FUJITSU

FUJITSU Software
Interstage Application Server V12.0.0

チューニングガイド

Windows/Solaris(64)/Linux

B1WS-1303-01Z0(00)
2017年7月

まえがき

本書の目的

本書は、運用形態を変更する場合やシステム規模を変更する場合などに必要な環境設定のチューニングについて説明しています。
本書は、Interstage Application Serverの運用を行う方を対象に記述されています。

前提知識

本書を読む場合、以下の知識が必要です。

- 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
- 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
- 使用するOS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10章 JDK/JRE 8のチューニング

JDK/JRE 8 のチューニングに関して、基本的な知識、Java VM、異常発生時の原因振り分け方法およびチューニング方法について説明します。

第11章 データベース連携サービスのチューニング Windows32/64

データベース連携サービスのiniファイルについて説明します。

付録A CORBAサービスの動作環境ファイル

CORBAサービスの環境定義について説明しています。

付録B コンポーネントトランザクションサービスの環境定義

コンポーネントトランザクションサービスの環境定義について説明しています。

付録C データベース連携サービスの環境定義

データベース連携サービスの環境定義について説明しています。

付録D イベントサービスの環境定義

イベントサービスの環境定義について説明しています。

付録E Interstage HTTP Serverの環境定義

Interstage HTTP Serverの環境定義について説明します。

付録F Interstage HTTP Server 2.2の環境定義

Interstage HTTP Serverの環境定義について説明します。

付録G Interstage シングル・サインオンの環境定義

Interstage シングル・サインオンを運用するための、環境定義のチューニングについて説明します。

付録H Portable-ORBの環境設定

Portable-ORBの環境設定について説明します。

付録I RHEL7のunitファイルでの環境定義 Linux32/64

RHEL7のunitファイルでの環境定義について説明します。

製品名称の略称について

本書では、以下の製品名称を略称で表記している箇所があります。

略称	製品名称
EE	Interstage Application Server Enterprise Edition
SJE	Interstage Application Server Standard-J Edition

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。

登録商標について

記載されている会社名、製品名などの固有名詞は、各社の商標または登録商標です。

本製品のマニュアルに記載されている他社製品の商標表示については、「マニュアル体系と読み方」の「マニュアルの読み方」－「登録商標について」を参照してください。

著作権

2017年7月 初版

目　次

第1章 必要資源

注意

- Interstage HTTP Server 2.2の運用時に必要な資源については、「Interstage HTTP Server 2.2 運用ガイド」の「チューニング」を参照してください。
- EE Java EE 6機能の運用時に必要な資源については、「Java EE運用ガイド(Java EE 6編)」の「Java EE 6機能のチューニング」－「必要資源」を参照してください。
- Java EE 7機能の運用時に必要な資源については、「Java EE 7 設計・構築・運用ガイド」の「Java EE 7機能のチューニング」－「必要資源」を参照してください。

1.1 運用時に必要なディスク容量

運用時に必要なディスク容量は次のとおりです。

1.1.1 サーバ機能を使用する場合

Interstageの各機能の運用時に必要となるディスク容量について説明します。
以下の表を参照し、使用する製品に応じて、各機能のディスク容量を見積もってください。表内で使用している略称については、「製品名称の略称について」を参照してください。

	EE	SJE
Interstage動作環境	○	○
Interstage管理コンソール	○	○
業務構成管理	○	○
Interstage HTTP Server	○	○
Interstage JMXサービス	○	○
Interstage シングル・サインオン	○	○
Interstage ディレクトリサービス	○	○
J2EE	○	×
IJServerワークユニット	○	×
Interstage JMS	○	×
Servletサービス(OperationManagement)	○	○
ワークユニット	○	○
CORBAサービス	○	○
イベントサービス	○	×
Portable-ORB	○	×
コンポーネントトランザクションサービス	○	○
データベース連携サービス	○	×
ロードバランス Windows32 Linux32	○	×
性能監視ツール	○	○
MessageQueueDirector Windows32/64 Linux32/64	○	×

○:該当製品で提供される機能です。
×:該当製品で提供されない機能です。

注)
Interstage シングル・サインオンの業務サーバだけでディスク容量を見積もります。

機能: Interstage動作環境

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
コンポーネントトランザクションサービスのインストールディレクトリ¥var¥td001 (Interstage動作環境定義ファイルの“TD path for system”で指定)	60 以上	Interstage動作環境作成時

機能: Interstage管理コンソール

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 Interstage管理コンソールのインストールディレクトリ¥isAdmin¥var¥download Solaris64 Linux32/64 /var/opt/FJSVisgui/tmp/download	(注)	ログ情報

注)
Interstage管理コンソールの以下の画面において、ログファイルをダウンロードする場合、同時にダウンロードするログファイルのサイズ分のディスク容量が一時的に必要となります。

機能	画面
Interstage HTTP Server	[システム] > [サービス] > [Webサーバ] > [Webサーバ名] > [ログ参照]タブ
	[システム] > [サービス] > [Webサーバ] > [Webサーバ名] > [バーチャルホスト] > [バーチャルホスト名] > [ログ参照]タブ
EE Webサーバコネクタ	[システム] > [サービス] > [Webサーバ] > [Webサーバ名] > [Webサーバコネクタ] > [ログ参照]タブ
EE IJServerワークユニット	[システム] > [ワークユニット] > [ワークユニット名] > [ログ参照]タブ

ログファイルのサイズについては、各機能のログ情報のディスク容量を参照し、運用の内容により必要とするサイズを検討してください。
なお、ログファイルのサイズが大きいため、ディスク容量の不足によりログファイルのダウンロードに失敗する場合は、FTPなどを使用してダウンロードしてください。

機能: 業務構成管理

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
業務構成管理のリポジトリ Windows32/64 Interstage JMXサービスのインストールディレクトリ¥var¥repository Solaris64 Linux32/64 /var/opt/FJSVisas/repository	運用の内容により、必要とするサイズを検討してください。(注)	デフォルトから変更した場合は、変更先

注)
業務構成管理のリポジトリの格納先のサイズは、“Interstage Application Server 運用ガイド(基本編)”の“業務構成管理機能”を参照してください。

機能: Interstage HTTP Server

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
アクセスログ、エラーログ、トレースログ格納ディレクトリ	運用の内容により、必要とするサイズを検討してください。	アクセスログ、エラーログ、トレースログ
Windows32/64 Interstage HTTP Serverのインストールディレクトリ¥var¥opelog Solaris64 Linux32/64 /var/opt/FJSVihs/var/opelog	2	オペレーションログ
Windows32/64 Interstage HTTP Serverのインストールディレクトリ¥var¥.ihsapi Solaris64 Linux32/64 /var/opt/FJSVihs/var/.ihsapi	10	保守ログ
コンテンツ格納ディレクトリ	運用の内容により、必要とするサイズを検討してください。	コンテンツ(HTML文書など)

機能: Interstage JMXサービス

Windows32/64

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Interstage JMXサービスのインストールディレクトリ	14以上 (注)	

Solaris64 Linux32/64

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
/var/opt Linux32	14 以上 (注)	
/var/opt Solaris64 Linux64	32 以上 (注)	
/etc/opt Solaris64 Linux32/64	0.1 以上	

注)
Interstage JMXサービスのカスタマイズでログインログのファイルサイズの上限値を変更している場合、以下のディスク所要量が必要となります。

ー ログインログ
ログインログのファイルサイズの上限値 × 2 (Mバイト)

上限値を変更していない場合、ログインログのファイルサイズの上限値は1に設定されています。

機能: Interstage シングル・サインオン

Interstage シングル・サインオンの業務サーバ機能

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 Interstage シングル・サインオンのインストールディレクトリ¥ssoatzag¥log Solaris64 Linux32/64 /var/opt/FJSVssoaz/log	運用の内容により、必要とするサイズを検討してください。(注1)	アクセスログなどのログ情報 (アクセスログファイルの出力先ディレクトリ)
Windows32/64 Interstage シングル・サインオンのインストールディレクトリ¥ssocm¥etc Solaris64 Linux32/64 /var/opt/FJSVssocm/etc	2	
Windows32/64 Interstage シングル・サインオンのインストールディレクトリ Solaris64 Linux32/64 /etc/opt	運用の内容により、必要とするサイズを検討してください。	アクセス制御情報

Interstage シングル・サインオンの認証サーバ機能

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 Interstage シングル・サインオンのインストールディレクトリ¥ssoatcag¥log Solaris64 Linux32/64 /var/opt/FJSVssoac/log	運用の内容により、必要とするサイズを検討してください。(注1)	アクセスログなどのログ情報 (アクセスログファイルの出力先ディレクトリ)
Windows32/64 Interstage シングル・サインオンのインストールディレクトリ¥ssofsv¥log Solaris64 Linux32/64 /var/opt/FJSVssofs/log	運用の内容により、必要とするサイズを検討してください。(注1)(注2)	
Windows32/64 Interstage シングル・サインオンのインストールディレクトリ¥ssocm¥etc Solaris64 Linux32/64 /var/opt/FJSVssocm/etc	2 (注3)(注4)	

Interstage シングル・サインオンのリポジトリサーバ機能

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 Interstage シングル・サインオンのインストールディレクトリ¥ssoatcsv¥log Solaris64 Linux32/64 /var/opt/FJSVssosv/log	運用の内容により、必要とするサイズを検討してください。(注1)(注5)	アクセスログなどのログ情報 (アクセスログファイル、およびセション管理ログファイルの出力先ディレクトリ)
Windows32/64 Interstage シングル・サインオンのインストールディレクトリ¥ssocm¥etc Solaris64 Linux32/64 /var/opt/FJSVssocm/etc	2	

注1)
デフォルト設定のままでは使用ディスクサイズの上限なしにログが採取されます。ディスク不足発生を防止するために、定期的に不要になったログファイルを削除するか、ログの採取方法を変更してください。

注2)
認証サーバ間連携を行わない場合は、0Mバイトです。

注3)
統合Windows認証を行う場合には、2Mバイトを加算してください。

注4)
認証サーバ間連携を行う場合には、2Mバイトを加算してください。

注5)
セション管理を行うリポジトリサーバをクラスタシステム上で運用する場合には、52Mバイトを加算してください。

機能: Interstage ディレクトリサービス

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 Interstage ディレクトリサービスのインストールディレクトリ¥var Solaris64 Linux32/64 /var/opt	20 × リポジトリ作成数 ＋ 20	ログ情報
Windows32/64 Interstage ディレクトリサービスのインストールディレクトリ¥etc Solaris64 Linux32/64 /etc/opt	0.5 × リポジトリ作成数	環境定義
Interstage ディレクトリサービスのアクセスログ作成ディレクトリ	Interstage管理コンソールのアクセスログの設定値に依存 サイズ × 世代管理数	アクセスログ
Windows32/64 Interstage ディレクトリサービス SDKのインストールディレクトリ¥var Solaris64 Linux32/64 /var/opt/FJSVirepc	Windows32/64 プロセス数 × 8 Solaris64 Linux32/64 同時接続数 × 8 (注)	Interstage ディレクトリサービス SDKのログ情報

注)
同時接続数には、以下の必要数の総和を指定して見積もってください。

- スレッド多重のアプリケーションを使用してリポジトリにアクセスする場合の、アプリケーションのプロセス数
- プロセス多重のアプリケーションを使用してリポジトリにアクセスする場合の、アプリケーションのプロセス多重度
- Interstage シングル・サインオン機能を使用する場合の、Interstage シングル・サインオンのリポジトリサーバ機能を組み込んだWebサーバのクライアント同時接続数

機能: J2EE

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 J2EE共通ディレクトリ Solaris64 Linux32/64 /var/opt/FJSVj2ee/deployment	運用の内容により、必要とするサイズを検討してください。	J2EEアプリケーションの資産一式

機能: IJServerワークユニット

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 J2EE共通ディレクトリ¥ijserver¥IJServer名¥logディレクトリ Solaris64 Linux32/64 J2EE共通ディレクトリ/ijserver/IJServer名/logディレクトリ	24 以上 (注1)	
Windows32/64 Interstageのインストールディレクトリ¥F3FMjs5¥logs¥jk2 Solaris64 Linux32/64 /var/opt/FJSVjs5/logs/jk2	2 以上 (注2)	
Windows32/64 J2EE共通ディレクトリ¥ijserver¥Session Registry Server(IJServer)名¥apps¥srs.ear¥srs.war¥serializedata¥sessionrecovery Solaris64 Linux32/64 J2EE共通ディレクトリ/ijserver/Session Registry Server(IJServer)名/apps / srs.ear/srs.war/serializedata/sessionrecovery	(注3)	Servletサービスのセッションリカバリ機能使用時 セッションの永続化有効時、Session Registry Serverを運用する環境で必要です。

注1)
IJServerワークユニット1つにつき以下を加算してください。

プロセス多重度 ×
4(コンテナログとコンテナ情報ログのデフォルトディスク使用量) ×
6(世代分のバックアップ) 以上

アプリケーションのタイムアウトが多発する場合、アプリケーションで短時間に大量のメッセージを出力する場合、およびデバッグ情報出力を行う場合は、“J2EE共通ディレクトリ/ijserver/IJServer名/log”配下のコンテナ情報ログのディスク使用量が大きくなります。このような操作が想定される場合は、十分なディスク容量をご用意ください。

注2)
Webサーバ1つにつきデフォルトで2Mバイトです。アプリケーションで短時間に大量のメッセージを出力する場合、デバッグ情報出力を行う場合は、ディスク使用量が大きくなります。このような操作が想定される場合は、十分なディスク容量をご用意ください。

注3)
セッションリカバリ機能を使用して、セッションの永続化を有効にした場合は、Session Registry Server環境定義ファイルで指定したセッションの永続化ファイルの保存先にセッションの永続化ファイルが生成されます。配備するWebアプリケーション1つについて、次のディスク容量が必要です。(単位:Mバイト)

Windows32/64
(0.005 ＋ (0.005 ＋ セッションの保持するデータ容量) × セッション数) × 2
Solaris64
(0.001 ＋ (0.002 ＋ セッションの保持するデータ容量) × セッション数) × 2
Linux32/64
(0.008 ＋ (0.008 ＋ セッションの保持するデータ容量) × セッション数) × 2

“セッションの保持するデータ容量”は、Webアプリケーションでセッションの属性(Attribute)にセットするオブジェクトおよびキーのサイズの合計値です。
上記の値は、利用しているファイルシステムによっては値が増減する場合があります。
なお、Session Registry Serverは、Interstage Application Server Enterprise Editionで運用可能です。

機能: Interstage JMS

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 Interstage JMSのインストールディレクトリ¥etc Solaris64 Linux32/64 /etc/opt	0.01 ＋ (durable Subscriber数 × 0.002)	定義情報
Windows32/64 Interstage JMSのインストールディレクトリ¥var Solaris64 Linux32/64 /var/opt	10 以上	コンソールファイル

機能: Servletサービス(OperationManagement)

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 Servletサービス(OperationManagement)インストールディレクトリ¥log Solaris64 Linux32/64 /var/opt/FJSVjs2su/log	12	ログ情報

機能: ワークユニット

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Interstage動作環境定義の定義項目“TD path for system”で指定	1つのワークユニット定義サイズ × ワークユニット定義数 (注)	ワークユニット定義登録時
	1つのワークユニット定義サイズ × ワークユニット起動数 (注)	ワークユニット運用時

注)
1つのワークユニット定義サイズ ＝

1000 ＋
(500 ×“[Application Program]セクション定義数”) ＋
(500 ×“[Resource Manager]セクション定義数”) ＋
(500 ×“[Nonresident Application Process]セクション定義数”) ＋
(500 ×“[Multiresident Application Process]セクション定義数”) ＋
ユーザ任意指定文字列データ長

機能: CORBAサービス

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 CORBAサービスのインストールディレクトリ¥etcフォルダ Solaris64 Linux32/64 /etc/opt	0.1 以上	インプリメンテーション情報、ネーミングサービス、インタフェースリポジトリのデータサイズに依存します。
	4.1 以上 (注1)	ネーミングサービス情報

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 CORBAサービスのインストールディレクトリ¥varフォルダ Solaris64 Linux32/64 /var/opt	8 以上	
	Windows32/64 42.0 Solaris64 Linux32/64 24.0 (デフォルト時の最大サイズ) (注2)	ログ情報
	2.0 以上 (注3)	内部ログ採取時(プレインストール型Javaライブラリ以外の場合)
	4.0 以下 (注1)	ネーミングサービスのユーザ例外ログ情報
	4.0 以下 (注1)	ネーミングサービスの実行トレース情報(サービス動作時のみ)
	32.3 以下	インタフェースリポジトリサービスのログ情報 (サービス動作時のみ)
Windows32/64 CORBAサービスのインストールディレクトリ¥var¥traceフォルダ Solaris64 /var/opt/FSUNod/trace Linux32/64 /var/opt/FJSVod/trace	10.4 (注4)	トレース情報
Windows32/64 コンポーネントトランザクションサービスインストールディレクトリ¥var¥IRDB Solaris64 Linux32/64 コンポーネントトランザクションサービスインストールディレクトリ/var/IRDB (Interstage統合コマンド使用時のデフォルト)	10.3 以上 (注5)	インタフェースリポジトリサービス情報
Solaris64 Linux32/64 /tmp	1.0 以上 IDL定義の量に依存 C/C++コンパイラ動作時には、別途作業用のディスク容量が必要	IDLコンパイラ動作時
Java VMのシステムプロパティのuser.dirで指定	(注6)	内部ログ採取時(プレインストール型Javaライブラリの場合)
環境変数OD_HTTPGW_HOMEまたはOD_HOMEで指定されたvarディレクトリ	2.0 以上 (注7)	HTTP-IIOPゲートウェイの内部ログ採取時

注1)
CORBAサービスのサーバマシンにネーミングサービスを構築する場合に、必要となるディスク容量について以下に示します。

ー ディスク容量

用途		容量
ネーミングサービス情報	オブジェクトリポジトリ	(固定)16Kバイト
	制御ファイル	(固定)2056Kバイト
	データファイル	(可変)2048(Kバイト) × コンテキスト数 ＋ (オブジェクトリファレンス長 × オブジェクト数 × 2)
実行トレース情報		(最大)4096Kバイト
ユーザ例外ログ情報		(最大)4096Kバイト

注2)
CORBAサービスのログ採取機能を使用している場合、最大で以下のディスク容量を使用します。(各パラメタはconfigファイルで定義)

access_log_size × 2 ＋ error_log_size × 2 ＋ process_log_size × 2 ＋ info_log_size × 2

Windowsでは、上記に加えてさらに以下のディスク容量を使用します。

error_log_size × 2 ＋ process_log_size × 2 ＋ info_log_size × 2

ログ採取機能については“トラブルシューティング集”の“CORBAサービスのログ情報の採取”を、上記パラメタについては“A.1 config”(CORBAサービス)を参照してください。

注3)
以下のディスク所要量(単位:バイト)が必要です。

Windows32/64
(max_processes(*) ＋ 2) × log_file_size(*) × 2
*: CORBAサービスのインストールフォルダ¥etc¥configファイルのパラメタ
Solaris64 Linux32/64
log_file_size(*)×2
*: configファイルで定義

なお、ログファイルは、不要になった時点で、削除してください。

Windows32/64
採取されるログファイルはlog、log.old以外にサーバアプリケーションごとに“appNNNN.log”、“appNNNN.old”(NNNNは英数字)の名前で採取されます。
自ホストでネーミングサービス、インタフェースリポジトリを動作させる場合には、それぞれ、4Mバイト、32Mバイトの領域が必要です。

注4)
CORBAサービスのトレース情報の採取機能を使用している場合、最大で以下のディスク容量を使用します。(各パラメタはconfigファイルで定義)

trace_size_per_process× max_processes × 2 ＋ trace_size_of_daemon × 2

トレース情報の採取機能については“トラブルシューティング集”の“CORBAサービスのトレース情報の採取”を、上記パラメタについては“A.1 config”(CORBAサービス)を参照してください。

注5)
インタフェースリポジトリを使用する場合のディスク容量について以下に示します。インタフェースリポジトリのデータベースのサイズは、以下の計算式に従って見積もり、ディスクを確保してください。
なお、インタフェースリポジトリのデータベースは、初期値(10240Kバイト)から自動拡張します。

ー ディスク容量

用途		容量
インタフェースリポジトリ	管理域	(固定)220Kバイト
	利用者定義領域	(初期値:可変)10240Kバイト

用途		容量
サービス情報		
実行トレース情報		(最大)33000Kバイト

ー 見積り式

利用者定義領域(オブジェクトに要するディスク容量)の見積り式を以下に示します。

項番	IDL定義	計算式(単位:バイト)
1	モジュール宣言	1708 ＋ ((a－1)÷32＋1)×176
2	インタフェース宣言	1712 ＋ ((a－1)÷32＋1)×176 ＋ ((b－1)÷32＋1)×176 ＋ 512×b
3	オペレーション宣言	2304 ＋ ((e－1)/32＋1)×176 ＋ ((f－1)÷32＋1)×176 ＋ ((g－1)÷32＋1)×176
4	属性宣言	2224
5	定数宣言	2160 ＋ c
6	例外宣言	1712 ＋ ((d－1)÷32＋1)×176 ＋ 836×d
7	データ型宣言	2220
8	文字列型宣言(ワイド文字列を含む)	1716
9	列挙型宣言	1824 ＋ ((j－1)÷32＋1)×176 ＋ 64×j
10	シーケンス型宣言	2228
11	構造体宣言	1712 ＋ ((h－1)÷32＋1)×176 ＋ 836×h
12	共用体宣言	2436 ＋ ((i－1)÷32＋1)×176 ＋ 972×i
13	固定小数点型宣言	1716
14	配列宣言	2228

記号	項目	意味
a	包含数	包含する型宣言数
b	継承数	インタフェース宣言が継承するインタフェース数
c	定数値長	定数宣言の値の長さ
d	例外構造体メンバ数	例外宣言の構造体のメンバ数
e	パラメタ数	オペレーション宣言でのパラメタ数
f	コンテキスト数	オペレーション宣言でのコンテキスト数
g	例外数	オペレーション宣言での例外数
h	構造体メンバ数	構造体宣言でのメンバ数
i	共用体メンバ数	共用体宣言でのメンバ数
j	列挙型メンバ数	列挙型宣言でのメンバ数

注6)
ログファイルのサイズの上限値は、CORBAサービスのconfigファイルのlog_file_sizeで設定することができます。アプリケーションごとにJVxxxxxxxxxx.log/JVxxxxxxxxxx.old(xxxxxxxxxxは一意の数字)の名前で採取されます。なお、ログファイルは、不要になった時点で、削除してください。

注7)
ログファイルのサイズの上限値は、HTTPトンネリングの“gwconfigファイル”の“max_log_file_size”で設定することができます。ディスク容量は、バックアップファイルを1つ残すため“max_log_file_size”で指定した値×2”となります。また、SolarisまたはLinuxで、WebサーバにInterstage HTTP Serverを使用している場合は、Interstage HTTP Serverの通信プロセスごとにログファイルが作成されます。なお、ログファイルは、不要になった時点で、削除してください。

機能: イベントサービス

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 イベントサービスのインストールディレクトリ¥etc Solaris64 Linux32/64 /etc/opt	Windows32/64 0.1 以上 Solaris64 Linux32/64 1.0 以上	チャネル情報
Windows32/64 イベントサービスのインストールディレクトリ¥var Solaris64 Linux32/64 /var/opt	61(Mバイト) ＋ essetcnfコマンドの-s -logsizeオプションの指定値×2(Kバイト)以上	ログ情報
	Windows32/64 Solaris64 トレースファイル(traceconfig)のバッファサイズ(trace_size) × 最大数(trace_file_number) Linux32/64 トレースの採取方法によって異なります。(注1)	トレース情報
Interstage管理コンソールで保存先(新規作成)の格納ディレクトリで指定、またはイベントサービスのユニット定義ファイルの“trandir”、“sysdir”、“userdir”で指定	Windows32/64 38 × イベントサービスで作成したユニット数以上 (注2) Solaris64 Linux32/64 運用の内容により、必要とするサイズを検討してください。(注2)	不揮発チャネル運用時

注1)

－ プロセス単位で内部トレースを採取する(traceconfigのtrace_buffer = process)場合

traceconfigファイルのtrace_size × イベントチャネルのプロセス数(*1) × トレースファイルの世代数

*1)イベントチャネルのプロセス数 ＝ 静的イベントチャネルグループ数 ＋ 動的イベントチャネルのプロセス数(*2)
*2)isinitコマンドでInterstage初期化時に設定したInterstage動作環境定義の定義「Event maximum Process」の指定値。イベントサービスとノーティフィケーションサービスの両方を使用している場合は、「動的イベントチャネルのプロセス数×2」としてください。

― イベントサービス単位で内部トレースを採取する(traceconfigのtrace_buffer = system)場合

トレースファイル(traceconfig)のバッファサイズ(trace_size) × 最大数(trace_file_number)

注2)
Interstage管理コンソールで設定した場合は、保存先(新規作成)の格納ディレクトリには、以下の容量が必要となります。

― イベントデータ用ファイル容量

― システム用ファイル容量

― トランザクション用ファイル容量:
((トランザクション多重度 × 4) + 256 + (1トランザクション内最大メッセージサイズ × 2)) × 16 (Kバイト)

ユニット定義ファイルで設定した場合は、各ユニット定義ファイルで指定した以下の容量が必要となります。

― 「sysdir」で指定したディレクトリ:「syssize」で指定したサイズ

― 「userdir」で指定したディレクトリ:「usersize」で指定したサイズ

― 「trandir」で指定したディレクトリ:((tranmax×4) + 256 + (tranunitmax × 2)) × 16 (Kバイト)

機能: Portable-ORB

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 Portable-ORBインストールディレクトリ (注1) Solaris64 Linux32/64 /var/opt (注1)	(注2)	ログ情報

注1)
アプレットとして動作する場合は、アプレットが動作するクライアントマシン上のローカルディスクに、porbeditenvコマンドで“ログ格納ディレクトリ”として指定したディレクトリとなります。

注2)
porbeditenvコマンドで“ログ情報を採取”を指定した場合、
設定した“ログファイルサイズ” × 2 × 動作するアプリケーション/アプレット数

機能: コンポーネントトランザクションサービス

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32/64 コンポーネントトランザクションサービスのインストールディレクトリ¥var Solaris64 /var/opt/FSUNtd/	25以上	ログトレースファイル
EE Solaris64 /opt/FSUNtd/etc/isreg(Interstageの動作環境ディレクトリ) Linux32/64 /opt/FJSVtd/etc/isreg(Interstageの動作環境ディレクトリ)	15.0 以上	動作環境
Solaris64 Linux32/64 /var	運用の内容により、必要とするサイズを検討してください。(注)	異常終了した場合に採取されるcoreファイル

注)
ディスク所要量の算出方法は、以下のとおりです。

Linux32

CORBAサービス関連の共有メモリサイズ(*1) × 3 ＋ コンポーネントトランザクションサービスの共有メモリサイズ(*2) ＋ ワークユニット数 × 0.26 ＋ ワークユニットに含まれるIDL定義のパラメタ数(*3) × 0.00005 ＋ 基本サイズ(*4)

Solaris64 Linux64

CORBAサービス関連の共有メモリサイズ(*1) × 3 ＋ ワークユニット数 × 0.26 ＋ 基本サイズ(*4)

*1)CORBAサービス関連の共有メモリサイズは、CORBAサービスのconfigファイル(/opt/FJSVod/config)の各パラメタから以下のように算出します。

```
limit_of_max_IIOP_resp_con × 0.016 ＋ limit_of_max_IIOP_resp_requests × 0.016 ＋
max_impl_rep_entries × 0.006 ＋ 0.01
```

*2)コンポーネントトランザクションサービスの共有メモリサイズは、以下のように算出します。

クライアント数 × 0.1 ＋ 100

クライアント数は、環境定義ファイル(/var/opt/FJSVtd/etc/sysdef)のSystem Scale: ステートメントに指定した値に応じて、以下のように見積もってください。

- small:50
- moderate:100
- large:500
- super:1000

*3)ワークユニットに含まれるIDL定義のパラメタ数は、ワークユニット数やIDL定義内のパラメタ数が多い場合に加算してください。各オペレーションのパラメタに構造体がある場合は、パラメタごとに構造体のメンバ数を加算してください。

*4)基本サイズは、環境定義ファイル(/var/opt/FJSVtd/etc/sysdef)のSystem Scale: ステートメントに指定した値に応じて、以下のように見積もってください。

Linux32

- small:250
- moderate:330
- large:840
- super:1400

Solaris64 Linux64

- small:270
- moderate:350
- large:860
- super:1420

機能: データベース連携サービス

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Solaris64 Linux32/64 －	(注)	システムログファイル
Windows32/64 ログファイル格納ディレクトリ	トランザクション数 × 0.008 ＋ 0.001	データベース連携サービス運用時

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
トレースログファイル格納ディレクトリ	運用環境の OTS_TRACE_SIZE × 0.001	
リソース管理トレースログファイル格納ディレクトリ	運用環境の RESOUCE_TRACE_ SIZE × 0.001	
リカバリトレースログファイル格納ディレクトリ	運用環境の RECOVERY_TRACE _SIZE × 0.001	
監視プロセストレースログファイル格納ディレクトリ	運用環境の OBSERVE_TRACE_ SIZE × 0.001	
Windows32/64 リソース定義ファイル格納ディレクトリ	登録したリソース定義 ファイル数 × 0.001	
Solaris64 /opt/FSUNots/var (otsgetdumpコマンドによるダンプファイル格納ディレクトリ)	5.0 以上	

注)
データベース連携サービスのシステムログファイルは、isgendefコマンドで指定したシステム規模により異なりますので、以下のように見積もってください。

- small:1Mバイト以上
- moderate:2Mバイト以上
- large:8Mバイト以上
- super:16Mバイト以上

機能: ロードバランス Windows32 Linux32

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Windows32 CORBAサービスのインストールディレクトリ¥etcフォルダ Linux32 /etc/opt	(注)	データファイル

注)
以下のディスク所要量が必要です。

ロードバランスグループ数 × ((1ロードバランスグループあたりのオブジェクトリファレンス数 × オブジェクトリファレンス長) ＋ 0.0005)

初期量として、8256Kバイトを、初回起動時に獲得します。
これを超過した場合、1024Kバイト単位で拡張します。

機能: 性能監視ツール

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
格納ディレクトリ	1.0以上 (注1)	性能ログファイル

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
Solaris64 /var	6.4 × 性能監視ツールの共有メモリサイズ(注2) × 6	異常終了した場合に採取されるcoreファイル

注1)
所要量 ＝ ispmakeenvで指定する共用メモリサイズ × (測定時間 ÷ インターバル時間)

注2)
ispmakeenvコマンドの-mオプションで指定する共有メモリサイズです。

機能: MessageQueueDirector Windows32/64 Linux32/64

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
－	運用の内容により、必要とするサイズを検討してください	詳細は“MessageQueueDirector 説明書”の“環境作成”－”環境作成の説明”－”ファイル資源の準備”－"“ファイル容量の見積り”を参照してください

1.1.2 クライアント機能を使用する場合 Solaris64 Linux32/64

本ソフトウェアを以下の運用で動作させるとき、各ディレクトリにはインストールに必要な“静的ディスク資源”に加えて以下のディスク容量が必要です。空き容量が足りない場合は、該当するファイルシステムのサイズを拡張してください。

機能: CORBAサービス

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
/var/opt	(注1)	ログ情報(プレインストール型Javaライブラリ以外の場合)
任意(Java VMのシステムプロパティのuser.dirで指定)	(注2)	ログ情報(プレインストール型Javaライブラリの場合)
/tmp	1.0 以上 IDL定義の量に依存	IDLコンパイラ動作時

注1)
ログファイルのサイズの上限値は、CORBAサービスのconfigファイルのlog_file_sizeで設定することができます。ディスク容量は、バックアップファイルを1つ残すため“ログファイルサイズの上限値×2”となります。configファイルの詳細については、“CORBAサービスの動作環境ファイル”－“A.1 config”を参照してください。なお、ログファイルは、不要になった時点で、削除してください。

注2)
ログファイルのサイズの上限値は、CORBAサービスのconfigファイルのlog_file_sizeで設定することができます。アプリケーションごとにJVxxxxxxxxxx.log/JVxxxxxxxxxx.old(xxxxxxxxxxは一意の数字)の名前で採取されます。なお、ログファイルは、不要になった時点で、削除してください。

機能: Portable-ORB

ディレクトリ	ディスク容量 (単位:Mバイト)	備考(用途)
/var/opt (注)	“ログファイルサイズ”×2×動作するアプリケーション/アプレット数	porbeditenvコマンドで“ログ情報を採取”を指定した場合のログ情報

注)
アプレットとして動作する場合は、アプレットが動作するクライアントマシン上のローカルディスクに、porbeditenvコマンドで“ログ格納ディレクトリ”として指定したディレクトリとなります。

1.2 メモリ容量

本ソフトウェアを動作させるために必要なメモリ容量は次のとおりです。

注意

動作させるために必要なメモリ容量が確保されていない場合、動作に不具合が生じる場合があります。

1.2.1 サーバ機能を使用する場合

Interstageの各機能を使用する場合のメモリ所要量について説明します。
以下の表を参照し、使用する製品に応じて、各機能のメモリ所要量を見積もってください。表内で使用している略称については、「製品名称の略称について」を参照してください。

	EE	SJE
Interstage管理コンソール	○	○
Interstage HTTP Server	○	○
Interstage JMXサービス	○	○
Interstage シングル・サインオン	○	○
Interstage ディレクトリサービス	○	○
IJServerワークユニット	○	×
Session Registry Server	○	×
CORBAサービス	○	○
イベントサービス／ノーティフィケーションサービス	○	×
Portable-ORB	○	×
コンポーネントトランザクションサービス	○	○
データベース連携サービス	○	×
ロードバランス Windows32 Linux32	○	×
MessageQueueDirector Windows32/64 Linux32/64	○	×

○:該当製品で提供される機能です。
×:該当製品で提供されない機能です。

注)
Interstage シングル・サインオンの業務サーバだけでメモリ容量を見積もります。

機能: Interstage管理コンソール

メモリ所要量(単位:Mバイト)	備考
250.0 以上	―

機能: Interstage HTTP Server

メモリ所要量(単位:Mバイト)	備考
Windows32 22.7 ＋ (0.05 × m) ＋ (0.06 × n) 以上 Windows64 25.2 ＋ (0.04 × m) ＋ (0.12 × n) 以上 Solaris64 25.0 ＋ (4.5 × n) 以上 Linux32/64 8.0 ＋ (3.0 × n) 以上 m: 環境定義ファイルで指定した最大リクエスト同時処理数(httpd.confファイルのThreadsPerChildディレクティブの設定値) n: クライアントからのHTMLファイル同時アクセス数	HTMLファイルを複数クライアント同時アクセス時

機能: Interstage JMXサービス

メモリ所要量(単位:Mバイト)			備考
Windows32/64	Solaris64	Linux32/64	
90.0 以上	270.0 以上	130.0 以上 Linux32 200.0 以上 Linux64	―

機能: Interstage シングル・サインオン

メモリ所要量(単位:Mバイト)	備考
10.0 以上 (注1)	業務サーバ機能
10.0 以上 (注2)	認証サーバ機能
10.0 以上 (注3)	リポジトリサーバ機能

注1)
運用に応じて以下の式で見積もった値を加算してください。(単位:バイト)

― (2,400 ＋ (ロール数 ＋ ロールセット数 ＋ (ロールセット数 × ロール数)) × 2,048以上) × パス定義数＋業務サーバ数 × (2,000,000 ＋ キャッシュサイズ × キャッシュ数)

ロール数　　: SSOリポジトリに定義した保護リソースの、チューニングを行う業務サーバのパス定義に設定したロールの総数
ロールセット数 : SSOリポジトリに定義した保護リソースの、チューニングを行う業務サーバのパス定義に設定したロールセットの総数
パス定義数　: SSOリポジトリに定義した保護リソースの、チューニングを行う業務サーバのパス定義の総数
キャッシュサイズ: “G.4 業務サーバを構築する場合のチューニング”を参照してください。
キャッシュ数　: “G.4 業務サーバを構築する場合のチューニング”を参照してください。

注2)
セション管理を行わない場合は、運用に応じて以下の式で見積もった値を加算してください。(単位:バイト)

― ((サイト定義数 × 1,024) ＋ (パス定義数 × 1,024)) × 2

サイト定義数: SSOリポジトリに定義したサイト定義の総数
パス定義数 : SSOリポジトリのすべてのサイト定義に定義したパス定義の総数

統合Windows認証を行う場合は、256Mバイトを加算してください。

認証サーバ間連携を行う場合は、256Mバイトを加算してください。

注3)
運用に応じて以下の式で見積もった値を加算してください。(単位:バイト)

－ ((ロール数 ＋ ロールセット数 ＋ ロールセット数 × ロール数) × 2,048以上) × 2

ロール数　　: SSOリポジトリに定義したロールの総数
ロールセット数: SSOリポジトリに定義したロールセットの総数

セション管理を行う場合は、上記の算出値に、以下の式から算出される値を加算してください。

－ 23,500,000 ＋ ((同時にシングル・サインオンシステムを使用する利用者数 × (2,560 ＋ α)) × 2)

【α:拡張ユーザ情報】
通知する拡張ユーザ情報の数に応じて、以下の値を加算する。
通知する拡張ユーザ情報のサイズ × 2

ユーザ情報を登録するディレクトリサービスにActive Directoryを使用し、シングル・サインオンの拡張スキーマを使用しない場合は、上記の算出値に、以下の式から算出される値を加算してください。

－ Active Directoryのロール/ロールセットに使用する属性の総数 × 524 × 2

機能: Interstage ディレクトリサービス

メモリ所要量(単位:Mバイト)			備考
Windows32/64	Solaris64	Linux32/64	
340.0 以上 (注)	400.0 以上 (注)	217.0以上 (注)	スタンドアロン、またはデータベース共用で運用する場合
2.0 以上			エントリ管理コマンドを使用する場合
22.0 以上	60.0 以上	60.0 以上	エントリ管理ツールを使用する場合
m × n × 3 m: 1エントリの登録に使用したLDIFファイルのサイズ n: 検索により通知されるエントリ数			エントリの検索時

注)
リポジトリを複数作成して運用する場合は、リポジトリ数を乗算してください。

機能: IJServerワークユニット

“WebアプリケーションとEJBアプリケーションを同一JavaVMで運用”時 (注1)(注2)

メモリ所要量(単位:Mバイト)			配備したアプリケーションの種類とファイルサイズ
Windows32/64	Solaris64	Linux32/64	
168.2以上 Windows32 184.5以上 Windows64	280.0以上	677.2 以上 Linux32 3189.6以上 Linux64	Webアプリケーション 14.3KB EJBアプリケーション 10.6KB (Session,BMP)
169.4以上 Windows32 187.9以上 Windows64	284.8以上	679.9 以上 Linux32 3187.2 以上 Linux64	Webアプリケーション 14.4KB EJBアプリケーション 8.5KB (Session,CMP1.1)

メモリ所要量(単位:Mバイト)			配備したアプリケーションの種類とファイルサイズ
Windows32/64	Solaris64	Linux32/64	
183.3以上 Windows32 198.7以上 Windows64	303.7以上	682.3 以上 Linux32 3195.3 以上 Linux64	Webアプリケーション 16.6KB EJBアプリケーション 13.7KB (Session,CMP2.0)
187.6以上 Windows32 200.7以上 Windows64	306.9 以上	690.9以上 Linux32 3216.2以上 Linux64	Webアプリケーション 10.9KB EJBアプリケーション 10.8KB （Session,MDB)

“Webアプリケーションのみ運用”時 (注1)

メモリ所要量(単位:Mバイト)			配備したアプリケーションの種類とファイルサイズ
Windows32/64	Solaris64	Linux32/64	
140.0以上 Windows32 184.6以上 Windows64	271.9以上	672.9以上 Linux32 3183.9以上 Linux64	Webアプリケーション 5.8KB

“EJBアプリケーションのみ運用”時 (注2)

メモリ所要量(単位:Mバイト)			配備したアプリケーションの種類とファイルサイズ
Windows32/64	Solaris64	Linux32/64	
104.6以上 Windows32 162.7以上 Windows64	239.6以上	838.5以上 Linux32 3338.9以上 Linux64	EJBアプリケーション 10.6KB (Session,BMP)
105.3以上 Windows32 169.5以上 Windows64	250.7以上	840.5以上 Linux32 3343.1以上 Linux64	EJBアプリケーション 8.5KB (Session,CMP1.1)
141.7以上 Windows32 174.7以上 Windows64	282.8以上	856.4以上 Linux32 3342.8以上 Linux64	EJBアプリケーション 13.7KB (Session,CMP2.0)
143.6以上 Windows32 176.4以上 Windows64	281.3以上	862.0以上 Linux32 3362.9以上 Linux64	EJBアプリケーション 10.8KB (Session,MDB)

注1)
詳細は以下の式で見積もってください。(単位:Mバイト)

- Webサーバコネクタ

Windows32/64
0.2 × k ＋ 2.0
Solaris64
1.9 × k ＋ 30.0

Linux32/64
1.0 × k ＋ 30.0

k: Servletサービスへの同時アクセス数

・ IJServerワークユニット:(プロセス多重度1当り)

― “WebアプリケーションとEJBアプリケーションを運用”の場合

Windows32/64
48.0 ＋ (1.4 × k) ＋ (0.7 × w) ＋ (P1＋P2＋P3＋..＋Pn)

Solaris64
121.0 ＋ (2.1 × k) ＋ (0.7 × w) ＋ (P1＋P2＋P3＋..＋Pn)

Linux32/64
28.0 ＋ (1.5 × k) ＋ (0.7 × w) ＋ (P1＋P2＋P3＋..＋Pn)

― “Webアプリケーションのみ運用”の場合

Windows32/64
47.0 ＋ (1.3 × k) ＋ (0.7 × w) ＋ (P1＋P2＋P3＋..＋Pn)

Solaris64
84.0 ＋ (2.5 × k) ＋ (0.7 × w) ＋ (P1＋P2＋P3＋..＋Pn)

Linux32/64
27.0 ＋ (1.3 × k) ＋ (0.7 × w) ＋ (P1＋P2＋P3＋..＋Pn)

k: Servletコンテナへの同時アクセス数
w: Webアプリケーションの数
Pn: 各ServletまたはJSPの実行サイズ。上記表では1Mバイトとして計算

セションリカバリ機能(Session Registry Client)を使用する場合は、以下の値を加算してください。

― (0.002 ＋ セションの保持するデータ容量(※)) × 想定されるセション数
※: Webアプリケーションでセションの属性(Attribute)にセットするオブジェクトおよびキーのサイズの合計値

なお、セションリカバリ機能(Session Registry Client)は、Interstage Application Server Enterprise Editionで運用可能です。

ServletはJava VM上で動作するため、実際のメモリ使用量(ヒープ領域を含む)は、以下に示す要因により異なります。

― newするクラス型

― newするインスタンスの個数

― インスタンスのライフサイクル

― GCの動作状況

― IJServerワークユニットの各種定義

― 使用するJava VM

そのため正確なメモリ使用量(ヒープ領域、Perm領域)は次のようにして実測することにより見積もることを推奨します。

― 本番運用のピーク時と同一条件で動作させます。Java VMが使用するメモリが不足すると、イベントログにメッセージが出力されますので、ヒープ領域やPerm領域の最大値を増やして、最適な値としてください。求めたヒープ領域やPerm領域の最大値をそのまま本番運用時の値として利用します。

注2)
以下を参考に、EJBサービス運用時のメモリ所要量を見積もってください。
EJBアプリケーション運用時、Java VMが使用するメモリ量(初期値、最大値)および1プロセスで必要な全メモリ量は、以下に示す要因により異なります。

― newするクラス型

― newするインスタンスの個数

― インスタンスのライフサイクル

― GCの動作状況

― EJBアプリケーションの各種定義

いずれのメモリ量も簡単には算出できないので、次のようにして実測することにより見積もってください。

1. Java VMが使用するメモリ量の初期値(javaコマンドの-Xmsオプションで指定する値)
   EJBアプリケーションを、本番運用の通常時(ピーク時ではない)と同一条件で動作させます。Java VMが使用するメモリ量(最大値)が不足すると、IJServer21033またはEJB1033メッセージが出力されますので、試行錯誤によりメモリ量(最大値)を最適な値としてください。このようにして求めたメモリ量(最大値)を本番運用時のメモリ量(初期値)として利用します。メモリ量(初期値)の省略値は2Mバイトです。
2. Java VMが使用するメモリ量の最大値(javaコマンドの-Xmxオプションで指定する値)
   EJBアプリケーションを、本番運用のピーク時と同一条件で動作させます。Java VMが使用するメモリ量(最大値)が不足すると、IJServer21033またはEJB1033メッセージが出力されますので、試行錯誤によりメモリ量(最大値)を最適な値としてください。このようにして求めたメモリ量(最大値)をそのまま本番運用時のメモリ量(最大値)として利用します。メモリ量(最大値)の省略値は64Mバイトです。
3. 1プロセスで必要な全メモリ量
   1)と2)でJava VMが使用するメモリ量を見積り時、同時に1プロセスで必要な全メモリ量も実測して見積もってください。

機能: Session Registry Server (J2EE)

メモリ所要量(単位:Mバイト)			備考
Windows32/64	Solaris64	Linux32/64	
(例)140 (注)	(例)254 (注)	(例)120 (注)	―

注)
詳細は以下の式で見積もってください。(単位:Mバイト)

Windows32/64
48.7 ＋ (1.3 × k) ＋ (0.01 × a) ＋ ((0.002 ＋ d) × s) × 2
Solaris64
85.7 ＋ (2.5 × k) ＋ (0.01 × a) ＋ ((0.002 ＋ d) × s) × 2
Linux32/64
28.7 ＋ (1.3 × k) ＋ (0.01 × a) ＋ ((0.002 ＋ d) × s) × 2

k: Session Registry Serverの同時処理数
a: (IJServerに配備している)Webアプリケーションの数
d: セションの保持するデータ容量 ＝
　Webアプリケーションでセションの属性(Attribute)にセットするオブジェクトおよびキーのサイズの合計値。
s: セション数

例: 対象とするIJServerは同時処理数64、アプリケーション1つ、セションに格納するデータ量が2KB、セション数が1000の場合。

Windows32/64
48.7 ＋ (1.3 × 64) ＋ (0.01 × 1) ＋ ((0.002 ＋ 0.002) × 1000) × 2
＝ 48.7 ＋ 83.2 ＋ 0.01 ＋ 8
≒ 140
Solaris64
85.7 ＋ (2.5 × 64) ＋ (0.01 × 1) ＋ ((0.002 ＋ 0.002) × 1000) × 2
＝ 85.7 ＋ 160 ＋ 0.01 ＋ 8
≒ 254
Linux32/64
28.7 ＋ (1.3 × 64) ＋ (0.01 × 1) ＋ ((0.002 ＋ 0.002) × 1000) × 2
＝ 28.7 ＋ 83.2 ＋ 0.01 ＋ 8
≒ 120

Session Registry ServerはJava VM上で動作するため、実際のメモリ使用量(ヒープ領域を含む)は、負荷やGCの動作状況により異なります。
そのため正確なメモリ使用量は次のようにして実測することにより見積もることを推奨します。

ー 本番運用のピーク時と同一条件で動作させます。Java VMが使用するメモリが不足すると、イベントログ/システムログにメッセージが出力されますので、ヒープ領域の最大値を増やして、最適な値としてください。求めたヒープ領域の最大値をそのまま本番運用時の値として利用します。

なお、Session Registry Serverは、Interstage Application Server Enterprise Editionで運用可能です。

機能: CORBAサービス

メモリ所要量(単位:Mバイト)	備考
16.0 以上 (注1)	ー
8.0 以上	ネーミングサービス運用時
45.6 以上 (注2)	インタフェースリポジトリ運用時
2.4	COBOL Webサブルーチン使用時

注1)
CORBAサービスの動作環境定義(configファイル)の設定により、16Mバイト ＋ 加算値(下表)が必要です。

運用形態	必要数(加算値)(単位:Kバイト)
CORBAサービス運用時	100.0 ＋ limit_of_max_IIOP_resp_con × 16.0 ＋ limit_of_max_IIOP_resp_requests × 16.0 ＋ max_impl_rep_entries × 6.0 (以上)
トレース機能を使用する場合	(CORBAサービス運用時) ＋ 20.0 ＋ max_processes × trace_size_per_process (以上)
スナップショット機能を使用する場合	(CORBAサービス運用時) ＋ 10.0 ＋ snap_size (以上)

なお、上記のlimit_of_[パラメタ名]のデフォルト値は以下となります。0が指定された場合も、以下と同様になります。

[パラメタ名] × 1.3 (小数部分切り捨て)

isconfig.xmlファイルの定義項目AutoConfigurationModeにMANUALを指定し、自動拡張を行わない設定にした場合は、[パラメタ名]となります。

また、CORBAアプリケーションを動作させる場合、1プロセスあたり1.5 Mバイト以上のメモリが必要となります。

注2)
インタフェースリポジトリは、起動時にデータベースに格納されているオブジェクトをメモリ上に展開します。インタフェースリポジトリを使用する場合のメモリ容量について説明します。

ー 固定使用領域
45.6 Mバイト

ー 可変使用領域
インタフェースリポジトリでは、オブジェクトごとにメモリが使用されます。
以下の計算式より、オブジェクトごとの使用メモリを算出することができます。

項番	IDL定義	計算式(単位:バイト)
1	モジュール宣言	3902＋a×(2×b＋2)
2	インタフェース宣言	3902＋a×(2×b＋2)＋a×b×c
3	オペレーション宣言	3934＋a×(3×b＋2＋f)＋a×b×g＋h×(12＋a＋a×b)
4	属性宣言	3910＋a×(3×b＋2)
5	定数宣言	7704＋a×(3×b＋3)＋d
6	例外宣言	3836＋a×(2×b＋e＋1)＋e×(78＋a＋a×b)

項番	IDL定義	計算式(単位:バイト)
7	文字列型宣言(ワイド文字列を含む)	3882＋a×(b＋1)
8	列挙型宣言	3918＋a×(2×b＋k＋2)
9	シーケンス型宣言	3882＋a×(2×b＋1)
10	構造体宣言	3766＋a×(2×b＋i＋1)＋i×(78＋a＋a×b)
11	共用体宣言	3840＋a×(3×b＋j＋1)＋j×(3880＋2×a＋a×b)
12	固定小数点型宣言	3882＋a×(b＋1)
13	配列宣言	3886＋a×(2×b＋1)

記号	項目	意味
a	識別子長	対象オブジェクトの識別子の長さ
b	階層数	対象オブジェクトの存在する階層
c	継承数	インタフェース宣言が継承するインタフェース数
d	定数値長	定数宣言の値の長さ
e	例外構造体メンバ数	例外宣言の構造体のメンバ数
f	コンテキスト数	オペレーション宣言でのコンテキスト数
g	例外数	オペレーション宣言での例外数
h	パラメタ数	オペレーション宣言でのパラメタ数
i	構造体メンバ数	構造体宣言でのメンバ数
j	共用体メンバ数	共用体宣言でのメンバ数
k	列挙型メンバ数	列挙型宣言でのメンバ数

機能: イベントサービス／ノーティフィケーションサービス

メモリ所要量(単位:Mバイト)			備考
Windows32/64	Solaris64	Linux32/64	
28.0 以上	30.0 以上	9.0 以上	イベントサービス運用時
23.0 以上	25.0 以上	6.0 以上	イベントファクトリ運用時
イベントチャネル数 × イベントチャネルのメモリ使用量 (注1)			揮発チャネル運用時
ユニット数×100＋ユニット定義ファイルのshmmaxの合計			不揮発チャネル運用時
((a＋d)＋(b×c)＋(e×f))×g (Kバイト) (注2)			essetcnfコマンドを実行して静的生成のイベントチャネルのコンシューマ数・サプライヤ数を拡張する場合
((a＋d)＋(b×c)＋(e×f))×h (Kバイト) (注2)			essetcnfコマンドを実行して動的生成のイベントチャネルのコンシューマ数・サプライヤ数を拡張する場合
((j＋m)＋(k×l)＋(n×o))×i (Kバイト) (注2)			essetcnfchnlコマンドを実行して静的生成のイベントチャネルのコンシューマ数・サプライヤ数を拡張する場合
メッセージ本文のサイズ×蓄積メッセージ数			イベントチャネルに蓄積するイベントデータの形式に、any型を使用する場合 (注3)

メモリ所要量(単位:Mバイト)			備考
Windows32/64	Solaris64	Linux32/64	
(メッセージ本文のサイズ＋(QoSプロパティ項目数×4Kバイト))×蓄積メッセージ数			イベントチャネルに蓄積するイベントデータの形式に、StructuredEvent型を使用する場合(注3)

注1)
イベントチャネルのメモリ使用量は、イベントチャネル作成時に指定する最大接続数(esmkchnlコマンドの-mオプションの設定値)に依存します。
最大接続数に対するメモリ使用量の目安を、参考として以下の表に示します。

最大接続数	メモリ使用量(Mバイト)				
	Windows32	Windows64	Solaris64	Linux32	Linux64
16	14	15	35	10	(RHEL6) 12 (RHEL7) 14
50	15	16	35	10	(RHEL6) 15 (RHEL7) 15
100	16	19	36	10	(RHEL6) 15 (RHEL7) 15
150	17	21	36	11	(RHEL6) 16 (RHEL7) 16
200	18	24	37	11	(RHEL6) 16 (RHEL7) 16
250	19	26	38	11	(RHEL6) 20 (RHEL7) 20
500	25	35	41	注)OS制限によりスレッドを生成できません。	(RHEL6) 23 (RHEL7) 23
1000	注)OS制限によりスレッドを生成できません。	60	47		(RHEL6) 31 (RHEL7) 31
2000		100	59		(RHEL6) 48 (RHEL7) 48
3000		140	68		(RHEL6) 62 (RHEL7) 66

注2)
以下の項目の設定値により算出してください。essetcnfコマンドおよびessetcnfchnlコマンドについては、“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。

記号	項目	設定方法
a	コンシューマ数の初期値	essetcnfコマンドの-coninitオプションの設定値
b	コンシューマ数が初期値を超えた場合の拡張数	essetcnfコマンドの-conextオプションの設定値
c	コンシューマ数が初期値を超えた場合の拡張回数	essetcnfコマンドの-conenumオプションの設定値
d	サプライヤ数の初期値	essetcnfコマンドの-supinitオプションの設定値
e	サプライヤ数が初期値を超えた場合の拡張数	essetcnfコマンドの-supextオプションの設定値

記号	項目	設定方法
f	サプライヤ数が初期値を超えた場合の拡張回数	essetcnfコマンドの-supenumオプションの設定値
g	イベントチャネルのグループ数(イベントチャネルの総グループ数－essetcnfchnlコマンドでの設定対象グループ数(iの値))	－
h	イベントチャネルの最大起動数	essetcnfコマンドの-dchmaxオプションの設定値
i	イベントチャネルのグループ数(essetcnfchnlコマンドでの設定対象)	－
j	コンシューマ数の初期値	essetcnfchnlコマンドの-coninitオプションの設定値
k	コンシューマ数が初期値を超えた場合の拡張数	essetcnfchnlコマンドの-conextオプションの設定値
l	コンシューマ数が初期値を超えた場合の拡張回数	essetcnfchnlコマンドの-conenumオプションの設定値
m	サプライヤ数の初期値	essetcnfchnlコマンドの-supinitオプションの設定値
n	サプライヤ数が初期値を超えた場合の拡張数	essetcnfchnlコマンドの-supextオプションの設定値
o	サプライヤ数が初期値を超えた場合の拡張回数	essetcnfchnlコマンドの-supenumオプションの設定値

注3)
イベントサービスの形式については、“アプリケーション作成ガイド(イベントサービス編)”の“イベントデータの形式”を参照してください。

機能: Portable-ORB

メモリ所要量(単位:Mバイト)		備考
Windows32 Linux32	Windows64 Solaris64 Linux64	
1.5 以上	3.0 以上	－

機能: コンポーネントトランザクションサービス

メモリ所要量(単位:Mバイト)			備考
Windows32/64	Solaris64	Linux32/64	
48.0 以上 (注1)	50.0 以上 (注2)		サービスの起動
－	4.0 以上 (注3)		

注1)
この値はCORBAサービスのメモリ容量を含んでいませんので、加算してください。

注2)
ユーザ認証機能を使用する場合は、0.9Mバイト加算してください。
アクセス制御を使用する場合は、0.6Mバイト加算してください。

注3)
1つのワークユニットでプロセス多重度を1とした場合の値です。
詳細は以下の式で見積もってください。

－ 4.0 × ワークユニット配下のプロセス数の総和

機能: データベース連携サービス

メモリ所要量(単位:Mバイト)	備考
Windows32 54.0 ＋ 0.010 × m ＋ 43.0 × n 以上 Windows64 64.0 ＋ 0.011 × m ＋ 32.0 × n 以上 Solaris64 30.0 ＋ 0.010 × m ＋ 75.0 × n 以上 Linux32 34.0 ＋ 0.009 × m ＋ 23.0 × n 以上 Linux64 36.0 ＋ 0.010 × m ＋ 23.0 × n 以上 m: 最大トランザクション数 n: リソース管理ごとの多重度＋1の総数	サービスの起動 (データベース連携サービス動作マシン)
Windows32 22.0 ＋ 27.0 × n 以上 Windows64 27.0 ＋ 27.0 × n 以上 Solaris64 27.0 ＋ 75.0 × n 以上 Linux32/64 14.0 ＋ 20.0 × n 以上 n: リソース管理ごとの多重度＋1の総数	サービスの起動 (リソース管理プログラムだけを起動するマシン)

機能: ロードバランス Windows32 Linux32

メモリ所要量(単位:Mバイト)	備考
2.0	―

機能: MessageQueueDirector Windows32/64 Linux32/64

注) MQDシステムが複数ある場合には、それぞれのMQDシステムについて見積もった値の合計が所要量になります。

メモリ所要量(単位:Mバイト)	備考
Windows32 100.0 ＋ m ＋ s ÷ 1000 以上 Windows64 Linux32/64 100.0 ＋ m m: MQD環境定義のMQDConfigurationセクションのMessageBufferMaxSize s: MQD環境定義のMemoryQueueセクションのsize	基本機能使用時
39.0 ＋ sc × 0.3 ＋ rc × 0.3 以上 sc: イベントチャネル連携サービスのCHANNELセクション定義数 rc: イベントチャネル連携サービスのRCHANNELセクション定義全部の総集信数	イベントチャネル連携サービス使用時
Windows32 10.0 以上	同報配信サービス使用時

1.2.2 クライアント機能を使用する場合

本ソフトウェアを以下の運用で動作させるときに使用するメモリ容量を示します。

機能: Interstage ディレクトリサービス

メモリ所要量(単位:Mバイト)	備考
22.0 以上	エントリ管理ツールを使用する場合

第2章 Interstageのチューニング

Interstageはシステム規模の指定だけで、システム運用が可能となるようなモデルケースを設定して各サービスの定義を登録しています。しかし、システムによっては、より詳細なシステム構築が必要となります。
Interstageをチューニングする場合は、isregistdefコマンドで各サービスの定義を登録した後で実施してください。チューニングした内容はInterstageの初期化機能で有効となり、Interstageの起動で反映されます。
Interstageのチューニングは、以下の定義ファイルに対して行います。

- Interstage動作環境定義
- CORBAサービスの動作環境ファイル
- コンポーネントトランザクションサービスの環境定義
- EE データベース連携サービスのシステム環境定義

EE 2.1 想定するシステム形態(マルチ言語サービス) Windows32 Linux32

Interstageの初期化時に、トランザクションアプリケーションを使用した連携をモデルケースとして設定しています。

トランザクションアプリケーションを利用した連携は、以下のとおりです。

- ローカルトランザクション連携
- グローバルトランザクション連携
- セション管理を利用した連携
- Windows32
  既存システム(グローバルサーバ)との連携

トランザクションアプリケーションは、以下の条件で設計しています。

- すべてのトランザクションアプリケーションは、他サーバ(自システム内の連携も含みます)と連携することを想定しています。
- トランザクションアプリケーションのオブジェクトのプロセス数は、最大接続クライアント数の10分の1です。
- トランザクションアプリケーションのオブジェクトから接続するサーバマシンは、1台です。
- リソースはサーバマシンに1つ、リソース管理プログラムの多重度は5です。
- データベース連携サービスの多重度は5、リカバリプログラムの多重度は2です。

CORBAアプリケーションの運用、ロードバランス、サーバマシン状態監視を使用する場合は、Interstageのチューニングが必要です。

2.2 定義ファイルの設定値

各定義ファイルには、Interstageシステム定義のSystem Scaleステートメントに設定されているシステム規模に応じた値が設定されます。

システム規模には、以下の4種類があり、isgendefコマンドでInterstageシステム定義を生成する際に指定します。isgendef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「isgendef」を参照してください。

- small(小規模システム)
- moderate(中規模システム)
- large(大規模システム)
- super(超大規模システム)

なお、インストール直後のセットアップ環境では、以下のシステム規模が設定されています。

- Windows32/64
  large（大規模システム）
- Solaris64 Linux32/64
  small（小規模システム）

各定義ファイルに設定される値を以下に示します。

■システム規模ごとの設定値

定義名	ステートメント	値(System Scaleごとの)			
		small	moderate	large	super
Interstage動作環境定義	Corba Host Name	ありません。			
	Corba Port Number	ありません。			
	IR path for DB file	Windows32/64 TD_HOME¥var¥IRDB (注1) Solaris64 Linux32/64 TD_HOME/var/IRDB (注2)			
	IR USE	ありません。(注3)			
	IR Host Name	ありません。 Solaris64 Linux32/64 (注3)			
	IR Port Number	8002			
	NS USE	ありません。(注3)			
	NS Host Name	ありません。 Solaris64 Linux32/64 (注3)			
	NS Port Number	8002			
	NS JP	no			
	NS Locale	Windows32/64 SJIS Solaris64 Linux32/64 EUC			
	EE Windows32 Linux32 LBO USE	no			
	TD path for system	Windows32/64 TD_HOME¥var (注1) Solaris64 /var/opt/FJSVisas/system/default/FSUNextp Linux32/64 /var/opt/FJSVisas/system/default/FJSVextp			
	EE OTS Multiple degree	5			
	EE OTS Recovery	2			
	EE OTS path for system log	ありません。(注4)			

定義名	ステートメント	値(System Scaleごとの)			
		small	moderate	large	super
	EE OTS maximum Transaction	Windows32/64 5 Solaris64 Linux32/64 50	Windows32/64 10 Solaris64 Linux32/64 100	Windows32/64 50 Solaris64 Linux32/64 500	Windows32/64 100 Solaris64 Linux32/64 1000
	EE OTS Setup mode	sys			
	EE Windows32/64 Linux32/64 OTS JTS's RMP Multiple degree of Process	5			
	EE Windows32/64 Linux32/64 OTS JTS's RMP Multiple degree of Thread	16			
	EE OTS Participate	4			
	EE OTS Host	ありません。			
	EE OTS Port	ありません。			
	EE OTS Locale	ありません。			
	EE Event Service	no			
	EE Event maximum Process	2			
	EE Event maximum Connection	Windows32/64 5 Solaris64 Linux32/64 50	Windows32/64 10 Solaris64 Linux32/64 100	Windows32/64 50 Solaris64 Linux32/64 500	Windows32/64 100 Solaris64 Linux32/64 1000
	EE Event Locale	Windows32/64 SJIS Solaris64 Linux32/64 EUC			
	EE Event Auto Disconnect	no			

定義名	ステートメント	値(System Scaleごとの)			
		small	moderate	large	super
	EE Event SSL	no			
	SSL USE	no			
	SSL Port Number	4433			
	IS Monitor Mode	mode2			
	EE FJapache	no			
CORBAサービスの動作環境ファイル (注5)	max_IIOP_resp_con	Windows32/64 33 Solaris64 Linux32/64 80	Windows32/64 40 Solaris64 Linux32/64 135	Windows32/64 100 Solaris64 Linux32/64 575	Windows32/64 175 (注6) Solaris64 Linux32/64 1024
	max_IIOP_resp_requests	Windows32/64 772 Solaris64 Linux32/64 1920	Windows32/64 896 Solaris64 Linux32/64 2944	Windows32/64 1920 Solaris64 Linux32/64 10112	Windows32/64 3968 Solaris64 Linux32/64 20352
	max_processes	Windows32/64 29 Solaris64 Linux32/64 31	Windows32/64 31 Solaris64 Linux32/64 36	Windows32/64 51 Solaris64 Linux32/64 76	Windows32/64 76 Solaris64 Linux32/64 126
	max_exec_instance	448	448	Windows32/64 448 Solaris64 Linux32/64 1046	Windows32/64 448 Solaris64 Linux32/64 2046
コンポーネントトランザクションサービスの環境定義	[SYSTEM ENVIRONMENT] System Scale	small	moderate	large	super
EE データベース連携サービスの環境定義	ありません。	ありません。			

注1)

TD_HOME:Interstageのインストールフォルダ¥td

注2)

TD_HOME:コンポーネントトランザクションサービスのインストールディレクトリ

注3)

運用形態が「TYPE3」の場合は、必ず指定してください。

注4)

運用形態が「TYPE2」の場合は、必ず指定してください。

注5)

CORBAサービスの動作環境ファイル内の値に表中の値がisregistdefコマンド実行時に**加算**されます。また、isregistdefコマンドの投入が初回でない場合は、前回のコマンド投入時に加算した値を、現在設定されている値から減算し、新たに指定したSystem Scaleの値が加算されます。詳細については、“■CORBAサービスの動作環境ファイルの設定について”を参照してください。

注6)

1024以上の値は、設定できません。

■CORBAサービスの動作環境ファイルの設定について

CORBAサービスの動作環境定義ファイルの定義値は、isregistdefコマンドによるセットアップの実行時に、以下のように設定されます。

- セットアップ実行時に、CORBAサービスの動作環境ファイルの定義値に対し、必要な値が加算されます。
- すでにセットアップ済みの環境に対し、スケールを変更した場合には、加算値の差分の値が反映されます。スケールを大きくした場合には、加算値の差分の値が加算され、スケールを小さくした場合には、加算値の差分の値が減算されます。
- 加算値は、スケールに対して一定です。

以下に定義値max_IIOP_resp_conに対する設定例を示します。

- Interstageがセットアップされていない環境に対し、isregistdefコマンドを実行した場合(システム規模:small)。
  - max_IIOP_resp_conの値8に対して、33を加算した値41が設定される。
  - max_IIOP_resp_requestsの値128に対して、772を加算した値900が設定される。
  - max_processの値20に対して、29を加算した値49が設定される。
  - max_exec_instanceの値512に対して、448を加算した値960が設定される。
- システム規模がsmallの環境に対し、システム規模をlargeに変更した場合
  - max_IIOP_resp_conの値41に対して、スケール間の加算値の差分+67(100-33)が反映された値108が設定される。
  - max_IIOP_resp_requestsの値900に対して、スケール間の加算値の差分+1148(1920-772)が反映された値2048が設定される。
  - max_processの値49に対して、スケール間の加算値の差分+22(51-29)が反映された値71が設定される。
  - max_exec_instanceの値960に対して、スケール間の加算値の差分+0(448-448)が反映された値960が設定される。
- システム規模がlargeの環境に対し、システム規模をsmallに変更した場合
  - max_IIOP_resp_conの値108に対して、スケール間の加算値の差分-67(33-100)が反映された値41が設定される。
  - max_IIOP_resp_requestsの値2048に対して、スケール間の加算値の差分-1148(772-1920)が反映された値900が設定される。
  - max_processの値71に対して、スケール間の加算値の差分-22(29-51)が反映された値49が設定される。
  - max_exec_instanceの値960に対して、スケール間の加算値の差分-0(448-448)が反映された値960が設定される。

なお、isregistdefコマンドを使用せずに、CORBAサービスの動作環境定義ファイルの値を変更した場合、それ以降に本セットアップを行っても、その変更時の差分の値は有効です。

注意

システムスケールを小さくする場合には、CORBAサービスの動作環境ファイルの定義値が減算されます。CORBAサービスの動作環境ファイルの定義値が、必要量を下回らないように注意してください(セットアップ後、CORBAサービスの動作環境定義ファイルの値を小さくカストマイズしなおしている場合に注意が必要です)。

2.3 チューニング方法(マルチ言語サービス)

2.3.1 アプリケーション追加によるチューニング

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を追加する場合に修正する各サービスの定義のステートメントと加算する値を説明します。

なお、追加するアプリケーションがCORBAアプリケーションかトランザクションアプリケーションにより加算する値は異なります。

以下に示すアプリケーションを追加した場合のチューニング方法について説明します。

- クライアントアプリケーションを追加した場合
- サーバアプリケーションを追加した場合
- クライアント、サーバ兼用アプリケーションを追加した場合

2.3.1.1 クライアントアプリケーションを追加した場合

CORBAアプリケーション

定義名	ステートメント	加算値
CORBAサービスの動作環境ファイル	max_processes (注1)	プロセス数の合計
	max_IIOP_resp_con (注1)(注2)	

注1) Solaris64 Linux32/64

max_processes、max_IIOP_resp_conを変更した場合は、システムパラメタの設定が必要です。

注2)

SSL接続のコネクションとSSL接続でないコネクションは別コネクションとして数える必要があります。そのため、SSL連携機能を使用する場合、加算値は“プロセス数の合計 × 2”となります。

EJBクライアントアプリケーション

定義名	ステートメント	加算値
CORBAサービスの動作環境ファイル	max_processes (注)	追加するクライアントアプリケーションのプロセス数

注) Solaris64 Linux32/64

max_processesを変更した場合は、システムパラメタの設定が必要です。

2.3.1.2 サーバアプリケーションを追加した場合

CORBAアプリケーション

定義名	ステートメント	加算値
CORBAサービスの動作環境ファイル	max_processes (注)	プロセス数の合計
	max_exec_instance	リクエスト実行用スレッドの合計

注) Solaris64 Linux32/64

max_processesを変更した場合は、システムパラメタの設定が必要です。

EJBアプリケーション

定義名	ステートメント	加算値
CORBAサービスの動作環境ファイル	max_processes (注)	追加するEJBアプリケーションのプロセス数
	max_exec_instance	Windows32/64 ・ EJBアプリケーション作成の際に、スレッド動作可としている場合 追加するEJBアプリケーションのプロセス数 × 16 ・ EJBアプリケーション作成の際に、スレッド動作不可としている場合 追加するEJBアプリケーションのプロセス数 Solaris64 Linux32/64 追加するEJBアプリケーションのプロセス数 × 16

注) Solaris64 Linux32/64

max_processesを変更した場合は、システムパラメタの設定が必要です。

トランザクションアプリケーション Windows32 Linux32

考慮の必要はありません。

2.3.1.3 クライアント、サーバ兼用アプリケーションを追加した場合

サーバアプリケーションから他のオブジェクトを呼び出したり、オブジェクトリファレンスを獲得したり、セション管理機能、XA連携などを使用する場合など、CORBAクライアントとしても動作するアプリケーションを示します。

CORBAアプリケーション

定義名	ステートメント	加算値
CORBAサービスの動作環境ファイル	max_processes (注1)	プロセス数の合計
	max_IIOP_resp_con (注1)(注2)	
	max_exec_instance	リクエスト実行用スレッドの合計

注1) Solaris64 Linux32/64

max_processes、max_IIOP_resp_conを変更した場合は、システムパラメタの設定が必要です。

注2)

SSL接続のコネクションとSSL接続でないコネクションは別コネクションとして数える必要があります。そのため、SSL連携機能を使用する場合、加算値は“プロセス数の合計 × 2”となります。

トランザクションアプリケーション Windows32 Linux32

定義名	ステートメント	加算値
CORBAサービスの動作環境ファイル	max_processes (注1)	プロセス数の合計
	max_IIOP_resp_con (注1)(注2)	

注1) Linux32

max_processes、max_IIOP_resp_conを変更した場合は、システムパラメタの設定が必要です。

注2)

SSL接続のコネクションとSSL接続でないコネクションは別コネクションとして数える必要があります。そのため、SSL連携機能を使用する場合、加算値は“プロセス数の合計 × 2”となります。

2.3.2 Interstage機能を使用するためのチューニング

Interstageの機能を使用する場合のチューニング方法について説明します。

以下の表を参照し、使用する製品に応じて、以降に示すサービスのチューニングを行ってください。表内で使用している略称については、「製品名称の略称について」を参照してください。

	EE	SJE
データベース連携サービス	○	×
ロードバランス Windows32 Linux32	○	×
イベントサービス (注)	○	×
サーバマシン状態監視 Windows32 Linux32	○	×

○:チューニングを行う必要があります。
×:チューニングを行う必要はありません(該当製品では、サービスが使用できないため)。

注) Interstage JMSを使用する場合、イベントサービスのチューニングを行う必要があります。

EE 2.3.2.1 データベース連携サービス

データベース連携サービスの多重度

データベース連携サービスの多重度を変更する場合は、以下の値を設定または加算します。

定義名	ステートメント	加算、設定値
Interstage動作環境定義	OTS Multiple degree (注1)	データベース連携サービスの多重度
CORBAサービスの動作環境ファイル	max_IIOP_resp_requests (注2) (注3)	
	max_exec_instance (注2)	

注1)

値を設定します。

注2)

値を加算します。

注3)

データベース連携サービスの多重度がmax_IIOP_resp_requestsより大きい場合は、データベース連携サービスの多重度の値を設定します。

リカバリプログラムの多重度のチューニング

リカバリプログラムの多重度をチューニングする場合は、以下の値を設定または加算します。

定義名	ステートメント	加算、設定値
Interstage動作環境定義	OTS Recovery (注1)	リカバリプログラムの多重度
CORBAサービスの動作環境ファイル	max_IIOP_resp_requests (注2)	
	max_exec_instance (注2)	

注1)

値を設定します。

注2)

値を加算します。

リソース管理プログラムのチューニング

リソース管理プログラムを複数起動する場合または、リソース管理プログラムの多重度を変更する場合は以下の値を加算します。

定義名	ステートメント	加算値
CORBAサービスの動作環境ファイル	max_processes (注)	(リソース管理プログラムの多重度 ＋ 1)の合計
	max_IIOP_resp_con (注)	
	max_exec_instance	

注) Solaris64 Linux32/64

max_processes、max_IIOP_resp_conを変更した場合は、システムパラメタの設定が必要です。

EE 2.3.2.2 ロードバランス Windows32 Linux32

ロードバランス機能を使用する場合は、以下の値を加算します。

定義名	ステートメント	加算値
CORBAサービスの動作環境ファイル	max_processes (注)	定数: 1
	max_IIOP_resp_con (注)	
	max_exec_instance	odsetlboコマンドの-mオプションに指定した値

注) Linux32

max_processes、max_IIOP_resp_conを変更した場合は、システムパラメタの設定が必要です。

EE 2.3.2.3 イベントサービス

イベントサービスを使用する場合は、以下の値を設定または加算します。

定義名	ステートメント	加算、設定値
CORBAサービスの動作環境ファイル	max_exec_instance	(注2)
	max_IIOP_local_init_con	以下のいずれかの最大値 ・ max_IIOP_local_init_con ・ 起動するコンシューマ／サプライヤのプロセス数の最大値 ＋ 3 (注3)
	max_IIOP_local_init_requests	以下のいずれかの最大値 ・ max_IIOP_local_init_requests ・ 起動するコンシューマ／サプライヤのプロセス数の最大値 ＋ 3 (注3) × mixモデルのコンシューマ／サプライヤが1コネクションで同時に接続(送信)できるリクエスト数 ・ 起動するコンシューマ／サプライヤのプロセス数の最大値 ＋ 3 (注3) × pushモデルのコンシューマ／pullモデルのサプライヤが1コネクションで同時に接続(受信)できるリクエスト数

定義名	ステートメント	加算、設定値
	max_IIOP_resp_con (注1)	すべてのイベントチャネルに接続するコンシューマ・サプライヤの合計値 ＋ 1 (注4)
	max_IIOP_resp_requests	以下のいずれかの最大値 ・ max_IIOP_resp_conの加算値 × (mixモデルのコンシューマ／サプライヤが1コネクションで同時に接続(送信)できるリクエスト数 ＋ 1) ・ max_IIOP_resp_conの加算値 × (pushモデルのコンシューマ／pullモデルのサプライヤが1コネクションで同時に接続(受信)できるリクエスト数 ＋ 1)
	max_processes (注1)	起動するイベントチャネル・コンシューマ・サプライヤのプロセス数の合計値 ＋ 2 (注4)
	max_impl_rep_entries	(作成する静的生成イベントチャネルのプロセス数・動的生成イベントチャネルのプロセス数 × 2)の合計 (注5)
	period_receive_timeout	異常が発生した場合にコネクションを回収するまでのタイムアウト時間 (注6)

注1) Solaris64 Linux32/64

max_IIOP_resp_con、およびmax_processesを変更した場合は、システムパラメタを設定してください。

注2)

イベントチャネル側のシステムと、コンシューマ・サプライヤ側のシステムで加算値が異なります。システムにより以下の値を加算してください。

Windows32/64

－ イベントチャネル側(イベントチャネルを静的起動した場合)
「イベントチャネルグループの接続数(esmkchnlコマンドの-mオプションの設定値) (*1)」の総和
*1) “「イベントチャネルグループの接続数」× 2”の値が“256”よりも小さい場合は、“256”として計算してください。

－ イベントチャネル側(イベントファクトリを使用する場合)
「イベントチャネルの最大プロセス数(isinitコマンドでInterstage初期化時に設定したInterstage動作環境定義の定義“Event maximum Process”の指定値)」×「イベントチャネルの最大接続数(isinitコマンドでInterstage初期化時に設定したInterstage動作環境定義の定義“Event maximum Connection”の指定値) (*2)」＋ 17
*2) “「接続数」× 2”の値が“256”よりも小さい場合は、“256”として計算してください。

－ コンシューマおよびサプライヤ側
「サーバアプリケーション数(Pushモデルのコンシューマ数、Pullモデルのサプライヤ数)」×「スレッド最大多重度(OD_impl_instコマンドの-axオプションで指定するthr_conc_maximumの設定値)」

Solaris64 Linux32/64

－ イベントチャネル側(イベントチャネルを静的起動した場合)
「イベントチャネルグループの接続数(esmkchnlコマンドの-mオプションの設定値) (*3)」の総和
*3) “「イベントチャネルグループの接続数」＋ 16”の値が“256”よりも小さい場合は、“256”として計算してください。

－ イベントチャネル側(イベントファクトリを使用する場合)
「イベントチャネルの最大プロセス数(isinitコマンドでInterstage初期化時に設定したInterstage動作環境定義の定義“Event maximum Process”の指定値)」×「イベントチャネルの最大接続数(isinitコマンドでInterstage初期化時に設定したInterstage動作環境定義の定義“Event maximum Connection”の指定値) (*4)」＋ 17
*4) “「接続数」＋ 16”の値が“256”よりも小さい場合は、“256”として計算してください。

ー コンシューマおよびサプライヤ側
「サーバアプリケーション数(Pushモデルのコンシューマ数、Pullモデルのサプライヤ数)」×「スレッド初期多重度(OD_impl_instコマンドの-axオプションで指定するthr_conc_initの設定値)」

注3)

イベントチャネルを動作させる場合は、さらに“3”を加算してください。

注4)

イベントチャネル通信中にイベントサービス運用コマンドを実行する場合は、1を加算してください。

注5)

静的生成イベントチャネルのプロセス数は、esmkchnlコマンドまたはInterstage管理コンソールで作成した静的生成イベントチャネルグループ数です。
動的生成イベントチャネル(イベントファクトリを使用する場合)のプロセス数は、isinitコマンドでInterstage初期化時に設定したInterstage動作環境定義の定義“Event maximum Process”の指定値です。

注6)

以下の見積もり式を参考にして見積もった値を加算してください。

```
period_receive_timeout × 5 ＞ イベントデータの待ち合わせ時間(essetcnf／essetcnfchnlコマンドの“-wtime”の設定値) ＋ 20
```

イベントデータの待ち合わせ時間より先にperiod_receive_timeoutによるタイムアウトが発生した場合は、以下の現象が発生する可能性があります。

ー イベントデータがロストします。

ー エラーメッセージ“od10605”が出力されて、応答の送信が失敗します。

ー エラーメッセージ“es10033”(CODE=138)が出力されて、イベントチャネルが異常終了します。

なお、イベントデータの待ち合わせ時間には、“0”を指定しないでください。“0”を指定すると、イベントデータの待ち合わせ時間は無限となり、period_receive_timeoutによるタイムアウトが発生します。

EE

2.3.2.4 サーバマシン状態監視 Windows32 Linux32

サーバマシン状態監視機能を使用する場合は、以下の値を加算します。

監視サーバのチューニング

定義名	ステートメント	加算値
CORBAサービスの動作環境ファイル	max_processes (注)	定数: 1
	max_IIOP_resp_con (注)	
	max_exec_instance	定数: 4

注) Linux32
max_processes、max_IIOP_resp_conを変更した場合は、システムパラメタの設定が必要です。

被監視サーバのチューニング

定義名	ステートメント	加算値
CORBAサービスの動作環境ファイル	max_processes (注)	定数: 1
	max_IIOP_resp_con (注)	
	max_exec_instance	

注) Linux32
max_processes、max_IIOP_resp_conを変更した場合は、システムパラメタの設定が必要です。

2.4 環境変数について

以下にInterstageで使用する環境変数を示します。

環境変数名	用途
OD_HOME	CORBAサービスのインストールパスを指定します。
	例) Windows32/64 OD_HOME=Interstageのインストールフォルダ¥ODWIN Solaris64 OD_HOME=/opt/FSUNod Linux32/64 OD_HOME=/opt/FJSVod
OD_CODE_SET	コード変換を行う際のクライアントコード系を指定します。 サポートコードは以下のとおりです。 ・ SJIS(ShiftJIS) ・ EUC ・ UNICODE ・ SJISMS(ShiftJIS MS) ・ U90 ・ JEF_LOWER(JEF英小文字) ・ JEF_KANA(JEFカナ文字) ・ JEF_ASCII(JEFASCII) ・ UTF8
	例) OD_CODE_SET=EUC
TD_HOME	コンポーネントトランザクションサービスのインストールパスを指定します。
	例) Windows32/64 TD_HOME=Interstageのインストールフォルダ¥td Solaris64 TD_HOME=/opt/FSUNtd Linux32/64 TD_HOME=/opt/FJSVtd
OTS_HOME	データベース連携サービスのインストールパスを指定します。 OTSが動作するためにOTS_HOMEの設定は必須ではありません。
	例) Windows32/64 OTS_HOME=Interstageのインストールフォルダ¥ots Solaris64 OTS_HOME=/opt/FSUNots Linux32/64 OTS_HOME=/opt/FJSVots
OTS_SCROLL_SIZE	otstranlist、otspendlistコマンドを使用して、1画面でトランザクションリストを表示する行数を指定します。
	例) OTS_SCROLL_SIZE=30
	デフォルト値:20

環境変数名	用途
OTS_INVOKE_MODE	旧バージョンレベルのInterstage上のOTSシステム、またはリソース管理プログラムと分散トランザクション連携する場合に、OTSシステム、またはリソース管理プログラムが動作するマシン上で指定します。2以外の値は、指定できません。
	例) OTS_INVOKE_MODE=2
Solaris64 Linux32/64 ES_HOME	イベントサービスのインストールパスを指定します。
	例) ES_HOME=/opt/FJSVes
PORB_HOME	Portable-ORBのインストールパスを指定します。 (Portable-ORBの動作環境ファイル検索時にも使用)
	例) Windows32/64 PORB_HOME=Portable-ORBのインストールフォルダ Solaris64 Linux32/64 PORB_HOME=/opt/FJSVporb
IS_ISSTOP_MONITOR_TIMER	Interstageやワークユニットの停止が無応答になる現象が発生した際に、自動的にトラブル調査資料を採取するまでの監視時間(秒単位)を、デフォルト値から変更したい場合に指定します。 本環境変数は、以下の方法で設定してください。 ・ Windows32/64 ブート時に取得可能なシステム環境変数として設定します。環境変数を有効とするには、システム環境変数の設定後、オペレーティングシステムを再起動する必要があります。 ・ Solaris64 Linux32/64 Interstage管理コンソールでInterstageを起動する場合は、システムの環境変数として設定します。 ・ Solaris64 Linux32/64 isstartコマンドでInterstageを起動する場合は、コマンド実行前に設定します。 ・ Solaris64 Linux32/64 (RHEL6) オペレーティングシステム起動時の自動起動でInterstageを起動する場合は、RCプロシジャに設定します。 ・ Linux32/64 (RHEL7) オペレーティングシステム起動時の自動起動でInterstageを起動する場合は、以下のファイルに設定します。 /etc/opt/FJSVtd/systemd/system/service_startis.sh
	例) IS_ISSTOP_MONITOR_TIMER=360
	デフォルト値:300秒(5分)
	注)isstop／isstopwuコマンドの-tオプションに監視時間を指定して、Interstageやワークユニットを停止する場合、本環境変数の設定値よりコマンドの-tオプションの設定値が優先されます。

2.5 IPv6環境での運用について

本項では、IPv6環境での運用について説明します。

注意

・ IPv6/IPv4デュアルスタック環境での運用のみをサポートします。IPv4を無効にした場合の運用はサポートしません。

- グローバルアドレスおよびユニークローカルアドレスを使用した運用が可能です。リンクローカルアドレスについては、機能ごとにサポート有無が異なります。詳細は、以降の各機能の説明を参照してください。
- ホスト名指定によりIPv6運用を行う場合、そのホスト名をIPv6アドレスで名前解決できる必要があります。pingコマンドなどを使用して、対象ホスト名に対して正しくIPv6通信できることを事前に確認してください。
- 一時アドレス対応(プライバシ拡張)を有効にした環境での動作はサポートしません。一時アドレス対応(プライバシ拡張)の詳細については、各OSのマニュアルを参照してください。
- IPv6アドレスをログ・メッセージに出力する場合、設定ファイルなどに指定した値(入力値)の形式と異なる形式で出力される場合があります。
- IPv6環境において運用可能な機能は、Interstage管理コンソールを使用してIPv6通信を設定することもできます。設定方法の詳細については、Interstage管理コンソールのヘルプを参照してください。

■運用可能な機能

IPv6環境において、以下の機能が使用できます。表内で使用している略称については、「製品名称の略称について」を参照してください。

機能名		SJE	EE
JDK/JRE		○	○
Java EE 6		―	○
Java EE 7		○	○
J2EE	EJB	―	○
	JTS/JTA	―	○
マルチ言語サービス	CORBAサービス	○	○
	イベントサービス	―	○
	データベース連携サービス	―	○
Interstage HTTP Server		○	○
Interstage シングル・サインオン		○	○
Interstage ディレクトリサービス		○	○
SMEEコマンドによる証明書/鍵管理環境		○	○

○:使用可 ―:使用不可

JDK/JRE

JDK/JREを使用してIPv6通信を行うことが可能です。

Java EE 6

IPv6環境でJava EE 6を使用することが可能です。

注意

- リンクローカルアドレスは使用できません。

Java EE 7

IPv6環境でJava EE 7を使用することが可能です。

注意

- リンクローカルアドレスは使用できません。

J2EE

以下の機能において、IPv6通信を行うことが可能です。ただし、SSL連携機能を使用した場合はIPv6通信を行えません。

- EJB
- JTS/JTA Windows32/64 Linux32/64

ポイント

CORBAサービスの環境設定が必要です。
CORBAサービスの動作環境ファイル(config)において、“IP-version”に“v4-dual”または“v6”を設定し、Interstageを再起動してください。デフォルトは“v4-dual”です。詳細は、A.1 configを参照してください。

注意

- Linux32/64
  リンクローカルアドレスは使用できません。
- 入力値としてIPv4射影アドレスを使用することはできません。

マルチ言語サービス

以下の機能において、IPv6通信を行うことが可能です。ただし、SSL連携機能を使用した場合はIPv6通信を行えません。

- CORBAサービス
- イベントサービス
- データベース連携サービス

ポイント

CORBAサービスの環境設定が必要です。
CORBAサービスの動作環境ファイル(config)において、“IP-version”に“v4-dual”または“v6”を設定し、Interstageを再起動してください。デフォルトは“v4-dual”です。詳細については、“A.1 config”を参照してください。

注意

- Linux32/64
  リンクローカルアドレスは使用できません。
- 入力値としてIPv4射影アドレスを使用することはできません。

Interstage HTTP Server

IPv6環境でのHTTP/HTTPS通信を行うことが可能です。

注意

- Linux32/64
  リンクローカルアドレスは使用できません。
- 入力値としてIPv4射影アドレスを使用することはできません。

Interstage シングル・サインオン

IPv6環境でInterstage シングル・サインオンを使用することが可能です。

注意

- リンクローカルアドレスは使用できません。
- 入力値としてIPv4射影アドレスを使用することはできません。
- IPv6アドレスを直接指定することはできません。ホスト名、またはFQDNを指定してください。

Interstage ディレクトリサービス

IPv6環境でInterstage ディレクトリサービスを使用することが可能です。

注意

- リンクローカルアドレスは使用できません。
- 入力値としてIPv4射影アドレスを使用することはできません。
- IPv6アドレスを直接指定することはできません。ホスト名を指定してください。

SMEEコマンドによる証明書/鍵管理環境

cmgetcrlコマンド（CRLの取得）において、IPv6通信を行うことが可能です。

注意

- リンクローカルアドレスは使用できません。
- 入力値としてIPv4射影アドレスを使用することはできません。

2.6 ホスト情報(IPアドレス/ホスト名)の変更方法について

Interstageを運用しているサーバのホスト情報(IPアドレスやホスト名)を変更する場合には、1台のサーバで、Interstageの資源移出と資源移入を行うことで実施できます。資源移入時に、変更したいホスト情報を指定して資源移入の操作を行ってください。詳細については、“運用ガイド(基本編)”の“メンテナンス(資源のバックアップ/他サーバへの資源移行/ホスト情報の変更)”を参照してください。

第3章 システムのチューニング

この章では、システムチューニングについて説明します。

3.1 サーバ機能運用時に必要なシステム資源 Solaris64 Linux32/64

Interstageの各サービスの運用に必要となるシステム資源について説明します。
以下の表を参照し、使用する製品に応じて、以降に示すサービスのチューニングを行ってください。各機能の説明を参照する前に“■システムパラメタについて”を参照してください。表内で使用している略称については、「製品名称の略称について」を参照してください。

なお、システムパラメタを算出するためのExcelファイルがマニュアルパッケージの“ApplicationServer¥tuning”配下のサブフォルダに“ISAS-IPCtuning.xlsx”として格納されています。Microsoft(R) Excel 2007もしくは以降のバージョンのMicrosoft(R) Excelをお持ちの場合は“ISAS-IPCtuning.xlsx”を使用してシステムパラメタを算出することが可能です。使用方法などの詳細については、当該Excelファイル内の説明記事を参照してください。

	EE	SJE
CORBAサービスのシステム資源の設定	○	○
コンポーネントトランザクションサービスのシステム資源の設定	○	○
データベース連携サービスのシステム資源の設定	○	×
イベントサービスのシステム資源の設定 (注1)	○	×
J2EE互換のシステム資源の設定	○	○
Interstage HTTP Serverのシステム資源の設定	○	○
MessageQueueDirectorのシステム資源の設定 Linux32/64	○	×
Interstage シングル・サインオンのシステム資源の設定	○	○
Interstage ディレクトリサービスのシステム資源の設定	○	○
Interstage管理コンソールのシステム資源の設定 Linux32/64	○	○
Interstage統合コマンドのシステム資源の設定	○	○
Webサーバコネクタのシステム資源の設定 Windows32/64 Solaris64 Linux32/64	○	○

○:チューニングを行う必要があります。
×:チューニングを行う必要はありません(該当製品ではサービスが使用できないため)。

注1) Interstage JMSを使用する場合、イベントサービスのシステム環境を設定する必要があります。

注意

Interstage HTTP Server 2.2のシステム資源については、「Interstage HTTP Server 2.2 運用ガイド」の「チューニング」を参照してください。

■システムパラメタについて

◆システムパラメタの変更方法 Solaris64

IPC資源のパラメタに値を設定するために、以下のいずれかの方法を選択してください。

- **/etc/systemファイルの修正**

  /etc/systemファイルを編集し、必要なパラメタ値を変更します。変更後は、変更した値を反映するためにシステムをリブートしてください。なお、変更方法の詳細については、Solarisのドキュメントを参照してください。

注) Solaris 10以降において、この方法でシステムチューニングする場合、shmmaxおよびshmmniは以下の関係が成り立つような値を設定してください。

```
project.max-shm-memory ＝ shmmax × shmmni
```

- **資源制御(Solaris 10以降)**

  以下の手順で、パラメタ値を変更してください。

  1. Interstageの停止
     Interstageを停止してください。もしInterstage管理コンソールを使用するためのサービスを起動している場合はそれらのサービスも停止してください。
  2. user.rootプロジェクトとsystemプロジェクトのパラメタの変更
     projmodコマンドでuser.rootプロジェクトとsystemプロジェクトのパラメタの値を変更してください。以下に例を示します。

     ```
     projmod -s -K 'project.max-sem-ids=(privileged,155,deny)' user.root
     projmod -s -K 'project.max-sem-ids=(privileged,155,deny)' system
     ```

     変更の特権レベルをprivileged、設定したしきい値を超える要求があった場合のアクションをdenyとします。
  3. 値の反映
     newtaskコマンドで変更した値をシステム反映します。

