECTION.
  COPY                  CORBA--COPY.
  COPY                  SIMPLE--COPY.
*
  PROCEDURE             DIVISION.
*
   METHOD-ID.           OP1 OVERRIDE.
   DATA                 DIVISION.
   WORKING-STORAGE      SECTION.
   01 SEQ-VALUE         TYPE                    CORBA-LONG.
   01 I                 TYPE                    CORBA-UNSIGNED-LONG.
   LINKAGE              SECTION.
   01 SEQ1              TYPE                    ODSAMPLE-SEQTEST-SAMPLESEQ.
   01 SEQ2              TYPE                    ODSAMPLE-SEQTEST-SAMPLESEQ.
   01 SEQ3              TYPE                    ODSAMPLE-SEQTEST-SAMPLESEQ.
   01 RESULT            TYPE                    ODSAMPLE-SEQTEST-SAMPLESEQ.
*
   PROCEDURE            DIVISION
                        USING     SEQ1
                                  SEQ2
                                  SEQ3
                        RETURNING RESULT
                        RAISING   CORBA-SYSTEMEXCEPTION.
*
*  SET OUT PARAMETER
     INVOKE SEQUENCE-LONG "NEW" RETURNING SEQ2.
     MOVE 4 TO SEQ-LENGTH OF SEQ1.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 4
       COMPUTE SEQ-VALUE = I * 100
       INVOKE SEQ2 "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.
*
*  SET INOUT PARAMETER
     INVOKE SEQUENCE-LONG "NEW" RETURNING SEQ3.
     MOVE 5 TO SEQ-LENGTH OF SEQ3.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 5
       COMPUTE SEQ-VALUE = I * 1000
       INVOKE SEQ3 "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.
*
*  SET RESULT
     INVOKE SEQUENCE-LONG "NEW" RETURNING RESULT.
     MOVE 6 TO SEQ-LENGTH OF SEQ3.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 6
       COMPUTE SEQ-VALUE = I * 10000
       INVOKE RESULT "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.
*
   END METHOD OP1.
*
*
*
   METHOD-ID.           OP2 OVERRIDE.
   DATA                 DIVISION.
   WORKING-STORAGE      SECTION.
   01 SEQ-VALUE         USAGE OBJECT REFERENCE CORBA-STRING.
   01 I                 TYPE                    CORBA-UNSIGNED-LONG.
   LINKAGE              SECTION.
   01 SEQ1              TYPE                    ODSAMPLE-SEQTEST-SAMPLESEQ2.
   01 SEQ2              TYPE                    ODSAMPLE-SEQTEST-SAMPLESEQ2.
   01 SEQ3              TYPE                    ODSAMPLE-SEQTEST-SAMPLESEQ2.
```

```
   01 RESULT            TYPE                   ODSAMPLE-SEQTEST-SAMPLESEQ2.
*
   PROCEDURE            DIVISION
                        USING     SEQ1
                                  SEQ2
                                  SEQ3
                        RETURNING RESULT
                        RAISING   CORBA-SYSTEMEXCEPTION.
*
*  SET OUT PARAMETER
     INVOKE SEQUENCE-CORBA-STRING "NEW" RETURNING SEQ2.
     MOVE 4 TO SEQ-LENGTH OF SEQ2.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 4
       INVOKE CORBA-STRING "NEW-WITH-VALUE" USING         "TEST"
                                            RETURNING     SEQ-VALUE
       INVOKE SEQ2 "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.
*
*  SET INOUT PARAMETER
     INVOKE SEQUENCE-CORBA-STRING "NEW" RETURNING SEQ3.
     MOVE 5 TO SEQ-LENGTH OF SEQ3.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 5
       INVOKE CORBA-STRING "NEW-WITH-VALUE" USING         "TEST"
                                            RETURNING     SEQ-VALUE
       INVOKE SEQ3 "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.
*
*  SET RESULT
     INVOKE SEQUENCE-CORBA-STRING "NEW" RETURNING RESULT.
     MOVE 6 TO SEQ-LENGTH OF RESULT.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 6
       INVOKE CORBA-STRING "NEW-WITH-VALUE" USING         "TEST"
                                            RETURNING     SEQ-VALUE
       INVOKE RESULT "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.
*
   END METHOD OP2.
*
 END OBJECT.
*
 END CLASS ODSAMPLE-SEQTEST-IMPL.
```

G.3.8.4 構造体

(1)IDLマッピング

IDL言語で構造体structを指定した場合、以下のデータ型で宣言します。

- 構造体(固定長):集団項目
- 構造体(可変長):クラス(構造体クラス)

以降では、以下のIDL定義例をもとに説明します。

IDL言語

［ファイル名: simple.idl］

```
module ODsample{
    struct samplefix {    // 構造体(固定長)
```

```
        long   para1;
        long   para2;
    };
    struct samplevar {    // 構造体(可変長)
        long   para1;
        string para2;
    };
    interface  structtest{
        samplefix  op2(
           in samplefix str1,
           out samplefix str2,
           inout samplefix str3
        );
        samplevar  op1(
            in samplevar str1,
            out samplevar str2,
            inout samplevar str3
        );
    };
};
```

OOCOBOL

［ファイル名: simple--copy.cbl］

```
 * 集団項目(構造体(固定長))
  01  ODSAMPLE-SAMPLEFIX TYPEDEF.
      02  PARA1 TYPE CORBA-LONG.
      02  PARA2 TYPE CORBA-LONG.
```

［ファイル名: ODSAMPLE-SAMPLEVAR.COB］

```
 * 構造体クラス(構造体(可変長))
  CLASS-ID. ODSAMPLE-SAMPLEVAR AS "ODSAMPLE-SAMPLEVAR"
      INHERITS STRUCTBASE.
 *
  OBJECT.
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   COPY CORBA--COPY.
 *
 * PROPERTY
   01  PARA1 TYPE CORBA-LONG PROPERTY.
   01  PARA2 OBJECT REFERENCE CORBA-STRING PROPERTY.
 *
   PROCEDURE DIVISION.
 *
 * CLONE METHOD
    METHOD-ID. CLONE AS "CLONE".
    DATA DIVISION.
    LINKAGE SECTION.
    01  O--CLONE-OBJ OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
    PROCEDURE DIVISION RETURNING O--CLONE-OBJ.
    END METHOD CLONE.
  END OBJECT.
  END CLASS ODSAMPLE-SAMPLEVAR.
```

(2)構造体クラスのメソッド／プロパティ

構造体クラスで定義されているメソッド／プロパティを以下に示します。

メソッド／プロパティ	機能
CLONEメソッド	(オブジェクトメソッド)構造体クラスを複製します。
メンバプロパティ	メンバの値の取得／設定を行います。

メソッド／プロパティの使用例を以下に示します。

```
＊ CLONEメソッド
＊   構造体クラスをSTV-CLONとして複製
 01  STV       USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01  STV-CLON  USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 INVOKE STV "CLONE"  RETURNING STV-CLON.

＊ メンバプロパティ(クラス以外のデータ型)
＊   メンバPARA1の値をSTV-MEMTに取得
 01  STV       USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01  STV-MEMT  TYPE  CORBA-LONG.
 MOVE PARA1 OF STV  TO STV-MEMT.

＊   メンバPARA1にSTV-MEMTを設定
 01  STV       USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01  STV-MEMT  TYPE  CORBA-LONG.
 MOVE STV-MEMT  TO PARA1 OF STV.

＊ メンバプロパティ(クラス)
＊   メンバPARA2の値をSTV-MEMCに取得
 01  STV       USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01  STV-MEMC  USAGE OBJECT REFERENCE CORBA-STRING.
 SET STV-MEMC  TO PARA2 OF STV.

＊   メンバPARA2にSTV-MEMCを設定
 01  STV       USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01  STV-MEMC  USAGE OBJECT REFERENCE CORBA-STRING.
 SET PARA2 OF STV  TO STV-MEMC.
```

(3)クライアントアプリケーション例

クライアントアプリケーション例を以下に示します。

```
  IDENTIFICATION          DIVISION.
  PROGRAM-ID.             "CLIENT-MAIN".
*
  ENVIRONMENT             DIVISION.
  CONFIGURATION           SECTION.
  REPOSITORY.
      COPY                CORBA--REP.
      COPY                COSNAMING--REP.
      COPY                SIMPLE--REP.
      .
  SPECIAL-NAMES.
      SYMBOLIC CONSTANT
      COPY                CORBA--CONST.
      COPY                COSNAMING--CONST.
      COPY                SIMPLE--CONST.
      .
*
  DATA                    DIVISION.
  WORKING-STORAGE         SECTION.
  COPY                    CORBA--COPY.
  COPY                    COSNAMING--COPY.
  COPY                    SIMPLE--COPY.
```

```
*
 01 ORB                 USAGE OBJECT REFERENCE CORBA-ORB.
 01 OBJ                 USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 NAMING-CONTEXT      USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT      USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NAME-ID             USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND           USAGE OBJECT REFERENCE CORBA-STRING.
*
 01 EXCEPTION-ID        USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE  PIC   X(50).
 01 API-NAME            PIC   X(50).
*
 01 TARGET-OBJ          USAGE OBJECT REFERENCE ODSAMPLE-STRUCTTEST.
 01 FIX0                TYPE                   ODSAMPLE-SAMPLEFIX.
 01 FIX1                TYPE                   ODSAMPLE-SAMPLEFIX.
 01 FIX2                TYPE                   ODSAMPLE-SAMPLEFIX.
 01 FIX3                TYPE                   ODSAMPLE-SAMPLEFIX.
 01 VAR0                USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01 VAR1                USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01 VAR2                USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01 VAR3                USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01 STR-WORK            USAGE OBJECT REFERENCE CORBA-STRING.
*
 PROCEDURE             DIVISION.
*
 DECLARATIVES.
*
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::Exception: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
 MAIN SECTION.
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
                         USING     "simple_c"
                                   FJ-OM_ORBID
                         RETURNING ORB.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                         USING     CORBA-ORB-OBJECTID_NAMESERVICE
                         RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                         USING     OBJ
                         RETURNING NAMING-CONTEXT.
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE"
                         USING     "ODsample::structtest"
                         RETURNING NAME-ID.
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE"
                         USING     " "
                         RETURNING NAME-KIND.
```

```
*
     INVOKE COSNAMING-NAMECOMPONENT "NEW"
                         RETURNING NAME-COMPONENT.
*
     SET IDL-ID OF NAME-COMPONENT TO NAME-ID.
     SET KIND   OF NAME-COMPONENT TO NAME-KIND.
*
     INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH"
                         USING     1
                         RETURNING NAME.
*
     INVOKE NAME "SET-VALUE"
                         USING     1
                                   NAME-COMPONENT.
*
     MOVE "CosNaming::NamingContext::resolve" TO API-NAME.
     INVOKE NAMING-CONTEXT "RESOLVE"
                         USING     NAME
                         RETURNING OBJ.
*
     INVOKE ODSAMPLE-STRUCTTEST "NARROW"
                         USING     OBJ
                         RETURNING TARGET-OBJ.
*
*  ODsample::samplefix
*
*  SET IN PARAMETER
     MOVE 10 TO PARA1 OF FIX1.
     MOVE 11 TO PARA2 OF FIX1.
*
*  SET INOUT PARAMETER
     MOVE 20 TO PARA1 OF FIX3.
     MOVE 21 TO PARA1 OF FIX3.
*
     MOVE "ODdemo::structtest::op2" TO API-NAME.
     INVOKE TARGET-OBJ "OP2"
                         USING     FIX1
                                   FIX2
                                   FIX3
                         RETURNING FIX0.
*
*  ODsample::samplevar
*
*  SET IN PARAMETER
     INVOKE ODSAMPLE-SAMPLEVAR "NEW" RETURNING VAR1.
     MOVE 5 TO PARA1 OF VAR1.
     INVOKE CORBA-STRING "NEW" RETURNING STR-WORK.
     INVOKE STR-WORK "SET-VALUE" USING "test".
     SET PARA2 OF VAR1 TO STR-WORK.
*
*  SET INOUT PARAMETER
     INVOKE ODSAMPLE-SAMPLEVAR "NEW" RETURNING VAR3.
     MOVE 4 TO PARA1 OF VAR1.
     INVOKE CORBA-STRING "NEW" RETURNING STR-WORK.
     INVOKE STR-WORK "SET-VALUE" USING "pro".
     SET PARA2 OF VAR3 TO STR-WORK.
*
     MOVE "ODdemo::structtest::op1" TO API-NAME.
     INVOKE TARGET-OBJ "OP1"
                         USING     VAR1
                                   VAR2
                                   VAR3
                         RETURNING VAR0.
```

```
*
     STOP RUN.
*
 END-MAIN.
*
 END PROGRAM "CLIENT-MAIN".
```

(4)サーバアプリケーション例 Windows32 Solaris32 Linux32

サーバアプリケーション例を以下に示します。

```
 CLASS-ID.              ODSAMPLE-STRUCTTEST-IMPL
     INHERITS           ODSAMPLE-STRUCTTEST.
*
 ENVIRONMENT            DIVISION.
 CONFIGURATION          SECTION.
 REPOSITORY.
     COPY               CORBA--REP.
     COPY               SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY               CORBA--CONST.
     COPY               SIMPLE--CONST.
     .
*
 OBJECT.
  DATA                  DIVISION.
  WORKING-STORAGE       SECTION.
  COPY                  CORBA--COPY.
  COPY                  SIMPLE--COPY.
*
  PROCEDURE             DIVISION.
*
   METHOD-ID.           OP1 OVERRIDE.
   DATA                 DIVISION.
   WORKING-STORAGE      SECTION.
   01 STR-WORK          USAGE OBJECT REFERENCE CORBA-STRING.
   LINKAGE              SECTION.
   01 STR1              USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
   01 STR2              USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
   01 STR3              USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
   01 STR0              USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
*
   PROCEDURE            DIVISION
                        USING     STR1
                                  STR2
                                  STR3
                        RETURNING STR0
                        RAISING   CORBA-SYSTEMEXCEPTION.
*
*  SET OUT PARAMETER
     INVOKE ODSAMPLE-SAMPLEVAR "NEW" RETURNING STR2.
     MOVE 12 TO PARA1 OF STR2.
     INVOKE CORBA-STRING "NEW" RETURNING STR-WORK.
     INVOKE STR-WORK "SET-VALUE" USING "(*str2)->para2".
     SET PARA2 OF STR2 TO STR-WORK.
*
*  SET INOUT PARAMETER
     INVOKE ODSAMPLE-SAMPLEVAR "NEW" RETURNING STR3.
     MOVE 16 TO PARA1 OF STR3.
     INVOKE CORBA-STRING "NEW" RETURNING STR-WORK.
```

```
     INVOKE STR-WORK "SET-VALUE" USING "(*str3)->para2".
     SET PARA2 OF STR3 TO STR-WORK.
*
*  SET RESULT
     INVOKE ODSAMPLE-SAMPLEVAR "NEW" RETURNING STR0.
     MOVE 11 TO PARA1 OF STR0.
     INVOKE CORBA-STRING "NEW" RETURNING STR-WORK.
     INVOKE STR-WORK "SET-VALUE" USING "(*str0)->para2".
     SET PARA2 OF STR0 TO STR-WORK.
*
   END METHOD OP1.
*
*
   METHOD-ID.           OP2 OVERRIDE.
   DATA                 DIVISION.
   LINKAGE              SECTION.
   01 STR1             TYPE                    ODSAMPLE-SAMPLEFIX.
   01 STR2             TYPE                    ODSAMPLE-SAMPLEFIX.
   01 STR3             TYPE                    ODSAMPLE-SAMPLEFIX.
   01 STR0             TYPE                    ODSAMPLE-SAMPLEFIX.
*
   PROCEDURE            DIVISION
                        USING     STR1
                                  STR2
                                  STR3
                        RETURNING STR0
                        RAISING   CORBA-SYSTEMEXCEPTION.
*
*  SET OUT PARAMETER
*
     MOVE 0 TO PARA1 OF STR3.
     MOVE 1 TO PARA2 OF STR3.
*  SET INOUT PARAMETER
     MOVE 2 TO PARA1 OF STR2.
     MOVE 3 TO PARA2 OF STR2.
*
*  SET RESULT
     MOVE 4 TO PARA1 OF STR0.
     MOVE 5 TO PARA2 OF STR0.
*
   END METHOD OP2.
*
 END OBJECT.
*
 END CLASS ODSAMPLE-STRUCTTEST-IMPL.
```

G.3.8.5 共用体

(1) IDLマッピング

IDL言語で共用体unionを指定した場合、クラス(共用体クラス)でデータ宣言します。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

［ファイル名: simple.idl］

```
module ODsample{
    union samplefix switch(long){    // 共用体(固定長)
        case 1: long    para1;
```

```
        case 2: long    para2;
    };
    union samplevar switch(long){    // 共用体(可変長)
        case 1: long    para1;
        case 2: string  para2;
    };
    interface  uniontest{
        samplefix  op2(
            in samplefix uni1,
            out samplefix uni2,
            inout samplefix uni3
        );
        samplevar  op1(
            in samplevar uni1,
            out samplevar uni2,
            inout samplevar uni3
        );
    };
};
```

OOCOBOL

［ファイル名:ODSAMPLE-SAMPLEFIX.COB］

```
* 共用体クラス(共用体(固定長))
 CLASS-ID. ODSAMPLE-SAMPLEFIX AS "ODSAMPLE-SAMPLEFIX"
     INHERITS UNIONBASE.
*
 OBJECT.
  PROCEDURE DIVISION.
*
* DISCRIMINATOR PROPERTY : GET
   METHOD-ID. GET PROPERTY DISCRIMINATOR.
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--RET-DISCRIMINATOR TYPE CORBA-LONG.
   PROCEDURE DIVISION RETURNING O--RET-DISCRIMINATOR.
   END METHOD.
*
* MEMBER PROPERTY : GET
   METHOD-ID. GET PROPERTY PARA1.
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--VALUE TYPE CORBA-LONG.
   PROCEDURE DIVISION RETURNING O--VALUE
     RAISING CORBA-BAD_PARAM.
   END METHOD.
*
* MEMBER PROPERTY : SET
   METHOD-ID. SET PROPERTY PARA1.
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--VALUE TYPE CORBA-LONG.
   PROCEDURE DIVISION USING O--VALUE.
   END METHOD.
*
* MEMBER PROPERTY : GET
   METHOD-ID. GET PROPERTY PARA2.
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--VALUE TYPE CORBA-LONG.
   PROCEDURE DIVISION RETURNING O--VALUE
```

```
      RAISING CORBA-BAD_PARAM.
   END METHOD.
*
* MEMBER PROPERTY : SET
   METHOD-ID. SET PROPERTY PARA2.
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--VALUE TYPE CORBA-LONG.
   PROCEDURE DIVISION USING O--VALUE.
   END METHOD.
*
* CLONE METHOD
   METHOD-ID. CLONE AS "CLONE".
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--CLONE-OBJ OBJECT REFERENCE ODSAMPLE-SAMPLEFIX.
   PROCEDURE DIVISION RETURNING O--CLONE-OBJ.
   END METHOD CLONE.
*
 END OBJECT.
 END CLASS ODSAMPLE-SAMPLEFIX.
```

［ファイル名:ODSAMPLE-SAMPLEVAR.COB］

```
* 共用体クラス(共用体(可変長)
 CLASS-ID. ODSAMPLE-SAMPLEVAR AS "ODSAMPLE-SAMPLEVAR"
     INHERITS UNIONBASE.
*
 OBJECT.
  PROCEDURE DIVISION.
*
* DISCRIMINATOR PROPERTY : GET
   METHOD-ID. GET PROPERTY DISCRIMINATOR.
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--RET-DISCRIMINATOR TYPE CORBA-LONG.
   PROCEDURE DIVISION RETURNING O--RET-DISCRIMINATOR.
   END METHOD.
*
* MEMBER PROPERTY : GET
   METHOD-ID. GET PROPERTY PARA1.
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--VALUE TYPE CORBA-LONG.
   PROCEDURE DIVISION RETURNING O--VALUE
     RAISING CORBA-BAD_PARAM.
   END METHOD.
*
* MEMBER PROPERTY : SET
   METHOD-ID. SET PROPERTY PARA1.
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--VALUE TYPE CORBA-LONG.
   PROCEDURE DIVISION USING O--VALUE.
   END METHOD.
*
* MEMBER PROPERTY : GET
   METHOD-ID. GET PROPERTY PARA2.
   DATA DIVISION.
   LINKAGE SECTION.
   01  O--VALUE OBJECT REFERENCE CORBA-STRING.
   PROCEDURE DIVISION RETURNING O--VALUE
```

```
      RAISING CORBA-BAD_PARAM.
    END METHOD.
*
* MEMBER PROPERTY : SET
    METHOD-ID. SET PROPERTY PARA2.
    DATA DIVISION.
    LINKAGE SECTION.
    01  O--VALUE OBJECT REFERENCE CORBA-STRING.
    PROCEDURE DIVISION USING O--VALUE.
    END METHOD.
*
* CLONE METHOD
    METHOD-ID. CLONE AS "CLONE".
    DATA DIVISION.
    LINKAGE SECTION.
    01  O--CLONE-OBJ OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
    PROCEDURE DIVISION RETURNING O--CLONE-OBJ.
    END METHOD CLONE.
*
 END OBJECT.
 END CLASS ODSAMPLE-SAMPLEVAR.
```

(2)共用体クラスのメソッド／プロパティ

共用体クラスで定義されているメソッド／プロパティを以下に示します。

メソッド／プロパティ	機能
CLONEメソッド	(オブジェクトメソッド)共用体クラスを複製します。
弁別子プロパティ	弁別子情報を取得します。
メンバプロパティ	メンバの値の取得／設定を行います。

メソッド／プロパティの使用例を以下に示します。

```
* CLONEメソッド
*   共用体クラスをUNI-CLONとして複製
 01  UNI       USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01  UNI-CLON  USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 INVOKE UNI "CLONE"  RETURNING UNI-CLON.

* 弁別子プロパティ
*   弁別子(DISCRIMINATOR)の値をUNI-MEMTに取得
 01  UNI       USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01  UNI-DISC  TYPE  CORBA-LONG.
 MOVE DISCRIMINATOR OF UNI  TO UNI-DISC.

* メンバプロパティ(クラス以外のデータ型)
*   メンバPARA1の値をUNI-MEMTに取得
 01  UNI       USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01  UNI-MEMT  TYPE  CORBA-LONG.
 MOVE PARA1 OF UNI  TO UNI-MEMT.

*   メンバPARA1にUNI-MEMTを設定
 01  UNI       USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01  UNI-MEMT  TYPE  CORBA-LONG.
 MOVE UNI-MEMT  TO PARA1 OF UNI.

* メンバプロパティ(クラス)
*   メンバPARA2の値をUNI-MEMCに取得
 01  UNI       USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01  UNI-MEMC  USAGE OBJECT REFERENCE CORBA-STRING.
```

```
 SET UNI-MEMC  TO PARA2 OF UNI.

*   メンバPARA2にUNI-MEMCを設定
 01  UNI       USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01  UNI-MEMC  USAGE OBJECT REFERENCE CORBA-STRING.
 SET PARA2 OF UNI  TO UNI-MEMC.
```

(3)クライアントアプリケーション例

クライアントアプリケーション例を以下に示します。

```
 IDENTIFICATION          DIVISION.
 PROGRAM-ID.             "CLIENT-MAIN".
*
 ENVIRONMENT             DIVISION.
 CONFIGURATION           SECTION.
 REPOSITORY.
     COPY                CORBA--REP.
     COPY                COSNAMING--REP.
     COPY                SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY                CORBA--CONST.
     COPY                COSNAMING--CONST.
     COPY                SIMPLE--CONST.
     .
*
 DATA                    DIVISION.
 WORKING-STORAGE         SECTION.
 COPY                    CORBA--COPY.
 COPY                    COSNAMING--COPY.
 COPY                    SIMPLE--COPY.
*
 01 ORB                  USAGE OBJECT REFERENCE CORBA-ORB.
 01 OBJ                  USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 NAMING-CONTEXT       USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                 TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT       USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NAME-ID              USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND            USAGE OBJECT REFERENCE CORBA-STRING.
*
 01 EXCEPTION-ID         USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE   PIC   X(50).
 01 API-NAME             PIC   X(50).
*
 01 TARGET-OBJ           USAGE OBJECT REFERENCE ODSAMPLE-UNIONTEST.
 01 FIX0                 USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEFIX.
 01 FIX1                 USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEFIX.
 01 FIX2                 USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEFIX.
 01 FIX3                 USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEFIX.
 01 VAR0                 USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01 VAR1                 USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01 VAR2                 USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01 VAR3                 USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
 01 STR-WORK             USAGE OBJECT REFERENCE CORBA-STRING.
*
 PROCEDURE               DIVISION.
*
 DECLARATIVES.
*
 OTHER-ERROR SECTION.
```

```
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::Exception: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
 MAIN SECTION.
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
                         USING     "simple_c"
                                   FJ-OM_ORBID
                         RETURNING ORB.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                         USING     CORBA-ORB-OBJECTID_NAMESERVICE
                         RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                         USING     OBJ
                         RETURNING NAMING-CONTEXT.
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE"
                         USING     "ODsample::uniontest"
                         RETURNING NAME-ID.
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE"
                         USING     " "
                         RETURNING NAME-KIND.
*
     INVOKE COSNAMING-NAMECOMPONENT "NEW"
                         RETURNING NAME-COMPONENT.
*
     SET IDL-ID OF NAME-COMPONENT TO NAME-ID.
     SET KIND   OF NAME-COMPONENT TO NAME-KIND.
*
     INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH"
                         USING     1
                         RETURNING NAME.
*
     INVOKE NAME "SET-VALUE"
                         USING     1
                                   NAME-COMPONENT.
*
     MOVE "CosNaming::NamingContext::resolve" TO API-NAME.
     INVOKE NAMING-CONTEXT "RESOLVE"
                         USING     NAME
                         RETURNING OBJ.
*
     INVOKE ODSAMPLE-UNIONTEST "NARROW"
                         USING     OBJ
                         RETURNING TARGET-OBJ.
*
*  ODsample::samplefix
*
*  SET IN PARAMETER
     INVOKE ODSAMPLE-SAMPLEFIX "NEW" RETURNING FIX1.
     MOVE 10 TO PARA1 OF FIX1.
```

```
*
*  SET INOUT PARAMETER
     INVOKE ODSAMPLE-SAMPLEFIX "NEW" RETURNING FIX3.
     MOVE 21 TO PARA2 OF FIX3.
*
     MOVE "ODsample::uniontest::op2" TO API-NAME.
     INVOKE TARGET-OBJ "OP2"
                        USING     FIX1
                                  FIX2
                                  FIX3
                        RETURNING FIX0.
*
*  ODsample::samplevar
*
*  SET IN PARAMETER
     INVOKE ODSAMPLE-SAMPLEVAR "NEW" RETURNING VAR1.
     INVOKE CORBA-STRING "NEW" RETURNING STR-WORK.
     INVOKE STR-WORK "SET-VALUE" USING "Z".
     SET PARA2 OF VAR1 TO STR-WORK.
*
*  SET INOUT PARAMETER
     INVOKE ODSAMPLE-SAMPLEVAR "NEW" RETURNING VAR3.
     INVOKE CORBA-STRING "NEW" RETURNING STR-WORK.
     INVOKE STR-WORK "SET-VALUE" USING "y".
     SET PARA2 OF VAR3 TO STR-WORK.
*
     MOVE "ODsample::uniontest::op1" TO API-NAME.
     INVOKE TARGET-OBJ "OP1"
                        USING     VAR1
                                  VAR2
                                  VAR3
                        RETURNING VAR0.
*
     STOP RUN.
*
 END-MAIN.
*
 END PROGRAM "CLIENT-MAIN".
```

(4)サーバアプリケーション例 Windows32 Solaris32 Linux32

サーバアプリケーション例を以下に示します。

```
 CLASS-ID.              ODSAMPLE-UNIONTEST-IMPL
     INHERITS           ODSAMPLE-UNIONTEST.
*
 ENVIRONMENT            DIVISION.
 CONFIGURATION          SECTION.
 REPOSITORY.
     COPY               CORBA--REP.
     COPY               SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY               CORBA--CONST.
     COPY               SIMPLE--CONST.
     .
*
 OBJECT.
  DATA                  DIVISION.
  WORKING-STORAGE       SECTION.
  COPY                  CORBA--COPY.
```

```
  COPY                 SIMPLE--COPY.
*
  PROCEDURE            DIVISION.
*
   METHOD-ID.          OP1 OVERRIDE.
   DATA                DIVISION.
   WORKING-STORAGE     SECTION.
   01 STR-WORK         USAGE OBJECT REFERENCE CORBA-STRING.
   LINKAGE             SECTION.
   01 UNI1             USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
   01 UNI2             USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
   01 UNI3             USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
   01 UNI0             USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEVAR.
*
   PROCEDURE           DIVISION
                       USING     UNI1
                                 UNI2
                                 UNI3
                       RETURNING UNI0
                       RAISING   CORBA-SYSTEMEXCEPTION.
*
*  SET OUT PARAMETER
     INVOKE ODSAMPLE-SAMPLEVAR "NEW" RETURNING UNI2.
     INVOKE CORBA-STRING "NEW" RETURNING STR-WORK.
     INVOKE STR-WORK "SET-VALUE" USING "(*uni2)->para2".
     SET PARA2 OF UNI2 TO STR-WORK.
*
*  SET INOUT PARAMETER
     INVOKE ODSAMPLE-SAMPLEVAR "NEW" RETURNING UNI3.
     INVOKE CORBA-STRING "NEW" RETURNING STR-WORK.
     INVOKE STR-WORK "SET-VALUE" USING "(*uni3)->para2".
     SET PARA2 OF UNI3 TO STR-WORK.
*
*  SET RESULT
     INVOKE ODSAMPLE-SAMPLEVAR "NEW" RETURNING UNI0.
     INVOKE CORBA-STRING "NEW" RETURNING STR-WORK.
     INVOKE STR-WORK "SET-VALUE" USING "(*uni0)->para2".
     SET PARA2 OF UNI0 TO STR-WORK.
*
   END METHOD OP1.
*
*
   METHOD-ID.          OP2 OVERRIDE.
   DATA                DIVISION.
   LINKAGE             SECTION.
   01 UNI1             USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEFIX.
   01 UNI2             USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEFIX.
   01 UNI3             USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEFIX.
   01 UNI0             USAGE OBJECT REFERENCE ODSAMPLE-SAMPLEFIX.
*
   PROCEDURE           DIVISION
                       USING     UNI1
                                 UNI2
                                 UNI3
                       RETURNING UNI0
                       RAISING   CORBA-SYSTEMEXCEPTION.
*
*  SET OUT PARAMETER
*
     INVOKE ODSAMPLE-SAMPLEFIX "NEW" RETURNING UNI2.
     MOVE 0 TO PARA1 OF UNI2.
*
*  SET INOUT PARAMETER
```

```
     INVOKE ODSAMPLE-SAMPLEFIX "NEW" RETURNING UNI3.
     MOVE 3 TO PARA2 OF UNI3.
*
*  SET RESULT
     INVOKE ODSAMPLE-SAMPLEFIX "NEW" RETURNING UNIO.
     MOVE 4 TO PARA1 OF UNIO.
*
   END METHOD OP2.
*
 END OBJECT.
*
 END CLASS ODSAMPLE-UNIONTEST-IMPL.
```

G.3.8.6 固定小数点型

(1)IDLマッピング

IDL言語で固定小数点型fixedを指定した場合、数字項目(PICTURE句)でデータ宣言します。
有効桁数と小数点位置による数字項目の定義形式を以下に示します。

IDL言語

```
// fixed<digits,scale> とした場合 (digits=有効桁数,scale=小数点位置)
typedef fixed <10,2>  fixed1;    // digits > scale > 0 のとき
typedef fixed <2,10>  fixed2;    // 0 < digits < scale のとき
typedef fixed <10,10> fixed3;    // digits = scale のとき
typedef fixed <10,0>  fixed4;    // scale = 0 のとき
typedef fixed <10,-2> fixed5;    // scale < 0 のとき
```

OOCOBOL

```
01 FIXED1 PIC S9(8)V9(2)  PACKED-DECIMAL.
01 FIXED2 PIC SVP(8)9(2)  PACKED-DECIMAL.
01 FIXED3 PIC SV9(10)     PACKED-DECIMAL.
01 FIXED4 PIC S9(10)      PACKED-DECIMAL.
01 FIXED5 PIC S9(10)P(2)  PACKED-DECIMAL.
```

以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample {
    typedef fixed <10,3>  fixed0;
    interface fixedtest {
          fixed0  op1(
              in       fixed <5,10>    fixed1,
              out      fixed <10,-3>   fixed2,
              inout    fixed <10,10>   fixed3 );
    };
};
```

OOCOBOL

```
* データ型宣言(固定小数点)
   01 FIXED0  PIC  S9(7)V9(3)  PACKED-DECIMAL.
```

(2)クライアントアプリケーション例

クライアントアプリケーション例を以下に示します。

```
 IDENTIFICATION          DIVISION.
 PROGRAM-ID.             "CLIENT-MAIN".
*
 ENVIRONMENT             DIVISION.
 CONFIGURATION           SECTION.
 REPOSITORY.
     COPY                CORBA--REP.
     COPY                COSNAMING--REP.
     COPY                SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY                CORBA--CONST.
     COPY                COSNAMING--CONST.
     COPY                SIMPLE--CONST.
     .
*
 DATA                    DIVISION.
 WORKING-STORAGE         SECTION.
 COPY                    CORBA--COPY.
 COPY                    COSNAMING--COPY.
 COPY                    SIMPLE--COPY.
*
 01 ORB                  USAGE OBJECT REFERENCE CORBA-ORB.
 01 OBJ                  USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 NAMING-CONTEXT       USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                 TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT       USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NAME-ID              USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND            USAGE OBJECT REFERENCE CORBA-STRING.
*
 01 EXCEPTION-ID         USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE   PIC   X(50).
 01 API-NAME             PIC   X(50).
*
 01 TARGET-OBJ           USAGE OBJECT REFERENCE ODSAMPLE-FIXEDTEST.
 01 FIXED0               TYPE ODSAMPLE-FIXED0.
 01 FIXED1               PIC SVP(5)9(5) PACKED-DECIMAL.
 01 FIXED2               PIC S9(10)P(3) PACKED-DECIMAL.
 01 FIXED3               PIC SV9(10)    PACKED-DECIMAL.
*
 PROCEDURE               DIVISION.
*
 DECLARATIVES.
*
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::Exception: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
 MAIN SECTION.
*
```

```
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
                         USING     "simple_c"
                                   FJ-OM_ORBID
                         RETURNING ORB.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                         USING     CORBA-ORB-OBJECTID_NAMESERVICE
                         RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                         USING     OBJ
                         RETURNING NAMING-CONTEXT.
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE"
                         USING     "ODsample::fixedtest"
                         RETURNING NAME-ID.
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE"
                         USING     " "
                         RETURNING NAME-KIND.
*
     INVOKE COSNAMING-NAMECOMPONENT "NEW"
                         RETURNING NAME-COMPONENT.
*
     SET IDL-ID OF NAME-COMPONENT TO NAME-ID.
     SET KIND   OF NAME-COMPONENT TO NAME-KIND.
*
     INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH"
                         USING     1
                         RETURNING NAME.
*
     INVOKE NAME "SET-VALUE"
                         USING     1
                                   NAME-COMPONENT.
*
     MOVE "CosNaming::NamingContext::resolve" TO API-NAME.
     INVOKE NAMING-CONTEXT "RESOLVE"
                         USING     NAME
                         RETURNING OBJ.
*
     INVOKE ODSAMPLE-FIXEDTEST "NARROW"
                         USING     OBJ
                         RETURNING TARGET-OBJ.
*
*  ODsample::samplefix
*
*  SET IN PARAMETER
     MOVE 0.  23 TO FIXED1.
*
*  SET INOUT PARAMETER
     MOVE 0.  33 TO FIXED3.
*
     MOVE "ODdemo::fixedtest::op1" TO API-NAME.
     INVOKE TARGET-OBJ "OP1"
                         USING     FIXED1
                                   FIXED2
                                   FIXED3
                         RETURNING FIXED0.
*
     STOP RUN.
*
```

```
 END-MAIN.
*
 END PROGRAM "CLIENT-MAIN".
```

(3)サーバアプリケーション例 Windows32 Solaris32 Linux32

サーバアプリケーション例を以下に示します。

```
 CLASS-ID.              ODSAMPLE-FIXEDTEST-IMPL
     INHERITS           ODSAMPLE-FIXEDTEST.
*
 ENVIRONMENT            DIVISION.
 CONFIGURATION          SECTION.
 REPOSITORY.
     COPY               CORBA--REP.
     COPY               SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY               CORBA--CONST.
     COPY               SIMPLE--CONST.
     .
*
 OBJECT.
  DATA                  DIVISION.
  WORKING-STORAGE       SECTION.
  COPY                  CORBA--COPY.
  COPY                  SIMPLE--COPY.
*
  PROCEDURE             DIVISION.
*
   METHOD-ID.           OP1 OVERRIDE.
   DATA                 DIVISION.
   WORKING-STORAGE      SECTION.
   01 STR-WORK         USAGE OBJECT REFERENCE CORBA-STRING.
   LINKAGE              SECTION.
 01 FIXED0             TYPE ODSAMPLE-FIXED0.
   01 FIXED1           PIC SVP(5)9(5) PACKED-DECIMAL.
   01 FIXED2           PIC S9(10)P(3) PACKED-DECIMAL.
   01 FIXED3           PIC SV9(10)    PACKED-DECIMAL.
*
   PROCEDURE            DIVISION
                        USING     FIXED1
                                  FIXED2
                                  FIXED3
                        RETURNING FIXED0
                        RAISING   CORBA-SYSTEMEXCEPTION.
*
*  SET OUT PARAMETER
     MOVE   0 TO FIXED2.
*
*  SET INOUT PARAMETER
     MOVE 0.    TO FIXED3.
*
*  SET RESULT
     MOVE 200.33 TO FIXED0.
*
   END METHOD OP1.
*
 END OBJECT.
*
 END CLASS ODSAMPLE-FIXEDTEST-IMPL.
```

G.3.8.7 属性宣言(attribute)のマッピング

(1) IDLマッピング

IDL言語で属性宣言(attribute)を指定した場合、オブジェクトデータの取得／設定を行うプロパティ(プロパティ名はIDLで定義された変数名)が定義されます。
以降では、以下のIDL定義例をもとに説明します。

IDL言語

```
module ODsample{
       interface       attrtest{
               attribute long  para1;
               attribute string para2;
               readonly attribute long para3;
       };
};
```

OOCOBOL

```
   CLASS-ID. ODSAMPLE-ATTRTEST AS "ODSAMPLE-ATTRTEST"
       INHERITS CORBA-OBJECT
       .
 *
  FACTORY.
   PROCEDURE DIVISION.
    METHOD-ID. NARROW AS "NARROW" PROTOTYPE.
    DATA DIVISION.
    LINKAGE SECTION.
    01 O--PARAM OBJECT REFERENCE CORBA-OBJECT.
    01 O--RESULT OBJECT REFERENCE SELF.
    PROCEDURE DIVISION
      USING O--PARAM
      RETURNING O--RESULT
      RAISING CORBA-SYSTEMEXCEPTION.
    END METHOD NARROW.
  END FACTORY.
 *
  OBJECT.
   PROCEDURE DIVISION.
 *
    METHOD-ID. GET PROPERTY PARA1.
    DATA DIVISION.
    LINKAGE SECTION.
    01  O--RESULT TYPE CORBA-LONG.
    PROCEDURE DIVISION
      RETURNING O--RESULT RAISING CORBA-SYSTEMEXCEPTION.
    END METHOD.
 *
    METHOD-ID. SET PROPERTY PARA1.
    DATA DIVISION.
    LINKAGE SECTION.
    01  O--PARAM TYPE CORBA-LONG.
    PROCEDURE DIVISION
      USING O--PARAM RAISING CORBA-SYSTEMEXCEPTION.
    END METHOD.
 *
    METHOD-ID. GET PROPERTY PARA2.
```

```
    DATA DIVISION.
    LINKAGE SECTION.
    01  O--RESULT OBJECT REFERENCE CORBA-STRING.
    PROCEDURE DIVISION
      RETURNING O--RESULT RAISING CORBA-SYSTEMEXCEPTION.
    END METHOD.
 *
    METHOD-ID. SET PROPERTY PARA2.
    DATA DIVISION.
    LINKAGE SECTION.
    01  O--PARAM OBJECT REFERENCE CORBA-STRING.
    PROCEDURE DIVISION
      USING O--PARAM RAISING CORBA-SYSTEMEXCEPTION.
    END METHOD.
 *
    METHOD-ID. GET PROPERTY PARA3.
    DATA DIVISION.
    LINKAGE SECTION.
    01  O--RESULT TYPE CORBA-LONG.
    PROCEDURE DIVISION
      RETURNING O--RESULT RAISING CORBA-SYSTEMEXCEPTION.
    END METHOD.
 *
  END OBJECT.
  END CLASS ODSAMPLE-ATTRTEST.
```

(2)オブジェクトデータのプロパティ

属性宣言により定義されるオブジェクトデータのプロパティを以下に示します。

プロパティ	機能
データプロパティ	オブジェクトデータの値の取得／設定を行います。

プロパティの使用例を以下に示します。

```
* データプロパティ(クラス以外のデータ型)
*   PARA1の値をATTR-LONGへ取得
 01  ATTR      USAGE OBJECT REFERENCE ODSAMPLE-ATTRTEST.
 01  ATTR-LONG TYPE  CORBA-LONG.
 MOVE PARA1 OF ATTR  TO ATTR-LONG.

*   PARA1にATTR-LONGを設定
 01  ATTR      USAGE OBJECT REFERENCE ODSAMPLE-ATTRTEST.
 01  ATTR-LONG TYPE  CORBA-LONG.
 MOVE ATTR-LONG  TO PARA1 OF ATTR.

* データプロパティ(クラス)
*   PARA2の値をATTR-STRへ取得
 01  ATTR      USAGE OBJECT REFERENCE ODSAMPLE-ATTRTEST.
 01  ATTR-STR  USAGE OBJECT REFERENCE CORBA-STRING.
 SET ATTR-STR  TO PARA2 OF ATTR.

*   PARA2にATTR-STRを設定
 01  ATTR      USAGE OBJECT REFERENCE ODSAMPLE-ATTRTEST.
 01  ATTR-STR  USAGE OBJECT REFERENCE CORBA-STRING.
 SET PARA2 OF ATTR  TO ATTR-STR.

* データプロパティ(readonly)
```

```
*   PARA3の値をATTR-LONGへ取得
 01  ATTR      USAGE OBJECT REFERENCE ODSAMPLE-ATTRTEST.
 01  ATTR-LONG TYPE  CORBA-LONG.
 MOVE PARA3 OF ATTR  TO ATTR-LONG.
```