     ```
     newtask -p user.root -c $$
     ```

  4. Interstageの起動
     Interstageを起動してください。必要に応じて、Interstage管理コンソールを使用するためのサービスを起動してください。

  資源制御の詳細については、Solarisのドキュメントを参照してください。

「種類」の意味

システムパラメタの表中の「種類」の意味は、以下のとおりです。

- 設定値
  必要数の条件に応じた値に変更してください。
- 加算値
  すでに設定されている値に、必要数を加算してください。

各パラメタの意味

システムパラメタの各パラメタ名と意味は、以下のとおりです。
なお、資源制御によるIPC資源のパラメタ設定は、Solaris 10以降で利用できます。

共用メモリ Solaris64

パラメタ	資源制御	意味
－	project.max-shm-memory	共用メモリの最大サイズ
shmmax	－	共用メモリの最大セグメントサイズ
shmmni	project.max-shm-ids	共用メモリIDの最大数

セマフォ Solaris64

パラメタ	資源制御	意味
semmni	project.max-sem-ids	セマフォIDの最大数

パラメタ	資源制御	意味
semmns	－	システム全体のセマフォの最大数
semvmx	－	セマフォに設定できる最大数
semmsl	process.max-sem-nsems	セマフォIDあたりのセマフォの最大数
semopm	process.max-sem-ops	セマフォコールあたりのセマフォ操作の最大数
semmnu	－	システム全体のセマフォ操作の取消記録グループ数
semume	－	プロセスあたりのセマフォ操作の取消記録の最大数

メッセージキュー Solaris64

パラメタ	資源制御	意味
msgmax		メッセージの最大サイズ
msgmnb	process.max-msg-qbytes	メッセージキュー上のメッセージの最大バイト数
msgmni	project.max-msg-ids	メッセージキューIDの最大数
msgtql	process.max-msg-messages	メッセージキューにあるメッセージの最大数

ファイルディスクリプタ Solaris64

パラメタ	資源制御	意味
rlim_fd_max	process.max-file-descriptor	ファイルディスクリプタの最大数

◆システムパラメタの変更方法 Linux32/64

/etc/sysctl.confを編集し、パラメタ値を変更します。変更後は“sysctl -p /etc/sysctl.conf”を実行するか、システムをリブートしてください。
変更方法の詳細については、OSのドキュメントを参照してください。

「種類」の意味

システムパラメタの表中の「種類」の意味は、以下のとおりです。

- 設定値
  必要数の条件に応じた値に変更してください。
- 加算値
  すでに設定されている値に、必要数を加算してください。

各パラメタの意味

システムパラメタの各パラメタ名と意味は、以下のとおりです。

共用メモリ Linux32/64

パラメタ	意味
kernel.shmall	システム全体の共用メモリの最大サイズ
kernel.shmmax	共用メモリの最大サイズ

パラメタ	意味
kernel.shmmni	共用メモリセグメントの最大数

セマフォ Linux32/64

セマフォの設定値は、各パラメタ値を以下の形式で指定します。

ー kernel.sem = *para1 para2 para3 para4*

パラメタ	意味
para1	セマフォIDあたりのセマフォの最大数
para2	システム全体のセマフォの最大数
para3	セマフォコールあたりのセマフォ操作の最大数
para4	セマフォIDの最大数

メッセージキュー Linux32/64

パラメタ	意味
kernel.msgmax	メッセージの最大サイズ
kernel.msgmnb	メッセージキュー上のメッセージの最大バイト数
kernel.msgmni	メッセージキューIDの最大数

ファイルディスクリプタ Linux32/64

パラメタ	意味
fs.file-max	システム全体のファイルディスクリプタの最大数

プロセス・スレッド Linux32/64

パラメタ	意味
kernel.threads-max	システム全体のプロセス・スレッドの最大数

◆リソース制限 Linux32/64

システムのリソースを制限するには、/etc/security/limits.confファイルを編集し、必要なパラメタ値を変更します。変更後は、変更した値を反映するためにシステムをリブートしてください。
変更方法の詳細については、OSのドキュメントを参照してください。

また、RHEL7の場合は各機能の説明も参照してください。

「種類」の意味

リソース制限の表中の「種類」の意味は、以下のとおりです。

- soft
  ソフトウェアリミット値を変更してください。
- hard
  ハードウェアリミット値を変更してください。

各パラメタの意味

リソース制限のパラメタ名と意味は、以下のとおりです。

ファイルディスクリプタ Linux32/64

パラメタ	意味
nofile	ユーザにおいてオープン可能なファイルディスクリプタの最大数

プロセス・スレッド Linux32/64

パラメタ	意味
nproc	ユーザにおいて作成可能なプロセス・スレッドの最大数

3.1.1 CORBAサービスのシステム資源の設定 Solaris64 Linux32/64

CORBAサービスを用いたシステムの運用時には、接続するクライアント/サーバ数、オブジェクト数などによりシステム資源を拡張する必要があります。ここでは、以下について説明します。

- システムパラメタ
  - CORBAサービス
  - インタフェースリポジトリ
  - ネーミングサービス
- プロセス・スレッド
- ファイルディスクリプタ

■システムパラメタ

一般的なCORBAサービスが使用する共用メモリ、セマフォ、メッセージキューのシステムパラメタのチューニングについて説明します。

CORBAサービスの他に共用メモリ、セマフォ、メッセージキューを使用するアプリケーションが存在する場合、そのアプリケーションが使用する資源にCORBAサービスの資源量を加算してください。

システムパラメタの変更方法や、各パラメタの意味については、“システムパラメタについて”を参照してください。

CORBAサービス

CORBAサービスで必要となるシステム資源について、以下に示します。

注意

各表に記述されているパラメタ名(max_IIOP_resp_conなど)は、CORBAサービスのconfigファイルで指定します。詳細については、“A.1 config”を参照してください。

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
shmmax	－	設定値	Solarisのドキュメントおよび“システムパラメタについて”を参照して値を決定してください。
－	project.max-shm-memory	加算値	max_IIOP_resp_con × 0.4K ＋ limit_of_max_IIOP_resp_con(注1) × 0.5K ＋ max_IIOP_resp_con_extend_number(注2) × 0.1K ＋ max_IIOP_resp_requests × 8K ＋ limit_of_max_IIOP_resp_requests(注1) × 3K ＋ max_IIOP_resp_requests_extend_number(注2) × 0.1K ＋ limit_of_number_of_common_buffer(注7) × 5K ＋ number_of_common_buffer_extend_number(注2) × 0.1K ＋

パラメタ	資源制御	種類	必要数
			max_processes × 0.6K ＋ max_exec_instance × 0.2K ＋ max_impl_rep_entries × 12K ＋ (max_IIOP_resp_con ＋ limit_of_max_IIOP_resp_con(注1) × 2) × max_impl_rep_entries × 0.004K ＋ max_bind_instances(注6) × 0.1K ＋ 3,200K 以上
			[SSL連携機能を使用する場合] 上記値 ＋ limit_of_max_IIOP_resp_con × 5K 以上
			[トレース機能を使用する場合] 上記値 ＋ max_processes × trace_size_per_process ＋ trace_size_of_daemon(注3) ＋ 20K 以上
			[スナップショット機能を使用する場合] 上記値 ＋ snap_size ＋ (max_impl_rep_entries ＋ max_processes) × 0.1K 以上
			[CORBAワークユニットを使用する場合] 上記値 ＋ (Buffer Size ＋ 0.2KB) × Buffer Number × “Buffer Size、Buffer Number(ワークユニット定義)を指定したアプリケーション数” (注5)
shmmni	project.max-shm-ids	加算値	max_IIOP_resp_con_extend_number(注2) ＋ max_IIOP_resp_requests_extend_number(注2) ＋ number_of_common_buffer_extend_number(注2) ＋ “Buffer Size、Buffer Number(ワークユニット定義)を指定したアプリケーション数” ＋ 13

注1)

limit_of_[パラメタ名]のデフォルト値は以下となります。0が指定された場合も、以下と同様になります。
[パラメタ名] × 1.3 (小数部分切り捨て)
isconfig.xmlファイルの定義項目AutoConfigurationModeにMANUALを指定し、自動拡張を行わない設定にした場合は、[パラメタ名]となります。

注2)

[パラメタ名]_extend_numberのデフォルト値は以下となります。0が指定された場合も、以下と同様になります。
(limit_of_[パラメタ名] － [パラメタ名]) ÷ [パラメタ名] (小数部分切り上げ)
isconfig.xmlファイルの定義項目AutoConfigurationModeにMANUALを指定し、自動拡張を行わない設定にした場合は、0となります。

注3)

デフォルトは以下です。0が指定された場合も、以下と同様になります。
trace_size_per_process × 32

注4)

デフォルトは以下です。0が指定された場合も、以下と同様になります。
max_IIOP_resp_requests × 0.2

注5)

Buffer Size、Buffer Numberを指定したCORBAワークユニット定義の中で、“(Buffer Size ＋ 0.2KB) × Buffer Number”の最大値に、“Buffer Size、Buffer Number(ワークユニット定義)を指定したアプリケーション数”を積算した値が該当します。
なお、“(Buffer Size ＋ 0.2KB) × Buffer Number”の最大値が 2,147,483,647より小さい値になるようにBuffer Size、Buffer Numberの値を設定してください。

注6)

デフォルトは以下です。0が指定された場合も、以下と同様になります。
max_processes × 1024 (計算結果が65,535を超えた場合は65,535)

注7)

デフォルトは以下です。0が指定された場合も、以下と同様になります。
limit_of_max_IIOP_resp_requests

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	以下の値のうち、最大値を指定します。 ・ max_IIOP_resp_con × 16K ＋ (max_IIOP_resp_con_extend_number(注1) ＋ 1) × 0.2K ＋ max_IIOP_resp_requests × 16K ＋ (max_IIOP_resp_requests_extend_number(注1) ＋ 1) × 0.2K ＋ max_impl_rep_entries × 6K ＋ max_bind_instances(注5) × 0.1K ＋ 100K 以上 ― [trace_use=yesの場合] 上記値 ＋ max_processes × trace_size_per_process ＋ trace_size_of_daemon(注2) ＋ 20K 以上 ― [snap_use=yesの場合] 上記値 ＋ snap_size ＋ 10K 以上 ・ number_of_common_buffer(注3) × 4K 以上 ＋ (number_of_common_buffer_extend_number(注1) ＋ 1) × 0.2K ・ max_exec_instance × 0.15KB 以上 ・ [Buffer Size、Buffer Number(ワークユニット定義)を指定したCORBAワークユニット、IJServer起動時] (Buffer Size ＋ 0.2K) × Buffer Number 以上 (注4)
kernel.shmmni	加算値	max_IIOP_resp_con_extend_number(注1) ＋ max_IIOP_resp_requests_extend_number(注1) ＋ number_of_common_buffer_extend_number(注1) ＋ “Buffer Size、Buffer Number(ワークユニット定義)を指定したアプリケーション数” ＋ 14

注1)

[パラメタ名]_extend_numberのデフォルト値は以下となります。0が指定された場合も、以下と同様になります。
limit_of_[パラメタ名]は、0が指定された場合は自動計算されます。計算式の詳細については、“config”を参照してください。
(limit_of_[パラメタ名] ― [パラメタ名]) ÷ [パラメタ名] (小数部分切り上げ)
isconfig.xmlファイルの定義項目AutoConfigurationModeにMANUALを指定し、自動拡張を行わない設定にした場合は、0となります。

注2)

デフォルトは以下です。0が指定された場合も、以下と同様になります。
trace_size_per_process × 32

注3)

デフォルトは以下です。0が指定された場合も、以下と同様になります。
max_IIOP_resp_requests × 0.2

注4)

Buffer Size、Buffer Numberを指定したワークユニット定義の中で、“(Buffer Size ＋ 0.2KB) × Buffer Number”の最大値が該当します。
なお、“(Buffer Size ＋ 0.2KB) × Buffer Number”の最大値が 2,147,483,647より小さい値になるようにBuffer Size、Buffer Numberの値を設定してください。

注5)

デフォルトは以下です。0が指定された場合も、以下と同様になります。
max_processes × 1024 (計算結果が65,535を超えた場合は65,535)

セマフォ Solaris64

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	以下の値のうち、最大値を指定します。 ・ 512 ・ max_IIOP_resp_con_extend_number(注1) × 5 ＋ max_IIOP_resp_requests_extend_number(注1) ＋ max_impl_rep_entries ＋ プロセスモードのCORBAサーバアプリケーションの起動プロセス数(注2) ＋ “Buffer Size、Buffer Number(ワークユニット定義)を指定したアプリケーション数” × 2 ＋ 100 以上
semmsl	process.max-sem-nsems	設定値	以下の値のうちの最大値以上の値を指定します。 ・ max_IIOP_resp_con ・ max_processes
semopm	process.max-sem-ops	設定値	50 以上

注1)

[パラメタ名]_extend_numberのデフォルト値は以下となります。0が指定された場合も、以下と同様になります。
(limit_of_[パラメタ名] － [パラメタ名]) ÷ [パラメタ名] (小数部分切り上げ)
isconfig.xmlファイルの定義項目AutoConfigurationModeにMANUALを指定し、自動拡張を行わない設定にした場合は、0となります。

注2)

起動プロセス数が分からない場合はmax_processesを指定してください。

セマフォ Linux32/64

パラメタ	種類	必要数
para1	設定値	以下の値のうちの最大値以上の値を指定します。 ・ max_IIOP_resp_con ・ max_processes
para2	加算値	limit_of_max_IIOP_resp_con(注1) × 4 ＋ max_IIOP_resp_con_extend_number(注2) ＋ max_IIOP_resp_requests_extend_number(注2) ＋ max_impl_rep_entries ＋ max_processes × 4 ＋ プロセスモードのCORBAサーバアプリケーションの起動プロセス数(注3) ＋ “Buffer Size、Buffer Number(ワークユニット定義)を指定したアプリケーション数” × 2 ＋ 14 以上

パラメタ	種類	必要数
		[トレース機能を使用する場合] 上記値 ＋ 1 以上
		[スナップショット機能を使用する場合] 上記値 ＋ 1 以上
		[SSL連携機能を使用する場合] 上記値 ＋ limit_of_max_IIOP_resp_con(注1) 以上
para3	設定値	50 以上
para4	加算値	以下の値のうち、最大値を指定します。 ・ 512 ・ max_IIOP_resp_con_extend_number(注2) × 5 ＋ max_IIOP_resp_requests_extend_number(注2) ＋ max_impl_rep_entries ＋ プロセスモードのCORBAサーバアプリケーションの起動プロセス数(注3) ＋ “Buffer Size、Buffer Number(ワークユニット定義)を指定したアプリケーション数” × 2 ＋ 100 以上

注1)

limit_of_[パラメタ名]のデフォルト値は以下となります。0が指定された場合も、以下と同様になります。
[パラメタ名] × 1.3 (小数部分切り捨て)
isconfig.xmlファイルの定義項目AutoConfigurationModeにMANUALを指定し、自動拡張を行わない設定にした場合は、[パラメタ名]となります。

注2)

[パラメタ名]_extend_numberのデフォルト値は以下となります。0が指定された場合も、以下と同様になります。
(limit_of_[パラメタ名] － [パラメタ名]) ÷ [パラメタ名] (小数部分切り上げ)
isconfig.xmlファイルの定義項目AutoConfigurationModeにMANUALを指定し、自動拡張を行わない設定にした場合は、0となります。

注3)

起動プロセス数が分からない場合はmax_processesを指定してください。

メッセージキュー Solaris64

パラメタ	資源制御	種類	必要数
msgmnb	process.max-msg-qbytes	設定値	32,768 以上
msgmni	project.max-msg-ids	加算値	512 以上

メッセージキュー Linux32/64

パラメタ	種類	必要数
kernel.msgmax	設定値	16,384 以上
kernel.msgmnb	設定値	32,768 以上
kernel.msgmni	加算値	512 以上

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場合に必要となるシステム資源を以下に示します。

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
shmmax	－	設定値	Solarisのドキュメントおよび“システムパラメタについて”を参照して値を決定してください。
－	project.max-shm-memory	加算値	[ログ採取時(EJBサービス未使用)] (logging memory size ＋ 16K) × 3 (注)
			EE [ログ採取時(EJBサービス使用)] (logging memory size ＋ 16K) × 4 (注)
shmmni	project.max-shm-ids	加算値	[EJBサービスを使用しない場合] 3
			EE [EJBサービスを使用する場合] 4

注)

“logging memory size”は、CORBAサービスのirconfigファイルで指定します。詳細については、“A.6 irconfig”を参照してください。

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	[ログ採取時] logging memory size ＋ 16K (注)

注)

“logging memory size”は、CORBAサービスのirconfigファイルで指定します。詳細については、“A.6 irconfig”を参照してください。

ネーミングサービス

ネーミングサービスにネーミングコンテキストを多数作成する場合に必要となるシステム資源を、以下に示します。

パラメタ	種類	必要数
nofile (注1) LimitNOFILE (注2) Linux32/64	加算値	ネーミングコンテキスト数 ＋ 16 以上

注1)

プロセス数あたりのオープン可能なファイル数です。
bash(Linux)または、ボーンシェルの場合はulimitコマンドを、Cシェルの場合はlimitコマンドを使用して、ネーミングサービスのプロセスが必要とするファイルをオープンできるだけの値を設定してください。コマンドの詳細については、OSのドキュメントを参照してください。

注2)

RHEL7の場合、環境定義用unitファイルにパラメタを設定します。環境定義用unitファイルに指定する起動用unitファイル名は、FJSVtd_start.serviceを指定します。
環境定義用unitファイルの詳細については、“付録I RHEL7のunitファイルでの環境定義”を参照してください。

■アプリケーションで使用するスレッド数・プロセス数

CORBAサービスでアプリケーションを実行する場合、アプリケーションから生成されるプロセス数・スレッド数が多くなる場合には、システムパラメタを変更する必要があります。
アプリケーションをマルチスレッドで作成している場合に、生成されるスレッド数の目安を以下に示します。

分類	スレッド数
CORBAサービス	25個 ＋ クライアントアプリケーションとの接続数 ＋ CORBAアプリケーションプロセス数
サーバアプリケーション	1プロセスにつき (6個 ＋ スレッド多重度数)
クライアントアプリケーション	1プロセスにつき8個

システムパラメタで、変更が必要となるものを以下に示します。

パラメタ	内容
max_nprocs Solaris64	システム全体で生成できるプロセス数
kernel.threads-max Linux32/64	システム全体で生成できるプロセス数とスレッド数の合計

システムパラメタ以外で、考慮するパラメタを以下に示します。

パラメタ	内容
stack (注1) Solaris64 Linux32/64 LimitSTACK (注3) Linux32/64	プロセスの最大スタックサイズ
nproc (注2) Linux32/64 LimitNPROC (注3) Linux32/64	使用できる最大のプロセス数

注1)

bashまたはボーンシェルの場合はulimitコマンドを、Cシェルの場合はlimitコマンドを使用して設定してください。この値にスレッド数を掛けた値が、プロセスのスタック領域として使用されます。1つのプロセスで使用可能なメモリを超過してスレッドは生成できないため、1つのプロセスが生成できるスレッド数は限界があります。
CORBAサーバアプリケーションおよびEJBアプリケーションのリクエスト処理多重度は“スレッド多重度 × プロセス多重度”で計算されます。1つのプロセスで使用可能なメモリサイズによってスレッド多重度を上げることができない場合は、プロセス多重度を上げることを検討してください。CORBAサーバアプリケーションのスレッド多重度・プロセス多重度については“リファレンスマニュアル(コマンド編)”の“OD_impl_inst”に記載されているproc_conc_max、thr_conc_init、thr_conc_maximumを参照してください。

EE

EJBアプリケーションのスレッド多重度については、“J2EE ユーザーズガイド(旧版互換)”－“EJBコンテナのチューニング”に記載されている“同時処理数”を参照してください。

注2) Linux32/64

bashまたはボーンシェルの場合はulimitコマンドを、Cシェルの場合はlimitコマンドを使用して設定してください。ユーザが生成するプロセス数とスレッド数の合計値以上の値に設定してください。

注3)

RHEL7の場合、環境定義用unitファイルにパラメタを設定します。環境定義用unitファイルに指定する起動用unitファイル名は、FJSVtd_start.serviceを指定します。
環境定義用unitファイルの詳細については、“付録I RHEL7のunitファイルでの環境定義”を参照してください。

■ファイルディスクリプタ数

CORBAサービスで多数のアプリケーションを動作させ(多端末接続時など)、使用するファイルディスクリプタ数がシステムのデフォルト値を超える場合は、システムパラメタにその値を設定してください。

Solaris64

パラメタ	内容
rlim_fd_cur (システムパラメタ)	使用するファイルディスクリプタ数がデフォルト値を超える場合に設定。

Linux32/64

システムパラメタで、変更が必要となるものを以下に示します。

パラメタ	内容
fs.file-max	システム全体のファイルディスクリプタの上限値。

Linux32/64

システムパラメタ以外で、考慮するパラメタを以下に示します。

パラメタ	内容
nofile (注1)	各ユーザのオープン可能なファイルディスクリプタの上限値

注1)

/etc/security/limits.conf ファイルを編集します。limits.conf ファイルの詳細については、オペレーティングシステムのドキュメントを参照してください。

3.1.2 コンポーネントトランザクションサービスのシステム資源の設定 Solaris64 Linux32/64

コンポーネントトランザクションサービスの動作時には、使用する機能によりシステム資源を拡張する必要があります。ここでは、以下について説明します。

- システムパラメタ
  - コンポーネントトランザクションサービスの基本機能
  - セション情報管理機能 Linux32
  - 性能監視ツール

注意

以降に示す値は、CORBAサービスの値を含んでいません。“3.1.1 CORBAサービスのシステム資源の設定”を参照し、必要な値を加算してください。

■システムパラメタ

コンポーネントトランザクションサービスが使用する共用メモリ、セマフォ、メッセージキューのシステムパラメタのチューニングについて説明します。
コンポーネントトランザクションサービスの基本機能のほかに各機能を使用する場合は、コンポーネントトランザクションサービスの基本機能の資源に各機能で使用する資源量を加算してください。
システムパラメタの変更方法や、各パラメタの意味については、“■システムパラメタについて”を参照してください。

コンポーネントトランザクションサービスの基本機能

コンポーネントトランザクションサービスの基本機能を使用する場合に必要となるシステム資源について、以下に示します。

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
―	project.max-shm-memory	加算値	EE ・システム規模がsmallの場合 36096468以上 ・システム規模がmoderateの場合 48500404以上 ・システム規模がlargeの場合 61728660以上 ・システム規模がsuperの場合 83718036以上 SJE ・システム規模がsmallの場合 35971540以上 ・システム規模がmoderateの場合 48257716以上 ・システム規模がlargeの場合 60543892以上 ・システム規模がsuperの場合 81355668以上
shmmax	―	設定値	・システム規模がsmallの場合 12498508以上 ・システム規模がmoderateの場合 23736108以上 ・システム規模がlargeの場合 34973708以上 ・システム規模がsuperの場合 54736908以上
shmmni	project.max-shm-ids	加算値	22

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	Linux32 [システム規模がsmallの場合] 12498508以上 [システム規模がmoderateの場合] 23736108以上 [システム規模がlargeの場合] 34973708以上

パラメタ	種類	必要数
		[システム規模がsuperの場合] 54736908以上
		Linux64 18,219,144 以上
kernel.shmmni	加算値	22

セマフォ Solaris64

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	29
semmsl	process.max-sem-nsems	設定値	12 以上
semopm	process.max-sem-ops	設定値	3 以上

セマフォ Linux32/64

パラメタ	種類	必要数
para1	設定値	12 以上
para2	加算値	21
para3	設定値	3 以上
para4	加算値	29

メッセージキュー Solaris64

パラメタ	資源制御	種類	必要数
msgmnb	process.max-msg-qbytes	設定値	4,572 ＋ (528 × 同時実行コマンド数) (注1)
msgmni	project.max-msg-ids	加算値	11
msgtql	process.max-msg-messages	設定値	15 ＋ 同時実行コマンド数 以上(注1)(注2)

EE

注1)

同時実行コマンド数とは、以下のコマンドを同時に実行した数のことです。

－ isstartwu、isstopwu、tdstartwu、tdstopwu、tdinhibitobj、tdpermitobj、tdmodifyprocnum、tdmodifywu

EE

注2)

AIM連携機能を使用する場合は、2,040を加算してください。

メッセージキュー Linux32/64

パラメタ	種類	必要数
kernel.msgmax	設定値	528 以上
kernel.msgmnb	設定値	4,572 ＋ (528 × 同時実行コマンド数) (注)
kernel.msgmni	加算値	11

EE

注)

同時実行コマンド数とは、以下のコマンドを同時に実行した数のことです。

- isstartwu、isstopwu、tdstartwu、tdstopwu、tdinhibitobj、tdpermitobj、tdmodifyprocnum、tdmodifywu

Linux32

Systemwalker Operation Manager／Interstage運用APIを使用して、以下の操作を行う場合は、同時に操作する回数が同時実行コマンド数となります。

- ワークユニットの起動／停止
- オブジェクト閉塞／閉塞解除

Linux64

Systemwalker Operation Managerを使用して、ワークユニットの起動／停止を行う場合は、同時に操作する回数が同時実行コマンド数となります。

セション情報管理機能 Linux32

セション情報管理機能を使用する場合に追加となるシステム資源について、以下に示します。

共用メモリ

パラメタ	種類	必要数
kernel.shmmni	加算値	1

セマフォ

パラメタ	種類	必要数
para2	加算値	1
para4	加算値	1

性能監視ツール

性能監視ツールを使用する場合に追加となるシステム資源については、“3.2 性能監視ツール使用時に必要なシステム資源”を参照してください。

3.1.3 データベース連携サービスのシステム資源の設定

データベース連携サービスの動作時には、利用するシステム形態によりシステム資源を拡張する必要があります。ここでは、以下について、利用するシステム形態ごとに説明します。

- システムパラメタ
  - ― OTSシステムのみが動作する場合
  - ― リソース管理プログラムのみが動作する場合
  - ― OTSシステムとリソース管理プログラムの両方が動作する場合

■システムパラメタ

データベース連携サービスが使用する共用メモリ、セマフォ、メッセージキューのシステムパラメタのチューニングについて、システム形態ごとに説明します。
システムパラメタの変更方法や、各パラメタの意味については、“■システムパラメタについて”を参照してください。

OTSシステムのみが動作する場合

OTSシステムのみが動作する場合に必要となるシステム資源について、以下に示します。

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
―	project.max-shm-memory	加算値	18,035,004 (注)
shmmax	―	設定値	17,830,204以上 (注)
shmmni	project.max-shm-ids	加算値	12

共用メモリ Linux32/64

パラメタ	種類	必要数	
		Linux32	Linux64
kernel.shmmax	設定値	17,830,204 (注)	18,035,004 (注)
kernel.shmmni	加算値	12	

注)

以下のファイルのデフォルト値で算出した値です。

― データベース連携サービスの環境定義のconfigファイル
OTS_TRACE_SIZE = 4096
RECOVERY_TRACE_SIZE = 4096
OBSERVE_TRACE_SIZE = 4096

― データベース連携サービスの環境定義のセットアップ情報ファイル
TRANMAX = 100
PARTICIPATE = 4

定義値を変更した場合は、以下の計算式で算出してください。

Linux32

必要数 ＝
(OTS_TRACE_SIZE ＋ RECOVERY_TRACE_SIZE ＋ OBSERVE_TRACE_SIZE) ×1,024 ＋ PARTICIPATE × TRANMAX × 2,048 ＋ TRANMAX × 284 ＋ 4,399,692

Solaris64 Linux64

必要数 ＝
(OTS_TRACE_SIZE ＋ RECOVERY_TRACE_SIZE ＋ OBSERVE_TRACE_SIZE) ×1,024 ＋ PARTICIPATE × TRANMAX × 2,560 ＋ TRANMAX × 284 ＋ 4,399,692

セマフォ Solaris64

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	8
semmsl	process.max-sem-nsems	設定値	12 以上
semopm	process.max-sem-ops	設定値	3 以上

セマフォ Linux32/64

パラメタ	種類	必要数
para1	設定値	12 以上
para2	加算値	24
para3	設定値	3 以上
para4	加算値	8

メッセージキュー Solaris64

パラメタ	資源制御	種類	必要数
msgmnb	process.max-msg-qbytes	設定値	4,572 以上
msgmni	project.max-msg-ids	加算値	3
msgtql	process.max-msg-messages	設定値	2,040以上

メッセージキュー Linux32/64

パラメタ	種類	必要数
kernel.msgmax	設定値	528 以上
kernel.msgmnb	設定値	4,572 以上
kernel.msgmni	加算値	3

リソース管理プログラムのみが動作する場合

リソース管理プログラムのみが動作する場合に必要となるシステム資源について、以下に示します。

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
－	project.max-shm-memory	加算値	12,840,416 (注)
shmmax	－	設定値	12,840,416以上 (注)
shmmni	project.max-shm-ids	加算値	リソース管理プログラムの種類 × 11

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	12,840,416 (注)
kernel.shmmni	加算値	リソース管理プログラムの種類 × 11

注)

リソース管理プログラムの種類が1つの場合で、かつ以下のファイルのデフォルト値で算出した値です。

－ データベース連携サービスの環境定義のconfigファイル
RESOURCE_TRANMAX = 10
RESOURCE_TRACE_SIZE = 4096
OBSERVE_TRACE_SIZE = 4096

－ リソース定義ファイル
OTS_RMP_PROC_CONC = 5

－ データベース連携サービスの環境定義のセットアップ情報ファイル
TRANMAX = 100

定義値を変更した場合は、以下の計算式で算出してください。
必要数 ＝
(RESOURCE_TRACE_SIZE ＋ OBSERVE_TRACE_SIZE) × 1,024 ＋
(TRANMAX ＋ 1) × 332 ＋
((リソース管理プログラムの種類 × RESOURCE_TRANMAX ×
OTS_RMP_PROC_CONC) × (144 ＋ 332)) ＋ 4,394,476

セマフォ Solaris64

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	リソース管理プログラムの種類 × 7

セマフォ Linux32/64

パラメタ	種類	必要数
para4	加算値	リソース管理プログラムの種類 × 7

OTSシステムとリソース管理プログラムの両方が動作する場合

OTSシステムとリソース管理プログラムの両方が動作する場合に必要となるシステム資源について、以下に示します。

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
－	project.max-shm-memory	加算値	17,830,204(注)
shmmax	－	設定値	17,830,204以上 (注)
shmmni	project.max-shm-ids	加算値	12 ＋ リソース管理プログラムの種類 × 11

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	17,830,204 (注)
kernel.shmmni	加算値	12 ＋ リソース管理プログラムの種類 × 11

注)

リソース管理プログラムの種類が1つの場合で、かつデフォルト値で算出した値です。
定義値を変更した場合は、以下の計算式で算出してください。
必要数 ＝
OTSシステムのみが動作する場合の必要数 ＋
リソース管理プログラムのみが動作する場合の必要数 － 4,915,600

セマフォ Solaris64

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	8 ＋ リソース管理プログラムの種類 × 7
semmsl	process.max-sem-nsems	設定値	12 以上

パラメタ	資源制御	種類	必要数
semopm	process.max-sem-ops	設定値	3 以上

セマフォ Linux32/64

パラメタ	種類	必要数
para1	設定値	12 以上
para2	加算値	24
para3	設定値	3 以上
para4	加算値	8 ＋ リソース管理プログラムの種類 × 7

メッセージキュー Solaris64

パラメタ	資源制御	種類	必要数
msgmnb	process.max-msg-qbytes	設定値	4,572 以上
msgmni	project.max-msg-ids	加算値	3
msgtql	process.max-msg-messages	設定値	2,040以上

メッセージキュー Linux32/64

パラメタ	種類	必要数
kernel.msgmax	設定値	528 以上
kernel.msgmnb	設定値	4,572 以上
kernel.msgmni	加算値	3

3.1.4 イベントサービスのシステム資源の設定 Solaris64 Linux32/64

イベントサービスを用いたシステムの運用時には、チャネル数、接続するコンシューマ/サプライヤ数などによりシステム資源を拡張する必要があります。ここでは、以下について説明します。

- システムパラメタ

注意

以降に示す値は、CORBAサービスの値を含んでいません。“3.1.1 CORBAサービスのシステム資源の設定”を参照し、必要な値を加算してください。

■システムパラメタ

一般的なイベントサービスが使用する共用メモリ、セマフォ、メッセージキューのシステムパラメタのチューニングについて説明します。
イベントサービスの他に共用メモリ、セマフォ、メッセージキューを使用するアプリケーションが存在する場合、そのアプリケーションが使用する資源にイベントサービスの資源量を加算してください。
システムパラメタの変更方法や、各パラメタの意味については、“システムパラメタについて”を参照してください。

注意

Solaris64

揮発チャネル運用と不揮発チャネル運用を併用している場合は、不揮発チャネル運用の必要数を使用してください。

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
－	project.max-shm-memory	加算値	[揮発チャネル運用の場合] 以下の値を加算します。 ・ 1,040 × イベントチャネル最大作成数(注1) ＋ 600K ・ traceconfigファイルのtrace_sizeパラメタに指定したバイト数
			[不揮発チャネル運用の場合] 以下の値を加算します。 ・ 1,040 × イベントチャネル最大作成数(注1) ＋ 184 × 同時実行可能なグローバルトランザクション数(注2) ＋ 600K ・ 17 × 1,024 × 1,024 ＋ 576 × トランザクションの多重度 ＋ 88 × (システム用データ格納域の数 ＋ イベントデータ用データ格納域の数) ＋ ユニットで使用する共用メモリサイズ × 1,024 × 1,024 (注3) ・ traceconfigファイルのtrace_sizeパラメタに指定したバイト数
shmmax	－	設定値	[揮発チャネル運用の場合] 以下の値のうち、最大値を指定します。 ・ 1,040 × イベントチャネル最大作成数(注1) ＋ 600K ・ traceconfigファイルのtrace_sizeパラメタに指定したバイト数
			[不揮発チャネル運用の場合] 以下の値のうち、最大値を指定します。 ・ 1,040 × イベントチャネル最大作成数(注1) ＋ 600K ・ 17 × 1,024 × 1,024 ＋ 576 × トランザクションの多重度 ＋ 88 × (システム用データ格納域の数 ＋ イベントデータ用データ格納域の数) ＋ ユニットで使用する共用メモリサイズ × 1,024 × 1,024 (注2)

パラメタ	資源制御	種類	必要数
			・ traceconfigファイルのtrace_sizeパラメタに指定したバイト数
shmmni	project.max-shm-ids	加算値	[揮発チャネル運用の場合] 4
			[不揮発チャネル運用の場合] ユニット数 × 100 以上

注1)

イベントチャネル最大作成数 ＝
静的生成イベントチャネル最大作成数 ＋ 動的生成イベントチャネル最大作成数

注2)

各値は、ユニット作成コマンド(esmkunitでユニットを作成した際、ユニット定義の以下の項目に指定した値です。
なお、複数のユニットを使用する場合は、それぞれのユニットに対して、算出してください。

	ユニット定義の項目
トランザクションの多重度	tranmaxの設定値
システム用データ格納域の数	sysqnumの設定値
イベントデータ用データ格納域の数	userqnumの設定値
ユニットで使用する共用メモリサイズ	shmmaxの設定値(42より小さい場合、42)

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	[揮発チャネル運用の場合] 2,064 × イベントチャネル最大作成数(注1) ＋ 600K ＋traceconfigファイルのtrace_sizeパラメタに指定したバイト数 ＋ 3,328
		[不揮発チャネル運用の場合] 以下の値のうち、最大値を指定します。 ・ 2,064 × イベントチャネル最大作成数(注1) ＋ 184 × 同時実行可能なグローバルトランザクション数(注2) ＋ 600K ＋traceconfigファイルのtrace_sizeパラメタに指定したバイト数 ＋ 3,328 ・ 134,217,728 以上 ・ ユニットで使用する共用メモリサイズ(複数ユニットがある場合、最大値)(注3)
kernel.shmmni	加算値	[揮発チャネル運用の場合] ・ プロセス単位で内部トレースを採取する(traceconfigファイルのtrace_buffer=process)場合 4 ＋ イベントチャネルのプロセス数(注4) ・ イベントサービス単位で内部トレースを採取する(traceconfigファイルのtrace_buffer=system)場合 4
		[不揮発チャネル運用の場合]

パラメタ	種類	必要数
		・ プロセス単位で内部トレースを採取する(traceconfigファイルのtrace_buffer=process)場合 100 以上の値 (ユニット単位に加算) ＋ イベントチャネルのプロセス数(注4) ・ イベントサービス単位で内部トレースを採取する(traceconfigファイルのtrace_buffer=system)場合 100 以上の値 (ユニット単位に加算)

注1)

イベントチャネル最大作成数 ＝
静的生成イベントチャネル最大作成数 ＋ 動的生成イベントチャネル最大作成数

注2)

同時実行可能なグローバルトランザクション数は、essetcnfコマンドでイベントサービスの構成情報を設定した際、-gtrnmaxオプションに指定した値です。

注3)

ユニットで使用する共用メモリサイズは、esmkunitコマンドでユニットを作成した際、ユニット定義ファイルの項目「shmmax」に指定した値です。

注4)

イベントチャネルのプロセス数 ＝
静的イベントチャネルグループ数 ＋ 動的イベントチャネルのプロセス数

動的イベントチャネルのプロセス数:
essetupコマンドでイベントサービスをセットアップした際、-pオプションに指定した値です。
ノーティフィケーションサービスを使用している場合は、「動的イベントチャネルのプロセス数 × 2」としてください。

セマフォ Solaris64

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	3

セマフォ Linux32/64

パラメタ	種類	必要数
para1	設定値	29
para2	加算値	[揮発チャネル運用の場合] 6 以上
		[不揮発チャネル運用の場合] ユニット数 × 29 ＋ 13 以上
para3	設定値	29
para4	加算値	ユニット数 × 256

メッセージキュー Solaris64

パラメタ	資源制御	種類	必要数
msgmnb	process.max-msg-qbytes	設定値	[不揮発チャネル運用の場合] 4,096 以上
msgmni	project.max-msg-ids	加算値	[揮発チャネル運用の場合] 2
			[不揮発チャネル運用の場合] ユニット数 × 3 ＋ 2
msgtql	process.max-msg-messages	設定値	[不揮発チャネル運用の場合] 以下をユニットごとに求めた合計値 6 ＋ (ユニット定義ファイルのtranmax値 × 4 × 1.3)

メッセージキュー Linux32/64

パラメタ	種類	必要数
kernel.msgmax	設定値	[不揮発チャネル運用の場合] 2,048 以上
kernel.msgmnb	設定値	[不揮発チャネル運用の場合] 4,096 以上
kernel.msgmni	加算値	[不揮発チャネル運用の場合] ユニット数 × 9

3.1.5 J2EE互換のシステム資源の設定 Solaris64 Linux32/64

IJServerまたはEJBサービスでは以下の機能を使用する場合に、システム資源を拡張する必要があります。ここでは、以下について説明します。

- システムパラメタ

■システムパラメタ

IJServerまたはEJBサービスを使用する際には、以下のシステムパラメタのチューニングを行ってください。
ワークユニット定義の通信バッファ数(Buffer Number)、通信バッファ長(BufferSize)を指定する場合は、“3.1.1 CORBAサービスのシステム資源の設定”についても参照してください。
システムパラメタの変更方法や、各パラメタの意味については、“■システムパラメタについて”を参照してください。

メッセージキュー Solaris64

パラメタ	資源制御	種類	必要数
msgmnb	process.max-msg-qbytes	設定値	4,096 以上
msgmni	project.max-msg-ids	加算値	2 以上
msgtql	process.max-msg-messages	設定値	1,024 以上

メッセージキュー Linux32/64

パラメタ	種類	必要数
kernel.msgmax	設定値	4,096 以上
kernel.msgmnb	設定値	4,096 以上
kernel.msgmni	加算値	2 以上

3.1.6 Interstage HTTP Serverのシステム資源の設定

Interstage HTTP Serverを用いたシステムの運用時には、システム資源を拡張する必要があります。ここでは、以下について説明します。

- システムパラメタ Linux32/64
- ファイルディスクリプタ
- プロセス数 Linux32/64

参照

- システムパラメタの変更方法や、パラメタの意味については、“システムパラメタについて”を参照してください。
- リソース制限(/etc/security/limits.conf)の変更方法や、パラメタの意味については、“リソース制限”を参照してください。

■システムパラメタ Linux32/64

以下の表を参照して、Interstage HTTP Serverを運用するために必要なシステムパラメタを算出し、本値がシステムのデフォルト値を超える場合、本値を設定してください。

セマフォ

パラメタ	種類	必要数
para1	設定値	1 以上
para2	加算値	Webサーバ数 × 2
para3	設定値	1 以上
para4	加算値	Webサーバ数 × 2

■ファイルディスクリプタ数 Solaris64

Interstage HTTP Serverを運用するために必要なファイルディスクリプタ数は、Webサーバで使用している機能、および環境定義ファイル(httpd.conf)の定義内容に応じて異なります。

以下の表を参照して必要なファイルディスクリプタ数を算出し、本値がシステムのデフォルト値を超える場合、本値を設定してください。

パラメタ	種類	必要数
rlim_fd_cur	設定値	130 以上
		[以下の機能を使用する場合] 上記に以下の値を加算してください。 ・ 基本認証機能:1 ・ オンライン照合機能:1 ・ SSL運用:21 ・ プロキシ機能:1 ・ CGI機能:5（注）
		[環境定義ファイル(httpd.conf)に以下のディレクティブを設定している場合] 上記に以下の値を加算してください。 ・ CustomLog("|コマンド実行文"指定):ディレクティブ数 × 10 ・ CustomLog(ファイル名指定):ディレクティブ数 ・ ErrorLog:ディレクティブ数 ・ Listen:ディレクティブ数

注）

動作するCGIプログラム内で必要となるファイルディスクリプタの数を別途加算してください。

■ファイルディスクリプタ数 Linux32/64

Interstage HTTP Serverを運用するために必要なファイルディスクリプタ数は、Webサーバで使用している機能、および環境定義ファイル(httpd.conf)の定義内容に応じて異なります。

以下の表を参照して必要なファイルディスクリプタ数を算出してください。
リソース制限（/etc/security/limits.conf）については、本値がシステムのデフォルト値を超える場合、本値を設定してください。
システムパラメタおよびリソース制限（unit）については、ファイルディスクリプタ数が不足する現象が発生した場合、本値を設定してください。

パラメタ	種類	必要数
[リソース制限(/etc/security/limits.conf)] nofile	設定値	20 以上
		[以下の機能を使用する場合] 上記に以下の値を加算してください。 ・ 基本認証機能:1 ・ オンライン照合機能:1 ・ SSL運用:21 ・ プロキシ機能:1 ・ CGI機能:5 (注1)
		[環境定義ファイル(httpd.conf)に以下のディレクティブを設定している場合] 上記に以下の値を加算してください。 ・ CustomLog("|コマンド実行文"指定):ディレクティブ数 × 2 ・ CustomLog(ファイル名指定):ディレクティブ数

パラメタ	種類	必要数
		・ ErrorLog:ディレクティブ数 ・ Listen:ディレクティブ数
[システムパラメタ] fs.file-max	加算値 (注2)	上記と同じ値
[リソース制限(unit)] LimitNOFILE (注3)	設定値 (注2)	上記と同じ値

注1)

動作するCGIプログラム内で必要となるファイルディスクリプタの数を別途加算してください。

注2)

複数のWebサーバを作成している場合は、Webサーバごとに必要数を加算してください。

注3)

RHEL7を使用して、オペレーティングシステム起動時にWebサーバを自動起動する場合、環境定義用unitファイル(FJSVihs_start.service)を作成し、本パラメタの値を以下の例のように記述します。環境定義用unitファイルの定義詳細や有効化手順については、"付録I RHEL7のunitファイルでの環境定義"を参照してください。

例

FJSVihs_start.serviceの記述例

```
.include /usr/lib/systemd/system/FJSVihs_start.service

[Service]
LimitNOFILE=値
```

■プロセス数 Linux32/64

以下の表を参照して必要なプロセス数を算出してください。
リソース制限(/etc/security/limits.conf)については、本値がシステムのデフォルト値を超える場合、本値を設定してください。
システムパラメタおよびリソース制限(unit)については、プロセス数が不足する現象が発生した場合、本値を設定してください。

パラメタ	種類	必要数
[リソース制限(/etc/security/limits.conf)] nproc	設定値 (注1) (注2)	m 以上
		[以下の機能を使用する場合] 上記に以下の値を加算してください。 ・ CGI機能: (注3)
[システムパラメタ] kernel.threads-max	加算値 (注2)	m ＋ 2 以上
		[以下の機能を使用する場合] 上記に以下の値を加算してください。 ・ SSL運用:1 ・ CGI機能: (注3)
		[環境定義ファイル(httpd.conf)に以下のディレクティブ("|コマンド実行文"指定)を設定している場合] 上記に以下の値を加算してください。 ・ CustomLog:ディレクティブ数 ・ ErrorLog:ディレクティブ数

パラメタ	種類	必要数
		・ IHSTraceLog：ディレクティブ数
[リソース制限(unit)] LimitNPROC (注4)	設定値 (注2)	上記と同じ値

m：環境定義ファイル(httpd.conf)のMaxClientsディレクティブの設定値

注1）

環境定義ファイル(httpd.conf)のUserディレクティブで設定したユーザに対する値を設定してください。

注2）

複数のWebサーバを作成している場合は、Webサーバごとに必要数を加算してください。

注3）

動作するCGIプログラム内で必要となるプロセス数・スレッド数を別途加算してください。

注4）

RHEL7を使用して、オペレーティングシステム起動時にWebサーバを自動起動する場合、環境定義用unitファイル（FJSVihs_start.service）を作成し、本パラメタの値を以下の例のように記述します。環境定義用unitファイルの定義詳細や有効化手順については、“付録I RHEL7のunitファイルでの環境定義”を参照してください。

例

FJSVihs_start.serviceの記述例

```
.include /usr/lib/systemd/system/FJSVihs_start.service

[Service]
LimitNPROC=値
```

3.1.7 MessageQueueDirectorのシステム資源の設定 Linux32/64

MessageQueueDirectorを用いたシステムの運用時には、MQDのシステム数およびMQDの上位サービスの種類などによりシステム資源を拡張する必要があります。ここでは、以下について説明します。

- システムパラメタ

■システムパラメタ

MessageQueueDirectorが使用するシステムパラメタのチューニングについて説明します。
システムパラメタの変更方法や、各パラメタの意味については、“■システムパラメタについて”を参照してください。

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	134,217,728 以上
kernel.shmmni	加算値	100 × MQDシステム数

セマフォ Linux32/64

パラメタ	種類	必要数
para1	設定値	29

パラメタ	種類	必要数
para2	加算値	MQDシステム数 × 29
para3	設定値	29
para4	加算値	MQDシステム数 × 256

メッセージキュー Linux32/64

パラメタ	種類	必要数
kernel.msgmax	設定値	16,384 以上
kernel.msgmnb	設定値	32,768 以上
kernel.msgmni	加算値	9 × MQDシステム数

3.1.8 Interstage シングル・サインオンのシステム資源の設定 Solaris64 Linux32/64

Interstage シングル・サインオンを用いたシステムの運用時には、システム資源を拡張する必要があります。ここでは、以下について説明します。

- システムパラメタ
  - Interstage シングル・サインオンのリポジトリサーバ機能を使用する際のシステムパラメタのチューニング
  - Interstage シングル・サインオンの認証サーバ機能を使用する際のシステムパラメタのチューニング
  - Interstage シングル・サインオンの業務サーバ機能を使用する際のシステムパラメタのチューニング

■システムパラメタ

Interstage シングル・サインオンが使用するシステムパラメタのチューニングについて説明します。
システムパラメタの変更方法や、各パラメタの意味については、“■システムパラメタについて”を参照してください。

Linux32/64

なお、以降に示す値は、Interstage HTTP Server、およびInterstage HTTP Server 2.2で必要な値を含んでいません。以下を参照し、必要な値を加算してください。

- Interstage HTTP Serverの場合
  “3.1.6 Interstage HTTP Serverのシステム資源の設定”
- Interstage HTTP Server 2.2の場合
  “Interstage HTTP Server 2.2 運用ガイド”の“チューニング”－“運用時に必要なシステム資源(Linux)”

Interstage シングル・サインオンのリポジトリサーバ機能を使用する際のシステムパラメタのチューニング

リポジトリサーバでは、“Interstage ディレクトリサービス”を使用しています。“Interstage ディレクトリサービス”で必要としているチューニングについては、“3.1.9 Interstage ディレクトリサービスのシステム資源の設定 ”を参照してください。

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
shmmax	－	設定値	Solarisのドキュメントおよび“■システムパラメタについて”を参照して値を決定してください。
－	project.max-shm-memory	加算値	26,000,000 ＋ (164 × サイト定義数(注1)) ＋ (ロール数(注2) ＋ ロールセット数(注3) ＋ ロールセット数(注3) × ロール数(注2)) × 1,024 × 3 (注4)
shmmni	project.max-shm-ids	加算値	13

注1)

SSOリポジトリの保護リソースに登録したサイト定義の総数

注2)

SSOリポジトリに定義したロールの総数

注3)

SSOリポジトリに定義したロールセットの総数

注4)

ユーザ情報を登録するディレクトリサービスにActive Directoryを使用し、シングル・サインオンの拡張スキーマを使用しない場合は、運用に応じて以下の式で見積もった値を加算してください。
Active Directoryのロール/ロールセットに使用する属性の総数 × 524 × 3

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	12,800,000 ＋ (164 × サイト定義数(注1)) ＋ (ロール数(注2) ＋ ロールセット数(注3) ＋ ロールセット数(注3) × ロール数(注2)) × 1,024 以上(注4)
kernel.shmmni	加算値	13

注1)

SSOリポジトリの保護リソースに登録したサイト定義の総数

注2)

SSOリポジトリに定義したロールの総数

注3)

SSOリポジトリに定義したロールセットの総数

注4)

ユーザ情報を登録するディレクトリサービスにActive Directoryを使用し、シングル・サインオンの拡張スキーマを使用しない場合は、運用に応じて以下の式で見積もった値を加算してください。
Active Directoryのロール/ロールセットに使用する属性の総数 × 524

セマフォ Solaris64

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	9

セマフォ Linux32/64

パラメタ	種類	必要数
para2	加算値	9
para4	加算値	9

ファイルディスクリプタ Solaris64

パラメタ	資源制御	種類	必要数
rlim_fd_max	process.max-file-descriptor	設定値	1024以上

ファイルディスクリプタ Linux32/64

パラメタ	種類	必要数
nofile	hard	1024以上

Interstage シングル・サインオンの認証サーバ機能を使用する際のシステムパラメタのチューニング

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
shmmax	－	設定値	Solarisのドキュメントおよび“■システムパラメタについて”を参照して値を決定してください。
－	project.max-shm-memory	加算値	27,000,000 ＋ パス定義の総数(注1) × 2,048
shmmni	project.max-shm-ids	加算値	11

注1)

SSOリポジトリの保護リソースに登録した各サイト定義に定義したパス定義の総数

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	14,000,000 ＋ パス定義の総数(注) × 2,048 以上
kernel.shmmni	加算値	11

注)

SSOリポジトリの保護リソースに登録した各サイト定義に定義したパス定義の総数

セマフォ Solaris64

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	10

セマフォ Linux32/64

パラメタ	種類	必要数
para2	加算値	10
para4	加算値	10

ファイルディスクリプタ Solaris64

パラメタ	資源制御	種類	必要数
rlim_fd_max	process.max-file-descriptor	設定値	1024以上

ファイルディスクリプタ Linux32/64

パラメタ	種類	必要数
nofile	hard	1024以上

Interstage シングル・サインオンの業務サーバ機能を使用する際のシステムパラメタのチューニング

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
shmmax	－	設定値	Solarisのドキュメントおよび“■システムパラメタについて”を参照して値を決定してください。
－	project.max-shm-memory	加算値	(13,000,000 ＋ (1 + キャッシュサイズ(注1)) × 1024 × キャッシュ数(注2) + (ロール数(注3) ＋ 拡張ユーザ情報数(注4)＋ 1) × パス定義の最大数(注5) × 1,200 × 2) × 業務サーバ数
shmmni	project.max-shm-ids	加算値	10 × 業務サーバ数

注1)

セションの管理を行う運用の場合、業務サーバでキャッシュする利用者の認証情報サイズ(Kバイト)を設定します。セションの管理を行わない運用の場合は、0で計算してください。
キャッシュサイズについては、“G.4 業務サーバを構築する場合のチューニング”を参照してください。

注2)

セションの管理を行う運用の場合、システムの同時アクセス最大数以上を設定します。セションの管理を行わない運用の場合は、0で計算してください。
キャッシュ数については、“G.4 業務サーバを構築する場合のチューニング”を参照してください。

注3)

SSOリポジトリに定義したロールの総数

注4)

業務アプリケーションに対して通知するユーザ情報の数
セションの管理を行う運用の場合、リポジトリサーバのInterstage管理コンソールの以下に設定されている属性名の数を設定します。セションの管理を行わない運用の場合は、0で計算してください。
[システム] > [セキュリティ] > [シングル・サインオン] > [認証基盤] > [リポジトリサーバ] > [環境設定] > [リポジトリサーバ詳細設定[表示]] > [業務システムに通知する情報] > [拡張ユーザ情報]

注5)

SSOリポジトリの保護リソースに登録した各サイト定義の保護パスに設定されているパス定義の最大数

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	13,000,000 ＋ (1 + キャッシュサイズ(注1)) × 1024 × キャッシュ数(注2) + (ロール数(注3) ＋ 拡張ユーザ情報数(注4)＋ 1) × パス定義の最大数(注5) × 1,200
kernel.shmmni	加算値	10 × 業務サーバ数

注1)

セションの管理を行う運用の場合、業務サーバでキャッシュする利用者の認証情報サイズ(Kバイト)を設定します。セションの管理を行わない運用の場合は、0で計算してください。
キャッシュサイズについては、“G.4 業務サーバを構築する場合のチューニング”を参照してください。

注2)

セションの管理を行う運用の場合、システムの同時アクセス最大数以上を設定します。セションの管理を行わない運用の場合は、0で計算してください。
キャッシュ数については、“G.4 業務サーバを構築する場合のチューニング”を参照してください。

注3)

SSOリポジトリに定義したロールの総数

注4)

業務アプリケーションに対して通知するユーザ情報の数
セションの管理を行う運用の場合、リポジトリサーバのInterstage管理コンソールの以下に設定されている属性名の数を設定します。セションの管理を行わない運用の場合は、0で計算してください。
[システム] > [セキュリティ] > [シングル・サインオン] > [認証基盤] > [リポジトリサーバ] > [環境設定] > [リポジトリサーバ詳細設定[表示]] > [業務システムに通知する情報] > [拡張ユーザ情報]

注5)

SSOリポジトリの保護リソースに登録した各サイト定義の保護パスに設定されているパス定義の最大数

セマフォ Solaris64

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	7 × 業務サーバ数

セマフォ Linux32/64

パラメタ	種類	必要数
para2	加算値	7 × 業務サーバ数
para4	加算値	7 × 業務サーバ数

3.1.9 Interstage ディレクトリサービスのシステム資源の設定 Solaris64 Linux32/64

Interstage ディレクトリサービスを用いたシステムの運用時には、システム資源を拡張する必要があります。ここでは、以下について説明します。

- システムパラメタ
  - Interstage ディレクトリサービスの運用
  - Symfoware Serverの運用
  - Oracleデータベースの運用

■システムパラメタ

Interstage ディレクトリサービスが使用する共用メモリ、セマフォ、メッセージキューのシステムパラメタのチューニングについて説明します。

Interstage ディレクトリサービスを運用する場合に必要となるシステム資源に加え、使用しているRDB（Symfoware Serverまたはracleデータベースⅰ）に応じて、それぞれ必要となるシステム資源を加算してください。

システムパラメタの変更方法や、各パラメタの意味については、“システムパラメタについて”を参照してください。

Interstage ディレクトリサービスの運用

Interstage ディレクトリサービスを運用する場合に必要となるシステム資源について、以下に示します。

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
－	project.max-shm-memory	加算値	(リポジトリ数 × 1,572,864 ＋ (5 × (リポジトリ数 × 1,843,200))) × (4 × リポジトリ数)
shmmax	－	設定値	リポジトリ数 × 1,572,864 ＋ (5 × (リポジトリ数 × 1,843,200)) 以上
shmmni	project.max-shm-ids	加算値	4 × リポジトリ数

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	5 × (リポジトリ数 × 1,843,200) 以上
kernel.shmmni	加算値	4 × リポジトリ数

Symfoware Serverの運用

Symfoware Serverの運用を行う場合に、追加で必要となるシステムパラメタの設定を以下に示します。ただし、Symfoware Server Lite Editionを使用している環境では、本設定は不要です。

- Symfoware ServerをSolarisにインストールした場合
- Symfoware ServerをLinuxにインストールした場合

ここで示す各システムパラメタの必要数の値は、Symfoware Serverのシステム用動作環境ファイル、またはRDB構成パラメタファイルのパラメタに以下の値が設定されていることを前提としています。これらのパラメタの設定値を変更する場合は、Symfoware Serverのマニュアルを参照して各システムパラメタの必要数を算出し直してください。

パラメタ	設定値
ローカル接続で使用するメモリ量 (COMMUNICATION_BUFFER)	32Kバイト
ローカル接続数(MAX_CONNECT_SYS) (注1)	256
デーモン多重度(RDBCNTNUM)	712
共有メモリ量(RDBEXTMEM)	13,208Kバイト

注1)

ローカル接続数は、Interstage ディレクトリサービスの使用時に必要となる、リポジトリからRDBへの最大コネクション数の合計に、他のアプリケーション等が使用するコネクション数を加えた値を算出してください。求めた値が、この設定値(256)を超える場合には、各システムパラメタの必要数を算出し直してください。

リポジトリの最大コネクション数の詳細は、“ディレクトリサービス運用ガイド”の“データベース共用”-“最大コネクション数の設定”を参照してください。

レプリケーション運用を行う場合は、Linkexpressの運用に必要なシステム資源を設定してください。設定内容は、Linkexpressのインストールガイドを参照してください。
Solarisの場合は、「Linkexpressへの同時転送依頼数」を「1」、「Linkexpress同時ファイル転送多重度」を「4」として計算してください。また、システムパラメタ「shmmin」(共用メモリセグメントの最小サイズ)は設定しないでください。

Symfoware ServerをSolarisにインストールした場合

共用メモリ

パラメタ	資源制御	種類	必要数
shmmax	project.max-shm-memory	設定値	13,524,992 以上
shmmni	project.max-shm-ids	加算値	10

セマフォ

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	300
semmsl	process.max-sem-nsems	設定値	48 以上

メッセージキュー

パラメタ	資源制御	種類	必要数
msgmnb	process.max-msg-qbytes	設定値	4,096 以上
msgmni	project.max-msg-ids	加算値	2
msgtql	process.max-msg-messages	設定値	64以上

Symfoware ServerをLinuxにインストールした場合

共用メモリ

パラメタ	種類	必要数
kernel.shmmax	設定値	13,524,992 以上
kernel.shmmni	加算値	10

セマフォ

パラメタ	種類	必要数
para1	設定値	48 以上
para2	加算値	1,112
para3		すでに設定されている値
para4	加算値	300

メッセージキュー

パラメタ	種類	必要数
kernel.msgmax	設定値	128 以上
kernel.msgmnb	設定値	4,096 以上
kernel.msgmni	加算値	2

Oracleデータベースの運用

Oracleデータベースの運用を行う場合に、追加で必要となるシステムパラメタの設定を以下に示します。

- OracleデータベースをSolarisにインストールした場合
- OracleデータベースをLinuxにインストールした場合

レプリケーション運用を行う場合は、レプリケーションの運用に必要なシステム資源を設定してください。設定内容については、Oracleデータベースのマニュアルを参照してください。

OracleデータベースをSolarisにインストールした場合

共用メモリ

パラメタ	資源制御	種類	必要数
shmmax	project.max-shm-memory	設定値	4,294,967,295 以上
shmmni	project.max-shm-ids	設定値	100 以上

セマフォ

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	設定値	100 以上
semmsl	project.max-sem-nsems	設定値	256 以上

OracleデータベースをLinuxにインストールした場合

共用メモリ

パラメタ	種類	必要数
kernel.shmall	設定値	2,097,152 以上
kernel.shmmax	設定値	物理メモリのサイズ(バイト)の1/2 以上
kernel.shmmni	設定値	4,096 以上

セマフォ

パラメタ	種類	必要数
para1	設定値	250 以上
para2	設定値	32,000 以上
para3	設定値	100 以上
para4	設定値	128 以上

ファイルシステム

パラメタ	種類	必要数
fs.file-max (最大ファイル・ハンドル数)	設定値	65,536 以上

ネットワーク

パラメタ	種類	必要数
net.ipv4.ip_local_port_range (ポート番号の範囲)	設定値	最小:1,024 最大:65,000

パラメタ	種類	必要数
net.core.rmem_default (受信用ウィンドウ・サイズのデフォルト値)	設定値	1,048,576 以上
net.core.rmem_max (受信用ウィンドウ・サイズの最大値)	設定値	1,048,576 以上
net.core.wmem_default (送信用ウィンドウ・サイズのデフォルト値)	設定値	262,144 以上
net.core.wmem_max (送信用ウィンドウ・サイズの最大値)	設定値	262,144 以上

3.1.10 Interstage管理コンソールのシステム資源の設定 Linux32/64

Interstage管理コンソールを用いたシステムの運用時には、システム資源を拡張する必要があります。ここでは、以下について説明します。

- システムパラメタ Linux32/64

注意

Interstage管理コンソールを使用する場合は、Interstage統合コマンドのシステム資源も拡張する必要があります。“3.1.11 Interstage統合コマンドのシステム資源の設定”を参照し、必要な値を加算してください。

■システムパラメタ Linux32/64

Interstage管理コンソールを運用するためのサービスが使用するセマフォのシステムパラメタのチューニングについて説明します。
システムパラメタの変更方法や、各パラメタの意味については、“■システムパラメタについて”を参照してください。

セマフォ

パラメタ	種類	必要数
para1	設定値	1以上
para2	加算値	1
para3	設定値	1以上
para4	加算値	1

3.1.11 Interstage統合コマンドのシステム資源の設定 Solaris64 Linux32/64

以下のどれかを利用する場合は、各機能で追加したシステム資源に加えて、Interstage統合コマンドで使用するシステム資源を拡張する必要があります。
ここでは、システムパラメタについて説明します。

EE

- マルチ言語サービス

EE ・ J2EE互換機能

■システムパラメタ

Interstage統合コマンドで使用する共用メモリ、セマフォ、メッセージキューのシステムパラメタのチューニングについて説明します。
システムパラメタの変更方法や、各パラメタの意味については、“■システムパラメタについて”を参照してください。

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
－	project.max-shm-memory	加算値	2015552 以上
shmmax	－	設定値	1106440 以上
shmmni	project.max-shm-ids	加算値	22

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	Linux32 12503496 以上
		Linux64 13208308 以上
kernel.shmmni	加算値	19

セマフォ Solaris64

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	2
semmsl	process.max-sem-nsems	設定値	12 以上
semopm	process.max-sem-ops	設定値	3 以上

セマフォ Linux32/64

パラメタ	種類	必要数
para1	設定値	12 以上
para2	加算値	2

パラメタ	種類	必要数
para3	設定値	3 以上
para4	加算値	2

メッセージキュー Solaris64

パラメタ	資源制御	種類	必要数
msgmnb	process.max-msg-qbytes	設定値	4572 ＋ (528 × 同時実行コマンド数) (注)
msgmni	project.max-msg-ids	加算値	12
msgtql	process.max-msg-messages	設定値	15 ＋ 同時実行コマンド数 以上(注)

注)

同時実行コマンド数とは、Interstage統合コマンドを同時に実行した数のことです。

EE
- オブジェクト閉塞／閉塞解除

メッセージキュー Linux32/64

パラメタ	種類	必要数
kernel.msgmax	設定値	528 以上
kernel.msgmnb	設定値	4572 ＋ (528 × 同時実行コマンド数) (注)
kernel.msgmni	加算値	12

注)

同時実行コマンド数とは、Interstage統合コマンドを同時に実行した数のことです。

Linux32

Systemwalker Operation Manager／Interstage運用APIを使用して、以下の操作を行う場合は、同時に操作する回数が同時実行コマンド数となります。

- ワークユニットの起動／停止

EE
- オブジェクト閉塞／閉塞解除

Linux64

Systemwalker Operation Managerを使用して、ワークユニットの起動／停止を行う場合は、同時に操作する回数が同時実行コマンド数となります。

3.1.12 Webサーバコネクタのシステム資源の設定 Windows32/64 Solaris64 Linux32/64

Webサーバコネクタを使用する場合には、システム資源を拡張する必要があります。ここでは、以下について説明します。
Webサーバコネクタ(Interstage HTTP Server 2.2用)を使用する場合も同様です。

- システムパラメタ
  - Webサーバコネクタを使用する場合
  - Webサーバコネクタの故障監視機能を使用する場合

注意

- Webサーバコネクタではエフェメラルポートを使用しています。エフェメラルポートについては、以下を参照してください。
  - 「3.3 TCP/IPパラメタのチューニング」
  - 「システム設計ガイド」の「付録B ポート番号」
- Webサーバには、Interstage HTTP ServerやInterstage HTTP Server 2.2などの種類がありますが、Webサーバの種類にかかわらず、Webサーバの数で計算してください。例えば、Interstage HTTP ServerのWebサーバが2つ、Interstage HTTP Server 2.2のWebサーバが1つの場合は、Webサーバの数は3になります。

EE
- なお、J2EEをお使いの場合は、「IJServerクラスタ」を「IJServerワークユニット」に読み替えて設定してください。

■システムパラメタ

Webサーバコネクタを使用する際には、以下のシステムパラメタのチューニングを行ってください。
Webサーバコネクタの故障監視機能を使用する場合は、Webサーバコネクタで使用する資源にWebサーバコネクタの故障監視機能で使用する資源量を加算してください。
システムパラメタの変更方法や、各パラメタの意味については、“■システムパラメタについて”を参照してください。

Webサーバコネクタ

Webサーバコネクタを使用する場合に必要となるシステム資源について、以下に示します。

セマフォ Solaris64

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	以下をWebサーバごとに求めた合算値 (注) ・ ((IJServerクラスタ数の合計値 ＋ 各IJServerクラスタのプロセス多重度の合計値 ＋ 2)×2) ＋ 4
semmsl	process.max-sem-nsems	設定値	1 以上
semopm	process.max-sem-ops	設定値	2 以上

注)

チューニングの例を以下に示します。

条件		
IJServerクラスタ	WU001	プロセス多重度:3
	WU002	プロセス多重度:3
	WU003	プロセス多重度:3
Webサーバ	web001	接続先:WU001、WU002
	web002	接続先:WU003

― semmni

web001の必要数 ＝
((2(IJServerクラスタ数の合計値) ＋
6(各IJServerクラスタのプロセス多重度の合計値) ＋ 2)×2) ＋ 4
＝ 24
web002の必要数 ＝
((1(IJServerクラスタ数の合計値) ＋
3(各IJServerクラスタのプロセス多重度の合計値) ＋ 2)×2) ＋ 4
＝ 16
semmni ＝ 24(web001の必要数) ＋ 16(web002の必要数) ＝ 40

セマフォ Linux32/64

パラメタ	種類	必要数
para1	設定値	1以上
para2	加算値	以下をWebサーバごとに求めた合算値 (注) ・ ((IJServerクラスタ数の合計値 ＋ 各IJServerクラスタのプロセス多重度の合計値 ＋ 2)×2) ＋ 4
para3	設定値	2 以上
para4	加算値	以下をWebサーバごとに求めた合算値 (注) ・ ((IJServerクラスタ数の合計値 ＋ 各IJServerクラスタのプロセス多重度の合計値 ＋ 2)×2) ＋ 4

注)

チューニングの例を以下に示します。

条件		
IJServerクラスタ	WU001	プロセス多重度:3
	WU002	プロセス多重度:3
	WU003	プロセス多重度:3
Webサーバ	web001	接続先:WU001、WU002
	web002	接続先:WU003

― para2

web001の必要数 ＝
((2(IJServerクラスタ数の合計値) ＋
6(各IJServerクラスタのプロセス多重度の合計値) ＋ 2)×2) ＋ 4
＝ 24
web002の必要数 ＝
((1(IJServerクラスタ数の合計値) ＋
3(各IJServerクラスタのプロセス多重度の合計値) ＋ 2)×2) ＋ 4
＝ 16
para2 ＝ 24(web001の必要数) ＋ 16(web002の必要数) ＝ 40

― para4

para2と同様に計算してください。

Webサーバコネクタの故障監視機能

Webサーバコネクタの故障監視機能を使用する場合に追加となるシステム資源について、以下に示します。

共用メモリ Solaris64

パラメタ	資源制御	種類	必要数
shmmax	－	設定値	6,720,012 以上
－	project.max-shm-memory	加算値	6,720,012 × (Webサーバ数 + 1)
shmmni	project.max-shm-ids	加算値	Webサーバ数 + 1

共用メモリ Linux32/64

パラメタ	種類	必要数
kernel.shmmax	設定値	6,720,012 以上
kernel.shmmni	加算値	Webサーバ数 + 1

セマフォ Solaris64

パラメタ	資源制御	種類	必要数
semmni	project.max-sem-ids	加算値	3

セマフォ Linux32/64

パラメタ	種類	必要数
para1	設定値	2 以上
para2	加算値	3
para3	設定値	2 以上
para4	加算値	3

EE

3.2 性能監視ツール使用時に必要なシステム資源

性能監視ツールで性能監視環境として使用する資源の見積もり方法を説明します。

EE

3.2.1 システム構成情報の見積もり方法 (Solarisの場合) Solaris64

システム構成情報は、以下の見積もり式を参考に見積もり、見積もり値以上の値を設定してください。

セマフォ

システム構成情報	資源制御	種類	見積もり
semsys: seminfo_semmni	project.max-sem-ids	加算値	1
semsys: seminfo_semmsl	process.max-sem-nsems	設定値	セマフォ数 (*) 以上の値

***)**

セマフォ数 ＝ 性能監視ツール起動時に指定する共有メモリ量(MB) × 10 ＋ 2
ただし、最大52

共有メモリ

システム構成情報	資源制御	種類	見積もり
shmsys: shminfo_shmmax	project.max-shm-memory	設定値	“3.2.3 共有メモリ量の見積もり方法”を参照
shmsys: shminfo_shmmni	project.max-shm-ids	加算値	1

EE

3.2.2 システム構成情報の見積もり方法 (Linuxの場合) Linux32/64

システム構成情報は、以下の見積もり式を参考に見積もり、見積もり値以上の値を設定してください。

セマフォ

kernel.sem = *para1 para2 para3 para4*

システム構成情報	種類	見積もり
para1	設定値	セマフォ数 (*) 以上の値
para2	加算値	セマフォ数 (*)
para3	設定値	セマフォ数 (*) 以上の値
para4	加算値	1

***)**

セマフォ数 ＝ 性能監視ツール起動時に指定する共有メモリ量(MB) × 10 ＋ 2
ただし、最大52

共有メモリ

システム構成情報	種類	見積もり
kernel.shmmax	設定値	“3.2.3 共有メモリ量の見積もり方法”を参照
kernel.shmmni	加算値	1

EE

3.2.3 共有メモリ量の見積もり方法

共有メモリ量は、以下の見積もり式を参考に必要な共有メモリ量を求め、その共有メモリ量をメガバイト単位に切り上げた値で見積もってください。

Solaris64

変更方法の詳細については、“■システムパラメタについて”を参照してください。

共有メモリ量の見積もり(単位:バイト)

1. 性能監視ツールの性能監視対象とする全オブジェクト(アプリケーション)に対して、**オブジェクト(アプリケーション)ごとに必要な共有メモリ量**を、以下の方法で求めてください。

   a. 各オブジェクト(アプリケーション)に定義されている**プロセス多重度**を確認してください。

   b. 各オブジェクト(アプリケーション)に定義されている**スレッド多重度**を確認してください。

   c. 各オブジェクト(アプリケーション)に登録されている**オペレーション数**を確認してください。
   オペレーション数はCORBAアプリケーションおよびトランザクションアプリケーションの場合、IDL定義ファイルから求めてください。EJBアプリケーションの場合、remoteインターフェースとHomeインターフェースのソースを確認し、publicであるメソッドの数を数えて、これに、継承されるインターフェースを加算して求めてください。

   d. 以下の計算で、**オブジェクト(アプリケーション)ごとに必要な共有メモリ量**を求めてください。

   - **オペレーション数の平均**が**3以下**の場合
     Windows32 Linux32
     (プロセス多重度 × スレッド多重度 × 1536) ＋ 400
     Windows64 Solaris64 Linux64
     4600 × プロセス多重度 × スレッド多重度＋ 276288

   - **オペレーション数の平均**が**3よりも多い**場合
     Windows32 Linux32
     (オペレーション数 × プロセス多重度 × スレッド多重度 × 546) ＋ 400
     Windows64 Solaris64 Linux64
     (4600 × プロセス多重度 × スレッド多重度 × オペレーション数 ÷ 3) ＋ 276288

2. 以下の計算で、**必要な共有メモリ量**を求めてください。

   オブジェクトごとに必要な共有メモリ量の合計 ＋ 261188

3. **共有メモリ量**を、以下の計算で求めてください。

   必要な共有メモリ量 ÷ 1048576

   なお、上記の計算で端数(小数点以下)が発生した場合は、切り上げてください。

注意

インターバル時間内に性能監視対象のオブジェクトを再起動する場合には、a.～d.で求めた「オブジェクトごとに必要な共有メモリ量の合計」について、再起動回数分、あらかじめ積算してください。

3.3 TCP/IPパラメタのチューニング

TCP/IPパラメタのチューニング方法を説明します。

■チューニング方法 Windows32/64

レジストリエディタを使用して以下のレジストリ情報を追加した後、システムを再起動してください。

- レジストリキー:
  HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥Tcpip¥Parameters
  - 名前:TcpTimedWaitDelay
  - 種類:REG_DWORD
  - 推奨値:1E(30秒)
- レジストリキー:
  HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥Tcpip¥Parameters
  - 名前:MaxUserPort
  - 種類:REG_DWORD

ー 推奨値:65534(10進数)

短時間に数多くのクライアントが接続する場合、TCP/IPのソケット不足になりWebブラウザに「Internal Server Error」が表示される場合があります。
上記情報を追加することで、使用可能なソケット数を増やし、また、使用済みのソケットを早く開放するようになります。

注意

- レジストリ情報がない場合は、新しく作成してください。
- TCP/IPパラメタをチューニングする際は、上記の2つのレジストリキーの値を変更してください。
- レジストリの編集操作を誤ると、システムが不安定になる可能性がありますので、操作の際には、指定されたキー以外は一切変更を加えないよう注意してください。また、操作前には必ずレジストリのバックアップを行ってください。
- TCP/IPパラメタのチューニングは、すべてのTCPに対して影響を与えるため、システム管理者に相談してから実施してください。

■チューニング方法 Solaris64

nddコマンドを使用してtcp_time_wait_intervalを60秒にしてください。OSのデフォルト値は60秒のため、値を変更していなければ設定する必要はありません。
エフェメラルポートが不足している場合、nddコマンドを使用してtcp_smallest_anon_portおよびtcp_largest_anon_portを変更し、エフェメラルポートに使用できるポートを増やしてください。
永続的に設定を有効する場合には、RCプロシジャ(/etc/rc2.d)に登録が必要です。
RCプロシジャの例を以下に示します。

```
#!/bin/sh
ndd -set /dev/tcp tcp_time_wait_interval 60000
```

短時間に数多くのクライアントが接続する場合、TCP/IPのソケット不足になりWebブラウザに「Internal Server Error」が表示される場合があります。
上記情報を追加することで、使用可能なソケット数を増やし、また、使用済みのソケットを早く開放するようになります。

注意

- TCP/IPパラメタのチューニングは、すべてのTCPに対して影響を与えるため、システム管理者に相談してから実施してください。
- エフェメラルポートに使用できるポートを増やす場合、Interstage、および他のアプリケーションが使用しているポートをエフェメラルポートに指定しないようにしてください。

■チューニング方法 Linux32/64

エフェメラルポートが不足している場合、システムパラメタnet.ipv4.ip_local_port_rangeを変更して、エフェメラルポートに使用できるポートを増やしてください。
システムパラメタの変更については、「3.1 サーバ機能運用時に必要なシステム資源」の「システムパラメタの変更方法」を参照してください。

注意

- TCP/IPパラメタのチューニングは、すべてのTCPに対して影響を与えるため、システム管理者に相談してから実施してください。
- エフェメラルポートに使用できるポートを増やす場合、Interstage、および他のアプリケーションが使用しているポートをエフェメラルポートに指定しないようにしてください。

3.4 IPC資源のカスタマイズ Solaris64 Linux32/64

ここでは、その他に必要となるカスタマイズ項目について説明します。

■System V IPC資源のIPCキー値のカスタマイズについて

Interstageでは、Interstageを構成するプロセス間通信のために、OSが提供するSystem V IPC資源(メッセージキュー、セマフォ、共有メモリ)を使用しています。このIPC資源は、作成時に指定する値(IPCキー値)により、システムで一意に識別されます。
IPCキー値は、システムで一意でなければなりませんが、任意の値を使用することが可能なため、ほかのIPC資源を使用する製品およびアプリケーションプログラムと重複することがあります。

IPCキー値の重複が発生した場合、Interstageでは、以下のようなメッセージを出力して、IPCキー値の重複を通知します。

例

IPCキー値の重複を検出したときにコンポーネントトランザクションサービスが出力するメッセージの例

- TD: エラー: td11038:必要なIPC資源が使用中のため獲得できませんでした(key=%x path=%s)

この場合、IPCキー値に対応したIPC資源を使用しているInterstageのサービスの各機能は使用できません。
このような状態に対処するため、Interstageでは以下に示す方法で、Interstageが使用するIPCキー値をカスタマイズすることが可能です。IPCキー値の重複発生を通知するメッセージが出力された場合は、この対処により運用することができます。

■概要

IPCキー値は4バイト(32ビット)で構成されますが、そのうちの下位12ビット(16進3桁)に任意の値を定義することで、ほかの製品が使用するIPCキー値と重複しないようにします。なお、上位残りの20ビットは、Interstageが決定します。

■IPCキー値の定義方法

以下のIPCキー値定義ファイルを新規に作成し、16進3桁でIPCキー値の下位12ビットを指定します。
MessageQueueDirectorを除くサービスは、共通定義ファイルの指定が有効です。MessageQueueDirectorは固有の定義ファイルの指定が有効です。

共通定義ファイル:

Solaris64
/var/opt/FJSVisas/system/システム名/FJSVisas/etc/ipc_key
Linux32/64
/var/opt/FJSVisas/system/default/FJSVisas/etc/ipc_key

MQDシステムの定義ファイル:

/opt/FJSVmqd/mqd/MQDシステム名/ipc_key

注意

- 定義ファイルの内容が16進3桁以外の場合は、IPCキー値の指定がない場合と同様に動作します。
- “MQDシステム名”については、“MessageQueueDirector 説明書”を参照してください。

- Linux32/64
  クラスタシステムを使用する場合、MQDシステム用の定義ファイルは、mqd環境定義ファイル中の以下で指定したディレクトリの中に定義ファイル(ipc_key)を作成してください。

  ー [Cluster]
    SystemDirectory = MQDのクラスタサービスが使用するディレクトリの名前

例

```
FFF
```

この定義例の場合、Interstageが使用する上位20ビットを0x01280とすると、IPCキー値は、16進表示で、以下のようになります。

```
0x01280FFF
```

注意

- IPCキー値の定義は、Interstageを全強制停止モードで停止し、さらにInterstage JMXサービスを停止してから行ってください。Interstageの運用中に本定義を変更しないでください。
- Solaris64
  IPCキー値の定義は、システム間で重複することのないように定義してください。

3.5 RHEL7でのシステムログ出力設定 Linux32/64

RHEL7で一定の期間内に大量のメッセージがシステムログに出力された場合、以降のメッセージのシステムログへの出力が抑止されることがあります。

すべてのシステムログを出力する場合は、以下の設定を行って下さい。

- /etc/systemd/journald.confを編集して、以下の値を設定します。

```
RateLimitInterval=0
```

- /etc/rsyslog.conf内を編集して、「$ModLoad imjournal」の次の行に、以下の値を設定します。

```
$imjournalRatelimitInterval 0
```

上記を設定後、オペレーティングシステムを再起動してください。

システムログの設定の詳細については、OSのドキュメントを参照してください。

EE

第4章 ワークユニットのチューニング

ワークユニットには、様々な機能があり、これらをチューニングすることで、最適な状態での運用を行うことが可能となります。ここでは、ワークユニットのチューニングについて説明します。

4.1 ワークユニット数、オブジェクト数、プロセス数のチューニング

起動できるワークユニット数、オブジェクト数およびプロセス数は以下の条件式を満たす必要があります。

(ワークユニット数×2)＋(オブジェクト総数×m)＋(プロセス総数×n)＋1 ≦ 2010

この条件式を超えてワークユニットを起動する場合は、大規模システム用環境定義ファイルをコピーする必要があります。なお、Linux for Intel64版では、標準で大規模システム用環境定義ファイルが使用されています。

大規模システム用環境定義ファイルを使用した場合、以下の条件式まで起動できるワークユニット数、オブジェクト数およびプロセス数が拡張されます。

(ワークユニット数×2)＋(オブジェクト総数×m)＋(プロセス総数×n)＋1 ≦ 6030

条件式のオブジェクト総数とは、ワークユニット定義の[Application Program]セクションの総数です。

IJServerの場合、各ワークユニットのオブジェクト数は以下となります。

IJServerタイプ	オブジェクト数
WebアプリケーションとEJBアプリケーションを同一JavaVMで運用 Webアプリケーションのみ運用 EJBアプリケーションのみ運用	1
WebアプリケーションとEJBアプリケーションを別JavaVMで運用	2

条件式のm、nについては、m＝1、n＝2を基準値とし、以下に該当する場合は、変更してください。

- islistwuおよびislistobjコマンドを使用する場合
  m＝2
- islistaplprocコマンドを使用する場合
  n＝3
- Interstage管理コンソールを使用する場合
  m＝2、n＝3

大規模システム用環境定義ファイル

	コピー元	コピー先
Windows32/64	Interstageインストール先フォルダ¥extp¥etc¥sys¥td¥extp.ini.large	Interstageインストール先フォルダ¥td¥var¥td001¥sys¥etc¥extp.ini
Solaris64	/opt/FSUNextp/etc/sys/td/extp.ini.large	/var/opt/FJSVisas/system/default/FSUNextp/td001/sys/etc/extp.ini
Linux32	/opt/FJSVextp/etc/sys/td/extp.ini.large	/var/opt/FJSVisas/system/default/FJSVextp/td001/sys/etc/extp.ini

通常システム用環境定義ファイル

通常のシステム定義に復元する場合は、以下の通常システム用環境定義ファイルをコピーしてください。

	コピー元	コピー先
Windows32/64	Interstageインストール先フォルダ¥extp¥etc¥sys¥td¥extp.ini	Interstageインストール先フォルダ¥td¥var¥td001¥sys¥etc¥extp.ini
Solaris64	/opt/FSUNextp/etc/sys/td/extp.ini	/var/opt/FJSVisas/system/default/FSUNextp/td001/sys/etc/extp.ini

	コピー元	コピー先
Linux32	/opt/FJSVextp/etc/sys/td/extp.ini	/var/opt/FJSVisas/system/default/ FJSVextp/td001/sys/etc/extp.ini

注意

- 大規模システム用環境定義ファイルを使用した場合、共有メモリ使用量が14メガバイト増加します。
- 環境定義ファイルをコピーする場合は、Interstageを停止してください。
- コピー先の環境定義ファイルを誤って削除した場合、システム規模に応じて環境定義ファイルを再度コピーしてください。

4.2 プロセス強制停止時間のチューニング

プロセス強制停止時間は、ワークユニット停止においてアプリケーションプロセスの停止処理がハングアップしていると判断する時間です。
プロセス強制停止時間をチューニングする場合は、アプリケーションプロセス停止処理に必要な時間を考慮してください。以下の条件式を参考にしてください。

■条件式

プロセス強制停止時間(秒) ＞ アプリケーションプロセス停止処理時間(秒)

アプリケーションプロセス停止処理時間:
以下の場合において、アプリケーションプロセス停止処理時間を考慮してください。その他の場合は、デフォルト値で問題ありません。

- CORBAワークユニットの場合
  サーバアプリケーションの活性化後の動作モードに“SYNC_END(サーバアプリケーションを活性化しても、活性化メソッドは復帰しません。)”を設定し、かつ、活性化メソッドの後に後処理を記述している場合、後処理の実行に必要な時間
  サーバアプリケーションの活性化後の動作モードについては、“リファレンスマニュアル(コマンド編) OD_impl_inst(「CORBAアプリケーション情報定義ファイルでの登録」の「mode」)”を参照してください。
- IJServerワークユニットの場合
  停止時実行クラスを登録している場合、停止時実行クラスの実行に必要な時間
  停止時実行クラスについては、“J2EE ユーザーズガイド(旧版互換)(「J2EEアプリケーションの設計」「J2EEアプリケーションが運用される環境(IJServer) 」の「起動/停止の実行クラス」)”を参照してください。

ポイント

ワークユニットを停止する時は、お客様の業務が終了していることを想定しています。そのため、デフォルト値には、ハングアップしていると判断して差し支えのない値として、180秒を設定しています。

業務運用中にワークユニットを停止することがある場合は、アプリケーション処理時間を考慮する必要があります。以下の条件式を参考にしてください。
プロセス強制停止時間はアプリケーションの処理時間とプロセスの停止処理時間の合計を上回るように設定してください。

プロセス強制停止時間(秒) ＞ アプリケーション処理時間(秒) ＋ アプリケーションプロセス停止処理時間(秒)

アプリケーション処理時間:
アプリケーション処理中にワークユニットを停止するような運用を実施する場合、そのアプリケーションの処理が正常に完了するために必要な、最も長い時間を設定してください。
ワークユニット同期停止は、アプリケーション処理中に実行すると、処理中の要求が完了するのを待ってからプロセスを停止します。
アプリケーション処理時間の考慮がされていないと、ワークユニット同期停止において、処理中の要求が途中で強制終了さ

れます。
注)IJServerワークユニットの通常停止は同期停止と同じ動きとなります。

アプリケーションプロセス停止処理時間:
■条件式の**アプリケーションプロセス停止処理時間**を参照してください。

例

アプリケーション処理時間が120秒で、アプリケーションプロセス停止処理時間が5秒の場合、プロセス強制停止時間は、126秒以上を設定してください。

4.3 CORBAワークユニットのチューニング

ワークユニットのチューニングについては、以下のマニュアルを参照してください。

- “OLTPサーバ運用ガイド”の“ワークユニットの作成”
- “OLTPサーバ運用ガイド”の“CORBAワークユニット”

4.4 IJServerワークユニットのチューニング

IJServerワークユニットについては、以下を参照してください。

- “J2EE ユーザーズガイド(旧版互換)”－“IJServerのチューニング”

4.5 トランザクションアプリケーションのチューニング Windows32 Linux32

トランザクションワークユニットについては、以下のマニュアルを参照してください。

- “OLTPサーバ運用ガイド”の“ワークユニットの作成”
- “OLTPサーバ運用ガイド”の“トランザクションアプリケーションのワークユニット”

4.6 ラッパーワークユニットのチューニング Windows32

ラッパーオブジェクトワークユニットについては、以下のマニュアルを参照してください。

- “OLTPサーバ運用ガイド”の“ワークユニットの作成”
- “OLTPサーバ運用ガイド”の“ラッパーオブジェクトのワークユニット”

4.7 ユーティリティワークユニットのチューニング Solaris64 Linux32/64

ユーティリティワークユニットについては、以下のマニュアルを参照してください。

- “OLTPサーバ運用ガイド”の“ワークユニットの作成”
- “OLTPサーバ運用ガイド”の“ユーティリティのワークユニット”

第5章 Java EE 6機能のチューニング

Java EE 6機能のチューニングについては、「Java EE運用ガイド(Java EE 6編)」の「Java EE 6機能のチューニング」を参照してください。

第6章 Java EE 7機能のチューニング

Java EE 7機能のチューニングについては、「Java EE 7 設計・構築・運用ガイド」の「Java EE 7機能のチューニング」を参照してください。

第7章 J2EEのチューニング

J2EEのチューニングについては、「J2EE ユーザーズガイド(旧版互換)」を参照してください。

第8章 業務構成管理機能のチューニング

業務構成管理機能が管理するリポジトリの設定値をチューニングすることで、最適な状態での運用を行うことが可能となります。ここでは、業務構成管理機能が管理するリポジトリのチューニングについて説明します。

8.1 リポジトリのチューニング

リポジトリのチューニングについては、以下のマニュアルを参照してください。

・“運用ガイド(基本編)”の“業務構成管理機能”

第9章 JDK/JRE 7のチューニング

CやC++では、メモリに割り当てた領域は、不要になった時点でプログラマーが明示的に解放する必要があります。しかし、この解放処理に漏れやミスがあると、メモリリーク、プログラム停止および暴走が発生するデメリットがあります。

Javaでは、ガーベジコレクション(GC)を導入したことにより、プログラマーがメモリ管理において作業の負担を軽減できるようになりました。その反面、GCが発生するたびにJavaアプリケーションが一時停止するため、パフォーマンス劣化の要因になることがあります。また、Java独特のメモリリークも存在します。

Javaでは、スレッドを管理しやすいため、大量のスレッドを生成する傾向があります。このため、スタックが大量に生成されて、メモリ不足になることもあります。

以上から、多くの場合において、JDK/JRE 7のチューニングは、スタックサイズ、または、GCの発生頻度とその処理時間を調整することになります。本章では、Javaアプリケーションのチューニングにあたって、必要な基礎知識やチューニング方法などを説明します。

9.1 基礎知識

JDK/JRE 7のチューニングを行う際に必要な知識を説明します。

9.1.1 JDK関連のドキュメント

JDKドキュメント

本製品に搭載しているJDK のドキュメントは、以下のURLにあります。

```
http://docs.oracle.com/javase/jp/7/
```

- Java VisualVMのドキュメントについて

  Java VisualVM に関しては、以下のURLを参照してください。

  ```
  http://docs.oracle.com/javase/jp/7/technotes/guides/visualvm/index.html
  ```

- jcmdのドキュメントについて

  jcmd に関しては、以下のURLを参照してください。

  - Windows32/64

    ```
    http://docs.oracle.com/javase/jp/7/technotes/tools/windows/jcmd.html
    ```

  - Solaris64 Linux32/64

    ```
    http://docs.oracle.com/javase/jp/7/technotes/tools/solaris/jcmd.html
    ```

javaコマンドおよびJava VMのオプションには、チューニングに関するオプションがあります。本章では、これらのオプションを随所で紹介します。
各オプションの詳細は、次を参照してください。なお、本マニュアル内で具体的に説明していないJava VMのチューニングに関するオプションは、富士通版Java VMではサポートしておりません。

- javaコマンドのオプション

  JDKドキュメントの「JDKツールとユーティリティ」の「java」

- Java HotSpot VM Options

  http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html

移行について

JDK1.4からJDK5.0への移行については、以下のURLを参照してください。

- http://www.oracle.com/technetwork/java/javase/compatibility-137462.html

JDK5.0からJDK6への移行については、以下のURLを参照してください。

- http://www.oracle.com/technetwork/java/javase/adoptionguide-137484.html

JDK6からJDK7への移行については、以下のURLを参照してください。

- http://docs.oracle.com/javase/jp/7/webnotes/adoptionGuide/index.html

9.1.2 Java VM

製品搭載のJava VM

JDK/JREには、Javaのバイトコードを解釈・実行するエンジン部であるJava仮想マシン(**Java VM**)があります。

“表9.1 Java VMの種類と特徴”に、**Java VM**の種類と特徴を示します。

表9.1 Java VMの種類と特徴

Java VMの名称	Java VMの特徴
Java HotSpot Client VM	アプリケーション起動時間を短縮し、メモリ消費を抑制するように設計されたクライアント環境向けの**Java VM**です。 富士通版**Java HotSpot Client VM**では、Oracle CorporationのJava VMであるJava HotSpot Client VMをベースに、富士通独自技術によるトラブルシューティングに関する機能強化などを追加実装しています。 そのため、ベースとしたJava HotSpot Client VMと機能的な互換性を基本的にもっています。 追加実装された機能項目については、“9.1.3 FJVM”を参照してください。
Java HotSpot Server VM (FJVM)	アプリケーション起動時間の短縮などよりも、安定性および、スループットの向上を考慮して設計されたサーバ環境向けの**Java VM**です。 富士通版**Java HotSpot Server VM**では、Oracle CorporationのJava VMであるJava HotSpot Server VMをベースに、富士通独自技術による性能改善やトラブルシューティングに関する機能強化などを追加実装しています。 そのため、ベースとしたJava HotSpot Server VMと機能的な互換性を基本的にもっています。 富士通版**Java HotSpot Server VM**は、Interstage Application Server搭載のJDK/JREにおけるデフォルトの**Java VM**であることから、このJava VMを特に**FJVM**と呼びます。 追加実装された機能項目については、“9.1.3 FJVM”を参照してください。

- Interstage Application Server搭載のJDK/JREでは、**Java HotSpot Client VM**と**FJVM**(=**Java HotSpot Server VM**)を搭載しています。
  デフォルトの**Java VM**は、**FJVM**です。これは、javaコマンドのオプションに、“-server”または“-fjvm”を指定することと同義です。**Java HotSpot Client VM**を使用したい場合は、オプションに“-client”を指定してください。

- Windows64 Solaris64 Linux64
  実行モードが64ビットモードのJDK/JREでは**FJVM**だけ使用可能です。**Java HotSpot Client VM**は搭載していません。