(3)クライアントアプリケーション例

クライアントアプリケーション例を以下に示します。

```
 IDENTIFICATION          DIVISION.
 PROGRAM-ID.             "CLIENT-MAIN".
*
 ENVIRONMENT             DIVISION.
 CONFIGURATION           SECTION.
 REPOSITORY.
     COPY                CORBA--REP.
     COPY                COSNAMING--REP.
     COPY                SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY                CORBA--CONST.
     COPY                COSNAMING--CONST.
     COPY                SIMPLE--CONST.
     .
*
 DATA                    DIVISION.
 WORKING-STORAGE         SECTION.
 COPY                    CORBA--COPY.
 COPY                    COSNAMING--COPY.
 COPY                    SIMPLE--COPY.
*
 01 ORB                  USAGE OBJECT REFERENCE CORBA-ORB.
 01 OBJ                  USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 NAMING-CONTEXT       USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                 TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT       USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NAME-ID              USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND            USAGE OBJECT REFERENCE CORBA-STRING.
*
 01 EXCEPTION-ID         USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE   PIC   X(50).
 01 API-NAME             PIC   X(50).
*
 01 TARGET-OBJ           USAGE OBJECT REFERENCE ODSAMPLE-ATTRTEST.
 01 NUM                  TYPE                   CORBA-LONG.
 01 RET-NUM              TYPE                   CORBA-LONG.
 01 STR1                 USAGE OBJECT REFERENCE CORBA-STRING.
 01 RET-STR1             USAGE OBJECT REFERENCE CORBA-STRING.
 01 RET-STR1-VALUE       PIC X(30).
*
 PROCEDURE               DIVISION.
*
 DECLARATIVES.
*
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::Exception: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
```

```
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
 MAIN SECTION.
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
                         USING     "simple_c"
                                   FJ-OM_ORBID
                         RETURNING ORB.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                         USING     CORBA-ORB-OBJECTID_NAMESERVICE
                         RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                         USING     OBJ
                         RETURNING NAMING-CONTEXT.
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE"
                         USING     "ODsample::attrtest"
                         RETURNING NAME-ID.
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE"
                         USING     " "
                         RETURNING NAME-KIND.
*
     INVOKE COSNAMING-NAMECOMPONENT "NEW"
                         RETURNING NAME-COMPONENT.
*
     SET IDL-ID OF NAME-COMPONENT TO NAME-ID.
     SET KIND   OF NAME-COMPONENT TO NAME-KIND.
*
     INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH"
                         USING     1
                         RETURNING NAME.
*
     INVOKE NAME "SET-VALUE"
                         USING     1
                                   NAME-COMPONENT.
*
     MOVE "CosNaming::NamingContext::resolve" TO API-NAME.
     INVOKE NAMING-CONTEXT "RESOLVE"
                         USING     NAME
                         RETURNING OBJ.
*
     INVOKE ODSAMPLE-ATTRTEST "NARROW"
                         USING     OBJ
                         RETURNING TARGET-OBJ.
*
*  para1(SET)
     MOVE 2 TO NUM.
     MOVE "ODsample::attrtest::para1(set)" TO API-NAME.
     MOVE NUM TO PARA1 OF TARGET-OBJ.
*
*  para1(GET)
     MOVE "ODsample::attrtest::para1(get)" TO API-NAME.
     MOVE PARA1 OF TARGET-OBJ TO RET-NUM.
     DISPLAY "ODsample::attrtest::para1 returns " RET-NUM.
*
*  para2(SET)
```

```
      INVOKE CORBA-STRING "NEW-WITH-VALUE" USING "test" RETURNING STR1.
      MOVE "ODsample::attrtest::para2(set)" TO API-NAME.
      SET PARA2 OF TARGET-OBJ TO STR1.
*
*  para2(GET)
      MOVE "ODsample::attrtest::para2(get)" TO API-NAME.
      SET RET-STR1 TO PARA2 OF TARGET-OBJ.
      INVOKE RET-STR1 "GET-VALUE" RETURNING RET-STR1-VALUE.
      DISPLAY "ODsample::attrtest::para2 returns " RET-STR1-VALUE.
*
*  para3(GET)
      MOVE "ODsample::attrtest::para3(get)" TO API-NAME.
      MOVE PARA3 OF TARGET-OBJ TO RET-NUM.
      DISPLAY "ODsample::attrtest::para3 returns " RET-NUM.
*
      STOP RUN.
*
 END-MAIN.
*
 END PROGRAM "CLIENT-MAIN".
```

(4)サーバアプリケーション例 Windows32 Solaris32 Linux32

サーバアプリケーション例を以下に示します。

```
  CLASS-ID.              ODSAMPLE-ATTRTEST-IMPL
      INHERITS           ODSAMPLE-ATTRTEST.
*
 ENVIRONMENT             DIVISION.
 CONFIGURATION           SECTION.
 REPOSITORY.
      COPY               CORBA--REP.
      COPY               SIMPLE--REP.
      .
 SPECIAL-NAMES.
      SYMBOLIC CONSTANT
      COPY               CORBA--CONST.
      COPY               SIMPLE--CONST.
      .
*
 OBJECT.
  DATA                   DIVISION.
  WORKING-STORAGE        SECTION.
  COPY                   CORBA--COPY.
  COPY                   SIMPLE--COPY.
  01 PARA1-VALUE         TYPE                    CORBA-LONG VALUE ZERO.
  01 PARA2-VALUE         USAGE OBJECT REFERENCE CORBA-STRING.
  01 PARA3-VALUE         TYPE                    CORBA-LONG VALUE 10000.
*
  PROCEDURE              DIVISION.
*
   METHOD-ID.            GET PROPERTY PARA1 OVERRIDE.
   DATA                  DIVISION.
   WORKING-STORAGE       SECTION.
   LINKAGE               SECTION.
   01 PARAM              TYPE                    CORBA-LONG.
*
   PROCEDURE             DIVISION
                         RETURNING PARAM
                         RAISING   CORBA-SYSTEMEXCEPTION.
*
      MOVE PARA1-VALUE TO PARAM.
```

```
*
   END METHOD.
*
   METHOD-ID.           SET PROPERTY PARA1 OVERRIDE.
   DATA                 DIVISION.
   WORKING-STORAGE      SECTION.
   LINKAGE              SECTION.
   01 PARAM             TYPE                   CORBA-LONG.
*
   PROCEDURE            DIVISION
                        USING     PARAM
                        RAISING   CORBA-SYSTEMEXCEPTION.
*
     MOVE PARAM TO PARA1-VALUE.
*
   END METHOD.
*
   METHOD-ID.           GET PROPERTY PARA2 OVERRIDE.
   DATA                 DIVISION.
   WORKING-STORAGE      SECTION.
   LINKAGE              SECTION.
   01 PARAM             USAGE OBJECT REFERENCE CORBA-STRING.
*
   PROCEDURE            DIVISION
                        RETURNING PARAM
                        RAISING   CORBA-SYSTEMEXCEPTION.
*
     SET PARAM TO PARA2-VALUE.
*
   END METHOD.
*
   METHOD-ID.           SET PROPERTY PARA2 OVERRIDE.
   DATA                 DIVISION.
   WORKING-STORAGE      SECTION.
   LINKAGE              SECTION.
   01 PARAM             USAGE OBJECT REFERENCE CORBA-STRING.
*
   PROCEDURE            DIVISION
                        USING     PARAM
                        RAISING   CORBA-SYSTEMEXCEPTION.
*
     SET PARA2-VALUE TO PARAM.
*
   END METHOD.
*
   METHOD-ID.           GET PROPERTY PARA3 OVERRIDE.
   DATA                 DIVISION.
   WORKING-STORAGE      SECTION.
   LINKAGE              SECTION.
   01 PARAM             TYPE                   CORBA-LONG.
*
   PROCEDURE            DIVISION
                        RETURNING PARAM
                        RAISING   CORBA-SYSTEMEXCEPTION.
*
     MOVE PARA3-VALUE TO PARAM.
*
   END METHOD.
*
 END OBJECT.
*
 END CLASS ODSAMPLE-ATTRTEST-IMPL.
```

G.3.8.8 パラメタ受渡しで使用するデータ型

クライアント／サーバアプリケーションでパラメタ受渡しに使用するデータ型(オペレーション宣言)を以下に示します。

CORBAデータ型		in	out	inout	復帰値
long		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
short		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
unsigned long		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
unsigned short		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
long long		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
float		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
double		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
char		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
wchar		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
octet		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
boolean		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
enum		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
string	サイズ指定あり	TEXT	TEXT	TEXT	TEXT
	サイズ指定なし	TEXT	BY REFERENCE	BY REFERENCE	BY REFERENCE
wstring	サイズ指定あり	TEXT	TEXT	TEXT	TEXT
	サイズ指定なし	TEXT	BY REFERENCE	BY REFERENCE	BY REFERENCE
sequence		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
struct		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
union		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
fixed		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
Object		BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE

CORBAデータ型	in	out	inout	復帰値
TypeCode	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE

注) TEXTはOOCOBOLの固定長テキストでNULL文字を含みません。

クライアント／サーバアプリケーションでパラメタ受渡しに使用するデータ型(属性宣言)を以下に示します。

CORBAデータ型		set	get
long		BY REFERENCE	BY REFERENCE
short		BY REFERENCE	BY REFERENCE
unsigned long		BY REFERENCE	BY REFERENCE
unsigned short		BY REFERENCE	BY REFERENCE
long long		BY REFERENCE	BY REFERENCE
float		BY REFERENCE	BY REFERENCE
double		BY REFERENCE	BY REFERENCE
char		BY REFERENCE	BY REFERENCE
wchar		BY REFERENCE	BY REFERENCE
octet		BY REFERENCE	BY REFERENCE
Boolean		BY REFERENCE	BY REFERENCE
enum		BY REFERENCE	BY REFERENCE
string	サイズ指定あり	TEXT	TEXT
	サイズ指定なし	BY REFERENCE	BY REFERENCE
wstring	サイズ指定あり	TEXT	TEXT
	サイズ指定なし	BY REFERENCE	BY REFERENCE
sequence		BY REFERENCE	BY REFERENCE
struct		BY REFERENCE	BY REFERENCE
union		BY REFERENCE	BY REFERENCE
fixed		BY REFERENCE	BY REFERENCE
Object		BY REFERENCE	BY REFERENCE
TypeCode		BY REFERENCE	BY REFERENCE

注) TEXTはOOCOBOLの固定長テキストでNULL文字を含みません。

注意

(1)NULLポインタの設定について

サーバアプリケーションでのout, inoutパラメタ、復帰値、およびクライアントアプリケーションでのin,inoutパラメタでは、文字列型、シーケンス型、構造体、共用体でNULLオブジェクトを設定できません。

(2)データ型の扱いについて

固定長sequenceの場合、lengthにmaximumを超える値を設定した場合、maximumがlengthに設定されます。指定したlengthは、無効となります。

G.3.9 OOCOBOLアプリケーション使用時の注意事項

“第2章 CORBAアプリケーション開発時の留意事項”についても参照してください。

関数名／変数名の扱い

- 関数名/変数名の長さは30文字までです。30文字を超える場合、マッピング情報ファイル（IDL.inf）を参照してください。
- 関数名/変数名の文字長が30文字を超える場合で、同一名になる場合、27文字目以降を数値に置き換えます。

予約語の扱い

- OOCOBOLの予約語の文字列がIDLファイルに記述されている場合、文字列の前に“IDL-”を付加します。予約語については、OOCOBOLのマニュアルを参照してください。

アプリケーションのビルド時の注意事項

- 原則として、IDLの順番にコンパイルします。以下のIDLの場合、Sのコンパイル後にAをコンパイルします。

```
//idl
struct S { ... };
interface A { S op(); };
```

interfaceのforward_dclは、上記規則に含みません。

```
forward_dcl
interface A;
interface A{}; // 中身が空
```

- 要素内に定義されているものは、先にコンパイルします。以下のIDLの場合、S1のコンパイル後にSをコンパイルします。

```
//idl
struct S {
    struct S1 { ... } mem1;
};
```

- 1つの要素に複数のファイルが出力される場合は、以下の順番でコンパイルします。

  ー インタフェース以外の要素（登録集に定義されるものは除く）
  要素本体のクラスファイル→Helperクラスファイル

  ー インタフェース（スタブ側）
  インタフェースファイル→Helperクラスファイル→スタブクラスファイル→Narrowスタブファイル

  ー インタフェース（スケルトン側）
  インタフェースファイル→Helperクラスファイル→tieクラスファイル/ユーザ定義クラス→インプリメンテーション登録ファイル

- 相互参照がある場合は、下の要素の本体クラスから“CREATE(REP)”オプションで翻訳します。以下のIDLのコンパイル順番は、A.cob(“CREATE(REP)”オプション付き)→S.cob→S--HELPER.cob→A.cob....(“CREATE(REP)”オプションなし）となります。

```
interface A;
struct S { A mem1; };
interface A { S op(); };
```

その他、IDLコンパイラが出力したクラスをリポジトリ宣言で使用しているファイルは、上記の翻訳が終わった後に行います。

Windows(R)アプリケーションの作成 Windows32/64

Windows(R)でOOCOBOLアプリケーションを作成する場合の注意点を以下に示します。

ポイント

CORBAサービスのインストールフォルダを“C:¥Interstage¥ODWIN”として説明します。

クライアントアプリケーションの作成

クライアントアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。OOCOBOLアプリケーションのビルドには“COBOLプロジェクトマネージャー”を使用します。

翻訳・リンク手順

1. IDLコンパイルを実行します。
   IDLc -oocob IDLファイル名
2. 1.で生成されたファイルをCOBOLソースファイルにデータ型、データ型ヘルパー、インタフェースクラス、インタフェースクラスヘルパー、スタブクラス、Narrowスタブ、クライアントmainの順に追加します。
3. 「翻訳オプション」には、以下を設定します。
   LIB、REPIN(“翻訳時のオプションの設定”参照)
4. ライブラリファイルを設定します。
   [UNICODE／UTF8以外の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODOOCOB.LIB
   C:¥Interstage¥ODWIN¥LIB¥ODCNOOCOB.LIB
   [UNICODE／UTF8の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODOOCOBUC.LIB
   C:¥Interstage¥ODWIN¥LIB¥ODCNOOCOBUC.LIB
5. 相互参照がある場合は、リポジトリファイルを作成します。「翻訳オプション」「その他の翻訳オプション」に“CREATE(REP)”として翻訳します。相互参照がない場合は、必要ありません。

リンク時に指定するライブラリ

リンク時に必ず指定するライブラリは、以下のとおりです。

[UNICODE／UTF8以外の場合]
C:¥Interstage¥ODWIN¥LIB¥ODOOCOB.LIB
[UNICODE／UTF8の場合]
C:¥Interstage¥ODWIN¥LIB¥ODOOCOBUC.LIB

ネーミングサービスを利用するときは、以下のライブラリを追加します。

[UNICODE／UTF8以外の場合]
C:¥Interstage¥ODWIN¥LIB¥ODCNOOCOB.LIB
[UNICODE／UTF8の場合]
C:¥Interstage¥ODWIN¥LIB¥ODCNOOCOBUC.LIB

サーバアプリケーションの作成

サーバアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。OOCOBOLアプリケーションのビルドには、“COBOLプロジェクトマネージャー”を使用します。

翻訳・リンク手順

1. IDLコンパイルを実行します。
   IDLc -oocob IDLファイル名
2. 1.で生成されたファイルを“インタフェース名.dll”のCOBOLソースファイルにデータ型、データ型ヘルパー、インタフェースクラス、インタフェースクラスヘルパー、ユーザ定義クラス、Tieクラス、Initプログラム、Newプログラム、Narrowスケルトンの順に追加します。
3. “インタフェース名.dll”に、以下のライブラリを設定します。
   [UNICODE／UTF8以外の場合]
   C:¥Interstage¥ODWIN¥LIB¥ODOOCOBSV.LIB

［**UNICODE／UTF8の場合**］
C:¥Interstage¥ODWIN¥LIB¥ODOOCOBSVUC.LIB

4. 実行ファイルのCOBOLソースファイルにサーバmainを追加し、ライブラリファイルに以下のものを追加します。
［**UNICODE／UTF8以外の場合**］
C:¥Interstage¥ODWIN¥LIB¥ODOOCOBSV.LIB
［**UNICODE／UTF8の場合**］
C:¥Interstage¥ODWIN¥LIB¥ODOOCOBSVUC.LIB

5. 相互参照がある場合は、リポジトリファイルを作成します。「翻訳オプション」「その他の翻訳オプション」に“CREATE(REP)”として翻訳します。相互参照がない場合は、必要ありません。

リンク時に指定するライブラリ

リンク時に必ず指定するライブラリは、以下のとおりです。

［**UNICODE／UTF8以外の場合**］
C:¥Interstage¥ODWIN¥LIB¥ODOOCOBSV.LIB
［**UNICODE／UTF8の場合**］
C:¥Interstage¥ODWIN¥LIB¥ODOOCOBSVUC.LIB

ネーミングサービスを利用するときは、以下のライブラリを追加します。

［**UNICODE／UTF8以外の場合**］
C:¥Interstage¥ODWIN¥LIB¥ODCNSOOCOB.LIB
［**UNICODE／UTF8の場合**］
C:¥Interstage¥ODWIN¥LIB¥ODCNSOOCOBUC.LIB

クライアント用とサーバ用のライブラリを同時に指定できません。サーバアプリケーションが別のサーバと通信する場合は、サーバ用のライブラリをリンクします。

翻訳時のオプションの設定

翻訳時のオプションを設定します（“プロジェクト”→“オプション”→“翻訳オプション”）。

LIB

登録集が格納されているフォルダを指定します。翻訳オプションの追加で“LIB”を選択し、登録集ファイルの入力先に“C:¥Interstage¥ODWIN¥INCLUDE¥OOCOB”を指定します。

REPIN

リポジトリファイルが格納されているフォルダを指定します。翻訳オプションの追加で“REPIN”を選択し、リポジトリファイルの入力先に“C:¥Interstage¥ODWIN¥REP”を指定します。UNICODEを使用する場合は、“C:¥Interstage¥ODWIN¥REP¥UNICODE”を指定します。

THREAD

アプリケーションの動作モードを指定します。スレッドモードのアプリケーション作成時は、“マルチスレッドとする”を選択し、プロセスモードのアプリケーション作成時は、“シングルスレッドとする”を選択します。

RCS

実行時のコード系を設定します。翻訳オプションの追加で、UNICODEアプリケーション、またはUTF8アプリケーションを作成する場合は、“RCS”を選択し、“UCS2-UNICODE(UCS2)を使用する”を選択します。

ALPHAL

英小文字を英大文字と等価に扱う(ALPHAL)か、扱わない(NOALPHAL)かを指定します。本オプションを指定する場合は、本製品が提供するOOCOBOLのAPI、ネーミングサービス、およびIDLコンパイラが出力したクラス、メソッドはすべて大文字で記述してください（IDLコンパイラが出力するOOCOBOLのファイルでは、コメント行以外はすべて大文字で記述されます）。

モジュールの単位

サーバアプリケーションは初期化する実行モジュール（EXE）と、オペレーションの実装部であるライブラリ（DLL）の2つを作成します。実行モジュール側では初期化（CORBA-ORB_INIT）を行い、サーバを活性化させる（CORBA-BOA-IMPL_IS_READYまたはCORBA-BOA-OBJ_IS_READY）ところまでの処理を行います。
ライブラリ側は、1つのライブラリに複数のインタフェースを実装することができます。インタフェースごとにライブラリを分ける場合は、

データ型クラスファイルはインタフェースのライブラリとは別のライブラリ(DLL)にし、インタフェースクラスのライブラリからデータ型クラスのライブラリをリンクするようにします。

```
// IDL
module M {
    struct S { string s; };
    typedef long L;
    interface I1 {
        ....
    };
    interface I2 {
        ...
    };
};
```

上記IDLのサーバアプリケーションのライブラリをインタフェースごとに分けるときは、以下の構成にします。

```
M-I1のライブラリ→
              (リンク) データ型のライブラリ
M-I2のライブラリ→
```

[M-I1のライブラリに含まれるファイル]

```
M-I1.COB
M-I1--HELPER.COB
M-I1--TIE.COB
M-I1--NARROWSKEL.COB
M-I1--NEW.COB
M-I1の実装クラスファイル
M-I1-INITの定義ファイル
```

[M-I2のライブラリに含まれるファイル]

```
M-I2.COB
M-I2--HELPER.COB
M-I2--TIE.COB
M-I2--NARROWSKEL.COB
M-I2--NEW.COB
M-I2の実装クラスファイル
M-I2-INITの定義ファイル
```

[データ型のライブラリに含まれるファイル]

```
M-S.COB
M-S--HELPER.COB
L--HELPER.COB
```

Solarisアプリケーションの作成 Solaris32

SolarisでOOCOBOLアプリケーションを作成する場合の注意点を以下に示します。

注意

- 開発環境として“NetCOBOL”を使用してUNICODEのアプリケーションを開発する場合、ソースファイルおよび登録集は、UNICODE(UTF-8)で作成してください。
- 翻訳／リンク／実行を行う場合、コート系には“UNICODE”を設定してください。