注意

エルゴノミクス機能によるJava VMの自動選択機能

富士通版JDK/JRE では、エルゴノミクス機能によるJava VMの自動選択機能（マシンのCPU数や物理メモリ量などに応じて、使用するJava VMを自動的に選択する機能）は無効になっています。

Java VM関係の資料

- **Java VM**の詳細は、JDKドキュメンテーションの次を参照してください。

  [Java 概念図の説明] > [Java 仮想マシン] および
  http://docs.oracle.com/javase/jp/7/technotes/guides/vm/index.html

- 他にも、**Java VM**に関連する資料があります。

  ― Java Language and Virtual Machine Specifications

  http://docs.oracle.com/javase/specs/

  ― Java HotSpot VM Options

  http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html

9.1.3 FJVM

本章におけるJava VMに関する情報は、デフォルトのJava VMである「富士通版Java HotSpot Server VM(=FJVM)」を中心に説明しています。なお、特に断り書きの無い限り、それらの情報は「富士通版Java HotSpot Client VM(=Java HotSpot Client VM)」に対しても、そのまま適用されます。

FJVMにおいて提供される「富士通独自技術により強化された機能および固有機能」は以下のとおりです。

FJVM固有機能を除き、Java HotSpot Client VMも同様です。

- New世代領域用制御処理並列化機能付きGC（パラレルGC）
- コンカレント・マーク・スイープGC付きパラレルGC（CMS付きパラレルGC）
- オブジェクト参照の圧縮機能 (注)
- ガーベジコレクションのログ出力
- コンパイラ異常発生時の自動リカバリ機能 (注)
- 長時間コンパイル処理の検出機能 (注)
- 動的コンパイル発生状況のログ出力機能 (注)
- クラスのインスタンス情報出力機能
- java.lang.System.gc()実行時におけるスタックトレース出力機能
- Java VM終了時における状態情報のメッセージ出力機能
- ログ出力における時間情報のフォーマット指定機能
- FJVMログ
- メモリ領域不足事象発生時のメッセージ出力機能の強化
- スタックオーバーフロー検出時のメッセージ出力機能

注) FJVMにだけ提供されるFJVM固有機能です。

9.1.4 仮想メモリと仮想アドレス空間

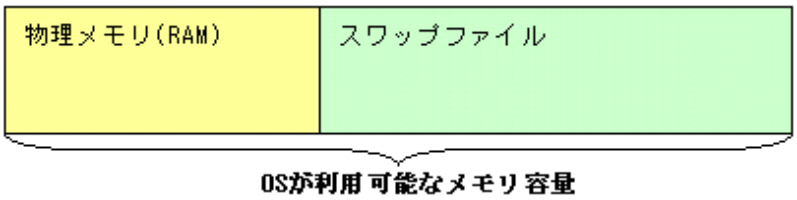
Javaアプリケーションを開発・運用するにあたり、OSのメモリ管理の概要を知っておく必要があります。本節では、OSの一般的なメモリ管理技法の1つである**仮想アドレス空間**を説明します。

なお、**仮想アドレス空間**の具体的なアーキテクチャーはOSごとに異なりますが、本節では各OSに共通する内容を説明します。

仮想メモリ

OSは、物理メモリ(RAM)だけでなくスワップファイルを活用することにより、多くのメモリ領域を使用することができます。このテクノロジーを**仮想メモリ**といいます。**仮想メモリ**の容量は、RAMとスワップファイルのサイズの合計になります。

図9.1 OSが利用可能なメモリ容量



ただし、ハードディスクへのアクセスはRAMより低速なので、メモリのスワッピングは性能に大きく影響を与えますので、注意が必要になります。

仮想アドレス空間

OS上でプログラムを起動すると、OSがプログラムを実行・管理する単位としてのプロセスが生成されます。同様にして、Javaアプリケーションを起動すると、Javaプロセスが生成されます。

OS上で生成されたプロセスには、**仮想アドレス空間**が割り当てられます。**仮想アドレス空間**は、それぞれのプロセスで独立したものであり、あるプロセスから別のプロセスの**仮想アドレス空間**にアクセスすることはできません。マシンに積んでいる物理メモリ(RAM)のサイズとは関係なく、**仮想アドレス空間**のサイズは、常に一定です。たとえば、32ビットアーキテクチャーのOSの場合、物理メモリ(RAM)のサイズに依存せず、**仮想アドレス空間**のサイズは常に4GB(2の32乗バイト)です。

このため、大量の**仮想メモリ**を用意しても、1つのプロセスで使用できるメモリ量の上限は**仮想アドレス空間**の上限になりますから、プロセスが**仮想アドレス空間**の大きさ(実際には後述のユーザ空間の大きさ)を超えるメモリ量を必要とする場合は、メモリ不足の状態になります。

逆に、プロセスが必要とするメモリ量が仮想アドレス空間の大きさ(実際には後述のユーザ空間の大きさ)の上限未満だったとしても、そのメモリ量に相当する仮想メモリ量がOS上になければ、メモリ不足の状態になります。

また、大量の**仮想メモリ**を用意しても、OS上で大量のプロセスが動作していて**仮想メモリ**を大量に消費している状態であれば、各プロセスに割り当てられた**仮想アドレス空間**を使い切っていなくても、メモリ不足の状態になる場合があります。

ユーザ空間

プロセスが持つ**仮想アドレス空間**のうち、実際にプロセスが使用できる空間を、**ユーザ空間**といいます。**ユーザ空間**には、プログラムの実体(Windows(R)でJavaアプリケーションを実行する場合は、java.exeなど)がコピーされるだけでなく、スタックやヒープなどのさまざまなセグメントがあります。更に**ユーザ空間**は、実行するプログラムだけでなく、そのプログラムを実行させるためのOS側のプログラムなどでも使用します。Javaプロセスの**ユーザ空間**の場合には、前述の各セグメントの他に、Javaオブジェクトを格納するセグメント(=Javaヒープ)があります。このため、Javaアプリケーションをチューニングする際の対象となるJavaヒープのサイズの上限値は、**ユーザ空間**のサイズよりも少なくなります。

Javaアプリケーションのチューニングを実施する際は、仮想メモリの容量やプロセスの使用状況など、システムの状態を考慮する必要があります。なおOSの種類やアプリケーションの実行モードによって、**ユーザ空間**として使用できる上限値が異なるため、注意が必要です。

なお、各セグメント獲得・解放時の実際の制御処理はOSが行います。そのため、各セグメントに関する管理方法/動作仕様/大きさについては、JDK/JREを実行する各OSの仕様に依存します。そして、プロセス外部からは**ユーザ空間**に空きが存在するように見える場合であっても、OSの制御処理上は利用できる空きが無いと判断され、セグメントに対応するメモリ領域が獲得できず、プロセス動作が異常となる場合がありますので注意してください。
また、OSの制御処理上、各セグメント間には、アプリケーションから利用できない隙間領域が存在する場合があります。そのため、通常、ユーザ空間としての上限値まで仮想メモリを使用することはできません。プロセスが使用するメモリ量として、ユーザ空間の上限値まで使えることを前提とした設計にはしないでください。

実行モード

実行モードが変わると、プログラムの実行に必要な基本メモリ量が変わります。
具体的には、プログラムの実行モードを32ビットモードから64ビットモードに変更して実行する場合、ポインタを扱うために必要となるメモリ量の単位が、4バイトから8バイトになります(OSによっては、整数を扱う際に必要となるメモリ量の単位も異なります)。そのため、同じソースから作られたプログラムの場合であっても、64ビットOS上で64ビットモードのプログラムを使ってアプリケーションを動作させる場合は、32ビットOSまたは64ビットOS上で32ビットモードのプログラムを使ってアプリケーションを動作させる場合よりも、より大きなメモリ量が必要となります。

64ビットモード時のCデータ型モデル

- Solaris64 Linux64
  LP64モデル：long型/ポインタが32ビット(4バイト)から64ビット(8バイト)へ変更されます。
- Windows64
  P64モデル ：ポインタが32ビット(4バイト)から64ビット(8バイト)へ変更されます。

Javaアプリケーションの場合もC/C++アプリケーションの場合と同様、実行モードが変わると、プログラムの実行に必要な基本メモリ量が変わります。
特に、オブジェクトを構成するデータ内容にはポインタを扱う情報も多数あるため、64ビットモードの場合、1オブジェクトあたりに必要となるメモリ域の大きさは32ビットモードの場合よりも大きくなります。
そのため、通常、64ビットOS上で64ビットモードのJDK/JREを使ってJavaアプリケーションを動作させる場合、32ビットOSまたは64ビットOS上で32ビットモードのJDK/JREを使ってJavaアプリケーションを動作させた場合の設定に対して、1.5～2倍のJavaヒープ量が必要です。

9.1.5 スタック

ここでは、**スタック**について簡単に説明します。

プログラムがスレッドを生成すると、OSがそのスレッドに対して**スタック**と呼ばれるメモリ領域をそのスレッドの終了時まで自動的に割り当てます。**スタック**は、スレッド上で実行されるメソッドや関数が使用するローカル変数などの一時的なデータを格納するための作業域として使用されます。

スレッド上では多くのメソッドや関数が入れ子状態で呼び出されるので、これらのメソッドや関数が使用する一時的な作業域を管理するために、OSは**フレーム**と呼ばれる区切りで個々の作業域をスタック上に積み上げることで管理しています。(メソッドや関数が呼び出されるごとに、これらが使用する作業域を**フレーム**という区切りでスタック上に積み上げ、**フレーム**内のデータは呼び出したメソッドや関数からの復帰時に破棄されます。)

このため、あるメソッドを再帰的に無限に呼び出すなどしてメソッドを深く呼び出した場合や、非常に大きなサイズのローカル変数などを使用するメソッドを呼び出した場合などには、**スタック**域の枯渇により、スタック上に**フレーム**を積み上げることができなくなり、スタックオーバーフローが発生する場合があります。

Javaアプリケーションの場合、通常、スタックオーバーフローが発生すると、java.lang.StackOverflowErrorがスローされます。ただし、Javaプロセス中のネイティブモジュール実行時の処理でスタックオーバーフローが発生した場合には、java.lang.StackOverflowErrorはスローされません。なお、FJVMを使用しているJavaプロセス中のネイティブモジュール実行時にスタックオーバーフローが発生した場合は、“9.6.3.1 スタックオーバーフロー検出時のメッセージ出力機能”によってスタックオーバーフローの発生を検出できる場合があります。

注意

スタック領域の管理

スタック領域の実際の管理はOSが行います。そのため、**スタック**領域に関する管理方法/動作仕様/大きさについては、JDK/JREを実行する各OSの仕様に依存します。
なお、Javaアプリケーション実行時のスタックオーバーフロー発生をJava VMが検出できる様にするために、**スタック**領域の一部をJava VMの制御用領域として使用します。そのため、Javaアプリケーションから使用できる**スタック**領域の大きさは、**スタック**として割り当てられた領域の大きさよりも若干小さな大きさになります。

注意

クラスファイル実行時のスタック領域の利用

Javaの実行環境であるJava VMが起動された後、Java VMは、実行対象プログラムであるクラスファイルを読み込み、クラスファイルを以下の2つの方法を用いて実行します。

- インタプリタによるバイトコードの実行
- 動的コンパイルにより、バイトコードを機械命令に翻訳してから実行
  あるクラスファイル中の同じJavaメソッドを実行する場合であっても、Java VMによる実行方法が異なる場合は、実行時に使用されるスタックの大きさが異なります。
  なお、クラスファイルの実行方法については、“9.3 動的コンパイル”を参照してください。

注意

ユーザ空間のメモリ不足によるスレッド生成エラー

スタックは、プロセス内のユーザ空間から割り当てられます。
そして、ユーザ空間のメモリ不足により**スタック**などを割り当てることができなった場合には、スレッド生成処理でエラーが発生します。
Javaプロセスの場合、Javaヒープなどのサイズを大きく確保すると、その分だけ**スタック**として割当て可能な領域が減少するため、Javaプロセス内で生成可能なスレッドの個数も減少します。
このため、Javaプロセス内のスレッド生成処理がエラーとなる要因の1つとして、Javaヒープなどのサイズを大きく確保したことによるユーザ空間のメモリ不足が考えられます。

注意

Javaプロセスの消滅 Windows32/64

Windows(R)では、Javaプロセス内でスタックオーバーフローが発生した場合、システム状態やプログラム状態によっては、OSからFJVMやWindowsエラー報告(Windows Error Reporting(WER))に制御が渡らず、痕跡を残さずにJavaプロセスが消滅する場合があります。
なお、Windowsエラー報告の説明は、"9.5.9.1 クラッシュダンプ"を参照してください。

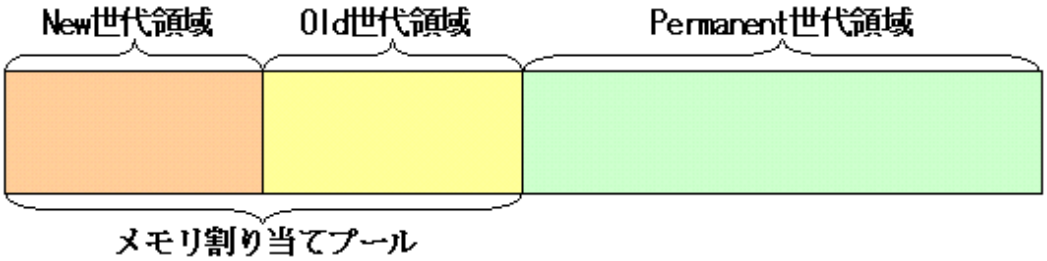
9.1.6 Javaヒープとガーベジコレクション

ここでは、**Javaヒープ**と**ガーベジコレクション(GC)**を説明します。

Javaヒープは、Javaプロセス内に存在するJavaオブジェクトを格納するための領域です。
Javaヒープは、**New世代領域**、**Old世代領域**および**Permanent世代領域**に大別され、Java VMが管理・制御します。なお、**New世代領域**と**Old世代領域**は、**メモリ割り当てプール**という形で各領域を合わせて一括的に管理・制御します。

Java VMは、Javaアプリケーションの実行時に、Javaオブジェクトを**Javaヒープ**の各領域に格納します。**Javaヒープ**の空き容量がなくなった場合は、java.lang.OutOfMemoryErrorがスローされます。
また、不要となったJavaオブジェクトは**GC**により回収され、**Javaヒープ**の空き領域が増加します。なお、**New世代領域**に存在する不要となったJavaオブジェクトを回収する**GC**処理を「**NewGC**処理(または**マイナーGC**処理)」といいます(NewGCまたはマイナーGCと略す場合もあります)。また、**New世代領域**だけでなく、**Old世代領域**や**Permanent世代領域**を含む**Javaヒープ全体**に存在する不要となったJavaオブジェクトを回収する**GC**処理を「**FullGC**処理」といいます(FullGCと略す場合もあります)。

図9.2 Javaヒープの構造



- **New世代領域**と**Old世代領域(メモリ割り当てプール)**

  New世代領域と**Old世代領域**は、インスタンスや配列などのJavaオブジェクトを管理する領域です。
  New世代領域は寿命の短いJavaオブジェクトを管理します。通常、Javaアプリケーションで要求されたオブジェクトは、**New世代領域**に生成されます。一定期間**New世代領域**で生存したJavaオブジェクトは、**GC**処理によって**Old世代領域**に移動されます。そして、**Old世代領域**に存在する不要となったJavaオブジェクトは、**FullGC**処理によって回収されます。
  領域がNew世代とOld世代に分かれるのは、世代別に**GC**処理を実行するためです。
  なお、**メモリ割り当てプール**全体のサイズの初期値および最大値は、それぞれ、"-Xms"オプションおよび"-Xmx"オプションで指定することができます。

- **Permanent世代領域**

  Permanent世代領域は、Javaのクラス、メソッドや定数など、永続的に参照される静的オブジェクトを管理する領域です。
  Permanent世代領域に存在する不要となったオブジェクトは、**FullGC**処理によって回収されます。
  なお、**Permanent世代領域**の大きさの初期値および最大値は、それぞれ、“-XX:PermSize”オプションおよび“-XX:MaxPermSize”オプションで指定することができます。

ユーザ空間

Javaヒープは、Javaプロセスのユーザ空間内に割り当てられます。

また指定された最大値まで**Javaヒープ**が利用できる環境にするため、Java VM起動時に、**メモリ割り当てプール**および**Permanent世代領域**を、各最大値の大きさで連続領域としてリザーブします(同一プロセス内の他の処理から使用できないようにします)。

このため、**Javaヒープ**の最大値を大きく設定すると、その分だけスタックなど他の処理に割り当てられるメモリ領域が減少します。

別な言い方をすれば、ユーザ空間内で使用できるメモリ量の上限から、Javaアプリケーション自身、Java VM、そしてネイティブモジュール(OSを含む)が使用するメモリ量などを差し引いた値が、**Javaヒープ**として使用できる上限となります。

なおJava VM起動時に指定された**Javaヒープ**の大きさが利用できない場合、Java VMは以下のメッセージを出力し、Javaプロセスを終了させます。

```
Error occurred during initialization of VM
Could not reserve enough space for object heap
```

このメッセージが出力された場合は、**Javaヒープ**を小さくするチューニングを行ってください。

またSolaris用およびLinux用のJava VMにおいて、Java VM起動時またはJavaアプリケーション実行中に「ユーザ空間不足」または「仮想メモリ不足」が発生した場合、Java VMは以下のメッセージを出力し、Javaプロセスを終了させます。

- Java VM起動時の場合

```
制御名: mmap failed: errno=エラー情報, 制御情報....
Error occurred during initialization of VM
mmap failure
```

- Javaアプリケーション実行中の場合

```
制御名: mmap failed: errno=エラー情報, 制御情報....
(上記メッセージに続いて、java.lang.OutOfMemoryErrorメッセージが出力される場合があります。)
```

制御名:

メモリ不足が発生した際のJava VMの制御名

エラー情報:

メモリ不足が発生した際のJava VMのエラー情報

制御情報:

メモリ不足が発生した際のJava VMの制御情報

ユーザ空間が不足している場合は、Javaヒープを小さくするチューニングを行ってください。

仮想メモリが不足している場合は、他の不要なプロセスを終了して仮想メモリに余裕を持たせるか、物理メモリ(RAM)またはスワップファイルを拡張して仮想メモリを増やすようにチューニングを行ってください。

Javaヒープにメモリを割り当てるタイミング

Java VMは、OSの仮想メモリ資源を効率的に利用するために、起動時に**Javaヒープ**の各領域の初期値を割り当て、各領域の最大値まで段階的に拡大する制御方法を用いています。
具体的には、Javaプロセスの起動時は**メモリ割り当てプール**および**Permanent世代領域**の各初期値のサイズを割り当てます。その後、各領域のサイズは各領域の最大値までの範囲内で、GC処理後や、オブジェクトの生成時のタイミングで、その時々の最適な値に自動調整されます(GC処理としてメモリ割り当てプールの最小使用量保証機能を無効にしたパラレルGCを使用している場合は、**メモリ割り当てプール**の大きさが、初期値よりも小さくなる場合があります)。

スワップ発生時の対処方法

各領域を拡大していく過程で、OS側で物理メモリの資源をディスクにスワップする処理が発生する場合があります。このスワップ処理により、各領域を拡大する**FullGC**処理に時間がかかる場合があります。**FullGC**処理中のスワップ処理発生による時間遅延が問題になる場合は、**Javaヒープ**の各領域に対する初期値と最大値は同じ値に指定してください。
パラレルGCについては、"9.2.3 New世代領域用制御処理並列化機能付きGC(パラレルGC)"を参照してください。

9.1.7 FJVMに対して指定可能なチューニング用オプション

Javaヒープのチューニングなど、FJVMに対して指定可能な**Java VMのチューニングに関するオプション**を、以下に示します。

各オプションの使用法については、本マニュアルを参照してください。
なお、本マニュアル内で具体的に説明していない**Java VMのチューニングに関するオプション**は、FJVMではサポートしておりません。

Javaヒープチューニング用のオプション

```
-Xms
-Xmx
-XX:NewSize
-XX:MaxNewSize
-XX:NewRatio
-XX:SurvivorRatio
-XX:TargetSurvivorRatio
-XX:PermSize
-XX:MaxPermSize
```

スタックサイズチューニング用のオプション

```
-Xss
-XX:CompilerThreadStackSize
```

使用するガーベジコレクション処理を選択するオプション

```
-XX:+UseSerialGC
-XX:+UseParallelGC
-XX:UseFJcmsGC
```

ガーベジコレクション処理のチューニング用オプション

```
パラレルGC用:
-XX:ParallelGCThreads
-XX:+UseAdaptiveSizePolicyMinHeapSizeLimit
-XX:-UseAdaptiveSizePolicyMinHeapSizeLimit
-XX:+AutomaticallyJavaHeapSizeSetting
-XX:GCTimeLimit
-XX:GCHeapFreeLimit
```

-XX:+UseGCOverheadLimit

CMS付きパラレルGC用:
-XX:ParallelGCThreads
-XX:ConcGCThreads

共通:
-XX:-UseCompressedOops (64ビットモード版JDK/JREの場合)

チューニングの際に使用するログ出力などのデバッグ用オプション

ガーベジコレクションのログ出力用:
-verbose:gc
-XX:+UseFJverbose
-XX:+ClassUnloadingInfo
-Xloggc

動的コンパイルのログ出力用:
-XX:+PrintCompilationCPUTime
-XX:+FJPrintCompilation

ログ出力共通:
-XX:FJverboseTime

その他:
-XX:-OmitStackTraceInFastThrow
-XX:+PrintClassHistogram
-XX:+PrintJavaStackAtSystemGC
-XX:+VMTerminatedMessage
-Xcheck:jni
-XX:+PrintCompilerRecoveryMessage
-XX:CompileTimeout

9.1.8 Java Native Interface(JNI)

アプリケーションミスのトラブルの確実な事前防止のためには、(CやC++等を使用した)ネイティブプログラムを、Java Native Interface(JNI)経由で利用してはいけません。実現しようとしている機能が、Javaにより記述できないか設計段階で十分に検討を行ってください。やむをえず利用する場合でも、JNIの利用は最小限にし、十分な確認とデバッグを実施してください。

JNIを利用する場合の前提スキルとして、以下は必須です。

- C/C++によるマルチスレッドプログラミングの経験がある
- トラブル発生時、自分でデバッグできる

注意

ファイナライズ処理を期待したリソース管理は、行わないでください。JNI関連で、もっともトラブルが多いのは、ネイティブプログラム側で確保したメモリの後処理漏れです。例えば、次のようなプログラミングをしてはいけません。

```
------- Java ---------------
class A {
  native long nativeAlloc();
  native void nativeFree(long a);
  long address;
  A() {
    address = nativeAlloc();
  }
```

```
  public void finalize() {
    nativeFree(address);
  }
}
------- Java ----------------
------- C -----------------
JNIEXPORT jlong JNICALL Java_A_nativeAlloc(JNIEnv *env, jobject o)
{
  return (jlong)malloc(10);
}
JNIEXPORT void JNICALL Java_A_nativeFree(JNIEnv *env, jobject o, jlong p)
{
  free((void*)p);
}
------- C -------------------
```

注意

エラー処理は、必ず行ってください。ネイティブプログラム側でJNI関数呼び出しをしたとき、色々なケースでJavaレベルのエラーになることがあります。このエラーに対する後処理をネイティブプログラム側で行わないと、例外がスローされている状態のままになり、それ以降のJNI関数の呼び出しに失敗し、アプリケーションが正しく動作しません。

なお「JNI関数」とは、以下のJNI仕様書に記述されている関数のことです。

http://docs.oracle.com/javase/jp/7/technotes/guides/jni/spec/functions.html

これらを使う場合は、その度にExceptionOccurredを使用しエラーチェックをする必要があります。

Solaris64 Linux32/64

Solaris、Linux上でJNIを利用される場合は、連携するネイティブプログラムまたは、ネイティブライブラリにおいてシグナルハンドラの書き替えを絶対に行わないでください。
なお、JDK/JRE 1.4.0以降でサポートされたシグナル連鎖機能(Signal Chaining)を利用する場合、マルチスレッド環境におけるシグナル動作など、OSやJava VM自身のシグナル動作およびプログラミングについての知識が前提として必要です。安定稼動が要求されるシステムの設計においては、この機能の使用もお勧めできません。

9.2 ガーベジコレクション(GC)

ガーベジコレクション(GC)について説明します。

9.2.1 FJVMでサポートされるガーベジコレクション処理

FJVMでサポートされるガーベジコレクション(GC)処理には、Javaヒープの**New**世代領域に対するGC制御方法の違いにより、以下の3種類があります。

- 標準GC(シリアルGC)
- New世代領域用制御処理並列化機能付きGC(パラレルGC)
- コンカレント・マーク・スイープGC付きパラレルGC(CMS付きパラレルGC)

なお、JDK/JREのバージョン、実行モード、Java VMの種類により、サポートされるGCの種類が異なります。また、Java VMの種類により、デフォルトで動作するGCが異なります。
“表9.2 サポートされるGCの種類”に、サポートされるGCの種類とデフォルトのGCを示します。

表9.2 サポートされるGCの種類

	JDK/JREの実行モード	32ビットモード		64ビットモード※
	Java VMの種類	Client VM	FJVM	FJVM
シリアルGC		◎	○	○

	JDK/JREの実行モード	32ビットモード		64ビットモード※
	Java VMの種類	Client VM	FJVM	FJVM
パラレルGC		○	◎	◎
CMS付きパラレルGC		□	□	□

○:

サポートされるGC

◎:

サポートされるGC、かつデフォルトのGC

□:

Interstage Application Server Enterprise EditionでだけサポートされるGC

※実行モードが64ビットモードのClient VMは提供していません。

GCは、デフォルトのGCの使用を推奨します。
通常、デフォルトのGCを変更する必要はありません。

注意

New世代領域用GC制御に対し「New世代領域サイズ自動調整機能」を追加して構成されたGC処理「**FJGC**」を提供していません。

Java HotSpot Client VMに対して"-XX:+UseFJGC"オプションを指定した場合は、以下のエラーメッセージが標準エラー出力へ出力され、Javaプロセスの起動に失敗します。

FJVMに対して"-XX:+UseFJGC"オプションを指定した場合は、以下ののワーニングメッセージが標準エラー出力へ出力され、オプションの指定は無効になります。

- Java HotSpot Client VMの場合

```
Unrecognized VM option '+UseFJGC'
```

- 実行モードが32ビットモードのFJVMの場合

```
warning: -XX:+UseFJGC is not supported in Java HotSpot Server VM.
```

- 実行モードが64ビットモードのFJVMの場合

```
warning: -XX:+UseFJGC is not supported in Java HotSpot 64-Bit Server VM.
```

New世代領域の使われ方

NewGC処理では、New世代領域を「eden space」、「from space」および「to space」の3つの内部領域に細分割し、当該領域上において、一般に世代別GC制御と言われる制御方法を用いて、Javaアプリケーションが生成要求したオブジェクトを管理・制御しています。
このうち、「from space」および「to space」は、Java VMがNewGC処理を行う際の作業域的な役割を持つ領域となっています。そのため、「from space」および「to space」の各領域が占める大きさのうち、Javaアプリケーションからのオブジェクト生成要求のために使用される大きさは、その領域の一部分だけとなります。
そのため、ガーベジコレクションのログ出力などの出力データにおいて、メモリ割り当てプールやNew世代領域に空きがあるように見える場合であっても、実際には空きがない場合があります（空きがあるように見える場合であっても、その差は、NewGC処理用の作業域的な役割で使用済となっている場合があります）。

ガーベジコレクション処理の実行抑止

"ガーベジコレクション処理の実行が抑止される機能"に示す機能を使用するJavaアプリケーションを実行すると、Javaヒープ内に存在する全オブジェクトの移動が禁止されるクリティカルセクションと呼ばれる状態が、当該機能の利用状況に応じて発生

します。
クリティカルセクション状態発生時は、オブジェクトの移動が禁止されるため、オブジェクト移動が必須となるGC処理の実行が抑止される状態となります。
Java VMは、Javaアプリケーションにより GC処理の実行が抑止されている際に発生したオブジェクト生成要求に対し、New世代領域に空きが無い場合には、Old世代領域の空き領域を一時的に使用して対応します。
そして、要求されたオブジェクト量を満たす空きがOld世代領域にない場合には、java.lang.OutOfMemoryError例外を発生させます。
そのため“ガーベジコレクション処理の実行が抑止される機能”の機能を多用するJavaアプリケーションの場合は、GC処理実行が抑止される状態も多数発生する可能性が高くなり、当該機能を多用しないJavaアプリケーションに比べ、GC処理実行抑止に因るjava.lang.OutOfMemoryError例外が発生しやすい状態にあります。
特にOld世代領域が小さい状態でJavaアプリケーションを実行した場合は、Old世代領域の空きとなり得る最大値（仮にOld世代領域を全く使用しない場合だと、Old世代領域自身の大きさ）もその大きさに比例して小さいため、その傾向が強まることがあります。

なおFJVMでは、GC処理の実行抑止によりjava.lang.OutOfMemoryError例外が発生したかどうかの情報を出力する機能を提供しています。
詳しくは“9.6.1.1 メモリ領域不足事象発生時のメッセージ出力機能の強化”を参照してください。

ガーベジコレクション処理の実行が抑止される機能

- GC処理の実行が抑止される状態となるJNI関数
  - GetPrimitiveArrayCritical()実行からReleasePrimitiveArrayCritical()実行までの間
  - GetStringCritical()実行からReleaseStringCritical()実行までの間
- GC処理の実行が抑止される状態となるJVMPI関数(注)
  - DisableGC()実行からEnableGC()実行までの間
- GC処理の実行が抑止される状態となるJVMPIイベント(注)
  - JVMPI_EVENT_THREAD_START
  - JVMPI_EVENT_CLASS_LOAD
  - JVMPI_EVENT_CLASS_UNLOAD
  - JVMPI_EVENT_JNI_GLOBALREF_ALLOC
  - JVMPI_EVENT_JNI_GLOBALREF_FREE
  - JVMPI_EVENT_JNI_WEAK_GLOBALREF_ALLOC
  - JVMPI_EVENT_JNI_WEAK_GLOBALREF_FREE
  - JVMPI_EVENT_OBJECT_ALLOC
  - JVMPI_EVENT_MONITOR_CONTENDED_ENTER
  - JVMPI_EVENT_MONITOR_CONTENDED_ENTERED
  - JVMPI_EVENT_MONITOR_CONTENDED_EXIT
  - JVMPI_EVENT_MONITOR_WAIT
  - JVMPI_EVENT_MONITOR_WAITED
  - JVMPI_EVENT_HEAP_DUMP
  - JVMPI_EVENT_METHOD_ENTRY
  - JVMPI_EVENT_METHOD_ENTRY2
  - JVMPI_EVENT_METHOD_EXIT

  注) Java Virtual Machine Profiling Interface(JVMPI)をサポートしていません。

JVMPIとJVMTI

Java Virtual Machine Profiling Interface(JVMPI)をサポートしていません。

JVMPI相当の機能を使用する場合には、Java Virtual Machine Tool Interface(JVMTI)を使用してください。

RMI処理の分散GC

JavaのRMI処理では、クライアントで不要となった参照に対するサーバ側のオブジェクトを破棄するため、Distributed GC(分散GC)という処理を行います。その処理の一貫として、以下のプロパティで設定された時間間隔(デフォルトの時間間隔は1時間)ごとに、java.lang.System.gc()実行によるFull GCが発生します。なお、分散GCとメモリ不足等による通常のガーベジコレクション(GC)が同時に発生した場合は、メモリ不足等による通常のGCが実行され、分散GCによるFullGCは実行されません（メモリ不足等による通常のGCがNewGC処理だった場合は、FullGCにはなりません）。

```
-Dsun.rmi.dgc.server.gcInterval=時間間隔(ミリ秒)
-Dsun.rmi.dgc.client.gcInterval=時間間隔(ミリ秒)
```

分散GCは独自のタイマー制御の元で実行されるため、メモリ不足等による通常のGCの実行とは関係せずに実行されます。そのため、GC処理の結果ログを見た場合、Javaアプリケーションとしての処理がほとんど動作していない時間帯やメモリ不足とは思われない状態のときに、FullGC処理が実行されているように見える場合があります。
またプロパティで指定された時間間隔が短い場合、Interstage Application Serverの予兆監視機能により警告を受ける(OM3204メッセージが出力される)場合がありますので、注意してください。

9.2.2 標準GC（シリアルGC）

New世代領域用GC制御に対して何も付加機能を追加していない「標準機能のみ」で構成されたGC処理です。後述の**パラレルGC**との対比から、標準GCのことを**シリアルGC**と呼ぶ場合もあります。

- **Java HotSpot Client VM**、および以下のオプションを指定した場合の**FJVM**では、**シリアルGC**が実行されます。なお、以下のオプションは、後述の**FJGC**によるGC制御を無効にする、またはシリアルGCによるGC制御を有効にするオプションです。

  ```
  -XX:+UseSerialGC
  ```

- **シリアルGC**使用時に利用可能となる「Javaヒープのチューニング用オプション」です。

  ```
  -Xms
  -Xmx
  -XX:NewSize
  -XX:MaxNewSize
  -XX:NewRatio
  -XX:SurvivorRatio
  -XX:TargetSurvivorRatio
  -XX:PermSize
  -XX:MaxPermSize
  ```

9.2.3 New世代領域用制御処理並列化機能付きGC（パラレルGC）

New世代領域用GC制御に対し、「当該処理を並列化して実行する機能」を追加して構成されたGC処理です。New世代領域用のGC制御を並列化して実行することから、このGCを**パラレルGC**と呼ぶ場合もあります。

パラレルGCを有効にする場合に指定するオプション

FJVMの場合は、デフォルトでこのGC処理が実行されます。

```
-XX:+UseParallelGC
```

Javaヒープチューニング用オプション（パラレルGC使用時）

パラレルGC使用時に利用可能となる「Javaヒープのチューニング用オプション」です。

なお、パラレルGC使用時は、JavaヒープのNew世代領域およびOld世代領域の大きさに関する値が自動的に調整および最適化されるため、通常、New世代領域の大きさやNew世代領域とOld世代領域の大きさのバランスをチューニングするためのオプションを使用する必要はありません。

```
-Xms
-Xmx
-XX:NewSize (*)
-XX:MaxNewSize (*)
-XX:NewRatio (*)
-XX:PermSize
-XX:MaxPermSize
```

(*) New世代領域の大きさや、New世代領域とOld世代領域の大きさのバランスをチューニングするためのオプションです。

注意

GC処理用スレッド数

パラレルGCを使用した場合は、実行するハードウェアに搭載しているCPU数に依存した数のGC処理用スレッドがJavaプロセス内に作成されます。そのため、GC処理用スレッドの数分だけ、スタック域などのスレッド用のメモリ領域が必要となります。

Javaプロセス内でのメモリ量を抑えるためなど、GC処理用スレッドの数を調整する場合には、以下のオプションでGC処理用スレッドの数を指定することにより、GC処理用スレッドの数を調整することができます。

なおGC処理用スレッドの数を抑制した分だけGC処理における性能がおちる場合もありますので、このオプションを用いる場合には、十分な性能確認を実施してください。また、一般的に、CPU数以上の数のGC処理用スレッドを作成しても、GC処理における性能向上にはつながりません。

- パラレルGCで使用するGC処理用スレッドの数を指定するオプション

```
-XX:ParallelGCThreads=New世代領域GC用スレッドの数
```

New世代領域用GC処理を行うGCスレッドの数を指定します。
「0」が指定された場合は、デフォルト値となります。デフォルト値は以下のとおりです。

— 実行するハードウェアに搭載しているCPU数が7以下の場合 = CPU数分

— 実行するハードウェアに搭載しているCPU数が8以上の場合 = 8

注意

メモリ割り当てプールの省略値自動調整機能

FJVMのパラレルGCでは、エルゴノミクス機能によるメモリ割り当てプールの初期値(-Xms)および最大値(-Xmx)の省略値自動調整機能（マシンの物理メモリ量などに応じて、-Xmsおよび-Xmxの各オプションに対する省略値を自動的に決定する機能）を無効にしています。

FJVMで、エルゴノミクス機能によるメモリ割り当てプールの省略値自動調整機能を有効にする場合は、以下のオプションを指定してください。

ただし、このオプション指定は、システムのメモリ資源不足の要因となる場合があるため、システム内に複数のJavaプロセスを起動、実行する場合には使用しないでください。

- メモリ割り当てプールの省略値自動調整機能を有効にするオプション

```
-XX:+AutomaticallyJavaHeapSizeSetting
```

注意

メモリ領域不足事象の検出機能

FJVMのパラレルGCでは、エルゴノミクス機能によるメモリ領域不足事象の検出機能（以下の各オプション指定値による条件が同時に成立した場合に、メモリ領域不足事象（java.lang.OutOfMemoryError）として検出する機能）を無効にしています。

- メモリ領域不足事象検出条件

  -XX:GCTimeLimit=GC処理に要する時間の上限値 (デフォルトは98)

  Javaアプリケーションの合計処理時間に対して、GC処理に要した時間の上限値をパーセント単位(%)で指定します。
  指定された上限値を超えた場合、検出条件の一方が成立します。

  -XX:GCHeapFreeLimit=GC処理後のJavaヒープ量の空きスペースの下限値 (デフォルトは2)

  メモリ割り当てプールの最大値に対する、GC処理後のJavaヒープ量の空きスペースの下限値をパーセント単位(%)で指定します。
  指定された下限値を下回った場合、検出条件の一方が成立します。

FJVMでエルゴノミクス機能によるメモリ領域不足事象検出機能を有効にする場合は、以下のオプションを指定してください。

ただし、このオプション指定で検出されるメモリ領域不足事象は、Javaヒープの使用量だけではなく、メモリ領域不足事象検出用オプションで指定された値、およびガーベジコレクション処理の動作状況から得られた統計情報などを元に決定されるため、Javaヒープの使用量が不足していない状態であっても、メモリ領域不足事象が検出される場合がありますので注意してください。

- メモリ領域不足事象検出を有効にするオプション

  -XX:+UseGCOverheadLimit

注意

パラレルGC処理のエルゴノミクス機能およびメモリ割り当てプールの最小使用量保証機能

FJVMでパラレルGCを使用した場合、パラレルGC処理のエルゴノミクス機能（メモリ割り当てプール内の各世代領域サイズの動的変更機能）により、Javaアプリケーションの実行状況や負荷／GC処理に掛かる時間などの情報から、GC処理としての最適動作状態になるように、メモリ割り当てプール内の各世代領域サイズが自動的に調整・変更および最適化されます。

その際、メモリ割り当てプールの使用量が-Xmsオプションで指定した値（メモリ割り当てプールの初期値）よりも小さくなることあります（-Xmsオプションで指定した値よりも下回るメモリ割り当てプールの使用量で、GC処理としての最適動作状態になることがあります）。

パラレルGCを使用してJavaアプリケーションを実行する際、メモリ割り当てプールの使用量についての操作を行う場合は、以下のオプションを指定してください。

- メモリ割り当てプールの最小使用量保証機能を操作するオプション
  - -Xmsオプションで指定した値よりもメモリ割り当てプールの使用量を小さくしない場合

    -XX:+UseAdaptiveSizePolicyMinHeapSizeLimit

    パラレルGC処理のエルゴノミクス機能動作時、-Xmsオプション指定値によるメモリ割り当てプールの最小使用量保証機能を有効にします。
    Javaアプリケーション実行時に使用されるメモリ割り当てプールの大きさは，「-Xmsオプション指定値」から「-Xmxオプション指定値」の範囲で変動します。（-Xmsオプション指定値と-Xmxオプション指定値が等しい場合、使用中となるメモリ割り当てプールの大きさは変動しません。）
    この状態は、パラレルGCを使用する場合のデフォルト状態です。

  - -Xmsオプションで指定した値よりもメモリ割り当てプールの使用量を小さくして良い場合

-XX:-UseAdaptiveSizePolicyMinHeapSizeLimit

パラレルGC処理のエルゴノミクス機能動作時、-Xmsオプション指定値によるメモリ割り当てプールの最小使用量保証機能を無効にします。
Javaアプリケーション実行時に使用されるメモリ割り当てプールの大きさは，「Java VMとしての下限値」から「-Xmxオプション指定値」の範囲で変動します。(-Xmsオプション指定値と-Xmxオプション指定値が等しい場合であっても、使用中となるメモリ割り当てプールの大きさは変動します。)
なお、Javaアプリケーションの実行時に使用されるメモリ割り当てプールの大きさが-Xmsオプション指定値よりも小さくなった場合、Full GCの発生間隔が、Javaプロセス起動時近辺における発生間隔よりも短くなってしまう場合があります。しかし、パラレルGC処理のエルゴノミクス機能は、GC処理に掛かる時間情報も考慮した上で、GC処理としての最適動作状態となるように調整しています(メモリ割り当てプールを縮小しても、Full GCに掛かる時間が少ない状態の場合に最適動作状態としています)。そのため、メモリ割り当てプールの縮小によりFull GCの発生間隔が短くなった場合でも、Javaアプリケーション実行時の性能に対する影響はほとんどありません。
なお、このオプションは、パラレルGCを動作させたJDK/JRE 5.0で実行していたアプリケーションを、パラレルGCを動作させるJDK/JRE を用いて実行させる際、使用中となるメモリ割り当てプールの大きさに対する挙動の互換性を向上させる必要がある場合に使用します。
通常、このオプションを指定する必要はありません。

なおメモリ割り当てプールの最小使用量保証機能の有効／無効に関わらず、Javaプロセス起動時に使用されるメモリ割り当てプールの大きさは、-Xmsオプションで指定された値となります。

EE

9.2.4 コンカレント・マーク・スイープGC付きパラレルGC（CMS付きパラレルGC）

New世代領域用GC制御を並列化して実行する機能、およびJavaアプリケーションと同時並列に動作するOld世代領域・Permanent世代領域用GC制御「コンカレント・マーク・スイープGC(CMS-GC)機能」を追加して構成されたGC処理です。CMS-GC機能を追加されたパラレルGC制御であることから、このGCを**CMS付きパラレルGC**と呼ぶ場合もあります。

CMS付きパラレルGCによるGC制御は、以下のエディションに搭載されたJDK/JRE にだけ提供しています。

- Interstage Application Server Enterprise Edition

CMS-GCは、Javaアプリケーションが Full GCによって停止されることにより影響を受ける「アプリケーションの応答性能の平準化」を改善するために実行される「Old世代領域内およびPermanent世代領域内の不要オブジェクト回収用のGC機構」です。

Javaアプリケーションを停止させて実行するFull GCに対し、CMS-GCはJavaアプリケーションと同時並列に動作し、Old世代領域内およびPermanent世代領域内の不要オブジェクトを回収します。CMS-GCの実行により、Old世代領域(New世代領域にあるオブジェクトの移動先や巨大オブジェクトの生成先となる領域) およびPermanent世代領域(Javaのクラス、メソッドや定数などの格納先となる領域)の空きを、Javaアプリケーション動作と並行して増加させることができるため、Full GCの発生を抑えることができます。これにより、JavaアプリケーションはFull GCにより停止される影響を受けにくくなり、応答性能平準化の改善が期待できます。

なおCMS-GC動作中は、NewGC処理/Full GC処理の実行開始が遅延する場合があります。そして遅延期間中は、Javaアプリケーションとしての動作も停止します。そのため、ガーベジコレクション処理の結果ログ内のNewGC処理/Full GC処理に対して出力されたGC処理実行時間よりも長い間、Javaアプリケーションとしての動作が停止している場合があります。

CMS付きパラレルGCを有効に場合に指定するオプション

-XX:UseFJcmsGC=タイプ

タイプ:

CMS-GCによる不要オブジェクトの回収対象を「Old世代領域内」とする場合

- type0
- type1
- type2

CMS-GCによる不要オブジェクトの回収対象を「Old世代領域内およびPermanent世代領域内」とする場合

- type0p
- type1p
- type2p

CMS-GCによる不要オブジェクトの回収対象を、Old世代領域内だけでなくPermanent世代領域内にまで拡大した場合、CMS-GCとしての処理対象域が増加することになるため、CMS-GC完了までの実行時間増加に繋がり易くなります。

その結果、実行するアプリケーションや実行環境によっては、CMS-GCによる回収処理が間に合わなくなり、FullGCの発生に繋がる場合もありえます。

そのため、回収対象が「Old世代領域内」で正常動作していた環境において、回収対象を「Old世代領域内およびPermanent世代領域内」にまで拡大する場合は、再度のチューニング作業が必要となります.

・ -XX:UseFJcmsGC=type0またはtype0pが指定された場合

以下は、-XX:UseFJcmsGC=type0およびtype0p指定時に利用可能となる「Javaヒープのチューニング用オプション」です。チューニング用オプションを用いて、利用者が細かくチューニング作業を行うことが可能なCMS付きパラレルGCを使用する場合に指定します。

Javaヒープチューニング用オプション（-XX:UseFJcmsGC=type0およびtype0p指定時）

-Xms -Xmx -XX:NewSize -XX:MaxNewSize -XX:SurvivorRatio -XX:TargetSurvivorRatio -XX:PermSize -XX:MaxPermSize

・ -XX:UseFJcmsGC=type1またはtype1pが指定された場合

New世代領域用GCでのオブジェクト回収を重視した調整が成されたCMS付きパラレルGCを使用する場合に指定します。

実行されるアプリケーションの特徴が「大半のオブジェクトが、少ない回数のNew世代領域用GCによって回収されるオブジェクト」である場合に、CMS-GCによる応答性能平準化の改善効果が得やすい調整になっています。

Javaヒープのチューニングを行う場合は、まず-Xms/-Xmxおよび-XX:PermSize/-XX:MaxPermSizeの各オプションにより、メモリ割り当てプールおよびPermanent世代領域の大きさを調整します。そして必要に応じて、-XX:NewSize/-XX:MaxNewSizeの各オプションでNew世代領域の大きさを調整します。

なおNew世代領域サイズとして、メモリ割り当てプール最大サイズ未満の値を指定できます。ただしNew世代領域サイズを大きくしすぎると、Full GCが発生しやすくなってしまうので注意が必要です。

以下は、-XX:UseFJcmsGC=type1およびtype1p指定時に利用可能となる「Javaヒープのチューニング用オプション」です。

Javaヒープチューニング用オプション（-XX:UseFJcmsGC=type1およびtype1p指定時）

-Xms -Xmx -XX:NewSize -XX:MaxNewSize -XX:PermSize -XX:MaxPermSize

・ -XX:UseFJcmsGC=type2またはtype2pが指定された場合

CMS-GCでのオブジェクト回収を重視した調整が成されたCMS付きパラレルGCを使用する場合に指定します。

実行されるアプリケーションの特徴が「大半のオブジェクトが、何回かのGCを経てから回収される(長期間常駐せず、ある程度の短期間で回収される)オブジェクト」である場合に、CMS-GCによる応答性能平準化の改善効果が得やすい調整になっています。

Javaヒープのチューニングを行う場合は、まず-Xms/-Xmxおよび-XX:PermSize/-XX:MaxPermSizeの各オプションにより、メモリ割り当てプールおよびPermanent世代領域の大きさを調整します。そして必要に応じて、-XX:NewSize/-XX:MaxNewSizeの各オプションでNew世代領域の大きさを調整します。

なおNew世代領域サイズとして、メモリ割り当てプール最大サイズ未満の値を指定できます。ただしNew世代領域サイズを大きくしすぎると、Full GCが発生しやすくなってしまうので注意が必要です。

以下は、-XX:UseFJcmsGC=type2およびtype2p指定時に利用可能となる「Javaヒープのチューニング用オプション」です。

Javaヒープチューニング用オプション（-XX:UseFJcmsGC=type2およびtype2p指定時）

-Xms -Xmx -XX:NewSize -XX:MaxNewSize -XX:PermSize -XX:MaxPermSize

GC処理用スレッド数

CMS付きパラレルGCを使用した場合は、実行するハードウェアに搭載しているCPU数に依存した数のGC処理用スレッドがJavaプロセス内に作成されます。そのため、GC処理用スレッドの数分だけ、スタック域などのスレッド用のメモリ領域が必要となります。

Javaプロセス内でのメモリ量を抑えるためなど、GC処理用スレッドの数を調整する場合には、以下のオプションでGC処理用スレッドの数を指定することにより、GC処理用スレッドの数を調整することができます。

なおGC処理用スレッドの数を抑制した分だけGC処理における性能がおちる場合もありますので、このオプションを用いる場合には、十分な性能確認を実施してください。また、一般的に、CPU数以上の数のGC処理用スレッドを作成しても、GC処理における性能向上にはつながりません。

・ CMS付きパラレルGCで使用するGC処理用スレッドの数を指定するオプション

-XX:ParallelGCThreads=New世代領域GC用スレッドの数

New世代領域用GC処理を行うGCスレッドの数を指定します。

最小値は「2」です。「0」または「1」が指定された場合は、デフォルト値となります。

デフォルト値は以下のとおりです。

- 実行するハードウェアに搭載しているCPU数が1の場合 = 2
- 実行するハードウェアに搭載しているCPU数が2以上7以下の場合 = CPU数分
- 実行するハードウェアに搭載しているCPU数が8以上の場合 = 8

CMS-GC処理用スレッド数

CMS付きパラレルGCを使用した場合は、以下のCMS-GC処理用スレッドがJavaプロセス内に作成されます。そのため、CMS-GC処理用スレッドの数分だけ、スタック域などのスレッド用のメモリ領域が必要となります。

・ CMSスレッド (必ず1つ作成されます)

・ コンカレント・マーク処理専用スレッド

CMSスレッドの他、CMS-GC処理の中のコンカレント・マーク処理を複数スレッドで並列化して実行するために、コンカレント・マーク処理専用スレッドを追加で作成することができます。以下のオプションでCMS-GC処理用スレッドの数を指定することにより、CMS-GC処理用スレッドの数を調整することができます。

なお一般的に、CPU数以上の数のCMS-GC処理用スレッドを作成しても、CMS-GC処理における性能向上にはつながりません。

・ CMS付きパラレルGCで使用するCMS-GC処理用スレッドの数を指定するオプション

-XX:ConcGCThreads=CMS-GC処理用スレッドの数

コンカレント・マーク処理専用スレッドの数を指定します。

最小値は「2」です。オプションの指定がない場合、または「1」が指定された場合は、CMSスレッドだけ生成されます。

「0」が指定された場合は、自動的に以下の値が設定されます。(実行するハードウェアに搭載しているCPU数の1/4(端数切り上げ)をAとした場合)

- Aが1の場合 = 0(CMSスレッドだけ生成されます)
- Aが2以上7以下の場合 = A
- Aが8以上の場合 = 8

なお指定値および自動設定値どちらの場合も、「-XX:ParallelGCThreads」の値よりも大きい場合は、「-XX:ParallelGCThreads」の値となります。

9.2.5 オブジェクト参照の圧縮機能

Javaアプリケーションを64ビットモードのJDK/JRE上で動作させる場合、通常のC/C++アプリケーションの場合と同様、実行モード上の制約から、Javaヒープに格納されるオブジェクト参照(ポインタ情報)で必要となる領域は「64ビット表現/8バイト域」が管理単位となります。

そのため64ビットモードのJDK/JRE上でJavaアプリケーションを動作させる場合、32ビットモードのJDK/JRE上で動作させる場合に対して、1.5～2倍のJavaヒープ量が必要となります。

しかし64ビットモードで実行されるFJVMの場合は、Javaヒープの大きさ(メモリ割り当てプールとPermanent世代領域の大きさの合計)が32GB未満の場合に限り、オブジェクト参照で必要となる領域を「32ビット表現/4バイト域」に圧縮して管理する機能「64ビットモード実行時における**オブジェクト参照圧縮機能**」により、当該機能のない64ビットモードで実行されるJDK/JREよりも少ないJavaヒープ量でJavaアプリケーションが実行できます。

1オブジェクト参照当たりで必要となるJavaヒープの領域が減るため、たとえば極端な例ですが、従来はある大きさに100個のオブジェクト参照を格納できていた場合、当該機能により200個のオブジェクト参照がJavaヒープ上に格納できるようになります。つまり当該機能がないもしくは使用しない場合に必要となるJavaヒープ量を、当該機構を用いた場合に適用した場合、相対的により大きなJavaヒープ値を指定したことと等価となります。その結果、ガーベジコレクション処理の発行回数が減り、アプリケーション実行性能の向上が期待できます。

注意

オブジェクト参照圧縮機能の無効化

本来「64ビット表現/8バイト域」が必要なメモリ域を「32ビット表現/4バイト域」に圧縮して管理するため、オブジェクト参照使用時には、圧縮した情報を「64ビット表現/8バイト域」に展開する必要があります。その圧縮／展開処理は従来よりも余分なCPU消費に繋がるため、Javaアプリケーションとしての実行性能が落ちる可能性が考えられます。

64ビットモードで実行されるFJVMの場合は、デフォルト状態でオブジェクト参照圧縮機能が有効になっています。そのため、Javaアプリケーションとしての実行性能が問題となった場合は、以下のオプションを指定することで、FJVMのオブジェクト参照圧縮機能を無効にすることができます。

- 64ビットモードで実行されるFJVMのオブジェクト参照圧縮機能を無効にするオプション

```
-XX:-UseCompressedOops
```

注) 64ビットモードで実行される場合に指定できます。

なお、Javaヒープの大きさ(メモリ割り当てプールとPermanent世代領域の大きさの合計)が32GB以上であった場合は、以下のメッセージを標準出力へ出力して、オブジェクト参照圧縮機能は自動的に無効となります。

Javaヒープの大きさは、ページサイズなどのシステム情報を元に、Java VMによる制御が最適となるように調整された値となるため、-Xms、-XX:MaxPermSizeなどで指定された値と若干異なる場合があります。このため、これらのオプションで指定したサイズの合計が、32GBより若干小さい場合でもオブジェクト参照圧縮機能が無効になる場合があります。

- FJVMのオブジェクト参照圧縮機能が自動的に無効となった場合に出力されるメッセージ

```
Java HotSpot(TM) 64-Bit Server VM warning: Max heap size too large for Compressed Oops
```

9.2.6 ガーベジコレクションのログ出力

ガーベジコレクション(GC)のログを採取する場合は、以下のオプションを指定します。

GC処理の結果ログを出力するオプション

```
-verbose:gc
```

本オプションの指定により、**GC**が発生するたびに、GC処理の結果ログが標準出力へ1行ずつ出力されます。

GCログの出力フォーマット

```
[GCの種類 GC前のヒープ使用量->GC後のヒープの使用量(ヒープのサイズ), GCの処理時間]
```

GCの種類:

"GC"の場合は**マイナーGC**(または**NewGC処理**)で、"FullGC"の場合は**FullGC**であることを示します。

なお"Javaヒープの中のメモリ割り当てプール"を"ヒープ"と略記しています。

CMS付きパラレルGCを使用している場合は、以下の出力フォーマットのGCログもあります。

```
[GC ヒープ使用量(ヒープのサイズ), マーク処理時間]
```

このフォーマットで出力された場合は、CMS-GC処理のinitialマーク処理またはfinalマーク処理が実行されたことを示します。

なお"Javaヒープの中のメモリ割り当てプール"を"ヒープ"と略記しています。

例

GCログの出力例

```
[GC 80229K->31691K(259776K), 0.4795163 secs]
[FullGC 57654K->4623K(259776K), 0.3844278 secs]
```

なおFJVMでは、GC処理の結果ログ出力機能の強化を行っています。より詳細なGC処理の結果ログ情報を得る場合には、当該機能を使用してください。詳細は、"9.2.6.1 ガーベジコレクション処理の結果ログ出力機能の強化"を参照してください。

注意

ログ出力量の増加

本オプションの指定により、ログ出力が増大します。
本オプションを指定する場合は、ログ出力量についての注意が必要です。

New世代領域の使われ方

NewGC処理では、New世代領域を「eden space」、「from space」および「to space」の3つの内部領域に細分割し、当該領域上において、一般に世代別GC制御と言われる制御方法を用いて、Javaアプリケーションが生成要求したオブジェクトを管理・制御しています。
このうち、「from space」および「to space」は、Java VMがNewGC処理を行う際の作業域的な役割を持つ領域となっています。そのため、「from space」および「to space」の各領域が占める大きさのうち、Javaアプリケーションからのオブジェクト生成要求のために使用される大きさは、その領域の一部分だけとなります。
そのため、ガーベジコレクションのログ出力において、メモリ割り当てプールやNew世代領域に空きがあるように見える場合であっても、実際には空きがない場合があります(空きがあるように見える場合であっても、その差は、NewGC処理用の作業域的な役割で使用済となっている場合があります)。

クラスのアンロード情報

Javaアプリケーションがクラスのアンローディング処理を行っていた場合に、FullGC処理中に該当するクラスのアンロード情報”[Unloading class アンロードされたクラス名]”を結果ログ出力の途中に挿入したい場合は、以下のオプションを指定してください。

- GC処理の結果ログに対してクラスのアンロード情報を出力するオプション

```
-XX:+ClassUnloadingInfo
```

- GC処理の結果ログの格納先ファイルを指定するオプション

```
-Xloggc:GC処理の結果ログの格納先ファイル名
```

GC処理の結果ログおよびクラスのインスタンス情報を、標準出力ではなく、指定したファイルへ出力先を切り換えることができます。

- このオプションを指定すると、自動的に“-verbose:gc”オプションも指定したと見做されます。そのため、“-verbose:gc”オプションの指定がない場合でも、GC処理の結果ログが出力され、また、ガーベジコレクション処理の結果ログ出力機能の強化を使用することもできます。

  なお、“-Xloggc”オプション指定により出力されたGC処理の結果ログの先頭には、「Java VMが起動されてからの経過時間(秒)」が「GC処理の実行開始時間」として自動的に付加されます（以下の形式で出力されます）。

  ```
  GC処理の実行開始時間: GC処理の結果ログ(「GCログの出力フォーマット」と同じ)
  ```

  「GC処理の実行開始時間」のフォーマットは変更できません。

  （ガーベジコレクション処理の結果ログ出力機能の強化も使用して出力されたGC処理の結果ログの場合は、「GC処理の実行開始時間」のフォーマットを、ログ出力における「9.5.7 ログ出力における時間情報のフォーマット指定機能」により指定できます。）

- 格納先ファイル名の指定には、絶対パスや相対パスのディレクトリ名を付加した形式も可能です。
- 格納先ファイル名の中にあるディレクトリが存在しないなど、何らかの理由で指定された格納先ファイルにアクセスできなかった場合、GC処理の結果ログは出力されません。

注意

Interstage Application Server配下でJavaアプリケーションを実行する際にオプションを指定した場合、以下の問題が発生します。

- オプションによるファイル出力処理には、ログローテーションなどの世代管理機能はありません。何らかの理由でJavaプロセスが自動的に再起動した場合、格納先として同じファイルが用いられることになるため、再起動後のGC処理の結果ログで再起動前の結果ログが上書きされ、ログ情報として使えなくなります。
- Javaアプリケーションを複数のプロセスで多重動作させる場合、同一のオプション定義で複数のプロセスが動作します。そのため、複数のプロセスから同じファイルに対してGC処理の結果ログを書き込むことになり、ログ情報として使えなくなります。
- Interstage Application Server配下でJavaアプリケーションを実行する際のログの出力先は、Interstage Application Serverとしての制御により、出力先ファイルが管理されています。オプション指定でGC処理の結果ログが別のファイルとなった場合、エラー発生時に、他のエラー情報とGC処理の結果ログが分離することになるため、エラーの解析が難しくなる場合があります。

そのため、Interstage Application Server配下でJavaアプリケーションを実行する場合には、上記のオプションを指定しないでください。上記のオプションは、Interstage Application Serverとは関係しない単独のJavaアプリケーションを実行する場合に、必要に応じて指定してください。

9.2.6.1 ガーベジコレクション処理の結果ログ出力機能の強化

FJVMでは、“-verbose:gc”オプション指定時に出力されるガーベジコレクション(GC)処理の結果ログを、より詳細に出力する「***ガーベジコレクション処理の結果ログ出力機能の強化***」を行っています。

GC処理の結果ログとして出力される情報を拡張するオプション

```
-XX:+UseFJverbose
```

“-verbose:gc”オプションを指定してGC処理の結果ログを出力する際、上記のオプションを追加指定することにより、GC処理の結果ログとして出力される情報が、「GC処理の結果ログとして出力される情報の拡張形式」に示す形式に拡張されます。

GC処理の結果ログとして出力される情報の拡張形式

```
$1: [$2, [$3 : $4->$5($6)], [$7 : $8->$9($10)] $11->$12($13), [$14 : $15->$16($17)], $18 secs]
```

$1: GC処理の実行開始時間(ログ出力時の時間)

GC処理の実行開始時間(ログ出力時の時間)を示します。
ログ出力時の時間のフォーマットは、ログ出力における「9.5.7 ログ出力における時間情報のフォーマット指定機能」により指定できます。
デフォルトは、「Java VMが起動されてからの経過時間(秒)」です。

$2: GC種別

実行したGC処理の種別を以下の名称で示します。

- GC

  New世代領域を対象とするGC処理(NewGC処理またはマイナーGC処理)における結果情報です。

- Full GC

  Javaヒープ域全体(メモリ割り当てプール(New世代領域、Old世代領域)およびPermanent世代領域)を対象とするGC処理(FullGC処理)における結果情報です。

- Full GC*

  使用されているGC処理がシリアルGCまたはCMS付きパラレルGCの場合で、かつ、直前に実行されたNewGC処理では十分な空き領域が確保できず、そのNewGC処理に続けて実行されたFullGC処理における結果情報です("Full GC"の後に"*"の付く表示になります)。
  なおこの名称は“-verbose:gc”オプションだけを指定した場合には出力されません(単に"Full GC"として出力されます)。

- CMS initial-mark

  Old世代領域を対象とするCMS-GC処理のうち、initialマーク処理における結果情報です。
  CMS-GCでは、不要オブジェクトを検出するために行う処理(initialマーク処理)を実行する際、極わずかな時間だけJavaアプリケーションを停止させます。
  注)不要オブジェクトの検出処理だけであるため、GC処理開始前後での各世代領域におけるオブジェクト量に変化はありません。

- CMS remark

  Old世代領域を対象とするCMS-GC処理のうち、finalマーク処理における結果情報です。
  CMS-GCでは、不要オブジェクトを検出するために行う処理(finalマーク処理)を実行する際、極わずかな時間だけJavaアプリケーションを停止させます。
  注)不要オブジェクトの検出処理だけであるため、GC処理開始前後での各世代領域におけるオブジェクト量に変化はありません。

$3: New世代領域の識別名

New世代領域の識別名を示します。
使用されているGC処理の違いにより、以下の識別名が出力されます。

- DefNew: シリアルGCの場合
- PSYoungGen: パラレルGCの場合
- ParNew: CMS付きパラレルGCの場合

$4: GC処理前のオブジェクト量(New世代領域)

GC処理実行前のNew世代領域に存在したオブジェクトの総量(バイト)です。

$5: GC処理後のオブジェクト量(New世代領域)

GC処理実行後のNew世代領域に存在するオブジェクトの総量(バイト)です。

$6: New世代領域の大きさ

New世代領域の大きさ(バイト)です。
注)使用されているGC処理がシリアルGC、パラレルGCまたはCMS付きパラレルGCの場合は、この大きさに「to space」領域の大きさが含まれません。
(シリアルGC、パラレルGCまたはCMS付きパラレルGCの場合、GC処理はNew世代領域を「eden space」、「from space」および「to space」の3つの内部領域に細分割して制御しています。)

$7: Old世代領域の識別名

Old世代領域の識別名を示します。
使用されているGC処理の違いにより、以下の識別名が出力されます。

- Tenured: シリアルGCの場合
- PSOldGen: パラレルGCの場合
- CMS: CMS付きパラレルGCの場合

$8: GC処理前のオブジェクト量(Old世代領域)

GC処理実行前のOld世代領域に存在したオブジェクトの総量(バイト)です。

$9: GC処理後のオブジェクト量(Old世代領域)

GC処理実行後のOld世代領域に存在するオブジェクトの総量(バイト)です。

$10: Old世代領域の大きさ

Old世代領域の大きさ(バイト)です。

$11: GC処理前のオブジェクト量(メモリ割り当てプール)

GC処理実行前のメモリ割り当てプールに存在したオブジェクトの総量(バイト)です。
$4+$8の値です。

$12: GC処理後のオブジェクト量(メモリ割り当てプール)

GC処理実行後のメモリ割り当てプールに存在するオブジェクトの総量(バイト)です。
$5+$9の値です。

$13: メモリ割り当てプールの大きさ

メモリ割り当てプールの大きさ(バイト)です。
$6+$10の値です。
注)使用されているGC処理がシリアルGC、パラレルGCまたはCMS付きパラレルGCの場合は、この大きさにNew世代領域の「to space」領域の大きさが含まれません。
(シリアルGC、パラレルGCまたはCMS付きパラレルGCの場合、GC処理はNew世代領域を「eden space」、「from space」および「to space」の3つの内部領域に細分割して制御しています。)

$14: Permanent世代領域の識別名

Permanent世代領域の識別名です。
使用されているGC処理の違いにより、以下の識別名が出力されます。

- Perm: シリアルGCの場合
- PSPermGen: パラレルGCの場合
- CMS Perm: CMS付きパラレルGCの場合

$15: GC処理前のオブジェクト量(Permanent世代領域)

GC処理実行前のPermanent世代領域に存在したオブジェクトの総量(バイト)です。

$16: GC処理後のオブジェクト量(Permanent世代領域)

GC処理実行後のPermanent世代領域に存在するオブジェクトの総量(バイト)です。

$17: Permanent世代領域の大きさ

Permanent世代領域の大きさ(バイト)です。

$18: GC処理実行時間

GC処理実行に要した時間(秒)です。
注)GC処理は、Javaアプリケーションとしての動作を停止させて実行されます。

参考

$2、$11、$12、$13、および$18に対する出力情報は、GC処理の結果ログ出力機能として“-verbose:gc”オプションだけを指定した際に出力される情報と対応します。

GC処理の結果ログとして出力される情報の拡張形式(CMS-GCの開始)

```
$1: CMS start
```

$1: CMS-GC処理の実行開始時間(ログ出力時の時間)

CMS-GC処理の実行開始時間(ログ出力時の時間)を示します。
ログ出力時の時間のフォーマットは、ログ出力における「9.5.7 ログ出力における時間情報のフォーマット指定機能」により指定できます。
デフォルトは、「Java VMが起動されてからの経過時間(秒)」です。

GC処理の結果ログとして出力される情報の拡張形式(FullGC発生によるCMS-GCの終了要求)

```
$1: CMS stop-req
```

$1: CMS-GC処理の終了要求時点(ログ出力時の時間)

Full GC要求発生時にCMS-GC処理が実行中であった場合、CMS-GC処理の終了要求時点(ログ出力時の時間)を示します。
ログ出力時の時間のフォーマットは、ログ出力における「9.5.7 ログ出力における時間情報のフォーマット指定機能」により指定できます。
デフォルトは、「Java VMが起動されてからの経過時間(秒)」です。

注)CMS-GC処理が動作している最中にJavaヒープ不足やjava.lang.System.gc()実行などによりFull GC要求が発生した場合、Full GC処理は、実行中のCMS-GC処理の終了を待ってから開始されます。その際、CMS-GCが処理しているデータの整合性を維持させるため、CMS-GC処理の終了は強制的な打ち切り終了ではなく、CMS-GC内で打ち切り可能な処理まで完了させてからの終了になります。そのため、CMS-GC処理が実行中であった場合、Full GC処理要求の発生から実際にFull GC処理が開始されるまでに、ある程度の時間差が生じる可能性があります。

なおCMS-GC処理の終了要求時点からCMS-GC処理の実行が終了するまで、Javaアプリケーションとしての動作は停止します。そのため、CMS-GC処理の終了要求時点からFull GC処理の実行完了までが、この出力があった場合のFull GC処理によるJavaアプリケーション動作の実際の停止期間となります。

GC処理の結果ログとして出力される情報の拡張形式(CMS-GCの終了)

- パターン1:

  CMS-GCの対象が「Old世代領域内」の場合

  ```
  $1: CMS stop($2), [CMS : $3->$4($5)], $9 secs
  ```

  CMS-GCの対象が「Old世代領域内およびPermanent世代領域内」の場合

  ```
  $1: CMS stop($2), [CMS : $3->$4($5)], [CMS Perm : $6->$7($8)], $9 secs
  ```

- パターン2:

```
$1: CMS stop($2), $9 secs
```

$1: CMS-GC処理の実行終了時間(ログ出力時の時間)

CMS-GC処理の実行終了時間(ログ出力時の時間)を示します。
ログ出力時の時間のフォーマットは、ログ出力における「9.5.7 ログ出力における時間情報のフォーマット指定機能」により指定できます。
デフォルトは、「Java VMが起動されてからの経過時間(秒)」です。

$2: 終了コード

CMS-GC処理の実行結果に対する終了コードを示します。
終了コードの違いにより、情報の出力形式パターンが異なります。
終了コードの種類および意味は以下のとおりです。

- 00: CMS-GC処理が終了しました。

  Old世代領域内またはOld世代領域内およびPermanent世代領域内で検出済の不要オブジェクトは回収しました。
  パターン1の出力形式で情報が出力されます。

- 10: Javaヒープ不足によるFull GC要求があったため、実行中のCMS-GC処理を終了しました。

  Old世代領域内またはOld世代領域内およびPermanent世代領域内で検出済の不要オブジェクトは回収しました。
  パターン1の出力形式で情報が出力されます。

- 20: java.lang.System.gc()などの外部要因によるFull GC要求があったため、実行中のCMS-GC処理を終了しました。

  Old世代領域内またはOld世代領域内およびPermanent世代領域内で検出済の不要オブジェクトは回収しました。
  パターン1の出力形式で情報が出力されます。

- 11: Javaヒープ不足によるFull GC要求があったため、実行中のCMS-GC処理を終了しました。

  Old世代領域内またはOld世代領域内およびPermanent世代領域内の不要オブジェクトは回収していません。
  パターン2の出力形式で情報が出力されます。

- 21: java.lang.System.gc()などの外部要因によるFull GC要求があったため、実行中のCMS-GC処理を終了しました。

  Old世代領域内またはOld世代領域内およびPermanent世代領域内の不要オブジェクトを回収していません。
  パターン2の出力形式で情報が出力されます。

$3: CMS-GC処理前のオブジェクト量(Old世代領域)

CMS-GC処理実行前のOld世代領域に存在したオブジェクトの総量(バイト)です。
通常はCMS-GC処理のfinalマーク処理実行時のOld世代領域に存在したオブジェクトの総量と等しくなります。
finalマーク処理実行時のOld世代領域に存在したオブジェクトの総量と等しくない場合は、CMS-GCによる不要オブジェクトの回収処理前にNew世代領域用GCが実行された、またはJavaアプリケーションの実行によりOld世代領域にオブジェクトが割り当てられたことを示します。

$4: CMS-GC処理後のオブジェクト量(Old世代領域)

CMS-GC処理実行後のOld世代領域に存在するオブジェクトの総量(バイト)です。

$5: Old世代領域の大きさ

Old世代領域の大きさ(バイト)です。

$6: CMS-GC処理前のオブジェクト量(Permanent世代領域)

CMS-GC処理実行前のPermanent世代領域に存在したオブジェクトの総量(バイト)です。
通常はfinalマーク実行時のPermanent世代領域に存在したオブジェクトの総量と等しくなります。
finalマーク処理実行時のPermanent世代領域に存在したオブジェクトの総量と等しくない場合は、CMS-GCによる不要オブジェクトの回収処理前に、Javaアプリケーションの実行によりPermanent世代領域にオブジェクトが割り当てられたことを示します。

$7: CMS-GC処理後のオブジェクト量(Permanent世代領域)

CMS-GC処理実行後のPermanent世代領域に存在するオブジェクトの総量(バイト)です。

$8: Permanent世代領域の大きさ

Permanent世代領域の大きさ(バイト)です。

$9: CMS-GC処理実行時間

CMS-GC処理実行に要した時間(秒)です(CMS-GC開始からの経過時間です)。

注意

有効なGC処理

このオプション指定により出力形式が拡張されるのは、使用するGC処理が以下の場合です。

- シリアルGC
- パラレルGC
- CMS付きパラレルGC

注意

ログ出力量の増加

本オプションの指定により、ログ出力が増大します。

本オプションを指定する場合は、ログ出力量についての注意が必要です。

ログの見方

GC処理の結果ログとして出力される情報の拡張形式の出力例を示します。

例

GC処理の結果ログとして出力される情報の拡張形式の出力例

```
23.646: [Full GC, [PSYoungGen : 1584K->0K(5504K)], [PSOldGen : 57764K->26302K(58304K)] 59348K->26302K(63808K), [PSPermGen : 4655K->4655K(16384K)], 0.1353549 secs]
```

この出力情報から、以下のことが分かります。

- Java VMが起動されてから23.646秒後にFullGC処理が実行された。
- 使用されているGC処理はパラレルGCである。
- GC処理後のNew世代領域の大きさは5504KBである。
- GC処理により、New世代領域に存在するオブジェクト量は1584KBから0KBになった。
  (不要オブジェクトが削除され、また必要に応じてOld世代領域へ生存オブジェクトが移動した)
- GC処理後のOld世代領域の大きさは58304KBである。
- GC処理により、Old世代領域に存在するオブジェクト量は57764KBから26302KBになった。
  (不要オブジェクトが削除され、また必要に応じてNew世代領域から生存オブジェクトが移動してきた)
- GC処理後のメモリ割り当てプールの大きさは63808KBである。
- GC処理により、メモリ割り当てプールに存在するオブジェクト総量は59348KBから26302KBになった。
  (不要オブジェクトが削除された)
- GC処理後のPermanent世代領域の大きさは16384KBである。
- GC処理により、Permanent世代領域に存在するオブジェクト量は変化していない。

- GC処理に要した時間は0.1353549秒である。

例

GC処理の結果ログとして出力される情報の拡張形式の出力例(CMS付きパラレルGCの場合-1)

```
150.207: CMS start
150.208: [CMS initial-mark, [ParNew : 1863K->1863K(14784K)], [CMS : 53791K->53791K(65536K)] 55654K->55654K(80320K), [CMS Perm : 4664K->4664K(16384K)], 0.0030212 secs]
150.351: [GC, [ParNew : 14782K->1598K(14784K)], [CMS : 53791K->57981K(65536K)] 68573K->59579K(80320K), [CMS Perm : 4664K->4664K(16384K)], 0.0328537 secs]
150.466: [CMS remark, [ParNew : 8277K->8277K(14784K)], [CMS : 57981K->57981K(65536K)] 66258K->66258K(80320K), [CMS Perm : 4664K->4664K(16384K)], 0.0097905 secs]
150.549: [GC, [ParNew : 14782K->1598K(14784K)], [CMS : 50163K->54371K(65536K)] 64946K->55969K(80320K), [CMS Perm : 4664K->4664K(16384K)], 0.0303271 secs]
150.583: CMS stop(00), [CMS : 57981K->54200K(65536K)], 0.3753996 secs
```

この出力情報から、以下のことが分かります。

- 使用されているGC処理はCMS付きパラレルGC(CMS-GCの対象はOld世代領域)である。
- Java VMが起動されてから150.207秒後にCMS-GC処理が開始され、150.583秒後に終了した。
- 終了したCMS-GCにより不要オブジェクトが回収された。
- CMS-GC処理後のOld世代領域の大きさは65536KBである。
- CMS-GC処理により、Old世代領域に存在するオブジェクト量は57981KBから54200KBになった。
- CMS-GC処理に要した時間は0.3753996秒である。
- CMS-GC処理中にNew世代領域用GC処理が実行された。またその実行開始は、遅延している可能性がある。

例

GC処理の結果ログとして出力される情報の拡張形式の出力例(CMS付きパラレルGCの場合-2)

```
137.803: CMS start
137.804: [CMS initial-mark, [ParNew : 206690K->206690K(314560K)], [CMS : 655731K->655731K(699072K)] 862421K->862421K(1013632K), [CMS Perm : 3892K->3892K(16384K)], 0.4101250 secs]
139.069: [GC, [ParNew : 279616K->34943K(314560K)], [CMS : 655731K->673280K(699072K)] 935347K->708223K(1013632K), [CMS Perm : 3892K->3892K(16384K)], 0.2177910 secs]
142.140: CMS stop-req
142.501: CMS stop(11), 4.6984060 secs
142.501: [Full GC, [ParNew : 314559K->0K(314560K)], [CMS : 673280K->657037K(699072K)] 987839K->657037K(1013632K), [CMS Perm : 3892K->3892K(16384K)], 1.8642510 secs]
```

この出力情報から、以下のことが分かります。

- 使用されているGC処理はCMS付きパラレルGC(CMS-GCの対象はOld世代領域)である。
- Java VMが起動されてから137.803秒後にCMS-GC処理が開始され、142.501秒後に終了した。
- Java VMが起動されてから142.140秒後にFullGC要求があり、そのため実行中のCMS-GCが終了した。
- 終了したCMS-GCによる不要オブジェクト回収は実行されていない。
- CMS-GC処理の終了要求時点(142.140)からCMS-GC処理の実行が終了(142.501)するまでの0.361秒、および142.501に実行開始したFullGCの実行時間1.8642510秒の合計2.225251秒の間、Javaアプリケーション動作が停止された。

9.3 動的コンパイル

動的コンパイルについて説明します。

C/C++やCOBOLなどで作られたプログラムを実行する場合は、各言語に対応したコンパイラによって、プログラムのソースコードを実行対象プラットフォーム上で動作可能な機械命令へ翻訳（後述の動的コンパイルとの対比で静的コンパイルと呼ばれることがあります）し、事前にプラットフォーム依存の実行バイナリを作成する必要があります。

Javaで作られたプログラムを実行する場合は、javacコマンドによって、プログラムのソースコードをJava VMが解釈／実行できる命令「バイトコード」へ変換し、事前にプラットフォーム非依存の実行バイナリである「クラスファイル」を作成する必要があります。

Javaの実行環境であるJava VMが起動された後、Java VMは、実行対象プログラムであるクラスファイルを読み込み、クラスファイルを以下の2つの方法を用いて実行します。

- インタプリタによるバイトコードの実行

  Java VMのインタプリタは、クラスファイル内のバイトコードを1命令ずつ解釈して実行します。
  機械命令の実行に比べ、実行性能が低速になります。

- 動的コンパイルにより、バイトコードを機械命令に翻訳してから実行

  Javaアプリケーションの実行中、Java VMは、クラスファイル内のJavaメソッドに対応するバイトコードを、実行対象プラットフォーム上で動作可能な機械命令へ自動的に翻訳してから実行します。Javaアプリケーション実行中に自動的に行われる翻訳処理であるため、その翻訳処理を**動的コンパイル**と呼びます。
  インタプリタによるバイトコードの実行に比べ、高速に実行できます。

  なお、動的コンパイル処理は、翻訳時における機械命令の最適化処理で必要となる各Javaメソッドの実行頻度や呼び出し関係などの情報を、Javaアプリケーション実行と同時に行われる各Javaメソッドに関するプロファイリング処理の結果から得ます。そのため、プロファイリング処理によりある程度の情報量が得られるまで何度か再翻訳が繰り返され、次第にJavaアプリケーションの実行状況に合った翻訳結果に最適化されることで、機械命令部分の実行性能が向上する傾向があります。

Java VMが行う動的コンパイル自体は、Javaアプリケーションの実行から見ると、オーバーヘッド部分になります。そのため、インタプリタによる実行性能(低速)、動的コンパイルによるオーバーヘッド、および動的コンパイル結果である機械命令による実行性能(高速)の三者をバランス良く調整することで、Javaアプリケーション全体としての実行性能を良くする必要があります。

Java VMは、実行頻度の高いJavaメソッドを優先的にコンパイルし、あまり使われることのないJavaメソッドはインタプリタ実行のままにすることで、インタプリタ実行、動的コンパイル、および動的コンパイル結果である機械命令実行のバランスを取り、Javaアプリケーション全体としての実行性能を良くする調整を行っています。

なお、実行対象となるJavaアプリケーション内で、各Javaメソッドの実行頻度がどの位になるのかについては、実際にJavaアプリケーションが実行されない限り分かりません。そのため、Java VMはJavaアプリケーション実行と同時に各Javaメソッドに関するプロファイリング処理を行い、その結果を用いて動的コンパイルの対象となるJavaメソッドを決定しています。プロファイリング処理によりある程度の情報量が得られるまで、すなわちJava VM起動直後はインタプリタ実行だけですが、次第にインタプリタ実行と動的コンパイル結果による機械命令実行との混合動作になります。

FJVMでは、動的コンパイルに関する以下の機能を富士通版独自機能として実装しています。

以下は、FJVM固有機能です。

- コンパイラ異常発生時の自動リカバリ機能
- 長時間コンパイル処理の検出機能
- 動的コンパイル発生状況のログ出力機能

9.3.1 コンパイラ異常発生時の自動リカバリ機能

コンパイラ異常発生時の自動リカバリ機能はFJVM固有機能です。

Java VMはJavaアプリケーションとして実行されるJavaメソッドに対して必要に応じて自動的にコンパイル処理を行いますが、コンパイル処理を行っている際にコンパイラ内で何らかの異常が発生すると、当該Javaメソッドに対するコンパイル処理だけでなくJava VMとしての動作も異常状態として停止させてしまう場合があります。

FJVMでは、コンパイラ内で何らかの異常が発生した場合に自動的にリカバリ処理を行い、Java VMとしての動作を継続させる機能を「コンパイラ異常発生時の自動リカバリ機能」として実装しています。

本機能によるリカバリ処理が行われた際にコンパイル対象となっていたJavaメソッドは、以降、コンパイル処理の対象とはなりません。当該Javaメソッドについてはコンパイルされず、インタプリタモードのままJavaアプリケーションとしての実行が継続されます。

リカバリ処理実施後に通知を受け取るオプション

本機能は**FJVM**の内部処理として動作する機能であるため、コンパイラ内で何らかの異常が発生してもリカバリ処理が正常に行われた場合には、外部に対する通知などは何も行いません。リカバリ処理が正常に行われた場合でもコンパイラ内で何らかの異常が発生したことを情報として受け取る場合には、以下のオプションを指定してください。

```
-XX:+PrintCompilerRecoveryMessage
```

リカバリ処理実施後に通知を受け取るオプションを指定した場合の出力形式

```
CompilerRecovery: Information:The compilation was canceled for method method_name
Reason for the cancellation: reason [code:c, addr:xxxxxxxx]
```

リカバリ処理の情報が上記の形式で標準出力に出力されます。

method_name:

コンパイル処理で異常が発生した際にコンパイル対象となっていたJavaメソッドの名前。

reason:

コンパイル処理で発生した異常の原因情報。原因情報として以下の項目があります。

- assert: コンパイル処理で内部処理矛盾を検出した
- error: コンパイル処理で何らかの誤りを検出した
- stack overflow: コンパイル処理でスタックオーバーフローを検出した

c:

異常コード。

xxxxxxxx:

コンパイル処理で異常が発生した際のアドレス。

9.3.2 長時間コンパイル処理の検出機能

コンパイラ異常発生時の自動リカバリ機能はFJVM固有機能です。

Java VMはJavaアプリケーションとして実行されるJavaメソッドに対して必要に応じて自動的にコンパイル処理を行い、通常は極短時間でその処理を完了します。
しかし、コンパイル処理自身や同じJavaプロセス内で動作させている他の処理の障害などによりCPU資源が占有され続けてしまうと、数分を経過してもコンパイル処理が終了しない場合が考えられます。このような状態が継続すると、システム全体に対して悪影響を与える可能性が考えられます。
このため、**FJVM**では、各Javaメソッドのコンパイル処理に要している時間を監視し、コンパイル処理で必要と考えられる程度の時間を経過してもコンパイル処理が終了していない場合には、Javaプロセス内の処理で何らかの問題が発生していると判断し、当該Javaプロセスを強制的に終了させる機能を「**長時間コンパイル処理の検出機能**」として実装しています。

長時間コンパイル処理の検出機能を有効にするオプション

本機能は、以下のオプションでコンパイル処理に対する監視時間(コンパイル処理に要する時間の上限値)を指定した場合に有効となります。ただし、オプション値として「0」を指定した場合には、本機能は有効となりません。

以下のオプションで指定された時間を超過してもコンパイル処理が終了していない場合、本機能はJavaプロセス内の処理で何からの問題が発生していると判断し、当該Javaプロセスを強制的に終了させます。

```
-XX:CompileTimeout=<nn>
```

<nn>

本機能による異常有無の判断条件とされるコンパイル処理に要する時間の上限値(単位:秒)を指定します。デフォルト値は「0」であり、本機能は無効となっています。
なお、本機能による時間監視の最小単位は30秒であるため、その単位での時間誤差があります。

長時間コンパイル処理の検出機能によるJavaプロセス強制終了時の出力メッセージ

本機能によってJavaプロセスを強制終了する場合、**FJVM**は以下のメッセージを標準出力に出力してから終了します。また、本機能によるJavaプロセスの強制終了時にはコアダンプも出力されます。

```
CompilerRecovery: Information: CompilerRecovery got the VM aborted
because the compiler thread(nnnnnnnn) has not completed.
(compiling method: method_name)
```

nnnnnnnn:

コンパイラスレッドの内部識別子

method_name:

本機能によるチェックでJavaプロセス内に異常が検出された際にコンパイル対象となっていたJavaメソッドの名前

注意

長時間コンパイル処理の検出機能に対する注意事項

何らかの要因によりJavaプロセス内のコンパイル処理へCPU資源が十分に割り当てられず、コンパイル処理自体が進んでいない場合でも、コンパイル処理開始から-XX:CompileTimeoutオプションで指定された監視時間を超過した場合には、本機能による当該Javaプロセスの強制終了となります。

このため、当該Javaプロセスを実行するシステムのCPU負荷が高い場合には、コンパイル処理に対してCPU資源が十分に割り当てられず、この結果として本機能による強制終了が発生する可能性が考えられます。

本機能による強制終了が発生した場合には、まず以下の事項を確認してください。

- 当該Javaプロセスを実行しているシステムのCPU資源量は十分か。
- 当該Javaプロセス以外の他のプロセスでCPU資源が占有されていないか。
- “-XX:CompileTimeout=0”を指定した場合に本機能による強制終了が回避され、かつ、当該Javaプロセスが正常に終了する、または未負荷時に正常なアイドル状態に遷移するか。

上記事項に合致する場合は、コンパイル処理に対してCPU資源が十分に割り当てられなかった結果として発生した強制終了と考えられます。
長時間コンパイル処理の検出機能を有効にしてこの状態が発生した場合には、“-XX:CompileTimeout”オプションで指定する監視時間として、より大きな値に設定する形で調整してください。

参考

長時間コンパイル処理の検出機能に対する監視メッセージの出力

以下のオプションを指定した場合、Javaメソッドのコンパイル処理において1分が経過すると、「長時間コンパイル処理の検出機能が出力する監視メッセージ」の形式で監視メッセージが出力されます。
その後は、30秒経過するごとに同じ監視メッセージが出力されます。
なお、本機能による時間監視の最小単位は30秒であるため、その単位での時間誤差があります。

- 長時間コンパイル処理の検出機能の監視メッセージ出力を有効にするオプション

```
-XX:+PrintCompilerRecoveryMessage
```

- 長時間コンパイル処理の検出機能が出力する監視メッセージ

```
CompilerRecovery: Information: The compiler thread(0xnnnnnnnn) might not return from compiling method
method_name.
```

nnnnnnnn:

コンパイラスレッドの内部識別子

method_name:

本機能によるチェックで検出された際にコンパイル対象となっていたJavaメソッドの名前

9.3.3 動的コンパイル発生状況のログ出力機能

動的コンパイル発生状況のログ出力機能はFJVM固有機能です。

Java VMは、Javaアプリケーションとして実行されるJavaメソッドに対して、必要に応じて自動的にコンパイル処理を行います(動的コンパイル)。

FJVMでは、動的コンパイルの発生状況を出力する機能を「動的コンパイル発生状況のログ出力機能」として実装しています。

動的コンパイル発生状況のログ出力機能では、以下の情報を出力します。

- コンパイラスレッドのCPU使用状況

  コンパイラスレッド(動的コンパイルを行っているスレッド)が20個のJavaメソッドをコンパイルするごとに、コンパイルを行うのに使用したCPU時間と経過時間を出力します。

  経過時間に対してCPU時間の割合が高い場合には、Javaアプリケーションの実行性能に動的コンパイルが影響を与えている可能性があります。

- 動的コンパイル結果情報

  どのJavaメソッドが、いつコンパイルされたかの情報を出力します。
  Javaメソッドのコンパイルが、短い間に連続して発生している場合、Javaアプリケーションの実行性能に動的コンパイルが影響を与えている可能性があります。

コンパイラスレッドのCPU使用状況を出力するオプション

コンパイラスレッドのCPU使用状況を出力する場合に指定します。

```
-XX:+PrintCompilationCPUTime
```

コンパイラスレッドのCPU使用状況および動的コンパイル結果情報を出力するオプション

コンパイラスレッドのCPU使用状況および動的コンパイル結果情報を出力する場合に指定します。

```
-XX:+FJPrintCompilation
```

上記の両方のオプションの指定により、動的コンパイルが発生するたびに、その発生状況ログが標準出力へ、以下に示す形式で出力されます。

コンパイラスレッドのCPU使用状況の出力形式

```
$1: [$2: cpu=$3ms elapsed=$4ms $5]
```

$1: ログ出力時の時間

ログ出力時の時間を示します。
ログ出力時の時間のフォーマットは、ログ出力における「9.5.7 ログ出力における時間情報のフォーマット指定機能」により指定できます。
デフォルトは、「Java VMが起動されてからの経過時間(秒)」です。

$2: コンパイラスレッド名

情報の出力対象となるコンパイラスレッドの名前を「CompilerThread数字」の形式で示します。

$3: CPU時間

$2のコンパイラスレッドが、20個のJavaメソッドのコンパイルを行うのに使用したCPU時間(ミリ秒)を示します。
なお、ネイティブメソッド自体はコンパイル処理の対象となりません。ネイティブメソッドに対しては、ネイティブプログラムを呼び出すための専用コードの作成処理が、コンパイラスレッドで実行されますが、作成処理は軽微なため、CPU時間を求める際の20個のJavaメソッドに含まれません。

$4: 経過時間

$2のコンパイラスレッドが、20個のJavaメソッドのコンパイルを行うまでに経過した時間(ミリ秒)を示します。

$5: 通し番号

コンパイラスレッドごとの本情報の出力数を、通し番号で示します。
通し番号×20」の値が、そのコンパイラスレッドでコンパイルしたJavaメソッドの累計数になります。

動的コンパイル結果情報の出力形式

```
$1: $2 $3 ($4 bytes) $5
```

$1: Javaメソッドのコンパイル要求発生時間(ログ出力時の時間)

Javaメソッドのコンパイル要求が発生した時間(ログ出力時の時間)を示します。
ログ出力時の時間のフォーマットは、ログ出力における「9.5.7 ログ出力における時間情報のフォーマット指定機能」により指定できます。
デフォルトは、「Java VMが起動されてからの経過時間(秒)」です。

$2: 通し番号

コンパイル要求の数(コンパイル要求が発生したJavaメソッドの数)を通し番号で示します。
通し番号の後ろに「%」がない場合は、コンパイル対象のJavaメソッド全体をコンパイルする要求です。
通し番号の後ろに「%」がある場合は、コンパイル対象のJavaメソッドを部分的にコンパイルする要求です。
通し番号の後ろに「%」がない場合とある場合は、別の通し番号になります。

$3: Javaメソッド名

コンパイル要求が発生したJavaメソッドの名前を示します。
Javaメソッドを部分的にコンパイルする要求の場合($2に「%」の表示が含まれる場合)、Javaメソッド名の後に、Javaメソッド(バイトコード)のどの部分からコンパイルの対象になっているかを示す情報「(@ 数字)」が付加されます。

$4: Javaメソッドのバイト数

コンパイル対象となったJavaメソッドの大きさ(バイトコード・サイズ)をバイト数で示します。
ネイティブメソッドの場合は、0 bytes と表示されます。

$5: 空白または(static)

コンパイル対象となったJavaメソッドが、static修飾子を持つネイティブメソッドだった場合は、“(static)”と出力されます。コンパイル対象となったJavaメソッドが、ネイティブメソッドでない場合、またはstatic修飾子を持たないネイティブメソッドだった場合、この位置には何も出力されません。

例

-XX:+PrintCompilationCPUTime指定時の出力例

```
0.586: [CompilerThread1: cpu=78.13ms elapsed=450.72ms 1]
0.822: [CompilerThread0: cpu=437.50ms elapsed=686.32ms 1]
1.312: [CompilerThread0: cpu=218.75ms elapsed=489.93ms 2]
1.637: [CompilerThread1: cpu=546.88ms elapsed=1050.52ms 2]
2.385: [CompilerThread0: cpu=296.88ms elapsed=1073.57ms 3]
3.365: [CompilerThread0: cpu=140.63ms elapsed=979.67ms 4]
3.557: [CompilerThread1: cpu=343.75ms elapsed=1919.97ms 3]
4.096: [CompilerThread1: cpu=390.63ms elapsed=539.47ms 4]
4.995: [CompilerThread1: cpu=140.63ms elapsed=898.45ms 5]
```

例

-XX:+FJPrintCompilation指定時の出力例

```
0.074: 1 java.util.Properties$LineReader::readLine (383 bytes)
0.102: 2 java.io.Win32FileSystem::normalize (143 bytes)
0.107: 3 java.lang.String::hashCode (60 bytes)
0.179: 4 sun.security.provider.SHA::implCompress (494 bytes)
0.206: 5 sun.reflect.UTF8::utf8Length (81 bytes)
0.229: 6 java.util.jar.Manifest$FastInputStream::readLine (167 bytes)
0.232: 7 sun.nio.cs.UTF_8$Decoder::decodeArrayLoop (1814 bytes)
0.244: 1% sun.text.NormalizerDataReader::read @ 38 (139 bytes)
0.261: 8 java.math.BigInteger::mulAdd (82 bytes)
(中略)
0.742: [CompilerThread1: cpu=406.25ms elapsed=677.52ms 1]
0.744: 40 java.lang.String::replace (142 bytes)
```

9.4 チューニング方法

チューニングのポイントとして、次があります。

- メモリ消費量と処理速度は深い関係にあり、メモリ消費量を抑制すれば処理速度が低下するのが一般的です。しかし、JDK/JREにおいては、Javaヒープのサイズを必要以上に大きく確保した場合、New GCの発生頻度が少なくなる反面、FullGCの処理時間が増大し、処理速度が低下するという特徴があります。
- プロセスに割り当てられたメモリ資源は限られています。JDK/JREにおいては、スタック、Javaヒープおよびネイティブモジュールの動作に必要な領域などの各セグメントがユーザ空間に割り当てられます。このため、あるセグメントの領域を大きく確保すれば、その分だけ他のセグメントの領域が少なくなります。

上記のポイントを踏まえた上で、JDK/JRE 7のチューニングを行います。

9.4.1 Javaヒープのチューニング

Javaヒープのチューニング方法および、チューニングによる影響範囲を説明します。

チューニング方法

Javaヒープの各領域のサイズは、“表9.3 Javaヒープに関するオプション”に示すオプションをJava起動時に指定することで設定ができます。

メモリ割り当てプールのデフォルトの初期値および最大値を、“表9.4 メモリ割り当てプールのデフォルトのサイズ”に示します。
また、**Permanent世代領域**のデフォルトの初期値および最大値を、“表9.5 Permanent世代領域のデフォルトのサイズ”に示します。
また各オプションにおいて、“表9.3 Javaヒープに関するオプション”中に記載されていないデフォルト値を、“表9.6 Javaヒープに関するオプション(-XX:NewSize/-XX:NewRatio)のデフォルト値”に示します。

表9.3 Javaヒープに関するオプション

オプション	オプションの機能 (注1)
-Xms	**メモリ割り当てプール**の初期値を指定します。 たとえば、**メモリ割り当てプール**の初期値を128MBに設定する場合、“-Xms128m”と指定します。 本オプションのデフォルト値は“表9.4 メモリ割り当てプールのデフォルトのサイズ”のとおりです。 なお、指定値が1MB未満の場合、または-XX:NewSizeオプションの値(デフォルト値を含む)以下の場合、初期化エラーとなり、Javaプロセスは終了します。

オプション	オプションの機能 (注1)
-Xmx	**メモリ割り当てプール**の最大値を指定します。 たとえば、**メモリ割り当てプール**の最大値を256MBに設定する場合、“-Xmx256m”と指定します。 (実際に使用される値は、ページサイズなどのシステム情報を元に、Java VMによる制御が最適となるように調整された値「調整値」となるため、指定された値と若干異なる場合があります。) 本オプションのデフォルト値は“表9.4 メモリ割り当てプールのデフォルトのサイズ”のとおりです。 なお、指定値(または調整値)が、-Xmsオプションで指定された値よりも小さな値の場合、初期化エラーとなり、Javaプロセスは終了します。
-XX:NewSize	**New世代領域**のヒープサイズを指定します。 たとえば、**New世代領域**のヒープサイズを128MBに設定する場合、“-XX:NewSize=128m”と指定します。 本オプションの指定が有効な場合のデフォルト値は以下のとおりです。 ・ type0またはtype0pのCMS付きパラレルGCを使用している場合は、メモリ割り当てプールの初期値の1/8です。 ・ type1またはtype1pのCMS付きパラレルGCを使用している場合は、メモリ割り当てプールの初期値の1/3です。 ・ type2またはtype2pのCMS付きパラレルGCを使用している場合は、メモリ割り当てプールの初期値の1/16です。 ・ 上記以外の場合のデフォルト値は“表9.6 Javaヒープに関するオプション(-XX:NewSize/-XX:NewRatio)のデフォルト値”のとおりです。 なお、指定値がJava VMとしての下限値未満の場合、初期化エラーとなり、Javaプロセスは終了します。そのため、本オプションを指定する場合は、1MB以上の値を指定してください。
-XX:MaxNewSize	**New世代領域**の最大ヒープサイズを設定します。 たとえば、**New世代領域**の最大ヒープサイズを128MBに設定する場合、“-XX:MaxNewSize=128m”と指定します。 本オプションの指定が有効な場合のデフォルト値は以下のとおりです。 ・ type0またはtype0pのCMS付きパラレルGCを使用している場合は、メモリ割り当てプールの最大値の1/8です。 ・ type1またはtype1pのCMS付きパラレルGCを使用している場合は、メモリ割り当てプールの最大値の1/3です。 ・ type2またはtype2pのCMS付きパラレルGCを使用している場合は、メモリ割り当てプールの最大値の1/16です。

オプション	オプションの機能 (注1)
	・ 上記以外の場合、デフォルト値はありません(本オプションの値を用いてNew世代領域の最大ヒープサイズは決定されません)。 なお、指定値が、-XX:NewSizeオプションで指定された値よりも小さな値の場合は、-XX:NewSizeオプションで指定された値となります。
-XX:NewRatio (注3)	**New世代領域**と**Old世代領域**のサイズ比率を指定します。 たとえば、**New世代領域**と**Old世代領域**のサイズ比率を2とする場合、“-XX:NewRatio=2”と指定します。 本オプションの指定が有効な場合のデフォルト値は“表9.6 Javaヒープに関するオプション(-XX:NewSize/-XX:NewRatio)のデフォルト値”のとおりです。
-XX:SurvivorRatio (注2)(注4)	**New世代領域**を構成する**Eden領域**と**Survivor領域**のサイズ比率を指定します。 たとえば、**Eden領域**と**Survivor領域**のサイズ比率を8とする場合、“-XX:SurvivorRatio=8”と指定します。 本オプションの指定が有効な場合のデフォルト値は8です。
-XX:TargetSurvivorRatio (注2)(注4)	**ガーベジコレクション(GC)**処理後の生存オブジェクトが**Survivor領域**を占める割合を、指定したパーセンテージ値に調整します。 たとえば、GC処理後の生存オブジェクトが**Survivor領域**を占める割合を半分とする場合、“-XX:TargetSurvivorRatio=50”と指定します。 本オプションの指定が有効な場合のデフォルト値は50です。
-XX:PermSize	**Permanent世代領域**の初期値を指定します。 たとえば、**Permanent世代領域**の初期値を32MBに設定する場合、“-XX:PermSize=32m”と指定します。 本オプションのデフォルト値は“表9.5 Permanent世代領域のデフォルトのサイズ”のとおりです。 なお、指定値が1MB未満の場合は初期化エラーとなり、Javaプロセスは終了します。
-XX:MaxPermSize	**Permanent世代領域**の最大値を指定します。 たとえば、**Permanent世代領域**の最大値を128MBに設定する場合、“-XX:MaxPermSize=128m”と指定します。 (実際に使用される値は、ページサイズなどのシステム情報を元に、Java VMによる制御が最適となるように調整された値「調整値」となるため、指定された値と若干異なる場合があります。) 本オプションのデフォルト値は“表9.5 Permanent世代領域のデフォルトのサイズ”のとおりです。 なお、指定値(または調整値)が、-XX:PermSizeオプションで指定された値よりも小さな値の場合は、-XX:PermSizeオプションで指定された値となります。

注1)

サイズを指定するオプションでは単位として次の文字を指定できます。
KB(キロバイト)を指定する場合: “k”または“K”
MB(メガバイト)を指定する場合: “m”または“M”
GB(ギガバイト)を指定する場合: “g”または“G”

注2)

パラレルGCを使用する場合、このオプションへの指定値は無効となります。

注3)

CMS付きパラレルGCを使用する場合、このオプションへの指定値は無効となります。

注4)

CMS付きパラレルGCを使用する場合、かつ-XX:UseFJcmsGC=type0またはtype0p指定でない場合、このオプションへの指定値は無効となります。

表9.4 メモリ割り当てプールのデフォルトのサイズ

OS	JDK/JREの実行モード	GC処理	初期値	最大値
Windows	32ビットモード	シリアルGC	5.0MB	96MB
		パラレルGC (注1)	7.5MB	
		CMS付きパラレルGC	(注2)	
Windows Server(R) x64 Editions	64ビットモード	シリアルGC	6.4375MB	126MB
		パラレルGC (注1)	8.5MB	
		CMS付きパラレルGC	(注2)	
Linux for x86 Linux for Intel64	32ビットモード	シリアルGC	5.0MB	96MB
		パラレルGC (注1)	7.5MB	
		CMS付きパラレルGC	(注2)	
Linux for Intel64	64ビットモード	シリアルGC	6.4375MB	126MB
		パラレルGC (注1)	8.5MB	
		CMS付きパラレルGC	(注2)	
Solaris	64ビットモード	シリアルGC	6.4375MB	128MB
		パラレルGC (注1)	12.0MB	
		CMS付きパラレルGC	(注2)	

注1)

デフォルトで使用されるGC処理です。

注2)

メモリ割り当てプールの最大値が64MB未満の場合は、メモリ割り当てプールの最大値と等しくなります。メモリ割り当てプールの最大値が64MB以上の場合は、64MBとなります。

表9.5 Permanent世代領域のデフォルトのサイズ

OS	JDK/JREの実行モード	Java VM	初期値	最大値
Windows Linux for x86	32ビットモード	Java HotSpot Client VM	12MB	64MB
Linux for Intel64		FJVM (注1)	16MB	
Windows Server(R) x64 Editions Linux for Intel64	64ビットモード	FJVM (注1)	20.75MB	84MB
Solaris	64ビットモード	FJVM (注1)	20.75MB または 24MB (注2)	84MB

注1)

デフォルトで使用されるJava VMです。

注2)

GC処理にパラレルGCを使用している場合は、24MBになります。シリアルGC または CMS付きパラレルGCを使用している場合は、20.75MB になります。

表9.6 Javaヒープに関するオプション(-XX:NewSize/-XX:NewRatio)のデフォルト値

OS	JDK/JREの実行モード	Java VM	-XX:NewSize	-XX:NewRatio
Windows Linux for x86	32ビットモード	Java HotSpot Client VM	1024KB	2
Linux for Intel64		FJVM (注1)		
Windows Server(R) x64 Editions Linux for Intel64	64ビットモード	FJVM (注1)	1280KB	
Solaris	64ビットモード	FJVM (注1)	1280KB	

注1)

デフォルトで使用されるJava VMです。

チューニングの方針

Javaヒープをチューニングする際、次の方針があります。

1. **FullGC**を実行したにもかかわらず、メモリ不足が発生する場合、**GC**のログを採取し、メモリ割り当てプールまたは**Permanent世代領域**のどちらかの領域が不足しているかどうかを確認します。
2. **FullGC**はコストがかかります。このため、メモリ不足が発生しなくても、Javaアプリケーションがハングアップしたかのように一時的に無反応になる場合、**FullGC**の影響を受けている場合があります。**GC**のログを採取し、**Javaヒープ**が必要以上に大きなサイズになっていれば、**Javaヒープ**のサイズを縮小する方針でチューニングします。
3. 効率的な**New GC**に対して、**FullGC**はコストがかかります。このため、**New世代領域**と**Old世代領域**のサイズのバランスを考慮する必要があります。なお、GC処理としてパラレルGCを使用する場合は、JavaヒープのNew世代領域お

よびOld世代領域の大きさに関する値が自動的に調整および最適化されるため、通常はこのバランスを考慮する必要はありません。

4. 仮想メモリに余裕がある場合は、Javaプロセスを複数起動して、プロセス多重度を上げる方法を検討します。プロセス多重度を上げることにより、プロセスごとのユーザ空間を有効に使うことが可能になります。ただし、メモリのスワッピングによるスローダウンに注意する必要があります。

チューニングの影響範囲

メモリ割り当てプールの大きさ(-Xmxオプションの指定値)や、New世代領域とOld世代領域の大きさのバランスを変更した場合の影響範囲を、次に示します。

- メモリ割り当てプールの大きさを縮小した場合、**GC**が頻発することがあります。
- メモリ割り当てプールの大きさを拡張した場合、**FullGC**に時間がかかることがあります。
- メモリ割り当てプールの大きさを拡張した場合、その分ユーザ空間や仮想メモリが少なくなるため、スタックやネイティブモジュールの動作に必要な領域を確保できず、メモリ不足になることがあります。
- New世代領域の大きさがメモリ割り当てプールの大きさの半分となるようなチューニングを行った場合、一般的に、FullGCが発生しやすくなる傾向があります。なお、Javaアプリケーション実行時のオブジェクト生成／解放などの特性に依存しますので、どのアプリケーションの場合でもFullGCが発生しやすいというわけではありません。

注意

overcommit memory機能が有効な場合の注意事項 Linux32/64

「overcommit memory機能」が有効な場合、Linuxは、Javaヒープの各領域の最大値に相当する仮想メモリ資源を、Java VMの起動時に、Javaプロセスに対して予約します。
このため、-Xms値と-Xmx値を異なる値にしてJavaプロセスを起動する場合、本機能の有効/無効によって、Javaプロセス起動時にJavaヒープとして必要となる仮想メモリの量が異なります。

- overcommit memory機能が無効、またはovercommit memory機能がないシステムの場合

  Javaヒープ用仮想メモリ量 ＝「-Xms値」＋「Perm域初期値」

- overcommit memory機能が有効なシステムの場合

  Javaヒープ用仮想メモリ量 ＝「-Xmx値」＋「-XX:MaxPermSize値」

この結果、仮に同量の仮想メモリ資源を持つシステムの場合であっても、本機能の有効/無効によって、同時に起動できるJavaプロセスの数が異なる場合があります。
Linuxで仮想メモリ資源の見積もりを行う場合には、overcommit memory機能の有無に注意してください。

9.4.2 スタックのチューニング

Javaアプリケーションで使用するスレッドの**スタック**のチューニング方法および、チューニングによる影響範囲を説明します。

チューニング方法

Java APIで生成するスレッドのスタックサイズは、“-Xss”オプションで指定することができます。

“-Xss”オプションは、バイト単位でスタックサイズを指定します。例えば、スタックサイズを512KBに設定する場合、“-Xss512k”と指定します。

またJDK/JRE内で実行されるJavaメソッドを自動的にコンパイルする専用スレッド(コンパイラスレッド)のスタックサイズは、“-XX:CompilerThreadStackSize”オプションで指定することができます。

通常、コンパイラスレッドのスタックサイズを指定する必要はありません。

“-XX:CompilerThreadStackSize”オプションは、キロバイト(Kバイト)単位でコンパイラスレッドのスタックサイズを指定します。例えば、スタックサイズを1024KBに設定する場合、”-XX:CompilerThreadStackSize=1024”と指定します。

Java APIで生成したスレッドおよびコンパイラスレッドのデフォルトのスタックサイズを、“表9.7 Java APIで生成したスレッドおよびコンパイラスレッドのデフォルトのスタックサイズ”に示します。

なお、スタック領域の実際の管理はOSが行います。そのためスタック領域に関する管理方法/動作仕様については、JDK/JREを実行する各OSの仕様に依存します。

表9.7 Java APIで生成したスレッドおよびコンパイラスレッドのデフォルトのスタックサイズ

OS	JDK/JREの実行モード	Java APIで生成したスレッド(注1)	コンパイラスレッド (Client VM)(注1)	コンパイラスレッド (FJVM)
Windows	32ビットモード	320KB	320KB	2048KB
Linux for x86 Linux for Intel64	32ビットモード	320KB	512KB	2048KB
Windows Server(R) x64 Editions Linux for Intel64	64ビットモード	1024KB	(注2)	4096KB
Solaris	64ビットモード	1024KB	(注2)	4096KB

注1)

Windows版JDK/JREにおけるスタックサイズは、java.exeなどWindows版JDK/JREが提供するJDKツールを用いた場合の値です。JNIを用いて独自にJava VMを起動しているWindowsアプリケーションの場合は、Java VMを起動したプログラムのメインスレッドに対するスタックサイズと同じ値になります。

注2)

実行モードが64ビットモードのJDK/JREでは、Java HotSpot Client VMは搭載していません。

チューニングの影響範囲

スタックのサイズを変更した場合の影響範囲を、次に示します。

- ***スタック***のサイズを縮小した場合、スタックオーバーフローが発生することがあります。
- ***スタック***のサイズを拡張した場合、その分ユーザ空間や仮想メモリが少なくなるため、Javaヒープやネイティブモジュールの動作に必要な領域を確保できず、メモリ不足になることがあります。

9.4.3 暖機運転

Javaアプリケーション実行において、動的コンパイルから受ける影響の確認方法、およびその対応方法である暖機運転について説明します。

チューニング方法

"9.3 動的コンパイル"で説明したように、Javaアプリケーションの実行は、Java VM起動直後はインタプリタ実行だけで行われ、次第にインタプリタ実行と動的コンパイル結果による機械命令実行との混合動作になります。また動的コンパイルにより翻訳された機械命令の内容も、Javaアプリケーションの実行が進むにつれて、次第にJavaアプリケーションの実行状況に合った翻訳結果に最適化されます。つまりJavaアプリケーションの実行には、以下の動的コンパイルによるオーバーヘッドや最適化状態の推移があるため、Javaアプリケーションとして安定した実行性能になるまで、プロセス起動時からある程度の時間を必要とする場合があります。

- 実行対象プログラムであるクラスファイルを実行時に読み込むため、Javaアプリケーションとしての起動直後には、クラスファイルのロードおよび内容検査によるオーバーヘッドが発生します。
- 実行時に機械命令への翻訳処理を行うため、そのオーバーヘッドが発生します。
- 機械命令への翻訳・最適化処理で必要とする情報を、Javaアプリケーション実行と同時に収集するため、Javaアプリケーション開始直後は最適化状態の推移が少なく、結果として実行性能が遅い機械命令となっている場合があります。

なお、Javaアプリケーションとして安定した実行性能になるまでに掛かる時間は、各アプリケーションによって異なります。

動的コンパイル発生状況の調査

Javaアプリケーション実行時における動的コンパイルから受ける影響の有無は、Javaアプリケーションによる業務が開始された際に、動的コンパイルの発生頻度が高いかどうかを判断することで行います。

具体的には、“9.3.3 動的コンパイル発生状況のログ出力機能”を用いてコンパイラスレッドのCPU使用状況や動的コンパイル結果情報を出力し、その結果から以下の傾向が見て取れ、かつJavaアプリケーションとして安定した実行性能になるまでに掛かる時間との関連性が見て取れるかどうかを元に判断します。

- コンパイラスレッドのCPU使用状況において、経過時間に対してCPU時間の割合が高い場合
- 動的コンパイル結果情報において、Javaメソッドのコンパイルが、短い間に連続して発生している場合

Javaアプリケーション起動時や、業務を開始してJavaアプリケーションへの入力が始まる時に、動的コンパイル処理が発生する頻度が高くなる傾向は、Javaアプリケーション実行時の一般的な傾向です。そのため、動的コンパイル発生状況の調査の結果として、「コンパイラスレッドのCPU時間の割合が高い」や「動的コンパイルの発生頻度が高い」などの傾向が見て取れる場合であっても、インタプリタ実行と機械命令による実行がバランス良く行われており、対応が不要な場合が多々あります。

動的コンパイル発生状況の調査は、Javaアプリケーションとして安定した実行性能になるまでに掛かる時間が実運用上の問題となった場合に行ってください。

Javaアプリケーション起動直後の性能に影響を与える処理は、動的コンパイル処理だけではありません。Javaアプリケーション自体の初期化処理や関連するアプリケーションの起動待ちなど、Javaアプリケーションの起動直後でだけ動作する動的コンパイル以外の要因についても考慮する必要があります。

ServletやEJBなどのJavaアプリケーションがInterstage Application Server配下で動作する場合、「Javaアプリケーションの起動(開始)」には以下の2つの意味があります。

a. Interstage Application Serverの起動(Java EE 6 環境のIJServerクラスタ起動)
b. 業務アプリケーションの運用開始

動的コンパイル処理が行われる頻度が高まるのは(a)の直後と(b)の直後の両方ですが、業務としての影響を確認することが目的であるため、動的コンパイル発生状況の調査対象は(b)の状況になります。

なお、動的コンパイル結果情報(-XX:+FJPrintCompilationオプション指定時の出力結果)から、(b)の始まりを調べるには、システムログなどから(a)の完了時刻を調べ、その時刻以降に発生した動的コンパイルを対象に調査を行います。

JavaアプリケーションがJSPで作成されている場合、アプリケーションを配備する際にJSPのプリコンパイル(javacコマンドによりJavaソースをクラスファイルへ変換する処理)が実施されていないと、Javaアプリケーション起動時にjavacコマンド実行によるオーバヘッドが発生する場合があります。

JSPのプリコンパイル運用ができる場合は、JSPのプリコンパイルを実施することで、Javaアプリケーションとして安定した実行性能になるまでに掛かる時間が小さくなるかどうかを確認してください。

暖機運転

Javaアプリケーションを起動した後、業務としての運用を開始する時点で安定した実行性能を得る状態にするためには、暖機運転という運用を行います。

暖機運転とは、Javaアプリケーションを起動した後、実際の業務を開始する前に、業務と同様のダミーデータを用いて疑似実行させることで、事前に主なJavaメソッドを動作させ、クラスファイルのロードやJavaメソッドの動的コンパイルを完了させておくことを言います。

暖機運転により、動的コンパイルなどのオーバーヘッドを減らすことができ、また機械命令への翻訳・最適化処理もその時点で行われるようになるため、業務としての運用開始時点から安定した実行性能を得ることができるようになります。

なお暖機運転をどの程度の期間行ったら良いのかについては、Javaアプリケーションの実行性能が運用要件を満たすところまで安定しているかのかどうかが、判断の基準になります。ただし、Javaアプリケーションを起動してから業務開始までの時間は無限にあるわけではないので、"9.3.3 動的コンパイル発生状況のログ出力機能"の動的コンパイル結果情報の出力結果を参考に、暖気運用に掛ける時間をある程度のところで区切る判断をする必要があります。

注意

通常、Javaアプリケーションの実行には、実行結果の表示、データの更新などの結果を伴います。暖機運転による実行結果が、Javaアプリケーションの運用自体に影響を与えないように注意する必要があります。

例えば、暖機運転用のダミーデータをそのままデータベースに登録してしまい、運用時に誤ったデータを返却するなどの問題が発生しないように注意してください。

注意

Javaアプリケーション自体の初期化処理や関連するアプリケーションの起動待ちなど、動的コンパイル以外の要因でJavaアプリケーションとして安定した実行性能になるまでに時間が掛かっている場合は、暖機運転では対処することができません。

例

動的コンパイル結果情報(-XX:+FJPrintCompilationオプション指定時)の出力例

```
0.133: 1 sun.misc.ASCIICaseInsensitiveComparator::compare (143 bytes)
0.137: 2 java.lang.String::charAt (33 bytes)
0.142: 3 java.lang.String::hashCode (64 bytes)
0.204: 4 java.lang.String::equals (88 bytes)
0.210: 5 java.lang.String::indexOf (151 bytes)
0.215: 6 java.lang.AbstractStringBuilder::append (40 bytes)
0.226: 7 java.lang.String::replace (142 bytes)
(中略)
0.444: 30 java.util.jar.Attributes::read (410 bytes)
0.470: 31 java.lang.String::<init> (111 bytes)
0.486: [CompilerThread0: cpu=140.63ms elapsed=367.55ms 1]
0.486: 32 java.util.jar.Attributes$Name::isValid (32 bytes)
0.487: 33 java.util.jar.Attributes$Name::isAlpha (30 bytes)
0.487: 34 sun.nio.cs.ext.MS932$Decoder::decodeSingle (19 bytes)
(中略) Application Server起動時は、頻繁に動的コンパイルが発生
15.449: 796 net.jxta.impl.document.LiteXMLElement::addAttribute (450 bytes)
15.538: 797 java.io.ObjectOutputStream::writeHandle (21 bytes)
15.574: 798 java.util.Hashtable$Enumerator::hasNext (5 bytes)
15.600: 799 com.sun.enterprise.config.ConfigBean::addXPathToChild (27 bytes)
15.606: [CompilerThread0: cpu=250.00ms elapsed=781.02ms 24]
15.630: 800 java.util.regex.Matcher::<init> (84 bytes)
23.535: 801 java.util.Arrays::copyOf (47 bytes)
25.542: 802 java.util.Arrays::copyOf (13 bytes)
25.545: 803 net.jxta.document.MimeMediaType::findNextSeperator (39 bytes)
43.567: 804 net.jxta.impl.document.LiteXMLElement::getTagRanges (1697 bytes)
この時点でInterstage Application Serverとしての起動が完了し、動的コンパイルの発生がいったん減少
(Interstage Application Serverの起動(JavaEE 6 環境のIJServerクラスタ起動)の完了時間は、Interstage Application
```

Serverのログ(サーバ情報のログ) から調べることができます。)
44.212: 805 org.apache.tomcat.util.buf.Ascii::toLower (14 bytes)
44.223: 806 org.apache.tomcat.util.buf.ByteChunk::equalsIgnoreCase (76 bytes)
44.224: 807 sun.nio.cs.ISO_8859_1$Decoder::decodeArrayLoop (263 bytes)
44.240: 18% java.util.Properties$LineReader::readLine @ 21 (452 bytes)
44.258: 808 org.apache.coyote.http11.InternalInputBuffer::parseHeader (585 bytes)
44.265: 809 java.util.Properties$LineReader::readLine (452 bytes)
44.290: 810 java.lang.ThreadLocal$ThreadLocalMap::nextIndex (15 bytes)
44.332: 811 java.util.Hashtable$Enumerator::next (27 bytes)
44.387: 812 java.net.URI::quote (208 bytes)
44.412: 19% sun.nio.cs.ext.DoubleByteEncoder::encodeArrayLoop @ 55 (608 bytes)
(中略) 業務が開始し、再び頻繁に動的コンパイルが発生
158.121: 1128 sun.util.calendar.ZoneInfo::getOffsetsByWall (8 bytes)
160.374: 1129 javax.management.NotificationBroadcasterSupport::sendNotification (118 bytes)
160.698: 1130 java.util.TimeZone$DisplayNames::access$000 (4 bytes)
161.405: 1131 com.sun.jmx.remote.internal.ArrayNotificationBuffer::addNotification (144 bytes)
163.572: 1132 sun.nio.cs.ext.DoubleByteDecoder::decodeLoop (28 bytes)
164.460: 1133 sun.util.calendar.CalendarDate::isDaylightTime (22 bytes)
170.743: 1134 java.util.AbstractCollection::toArray (116 bytes)
業務開始からしばらく経過した時点で、動的コンパイルの発生が減少

9.5 チューニング/デバッグ技法

チューニング技法およびデバッグ技法を紹介します。

9.5.1 スタックトレース

Javaアプリケーションで例外(java.lang.Throwableのインスタンス)がスローされた場合などに出力される**スタックトレース**は、エラーが発生するまでの経緯(メソッドの呼び出し順番)が示されています。このスタックトレースを解析することにより、エラーが発生した箇所と原因を確認することができます。

スタックトレースの出力先

スタックトレースの出力先は、標準エラーです。通常のJavaアプリケーションの場合は、コンソールに出力されますが、Servlet/JSP/EJBアプリケーションの場合は、ログファイル(標準出力、標準エラー出力あるいはJava VMの出力を格納するファイルなど)に出力されます。

スタックトレースの出力方法

Javaでスローされた例外をcatch節でキャッチし、例外のprintStackTraceメソッドを実行することにより、**スタックトレース**を出力することができます。

java.lang.Throwable.printStackTrace()メソッドでスタックトレースを出力する方法

```
try {
    SampleBMPSessionRemote bmpSessionRemote = bmpSessionHome.create();
} catch(Exception e) {
    e.printStackTrace();
}
```

なお、スローされた例外をtry-catch構文で処理するメソッドがスレッドにない場合、そのスレッドは停止され、Java VMによって**スタックトレース**が出力されます。

スタックトレースの出力フォーマット

スタックトレースの出力フォーマット

```
例外クラス名: エラーメッセージ
  at クラス名.メソッド名1(ソース名:行番号)     呼び出し先
```

```
at クラス名.メソッド名2(ソース名:行番号)
            :
            :
at クラス名.メソッド名N(ソース名:行番号)     呼び出し元
```

- 最初の1行目は、スローされた例外のクラス名とエラーメッセージです。
  エラーメッセージがない場合もあります。
- 2行目以降は、メソッドの呼び出し元(クラス名.メソッド名N)から呼び出し先(クラス名.メソッド名1)に向かって下から上に出力されます。
  2行目(クラス名.メソッド名1)が、例外をスローしたメソッドの情報です。
- “メソッド名”が“<init>”の場合、コンストラクタを示します。
- “メソッド名”が“<clinit>”の場合、static initializerを示します。
- “(ソース名:行番号)”が“(Native Method)”の場合、Javaのネイティブメソッド(.soや.dllファイル)を示します。
- クラスのコンパイル時にデバッグ情報を削除した場合、“(ソース名:行番号)”には、ソース名しか表示されなかったり、“Unknown Source”と表示されたりする場合があります。

9.5.1.1 スタックトレースの解析方法(その1)

以下の出力例をもとにして、解析方法を説明します。
先頭の“数字:”は、説明の便宜上、付加しています。

```
1:java.lang.NullPointerException
2:  at agency.attestation.CheckLoginInfo.doCheck(CheckLoginInfo.java:150)
3:  at agency.attestation.AttestationServlet.doGet(AttestationServlet.java:96)
4:  at agency.attestation.AttestationServlet.doPost(AttestationServlet.java:161)
5:  at javax.servlet.http.HttpServlet.service(HttpServlet.java:772)
6:  at javax.servlet.http.HttpServlet.service(HttpServlet.java:865)
            :
```

読み方

スタックトレースは、6行目から上方向に読むと、次の流れで例外が発生したことがわかります。

1. javax.servlet.http.HttpServlet.service()が、HttpServlet.javaの865行目で、javax.servlet.http.HttpServlet.service()を実行し、
2. javax.servlet.http.HttpServlet.service() が、HttpServlet.java の 772 行目で、agency.attestation.AttestationServlet.doPost()を実行し、
3. agency.attestation.AttestationServlet.doPost() が、AttestationServlet.java の 161 行目で、agency.attestation.AttestationServlet.doGet()を実行し、
4. agency.attestation.AttestationServlet.doGet() が、AttestationServlet.java の 96 行目で、agency.attestation.CheckLoginInfo.doCheck()を実行した結果、
5. agency.attestation.CheckLoginInfo.doCheck() 内の CheckLoginInfo.java の 150 行目で、java.lang.NullPointerExceptionという例外が発生した

解析方法

スタックトレースの解析例を、次に示します。

1. 1行目の例外情報から、原因を特定できるかどうか確認します。
   NullPointerExceptionがスローされていることがわかります。
2. 2行目のCheckLoginInfo.javaの開発担当者であれば、150行目の実装に問題がないかどうかを確認します。
3. 2行目のCheckLoginInfo.javaの開発担当者でない場合、**スタックトレース**中で最上行にある開発担当者が開発したクラスを探します。そして、そのクラスの実装に問題がないかどうかを確認します。それでも、原因を特定できない場合は、開発したクラスが使用しているクラスの提供元に調査を依頼します。

または、**スタックトレース**が、想定された流れでメソッドを実行しているかどうかを確認するのも1つの方法です。

9.5.1.2 スタックトレースの解析方法(その2)

以下の出力例をもとにして、解析方法を説明します。
先頭の“数字:”は、説明の便宜上、付加しています。

```
1:java.util.MissingResourceException: Can't find bundle for base name sample.SampleResource, locale ja_JP
2:  at java.util.ResourceBundle.throwMissingResourceException(Unknown Source)
3:  at java.util.ResourceBundle.getBundleImpl(Unknown Source)
4:  at java.util.ResourceBundle.getBundle(Unknown Source)
5:  at sample.SampleMessage.getMessage(SampleMessage.java:15)
6:  at sample.SampleServlet.doGet(SampleServlet.java:10)
7:  at javax.servlet.http.HttpServlet.service(HttpServlet.java:696)
8:  at javax.servlet.http.HttpServlet.service(HttpServlet.java:809)
    :
    :
```

解析方法

スタックトレースの解析例を、次に示します。

1. 1行目の例外情報から、原因を特定できないかを確認します。

   APIリファレンスによると、java.util.MissingResourceExceptionは、Javaのリソースがない場合に発生する例外です。また、エラーメッセージによると、sample.SampleResourceというリソースファイルの日本語版(ja_JP)がないということがわかります。

2. リソースファイルを確認します。

   a. リソースファイル名を誤っていないか

      SampleMessage.java の 15 行目の sample.SampleMessage.getMessage () 内で、java.util.ResourceBundle.getBundle()を実行した結果、例外がスローされています。したがって、そこでjava.util.ResourceBundle.getBundle()に渡しているリソースファイル名に誤りがないかどうかを確認します。

   b. リソースファイルが、所定のディレクトリ構成内に存在するか

      a)のリソースファイル名が正しい場合、所定のディレクトリ構成(/sample/)に、次のどれかのリソースファイルがあるかどうかを確認します。

      - SampleResource_ja_JP.properties
      - SampleResource_ja_JP.class
      - SampleResource_ja.properties
      - SampleResource_ja.class
      - SampleResource.properties
      - SampleResource.class

9.5.1.3 スタックトレースの解析方法(その3)

JDK/JRE 1.4以降、java.lang.Throwableに次のコンストラクタとメソッドが追加されました。

- Throwable(java.lang.String, java.lang.Throwable)
- Throwable(java.lang.Throwable)
- initCause(java.lang.Throwable)

これにより、**スタックトレース**には、原因となる例外の**スタックトレース**も出力されるようになりました。

以下のサンプルを使って、説明します。

```
1 :public class Test {
2 :
3 :    public static void main(String[] args) {
4 :       new Test();
5 :    }
6 :
7 :    Test() {
8 :        try{
9 :            parentMethod();
10:        } catch (Exception e) {
11:            e.printStackTrace();
12:        }
13:    }
14:
15:    void parentMethod() throws HiLevelException {
16:        try {
17:            childMethod();
18:        } catch (Exception e) {
19:            throw new HiLevelException("HiLevel", e);
20:        }
21:    }
22:
23:    void childMethod() throws LowLevelException {
24:        throw new LowLevelException("LowLevel");
25:    }
26:}
27:
28:class HiLevelException extends Exception {
29:    HiLevelException(String msg, Throwable cause) {
30:        super(msg, cause);
31:    }
32:}
33:
34:class LowLevelException extends Exception {
35:    LowLevelException(String msg) {
36:        super(msg);
37:    }
38:}
```

サンプルを実行すると、以下の**スタックトレース**が出力されます。

```
HiLevelException: HiLevel
    at Test.parentMethod(Test.java:19)
    at Test.<init>(Test.java:9)
    at Test.main(Test.java:4)
Caused by: LowLevelException: LowLevel
    at Test.childMethod(Test.java:24)
    at Test.parentMethod(Test.java:17)
    ... 2 more
```

HiLevelExceptionに続いて、“Caused by:”以降に、原因となるLowLevelExceptionのスタックトレースが出力されています。
最終行の“... 2 more”は、“Caused by:”の直前の2行が続きの**スタックトレース**であることを示しています。
つまり、以下のように解釈することができます。

原因となる例外の解釈

```
Caused by: LowLevelException: LowLevel
    at Test.childMethod(Test.java:24)
    at Test.parentMethod(Test.java:17)
    at Test.<init>(Test.java:9)
    at Test.main(Test.java:4)
```

以上から、次のことがわかります。

- **スタックトレース**の原因は、LowLevelExceptionである
- Test.main(Test.javaの4行目)が、最初の呼び出し元である。

詳細は、Java APIリファレンスのjava.lang.ThrowableのprintStackTraceメソッドの解説を参照してください。

9.5.2 例外発生時のスタックトレース出力

FJVMを使用してJavaアプリケーションを実行している場合、以下の例外については、実行性能の観点から、Java VMの動的コンパイル処理が行う最適化処理により例外発生時のスタックトレース出力処理が省略され、例外発生時のスタックトレースが出力されない場合があります(例外発生時のスタックトレース出力抑止)。

- java.lang.NullPointerException
- java.lang.ArithmeticException
- java.lang.ArrayIndexOutOfBoundsException
- java.lang.ArrayStoreException
- java.lang.ClassCastException

動的コンパイル処理により例外発生時のスタックトレース出力処理が省略されないようにする場合は、"-XX:-OmitStackTraceInFastThrow"オプションを指定します。

ただし該当する例外の発生頻度が高い場合に当該オプションを指定し、例外発生時のスタックトレース出力を行った場合は、Javaアプリケーションの実行性能が低下する場合があります。

当該オプションを指定する場合は、性能検証を行った上で使用するか、開発作業において例外が発生している場所を特定したい場合においてだけ使用してください。

"-XX:-OmitStackTraceInFastThrow"オプションはFJVM固有機能です。

インタプリタで実行しているJavaメソッド内で上記例外が発生した場合は、当該オプションの指定に関係なく、例外発生時のスタックトレースが出力されます。

9.5.3 スレッドダンプ

スレッドダンプには、Javaプロセスの各スレッドの情報(スタックトレース形式)が含まれているため、ハングアップやデッドロックなどの動作具合を調査することができます。

スレッドダンプの出力先は、標準出力です。**スレッドダンプ**が出力される契機および出力先を、"表9.8 スレッドダンプの出力契機と出力先"に示します。

表9.8 スレッドダンプの出力契機と出力先

プログラムの種類	出力契機	出力先
EE Java EE 6 アプリケーション	一定の条件を満たした場合、コンテナの機能により自動的に採取される場合と利用者の任意のタイミングで手動による採取があります。 ・ 自動採取:アプリケーションがタイムアウトまたは無応答になった場合 ・ 手動採取: Windows32/64 "スレッドダンプツール"で採取することができます。スレッドダンプツールの詳細は、"トラブルシューティング集"の"スレッドダンプツール"を参照してください。 Solaris64 Linux32/64 kill -QUIT [プロセスID]でJava VMに対してQUITシグナルを送り採取することができます。	標準出力、JavaVMの出力をロギングしているファイル

プログラムの種類	出力契機	出力先
上記以外のJavaプログラム	利用者の任意のタイミングで手動で採取することができます。 Windows32/64 ・コマンドプロンプトからJavaプログラムを起動した場合: 以下、どちらかの方法で採取できます。 1) [Ctrl]+[Break]キー押下 2) “スレッドダンプツール” ・コマンドプロンプト以外からJavaプログラムを起動した場合: “スレッドダンプツール”で採取します。 スレッドダンプツールの詳細は、“トラブルシューティング集”の“スレッドダンプツール”を参照してください。 Solaris64 Linux32/64 ・ターミナルからJavaプログラムを起動した場合: 以下、どちらかの方法で採取できます。 1) [Ctrl]+[¥]キー(英語キーボードの場合バックスラッシュキー)押下 2) kill -QUIT [プロセスID] ・ターミナル以外からJavaプログラムを起動した場合: kill -QUIT [プロセスID]で採取します。	コンソール(標準出力)

注意

“-Xrs”オプションが指定されたJavaプロセスの場合、当該プロセスへ送られた[Ctrl]+[Break]キー押下またはQUITシグナルに対する動作は、OSのデフォルト動作になります。

そのため、“-Xrs”オプションを指定したJavaプロセスに対して[Ctrl]+[Break]キー押下またはQUITシグナルが送られると、当該Javaプロセスは強制終了または異常終了します。
スレッドダンプを出力する可能性があるJavaプロセスに対して、“-Xrs”オプションは指定しないでください。
ただし、Windows(R)でサービスとして登録されるJavaプロセスの場合は、“-Xrs”オプションを指定しない場合、ログオフ時に強制終了してしまいます。これが不都合な場合は、“-Xrs”オプションを指定してください。

以下のサンプルプログラムをもとにして、**スレッドダンプ**の解析方法を説明します。

```
1 :public class DeadlockSample {
2 :    static boolean flag;
3 :    static Thread1 thread1;
4 :    static Thread2 thread2;
5 :
6 :    public static void main(String[] args) {
7 :        thread1 = new Thread1();
8 :        thread2 = new Thread2();
9 :        thread1.start();
10:        thread2.start();
11:    }
12:}
13:
14:class Thread1 extends Thread {
15:    public Thread1(){
```

```
16:         super("Thread1");
17:     }
18:
19:     public void run(){
20:         synchronized(this){
21:             System.out.println("Thread1開始");
22:             while(DeadlockSample.flag==false){  // Thread2が開始するのを待つ
23:                 yield();
24:             }
25:             DeadlockSample.thread2.method();
26:             notify();
27:         }
28:     }
29:
30:     public synchronized void method(){
31:         try{wait(1000);}catch(InterruptedException ex){}
32:         System.out.println("Thread1.method()終了");
33:     }
34:}
35:
36:class Thread2 extends Thread {
37:     public Thread2(){
38:         super("Thread2");
39:     }
40:
41:     public void run() {
42:         synchronized(this){
43:             DeadlockSample.flag = true;
44:             System.out.println("Thread2開始");
45:             DeadlockSample.thread1.method();
46:             notify();
47:         }
48:     }
49:
50:     public synchronized void method() {
51:         try{wait(1000);}catch(InterruptedException ex){}
52:         System.out.println("Thread2.method()終了");
53:     }
54:}
```

サンプルでは、Thread1とThread2がお互いに排他処理を行っています。
このサンプルを実行すると、次のように処理が進められます。

1. Thread1で、Thread1のロックを獲得する(20行目のsynchronized節)
2. Thread2で、Thread2のロックを獲得する(42行目のsynchronized節)
3. Thread1で、Thread2.method()を実行しようとして、ロック解放待ちになる(50行目のsynchronized修飾子)
4. Thread2で、Thread1.method()を実行しようとして、ロック解放待ちになる(30行目のsynchronized修飾子)

この結果、Thead1とThread2がお互いに、解放されないロックを待ち続けるデッドロック状態になります。
デッドロック状態で、**スレッドダンプ**を採取したものを、以下に示します。

スレッドダンプ

```
"DestroyJavaVM" prio=5 tid=0x002856c8 nid=0x5f4 waiting on condition [0..6fad8]

"Thread2" prio=5 tid=0x0092f4d8 nid=0x640 waiting for monitor entry [182ef000..182efd64]
    at Thread1.method(DeadlockSample.java:31)
    - waiting to lock <0x1002ffe8> (a Thread1)
    at Thread2.run(DeadlockSample.java:45)
    - locked <0x10030ca0> (a Thread2)

"Thread1" prio=5 tid=0x0092f370 nid=0x294 waiting for monitor entry [182af000..182afd64]
```

```
    at Thread2.method(DeadlockSample.java:51)
    - waiting to lock <0x10030ca0> (a Thread2)
    at Thread1.run(DeadlockSample.java:25)
    - locked <0x1002ffe8> (a Thread1)

"Signal Dispatcher" daemon prio=10 tid=0x0098eb80 nid=0x634 waiting on condition [0..0]

"Finalizer" daemon prio=9 tid=0x0092a540 nid=0x5e8 in Object.wait() [1816f000..1816fd64]
    at java.lang.Object.wait(Native Method)
    - waiting on <0x10010498> (a java.lang.ref.ReferenceQueue$Lock)
    at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:111)
    - locked <0x10010498> (a java.lang.ref.ReferenceQueue$Lock)
    at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:127)
    at java.lang.ref.Finalizer$FinalizerThread.run(Finalizer.java:159)

"Reference Handler" daemon prio=10 tid=0x0096da70 nid=0x5e4 in Object.wait() [1812f000..1812fd64]
    at java.lang.Object.wait(Native Method)
    - waiting on <0x10010388> (a java.lang.ref.Reference$Lock)
    at java.lang.Object.wait(Object.java:429)
    at java.lang.ref.Reference$ReferenceHandler.run(Reference.java:115)
    - locked <0x10010388> (a java.lang.ref.Reference$Lock)

"VM Thread" prio=5 tid=0x0096c950 nid=0x624 runnable

"VM Periodic Task Thread" prio=10 tid=0x0092c008 nid=0x2a0 waiting on condition
"Suspend Checker Thread" prio=10 tid=0x0098e118 nid=0x478 runnable

Found one Java-level deadlock:
=============================
"Thread2":
  waiting to lock monitor 0x00929c3c (object 0x1002ffe8, a Thread1),
  which is held by "Thread1"
"Thread1":
  waiting to lock monitor 0x00929c5c (object 0x10030ca0, a Thread2),
  which is held by "Thread2"

Java stack information for the threads listed above:
===================================================
"Thread2":
    at Thread1.method(DeadlockSample.java:31)
    - waiting to lock <0x1002ffe8> (a Thread1)
    at Thread2.run(DeadlockSample.java:45)
    - locked <0x10030ca0> (a Thread2)
"Thread1":
    at Thread2.method(DeadlockSample.java:51)
    - waiting to lock <0x10030ca0> (a Thread2)
    at Thread1.run(DeadlockSample.java:25)
    - locked <0x1002ffe8> (a Thread1)

Found 1 deadlock.
```

解析方法

スレッドダンプの各スレッドの情報は、スタックトレース形式です。
Thread1とThread2の両方のスタックトレースには、“locked”と“waiting to lock”があります。また、**スレッドダンプ**の下の方にも、“deadlock”の文字列があり、デットロックが発生していることが確認できます。

このように、**スレッドダンプ**で全スレッドの動作状況を確認するにより、Javaプロセスがハングアップしているか、あるいは、デッドロック状態かを確認することができます。特に、短い間隔で複数のスレッドダンプを採取し、スレッドに動きがなければ、ハングアップの可能性があります。

スレッドダンプの詳細は、“トラブルシューティング集”も参照してください。

オブジェクトをロックしているスレッドがスレッドダンプ上に出ない

通常スレッドダンプ上のあるスレッドで次のように表示される場合があります。

```
- waiting to lock <オブジェクトID> (a クラス名)
```

このような場合、別のスレッドがそのオブジェクトIDのロックを持っていて、そのスレッドのトレース上のどこかで次の表示がされています。

```
- locked <オブジェクトID> (a クラス名)
```

しかし、スレッドダンプを表示するタイミングによっては“- locked <オブジェクトID> (a クラス名)”の表示がどのスレッドにも現われず、“- waiting to lock <オブジェクトID> (a クラス名)”だけ表示される場合があります。

以下のプログラムを例とします。

```
 1  class NoLockOwner extends Thread
 2  {
 3    static Object lock = new Object();
 4
 5    public static void main(String[] arg)
 6    {
 7      new NoLockOwner().start();
 8      new NoLockOwner().start();
 9    }
10
11    public void run()
12    {
13      while (true) {
14        synchronized (lock) {
15          dumb();
16        }
17      }
18    }
19
20    void dumb()
21    {
22      int n = 0;
23      for (int i = 0 ; i < 1000 ; ++i)
24        n += i;
25    }
26
27  }
```

(0) スレッドダンプを取ると、通常はこのようになります。

```
"Thread-1" prio=1 tid=0x10 nid=0x5 waiting for monitor entry [0x3000..0x4000]
    at NoLockOwner.run(NoLockOwner.java:14)
    - waiting to lock <0x800> (a java.lang.Object)
"Thread-0" prio=1 tid=0x20 nid=0x6 runnable [0x5000..0x6000]
    at NoLockOwner.dumb(NoLockOwner.java:23)
    at NoLockOwner.run(NoLockOwner.java:15)
    - locked <0x800> (a java.lang.Object)
```

(1) 先頭フレームでオブジェクトをロックしている場合は“- locked”ではなく、“- waiting to lock”と表示されることがあります。

```
"Thread-1" prio=1 tid=0x10 nid=0x5 waiting for monitor entry [0x3000..0x4000]
    at NoLockOwner.run(NoLockOwner.java:14)
    - waiting to lock <0x800> (a java.lang.Object)
"Thread-0" prio=1 tid=0x20 nid=0x6 runnable [0x5000..0x6000]
    at NoLockOwner.run(NoLockOwner.java:14)
    - waiting to lock <0x800> (a java.lang.Object)
```

この場合、スレッドの状態を見て、runnableであれば、ロック待ちではなく、ロック取得後同じフレームを実行中の状態であると考えます。

Thread-0、Thread-1ともに、“- waiting to lock <0x800> (a java.lang.Object)”と表示されているので、どちらもロック待ちのように見えます。

しかし、Thread-0の状態は、runnbleなので、ロック待ち状態ではありません。

(2)“- waiting to lock”と表示されるのは、ロック待ちの状態だけではなく、ロック獲得処理の最中でもそのように表示されます。

```
"Thread-1" prio=1 tid=0x10 nid=0x5 waiting for monitor entry [0x3000..0x4000]
    at NoLockOwner.run(NoLockOwner.java:14)
    - waiting to lock <0x800> (a java.lang.Object)
"Thread-0" prio=1 tid=0x20 nid=0x6 waiting for monitor entry [0x5000..0x6000]
    at NoLockOwner.run(NoLockOwner.java:14)
    - waiting to lock <0x800> (a java.lang.Object)
```

Thread-0、Thread-1ともに、“- waiting to lock <0x800> (a java.lang.Object)”と表示されているので、どちらもロック待ちのように見えます。

さらに、スレッドの状態も、どちらも、“waiting for monitor entry”になっています。

“- waiting to lock”と表示されるのは、ロック待ちの状態だけではなく、ロック獲得処理の最中でもそのように表示されます。

したがって、この場合、Thread-0またはThread-1のどちらか一方、あるいは両方がロック獲得処理の最中であると考えます。

しかし、この状態は長く続かず、短時間で(0)または(1)に移行します。

(3) ロックを開放した直後の状態

```
"Thread-1" prio=1 tid=0x10 nid=0x5 waiting for monitor entry [0x3000..0x4000]
    at NoLockOwner.run(NoLockOwner.java:14)
    - waiting to lock <0x800> (a java.lang.Object)
"Thread-0" prio=1 tid=0x20 nid=0x6 runnable [0x5000..0x6000]
    at NoLockOwner.run(NoLockOwner.java:16)
```

Thraed-0がちょうどロックを開放した直後の状態です。

この状態も長く続くとはなく、短時間で(0)または(1)に移行します。

スレッドダンプからアプリケーションの状態を判断する場合、ひとつのスレッドダンプから状態を判断することは困難です。

適切な判断をするためには、複数回のスレッドダンプを総合的に見ることが必要です。

スレッドダンプ中に表示されるsynchronizedメソッドの行番号について

次のようなプログラムを考えます。

```
 1   class SyncMethod extends Thread
 2   {
 3     static volatile int k;
 4
 5     public static void main(String[] arg)
 6     {
 7       new SyncMethod().start();
 8     }
 9
10     public void run()
11     {
12       while (true) {
13         dumb();
14       }
15     }
16
17     synchronized void dumb()
18     {
19       /*
20         meaningless comments
21        */
22       int i = 0;
23       for ( ; i < 10 ; ++i)
```

```
24          k += i;
25      }
26
27  }
```

このようなプログラムのスレッドダンプを取ると以下のように出力されることがあります。

```
"Thread-0" prio=1 tid=0x300 nid=0x61 runnable [1000..2000]
    at SyncMethod.dumb(SyncMethod.java:23)
    - waiting to lock <0xa00> (a SyncMethod)
    at SyncMethod.run(SyncMethod.java:13)
```

23行目でロックを獲得しているように見えますが、23行目は、“for (; i < 10 ; ++i)”であり、何もロック獲得に関係しているようにソースコード上は見えません。

これは、synchronizedメソッドがロックを獲得する行番号は、最初に実行される行番号になるためです。

注意

スレッドダンプ中、複数のスレッドがロックを獲得している場合ついて

次のようなプログラムを考えます。

```
1   class NoNotify extends Thread
2   {
3     static Object o = new Object();
4
5     public static void main(String[] arg)
6     {
7       new NoNotify().start();
8       new NoNotify().start();
9     }
10
11    public void run()
12    {
13      try {
14        synchronized (o) {
15          o.wait();
16        }
17      } catch (Exception e) {}
18    }
19
20  }
```

このプログラムには誤りがあります。

このプログラムでは誰もnotifyするスレッドがないため、どちらのスレッドも永久に起こされることはありません。

このプログラムのスレッドダンプを採取すると次のように示される場所があります。

```
"Thread-1" prio=1 tid=0x800 nid=0x6 in Object.wait() [1000..2000]
    at java.lang.Object.wait(Native Method)
    - waiting on <0x200> (a java.lang.Object)
    at java.lang.Object.wait(Object.java:429)
    at NoNotify.run(NoNotify.java:15)
    - locked <0x200> (a java.lang.Object)
"Thread-0" prio=1 tid=0x900 nid=0x7 in Object.wait() [3000..4000]
    at java.lang.Object.wait(Native Method)
    - waiting on <0x200> (a java.lang.Object)
    at java.lang.Object.wait(Object.java:429)
    at NoNotify.run(NoNotify.java:15)
    - locked <0x200> (a java.lang.Object)
```

このスレッドダンプから分かることは、複数のスレッドがロックを獲得しているのではなく、どのスレッドもロックを獲得していないということです。

Thread-0、Thraed-1ともに、同じオブジェクト“ID<0x200>”のオブジェクトをロックしているように見えます。

しかし“- locked”と表示されているスレッドが、現在のロックオーナであることは意味しません。(“- locked”の正確な意味するところは、そのフレームにおいてロックしたということに過ぎません。)

先頭フレームでは、“- waiting on”と表示されています。

これは、ロックが解放されたら起こされる可能性のある“- waiting to lock”とは異なり、ロックが解放されても自動的に起こされることを意味しません。

注意

Object.wait()を呼び出していないにも関わらず、「in Object.wait()」と表示される

アプリケーションから直接Object.wait()を呼び出していないにも関わらず、以下のように「in Object.wait()」と表示される場合があります。

```
"Thread-1" prio=10 tid=0x70257c00 nid=0x3818 in Object.wait() [0x6fe9c000]
   java.lang.Thread.State: RUNNABLE
        at ClassLoadContention.run(ClassLoadContention.java:10)
```

これは、以下の例のように、複数スレッドで同じクラスをロードしようして、Java VM内部でObject.wait()と同等の処理が実行されているためです。

```
 1   class ClassLoadContention extends Thread
 2   {
 3     public static void main(String[] arg)
 4     {
 5       new ClassLoadContention().start();
 6       new ClassLoadContention().start();
 7     }
 8
 9     public void run() {
10       TargetClass.N++;
11     }
12
13     static class TargetClass {
14       static int N;
15       static {
16         try {
17            Thread.sleep(1000);
18         } catch (Exception e) {      }
19       }
20     }
21   }
```

ポイント

JavaVM制御用のスレッド

スレッドダンプにおいて、以下の名前で出力されているスレッドは、Java VM自身の制御用スレッドです。そのため、以下の名前で出力されているスレッドの情報には、アプリケーションとしての動作状態を解析する際の直接的な情報は含まれていません。

- "Attach Listener"
- "C2 CompilerThread*" ("*"部分は数字です)
- "Finalizer"
- "RAS Control Thread"

- "Reference Handler"
- "Signal Dispatcher"
- "VM Periodic Task Thread"
- "VM Thread"
- "Service Thread"
- "GC task thread#* (ParallelGC)" (パラレルGC使用時に存在します) (注1)
- "Concurrent Mark-Sweep GC Thread" (CMS付きパラレルGC使用時に存在します)
- "Surrogate Locker Thread (Concurrent GC)" (CMS付きパラレルGC使用時に存在します)
- "Gang worker#* (Parallel GC Threads)" (CMS付きパラレルGC使用時に存在します) (注1)
- "Gang worker#* (Parallel CMS Threads)" (CMS付きパラレルGC使用時に存在します) (注2)

注1)

"-XX:ParallelGCThreads"オプションで指定したGC処理用スレッドの数だけ存在します。なお、名前の"*"部分は数字です。

注2)

"-XX:ConcGCThreads"オプションで指定したCMS-GC処理用スレッドの数だけ存在します。なお、名前の"*"部分は数字です。

ポイント

Javaヒープ領域に関する情報の出力

スレッドダンプの出力と合わせて、Javaヒープ領域に関する情報も出力されます。
Javaヒープ領域に関する情報は、各ガーベジコレクション処理の違いにより、New世代領域、Old世代領域、Permanent世代領域の各領域に対応する出力文字列が異なります。
なお、パーセントで示されている値は、情報出力時点でJava VMがJavaヒープ用に利用可能な状態にしている(コミットしている)メモリ量に対する比率です。利用可能な上限値に対する比率ではありません。そのため、パーセントで示されている値は参照せず、K(キロ)単位で表示されているメモリ使用量の値と、オプションで指定された値(デフォルト値を含む)とを比較する方法で各値を利用してください。

- シリアルGC使用時:

  「def new generation」が「New世代領域」、「tenured generation」が「Old世代領域」、「compacting perm gen」が「Permanent世代領域」に関する情報です。

- パラレルGC使用時:

  「PSYoungGen」が「New世代領域」、「PSOldGen」が「Old世代領域」、「PSPermGen」が「Permanent世代領域」に関する情報です。

- CMS付きパラレルGC使用時:

  「par new generation」が「New世代領域」、「concurrent mark-sweep generation」が「Old世代領域」、「concurrent-mark-sweep perm gen」が「Permanent世代領域」に関する情報です。
  なお「Old世代領域」および「Permanent世代領域」における「object space」についての情報は出力されません。

出力例:

```
Heap
 PSYoungGen      total 7168K, used 5158K [0x0fd60000, 0x10470000, 0x10470000)
  eden space 7104K, 72% used [0x0fd60000,0x102658f0,0x10450000)
  from space 64K, 25% used [0x10450000,0x10454000,0x10460000)
  to   space 64K, 0% used [0x10460000,0x10460000,0x10470000)
 PSOldGen        total 4096K, used 162K [0x0c470000, 0x0c870000, 0x0fd60000)
  object space 4096K, 3% used [0x0c470000,0x0c498870,0x0c870000)
```

```
PSPermGen       total 16384K, used 2103K [0x08470000, 0x09470000, 0x0c470000)
 object space 16384K, 12% used [0x08470000,0x0867dd40,0x09470000)
```

9.5.4 クラスのインスタンス情報出力機能

以下のオプションを指定したJavaプロセスに対してスレッドダンプ出力の操作を行った場合、スレッドダンプの出力に続いて、Javaヒープ内に生存する各クラスのインスタンス情報が「クラスのインスタンス情報の出力形式」の形式で出力されます。FJVMでは、当該機能を「**クラスのインスタンス情報出力機能**」として実装しています。

クラスのインスタンス情報として、クラス毎のインスタンス数および合計サイズが出力されるため、Javaヒープ内におけるメモリリークなどの調査で利用することができます。

クラスのインスタンス情報の出力先は、標準出力です。クラスのインスタンス情報が出力される契機および出力先は、“9.5.3 スレッドダンプ”が出力される契機および出力先と同じです。

なお、-Xloggcオプションの指定がある場合は、クラスのインスタンス情報の出力先が標準出力から-Xloggcオプションで指定したファイルへ切り替わります。

スレッドダンプに続いて、クラスのインスタンス情報を出力する機能を有効にするオプション

```
-XX:+PrintClassHistogram
```

クラスのインスタンス情報の出力形式

```
num     #instances         #bytes  class name
----------------------------------------------
 $1:            $2             $3  $4
           :
           (略)
           :
Total           $5             $6
```

$1:

順位:クラスのインスタンス情報は、各クラスのインスタンスの合計サイズが大きい順に、ソートされて出力されます。

$2:

クラスのインスタンス数

$3:

クラスのインスタンスの合計サイズ

$4:

クラス名

$5:

$2の値の合計

$6:

$3の値の合計

注意

クラスのインスタンス情報出力機能では、不要なインスタンスを排除した後の情報を採取・出力します。そのため、事前処理としてFullGCを実行します。クラスのインスタンス情報出力の過度の使用は、FullGCの多発となるので注意してください。

注意

クラスのインスタンス情報出力に先立って行われるはずのFullGCが、ガーベジコレクション処理の実行抑止により実行できない状態にある場合は、以下のメッセージを標準出力へ出力した後、クラスのインスタンス情報出力の要求は取り消されます。

```
The PrintClassHistogram operation was canceled because GC could not be run.
```

9.5.5 java.lang.System.gc()実行時におけるスタックトレース出力機能

Javaアプリケーションが以下のJavaメソッドを頻繁に実行すると、Java VMに負荷がかかり、アプリケーションの応答性能を低下させる要因になることがあります。

- java.lang.System.gc()
- java.lang.Runtime.gc()

以降、java.lang.System.gc()(以降、System.gc()と略する)で代表して説明します。

FJVMでは、Javaアプリケーション実行時にSystem.gc()メソッドの実行状態の確認ができるように、当該メソッドを実行したJavaスレッドのスタックトレースを出力する機能を「**java.lang.System.gc()実行時におけるスタックトレース出力機能**」として実装しています。
java.lang.System.gc()実行時におけるスタックトレース出力機能は、以下のオプションを指定した場合に有効となります。

```
-XX:+PrintJavaStackAtSystemGC
```

本機能が有効な場合、Javaアプリケーション内でSystem.gc()メソッドが実行された場合には、当該メソッドを実行したJavaスレッドのスタックトレース情報が、以下のような形で標準出力へ出力されます。

例

java.lang.System.gc()実行時におけるスタックトレース出力機能による出力例

```
"main" prio=10 tid=0x087c1c00 nid=0xd2a runnable [0xb75ab000..0xb75ab214]
   java.lang.Thread.State: RUNNABLE
                        at java.lang.Runtime.gc(Native Method)
                        at java.lang.System.gc(System.java:928)
                        at SystemGC.main(SystemGC.java:8)

"main" prio=10 tid=0x087c1c00 nid=0xd2a runnable [0xb75ab000..0xb75ab214]
   java.lang.Thread.State: RUNNABLE
                        at java.lang.Runtime.gc(Native Method)
                        at SystemGC.foo(SystemGC.java:4)
                        at SystemGC.main(SystemGC.java:10)
```

SystemGC.mainからjava.lang.System.gc()が実行され、SystemGC.fooからjava.lang.Runtime.gc()が実行されたことを確認することができます。

なお、標準出力への出力結果は、**FJVM**のログ情報としてファイルへも格納されます。また“-verbose:gc”オプション指定などでガーベジコレクション処理の結果ログ出力を指定していた場合は、java.lang.System.gc()実行時におけるスタックトレース出力後に出力された結果ログ出力も、合わせてファイルへ格納されます。ファイル名や格納先はJava VM異常終了時のログ出力時と同じです。“9.5.8 FJVMログ”を参照してください。

参考

スタックトレース情報にスレッドの状態を表す情報(トレース情報の前の行)が表示されます。

出力例では「java.lang.Thread.State: RUNNABLE」となっていますが、"RUNNABLE"の部分が、"NEW"、"TIMED_WAITING (sleeping)"、"WAITING (on object monitor)"、"TIMED_WAITING (on object

monitor)" 、 "WAITING (parking)" 、 "TIMED_WAITING (parking)" 、 "BLOCKED (on object monitor)"、"TERMINATED"、"UNKNOWN"などの表示の場合があります。

9.5.6 Java VM終了時における状態情報のメッセージ出力機能

特別なメッセージ出力などがないまま、Javaプロセスが予想外の状態で終了してしまった場合の原因の1つとして、Javaアプリケーションが以下のどれかの処理を実行した場合が考えられます。

- java.lang.System.exit()を予想外の箇所で実行した
- java.lang.Runtime.exit()を予想外の箇所で実行した
- java.lang.Runtime.halt()を予想外の箇所で実行した

以降は、java.lang.System.exit()(以降、System.exit()と略する)で代表して説明します。

System.exit()の実行によりJavaアプリケーションが明示的にJavaプロセスを終了させた場合、Java VM側から見ると正常な仕様動作であるため、Java VMとして特別なメッセージ出力などは行いません。このため、ソースがないなど、内部処理動作の詳細が不明なJavaアプリケーションが予想外の状態で終了した場合には、ソース確認などが行えないため、System.exit()が実行されたかどうかを確認することができません。

このため**FJVM**では、Javaプロセス終了時にSystem.exit()が実行されたかどうかを確認可能にするための機能を、「**Java VM終了時における状態情報のメッセージ出力機能**」として実装しています。
Java VM終了時における状態情報のメッセージ出力機能は、以下のオプションを指定した場合に有効となります。

```
-XX:+VMTerminatedMessage
```

本機能が有効な場合、JavaプロセスがSystem.exit()の実行で終了した場合には、System.exit()を実行したスレッドのスタックトレース情報などが、以下の出力例のような形で標準出力へ出力されます。スタックトレース情報の出力の有無および内容を確認することにより、System.exit()実行の有無を確認することができます。
なお、標準出力への出力結果は、**FJVM**のログ情報としてファイルへも格納されます。ファイル名や格納先はJava VM異常終了時のログ出力時と同じです。“9.5.8 FJVMログ”を参照してください。

そして、本機能が有効な場合で、かつJavaプロセス終了時に以下の出力例のようなスタックトレースの情報が出力されなかった場合(「#### JavaVM terminated: …」から始まるメッセージ出力だけの場合)は、System.exit()が使用されず、Javaアプリケーション側の制御論理として終了したと考えられます。
また、本機能が有効な場合で、かつJavaプロセス終了時に以下の情報が何も出力されなかった場合は、ネイティブモジュールの中からCランタイムのexit()関数呼び出しによりJavaプロセスが終了したなど、別原因によるJavaプロセスの終了だと考えられます。

例

Java VM終了時における状態情報のメッセージ出力機能による出力例

```
Thread dump at JVM_Halt(status code=1234):
"main" prio=3 tid=0x00030000 nid=0x2 runnable [0xfe5ff000..0xfe5ffd80]
   java.lang.Thread.State: RUNNABLE
                        at java.lang.Shutdown.halt0(Native Method)
                        at java.lang.Shutdown.halt(Shutdown.java:105)
        - locked <0xfa81e7d0> (a java.lang.Shutdown$Lock)
                        at java.lang.Shutdown.exit(Shutdown.java:179)
        - locked <0xf3dc8dd8> (a java.lang.Class for java.lang.Shutdown)
                        at java.lang.Runtime.exit(Runtime.java:90)
                        at java.lang.System.exit(System.java:906)
                        at JVM_Halt.main(JVM_Halt.java:5)

#### JavaVM terminated: Java HotSpot(TM) Server VM
(**.*.**_FUJITSU_MODIFIED-B** mixed mode), [pid=29500] TimeMillis=1243580170483
Time=Fri May 29 15:56:10 2012
```

JVM_Halt.main()がSystem.exit()を実行し、その結果としてJavaプロセスが終了したことを確認することができます。

参考

スタックトレース情報にスレッドの状態を表す情報(トレース情報の前の行)が表示されます。

出力例では「java.lang.Thread.State: RUNNABLE」となっていますが、"RUNNABLE"の部分が、"NEW"、"TIMED_WAITING (sleeping)"、"WAITING (on object monitor)"、"TIMED_WAITING (on object monitor)"、"WAITING (parking)"、"TIMED_WAITING (parking)"、"BLOCKED (on object monitor)"、"TERMINATED"、"UNKNOWN"などの表示の場合があります。

この情報は**Java VM終了時における状態**の判定には使用しません。

9.5.7 ログ出力における時間情報のフォーマット指定機能

以下のオプションで出力されるログに含まれる「ログ出力時の時間」情報のフォーマットを、「ログ出力における時間情報のフォーマットを指定するオプション」で指定することができます。

- ガーベジコレクションのログ出力

  -verbosegc -XX:+UseFJverbose

- 動的コンパイル発生状況のログ出力機能

  -XX:+PrintCompilationCPUTime
  -XX:+FJPrintCompilation

ログ出力における時間情報のフォーマットを指定するオプション

-XX:FJverboseTime=タイプ タイプとして以下の値が指定できます type1 type2 type3

-XX:FJverboseTime=type1が指定された場合

ログ出力時の時間を、「Java VMが起動されてからの経過時間(秒)」で示します。
例) 0.165:

-XX:FJverboseTime=type2が指定された場合

ログ出力時の時間を、「日時(iso8601フォーマット)」で示します。
例) 2010-10-01T13:37:36.881+0900:

-XX:FJverboseTime=type3が指定された場合

ログ出力時の時間を、「日時(iso8601フォーマット)」とJava VMが起動されてからの「経過時間(秒)」で示します。出力順序は、「日時」「経過時間」です。
例) 2010-10-01T13:37:45.830+0900: 0.164:

このオプションによる指定がない場合、ログ出力時の時間は「Java VMが起動されてからの経過時間(秒)」(「-XX:FJverboseTime=type1」が指定された場合の形式)で出力されます。

9.5.8 FJVMログ

FJVMでは「Java VM異常終了時のログ出力機能」の強化を行っています。
何らかの原因でJavaプロセスが異常終了した場合、Java VM異常終了時のログとして**FJVMログ**が出力されます。
Javaプロセスが異常終了した原因の調査のために、この**FJVMログ**を活用することができます。

FJVMログの出力先

FJVMログは、Javaプロセスのカレントディレクトリに、以下のファイル名で出力されます。

fjvm_pid***.log (***は異常終了したJavaプロセスのプロセスID)

IJServerクラスタ使用時のカレントディレクトリの詳細は、“Java EE運用ガイド(Java EE 6編)”の“IJServerクラスタ”を参照してください。

FJVMログの調査

FJVMログとしてJavaプロセス異常終了時の各種情報が格納されますが、その中から以下の情報を原因調査用の情報として使用することができます。

- 9.5.8.1 異常終了箇所の情報
- 9.5.8.2 異常終了時のシグナルハンドラ情報 Solaris64 Linux32/64
- 9.5.8.3 異常終了時のJavaヒープに関する情報

9.5.8.1 異常終了箇所の情報

異常終了箇所に関する情報

異常終了箇所に関する情報が確認できます。

1. 異常終了時に発生した例外に関する情報(シグナルコードおよび例外発生アドレス)

   「Unexpected Signal :」から始まる情報です。
2. 異常終了した関数名(実際には異常終了したアドレスに一番近いシンボル名)

   「Function name=」から始まる情報です。
3. 異常終了した関数を含むライブラリ名

   「Library=」から始まる情報です。
4. 異常終了時のJavaスレッドのスタックトレース

   「Current Java thread:」から始まる情報です。
5. 異常終了時のダイナミックライブラリ一覧

   「Dynamic libraries:」から始まる情報です。
6. 発生時間

   「Local Time =」から始まる情報です。

調査手順

「異常終了箇所に関する情報」の1.～3.の情報で異常終了した関数を特定し、実行しているJavaアプリケーションから呼び出す関数かどうかを確認します。ただし、2.の異常終了した関数名として出力される名前は、異常終了したアドレスに一番近いシンボル名情報であるため、実際に異常終了した関数とは別の名前が出力されている場合がありますので注意してください。
そして、実行しているJavaアプリケーションが使用する関数の場合には、当該関数使用に際して何らかの問題がないか確認します。
実行しているJavaアプリケーションで使用していない関数の場合には、4.のスタックトレースを調査します。

スタックトレース情報の最初のメソッドがネイティブメソッドだった場合（メソッド名の後ろに「(Native Method)」が付加されている場合）はJNI処理に関係した問題である可能性が高いため、スタックトレース情報で出力された処理のJNI処理に関わる制御で何らかの問題がないか確認します。

また異常終了した関数を含むライブラリ名が利用者作成のライブラリである場合は、利用者側作成のライブラリ内の問題である可能性が高いため、当該ライブラリ内の処理および当該ライブラリを呼び出すJNI処理に何らかの問題がないか確認します。

スタックトレースの調査方法は、“9.5.1 スタックトレース”を参照してください。

スタックオーバーフローの検出

「異常終了箇所に関する情報」の1.の異常終了時に発生した例外に関する情報に、以下のシグナルコードの表記がある場合、例外が発生したスレッドでスタックオーバーフローが発生した(a)、c)の表記)、または発生した可能性がある(b)、d)の表記)

ことを示しています。
この場合、例外が発生したスレッドに対するスタックのサイズを大きくすることで問題が解決する可能性があります。

スタックオーバーフロー発生の原因が、Java APIで生成されたスレッドに対するスタックのサイズにある場合は、“9.4.2 スタックのチューニング”を参照して、Java APIで生成されるスレッドに対するスタックのサイズをチューニングしてください。

スタックオーバーフローを示すシグナルコード

Windows32/64

a.「EXCEPTION_STACK_OVERFLOW」

b.「EXCEPTION_ACCESS_VIOLATION (Stack Overflow ?)」

Solaris64 Linux32/64

a.「SIGSEGV (Stack Overflow)」

b.「SIGSEGV (Stack Overflow ?)」

注意

Windowsエラー報告(Windows Error Reporting(WER))の分析 Windows32/64

スタックオーバーフローが原因で発生した異常終了の場合、OS側からFJVM側の処理へ制御が渡らず、そのままWindowsエラー報告へ制御が渡されることがあります。この場合は、FJVMログが出力されないため、Windowsエラー報告を確認してください。
Windowsエラー報告に以下の例外番号が出力されている場合には、スタックオーバーフローが原因と考えられます。
なお、Windowsエラー報告の説明は、“9.5.9.1 クラッシュダンプ”を参照してください。

スタックオーバーフローを示す例外番号

c00000fd (スタックオーバーフロー)

9.5.8.2 異常終了時のシグナルハンドラ情報 Solaris64 Linux32/64

Java VMの実行制御で必要となる“表9.9 Java VMの制御で必要となるシグナル”の各シグナルに対するシグナルハンドラ情報が確認できます。

表9.9 Java VMの制御で必要となるシグナル

Solaris版Java VM	Linux版Java VM
SIGSEGV	SIGSEGV
SIGPIPE	SIGPIPE
SIGBUS	SIGBUS
SIGILL	SIGILL
SIGFPE	SIGFPE
INTERRUPT_SIGNAL (デフォルトはSIGUSR1)	INTERRUPT_SIGNAL (デフォルトはSIGUSR1) (注2)
ASYNC_SIGNAL (デフォルトはSIGUSR2)	SR_SIGNUM (デフォルトはSIGUSR2)
SIGQUIT (注1)	SIGQUIT (注1)
SIGINT (注1)	SIGINT (注1)
SIGHUP (注1)	SIGHUP (注1)
SIGTERM (注1)	SIGTERM (注1)
SIGXFSZ (注3)	SIGXFSZ (注3)

注1)

-Xrsオプションで操作対象となるシグナルです。

注2)

Java VMでリザーブしているシグナルです。
注1)および注2)のシグナルに関するシグナルハンドラ情報は出力しません。

注3)

Java VMで使用されているシグナルです。

シグナルハンドラ情報として、以下の情報が出力されています。

- 登録されているシグナルハンドラのアドレス
- 登録されているシグナルハンドラがJava VMで登録したシグナルハンドラかどうかの正否情報(Java VM以外の処理で登録されたシグナルハンドラの場合には、該当するシグナルハンドラ情報の行に“(not in VM)”が出力されます)

“表9.9 Java VMの制御で必要となるシグナル”のシグナルハンドラがJava VM以外の処理で登録されていた場合、Java VMは正常に動作しません。この場合、当該シグナルハンドラを登録しないようにアプリケーションを修正してください。

9.5.8.3 異常終了時のJavaヒープに関する情報

異常終了時のJavaヒープの使用状況が確認できます。

Javaヒープのサイズによる異常終了の場合、異常終了時にどのJavaヒープの枯渇により異常終了が発生したかが確認できます。

9.5.8.4 出力例と調査例

出力例を元に説明します。

```
--------------------------------------------------------------------------------
#### Java VM: Java HotSpot(TM) 64-Bit Server VM (**.*.**_FUJITSU_MODIFIED-B**[*****] mixed mode)
>>>> Logging process start. [pid=879] Time=Tue Mar  7 19:54:10 2017
```

(1) 異常終了箇所の情報

異常終了箇所に関する情報が確認できます。
libjvm.soのsysThreadAvailableStackWithSlack関数の近くでSIGSEGV(メモリアクセスで不正なセグメントを参照)が発生しています。
本例の場合、Java VM内で異常が発生していると判断します。
異常終了箇所がJavaアプリケーション内でないため、異常発生時のスタックトレース情報を調査します。
本例の場合、com.appli.ap.business.AL02ABB00000.toStringの延長で不正なアクセスが発生しているので、そこからAL02ABB00000.javaの489行目で不正なアクセスを招きそうな箇所がないか調べます。

```
Unexpected Signal : SIGSEGV [0xb] occurred at PC= ffffffff65c0067c, pid= 879, nid=1
Function name=sysThreadAvailableStackWithSlack
Library=/opt/FJSVawjbk/jdk7/jre/lib/sparcv9/fjvm/libjvm.so

Current Java thread:
0xffffffff7b8e2850 - 0xffffffff7b8e4b7c at com.appli.ap.business.AL02ABB00000.toString(AL02ABB00000.java:489)
0xffffffff7b8e2850 - 0xffffffff7b8e4b7c at com.appli.ap.business.AL02ABB00000.toString(AL02ABB00000.java:520)
                                          at java.lang.String.valueOf(String.java:1942)
                                          at java.lang.StringBuffer.append(StringBuffer.java:365)
- locked <0xffffffff72fdea48> (a java.lang.StringBuffer)
                                         at com.appli.ap.business.AL02ABB25201.doExecute(AL02ABB25201.java:774)
                                          at com.appli.ap.formula.AFCC6842.doDelegate(AFCC6842.java:221)
                                         at
com.appli.ap.formula.ejb.session.AFSF6801.doExecuteOrdinarily(AFSF6801.java:381)
                                          at
com.appli.ap.formula.ejb.session.FJAFSF6801_AFSF6801RemoteImpl.doExecuteOrdinarily(FJAFSF6801_AFSF6801Remote
Impl.java:464)
- locked <0xffffffff72fdef70> (a com.appli.ap.formula.ejb.session.FJAFSF6801_AFSF6801RemoteImpl)
                                          at
com.appli.ap.formula.ejb.session._FJAFSF6801_AFSF6801RemoteImpl_Tie._invoke(_FJAFSF6801_AFSF6801RemoteImpl_T
ie.java:76)
```

```
                                        at
com.fujitsu.ObjectDirector.CORBA.ServerRequest.call_invoke(ServerRequest.java:961)
                                      at com.fujitsu.ObjectDirector.PortableServer.POA.MsgRecv(POA.java:2578)
                                        at
com.fujitsu.ObjectDirector.PortableServer.POAManager.MsgRecv(POAManager.java:1061)
                                    at com.fujitsu.ObjectDirector.PortableServer.POAnc.MsgRecv(POAnc.java:163)

Dynamic libraries:
0x0000000100000000 /opt/FJSVawjbk/jdk7/bin/java
0xffffffff7f200000 /usr/lib/libthread.so.1
0xffffffff7ed00000 /usr/lib/libdl.so.1
～～～～～～～～
略
～～～～～～～～
0xffffffff7be00000 /lib/64/libuutil.so.1
0xffffffff7bc00000 /lib/64/libgen.so.1

Local Time = Wed May 30 19:54:10 2017
Elapsed Time = 22
```

注意:

Error IDの行が出力されている場合、Error IDとして出力されている値は、Java VMが内部処理矛盾を自己検出した場合に出力する内部情報コードです。

```
#
# HotSpot Virtual Machine Error : SIGSEGV (0xb)
#  [ pc= ffffffff65c0067c, pid=879(0x36f), nid=1(0x0000000000000001), tid=0x000000010010e000 ]
#
# Please report this error to FUJITSU
#
# JRE version: Java(TM) SE Runtime Environment (7.0_***-b**) (build 1.7.0_***-b**_Fujitsu_**-**-**_**:**)

# Java VM: Java HotSpot(TM) 64-Bit Server VM (**.*.**_FUJITSU_MODIFIED-B** mixed mode solaris-sparc compressed
oops)


～～～～～～～～
略
～～～～～～～～
```

(2)異常終了時のシグナルハンドラ情報 Solaris64 Linux32/64

異常終了時のシグナルハンドラに関する情報が確認できます。
本例では、すべて「(in VM)」表示なので、シグナルハンドラの登録変更に関する問題はありません。

```
##>> Signal Handlers
 VM signal handler[1]=0xffffffff7d68eef8, VM signal handler[2]=0xffffffff7dff7fdc,
SIG_DFL=0x0000000000000000,

SIG_IGN=0x0000000000000001, INT_SIG=(16,39), ASYNC_SIG=(17,40)
  SIGSEGV       :signal handler=0xffffffff7dff7fdc (in VM *)
  SIGPIPE       :signal handler=0xffffffff7d68eef8 (in VM)
  SIGBUS        :signal handler=0xffffffff7dff7fdc (in VM *)
  SIGILL        :signal handler=0xffffffff7d68eef8 (in VM)
  SIGFPE        :signal handler=0xffffffff7d68eef8 (in VM)
  SIGXFSZ       :signal handler=0xffffffff7d68eef8 (in VM)
  ALT_INTERRUPT_SIGNAL  :signal handler=0xffffffff7de603d8 (in VM +)
  ALT_ASYNC_SIGNAL      :signal handler=0xffffffff7d68eef8 (in VM)
```

(3)異常終了時のJavaヒープ領域に関する情報

異常終了時のJavaヒープ領域に関する情報が確認できます。

パラレルGC使用時:
「PSYoungGen」が「New世代領域」、
「PSOldGen」が「Old世代領域」、
「PSPermGen」が「Permanent世代領域」
に関する情報です。

CMS付きパラレルGC使用時:
「par new generation」が「New世代領域」、
「concurrent mark-sweep generation」が「Old世代領域」、
「concurrent-mark-sweep perm gen」が「Permanent世代領域」
に関する情報です。

なお「Old世代領域」および「Permanent世代領域」における「object space」についての情報は出力されません。

シリアルGC使用時:
「def new generation」が「New世代領域」、
「tenured generation」が「Old世代領域」、
「compacting perm gen」が「Permanent世代領域」
に関する情報です。

本例の場合、異常終了時点における「New世代領域」＋「Old世代領域」の領域(-Xmxで最大量が指定される領域)には、空きがあることがわかります。
また「Permanent世代領域」に対しても、余裕があることがわかります。

注意:
パーセントで示されている値は、異常終了した時点でFJVMがJavaヒープ用に利用可能な状態にしている(コミットしている)メモリ量に対する比率です。利用可能な上限値に対する比率ではありません。
パーセントで示されている値は参照せず、K(キロ)単位で表示されているメモリ使用量の値と、オプションで指定された値(デフォルト値を含む)とを比較して判断してください。

注意:
パラレルGCを使用していた場合、以下の「-Xms=」「-Xmx=」に続いて表示される値は、-Xmsオプション/-Xmxオプションで指定された値およびページサイズなどのシステム情報を元に、Java VMが最適となるよう初期値/最大値を計算し直し、実際にJava VMが使用した値が出力されます。そのため、指定された値と異なる場合があります。

```
##>> Heap
 PSYoungGen      total 3584K, used 184K [0xffffffff72c00000, 0xffffffff73000000, 0xffffffff75800000)
  eden space 3072K, 6% used [0xffffffff72c00000,0xffffffff72c2e170,0xffffffff72f00000)
  from space 512K, 0% used [0xffffffff72f00000,0xffffffff72f00000,0xffffffff72f80000)
  to   space 512K, 0% used [0xffffffff72f80000,0xffffffff72f80000,0xffffffff73000000)
 PSOldGen        total 8192K, used 593K [0xffffffff6d800000, 0xffffffff6e000000, 0xffffffff72c00000)
  object space 8192K, 7% used [0xffffffff6d800000,0xffffffff6d894478,0xffffffff6e000000)
 PSPermGen       total 24576K, used 3098K [0xffffffff68400000, 0xffffffff69c00000, 0xffffffff6d800000)
  object space 24576K, 12% used [0xffffffff68400000,0xffffffff68706ae8,0xffffffff69c00000)
(-Xms=12288K, -Xmx=131072K, -XX:PermSize=21248K, -XX:MaxPermSize=86016K)
```

9.5.9 クラッシュダンプ・コアダンプ

Javaアプリケーションが異常終了(プロセスが消滅)したときに、各OS上に用意されたクラッシュダンプやコアダンプを採取することにより、異常終了の原因を調査することができる場合があります。

9.5.9.1 クラッシュダンプ Windows32/64

Windows(R)上で異常を調査する場合に採取する、クラッシュダンプの採取方法を説明します。

クラッシュダンプの採取には、Windowsエラー報告（Windows Error Reporting（WER））の機能を使用します。

次の例を参考にして、WERを設定してください。

例

WERの設定例

1. MS-DOSコマンドプロンプトなどで“regedit”コマンドを投入し、レジストリエディタを起動します。
2. 「HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows¥Windows Error Reporting¥LocalDumps」キーを作成します。
3. LocalDumpsキーにREG_DWORD型でDumpTypeという値を作成し、「2」を設定します。

WERに関する設定の方法については、以下の情報も参照ください。

- 富士通製サーバ製品の場合：

  http://primeserver.fujitsu.com/primergy/soft/ssupportguide/

- 上記製品以外の場合：

  http://msdn.microsoft.com/en-us/library/bb787181.aspx

9.5.9.2 コアダンプ（Solaris） Solaris64

Solaris上でのコアダンプ採取のための注意事項を説明します。

コアダンプが出力されない場合の確認

コアダンプが出力されない場合の原因として、システムリソース等の問題がまず考えられます。カレントディレクトリの書込み権、ディスク容量、limit(1)コマンド結果を確認してください。

9.5.9.3 コアダンプ（Linux） Linux32/64

Linux上でのコアダンプ採取のための注意事項を説明します。

コアダンプが出力されない場合の確認

- コアダンプが出力されない場合の原因として、システムリソース等の問題がまず考えられます。カレントディレクトリの書込み権、ディスク容量、limit(1)コマンド結果を確認してください。
- ハード／オペレーティングシステムの出荷時、またはオペレーティングシステムのUpdate適用により、デフォルトではコアダンプの出力が設定されていない場合があります。以下を参照して、コアダンプが出力されるように設定してください。

コアダンプ出力の設定方法

- コマンドでInterstageを起動させる場合

  sh(bash)で"ulimit -c unlimited"コマンド実行後、Interstageを起動させます。クラスタ起動ユーザがInterstage起動ユーザと違う場合は、クラスタ起動前に"ulimit -c unlimited"コマンドを実行してから、クラスタを起動させます。

- オペレーティングシステム起動時の自動起動でInterstageを起動する場合(RHEL6)

  以下のファイルの記述を変更することにより、オペレーティングシステムの再起動後にcoreが出力されるようになります。

  /etc/init.d/functions
  「ulimit -S -c unlimited >/dev/null 2>&1」に変更します。

  【修正前】

```
# make sure it doesn't core dump anywhere; while this could mask
# problems with the daemon, it also closes some security problems

ulimit -S -c 0 >/dev/null 2>&1
  または、
ulimit -S -c ${DAEMON_COREFILE_LIMIT:-0} >/dev/null 2>1
```

【修正後】

```
# make sure it doesn't core dump anywhere; while this could mask
# problems with the daemon, it also closes some security problems

ulimit -S -c unlimited >/dev/null 2>&1
```

/etc/rc2.d/S99startis

「ulimit -c unlimited」を追加します。

【修正前】

```
#!/bin/sh
# Interstage Application Server
# S99starttis : Interstage Application Server start procedure

  OD_HOME=/opt/FJSVod
  export OD_HOME

  /opt/FJSVod/bin/odalive > /dev/null
  while [ "$?" != "0" ]
  do
      sleep 1
      /opt/FJSVod/bin/odalive > /dev/null
  done

  /opt/FJSVtd/bin/isstart
```

【修正後】

```
#!/bin/sh
# Interstage Application Server
# S99starttis : Interstage Application Server start procedure

  OD_HOME=/opt/FJSVod
  export OD_HOME

  ulimit -c unlimited

  /opt/FJSVod/bin/odalive > /dev/null
  while [ "$?" != "0" ]
  do
      sleep 1
      /opt/FJSVod/bin/odalive > /dev/null
  done

  /opt/FJSVtd/bin/isstart
```

- オペレーティングシステム起動時の自動起動でInterstageを起動する場合(RHEL7)

  unitファイルに以下の設定を追加してください。unitファイルでの定義方法については、"付録I RHEL7のunitファイルでの環境定義"を参照してください。

記載するセクション	設定項目	設定値
[Service]	LimitCORE	infinity

9.5.10 JNI処理異常時のメッセージ出力

Java以外の言語と連携する場合、Java Native Interface(JNI)を使用します。
しかし、JNIの使用方法を誤ると、Javaプロセスの終了(異常終了)などの原因となります。

このとき、以下のオプションを指定することにより、JNI処理で異常が発生した場合にメッセージが出力されますので、JNIのパラメーターなどの確認に活用してください。

JNI処理異常時にメッセージを出力するオプション

```
-Xcheck:jni
```

“-Xcheck:jni”パラメーターを指定したときに、以下のメッセージが出力されることがあります。

JNI処理異常時に出力されるメッセージ

```
FATAL ERROR in native method: (詳細メッセージ)
```

(詳細メッセージ)

以降で説明する文字列が出力されます。

(詳細メッセージ)が出力される例と注意事項を説明します。
以降の説明を参考にして、JNIの処理部分を見直してください。

メッセージの説明

JNI received a class argument that is not a class

[異常例] AllocObject関数の第2引数にJNIハンドル経由で受け取ったjclass型ではない型(buf)を指定した場合:

```
(*env)->AllocObject(env, (jclass)buf);
```

JNI string operation received a non-string

[異常例] GetStringUTFChars関数の第2引数にNULLを指定した場合:

```
(*env)->GetStringUTFChars(env, NULL, 0);
```

Non-array passed to JNI array operations

[異常例] GetArrayLength関数の第2引数にjarray型ではない型を指定した場合:

```
(*env)->GetArrayLength(env, (jarray)(*env)->NewStringUTF(env, "abc"));
```

Static field ID passed to JNI

[異常例] GetIntField関数の第3引数にstaticフィールドを指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
jjfieldID fid = (*env)->GetFieldID(env, cls, "static_data", "I");
(*env)->GetIntField(env, obj, fid);
```

"static_data"は、JNIハンドル経由で受け取ったobjオブジェクトのstaticフィールド名

Null object passed to JNI

[異常例] GetIntField関数の第2引数にNULLを指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
jfieldID fid = (*env)->GetFieldID(env, cls, "instance_data", "I");
(*env)->GetIntField(env, NULL, fid);
```

"instance_data" は、JNIハンドル経由で受け取ったobjオブジェクトのinstanceフィールド名

注意

instance変数かどうかのチェック時だけに出力されるメッセージです。

以下のように、GetObjectClass関数の第2引数にNULLを指定した場合、“-Xcheck:jni”オプションによるメッセージは出力されません。

```
(*env)->GetObjectClass(env, NULL);
```

Wrong field ID passed to JNI

[異常例] GetIntFieldの第3引数に数値を指定した場合:

```
(*env)->GetIntField(env, obj, -1);
```

注意

instance変数かどうかのチェック時だけに出力されるメッセージです。

Non-static field ID passed to JNI

[異常例] GetStaticIntField関数の第3引数に数値を指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
(*env)->GetStaticIntField(env, cls, -1);
```

注意

以下のように、GetStaticFieldID関数の第2引数にinstanceフィールド名を指定した場合、“-Xcheck:jni”オプションによるメッセージは出力されません。

```
jclass cls = (*env)->GetObjectClass(env, obj);
jfieldID fid = (*env)->GetStaticFieldID(env, cls, "instance_data", "I");
(*env)->GetStaticIntField(env, cls, fid);
```

"instance_data" は、JNIハンドル経由で受け取ったobjオブジェクトのinstanceフィールド名

Array element type mismatch in JNI

[異常例] GetFloatArrayElements関数の第2引数にjintArray型を指定した場合:

```
jintArray intarray = (*env)->NewIntArray(env, 2);

(*env)->GetFloatArrayElements(env, intarray, 0)
```

Object array expected but not received for JNI array operation

[異常例] GetIntArrayElements関数の第2引数にjobjectArray型を指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
jobjectArray objarray = (*env)->NewObjectArray(env, 1, cls, obj);
(*env)->GetIntArrayElements(env, objarray, 0);
```

Field type (static) mismatch in JNI get/set field operations

[異常例] GetStaticFloatField関数の第3引数にint型のjfieldIDを指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
jfieldID fid = (*env)->GetStaticFieldID(env, cls, "static_data", "I");
(*env)->GetStaticFloatField(env, cls, fid);
```

"static_data"は、JNIハンドル経由で受け取ったobjオブジェクトのstaticフィールド名

Field type (instance) mismatch in JNI get/set field operations

[異常例] GetFloatField関数の第3引数にint型のjfieldIDを指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
jfieldID fid = (*env)->GetFieldID(env, cls, "instance_data", "I");
(*env)->GetFloatField(env, obj, fid);
```