ポイント

CORBAサービスのインストールディレクトリを“/opt/FSUNod”として説明します。

クライアントアプリケーションの作成

クライアントアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。

翻訳・リンク手順

1. CORBA-OOCOBOLマッピングを出力します。
   ［UNICODEモジュール以外の場合］
   IDLc -oocob IDLファイル名
   ［UNICODEモジュール(NetCOBOL)の場合］
   IDLc -oocob -unic IDLファイル名
2. 相互参照がある場合は、リポジトリファイルを作成します（XXX.cob: 主プログラム名）。相互参照がない場合は、必要ありません。
   cobol -WC,"CREATE(REP)" XXX.cob
3. 主プログラム以外を翻訳し、オブジェクトを作成します（翻訳の順番は、“アプリケーションのビルド時の注意事項”参照）。
   cobol -c -R/opt/FSUNod/rep -I/opt/FSUNod/include/oocob -WC,"THREAD(MULTI)" XXX.cob
4. 主プログラムを作成します。-Mオプションを指定してオブジェクトを作成します。
   cobol -c -M -R/opt/FSUNod/rep -I/opt/FSUNod/include/oocob -WC,"THREAD(MULTI)" XXX.cob
5. ファイルをリンクし、実行ファイルを作成します。-Tmオプションを指定すると、マルチスレッドで結合します。
   ［UNICODEモジュール以外の場合］
   cobol -Tm -lnsl -lsocket -L/opt/FSUNod/lib -lOMoocob -lOMcnoocob -lrcobol -o 実行ファイル名
   ［UNICODEモジュール(NetCOBOL)の場合］
   cobol -Tm -lnsl -lsocket -L/opt/FSUNod/lib -lOMoocobUC -lOMcnoocobUC -lrcobol -o 実行ファイル名

サーバアプリケーションの作成

サーバアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。

翻訳・リンク手順

1. CORBA-OOCOBOLマッピングを出力します。
   ［UNICODEモジュール以外の場合］
   IDLc -oocob IDLファイル名
   ［UNICODEモジュール(NetCOBOL)の場合］
   IDLc -oocob -unic IDLファイル名
2. 相互参照がある場合は、リポジトリファイルを作成します（XXX.cob: 主プログラム名）。相互参照がない場合は、必要ありません。
   cobol -WC,"CREATE(REP)" XXX.cob
3. 主プログラム以外を翻訳し、オブジェクトを作成します（翻訳の順番は、“アプリケーションのビルド時の注意事項”参照）。
   cobol -c -R/opt/FSUNod/rep -I/opt/FSUNod/include/oocob -WC,"THREAD(MULTI)" XXX.cob
4. 主プログラムを作成します。-Mオプションを指定してオブジェクトを作成します。
   cobol -c -M -R/opt/FSUNod/rep -I/opt/FSUNod/include/oocob -WC,"THREAD(MULTI)" XXX.cob
5. ファイルをリンクし、実行ファイルを作成します。-Tmオプションを指定すると、マルチスレッドで結合します。
   ［UNICODEモジュール以外の場合］
   cobol -Tm -lnsl -lsocket -L/opt/FSUNod/lib -lOMoocob -lrcobol -o 実行ファイル名
   ［UNICODEモジュール(NetCOBOL)の場合］
   cobol -Tm -lnsl -lsocket -L/opt/FSUNod/lib -lOMoocobUC -lrcobol -o 実行ファイル名
6. 共用オブジェクトを作成します。
   ［UNICODEモジュール以外の場合］
   cobol -G -Tm -lnsl -lsocket -L/opt/FSUNod/lib -lOMoocob -lrcobol -o 共用オブジェクト名

［UNICODEモジュール(NetCOBOL)の場合］
cobol -G -Tm -lnsl -lsocket -L/opt/FSUNod/lib -lOMoocobUC -lrcobol -o 共用オブジェクト名

リンク時に指定する共用オブジェクト（クライアントアプリケーション／サーバアプリケーション共通）

リンク時に必ず指定するライブラリは、以下のとおりです。
［UNICODEモジュール以外の場合］
/opt/FSUNod/lib/libOMoocob.so
［UNICODEモジュール(NetCOBOL)の場合］
/opt/FSUNod/lib/libOMoocobUC.so

ネーミングサービスを利用するときは、以下のライブラリを追加します。
［UNICODEモジュール以外の場合］
/opt/FSUNod/lib/libOMcnoocob.so
［UNICODEモジュール(NetCOBOL)の場合］
/opt/FSUNod/lib/libOMcnoocobUC.so

Linuxアプリケーションの作成 Linux32

LinuxでOOCOBOLアプリケーションを作成する場合の注意点を以下に示します。

注意

- UNICODEのアプリケーションを開発する場合、ソースファイルおよび登録集は、UNICODE（UTF-8）で作成してください。
- 翻訳／リンク／実行を行う場合、コード系には“UNICODE”を設定してください。

ポイント

CORBAサービスのインストールディレクトリを“/opt/FJSVod”として説明します。

クライアントアプリケーションの作成

クライアントアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。

翻訳・リンク手順

1. CORBA-OOCOBOLマッピングを出力します。
   IDLc -oocob IDLファイル名
2. 相互参照がある場合は、リポジトリファイルを作成します（XXX.cob: 主プログラム名）。相互参照がない場合は、必要ありません。
   cobol -WC,"CREATE(REP)" XXX.cob
3. 主プログラム以外を翻訳し、オブジェクトを作成します（翻訳の順番は、“アプリケーションのビルド時の注意事項”参照）。
   cobol -Tm -c -R/opt/FJSVod/rep -I/opt/FJSVod/include/oocob XXX.cob
4. 主プログラムを作成します。-Mオプションを指定してオブジェクトを作成します。
   cobol -Tm -c -M -R/opt/FJSVod/rep -I/opt/FJSVod/include/oocob XXX.cob
5. ファイルをリンクし、実行ファイルを作成します。-Tmオプションを指定すると、マルチスレッドで結合します。
   ［UNICODEモジュール以外の場合］
   cobol -Tm -lnsl -lsocket -L/opt/FJSVod/lib -lOMoocob -lOMcnoocob -lrcobol -o 実行ファイル名 XXX.o
   ［UNICODEモジュールの場合］
   cobol -Tm -lnsl -lsocket -L/opt/FJSVod/lib -lOMoocobUC -lOMcnoocobUC -lrcobol -o 実行ファイル名 XXX.o

サーバアプリケーションの作成

サーバアプリケーションを作成する場合の翻訳・リンク手順を以下に示します。

翻訳・リンク手順

1. CORBA-OOCOBOLマッピングを出力します。
   IDLc -oocob IDLファイル名
2. 相互参照がある場合は、リポジトリファイルを作成します（XXX.cob: 主プログラム名）。相互参照がない場合は、必要ありません。
   cobol -WC,"CREATE(REP)" XXX.cob
3. 主プログラム以外を翻訳し、オブジェクトを作成します（翻訳の順番は、“アプリケーションのビルド時の注意事項”参照）。
   cobol -Tm -c -R/opt/FJSVod/rep -I/opt/FJSVod/include/oocob XXX.cob
4. 主プログラムを作成します。-Mオプションを指定してオブジェクトを作成します。
   cobol -Tm -c -M -R/opt/FJSVod/rep -I/opt/FJSVod/include/oocob XXX.cob
5. ファイルをリンクし、実行ファイルを作成します。-Tmオプションを指定すると、マルチスレッドで結合します。
   ［UNICODEモジュール以外の場合］
   cobol -Tm -lnsl -L/opt/FJSVod/lib -lOMoocob -lrcobol -o 実行ファイル名 XXX.o
   ［UNICODEモジュールの場合］
   cobol -Tm -lnsl -L/opt/FJSVod/lib -lOMoocobUC -lrcobol -o 実行ファイル名 XXX.o
6. 共用オブジェクトを作成します。
   ［UNICODEモジュール以外の場合］
   cobol -G -Tm -lnsl -L/opt/FJSVod/lib -lOMoocob -lrcobol -o 共用オブジェクト名 XXX.o
   ［UNICODEモジュールの場合］
   cobol -G -Tm -lnsl -L/opt/FJSVod/lib -lOMoocobUC -lrcobol -o 共用オブジェクト名 XXX.o

リンク時に指定する共用オブジェクト（クライアントアプリケーション／サーバアプリケーション共通）

リンク時に必ず指定するライブラリは以下のとおりです。
［UNICODEモジュール以外の場合］
/opt/FJSVod/lib/libOMoocob.so
［UNICODEモジュールの場合］
/opt/FJSVod/lib/libOMoocobUC.so

ネーミングサービスを利用するときは以下のライブラリを追加します。
［UNICODEモジュール以外の場合］
/opt/FJSVod/lib/libOMcnoocob.so
［UNICODEモジュールの場合］
/opt/FJSVod/lib/libOMcnoocobUC.so

G.3.10 OOCOBOL用登録集について

CORBAサービスが提供する関数、データ宣言等に関する登録集原文について、以下に示します。

格納パス(インストールパスはデフォルト) Windows32/64

```
C:¥Interstage¥ODWIN¥INCLUDE¥OOCOB
```

格納パス(インストールパスはデフォルト) Solaris32

```
/opt/FSUNod/include/oocob
```

格納パス(インストールパスはデフォルト) Linux32

```
/opt/FJSVod/include/oocob
```

G.3.10.1 使用例

(1) 環境部の構成節での宣言

“COPY CORBA--CONST.”宣言をSPECIAL-NAMES内でSYMBOLIC CONSTANT句により行います。

```
000013  ENVIRONMENT    DIVISION.
000014  CONFIGURATION  SECTION.
000015  REPOSITORY.
000016      COPY       CORBA--REP.
000017      .
000018  SPECIAL-NAMES.
000019      SYMBOLIC CONSTANT
000020      COPY CORBA--CONST.
000021      .
```

(2) データ部の作業場所節での宣言

“COPY CORBA--COPY.”宣言をWORKING-STORAGE SECTIONの真下に宣言を行います。レベル番号は、設定しません。

```
000022  WORKING-STORAGE SECTION.
000023  COPY CORBA--COPY.
```

G.3.10.2 登録集原文

OOCOBOL

CORBA--REP.cbl	リポジトリ宣言用登録集
CORBA--CONST.cbl	定数宣言用登録集
CORBA--COPY.cbl	TYPEDEF宣言用登録集

OOCOBOLネーミングサービス

COSNAMING--REP.cbl	ネーミングサービスのリポジトリ宣言用登録集
COSNAMING--CONST.cbl	定数宣言用登録集
COSNAMING--COPY.cbl	TYPEDEF宣言用登録集

OOCOBOLロードバランス Windows32 Solaris32 Linux32

NSLBO--REP.cbl	ロードバランスのリポジトリ宣言用登録集
NSLBO-CONST.cbl	定数宣言用登録集
NSLBO--COPY.cbl	TYPEDEF宣言用登録集

OOCOBOLインタフェースリポジトリ

INTERFACEREP--REP.cbl	インタフェースリポジトリのリポジトリ宣言用登録集
INTERFACEREP--CONST.cbl	定数宣言用登録集
INTERFACEREP--COPY.cbl	TYPEDEF宣言用登録集

G.3.10.3 リポジトリファイル

翻訳時にコンパイラがクラスの情報を参照し、呼び出すメソッドがクラスに存在するか、パラメタが一致するかをチェックするために使用します。

格納ディレクトリ(インストールパスはデフォルト) Windows32/64

```
C:¥Interstage¥ODWIN¥REP
```

格納ディレクトリ(インストールパスはデフォルト) Solaris32/64

UNICODEモジュール以外の場合:

```
/opt/FSUNod/rep
```

UNICODEモジュール(NetCOBOL)の場合:

```
/opt/FSUNod/rep/unicode
```

格納ディレクトリ(インストールパスはデフォルト) Linux32/64

UNICODEモジュール以外の場合:

```
/opt/FJSVod/rep
```

UNICODEモジュールの場合:

```
/opt/FJSVod/rep/unicode
```

(1) データ型のベースクラス

```
ARRAYBASE.rep
OBJECTBASE.rep
SEQUENCEBASE.rep
STRUCTBASE.rep
TIEBASE.rep
UNIONBASE.rep
```

(2) CORBAモジュール配下のデータクラス

```
CORBA-ANY.rep
CORBA-STRING.rep
CORBA-WSTRING.rep
```

(3) 例外クラス

```
CORBA-EXCEPTION.rep
CORBA-SYSTEMEXCEPTION.rep
```

(4) システム例外クラス

```
CORBA-BAD_CONTEXT.rep
CORBA-BAD_INV_ORDER.rep
CORBA-BAD_OPERATION.rep
CORBA-BAD_PARAM.rep
CORBA-BAD_QOS.rep
CORBA-BAD_TYPECODE.rep
CORBA-CODESET_INCOMPATIBLE.rep
CORBA-COMM_FAILURE.rep
CORBA-DATA_CONVERSION.rep
CORBA-FREE_MEM.rep
CORBA-IMP_LIMIT.rep
CORBA-INITIALIZE.rep
CORBA-INTERNAL.rep
CORBA-INTF_REPOS.rep
CORBA-INVALID_TRANSACTION.rep
CORBA-INV_FLAG.rep
CORBA-INV_IDENT.rep
CORBA-INV_OBJREF.rep
CORBA-INV_POLICY.rep
CORBA-MARSHAL.rep
```

```
CORBA-NO_IMPLEMENT.rep
CORBA-NO_MEMORY.rep
CORBA-NO_PERMISSION.rep
CORBA-NO_RESOURCES.rep
CORBA-NO_RESPONSE.rep
CORBA-OBJECT_NOT_EXIST.rep
CORBA-OBJ_ADAPTER.rep
CORBA-ORB-INVALIDNAME.rep
CORBA-PERSIST_STORE.rep
CORBA-REBIND.rep
CORBA-TIMEOUT.rep
CORBA-TRANSACTION_MODE.rep
CORBA-TRANSACTION_REQUIRED.rep
CORBA-TRANSACTION_ROLLEDBACK.rep
CORBA-TRANSACTION_UNAVAILABLE.rep
CORBA-TRANSIENT.rep
CORBA-TYPECODE-BADKIND.rep
CORBA-TYPECODE-BOUNDS.rep
CORBA-UNKNOWN.rep
EXCEPTIONLIST-ELEMENT.rep
```

(5) ユーザ例外クラス

```
CORBA-USEREXCEPTION.rep
CORBA-UNKNOWNUSEREXCEPTION.rep
CORBA-BOUNDS.rep
```

(6) 疑似オブジェクトクラス

```
CORBA-BOA.rep
CORBA-CONTEXT.rep
CORBA-CONTEXTLIST.rep
CORBA-DELEGATE.rep
CORBA-ENVIRONMENT.rep
CORBA-EXCEPTIONLIST.rep
CORBA-NAMEDVALUE.rep
CORBA-NVLIST.rep
CORBA-OBJECT.rep
CORBA-ORB.rep
CORBA-PRINCIPAL.rep
CORBA-REQUEST.rep
CORBA-SERVERREQUEST.rep
CORBA-SERVERREQUESTIMPL.rep
CORBA-TYPECODE.rep
CORBA.rep
NVLIST-ELEMENT.rep
```

(7) 基本型のsequenceクラス

```
SEQUENCE-ANY--HELPER.rep
SEQUENCE-ANY.rep
SEQUENCE-BOOLEAN--HELPER.rep
SEQUENCE-BOOLEAN.rep
SEQUENCE-BUFFER.rep
SEQUENCE-CHAR--HELPER.rep
SEQUENCE-CHAR.rep
SEQUENCE-CORBA-STRING--HELPER.rep
SEQUENCE-CORBA-STRING.rep
SEQUENCE-CORBA-WSTRING--HELPER.rep
SEQUENCE-CORBA-WSTRING.rep
SEQUENCE-DOUBLE--HELPER.rep
SEQUENCE-DOUBLE.rep
SEQUENCE-FLOAT--HELPER.rep
SEQUENCE-FLOAT.rep
```

```
SEQUENCE-LONG--HELPER.rep
SEQUENCE-LONG-LONG--HELPER.rep
SEQUENCE-LONG-LONG.rep
SEQUENCE-LONG.rep
SEQUENCE-OBJECT--HELPER.rep
SEQUENCE-OBJECT.rep
SEQUENCE-OCTET--HELPER.rep
SEQUENCE-OCTET.rep
SEQUENCE-SHORT--HELPER.rep
SEQUENCE-SHORT.rep
SEQUENCE-TYPECODE--HELPER.rep
SEQUENCE-TYPECODE.rep
SEQUENCE-UNSIGNED-LONG--HELPER.rep
SEQUENCE-UNSIGNED-LONG.rep
SEQUENCE-UNSIGNED-SHORT--HELPER.rep
SEQUENCE-UNSIGNED-SHORT.rep
SEQUENCE-WCHAR--HELPER.rep
SEQUENCE-WCHAR.rep
```

(8) その他のORBコアクラス

```
CORBA-IMPLEMENTATIONDEF.rep
CORBA-INPUTSTREAM.rep
CORBA-INTERFACEDEF.rep
CORBA-OUTPUTSTREAM.rep
CORBA-REPOSITORY.rep
FJ-IMPLEMENTATIONREP.rep
FJ-NAMEDOESNTEXIST.rep
```

(9) ネーミングサービスクラス

```
COSNAMING-BINDING--HELPER.rep
COSNAMING-BINDING.rep
COSNAMING-BINDINGITERATOR--HELPER.rep
COSNAMING-BINDINGITERATOR--STUB.rep
COSNAMING-BINDINGITERATOR.rep
COSNAMING-BINDINGLIST--HELPER.rep
COSNAMING-BINDINGOBJECTGROUP.rep
COSNAMING-BINDINGOBJECTGROUP--HELPER.rep
COSNAMING-BINDINGTYPE--HELPER.rep
COSNAMING-INSTALLER--HELPER.rep
COSNAMING-INSTALLER--STUB.rep
COSNAMING-INSTALLER.rep
COSNAMING-ISTRING--HELPER.rep
COSNAMING-NAME--HELPER.rep
COSNAMING-NAMECOMPONENT--HELPER.rep
COSNAMING-NAMECOMPONENT.rep
COSNAMING-NAMINGCONTEXT--HELPER.rep
COSNAMING-NAMINGCONTEXT--STUB.rep
COSNAMING-NAMINGCONTEXT-ALREADYBOUND--HELPER.rep
COSNAMING-NAMINGCONTEXT-ALREADYBOUND.rep
COSNAMING-NAMINGCONTEXT-CANNOTPROCEED--HELPER.rep
COSNAMING-NAMINGCONTEXT-CANNOTPROCEED.rep
COSNAMING-NAMINGCONTEXT-INVALIDNAME--HELPER.rep
COSNAMING-NAMINGCONTEXT-INVALIDNAME.rep
COSNAMING-NAMINGCONTEXT-NOTEMPTY--HELPER.rep
COSNAMING-NAMINGCONTEXT-NOTEMPTY.rep
COSNAMING-NAMINGCONTEXT-NOTFOUND--HELPER.rep
COSNAMING-NAMINGCONTEXT-NOTFOUND.rep
COSNAMING-NAMINGCONTEXT-NOTFOUNDREASON--HELPER.rep
COSNAMING-NAMINGCONTEXT.rep
COSNAMING-NAMINGCONTEXTEXT.rep
COSNAMING-NAMINGCONTEXTEXT-ADDRESS--HELPER.rep
COSNAMING-NAMINGCONTEXTEXT--HELPER.rep
```

```
COSNAMING-NAMINGCONTEXTEXT-INVALIDADDRESS.rep
COSNAMING-NAMINGCONTEXTEXT-INVALIDADDRESS--HELPER.rep
COSNAMING-NAMINGCONTEXTEXT-STRINGNAME--HELPER.rep
COSNAMING-NAMINGCONTEXTEXT--STUB.rep
COSNAMING-NAMINGCONTEXTEXT-URLSTRING--HELPER.rep
COSNAMING-OBJECTGROUP.rep
COSNAMING-OBJECTGROUP--HELPER.rep
COSNAMING-OBJECTGROUPLIST--HELPER.rep
COSNAMING-OBJECTGROUP--STUB.rep
COSNAMING-OBJECTGROUPTYPE--HELPER.rep
SEQUENCE-COSNAMING-BINDING--HELPER.rep
SEQUENCE-COSNAMING-BINDING.rep
SEQUENCE-COSNAMING-BINDINGOBJECTGROUP.rep
SEQUENCE-COSNAMING-BINDINGOBJECTGROUP--HELPER.rep
SEQUENCE-COSNAMING-NAMECOMPONENT--HELPER.rep
SEQUENCE-COSNAMING-NAMECOMPONENT.rep
```

(10) ロードバランスクラス Windows32 Solaris32 Linux32

```
ISOD-LBG.rep
ISOD-LBG-ALREADYBOUND.rep
ISOD-LBG-ALREADYBOUND--HELPER.rep
ISOD-LBG-BADOBJECT.rep
ISOD-LBG-BADOBJECT--HELPER.rep
ISOD-LBG-CANNOTPROCEED.rep
ISOD-LBG-CANNOTPROCEED2.rep
ISOD-LBG-CANNOTPROCEED2--HELPER.rep
ISOD-LBG-CANNOTPROCEED--HELPER.rep
ISOD-LBG--HELPER.rep
ISOD-LBG-LOADBALANCETYPE--HELPER.rep
ISOD-LBG-NOTFOUND.rep
ISOD-LBG-NOTFOUND--HELPER.rep
ISOD-LBG-OBJECTLIST--HELPER.rep
ISOD-LBG-OPERATIONBUSY.rep
ISOD-LBG-OPERATIONBUSY--HELPER.rep
ISOD-LBG--STUB.rep
ISOD-LBO.rep
ISOD-LBO-ALREADYEXIST.rep
ISOD-LBO-ALREADYEXIST--HELPER.rep
ISOD-LBO-BADOBJECT.rep
ISOD-LBO-BADOBJECT--HELPER.rep
ISOD-LBO-BINDINGLBG.rep
ISOD-LBO-BINDINGLBG--HELPER.rep
ISOD-LBO-CANNOTPROCEED.rep
ISOD-LBO-CANNOTPROCEED2.rep
ISOD-LBO-CANNOTPROCEED2--HELPER.rep
ISOD-LBO-CANNOTPROCEED--HELPER.rep
ISOD-LBO-DOWNSERVERLIST--HELPER.rep
ISOD-LBO-DUPLICATETYPE--HELPER.rep
ISOD-LBO--HELPER.rep
ISOD-LBO-INVALIDARGUMENT.rep
ISOD-LBO-INVALIDARGUMENT--HELPER.rep
ISOD-LBO-INVALIDNAME.rep
ISOD-LBO-INVALIDNAME--HELPER.rep
ISOD-LBO-INVALIDTYPE.rep
ISOD-LBO-INVALIDTYPE--HELPER.rep
ISOD-LBO-LBGLIST--HELPER.rep
ISOD-LBO-LOADBALANCETYPE--HELPER.rep
ISOD-LBO-NOTEMPTY.rep
ISOD-LBO-NOTEMPTY--HELPER.rep
ISOD-LBO-NOTFOUND.rep
ISOD-LBO-NOTFOUND--HELPER.rep
ISOD-LBO-NOTFOUNDREASON--HELPER.rep
```

```
ISOD-LBO-OPERATIONBUSY.rep
ISOD-LBO-OPERATIONBUSY--HELPER.rep
ISOD-LBO--STUB.rep
SEQUENCE-ISOD-LBO-BINDINGLBG.rep
SEQUENCE-ISOD-LBO-BINDINGLBG--HELPER.rep
```

(11) インタフェースリポジトリクラス

```
CORBA-ALIASDEF--HELPER.rep
CORBA-ALIASDEF--STUB.rep
CORBA-ALIASDEF.rep
CORBA-ARRAYDEF--HELPER.rep
CORBA-ARRAYDEF--STUB.rep
CORBA-ARRAYDEF.rep
CORBA-ATTRDESCRIPTIONSEQ--HELPER.rep
CORBA-ATTRIBUTEDEF--HELPER.rep
CORBA-ATTRIBUTEDEF--STUB.rep
CORBA-ATTRIBUTEDEF.rep
CORBA-ATTRIBUTEDESCRIPTION--HELPER.rep
CORBA-ATTRIBUTEDESCRIPTION.rep
CORBA-ATTRIBUTEMODE--HELPER.rep
CORBA-CONSTANTDEF--HELPER.rep
CORBA-CONSTANTDEF--STUB.rep
CORBA-CONSTANTDEF.rep
CORBA-CONSTANTDESCRIPTION--HELPER.rep
CORBA-CONSTANTDESCRIPTION.rep
CORBA-CONTAINED--HELPER.rep
CORBA-CONTAINED--STUB.rep
CORBA-CONTAINED-DESCRIPTION--HELPER.rep
CORBA-CONTAINED-DESCRIPTION.rep
CORBA-CONTAINED.rep
CORBA-CONTAINEDSEQ--HELPER.rep
CORBA-CONTAINER--HELPER.rep
CORBA-CONTAINER--STUB.rep
CORBA-CONTAINER-DESCRIPTION--HELPER.rep
CORBA-CONTAINER-DESCRIPTION.rep
CORBA-CONTAINER-DESCRIPTIONSEQ--HELPER.rep
CORBA-CONTAINER.rep
CORBA-CONTEXTIDENTIFIER--HELPER.rep
CORBA-CONTEXTIDSEQ--HELPER.rep
CORBA-DEFINITIONKIND--HELPER.rep
CORBA-ENUMDEF--HELPER.rep
CORBA-ENUMDEF--STUB.rep
CORBA-ENUMDEF.rep
CORBA-ENUMMEMBERSEQ--HELPER.rep
CORBA-EXCDESCRIPTIONSEQ--HELPER.rep
CORBA-EXCEPTIONDEF--HELPER.rep
CORBA-EXCEPTIONDEF--STUB.rep
CORBA-EXCEPTIONDEF.rep
CORBA-EXCEPTIONDEFSEQ--HELPER.rep
CORBA-EXCEPTIONDESCRIPTION--HELPER.rep
CORBA-EXCEPTIONDESCRIPTION.rep
CORBA-FIXEDDEF--HELPER.rep
CORBA-FIXEDDEF--STUB.rep
CORBA-FIXEDDEF.rep
CORBA-IDENTIFIER--HELPER.rep
CORBA-IDLTYPE--HELPER.rep
CORBA-IDLTYPE--STUB.rep
CORBA-IDLTYPE.rep
CORBA-INTERFACEDEF--HELPER.rep
CORBA-INTERFACEDEF--STUB.rep
CORBA-INTERFACEDEF-FULLINTERFACEDESCRIPTION--HELPER.rep
CORBA-INTERFACEDEF-FULLINTERFACEDESCRIPTION.rep
```

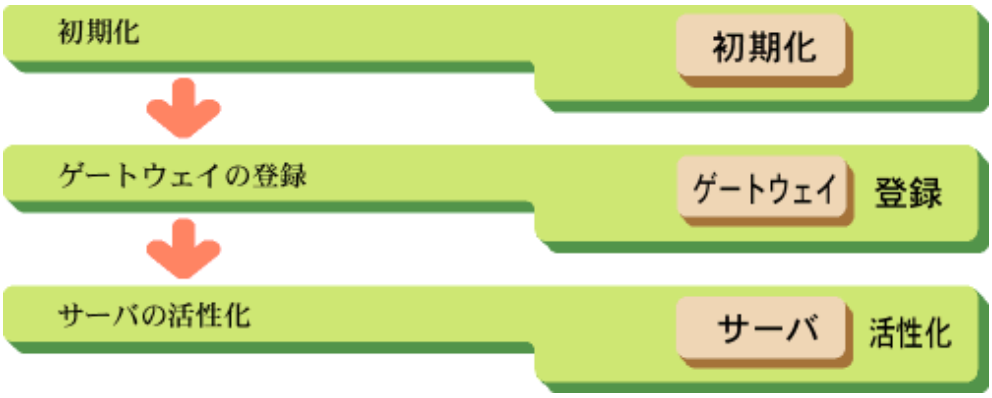
```
CORBA-INTERFACEDEF.rep
CORBA-INTERFACEDEFSEQ--HELPER.rep
CORBA-INTERFACEDESCRIPTION--HELPER.rep
CORBA-INTERFACEDESCRIPTION.rep
CORBA-INTERFACEREP--HELPER.rep
CORBA-INTERFACEREP--STUB.rep
CORBA-INTERFACEREP.rep
CORBA-IROBJECT--HELPER.rep
CORBA-IROBJECT--STUB.rep
CORBA-IROBJECT.rep
CORBA-MODULEDEF--HELPER.rep
CORBA-MODULEDEF--STUB.rep
CORBA-MODULEDEF.rep
CORBA-MODULEDESCRIPTION--HELPER.rep
CORBA-MODULEDESCRIPTION.rep
CORBA-OPDESCRIPTIONSEQ--HELPER.rep
CORBA-OPERATIONDEF--HELPER.rep
CORBA-OPERATIONDEF--STUB.rep
CORBA-OPERATIONDEF.rep
CORBA-OPERATIONDESCRIPTION--HELPER.rep
CORBA-OPERATIONDESCRIPTION.rep
CORBA-OPERATIONMODE--HELPER.rep
CORBA-PARAMETERDESCRIPTION--HELPER.rep
CORBA-PARAMETERDESCRIPTION.rep
CORBA-PARAMETERMODE--HELPER.rep
CORBA-PARDESCRIPTIONSEQ--HELPER.rep
CORBA-PRIMITIVEDEF--HELPER.rep
CORBA-PRIMITIVEDEF--STUB.rep
CORBA-PRIMITIVEDEF.rep
CORBA-PRIMITIVEKIND--HELPER.rep
CORBA-REPOSITORY--HELPER.rep
CORBA-REPOSITORY--STUB.rep
CORBA-REPOSITORY.rep
CORBA-REPOSITORYID--HELPER.rep
CORBA-REPOSITORYIDSEQ--HELPER.rep
CORBA-SCOPEDNAME--HELPER.rep
CORBA-SEQUENCEDEF--HELPER.rep
CORBA-SEQUENCEDEF--STUB.rep
CORBA-SEQUENCEDEF.rep
CORBA-STRINGDEF--HELPER.rep
CORBA-STRINGDEF--STUB.rep
CORBA-STRINGDEF.rep
CORBA-STRUCTDEF--HELPER.rep
CORBA-STRUCTDEF--STUB.rep
CORBA-STRUCTDEF.rep
CORBA-STRUCTMEMBER--HELPER.rep
CORBA-STRUCTMEMBER.rep
CORBA-STRUCTMEMBERSEQ--HELPER.rep
CORBA-TYPEDEFDEF--HELPER.rep
CORBA-TYPEDEFDEF--STUB.rep
CORBA-TYPEDEFDEF.rep
CORBA-TYPEDESCRIPTION--HELPER.rep
CORBA-TYPEDESCRIPTION.rep
CORBA-UNIONDEF--HELPER.rep
CORBA-UNIONDEF--STUB.rep
CORBA-UNIONDEF.rep
CORBA-UNIONMEMBER--HELPER.rep
CORBA-UNIONMEMBER.rep
CORBA-UNIONMEMBERSEQ--HELPER.rep
CORBA-VERSIONSPEC--HELPER.rep
CORBA-WSTRINGDEF--HELPER.rep
CORBA-WSTRINGDEF--STUB.rep
CORBA-WSTRINGDEF.rep
```

```
SEQUENCE-CORBA-ATTRIBUTEDESCRIPTION--HELPER.rep
SEQUENCE-CORBA-ATTRIBUTEDESCRIPTION.rep
SEQUENCE-CORBA-CONTAINED--HELPER.rep
SEQUENCE-CORBA-CONTAINED.rep
SEQUENCE-CORBA-CONTAINER-DESCRIPTION--HELPER.rep
SEQUENCE-CORBA-CONTAINER-DESCRIPTION.rep
SEQUENCE-CORBA-CONTEXTIDENTIFIER--HELPER.rep
SEQUENCE-CORBA-CONTEXTIDENTIFIER.rep
SEQUENCE-CORBA-EXCEPTIONDEF--HELPER.rep
SEQUENCE-CORBA-EXCEPTIONDEF.rep
SEQUENCE-CORBA-EXCEPTIONDESCRIPTION--HELPER.rep
SEQUENCE-CORBA-EXCEPTIONDESCRIPTION.rep
SEQUENCE-CORBA-IDENTIFIER--HELPER.rep
SEQUENCE-CORBA-IDENTIFIER.rep
SEQUENCE-CORBA-INTERFACEDEF--HELPER.rep
SEQUENCE-CORBA-INTERFACEDEF.rep
SEQUENCE-CORBA-OPERATIONDESCRIPTION--HELPER.rep
SEQUENCE-CORBA-OPERATIONDESCRIPTION.rep
SEQUENCE-CORBA-PARAMETERDESCRIPTION--HELPER.rep
SEQUENCE-CORBA-PARAMETERDESCRIPTION.rep
SEQUENCE-CORBA-REPOSITORYID--HELPER.rep
SEQUENCE-CORBA-REPOSITORYID.rep
SEQUENCE-CORBA-STRUCTMEMBER--HELPER.rep
SEQUENCE-CORBA-STRUCTMEMBER.rep
SEQUENCE-CORBA-UNIONMEMBER--HELPER.rep
SEQUENCE-CORBA-UNIONMEMBER.rep
```

G.4 動的スケルトンインタフェースのプログラミング

G.4.1 C言語の開発

動的スケルトンインタフェース(DSI)のサーバアプリケーションは、初期化処理部とサーバアプリケーションが実装するゲートウェイから構成されます。初期化処理部では、以下に示す処理を行います。



G.4.1.1 初期化

CORBAの初期化メソッドCORBA_ORB_init()を呼び出し、初期化処理を行います。結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。
また、基本オブジェクトアダプタの初期化処理を行います。

G.4.1.2 ゲートウェイの登録

サーバアプリケーションが実装するゲートウェイをORBに通知するため、以下の処理を行います。

(1)インプリメンテーションリポジトリのオブジェクトリファレンスの獲得

CORBA_ORB_resolve_initial_references()により、インプリメンテーションリポジトリのオブジェクトリファレンスを取り出します。パラメタとして、CORBA_ORB_ObjectId_ImplementationRepositoryを指定します。

(2)ImplementationRepオブジェクトのオブジェクトリファレンスの獲得

FJ_ImplementationRep_lookup_id()により、サーバアプリケーションのImplementationRepオブジェクトのオブジェクトリファレンスを求めます。

(3)ゲートウェイの登録

CORBA_BOA_set_impl_dsi()により、サーバアプリケーションが実装するゲートウェイを登録します。

```
CORBA_BOA_set_impl_dsi(
    boa, &env, (CORBA_DynamicImplementationRoutine)dsi );
```

G.4.1.3 サーバの活性化

サーバアプリケーションの初期化が完了すると、ORBに対してその旨を通知します。ORBは、この命令が発行された時点で、クライアントからの要求をサーバアプリケーションに伝えます。

G.4.1.4 ゲートウェイの処理

ゲートウェイでは、以下に示す処理を行います。

- メソッドの解析
- パラメタの組立て
- パラメタの解析
- 処理部の起動
- 復帰情報の設定
- 例外情報の設定

(1)メソッドの解析

CORBA_ServerRequest_op_name()により、メソッド名の解析を行います。下の例ではゲートウェイ関数としてstaticな関数を使用していますが、staticである必要はありません。ゲートウェイ関数の第1引数はサーバオブジェクトリファレンス、第2引数はServerRequestオブジェクト、第3引数は例外情報を設定するためのenv構造体です。

```
static void
dsi(
    CORBA_Object         obj,
    CORBA_ServerRequest  request,
    CORBA_Environment    *env )
{

    CORBA_Identifier  dsi_op_name;
```

```
    dsi_op_name = CORBA_ServerRequest_op_name(
                        request,
                        env );
    if( strcmp( dsi_op_name, "calculate" ) == 0 )
        method_calculate( request, env );
    CORBA_free( dsi_op_name );
    return;
}
```

(2)パラメタの組立て

リストオブジェクトの生成

CORBA_ORB_create_list()により、サーバアプリケーションに渡すパラメタを格納するための領域を保持するリストオブジェクト(NVListオブジェクト)を生成します。パラメタとして、何個のパラメタを格納するかを指定します。リストオブジェクトのオブジェクトリファレンスが返されます。

```
CORBA_NVList  arg_list;
CORBA_ORB_create_list( orb, 2, &arg_list, env );
```

パラメタリストの設定

CORBA_NVList_add_item()により、サーバアプリケーションに渡すパラメタをリストオブジェクトに設定します。パラメタには、CORBA_NVListオブジェクトリファレンス、サーバアプリケーションのパラメタの名前、型、値および長さを指定します。

```
CORBA_NVList_add_item(
    arg_list,
    "a", TC_long, NULL,
    sizeof( CORBA_long ),
    CORBA_ARG_IN,
    env );

CORBA_NVList_add_item(
    arg_list,
    "b", TC_long, NULL,
    sizeof( CORBA_long ),
    CORBA_ARG_IN,
    env );
```

(3)パラメタの解析

CORBA_ServerRequest_params()により、パラメタを解析します。

```
CORBA_ServerRequest_params(
    dsi_request,
    arg_list,
    env );
```

(4)復帰情報の設定

CORBA_ServerRequest_result()により、復帰情報を設定します。

```
  CORBA_long                  a, b;
  ODdemo_calculator_result    *res;
  CORBA_any                   any_value;

  a = *(long *)(arg_list[0].argument._value );
  b = *(long *)(arg_list[1].argument._value );

  if( b == 0 ){
      CORBA_ServerRequest_exception(
```

```
        dsi_request,
        CORBA_USER_EXCEPTION,
        ex_ODdemo_calculator_ZEROPARAM,
        NULL,
        env );
    return;
}
res = ODdemo_calculator_result_alloc();
res->add_result = a+b;
res->subtract_result = a/b;
res->multiple_result = a*b;
res->divide_result = (CORBA_float)a/b;

any_value._type = TC_ODdemo_calculator_result;
any_value._value = res;
CORBA_ServerRequest_result(
    dsi_request,
    any_value,
    env );
```

(5)例外情報の設定

CORBA_ServerRequest_exception()を発行して、例外情報を設定します。

G.4.1.5 サーバの非活性化

サーバアプリケーションは、利用者などからの停止要求を受けた場合、クライアントからの要求を以降受け付けない旨をORBに対して通知します。ORBはこの通知を受けて、クライアントからの処理要求をサーバアプリケーションに伝えず、クライアントに例外を返します。

G.4.1.6 パラメタの獲得／解放

動的スケルトンインタフェースを使用して動的にパラメタを作成する方法について説明します。
パラメタはCORBA_NVList_add_item()で設定します。

inパラメタ

クライアントアプリケーションからinパラメタを受け取る場合、パラメタ領域の獲得／解放を行う必要はありません。
CORBA_NVList_add_item()は、以下のように指定します。

```
CORBA_NVList_add_item(
    arg_list,
    name,               /* IDLで指定したパラメタの名前を設定します。 */
    type,               /* パラメタのTypeCodeを設定します。 */
    NULL,               /* NULLを設定します */
    0,                  /* 0を設定します */
    CORBA_ARG_IN,       /* CORBA_ARG_INを設定します */
    &env );
```

outパラメタ

サーバアプリケーションの処理結果をoutパラメタでクライアントアプリケーションに渡す場合、各データ領域獲得関数(CORBA_long_alloc()など)で領域を獲得し、そのポインタをCORBA_NVList_add_item()の第4パラメタに指定します。

```
CORBA_NVList_add_item(
    arg_list,
```

```
        name,                   /* IDLで指定したパラメタの名前を設定します。 */
        type,                   /* パラメタのTypeCodeを設定します。 */
        &param,                 /* パラメタ領域へのポインタを設定します */
        sizeof( CORBA_long ),   /* パラメタのサイズを設定します */
        CORBA_ARG_OUT,          /* CORBA_ARG_OUTを設定します */
        &env );
```

獲得した領域は、CORBAサービス内でリクエスト復帰後に解放されます。

inoutパラメタ

クライアントアプリケーションからinoutパラメタを受け取る場合には、パラメタ領域の獲得／解放を行う必要はありません。CORBA_NVList_add_item()は、以下のように指定します。

```
CORBA_NVList_add_item(
        arg_list,
        name,                   /* IDLで指定したパラメタの名前を設定します。 */
        type,                   /* パラメタのTypeCodeを設定します。 */
        NULL,                   /* NULLを設定します */
        0,                      /* 0を設定します */
        CORBA_ARG_INOUT,        /* CORBA_ARG_INOUTを設定します */
        &env );
```

サーバアプリケーションの処理結果をinoutパラメタでクライアントアプリケーションに渡す場合は、以下のようにします。

固定長データの場合

CORBA_ServerRequest_params()で通知されたパラメタ領域に値を設定します。

可変長データの場合

CORBA_ServerRequest_params()で通知されたパラメタ領域を、CORBA_free()でいったん解放した後、データ域獲得関数（CORBA_string_alloc()等）により領域を獲得し、そのポインタを再設定します。受け渡すデータはその領域に設定します。

inoutパラメタの領域は、リクエスト復帰後にスケルトンで解放されます。

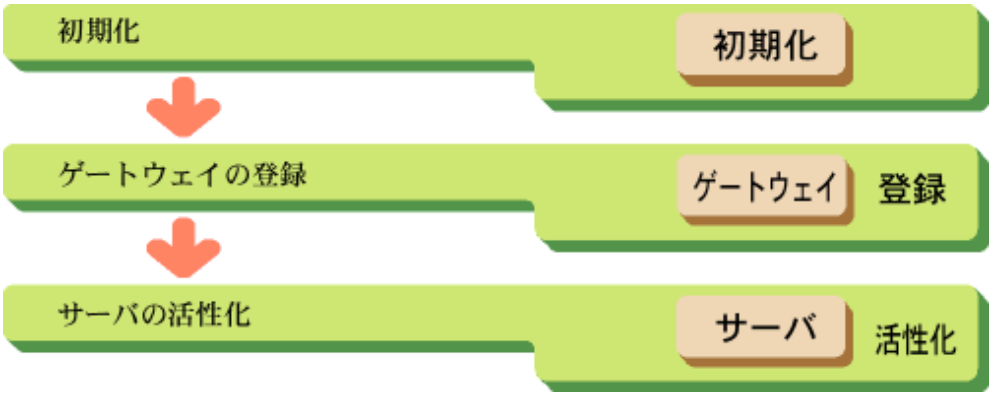
復帰パラメタ

サーバアプリケーションの処理結果を復帰パラメタ渡す場合、データ域獲得関数（CORBA_long_alloc()など）で領域を獲得します。

G.4.2 C++言語の開発

G.4.2.1 Basic Object Adapter : BOA

動的スケルトンインタフェース(DSI)のサーバアプリケーションは、初期化処理部とサーバアプリケーションが実装するゲートウェイから構成されます。初期化処理部では、以下に示す処理を行います。



G.4.2.1.1 初期化

CORBAの初期化メソッドCORBA::ORB_init()を呼び出し、初期化処理を行います。結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。
また、基本オブジェクトアダプタの初期化処理を行います。