"instance_data" は、JNIハンドル経由で受け取ったobjオブジェクトのinstanceフィールド名

Wrong static field ID passed to JNI

[異常例] GetStaticObjectField関数の第2引数に不正なjclassを指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
jclass cls2 = (*env)->GetObjectClass(env, (*env)->NewStringUTF(env, "abc"));
jfieldID fid = (*env)->GetStaticFieldID(env, cls, "static_data", "I");
(*env)->GetStaticObjectField(env, cls2, fid);
```

"static_data"は、JNIハンドル経由で受け取ったobjオブジェクトのstaticフィールド名

Using JNIEnv in the wrong thread

[解説]

実行しているスレッドのためのものではないJNIEnvを使用したためのエラーです。
Java VMは、JNIインタフェースポインタ(JNIEnv)が参照する領域を、スレッド固有のデータ領域に割り当てることがあります。このため、JNIインタフェースポインタは、カレントスレッドに対してだけ有効です。ネイティブメソッドは、JNIインタフェースポインタを別のスレッドに渡すといった使い方はできません。

JNI call made with exception pending

[解説]

ネイティブプログラムで何らかの例外が発生後、その例外を処理せずに引き続きJNI関数を実行したためのエラーです。
ネイティブプログラムでJNI関数を呼び出した後は、その都度ExceptionOccurredを使用して例外の発生状況をチェックし、必要に応じて、例外のクリア、または、例外を上位メソッドへスローしてください。

9.6 異常発生時の原因振り分け

異常が発生したときに、原因を振り分ける方法を説明します。

9.6.1 java.lang.OutOfMemoryErrorがスローされた場合

OutOfMemoryErrorがスローされた場合、考えられる原因とその対処方法を説明します。

なお、OutOfMemoryErrorがスローされた場合に出力されるメッセージ情報については、“9.6.1.1 メモリ領域不足事象発生時のメッセージ出力機能の強化”も参照してください。

想定される原因(メモリリーク)

VMがガーベジコレクションを繰り返しても、時間の経緯とともにメモリ消費量が増大していく場合、プログラム中メモリリークを起こしている可能性があります。

メモリリークの結果、Javaのヒープ不足が発生し**OutOfMemoryError**がスローされる場合があります。

この場合、ガーベジコレクションのログを採取して、Javaヒープの消費状況を確認してください。ガーベジコレクションのログを採取する方法は、“9.2.6 ガーベジコレクションのログ出力”を参照してください。

想定される原因(Javaヒープ不足)

通常、**OutOfMemoryError**は、Javaヒープ不足が原因でスローされます。

ガーベジコレクションのログを採取して、Javaヒープの消費状況を確認してください。
Javaヒープの空き容量がないことが確認されたら、Javaヒープをチューニングしてください。

ガーベジコレクションのログを採取する方法は、"9.2.6 ガーベジコレクションのログ出力"を参照してください。
Javaヒープのチューニング方法は、"9.4.1 Javaヒープのチューニング"を参照してください。

想定される原因(ユーザ空間不足)

多量のスレッドを生成して、多量のスタックがユーザ空間内に割り当てられ、ユーザ空間不足になった場合、次の**OutOfMemoryError**がスローされる、あるいはエラーメッセージとして表示を行いプロセスが終了します。

```
java.lang.OutOfMemoryError: unable to create new native thread
```

また、JavaヒープやOSの仮想メモリに余裕があるにもかかわらず、ユーザ空間内にメモリを確保できなかった場合、次の**OutOfMemoryError**が出力されプログラムが終了します。

```
java.lang.OutOfMemoryError: requested サイズ bytes 制御名. Out of swap space?
```

サイズ:

確保できなかったメモリの大きさ

制御名:

メモリが確保できなかったJava VMの制御名(該当情報がある場合にだけ表示)

ユーザ空間が不足している場合は、Javaヒープまたはスタックのサイズを小さくするなどのチューニングを行ってください。
スタックのサイズをチューニングする方法は、"9.4.2 スタックのチューニング"を参照してください。
Javaヒープのチューニング方法は、"9.4.1 Javaヒープのチューニング"を参照してください。

なお、仮想メモリに余裕がある場合は、Javaプロセスを複数起動して、プロセス多重度を上げる方法もあります。Java EE 6アプリケーションの場合、Java EE 6のチューニングを行ってください。Java EE 6のチューニング方法の詳細は、Java EE運用ガイド(Java EE 6編)を参照してください。

想定される原因(仮想メモリ不足)

仮想メモリが不足してスレッドが生成できない場合、次の**OutOfMemoryError**がスローされる、あるいはエラーメッセージとして表示を行いプロセスが終了します。

```
java.lang.OutOfMemoryError: unable to create new native thread
```

また、OSの仮想メモリが不足した場合、次の**OutOfMemoryError**が出力されプログラムが終了します。

```
java.lang.OutOfMemoryError: requested サイズ bytes 制御名. Out of swap space?
```

サイズ:

確保できなかったメモリの大きさ

制御名:

メモリが確保できなかったJava VMの制御名(該当情報がある場合にだけ表示)

仮想メモリが不足した場合は、他の不要なプロセスを終了して仮想メモリに余裕を持たせるか、物理メモリ(RAM)またはスワップファイルを拡張して仮想メモリを増やすようにチューニングを行ってください。

想定される原因(ガーベジコレクション処理の実行抑止)

ガーベジコレクション処理(GC処理)の実行抑止により、クリティカルセクション状態時に**OutOfMemoryError**がスローされた場合は、必要に応じて、GC処理の実行抑止による影響ができるだけ小さくなるように、実行するアプリケーションの処理内容を見直してください(アプリケーション処理内の、GC処理の実行抑止に関係する機能の利用見直しを行ってください)。

GC処理の実行抑止については、“9.2.1 FJVMでサポートされるガーベジコレクション処理”を参照してください。
なお、Javaヒープのチューニングにより、GC処理の実行抑止による**OutOfMemoryError**の発生が緩和できる場合があります。

- クリティカルセクション状態時に、ネイティブプログラムからJNIを利用してJavaのオブジェクトの生成要求を行っていないアプリケーションの場合は、Old世代領域を大きくする(メモリ割り当てプール全体を大きくする)チューニングで、GC処理の実行抑止による**OutOfMemoryError**の発生が緩和できる場合があります。
- クリティカルセクション状態時に、ネイティブプログラムからJNIを利用してJavaのオブジェクトの生成要求を行っているアプリケーションの場合は、New世代領域を大きくするチューニングで、GC処理の実行抑止による**OutOfMemoryError**の発生が緩和できる場合があります。

ガーベジコレクションのログを採取する方法は、“9.2.6 ガーベジコレクションのログ出力”を参照してください。
Javaヒープのチューニング方法は、“9.4.1 Javaヒープのチューニング”を参照してください。

注意

GC処理の実行抑止により、クリティカルセクション状態で**OutOfMemoryError**がスローされたかどうかは、“メモリ領域不足事象発生時のメッセージ出力機能の強化”で出力されるメッセージを参照して判断してください。
また、PCMI1105メッセージが出力されている場合は、IJServerクラスタのJava VMログ(console.log)に出力される「Java VMのヒープ域不足の詳細情報」を参照して判断してください。

9.6.1.1 メモリ領域不足事象発生時のメッセージ出力機能の強化

FJVMでは、メモリ領域不足事象発生時に出力されるメッセージ情報の強化を行っています。
これにより**FJVM**では、メモリ領域不足事象が発生した場合に、java.lang.OutOfMemoryErrorの例外メッセージ情報に加え、不足した領域の種別情報を以下の形式で出力します。

メモリ領域不足事象が発生した場合に出力される不足した領域の種別情報

```
The memory was exhausted area_name
  Java heap size / max Java heap size = heap_size / max_heap_size
  Java perm size / max Java perm size = perm_size / max_perm_size
```

area_name:

メモリ領域不足事象が発生した領域の名前や領域不足となったオブジェクトの要求サイズ(不足領域情報)を表示します。
不足領域情報としては以下の項目があります。

- on Java heap space. : requested <NNNN> bytes

  NNNNバイトのオブジェクト生成要求において、メモリ割り当てプール(New世代領域またはOld世代領域)に対してメモリ領域不足事象が発生した場合です。
  なお、エルゴノミクス機能によるメモリ領域不足事象の検出機能が有効な場合で、かつ当該機能によりメモリ領域不足事象を検出した場合、この項目になる場合があります。

- on Java heap space. : requested <NNNN> bytes (in critical section)

  意味は「on Java heap space. : requested <NNNN> bytes」と同じですが、メモリ領域不足事象発生時、クリティカルセクション状態でGC処理の実行が抑止されていたことを示しています。

- on Java perm space. : requested <NNNN> bytes

  NNNNバイトのオブジェクト生成要求において、Permanent世代領域に対してメモリ領域不足事象が発生した場合です。
  なお、エルゴノミクス機能によるメモリ領域不足事象の検出機能が有効な場合で、かつ当該機能によりメモリ領域不足事象を検出した場合、この項目になる場合があります。

- on Java perm space. : requested <NNNN> bytes (in critical section)

  意味は「on Java perm space. : requested <NNNN> bytes」と同じですが、メモリ領域不足事象発生時、クリティカルセクション状態でGC処理の実行が抑止されていたことを示しています。

ー (なし)

スタックやヒープなど、Javaヒープ以外の領域に対してメモリ領域不足事象が発生した場合です。特にjava.lang.OutOfMemoryErrorの例外メッセージ情報が“java.lang.OutOfMemoryErrorがスローされた場合”の「ユーザ空間不足」または「仮想メモリ不足」の場合に出力される形式の場合は、スタックやヒープなど、Javaヒープ以外の領域に対してメモリ領域不足事象が発生した場合と断定できます。
またはJavaアプリケーション実行時における配列生成式の評価の段階で、配列オブジェクトの長さ(配列要素の数)から、当該配列オブジェクトを割り当てるための領域が十分でないと評価された場合(配列の長さ(配列要素の数)が大きすぎ、配列オブジェクトとしての大きさが2ギガバイト程度もしくはそれ以上の大きさになる配列の定義がある場合)。
またはクラスのロード処理でメモリ不足が発生した場合。

なお、エルゴノミクス機能によるメモリ領域不足事象の検出機能が有効な場合で、かつ当該機能によりメモリ領域不足事象を検出した場合、この項目になる場合があります。

heap_size:

メモリ領域不足事象が発生した際に使用中となっているメモリ割り当てプールのサイズ(単位:byte)。

max_heap_size:

利用可能なメモリ割り当てプールの最大サイズ(単位:byte)。

perm_size:

メモリ領域不足事象が発生した際に使用中となっているPermanent世代領域のサイズ(単位:byte)。

max_perm_size:

利用可能なPermanent世代領域の最大サイズ(単位:byte)。

注意

メモリ領域不足事象が発生した際に出力される各領域の使用中サイズ(heap_size、perm_size)には、メモリ領域不足の原因となったオブジェクトの大きさは含まれません。

そのため巨大サイズのオブジェクト生成要求などによりメモリ領域不足事象が発生した場合には、「最大サイズ」と「使用中サイズ」の差が大きい場合(空き領域がたくさんあるように見える場合)がありますので注意してください。

注意

NewGC処理では、New世代領域を「eden space」、「from space」および「to space」の3つの内部領域に細分割し、当該領域上において、一般に世代別GC制御と言われる制御方法を用いて、Javaアプリケーションが生成要求したオブジェクトを管理・制御しています。

このうち、「from space」および「to space」は、Java VMがNewGC処理を行う際の作業域的な役割を持つ領域となっています。そのため、「from space」および「to space」の各領域が占める大きさのうち、Javaアプリケーションからのオブジェクト生成要求のために使用される大きさは、その領域の一部分だけとなります。

そのため、出力データにおいて、メモリ割り当てプールやNew世代領域に空きがあるように見える場合であっても、実際には空きがない場合がありますので注意してください(空きがあるように見える場合であっても、その差は、NewGC処理用の作業域的な役割で使用済となっている場合があります)。

注意

メモリ割り当てプールに対してメモリ領域不足事象が発生した場合に出力されるheap_sizeの値は、New世代領域での使用中サイズとOld世代領域での使用中サイズの合計値です。
New世代領域とOld世代領域は別々のオブジェクト格納域として管理・制御されますから、max_heap_sizeとheap_sizeの差の大きさが、そのまま生成要求できるオブジェクトの最大サイズにはなりませんので注意してください。

例

メモリ領域不足事象が発生した場合に出力されるメッセージ出力例

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
The memory was exhausted on Java heap space. : requested 4016 bytes
Java heap size / max Java heap size = 495974032 / 536870912
Java perm size / max Java perm size = 1678376 / 67108864
```

4016バイトのオブジェクト生成要求において、メモリ割り当てプールに対してメモリ領域不足が発生したことを確認することができます。

9.6.2 PCMI1105メッセージが出力された場合

Interstage Application Server配下のJavaアプリケーション実行時において、以下のメッセージが出力された場合は、メモリ領域不足事象が原因による異常となります。
Javaヒープをチューニングしてください。

Javaヒープのチューニング方法は、“9.4.1 Javaヒープのチューニング”を参照してください。

- PCMI1105メッセージ

参照

各メッセージの内容については、“メッセージ集”を参照してください。

なお、IJServerクラスタのJava VMログ(console.log)に出力される「Java VMのヒープ域不足の詳細情報」の出力形式は、以下のとおりです。

Java VMのヒープ域不足の詳細情報の出力形式

```
---------------------------------------------------------------
OutOfMemory Log
---------------------------------------------------------------
pid=process_id
heap_type=heap_type_code
heap_size=heap_size
max_heap_size=max_heap_size
perm_size=perm_size
max_perm_size=max_perm_size
requested_size=requested_size
---------------------------------------------------------------
VM is terminated by occurred OutOfMemoryError on heap_type.
stack_trace
```

process_id:

メモリ領域不足事象が発生したJavaプロセスのプロセス番号。

heap_type_code:

メモリ領域不足事象が発生した領域に対応する番号です。表示される番号については、heap_typeの説明を参照してください。

heap_size:

メモリ領域不足事象が発生した際に使用中となっているメモリ割り当てプールのサイズ(単位:byte)。

max_heap_size:

利用可能なメモリ割り当てプールの最大サイズ(単位:byte)。

perm_size:

メモリ領域不足事象が発生した際に使用中となっているPermanent世代領域のサイズ(単位:byte)。

max_perm_size:

利用可能なPermanent世代領域の最大サイズ(単位:byte)。

requested_size:

領域不足となったオブジェクトの要求サイズ(単位:byte)。

heap_type:

不足領域情報(メモリ領域不足事象が発生した領域の名前など)を表示します。
不足領域情報としては以下の項目があります。

- Java heap

  requested_sizeバイトのオブジェクト生成要求において、メモリ割り当てプール(New世代領域またはOld世代領域)に対してメモリ領域不足事象が発生した場合です。
  この項目のときのheap_type_codeは「1」になります。
  なお、エルゴノミクス機能によるメモリ領域不足事象の検出機能が有効な場合で、かつ当該機能によりメモリ領域不足事象を検出した場合、この項目になる場合があります。

- Java heap in critical section

  意味は「Java heap」と同じですが、メモリ領域不足事象発生時、クリティカルセクション状態でGC処理の実行が抑止されていたことを示しています。

- Java Perm

  requested_sizeバイトのオブジェクト生成要求において、Permanent世代領域に対してメモリ領域不足事象が発生した場合です。
  この項目のときのheap_type_codeは「2」になります。
  なお、エルゴノミクス機能によるメモリ領域不足事象の検出機能が有効な場合で、かつ当該機能によりメモリ領域不足事象を検出した場合、この項目になる場合があります。

- Java Perm space in critical section

  意味は「Java Perm」と同じですが、メモリ領域不足事象発生時、クリティカルセクション状態でGC処理の実行が抑止されていたことを示しています。

- C heap 制御情報

  スタックやヒープなど、Javaヒープ以外の領域に対してメモリ領域不足事象が発生した場合(requested_sizeバイトのメモリ割り当て要求が失敗した場合)です。
  この項目のときのheap_type_codeは「0」になります。
  なお、現象発生時の要求サイズが不明な場合のrequested_sizeは「0」になります。
  また、次行に制御情報(メモリが確保できなかったJava VM制御の情報)が出力される場合があります。

- unknown space

  Javaアプリケーション実行時における配列生成式の評価の段階で、配列オブジェクトの長さ(配列要素の数)から、当該配列オブジェクトを割り当てるための領域が十分でないと評価された場合(配列の長さ(配列要素の数)が大きすぎ、配列オブジェクトとしての大きさが2ギガバイト程度もしくはそれ以上の大きさになる配列の定義がある場合)。
  またはクラスのロード処理でメモリ不足が発生した場合。
  なお、エルゴノミクス機能によるメモリ領域不足事象の検出機能が有効な場合で、かつ当該機能によりメモリ領域不足事象を検出した場合、この項目になる場合があります。
  この項目のときのheap_type_codeは「-1」になります。
  なお、この項目の場合のrequested_sizeは「0」になります。

stack_trace:

メモリ領域不足事象が発生したスレッドがJavaアプリケーションを実行しているスレッドだった場合は、当該スレッドのスタックトレースが出力されます。それ以外のスレッドの場合、またはリソース不足によりスタック情報が取り出せない場合は、スタックトレースは出力されません。

注意

メモリ領域不足事象が発生した際に出力される各領域の使用中サイズ(heap_size、perm_size)には、メモリ領域不足の原因となったオブジェクトの大きさは含まれません。

そのため巨大サイズのオブジェクト生成要求などによりメモリ領域不足事象が発生した場合には、「最大サイズ」と「使用中サイズ」の差が大きい場合（空き領域がたくさんあるように見える場合）がありますので注意してください。

注意

NewGC処理では、New世代領域を「eden space」、「from space」および「to space」の3つの内部領域に細分割し、当該領域上において、一般に世代別GC制御と言われる制御方法を用いて、Javaアプリケーションが生成要求したオブジェクトを管理・制御しています。

このうち、「from space」および「to space」は、Java VMがNewGC処理を行う際の作業域的な役割を持つ領域となっています。そのため、「from space」および「to space」の各領域が占める大きさのうち、Javaアプリケーションからのオブジェクト生成要求のために使用される大きさは、その領域の一部分だけとなります。

そのため、出力データにおいて、メモリ割り当てプールやNew世代領域に空きがあるように見える場合であっても、実際には空きがない場合がありますので注意してください（空きがあるように見える場合であっても、その差は、NewGC処理用の作業域的な役割で使用済となっている場合があります）。

注意

メモリ割り当てプールに対してメモリ領域不足事象が発生した場合に出力されるheap_sizeの値は、New世代領域での使用中サイズとOld世代領域での使用中サイズの合計値です。
New世代領域とOld世代領域は別々のオブジェクト格納域として管理・制御されますから、max_heap_sizeとheap_sizeの差の大きさが、そのまま生成要求できるオブジェクトの最大サイズにはなりませんので注意してください。

例

Java VMのヒープ域不足の詳細情報の出力例

スタックトレース情報にスレッドの状態を表す情報（トレース情報の前の行）が表示されます。

```
---------------------------------------------------------------------------
OutOfMemory Log
---------------------------------------------------------------------------
pid=4696
heap_type=1
heap_size=136800
max_heap_size=6291456
perm_size=2052320
max_perm_size=67108864
requested_size=40000016
---------------------------------------------------------------------------
VM is terminated by occurred OutOfMemoryError on Java heap.
"main" prio=6 tid=0x00307000 nid=0x12a8 runnable [0x0092f000]
java.lang.Thread.State: RUNNABLE
at test.<init>(test.java:10)
at test.main(test.java:5)
```

「java.lang.Thread.State: RUNNABLE」となっていますが、"RUNNABLE"の部分が、"NEW"、"TIMED_WAITING (sleeping)"、"WAITING (on object monitor)"、"TIMED_WAITING (on object monitor)"、"WAITING

(parking)"、"TIMED_WAITING (parking)"、"BLOCKED (on object monitor)"、"TERMINATED"、"UNKNOWN"などの表示の場合があります。

9.6.3 java.lang.StackOverflowErrorがスローされた場合

StackOverflowErrorがスローされた場合、スタックオーバーフローが原因です。

スタックのサイズをチューニングしてください。
スタックのチューニング方法は、“9.4.2 スタックのチューニング”を参照してください。

なお、**StackOverflowError**がスローされず、そのままJavaプロセスが異常終了する場合もあります。

その場合の解析方法については、“9.6.3.1 スタックオーバーフロー検出時のメッセージ出力機能”を参照してください。

9.6.3.1 スタックオーバーフロー検出時のメッセージ出力機能

Javaプロセスが不当なメモリアクセスにより異常終了した場合の原因の1つとして、スレッドに対するスタックのサイズ不足、すなわちスタックオーバーフローの発生が考えられます。

FJVMでは、スタックオーバーフローが原因と考えられる不当なメモリアクセスによりJavaプロセスが異常終了した場合、その旨を原因調査情報としてFJVMログへ出力する機能を、「**スタックオーバーフロー検出時のメッセージ出力機能**」として実装しています。
FJVMログの見方については、“9.5.8 FJVMログ”を参照してください。

スタックオーバーフロー発生の原因が、Java APIで生成されたスレッドに対するスタックのサイズにある場合は、“9.4.2 スタックのチューニング”を参照して、Java APIで生成されるスレッドに対するスタックのサイズをチューニングしてください。

注意

検出対象となるスレッド

本機能でスタックオーバーフローの検出対象となるスレッドは、原則Java APIで生成されたスレッドです。
次のスレッドは、本機能による検出対象スレッドとはなりません。

- ネイティブモジュールからOSのAPIを直接使用して生成されたスレッド
- mainメソッドを実行するスレッド(Java APIで生成されたスレッドではないため)

スタックオーバフローの検出には、OSの機能を利用しています。ご使用中のOSがWindowsの場合は、上記スレッドの場合であっても、そのスレッドで実行されるJavaメソッドの中から呼び出されたネイティブメソッド内で直接発生したスタックオーバーフローについては、本機能による検出の対象となります。

注意

スタックオーバーフローが発生しても、OS側からFJVM側の処理へ制御が渡らないことがあります。その場合はFJVMログが出力されません。 Windows32/64

OSの制御処理がWindowsエラー報告(Windows Error Reporting(WER))へ直接例外制御を渡した場合には、Windowsエラー報告が出力するログファイルを確認してください。Windowsエラー報告の説明は、“9.5.9.1 クラッシュダンプ”を参照してください。

なお、スタックオーバーフロー発生時のスタック残量が少ない場合には、以下の状態でJavaアプリケーションが終了する場合があるので注意してください。

- 以下のメッセージが標準出力へ出力されただけで、FJVMログが出力されないままJavaアプリケーションが終了してしまうことがあります。

  スタックオーバーフロー検出に関するFJVMログが出力されない場合であっても、以下のメッセージが出力された場合は、スタックオーバーフローが発生したことを示します。

  ```
  An unrecoverable stack overflow has occurred.
  ```

- スタックオーバーフロー発生例外に対する制御が、OS側からFJVM側の処理およびWindowsエラー報告（Windows Error Reporting（WER））のどちらにも渡らず、そのままJavaアプリケーションが終了してしまうことがあります。

  その場合は、スタックオーバーフロー発生を検出することができません。

9.6.4 SIGBUS発生により異常終了した場合 Solaris64

Solaris上で実行しているプロセスが、以下の状態のSIGBUS発生により異常終了した場合は、システムのメモリ資源／スワップ不足により発生した異常終了です。

- signal no : 10(SIGBUS)
- signal code : 3(BUS_OBJERR)
- signal error: 12(ENOMEM)

そして、Javaプロセスが異常終了した場合に出力されるFJVMログにおいて、以下の情報が出力されている場合が、上記状態に相当します(JDK/JREのバージョンにより異なります)。

```
siginfo:si_signo=SIGBUS: si_errno=Not enough space, si_code=3(BUS_OBJERR), si_addr=16進数
```

または

```
siginfo:si_signo=10, si_errno=12, si_code=3, si_addr=16進数
```

異常終了時の情報として上記情報が出力された場合は、不要なプロセスを終了して仮想メモリに余裕を持たせるか、物理メモリ(RAM)またはスワップファイルを拡張して仮想メモリを増やすようにチューニングを行ってください。

9.6.5 プロセスが消滅(異常終了)した場合

何の痕跡も残さずに突然プロセスが消滅した場合に、考えられる原因とその対処方法を説明します。

想定される原因(スタックオーバーフロー)

FJVMには、**スタックオーバーフロー**検出時にメッセージを出力する機能を備えています。FJVMログを分析することにより、スタックオーバーフローが発生したかどうかを確認することができます。FJVMログの分析方法は、“9.6.3.1 スタックオーバーフロー検出時のメッセージ出力機能”を参照してください。

スタックオーバーフローが発生したことを確認できた場合、該当するスタックのサイズをチューニングしてください。スタックのチューニング方法は、“9.4.2 スタックのチューニング”を参照してください。

Windows32/64

通常、スタックオーバーフローが発生した場合、java.lang.StackOverflowErrorがスローされ、Windowsエラー報告（Windows Error Reporting（WER））が検知してユーザダンプを出力します。

しかし、OSが高負荷状態になったり、スタックオーバーフロー発生時のスタック残量が少なかったりすると、OSからFJVMにもWindowsエラー報告にも制御が渡らないまま、痕跡を残さずにプロセスが消滅することがあります。

したがって、プロセスが消滅した原因が不明な場合は、スタックのサイズを拡張して、現象が改善できるかどうかを確認してください。スタックのサイズを拡張しても改善できない場合は、別の原因を調査してください。

なお、Windowsエラー報告の説明は、“9.5.9.1 クラッシュダンプ”を参照してください。

想定される原因(長時間コンパイル処理の検出機能による終了)

FJVMの“長時間コンパイル処理の検出機能”による終了の可能性があります。
詳細は、“9.3.2 長時間コンパイル処理の検出機能”を参照してください。

Javaアプリケーションを“-XX:CompileTimeout”オプション付きで起動した場合は、標準出力にFJVMからのメッセージが出力されていないかどうかを確認してください。

想定される原因(シグナルハンドラ) Solaris64 Linux32/64

Java VM以外のモジュールで、シグナルハンドラを登録した場合、Javaアプリケーションが正常に動作せずに、異常終了することがあります。詳細は、“9.5.8.2 異常終了時のシグナルハンドラ情報 ”を参照してください。

FJVMを使用している場合は、FJVMログにシグナルハンドラ情報が出力されますので、それを確認してください。

想定される原因(JNI処理の異常)

JNI経由でJava以外の言語で開発したネイティブモジュールと連携する際、JNIの使用方法を誤ると、プロセス消滅の原因となります。

このようなときは、“-Xcheck:jni”オプションを指定して、JNI処理でメッセージが出力されないかどうかを確認してください。“-Xcheck:jni”オプションの詳細は、“9.5.10 JNI処理異常時のメッセージ出力”を参照してください。

JNI処理に誤りがなくても、ネイティブモジュールで異常終了またハングアップが発生すると、Javaアプリケーションのプロセスが消滅する場合があります。たとえば、スレッドアンセーフな関数を使用している場合は、注意が必要です。

例

スレッドアンセーフな関数の例 Solaris64

次の関数を使用したときに、障害が発生した事例があります。

- vfork

想定される原因(プログラムによる終了)

Javaプロセスが、特別なメッセージ出力などがないまま、予想外の状態で終了した場合、原因の1つとして、次のどれかが考えられます。

- java.lang.Runtime.exit()を予想外の箇所で実行した
- java.lang.Runtime.halt()を予想外の箇所で実行した
- java.lang.System.exit()を予想外の箇所で実行した

FJVMを使用している場合は、“9.5.6 Java VM終了時における状態情報のメッセージ出力機能”を参照して、対処してください。

9.6.6 ハングアップ(フリーズ)した場合

本節では、Javaプロセスが残っているにもかかわらず、プログラムが無応答になった場合、考えられる原因とその対処方法を説明します。

想定される原因(デッドロック)

デッドロックが発生した場合、そのスレッドが停止されます。
ハングアップしたときに、スレッドダンプを採取して、デッドロックがないかどうかを確認してください。

スレッドダンプの採取方法および解析方法の詳細は、“9.5.3 スレッドダンプ”を参照してください。

想定される原因(ガーベジコレクション)

ガーベジコレクションが発生すると、ガーベジコレクションが終了するまでの間、Javaアプリケーションのすべてのスレッドが停止されます。
これにより、Javaアプリケーションがハングアップしたかのように見える場合があります。

ガーベジコレクションのログを採取して、ガーベジコレクションが動作したタイミングを照合してください。ガーベジコレクションが原因で無応答のような現象になる場合は、Javaヒープをチューニングして、ガーベジコレクションの動作具合を改善してください。

ガーベジコレクションのログを採取する方法は、“9.2.6 ガーベジコレクションのログ出力”を参照してください。
Javaヒープのチューニング方法は、“9.4.1 Javaヒープのチューニング”を参照してください。

想定される原因(JNI処理の異常)

JNI経由でJava以外の言語で開発したネイティブモジュールと連携する際、JNIの使用方法を誤ると、ハングアップの原因となります。

このようなときは、“-Xcheck:jni”オプションを指定して、JNI処理でメッセージが出力されないかどうかを確認してください。“-Xcheck:jni”オプションの詳細は、“9.5.10 JNI処理異常時のメッセージ出力”を参照してください。

JNI処理に誤りがなくても、JNIモジュールで異常終了またハングアップが発生すると、Javaアプリケーションがハングアップする場合があります。たとえば、スレッドアンセーフな関数を使用している場合は、注意が必要です。

スレッドアンセーフな関数の例 Solaris64

次の関数を使用したときに、障害が発生した事例があります。

- vfork

9.6.7 スローダウンが発生した場合

Javaアプリケーションの動きが遅くなる現象(スローダウン)が発生した場合に、考えられる原因とその対処方法を説明します。

想定される原因(ガーベジコレクション)

ガーベジコレクションが発生すると、ガーベジコレクションが終了するまでの間、Javaアプリケーションのすべてのスレッドが停止されます。このため、Javaアプリケーションのレスポンス(応答)が遅くなる場合があります。

ガーベジコレクションのログを採取して、ガーベジコレクションが動作したタイミングとスローダウンが発生したタイミングを照合してください。ガーベジコレクションが原因でスローダウンになる場合は、Javaヒープをチューニングして、ガーベジコレクションの動作具合を改善してください。

ガーベジコレクションのログを採取する方法は、“9.2.6 ガーベジコレクションのログ出力”を参照してください。
Javaヒープのチューニング方法は、“9.4.1 Javaヒープのチューニング”を参照してください。

スローダウンの事例

同じソフトウェアとJavaアプリケーションが動作している複数のWebサーバのうち、一部のサーバだけがスローダウンしたという事例があります。この原因は、マシンに搭載されている物理メモリ(RAM)の容量の違いでした。
物理メモリ(RAM)が少ないマシンの場合、ガーベジコレクションの実行に伴い、ディスクのスワッピングが発生し、スローダウンすることがあります。

9.7 Javaツール機能

本製品では、Javaプログラムのチューニングやトラブルシューティングに有用なツールを提供しています。ツールには、次の5つがあります。

- メソッドのトレースを出力するメソッドトレース機能
- Javaヒープ使用量を出力するjheap
- スレッドダンプを出力するスレッドダンプツール Windows32/64
- Java監視機能
- JDKに含まれるトラブルシューティングに役立つツール

ツールの格納先

- メソッドのトレースを出力するメソッドトレース機能

- Javaヒープ使用量を出力するjheap
- スレッドダンプを出力するスレッドダンプツール Windows32/64

  Windows32/64

  ー JDK 使用時: [Interstageインストールフォルダ]¥jdk7¥tools

  ー JRE 使用時: [Interstageインストールフォルダ]¥jre7¥tools

  Solaris64 Linux32/64
  以下の“$DIR”はインストール時に指定する相対ディレクトリ名です。“$DIR”のシステム推奨名は“opt”です。

  ー JDK 使用時: /$DIR/FJSVawjbk/jdk7/tools

  ー JRE 使用時: /$DIR/FJSVawjbk/jre7/tools
- Java監視機能

  Windows32/64

  ー [Interstageインストールフォルダ]¥jdk7¥tools

  Solaris64 Linux32/64
  以下の“$DIR”はインストール時に指定する相対ディレクトリ名です。“$DIR”のシステム推奨名は“opt”です。

  ー /$DIR/FJSVawjbk/jdk7/tools
- JDKに含まれるトラブルシューティングに役立つツール

  Windows32/64

  ー [Interstageインストールフォルダ]¥jdk7¥bin

  Solaris64 Linux32/64
  以下の“$DIR”はインストール時に指定する相対ディレクトリ名です。“$DIR”のシステム推奨名は“opt”です。

  ー /$DIR/FJSVawjbk/jdk7/bin

各ツールの詳細は、“トラブルシューティング集”の“Javaツール機能”を参照してください。

第10章　JDK/JRE 8のチューニング

CやC++では、メモリに割り当てた領域は、不要になった時点でプログラマーが明示的に解放する必要があります。しかし、この解放処理に漏れやミスがあると、メモリリーク、プログラム停止および暴走が発生するデメリットがあります。

Javaでは、ガーベジコレクション(GC)を導入したことにより、プログラマーがメモリ管理において作業の負担を軽減できるようになりました。その反面、GCが発生するたびにJavaアプリケーションが一時停止するため、パフォーマンス劣化の要因になることがあります。また、Java独特のメモリリークも存在します。

Javaでは、スレッドを管理しやすいため、大量のスレッドを生成する傾向があります。このため、スタックが大量に生成されて、メモリ不足になることもあります。

以上から、多くの場合において、JDK/JRE 8のチューニングは、スタックサイズ、または、GCの発生頻度とその処理時間を調整することになります。本章では、Javaアプリケーションのチューニングにあたって、必要な基礎知識やチューニング方法などを説明します。

10.1 基礎知識

JDK/JRE 8のチューニングを行う際に必要な知識を説明します。

10.1.1 JDK関連のドキュメント

JDKドキュメント

本製品に搭載しているJDK のドキュメントは、以下のURLにあります。

```
https://docs.oracle.com/javase/jp/8/docs/
```

- JavaFXのドキュメントについて

  JavaFX に関しては、以下のURLを参照してください。

  ```
  http://www.oracle.com/technetwork/jp/java/javafx/documentation/index.html
  ```

- Java VisualVMのドキュメントについて

  Java VisualVM に関しては、以下のURLを参照してください。

  ```
  https://docs.oracle.com/javase/jp/8/docs/technotes/guides/visualvm/intro.html
  ```

- jcmdのドキュメントについて

  jcmd に関しては、以下のURLを参照してください。

  - Windows32/64

    ```
    http://docs.oracle.com/javase/jp/8/docs/technotes/tools/windows/jcmd.html
    ```

  - Solaris64 Linux32/64

    ```
    http://docs.oracle.com/javase/jp/8/docs/technotes/tools/unix/jcmd.html
    ```

javaコマンドおよびJava VMのオプションには、チューニングに関するオプションがあります。本章では、これらのオプションを随所で紹介します。
各オプションの詳細は、次を参照してください。なお、本マニュアル内で具体的に説明していないJava VMのチューニングに関するオプションは、富士通版Java VMではサポートしておりません。

- javaコマンドのオプション

  JDKドキュメントの「JDKツールとユーティリティ」の「java」

- Java HotSpot VM Options
  - Windows32/64

    http://docs.oracle.com/javase/8/docs/technotes/tools/windows/java.html
  - Solaris64 Linux32/64

    http://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html

移行について

JDK1.4からJDK5.0への移行については、以下のURLを参照してください。

- http://www.oracle.com/technetwork/java/javase/compatibility-137462.html

JDK5.0からJDK6への移行については、以下のURLを参照してください。

- http://www.oracle.com/technetwork/java/javase/adoptionguide-137484.html

JDK6からJDK7への移行については、以下のURLを参照してください。

- http://docs.oracle.com/javase/jp/7/webnotes/adoptionGuide/index.html

JDK7からJDK8への移行については、以下のURLを参照してください。

- http://www.oracle.com/technetwork/jp/java/javase/jdk8-adoption-guide-2157601-ja.html

10.1.2 Java VM

製品搭載のJava VM

JDK/JREには、Javaのバイトコードを解釈・実行するエンジン部であるJava仮想マシン(**Java VM**)があります。

“表10.1 Java VMの種類と特徴”に、**Java VM**の種類と特徴を示します。

表10.1 Java VMの種類と特徴

Java VMの名称	Java VMの特徴
Java HotSpot Client VM	アプリケーション起動時間を短縮し、メモリ消費を抑制するように設計されたクライアント環境向けの**Java VM**です。 富士通版**Java HotSpot Client VM**では、Oracle CorporationのJava VMであるJava HotSpot Client VMをベースに、富士通独自技術によるトラブルシューティングに関する機能強化などを追加実装しています。 そのため、ベースとしたJava HotSpot Client VMと機能的な互換性を基本的にもっています。 追加実装された機能項目については、“10.1.3 FJVM”を参照してください。
Java HotSpot Server VM (FJVM)	アプリケーション起動時間の短縮などよりも、安定性および、スループットの向上を考慮して設計されたサーバ環境向けの**Java VM**です。 富士通版**Java HotSpot Server VM**では、Oracle CorporationのJava VMであるJava HotSpot Server VMをベースに、富士通独自技術による性能改善やトラブルシューティングに関する機能強化などを追加実装しています。 そのため、ベースとしたJava HotSpot Server VMと機能的な互換性を基本的にもっています。 富士通版**Java HotSpot Server VM**は、Interstage Application Server搭載のJDK/JREにおけるデフォルトの**Java VM**であることから、このJava VMを特に**FJVM**と呼びます。 追加実装された機能項目については、“10.1.3 FJVM”を参照してください。

- Interstage Application Server搭載のJDK/JREでは、**Java HotSpot Client VM**と**FJVM**(=**Java HotSpot Server VM**)を搭載しています。

デフォルトの**Java VM**は、**FJVM**です。これは、javaコマンドのオプションに、“-server”または“-fjvm”を指定することと同義です。**Java HotSpot Client VM**を使用したい場合は、オプションに“-client”を指定してください。

- Windows64 Solaris64 Linux64
  実行モードが64ビットモードのJDK/JREでは**FJVM**だけ使用可能です。**Java HotSpot Client VM**は搭載していません。

注意

エルゴノミクス機能によるJava VMの自動選択機能

富士通版JDK/JRE では、エルゴノミクス機能によるJava VMの自動選択機能（マシンのCPU数や物理メモリ量などに応じて、使用するJava VMを自動的に選択する機能）は無効になっています。

Java VM関係の資料

- **Java VM**の詳細は、JDKドキュメンテーションの次を参照してください。

  [Java 概念図の説明] > [Java 仮想マシン] および
  https://docs.oracle.com/javase/jp/8/docs/technotes/guides/vm/index.html

- 他にも、**Java VM**に関連する資料があります。

  ― Java Language and Virtual Machine Specifications

  http://docs.oracle.com/javase/specs/

  ― Java HotSpot VM Options

  http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html

10.1.3 FJVM

本章におけるJava VMに関する情報は、デフォルトのJava VMである「富士通版Java HotSpot Server VM(=FJVM)」を中心に説明しています。なお、特に断り書きの無い限り、それらの情報は「富士通版Java HotSpot Client VM(=Java HotSpot Client VM)」に対しても、そのまま適用されます。

FJVMにおいて提供される「富士通独自技術により強化された機能および固有機能」は以下のとおりです。

FJVM固有機能を除き、Java HotSpot Client VMも同様です。

- New世代領域用制御処理並列化機能付きGC（パラレルGC）
- コンカレント・マーク・スイープGC付きパラレルGC（CMS付きパラレルGC）
- コンパイラ異常発生時の自動リカバリ機能 (注)
- 10.3.2 長時間コンパイル処理の検出機能 (注)
- 10.5.4 クラスのインスタンス情報出力機能
- java.lang.System.gc()実行時におけるスタックトレース出力機能
- Java VM終了時における状態情報のメッセージ出力機能
- FJVMログ
- メモリ領域不足事象発生時のメッセージ出力機能の強化

注) FJVMにだけ提供されるFJVM固有機能です。

10.1.4 仮想メモリと仮想アドレス空間

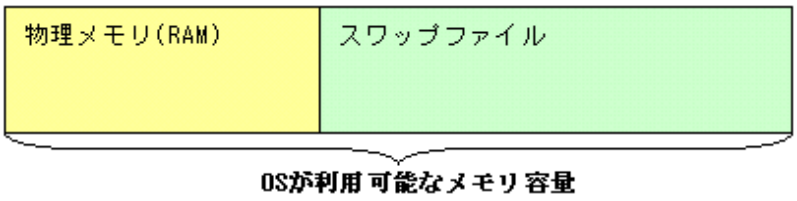
Javaアプリケーションを開発・運用するにあたり、OSのメモリ管理の概要を知っておく必要があります。本節では、OSの一般的なメモリ管理技法の1つである**仮想アドレス空間**を説明します。

なお、**仮想アドレス空間**の具体的なアーキテクチャーはOSごとに異なりますが、本節では各OSに共通する内容を説明します。

仮想メモリ

OSは、物理メモリ(RAM)だけでなくスワップファイルを活用することにより、多くのメモリ領域を使用することができます。このテクノロジーを**仮想メモリ**といいます。**仮想メモリ**の容量は、RAMとスワップファイルのサイズの合計になります。

図10.1 OSが利用可能なメモリ容量



ただし、ハードディスクへのアクセスはRAMより低速なので、メモリのスワッピングは性能に大きく影響を与えますので、注意が必要になります。

仮想アドレス空間

OS上でプログラムを起動すると、OSがプログラムを実行・管理する単位としてのプロセスが生成されます。同様にして、Javaアプリケーションを起動すると、Javaプロセスが生成されます。

OS上で生成されたプロセスには、**仮想アドレス空間**が割り当てられます。**仮想アドレス空間**は、それぞれのプロセスで独立したものであり、あるプロセスから別のプロセスの**仮想アドレス空間**にアクセスすることはできません。マシンに積んでいる物理メモリ(RAM)のサイズとは関係なく、**仮想アドレス空間**のサイズは、常に一定です。たとえば、32ビットアーキテクチャーのOSの場合、物理メモリ(RAM)のサイズに依存せず、**仮想アドレス空間**のサイズは常に4GB(2の32乗バイト)です。

このため、大量の**仮想メモリ**を用意しても、1つのプロセスで使用できるメモリ量の上限は**仮想アドレス空間**の上限になりますから、プロセスが**仮想アドレス空間**の大きさ(実際には後述のユーザ空間の大きさ)を超えるメモリ量を必要とする場合は、メモリ不足の状態になります。

逆に、プロセスが必要とするメモリ量が仮想アドレス空間の大きさ(実際には後述のユーザ空間の大きさ)の上限未満だったとしても、そのメモリ量に相当する仮想メモリ量がOS上になければ、メモリ不足の状態になります。

また、大量の**仮想メモリ**を用意しても、OS上で大量のプロセスが動作していて**仮想メモリ**を大量に消費している状態であれば、各プロセスに割り当てられた**仮想アドレス空間**を使い切っていなくても、メモリ不足の状態になる場合があります。

ユーザ空間

プロセスが持つ**仮想アドレス空間**のうち、実際にプロセスが使用できる空間を、**ユーザ空間**といいます。**ユーザ空間**には、プログラムの実体(Windows(R)でJavaアプリケーションを実行する場合は、java.exeなど)がコピーされるだけでなく、スタックやヒープなどのさまざまなセグメントがあります。更に**ユーザ空間**は、実行するプログラムだけでなく、そのプログラムを実行させるためのOS側のプログラムなどでも使用します。Javaプロセスの**ユーザ空間**の場合には、前述の各セグメントの他に、Javaオブジェクトを格納するセグメント(=Javaヒープ)があります。このため、Javaアプリケーションをチューニングする際の対象となるJavaヒープのサイズの上限値は、**ユーザ空間**のサイズよりも少なくなります。

Javaアプリケーションのチューニングを実施する際は、仮想メモリの容量やプロセスの使用状況など、システムの状態を考慮する必要があります。なおOSの種類やアプリケーションの実行モードによって、**ユーザ空間**として使用できる上限値が異なるため、注意が必要です。

なお、各セグメント獲得・解放時の実際の制御処理はOSが行います。そのため、各セグメントに関する管理方法/動作仕様/大きさについては、JDK/JREを実行する各OSの仕様に依存します。そして、プロセス外部からは**ユーザ空間**に空きが存在するように見える場合であっても、OSの制御処理上は利用できる空きが無いと判断され、セグメントに対応するメモリ領域が獲得できず、プロセス動作が異常となる場合がありますので注意してください。
また、OSの制御処理上、各セグメント間には、アプリケーションから利用できない隙間領域が存在する場合があります。そのため、通常、ユーザ空間としての上限値まで仮想メモリを使用することはできません。プロセスが使用するメモリ量として、ユーザ空間の上限値まで使えることを前提とした設計にはしないでください。

実行モード

実行モードが変わると、プログラムの実行に必要な基本メモリ量が変わります。
具体的には、プログラムの実行モードを32ビットモードから64ビットモードに変更して実行する場合、ポインタを扱うために必要

となるメモリ量の単位が、4バイトから8バイトになります(OSによっては、整数を扱う際に必要となるメモリ量の単位も異なります)。そのため、同じソースから作られたプログラムの場合であっても、64ビットOS上で64ビットモードのプログラムを使ってアプリケーションを動作させる場合は、32ビットOSまたは64ビットOS上で32ビットモードのプログラムを使ってアプリケーションを動作させる場合よりも、より大きなメモリ量が必要となります。

64ビットモード時のCデータ型モデル

- Solaris64 Linux64
  LP64モデル:long型/ポインタが32ビット(4バイト)から64ビット(8バイト)へ変更されます。
- Windows64
  P64モデル :ポインタが32ビット(4バイト)から64ビット(8バイト)へ変更されます。

Javaアプリケーションの場合もC/C++アプリケーションの場合と同様、実行モードが変わると、プログラムの実行に必要な基本メモリ量が変わります。
特に、オブジェクトを構成するデータ内容にはポインタを扱う情報も多数あるため、64ビットモードの場合、1オブジェクトあたりに必要となるメモリ域の大きさは32ビットモードの場合よりも大きくなります。
そのため、通常、64ビットOS上で64ビットモードのJDK/JREを使ってJavaアプリケーションを動作させる場合、32ビットOSまたは64ビットOS上で32ビットモードのJDK/JREを使ってJavaアプリケーションを動作させた場合の設定に対して、1.5～2倍のJavaヒープ量が必要です。

10.1.5 スタック

ここでは、**スタック**について簡単に説明します。

プログラムがスレッドを生成すると、OSがそのスレッドに対して**スタック**と呼ばれるメモリ領域をそのスレッドの終了時まで自動的に割り当てます。**スタック**は、スレッド上で実行されるメソッドや関数が使用するローカル変数などの一時的なデータを格納するための作業域として使用されます。

スレッド上では多くのメソッドや関数が入れ子状態で呼び出されるので、これらのメソッドや関数が使用する一時的な作業域を管理するために、OSは**フレーム**と呼ばれる区切りで個々の作業域をスタック上に積み上げることで管理しています。(メソッドや関数が呼び出されるごとに、これらが使用する作業域を**フレーム**という区切りでスタック上に積み上げ、**フレーム**内のデータは呼び出したメソッドや関数からの復帰時に破棄されます。)

このため、あるメソッドを再帰的に無限に呼び出すなどしてメソッドを深く呼び出した場合や、非常に大きなサイズのローカル変数などを使用するメソッドを呼び出した場合などには、**スタック**域の枯渇により、スタック上に**フレーム**を積み上げることができなくなり、スタックオーバーフローが発生する場合があります。

Javaアプリケーションの場合、通常、スタックオーバーフローが発生すると、java.lang.StackOverflowErrorがスローされます。ただし、Javaプロセス中のネイティブモジュール実行時の処理でスタックオーバーフローが発生した場合には、java.lang.StackOverflowErrorはスローされません。

注意

スタック領域の管理

スタック領域の実際の管理はOSが行います。そのため、**スタック**領域に関する管理方法/動作仕様/大きさについては、JDK/JREを実行する各OSの仕様に依存します。
なお、Javaアプリケーション実行時のスタックオーバーフロー発生をJava VMが検出できる様にするために、**スタック**領域の一部をJava VMの制御用領域として使用します。そのため、Javaアプリケーションから使用できる**スタック**領域の大きさは、**スタック**として割り当られた領域の大きさよりも若干小さな大きさになります。

注意

クラスファイル実行時のスタック領域の利用

Javaの実行環境であるJava VMが起動された後、Java VMは、実行対象プログラムであるクラスファイルを読み込み、クラスファイルを以下の2つの方法を用いて実行します。

- インタプリタによるバイトコードの実行

- 動的コンパイルにより、バイトコードを機械命令に翻訳してから実行
  あるクラスファイル中の同じJavaメソッドを実行する場合であっても、Java VMによる実行方法が異なる場合は、実行時に使用されるスタックの大きさが異なります。
  なお、クラスファイルの実行方法については、“10.3 動的コンパイル”を参照してください。

注意

ユーザ空間のメモリ不足によるスレッド生成エラー

スタックは、プロセス内のユーザ空間から割り当てられます。
そして、ユーザ空間のメモリ不足により**スタック**などを割り当てることができなかった場合には、スレッド生成処理でエラーが発生します。
Javaプロセスの場合、Javaヒープなどのサイズを大きく確保すると、その分だけ**スタック**として割当て可能な領域が減少するため、Javaプロセス内で生成可能なスレッドの個数も減少します。
このため、Javaプロセス内のスレッド生成処理がエラーとなる要因の1つとして、Javaヒープなどのサイズを大きく確保したことによるユーザ空間のメモリ不足が考えられます。

注意

Javaプロセスの消滅 Windows32/64

Windows(R)では、Javaプロセス内でスタックオーバーフローが発生した場合、システム状態やプログラム状態によっては、OSからFJVMやWindowsエラー報告（Windows Error Reporting（WER））に制御が渡らず、痕跡を残さずにJavaプロセスが消滅する場合があります。
なお、Windowsエラー報告の説明は、“10.5.8.1 クラッシュダンプ”を参照してください。

10.1.6 Javaヒープ、メタスペースとガーベジコレクション

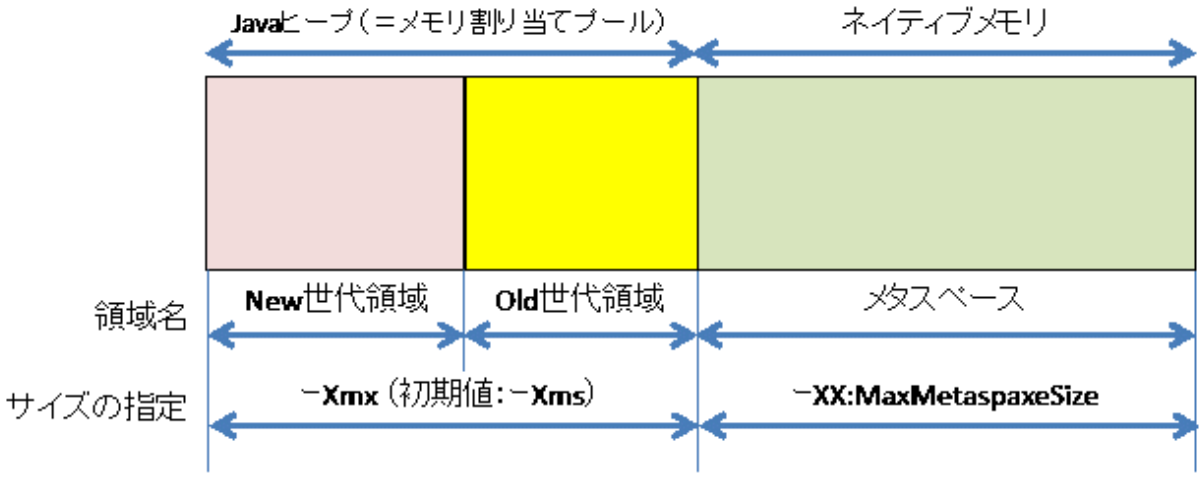
ここでは、**Javaヒープ**、**メタスペース**と**ガーベジコレクション(GC)**を説明します。

Javaヒープは、Javaプロセス内に存在するJavaオブジェクトを格納するための領域です。
Javaヒープは、**New世代領域**、**Old世代領域**に大別され、Java VMが管理・制御します。

JDK/JRE 7以前でJavaヒープに存在していた、Permanent世代領域は、JDK/JRE 8では廃止されました。Permanent世代領域の代替として、メタスペースが導入されました。

Java VMは、Javaアプリケーションの実行時に、Javaオブジェクトを**Javaヒープ**の各領域に格納します。**Javaヒープ**の空き容量がなくなった場合は、java.lang.OutOfMemoryErrorがスローされます。
また、不要となったJavaオブジェクトは**GC**により回収され、**Javaヒープ**の空き領域が増加します。なお、**New世代領域**に存在する不要となったJavaオブジェクトを回収する**GC**処理を「**NewGC**処理(または**マイナーGC**処理)」といいます(NewGCまたはマイナーGCと略す場合もあります)。また、**New世代領域**だけでなく、**Old世代領域**を含む**Javaヒープ全体**に存在する不要となったJavaオブジェクトを回収する**GC**処理を「**FullGC**処理」といいます(FullGCと略す場合もあります)。

図10.2 Javaヒープの構造



- **New世代領域とOld世代領域(メモリ割り当てプール)**

  New世代領域と**Old世代領域**は、インスタンスや配列などのJavaオブジェクトを管理する領域です。
  New世代領域は寿命の短いJavaオブジェクトを管理します。通常、Javaアプリケーションで要求されたオブジェクトは、**New世代領域**に生成されます。一定期間**New世代領域**で生存したJavaオブジェクトは、**GC**処理によって**Old世代領域**に移動されます。そして、**Old世代領域**に存在する不要となったJavaオブジェクトは、**FullGC**処理によって回収されます。
  領域がNew世代とOld世代に分かれるのは、世代別に**GC**処理を実行するためです。
  なお、**メモリ割り当てプール**全体のサイズの初期値および最大値は、“-Xms”オプションおよび“-Xmx”オプションで指定することができます。

- **メタスペース**

  メタスペースは、Javaのクラス、メソッドなど、永続的に参照される静的オブジェクトを管理する領域です。

  メタスペースはJavaヒープには存在せず、OSが管理するネイティブメモリ上の領域に取られます。そのため、OSが領域を獲得できれば上限はありません（以下で説明する“-XX:MaxMetaspaceSize”オプションでサイズを指定しない場合）。JDK/JRE 7のPermanent世代領域と比較して、OutOfMemoryErrorが発生しにくい、という特徴があります。

  メタスペースは、クラスローダーごとに分けて管理しています。クラスローダーがFullGC処理によって回収されると、対応する**メタスペース**も削除されます。あるクラスローダーでロードされたクラス情報を別のクラスローダーが管理する**メタスペース**に格納できません。また、各クラスローダーの**メタスペース**は、クラス情報ごとに取得するのではなく、ある程度のまとまったサイズごとに取得して、そこにクラス情報を設定していきます。

  メタスペースの大きさの最大値は、“-XX:MaxMetaspaceSize”オプションで指定することができます。オプションを指定しなかった場合、メタスペースはメモリ資源のある限り無制限です。しかし、複数のJavaプロセスを同時に起動した場合に、全体で使用するメモリ量をチューニングする場合は、本オプションを指定することを推奨します。

  “-XX:MetaspaceSize”オプションは、初めて超えたときにFullGCを発生させるしきい値となるサイズを指定します。Javaプロセスの起動時に確保する初期サイズではありません。

  JDK/JRE 7のPermanent世代領域に関連するオプションが指定された場合は無視されます。

注意

Compressed Class Space

64ビットモードのJDK/JRE 8では、通常のメタスペースの他に、クラス情報の一部を格納する**Compressed Class Space**と呼ばれる領域を使用します。この領域は、Java VM起動時に1GBのサイズだけリザーブされます。

参考

ユーザ空間

Javaヒープは、Javaプロセスのユーザ空間内に割り当てられます。

また指定された最大値まで**Javaヒープ**が利用できる環境にするため、Java VM起動時に、**メモリ割り当てプール**を、最大値の大きさで連続領域としてリザーブします(同一プロセス内の他の処理から使用できないようにします)。

このため、**Javaヒープ**の最大値を大きく設定すると、その分だけスタックなど他の処理に割り当てられるメモリ領域が減少します。

別な言い方をすれば、ユーザ空間内で使用できるメモリ量の上限から、Javaアプリケーション自身、Java VM、そしてネイティブモジュール(OSを含む)が使用するメモリ量などを差し引いた値が、**Javaヒープ**として使用できる上限となります。

なおJava VM起動時に指定された**Javaヒープ**の大きさが利用できない場合、Java VMは以下のメッセージを出力し、Javaプロセスを終了させます。

```
Error occurred during initialization of VM
Could not reserve enough space for object heap
```

このメッセージが出力された場合は、**Javaヒープ**を小さくするチューニングを行ってください。

またSolaris用およびLinux用のJava VMにおいて、Java VM起動時またはJavaアプリケーション実行中に「ユーザ空間不足」または「仮想メモリ不足」が発生した場合、Java VMは以下のメッセージを出力し、Javaプロセスを終了させます。

- Java VM起動時の場合

```
制御名: mmap failed: errno=エラー情報, 制御情報....
Error occurred during initialization of VM
mmap failure
```

- Javaアプリケーション実行中の場合

```
制御名: mmap failed: errno=エラー情報, 制御情報....
(上記メッセージに続いて、java.lang.OutOfMemoryErrorメッセージが出力される場合があります。)
```

制御名:

メモリ不足が発生した際のJava VMの制御名

エラー情報:

メモリ不足が発生した際のJava VMのエラー情報

制御情報:

メモリ不足が発生した際のJava VMの制御情報

ユーザ空間が不足している場合は、Javaヒープを小さくするチューニングを行ってください。

仮想メモリが不足している場合は、他の不要なプロセスを終了して仮想メモリに余裕を持たせるか、物理メモリ(RAM)またはスワップファイルを拡張して仮想メモリを増やすようにチューニングを行ってください。

参考

Javaヒープにメモリを割り当てるタイミング

Java VMは、OSの仮想メモリ資源を効率的に利用するために、起動時に**Javaヒープ**の各領域の初期値を割り当て、各領域の最大値まで段階的に拡大する制御方法を用いています。
具体的には、Javaプロセスの起動時は**メモリ割り当てプール**の各初期値のサイズを割り当てます。その後、各領域のサイズは各領域の最大値までの範囲内で、GC処理後や、オブジェクトの生成時のタイミングで、その時々の最適な値に自動調整されます(GC処理としてメモリ割り当てプールの最小使用量保証機能を無効にしたパラレルGCを使用している場合は、**メモリ割り当てプール**の大きさが、初期値よりも小さくなる場合があります)。

参考

スワップ発生時の対処方法

Javaヒープの各領域を拡大していく過程で、OS側で物理メモリの資源をディスクにスワップする処理が発生する場合があります。このスワップ処理により、各領域を拡大するFullGC処理に時間がかかる場合があります。FullGC処理中のスワップ処理発生による時間遅延が問題になる場合は、物理メモリを増やす、またはJavaヒープを小さくするチューニングを行ってください。

パラレルGCについては、“10.2.3 New世代領域用制御処理並列化機能付きGC（パラレルGC）”を参照してください。

10.1.7 FJVMに対して指定可能なチューニング用オプション

Javaヒープやメタスペースのチューニングなど、FJVMに対して指定可能な**Java VMのチューニングに関するオプション**を、以下に示します。

各オプションの使用法については、本マニュアルを参照してください。
なお、本マニュアル内で具体的に説明していない**Java VMのチューニングに関するオプション**は、FJVMではサポートしておりません。

Javaヒープおよびメタスペースチューニング用のオプション

-Xms
-Xmx
-XX:NewSize
-XX:MaxNewSize
-XX:NewRatio
-XX:SurvivorRatio
-XX:TargetSurvivorRatio
-XX:MaxTenuringThreshold
-XX:InitialTenuringThreshold
-XX:MetaspaceSize
-XX:MaxMetaspaceSize
-XX:CompressedClassSpaceSize

各オプションの説明は、“10.4.1 Javaヒープおよびメタスペースのチューニング”を参照してください。

スタックサイズチューニング用のオプション

-Xss
-XX:CompilerThreadStackSize

各オプションの説明は、“10.4.2 スタックのチューニング”を参照してください。

使用するガーベジコレクション処理を選択するオプション

-XX:+UseSerialGC
※オプションの説明は、“10.2.2 標準GC（シリアルGC）”を参照してください。

-XX:+UseParallelGC
※オプションの説明は、“10.2.3 New世代領域用制御処理並列化機能付きGC（パラレルGC）”を参照してください。

-XX:+UseConcMarkSweepGCUse
※オプションの説明は、“10.2.4 コンカレント・マーク・スイープGC付きパラレルGC（CMS付きパラレルGC）”を参照してください。

ガーベジコレクション処理のチューニング用オプション

パラレルGC用

-XX:ParallelGCThreads
-XX:+AutomaticallyJavaHeapSizeSetting
-XX:GCTimeLimit
-XX:GCHeapFreeLimit
-XX:+UseGCOverheadLimit

※各オプションの説明は、“10.2.3 New世代領域用制御処理並列化機能付きGC（パラレルGC）”を参照してください。

CMS付きパラレルGC用：

-XX:ParallelGCThreads
-XX:ConcGCThreads

※各オプションの説明は、“10.2.4 コンカレント・マーク・スイープGC付きパラレルGC（CMS付きパラレルGC）”を参照してください。

共通：

-XX:-UseCompressedOops (64ビットモード版JDK/JREの場合)

オプションの説明は、“10.2.5 オブジェクト参照の圧縮機能”を参照してください。

チューニングの際に使用するログ出力などのデバッグ用オプション

ガーベジコレクションのログ出力用：

-verbose:gc
-XX:+ClassUnloadingInfo
-Xloggc

※各オプションの説明は、“10.2.6 ガーベジコレクションのログ出力”を参照してください。

-XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
各オプションの説明は、“10.2.6.1 ガーベジコレクション処理の結果ログ詳細出力機能”を参照してください。

動的コンパイルのログ出力用：

-XX:+PrintCompilation

オプションの説明は、“10.3.3 動的コンパイル発生状況のログ出力機能”を参照してください。

その他：

-XX:-OmitStackTraceInFastThrow
※オプションの説明は、“10.5.2 例外発生時のスタックトレース出力”を参照してください。

-XX:+PrintClassHistogram
※オプションの説明は、“10.5.4 クラスのインスタンス情報出力機能”を参照してください。

-XX:+PrintJavaStackAtSystemGC
※オプションの説明は、“10.5.5 java.lang.System.gc()実行時におけるスタックトレース出力機能”を参照してください。

-XX:+VMTerminatedMessage
※オプションの説明は、“10.5.6 Java VM終了時における状態情報のメッセージ出力機能”を参照してください。

-Xcheck:jni
※オプションの説明は、“10.5.9 JNI処理異常時のメッセージ出力”を参照してください。

-XX:+PrintCompilerRecoveryMessage
※オプションの説明は、“10.3.1 コンパイラ異常発生時の自動リカバリ機能”を参照してください。

-XX:CompileTimeout
※オプションの説明は、“10.3.2 長時間コンパイル処理の検出機能”を参照してください。

10.1.8 Java Native Interface(JNI)

アプリケーションミスのトラブルの確実な事前防止のためには、(CやC++等を使用した)ネイティブプログラムを、Java Native Interface(JNI)経由で利用してはいけません。実現しようとしている機能が、Javaにより記述できないか設計段階で十分に検討を行ってください。やむをえず利用する場合でも、JNIの利用は最小限にし、十分な確認とデバッグを実施してください。

JNIを利用する場合の前提スキルとして、以下は必須です。

- C/C++によるマルチスレッドプログラミングの経験がある
- トラブル発生時、自分でデバッグできる

注意

ファイナライズ処理を期待したリソース管理は、行わないでください。JNI関連で、もっともトラブルが多いのは、ネイティブプログラム側で確保したメモリの後処理漏れです。例えば、次のようなプログラミングをしてはいけません。

```
------- Java ---------------
class A {
  native long nativeAlloc();
  native void nativeFree(long a);
  long address;
  A() {
    address = nativeAlloc();
  }
  public void finalize() {
    nativeFree(address);
  }
}
------- Java ---------------
------- C ------------------
JNIEXPORT jlong JNICALL Java_A_nativeAlloc(JNIEnv *env, jobject o)
{
  return (jlong)malloc(10);
}
JNIEXPORT void JNICALL Java_A_nativeFree(JNIEnv *env, jobject o, jlong p)
{
  free((void*)p);
}
------- C ------------------
```

注意

エラー処理は、必ず行ってください。ネイティブプログラム側でJNI関数呼び出しをしたとき、色々なケースでJavaレベルのエラーになることがあります。このエラーに対する後処理をネイティブプログラム側で行わないと、例外がスローされている状態のままになり、それ以降のJNI関数の呼び出しに失敗し、アプリケーションが正しく動作しません。

なお「JNI関数」とは、以下のJNI仕様書に記述されている関数のことです。

https://docs.oracle.com/javase/jp/8/docs/technotes/guides/jni/spec/functions.html

これらを使う場合は、その度にExceptionOccurredを使用しエラーチェックをする必要があります。

Solaris64 Linux32/64

Solaris、Linux上でJNIを利用される場合は、連携するネイティブプログラムまたは、ネイティブライブラリにおいてシグナルハンドラの書き替えを絶対に行わないでください。
なお、JDK/JRE 1.4.0以降でサポートされたシグナル連鎖機能(Signal Chaining)を利用する場合、マルチスレッド環境におけるシグナル動作など、OSやJava VM自身のシグナル動作およびプログラミングについての知識が前提として必要です。安定稼動が要求されるシステムの設計においては、この機能の使用もお勧めできません。

10.2 ガーベジコレクション(GC)

ガーベジコレクション(GC)について説明します。

10.2.1 FJVMでサポートされるガーベジコレクション処理

FJVMでサポートされるガーベジコレクション(GC)処理には、Javaヒープの New世代領域に対するGC制御方法の違いにより、以下の3種類があります。

- 標準GC(シリアルGC)
- New世代領域用制御処理並列化機能付きGC(パラレルGC)
- コンカレント・マーク・スイープGC付きパラレルGC(CMS付きパラレルGC)

なお、JDK/JREのバージョン、実行モード、Java VMの種類により、サポートされるGCの種類が異なります。また、Java VMの種類により、デフォルトで動作するGCが異なります。
“表10.2 サポートされるGCの種類”に、サポートされるGCの種類とデフォルトのGCを示します。

表10.2 サポートされるGCの種類

	JDK/JREの実行モード	32ビットモード		64ビットモード※
	Java VMの種類	Client VM	FJVM	FJVM
シリアルGC		◎	○	○
パラレルGC		○	◎	◎
CMS付きパラレルGC		□	□	□

○:

サポートされるGC

◎:

サポートされるGC、かつデフォルトのGC

□:

Interstage Application Server Enterprise EditionでだけサポートされるGC

※実行モードが64ビットモードのClient VMは提供していません。

GCは、デフォルトのGCの使用を推奨します。
通常、デフォルトのGCを変更する必要はありません。

注意

New世代領域用GC制御に対し「New世代領域サイズ自動調整機能」を追加して構成されたGC処理「**FJGC**」を提供していません。

Java HotSpot Client VMに対して“-XX:+UseFJGC”オプションを指定した場合は、以下のエラーメッセージが標準エラー出力へ出力され、Javaプロセスの起動に失敗します。

FJVMに対して“-XX:+UseFJGC”オプションを指定した場合は、以下のワーニングメッセージが標準エラー出力へ出力され、オプションの指定は無効になります。

```
Unrecognized VM option '+UseFJGC'
```

New世代領域の使われ方

NewGC処理では、New世代領域を「eden space」、「from space」および「to space」の3つの内部領域に細分割し、当該領域上において、一般に世代別GC制御と言われる制御方法を用いて、Javaアプリケーションが生成要求したオブジェクトを管理・制御しています。
このうち、「from space」および「to space」は、Java VMがNewGC処理を行う際の作業域的な役割を持つ領域となっています。
そのため、「from space」および「to space」の各領域が占める大きさのうち、Javaアプリケーションからのオブジェクト生成要求のために使用される大きさは、その領域の一部分だけとなります。
そのため、ガーベジコレクションのログ出力などの出力データにおいて、メモリ割り当てプールやNew世代領域に空きがあるように見える場合であっても、実際には空きがない場合があります（空きがあるように見える場合であっても、その差は、NewGC処理用の作業域的な役割で使用済となっている場合があります）。

ガーベジコレクション処理の実行抑止

“ガーベジコレクション処理の実行が抑止される機能”に示す機能を使用するJavaアプリケーションを実行すると、Javaヒープ内に存在する全オブジェクトの移動が禁止されるクリティカルセクションと呼ばれる状態が、当該機能の利用状況に応じて発生します。
クリティカルセクション状態発生時は、オブジェクトの移動が禁止されるため、オブジェクト移動が必須となるGC処理の実行が抑止される状態となります。
Java VMは、Javaアプリケーションにより GC処理の実行が抑止されている際に発生したオブジェクト生成要求に対し、New世代領域に空きが無い場合には、Old世代領域の空き領域を一時的に使用して対応します。
そして、要求されたオブジェクト量を満たす空きがOld世代領域にない場合には、java.lang.OutOfMemoryError例外を発生させます。
そのため“ガーベジコレクション処理の実行が抑止される機能”の機能を多用するJavaアプリケーションの場合は、GC処理実行が抑止される状態も多数発生する可能性が高くなり、当該機能を多用しないJavaアプリケーションに比べ、GC処理実行抑止に因るjava.lang.OutOfMemoryError例外が発生しやすい状態にあります。
特にOld世代領域が小さい状態でJavaアプリケーションを実行した場合は、Old世代領域の空きとなり得る最大値（仮にOld世代領域を全く使用しない場合だと、Old世代領域自身の大きさ）もその大きさに比例して小さいため、その傾向が強まることがあります。

なおFJVMでは、GC処理の実行抑止によりjava.lang.OutOfMemoryError例外が発生したかどうかの情報を出力する機能を提供しています。
詳しくは“10.6.1.1 メモリ領域不足事象発生時のメッセージ出力機能の強化”を参照してください。

ガーベジコレクション処理の実行が抑止される機能

- GC処理の実行が抑止される状態となるJNI関数
  - GetPrimitiveArrayCritical()実行からReleasePrimitiveArrayCritical()実行までの間
  - GetStringCritical()実行からReleaseStringCritical()実行までの間
- GC処理の実行が抑止される状態となるJVMPI関数(注)
  - DisableGC()実行からEnableGC()実行までの間

・ GC処理の実行が抑止される状態となるJVMPIイベント(注)

- JVMPI_EVENT_THREAD_START
- JVMPI_EVENT_CLASS_LOAD
- JVMPI_EVENT_CLASS_UNLOAD
- JVMPI_EVENT_JNI_GLOBALREF_ALLOC
- JVMPI_EVENT_JNI_GLOBALREF_FREE
- JVMPI_EVENT_JNI_WEAK_GLOBALREF_ALLOC
- JVMPI_EVENT_JNI_WEAK_GLOBALREF_FREE
- JVMPI_EVENT_OBJECT_ALLOC
- JVMPI_EVENT_MONITOR_CONTENDED_ENTER
- JVMPI_EVENT_MONITOR_CONTENDED_ENTERED
- JVMPI_EVENT_MONITOR_CONTENDED_EXIT
- JVMPI_EVENT_MONITOR_WAIT
- JVMPI_EVENT_MONITOR_WAITED
- JVMPI_EVENT_HEAP_DUMP
- JVMPI_EVENT_METHOD_ENTRY
- JVMPI_EVENT_METHOD_ENTRY2
- JVMPI_EVENT_METHOD_EXIT

注) Java Virtual Machine Profiling Interface(JVMPI)をサポートしていません。

JVMPIとJVMTI

Java Virtual Machine Profiling Interface(JVMPI)をサポートしていません。

JVMPI相当の機能を使用する場合には、Java Virtual Machine Tool Interface(JVMTI)を使用してください。

RMI処理の分散GC

JavaのRMI処理では、クライアントで不要となった参照に対するサーバ側のオブジェクトを破棄するため、Distributed GC(分散GC)という処理を行います。その処理の一貫として、以下のプロパティで設定された時間間隔(デフォルトの時間間隔は1時間)ごとに、java.lang.System.gc()実行によるFull GCが発生します。なお、分散GCとメモリ不足等による通常のガーベジコレクション(GC)が同時に発生した場合は、メモリ不足等による通常のGCが実行され、分散GCによるFullGCは実行されません（メモリ不足等による通常のGCがNewGC処理だった場合は、FullGCにはなりません）。

```
-Dsun.rmi.dgc.server.gcInterval=時間間隔(ミリ秒)
-Dsun.rmi.dgc.client.gcInterval=時間間隔(ミリ秒)
```

分散GCは独自のタイマー制御の元で実行されるため、メモリ不足等による通常のGCの実行とは関係せずに実行されます。そのため、GC処理の結果ログを見た場合、Javaアプリケーションとしての処理がほとんど動作していない時間帯やメモリ不足とは思われない状態のときに、FullGC処理が実行されているように見える場合があります。
またプロパティで指定された時間間隔が短い場合、Interstage Application Serverの予兆監視機能により警告を受ける(EXTP4368メッセージやOM3204メッセージが出力される)場合がありますので、注意してください。

10.2.2 標準GC（シリアルGC）

New世代領域用GC制御に対して何も付加機能を追加していない「標準機能のみ」で構成されたGC処理です。後述の**パラレルGC**との対比から、標準GCのことを**シリアルGC**と呼ぶ場合もあります。

- **Java HotSpot Client VM**、および以下のオプションを指定した場合の**FJVM**では、**シリアルGC**が実行されます。なお、以下のオプションは、後述の**FJGC**によるGC制御を無効にする、またはシリアルGCによるGC制御を有効にするオプションです。

```
-XX:+UseSerialGC
```

- **シリアルGC**使用時に利用可能となる「Javaヒープおよびメタスペースのチューニング用オプション」です。

```
-Xms
-Xmx
-XX:NewSize
-XX:MaxNewSize
-XX:NewRatio
-XX:SurvivorRatio
-XX:TargetSurvivorRatio
-XX:MetaspaceSize
-XX:MaxMetaspaceSize
```

10.2.3 New世代領域用制御処理並列化機能付きGC（パラレルGC）

New世代領域用GC制御に対し、「当該処理を並列化して実行する機能」を追加して構成されたGC処理です。New世代領域用のGC制御を並列化して実行することから、このGCを**パラレルGC**と呼ぶ場合もあります。

パラレルGCを有効にする場合に指定するオプション

FJVMの場合は、デフォルトでこのGC処理が実行されます。

```
-XX:+UseParallelGC
```

Javaヒープおよびメタスペースのチューニング用オプション（パラレルGC使用時）

パラレルGC使用時に利用可能となる「Javaヒープおよびメタスペースのチューニング用オプション」です。

なお、パラレルGC使用時は、JavaヒープのNew世代領域およびOld世代領域の大きさに関する値が自動的に調整および最適化されるため、通常、New世代領域の大きさやNew世代領域とOld世代領域の大きさのバランスをチューニングするためのオプションを使用する必要はありません。

```
-Xms
-Xmx
-XX:NewSize (*)
-XX:MaxNewSize (*)
-XX:NewRatio (*)
-XX:MetaspaceSize
-XX:MaxMetaSpaceSize
```

(*) New世代領域の大きさや、New世代領域とOld世代領域の大きさのバランスをチューニングするためのオプションです。

GC処理用スレッド数

パラレルGCを使用した場合は、実行するハードウェアに搭載しているCPU数に依存した数のGC処理用スレッドがJavaプロセス内に作成されます。そのため、GC処理用スレッドの数分だけ、スタック域などのスレッド用のメモリ領域が必要となります。

Javaプロセス内でのメモリ量を抑えるためなど、GC処理用スレッドの数を調整する場合には、以下のオプションでGC処理用スレッドの数を指定することにより、GC処理用スレッドの数を調整することができます。

なおGC処理用スレッドの数を抑制した分だけGC処理における性能がおちる場合もありますので、このオプションを用いる場合には、十分な性能確認を実施してください。また、一般的に、CPU数以上の数のGC処理用スレッドを作成しても、GC処理における性能向上にはつながりません。

・ パラレルGCで使用するGC処理用スレッドの数を指定するオプション

```
-XX:ParallelGCThreads=New世代領域GC用スレッドの数
```

New世代領域用GC処理を行うGCスレッドの数を指定します。
「0」が指定された場合は、デフォルト値となります。デフォルト値は以下のとおりです。

ー 実行するハードウェアに搭載しているCPU数が7以下の場合 = CPU数分

ー 実行するハードウェアに搭載しているCPU数が8以上の場合 = 8

メモリ割り当てプールの省略値自動調整機能

FJVMのパラレルGCでは、エルゴノミクス機能によるメモリ割り当てプールの初期値(-Xms)および最大値(-Xmx)の省略値自動調整機能(マシンの物理メモリ量などに応じて、-Xmsおよび-Xmxの各オプションに対する省略値を自動的に決定する機能)を無効にしています。

FJVMで、エルゴノミクス機能によるメモリ割り当てプールの省略値自動調整機能を有効にする場合は、以下のオプションを指定してください。

ただし、このオプション指定は、システムのメモリ資源不足の要因となる場合があるため、システム内に複数のJavaプロセスを起動、実行する場合には使用しないでください。

・ メモリ割り当てプールの省略値自動調整機能を有効にするオプション

```
-XX:+AutomaticallyJavaHeapSizeSetting
```

メモリ領域不足事象の検出機能

FJVMのパラレルGCでは、エルゴノミクス機能によるメモリ領域不足事象の検出機能(以下の各オプション指定値による条件が同時に成立した場合に、メモリ領域不足事象(java.lang.OutOfMemoryError)として検出する機能)を無効にしています。

・ メモリ領域不足事象検出条件

```
-XX:GCTimeLimit=GC処理に要する時間の上限値 (デフォルトは98)
```

Javaアプリケーションの合計処理時間に対して、GC処理に要した時間の上限値をパーセント単位(%)で指定します。
指定された上限値を超えた場合、検出条件の一方が成立します。

```
-XX:GCHeapFreeLimit=GC処理後のJavaヒープ量の空きスペースの下限値 (デフォルトは2)
```

メモリ割り当てプールの最大値に対する、GC処理後のJavaヒープ量の空きスペースの下限値をパーセント単位(%)で指定します。
指定された下限値を下回った場合、検出条件の一方が成立します。

FJVMでエルゴノミクス機能によるメモリ領域不足事象検出機能を有効にする場合は、以下のオプションを指定してください。

ただし、このオプション指定で検出されるメモリ領域不足事象は、Javaヒープの使用量だけではなく、メモリ領域不足事象検出用オプションで指定された値、およびガーベジコレクション処理の動作状況から得られた統計情報などを元に決定されるため、Javaヒープの使用量が不足していない状態であっても、メモリ領域不足事象が検出される場合がありますので注意してください。

・ メモリ領域不足事象検出を有効にするオプション

```
-XX:+UseGCOverheadLimit
```

10.2.4 コンカレント・マーク・スイープGC付きパラレルGC（CMS付きパラレルGC）

New世代領域用GC制御を並列化して実行する機能、およびJavaアプリケーションと同時並列に動作するOld世代領域用GC制御「コンカレント・マーク・スイープGC(CMS-GC)機能」を追加して構成されたGC処理です。CMS-GC機能を追加されたパラレルGC制御であることから、このGCを**CMS付きパラレルGC**と呼ぶ場合もあります。

CMS-GCは、Javaアプリケーションが Full GCによって停止されることにより影響を受ける「アプリケーションの応答性能の平準化」を改善するために実行される「Old世代領域内の不要オブジェクト回収用のGC機構」です。

Javaアプリケーションを停止させて実行するFull GCに対し、CMS-GCはJavaアプリケーションと同時並列に動作し、Old世代領域内の不要オブジェクトを回収します。CMS-GCの実行により、Old世代領域(New世代領域にあるオブジェクトの移動先や巨大オブジェクトの生成先となる領域)の空きを、Javaアプリケーション動作と並行して増加させることができるため、Full GCの発生を抑えることができます。これにより、JavaアプリケーションはFull GCにより停止される影響を受けにくくなり、応答性能平準化の改善が期待できます。

なおCMS-GC動作中は、NewGC処理/Full GC処理の実行開始が遅延する場合があります。そして遅延期間中は、Javaアプリケーションとしての動作も停止します。そのため、ガーベジコレクション処理の結果ログ内のNewGC処理/Full GC処理に対して出力されたGC処理実行時間よりも長い間、Javaアプリケーションとしての動作が停止している場合があります。

CMS付きパラレルGCを有効に場合に指定するオプション

```
-XX:+UseConcMarkSweepGC
```

注意

“-XX:UseFJcmsGC=タイプ”オプションは、JDK/JRE 7では指定可能ですが、JDK/JRE 8では指定できません。

“-XX:UseFJcmsGC=タイプ”に相当するオプションは以下を参照してください。

- 利用者が細かくチューニング作業を行うことが可能なCMS付きパラレルGCを使用する場合
  (JDK/JRE7の-XX:UseFJcmsGC=type0またはtype0p指定に相当)

```
-XX:NewRatio=7
-XX:SurvivorRatio=8
-XX:TargetSurvivorRatio=50
-XX:MaxTenuringThreshold=4
-XX:InitialTenuringThreshold=4
```

  以下は、上記オプション指定時に利用可能となる「Javaヒープおよびメタスペースのチューニング用オプション」です。チューニング用オプションを用いて、利用者が細かくチューニング作業を行うことが可能なCMS付きパラレルGCを使用する場合に指定します。

  Javaヒープおよびメタスペースのチューニング用オプション（利用者が細かくチューニング作業をするオプション指定の場合）

```
-Xms
-Xmx
-XX:NewSize
-XX:MaxNewSize
-XX:SurvivorRatio
-XX:TargetSurvivorRatio
-XX:MetaspaceSize
-XX:MaxMetaSpaceSize
```

- New世代領域用GCでのオブジェクト回収を重視したCMS付きパラレルGCを使用する場合
  (-XX:UseFJcmsGC=type1またはtype1p指定に相当)

```
-XX:NewRatio=2
-XX:SurvivorRatio=8
-XX:TargetSurvivorRatio=90
-XX:MaxTenuringThreshold=8
-XX:InitialTenuringThreshold=4
```

New世代領域用GCでのオブジェクト回収を重視したCMS付きパラレルGCを使用する場合に指定します。

実行されるアプリケーションの特徴が「大半のオブジェクトが、少ない回数のNew世代領域用GCによって回収されるオブジェクト」である場合に、CMS-GCによる応答性能平準化の改善効果が得やすい調整になっています。

Javaヒープおよびメタスペースのチューニングを行う場合は、まず-Xms/-Xmxおよび-XX: MetaspaceSize/-XX: MaxMetaSpaceSizeの各オプションにより、メモリ割り当てプールおよびメタスペースの大きさを調整します。そして必要に応じて、-XX:NewSize/-XX:MaxNewSizeの各オプションでNew世代領域の大きさを調整します。

なおNew世代領域サイズとして、メモリ割り当てプール最大サイズ未満の値を指定できます。ただしNew世代領域サイズを大きくしすぎると、Full GCが発生しやすくなってしまうので注意が必要です。

以下は、New世代領域用GCでのオブジェクト回収を重視したオプション指定時に利用可能となる「Javaヒープおよびメタスペースのチューニング用オプション」です。

Javaヒープおよびメタスペースのチューニング用オプション（New世代領域用GCでのオブジェクト回収を重視したオプション指定時）

```
-Xms
-Xmx
-XX:NewSize
-XX:MaxNewSize
-XX:MetaspaceSize
-XX:MaxMetaSpaceSize
```

- CMS-GCでのオブジェクト回収を重視したCMS付きパラレルGCを使用する場合
(-XX:UseFJcmsGC=type2またはtype2pが指定に相当)

```
-XX:NewRatio=15
-XX:SurvivorRatio=1024
-XX:TargetSurvivorRatio=0
-XX:MaxTenuringThreshold=0
-XX:InitialTenuringThreshold=0
```

CMS-GCでのオブジェクト回収を重視したCMS付きパラレルGCを使用する場合に指定します。

実行されるアプリケーションの特徴が「大半のオブジェクトが、何回かのGCを経てから回収される(長期間常駐せず、ある程度の短期間で回収される)オブジェクト」である場合に、CMS-GCによる応答性能平準化の改善効果が得やすい調整になっています。

Javaヒープおよびメタスペースのチューニングを行う場合は、まず-Xms/-Xmxおよび-XX: MetaspaceSize/-XX: MaxMetaspaceSizeの各オプションにより、メモリ割り当てプールおよびメタスペースの大きさを調整します。そして必要に応じて、-XX:NewSize/-XX:MaxNewSizeの各オプションでNew世代領域の大きさを調整します。

なおNew世代領域サイズとして、メモリ割り当てプール最大サイズ未満の値を指定できます。ただしNew世代領域サイズを大きくしすぎると、Full GCが発生しやすくなってしまうので注意が必要です。

以下は、-XX:UseFJcmsGC=type2およびtype2p指定時に利用可能となる「Javaヒープおよびメタスペースのチューニング用オプション」です。

Javaヒープおよびメタスペースのチューニング用オプション（CMS-GCでのオブジェクト回収を重視したオプション指定時）

```
-Xms
-Xmx
-XX:NewSize
-XX:MaxNewSize
```

```
-XX:MetaspaceSize
-XX:MaxMetaSpaceSize
```

GC処理用スレッド数

CMS付きパラレルGCを使用した場合は、実行するハードウェアに搭載しているCPU数に依存した数のGC処理用スレッドがJavaプロセス内に作成されます。そのため、GC処理用スレッドの数分だけ、スタック域などのスレッド用のメモリ領域が必要となります。

Javaプロセス内でのメモリ量を抑えるためなど、GC処理用スレッドの数を調整する場合には、以下のオプションでGC処理用スレッドの数を指定することにより、GC処理用スレッドの数を調整することができます。

なおGC処理用スレッドの数を抑制した分だけGC処理における性能がおちる場合もありますので、このオプションを用いる場合には、十分な性能確認を実施してください。また、一般的に、CPU数以上の数のGC処理用スレッドを作成しても、GC処理における性能向上にはつながりません。

- CMS付きパラレルGCで使用するGC処理用スレッドの数を指定するオプション

```
-XX:ParallelGCThreads=New世代領域GC用スレッドの数
```

New世代領域用GC処理を行うGCスレッドの数を指定します。

最小値は「2」です。「0」または「1」が指定された場合は、デフォルト値となります。

デフォルト値は以下のとおりです。

- 実行するハードウェアに搭載しているCPU数が1の場合 = 2
- 実行するハードウェアに搭載しているCPU数が2以上7以下の場合 = CPU数分
- 実行するハードウェアに搭載しているCPU数が8以上の場合 = 8

CMS-GC処理用スレッド数

CMS付きパラレルGCを使用した場合は、以下のCMS-GC処理用スレッドがJavaプロセス内に作成されます。そのため、CMS-GC処理用スレッドの数分だけ、スタック域などのスレッド用のメモリ領域が必要となります。

- CMSスレッド (必ず1つ作成されます)
- コンカレント・マーク処理専用スレッド

CMSスレッドの他、CMS-GC処理の中のコンカレント・マーク処理を複数スレッドで並列化して実行するために、コンカレント・マーク処理専用スレッドを追加で作成することができます。以下のオプションでCMS-GC処理用スレッドの数を指定することにより、CMS-GC処理用スレッドの数を調整することができます。

なお一般的に、CPU数以上の数のCMS-GC処理用スレッドを作成しても、CMS-GC処理における性能向上にはつながりません。

- CMS付きパラレルGCで使用するCMS-GC処理用スレッドの数を指定するオプション

```
-XX:ConcGCThreads=CMS-GC処理用スレッドの数
```

コンカレント・マーク処理専用スレッドの数を指定します。

最小値は「2」です。オプションの指定がない場合、または「1」が指定された場合は、CMSスレッドだけ生成されます。

「0」が指定された場合は、自動的に以下の値が設定されます。(実行するハードウェアに搭載しているCPU数の1/4(端数切り上げ)をAとした場合)

- Aが1の場合 = 0(CMSスレッドだけ生成されます)
- Aが2以上7以下の場合 = A
- Aが8以上の場合 = 8

なお指定値および自動設定値どちらの場合も、「-XX:ParallelGCThreads」の値よりも大きい場合は、「-XX:ParallelGCThreads」の値となります。

10.2.5 オブジェクト参照の圧縮機能

Javaアプリケーションを64ビットモードのJDK/JRE上で動作させる場合、通常のC/C++アプリケーションの場合と同様、実行モード上の制約から、Javaヒープに格納されるオブジェクト参照（ポインタ情報）で必要となる領域は「64ビット表現/8バイト域」が管理単位となります。

そのため64ビットモードのJDK/JRE上でJavaアプリケーションを動作させる場合、32ビットモードのJDK/JRE上で動作させる場合に対して、1.5～2倍のJavaヒープ量が必要となります。

しかし64ビットモードで実行されるFJVMの場合は、Javaヒープの大きさ（メモリ割り当てプール）が32GB未満の場合に限り、オブジェクト参照で必要となる領域を「32ビット表現/4バイト域」に圧縮して管理する機能「64ビットモード実行時における**オブジェクト参照圧縮機能**」により、当該機能のない64ビットモードで実行されるJDK/JREよりも少ないJavaヒープ量でJavaアプリケーションが実行できます。

1オブジェクト参照当たりで必要となるJavaヒープの領域が減るため、たとえば極端な例ですが、従来はある大きさに100個のオブジェクト参照を格納できていた場合、当該機能により200個のオブジェクト参照がJavaヒープ上に格納できるようになります。つまり当該機能がないもしくは使用しない場合に必要となるJavaヒープ量を、当該機構を用いた場合に適用した場合、相対的により大きなJavaヒープ値を指定したことと等価となります。その結果、ガーベジコレクション処理の発行回数が減り、アプリケーション実行性能の向上が期待できます。

注意

オブジェクト参照圧縮機能の無効化

本来「64ビット表現/8バイト域」が必要なメモリ域を「32ビット表現/4バイト域」に圧縮して管理するため、オブジェクト参照使用時には、圧縮した情報を「64ビット表現/8バイト域」に展開する必要があります。その圧縮／展開処理は従来よりも余分なCPU消費に繋がるため、Javaアプリケーションとしての実行性能が落ちる可能性が考えられます。

64ビットモードで実行されるFJVMの場合は、デフォルト状態でオブジェクト参照圧縮機能が有効になっています。そのため、Javaアプリケーションとしての実行性能が問題となった場合は、以下のオプションを指定することで、FJVMのオブジェクト参照圧縮機能を無効にすることができます。

- 64ビットモードで実行されるFJVMのオブジェクト参照圧縮機能を無効にするオプション

```
-XX:-UseCompressedOops
```

注) 64ビットモードで実行される場合に指定できます。

なお、Javaヒープの大きさ（メモリ割り当てプール）が32GB以上であった場合は、以下のメッセージを標準出力へ出力して、オブジェクト参照圧縮機能は自動的に無効となります。

Javaヒープの大きさは、ページサイズなどのシステム情報を元に、Java VMによる制御が最適となるように調整された値となるため、-Xmxで指定された値と若干異なる場合があります。このため、このオプションで指定したサイズが、32GBより若干小さい場合でもオブジェクト参照圧縮機能が無効になる場合があります。

- FJVMのオブジェクト参照圧縮機能が自動的に無効となった場合に出力されるメッセージ

```
Java HotSpot(TM) 64-Bit Server VM warning: Max heap size too large for Compressed Oops
```

10.2.6 ガーベジコレクションのログ出力

ガーベジコレクション(GC)のログを採取する場合は、以下のオプションを指定します。

GC処理の結果ログを出力するオプション

```
-verbose:gc
```

本オプションの指定により、**GC**が発生するたびに、GC処理の結果ログが標準出力へ1行ずつ出力されます。

GCログの出力フォーマット

```
[GCの種類 (GCの発生理由) GC前のヒープ使用量->GC後のヒープの使用量(ヒープのサイズ), GCの処理時間]
```

GCの種類:

"GC"の場合は**マイナーGC**(または**NewGC処理**)で、"FullGC"の場合は**FullGC**であることを示します。

なお"Javaヒープの中のメモリ割り当てプール"を"ヒープ"と略記しています。

GCの発生理由:

GCの発生理由を表す文字列(内部制御の情報)が表示されます。

例. System.gc()/Allocation Failure/Ergonomics

例

GCログの出力例

```
[Full GC (System.gc()) 911K->911K(251392K), 0.0126174 secs]
[GC (Allocation Failure) 146967K->100836K(251392K), 0.1611865 secs]
```

より詳細なGC処理の結果ログ情報を得る場合は、"10.2.6.1 ガーベジコレクション処理の結果ログ詳細出力機能"を参照してください。

注意

ログ出力量の増加

本オプションの指定により、ログ出力が増大します。
本オプションを指定する場合は、ログ出力量についての注意が必要です。

New世代領域の使われ方

NewGC処理では、New世代領域を「eden space」、「from space」および「to space」の3つの内部領域に細分割し、当該領域上において、一般に世代別GC制御と言われる制御方法を用いて、Javaアプリケーションが生成要求したオブジェクトを管理・制御しています。
このうち、「from space」および「to space」は、Java VMがNewGC処理を行う際の作業域的な役割を持つ領域となっています。
そのため、「from space」および「to space」の各領域が占める大きさのうち、Javaアプリケーションからのオブジェクト生成要求のために使用される大きさは、その領域の一部分だけとなります。
そのため、ガーベジコレクションのログ出力において、メモリ割り当てプールやNew世代領域に空きがあるように見える場合であっても、実際には空きがない場合があります(空きがあるように見える場合であっても、その差は、NewGC処理用の作業域的な役割で使用済となっている場合があります)。

クラスのアンロード情報

Javaアプリケーションがクラスのアンローディング処理を行っていた場合に、FullGC処理中に該当するクラスのアンロード情報"[Unloading class アンロードされたクラス名]"を結果ログ出力の途中に挿入したい場合は、以下のオプションを指定してください。