G.4.2.1.2 ゲートウェイの登録

サーバアプリケーションが実装するゲートウェイをORBに対して通知します。このため、以下の手順で処理を行います。

1. CORBA::ORB::resolve_initial_references()によりインプリメンテーションリポジトリのオブジェクトリファレンスを取り出します。メソッドのパラメタとしてCORBA_ORB_ObjectId_ImplementationRepositoryを指定します。
2. サーバアプリケーションのImplementationRepオブジェクトのオブジェクトリファレンスを求めるため、FJ::ImplementationRep::lookup_id()を発行します。
3. サーバアプリケーションが実装するゲートウェイを登録するために、CORBA::BOA::set_impl_dsi()を発行します。

```
boa->set_impl_dsi(
    boa,
    *env,
    (CORBA_DynamicImplementationRoutine_cpp)dsi );
```

G.4.2.1.3 サーバの活性化

サーバアプリケーションの初期化が完了すると、ORBに対してその旨を通知します。ORBは、この命令が発行された時点で、クライアントからの要求をサーバアプリケーションに伝えます。

G.4.2.1.4 ゲートウェイの処理

ゲートウェイでは、以下に示す処理を行います。

- メソッドの解析
- パラメタの組立て
- パラメタの解析
- 処理部の起動
- 復帰情報の設定
- 例外情報の設定

ゲートウェイの実装例を以下に示します。

```
static void
dsi(
    CORBA::Object_ptr          obj,
    CORBA::ServerRequest_ptr   request,
    CORBA::Environment         &ev )
{
    CORBA::RepositoryId dsi_op_name;

    try {
        // (1) メソッドの解析
        dsi_op_name = request->op_name( ev );
        if( strcmp( dsi_op_name, "calculate" ) == 0 ) {
            method_calculate( request );
```

```
        }
        CORBA::string_free( dsi_op_name );
    }
    // (5) 例外情報の設定
    catch ( ODdemo::calculator::ZEROPARAM &_zeroparam ) {
        CORBA::Any *_any = new CORBA::Any( _tc_ODdemo_calculator_ZEROPARAM, &_zeroparam );
        request->exception( _any, ev );
    }
    catch ( CORBA::Exception e ){
        // 例外発生時の処理を記述してください。
        // 復帰情報あるいは例外情報を設定してください
    }
    return;
}

void
method_calculate( CORBA::ServerRequest_ptr request )
throw( CORBA::Exception )
{
    try {
        // (2) パラメタの組立て
        // パラメタリストの生成
        CORBA::NVList_ptr    arg_list;
        orb->create_list( 2, arg_list, *env );

        CORBA::Any p1, p2;

        CORBA::Long l;

        // パラメタ情報の登録
        l = 0;
        p1 <<= l;
        arg_list->add_value(
                            "a",
                            p1,
                            CORBA::ARG_IN,
                            *env );
        p2 <<= l;
        arg_list->add_value(
                            "b",
                            p2,
                            CORBA::ARG_IN,
                            *env );

        // (3) パラメタの解析
        request->params(
                        arg_list,
                        *env );

        CORBA::NamedValue_ptr    nvp1, nvp2;

        nvp1 = arg_list->item(0,*env);
        nvp2 = arg_list->item(1,*env);

        CORBA::Any *r1, *r2;
        r1 = nvp1->value(*env);
        r2 = nvp2->value(*env);

        CORBA::Long a,b;
         (*r1) >>= a;
         (*r2) >>= b;
        cout << " a = [" << a << "] b = [" << b << "]" << endl;
```

```
        if ( b == 0 ){
            ODdemo::calculator::ZEROPARAM excep = ODdemo::calculator::ZEROPARAM();
            throw( excep );
        }

        // (4) 復帰情報の設定
        ODdemo::calculator::result *res =
            new ODdemo::calculator::result();
        res->add_result = a+b;
        res->subtract_result = a-b;
        res->multiple_result = a*b;
        res->divide_result = (CORBA::Float)a/b;

        CORBA::Any *tmp_any = new CORBA::Any;
        tmp_any->replace( _tc_ODdemo_calculator_result, (void *)res, CORBA_TRUE );
        request->result( tmp_any, *env );
    }
    catch ( ODdemo::calculator::ZEROPARAM &_zeroparam ) {
        throw( _zeroparam );
    }
    catch( CORBA::Exception e ) {
        throw( e );
    }
}
```

(1)メソッドの解析

CORBA::ServerRequest::op_name()を発行して、メソッド名を解析します。

(2)パラメタの組立て

パラメタリストの生成

CORBA::ORB::create_list()メソッドにより、アプリケーションに渡すパラメタを格納するための領域を保持するリストオブジェクトを生成します。このとき、引数として何個のパラメタを格納するかを指定します。結果として、CORBA::NVListオブジェクトが返されます。

パラメタ情報の登録

CORBA::NVList::add_item()により、サーバアプリケーションに渡すパラメタをリストオブジェクトに設定します。パラメタとして、CORBA::NVListオブジェクトリファレンス、サーバアプリケーションのパラメタの名前、型、値、および長さを設定します。

(3)パラメタの解析

CORBA::ServerRequest::params()を発行して、パラメタを解析します。この結果、アプリケーションに渡すパラメタ値が取得されます。

(4)復帰情報の設定

CORBA::ServerRequest::result()を発行して、復帰情報を設定します。

(5)例外情報の設定

CORBA::ServerRequest::exception()を発行して、例外情報を設定します。

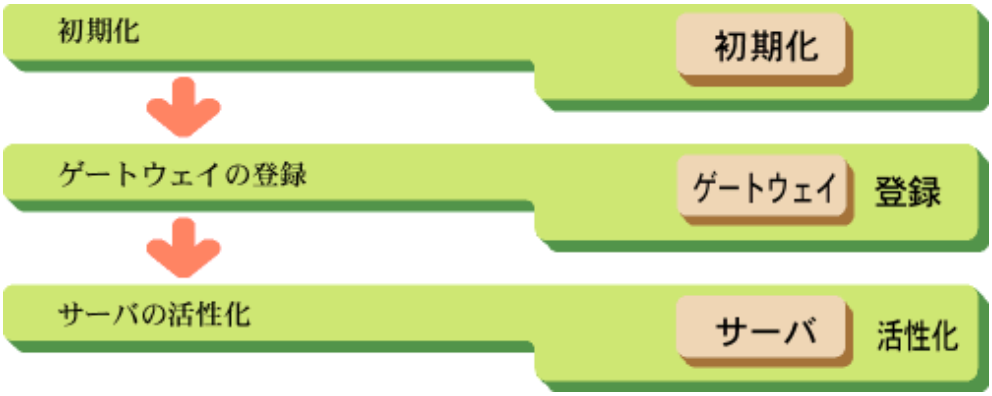
G.4.2.1.5 サーバの非活性化

サーバアプリケーションは、利用者等からの停止要求を受けた場合、クライアントからの要求を以降受け付けない旨をORBに対して通知します。ORBはこの通知を受けて、クライアントからの処理要求をサーバアプリケーションに伝えず、クライアントに例外を返します。

G.4.2.2 Portable Object Adpter : POA

動的スケルトンインタフェース(DSI)のサーバアプリケーションは、初期化処理部とインタフェースの実装部分から構成されます。

初期化処理部では、以下に示す処理を行います。以下の説明では、Servantオブジェクトを Default Servantとして扱っています。



G.4.2.2.1 初期化

初期化処理として以下の処理を行います。

- CORBA::ORB_init()によりORBオブジェクトを取得します。
- RootPOAのオブジェクトリファレンスを取得します。
- RootPOAから、Default Servantを使用するポリシで子POAを生成します。

G.4.2.2.2 ゲートウェイの登録

ゲートウェイを実装したServantオブジェクトを生成し、Default Servantとして子POAに登録します。これによりクライアントからの要求に対して、Servant内に記述したinvoke()メソッドが起動されます。

G.4.2.2.3 サーバの活性化

Servantオブジェクトを設定したPOAに関連付けられたPOAManagerのインスタンスに対て、しactivate()メソッドを発行します。

G.4.2.2.4 ゲートウェイの処理

ゲートウェイでは、以下に示す処理を行います。

- メソッドの解析
- パラメタの組立て
- パラメタの解析
- 復帰情報の設定

・ 例外情報の設定

ゲートウェイの実装例を以下に示します。

【ゲートウェイ実装例】

```
class DSIServant : public virtual PortableServer::DynamicImplementation
{
public:
    DSIServant(){ _rep_id = "IDL:ODdemo/calculator:1.0"; };
    ~DSIServant(){};

    CORBA::Boolean invoke(
        CORBA::ServerRequest_ptr sr,
        CORBA::Environment &env )
    {
        try{
            // (1)メソッド名の取得
            char *_name = sr->op_name();
            //メソッド名の解析
            if( !strcmp( _name, "calculate" )){
                // (2)パラメタリストの設定
                //パラメタリストの作成
                CORBA::NVList_ptr _list = NULL;
                orb->create_list( 2, _list );

                //パラメタ情報の登録
                CORBA::Any _p1, _p2;
                _p1 <<= (CORBA::Long)0;
                _p2 <<= (CORBA::Long)0;

                _list->add_value( "a", _p1, CORBA::ARG_IN );
                _list->add_value( "b", _p2, CORBA::ARG_IN );

                // (3)パラメタの解析
                sr->params(_list);
                CORBA::Long _param1, _param2;
                *(_list->item(0)->value()) >>= _param1;
                *(_list->item(1)->value()) >>= _param2;

                //処理部の起動
                UserServant *_sv = new UserServant();
                ODdemo::calculator::result *_result = _sv->calculate( _param1, _param2, env );

                // (4)復帰情報の設定
                CORBA::Any *_res = new CORBA::Any();
                _res->replace( _tc_ODdemo_calculator_result, (void*)_result, CORBA_TRUE );
                sr->result(_res);

                _sv->servant_delete();
            }
            CORBA::string_free(_name);

        } catch( ODdemo::calculator::ZEROPARAM *_zeroparam ) {
            // (5) 例外情報の設定
            env.exception(_zeroparam);
            CORBA::Any *_any = new CORBA::Any( _tc_ODdemo_calculator_ZEROPARAM, _zeroparam );
            sr->exception( _any, env );
            return true;
        } catch( CORBA::Exception e ) {
            // 例外発生時の処理を記述してください。
            // 復帰情報あるいは例外情報を設定してください
```

```
        }

        return true;
    };

    // (6)実装が必要なメソッド
    CORBA::RepositoryId _primary_interface(
        const PortableServer::ObjectId &oid,
        PortableServer::POA_ptr poa,
        CORBA::Environment&  env )
    {
        return _rep_id;
    };

private:
    CORBA::RepositoryId _rep_id;
};

//処理部
class UserServant : public virtual PortableServer::ServantBase
{
public:
    UserServant(){};
    ~UserServant(){};

    ODdemo::calculator::result    *calculate(
        CORBA::Long a,
        CORBA::Long b,
        CORBA::Environment &env )
        throw( CORBA::Exception )
    {
        if( b == 0 ){
            throw new ODdemo::calculator::ZEROPARAM();
        }
        ODdemo::calculator::result *_result =
            new ODdemo::calculator::result();
        _result->add_result = a + b;
        _result->subtract_result = a - b;
        _result->multiple_result = a * b;
        _result->divide_result = (CORBA::Float)(a / b);
        return _result;
    }
};
```

ゲートウェイを実装するServantオブジェクトは、PortableServer::DynamicImplementationを継承するものとして作成します。また、クライアントからのリクエストに対して起動されるメソッドとして**invoke()**メソッドを実装します。起動の際、invoke()の引数としてServerRequestクラスのオブジェクト(上記例ではreq)が渡されます。
処理部では、ゲートウェイを実装するServantオブジェクト内で、そのServantに対してservant_delete()を発行するために、PortableServer::ServantBaseクラスを継承します。

【ゲートウェイServantの必須メソッド(1)】

```
CORBA::boolean invoke( CORBA::ServerRequest_ptr req )
```

(1)メソッドの取得と解析

渡されたServerRequestクラスに登録されているオペレーション名をCORBA::ServerRequest::op_name()メソッドにより取得します。

(2)パラメタの組立て

パラメタリストの生成

CORBA::ORB::create_list()メソッドにより、アプリケーションに渡すパラメタを格納するための領域を保持するリストオブジェクトを生成します。このとき、引数として何個のパラメタを格納するかを指定します。結果として、CORBA::NVListオブジェクトが返されます。上記例では、2です。

パラメタ情報の登録

CORBA::NVList::add_value()メソッドを使用して、パラメタ情報をNVListオブジェクトに格納します。引数として、パラメタ名、パラメタの値を格納するAny型オブジェクト、パラメタ種別（入出力）を設定します。

(3)パラメタの解析

生成したNVListオブジェクトを引数として、CORBA::ServerRequest::params()メソッドを呼び出します。この結果、アプリケーションに渡すパラメタ値が取得されます。

(4)復帰情報の設定

CORBA::ServerRequest::result()メソッドを発行して復帰情報を設定します。

(5)例外情報の設定

CORBA::ServerRequest::exception()メソッドを発行して例外情報を設定します。

(6)その他実装が必要なメソッド

ゲートウェイを実装するServantは、OMG規約上_primary_interface()メソッドを実装しておく必要があります。上記の例のように、PortableServer::POAクラスのオブジェクト、PortableServer::ObjectIdオブジェクトを引数とし、インタフェースリポジトリID相当の文字列（"IDL:xxx:1.0"）を戻り値とする形式で実装します。引数の値をメソッド内部で扱う必要はありません。

【ゲートウェイServantの必須メソッド(2)】

```
virtual CORBA::RepositoryId  _primary_interface(
                               const PortableServer::ObjectId& oid, );
                               PortableServer::POA_ptr         poa,
                               CORBA::Environment& = CORBA::Environment())
                             throw( CORBA::Exception ) = 0;
```

G.4.2.2.5 DSI使用時の注意点

POAでDSIを使用する場合は、RTTIをサポートするC++コンパイラを使用してください。

Windows32/64

【Visual C++の場合】
「プロジェクトの設定」で、RTTIを使用するようにします。

【Makefileでnmakeを使用する場合】
/GRオプションを追加してください。

G.4.2.3 パラメタの獲得／解放

動的スケルトンインタフェースを使用して動的にパラメタを作成する方法について説明します。
パラメタはCORBA::NVList::add_value()を使用して設定します。

inパラメタ

クライアントアプリケーションからinパラメタを受け取る場合、パラメタ領域の獲得／解放を行う必要はありません。add_value()は、以下のように指定します。

```
CORBA::Any        p1( ((*params)[0])->type, &i, CORBA_TRUE );
arg_list->add_value(
      name,                       /* IDLで指定したパラメタの名前を設定します */
      p1,                         /* inで使用する値を設定したAny型を設定します */
      CORBA::ARG_IN,              /* CORBA::ARG_INを設定します */
      *env );
```

outパラメタ

サーバアプリケーションの処理結果をoutパラメタで渡す場合、データ域獲得関数（CORBA::long_alloc()など）で領域を獲得し、そのポインタをadd_value()の第2パラメタに指定します。

```
CORBA::Any        *p2 = new CORBA::Any( ((*params)[1])->type, &i, CORBA_TRUE );
arg_list->add_value(
      name,                       /* IDLで指定したパラメタの名前を設定します */
      *p2,                        /* outの値を設定するAny型を設定します */
      CORBA::ARG_OUT,             /* CORBA::ARG_OUTを設定します */
      *env );
```

獲得した領域は、CORBAサービス内でリクエスト復帰後に解放されます。

inoutパラメタ

クライアントアプリケーションからinoutパラメタを受け取る場合、パラメタ領域の獲得／解放を行う必要はありません。add_value()は、以下のように指定します。

```
CORBA::Any        *p3 = new CORBA::Any(( (*params)[2])->type, &i, CORBA_TRUE );
arg_list->add_value(
      name,                       /* IDLで指定したパラメタの名前を設定します */
      *p3,                        /* inoutの値を設定するAny型を設定します */
      CORBA::ARG_INOUT,           /* CORBA::ARG_INOUTを設定します */
      *env );
```

サーバアプリケーションの処理結果をinoutでクライアントアプリケーションに渡す場合、以下のようにします。

固定長データの場合

CORBA::ServerRequest::params()で通知されたパラメタ領域に値を設定します。

可変長データの場合

CORBA::ServerRequest::params()で通知されたパラメタ領域を、CORBA::free()でいったん解放した後、データ域獲得関数（CORBA::string_alloc()など）で領域を獲得し、そのポインタを再設定します。新しいデータをその領域に設定します。

inoutパラメタとして使用した領域は、リクエスト復帰後にスケルトンで解放されます。

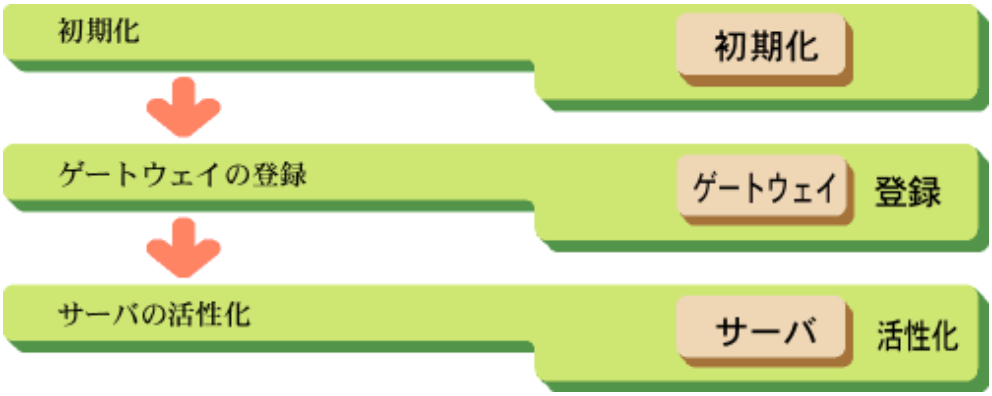
復帰パラメタ

サーバアプリケーションの処理結果を復帰パラメタ渡す場合、データ域獲得関数（CORBA::long_alloc()など）で領域を獲得します。

G.4.3 Java言語の開発

動的スケルトンインタフェース(DSI)のサーバアプリケーションは、初期化処理部とインタフェースの実装部分から構成されます。

初期化処理部では以下に示す処理を行います。以下の説明ではServantオブジェクトをDefault Servantとして扱っています。



G.4.3.1 初期化

初期化処理として以下の処理を行います。

- org.omg.CORBA.ORB.init()によりORBオブジェクトを取得します。
- RootPOAのオブジェクトリファレンスを取得します。
- RootPOAから、Default Servantを使用するポリシで子POAを生成します。

G.4.3.2 ゲートウェイの登録

ゲートウェイを実装したServantオブジェクトを生成し、Default Servantとして子POAに登録します。これによりクライアントからの要求に対して、Servant内に記述したinvoke()メソッドが起動されます。

G.4.3.3 サーバの活性化

Servantオブジェクトを設定したPOAに関連付けられたPOAManagerのインスタンスに対しactivate()メソッドを発行します。

G.4.3.4 ゲートウェイの処理

ゲートウェイでは、以下に示す処理を行います。

- メソッドの解析
- パラメタの組立て
- パラメタの解析
- 復帰情報の設定
- 例外情報の設定

ゲートウェイの実装例を以下に示します。

【ゲートウェイ実装例】

```
class DsiServant
extends org.omg.PortableServer.DynamicImplementation {

    public void
    invoke(org.omg.CORBA.ServerRequest req)
    {
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();

        String name = req.operation();  //(1) メソッド名の取得
        if (name.equals("div"))         //メソッド名の解析
        {

            //(2) パラメタリストの設定
            //パラメタリストの作成
            org.omg.CORBA.NVList args = orb.create_list(2);
            //パラメタ情報の登録
            org.omg.CORBA.Any _a = orb.create_any();
            _a.type(orb.get_primitive_tc(org.omg.CORBA.TCKind.tk_long));
            args.add_value("a", _a, org.omg.CORBA.ARG_IN.value);
            org.omg.CORBA.Any _b = orb.create_any();
            _b.type(orb.get_primitive_tc(org.omg.CORBA.TCKind.tk_long));
            args.add_value("b", _b, org.omg.CORBA.ARG_IN.value);

            //(3) パラメタの解析
            req.arguments(args);
            int a = _a.extract_long();
            int b = _b.extract_long();

            //処理部の起動
            org.omg.CORBA.DoubleHolder ret = new org.omg.CORBA.DoubleHolder();
            UserMethod um = new UserMethod();
            try
            {
                ret.value = um.method1(a, b);

                //(4) 復帰情報の設定
                org.omg.CORBA.Any _ret = orb.create_any();
                _ret.type(orb.get_primitive_tc(
                    org.omg.CORBA.TCKind.tk_double));
                _ret.insert_double(ret.value);
                req.set_result(_ret);
            }
            catch(ZeroDivException e)
            {
                //(5) 例外情報の設定
                org.omg.CORBA.Any exp = orb.create_any();
                org.omg.CORBA.StructMember[] members = {
                    new org.omg.CORBA.StructMember("detail",orb.get_primitive_tc(org.omg.CORBA.TCKind.tk_string), null)
                };
                org.omg.CORBA.TypeCode type =
                    orb.create_exception_tc("IDL:ODsample/ZeroDivException:1.0","ZeroDivException",members);
                exp.type(type);
                org.omg.CORBA.portable.OutputStream os = exp.create_output_stream();
                //例外のIDの設定
                os.write_string("IDL:ODsample/ZeroDivException:1.0");
                //例外のメンバの設定
                os.write_string(zde.detail);
                req.set_exception(exp);
            }
            return;
        }
        throw (new org.omg.CORBA.BAD_OPERATION(
               0, org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE));
```

```
    }

    //(6) 実装が必要なメソッド
    public java.lang.String[] _all_interfaces(
        org.omg.PortableServer.POA poa,
        byte[] objectId) {
          return ids;
    }
    private static java.lang.String[] ids = {"IDL:ODsample/intf:1.0"};
}

//処理部
class UserMethod
{
    public double method1( int a, int b ) throws ZeroDivException{
        if(b == 0)
        {
            throw new ZeroDivException("2nd parameter is 0.");
        }

        //必要な処理を記述
        return( a / b );
    }
}
```

ゲートウェイを実装するServantオブジェクトは、org.omg.PortableServer.DynamicImplementationを継承するものとして作成します。また、クライアントからのリクエストに対して起動されるメソッドとして**invoke()**メソッドを実装します。起動の際、invoke()の引数としてServerRequestクラスのオブジェクト(上記例ではreq)が渡されます。

【ゲートウェイServantの必須メソッド(1)】

```
public void invoke(org.omg.CORBA.ServerRequest req)
```

(1)メソッドの取得と解析

渡されたServerRequestクラスに登録されているオペレーション名をorg.omg.CORBA.ServerRequest.operation()メソッドにより取得します。

(2)パラメタの組立て

パラメタリストの生成

org.omg.CORBA.ORB.create_list()メソッドにより、アプリケーションに渡すパラメタを格納するための領域を保持するリストオブジェクトを生成します。このとき、引数として何個のパラメタを格納するかを指定します。結果として、org.omg.CORBA.NVListオブジェクトが返されます。上記例では、2です。

パラメタ情報の登録

org.omg.CORBA.add_value()メソッドを使用して、パラメタ情報をNVListオブジェクトに格納します。引数として、パラメタ名、パラメタの値を格納するAny型オブジェクト、パラメタ種別(入出力)を設定します。

(3)パラメタの解析

生成したNVListオブジェクトを引数としてorg.omg.ServerRequest.arguments()メソッドを呼び出します。この結果、アプリケーションに渡すパラメタ値が取得されます。

(4)復帰情報の設定

org.omg.CORBA.ServerRequest.set_result()メソッドを発行して復帰情報を設定します。

(5)例外情報の設定

org.omg.CORBA.ServerRequest.set_exception()メソッドを発行して例外情報を設定します。

(6)その他実装が必要なメソッド

ゲートウェイを実装するServantはOMG規約上_all_interfaces()メソッドを実装しておく必要があります。上記の例のように、org.omg.PortableServer.POAクラスのオブジェクト、byte配列オブジェクトを引数とし、インタフェースリポジトリID相当の文字列("IDL:xxx:1.0")を最初の要素に持つString配列オブジェクトを戻り値とする形式で実装します。引数の値をメソッド内部で扱う必要はありません。

【ゲートウェイServantの必須メソッド(2)】

```
public java.lang.String[] _all_interfaces(
          org.omg.PortableServer.POA poa,
          byte[] objectId)
```

G.4.4 COBOLの開発

動的スケルトンインタフェース(DSI)のサーバアプリケーションは、初期化処理部とサーバアプリケーションが実装するゲートウェイから構成されます。

初期化処理部では、以下に示す処理を行います。

注意

COBOLではC言語、C++言語にあるCORBA_BOA_set_impl_dsi関数によるゲートウェイの登録は行わず、OD_impl_instコマンドを使用してインプリメンテーションリポジトリに登録します。登録方法については、“アプリケーションの開発(COBOL)”－“サーバアプリケーションの登録”の“6.4.1 インプリメンテーションリポジトリへの登録”を参照してください。

G.4.4.1 初期化

CORBAの初期化メソッドCORBA-ORB-INITを呼び出して初期化処理を行います。本メソッドの結果として、ORBのオブジェクトリファレンスが返されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。
また、基本オブジェクトアダプタの初期化処理を行います。

G.4.4.2 サーバの活性化

サーバアプリケーションの初期化が完了すると、ORBに対してその旨を通知します。ORBは、この命令が発行された時点で、クライアントからの要求をサーバアプリケーションに伝えます。

G.4.4.3 ゲートウェイの処理

ゲートウェイでは、以下に示す処理を行います。

- メソッドの解析
- パラメタの組立て
- パラメタの解析
- 処理部の起動
- 復帰情報の設定
- 例外情報の設定

ゲートウェイの実装例を以下に示します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "ODDEMO-CALCULATOR".
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
     ARGUMENT-NUMBER IS ARG-C
     ARGUMENT-VALUE  IS ARG-V
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .
 DATA DIVISION.
*
 WORKING-STORAGE SECTION.
 01 COPY REPOSITORYID     IN CORBA        REPLACING
                 CORBA-REPOSITORYID      BY DSI-OP-NAME.

 01 TMP-BUF               USAGE POINTER.
 01 COPY ULONG            IN CORBA        REPLACING
                 CORBA-UNSIGNED-LONG     BY STRING-LENGTH.
 01 COPY ULONG            IN CORBA        REPLACING
                 CORBA-UNSIGNED-LONG     BY RET-VAL.
 01 STR-BUF               PIC X(20).
 01 COPY SERVERREQUEST    IN CORBA        REPLACING
                 CORBA-SERVERREQUEST     BY REQUEST-V.
 01 COPY ENVIRONMENT      IN CORBA        REPLACING
                 CORBA-ENVIRONMENT       BY W-ENV.
 01 COPY BOOLEAN  IN CORBA        REPLACING
                 CORBA-BOOLEAN   BY RET.

*
 LINKAGE SECTION.
 01 COPY OBJECT           IN CORBA        REPLACING
                 CORBA-OBJECT            BY OBJ.
 01 COPY SERVERREQUEST    IN CORBA        REPLACING
                 CORBA-SERVERREQUEST     BY REQUEST.
 01 COPY ENVIRONMENT      IN CORBA        REPLACING
                 CORBA-ENVIRONMENT       BY ENV.
*
 PROCEDURE DIVISION USING OBJ REQUEST ENV.

 MAIN.
```

```
* (1) メソッドの解析
       CALL "CORBA-SERVERREQUEST-OP-NAME" USING
                            REQUEST
                            ENV
                            DSI-OP-NAME.

       MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
       CALL "CORBA-STRING-GET" USING
                            DSI-OP-NAME
                            STRING-LENGTH
                            STR-BUF.
       IF STR-BUF = "calculate"
           CALL "ODDEMO-CALCULATOR-CALCULATE" USING REQUEST ENV
           MOVE "CORBA-SERVERREQUEST-OP-NAME" TO MESS
       END-IF.
       CALL "CORBA-FREE" USING TMP-BUF.

       EXIT PROGRAM.
   MAIN-END.
   END PROGRAM "ODDEMO-CALCULATOR".

   IDENTIFICATION DIVISION.
   PROGRAM-ID. "ODDEMO-CALCULATOR-CALCULATE".
   ENVIRONMENT DIVISION.
   CONFIGURATION SECTION.
   SPECIAL-NAMES.
       ARGUMENT-NUMBER IS ARG-C
       ARGUMENT-VALUE  IS ARG-V
       SYMBOLIC CONSTANT
       COPY SYMBOL-CONST IN CORBA.
       .
   DATA DIVISION.
  *
   WORKING-STORAGE SECTION.
   COPY CONST IN CORBA.
   01 MESS PIC X(30).
   01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
   01 COPY ORB IN CORBA REPLACING CORBA-ORB  BY ORB.
   01 COPY BOA IN CORBA REPLACING CORBA-BOA  BY BOA.
   01 ARG-LIST USAGE POINTER.
   01 COPY TYPECODE IN CORBA REPLACING CORBA-TYPECODE  BY TC-TYPE.
   01 COPY FLAGS  IN CORBA REPLACING CORBA-FLAGS BY FLAG.
   01 STRING-POINTER USAGE POINTER.
   01 PARAM-POINTER USAGE POINTER.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
   01 STRING-TMP  PIC X(10).
   01 COPY ULONG  IN CORBA REPLACING CORBA-UNSIGNED-LONG BY FLAGS.
   01 TEMP-BUF USAGE POINTER.
   01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY RET-VAL.
   01 EXC-POINTER  USAGE POINTER.
   01 COPY EXCEPTION-TYPE IN CORBA REPLACING CORBA-EXCEPTION-TYPE BY ENV-FLAGS.
   01 USER-ID USAGE POINTER.
   01 EX-ODDEMO-CALCULATOR-ZEROPARAM PIC X(35) VALUE
                  "IDL:ODdemo/calculator/ZEROPARAM:1.0".
   01 TC-ODDEMO-CALCULATOR-RESULT.
       03 MEMBER1 PIC X(100) VALUE
"131,131,0,0,0,0,15,4,0,16,0,33,73,68,76,58,79,68,100,101,109,111,47,99,97,108,99,117,108,97,116,111,".
       03 MEMBER2 PIC X(100) VALUE
"114,47,114,101,115,117,108,116,58,49,46,48,0,0,7,114,101,115,117,108,116,0,0,4,0,11,97,100,100,95,11".
       03 MEMBER3 PIC X(100) VALUE
"4,101,115,117,108,116,0,0,1,3,0,16,115,117,98,116,114,97,99,116,95,114,101,115,117,108,116,0,0,1,3,0".
       03 MEMBER4 PIC X(100) VALUE
```

```
",16,109,117,108,116,105,112,108,101,95,114,101,115,117,108,116,0,0,1,3,0,14,100,105,118,105,100,101,".
       03 MEMBER5 PIC X(100) VALUE "95,114,101,115,117,108,116,0,0,1,6".
   01 TC-ODDEMO-CALCULATOR-ZEROPARAM.
       03 MEMBER1 PIC X(100) VALUE
"73,73,0,0,0,0,22,4,0,4,0,36,73,68,76,58,79,68,100,101,109,111,47,99,97,108,99,117,108,97,116,111,114".
       03 MEMBER2 PIC X(100) VALUE
",47,90,69,82,79,80,65,82,65,77,58,49,46,48,0,0,10,90,69,82,79,80,65,82,65,77,0,0,1,0,8,101,120,99,10".
       03 MEMBER3 PIC X(100) VALUE "1,112,95,108,0,0,1,3".
   01 COPY EXCEPTION-TYPE IN CORBA REPLACING CORBA-EXCEPTION-TYPE BY EX-STATUS.
   01 COPY ANY IN CORBA REPLACING CORBA-ANY BY RESULT.
   01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY C-ARG-C.
   01 C-ARG-V.
      02 FILLER OCCURS 2.
         03 ARG-V-VALUE USAGE POINTER.
   01 APLI-NAME PIC X(8) VALUE "simple_s".
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY P1.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY P2.

   LINKAGE SECTION.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARAM1.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARAM2.
   01 A-RESULT.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY ADD-RESULT.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SUBTRACT-RESULT.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY MULTIPLE-RESULT.
      02 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY DEVIDE-RESULT.
   01 LIST-VALUE.
      03 FILLER OCCURS 2.
         05 COPY NAMEDVALUE IN CORBA REPLACING CORBA-NAMEDVALUE BY TMP-NAMVAL.
   01 EXCEP.
      03  COPY LONG IN CORBA REPLACING CORBA-LONG BY EXCEP-L.
   01 COPY ANY IN CORBA REPLACING CORBA-ANY BY EXC-VALUE.
   01 COPY SERVERREQUEST    IN CORBA        REPLACING
                   CORBA-SERVERREQUEST     BY REQUEST.
   01 COPY ENVIRONMENT      IN CORBA        REPLACING
                   CORBA-ENVIRONMENT       BY ENV.
  *
   PROCEDURE DIVISION USING REQUEST ENV.
  *
   MAIN.

       COMPUTE C-ARG-C = 1.
       MOVE FUNCTION LENG (APLI-NAME) TO STRING-LENGTH.
       CALL "CORBA-STRING-SET" USING
             ARG-V-VALUE (1)
             STRING-LENGTH
             APLI-NAME.
       SET ARG-V-VALUE(2) TO NULL.

       MOVE    12 TO STRING-LENGTH.
       CALL    "CORBA-STRING-SET"    USING
             TEMP-BUF
             STRING-LENGTH
             FJ-OM-ORB-ID.
       CALL "CORBA-ORB-INIT" USING
             C-ARG-C
             C-ARG-V
             TEMP-BUF
             ENV
             ORB.
       MOVE "CORBA-ORB-INIT" TO MESS.
       PERFORM ENV-CHECK
```

```
      CALL "CORBA-FREE" USING
            TEMP-BUF.

      MOVE 15 TO STRING-LENGTH.
      CALL    "CORBA-STRING-SET" USING
            TEMP-BUF
            STRING-LENGTH
            CORBA-BOA-OA-ID.
      CALL "CORBA-ORB-BOA-INIT" USING
            ORB
            C-ARG-C
            C-ARG-V
            TEMP-BUF
            ENV
            BOA.
      MOVE "CORBA-ORB-BOA-INIT" TO MESS.
      PERFORM ENV-CHECK.

*(2) パラメタの組み立て
*    パラメタリストの生成
      MOVE 2 TO NUM.
      CALL "CORBA-ORB-CREATE-LIST" USING
                  ORB
                  NUM
                  ARG-LIST
                  ENV
                  RET-VAL.
      MOVE "CORBA-ORB-CREATE-LIST" TO MESS.
      PERFORM ENV-CHECK.

  *   パラメタ情報の登録
      MOVE "a" TO STRING-TMP.
      MOVE FUNCTION LENG (STRING-TMP) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
                  STRING-POINTER
                  STRING-LENGTH
                  STRING-TMP.

      MOVE FUNCTION LENG (TC-LONG) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
                  TEMP-BUF
                  STRING-LENGTH
                  TC-LONG.
      CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
                  TEMP-BUF
                  TC-TYPE.
      CALL "CORBA-FREE" USING
                  TEMP-BUF.

      MOVE FUNCTION ADDR( P1 ) TO  PARAM-POINTER.

      MOVE 0 TO NUM.

      SET CORBA-ARG-IN OF FLAG TO TRUE.

      CALL "CORBA-NVLIST-ADD-ITEM" USING
                  ARG-LIST
                  STRING-POINTER
                  TC-TYPE
                  PARAM-POINTER
                  NUM
                  FLAG
                  ENV
```

```
                RET-VAL.
       MOVE "CORBA-NVLIST-ADD-ITEM" TO MESS.
       PERFORM ENV-CHECK.

       MOVE "b" TO STRING-TMP.
       MOVE FUNCTION LENG (STRING-TMP) TO STRING-LENGTH.
       CALL "CORBA-STRING-SET" USING
                STRING-POINTER
                STRING-LENGTH
                STRING-TMP.

       MOVE FUNCTION LENG (TC-LONG) TO STRING-LENGTH.
       CALL "CORBA-STRING-SET" USING
                TEMP-BUF
                STRING-LENGTH
                TC-LONG.
       CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
                TEMP-BUF
                TC-TYPE.
       CALL "CORBA-FREE" USING
                TEMP-BUF.

       MOVE FUNCTION ADDR( P2 ) TO PARAM-POINTER.

       MOVE 0 TO NUM.

       SET CORBA-ARG-IN OF FLAG TO TRUE.

       CALL "CORBA-NVLIST-ADD-ITEM" USING
                ARG-LIST
                STRING-POINTER
                TC-TYPE
                PARAM-POINTER
                NUM
                FLAG
                ENV
                RET-VAL.
       MOVE "CORBA-NVLIST-ADD-ITEM" TO MESS.
       PERFORM ENV-CHECK.

* (3) パラメタの解析
       CALL "CORBA-SERVERREQUEST-PARAMS" USING
                REQUEST
                ARG-LIST
                ENV.
       MOVE "CORBA-SERVERREQUEST-PARAMS" TO MESS.
       PERFORM ENV-CHECK.

       SET ADDRESS OF LIST-VALUE TO ARG-LIST.
       SET ADDRESS OF PARAM1   TO ANY-VALUE OF  LIST-VALUE(1).
       SET ADDRESS OF PARAM2   TO ANY-VALUE OF  LIST-VALUE(2).

* 例外判定
       IF PARAM2 = 0
* (5) 例外情報の設定
           SET CORBA-USER-EXCEPTION OF ENV-FLAGS TO TRUE

           MOVE FUNCTION LENG ( EX-ODDEMO-CALCULATOR-ZEROPARAM ) TO STRING-LENGTH
           CALL "CORBA-STRING-SET" USING
                       USER-ID
                       STRING-LENGTH
                       EX-ODDEMO-CALCULATOR-ZEROPARAM
```

```
           CALL "CORBA-ANY-ALLOC" USING EXC-POINTER
           SET ADDRESS OF EXC-VALUE TO EXC-POINTER

           MOVE FUNCTION LENG (TC-ODDEMO-CALCULATOR-ZEROPARAM) TO STRING-LENGTH
           CALL "CORBA-STRING-SET" USING
                           TEMP-BUF
                           STRING-LENGTH
                           TC-ODDEMO-CALCULATOR-ZEROPARAM
           CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
                           TEMP-BUF
                           ANY-TYPE OF EXC-VALUE

           CALL "ODDEMO-CALCULATOR-ZEROPARAM-ALLOC" USING  ANY-VALUE OF EXC-VALUE
           SET ADDRESS OF EXCEP TO ANY-VALUE OF EXC-VALUE

           MOVE  0 TO  EXCEP-L OF EXCEP


           CALL "CORBA-SERVERREQUEST-EXCEPTION" USING
                       REQUEST
                       ENV-FLAGS
                       USER-ID
                       EXC-POINTER
                       ENV
           MOVE "CORBA-SERVERREQUEST-EXCEPTION" TO MESS
           PERFORM ENV-CHECK

       ELSE
* (4) 復帰情報の設定
           MOVE FUNCTION LENG (TC-ODDEMO-CALCULATOR-RESULT) TO STRING-LENGTH
           CALL "CORBA-STRING-SET" USING
                           TEMP-BUF
                           STRING-LENGTH
                           TC-ODDEMO-CALCULATOR-RESULT
           CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
                           TEMP-BUF
                           ANY-TYPE OF RESULT

           CALL "ODDEMO-CALCULATOR-RESULT-ALLOC" USING ANY-VALUE OF RESULT
           SET ADDRESS OF A-RESULT TO ANY-VALUE OF RESULT
           COMPUTE  ADD-RESULT      OF A-RESULT = PARAM1 + PARAM2
           COMPUTE  SUBTRACT-RESULT OF A-RESULT = PARAM1 - PARAM2
           COMPUTE  MULTIPLE-RESULT OF A-RESULT = PARAM1 * PARAM2
           COMPUTE  DEVIDE-RESULT   OF A-RESULT = PARAM1 / PARAM2

           CALL "CORBA-SERVERREQUEST-RESULT" USING
                           REQUEST
                           RESULT
                           ENV
           MOVE "CORBA-SERVERREQUEST-PARAMS" TO MESS
           PERFORM ENV-CHECK
       END-IF.
       EXIT PROGRAM.
   MAIN-END.