- GC処理の結果ログに対してクラスのアンロード情報を出力するオプション

```
-XX:+ClassUnloadingInfo
```

- GC処理の結果ログの格納先ファイルを指定するオプション

```
-Xloggc:GC処理の結果ログの格納先ファイル名
```

GC処理の結果ログおよびクラスのインスタンス情報を、標準出力ではなく、指定したファイルへ出力先を切り換えることができます。

- このオプションを指定すると、自動的に“-verbose:gc”オプションも指定したと見做されます。そのため、“-verbose:gc”オプションの指定がない場合でも、GC処理の結果ログが出力され、また、ガーベジコレクション処理の結果ログ詳細出力機能を使用することもできます。

  なお、“-Xloggc”オプション指定により出力されたGC処理の結果ログの先頭には、「Java VMが起動されてからの経過時間(秒)」が「GC処理の実行開始時間」として自動的に付加されます(以下の形式で出力されます)。

  ```
  GC処理の実行開始時間: GC処理の結果ログ(「GCログの出力フォーマット」と同じ)
  ```

  「GC処理の実行開始時間」のフォーマットは変更できません。

- 格納先ファイル名の指定には、絶対パスや相対パスのディレクトリ名を付加した形式も可能です。
- 格納先ファイル名の中にあるディレクトリが存在しないなど、何らかの理由で指定された格納先ファイルにアクセスできなかった場合、GC処理の結果ログは出力されません。

注意

Interstage Application Server配下でJavaアプリケーションを実行する際にオプションを指定した場合、以下の問題が発生します。

- オプションによるファイル出力処理には、ログローテーションなどの世代管理機能はありません。何らかの理由でJavaプロセスが自動的に再起動した場合、格納先として同じファイルが用いられることになるため、再起動後のGC処理の結果ログで再起動前の結果ログが上書きされ、ログ情報として使えなくなります。
- Javaアプリケーションを複数のプロセスで多重動作させる場合、同一のオプション定義で複数のプロセスが動作します。そのため、複数のプロセスから同じファイルに対してGC処理の結果ログを書き込むことになり、ログ情報として使えなくなります。
- Interstage Application Server配下でJavaアプリケーションを実行する際のログの出力先は、Interstage Application Serverとしての制御により、出力先ファイルが管理されています。オプション指定でGC処理の結果ログが別のファイルとなった場合、エラー発生時に、他のエラー情報とGC処理の結果ログが分離することになるため、エラーの解析が難しくなる場合があります。

そのため、Interstage Application Server配下でJavaアプリケーションを実行する場合には、上記のオプションを指定しないでください。上記のオプションは、Interstage Application Serverとは関係しない単独のJavaアプリケーションを実行する場合に、必要に応じて指定してください。

10.2.6.1 ガーベジコレクション処理の結果ログ詳細出力機能

“-verbose:gc”オプション指定時に出力されるガーベジコレクション(GC)処理の結果ログを、より詳細に出力する機能について説明します。

GC処理の結果ログとして出力される情報を拡張するオプション

```
-XX:+PrintGCDetails
```

上記のオプションを指定することにより、GC処理の結果ログとして出力される情報が、「GC処理の結果ログとして出力される情報の拡張形式」に示す形式に拡張されます。

```
-XX:+PrintGCTimeStamps
```

上記のオプションを指定することにより、GC処理の結果ログにタイムスタンプを付加する形式に拡張されます。

上記2つのオプションを指定した場合に表示されるGC処理の結果ログの形式を以下に示します。

GC処理の結果ログとして出力される情報の拡張形式

```
$1: [$2 ($3) [$4 : $5->$6($7)] [$8 : $9->$10($11)] $12->$13($14), [$15 : $16->$17($18)], $19 secs] [Times: user=$20 sys=$21, real=$22 secs]
```

$1: GC処理の実行開始時間(ログ出力時の時間)

GC処理の実行開始時間(ログ出力時の時間)を示します。
時間は、起動されてからの経過時間(秒)です。

$2: GC種別

実行したGC処理の種別を以下の名称で示します。

- GC

  New世代領域を対象とするGC処理(NewGC処理またはマイナーGC処理)における結果情報です。

- Full GC

  Javaヒープ域全体(メモリ割り当てプール(New世代領域、Old世代領域)およびメタスペース)を対象とするGC処理(FullGC処理)における結果情報です。

- CMS initial-mark

  Old世代領域を対象とするCMS-GC処理のうち、initialマーク処理における結果情報です。
  CMS-GCでは、不要オブジェクトを検出するために行う処理(initialマーク処理)を実行する際、極わずかな時間だけJavaアプリケーションを停止させます。
  注)不要オブジェクトの検出処理だけであるため、GC処理開始前後での各世代領域におけるオブジェクト量に変化はありません。

- CMS remark

  Old世代領域を対象とするCMS-GC処理のうち、finalマーク処理における結果情報です。
  CMS-GCでは、不要オブジェクトを検出するために行う処理(finalマーク処理)を実行する際、極わずかな時間だけJavaアプリケーションを停止させます。
  注)不要オブジェクトの検出処理だけであるため、GC処理開始前後での各世代領域におけるオブジェクト量に変化はありません。

- CMS-concurrent-*phase*-start/CMS-concurrent-*phase*

  CMS-GCスレッドで実行される各CMS-GC処理*phase*の開始と終了における情報です。

  *phase*には以下があります。これらの*phase*は、Javaアプリケーションを停止することなく、CMS-GCスレッドで実行されます。

  - mark
  - preclean
  - sweep
  - reset
  - abortable-preclean

  注意

  CMS付きパラレルGCのログ

  GC種別が"CMS-concurrent-*phase*-start"および"CMS-concurrent-*phase*"のログは、CMS-GCスレッドで実行されたCMS-GCの内部処理のログです(*phase*は、mark/preclean/sweep/reset/abortable-preclean)。このログの対象となるCMS-GC処理は、Javaアプリケーションを停止することなく動作するために通常は意識する必要はありません。このログは、保守サポート向けの情報(CMS-GCの処理ロジックを理解した高度なスキルを必要とする情報です)。このため、*phase*の説明は省略しています。

$3: GCの発生理由

GCの発生理由を表す文字列(内部制御の情報)が表示されます。

例. System.gc()/Allocation Failure/Ergonomics

$4: New世代領域の識別名

New世代領域の識別名を示します。
使用されているGC処理の違いにより、以下の識別名が出力されます。

- DefNew: シリアルGCの場合
- PSYoungGen: パラレルGCの場合
- ParNew: CMS付きパラレルGCの場合

$5: GC処理前のオブジェクト量(New世代領域)

GC処理実行前のNew世代領域に存在したオブジェクトの総量(バイト)です。

$6: GC処理後のオブジェクト量(New世代領域)

GC処理実行後のNew世代領域に存在するオブジェクトの総量(バイト)です。

$7: New世代領域の大きさ

New世代領域の大きさ(バイト)です。
注)使用されているGC処理がシリアルGC、パラレルGCまたはCMS付きパラレルGCの場合は、この大きさに「to space」領域の大きさが含まれません。
(シリアルGC、パラレルGCまたはCMS付きパラレルGCの場合、GC処理はNew世代領域を「eden space」、「from space」および「to space」の3つの内部領域に細分割して制御しています。)

$8: Old世代領域の識別名

Old世代領域の識別名を示します。
使用されているGC処理の違いにより、以下の識別名が出力されます。

- Tenured: シリアルGCの場合
- ParOldGen: パラレルGCの場合
- CMS: CMS付きパラレルGCの場合

$9: GC処理前のオブジェクト量(Old世代領域)

GC処理実行前のOld世代領域に存在したオブジェクトの総量(バイト)です。

$10: GC処理後のオブジェクト量(Old世代領域)

GC処理実行後のOld世代領域に存在するオブジェクトの総量(バイト)です。

$11: Old世代領域の大きさ

Old世代領域の大きさ(バイト)です。

$12: GC処理前のオブジェクト量(メモリ割り当てプール)

GC処理実行前のメモリ割り当てプールに存在したオブジェクトの総量(バイト)です。
$5+$9の値です。

$13: GC処理後のオブジェクト量(メモリ割り当てプール)

GC処理実行後のメモリ割り当てプールに存在するオブジェクトの総量(バイト)です。
$6+$10の値です。

$14: メモリ割り当てプールの大きさ

メモリ割り当てプールの大きさ(バイト)です。
$7+$11の値です。

使用されているGC処理がシリアルGC、パラレルGCまたはCMS付きパラレルGCの場合は、この大きさにNew世代領域の「to space」領域の大きさが含まれません。
(シリアルGC、パラレルGCまたはCMS付きパラレルGCの場合、GC処理はNew世代領域を「eden space」、「from space」および「to space」の3つの内部領域に細分割して制御しています。)

$15: メタスペースの識別名

メタスペースの識別名です。
使用されているGC処理の違いに関わらず、以下の識別名が出力されます。

- Metaspace

$16: GC処理前のオブジェクト量(メタスペース)

GC処理実行前のメタスペースに存在したオブジェクトの総量(バイト)です。

$17: GC処理後のオブジェクト量(メタスペース)

GC処理実行後のメタスペースに存在するオブジェクトの総量(バイト)です

$18: メタスペースの大きさ

メタスペースの大きさ(バイト)です。

$19: GC処理実行時間

GC処理実行に要した時間(秒)です。
注)GC処理は、Javaアプリケーションとしての動作を停止させて実行されます。

$20: CPU時間(ユーザ時間)

GC処理期間中に、GC処理が使用したCPU時間(秒)です。

$21: CPU時間(システム時間)

GC処理期間中に、システム(OS)が使用したCPU時間(秒)です。

$22: 実時間

GC処理にかかった実時間(秒)です。

注意

有効なGC処理

このオプション指定により出力形式が拡張されるのは、使用するGC処理が以下の場合です。

- シリアルGC
- パラレルGC
- CMS付きパラレルGC

ログ出力量の増加

本オプションの指定により、ログ出力が増大します。

本オプションを指定する場合は、ログ出力量についての注意が必要です。

参考

$2、$3、$12、$13、$14、および$19に対する出力情報は、GC処理の結果ログ出力機能として“-verbose:gc”オプションだけを指定した際に出力される情報と対応します。

注意

マイナーGCの場合、Old世代領域の情報[$8 : $9->$10($11)] および、メタスペースの情報[$15 : $16->$17($18)]は出力されません。

ログの見方

GC処理の結果ログとして出力される情報の拡張形式の出力例を示します。

例

GC処理の結果ログとして出力される情報の拡張形式の出力例

```
19.980: [Full GC (Ergonomics) [PSYoungGen: 9664K->0K(164352K)] [ParOldGen: 174043K-
>120589K(201728K)] 183707K->120589K(366080K), [Metaspace: 7696K->7696K(1056768K)], 0.3491370 secs]
[Times: user=0.68 sys=0.01, real=0.35 secs]
```

この出力情報から、以下のことが分かります。

- Java VMが起動されてから19.980秒後にFullGC処理が実行された。
- 使用されているGC処理はパラレルGCである。
- GC処理後のNew世代領域の大きさは164352KBである。
- GC処理により、New世代領域に存在するオブジェクト量は9664KBから0KBになった。
  （不要オブジェクトが削除され、また必要に応じてOld世代領域へ生存オブジェクトが移動した）
- GC処理後のOld世代領域の大きさは201728KBである。
- GC処理により、Old世代領域に存在するオブジェクト量は174043KBから120589KBになった。
  （不要オブジェクトが削除され、また必要に応じてNew世代領域から生存オブジェクトが移動してきた）
- GC処理後のメモリ割り当てプールの大きさは(366080KB)である。
- GC処理により、メモリ割り当てプールに存在するオブジェクト総量は183707KBから120589KBになった。
  （不要オブジェクトが削除された）
- GC処理後のメタスペースの大きさは1056768KBである。
- GC処理により、メタスペースに存在するオブジェクト量は変化していない。
- GC処理に要した時間は0.3491370秒である。
- GC処理に使用したCPU時間のうち、ユーザ時間は0.68秒で、システム時間は0.01秒である。また、実時間は0.35秒である。

10.3 動的コンパイル

動的コンパイルについて説明します。

C/C++やCOBOLなどで作られたプログラムを実行する場合は、各言語に対応したコンパイラによって、プログラムのソースコードを実行対象プラットフォーム上で動作可能な機械命令へ翻訳（後述の動的コンパイルとの対比で静的コンパイルと呼ばれることがあります）し、事前にプラットフォーム依存の実行バイナリを作成する必要があります。

Javaで作られたプログラムを実行する場合は、javacコマンドによって、プログラムのソースコードをJava VMが解釈／実行できる命令「バイトコード」へ変換し、事前にプラットフォーム非依存の実行バイナリである「クラスファイル」を作成する必要があります。

Javaの実行環境であるJava VMが起動された後、Java VMは、実行対象プログラムであるクラスファイルを読み込み、クラスファイルを以下の2つの方法を用いて実行します。

- インタプリタによるバイトコードの実行

  Java VMのインタプリタは、クラスファイル内のバイトコードを1命令ずつ解釈して実行します。
  機械命令の実行に比べ、実行性能が低速になります。

- 動的コンパイルにより、バイトコードを機械命令に翻訳してから実行

  Javaアプリケーションの実行中、Java VMは、クラスファイル内のJavaメソッドに対応するバイトコードを、実行対象プラットフォーム上で動作可能な機械命令へ自動的に翻訳してから実行します。Javaアプリケーション実行中に自動的に行わ

れる翻訳処理であるため、その翻訳処理を**動的コンパイル**と呼びます。
インタプリタによるバイトコードの実行に比べ、高速に実行できます。

なお、動的コンパイル処理は、翻訳時における機械命令の最適化処理で必要となる各Javaメソッドの実行頻度や呼び出し関係などの情報を、Javaアプリケーション実行と同時に行われる各Javaメソッドに関するプロファイリング処理の結果から得ます。そのため、プロファイリング処理によりある程度の情報量が得られるまで何度か再翻訳が繰り返され、次第にJavaアプリケーションの実行状況に合った翻訳結果に最適化されることで、機械命令部分の実行性能が向上する傾向があります。

Java VMが行う動的コンパイル自体は、Javaアプリケーションの実行から見ると、オーバーヘッド部分になります。そのため、インタプリタによる実行性能(低速)、動的コンパイルによるオーバーヘッド、および動的コンパイル結果である機械命令による実行性能(高速)の三者をバランス良く調整することで、Javaアプリケーション全体としての実行性能を良くする必要があります。

Java VMは、実行頻度の高いJavaメソッドを優先的にコンパイルし、あまり使われることのないJavaメソッドはインタプリタ実行のままにすることで、インタプリタ実行、動的コンパイル、および動的コンパイル結果である機械命令実行のバランスを取り、Javaアプリケーション全体としての実行性能を良くする調整を行っています。

なお、実行対象となるJavaアプリケーション内で、各Javaメソッドの実行頻度がどの位になるのかについては、実際にJavaアプリケーションが実行されない限り分かりません。そのため、Java VMはJavaアプリケーション実行と同時に各Javaメソッドに関するプロファイリング処理を行い、その結果を用いて動的コンパイルの対象となるJavaメソッドを決定しています。プロファイリング処理によりある程度の情報量が得られるまで、すなわちJava VM起動直後はインタプリタ実行だけですが、次第にインタプリタ実行と動的コンパイル結果による機械命令実行との混合動作になります。

FJVMでは、動的コンパイルに関する以下の機能を富士通版独自機能として実装しています。

以下は、FJVM固有機能です。

- コンパイラ異常発生時の自動リカバリ機能
- 長時間コンパイル処理の検出機能
- 動的コンパイル発生状況のログ出力機能

10.3.1 コンパイラ異常発生時の自動リカバリ機能

コンパイラ異常発生時の自動リカバリ機能はFJVM固有機能です。

Java VMはJavaアプリケーションとして実行されるJavaメソッドに対して必要に応じて自動的にコンパイル処理を行いますが、コンパイル処理を行っている際にコンパイラ内で何らかの異常が発生すると、当該Javaメソッドに対するコンパイル処理だけでなくJava VMとしての動作も異常状態として停止させてしまう場合があります。

FJVMでは、コンパイラ内で何らかの異常が発生した場合に自動的にリカバリ処理を行い、Java VMとしての動作を継続させる機能を「コンパイラ異常発生時の自動リカバリ機能」として実装しています。

本機能によるリカバリ処理が行われた際にコンパイル対象となっていたJavaメソッドは、以降、コンパイル処理の対象とはなりません。当該Javaメソッドについてはコンパイルされず、インタプリタモードのままJavaアプリケーションとしての実行が継続されます。

リカバリ処理実施後に通知を受け取るオプション

本機能は**FJVM**の内部処理として動作する機能であるため、コンパイラ内で何らかの異常が発生してもリカバリ処理が正常に行われた場合には、外部に対する通知などは何も行いません。リカバリ処理が正常に行われた場合でもコンパイラ内で何らかの異常が発生したことを情報として受け取る場合には、以下のオプションを指定してください。

```
-XX:+PrintCompilerRecoveryMessage
```

リカバリ処理実施後に通知を受け取るオプションを指定した場合の出力形式

```
CompilerRecovery: Information:The compilation was canceled for method method_name

Reason for the cancellation: reason [code:c, addr:xxxxxxxx]
```

リカバリ処理の情報が上記の形式で標準出力に出力されます。

method_name:

コンパイル処理で異常が発生した際にコンパイル対象となっていたJavaメソッドの名前。

reason:

コンパイル処理で発生した異常の原因情報。原因情報として以下の項目があります。

- assert: コンパイル処理で内部処理矛盾を検出した
- error: コンパイル処理で何らかの誤りを検出した
- stack overflow: コンパイル処理でスタックオーバーフローを検出した

c:

異常コード。

xxxxxxxx:

コンパイル処理で異常が発生した際のアドレス。

10.3.2 長時間コンパイル処理の検出機能

コンパイラ異常発生時の自動リカバリ機能はFJVM固有機能です。

Java VMはJavaアプリケーションとして実行されるJavaメソッドに対して必要に応じて自動的にコンパイル処理を行い、通常は極短時間でその処理を完了します。
しかし、コンパイル処理自身や同じJavaプロセス内で動作させている他の処理の障害などによりCPU資源が占有され続けてしまうと、数分を経過してもコンパイル処理が終了しない場合が考えられます。このような状態が継続すると、システム全体に対して悪影響を与える可能性が考えられます。
このため、**FJVM**では、各Javaメソッドのコンパイル処理に要している時間を監視し、コンパイル処理で必要と考えられる程度の時間を経過してもコンパイル処理が終了していない場合には、Javaプロセス内の処理で何らかの問題が発生していると判断し、当該Javaプロセスを強制的に終了させる機能を「**長時間コンパイル処理の検出機能**」として実装しています。

長時間コンパイル処理の検出機能を有効にするオプション

本機能は、以下のオプションでコンパイル処理に対する監視時間(コンパイル処理に要する時間の上限値)を指定した場合に有効となります。ただし、オプション値として「0」を指定した場合には、本機能は有効となりません。

以下のオプションで指定された時間を超過してもコンパイル処理が終了していない場合、本機能はJavaプロセス内の処理で何からの問題が発生していると判断し、当該Javaプロセスを強制的に終了させます。

```
-XX:CompileTimeout=<nn>
```

<nn>

本機能による異常有無の判断条件とされるコンパイル処理に要する時間の上限値(単位:秒)を指定します。デフォルト値は「0」であり、本機能は無効となっています。
なお、本機能による時間監視の最小単位は30秒であるため、その単位での時間誤差があります。

長時間コンパイル処理の検出機能によるJavaプロセス強制終了時の出力メッセージ

本機能によってJavaプロセスを強制終了する場合、**FJVM**は以下のメッセージを標準出力に出力してから終了します。また、本機能によるJavaプロセスの強制終了時にはコアダンプも出力されます。

```
CompilerRecovery: Information: CompilerRecovery got the VM aborted
because the compiler thread(nnnnnnnn) has not completed.
(compiling method: method_name)
```

nnnnnnnn:

コンパイラスレッドの内部識別子

method_name:

本機能によるチェックでJavaプロセス内に異常が検出された際にコンパイル対象となっていたJavaメソッドの名前

注意

長時間コンパイル処理の検出機能に対する注意事項

何らかの要因によりJavaプロセス内のコンパイル処理へCPU資源が十分に割り当てられず、コンパイル処理自体が進んでいない場合でも、コンパイル処理開始から-XX:CompileTimeoutオプションで指定された監視時間を超過した場合には、本機能による当該Javaプロセスの強制終了となります。

このため、当該Javaプロセスを実行するシステムのCPU負荷が高い場合には、コンパイル処理に対してCPU資源が十分に割り当てられず、この結果として本機能による強制終了が発生する可能性が考えられます。

本機能による強制終了が発生した場合には、まず以下の事項を確認してください。

- 当該Javaプロセスを実行しているシステムのCPU資源量は十分か。
- 当該Javaプロセス以外の他のプロセスでCPU資源が占有されていないか。
- “-XX:CompileTimeout=0”を指定した場合に本機能による強制終了が回避され、かつ、当該Javaプロセスが正常に終了する、または未負荷時に正常なアイドル状態に遷移するか。

上記事項に合致する場合は、コンパイル処理に対してCPU資源が十分に割り当てられなかった結果として発生した強制終了と考えられます。
長時間コンパイル処理の検出機能を有効にしてこの状態が発生した場合には、“-XX:CompileTimeout”オプションで指定する監視時間として、より大きな値に設定する形で調整してください。

参考

長時間コンパイル処理の検出機能に対する監視メッセージの出力

以下のオプションを指定した場合、Javaメソッドのコンパイル処理において1分が経過すると、「長時間コンパイル処理の検出機能が出力する監視メッセージ」の形式で監視メッセージが出力されます。
その後は、30秒経過するごとに同じ監視メッセージが出力されます。
なお、本機能による時間監視の最小単位は30秒であるため、その単位での時間誤差があります。

- 長時間コンパイル処理の検出機能の監視メッセージ出力を有効にするオプション

```
-XX:+PrintCompilerRecoveryMessage
```

- 長時間コンパイル処理の検出機能が出力する監視メッセージ

```
CompilerRecovery: Information: The compiler thread(0xnnnnnnnn) might not return from compiling method method_name.
```

nnnnnnnn:

コンパイラスレッドの内部識別子

method_name:

本機能によるチェックで検出された際にコンパイル対象となっていたJavaメソッドの名前

10.3.3 動的コンパイル発生状況のログ出力機能

Java VMは、Javaアプリケーションとして実行されるJavaメソッドに対して、必要に応じて自動的にコンパイル処理を行います(動的コンパイル)。

動的コンパイル発生状況のログ出力機能では、以下の情報を出力します。

動的コンパイル結果情報

どのJavaメソッドが、いつコンパイルされたかの情報を出力します。
Javaメソッドのコンパイルが、短い間に連続して発生している場合、Javaアプリケーションの実行性能に動的コンパイルが影響を与えている可能性があります。

コンパイラスレッドの動的コンパイル結果情報を出力するオプション

コンパイラスレッドのCPU使用状況および動的コンパイル結果情報を出力する場合に指定します。

```
-XX:+PrintCompilation
```

オプションの指定に関しては、以下のURLを参照してください。

- Windows32/64

  ```
  http://docs.oracle.com/javase/8/docs/technotes/tools/windows/java.html
  ```

- Solaris64 Linux32/64

  ```
  http://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html
  ```

上記のオプションの指定により、動的コンパイルが発生するたびに、その発生状況ログが標準出力へ、以下に示す形式で出力されます。

動的コンパイル結果情報の出力形式

```
$1: $2 $3 $4 $5 ($6 bytes) $7
```

$1: Javaメソッドのコンパイル要求発生時間(ログ出力時の時間)

Javaメソッドのコンパイル要求が発生した時間(ログ出力時の時間)を示します。
時間は、Java VMが起動されてからの経過時間(ミリ秒)です。

$2: 通し番号

コンパイル要求の数(コンパイル要求が発生したJavaメソッドの数)を通し番号で示します。

$3: 空白またはメソッドのコンパイル種別

空白、または、メソッドのコンパイル種別を表す5桁の文字列"%s!bn"を示します。5桁の文字は、それぞれ、動的コンパイラが管理するメソッドの種別を表し、該当しない場合は空白が表示されます。

注意

メソッドのコンパイル種別

メソッドのコンパイル種別を表す5桁の文字列は、動的コンパイラの内部処理を解析するための、保守サポート向けの情報(動的コンパイラ処理ロジックを理解した高度なスキルを必要とする情報です)。このため、種別の説明は省略しています。

$4:空白またはメソッドのコンパイルレベル

動的コンパイラが管理するメソッドのコンパイルレベルを表す数値を示します。
Client VMの場合は、空白が表示されます。

$5: Javaメソッド名

コンパイル要求が発生したJavaメソッドの名前を示します。
Javaメソッドを部分的にコンパイルする要求の場合($2に「%」の表示が含まれる場合)、Javaメソッド名の後に、Javaメソッド(バイトコード)のどの部分からコンパイルの対象になっているかを示す情報「(@ 数字)」が付加されます。

$6: Javaメソッドのバイト数または(native)

コンパイル対象となったJavaメソッドの大きさ(バイトコード・サイズ)をバイト数で示します。
ネイティブメソッドの場合は、(native)が表示されます。

$7: 空白または(static)または made not entrant

コンパイル対象となったJavaメソッドが、static修飾子を持つネイティブメソッドだった場合は、“(static)”と出力されます。コンパイル対象となったJavaメソッドが、ネイティブメソッドでない場合、またはstatic修飾子を持たないネイティブメソッドだった場合、この位置には何も出力されません。

“made not entrant”は、これまでにコンパイラによってつくられた機械命令のコードを無効にしたことを示します。

例

-XX:+PrintCompilation指定時の出力例

```
263 1 3 java.lang.String::hashCode (55 bytes)
266 2 3 java.lang.String::equals (81 bytes)
272 3 n 0 java.lang.System::arraycopy (native) (static)
273 5 4 java.lang.String::charAt (29 bytes)
275 6 4 java.lang.String::length (6 bytes)
276 7 3 java.lang.Character::toLowerCase (9 bytes)
277 8 3 java.lang.CharacterData::of (120 bytes)
278 9 3 java.lang.CharacterDataLatin1::toLowerCase (39 bytes))
278 10 3 java.lang.CharacterDataLatin1::getProperties (11 bytes)
(中略)
1081 108 % 3 java.math.BigInteger::multiplyToLen @ 135 (216 bytes)
1081 108 % 3 java.math.BigInteger::multiplyToLen @ 135 (216 bytes)
```

10.4 チューニング方法

チューニングのポイントとして、次があります。

- メモリ消費量と処理速度は深い関係にあり、メモリ消費量を抑制すれば処理速度が低下するのが一般的です。しかし、JDK/JREにおいては、Javaヒープのサイズを必要以上に大きく確保した場合、New GCの発生頻度が少なくなる反面、FullGCの処理時間が増大し、処理速度が低下するという特徴があります。
- プロセスに割り当てられたメモリ資源は限られています。JDK/JREにおいては、スタック、Javaヒープおよびネイティブモジュールの動作に必要な領域などの各セグメントがユーザ空間に割り当てられます。このため、あるセグメントの領域を大きく確保すれば、その分だけ他のセグメントの領域が少なくなります。

上記のポイントを踏まえた上で、JDK/JRE 8のチューニングを行います。

10.4.1 Javaヒープおよびメタスペースのチューニング

Javaヒープおよび**メタスペース**のチューニング方法および、チューニングによる影響範囲を説明します。

チューニング方法

Javaヒープおよび**メタスペース**の各領域のサイズは、“表10.3 Javaヒープおよびメタスペースに関するオプション”に示すオプションをJava起動時に指定することで設定ができます。

メモリ割り当てプールのデフォルトの初期値および最大値を、“表10.4 メモリ割り当てプールのデフォルトのサイズ”に示します。
また、初めて超えたときにFullGCを発生させるしきい値となる**メタスペース**のサイズを、“表10.5 初めて超えたときにFullGCを発生させるしきい値となるメタスペースのサイズ”に示します。
また各オプションにおいて、“表10.3 Javaヒープおよびメタスペースに関するオプション”中に記載されていないデフォルト値を、“表10.6 Javaヒープに関するオプション(-XX:NewSize/-XX:NewRatio)のデフォルト値”に示します。

表10.3 Javaヒープおよびメタスペースに関するオプション

オプション	オプションの機能 (注1)
-Xms	**メモリ割り当てプール**の初期値を指定します。 たとえば、**メモリ割り当てプール**の初期値を128MBに設

オプション	オプションの機能 (注1)
	定する場合、“-Xms128m”と指定します。 本オプションのデフォルト値は“表10.4 メモリ割り当てプールのデフォルトのサイズ”のとおりです。 なお、指定値が1MB未満の場合、または-XX:NewSizeオプションの値(デフォルト値を含む)以下の場合、初期化エラーとなり、Javaプロセスは終了します。
-Xmx	**メモリ割り当てプール**の最大値を指定します。 たとえば、**メモリ割り当てプール**の最大値を256MBに設定する場合、“-Xmx256m”と指定します。 (実際に使用される値は、ページサイズなどのシステム情報を元に、Java VMによる制御が最適となるように調整された値「調整値」となるため、指定された値と若干異なる場合があります。) 本オプションのデフォルト値は“表10.4 メモリ割り当てプールのデフォルトのサイズ”のとおりです。 なお、指定値(または調整値)が、-Xmsオプションで指定された値よりも小さな値の場合、初期化エラーとなり、Javaプロセスは終了します。
-XX:NewSize	**New世代領域**のヒープサイズを指定します。 たとえば、**New世代領域**のヒープサイズを128MBに設定する場合、“-XX:NewSize=128m”と指定します。 本オプションのデフォルト値は“表10.6 Javaヒープに関するオプション(-XX:NewSize/-XX:NewRatio)のデフォルト値”のとおりです。 なお、指定値がJava VMとしての下限値未満の場合、初期化エラーとなり、Javaプロセスは終了します。そのため、本オプションを指定する場合は、1MB以上の値を指定してください。
-XX:MaxNewSize	**New世代領域**の最大ヒープサイズを設定します。 たとえば、**New世代領域**の最大ヒープサイズを128MBに設定する場合、“-XX:MaxNewSize=128m”と指定します。 本オプションのデフォルト値はありません。 なお、指定値が、-XX:NewSizeオプションで指定された値よりも小さな値の場合は、-XX:NewSizeオプションで指定された値となります。
-XX:NewRatio	**New世代領域**と**Old世代領域**のサイズ比率を指定します。 たとえば、**New世代領域**と**Old世代領域**のサイズ比率を2とする場合、“-XX:NewRatio=2”と指定します。 本オプションの指定が有効な場合のデフォルト値は“表10.6 Javaヒープに関するオプション(-XX:NewSize/-XX:NewRatio)のデフォルト値”のとおりです。
-XX:SurvivorRatio (注2)	**New世代領域**を構成する**Eden領域**と**Survivor領域**のサイズ比率を指定します。 たとえば、**Eden領域**と**Survivor領域**のサイズ比率を8とする場合、“-XX:SurvivorRatio=8”と指定します。 本オプションの指定が有効な場合のデフォルト値は8です。
-XX:TargetSurvivorRatio (注2)	**ガーベジコレクション(GC)**処理後の生存オブジェクトが**Survivor領域**を占める割合を、指定したパーセンテージ値に調整します。 たとえば、GC処理後の生存オブジェクトが**Survivor領域**

オプション	オプションの機能 (注1)
	を占める割合を半分とする場合、“-XX:TargetSurvivorRatio=50”と指定します。 本オプションの指定が有効な場合のデフォルト値は50です。
-XX:MetaspaceSize	初めて超えたときにFullGCを発生させるしきい値となる**メタスペース**のサイズを指定します。 たとえば、値を32MBに設定する場合、“-XX:MetaspaceSize=32m”と指定します。 指定できる範囲は以下になります。 ・ 32ビットモードのJDK/JRE 8の場合 256KB以上かつ、4GB未満 ・ 64ビットモードのJDK/JRE 8の場合 256KB以上かつ、8,589,934,592GB未満 本オプションのデフォルト値は“表10.5 初めて超えたときにFullGCを発生させるしきい値となるメタスペースのサイズ”のとおりです。 なお、指定値(または調整値)が、-XX:MaxMetaspaceSizeオプションで指定された値よりも大きな値の場合は、-XX:MaxMetaspaceSizeオプションで指定された値となります。
-XX:MaxMetaspaceSize	**メタスペース**の最大値を指定します。 たとえば、**メタスペース**の最大値を128MBに設定する場合、“-XX:MaxMetaspaceSize=128m”と指定します。 指定できる範囲は以下になります。 ・ 32ビットモードのJDK/JRE 8の場合 256KB以上かつ、4GB未満 ・ 64ビットモードのJDK/JRE 8の場合 256KB以上かつ、8,589,934,592GB未満 (実際に使用される値は、ページサイズなどのシステム情報を元に、Java VMによる制御が最適となるように調整された値「調整値」となるため、指定された値と若干異なる場合があります。) オプションが指定されなかった場合は無制限(OSが領域を獲得できれば上限なし)に設定されます。
-XX:CompressedClassSpaceSize	64ビットモードのJDK/JRE 8 の場合に、**Compressed Class Space** の最大値を指定します。 たとえば、値を512MBに設定する場合、“-XX:CompressedClassSpaceSize=512m”と指定します。 デフォルトサイズは、1GBです。

注1)

サイズを指定するオプションでは単位として次の文字を指定できます。
KB(キロバイト)を指定する場合: “k”または“K”
MB(メガバイト)を指定する場合: “m”または“M”
GB(ギガバイト)を指定する場合: “g”または“G”

注2)

パラレルGCを使用する場合、このオプションへの指定値は無効となります。

表10.4 メモリ割り当てプールのデフォルトのサイズ

OS	JDK/JREの実行モード	GC処理	初期値	最大値
Windows Linux for x86 Linux for Intel64	32ビットモード	シリアルGC	6.0MB	96MB
		パラレルGC (注1)	6.0MB	
		CMS付きパラレルGC	6.0MB	
Windows Server(R) x64 Editions Linux for Intel64	64ビットモード	シリアルGC	8.0MB	126MB
		パラレルGC (注1)	8.0MB	
		CMS付きパラレルGC	8.0MB	
Solaris	64ビットモード	シリアルGC	8.0MB	128MB
		パラレルGC (注1)	8.0MB	
		CMS付きパラレルGC	8.0MB	

注1)

デフォルトで使用されるGC処理です。

表10.5 初めて超えたときにFullGCを発生させるしきい値となるメタスペースのサイズ

OS	JDK/JREの実行モード	Java VM	FullGCのしきい値
Windows Linux for x86 Linux for Intel64	32ビットモード	Java HotSpot Client VM	12MB
		FJVM (注1)	16MB
Windows Server(R) x64 Editions Linux for Intel64 Solaris	64ビットモード	FJVM (注1)	20.796875MB

注1)

デフォルトで使用されるJava VMです。

注意

64ビットモードのJDK/JRE 8では、通常のメタスペースの他に、クラス情報の一部を格納する**Compressed Class Space**と呼ばれる領域を使用します。この領域は、Java VM起動時に1GBのサイズだけリザーブされます。

表10.6 Javaヒープに関するオプション(-XX:NewSize/-XX:NewRatio)のデフォルト値

OS	JDK/JREの実行モード	Java VM	-XX:NewSize	-XX:NewRatio
Windows Linux for x86 Linux for Intel64	32ビットモード	Java HotSpot Client VM	2048KB	2
		FJVM (注1)		

OS	JDK/JREの実行モード	Java VM	-XX:NewSize	-XX:NewRatio
Windows Server(R) x64 Editions Linux for Intel64 Solaris	64ビットモード	FJVM (注1)	パラレルGCの場合: 2560KB シリアルGC、CMS付きパラレルGCの場合: 2688KB	

注1)

デフォルトで使用されるJava VMです。

チューニングの方針

Javaヒープをチューニングする際、次の方針があります。

1. **FullGC**を実行したにもかかわらず、メモリ不足が発生する場合、**GC**のログを採取し、メモリ割り当てプールまたは**メタスペース**のどちらかの領域が不足しているかどうかを確認します。
2. **FullGC**はコストがかかります。このため、メモリ不足が発生しなくても、Javaアプリケーションがハングアップしたかのように一時的に無反応になる場合、**FullGC**の影響を受けている場合があります。**GC**のログを採取し、**Javaヒープ**が必要以上に大きなサイズになっていれば、**Javaヒープ**のサイズを縮小する方針でチューニングします。
3. 効率的な**New GC**に対して、**FullGC**はコストがかかります。このため、**New世代領域**と**Old世代領域**のサイズのバランスを考慮する必要があります。なお、GC処理としてパラレルGCを使用する場合は、JavaヒープのNew世代領域およびOld世代領域の大きさに関する値が自動的に調整および最適化されるため、通常はこのバランスを考慮する必要はありません。
4. 仮想メモリに余裕がある場合は、Javaプロセスを複数起動して、プロセス多重度を上げる方法を検討します。プロセス多重度を上げることにより、プロセスごとのユーザ空間を有効に使うことが可能になります。ただし、メモリのスワッピングによるスローダウンに注意する必要があります。

チューニングの影響範囲

メモリ割り当てプールの大きさ(-Xmxオプションの指定値)や、New世代領域とOld世代領域の大きさのバランスを変更した場合の影響範囲を、次に示します。

- メモリ割り当てプールの大きさを縮小した場合、**GC**が頻発することがあります。
- メモリ割り当てプールの大きさを拡張した場合、**FullGC**に時間がかかることがあります。
- メモリ割り当てプールの大きさを拡張した場合、その分ユーザ空間や仮想メモリが少なくなるため、スタックやネイティブモジュールの動作に必要な領域を確保できず、メモリ不足になることがあります。
- New世代領域の大きさがメモリ割り当てプールの大きさの半分となるようなチューニングを行った場合、一般的に、FullGCが発生しやすくなる傾向があります。なお、Javaアプリケーション実行時のオブジェクト生成／解放などの特性に依存しますので、どのアプリケーションの場合でもFullGCが発生しやすいというわけではありません。

overcommit memory機能が有効な場合の注意事項 Linux32/64

「overcommit memory機能」が有効な場合、Linuxは、Javaヒープの各領域の最大値に相当する仮想メモリ資源を、Java VMの起動時に、Javaプロセスに対して予約します。
このため、-Xms値と-Xmx値を異なる値にしてJavaプロセスを起動する場合、本機能の有効/無効によって、Javaプロセス起動時にJavaヒープとして必要となる仮想メモリの量が異なります。

- overcommit memory機能が無効、またはovercommit memory機能がないシステムの場合

  Javaヒープ用仮想メモリ量 ＝「-Xms値」

- overcommit memory機能が有効なシステムの場合

  Javaヒープ用仮想メモリ量 ＝「-Xmx値」

この結果、仮に同量の仮想メモリ資源を持つシステムの場合であっても、本機能の有効/無効によって、同時に起動できるJavaプロセスの数が異なる場合があります。
Linuxで仮想メモリ資源の見積もりを行う場合には、overcommit memory機能の有無に注意してください。

10.4.2 スタックのチューニング

Javaアプリケーションで使用するスレッドの**スタック**のチューニング方法および、チューニングによる影響範囲を説明します。

チューニング方法

Java APIで生成するスレッドのスタックサイズは、“-Xss”オプションで指定することができます。

“-Xss”オプションは、バイト単位でスタックサイズを指定します。例えば、スタックサイズを512KBに設定する場合、“-Xss512k”と指定します。

またJDK/JRE内で実行されるJavaメソッドを自動的にコンパイルする専用スレッド(コンパイラスレッド)のスタックサイズは、“-XX:CompilerThreadStackSize”オプションで指定することができます。

通常、コンパイラスレッドのスタックサイズを指定する必要はありません。

“-XX:CompilerThreadStackSize”オプションは、キロバイト(Kバイト)単位でコンパイラスレッドのスタックサイズを指定します。例えば、スタックサイズを1024KBに設定する場合、”-XX:CompilerThreadStackSize=1024”と指定します。

Java APIで生成したスレッドおよびコンパイラスレッドのデフォルトのスタックサイズを、“表10.7 Java APIで生成したスレッドおよびコンパイラスレッドのデフォルトのスタックサイズ”に示します。

なお、スタック領域の実際の管理はOSが行います。そのためスタック領域に関する管理方法/動作仕様については、JDK/JREを実行する各OSの仕様に依存します。

表10.7 Java APIで生成したスレッドおよびコンパイラスレッドのデフォルトのスタックサイズ

OS	JDK/JREの実行モード	Java APIで生成したスレッド(注1)	コンパイラスレッド (Client VM)(注1)	コンパイラスレッド (FJVM)
Windows	32ビットモード	320KB	320KB	1024KB
Linux for x86 Linux for Intel64	32ビットモード	320KB	512KB	1024KB
Windows Server(R) x64 Editions Linux for Intel64 Solaris	64ビットモード	1024KB	(注2)	2048KB

注1)

Windows版JDK/JREにおけるスタックサイズは、java.exeなどWindows版JDK/JREが提供するJDKツールを用いた場合の値です。JNIを用いて独自にJava VMを起動しているWindowsアプリケーションの場合は、Java VMを起動したプログラムのメインスレッドに対するスタックサイズと同じ値になります。

注2)

実行モードが64ビットモードのJDK/JREでは、Java HotSpot Client VMは搭載していません。

チューニングの影響範囲

スタックのサイズを変更した場合の影響範囲を、次に示します。

- **スタック**のサイズを縮小した場合、スタックオーバーフローが発生することがあります。
- **スタック**のサイズを拡張した場合、その分ユーザ空間や仮想メモリが少なくなるため、Javaヒープやネイティブモジュールの動作に必要な領域を確保できず、メモリ不足になることがあります。

10.4.3 暖機運転

Javaアプリケーション実行において、動的コンパイルから受ける影響の確認方法、およびその対応方法である暖機運転について説明します。

チューニング方法

“10.3 動的コンパイル”で説明したように、Javaアプリケーションの実行は、Java VM起動直後はインタプリタ実行だけで行われ、次第にインタプリタ実行と動的コンパイル結果による機械命令実行との混合動作になります。また動的コンパイルにより翻訳された機械命令の内容も、Javaアプリケーションの実行が進むにつれて、次第にJavaアプリケーションの実行状況に合った翻訳結果に最適化されます。つまりJavaアプリケーションの実行には、以下の動的コンパイルによるオーバーヘッドや最適化状態の推移があるため、Javaアプリケーションとして安定した実行性能になるまで、プロセス起動時からある程度の時間を必要とする場合があります。

- 実行対象プログラムであるクラスファイルを実行時に読み込むため、Javaアプリケーションとしての起動直後には、クラスファイルのロードおよび内容検査によるオーバーヘッドが発生します。
- 実行時に機械命令への翻訳処理を行うため、そのオーバーヘッドが発生します。
- 機械命令への翻訳・最適化処理で必要とする情報を、Javaアプリケーション実行と同時に収集するため、Javaアプリケーション開始直後は最適化状態の推移が少なく、結果として実行性能が遅い機械命令となっている場合があります。

なお、Javaアプリケーションとして安定した実行性能になるまでに掛かる時間は、各アプリケーションによって異なります。

動的コンパイル発生状況の調査

Javaアプリケーション実行時における動的コンパイルから受ける影響の有無は、Javaアプリケーションによる業務が開始された際に、動的コンパイルの発生頻度が高いかどうかを判断することで行います。

具体的には、“10.3.3 動的コンパイル発生状況のログ出力機能”を用いて動的コンパイル結果情報を出力し、その結果から以下の傾向が見て取れ、かつJavaアプリケーションとして安定した実行性能になるまでに掛かる時間との関連性が見て取れるかどうかを元に判断します。

- 動的コンパイル結果情報において、Javaメソッドのコンパイルが、短い間に連続して発生している場合

注意

Javaアプリケーション起動時や、業務を開始してJavaアプリケーションへの入力が始まる時に、動的コンパイル処理が発生する頻度が高くなる傾向は、Javaアプリケーション実行時の一般的な傾向です。そのため、動的コンパイル発生状況の調査の結果として、「動的コンパイルの発生頻度が高い」などの傾向が見て取れる場合であっても、インタプリタ実行と機械命令による実行がバランス良く行われており、対応が不要な場合が多々あります。

動的コンパイル発生状況の調査は、Javaアプリケーションとして安定した実行性能になるまでに掛かる時間が実運用上の問題となった場合に行ってください。

注意

Javaアプリケーション起動直後の性能に影響を与える処理は、動的コンパイル処理だけではありません。Javaアプリケーション自体の初期化処理や関連するアプリケーションの起動待ちなど、Javaアプリケーションの起動直後でだけ動作する動的コンパイル以外の要因についても考慮する必要があります。

注意

ServletやEJBなどのJavaアプリケーションがInterstage Application Server配下で動作する場合、「Javaアプリケーションの起動(開始)」には以下の2つの意味があります。

a. Interstage Application Serverの起動(J2EE環境のワークユニット起動、Java EE 7環境のIJServerクラスタ起動)

b. 業務アプリケーションの運用開始

動的コンパイル処理が行われる頻度が高まるのは(a)の直後と(b)の直後の両方ですが、業務としての影響を確認することが目的であるため、動的コンパイル発生状況の調査対象は(b)の状況になります。

なお、動的コンパイル結果情報(-XX:+ PrintCompilationオプション指定時の出力結果)から、(b)の始まりを調べるには、システムログなどから(a)の完了時刻を調べ、その時刻以降に発生した動的コンパイルを対象に調査を行います。

注意

JavaアプリケーションがJSPで作成されている場合、アプリケーションを配備する際にJSPのプリコンパイル(javacコマンドによりJavaソースをクラスファイルへ変換する処理)が実施されていないと、Javaアプリケーション起動時にjavacコマンド実行によるオーバヘッドが発生する場合があります。

JSPのプリコンパイル運用ができる場合は、JSPのプリコンパイルを実施することで、Javaアプリケーションとして安定した実行性能になるまでに掛かる時間が小さくなるかどうかを確認してください。

暖機運転

Javaアプリケーションを起動した後、業務としての運用を開始する時点で安定した実行性能を得る状態にするためには、暖機運転という運用を行います。
暖機運転とは、Javaアプリケーションを起動した後、実際の業務を開始する前に、業務と同様のダミーデータを用いて疑似実行させることで、事前に主なJavaメソッドを動作させ、クラスファイルのロードやJavaメソッドの動的コンパイルを完了させておくことを言います。

暖機運転により、動的コンパイルなどのオーバーヘッドを減らすことができ、また機械命令への翻訳・最適化処理もその時点で行われるようになるため、業務としての運用開始時点から安定した実行性能を得ることができるようになります。

なお暖機運転をどの程度の期間行ったら良いのかについては、Javaアプリケーションの実行性能が運用要件を満たすところまで安定しているかのかどうかが、判断の基準になります。ただし、Javaアプリケーションを起動してから業務開始までの時間は無限にあるわけではないので、“10.3.3 動的コンパイル発生状況のログ出力機能”の動的コンパイル結果情報の出力結果を参考に、暖気運用に掛ける時間をある程度のところで区切る判断をする必要があります。

注意

通常、Javaアプリケーションの実行には、実行結果の表示、データの更新などの結果を伴います。暖機運転による実行結果が、Javaアプリケーションの運用自体に影響を与えないように注意する必要があります。

例えば、暖機運転用のダミーデータをそのままデータベースに登録してしまい、運用時に誤ったデータを返却するなどの問題が発生しないように注意してください。

注意

Javaアプリケーション自体の初期化処理や関連するアプリケーションの起動待ちなど、動的コンパイル以外の要因でJavaアプリケーションとして安定した実行性能になるまでに時間が掛かっている場合は、暖機運転では対処することができません。

10.5 チューニング/デバッグ技法

チューニング技法およびデバッグ技法を紹介します。

10.5.1 スタックトレース

Javaアプリケーションで例外(java.lang.Throwableのインスタンス)がスローされた場合などに出力される**スタックトレース**は、エラーが発生するまでの経緯(メソッドの呼び出し順番)が示されています。このスタックトレースを解析することにより、エラーが発生した箇所と原因を確認することができます。

スタックトレースの出力先

スタックトレースの出力先は、標準エラーです。通常のJavaアプリケーションの場合は、コンソールに出力されますが、Servlet/JSP/EJBアプリケーションの場合は、ログファイル（標準出力、標準エラー出力あるいはJava VMの出力を格納するファイルなど）に出力されます。

スタックトレースの出力方法

Javaでスローされた例外をcatch節でキャッチし、例外のprintStackTraceメソッドを実行することにより、**スタックトレース**を出力することができます。

java.lang.Throwable.printStackTrace()メソッドでスタックトレースを出力する方法

```
try {
    SampleBMPSessionRemote bmpSessionRemote = bmpSessionHome.create();
} catch(Exception e) {
    e.printStackTrace();
}
```

なお、スローされた例外をtry-catch構文で処理するメソッドがスレッドにない場合、そのスレッドは停止され、Java VMによって**スタックトレース**が出力されます。

スタックトレースの出力フォーマット

スタックトレースの出力フォーマット

```
例外クラス名: エラーメッセージ
  at クラス名.メソッド名1(ソース名:行番号)     呼び出し先
  at クラス名.メソッド名2(ソース名:行番号)
              :
              :
  at クラス名.メソッド名N(ソース名:行番号)     呼び出し元
```

- 最初の1行目は、スローされた例外のクラス名とエラーメッセージです。
  エラーメッセージがない場合もあります。
- 2行目以降は、メソッドの呼び出し元(クラス名.メソッド名N)から呼び出し先(クラス名.メソッド名1)に向かって下から上に出力されます。
  2行目(クラス名.メソッド名1)が、例外をスローしたメソッドの情報です。
- “メソッド名”が“<init>”の場合、コンストラクタを示します。
- “メソッド名”が“<clinit>”の場合、static initializerを示します。
- “(ソース名:行番号)”が“(Native Method)”の場合、Javaのネイティブメソッド(.soや.dllファイル)を示します。
- クラスのコンパイル時にデバッグ情報を削除した場合、“(ソース名:行番号)”には、ソース名しか表示されなかったり、“Unknown Source”と表示されたりする場合があります。

10.5.1.1 スタックトレースの解析方法(その1)

以下の出力例をもとにして、解析方法を説明します。
先頭の“数字:”は、説明の便宜上、付加しています。

```
1:java.lang.NullPointerException
2:  at agency.attestation.CheckLoginInfo.doCheck(CheckLoginInfo.java:150)
3:  at agency.attestation.AttestationServlet.doGet(AttestationServlet.java:96)
4:  at agency.attestation.AttestationServlet.doPost(AttestationServlet.java:161)
5:  at javax.servlet.http.HttpServlet.service(HttpServlet.java:772)
6:  at javax.servlet.http.HttpServlet.service(HttpServlet.java:865)
              :
```

読み方

スタックトレースは、6行目から上方向に読むと、次の流れで例外が発生したことがわかります。

1. javax.servlet.http.HttpServlet.service()が、HttpServlet.javaの865行目で、javax.servlet.http.HttpServlet.service()を実行し、
2. javax.servlet.http.HttpServlet.service() が、HttpServlet.java の 772 行目で、agency.attestation.AttestationServlet.doPost()を実行し、
3. agency.attestation.AttestationServlet.doPost() が、AttestationServlet.java の 161 行目で、agency.attestation.AttestationServlet.doGet()を実行し、
4. agency.attestation.AttestationServlet.doGet() が、AttestationServlet.java の 96 行目で、agency.attestation.CheckLoginInfo.doCheck()を実行した結果、
5. agency.attestation.CheckLoginInfo.doCheck() 内の CheckLoginInfo.java の 150 行目で、java.lang.NullPointerExceptionという例外が発生した

解析方法

スタックトレースの解析例を、次に示します。

1. 1行目の例外情報から、原因を特定できるかどうか確認します。
   NullPointerExceptionがスローされていることがわかります。
2. 2行目のCheckLoginInfo.javaの開発担当者であれば、150行目の実装に問題がないかどうかを確認します。
3. 2行目のCheckLoginInfo.javaの開発担当者でない場合、**スタックトレース**中で最上行にある開発担当者が開発したクラスを探します。そして、そのクラスの実装に問題がないかどうかを確認します。それでも、原因を特定できない場合は、開発したクラスが使用しているクラスの提供元に調査を依頼します。

または、**スタックトレース**が、想定された流れでメソッドを実行しているかどうかを確認するのも1つの方法です。

10.5.1.2 スタックトレースの解析方法(その2)

以下の出力例をもとにして、解析方法を説明します。
先頭の“数字:”は、説明の便宜上、付加しています。

```
1:java.util.MissingResourceException: Can't find bundle for base name sample.SampleResource, locale ja_JP
2:  at java.util.ResourceBundle.throwMissingResourceException(Unknown Source)
3:  at java.util.ResourceBundle.getBundleImpl(Unknown Source)
4:  at java.util.ResourceBundle.getBundle(Unknown Source)
5:  at sample.SampleMessage.getMessage(SampleMessage.java:15)
6:  at sample.SampleServlet.doGet(SampleServlet.java:10)
7:  at javax.servlet.http.HttpServlet.service(HttpServlet.java:696)
8:  at javax.servlet.http.HttpServlet.service(HttpServlet.java:809)
    :
    :
```

解析方法

スタックトレースの解析例を、次に示します。

1. 1行目の例外情報から、原因を特定できないかを確認します。

   APIリファレンスによると、java.util.MissingResourceExceptionは、Javaのリソースがない場合に発生する例外です。また、エラーメッセージによると、sample.SampleResourceというリソースファイルの日本語版(ja_JP)がないということがわかります。

2. リソースファイルを確認します。

   a. リソースファイル名を誤っていないか

      SampleMessage.java の 15 行目の sample.SampleMessage.getMessage () 内で、java.util.ResourceBundle.getBundle()を実行した結果、例外がスローされています。したがって、そこでjava.util.ResourceBundle.getBundle()に渡しているリソースファイル名に誤りがないかどうかを確認します。

b. リソースファイルが、所定のディレクトリ構成内に存在するか

a)のリソースファイル名が正しい場合、所定のディレクトリ構成(/sample/)に、次のどれかのリソースファイルがあるかどうかを確認します。

- SampleResource_ja_JP.properties
- SampleResource_ja_JP.class
- SampleResource_ja.properties
- SampleResource_ja.class
- SampleResource.properties
- SampleResource.class

10.5.1.3 スタックトレースの解析方法(その3)

JDK/JRE 1.4以降、java.lang.Throwableに次のコンストラクタとメソッドが追加されました。

- Throwable(java.lang.String, java.lang.Throwable)
- Throwable(java.lang.Throwable)
- initCause(java.lang.Throwable)

これにより、**スタックトレース**には、原因となる例外の**スタックトレース**も出力されるようになりました。

以下のサンプルを使って、説明します。

```
1 :public class Test {
2 :
3 :    public static void main(String[] args) {
4 :       new Test();
5 :    }
6 :
7 :    Test() {
8 :        try{
9 :            parentMethod();
10:        } catch (Exception e) {
11:            e.printStackTrace();
12:        }
13:    }
14:
15:    void parentMethod() throws HiLevelException {
16:        try {
17:            childMethod();
18:        } catch (Exception e) {
19:            throw new HiLevelException("HiLevel", e);
20:        }
21:    }
22:
23:    void childMethod() throws LowLevelException {
24:        throw new LowLevelException("LowLevel");
25:    }
26:}
27:
28:class HiLevelException extends Exception {
29:    HiLevelException(String msg, Throwable cause) {
30:        super(msg, cause);
31:    }
32:}
33:
34:class LowLevelException extends Exception {
35:    LowLevelException(String msg) {
36:        super(msg);
```

```
37:     }
38:}
```

サンプルを実行すると、以下の**スタックトレース**が出力されます。

```
HiLevelException: HiLevel
    at Test.parentMethod(Test.java:19)
    at Test.<init>(Test.java:9)
    at Test.main(Test.java:4)
Caused by: LowLevelException: LowLevel
    at Test.childMethod(Test.java:24)
    at Test.parentMethod(Test.java:17)
    ... 2 more
```

HiLevelExceptionに続いて、“Caused by:”以降に、原因となるLowLevelExceptionのスタックトレースが出力されています。
最終行の“... 2 more”は、“Caused by:”の直前の2行が続きの**スタックトレース**であることを示しています。
つまり、以下のように解釈することができます。

原因となる例外の解釈

```
Caused by: LowLevelException: LowLevel
    at Test.childMethod(Test.java:24)
    at Test.parentMethod(Test.java:17)
    at Test.<init>(Test.java:9)
    at Test.main(Test.java:4)
```

以上から、次のことがわかります。

- **スタックトレース**の原因は、LowLevelExceptionである
- Test.main(Test.javaの4行目)が、最初の呼び出し元である。

詳細は、Java APIリファレンスのjava.lang.ThrowableのprintStackTraceメソッドの解説を参照してください。

10.5.2 例外発生時のスタックトレース出力

FJVMを使用してJavaアプリケーションを実行している場合、以下の例外については、実行性能の観点から、Java VMの動的コンパイル処理が行う最適化処理により例外発生時のスタックトレース出力処理が省略され、例外発生時のスタックトレースが出力されない場合があります（例外発生時のスタックトレース出力抑止）。

- java.lang.NullPointerException
- java.lang.ArithmeticException
- java.lang.ArrayIndexOutOfBoundsException
- java.lang.ArrayStoreException
- java.lang.ClassCastException

動的コンパイル処理により例外発生時のスタックトレース出力処理が省略されないようにする場合は、“-XX:-OmitStackTraceInFastThrow”オプションを指定します。

ただし該当する例外の発生頻度が高い場合に当該オプションを指定し、例外発生時のスタックトレース出力を行った場合は、Javaアプリケーションの実行性能が低下する場合があります。

当該オプションを指定する場合は、性能検証を行った上で使用するか、開発作業において例外が発生している場所を特定したい場合においてだけ使用してください。

“-XX:-OmitStackTraceInFastThrow”オプションはFJVM固有機能です。

インタプリタで実行しているJavaメソッド内で上記例外が発生した場合は、当該オプションの指定に関係なく、例外発生時のスタックトレースが出力されます。

10.5.3 スレッドダンプ

スレッドダンプには、Javaプロセスの各スレッドの情報(スタックトレース形式)が含まれているため、ハングアップやデッドロックなどの動作具合を調査することができます。

スレッドダンプの出力先は、標準出力です。**スレッドダンプ**が出力される契機および出力先を、“表10.8 スレッドダンプの出力契機と出力先”に示します。

表10.8 スレッドダンプの出力契機と出力先

プログラムの種類	出力契機	出力先
Java EE 7 アプリケーション EE J2EEアプリケーション	一定の条件を満たした場合、コンテナの機能により自動的に採取される場合と利用者の任意のタイミングで手動による採取があります。 ・ 自動採取:アプリケーションがタイムアウトまたは無応答になった場合 ・ 手動採取: Windows32/64 “スレッドダンプツール”で採取することができます。スレッドダンプツールの詳細は、“トラブルシューティング集”の“スレッドダンプツール”を参照してください。 Solaris64 Linux32/64 kill -QUIT [プロセスID]でJava VMに対してQUITシグナルを送り採取することができます。	標準出力、JavaVMの出力をロギングしているファイル
上記以外のJavaプログラム	利用者の任意のタイミングで手動で採取することができます。 Windows32/64 ・ コマンドプロンプトからJavaプログラムを起動した場合: 以下、どちらかの方法で採取できます。 1) [Ctrl]+[Break]キー押下 2) “スレッドダンプツール” ・ コマンドプロンプト以外からJavaプログラムを起動した場合: “スレッドダンプツール”で採取します。 スレッドダンプツールの詳細は、“トラブルシューティング集”の“スレッドダンプツール”を参照してください。 Solaris64 Linux32/64 ・ ターミナルからJavaプログラムを起動した場合: 以下、どちらかの方法で採取できます。 1) [Ctrl]+[¥]キー(英語キーボードの場合バックスラッシュキー)押下 2) kill -QUIT [プロセスID] ・ ターミナル以外からJavaプログラムを起動した場合: kill -QUIT [プロセスID]で採取します。	コンソール(標準出力)

注意

“-Xrs”オプションが指定されたJavaプロセスの場合、当該プロセスへ送られた[Ctrl]+[Break]キー押下またはQUITシグナルに対する動作は、OSのデフォルト動作になります。

そのため、“-Xrs”オプションを指定したJavaプロセスに対して[Ctrl]+[Break]キー押下またはQUITシグナルが送られると、当該Javaプロセスは強制終了または異常終了します。
スレッドダンプを出力する可能性があるJavaプロセスに対して、“-Xrs”オプションは指定しないでください。
ただし、Windows(R)でサービスとして登録されるJavaプロセスの場合は、“-Xrs”オプションを指定しない場合、ログオフ時に強制終了してしまいます。これが不都合な場合は、“-Xrs”オプションを指定してください。

以下のサンプルプログラムをもとにして、**スレッドダンプ**の解析方法を説明します。

```
1 :public class DeadlockSample {
2 :    static boolean flag;
3 :    static Thread1 thread1;
4 :    static Thread2 thread2;
5 :
6 :    public static void main(String[] args) {
7 :        thread1 = new Thread1();
8 :        thread2 = new Thread2();
9 :        thread1.start();
10:        thread2.start();
11:    }
12:}
13:
14:class Thread1 extends Thread {
15:    public Thread1(){
16:        super("Thread1");
17:    }
18:
19:    public void run(){
20:        synchronized(this){
21:            System.out.println("Thread1開始");
22:            while(DeadlockSample.flag==false){  // Thread2が開始するのを待つ
23:                yield();
24:            }
25:            DeadlockSample.thread2.method();
26:            notify();
27:        }
28:    }
29:
30:    public synchronized void method(){
31:        try{wait(1000);}catch(InterruptedException ex){}
32:        System.out.println("Thread1.method()終了");
33:    }
34:}
35:
36:class Thread2 extends Thread {
37:    public Thread2(){
38:        super("Thread2");
39:    }
40:
41:    public void run() {
42:        synchronized(this){
43:            DeadlockSample.flag = true;
44:            System.out.println("Thread2開始");
45:            DeadlockSample.thread1.method();
46:            notify();
47:        }
48:    }
49:
```

```
50:    public synchronized void method() {
51:        try{wait(1000);}catch(InterruptedException ex){}
52:        System.out.println("Thread2.method()終了");
53:    }
54:}
```

サンプルでは、Thread1とThread2がお互いに排他処理を行っています。
このサンプルを実行すると、次のように処理が進められます。

1. Thread1で、Thread1のロックを獲得する(20行目のsynchronized節)
2. Thread2で、Thread2のロックを獲得する(42行目のsynchronized節)
3. Thread1で、Thread2.method()を実行しようとして、ロック解放待ちになる(50行目のsynchronized修飾子)
4. Thread2で、Thread1.method()を実行しようとして、ロック解放待ちになる(30行目のsynchronized修飾子)

この結果、Thead1とThread2がお互いに、解放されないロックを待ち続けるデッドロック状態になります。
デッドロック状態で、**スレッドダンプ**を採取したものを、以下に示します。

スレッドダンプ

```
"DestroyJavaVM" prio=5 tid=0x002856c8 nid=0x5f4 waiting on condition [0..6fad8]

"Thread2" prio=5 tid=0x0092f4d8 nid=0x640 waiting for monitor entry [182ef000..182efd64]
    at Thread1.method(DeadlockSample.java:31)
    - waiting to lock <0x1002ffe8> (a Thread1)
    at Thread2.run(DeadlockSample.java:45)
    - locked <0x10030ca0> (a Thread2)

"Thread1" prio=5 tid=0x0092f370 nid=0x294 waiting for monitor entry [182af000..182afd64]
    at Thread2.method(DeadlockSample.java:51)
    - waiting to lock <0x10030ca0> (a Thread2)
    at Thread1.run(DeadlockSample.java:25)
    - locked <0x1002ffe8> (a Thread1)

"Signal Dispatcher" daemon prio=10 tid=0x0098eb80 nid=0x634 waiting on condition [0..0]

"Finalizer" daemon prio=9 tid=0x0092a540 nid=0x5e8 in Object.wait() [1816f000..1816fd64]
    at java.lang.Object.wait(Native Method)
    - waiting on <0x10010498> (a java.lang.ref.ReferenceQueue$Lock)
    at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:111)
    - locked <0x10010498> (a java.lang.ref.ReferenceQueue$Lock)
    at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:127)
    at java.lang.ref.Finalizer$FinalizerThread.run(Finalizer.java:159)

"Reference Handler" daemon prio=10 tid=0x0096da70 nid=0x5e4 in Object.wait() [1812f000..1812fd64]
    at java.lang.Object.wait(Native Method)
    - waiting on <0x10010388> (a java.lang.ref.Reference$Lock)
    at java.lang.Object.wait(Object.java:429)
    at java.lang.ref.Reference$ReferenceHandler.run(Reference.java:115)
    - locked <0x10010388> (a java.lang.ref.Reference$Lock)

"VM Thread" prio=5 tid=0x0096c950 nid=0x624 runnable

"VM Periodic Task Thread" prio=10 tid=0x0092c008 nid=0x2a0 waiting on condition
"Suspend Checker Thread" prio=10 tid=0x0098e118 nid=0x478 runnable

Found one Java-level deadlock:
=============================
"Thread2":
  waiting to lock monitor 0x00929c3c (object 0x1002ffe8, a Thread1),
  which is held by "Thread1"
"Thread1":
  waiting to lock monitor 0x00929c5c (object 0x10030ca0, a Thread2),
```

```
  which is held by "Thread2"

Java stack information for the threads listed above:
===================================================
"Thread2":
    at Thread1.method(DeadlockSample.java:31)
    - waiting to lock <0x1002ffe8> (a Thread1)
    at Thread2.run(DeadlockSample.java:45)
    - locked <0x10030ca0> (a Thread2)
"Thread1":
    at Thread2.method(DeadlockSample.java:51)
    - waiting to lock <0x10030ca0> (a Thread2)
    at Thread1.run(DeadlockSample.java:25)
    - locked <0x1002ffe8> (a Thread1)

Found 1 deadlock.
```

解析方法

スレッドダンプの各スレッドの情報は、スタックトレース形式です。
Thread1とThread2の両方のスタックトレースには、“locked”と“waiting to lock”があります。また、**スレッドダンプ**の下の方にも、“deadlock”の文字列があり、デットロックが発生していることが確認できます。

このように、**スレッドダンプ**で全スレッドの動作状況を確認するにより、Javaプロセスがハングアップしているか、あるいは、デッドロック状態かを確認することができます。特に、短い間隔で複数のスレッドダンプを採取し、スレッドに動きがなければ、ハングアップの可能性があります。

スレッドダンプの詳細は、“トラブルシューティング集”も参照してください。

オブジェクトをロックしているスレッドがスレッドダンプ上に出ない

通常スレッドダンプ上のあるスレッドで次のように表示される場合があります。

```
- waiting to lock <オブジェクトID> (a クラス名)
```

このような場合、別のスレッドがそのオブジェクトIDのロックを持っていて、そのスレッドのトレース上のどこかで次の表示がされています。

```
- locked <オブジェクトID> (a クラス名)
```

しかし、スレッドダンプを表示するタイミングによっては“- locked <オブジェクトID> (a クラス名)”の表示がどのスレッドにも現われず、“- waiting to lock <オブジェクトID> (a クラス名)”だけ表示される場合があります。

以下のプログラムを例とします。

```
 1  class NoLockOwner extends Thread
 2  {
 3    static Object lock = new Object();
 4
 5    public static void main(String[] arg)
 6    {
 7      new NoLockOwner().start();
 8      new NoLockOwner().start();
 9    }
10
11    public void run()
12    {
13      while (true) {
14        synchronized (lock) {
15          dumb();
16        }
17      }
18    }
19
```

```
20    void dumb()
21    {
22      int n = 0;
23      for (int i = 0 ; i < 1000 ; ++i)
24        n += i;
25    }
26
27  }
```

(0) スレッドダンプを取ると、通常はこのようになります。

```
"Thread-1" prio=1 tid=0x10 nid=0x5 waiting for monitor entry [0x3000..0x4000]
    at NoLockOwner.run(NoLockOwner.java:14)
    - waiting to lock <0x800> (a java.lang.Object)
"Thread-0" prio=1 tid=0x20 nid=0x6 runnable [0x5000..0x6000]
    at NoLockOwner.dumb(NoLockOwner.java:23)
    at NoLockOwner.run(NoLockOwner.java:15)
    - locked <0x800> (a java.lang.Object)
```

(1) 先頭フレームでオブジェクトをロックしている場合は“- locked”ではなく、“- waiting to lock”と表示されることがあります。

```
"Thread-1" prio=1 tid=0x10 nid=0x5 waiting for monitor entry [0x3000..0x4000]
    at NoLockOwner.run(NoLockOwner.java:14)
    - waiting to lock <0x800> (a java.lang.Object)
"Thread-0" prio=1 tid=0x20 nid=0x6 runnable [0x5000..0x6000]
    at NoLockOwner.run(NoLockOwner.java:14)
    - waiting to lock <0x800> (a java.lang.Object)
```

この場合、スレッドの状態を見て、runnableであれば、ロック待ちではなく、ロック取得後同じフレームを実行中の状態であると考えます。

Thread-0、Thread-1ともに、“- waiting to lock <0x800> (a java.lang.Object)”と表示されているので、どちらもロック待ちのように見えます。

しかし、Thread-0の状態は、runnbleなので、ロック待ち状態ではありません。

(2) “- waiting to lock”と表示されるのは、ロック待ちの状態だけではなく、ロック獲得処理の最中でもそのように表示されます。

```
"Thread-1" prio=1 tid=0x10 nid=0x5 waiting for monitor entry [0x3000..0x4000]
    at NoLockOwner.run(NoLockOwner.java:14)
    - waiting to lock <0x800> (a java.lang.Object)
"Thread-0" prio=1 tid=0x20 nid=0x6 waiting for monitor entry [0x5000..0x6000]
    at NoLockOwner.run(NoLockOwner.java:14)
    - waiting to lock <0x800> (a java.lang.Object)
```

Thread-0、Thread-1ともに、“- waiting to lock <0x800> (a java.lang.Object)”と表示されているので、どちらもロック待ちのように見えます。

さらに、スレッドの状態も、どちらも、“waiting for monitor entry”になっています。

“- waiting to lock”と表示されるのは、ロック待ちの状態だけではなく、ロック獲得処理の最中でもそのように表示されます。

したがって、この場合、Thread-0またはThread-1のどちらか一方、あるいは両方がロック獲得処理の最中であると考えます。

しかし、この状態は長く続かず、短時間で(0)または(1)に移行します。

(3) ロックを開放した直後の状態

```
"Thread-1" prio=1 tid=0x10 nid=0x5 waiting for monitor entry [0x3000..0x4000]
    at NoLockOwner.run(NoLockOwner.java:14)
    - waiting to lock <0x800> (a java.lang.Object)
"Thread-0" prio=1 tid=0x20 nid=0x6 runnable [0x5000..0x6000]
    at NoLockOwner.run(NoLockOwner.java:16)
```

Thraed-0がちょうどロックを開放した直後の状態です。

この状態も長く続くとはなく、短時間で(0)または(1)に移行します。

スレッドダンプからアプリケーションの状態を判断する場合、ひとつのスレッドダンプから状態を判断することは困難です。

適切な判断をするためには、複数回のスレッドダンプを総合的に見ることが必要です。

スレッドダンプ中に表示されるsynchronizedメソッドの行番号について

次のようなプログラムを考えます。

```
 1   class SyncMethod extends Thread
 2   {
 3     static volatile int k;
 4
 5     public static void main(String[] arg)
 6     {
 7       new SyncMethod().start();
 8     }
 9
10     public void run()
11     {
12       while (true) {
13         dumb();
14       }
15     }
16
17     synchronized void dumb()
18     {
19       /*
20         meaningless comments
21        */
22       int i = 0;
23       for ( ; i < 10 ; ++i)
24         k += i;
25     }
26
27   }
```

このようなプログラムのスレッドダンプを取ると以下のように出力されることがあります。

```
"Thread-0" prio=1 tid=0x300 nid=0x61 runnable [1000..2000]
    at SyncMethod.dumb(SyncMethod.java:23)
    - waiting to lock <0xa00> (a SyncMethod)
    at SyncMethod.run(SyncMethod.java:13)
```

23行目でロックを獲得しているように見えますが、23行目は、“for (; i < 10 ; ++i)”であり、何もロック獲得に関係しているようにソースコード上は見えません。

これは、synchronizedメソッドがロックを獲得する行番号は、最初に実行される行番号になるためです。

注意

スレッドダンプ中、複数のスレッドがロックを獲得している場合ついて

次のようなプログラムを考えます。

```
 1   class NoNotify extends Thread
 2   {
 3     static Object o = new Object();
 4
 5     public static void main(String[] arg)
 6     {
 7       new NoNotify().start();
 8       new NoNotify().start();
 9     }
10
11     public void run()
```

```
12      {
13        try {
14          synchronized (o) {
15            o.wait();
16          }
17        } catch (Exception e) {}
18      }
19
20    }
```

このプログラムには誤りがあります。

このプログラムでは誰もnotifyするスレッドがないため、どちらのスレッドも永久に起こされることはありません。

このプログラムのスレッドダンプを採取すると次のように示される場所があります。

```
"Thread-1" prio=1 tid=0x800 nid=0x6 in Object.wait() [1000..2000]
    at java.lang.Object.wait(Native Method)
    - waiting on <0x200> (a java.lang.Object)
    at java.lang.Object.wait(Object.java:429)
    at NoNotify.run(NoNotify.java:15)
    - locked <0x200> (a java.lang.Object)
"Thread-0" prio=1 tid=0x900 nid=0x7 in Object.wait() [3000..4000]
    at java.lang.Object.wait(Native Method)
    - waiting on <0x200> (a java.lang.Object)
    at java.lang.Object.wait(Object.java:429)
    at NoNotify.run(NoNotify.java:15)
    - locked <0x200> (a java.lang.Object)
```

このスレッドダンプから分かることは、複数のスレッドがロックを獲得しているのではなく、どのスレッドもロックを獲得していないということです。

Thread-0、Thraed-1ともに、同じオブジェクト“ID<0x200>”のオブジェクトをロックしているように見えます。

しかし“- locked”と表示されているスレッドが、現在のロックオーナであることは意味しません。(“- locked”の正確な意味するところは、そのフレームにおいてロックしたということに過ぎません。)

先頭フレームでは、“- waiting on”と表示されています。

これは、ロックが解放されたら起こされる可能性のある“- waiting to lock”とは異なり、ロックが解放されても自動的に起こされることを意味しません。

注意

Object.wait()を呼び出していないにも関わらず、「in Object.wait()」と表示される

アプリケーションから直接Object.wait()を呼び出していないにも関わらず、以下のように「in Object.wait()」と表示される場合があります。

```
"Thread-1" prio=10 tid=0x70257c00 nid=0x3818 in Object.wait() [0x6fe9c000]
   java.lang.Thread.State: RUNNABLE
        at ClassLoadContention.run(ClassLoadContention.java:10)
```

これは、以下の例のように、複数スレッドで同じクラスをロードしようして、Java VM内部でObject.wait()と同等の処理が実行されているためです。

```
1   class ClassLoadContention extends Thread
2   {
3     public static void main(String[] arg)
4     {
5       new ClassLoadContention().start();
6       new ClassLoadContention().start();
7     }
8
```

```
 9      public void run() {
10        TargetClass.N++;
11      }
12
13      static class TargetClass {
14        static int N;
15        static {
16          try {
17              Thread.sleep(1000);
18          } catch (Exception e) {      }
19        }
20      }
21    }
```

ポイント

JavaVM制御用のスレッド

スレッドダンプにおいて、以下の名前で出力されているスレッドは、Java VM自身の制御用スレッドです。そのため、以下の名前で出力されているスレッドの情報には、アプリケーションとしての動作状態を解析する際の直接的な情報は含まれていません。

- "Attach Listener"
- "C2 CompilerThread*" ("*"部分は数字です)
- "Finalizer"
- "RAS Control Thread"
- "Reference Handler"
- "Signal Dispatcher"
- "VM Periodic Task Thread"
- "VM Thread"
- "Service Thread"
- "GC task thread#* (ParallelGC)" (パラレルGC使用時に存在します) (注1)
- "Concurrent Mark-Sweep GC Thread" (CMS付きパラレルGC使用時に存在します)
- "Surrogate Locker Thread (Concurrent GC)" (CMS付きパラレルGC使用時に存在します)
- "Gang worker#* (Parallel GC Threads)" (CMS付きパラレルGC使用時に存在します) (注1)
- "Gang worker#* (Parallel CMS Threads)" (CMS付きパラレルGC使用時に存在します) (注2)

注1)

"-XX:ParallelGCThreads"オプションで指定したGC処理用スレッドの数だけ存在します。なお、名前の"*"部分は数字です。

注2)

"-XX:ConcGCThreads"オプションで指定したCMS-GC処理用スレッドの数だけ存在します。なお、名前の"*"部分は数字です。

ポイント

Javaヒープ領域に関する情報の出力

スレッドダンプの出力と合わせて、Javaヒープ領域およびメタスペースに関する情報も出力されます。
Javaヒープ領域に関する情報は、各ガーベジコレクション処理の違いにより、New世代領域、Old世代領域、メタスペースの各領域に対応する出力文字列が異なります。
なお、パーセントで示されている値は、情報出力時点でJava VMがJavaヒープ用に利用可能な状態にしている(コミットしている)メモリ量に対する比率です。利用可能な上限値に対する比率ではありません。そのため、パーセントで示されている値は参照せず、K(キロ)単位で表示されているメモリ使用量の値と、オプションで指定された値(デフォルト値を含む)とを比較する方法で各値を利用してください。

- シリアルGC使用時:

  「def new generation」が「New世代領域」、「tenured generation」が「Old世代領域」、「Metaspace」が「メタスペース」に関する情報です。

- パラレルGC使用時:

  「PSYoungGen」が「New世代領域」、「ParOldGen」が「Old世代領域」、「Metaspace」が「メタスペース」に関する情報です。

- CMS付きパラレルGC使用時:

  「par new generation」が「New世代領域」、「concurrent mark-sweep generation」が「Old世代領域」、「Metaspace」が「メタスペース」に関する情報です。
  なお「Old世代領域」における「object space」についての情報は出力されません。

出力例:

```
Heap
 PSYoungGen      total 7168K, used 5158K [0x0fd60000, 0x10470000, 0x10470000)
  eden space 7104K, 72% used [0x0fd60000,0x102658f0,0x10450000)
  from space 64K, 25% used [0x10450000,0x10454000,0x10460000)
  to   space 64K, 0% used [0x10460000,0x10460000,0x10470000)
 ParOldGen       total 4096K, used 162K [0x0c470000, 0x0c870000, 0x0fd60000)
  object space 4096K, 3% used [0x0c470000,0x0c498870,0x0c870000)
 Metaspace       used 5118K, capacity 5162K, committed 5248K, reserved 5504K
```

10.5.4 クラスのインスタンス情報出力機能

以下のオプションを指定したJavaプロセスに対してスレッドダンプ出力の操作を行った場合、スレッドダンプの出力に続いて、Javaヒープ内に生存する各クラスのインスタンス情報が「クラスのインスタンス情報の出力形式」の形式で出力されます。FJVMでは、当該機能を「**クラスのインスタンス情報出力機能**」として実装しています。

クラスのインスタンス情報として、クラス毎のインスタンス数および合計サイズが出力されるため、Javaヒープ内におけるメモリリークなどの調査で利用することができます。

クラスのインスタンス情報の出力先は、標準出力です。クラスのインスタンス情報が出力される契機および出力先は、“10.5.3 スレッドダンプ”が出力される契機および出力先と同じです。

なお、-Xloggcオプションの指定がある場合は、クラスのインスタンス情報の出力先が標準出力から-Xloggcオプションで指定したファイルへ切り替わります。

スレッドダンプに続いて、クラスのインスタンス情報を出力する機能を有効にするオプション

```
-XX:+PrintClassHistogram
```

クラスのインスタンス情報の出力形式

```
num     #instances         #bytes  class name
----------------------------------------------
 $1:            $2            $3  $4
          :
          (略)
          :
Total           $5            $6
```

$1:

順位:クラスのインスタンス情報は、各クラスのインスタンスの合計サイズが大きい順に、ソートされて出力されます。

$2:

クラスのインスタンス数

$3:

クラスのインスタンスの合計サイズ

$4:

クラス名

$5:

$2の値の合計

$6:

$3の値の合計

注意

クラスのインスタンス情報出力機能では、不要なインスタンスを排除した後の情報を採取・出力します。そのため、事前処理としてFullGCを実行します。クラスのインスタンス情報出力の過度の使用は、FullGCの多発となるので注意してください。

注意

クラスのインスタンス情報出力に先立って行われるはずのFullGCが、ガーベジコレクション処理の実行抑止により実行できない状態にある場合は、以下のメッセージを標準出力へ出力した後、クラスのインスタンス情報出力の要求は取り消されます。

```
The PrintClassHistogram operation was canceled because GC could not be run.
```

10.5.5 java.lang.System.gc()実行時におけるスタックトレース出力機能

Javaアプリケーションが以下のJavaメソッドを頻繁に実行すると、Java VMに負荷がかかり、アプリケーションの応答性能を低下させる要因になることがあります。

- java.lang.System.gc()
- java.lang.Runtime.gc()

以降、java.lang.System.gc()(以降、System.gc()と略する)で代表して説明します。

FJVMでは、Javaアプリケーション実行時にSystem.gc()メソッドの実行状態の確認ができるように、当該メソッドを実行したJavaスレッドのスタックトレースを出力する機能を「**java.lang.System.gc()実行時におけるスタックトレース出力機能**」として実装しています。
java.lang.System.gc()実行時におけるスタックトレース出力機能は、以下のオプションを指定した場合に有効となります。

```
-XX:+PrintJavaStackAtSystemGC
```

本機能が有効な場合、Javaアプリケーション内でSystem.gc()メソッドが実行された場合には、当該メソッドを実行したJavaスレッドのスタックトレース情報が、以下のような形で標準出力へ出力されます。

例

java.lang.System.gc()実行時におけるスタックトレース出力機能による出力例

```
"main" #1 prio=5 os_prio=0 tid=0x002cc000 nid=0x1e54 runnable [0x0252f000]
   java.lang.Thread.State: RUNNABLE
                        at java.lang.Runtime.gc(Native Method)
                        at java.lang.System.gc(System.java:928)
                        at SystemGC.main(SystemGC.java:8)

"main" #1 prio=5 os_prio=0 tid=0x002cc000 nid=0x1e54 runnable [0x0252f000]
   java.lang.Thread.State: RUNNABLE
                        at java.lang.Runtime.gc(Native Method)
                        at SystemGC.foo(SystemGC.java:4)
                        at SystemGC.main(SystemGC.java:10)
```

SystemGC.mainからjava.lang.System.gc()が実行され、SystemGC.fooからjava.lang.Runtime.gc()が実行されたことを確認することができます。

参考

スタックトレース情報にスレッドの状態を表す情報(トレース情報の前の行)が表示されます。

出力例では「java.lang.Thread.State: RUNNABLE」となっていますが、"RUNNABLE"の部分が、"NEW"、"TIMED_WAITING (sleeping)"、"WAITING (on object monitor)"、"TIMED_WAITING (on object monitor)"、"WAITING (parking)"、"TIMED_WAITING (parking)"、"BLOCKED (on object monitor)"、"TERMINATED"、"UNKNOWN"などの表示の場合があります。

10.5.6 Java VM終了時における状態情報のメッセージ出力機能

特別なメッセージ出力などがないまま、Javaプロセスが予想外の状態で終了してしまった場合の原因の1つとして、Javaアプリケーションが以下のどれかの処理を実行した場合が考えられます。

- java.lang.System.exit()を予想外の箇所で実行した
- java.lang.Runtime.exit()を予想外の箇所で実行した
- java.lang.Runtime.halt()を予想外の箇所で実行した

以降は、java.lang.System.exit()(以降、System.exit()と略する)で代表して説明します。

System.exit()の実行によりJavaアプリケーションが明示的にJavaプロセスを終了させた場合、Java VM側から見ると正常な仕様動作であるため、Java VMとして特別なメッセージ出力などは行いません。このため、ソースがないなど、内部処理動作の詳細が不明なJavaアプリケーションが予想外の状態で終了した場合には、ソース確認などが行えないため、System.exit()が実行されたかどうかを確認することができません。

このため**FJVM**では、Javaプロセス終了時にSystem.exit()が実行されたかどうかを確認可能にするための機能を、「**Java VM終了時における状態情報のメッセージ出力機能**」として実装しています。
Java VM終了時における状態情報のメッセージ出力機能は、以下のオプションを指定した場合に有効となります。

```
-XX:+VMTerminatedMessage
```

本機能が有効な場合、JavaプロセスがSystem.exit()の実行で終了した場合には、System.exit()を実行したスレッドのスタックトレース情報などが、以下の出力例のような形で標準出力へ出力されます。スタックトレース情報の出力の有無および内容を確認することにより、System.exit()実行の有無を確認することができます。

そして、本機能が有効な場合で、かつJavaプロセス終了時に以下の出力例のようなスタックトレースの情報が出力されなかった場合(「#### JavaVM terminated: …」から始まるメッセージ出力だけの場合)は、System.exit()が使用されず、Javaアプリケーション側の制御論理として終了したと考えられます。
また、本機能が有効な場合で、かつJavaプロセス終了時に以下の情報が何も出力されなかった場合は、ネイティブモジュールの中からCランタイムのexit()関数呼び出しによりJavaプロセスが終了したなど、別原因によるJavaプロセスの終了だと考えられます。

例

Java VM終了時における状態情報のメッセージ出力機能による出力例

```
Thread dump at JVM_Halt(status code=1234):
"main" #1 prio=5 os_prio=0 tid=0x0030c000 nid=0x21fc runnable [0x0070f000]
   java.lang.Thread.State: RUNNABLE
                        at java.lang.Shutdown.halt0(Native Method)
                        at java.lang.Shutdown.halt(Shutdown.java:105)
        - locked <0xfa81e7d0> (a java.lang.Shutdown$Lock)
                        at java.lang.Shutdown.exit(Shutdown.java:179)
        - locked <0xf3dc8dd8> (a java.lang.Class for java.lang.Shutdown)
                        at java.lang.Runtime.exit(Runtime.java:90)
                        at java.lang.System.exit(System.java:906)
                        at JVM_Halt.main(JVM_Halt.java:5)

#### JavaVM terminated: Java HotSpot(TM) Server VM
(**.*.**_FUJITSU_MODIFIED-B**[**.**.**] mixed mode), [pid=29500] TimeMillis=1489491901615
Time=Tue Mar 14 20:45:01 2017
```

JVM_Halt.main()がSystem.exit()を実行し、その結果としてJavaプロセスが終了したことを確認することができます。

参考

スタックトレース情報にスレッドの状態を表す情報(トレース情報の前の行)が表示されます。

出力例では「java.lang.Thread.State: RUNNABLE」となっていますが、"RUNNABLE"の部分が、"NEW"、"TIMED_WAITING (sleeping)"、"WAITING (on object monitor)"、"TIMED_WAITING (on object monitor)"、"WAITING (parking)"、"TIMED_WAITING (parking)"、"BLOCKED (on object monitor)"、"TERMINATED"、"UNKNOWN"などの表示の場合があります。

この情報は**Java VM終了時における状態**の判定には使用しません。

10.5.7 FJVMログ

FJVMでは「Java VM異常終了時のログ出力機能」の強化を行っています。
何らかの原因でJavaプロセスが異常終了した場合、Java VM異常終了時のログとして**FJVMログ**が出力されます。
Javaプロセスが異常終了した原因の調査のために、この**FJVMログ**を活用することができます。

FJVMログの出力先

FJVMログは、Javaプロセスのカレントディレクトリに、以下のファイル名で出力されます。

fjvm_pid***.log (***は異常終了したJavaプロセスのプロセスID)

IJServerクラスタ使用時のカレントディレクトリの詳細は、“Java EE 7 設計・構築・運用ガイド”の“IJServerクラスタ”を参照してください。

IJServer使用時のカレントディレクトリの詳細は、“J2EE ユーザーズガイド(旧版互換)”の“J2EEアプリケーションが運用される環境(IJServer)”を参照してください。

FJVMログの調査

FJVMログとしてJavaプロセス異常終了時の各種情報が格納されますが、その中から以下の情報を原因調査用の情報として使用することができます。

- 10.5.7.1 異常終了箇所の情報
- 10.5.7.2 異常終了時のシグナルハンドラ情報 Solaris64 Linux32/64
- 10.5.7.3 異常終了時のJavaヒープに関する情報

10.5.7.1 異常終了箇所の情報

異常終了箇所に関する情報

異常終了箇所に関する情報が確認できます。

1. 異常終了時に発生した例外に関する情報(シグナルコードおよび例外発生アドレス)

   ログの先頭部分に表示される情報です。以下の出力例の(1)の情報です。

2. 異常終了した関数名(実際には異常終了したアドレスに一番近いシンボル名)

   ログの先頭部分に表示される情報です。以下の出力例の(2)の情報です。

3. 異常終了した関数を含むライブラリ名

   ログの先頭部分に表示される情報です。以下の出力例の(2)の情報です。

4. 異常終了時のJavaスレッドのスタックトレース

   「Java frames:」から始まる情報です。以下の出力例の(3)の情報です。

5. 異常終了時のダイナミックライブラリ一覧

   「Dynamic libraries:」から始まる情報です。以下の出力例の(4)の情報です。

6. 発生時間

   「time:」から始まる情報です。以下の出力例の(5)の情報です。

   出力例

```
#
# A fatal error has been detected by the Java Runtime Environment:
#
(1) # SIGSEGV (0xb) at pc=0xffffffff1db00684, pid=12046, tid=0x0000000000000002
#
# JRE version: Java(TM) SE Runtime Environment (8.0_***-b**) (build 1.8.0_***-b**_Fujitsu_***-**-***)
# Java VM: Java HotSpot(TM) 64-Bit Server VM (1.8.0_***_FUJITSU_MODIFIED-B**[**.**.**] mixed mode solaris-sparc compressed oops)
#
# Problematic frame:
(2) # C [libfjapp.so+0x684] Java_com_fujitsu_jdk_ap_GW00D00100_calcStatus+0x14
#
# Core dump written. Default location: /tmp/core or core.12046
#

～～～～～～～
略
～～～～～～～
(3) Java frames: (J=compiled Java code, j=interpreted, Vv=VM code)
j com.fujitsu.jdk.ap.GW00D00100.calcStatus()V+0
j com.fujitsu.jdk.ap.AL00C00253.jni_calc(II)V+13
j com.fujitsu.jdk.ap.AL00B00221.execute(II)V+15
j com.fujitsu.jdk.ap.AL00A00001.main([Ljava/lang/String;)V+31
v ~StubRoutines::call_stub
～～～～～～～
略
～～～～～～～
(4) Dynamic libraries:
0x0000000100000000 /opt/FJSVawjbk/jdk8/bin/java
0xffffffff7f000000 opt/FJSVawjbk/jdk8/bin/../lib/sparcv9/jli/libjli.so
0xffffffff7f200000 /usr/lib/libthread.so.1
0xffffffff7ed00000 /usr/lib/libdl.so.1
～～～～～～～
```

```
略
～～～～～～～～
(5) time: Wed Mar 22 13:11:40 2017
～～～～～～～～
略
～～～～～～～～
```

調査手順

「異常終了箇所に関する情報」の1.～3.の情報で異常終了した関数を特定し、実行しているJavaアプリケーションから呼び出す関数かどうかを確認します。ただし、2.の異常終了した関数名として出力される名前は、異常終了したアドレスに一番近いシンボル名情報であるため、実際に異常終了した関数とは別の名前が出力されている場合がありますので注意してください。
そして、実行しているJavaアプリケーションが使用する関数の場合には、当該関数使用に際して何らかの問題がないか確認します。
実行しているJavaアプリケーションで使用していない関数の場合には、4.のスタックトレースを調査します。

スタックトレース情報の最初のメソッドがネイティブメソッドだった場合はJNI処理に関係した問題である可能性が高いため、スタックトレース情報で出力された処理のJNI処理に関わる制御で何らかの問題がないか確認します。

また異常終了した関数を含むライブラリ名が利用者作成のライブラリである場合は、利用者側作成のライブラリ内の問題である可能性が高いため、当該ライブラリ内の処理および当該ライブラリを呼び出すJNI処理に何らかの問題がないか確認します。

スタックトレースの調査方法は、“10.5.1 スタックトレース”を参照してください。

スタックオーバーフローの検出

「異常終了箇所に関する情報」の1.の異常終了時に発生した例外に関する情報に、以下のシグナルコードの表記がある場合、例外が発生したスレッドでスタックオーバーフローが発生した(a)、c)の表記)、または発生した可能性がある(b)、d)の表記)ことを示しています。
この場合、例外が発生したスレッドに対するスタックのサイズを大きくすることで問題が解決する可能性があります。

スタックオーバーフロー発生の原因が、Java APIで生成されたスレッドに対するスタックのサイズにある場合は、“10.4.2 スタックのチューニング”を参照して、Java APIで生成されるスレッドに対するスタックのサイズをチューニングしてください。

スタックオーバーフローを示すシグナルコード

Windows32/64

a.「EXCEPTION_STACK_OVERFLOW」

b.「EXCEPTION_ACCESS_VIOLATION (Stack Overflow ?)」

Solaris64 Linux32/64

a.「SIGSEGV (Stack Overflow)」

b.「SIGSEGV (Stack Overflow ?)」

注意

Windowsエラー報告(Windows Error Reporting(WER))の分析 Windows32/64

スタックオーバーフローが原因で発生した異常終了の場合、OS側から**FJVM**側の処理へ制御が渡らず、そのままWindowsエラー報告へ制御が渡されることがあります。この場合は、**FJVMログ**が出力されないため、Windowsエラー報告を確認してください。
Windowsエラー報告に以下の例外番号が出力されている場合には、スタックオーバーフローが原因と考えられます。
なお、Windowsエラー報告の説明は、“10.5.8.1 クラッシュダンプ”を参照してください。