   ENV-CHECK SECTION.
       EVALUATE TRUE
           WHEN CORBA-NO-EXCEPTION OF MAJOR OF ENV
                CONTINUE
           WHEN CORBA-USER-EXCEPTION OF MAJOR OF ENV
                EXIT PROGRAM
           WHEN CORBA-SYSTEM-EXCEPTION OF MAJOR OF ENV
                EXIT PROGRAM
```

```
      END-EVALUATE.
  ENV-CHECK-END.
      EXIT.
*
  END PROGRAM  "ODDEMO-CALCULATOR-CALCULATE".
```

(1)メソッドの解析

CORBA-SERVERREQUEST-OP-NAMEメソッドを発行して、メソッド名を解析します。

(2)パラメタの組み立て

1. CORBA-ORB-CREATE-LISTメソッドにより、サーバアプリケーションに渡すパラメタを格納するための領域を保持するリストオブジェクトを生成します。このとき、パラメタとして、何個のパラメタを格納するかを指定します。その結果、NVListオブジェクトリファレンスが返されます。
2. CORBA-NVLIST-ADD-ITEMメソッドにより、サーバアプリケーションに渡すパラメタをリストオブジェクトに設定します。パラメタとして、CORBA-NVLISTオブジェクトリファレンス、サーバアプリケーションのパラメタの名前、型、値および長さを設定します。

(3)パラメタの解析

CORBA-SERVERREQUEST-PARAMSメソッドを発行して、パラメタを解析します。

(4)復帰情報の設定

CORBA-SERVERREQUEST-RESULTメソッドを発行して、復帰情報を設定します。

(5)例外情報の設定

CORBA-SERVERREQUEST-EXCEPTIONメソッドを発行して例外情報を設定します。
サーバアプリケーションのメソッドが発生するユーザ例外を設定し、クライアントへ通知します。メソッドが発生するユーザ例外は、IDLのオペレーション定義で記述されます。

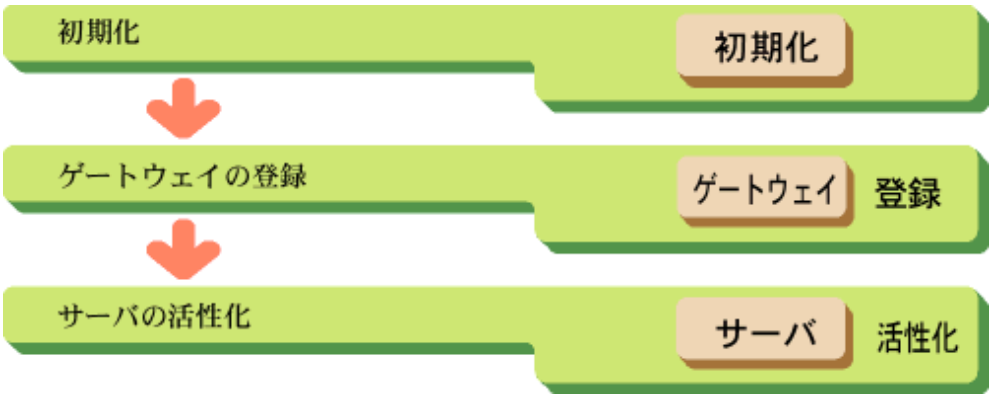
G.4.4.4 サーバの非活性化処理

サーバアプリケーションは、利用者からの停止要求を受けた場合、クライアントからの要求を以降受け付けない旨をORBに対して通知します。

G.4.5 OOCOBOLの開発

Windows32 Solaris32 Linux32

動的スケルトンインタフェース(DSI)のサーバアプリケーションは、初期化処理部とサーバアプリケーションが実装するゲートウェイから構成されます。
初期化処理部では、以下に示す処理を行います。



G.4.5.1 初期化

CORBAの初期化メソッドCORBA-ORB_INITを呼び出して初期化処理を行います。本関数の結果として、ORBのオブジェクトリファレンスが通知されます。このオブジェクトリファレンスは、以降で呼び出すORBインタフェースを使用する場合に指定します。

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "SERVER-MAIN".
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     COPY CORBA--REP.
     COPY SIMPLE--REP.
     CLASS ODDEMO-CALCULATOR-IMPL.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY CORBA--CONST.
     COPY SIMPLE--CONST.
     .
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 COPY CORBA--COPY.
 COPY SIMPLE--COPY.
 01  API-NAME        PIC   X(50).
 01  APL-NAME        PIC   X(8)    VALUE     "Simple_s".
 01  ORB             USAGE OBJECT REFERENCE CORBA-ORB.
 01  BOA             USAGE OBJECT REFERENCE CORBA-BOA.
 01  IMPL-REP        USAGE OBJECT REFERENCE FJ-IMPLEMENTATIONREP.
 01  IMPL            USAGE OBJECT REFERENCE CORBA-IMPLEMENTATIONDEF.
 01  REP-ID          PIC   X(25)   VALUE     "IDL:ODdemo/calculator:1.0".
 01  OBJ             USAGE OBJECT REFERENCE CORBA-OBJECT.
 01  EXCEPT-ID       USAGE OBJECT REFERENCE CORBA-STRING.
 01  EXCEPT-ID-VALUE PIC   X(50).
 01  L-DSIBASE       USAGE OBJECT REFERENCE DSIBASE.
*
 PROCEDURE DIVISION.
*
 DECLARATIVES.
 ERR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     SET EXCEPT-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPT-ID "GET-VALUE" RETURNING EXCEPT-ID-VALUE.
     DISPLAY API-NAME " is failed.".
     DISPLAY "Exception-id is " EXCEPT-ID-VALUE.
     EXIT PROGRAM.
```

```
 END-ERR.
 END DECLARATIVES.
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT" USING      APL-NAME
                                        FJ-OM_ORBID
                              RETURNING  ORB.
*
     MOVE "CORBA::ORB::BOA_init" TO API-NAME.
     INVOKE ORB "BOA_INIT" USING      APL-NAME
                                      CORBA-BOA_OAID
                            RETURNING  BOA.
```

G.4.5.2 ゲートウェイの登録

サーバアプリケーションが実装するゲートウェイをORBに対して通知します。このため、以下の手順で処理を行います。

CORBA-ORB-RESOLVE_INITIAL_REFERENCESメソッドによりインプリメンテーションリポジトリのオブジェクトリファレンスを取り出します。メソッドのパラメタとしてCORBA-OBJECTID_IMPLEMENTAT-001（値は“ImplementationRepository”）を指定します。

サーバアプリケーションのImplementationRepオブジェクトのオブジェクトリファレンスを求めるため、CORBA-IMPLEMENTATIONDEF-LOOKUP_IDを発行します。

サーバアプリケーションが実装するゲートウェイを登録するために、CORBA-BOA-SET_IMPL_DSIを発行します。

```
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES" USING      CORBA-OBJECTID_IMPLEMENTAT-001
                                             RETURNING  OBJ.
     INVOKE FJ-IMPLEMENTATIONREP "NARROW" USING      OBJ
                                          RETURNING  IMPL-REP.
*
     MOVE "FJ::ImplementationRep::lookup_id" TO API-NAME.
     INVOKE IMPL-REP "LOOKUP_ID" USING      REP-ID
                                 RETURNING  OBJ.
     INVOKE CORBA-IMPLEMENTATIONDEF "NARROW" USING      OBJ
                                             RETURNING  IMPL.
*
     INVOKE ODDEMO-CALCULATOR-IMPL "NEW"  RETURNING  L-DSIBASE.
     INVOKE BOA "SET_IMPL_DSI" USING L-DSIBASE.
```

G.4.5.3 サーバの活性化

サーバアプリケーションの初期化が完了すると、ORBに対してその旨を通知します。ORBは、この命令が発行された時点で、クライアントからの要求をサーバアプリケーションに伝えます。

G.4.5.4 ゲートウェイの処理

ゲートウェイでは、以下に示す処理を行います。

(1)メソッドの取得と解析
(2)パラメタの組立て
(3)パラメタの解析
(4)復帰情報の設定
(5)例外情報の設定

ゲートウェイの実装例を以下に示します。

【ゲートウェイ実装例】

```
 CLASS-ID. ODDEMO-CALCULATOR-IMPL INHERITS DSIBASE.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
     COPY CORBA--REP.
     COPY SIMPLE--REP.
     .
 SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY CORBA--CONST.
     COPY SIMPLE--CONST.
     .
*
 OBJECT.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 COPY CORBA--COPY.
 COPY SIMPLE--COPY.
 PROCEDURE DIVISION.
*
 METHOD-ID. INVOKE-METHOD OVERRIDE.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 EXCEP             OBJECT REFERENCE CORBA-SYSTEMEXCEPTION.
 01 L-OP-NAME         OBJECT REFERENCE CORBA-STRING.
 01 L-BUF             PIC X(30).
 01 L-NVLIST          OBJECT REFERENCE CORBA-NVLIST.
 01 NV-A              OBJECT REFERENCE CORBA-NAMEDVALUE.
 01 NV-B              OBJECT REFERENCE CORBA-NAMEDVALUE.
 01 NV-RESULT         OBJECT REFERENCE CORBA-NAMEDVALUE.
 01 ANY-A             OBJECT REFERENCE CORBA-ANY.
 01 ANY-B             OBJECT REFERENCE CORBA-ANY.
 01 ANY-RESULT        OBJECT REFERENCE CORBA-ANY.
 01 CALCULATE-A       TYPE CORBA-LONG.
 01 CALCULATE-B       TYPE CORBA-LONG.
 01 CALCULATE-RESULT  TYPE ODDEMO-CALCULATOR-RESULT.
 01 RET               TYPE CORBA-STATUS.
 01 API-NAME          PIC   X(50).
 01 APL-NAME          PIC   X(8)   VALUE     "SIMPLE_S".
 01 ORB               OBJECT REFERENCE CORBA-ORB.
 01 TMP_TC            OBJECT REFERENCE CORBA-TYPECODE.
 01 TMP-TCK           TYPE CORBA-TCKIND.
 01 FLAG              TYPE CORBA-FLAGS.
*
 LINKAGE SECTION.
 01 L-SVREQ           OBJECT REFERENCE CORBA-SERVERREQUEST.
*
 PROCEDURE DIVISION  USING    L-SVREQ
                     RAISING  CORBA-SystemException.
*  メソッド名の取得
       INVOKE L-SVREQ "OPERATION" RETURNING L-OP-NAME.
       INVOKE L-OP-NAME "GET-VALUE" RETURNING L-BUF.
* (1) メソッド名の解析
       IF NOT L-BUF = "calculate"
           INVOKE CORBA-BAD_OPERATION "NEW" RETURNING EXCEP
           EXIT METHOD RAISING EXCEP
       END-IF.
       MOVE "CORBA::ORB_init" TO API-NAME.
       INVOKE CORBA "ORB_INIT" USING APL-NAME FJ-OM_ORBID RETURNING ORB.
* (2) パラメタリストの作成
       INVOKE ORB "CREATE_LIST" USING   2
```

```
                                      L-NVLIST.
       INVOKE CORBA-ANY "NEW" RETURNING ANY-A.
       INVOKE CORBA-ANY "NEW" RETURNING ANY-B.
       INVOKE CORBA-ANY "NEW" RETURNING ANY-RESULT.
       SET CORBA-TK_LONG OF TMP-TCK TO TRUE.
       INVOKE ORB "GET_PRIMITIVE_TC"    USING     TMP-TCK
                                        RETURNING TMP_TC.
       INVOKE ANY-A  "SET-TYPE" USING TMP_TC.
       INVOKE ANY-B  "SET-TYPE" USING TMP_TC.
       INVOKE ODDEMO-CALCULATOR-RESULT--H "TYPE" RETURNING TMP_TC.
       INVOKE ANY-RESULT   "SET-TYPE" USING TMP_TC.
       MOVE "A" TO L-BUF.
       SET CORBA-ARG_IN OF FLAG TO TRUE.
       INVOKE L-NVLIST "ADD_VALUE"  USING      L-BUF
                                               ANY-A
                                               FLAG
                                    RETURNING  NV-A.
       MOVE "B" TO L-BUF
       INVOKE L-NVLIST "ADD_VALUE"  USING      L-BUF
                                               ANY-B
                                               FLAG
                                    RETURNING  NV-B.
* (3) パラメタの解析
       INVOKE L-SVREQ "ARGUMENTS" USING L-NVLIST.
       INVOKE ANY-A "EXTRACT_LONG" RETURNING CALCULATE-A.
       INVOKE ANY-B "EXTRACT_LONG" RETURNING CALCULATE-B.
* 処理部の起動
       INVOKE SELF "CALCULATE"  USING      CALCULATE-A
                                           CALCULATE-B
                                RETURNING  CALCULATE-RESULT.
* (4) 復帰情報の起動
       INVOKE ODDEMO-CALCULATOR-RESULT--H "INSERT" USING ANY-RESULT
                                                         CALCULATE-RESULT.
       INVOKE L-SVREQ "SET_RESULT" USING ANY-RESULT.
*
 END METHOD  INVOKE-METHOD.

*** 処理部

 METHOD-ID. CALCULATE.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 CALCULATE-ZEROPARAM USAGE OBJECT REFERENCE ODDEMO-CALCULATOR-ZEROPARAM.
 LINKAGE SECTION.
 01 CALCULATE-A TYPE CORBA-LONG.
 01 CALCULATE-B TYPE CORBA-LONG.
 01 CALCULATE-RESULT TYPE ODDEMO-CALCULATOR-RESULT.
*
 PROCEDURE DIVISION
          USING      CALCULATE-A
                     CALCULATE-B
          RETURNING  CALCULATE-RESULT
          RAISING    CORBA-SystemException
                     ODDEMO-CALCULATOR-ZEROPARAM.
*
      IF  CALCULATE-B  = 0
          INVOKE  ODDEMO-CALCULATOR-ZEROPARAM "NEW" RETURNING CALCULATE-ZEROPARAM
          EXIT METHOD RAISING CALCULATE-ZEROPARAM
      END-IF.
*
      COMPUTE ADD_RESULT      OF CALCULATE-RESULT = CALCULATE-A + CALCULATE-B.
      COMPUTE SUBTRACT_RESULT OF CALCULATE-RESULT = CALCULATE-A - CALCULATE-B.
      COMPUTE MULTIPLE_RESULT OF CALCULATE-RESULT = CALCULATE-A * CALCULATE-B.
```

```
      COMPUTE DIVIDE_RESULT   OF CALCULATE-RESULT = CALCULATE-A / CALCULATE-B.
 END METHOD CALCULATE.
*
 END OBJECT.
*
 END CLASS ODDEMO-CALCULATOR-IMPL.
```

(1)メソッドの取得と解析

渡されたServerRequestクラスに登録されているオペレーション名をCORBA-SERVERREQUEST-OPERATIONメソッドにより取得します。

(2)パラメタの組立て

CORBA-ORB-CREATE_LISTメソッドにより、アプリケーションに渡すパラメタを格納するための領域を保持するリストオブジェクトを生成します。このとき、引数として何個のパラメタを格納するかを指定します。結果としてCORBA-NVLISTオブジェクトが返されます。

(3)パラメタの解析

生成したNVLISTオブジェクトを引数としてCORBA-SERVERREQUEST-ARGUMENTSメソッドを呼び出します。この結果、アプリケーションに渡すパラメタ値が取得されます。

(4)復帰情報の設定

CORBA-SERVERREQUEST-SET_RESULTメソッドを発行して復帰情報を設定します。

ゲートウェイを実装するIMPLオブジェクトは、DSIBASEを継承するものとして作成します。また、クライアントからのリクエストに対して起動されるメソッドとしてCORBA-SERVERREQUEST-INVOKE-METHODメソッドを実装します。起動の際、CORBA-SERVERREQUEST-INVOKE-METHODの引数としてServerRequestクラスのオブジェクト（上記例ではANY-RESULT）が渡されます。

(5)例外情報の設定

CORBA-SERVERREQUEST-SET_EXCEPTIONメソッドを発行して例外情報を設定します。
サーバアプリケーションのメソッドが発生するユーザ例外を設定し、クライアントへ通知します。メソッドが発生するユーザ例外は、IDLのオペレーション定義で記述されます。

```
 REPOSITORY.
      CLASS CORBA-ANY
      CLASS ODDEMO-CALCULATOR-ZEROPARAM-HELPER
      CLASS ODDEMO-CALCULATOR-ZEROPARAM

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 DSI-REQUEST   USAGE OBJECT REFERENCE CORBA-SEQRVERREQUEST.
 01 EX            USAGE OBJECT REFERENCE CORBA-ANY.
 01 EX-TYPE       USAGE OBJECT REFERENCE CORBA-TYPECODE.
 PROCEDURE DIVISION.

    INVOKE ODDEMO-CALCULATOR-ZEROPARAM--HELPER "TYPE" RETURNING EX-TYPE.
    INVOKE CORBA-ANY "NEW" RETURNING EX.
    INVOKE EX "SET-TYPE" USING EX-TYPE.

    INVOKE DSI-REQUEST "SET_EXCEPTION" USING EX.
```

【ゲートウェイIMPLクラスの必須メソッド】

```
METHOD-ID. INVOKE-METHOD OVERRIDE.
DATA DIVISION.
LINKAGE SECTION.
01 L-SVREQ USAGE OBJECT REFERENCE CORBA-SERVERREQUEST.
PROCEDURE DIVISION  USING    L-SVREQ
                    RAISING  CORBA-SystemException.
```

索　引

[記号]

[ ]演算子.....227,230,239

[A]

AdapterActivatorオブジェクト.....94,108,303,316
allocbuf関数.....202
any.....52,71,186,222,383,462,513,819
Any*コンストラクタ.....232
Any*代入演算子.....233
Any_varクラス.....231
AOM.....101,117,310,324
attribute.....68,218,416,505,895

[C]

char*コンストラクタ.....225
char *代入演算子.....226
COBOL登録集.....531
COBOLとの整合性.....512
const.....419
Contextインタフェース.....685
Contextオブジェクト.....685
corbaloc URLスキーマ.....592,693
corbaname URLスキーマ.....592
CORBAアプリケーションの開発手順.....10
CORBAサービスのインクルードファイル.....725
CORBAサービスのライブラリ.....727

[D]

Default Servantオブジェクト.....93,302

[E]

enum.....378

[F]

Factory.....84,294,645
Factory-1方式.....83,87,292,297
Factory-2方式.....83,88,293,297
fixed.....491,891
freebuf関数.....202

[H]

helper関数.....194

[I]

IDL.....696
implクラス方式.....77
InterfaceDef.....548,554,573
IOR URLスキーマ.....591

[J]

Javaアプレット.....240,244

[L]

lengthアクセス関数.....202

[M]

maximumアクセス関数.....202
maximumコンストラクタ.....200

[N]

noinclude.....656
noincludeオプション.....656
NVListインタフェース.....684
NVListオブジェクト.....684
NVList使用方法.....514

[O]

odexportir.....732
odimportir.....733
OD_impl_inst.....39,172,251,435,854
OD_or_adm.....40,117,173,325,438,856
OLEクライアント.....815
OOCOBOL登録集.....909
OperationDef.....549,555,574

[P]

POA.....91,300
POA ID.....94,303
POAManager.....118,325
POAManagerオブジェクト.....93,103,302,312
POAオブジェクト.....93,94,302,303
POAポリシ.....92,94,301,303
Portable-ORB.....4
Portable-ORBの動作環境ファイル.....261
Portable-ORBのライブラリ.....731
private領域.....647

[R]

replace関数.....194
RootPOAオブジェクト.....96,305

[S]

sequence.....55,71,195,222,389,468,513,820,868
ServantActivatorオブジェクト.....104,313
ServantLocatorオブジェクト.....106,314
ServantManagerオブジェクト.....93,302
Servantオブジェクト.....93,302
string.....47,182,372,452,862
String.....818
String_varクラス.....224
struct.....58,204,394,821,877

[T]

T*dataコンストラクタ.....200
T*コンストラクタ.....235
T*代入演算子.....237
TIEクラス方式.....77
try－catch.....179
TypeCodeインタフェース.....682
TypeCodeオブジェクト.....681
TypeCode関連.....538
type関数.....194

[U]

union.....61,207,401,484,821,883
URLスキーマ.....591

[V]
value関数.....195
varクラス.....223

[W]
WChar*コンストラクタ.....228
WChar *代入演算子.....229
wstring.....49,184,375,457,865
WString_varクラス.....227

[あ]
アプレットのデジタル署名.....266
インスタンス管理.....91,300
インスタンス管理とアプリケーション形態.....82,291
インタフェース情報.....629
インタフェース宣言.....703
インタフェース定義情報の移出.....732
インタフェース定義情報の移入.....733
インタフェース定義情報の登録.....732
インタフェースリポジトリサービス.....618
インタフェースリポジトリ用宣言.....537
エラー例外用宣言.....534
オブジェクトID.....94,303
オブジェクトの検索.....593
オブジェクトの登録.....600
オブジェクトの呼び出し.....658
オブジェクトリファレンス.....716
オペレーション宣言.....706

[か]
環境設定.....674
基本データ型.....47,181,372,711,817
共用体.....714
キーストア.....271
クライアントアプリケーションの例外処理.....44,178,284,446,846
継承方式.....120,328
交換演算子.....234,238
構造体.....713
構造体型のバインディング名.....584
固定小数点型.....491,891
固定小数点型.....715
コピーコンストラクタ.....191,200,211,225,229,232,236
コンストラクタ.....191

[さ]
サンプルプログラムについて.....734
サーバアプリケーションの起動タイプ.....7
システム例外.....44,180,284,446,824,846
事前生成.....110,319
子孫POAオブジェクト.....97,306
実行手順（Solaris/Linux）.....767
実行手順（Windows(R)）.....741
終了処理.....651
使用時の注意事項.....740
シーケンス型.....712
スレッド生成.....672
スレッドモード.....671
静的起動インタフェース.....9,43,176,281,442,837
静的スケルトンインタフェース.....10,248,322
静的スケルトンインタフェース.....113
属性.....707,823
属性宣言.....707

[た]
代入演算子.....192,201,212,226,230,233,237
タイプコード.....716
代理方式.....120,328
多重制御.....8
定数型宣言.....533
定数宣言.....708
デジタル署名手順.....267
デストラクタ.....192,201,212,225,229,232,236
デフォルトインスタンス方式.....82,86,292,296
デフォルトコンストラクタ.....191,199,211,225,228,231,235
添字演算子.....202
データ型宣言.....533
同期送信.....551,557,570,578
動的起動インタフェース.....9,552,558
動的スケルトンインタフェース.....10,917,921,925,930,933
動的生成.....110,318

[な]
ネーミングコンテキスト.....580
ネーミングサービス.....554,580
ネーミングサービス用宣言.....536
ネームコンポーネント.....584

[は]
配列.....65,409,497,717,822
バインディングリスト.....585
パラメタ受渡し.....70,220,420,509,901
左シフト代入演算子.....193,203
非同期送信.....551,557,570,578
プラグマ.....701
フルスコープ.....699
プレインストール型.....4
プロセス生成.....672
プロセスモード.....671
プロパティ権限.....278
弁別子アクセス関数.....213
ポインタ演算子.....233,238
包含関係.....618

[ま]
右シフト代入演算子.....193,203
メンバアクセス関数.....213
モジュール宣言.....702

[や]
ユーザインスタンス管理方式.....90,299
ユーザインスタンス管理方式.....84,293
ユーザ定義型データ.....234
ユーザ例外.....37,44,180,284,432,447,824,846
要素名.....714
予約語の扱い.....514

[ら]

ライブラリ化.....667
ランタイム権限.....278
リストの取得.....608
リリースフラグ.....71,222,513
ルートリポジトリ.....629
例外情報.....853
例外宣言.....717
列挙型.....819
ロードバランスオブジェクトグループ.....827
ロードバランスオプション.....827
ロードバランスオプション用宣言.....537