スタックオーバーフローを示す例外番号

```
c00000fd (スタックオーバーフロー)
```

10.5.7.2 異常終了時のシグナルハンドラ情報 Solaris64 Linux32/64

Java VMの実行制御で必要となる“表10.9 Java VMの制御で必要となるシグナル”の各シグナルに対するシグナルハンドラ情報が確認できます。

表10.9 Java VMの制御で必要となるシグナル

Solaris版Java VM	Linux版Java VM
SIGSEGV	SIGSEGV
SIGPIPE	SIGPIPE
SIGBUS	SIGBUS
SIGILL	SIGILL
SIGFPE	SIGFPE
INTERRUPT_SIGNAL (デフォルトはSIGUSR1)	INTERRUPT_SIGNAL (デフォルトはSIGUSR1) (注2)
ASYNC_SIGNAL (デフォルトはSIGUSR2)	SR_SIGNUM (デフォルトはSIGUSR2)
SIGQUIT (注1)	SIGQUIT (注1)
SIGINT (注1)	SIGINT (注1)
SIGHUP (注1)	SIGHUP (注1)
SIGTERM (注1)	SIGTERM (注1)
SIGXFSZ (注3)	SIGXFSZ (注3)

注1)

-Xrsオプションで操作対象となるシグナルです。

注2)

Java VMでリザーブしているシグナルです。

注3)

Java VMで使用されているシグナルです。

シグナルハンドラ情報として、以下の情報が出力されています。

- 登録されているシグナルハンドラのアドレス
- sa_maskおよびsa_flags

“表10.9 Java VMの制御で必要となるシグナル”のシグナルハンドラがJava VM以外の処理で登録されていた場合、Java VMは正常に動作しません。この場合、当該シグナルハンドラを登録しないようにアプリケーションを修正してください。

10.5.7.3 異常終了時のJavaヒープに関する情報

異常終了時のJavaヒープの使用状況が確認できます。

Javaヒープのサイズによる異常終了の場合、異常終了時にどのJavaヒープの枯渇により異常終了が発生したかが確認できます。

10.5.7.4 出力例と調査例

出力例を元に説明します。

(1) 異常終了箇所の情報

異常終了箇所に関する情報が確認できます。
libfjapp.soのJava_com_fujitsu_jdk_ap_GW00D00100_calcStatus関数の近くでSIGSEGV(メモリアクセスで不正なセグメントを参照)が発生しています。
本例の場合、ネイティブライブラリで異常が発生していると判断し、異常発生時のスタックトレース情報を調査します。
com.fujitsu.jdk.ap.AL00C00253.jni_calc()の延長で呼び出しているネイティブメソッドの
com.fujitsu.jdk.ap.GW00D00100.calcStatus()で不正なメモリアクセスが発生しているので、ネイティブコード内で不正なメモリアクセスを招きそうな箇所がないか調べます。

```
#
# A fatal error has been detected by the Java Runtime Environment:
```

```
#
#  SIGSEGV (0xb) at pc=0xffffffff1db00684, pid=12046, tid=0x0000000000000002
#
# JRE version: Java(TM) SE Runtime Environment (8.0_***-b**) (build 1.8.0_***-b**_Fujitsu_**-**-****)
# Java VM: Java HotSpot(TM) 64-Bit Server VM (1.8.0_***_FUJITSU_MODIFIED-B**[**.**.**] mixed mode solaris-
sparc compressed oops)
#
# Problematic frame:
# C  [libfjapp.so+0x684]  Java_com_fujitsu_jdk_ap_GW00D00100_calcStatus+0x14
#
# Core dump written. Default location: /tmp/core or core.12046
#

>>>> Start of VM status report

##>> Thread:
Current thread (0x000000010010e800):  JavaThread "main" [_thread_in_native, id=2,
stack(0xffffffff7b900000,0xffffffff7ba00000)]
～～～～～～～
略
～～～～～～～
Java frames: (J=compiled Java code, j=interpreted, Vv=VM code)
j  com.fujitsu.jdk.ap.GW00D00100.calcStatus()V+0
j  com.fujitsu.jdk.ap.AL00C00253.jni_calc(II)V+13
j  com.fujitsu.jdk.ap.AL00B00221.execute(II)V+15
j  com.fujitsu.jdk.ap.AL00A00001.main([Ljava/lang/String;)V+31
v  ~StubRoutines::call_stub

##>> Process:
～～～～～～～
略
～～～～～～～
```

(2)異常終了時のJavaヒープ領域に関する情報

異常終了時のJavaヒープ領域およびメタスペースに関する情報が確認できます。
パラレルGC使用時:
「PSYoungGen」が「New世代領域」、
「ParOldGen」が「Old世代領域」、
「Metaspace」が「メタスペース」
に関する情報です。

CMS付きパラレルGC使用時:
「par new generation」が「New世代領域」、
「concurrent mark-sweep generation」が「Old世代領域」、
「Metaspace」が「メタスペース」
に関する情報です。
なお「Old世代領域」における「object space」についての情報は出力されません。

シリアルGC使用時:
「def new generation」が「New世代領域」、
「tenured generation」が「Old世代領域」、
「Metaspace」が「メタスペース」
に関する情報です。

本例の場合、異常終了時点における「New世代領域」＋「Old世代領域」の領域(-Xmxで最大量が指定される領域)には、空きがあることがわかります。
また「メタスペース」に対しても、余裕があることがわかります。

注意:
パーセントで示されている値は、異常終了した時点でFJVMがJavaヒープ用に利用可能な状態にしている(コミットし

ている)メモリ量に対する比率です。利用可能な上限値に対する比率ではありません。
パーセントで示されている値は参照せず、K(キロ)単位で表示されているメモリ使用量の値と、オプションで指定された値(デフォルト値を含む)とを比較して判断してください。

```
Heap:
 PSYoungGen      total 2048K, used 896K [0xfffffffff67980000, 0xfffffffff67c00000, 0xfffffffff6a400000)
  eden space 1536K, 58% used [0xfffffffff67980000,0xfffffffff67a60308,0xfffffffff67b00000)
  from space 512K, 0% used [0xfffffffff67b80000,0xfffffffff67b80000,0xfffffffff67c00000)
  to   space 512K, 0% used [0xfffffffff67b00000,0xfffffffff67b00000,0xfffffffff67b80000)
 ParOldGen       total 5632K, used 0K [0xfffffffff62400000, 0xfffffffff62980000, 0xfffffffff67980000)
  object space 5632K, 0% used [0xfffffffff62400000,0xfffffffff62400000,0xfffffffff62980000)
 Metaspace       used 2767K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 293K, capacity 386K, committed 512K, reserved 1048576K
～～～～～～～
略
～～～～～～～
```

(3)異常終了時のシグナルハンドラ情報 Solaris64 Linux32/64

異常終了時のシグナルハンドラに関する情報が確認できます。

```
Signal Handlers:
SIGSEGV: [libjvm.so+0xc4d80c], sa_mask[0]=11111111011111111111110111111111, sa_flags=SA_ONSTACK|SA_RESTART|
SA_SIGINFO
SIGBUS: [libjvm.so+0xc4d80c], sa_mask[0]=11111111011111111111110111111111, sa_flags=SA_RESTART|SA_SIGINFO
SIGFPE: [libjvm.so+0xa26340], sa_mask[0]=11111111011111111111110111111111, sa_flags=SA_RESTART|SA_SIGINFO
SIGPIPE: [libjvm.so+0xa26340], sa_mask[0]=11111111011111111111110111111111, sa_flags=SA_RESTART|SA_SIGINFO
SIGXFSZ: [libjvm.so+0xa26340], sa_mask[0]=11111111011111111111110111111111, sa_flags=SA_RESTART|SA_SIGINFO
SIGILL: [libjvm.so+0xa26340], sa_mask[0]=11111111011111111111110111111111, sa_flags=SA_RESTART|SA_SIGINFO
SIGUSR1: SIG_DFL, sa_mask[0]=00000000000000000000000000000000, sa_flags=none
SIGUSR2: SIG_DFL, sa_mask[0]=00000000000000000000000000000000, sa_flags=none
SIGQUIT: [libjvm.so+0xa20cb0], sa_mask[0]=11111111011111111111110111111111, sa_flags=SA_RESTART
SIGHUP: [libjvm.so+0xa20cb0], sa_mask[0]=11111111011111111111110111111111, sa_flags=SA_RESTART
SIGINT: [libjvm.so+0xa20cb0], sa_mask[0]=11111111011111111111110111111111, sa_flags=SA_RESTART
SIGTERM: [libjvm.so+0xa20cb0], sa_mask[0]=11111111011111111111110111111111, sa_flags=SA_RESTART
SIG39: [libjvm.so+0xa26390], sa_mask[0]=00000000000000000000000000000000, sa_flags=SA_SIGINFO
SIG40: [libjvm.so+0xa26340], sa_mask[0]=11111111011111111111110111111111, sa_flags=SA_RESTART|SA_SIGINFO
```

10.5.8 クラッシュダンプ・コアダンプ

Javaアプリケーションが異常終了(プロセスが消滅)したときに、各OS上に用意されたクラッシュダンプやコアダンプを採取することにより、異常終了の原因を調査することができる場合があります。

10.5.8.1 クラッシュダンプ Windows32/64

Windows(R)上で異常を調査する場合に採取する、クラッシュダンプの採取方法を説明します。

クラッシュダンプの採取には、Windowsエラー報告(Windows Error Reporting(WER))の機能を使用します。

次の例を参考にして、WERを設定してください。

例

WERの設定例

1. MS-DOSコマンドプロンプトなどで“regedit”コマンドを投入し、レジストリエディタを起動します。
2. 「HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows¥Windows Error Reporting¥LocalDumps」キーを作成します。
3. LocalDumpsキーにREG_DWORD型でDumpTypeという値を作成し、「2」を設定します。

WERに関する設定の方法については、以下の情報も参照ください。

- 富士通製サーバ製品の場合:

  http://primeserver.fujitsu.com/primergy/soft/ssupportguide/

- 上記製品以外の場合:

  http://msdn.microsoft.com/en-us/library/bb787181.aspx

10.5.8.2 コアダンプ(Solaris) Solaris64

Solaris上でのコアダンプ採取のための注意事項を説明します。

コアダンプが出力されない場合の確認

コアダンプが出力されない場合の原因として、システムリソース等の問題がまず考えられます。カレントディレクトリの書込み権、ディスク容量、limit(1)コマンド結果を確認してください。

10.5.8.3 コアダンプ(Linux) Linux32/64

Linux上でのコアダンプ採取のための注意事項を説明します。

コアダンプが出力されない場合の確認

- コアダンプが出力されない場合の原因として、システムリソース等の問題がまず考えられます。カレントディレクトリの書込み権、ディスク容量、limit(1)コマンド結果を確認してください。
- ハード/オペレーティングシステムの出荷時、またはオペレーティングシステムのUpdate適用により、デフォルトではコアダンプの出力が設定されていない場合があります。以下を参照して、コアダンプが出力されるように設定してください。

コアダンプ出力の設定方法

- コマンドでInterstageを起動させる場合

  sh(bash)で"ulimit -c unlimited"コマンド実行後、Interstageを起動させます。クラスタ起動ユーザがInterstage起動ユーザと違う場合は、クラスタ起動前に"ulimit -c unlimited"コマンドを実行してから、クラスタを起動させます。

- オペレーティングシステム起動時の自動起動でInterstageを起動する場合(RHEL6)

  以下のファイルの記述を変更することにより、オペレーティングシステムの再起動後にcoreが出力されるようになります。

  /etc/init.d/functions
  「ulimit -S -c unlimited >/dev/null 2>&1」に変更します。

  【修正前】

```
# make sure it doesn't core dump anywhere; while this could mask
# problems with the daemon, it also closes some security problems

ulimit -S -c 0 >/dev/null 2>&1
  または、
ulimit -S -c ${DAEMON_COREFILE_LIMIT:-0} >/dev/null 2>1
```

  【修正後】

```
# make sure it doesn't core dump anywhere; while this could mask
# problems with the daemon, it also closes some security problems

ulimit -S -c unlimited >/dev/null 2>&1
```

  /etc/rc2.d/S99startis
  「ulimit -c unlimited」を追加します。

  【修正前】

```
#!/bin/sh
# Interstage Application Server
# S99starttis : Interstage Application Server start procedure

  OD_HOME=/opt/FJSVod
  export OD_HOME

  /opt/FJSVod/bin/odalive > /dev/null
  while [ "$?" != "0" ]
  do
      sleep 1
      /opt/FJSVod/bin/odalive > /dev/null
  done

  /opt/FJSVtd/bin/isstart
```

【修正後】

```
#!/bin/sh
# Interstage Application Server
# S99starttis : Interstage Application Server start procedure

  OD_HOME=/opt/FJSVod
  export OD_HOME

  ulimit -c unlimited

  /opt/FJSVod/bin/odalive > /dev/null
  while [ "$?" != "0" ]
  do
      sleep 1
      /opt/FJSVod/bin/odalive > /dev/null
  done

  /opt/FJSVtd/bin/isstart
```

- オペレーティングシステム起動時の自動起動でInterstageを起動する場合(RHEL7)

  unitファイルに以下の設定を追加してください。unitファイルでの定義方法については、"付録I RHEL7のunitファイルでの環境定義"を参照してください。

記載するセクション	設定項目	設定値
[Service]	LimitCORE	infinity

10.5.9 JNI処理異常時のメッセージ出力

Java以外の言語と連携する場合、Java Native Interface(JNI)を使用します。
しかし、JNIの使用方法を誤ると、Javaプロセスの終了(異常終了)などの原因となります。

このとき、以下のオプションを指定することにより、JNI処理で異常が発生した場合にメッセージが出力されますので、JNIのパラメーターなどの確認に活用してください。

JNI処理異常時にメッセージを出力するオプション

```
-Xcheck:jni
```

“-Xcheck:jni”パラメーターを指定したときに、以下のメッセージが出力されることがあります。

JNI処理異常時に出力されるメッセージ

```
FATAL ERROR in native method: (詳細メッセージ)
```

(詳細メッセージ)

以降で説明する文字列が出力されます。

(詳細メッセージ)が出力される例と注意事項を説明します。
以降の説明を参考にして、JNIの処理部分を見直してください。

メッセージの説明

JNI received a class argument that is not a class

[異常例] AllocObject関数の第2引数にJNIハンドル経由で受け取ったjclass型ではない型(buf)を指定した場合:

```
(*env)->AllocObject(env, (jclass)buf);
```

JNI string operation received a non-string

[異常例] GetStringUTFChars関数の第2引数にNULLを指定した場合:

```
(*env)->GetStringUTFChars(env, NULL, 0);
```

Non-array passed to JNI array operations

[異常例] GetArrayLength関数の第2引数にjarray型ではない型を指定した場合:

```
(*env)->GetArrayLength(env, (jarray)(*env)->NewStringUTF(env, "abc"));
```

Static field ID passed to JNI

[異常例] GetIntField関数の第3引数にstaticフィールドを指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
jjfieldID fid = (*env)->GetFieldID(env, cls, "static_data", "I");
(*env)->GetIntField(env, obj, fid);
```

"static_data"は、JNIハンドル経由で受け取ったobjオブジェクトのstaticフィールド名

Null object passed to JNI

[異常例] GetIntField関数の第2引数にNULLを指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
jfieldID fid = (*env)->GetFieldID(env, cls, "instance_data", "I");
(*env)->GetIntField(env, NULL, fid);
```

"instance_data" は、JNIハンドル経由で受け取ったobjオブジェクトのinstanceフィールド名

注意

instance変数かどうかのチェック時だけに出力されるメッセージです。

以下のように、GetObjectClass関数の第2引数にNULLを指定した場合、“-Xcheck:jni”オプションによるメッセージは出力されません。

```
(*env)->GetObjectClass(env, NULL);
```

Wrong field ID passed to JNI

[異常例] GetIntFieldの第3引数に数値を指定した場合:

```
(*env)->GetIntField(env, obj, -1);
```

注意

instance変数かどうかのチェック時だけに出力されるメッセージです。

Non-static field ID passed to JNI

[異常例] GetStaticIntField関数の第3引数に数値を指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
(*env)->GetStaticIntField(env, cls, -1);
```

注意

以下のように、GetStaticFieldID関数の第2引数にinstanceフィールド名を指定した場合、“-Xcheck:jni”オプションによるメッセージは出力されません。

```
jclass cls = (*env)->GetObjectClass(env, obj);
jfieldID fid = (*env)->GetStaticFieldID(env, cls, "instance_data", "I");
(*env)->GetStaticIntField(env, cls, fid);
```

"instance_data" は、JNIハンドル経由で受け取ったobjオブジェクトのinstanceフィールド名

Array element type mismatch in JNI

[異常例] GetFloatArrayElements関数の第2引数にjintArray型を指定した場合:

```
jintArray intarray = (*env)->NewIntArray(env, 2);

(*env)->GetFloatArrayElements(env, intarray, 0)
```

Object array expected but not received for JNI array operation

[異常例] GetIntArrayElements関数の第2引数にjobjectArray型を指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
jobjectArray objarray = (*env)->NewObjectArray(env, 1, cls, obj);
(*env)->GetIntArrayElements(env, objarray, 0);
```

Field type (static) mismatch in JNI get/set field operations

[異常例] GetStaticFloatField関数の第3引数にint型のjfieldIDを指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
jfieldID fid = (*env)->GetStaticFieldID(env, cls, "static_data", "I");
(*env)->GetStaticFloatField(env, cls, fid);
```

"static_data"は、JNIハンドル経由で受け取ったobjオブジェクトのstaticフィールド名

Field type (instance) mismatch in JNI get/set field operations

[異常例] GetFloatField関数の第3引数にint型のjfieldIDを指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
jfieldID fid = (*env)->GetFieldID(env, cls, "instance_data", "I");
(*env)->GetFloatField(env, obj, fid);
```

"instance_data" は、JNIハンドル経由で受け取ったobjオブジェクトのinstanceフィールド名

Wrong static field ID passed to JNI

[異常例] GetStaticObjectField関数の第2引数に不正なjclassを指定した場合:

```
jclass cls = (*env)->GetObjectClass(env, obj);
jclass cls2 = (*env)->GetObjectClass(env, (*env)->NewStringUTF(env, "abc"));
jfieldID fid = (*env)->GetStaticFieldID(env, cls, "static_data", "I");
(*env)->GetStaticObjectField(env, cls2, fid);
```

"static_data"は、JNIハンドル経由で受け取ったobjオブジェクトのstaticフィールド名

Using JNIEnv in the wrong thread

[解説]

実行しているスレッドのためのものではないJNIEnvを使用したためのエラーです。
Java VMは、JNIインタフェースポインタ(JNIEnv)が参照する領域を、スレッド固有のデータ領域に割り当てることがあります。このため、JNIインタフェースポインタは、カレントスレッドに対してだけ有効です。ネイティブメソッドは、JNIインタフェースポインタを別のスレッドに渡すといった使い方はできません。

JNI call made with exception pending

[解説]

ネイティブプログラムで何らかの例外が発生後、その例外を処理せずに引き続きJNI関数を実行したためのエラーです。
ネイティブプログラムでJNI関数を呼び出した後は、その都度ExceptionOccurredを使用して例外の発生状況をチェックし、必要に応じて、例外のクリア、または、例外を上位メソッドへスローしてください。

10.6 異常発生時の原因振り分け

異常が発生したときに、原因を振り分ける方法を説明します。

10.6.1 java.lang.OutOfMemoryErrorがスローされた場合

OutOfMemoryErrorがスローされた場合、考えられる原因とその対処方法を説明します。

なお、OutOfMemoryErrorがスローされた場合に出力されるメッセージ情報については、“10.6.1.1 メモリ領域不足事象発生時のメッセージ出力機能の強化”も参照してください。

想定される原因(メモリリーク)

VMがガーベジコレクションを繰り返しても、時間の経緯とともにメモリ消費量が増大していく場合、プログラム中メモリリークを起こしている可能性があります。

メモリリークの結果、Javaのヒープ不足が発生し**OutOfMemoryError**がスローされる場合があります。

この場合、ガーベジコレクションのログを採取して、Javaヒープの消費状況を確認してください。ガーベジコレクションのログを採取する方法は、“10.2.6 ガーベジコレクションのログ出力”を参照してください。

想定される原因(Javaヒープ不足)

通常、**OutOfMemoryError**は、Javaヒープ不足が原因でスローされます。

ガーベジコレクションのログを採取して、Javaヒープの消費状況を確認してください。
Javaヒープの空き容量がないことが確認されたら、Javaヒープをチューニングしてください。

ガーベジコレクションのログを採取する方法は、“10.2.6 ガーベジコレクションのログ出力”を参照してください。
Javaヒープのチューニング方法は、“10.4.1 Javaヒープおよびメタスペースのチューニング”を参照してください。

想定される原因(メタスペース不足)

メタスペースが不足によって、**OutOfMemoryError**がスローされる場合があります。

ガーベジコレクションのログを採取して、メタスペースの消費状況を確認してください。
メタスペースの空き容量がないことが確認されたら、メタスペースをチューニングしてください。

ガーベジコレクションのログを採取する方法は、“10.2.6 ガーベジコレクションのログ出力”を参照してください。
メタスペースのチューニング方法は、“10.4.1 Javaヒープおよびメタスペースのチューニング”を参照してください。

注意

メタスペースは、クラスローダーごとに管理するため、あるクラスローダーでロードされたクラス情報を、別のクラスローダーが管理するメタスペースに格納できません。

また、各クラスローダーのメタスペースは、クラス情報ごとに取得するのではなく、ある程度のまとまったサイズごとに取得して、そこにクラス情報を設定していきます。

このため、取得済みのメタスペースに空きがある場合でも、別のクラスローダーが管理するものであれば、その空きを使用できません(使用済みと扱います)。

その結果、使用済みのサイズが“-XX:MaxMetaspaceSize”オプションで指定した値に到達していないにも関わらず、OutOfMemoryErrorがスローされる場合があります。

想定される原因(ユーザ空間不足)

多量のスレッドを生成して、多量のスタックがユーザ空間内に割り当てられ、ユーザ空間不足になった場合、次の**OutOfMemoryError**がスローされる、あるいはエラーメッセージとして表示を行いプロセスが終了します。

```
java.lang.OutOfMemoryError: unable to create new native thread
```

ユーザ空間が不足している場合は、Javaヒープまたはスタックのサイズを小さくするなどのチューニングを行ってください。
スタックのサイズをチューニングする方法は、“10.4.2 スタックのチューニング”を参照してください。
Javaヒープおよびメタスペースのチューニング方法は、“10.4.1 Javaヒープおよびメタスペースのチューニング”を参照してください。

なお、仮想メモリに余裕がある場合は、Javaプロセスを複数起動して、プロセス多重度を上げる方法もあります。J2EEアプリケーションまたはJava EE 7 アプリケーションの場合、J2EEまたはJava EE 7のチューニングを行ってください。J2EEまたはJava EE 7のチューニング方法の詳細は、それぞれのマニュアルを参照してください。

想定される原因(仮想メモリ不足)

仮想メモリが不足してスレッドが生成できない場合、次の**OutOfMemoryError**がスローされる、あるいはエラーメッセージとして表示を行いプロセスが終了します。

```
java.lang.OutOfMemoryError: unable to create new native thread
```

仮想メモリが不足した場合は、他の不要なプロセスを終了して仮想メモリに余裕を持たせるか、物理メモリ(RAM)またはスワップファイルを拡張して仮想メモリを増やすようにチューニングを行ってください。

想定される原因(ガーベジコレクション処理の実行抑止)

ガーベジコレクション処理(GC処理)の実行抑止により、クリティカルセクション状態時に**OutOfMemoryError**がスローされた場合は、必要に応じて、GC処理の実行抑止による影響ができるだけ小さくなるように、実行するアプリケーションの処理内容を見直してください(アプリケーション処理内の、GC処理の実行抑止に関係する機能の利用見直しを行ってください)。
GC処理の実行抑止については、“10.2.1 FJVMでサポートされるガーベジコレクション処理”を参照してください。
なお、Javaヒープのチューニングにより、GC処理の実行抑止による**OutOfMemoryError**の発生が緩和できる場合があります。

- クリティカルセクション状態時に、ネイティブプログラムからJNIを利用してJavaのオブジェクトの生成要求を行っていないアプリケーションの場合は、Old世代領域を大きくする(メモリ割り当てプール全体を大きくする)チューニングで、GC処理の実行抑止による**OutOfMemoryError**の発生が緩和できる場合があります。
- クリティカルセクション状態時に、ネイティブプログラムからJNIを利用してJavaのオブジェクトの生成要求を行っているアプリケーションの場合は、New世代領域を大きくするチューニングで、GC処理の実行抑止による**OutOfMemoryError**の発生が緩和できる場合があります。

ガーベジコレクションのログを採取する方法は、“10.2.6 ガーベジコレクションのログ出力”を参照してください。
Javaヒープおよびメタスペースのチューニング方法は、“10.4.1 Javaヒープおよびメタスペースのチューニング”を参照してください。

注意

GC処理の実行抑止により、クリティカルセクション状態で**OutOfMemoryError**がスローされたかどうかは、“メモリ領域不足事象発生時のメッセージ出力機能の強化”で出力されるメッセージを参照して判断してください。
また、EXTP4435メッセージまたはPCMI1105メッセージが出力されている場合は、IJServerのコンテナ情報ログ(info.log)およびIJServerクラスタのJava VMログ(console.log)に出力される「Java VMのヒープ域不足の詳細情報」を参照して判断してください。

10.6.1.1 メモリ領域不足事象発生時のメッセージ出力機能の強化

FJVMでは、メモリ領域不足事象発生時に出力されるメッセージ情報の強化を行っています。
これにより**FJVM**では、メモリ領域不足事象が発生した場合に、java.lang.OutOfMemoryErrorの例外メッセージ情報に加え、不足した領域の種別情報を以下の形式で出力します。

メモリ領域不足事象が発生した場合に出力される不足した領域の種別情報

```
The memory was exhausted area_name
  Java heap size / max Java heap size = heap_size / max_heap_size
  Java metaspace size / max Java metaspace size = metaspace_size / max_metaspace_size
```

area_name:

メモリ領域不足事象が発生した領域の名前や領域不足となったオブジェクトの要求サイズ(不足領域情報)を表示します。
不足領域情報としては以下の項目があります。

- on Java heap space. : requested <NNNN> bytes

  NNNNバイトのオブジェクト生成要求において、メモリ割り当てプール(New世代領域またはOld世代領域)に対してメモリ領域不足事象が発生した場合です。
  なお、エルゴノミクス機能によるメモリ領域不足事象の検出機能が有効な場合で、かつ当該機能によりメモリ領域不足事象を検出した場合、この項目になる場合があります。

- on Java heap space. : requested <NNNN> bytes (in critical section)

  意味は「on Java heap space. : requested <NNNN> bytes」と同じですが、メモリ領域不足事象発生時、クリティカルセクション状態でGC処理の実行が抑止されていたことを示しています。

- on Metaspace. : requested <NNNN> bytes

  NNNNバイトのオブジェクト生成要求において、メタスペースに対してメモリ領域不足事象が発生した場合です。

- on Metaspace. : requested <NNNN> bytes (in critical section)

  意味は「on Metaspace. : requested <NNNN> bytes」と同じですが、メモリ領域不足事象発生時、クリティカルセクション状態でGC処理の実行が抑止されていたことを示しています。

- (なし)

  スタックやヒープなど、Javaヒープやメタスペース以外の領域に対してメモリ領域不足事象が発生した場合です。特にjava.lang.OutOfMemoryErrorの例外メッセージ情報が“java.lang.OutOfMemoryErrorがスローされた場合”の「ユーザ空間不足」または「仮想メモリ不足」の場合に出力される形式の場合は、スタックやヒープなど、Javaヒープやメタスペース以外の領域に対してメモリ領域不足事象が発生した場合と断定できます。
  またはJavaアプリケーション実行時における配列生成式の評価の段階で、配列オブジェクトの長さ(配列要素の数)から、当該配列オブジェクトを割り当てるための領域が十分でないと評価された場合(配列の長さ(配列要素の数)が大きすぎ、配列オブジェクトとしての大きさが2ギガバイト程度もしくはそれ以上の大きさになる配列の定義がある場合)。
  またはクラスのロード処理でメモリ不足が発生した場合。

  なお、エルゴノミクス機能によるメモリ領域不足事象の検出機能が有効な場合で、かつ当該機能によりメモリ領域不足事象を検出した場合、この項目になる場合があります。

heap_size:

メモリ領域不足事象が発生した際に使用中となっているメモリ割り当てプールのサイズ(単位:byte)。

max_heap_size:

利用可能なメモリ割り当てプールの最大サイズ(単位:byte)。

metaspace_size:

メモリ領域不足事象が発生した際に使用中となっているメタスペースのサイズ(単位:byte)。

max_metaspace_size:

利用可能なメタスペースの最大サイズ(単位:byte)。

注意

メモリ領域不足事象が発生した際に出力される各領域の使用中サイズ(heap_size、metaspace_size)には、メモリ領域不足の原因となったオブジェクトの大きさは含まれません。

そのため巨大サイズのオブジェクト生成要求などによりメモリ領域不足事象が発生した場合には、「最大サイズ」と「使用中サイズ」の差が大きい場合（空き領域がたくさんあるように見える場合）がありますので注意してください。

注意

NewGC処理では、New世代領域を「eden space」、「from space」および「to space」の3つの内部領域に細分割し、当該領域上において、一般に世代別GC制御と言われる制御方法を用いて、Javaアプリケーションが生成要求したオブジェクトを管理・制御しています。

このうち、「from space」および「to space」は、Java VMがNewGC処理を行う際の作業域的な役割を持つ領域となっています。そのため、「from space」および「to space」の各領域が占める大きさのうち、Javaアプリケーションからのオブジェクト生成要求のために使用される大きさは、その領域の一部分だけとなります。

そのため、出力データにおいて、メモリ割り当てプールやNew世代領域に空きがあるように見える場合であっても、実際には空きがない場合がありますので注意してください（空きがあるように見える場合であっても、その差は、NewGC処理用の作業域的な役割で使用済となっている場合があります）。

注意

メモリ割り当てプールに対してメモリ領域不足事象が発生した場合に出力されるheap_sizeの値は、New世代領域での使用中サイズとOld世代領域での使用中サイズの合計値です。
New世代領域とOld世代領域は別々のオブジェクト格納域として管理・制御されますから、max_heap_sizeとheap_sizeの差の大きさが、そのまま生成要求できるオブジェクトの最大サイズにはなりませんので注意してください。

例

メモリ領域不足事象が発生した場合に出力されるメッセージ出力例

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
The memory was exhausted on Java heap space. : requested 4016 bytes
Java heap size / max Java heap size = 361797384 / 536870912
Java metaspace size / max Java metaspace size = 2662408 / 1073741824
```

4016バイトのオブジェクト生成要求において、メモリ割り当てプールに対してメモリ領域不足が発生したことを確認することができます。

10.6.2 EXTP4435メッセージまたはPCMI1105メッセージが出力された場合

Interstage Application Server配下のJavaアプリケーション実行時において、以下のメッセージが出力された場合は、メモリ領域不足事象が原因による異常となります。
Javaヒープおよびメタスペースをチューニングしてください。

Javaヒープおよびメタスペースのチューニング方法は、“10.4.1 Javaヒープおよびメタスペースのチューニング”を参照してください。

- EXTP4435メッセージ
- PCMI1105メッセージ

参照

各メッセージの内容については、“メッセージ集”を参照してください。

なお、IJServerのコンテナ情報ログ(info.log)およびIJServerクラスタのJava VMログ(console.log)に出力される「Java VMのヒープ域不足の詳細情報」の出力形式は、以下のとおりです。

Java VMのヒープ域不足の詳細情報の出力形式

```
------------------------------------------------------------------
OutOfMemory Log
------------------------------------------------------------------
pid=process_id
heap_type=heap_type_code
heap_size=heap_size
max_heap_size=max_heap_size
metaspace_size=metaspace_size
max_metaspace_size=max_metaspace_size
requested_size=requested_size
------------------------------------------------------------------
VM is terminated by occurred OutOfMemoryError on heap_type.
stack_trace
```

process_id:

メモリ領域不足事象が発生したJavaプロセスのプロセス番号。

heap_type_code:

メモリ領域不足事象が発生した領域に対応する番号です。表示される番号については、heap_typeの説明を参照してください。

heap_size:

メモリ領域不足事象が発生した際に使用中となっているメモリ割り当てプールのサイズ(単位:byte)。

max_heap_size:

利用可能なメモリ割り当てプールの最大サイズ(単位:byte)。

metaspace_size:

メモリ領域不足事象が発生した際に使用中となっているメタスペースのサイズ(単位:byte)。

max_metaspace_size:

利用可能なメタスペースの最大サイズ(単位:byte)。

requested_size:

領域不足となったオブジェクトの要求サイズ(単位:byte)。

heap_type:

不足領域情報(メモリ領域不足事象が発生した領域の名前など)を表示します。
不足領域情報としては以下の項目があります。

- Java heap

  requested_sizeバイトのオブジェクト生成要求において、メモリ割り当てプール(New世代領域またはOld世代領域)に対してメモリ領域不足事象が発生した場合です。
  この項目のときのheap_type_codeは「1」になります。
  なお、エルゴノミクス機能によるメモリ領域不足事象の検出機能が有効な場合で、かつ当該機能によりメモリ領域不足事象を検出した場合、この項目になる場合があります。

- Java heap in critical section

  意味は「Java heap」と同じですが、メモリ領域不足事象発生時、クリティカルセクション状態でGC処理の実行が抑止されていたことを示しています。

- Metaspace

  requested_sizeバイトのオブジェクト生成要求において、メタスペースに対してメモリ領域不足事象が発生した場合です。
  この項目のときのheap_type_codeは「2」になります。

- Metaspace in critical section

  意味は「Metaspace」と同じですが、メモリ領域不足事象発生時、クリティカルセクション状態でGC処理の実行が抑止されていたことを示しています。

- C heap 制御情報

  スタックやヒープなど、Javaヒープ以外の領域に対してメモリ領域不足事象が発生した場合(requested_sizeバイトのメモリ割り当て要求が失敗した場合)です。
  この項目のときのheap_type_codeは「0」になります。
  なお、現象発生時の要求サイズが不明な場合のrequested_sizeは「0」になります。
  また、次行に制御情報(メモリが確保できなかったJava VM制御の情報)が出力される場合があります。

- unknown space

  Javaアプリケーション実行時における配列生成式の評価の段階で、配列オブジェクトの長さ(配列要素の数)から、当該配列オブジェクトを割り当てるための領域が十分でないと評価された場合(配列の長さ(配列要素の数)が大きすぎ、配列オブジェクトとしての大きさが2ギガバイト程度もしくはそれ以上の大きさになる配列の定義がある場合)。
  またはクラスのロード処理でメモリ不足が発生した場合。
  なお、エルゴノミクス機能によるメモリ領域不足事象の検出機能が有効な場合で、かつ当該機能によりメモリ領域不足事象を検出した場合、この項目になる場合があります。
  この項目のときのheap_type_codeは「-1」になります。
  なお、この項目の場合のrequested_sizeは「0」になります。

stack_trace:

メモリ領域不足事象が発生したスレッドがJavaアプリケーションを実行しているスレッドだった場合は、当該スレッドのスタックトレースが出力されます。それ以外のスレッドの場合、またはリソース不足によりスタック情報が取り出せない場合は、スタックトレースは出力されません。

注意

メモリ領域不足事象が発生した際に出力される各領域の使用中サイズ(heap_size、metaspace_size)には、メモリ領域不足の原因となったオブジェクトの大きさは含まれません。

そのため巨大サイズのオブジェクト生成要求などによりメモリ領域不足事象が発生した場合には、「最大サイズ」と「使用中サイズ」の差が大きい場合(空き領域がたくさんあるように見える場合)がありますので注意してください。

注意

NewGC処理では、New世代領域を「eden space」、「from space」および「to space」の3つの内部領域に細分割し、当該領域上において、一般に世代別GC制御と言われる制御方法を用いて、Javaアプリケーションが生成要求したオブジェクトを管理・制御しています。

このうち、「from space」および「to space」は、Java VMがNewGC処理を行う際の作業域的な役割を持つ領域となっています。そのため、「from space」および「to space」の各領域が占める大きさのうち、Javaアプリケーションからのオブジェクト生成要求のために使用される大きさは、その領域の一部分だけとなります。

そのため、出力データにおいて、メモリ割り当てプールやNew世代領域に空きがあるように見える場合であっても、実際には空きがない場合がありますので注意してください（空きがあるように見える場合であっても、その差は、NewGC処理用の作業域的な役割で使用済となっている場合があります）。

注意

メモリ割り当てプールに対してメモリ領域不足事象が発生した場合に出力されるheap_sizeの値は、New世代領域での使用中サイズとOld世代領域での使用中サイズの合計値です。
New世代領域とOld世代領域は別々のオブジェクト格納域として管理・制御されますから、max_heap_sizeとheap_sizeの差の大きさが、そのまま生成要求できるオブジェクトの最大サイズにはなりませんので注意してください。

例

Java VMのヒープ域不足の詳細情報の出力例

スタックトレース情報にスレッドの状態を表す情報（トレース情報の前の行）が表示されます。

```
---------------------------------------------------------------------------
OutOfMemory Log
---------------------------------------------------------------------------
pid=4696
heap_type=1
heap_size=136800
max_heap_size=6291456
metaspace_size=2052320
max_metaspace_size=67108864
requested_size=40000016
---------------------------------------------------------------------------
VM is terminated by occurred OutOfMemoryError on Java heap.
"main" prio=6 tid=0x00307000 nid=0x12a8 runnable [0x0092f000]
java.lang.Thread.State: RUNNABLE
at test.<init>(test.java:10)
at test.main(test.java:5)
```

「java.lang.Thread.State: RUNNABLE」となっていますが、"RUNNABLE"の部分が、"NEW"、"TIMED_WAITING (sleeping)"、"WAITING (on object monitor)"、"TIMED_WAITING (on object monitor)"、"WAITING (parking)"、"TIMED_WAITING (parking)"、"BLOCKED (on object monitor)"、"TERMINATED"、"UNKNOWN"などの表示の場合があります。

10.6.3 java.lang.StackOverflowErrorがスローされた場合

StackOverflowErrorがスローされた場合、スタックオーバーフローが原因です。

スタックのサイズをチューニングしてください。
スタックのチューニング方法は、“10.4.2 スタックのチューニング”を参照してください。

なお、StackOverflowErrorがスローされず、そのままJavaプロセスが異常終了する場合もあります。

10.6.4 SIGBUS発生により異常終了した場合 Solaris64

Solaris上で実行しているプロセスが、以下の状態のSIGBUS発生により異常終了した場合は、システムのメモリ資源／スワップ不足により発生した異常終了です。

- signal no : 10(SIGBUS)
- signal code : 3(BUS_OBJERR)
- signal error: 12(ENOMEM)

そして、Javaプロセスが異常終了した場合に出力されるFJVMログにおいて、以下の情報が出力されている場合が、上記状態に相当します(JDK/JREのバージョンにより異なります)。

```
siginfo:si_signo=SIGBUS: si_errno=Not enough space, si_code=3(BUS_OBJERR), si_addr=16進数
```

または

```
siginfo:si_signo=10, si_errno=12, si_code=3, si_addr=16進数
```

異常終了時の情報として上記情報が出力された場合は、不要なプロセスを終了して仮想メモリに余裕を持たせるか、物理メモリ(RAM)またはスワップファイルを拡張して仮想メモリを増やすようにチューニングを行ってください。

10.6.5 プロセスが消滅(異常終了)した場合

何の痕跡も残さずに突然プロセスが消滅した場合に、考えられる原因とその対処方法を説明します。

想定される原因(スタックオーバーフロー)

Windows32/64

通常、スタックオーバーフローが発生した場合、java.lang.StackOverflowErrorがスローされ、Windowsエラー報告（Windows Error Reporting（WER））が検知してユーザダンプを出力します。

しかし、OSが高負荷状態になったり、スタックオーバーフロー発生時のスタック残量が少なかったりすると、OSからFJVMにもWindowsエラー報告にも制御が渡らないまま、痕跡を残さずにプロセスが消滅することがあります。

したがって、プロセスが消滅した原因が不明な場合は、スタックのサイズを拡張して、現象が改善できるかどうかを確認してください。スタックのサイズを拡張しても改善できない場合は、別の原因を調査してください。

なお、Windowsエラー報告の説明は、“10.5.8.1 クラッシュダンプ”を参照してください。

スタックオーバーフローが発生したことを確認できた場合、該当するスタックのサイズをチューニングしてください。スタックのチューニング方法は、“10.4.2 スタックのチューニング”を参照してください。

想定される原因(長時間コンパイル処理の検出機能による終了)

FJVMの“長時間コンパイル処理の検出機能”による終了の可能性があります。
詳細は、“10.3.2 長時間コンパイル処理の検出機能”を参照してください。

Javaアプリケーションを“-XX:CompileTimeout”オプション付きで起動した場合は、標準出力にFJVMからのメッセージが出力されていないかどうかを確認してください。

想定される原因(シグナルハンドラ) Solaris64 Linux32/64

Java VM以外のモジュールで、シグナルハンドラを登録した場合、Javaアプリケーションが正常に動作せずに、異常終了することがあります。詳細は、“10.5.7.2 異常終了時のシグナルハンドラ情報 ”を参照してください。

FJVMを使用している場合は、FJVMログにシグナルハンドラ情報が出力されますので、それを確認してください。

想定される原因(JNI処理の異常)

JNI経由でJava以外の言語で開発したネイティブモジュールと連携する際、JNIの使用方法を誤ると、プロセス消滅の原因となります。

このようなときは、“-Xcheck:jni”オプションを指定して、JNI処理でメッセージが出力されないかどうかを確認してください。“-Xcheck:jni”オプションの詳細は、“10.5.9 JNI処理異常時のメッセージ出力”を参照してください。

JNI処理に誤りがなくても、ネイティブモジュールで異常終了またハングアップが発生すると、Javaアプリケーションのプロセスが消滅する場合があります。たとえば、スレッドアンセーフな関数を使用している場合は、注意が必要です。

例

スレッドアンセーフな関数の例 Solaris64

次の関数を使用したときに、障害が発生した事例があります。

- vfork

想定される原因(プログラムによる終了)

Javaプロセスが、特別なメッセージ出力などがないまま、予想外の状態で終了した場合、原因の1つとして、次のどれかが考えられます。

- java.lang.Runtime.exit()を予想外の箇所で実行した
- java.lang.Runtime.halt()を予想外の箇所で実行した
- java.lang.System.exit()を予想外の箇所で実行した

FJVMを使用している場合は、“10.5.6 Java VM終了時における状態情報のメッセージ出力機能”を参照して、対処してください。

10.6.6 ハングアップ(フリーズ)した場合

本節では、Javaプロセスが残っているにもかかわらず、プログラムが無応答になった場合、考えられる原因とその対処方法を説明します。

想定される原因(デッドロック)

デッドロックが発生した場合、そのスレッドが停止されます。
ハングアップしたときに、スレッドダンプを採取して、デッドロックがないかどうかを確認してください。

スレッドダンプの採取方法および解析方法の詳細は、“10.5.3 スレッドダンプ”を参照してください。

想定される原因(ガーベジコレクション)

ガーベジコレクションが発生すると、ガーベジコレクションが終了するまでの間、Javaアプリケーションのすべてのスレッドが停止されます。
これにより、Javaアプリケーションがハングアップしたかのように見える場合があります。

ガーベジコレクションのログを採取して、ガーベジコレクションが動作したタイミングを照合してください。ガーベジコレクションが原因で無応答のような現象になる場合は、Javaヒープをチューニングして、ガーベジコレクションの動作具合を改善してください。

ガーベジコレクションのログを採取する方法は、“10.2.6 ガーベジコレクションのログ出力”を参照してください。
Javaヒープおよびメタスペースのチューニング方法は、“10.4.1 Javaヒープおよびメタスペースのチューニング”を参照してください。

想定される原因(JNI処理の異常)

JNI経由でJava以外の言語で開発したネイティブモジュールと連携する際、JNIの使用方法を誤ると、ハングアップの原因となります。

このようなときは、“-Xcheck:jni”オプションを指定して、JNI処理でメッセージが出力されないかどうかを確認してください。“-Xcheck:jni”オプションの詳細は、“10.5.9 JNI処理異常時のメッセージ出力”を参照してください。

JNI処理に誤りがなくても、JNIモジュールで異常終了またハングアップが発生すると、Javaアプリケーションがハングアップする場合があります。たとえば、スレッドアンセーフな関数を使用している場合は、注意が必要です。

例

スレッドアンセーフな関数の例 Solaris64

次の関数を使用したときに、障害が発生した事例があります。

- vfork

10.6.7 スローダウンが発生した場合

Javaアプリケーションの動きが遅くなる現象(スローダウン)が発生した場合に、考えられる原因とその対処方法を説明します。

想定される原因(ガーベジコレクション)

ガーベジコレクションが発生すると、ガーベジコレクションが終了するまでの間、Javaアプリケーションのすべてのスレッドが停止されます。このため、Javaアプリケーションのレスポンス(応答)が遅くなる場合があります。

ガーベジコレクションのログを採取して、ガーベジコレクションが動作したタイミングとスローダウンが発生したタイミングを照合してください。ガーベジコレクションが原因でスローダウンになる場合は、Javaヒープをチューニングして、ガーベジコレクションの動作具合を改善してください。

ガーベジコレクションのログを採取する方法は、“10.2.6 ガーベジコレクションのログ出力”を参照してください。
Javaヒープおよびメタスペースのチューニング方法は、“10.4.1 Javaヒープおよびメタスペースのチューニング”を参照してください。

例

スローダウンの事例

同じソフトウェアとJavaアプリケーションが動作している複数のWebサーバのうち、一部のサーバだけがスローダウンしたという事例があります。この原因は、マシンに搭載されている物理メモリ(RAM)の容量の違いでした。
物理メモリ(RAM)が少ないマシンの場合、ガーベジコレクションの実行に伴い、ディスクのスワッピングが発生し、スローダウンすることがあります。

10.7 Javaツール機能

本製品では、Javaプログラムのチューニングやトラブルシューティングに有用なツールを提供しています。ツールには、次の4つがあります。

- メソッドのトレースを出力するメソッドトレース機能
- Javaヒープ使用量を出力するjheap
- スレッドダンプを出力するスレッドダンプツール Windows32/64
- JDKに含まれるトラブルシューティングに役立つツール

ツールの格納先

- メソッドのトレースを出力するメソッドトレース機能
- Javaヒープ使用量を出力するjheap
- スレッドダンプを出力するスレッドダンプツール Windows32/64

  Windows32/64

－ JDK 使用時: [Interstageインストールフォルダ]¥jdk8¥tools

－ JRE 使用時: [Interstageインストールフォルダ]¥jre8¥tools

Solaris64 Linux32/64

以下の“$DIR”はインストール時に指定する相対ディレクトリ名です。“$DIR”のシステム推奨名は“opt”です。

－ JDK 使用時: /$DIR/FJSVawjbk/jdk8/tools

－ JRE 使用時: /$DIR/FJSVawjbk/jre8/tools

・ JDKに含まれるトラブルシューティングに役立つツール

Windows32/64

－ [Interstageインストールフォルダ]¥jdk8¥bin

Solaris64 Linux32/64

以下の“$DIR”はインストール時に指定する相対ディレクトリ名です。“$DIR”のシステム推奨名は“opt”です。

－ /$DIR/FJSVawjbk/jdk8/bin

各ツールの詳細は、“トラブルシューティング集”の“Javaツール機能”を参照してください。

EE

第11章　データベース連携サービスのチューニング Windows32/64

Interstageのシステム構成に応じて、データベース連携サービスで提供するパラメタをチューニングする必要があります。

ここでは、データベース連携サービスのiniファイル内のパラメタについて説明します。

11.1 データベース連携サービスのiniファイル設定情報 Windows32/64

データベース連携サービスのiniファイルには、以下の表に示すパラメタを設定します。

システムチューニングを行ってデータベース連携サービスを使用する場合は、iniファイルを変更します。iniファイルを変更しない場合、データベース連携サービスは、初期値で動作します。

iniファイルは、インストール時に、以下に格納されます。設定例については、“11.2 iniファイルの設定例”を参照してください。

■ファイル名

```
C:¥Interstage¥ots¥etc¥ots.ini
```

注）本製品のインストールパスがデフォルトの場合のパスです。

■パラメター覧

タイプ	パラメタ	意味	初期値	最小値	最大値	チューニング
共用メモリ	shmmni	共用メモリ識別子の数	100	25	16777215	－
	shmseg	プロセスごとのセグメント数	100	25	16777215	－
セマフォ	semmni	セマフォid数 (注1)	100	25	16777215	○
	semmsl	idごとの最大セマフォ数 (注1)	25	25	16777215	○
	semvmx	セマフォ最大値	32768	32768	16777215	－
メッセージキュー	msgmap	messageマップ内のエントリ数	50	25	16777215	－
	msgmax	メッセージ最大サイズ	4096	4096	16777215	－
	msgmni	メッセージ待ち行列id数	10	10	16777215	－
	msgssz	メッセージセグメントサイズ	8	8	255	－
	msgtql	メッセージキューidごとのヘッダ数	20	20	16777215	－
	msgseg	メッセージセグメント数	2048	2048	16777215	－
Windows(R)固有パラメタ	msgemuwait	メッセージキューwaitプロセス数 (注2)	64	64	16777215	○
	insmax	内部プロセス間通信メッセージ長	32768	32768	16777215	－

タイプ	パラメタ	意味	初期値	最小値	最大値	チューニング
	msgwait	内部プロセス間通信待ち合わせ数 (注2)	64	64	16777215	○
	execmax	最大プロセス数 (注2)	64	64	16777215	○
	prntmax	最大親プロセス数 (注2)	64	10	16777215	○
	ftokmax	最大ファイル名数 (注2)	64	64	16777215	○
	interval	プロセス終了監視間隔時間(秒)	1	1	16777215	－
	inthndl	資源監視間隔時間(秒)	10	1	16777215	－
	mutexmax	最大mutex数	300	150	16777215	－

○:チューニング可能です。
－:基本的には、初期値から変更しないでください。

注1)“11.3 セマフォ資源”を参照して設定してください。
注2)“11.4 Windows(R)固有パラメタ”を参照して設定してください。

11.2 iniファイルの設定例 Windows32/64

iniファイルの設定例を以下に示します。

```
[shminfo]
#       shared memory id count
shmmni=100
#       segment count of process
shmseg=100
[seminfo]
#       semaphore id count
semmni=100
#       semaphore count of id
semmsl=25
#       maximum semaphore
semvmx=32768
[msginfo]
#       entry count in message map
msgmap=50
#       maximum message size
msgmax=4096
#       message queue count(id count)
msgmni=10
#       message segment size
msgssz=8
#       header count of system message
msgtql=20
#       message segment count
msgseg=2048
#       message queue WAIT process count
msgemuwait=64
[insinfo]
#       length of interprocess communication
insmax=32768
```

```
#       wait count of interprocess communication
msgwait=64
#       maximum process count
execmax=64
#       maximum parent process count
prntmax=64
#       maximum count of ftok file name
ftokmax=64
#       process finish observation space time
interval=1
#       space time
inthndl=10
#       mutex count
mutexmax=300
```

#で始まる行は、コメントです。
設定値には、半角数字だけ使用できます。

11.3 セマフォ資源 Windows32/64

データベース連携サービスを使用する場合、同期制御／排他制御のため、セマフォ資源を使用します。
セマフォ資源は、以下の算出式で見積ります。

セマフォ資源	意味	種類	必要数
semmni	セマフォの組	加算値	5 ＋ x(注)
semmsl	組あたりのセマフォの最大数	設定値	25 以上

注)x:起動するリソース管理プログラムの種類 × 4(リソース管理プログラムを起動する場合)

11.4 Windows(R)固有パラメタ Windows32/64

Windows(R)固有パラメタは、以下の算出式で見積ります。

パラメタ	意味	種類	初期値	必要数
msgemuwait	メッセージキューwaitプロセス数	設定値	64	10 ＋ a(注1)
msgwait	内部プロセス間通信待ち合わせ数	設定値	64	10 ＋ b ＋ c(注2)
execmax	最大プロセス数	設定値	64	10 ＋ b ＋ c(注2)
prntmax	最大親プロセス数	設定値	64	10 ＋ b ＋ c(注2)
ftokmax	最大ファイル数	設定値	64	25 ＋ d(注3)

注1)a:リソース管理プログラムの最大多重度
注2)b:リソース管理プログラムの多重度の合計、c:ワークユニットの多重度の合計(サーバアプリケーションの合計)
注3)d:リソース定義ファイルの合計

付録A　CORBAサービスの動作環境ファイル

CORBAサービスの動作環境ファイルについて説明します。各ファイルは、以下に格納されます。

格納パス

Windows32/64

C:¥Interstage¥ODWIN¥etc (インストールパスはデフォルト)

Solaris64

/etc/opt/FSUNod
(インストールパスはデフォルト、
インストール時に動作環境ディレクトリ(“Fixed configuration install directory”)として変更可能)

Linux32/64

/etc/opt/FJSVod

ファイル

(Interstage Application Server Enterprise Editionにおいて提供)

config
gwconfig
inithost/initial_hosts
queue_policy Windows32 Linux32
nsconfig
irconfig

(Interstage Application Server Standard-J Editionにおいて提供)

config
inithost/initial_hosts
queue_policy Windows32 Linux32
nsconfig
irconfig

- 上記以外のファイルは、CORBAサービスの動作環境としてカスタマイズできません。エディタなどで編集しないでください。
- 上記のファイル内に日本語は記載できません。
- システムの異常停止などに起因して動作環境ファイルなどの資源が破壊されると、CORBAサービスが正常に起動できない場合があります。
  正常に起動できない、かつ以下のメッセージが発生する場合は、資源が破壊されている可能性があるので、環境の再構築を行うか、バックアップした資源を復元してCORBAサービスの再起動を行う必要があります。
  メッセージ番号:od10400, od10402, od10404, od10406, od10504, od10509, od10510
  万が一のため、運用環境を構築したら資源のバックアップを行うことを推奨します。
  バックアップについては、“運用ガイド(基本編)”の“メンテナンス(資源のバックアップ/他サーバへの資源移行/ホスト情報の変更)”で説明されています。

A.1 config

■概要

configファイルは、CORBAサービスの各種動作環境に関する定義が格納されたファイルです。

■ファイル名

Windows32/64

C:¥Interstage¥ODWIN¥etc¥config (インストールパスはデフォルト)

Solaris64

Solarisサーバ:
/etc/opt/FSUNod/config (インストールパスはデフォルト)
Windows(R)クライアント:
C:¥Interstage¥ODWIN¥etc¥config (インストールパスはデフォルト)

Linux32/64

Linuxサーバ:
/etc/opt/FJSVod/config (インストールパスはデフォルト)
Windows(R)クライアント:
C:¥Interstage¥ODWIN¥etc¥config (インストールパスはデフォルト)

■ファイル内情報

configファイルは、以下の形式で値を設定します。

◆形式:

パラメタ名 = 設定値

半角のシャープ(#)を行の先頭に指定した場合は、その行はコメントとして扱われます。また、空行は解析時に無視されます。
コメント

◆記述例:

```
# comment
period_receive_timeout = 72
```

◆パラメタ:

以下の動作環境について、パラメタ設定値を変更することができます。

- ホスト情報に関する動作環境
- ネットワーク環境に関する動作環境
- アプリケーション資源に関する動作環境
- タイムアウト監視に関する動作環境
- セキュリティ機能に関する動作環境
- コード変換機能に関する動作環境
- 保守機能に関する動作環境
- 旧バーションの互換に関する動作環境

- パラメタ値を変更した場合、次回のCORBAサービス起動時より有効となります。
- configファイル内に日本語は記載できません。
- 下表に記載していないパラメタは初期値を変更しないでください。
- 備考欄に“必須パラメタ”と記載されているパラメタは省略することはできません。(Solaris版およびLinux版には必須パラメタはありません)
- 数値を指定するパラメタ(period_receive_timeout等)に数値以外の文字列(“abc”など)を指定した場合、“0”が設定されたものとして扱われます。

◆ホスト情報に関する動作環境

パラメタ名	初期値 省略値 指定範囲	意味	備考
IIOP_hostname	－ － －	マシンにIPアドレス(またはホスト名)が複数設定されている場合に、CORBAサーバアプリケーションで使用するIPアドレスを限定した運用を行う場合に設定します。 IPアドレス(またはホスト名)を設定すると、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生成時には、ここで設定したIPアドレスが組み込まれ、クライアントからの接続時に使用されます。また、CORBAサービスは指定されたIPアドレスのみでバインドを行います。 本パラメタが省略された場合はすべてのIPアドレスに対してバインドが行われます。ただし、IPv4/IPv6のどちらの(または両方の)IPアドレスをバインドするかどうかはIP-versionパラメタの設定に依存します。	サーバ機能のみ有効。 (注1) (注2)
IIOP_port	8002 － －	CORBAサービスが使用するポート番号。省略できません。	

注1)
例えばLANカードが複数装着されたマシンにおいて、1つのLANカードからのみ接続要求を受け付けることができます。
ホスト名が指定された場合はIP-versionの値に従って名前解決が行われます。
IP-versionがv4-dualの場合はIPv4での名前解決が優先的に行われます。
IP-versionがv6の場合はIPv6での名前解決が優先的に行われます。
Windows版においてリンクローカルのIPv6アドレスを指定する場合はscope-idも記載する必要があります。(例:fe80::1234:5678:9abc:def0%4)

注2)
必要のない限り本パラメタを設定しないでください。注1)に記述されているような特殊用途以外では設定する必要はありません。誤ったホスト名を設定するとInterstageの起動が失敗します。
また、“localhost”を設定すると“127.0.0.1”(IPv4環境の場合)のみでバインドが行われ、他ホストからのリクエストが受け付けられなくなりますので“localhost”と設定しないで下さい。“127.0.0.1”(IPv4環境の場合)のIPアドレスで定義されているホスト名を設定した場合も同様に他ホストからのリクエストが受け付けられなくなります。
LinuxではOSインストール直後の状態では自ホスト名に対するIPアドレスが127.0.0.1に設定されており、自ホスト名をIIOP_hostnameに設定すると他ホストからの接続を受け付けることができなくなります。

◆ネットワーク環境に関する動作環境

パラメタ名	初期値 省略値 指定範囲	意味	備考
con_accept	all all all, localhost	・ all: すべてのマシンからの接続を受け付けます。 ・ localhost: クライアントからの接続受付を自ホストに制限する場合に指定します。自ホストからの接続のみを受け付け、他ホストからの接続を受け付けません。 システムセキュリティなどの理由で、他ホストからの接続要求を許可しない場合に指定してください。	サーバ機能のみ有効。
IP-version	v4-dual v4-dual v4, v4-dual, v6	運用するIPバージョンを設定します。 ・ v4: IPv4のみを利用してCORBAアプリケーションを運用します(IPv6は使用しません)。 ・ v4-dual: IPv4およびIPv6を利用してCORBAアプリケーションを運用します。 サーバとして動作する際はIPv4およびIPv6の両方を受け付けます。 クライアントとしての動作する際にIPv4を優先的に利用します。 ・ v6: IPv4およびIPv6環境で、CORBAアプリケーションを運用します。 サーバとして動作する際はIPv4およびIPv6の両方を受け付けます。 クライアントとしての動作する際にIPv6を優先的に利用します。 IPv6に対応していない環境で“v4-dual”もしくは“v6”を指定した場合、“v4”が設定されます。	
read_interval_timeout	30 30 0～ 100000000	ソケットに対する読み込みの待機時間。 この時間を超えても読み込みできない状態が持続する場合、アプリケーションにシステム例外(COMM_FAILURE)が通知されます。 この値が実際の時間(秒)となります。0を設定した場合、時間監視をしません。 このパラメタによる監視は電文の受信処理が始まってから開始されます。例えば、リプライの受信待ちの状態においてパケットが1つも届かなければread_interval_timeoutによる監視は行いません。この場合はperiod_receive_timeoutによる監視が行われます。パケットが1つでも届くと電文の受信処理を開始するためread_interval_timeoutによる監視が行われます。	
write_interval_timeout	30 30 0～ 100000000	ソケットに対する書き込みの待機時間。 この時間を超えて書き込みできない状態が持続する場合、アプリケーションにシステム例外(COMM_FAILURE)が通知されます。 この値が実際の時間(秒)となります。0を設定した場合、時間監視をしません。 このパラメタによる監視は電文の送信処理が始まってから開始されます。	
tcp_nodelay	no no	TCP_NODELAY機能を有効にするか無効にするかを設定します。	

パラメタ名	初期値 省略値 指定範囲	意味	備考
	yes, no	・ yes: 電文送信時においてNagleアルゴリズムを無効にします。 ・ no : Nagleアルゴリズムは有効となります。 Nagleアルゴリズムが有効の場合、送信データのバッファリングを行うためネットワーク使用効率が上がります。Nagleアルゴリズムを無効にした場合、送信データのバッファリングを行わないためネットワーク使用効率が下がり通信全体の性能が下がる可能性がありますが、データの送受信に発生するタイムラグが減少し、応答性能が向上する場合があります。	

◆アプリケーション資源に関する動作環境(プロセス/スレッド多重度、使用コネクション数など)

これらのパラメタに実際に指定可能な値はOSの資源によって制限されます。

パラメタ名	初期値 省略値 指定範囲	意味	備考
max_exec_instance	512 (注6) 256 16～1000000	サーバアプリケーションのリクエスト実行用スレッド(またはプロセス)の最大数。	サーバ機能のみ有効。 (注1) (注2)
max_IIOP_local_init_con	256 256 1～1000000	クライアントアプリケーションが使用するサーバホストへのコネクションの最大値。	ポイント参照
max_IIOP_local_init_requests	4096 4096 1～1000000	クライアントアプリケーションが同時に送信できるリクエスト数の最大値。	
max_IIOP_resp_con	8 (注6) 8 1～500000	クライアントアプリケーションと確立できる接続の最大値。	サーバ機能のみ有効。 (注2) (注9) (注10) ポイント参照
limit_of_max_IIOP_resp_con	0 (意味参照) 0 (意味参照) 0～1000000	max_IIOP_resp_conの自動拡張の最大値。 0またはmax_IIOP_resp_con以上の値を指定してください。 0を指定すると以下の値が設定されます。 max_IIOP_resp_con × 1.3 (小数部分切り捨て)	サーバ機能のみ有効。 (注2) (注4)
max_IIOP_resp_con_extend_number	0 (意味参照) 0 (意味参照)	max_IIOP_resp_conの自動拡張の拡張数。 0を指定すると以下の値が設定されます。 (limit_of_max_IIOP_resp_con －	サーバ機能のみ有効。 (注4) (注5)

パラメタ名	初期値 省略値 指定範囲	意味	備考
	0～1000000	max_IIOP_resp_con) ÷ max_IIOP_resp_con (小数部分切り上げ)	
max_IIOP_resp_requests	128 (注6) 128 1～500000	サーバホストにおいて同時に受信できるリクエスト数の最大値。	サーバ機能のみ有効。 (注2) (注8)
limit_of_max_IIOP_resp_requests	0 (意味参照) 0 (意味参照) 0～1000000	max_IIOP_resp_requestsの自動拡張の最大値。 0またはmax_IIOP_resp_requests以上の値を指定してください。 0を指定すると以下の値が設定されます。 max_IIOP_resp_requests × 1.3 (小数部分切り捨て)	サーバ機能のみ有効。 (注2) (注4) (注8)
max_IIOP_resp_requests_extend_number	0 (意味参照) 0 (意味参照) 0～1000000	max_IIOP_resp_requestsの自動拡張の拡張数。 0を指定すると以下の値が設定されます。 (limit_of_max_IIOP_resp_requests － max_IIOP_resp_requests) ÷ max_IIOP_resp_requests (小数部分切り上げ)	サーバ機能のみ有効。 (注1) (注6) (注7)
max_processes	20 (注6) 16 1～1000000	最大プロセス数。(起動クライアント ＋ サーバ数)	サーバ機能のみ有効。 (注4) (注5)
max_impl_rep_entries	512 256 100～1000000	インプリメンテーションリポジトリの最大登録数。	サーバ機能のみ有効。
number_of_common_buffer	0 (意味参照) 0 (意味参照) 0～500000 (注8)	CORBAサービスのキュー制御で使用するデフォルトバッファのバッファ数を指定します。 ワークユニット運用されているCORBAアプリケーションでワークユニット定義の“Buffer Number:通信バッファ数”を指定しているCORBAアプリケーションを除く、CORBAサービスの通信で使用します。 サーバマシン上で同時に処理される最大リクエスト数を指定してください。 0を指定すると、以下の値が設定されます。 max_IIOP_resp_requests × 0.2 (小数部分切り捨て)	サーバ機能のみ有効。 (注2)
limit_of_number_of_common_buffer	0 (意味参照) 0 (意味参照) 0～1000000 (注8)	number_of_common_bufferの自動拡張の最大値。 0またはnumber_of_common_buffer以上の値を指定してください。 0を指定すると以下の値が設定されます。 limit_of_max_IIOP_resp_requests	サーバ機能のみ有効。 (注2) (注4)
number_of_common_buffer_extend_number	0 (意味参照) 0 (意味参照)	number_of_common_bufferの自動拡張の拡張数。 0を指定すると以下の値が設定されます。 (limit_of_number_of_common_buffer －	サーバ機能のみ有効。 (注4) (注5)

パラメタ名	初期値 省略値 指定範囲	意味	備考
	0～1000000	number_of_common_buffer) ÷ number_of_common_buffer (小数部分切り上げ)	
max_bind_instances	0 (意味参照) 0 (意味参照) 0～1000000	CORBAサービスに登録可能な「サーバプロセスとオブジェクトのバインド関係」の数。 0を指定すると以下の値が設定されます。 1024 × max_processes (計算結果が65535を超えた場合は65535)	サーバ機能のみ有効。 (注2) (注7)

注1)

設定値の目安:

Windows32/64

登録アプリケーション数(*1) × プロセス最大多重度(*2) × スレッド最大多重度(*3)＋ 接続クライアント数(*4) ＋ 64

Solaris64 Linux32/64

登録アプリケーション数(*1) × プロセス最大多重度(*2) × スレッド最大多重度(*3)＋ 接続クライアント数(*4) ＋ 172

*1) OD_impl_instコマンドで登録したアプリケーション数
*2) OD_impl_instコマンドで指定するproc_conc_max値
*3) OD_impl_instコマンドで指定するthr_conc_maximum値
*4) isgendefコマンドのscale-valueに対応した接続クライアント数

注2)

サーバ機能では、本パラメタの設定値および実際の消費量をodprtcurparamコマンドにより確認することができます。

Solaris64 Linux32/64

初期値より増加させる場合、システム資源(共用メモリ、ファイルディスクリプタなど)のチューニングが必要です。詳細については、“3.1.1 CORBAサービスのシステム資源の設定”を参照してください。

ファイルディスクリプタ数については、“max_IIOP_resp_con値 ＋ max_processes値”だけ拡張してからCORBAサービスおよびCORBAアプリケーションを起動してください。

注3)

CORBAサービスのプロセス(CORBAサービス、ネーミングサービス、インタフェースリポジトリサーバ、インタフェースリポジトリキャッシュサーバ)も含みます。見積もりを行う場合、Interstageのサービスの使用分(20)にアプリケーション使用分を加算してください。

CORBAサービスのコマンドも含みます。コマンドを同時に複数起動する場合は、その数を加算してください。

注4)

自動拡張について

自動拡張を行うパラメタについてはlimit_of_[パラメタ名]というパラメタと[パラメタ名]_extend_numberというパラメタが存在します。例えば、max_IIOP_resp_conというパラメタについてはlimit_of_max_IIOP_resp_con・max_IIOP_resp_con_extend_numberが存在します。

そして、各パラメタは初期値を[パラメタ名]、最大値をlimit_of_[パラメタ名]として、[パラメタ名]_extend_number分割で必要に応じて拡張を行います。

以下に例を示します。

```
---------------------------------------------------------------------
max_IIOP_resp_con = 100
limit_of_max_IIOP_resp_con = 140
max_IIOP_resp_con_extend_number = 2
---------------------------------------------------------------------
```

上記のパラメタの場合、max_IIOP_resp_conは初期値を100として、120、140と最大2回の拡張を行います。

なお、isconfig.xmlファイルの定義項目AutoConfigurationModeにMANUALを指定した場合、自動拡張に関するパラメタは無視され拡張は行いません。isconfig.xmlについての詳細に関しては“運用ガイド(基本編)”を参照してください。

注5)
一度の拡張処理で増加できるサイズは初期値のサイズに制限されます。
一度の拡張サイズが初期値のサイズを超える拡張を行う設定がされた場合、拡張数は0が設定された場合と同様の値に補正されます。
また、自動拡張の最大値と初期値との差分よりも拡張数が大きい場合は、拡張数は自動拡張の最大値と初期値との差分の値に補正されます。

---
max_IIOP_resp_con = 100
limit_of_max_IIOP_resp_con = 300
max_IIOP_resp_con_extend_number = 1

上記のパラメタの場合、max_IIOP_resp_con_extend_numberは2に補正されます。

注6)
以下の場合は初期値が異なります。

Solaris64

標準インストール・カスタムインストール・GUIインストーラを使用したインストールを行った場合(pkgaddコマンドによるインストールを行わない場合)。

Linux32/64

標準インストール・カスタムインストール・GUIインストーラを使用したインストールを行った場合(rpmコマンドによるインストールを行わない場合)。

初期値は以下のように変更されています。

パラメタ名	初期値
max_IIOP_resp_con	512
max_IIOP_resp_requests	2048
max_processes	512
max_exec_instance	16384

注7)
C++のCORBAアプリケーションがCORBA::ORB::bind_object関数を発行して登録するオブジェクト数と別JavaVMから呼び出されるEJBアプリケーションについてSession BeanとEntity BeanのEJB objectのインスタンス数の加算値よりも大きな値を設定してください。

注8)
Solaris64
number_of_common_bufferとlimit_of_number_of_common_bufferの設定可能な値は65535が最大値になります。
Linux32/64
number_of_common_bufferとlimit_of_number_of_common_bufferの設定可能な値はSEMVMXのOS実装値(32767)が最大値になります。

なお、number_of_common_bufferとlimit_of_number_of_common_bufferの設定を省略した場合は、max_IIOP_resp_requestsとlimit_of_max_IIOP_resp_requestsの値から求まる各パラメタの値が設定可能な最大値を超過しないか確認をお願いします。

注9)
Solaris64 Linux32/64
以下に示すOSでは、"max_IIOP_resp_con"の最大値がOSの仕様により制限されます。

- Red Hat Enterprise Linux 5以前
  65520
- Solaris
  システムパラメタのsemmslまたは、process.max-sem-nsemsの最大値に制限されます。
  システムパラメタの最大値は、Solarisのドキュメントを参照してください。

注10)

Solaris64 Linux32/64

OSのメモリ使用状況によっては、指定可能範囲であってもod10730メッセージが出力される場合があります。
od10730メッセージが出力された場合、以下の対処を行ってください。

- Interstage起動時
  Interstageを再度起動してください。
- CORBAアプリケーション運用時
  CORBAサーバアプリケーションへ再接続してください。

ポイント: max_IIOP_local_init_con、max_IIOP_resp_conについて

CORBAサービスは、サーバアプリケーションが動作しているマシンごとに1つのコネクションを使用します。
max_IIOP_local_init_conは、各アプリケーションが使用するサーバホストへのコネクション数の最大値を指定します。
max_IIOP_resp_conは、各ホストで使用するアプリケーション間のコネクション数を指定します。

原則として、アプリケーション間のコネクションはクライアントアプリケーションのプロセス単位に生成されます。例えば、クライアントアプリケーションから1つのサーバアプリケーションに複数のリクエストが同時に発行されても、コネクション数は1になります。
SSL連携機能を使用する場合、SSL接続のコネクションとSSL接続でないコネクションは別コネクションとして数える必要があります。例えば、クライアントアプリケーションから、1つのサーバマシンにSSL接続のコネクションとSSL接続でないコネクションを使用した場合、コネクション数は2になります。

なお、以下の場合にはコネクションを使用するので必要に応じて加算する必要があります。

- CORBAサービスのコマンド実行時は1コネクション使用します。コマンドを同時に複数起動する場合は、その数を加算して指定しておいてください。
- インタフェースリポジトリ動作時は、1コネクションを使用します。
- Windows32 Linux32
  ロードバランス機能の動作時は、ネーミングサービスおよびロードバランス機能はそれぞれをサーバとしたクライアントとして動作するため、2コネクションを使用します。
- インタフェースリポジトリ、ネーミングサービス、ロードバランス機能などの各サービスはサーバアプリケーションです。そのため、ネーミングサービスへ参照、登録などのリクエストを発行すると1コネクションを使用します。他ホスト上のインタフェースリポジトリ、ネーミングサービスを参照する設定の場合は、参照先ホストのIIOP_resp_conが1消費されます。例えばTYPE3 EJBで初期化したホスト上でネーミングサービスへアクセスするアプリケーションを起動すると、ネーミングサービスが起動しているホスト上のIIOP_resp_con資源が1消費されることになります。

以下に、各パラメタのコネクション数のカウント方法を示します。

max_IIOP_local_init_con:

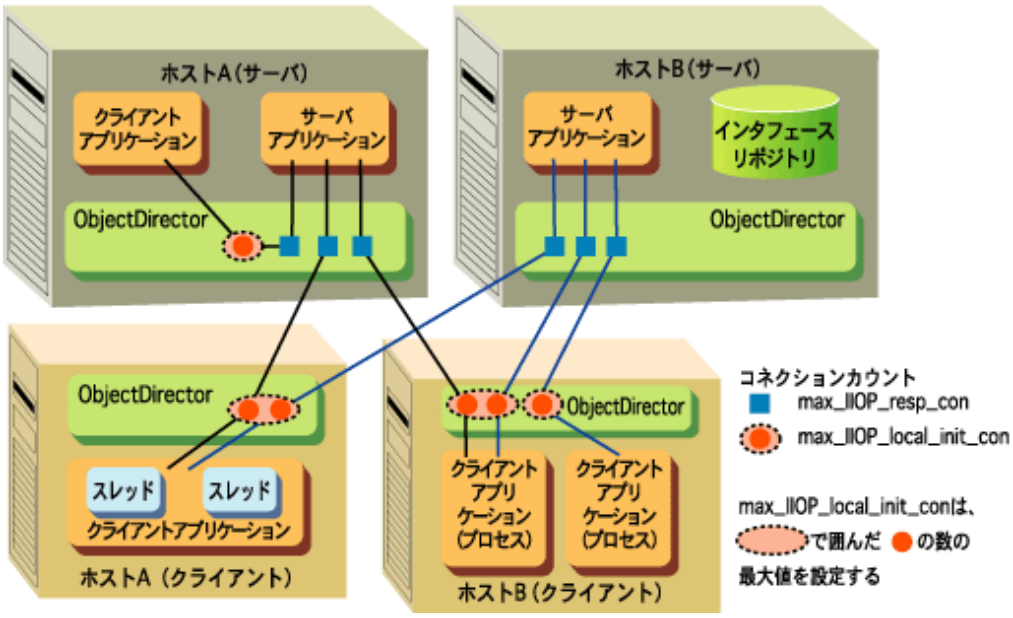
クライアントアプリケーションが動作しているホスト上で、クライアントアプリケーション(プロセス単位)からサーバアプリケーション(ホスト単位)へのコネクション数の最大値を指定します。

- **設定値の目安(インタフェースリポジトリ動作時):**
  max_IIOP_local_init_con ＝
  [1つのクライアントアプリケーションが接続するサーバホスト数の最大値]と
  256のうちの最大値
- **設定値の目安(インタフェースリポジトリ動作時、SSL連携機能を使用):**
  max_IIOP_local_init_con ＝
  [1つのクライアントアプリケーションが接続するサーバホスト数の最大値×2]と
  256のうちの最大値

max_IIOP_resp_con:

サーバアプリケーションが動作しているホスト上で、接続するクライアントアプリケーションのプロセス数の合計を指定します。同一ホスト上で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が接続する場合も、そのコネクション数を加算する必要があります。

- **設定値の目安(インタフェースリポジトリ動作時):**
  max_IIOP_resp_con ＝
  接続するクライアントアプリケーションのプロセス数 ＋ 2
- **設定値の目安(インタフェースリポジトリ動作時、SSL連携機能を使用):**
  max_IIOP_resp_con ＝
  (接続するクライアントアプリケーションのプロセス数 × 2) ＋ 2

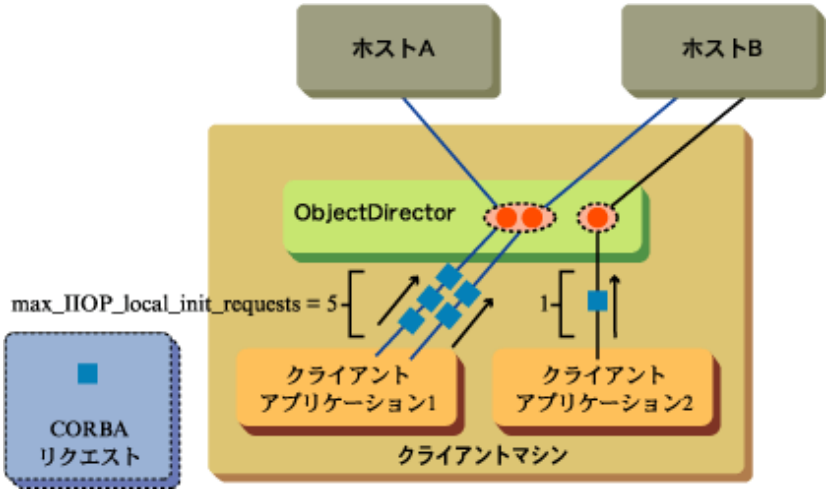


ポイント: max_IIOP_local_init_requests、max_IIOP_resp_requestsについて

CORBAサービスでは、クライアントアプリケーションが同時に送信するリクエスト数に応じてmax_IIOP_local_init_requestsを設定する必要があります。また、サーバアプリケーションが同時に受信するリクエスト数に応じてmax_IIOP_resp_requestsを設定する必要があります。

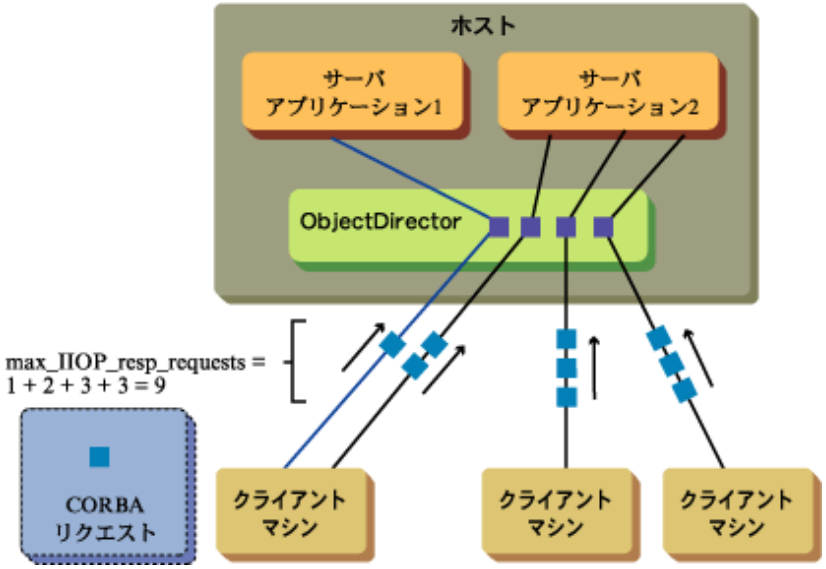
max_IIOP_local_init_requests:

クライアントアプリケーションが同時に送信できるリクエスト数の最大値を指定します。下の図ではクライアントアプリケーション1が5個のリクエストを同時に送信し、アプリケーション2が1個のリクエストを同時に送信しています。このため、max_IIOP_local_init_requestsは5以上の値を設定する必要があります。
ただし、算出された値が4096以下の場合は初期値の4096のままで問題ありません。この例では4096に満たないのでmax_IIOP_local_init_requestsはデフォルトの4096から変更の必要はありません。



max_IIOP_resp_requests:

CORBAサーバアプリケーションが同時に受信できるリクエスト数の最大値を指定します。
それぞれのクライアントマシンから発行されたリクエストがサーバマシンに到達し、CORBAサーバアプリケーションで同時に処理される数になるので、個々のクライアントマシンから同時に発行されるリクエストの合計値を見積もる必要があります。
下の図ではそれぞれのクライアントマシンから発行されたリクエストが同時に9個サーバマシンに到達しているので、max_IIOP_resp_requestsには9以上を設定する必要があります。



◆タイムアウト監視に関する動作環境

パラメタ名	初期値 省略値 指定範囲	意味	備考
period_client_idle_con_timeout	96 (480秒) 96 (480秒) 0～20000000	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)の監視時間(リクエスト返信完了後のコネクション維持時間)。 この時間を超え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の切断を行います。(注2) この値に5を乗じた値が実際の時間(秒)となります。0を設定すると無通信監視を行いません。	
period_idle_con_timeout	120 (600秒) 1 (5秒) 0～20000000	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)の監視時間(リクエスト返信完了後のコネクション維持時間)。 この時間を超え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、リクエスト処理に使用したメモリ資源を解放します。 この値に5を乗じた値が実際の時間(秒)となります。0を設定すると無通信監視を行いません。	サーバ機能のみ有効。 (注3)
period_receive_timeout	72 (360秒) 72 (360秒) 0～20000000	クライアントにおける、リクエスト送信から返信までの待機時間。この時間を超えてもサーバからの返信がない場合、クライアントにタイムアウトが通知されます。 この値に5を乗じた値が実際の時間(秒)となります。	
period_server_timeout	120 (600秒) (注1) 120 (600秒)	Persistentタイプ以外のサーバアプリケーションとその他のアプリケーションで意味が異なります。 Persistentタイプ以外のサーバアプリケーションにおいてはアプリケーション起動からCORBA_ORB_initメソッド完了までの監視時間となります。この時間以内に	サーバ機能のみ有効。

パラメタ名	初期値 省略値 指定範囲	意味	備考
	1～ 20000000	CORBA_ORB_initメソッドが完了しないとクライアントにシステム例外(NO_IMPLEMENT)が通知されます。 クライアントアプリケーションとPersistentタイプのサーバアプリケーションにおいてはCORBA_ORB_initメソッド発行からCORBA_ORB_initメソッド完了までの監視時間となります。 この値に5を乗じた値が実際の時間(秒)となります(0は指定できません)。	

注1)
初期値より減少させた場合は、インタフェースリポジトリの起動に失敗することがあります。

注2)
次回リクエスト送信時にサーバとのコネクションの再接続を行います。
なお、クライアントアプリケーションがプロセスモードの場合は、時間超過のタイミングではコネクション切断は行わず、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。

注3)
サーバ側の無通信監視時間超過によるコネクション切断のタイミングでクライアントがリクエストを発行すると、クライアントに通信異常が通知される場合があります。この問題を回避するためには、クライアント側のperiod_client_idle_con_timeoutにサーバ側のperiod_idle_con_timeoutよりも小さな値を設定してください。

タイムアウト時間は、連携するアプリケーションに適用されるタイムアウト時間を考慮して設定する必要があります。詳細は“OLTPサーバ運用ガイド”の“CORBAアプリケーション運用時のタイマ監視”を参照してください。

◆セキュリティ機能に関する動作環境(アプリケーション間通信)

パラメタ名	初期値 省略値 指定範囲	意味	備考
http_proxy	proxy_host null (値は設定されません) ―	HTTPプロキシサーバのホスト名。	(注1)
http_proxy_port	8080 0 ―	HTTPプロキシサーバのポート番号。	(注1)
http_proxy_use	no no yes, no	HTTPプロキシサーバの使用を指定。 ・ yes:使用する ・ no :使用しない	(注1)
UNO_IIOP_ssl_use	no no yes, no	SSL連携の有効/無効を選択。 ・ yes:有効 ・ no :無効	(注2)

パラメタ名	初期値 省略値 指定範囲	意味	備考
UNO_IIOP_ssl_port	4433 4433 ―	SSL連携で使用するポート番号。 UNO_IIOP_ssl_useが“yes”の場合に有効です。	

注1)
プレインストール型ランタイム(Portable-ORB以外の実行環境)でHTTPプロキシサーバを経由してHTTPトンネリングを使用する場合に指定します。http_proxy、http_proxy_portは、“http_proxy_use=yes”のときに有効であり、Webブラウザで使用しているHTTPプロキシサーバのホスト名とポート番号を指定します。

注2)
SSL接続のコネクションとSSL接続でないコネクションは別コネクションとして数える必要があります。max_IIOP_resp_con、max_IIOP_local_init_conパラメタを見積もる際には注意してください。

◆セキュリティ機能に関する動作環境(CORBAサービス資源) Solaris64 Linux32/64

パラメタ名	初期値 省略値 指定範囲	意味	備考
iss_use	no no yes, no	CORBAサービス資源のセキュリティ強化機能の有効/無効を指定。“yes”を指定すると、CORBAアプリケーションはiss_groupのグループに属するユーザ(またはroot)のみが起動可能となります。	(注1) (注2)
iss_group	root(0) root(0) ―	CORBAサービス資源のセキュリティ強化機能有効時(iss_use=yes指定)のアプリケーション動作のグループIDを指定します。	(注1) (注2) (注3) (注4)

注1)
インストール時のセキュリティ設定として“強化セキュリティモード”を選択した場合、初期値は以下のように変更されています。

パラメタ名	初期値
iss_use	yes
iss_group	インストール時に指定した“Interstage運用グループ名”

注2)
CORBAサービス資源のセキュリティ強化機能に関する設定を変更する場合、issetsecuritymodeコマンドの使用を推奨します。詳細は“セキュリティシステム運用ガイド”の“共通の対策”および“リファレンスマニュアル(コマンド編)”の“issetsecuritymode”を参照してください。

注3)
すでにシステムに登録されているグループを指定してください。
なお、指定したグループのエントリ情報の取得処理が行われます。また、指定した値に関わらず、sysとotherグループのエントリ情報の取得処理も行われます。このとき、OSの設定(nsswitch.conf)によってはLDAP等と通信する場合もあります。詳細はOSのマニュアルを参照してください。

注4)
CORBAアプリケーションの実行は、iss_groupに指定したグループに属するユーザまたはrootに限定され、他の一般ユーザは実行できなくなりますので、アプリケーションの実行ユーザに注意してください(“リファレンスマニュアル(コマンド編)”の“OD_impl_inst”を参照)。

◆コード変換機能に関する動作環境

パラメタ名	初期値 省略値 指定範囲	意味	備考
undefined_char_conversion	single single single, multi, fail	未定義文字をコード変換した場合の動作を設定します。 ・ single: デフォルト すべての未定義文字は半角のアンダースコアに変換します。(注1) ・ multi 半角の未定義文字は半角のアンダースコアに変換し、全角の未定義文字は全角のアンダースコアに変換します。 ・ fail 未定義文字の変換を行うとアプリケーションにシステム例外DATA_CONVERSIONを通知します。	(注2)

注1)
文字コードがUNICODEまたはUTF8の場合は、“multi”を指定した場合と同様に全角の未定義文字は全角のアンダースコアに変換します。

注2)
クライアントおよびサーバの両方で設定してください。コード変換機能の詳細については“OLTPサーバ運用ガイド”の“コード変換”を参照してください。

◆保守機能に関する動作環境

パラメタ名	初期値 省略値 指定範囲	意味	備考
access_log_policy	start (初期値を推奨) start start, standby	CORBAサービス起動時のアクセスログの採取/非採取の状態。 ・ start: 起動時からログ採取する ・ standby: ログ採取しない	サーバ機能のみ有効 (注1)
access_log_size	3000000 3000000 1～2147483647 (longの最大値)	アクセスログファイルの最大サイズ。(バイト単位)	サーバ機能のみ有効 (注1)
access_log_level	send_stex: recv_stex: send_userex: recv_userex: close_resp_info	アクセスログ採取レベルのキーワードを連結して指定。 (区切り文字はコロン(“:”)、空白は指定不可) “all”を指定すると、すべての採取レベルを指定したものとみなされます。	サーバ機能のみ有効 (注1) (注2)

パラメタ名	初期値 省略値 指定範囲	意味	備考
	send_stex: recv_stex: send_userex: recv_userex: close_resp_info — —		
error_log_policy	start (初期値を推奨) start start, standby	CORBAサービス起動時のエラーログの採取/非採取の状態。 ・ start: 起動時からログ採取する ・ standby: ログ採取しない	(注1)
error_log_size	3000000 3000000 1～2147483647 (longの最大値)	エラーログファイルの最大サイズ。(バイト単位)	(注1)
info_log_policy	start (初期値を推奨) start start, standby	CORBAサービス起動時のインフォメーションログの採取/非採取の状態。 ・ start: 起動時からログ採取する ・ standby: ログ採取しない	(注1)
info_log_size	3000000 3000000 1～2147483647 (longの最大値)	インフォメーションログファイルの最大サイズ。(バイト単位)	(注1)
logging	no no yes, no	内部ログの採取を指定。 ・ yes: 採取する ・ no : 採取しない	(注3)
log_file_size	10000000 -1 4096～2147483647 (longの最大値)	内部ログのファイルサイズの上限値。(バイト単位) “logging = yes”とする場合、本パラメタは省略しないでください。	(注3)

パラメタ名	初期値 省略値 指定範囲	意味	備考
process_log_policy	start (初期値を推奨) start start, standby	CORBAサービス起動時のプロセスログの採取/非採取の状態。 ・ start: 起動時からログ採取する ・ standby: ログ採取しない	(注1)
process_log_size	3000000 3000000 1～2147483647 (longの最大値)	プロセスログファイルの最大サイズ。(バイト単位)	(注1)
log_file_path	－ － －	ログファイルの出力先を絶対パスで指定します。本パラメタで指定したパスには、以下のログファイルが出力されます。 ・ アクセスログ ・ エラーログ ・ インフォメーションログ ・ プロセスログ ・ 内部ログ 本パラメタで指定したパスが存在しなかった場合、CORBAサービスの起動に失敗します。 128バイトより長いパスは指定できません。128バイトより長いパスを指定した場合、無効となります。 また、空白および“=”を含むパスは指定できません。空白または“=”を含むパスを指定した場合、その直前までが有効となります。 Windows32/64 “¥”と“/”は区別されず、共にフォルダの区切りとして使用されます。	(注1) (注3)
snap_size	40000 40000 1024～2147483647 (longの最大値)	スナップショットサイズの上限値。(バイト単位)	サーバ機能のみ有効
snap_use	yes yes yes, no	スナップショットの採取を指定。 ・ yes: 採取する ・ no : 採取しない	サーバ機能のみ有効
trace_file_synch_level	stop stop －	トレースファイルへの出力タイミングを指定。複数指定可能(セパレータは“&”)。 ・ none:odformtraceコマンド実行時のみ出力。	サーバ機能のみ有効

パラメタ名	初期値 省略値 指定範囲	意味	備考
		・ exit:アプリケーション正常終了時、終了したアプリケーションのトレース情報を出力。 ・ vanish:アプリケーション異常終了時、終了したアプリケーションのトレース情報を出力。 ・ stop:CORBAサービス終了時、すべてのアプリケーションのトレース情報を出力。 ・ loop:メモリ上に採取されたトレース情報のサイズがtrace_size_per_processを超えた場合に出力。	
trace_size_per_process	10000 10000 1024～100000000	プロセスごとのトレース情報サイズの最大値(バイト単位)。	サーバ機能のみ有効
trace_size_of_daemon	0 0 0, 1024～100000000	CORBAサービスのデーモンプロセスに対するトレース情報サイズの最大値(バイト単位)。 0を指定すると、“trace_size_per_process × 32”が設定されます。 なお、計算結果が100000000を超えた場合は100000000が設定されます。 trace_size_per_processよりも小さな値を設定した場合は、trace_size_per_processの値に補正されます。	
trace_use	yes (初期値を推奨) yes yes, no	トレース情報の採取を指定 ・ yes: 採取する ・ no : 採取しない	サーバ機能のみ有効

注1)

アクセスログ・プロセスログ・エラーログ・インフォメーションログは、“log_file_path”で指定したパスに採取されます。“log_file_path”を指定しなかった場合は、以下のパスに採取されます。

また、ディスク領域として、以下のログファイルサイズの合計分が必要となります。

ログファイル格納パス

Windows32/64

C:¥Interstage¥ODWIN¥var 配下 (インストールパスはデフォルト)

Solaris64

/var/opt/FSUNod 配下 (インストールパスはデフォルト)

Linux32/64

/var/opt/FJSVod 配下

ログファイル名とファイルサイズ

ログ名	ログファイル名	ログファイルサイズ
アクセスログ	accesslog accesslog.old	access_log_size×2

ログ名	ログファイル名	ログファイルサイズ
プロセスログ (サーバ用ライブラリ(ODSV.DLL, libOM.so)をリンクしている場合)	proclog proclog.old	process_log_size×2
プロセスログ (クライアント用ライブラリ(ODWIN.DLL)をリンクしている場合)	proclogcl proclogcl.old	process_log_size×2
エラーログ (サーバ用ライブラリ(ODSV.DLL, libOM.so)をリンクしている場合)	errlog errlog.old	error_log_size×2
エラーログ (クライアント用ライブラリ(ODWIN.DLL)をリンクしている場合)	errlogcl errlogcl.old	error_log_size×2
インフォメーションログ (サーバ用ライブラリ(ODSV.DLL, libOM.so)をリンクしている場合)	infolog infolog.old	info_log_size×2
インフォメーションログ (クライアント用ライブラリ(ODWIN.DLL)をリンクしている場合)	infologcl infologcl.old	info_log_size×2

以下のログファイルを採取するためには、ログファイル格納パスに、管理者権限グループに対する書き込みのアクセス許可が必要です。

- アクセスログ
- プロセスログ(サーバ用ライブラリをリンクしている場合)
- エラーログ(サーバ用ライブラリをリンクしている場合)
- インフォメーションログ(サーバ用ライブラリをリンクしている場合)

また、以下のログファイルを採取するためには、ログファイル格納パスに、クライアントアプリケーションを実行するユーザが所属しているグループに対する書き込みのアクセス許可が必要です。

- プロセスログ(クライアント用ライブラリをリンクしている場合)
- エラーログ(クライアント用ライブラリをリンクしている場合)
- インフォメーションログ(クライアント用ライブラリをリンクしている場合)

上記のアクセス許可がない場合はログファイルの出力に失敗しますが、この際、特にエラーメッセージなどが表示されない場合があります。このため、ログファイルを採取する際には、運用前にログファイル格納パスのアクセス許可が正しく設定されているか確認してください。

注2)

access_log_level(アクセスログ採取レベル)に指定可能なキーワードは、“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”を参照してください。

注3)

“logging=yes”を指定した場合、内部ログファイルへの出力処理に時間を要するため、CORBAサービスの性能が劣化し、CORBAアプリケーションのレスポンス性能が低下します。また、インタフェースリポジトリやネーミングサービスの起動に時間がかかります。

なお、インタフェースリポジトリおよびネーミングサービスの起動に1分以上かかった場合、Interstageの起動に失敗しますので注意してください。“logging=yes”を指定してInterstageの起動に失敗した場合は、“トラブルシューティング集”の“Interstageの起動/停止時の異常”を参照して対処を実施してください。

“logging=yes”を指定した場合、内部ログは“log_file_path”で指定したパスに採取されます。“log_file_path”を指定しなかった場合は、以下のパスに出力されます。ファイル名は“log_file_path”の指定にかかわらず共通です。

Windows32/64

パス: C:¥Interstage¥ODWIN¥var
ファイル名:
・log (log.old)
・アプリケーションごとのappNNNN.log(appNNNN.old) (NNNNは英数字)

Solaris64

パス: /var/opt/FSUNod
ファイル名: /log (log.old)

Linux32/64

パス: /var/opt/FJSVod
ファイル名: /log (log.old)

プレインストール型Javaライブラリ使用時は、上記に加えて、以下のファイルに出力されます。“log_file_path”の値は影響しません。

ー ユーザ作業ディレクトリ(Java VMのシステムプロパティのuser.dirの指す位置)配下
JVxxxxxxxxxx.log (xxxxxxxxxxは数字)
アプレット運用の場合、user.dirはJava VMの起動オプションで変更可能です。

◆旧バージョンの互換に関する動作環境

パラメタ名	初期値 省略値 指定範囲	意味	備考
msg_output_compatible	no no od10301, od10605, od10924, od10925, od10926, od10941, od11101, od60003, no	以下のメッセージの出力有無を指定します。システムログに以下のメッセージを出力する場合は、メッセージ番号を指定します。複数指定する場合は、アンパサンド(&)で区切って指定します。以下のメッセージを出力しない場合は、“no”を指定します。 ・ od10301 ・ od10605 ・ od10924 ・ od10925 ・ od10926 ・ od10941 ・ od11101 ・ od60003	

A.2 gwconfig

■概要

gwconfigファイルは、HTTPトンネリング使用時に、Webサーバで起動されるHTTP-IIOPゲートウェイの動作環境を定義するファイルです。

CORBAサービスのタイムアウト監視に関連する項目を修正した場合に、同様の項目について定義を修正する必要があります。

なお、初期値から変更しない場合、gwconfigファイルは必要ありません。

■ファイル名

Windows32/64

C:¥Interstage¥ODWIN¥etc¥gwconfig (インストールパスはデフォルト)

Solaris64

/etc/opt/FSUNod/gwconfig (インストールパスはデフォルト)

Linux32/64

/etc/opt/FJSVod/gwconfig

■ファイル内情報

gwconfigファイルは、以下の形式で値を設定します。

◆形式:

パラメタ名 = 設定値

半角のシャープ(#)を行の先頭に指定した場合は、その行はコメントとして扱われます。また、空行は解析時に無視されます。
コメント

◆記述例:

timeout_response=60

◆パラメタ:

設定値を変更することのできるパラメタを下表に示します。

パラメタ名	初期値 指定範囲	意味
timeout_response	360 0～100000000	リクエストの返信待ち時間 HTTP-IIOPゲートウェイにおける、リクエスト送信から返信までの待機時間(秒)。 この時間内にサーバメソッドから返信されないと、クライアントにタイムアウトが通知されます。クライアント側で、CORBAサービスのperiod_receive_timeout(クライアント側のリクエスト送信から返信までの待機時間。configファイルで定義)を変更した場合、このパラメタは、period_receive_timeout値以下になるよう変更する必要があります。0を指定するとタイムアウトを行いません。
timeout_session	180 0～2147483647	セション保持時間(クライアント間無通信監視時間) HTTP-IIOPゲートウェイにおける、クライアントの管理単位の保持時間(秒)。 サーバからの返信待ちがない状態で、この時間内にクライアントから新たなリクエスト送信がない場合には、クライアントの管理情報は破棄されます。 0を指定するとタイムアウトを行いません。
timeout_connection	60 0～2147483647	コネクション保持時間(サーバ間無通信監視時間) HTTP-IIOPゲートウェイにおける、無通信状態の監視時間。 サーバからの返信待ちがない状態で、この時間内にクライアントから新たなリクエスト送信がない場合には、サーバとのコネクションを切断します。0を指定するとタイムアウトを行いません。
logmode	5 1～5	HTTP-IIOPゲートウェイの内部ログ採取の有無 (注) 以下から指定します。 ・ 5:内部ログ情報を採取しません。

パラメタ名	初期値 指定範囲	意味
		• 3:リクエストデータおよびエラー発生時の情報を採取します。 • 2:“3”で採取する情報に加えて、リプライデータおよび内部処理の情報を採取します。 • 1:“2”で採取する情報に加えて、トレース情報を採取します。
max_log_file_size	1048576 1048576～10485760	ログファイルサイズ(バイト単位)

注)

- HTTP-IIOPゲートウェイの内部ログは、以下に出力されます。
  内部ログの出力先は環境変数OD_HTTPGW_HOMEまたはOD_HOMEより変更可能です。

  Windows32/64

  C:¥Interstage¥ODWIN¥var¥httpgw*_0.log(httpgw*_1.log) (*は英数字)

  Solaris64

  /opt/FSUNod/var/httpgw*_0.log(httpgw*_1.log) (*は英数字)

  Linux32/64

  /opt/FJSVod/var/httpgw*_0.log(httpgw*_1.log) (*は英数字)

- 内部ログ採取を終了するためには、gwconfigにパラメタを設定した後、Webサーバを再起動する必要があります。

- Solaris64 Linux32/64
  HTTP-IIOPゲートウェイの内部ログが出力されるディレクトリにInterstage HTTP Serverのサーバプロセスを実行するユーザ名/グループ名の書き込み権を設定してください。書き込み権がない場合、システムログにメッセージod40102が出力されます。

- 定義情報を変更した場合、次回のWebサーバ起動時より有効となります。
- gwconfigファイルの格納位置
  gwconfigファイルの格納場所は、環境変数OD_HTTPGW_HOMEまたはOD_HOMEで指定することが可能です。両方指定されている場合にはOD_HTTPGW_HOMEが優先され、指定されたディレクトリ配下のetcディレクトリに格納します。また、同ディレクトリ配下にvarディレクトリを作成してください。
- 最終行には、必ず改行を入れてください。
- 設定値に数字以外が含まれた場合、数字までを有効とします。
- SolarisまたはLinuxの場合、通信ごとにコネクションを解放します。このため、timeout_sessionおよびtimeout_connectionを設定する必要はありません(設定しても値は無視されます)。
  Windowsの場合、timeout_sessionおよびtimeout_connectionのタイムアウトの契機でコネクションを解放します。このため、これらのパラメタに0を設定すると、コネクションがリークする可能性があります。

A.3 inithost/initial_hosts

■概要

inithost/initial_hostsファイルは、ネーミングサービス、インタフェースリポジトリのホスト情報を定義するファイルです(ネーミングサービス、インタフェースリポジトリは、CORBAアプリケーション連携を行う場合に必要となる、アプリケーションの所在、インタフェース情報が登録されているサービスです)。

inithost/initial_hostsには、サービスが存在するホスト名(またはIPアドレス)とCORBAサービスのポート番号(デフォルト8002)を指定します。ホスト名・ポート番号は最大16組まで指定できます。
サービスの問い合わせは定義順に行われ、参照するサービスが存在しない場合は次行に定義されているホストに問い合わせを行います。ただしCORBAサービス起動時に名前解決に失敗したホストに対する問い合わせは、名前解決に成功した全てのホストへの問い合わせに失敗した後に行われます。
なお、ネーミングサービス・インタフェースリポジトリをローカルホスト上で動作させる場合は、ホスト名・ポート番号の設定は必要ありません。

■ファイル名

Windows32/64

C:¥Interstage¥ODWIN¥etc¥inithost (インストールパスはデフォルト)

Solaris64

Solarisサーバ:
/etc/opt/FSUNod/initial_hosts (インストールパスはデフォルト)
Windows(R)クライアント:
C:¥Interstage¥ODWIN¥etc¥inithost (インストールパスはデフォルト)

Linux32/64

Linuxサーバ:
/etc/opt/FJSVod/initial_hosts (インストールパスはデフォルト)
Windows(R)クライアント:
C:¥Interstage¥ODWIN¥etc¥inithost (インストールパスはデフォルト)

■ファイル内情報

inithost/initial_hostsファイルは、以下の形式で値を設定します。

◆形式:

ホスト名 ポート番号

半角のシャープ(#)を行の先頭に指定した場合は、その行はコメントとして扱われます。また、空行は解析時に無視されます。
コメント

◆記述例:

```
# comment
 hostname 8002
```

◆パラメタ:

設定値を変更することのできるパラメタを下表に示します。

パラメタ名	初期値	意味
ホスト名	なし	ネーミングサービス、またはインタフェースリポジトリサービスが動作しているホスト名(またはIPアドレス)を指定します。ホスト名の長さとして、64バイトまで指定できます。(注1)(注2)

パラメタ名	初期値	意味
ポート番号	なし	上記サービスが動作しているホストで定義されているCORBAサービスのポート番号を指定します。

注1)
サーバ側のイニシャルサービスやネーミングサービスに登録されているオブジェクトリファレンスに設定されているホスト名に対して名前解決(IPアドレスへの変換)が可能である必要があります。登録されているオブジェクトリファレンスの情報を参照する場合は、サーバ側でOD_or_admコマンド(-lオプション)及びodlistnsコマンド(-lオプション)を実行してください。
なお、自ホスト側/サーバ側(サービスが動作しているホスト)のホスト名定義と統一させて指定する必要があります。

[自ホスト側] Windows32/64
Windows(R)システムフォルダ¥system32¥drivers¥etc配下のlmhostsまたはhosts

[自ホスト側] Solaris64 Linux32/64
/etc/hostsまたはNIS+など

[サーバ側]
サーバ側のホスト名定義

注2)
ホスト名に自ホストのホスト名やIPアドレスは指定しないでください。自ホスト名を指定した場合、サービスの問い合わせがループする場合があります。

- 定義情報の変更について
  定義情報を変更した場合、次回のCORBAサービス起動時より有効となります。

- isinit、ismodifyserviceコマンドの実行および設定について
  - isinitおよびismodifyserviceコマンドを実行する場合は、inithost/initial_hostsファイルに指定されているインタフェースリポジトリサービスおよびネーミングサービスのホスト名を、コメントまたは削除してください。
  - inithost/initial_hostsファイルの設定は、isinitおよびismodifyserviceコマンドの実行後に可能になります。
  - inithost/initial_hostsファイルにホスト名が設定されている場合でも、isinitおよびismodifyserviceコマンドで設定したホスト名が優先されます。
    また、isinitおよびismodifyserviceコマンドで設定したホスト名は、inithost/initial_hostsファイルに設定する必要はありません。
  - isinitおよびismodifyserviceコマンドを実行して環境設定を行った場合、inithost/initial_hostsファイルを使用してネーミングサービスのリモートホスト運用を行うことはできません。それは、isinitおよびismodifyserviceコマンドを使用すると、ネーミングサービスのリモートホスト名がイニシャルサービスに設定されるため、inithost/initial_hostsファイルが使用されないためです。
  - inithost/initial_hostsファイルを使用してネーミングサービスのリモートホスト運用が行えるのは、isinitおよびismodifyserviceコマンドが含まれないCORBAサービスクライアントでの運用に限られます。

- 不要なホスト情報定義について
  Windows(R)クライアントのinithostに、存在しない、または通信不可状態のホストが定義された場合、クライアントアプリケーションの動作が遅くなることがあります。不要なホスト名は削除してください。

ホスト名・ポート番号の設定は、odsethostコマンドでも行うことができます。

A.4 queue_policy Windows32 Linux32

■概要

queue_policyファイルは、キュー制御機能のキューポリシーとして使用されるファイルです。
サンプルファイルとしてqueue_policy.defaultを提供していますので、キュー制御機能を使用する場合は、queue_policy.defaultを編集してqueue_policyファイルを作成してください。

■ファイル名

Windows32

C:¥Interstage¥ODWIN¥etc¥queue_policy (インストールパスはデフォルト)

Linux32

/etc/opt/FJSVod/queue_policy

■ファイル内情報

queue_policyには、[QUEUEGROUP]セクション、[QUEUE]セクション、[GUARANTY]セクションに分かれています。[QUEUEGROUP]セクションおよび[QUEUE]セクションは、odsetqueコマンドで更新します。
[GUARANTY]セクションはエディタなどで変更してください。[GUARANTY]セクションが未定義の場合の最大値は、“リファレンスマニュアル(コマンド編)”の“odsetque”を参照してください。

◆形式:

[GUARANTY]
キュー名 = キューの上限値

◆記述例:

[GUARANTY]
queue1 = 64

◆パラメタ:

設定値を変更することのできるパラメタを下表に示します。

パラメタ名	指定範囲	意味
キュー名 (注2)	−	odsetqueコマンドで登録したキュー名を指定します。
キューの上限値	1～2147483647 (longの最大値)	キューの上限値を指定します。(省略不可) (注1)

注1)
すべての上限値の合計が、limit_of_max_IIOP_resp_requests以下になるように設定する必要があります。

limit_of_max_IIOP_resp_requestsのデフォルト値は以下となります。

max_IIOP_resp_requests × 1.3 (小数部分切り捨て)

0が指定された場合、デフォルト値となります。

isconfig.xmlファイルの定義項目AutoConfigurationModeにMANUALを指定し、自動拡張を行わない設定にした場合は、max_IIOP_resp_requestsとなります。

注2)
CORBAサービスが持っているキューと設定値について下表に示します。

キュー名	設定値(キューの上限値)
SYSTEM_GLOBAL	変更不可
OD_ORB_QUEUE	変更不可
COS_NAMING_QUE	max_IIOP_resp_con以上の値
INTERFACE_REP_QUE	max_IIOP_resp_con以上の値

- 定義情報を変更した場合、次回のCORBAサービス起動時より有効となります。
- odsetqueコマンドで登録した際、[GUARANTY]にはキュー情報が追加されません。キューの上限値を設定する場合、新たに定義を追加する必要があります。

A.5 nsconfig

■概要

nsconfigファイルは、ネーミングサービスの動作環境を設定するファイルです。

■ファイル名

Windows32/64

C:¥Interstage¥ODWIN¥etc¥nsconfig (インストールパスはデフォルト)

Solaris64

/etc/opt/FSUNod/nsconfig (インストールパスはデフォルト)

Linux32/64

/etc/opt/FJSVod/nsconfig

■ファイル内情報

nsconfigファイルは、以下の形式で値を設定します。

◆形式:

パラメタ名 = 設定値

パラメタ名と設定値の間の文字列は、“ = ”(半角スペース＋半角イコール＋半角スペース)を設定します。
半角のシャープ(#)を行の先頭に指定した場合は、その行はコメントとして扱われます。また、空行は解析時に無視されます。
コメント

◆記述例:

```
file_sync = yes
trace_level = update
bl_how_many = 65536
ogl_how_many = 256
ext_intf = yes
```

cn_userexception_log_use = yes
cn_userexception_log_size = 2000000

◆パラメタ:

設定値を変更することのできるパラメタを下表に示します。なお、指定が必須となるパラメタはありません。

パラメタ名	初期値 指定範囲	意味
file_sync	yes yes, no	ネーミングサービスのデータベースへの更新処理でファイルの同期書き込みを行うかを設定します。 ・ yes: ファイルの同期書き込みを行う。 ・ no : ファイルの同期書き込みを行わない。 初期創成などの大量データの更新時に、このパラメタをnoにすることにより処理を高速化することができます。ネーミングサービスの運用中には、信頼性を向上させるため、本パラメタはyesにしてください。
trace_level	update update, all	メソッド実行の自動トレースを採取するトレースレベルを指定します。 ・ update: 更新ログだけを採取する。 ・ all: すべてのトレースを採取する。
bl_how_many	65536 0～65536	NamingContext::list, BindingIterator::next_nで返されるバインディング数の最大値を指定します。
Windows32 Linux32 ogl_how_many	256(初期値を推奨) 128～256	ロードバランスオブジェクトグループのリストの最大値を指定します。 注) ネーミングサービスでは、ロードバランスオブジェクトグループのリストを作成する際、本パラメタの設定値によりメモリを獲得します。使用するメモリ容量に影響が発生するため、メモリ容量を考慮して必要最小限の値を設定してください。
ext_intf	yes yes, no	ネーミングサービスの拡張機能を使用するかを指定します。 ・ yes: ネーミングサービスの拡張機能を使用する。 ・ no : ネーミングサービスの拡張機能を使用しない。 noを指定すると、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)を使用できません。V2.X以前のクライアントアプリケーションを動作させる場合は、noを指定する必要があります。
cn_userexception_log_use	yes yes, no	ユーザ例外ログを採取するかを指定します。 ・ yes: ユーザ例外ログを採取する。 ・ no : ユーザ例外ログを採取しない。
cn_userexception_log_size	2000000 1000～2000000	ユーザ例外ログのファイルサイズを指定します。

・ 変更した値は、次回のネーミングサービス起動時より有効となります。

・ パラメタext_intfに“no”を設定してV2.X以前のクライアントアプリケーションを動作させる場合は、運用に応じた対処が必要となる場合があります。V2.X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は、“アプリケーション作成ガイド(CORBAサービス編)”の“旧バージョンからの移行上の注意”－“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)”を参照してください。

A.6 irconfig

■概要

irconfigファイルは、インタフェースリポジトリのバックアップやログ情報などの動作環境を設定するファイルです。

■ファイル名

Windows32/64

C:¥Interstage¥ODWIN¥etc¥irconfig (インストールパスはデフォルト)

Solaris64

/etc/opt/FSUNod/irconfig (インストールパスはデフォルト)

Linux32/64

/etc/opt/FJSVod/irconfig

■ファイル内情報

irconfigファイルは、以下の形式で値を設定します。

◆形式:

パラメタ名 = 設定値

半角のシャープ(#)を行の先頭に指定した場合は、その行はコメントとして扱われます。また、空行は解析時に無視されます。
コメント

◆記述例:

```
auto backup = no(yes)
auto backup path =
auto recovery = no(yes)
logging = no(yes)
logging memory size = 512
logfile path =
sync = no
select cache obj =
```

◆パラメタ:

設定値を変更することのできるパラメタを下表に示します。なお、指定が必須となるパラメタはありません。

パラメタ名	初期値 指定範囲	意味
auto backup	no yes, no	インタフェースリポジトリ起動時に自動的にバックアップを行うかを指定します。 ・ yes: 自動的にバックアップを行う。 ・ no : 自動的にバックアップを行わない。

パラメタ名	初期値 指定範囲	意味
		注）バックアップは、インタフェースリポジトリ起動時に1回だけ行います。
auto backup path	－ －	バックアップデータの格納場所を指定します。auto backup=yesと指定した場合、必ずパスを指定する必要があります。パスを指定しないとバックアップは行われません。 注）格納場所には、作成したデータベースサイズ以上の空き領域が必要です。
auto recovery	no yes, no	トランザクション処理中のシステムダウンなどによりデータベースの異常を検出した場合に、バックアップデータを元に自動的にリカバリを行うかを指定します。 ・ yes: 自動的にリカバリを行う。 ・ no : 自動的にリカバリを行わない。 注）本機能を使用する場合は“auto backup=yes”と指定し、auto backup pathを指定する必要があります。
ir_timeout	1800(秒) 0～1000000000(秒)	IDLコンパイル(IDLc)およびインタフェース情報移入(odimportir)において、インタフェースリポジトリへのリクエストの復帰までの待機時間を指定します。0を指定すると、リクエスト復帰までの待機時間は監視されません。
Solaris64 Linux32/64 iss_use	no yes, no	資源保護機能の有効/無効を指定します。 ・ yes: 資源保護機能を有効とする。 ・ no : 資源保護機能を無効とする。 “yes”を指定すると、インタフェースリポジトリはデータベース管理者(デフォルト:root)のみが運用可能となります。 注）インストール時のセキュリティ設定として“強化セキュリティモード”を選択した場合、初期値は“yes”になります。
logging	no yes, no	トラブル発生時にログ情報を採取するかを指定します。 ・ yes : ログ情報を採取する。 ・ no : ログ情報を採取しない。 採取したログは、irlogdumpコマンドによりファイルに出力できます。通常は、初期値(no)で運用します。
logging memory size	512(KB) 1～4096(KB)	ログ情報を格納する共用メモリのサイズを指定します。logging=noとした場合、この値は意味を持ちません。
logfile path	－ －	“logging=yes”とした場合、irlogdumpコマンドにより出力されるログ情報の格納ディレクトリをフルパスで指定します。パスを指定しない場合は、CORBAサービスの動作ディレクトリと同一のディレクトリに格納されます(“A.1 config”参照)。 “logging=no”と指定した場合、この値は意味を持ちません。
select cache obj	－ －	インタフェースリポジトリ起動時にキャッシュ対象とするオブジェクトを指定します。 キャッシュ対象オブジェクトは、テキストファイル内にキャッシュ対象オブジェクトのリポジトリIDを記述し、そのファイル名をフルパスで指定します。 ファイル名が指定されない場合は、全登録オブジェクトがキャッシュ対象となります。

パラメタ名	初期値 指定範囲	意味
		ファイルの作成例および注意事項については、以下のキャッシュ対象オブジェクトの指定方法を参照してください。
sync	no yes, no	同期モードを指定します。 yes:同期モードを指定します。 no:同期モードを指定しません。 “no”を指定すると、データベースへの書き込みと同期しないモードで動作するため、インタフェースリポジトリ更新処理のスループットが向上します。ただし、更新中のシステムダウンなどで発生するデータベース破壊を認識できない場合があります。 “yes”を指定すると、トランザクション単位での書き込みを保証するため、同期モードで動作します。信頼性が要求されるシステムを構築する場合に設定します(データベース破壊の認識可)。 同期モードを指定した場合、指定しない場合に比べて更新処理のスループットは低下します。このとき、クライアントへのサーバメソッドの復帰時間が長くなるため、タイムアウトが発生する可能性があります。これを防ぐには、“A.1 config”のperiod_receive_timeout値をチューニングしてください。

◆キャッシュ対象オブジェクトの指定方法

インタフェースリポジトリ起動時、キャッシュ対象オブジェクトを限定することにより、インタフェースリポジトリに大量のオブジェクトが登録されている場合の起動性能を改善することができます。
ただし、キャッシュ対象オブジェクトを指定した場合は、キャッシュ対象としないオブジェクトに対する参照性能は低下するため、運用に関しては注意が必要です。
また、キャッシュ対象オブジェクトを指定してインタフェースリポジトリを起動した場合は、以後、インタフェースリポジトリに対する登録/更新(IDLc, tdc, odimportir)を行うことができません。運用時(インタフェースリポジトリの登録/更新を行わない)に使用してください。

キャッシュ対象オブジェクトは、ユーザがテキストファイル内にキャッシュ対象オブジェクトのリポジトリIDを記述します。
リポジトリIDは、Repositoryオブジェクト(ルートオブジェクト)に直接包含されるModuleDefオブジェクトまたはInterfaceDefオブジェクトのみ指定可能で、指定されたオブジェクトに包含されるオブジェクトすべてがキャッシュ対象となります。
キャッシュ対象オブジェクトが継承またはスコープ参照で他のモジュールと関連付けられている場合は、そのモジュールもキャッシュ対象として指定する必要があります。
インタフェースリポジトリで管理するオブジェクトの種類、およびインタフェースリポジトリオブジェクトの包含/継承関係については“アプリケーション作成ガイド(CORBAサービス編)”の“インタフェースリポジトリサービスのプログラミング”を参照してください。
なお、インタフェースリポジトリに登録されているオブジェクトと包含関係については、odlistirコマンドで表示できます。

キャッシュ対象オブジェクトを指定するためのファイルの記述例を以下に示します。

```
IDL:testmodule1:1.0
IDL:testmodule2:1.0
IDL:testmodule3:1.0
```

注) 1行に、キャッシュ対象オブジェクトのリポジトリIDを1つだけ記述できます。
コメントは、使用できません。

- 変更した値は、次回のインタフェースリポジトリサービス起動時より有効となります。
- ismodifyserviceで環境設定を行う際に“auto backup”が指定されていた場合は、データベースが空の状態でバックアップが行われます。この状態でバックアップされたデータベースは、使用できません。

- auto backup機能はインタフェースリポジトリ起動時の状態でバックアップが行われます。起動後に更新されたインタフェース定義情報はバックアップに反映されません。起動後に更新された情報が必要な場合は、Interstage(インタフェースリポジトリ)を再起動するか、またはodbackupsysコマンドを使用してバックアップを行ってください。
- odbackupsysコマンドでは、キャッシュ対象オブジェクトを指定したファイルをバックアップできません。ファイルのコピーコマンドなどでバックアップを行ってください。
- インタフェースリポジトリのデータベース管理者がroot(デフォルト)以外であり、かつirconfigファイルに“iss_use=yes”を設定すると、ログ情報採取機能の有効化(irconfigファイルの“logging=yes”)を指定しても、インタフェースリポジトリ(データベースアクセス機能)のログ情報が採取されなくなります(キャッシュサーバのログ情報は採取されます)。
  また、irconfigファイルに"iss_use=yes"を設定した場合は、irlogdumpコマンド(ログ情報の出力・制御)は管理者権限(root)で実行する必要があります。

付録B　コンポーネントトランザクションサービスの環境定義

コンポーネントトランザクションサービスの環境定義ファイルについて説明します。本定義ファイルは、以下に格納されます。

格納パス

Windows32/64 (インストールパスはデフォルト)

```
C:¥Interstage¥etc¥sysdef
```

Solaris64 (インストールパスはデフォルト)

```
/var/opt/FSUNtd/etc/sysdef
```

Linux32/64

```
/var/opt/FJSVtd/etc/sysdef
```

本定義ファイルは、コンポーネントトランザクションサービスが停止している時に、変更可能です。
コンポーネントトランザクションサービスの環境定義の記述形式と制御文について説明します。

注意

万一のため、運用環境を構築したら資源のバックアップを行うことを推奨します。バックアップについては、“運用ガイド(基本編)”の“メンテナンス(資源のバックアップ/他サーバへの資源移行/ホスト情報の変更)”を参照してください。

B.1　記述形式

通常ファイル記述形式は、以下の構文により構成されます。なお、通常ファイルの記述形式に誤りがあった場合、構文エラーとなります。構文エラー時には、そのファイルに記述されているすべての内容が無効となります。

- ステートメント
- セクション
- コメント行
- 空行

B.1.1　ステートメント

ステートメントとは、情報を設定するための行であり、以下の形式で記述します。

```
キーワード：設定内容(¥n)
```

ステートメントは、キーワード、“:”(コロン)、および設定内容から構成されています。
ステートメントの記述規則を以下に示します。

- ステートメントを省略する場合、対象ステートメントを削除するか、設定内容だけ省略します。
- ステートメントを記述している行にコメントは記述できません。

ステートメントを構成する情報の詳細を以下に説明します。

■キーワード

固有のキーワードを設定します。キーワードには、以下の規則があります。

・ キーワードは対象行の先頭の英数字から“:”(コロン)の直前までを意味します。
・ キーワードには英数字で始まる英数字とスペースで構成されている文字列を指定できます。
・ キーワード中に指定されている英字の大文字/小文字の区別はしません。
・ キーワード中にスペースを含むことができます。
・ キーワード中に連続したスペースが指定された場合、1つのスペースが指定されたものとみなされます。
・ 対象行の先頭にスペースやタブが指定されている場合、そのスペースやタブは無視されます。

■:(コロン)

キーワードと設定内容の区切りをあらわす文字として使用し、以下の規則があります。

・ コロンは必ず半角で指定する必要があります。
・ コロンの前後にスペースやタブが記述されている場合、そのスペースやタブは無視されます 。

■設定内容

キーワードに対応する内容を設定します。設定内容には、以下の規則があります。

・ 設定内容は、対象行の“:”の直後から指定できます。それ以降の“:”は設定内容に含まれる文字として扱われます。
・ 設定内容の終わりはスペース、タブ、改行('¥n')、またはEOFで示します。
・ 設定内容に指定されている英字の大文字/小文字は区別されます。
・ 設定内容は、1つの文字列しか指定できません。
・ 設定内容中にスペースやタブが含まれる場合、二重引用符で囲む必要があります。
・ 設定内容を複数記述する場合、ステートメントを繰り返し記述します。

ステートメントの設定例

例1)キーワード“Keyword:”には“Information”を設定します。

```
Keyword: Information(¥n)
KEYWORD: Information(¥n)
KeyWord: Information(¥n)
Keyword: Information(¥n)
Keyword: Information(¥n)
```

以上のステートメントは、すべて同じように解析されます。

例2)キーワード“This is a Keyword:”には“Information Area”を設定します。

```
This is a Keyword: "Information Area"(¥n)
THIS IS A KEYWORD: "Information Area"(¥n)
This is a keyword: "Information Area"(¥n)
This is a Keyword: "Information  Area"(¥n)
```

以上のステートメントはすべて同じように解析されます。

構文エラーとなるステートメントの例

設定内容が2つ指定されている(¥n)

```
Keyword:  Information  Area(¥n)
```

ステートメントを記述している行にコメントが指定されている(¥n)

```
Keyword:  Information   # This is a Statement(¥n)
```

二重引用符で終了していない(¥n)

```
Keyword: "START  Information. (¥n)
```

キーワードと設定内容が2行で指定されている(¥n)

```
Keyword: "START  Information. (¥n)
Information  END" (¥n)
```

また、登録されていないキーワードを指定した場合も構文エラーとなります。

B.1.2 セクション

セクションとは、ステートメントの集合(グループ)であり、以下の形式で記述します。

```
[セクション名](¥n)
        キーワード:  設定内容 (¥n)
        キーワード:  設定内容 (¥n)
```

セクションは、“[セクション名]”と複数のステートメントから構成されており、以下の規則があります。

- セクションとは“[セクション名]”で始まり、次のセクション、またはEOFまでを意味します。
- セクションを省略する場合、セクションをすべて削除するか、コメントにする必要があります。
- “[セクション名]”だけのセクションは指定できません。
- “[セクション名]”を記述する行に“[セクション名]”以外は記述できません。
- セクション名は必ず[ ](角括弧)で囲む必要があります。
- セクション名には英数字で始まる英数字とスペースで構成されている文字列を指定できます。
- セクション名に指定されている英字の大文字/小文字の区別はしません。
- “[セクション名]”を記述している行にコメントは指定できません。

正常なセクションの設定例

セクションにステートメントが1つある(¥n)

```
[Section] (¥n)
  Keyword:  Information(¥n)
```

セクションにステートメントが3つある(¥n)

```
[Section] (¥n)
  Keyword1: Information(¥n)
  Keyword2: Information(¥n)
  Keyword3: Information(¥n)
```

構文エラーとなるセクションの例

[セクション名]が記述されている行にコメントが指定されている(¥n)

```
[Section]           #  This is a Section(¥n)
  Keyword:  Information(¥n)
```

[セクション名]の角括弧が正しく終了していない(¥n)

```
[Section(¥n)
  Keyword:  Information.  (¥n)
```

また、登録されていないセクション名を指定した場合も構文エラーとなります。

B.1.3 コメント行

コメント行は、通常ファイル中にコメント(注釈)を記述する時に使用します。以下の形式で記述します。

```
# コメント(¥n)
```

コメント行には、以下の規則があります。

- コメント行の先頭に#(シャープ)を指定します。
- #は半角文字で指定する必要があります。

B.1.4 空行

空行を記述することができます。
空行は、解析時無視されます。

B.2 環境定義ファイルの制御文

B.2.1 [SYSTEM ENVIRONMENT]セクション

◆形式

[SYSTEM ENVIRONMENT]セクションは以下の形式で記述します。なお、本セクションは省略できません。

Windows32/64
[SYSTEM ENVIRONMENT]

System Scale:	システムの規模
Using Session Information Management Object: Windows32	SMOの使用の有無
Name of Session Information Management Object: Windows32	SMOのオブジェクト名
Number of Maximum WRAPPER Hold Session: Windows32	システム最大保留セション数
Number of Communication Buffer: Windows32	トランザクションアプリケーションの通信バッファ数
Using Interface Check:	インタフェース情報チェック機能使用の有無

Solaris64
[SYSTEM ENVIRONMENT]

System Scale:	システムの規模
Using Interface Check:	インタフェース情報チェック機能使用の有無
IP version:	使用するネットワークのバージョン

Linux32/64
[SYSTEM ENVIRONMENT]

System Scale:	システムの規模
Using Session Information Management Object: Linux32	SMOの使用の有無
Name of Session Information Management Object: Linux32	SMOのオブジェクト名
Number of Communication Buffer: Linux32	トランザクションアプリケーションの通信バッファ数
Using Interface Check:	インタフェース情報チェック機能使用の有無

◆設定内容

System Scale:システムの規模

システムの規模を指定します。
Interstage統合コマンドにより初期化を行った場合には、適切なシステム規模が設定されますので、修正する必要はありません。
システム規模は、接続クライアント数より選択します。各定義値の意味を、以下に示します。

設定値	システム規模	接続クライアント数	
		Windows32/64	Solaris64 Linux32/64
small	小規模	1～5	1～50
moderate	中規模	6～10	51～100
large	大規模	11～50	101～500
super	超大規模	51～100	501～1000

Using Session Information Management Object:セション情報管理機能の使用の有無 Windows32 Linux32

セション情報管理機能の使用の有無を指定します。

- YES: セション情報管理機能を使用する。
- NO : セション情報管理機能を使用しない。省略値。

Name of Session Information Management Object:セション情報管理機能のオブジェクト名 Windows32 Linux32

セション情報管理機能のネーミングサービス登録時の名前を指定します。
指定できる値は、OD_or_admコマンドの-nオプションに指定可能な255バイト以内の値です。ただし、ネーミングコンテキストの指定はできません。省略値は、“ISTD::SMO”です。
Using Session Information Management Object ステートメントに“NO”が指定された場合は、本ステートメントは無視されます。

Number of Maximum WRAPPER Hold Session:システム最大保留セション数 Windows32

システム全体でのセション保留数を指定します。
0を指定した場合にはセションの抑制は行いません。
本定義は、[WRAPPER]セクションで指定するPSYS Nameステートメントおよび Number of Maximum Sessionステートメントを定義した場合に有効となります。
指定できる値は、0～1000です。省略値は、0です。

Number of Communication Buffer:トランザクションアプリケーションの通信バッファ数 Windows32 Linux32

トランザクションアプリケーションに対する通信時に使用するバッファの数です。本ステートメントで指定した通信バッファ数に、4096byteを積算したサイズの通信バッファが用意されます。
指定できる値は、500～10000です。
1回の通信データ長×同時接続クライアント数を計算し、その値をもとに通信バッファ数を見積もってください。
通信バッファは、十分なサイズを用意してください。
なお、システムスケールにより、以下の省略値が設定されます。

small	500
moderate	1000
large	1500
super	2000

Using Interface Check:インタフェース情報チェック機能使用の有無

インタフェース情報チェック機能使用の有無を指定します。

- YES: インタフェース情報チェック機能を使用する。
- NO : インタフェース情報チェック機能を使用しない。省略値。

IP version:使用するネットワークのバージョン Solaris64

使用するネットワークのバージョンを指定します。

- v6: IPv6ネットワークを使用する。
- v4: 従来のIPv4ネットワークを使用する。省略値。

B.2.2 [WRAPPER]セクション Windows32

◆形式

[WRAPPER]セクションは以下の形式で記述します。

[WRAPPER]

PSYS Name:	負荷抑制対象DPCF通信パス名
Number of Maximum Session:	最大同時通信セション数

◆設定内容

PSYS Name:負荷抑制対象DPCF通信パス名

負荷抑制の対象とするDPCF通信パス名を8文字以内の英数字で指定します。
DPCF通信パス名については、「IDCMヘルプ」を参照してください。

Number of Maximum Session:最大同時通信セション数

PSYS Nameステートメントで設定したDPCF通信パスにおける同時に通信できるセション数の最大値を指定します。
指定できる値は、1～1000です。この値を超えたセションが、負荷抑制の対象となります。

注意

- 負荷抑制機能を使用する場合、[SYSTEM ENVIRONMENT]セクションと[WRAPPER]セクションの各ステートメントをすべて定義してください。
- [WRAPPER]セクションのPSYS NameステートメントとNumber of Maximum Sessionステートメントはペアで指定します。
- 複数のDPCF通信パスに対して負荷抑制を行う場合には、複数の[WRAPPER]セクションにPSYS NameステートメントとNumber of Maximum Sessionステートメントをペアで指定します。
- Number of Maximum Sessionステートメントに指定する値は、IDCMネットワーク定義で指定した優先会話コネクション数以下の値を指定してください。優先会話コネクション数より大きな値を指定した場合、負荷抑制されないことがあります。IDCMネットワーク定義の詳細は、「IDCMヘルプ」を参照してください。

EE

付録C　データベース連携サービスの環境定義

データベース連携サービスの環境定義について説明します。

C.1　configファイル

■概要

configファイルは、OTSシステム起動時に反映される情報を管理している定義ファイルです。

注意

- configファイルの修正を反映させるには、OTSシステムを再起動する必要があります。
- 本ファイルで編集した値をInterstage管理コンソールから参照するためには、Interstage管理コンソールを再起動する必要があります。Interstage管理コンソールの起動・停止方法は、“運用ガイド(基本編)”の“Interstage管理コンソールによるInterstage運用”－“Interstage管理コンソールの起動・停止”を参照してください。

■ファイル名

configファイルは、インストール時に、以下に格納されます。

Windows32/64 (インストールパスはデフォルト)

```
C:¥Interstage¥ots¥etc¥config
```

Solaris64 (インストールパスはデフォルト)

```
/opt/FSUNots/etc/config
```

Linux32/64

```
/opt/FJSVots/etc/config
```

■ファイル内情報

◆形式:

```
キー名=設定値
```

◆設定例

```
OBSERVE_CYCLE_TIME=6 （注）
TRAN_TIME_OUT=300
2PC_TIME_OUT=60
COM_RETRY_TIME=2 （注）
COM_RETRY_MAX=3 （注）
RECOVER_RETRY_TIME=30 （注）
RECOVER_RETRY_MAX=60 （注）
RESOURCE_TRANMAX=5
OTS_TRACE_SIZE=4096 （注）
RESOURCE_TRACE_SIZE=4096 （注）
RECOVERY_TRACE_SIZE=4096 （注）
OBSERVE_TRACE_SIZE=4096 （注）
DATABASE_RETRY_TIME=5 （注）
```

```
DATABASE_RETRY_MAX=5 （注）
MEM_RETRY_TIME=5 （注）
MEM_RETRY_MAX=5 （注）
RSCSTOP_CHECK_COUNT=100 （注）
OTS_VERSION=5 （注）
JTS_VERSION=5 （注）
TRACE_MODE=1
TRACE_LEVEL=1
JAVA_VERSION=14
PATH=C:¥Interstage¥JDK8¥bin¥java.exe....(Windows(R)の場合)
PATH=/opt/FJSVawjbk/jdk8/bin/java.......(Solaris/Linuxの場合)
```

注)インストール時に作成されたconfigファイルには、該当項目は記載されていません。項目を指定すると、値は有効となりますが、省略値から変更しないことを推奨します。

ポイント

- タイムアウトの詳細については、“OLTPサーバ運用ガイド”を参照してください。
- configファイルの項目は、すべて省略可能です。省略時は、省略値が有効となります。

◆キー一覧

キー	意味
OBSERVE_CYCLE_TIME	監視周期の指定
TRAN_TIME_OUT	トランザクションタイムアウト検出時間の指定
2PC_TIME_OUT	フェーズ間タイムアウト検出時間の指定
COM_RETRY_TIME	トランザクション処理エラー時のリトライ間隔指定
COM_RETRY_MAX	トランザクション処理リトライ上限回数の指定
RECOVER_RETRY_TIME	OTSシステムリカバリ処理リトライ間隔指定
RECOVER_RETRY_MAX	OTSシステムリカバリ処理リトライ上限回数の指定
RESOURCE_TRANMAX	1リソース管理プログラムのトランザクションの最大多重度
OTS_TRACE_SIZE	OTSシステムのトレースログサイズ指定
RESOURCE_TRACE_SIZE	リソース管理プログラムのトレースログサイズ指定
RECOVERY_TRACE_SIZE	リカバリプロセスのトレースログサイズ指定
OBSERVE_TRACE_SIZE	監視プロセスのトレースログサイズ指定
DATABASE_RETRY_TIME	データベースシステムアクセスのリトライ間隔指定
DATABASE_RETRY_MAX	データベースシステムアクセスのリトライ上限回数指定
MEM_RETRY_TIME	OTSシステム処理中のエラーでのリトライ間隔指定
MEM_RETRY_MAX	OTSシステム処理中のエラーでのリトライ上限回数指定
RSCSTOP_CHECK_COUNT	通常停止からのトランザクション待ち合わせ回数指定
OTS_VERSION	OTSのバージョン
JTS_VERSION	JTSのバージョン
JAVA_VERSION	JDK/JREのバージョン
PATH	JDK/JREのパス
TRACE_MODE	トレースの出力形式
TRACE_LEVEL	トレースの出力レベル

◆キー詳細

OBSERVE_CYCLE_TIME:監視周期の指定

OTSシステムが監視を実施する周期(秒単位)を指定します。以下の点に注意して設定してください。

ー 本値に小さい値を設定すると、頻繁に監視が行われるため、システムのパフォーマンスが低下します。

ー 本値に大きい値を設定すると、監視の間隔が長くなるため、異常の検出が遅くなります。

指定可能な範囲は、1～60です。省略値は、5です。

TRAN_TIME_OUT:トランザクションタイムアウト検出時間の指定

データベース連携サービスがトランザクションタイムアウト(beginからcommitまで)を検出する時間(秒単位)を指定します。
アプリケーションにおいてset_timeoutメソッドでタイムアウト時間を設定した場合は、アプリケーションの設定が有効となります。
指定可能な範囲は、1～2147483647です。省略値は、300です。

2PC_TIME_OUT:フェーズ間タイムアウト検出時間の指定

OTSシステムがトランザクションの2PC(2フェーズコミット)で、リソース管理プログラムにおいて1フェーズと2フェーズ間のタイムアウトを検出する時間(秒単位)を指定します。
指定可能な範囲は、1～2147483647です。省略値は、60です。

注意

CORBAサービスのクライアント側の無通信監視時間(CORBAサービスの動作環境ファイルのパラメタ“period_client_idle_con_timeout”の設定値×5)が“0”ではない場合、本値にはこの値よりも小さい値を設定してください。

COM_RETRY_TIME:トランザクション処理エラー時のリトライ間隔指定

トランザクション処理で通信異常などのエラーが発生した場合に、その通信をリトライする間隔(秒単位)を指定します。
指定可能な範囲は、1～600です。省略値は、2です。

COM_RETRY_MAX:トランザクション処理リトライ上限回数の指定

トランザクション処理で通信異常などのエラーが発生した場合に、その通信をリトライする上限回数を指定します。
指定可能な範囲は、1～2147483647です。省略値は、3です。

RECOVER_RETRY_TIME:OTSシステムリカバリ処理リトライ間隔指定

OTSシステムのリカバリ処理で通信異常などのエラーが発生した場合に、その通信をリトライする間隔(秒単位)を指定します。
指定可能な範囲は、1～600です。省略値は、30です。

RECOVER_RETRY_MAX:OTSシステムリカバリ処理リトライ上限回数の指定

OTSシステムのリカバリ処理で通信異常などのエラーが発生した場合に、その通信をリトライする上限回数を指定します。
指定可能な範囲は、1～2147483647です。省略値は、60です。

RESOURCE_TRANMAX:1リソース管理プログラムのトランザクションの最大多重度

1リソース管理プログラムのトランザクションの最大多重度を指定します。
指定可能な範囲は、1～2147483647です。省略値は、5です。

ポイント

OTSシステムのスレッド多重度と1リソース管理プログラムのトランザクションの最大多重度は、以下の関係を保つように設定してください。

OTSシステムのスレッド多重度 ＝< 1リソース管理プログラムのトランザクションの最大多重度

OTS_TRACE_SIZE:OTSシステムのトレースログサイズ指定

OTSシステムのトレースログサイズ(Kバイト単位)を指定します。
指定可能な範囲は、128～2147483647です。省略値は、4096(Kバイト)です。

RESOURCE_TRACE_SIZE:リソース管理プログラムのトレースログサイズ指定

リソース管理プログラムのトレースログサイズ(Kバイト単位)を指定します。
指定可能な範囲は、128～2147483647です。省略値は、4096(Kバイト)です。

RECOVERY_TRACE_SIZE:リカバリプロセスのトレースログサイズ指定

リカバリプロセスのトレースログサイズ(Kバイト単位)を指定します。
指定可能な範囲は、128～2147483647です。省略値は、4096(Kバイト)です。

OBSERVE_TRACE_SIZE:監視プロセスのトレースログサイズ指定

監視プロセスのトレースログサイズ(Kバイト単位)を指定します。
指定可能な範囲は、128～2147483647です。省略値は、4096です。

DATABASE_RETRY_TIME:データベースシステムアクセスのリトライ間隔指定

OTSシステムでのデータベースシステムへのアクセス時に、メモリ資源不足などのリトライ可能なエラーが発生した場合のリトライ間隔(秒単位)を指定します。
指定可能な範囲は、1～600です。省略値は、5です。

DATABASE_RETRY_MAX:データベースシステムアクセスのリトライ上限回数指定

OTSシステムでのデータベースシステムへのアクセス時に、メモリ資源不足などのリトライ可能なエラーが発生した場合にリトライする上限回数を指定します。
指定可能な範囲は、1～2147483647です。省略値は、5です。

MEM_RETRY_TIME:OTSシステム処理中のエラーでのリトライ間隔指定

OTSシステムの処理中に、メモリ資源不足などのリトライ可能なエラーが発生した場合のリトライ間隔(秒単位)を指定します。
指定可能な範囲は、1～600です。省略値は、5です。

MEM_RETRY_MAX:OTSシステム処理中のエラーでのリトライ上限回数指定

OTSシステムの処理中に、メモリ資源不足などのリトライ可能なエラーが発生した場合にリトライする上限回数を指定します。
指定可能な範囲は、1～2147483647です。省略値は、5です。

RSCSTOP_CHECK_COUNT:通常停止からのトランザクション待合せ回数指定

トランザクション処理中にリソース管理プログラムを通常停止し、トランザクション完了をOBSERVE_CYCLE_TIMEの監視同期に合わせた待合せ回数を指定します。
OBSERVE_CYCLE_TIME × RSCSTOP_CHECK_COUNT秒間、トランザクションの完了を待ち合わせ、時間内に完了しない場合は、リソース管理プログラムの停止を通常停止から強制停止に切り替えます。
指定可能な範囲は、1～2147483647です。省略値は、100です。

OTS_VERSION:OTSのバージョン

OTSのバージョンを指定します。通常、変更しないでください。
省略値は、5です。

JTS_VERSION:JTSのバージョン Windows32/64 Linux32/64

JTSのバージョンです。通常、変更しないでください。
省略値は、5です。

JAVA_VERSION:JDK/JREのバージョン Windows32/64 Linux32/64

JTS用リソース管理プログラムが利用するJavaのバージョンです。14を指定します。
省略値は、14です。

PATH:JDK/JREのパス Windows32/64 Linux32/64

JTS用リソース管理プログラムが利用するjavaコマンドへのパスをフルパスで指定します。java実行体を含めるパスを指定してください。
初期値は、JDK/JRE 8のバージョンに対応したパスです。通常、変更しないでください。

注意

- JTS用のリソース管理プログラムを利用する場合は、必ず指定してください。
- Interstage Application Serverに同梱されているJDK/JREのパスを指定してください。

TRACE_MODE:トレースの出力形式 Windows32/64 Linux32/64

JTSを利用した環境で出力されるトレースの出力形式を、以下から選択して指定します。

- 異常発生時に、OTSのインストールパス/var配下にトレースファイルを出力する場合：“1”
  注）通常、“1”を選択してください。
- システムの状態に関係なく、常時、OTSのインストールパス/var配下にトレースファイルを出力する場合：“2”
  注）常時出力されるため、ファイルサイズに注意してください。
- トレースファイルを出力しない場合：“3”

TRACE_LEVEL:トレースの出力レベル Windows32/64 Linux32/64

JTSを利用した環境で出力されるトレースのモードを、“1”から“5”までの数値で指定します。数値が大きいほど、詳細なトレース情報を出力します。
通常運用時は、“1”が指定されます。パフォーマンスに影響があるため、通常、変更しないでください。

C.2 セットアップ情報ファイル

■概要

セットアップ情報ファイルは、otssetupコマンドによるOTSシステム動作環境の設定時に指定するファイルです。isinitコマンドでInterstageの初期化を行う場合は、“Interstage動作環境定義”をカスタマイズする必要があります。Interstage動作環境定義の詳細については、“運用ガイド(基本編)”を参照してください。
セットアップ情報ファイルは、otssetupコマンド実行時に作成します。1度作成したセットアップ情報ファイルは、次回のセットアップ時にも使用可能であるため、保管しておいてください。

■ファイル内情報

◆形式:

```
パラメタ名=設定値
```

◆設定例

```
MODE=SYS
LOGFILE=c:¥ots¥logfile........(Windows(R)の場合)
LOGFILE=/dev/rdsk/c0t0d0s0....(Solarisの場合)
LOGFILE=/dev/raw/raw1.........(Linuxの場合)
TRANMAX=10
PARTICIPATE=4
OTS_FACT_THR_CONC=5
OTS_RECV_THR_CONC=2
JTS_RMP_PROC_CONC=5
JTS_RMP_THR_CONC=16
HOST=otshost
PORT=8002
LOCALE=EUC
```

ポイント

太字以外の項目は、省略可能です。省略時は、省略値が有効となります。

C.2.1 MODE: セットアップ種別

OTSシステムが動作するホストか、リソース管理プログラムだけが動作するホストかを、以下から選択して指定します。大文字で指定してください。

- OTSシステムおよびリソース管理プログラムが動作するホストの場合:“SYS”
  OTSシステム動作環境のセットアップ、リソース管理プログラムの動作環境のセットアップ、およびシステムログファイルの作成を行います。
- リソース管理プログラムだけが動作するホストの場合:“RMP”
  リソース管理プログラムの動作環境のセットアップだけを行います。
  注)“RMP”を設定してセットアップした環境では、OTSシステムを起動できません。

Interstage動作環境定義ファイルの“OTS Setup mode”に相当します。

“RMP”を設定する場合、リソース管理プログラムを正しく動作させるために、OTSシステムが動作するホストのネーミングサービスを参照する必要があります。以下のどちらかの方法でセットアップを行ってください。

- “RMP”を設定すると同時に、OTSシステムが動作しているホストの“HOST”／“PORT”／“LOCALE”を指定してセットアップを行います。この場合、ネーミングサービスはOTSシステムではなく、“RMP”を設定したホストのネーミングサービスが利用されます。“RMP”を設定したホストとOTSシステムが動作するホストのネーミングサービスをそれぞれ独立させて運用させることが可能となります。

- isinitコマンドに“type3”を指定してInterstageの初期化(NS Use／NS Host／NS Port Number／NSに、OTSシステムが動作するホストのネーミングサービスを設定)を行った後、“RMP”を設定してotssetupコマンドでセットアップを行ってください。これにより、OTSシステムが動作するホストと“RMP”を設定したホストのネーミングサービスは、共有されます。

注意

- “SYS”を設定したホスト同士のネーミングサービスを共有することはできません。必ず1つのネーミングサービスには1つの“SYS”を設定したホストになるようにしてください。“RMP”を設定する場合は、複数のホストでネーミングサービスを共有できます。
- “RMP”を設定した場合、otsstartコマンドではOTSシステムを起動できません。otsstartrscコマンドでリソース管理プログラムの起動だけを行うことができます。

C.2.2 LOGFILE: システムログファイルのパス

OTSシステムのシステムログファイルを指定します。

Windows32/64

OTSシステムのシステムログファイルへのパス(ドライブ名を含む絶対パス)を、制御文字(ShiftJISコードの0x00～0x1F, 0x7F)を除く文字列で指定します。半角英文字の大文字と小文字、全角英文字の大文字と小文字は区別されません。

Solaris64 Linux32/64

OTSシステムのシステムログファイルと使用するローデバイスまたはファイル名を、スラッシュ(/)で始まる空白文字と半角カナを除く文字列で指定します。

“MODE”に“SYS”を設定した場合に有効となります。
最大長は、255文字です。
Interstage動作環境定義ファイルの“OTS path for system log”に相当します。

ポイント

Linux32/64
ローデバイスの作成手順を以下に示します。

1. オペレーティングシステムのpartedコマンド／fdiskコマンドで、ローデバイスのパーティションを作成します。
2. ディスクのパーティションに対応するudevのブロックデバイス名を特定します。
   partedコマンドを使用した場合の例を以下に示します(#:プロンプト)。

```
# parted /dev/sda
(parted) p
      :

番号  開始    終了    サイズ  タイプ     ファイルシステム  フラグ
1     1049kB  211MB   210MB   primary    ext4              boot
2     211MB   32.4GB  32.2GB  primary    ext4
      :
8     77.5GB  78.5GB  974MB   logical

(parted) q
# udevadm info --query=path --name=/dev/sda8
/devices/pci0000:00/0000:00:1f.2/host0/target0:0:0/0:0:0:0/block/sda/sda8
# udevadm info --query=property --path=/devices/pci0000:00/0000:00:1f.2/host0/target0:0:0/0:0:0:0/
block/sda/sda8 | grep ID_PATH
ID_PATH=pci-0000:00:1f.2-scsi-0:0:0:0
```

3. udevの設定ファイル(/etc/udev/rules.d/60-raw.rules)を編集し、作成したパーティションをバインドします。

```
ACTION=="add", KERNEL=="sda8", ENV{ID_PATH}=="pci-0000:00:1f.2-scsi-0:0:0:0", RUN+="/bin/raw /dev/raw/raw1 %N"
```

4. udevによりローデバイスのアクセス権限が正しく設定されるように、/etc/udev/rules.d/配下の追加パーミッションルールファイルを必要に応じて編集します。

注意

- ローデバイスをバインドするブロックデバイスは、パーティションを指定してください。パーティション番号のないハードディスクデバイス(/dev/sdgなど)は、ディスクラベル(パーティションテーブル)を含んでいるため、ローデバイスとして使用しないでください。
- セットアップ情報ファイルのログファイルの指定には、必ずキャラクタデバイスにバインドしたデバイス名を指定してください。

C.2.3 TRANMAX: 最大トランザクション多重度

トランザクションの最大数を指定します。必ず指定する必要があります。
また、MODEに“RMP”を設定した場合は、連携するOTSシステム(MODEに“SYS”を設定しているシステム)と同じ値を設定してください。

Windows32/64

設定可能な範囲は、1～256です。

Solaris64 Linux32/64

設定可能な範囲は、1～1024です。

Interstage動作環境定義ファイルの“OTS maximum Transaction”に相当します。

C.2.4 PARTICIPATE: 1トランザクションに参加するリソース数

1トランザクションに参加するリソースの最大数を指定します。
MODEに“SYS”を設定した場合に有効となります。
設定可能な範囲は、2～32です。省略値は、4です。
Interstage動作環境定義ファイルの“OTS Participate”に相当します。

C.2.5 OTS_FACT_THR_CONC: OTSシステムのスレッド多重度

OTSシステムのスレッド多重数を指定します。指定した数だけ、begin/commit/rollbackなどのCurrentインタフェース、およびUserTransactionインタフェースを同時に動作させることが可能となります。
MODEに“SYS”を設定した場合に有効となります。
設定可能な範囲は、1～31です。省略値は、5です。
Interstage動作環境定義ファイルの“OTS Multiple degree”に相当します。
最大値を超えた場合は、警告メッセージ“ots9013”が出力され、自動的に31が設定されます。

ポイント

OTSシステムのスレッド多重度は、トランザクション処理性能を最大限に引き出すようにチューニングされているため、省略値から変更する必要はありません。
変更する場合は、以下の関係を保つように設定してください。

```
OTSシステムのスレッド多重度 ＝< リソース管理プログラムの多重度(注)
OTSシステムのスレッド多重度 ＝< 1リソース管理プログラムのトランザクションの最大多重度
```

注) Windows32/64 Linux32/64
JTS用リソース管理プログラムにおける多重度は、以下の算出式で見積もってください。

```
JTS用のリソース管理プログラムのプロセス多重度（JTS_RMP_PROC_CONC） ×
JTS用のリソース管理プログラムのスレッド多重度（JTS_RMP_THR_CONC）
```

C.2.6 OTS_RECV_THR_CONC: リカバリプロセスのスレッド多重度

リカバリプログラムのスレッド多重数を指定します。指定した数だけ、リカバリ処理を同時に動作させることが可能となります。
MODEに“SYS”を設定した場合に有効となります。
設定可能な範囲は、1～2147483647です。省略値は、2です。
Interstage動作環境定義ファイルの“OTS Recovery”に相当します。

C.2.7 JTS_RMP_PROC_CONC: JTS用のリソース管理プログラムのプロセス多重度 Windows32/64 Linux32/64

JTS用のリソース管理プログラムのプロセス多重度を指定します。使用するリソース(データベース、リソースアダプタなど)の数だけ指定することを推奨します。
設定可能な範囲は、1～32です。省略値は、2です。5以下の場合、変更する必要はありません。
Interstage動作環境定義ファイルの“OTS JTS's RMP Multiple degree of Process”に相当します。
最大値を超えた場合は、警告メッセージ“ots9017”が出力され、自動的に32が設定されます。

ポイント

リソース管理プログラムの多重度は、トランザクション処理性能を最大限に引き出すようにチューニングされているため、省略値から変更する必要はありません。
変更する場合は、OTSシステムのスレッド多重度とリソース管理プログラムの多重度の関係を以下のように設定してください。

```
OTSシステムのスレッド多重度 ＝< リソース管理プログラムの多重度(注)
```

注) JTS用リソース管理プログラムにおける多重度は、以下の算出式で見積もってください。

```
JTS用のリソース管理プログラムのプロセス多重度（JTS_RMP_PROC_CONC） ×
JTS用のリソース管理プログラムのスレッド多重度（JTS_RMP_THR_CONC）
```

C.2.8 JTS_RMP_THR_CONC: JTS用のリソース管理プログラムのスレッド多重度 Windows32/64 Linux32/64

JTS用のリソース管理プログラムのスレッド多重度を指定します。
設定可能な値は、1～2147483647です。省略値は、16です。
通常、変更する必要はありません。
Interstage動作環境定義ファイルの“OTS JTS's RMP Multiple degree of Thread”に相当します。

ポイント

リソース管理プログラムの多重度は、トランザクション処理性能を最大限に引き出すようにチューニングされているため、省略値から変更する必要はありません。
変更する場合は、OTSシステムのスレッド多重度とリソース管理プログラムの多重度の関係を以下のように設定してください。

```
OTSシステムのスレッド多重度 ＝< リソース管理プログラムの多重度(注)
```

注) JTS用リソース管理プログラムにおける多重度は、以下の算出式で見積もってください。

```
JTS用のリソース管理プログラムのプロセス多重度:JTS_RMP_PROC_CONC ×
JTS用のリソース管理プログラムのスレッド多重度:JTS_RMP_THR_CONC
```

C.2.9 HOST: OTSシステムが動作するホスト名

OTSシステムが利用するネーミングサービスのホスト名を、英数字、マイナス記号(-)、またはピリオド(.)から構成される64文字以内の英字から始まる文字列で指定します。最後の文字には、マイナス記号(-)、またはピリオド(.)を記述できません。
MODEに“RMP”を設定した場合に有効となります。
本ステートメントは、省略可能です。設定する場合は、PORTを同時に設定する必要があります。
本ステートメントの利用方法については、MODEを参照してください。
Interstage動作環境定義ファイルの“OTS Host”に相当します。

注意

“isinit”コマンドで“type3”を選択している場合は、使用しないでください。

C.2.10 PORT: OTSシステムが動作するホストのCORBAサービスのポート番号

OTSシステムが利用するネーミングサービスのポート番号を指定します。
MODEに“RMP”を設定した場合に有効となります。
設定可能な範囲は、1024～65535です。
本ステートメントは、省略可能です。設定する場合は、HOSTを同時に設定する必要があります。
本ステートメントの利用方法については、MODEを参照してください。
Interstage動作環境定義ファイルの“OTS Port”に相当します。

注意

isinitコマンドに“type3”を指定してInterstageを初期化した場合は、使用しないでください。

C.3 RMPプロパティ Windows32/64 Linux32/64

■概要

JTS用リソース管理プログラムに対するプロパティファイルです。
RMPプロパティは、Javaのプロパティリストの形式で、その他のキーを設定することもできます。設定されたキーと値は、JTS用リソース管理プログラムが動作するJavaVMのシステムプロパティに反映されます。

■ファイル名

RMPプロパティファイルは、インストール時に、以下に格納されます。

Windows32/64

```
C:¥Interstage¥ots¥etc¥RMP.properties
```

Linux32/64

```
/opt/FJSVots/etc/RMP.properties
```

注意

本製品のインストールパスがデフォルトの場合のパスです。

■ファイル内情報

◆形式:

```
パラメタ名=設定値
```

◆パラメタ

RecoveryTarget: リカバリ対象

JTS用のリソース管理プログラムの起動時にリカバリを行う対象をリソース定義名で指定します。リカバリ対象が設定されていない場合、リカバリ対象が複数の場合は、空白で区切って指定してください。

ポイント

RecoveryTargetを省略した場合でも、登録済みリソースに対してリカバリ処理を行います。

例

リカバリ対象が3つの場合

```
RecoveryTarget=Oracle_resource1 Oracle_resource2 Oracle_resource3
```

JavaPath: Javaコマンドへのパス

必要な場合に、記述を追加します。通常、指定しないでください。指定した場合、動作の保証はできません。

JavaCommandOption: Javaコマンドに受け渡すオプション

必要な場合に、記述を追加します。通常、指定しないでください。指定した場合、動作の保証はできません。

Classpath: 追加するクラスパス

Windows32/64

必要な場合に、記述を追加します。通常、指定しないでください。指定した場合、動作の保証はできません。

Linux32/64

クラスタサービス機能利用時に、リソースと連携するために必要となるクラスライブラリへのパスを設定します。クラスライブラリの詳細については、“アプリケーション作成ガイド(データベース連携サービス編)”の“リソース管理プログラムの作成から起動まで”を参照してください。

Librarypath:追加するライブラリパス

リソースと連携するために必要となるライブラリへのパスを指定します。

Windows32/64

JTS用のリソース管理プログラムの環境変数PATHに追加されます。

Linux32/64

JTS用のリソース管理プログラムの環境変数LD_LIBRARY_PATHに追加されます。

Environ:追加する環境変数

リソースと連携するために必要となる環境変数を指定します。

例

環境変数ORACLE_HOMEを指定する場合

Windows32/64

```
Environ ORACLE_HOME=C:¥app¥user¥product¥11.2.0¥db
```

Linux32/64

```
Environ ORACLE_HOME=/u01/app/oracle/product/11.2.0/dbhome
```

C.4 リソース定義ファイル

■概要

OTS／JTSが連携するリソース(データベース、リソースアダプタなど)に接続するための情報を定義するファイルです。
otssetrscコマンドを利用して、リソース単位に登録します。

■ファイル内情報

◆形式:

```
キー名=設定値
```

◆設定例

OTS用リソース定義ファイル

```
# 環境変数
ENVIRON ORACLE_SID=orac
ENVIRON ORACLE_HOME=/opt/oracle.............(Solaris/Linuxの場合)
ENVIRON LD_LIBRARY_PATH=/opt/oracle/lib.....(Solaris/Linuxの場合)

# 使用するデータベースシステム名とOPENINFO文字列、CLOSEINFO文字列
NAME=oracle_rmp_thread
RMNAME=Oracle_XA
OPENINFO=Oracle_XA+Acc=P/system/manager+SesTm=0+Threads=true
CLOSEINFO=
THREADS=TRUE................................(Solaris/Linuxの場合)
```

JTS用リソース定義ファイル Windows32/64 Linux32/64

```
# database1
name=xads1
rscType=JTS
type=JDBC
lookUpName=jdbc/XADataSource
initialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory
```

```
providerURL=file:/tmp/JNDI
user=dbuser
password=dbpass
logfileDir=c:¥interstage¥ots¥var.....(Windows(R)の場合)
logfileDir=/opt/FJSVots/var.....(Linuxの場合)
```

注意

キーの名前は、OTSおよびJTSで大文字／小文字が異なりますが、同じ意味を持ちます。

◆キー一覧

キー(OTS)	キー(JTS)	意味
ENVIRON	－	環境変数の設定
NAME	name	リソース定義名
RMNAME	－	リソースマネージャ名
OPENINFO	－	オープン文字列
CLOSEINFO	－	クローズ文字列
THREADS	－	スレッドモード
OTS_RMP_PROC_CONC	－	OTS用のリソース管理プログラムの多重度
RSCTYPE	rscType	リソース定義ファイルの種類
－	type	リソースの種類
－	lookupName	リソースの検索名
－	initialContextFactory	initialContextFactory名
－	providerURL	プロバイダURL
USER	－	ユーザ名
－	user	ユーザ名
－	password	パスワード
GROUP	－	グループ名
－	logfileDir	リソースのログファイル格納ディレクトリ

◆キー詳細

ENVIRON: 環境変数の設定(OTS)

値*data*に、リソース管理プログラム、またはリソース管理プログラムと同じプロセス内で動作するデータベースライブラリに渡す環境変数envを指定します。省略可。

Solaris64 Linux32/64

リソース管理プログラムを使用するサーバアプリケーションの起動時に指定するデータベースへの環境変数と同一の環境変数を指定してください。
また、リソース定義ファイルには、以下のような$指定できません。

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/oracle/lib
```

使用するデータベースがSymfoware/RDBの場合は、環境変数LD_LIBRARY_PATHにSymfoware/RDBの必須製品であるライブラリのパス名を指定してください。

NAME、name: リソース定義名(OTS、JTS)

otssetrscコマンドでの登録時に、本パラメタに指定したリソース定義名で登録します。1度登録されたリソース定義ファイルは、すべてリソース定義名で扱うことができます。リソース定義名は、32文字以内で指定します。省略不可。
“JTSRMP”は予約語であるため、リソース定義名として使用できません(一部またはすべてを小文字にしても使用できません)。

Windows32/64 Linux32/64
JTS用リソース定義ファイルでは、isj2eeadminコマンドで登録するJ2EEリソース定義の接続対象となるリソースの“定義名”を指定してください。詳細については、“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

RMNAME: リソースマネージャ名(OTS)

*system_name*に、データベースのシステム名を以下から選択して指定します。

- Oracleの場合:“Oracle_XA”
- Symfoware/RDBの場合:“RDBII”
- Windows32/64
  MQDの場合:“XA_MQD”

OPENINFO: オープン文字列(OTS)

*open_data*に、データベースのベンダが公開する、データベースをオープンする場合に必要なopen文字列を、256文字以内の文字列で指定します。
指定する内容については、各データベースのマニュアルを参照してください。

注意

- OPENINFOに指定するユーザ名は、各データベースに対するアクセス権限がないと、リソース管理プログラムの起動に失敗します。必要な権限については、各データベースのマニュアルを参照してください。
- Solaris64 Linux32/64
  プロセスモード/スレッドモードのタイプが、リソース管理プログラム作成時と動作時(リソース定義ファイル内のスレッド指定)とで異なる場合、リソース管理プログラムの起動が誤動作する可能性があります。必ずタイプをあわせて運用してください。

CLOSEINFO: クローズ文字列(OTS)

*close_data*に、データベースのベンダが公開する、データベースをクローズする場合に必要なclose文字列を、256文字以内の文字列で指定します。
指定する内容については、各データベースのマニュアルを参照してください。

THREADS: スレッドモード(OTS) Solaris64 Linux32/64

リソース管理プログラムのモードを以下から選択して指定します。

- プロセスモードの場合:“FALSE”(省略時)
- スレッドモードの場合:“TRUE”

OTS_RMP_PROC_CONC: OTS用のリソース管理プログラムの多重度(OTS)

OTS用のリソース管理プログラムの多重数を指定します。通常、変更する必要はありません。
指定可能な範囲は、1～31です。省略値は、5です。
最大値を超えた場合は、警告メッセージ“ots9017”を出力し、31を自動的に設定します。

注意

リソース管理プログラムの多重度は、トランザクション処理性能を最大限に引き出すようにチューニングされているため、省略値から変更する必要はありません。
変更する場合は、OTSシステムのスレッド多重度とリソース管理プログラムの多重度の関係を以下のように設定してください。

```
OTSシステムのスレッド多重度 ＝< リソース管理プログラムの多重度
```

RSCTYPE、rscType: リソース定義ファイルの種類(OTS、JTS)

リソース定義ファイルの種別を以下から選択して指定します。

- OTSを利用する場合:“OTS”(省略時)
- JTSを利用する場合:“JTS” Windows32/64 Linux32/64
  注)JTSを利用する場合は、必ず“JTS”を指定してください。

type: リソースの種類(JTS) Windows32/64 Linux32/64

リソースの種類を以下から選択して指定します。省略不可。

- JDBCを利用してデータベースと接続する場合:“JDBC”／“DBMS”(旧バージョンでの指定方法)
- J2EE Connector Architectureを利用してリソースアダプタと接続する場合は、“JCA”

lookupName: リソースの検索名(JTS) Windows32/64 Linux32/64

JDBCを利用してデータベースと接続する場合は、データベースが提供するデータソースをバインドした名前を指定します。
isj2eeadminコマンドで登録するJ2EEリソース定義で設定したデータソース名と同じ値を指定してください。
J2EE Connector Architectureを利用してリソースアダプタと接続する場合は、リソースアダプタの配備時に設定した“リソース名”を指定してください。

initialContextFactory: initialContextFactory名(JTS) Windows32/64 Linux32/64

バインドされたデータソース参照時に使用するinitialContextFactoruy名を指定します。isj2eeadminコマンドで登録するJ2EEリソース定義で設定したクラス名と同じ値を指定してください。JDBCを利用してデータベースと接続する場合は、必ず指定してください。

providerURL: プロバイダURL(JTS) Windows32/64 Linux32/64

バインドされたデータソース参照時に使用するprovider URLを指定します。isj2eeadminコマンドで登録するJ2EEリソース定義で設定したクラス名と同じ値を指定してください。

USER: ユーザ名(OTS) Solaris64 Linux32/64

リソース管理プログラムの実行ユーザを指定します。otssetrscコマンド実行時に、-uオプションを指定した場合は、オプションに指定されたユーザ名が有効となります。
“GROUP”と同時に指定する必要があります。
指定したユーザは、“GROUP”に指定するグループに所属している必要があります。
強化セキュリティモードの場合は、強化セキュリティモード設定時に指定したグループに所属している必要があります。

user: ユーザ名(JTS) Windows32/64 Linux32/64

リソースとの接続時にユーザ名が必要な場合に指定します。isj2eeadminコマンドで登録するJ2EEリソース定義によって設定したユーザ名を指定してください。

password: パスワード(JTS) Windows32/64 Linux32/64

リソースとの接続時にパスワードが必要な場合に指定します。isj2eeadminコマンドで登録するJ2EEリソース定義によって設定したユーザのパスワードを指定してください。

GROUP: グループ名(OTS) Solaris64 Linux32/64

リソース管理プログラムの実行ユーザを指定します。otssetrscコマンド実行時に、-gオプションを指定した場合は、オプションに指定されたグループ名が有効となります。
“USER”と同時に指定する必要があります。
強化セキュリティモードの場合、本設定は無効となり、強化セキュリティモード設定時に指定したグループ名が有効となります。

logfileDir: リソースのログファイル格納ディレクトリ(JTS) Windows32/64 Linux32/64

接続したリソースのトラブルを調査する場合は、トレースログを採取するディレクトリを指定してください。ディレクトリ名の最後に、セパレータを付加しないでください。
通常、指定しません。

EE

付録D　イベントサービスの環境定義

イベントサービスの動作環境ファイル、およびイベントチャネル・サプライヤ・コンシューマ総数の見積もり方法について説明します。
イベントサービスの動作環境ファイルは、以下に格納されます。

格納パス

Windows32/64 (インストールパスはデフォルト)

```
C:¥Interstage¥eswin¥etc
```

Solaris64 (インストールパスはデフォルト)

```
/etc/opt/FJSVes
```

Linux32/64

```
/etc/opt/FJSVes
```

ファイル

traceconfig

注意

上記以外のファイルは、イベントサービスの動作環境としてカスタマイズできません。エディタなどで編集しないでください。

D.1　traceconfig

■概要

traceconfigファイルは、イベントサービスのトレース動作環境に関する定義が格納されたファイルです。

■ファイル名

Windows32/64 (インストールパスはデフォルト)

```
C:¥Interstage¥eswin¥etc¥traceconfig
```

Solaris64 (インストールパスはデフォルト)

```
/etc/opt/FJSVes/traceconfig
```

Linux32/64

```
/etc/opt/FJSVes/traceconfig
```

■ファイル内情報

traceconfigファイルは、以下の形式で値を設定します。

◆形式:

パラメタ名 = 設定値

◆パラメタ:

以下の動作環境について、パラメタ設定値を変更することができます。

注意

パラメタ値を変更した場合、次回のイベントサービス起動時より有効となります。

パラメタ名	初期値 省略値 指定範囲	意味
trace_size	1024 512 512～102400	トレース情報の採取に使用するバッファのサイズをキロバイト単位で指定します。(注1)
trace_file_number	Windows32/64 Solaris64 50 Linux32/64 10	採取するトレース情報ファイルの最大数を指定します。トレース情報ファイルの数が指定値を超えた場合は、古いトレース情報ファイルに上書きします。
	Windows32/64 Solaris64 50 Linux32/64 10	
	Windows32/64 Solaris64 50～1000 Linux32/64 10～1000	
trace_auto	yes yes yes, no	トレース情報の自動採取を有効にするかを指定します。 ・ yes: トレース情報の自動採取を有効とする。(注2) ・ no : トレース情報の自動採取を無効とする。
Linux32/64 trace_buffer	process process process, system	内部トレースを採取する単位を指定します。 ・ process: プロセス単位で内部トレースを採取する。 ・ system: イベントサービス単位で内部トレースを採取する。

注1)
トレース情報の出力サイズはチャネル数、コンシューマ数、サプライヤ数、および通信頻度によって異なります。起動処理系、通信処理系、および停止処理系で使用するトレース情報のバッファサイズを以下に記載します。

- 起動処理系
  イベントチャネル起動: 3.2KB

サプライヤ起動処理(pushメソッドを出すまで): 1.0KB
コンシューマ起動処理(pullメソッドを出すまで): 1.0KB

- 通信処理系
  pushメソッド: 0.8KB
  pullメソッド(受信成功): 1.2KB
  pullメソッド(COMM_FAILURE[minor=0x464a09c1]): 1.0KB

- 停止処理系
  イベントチャネル停止: 3.4KB
  サプライヤdisconnect処理: 0.5KB
  コンシューマdisconnect処理: 0.8KB

トレース情報バッファサイズを初期設定で運用した場合の計算例を以下に示します。

例

1チャネルで、コンシューマ数:サプライヤ数が1:1の場合

1回の通信(push/pull)で2.0KB(0.8KB＋1.2KB)のバッファサイズが必要となります。
トレース情報バッファは、バッファを半分ずつサイクリックに使用するため、トレース情報バッファ(サイズ:1024KB)に格納できる通信のトレース情報数は256回となります。
(トレース情報バッファサイズ ÷ 2) ÷ 1回の通信に必要なバッファサイズ ＝
(1024KB ÷ 2) ÷ 2.0KB ＝ 256回

40秒に1回の通信を行うと仮定した場合、約2.8時間の通信をロギングできることになります。
256 × 40秒 ＝ 10240秒 ＝ 約2.8時間
上記の例では、トレース情報を自動採取する事象が発生するまでの約2.8時間分のトレース情報を採取することができます。

トレース情報バッファサイズは、少なくとも5分以上のトレース情報が採取可能なサイズを指定してください。
トレース情報バッファサイズを初期値から変更した場合、その増分だけ共用メモリ使用量が増加します(キロバイト単位)。

注2)
トレース情報の自動採取を有効とする(trace_auto = yes)場合、トレースファイルは以下のファイル名で出力されます。(XXX: 001～1000の通番)

Windows32/64

```
C:¥Interstage¥eswin¥var¥ESLOGXXX
```

Solaris64

```
/var/opt/FJSVes/ESLOGXXX
```

Linux32/64

プロセス単位で内部トレースを採取する(trace_buffer = process)場合

- イベントサービスのデーモンプロセスのログ情報

```
/var/opt/FJSVes/ESLOGDUMPDAEMONXXX
```

- イベントファクトリプロセスのログ情報

```
/var/opt/FJSVes/ESLOGDUMPFACTORYXXX
```

- 静的イベントチャネルプロセスのログ情報

```
/var/opt/FJSVes/ESLOGDUMPグループ名XXX
```

・ 動的イベントチャネルプロセスのログ情報

```
/var/opt/FJSVes/ESLOGDUMPインプリメンテーション名XXX
```

イベントサービス単位で内部トレースを採取する(trace_buffer = system)場合

```
/var/opt/FJSVes/ESLOGXXX
```

D.2 サプライヤ・コンシューマ総数の見積もり方法

不揮発チャネル運用時、同じユニットを使用するイベントチャネルに接続するサプライヤ・コンシューマの総数は、以下の見積もり式を参考にして見積もってください。

```
イベントチャネルの最大接続数の合計値（注） ＋ 10 ＜ ユニット定義ファイルのtranmax値
```

注）

同じユニットを使用するすべてのイベントチャネルについて、イベントチャネルごとに以下の値を求めて、その合計値を算出してください。

Windows32/64

```
イベントチャネルの最大接続数（esmkchnlコマンドの-mオプションの指定値(省略値：16)） × 2
```

Solaris64 Linux32/64

```
イベントチャネルの最大接続数（esmkchnlコマンドの-mオプションの指定値(省略値：16)） ＋ 16
```

付録E　Interstage HTTP Serverの環境定義

Interstage HTTP Serverの運用状態をチューニングするには、Interstage管理コンソールを使用して設定する方法と、Interstage HTTP Serverの環境定義ファイル(httpd.conf)を使用して設定する方法があります。

Interstage管理コンソールを使用して設定する場合、以下の手順で環境設定を行います。

1. Interstage管理コンソールにログインします。
2. [システム]＞[サービス]＞[Webサーバ]＞[Webサーバ名]＞[環境設定]タブ＞[Webサーバ:環境設定](詳細設定[表示])画面を使用して環境設定を行います。

参照

- Interstage管理コンソールの起動については、「運用ガイド(基本編)」の「Interstage管理コンソールによるInterstage運用」を参照してください。
- Interstage管理コンソールの定義詳細については、Interstage管理コンソールのヘルプを参照してください。

ここでは、環境定義ファイル(httpd.conf)の定義方法について説明します。

ポイント

Interstage HTTP Serverの環境定義ファイルは、以下に格納されています。

Windows32/64 (インストールパスはデフォルト)

```
C:¥Interstage¥F3FMihs¥servers¥(Webサーバ名)¥conf¥httpd.conf
```

Solaris64 (インストールパスはデフォルト) Linux32/64

```
/var/opt/FJSVihs/servers/(Webサーバ名)/conf/httpd.conf
```

E.1　タイムアウト時間

■タイムアウト時間の設定

タイムアウト時間を設定する場合、環境定義ファイル(httpd.conf)の以下のディレクティブを編集します。

Timeout クライアント送受信タイムアウト時間(秒)

クライアントとの間でデータパケットを送受信するときに待機する最長の時間(秒)を指定します。待機時間には、0から65535までを指定できます。

Interstage HTTP Serverは、指定された時間に達してもパケットを受信できない場合、TCPコネクションが切断されます。接続しているネットワークのトラフィックが増大し、TCPコネクションの接続が頻繁に中断される場合は、待機時間を増やすことにより中断回数を減少させることができます。

クライアント送受信タイムアウト時間の初期値は「600」、省略値は「300」です。

ポイント

クライアントからTCPコネクションが接続されたあとにリクエストが届かない場合は、指定した時間(秒)に達すると、TCPコネクションが切断されます。

SSLHandshakeTimeout SSLコネクション確立時の送受信タイムアウト時間(秒)

SSLコネクションの確立処理でクライアントからのデータパケットを送受信するときに待機する最長の時間(秒)を設定します。待機時間には、0から65535までを指定できます(0を指定した場合、無制限)。

Interstage HTTP Serverは、指定された時間に達してもパケットを受信できない場合は、TCPコネクションが切断されます。通常、SSLコネクションの確立処理にかかる時間をチューニングしたい場合に設定します。

SSLコネクション確立時の送受信タイムアウト時間の初期値は、ありません。省略値は、Timeoutディレクティブの設定値です。

■HTTP Keep-Alive機能の設定

HTTP Keep-Alive機能を設定する場合、環境定義ファイル(httpd.conf)の以下のディレクティブを編集します。

KeepAlive On|Off

Interstage HTTP Serverでは、クライアント(Webブラウザ)との間で持続的な接続を維持できます。

「Off」を指定した場合は、1つの要求が完了するたびに接続を閉じて、次の要求に対して新しく接続しますが、「On」を指定することにより複数の要求を同じ接続で繰り返し使えるため、クライアントのレスポンスが向上します。

初期値・省略値は、「On」です。

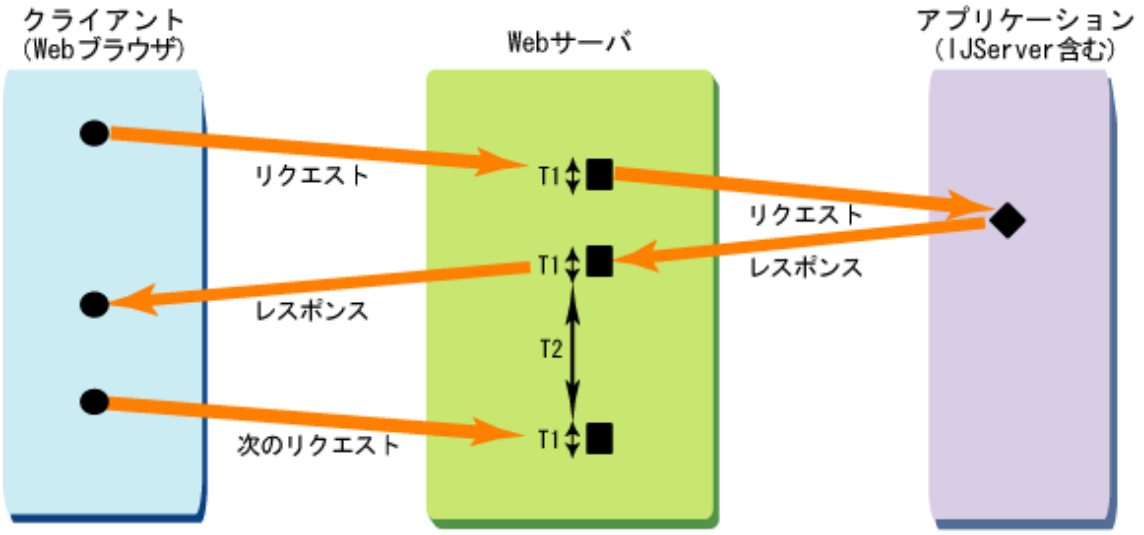
KeepAliveTimeout 次のリクエストまでのタイムアウト時間(秒)

クライアントの1つのリクエストが完了してから、コネクションを閉じずに次の新しいリクエストを待つ時間(秒)を指定します(「KeepAlive On」の場合のみ有効)。接続維持時間には、0から65535までを指定できます。この時間を経過しても次のリクエストがない場合は、接続を閉じます。

接続維持時間の初期値・省略値は、「15」です。

■タイムアウト時間の構成図

タイムアウト時間(TimeoutおよびKeepAliveTimeout)の構成図を以下に示します。



T1:Timeout(初期値 600秒)

クライアントとの間でデータパケットを送受信するときに待機する最長の時間(秒)。

注意

POSTまたはPUTリクエストにおいて、データを分割して複数回送受信される場合は、個々のデータの送受信時にかかる最長の時間となります。

T2:KeepAliveTimeout(初期値 15秒)

クライアントの1つのリクエストが完了してから、コネクションを閉じずに次の新しいリクエストを待つ時間(秒)。

E.2 クライアント同時接続数

クライアント同時接続数を設定する場合、環境定義ファイル(httpd.conf)の以下のディレクティブを編集します。

クライアント同時接続数の構成図については、「Interstage HTTP Server 運用ガイド」の「概要」－「Webサーバのプロセス構成(Windows(R))」／「Webサーバのプロセス構成(Solaris/Linux)」を参照してください。

Windows32/64

- ThreadsPerChild
- ThreadLimit
- ListenBacklog

Solaris64 Linux32/64

- MaxClients
- ServerLimit
- ListenBacklog

ThreadsPerChild クライアント同時接続数 Windows32/64

Interstage HTTP Serverが、クライアント(Webブラウザ)からの接続要求を同時に受け付けることができる最大数を設定します。最大数には、1からThreadLimitディレクティブの設定値までの数値を指定できます。

クライアント同時接続数の最大数の初期値は「50」、省略値は「64」です。

注意

本ディレクティブの設定値を大きくすることにより同時アクセス可能な数は多くなりますが、メモリ資源や一時ファイルなどの消費に伴いシステム全体の性能が劣化する可能性があります。

ポイント

本ディレクティブの設定値を超過するクライアントの接続要求があった場合は、以下のキューに保存されます。

本ディレクティブの設定値を超過したクライアント接続要求	キュー	キュー数
1つ目のクライアント接続要求	Webサーバ内の接続待ちキュー	1
2つ目以降のクライアント接続要求	オペレーティングシステム内の接続待ちキュー	ListenBacklogディレクティブの設定値

ThreadLimit 通信スレッドの上限数 Windows32/64

ThreadsPerChildディレクティブに設定するクライアント数の上限値を、1から15000までの数値で設定します。1より小さな値を指定した場合は、1で動作します。15000より大きな値を指定した場合は、15000で動作します。

本ディレクティブは、ThreadsPerChildディレクティブに1920よりも大きな値に設定する必要がある場合にだけ使用してください。

通信スレッドの上限数の初期値は、ありません。省略値は「1920」です。

ListenBacklog 接続待ちキューの最大数 Windows32/64

ThreadsPerChildディレクティブに設定したクライアント同時接続数よりも多くの接続要求があった場合に、オペレーティングシステム内にキューイングする数の最大値を設定します。接続待ちキューの最大数は、1から200までの数値で指定できます。

接続待ちキューの最大数の省略値は、「200」です。

注意

クライアントからの接続要求が以下の値を超過した場合は、この接続要求を受け付けず、ステータスコードも返却しません。

クライアント同時接続数(ThreadsPerChildディレクティブの設定値)＋接続待ちキューの最大数(本ディレクティブの設定値＋1)

MaxClients クライアント同時接続数 Solaris64 Linux32/64

Interstage HTTP Serverが、クライアント(Webブラウザ)からの接続要求を同時に受け付けることができる最大数を指定します。最大数には、1からServerLimitディレクティブの設定値までの数値を指定できます。1より小さな値を指定した場合は、1で動作します。ServerLimitディレクティブの設定値より大きな値を指定した場合は、ServerLimitディレクティブで指定した値で動作します。

クライアント同時接続数の初期値は「50」、省略値は「256」です。

注意

本ディレクティブの設定値を大きくすることにより同時アクセス可能な数は多くなりますが、メモリ資源や一時ファイルなどの消費に伴いシステム全体の性能が劣化する可能性があります。

ポイント

本ディレクティブの設定値を超過するクライアントの接続要求があった場合は、接続待ちキューに保存されます。接続待ちキュー数は、ListenBacklogディレクティブで設定します。

ServerLimit 通信プロセスの上限数 Solaris64 Linux32/64

MaxClientsディレクティブに設定するクライアントの同時接続数の上限値を、1から20000までの数値で設定します。1より小さな値を指定した場合は、1で動作します。20000より大きな値を指定した場合は、20000で動作します。

本ディレクティブは、MaxClientsディレクティブに256よりも大きな値に設定する必要がある場合にだけ使用してください。

通信プロセスの上限数の初期値は、ありません。省略値は「256」です。

ListenBacklog 接続待ちキューの最大数 Solaris64 Linux32/64

MaxClientsディレクティブで設定したクライアントの同時接続数よりも多くの接続要求があった場合に、オペレーティングシステム内にキューイングする数の最大値として、以下の条件に応じた値を設定します。接続待ちキュー数の最大数は、1から2147483647までの数値を指定できます。

接続待ちキューの最大数の省略値は、「511」です。

条件	接続待ちキューの最大数
本ディレクティブの設定値 ≦ 待機中TCPコネクションの最大値 (注)	本ディレクティブの設定値
本ディレクティブの設定値 ＞ 待機中TCPコネクションの最大値 (注)	待機中TCPコネクションの最大値 (注)

注)待機中TCPコネクションの最大値は、オペレーティングシステムに設定されています。以下のコマンドを実行して確認してください。待機中TCPコネクションの設定およびコマンドの詳細については、オペレーティングシステムのドキュメントを参照してください。

	待機中TCPコネクションの最大値	コマンド実行例
Solaris64	tcp_conn_req_max_q	/usr/sbin/ndd /dev/tcp tcp_conn_req_max_q
Linux32/64	/proc/sys/net/core/somaxconn	/sbin/sysctl -n net.core.somaxconn

注意

クライアントからの接続要求が以下の値を超過した場合は、この接続要求を受け付けず、ステータスコードも返却しません。

クライアント同時接続数(MaxClientsディレクティブの設定値)＋接続待ちキューの最大数(本ディレクティブの設定値＋α)

α:オペレーティングシステムの仕様に応じた値

付録F　Interstage HTTP Server 2.2の環境定義

Interstage HTTP Server 2.2の環境定義については、「Interstage HTTP Server 2.2運用ガイド」の「チューニング」－「環境定義」を参照してください。

付録G　Interstage シングル・サインオンの環境定義

Interstage シングル・サインオンを運用するための、環境定義のチューニングについて説明します。

G.1　1台のサーバにリポジトリサーバを構築する場合のチューニング

1台のサーバに、リポジトリサーバを構築する場合のチューニングについて説明します。

■Webサーバ(Interstage HTTP Server)のチューニング

リポジトリサーバのチューニングは、Webサーバ(Interstage HTTP Server)の環境定義を変更することにより行います。
詳細については、“付録E Interstage HTTP Serverの環境定義”を参照してください。

クライアントの同時接続数

以下のディレクティブに、想定する同時アクセス最大数以上を設定します。(初期値:50)

Windows32/64

ThreadsPerChild

Solaris64 Linux32/64

MaxClients

クライアント送受信タイムアウト時間

Timeoutにクライアントとの間でデータパケットを送受信するときに待機する最長の時間(秒)を指定します。(初期値:600)

◆チューニングの例

例

想定する同時アクセス最大数が256ユーザのシステムの場合

Windows32/64

Interstage HTTP Server
ThreadsPerChild＝256＋α(注1)
Timeout＝600(注2)

Solaris64 Linux32/64

Interstage HTTP Server
MaxClients＝256＋α(注1)
Timeout＝600(注2)

注1) システムを安定稼動させるため、αには10～100までの値を設定してください。
注2) 接続しているネットワークのトラフィックが増大し、接続が頻繁に中断される場合には、本時間を増やしてください。

■TCP/IPパラメタのチューニング Windows32/64

セション管理の運用を行う場合は、リポジトリサーバ(更新系)のマシンのTCP/IPパラメタのチューニングを行います。(注)

詳細については、“3.3 TCP/IPパラメタのチューニング”を参照してください。

注)リポジトリサーバ(更新系)への同時アクセス最大数が増大し、セション管理サーバとの通信に失敗し、以下のメッセージが出力される場合があります。この場合には、TCP/IPパラメタのチューニングを行ってください。

- sso00114
- sso00119

G.2 1台のサーバに認証サーバを構築する場合のチューニング

1台のサーバに、認証サーバを構築する場合のチューニングについて説明します。

■Webサーバ(Interstage HTTP Server)のチューニング

認証サーバのチューニングは、Webサーバ(Interstage HTTP Server)の環境定義を変更することにより行います。
詳細については、“付録E Interstage HTTP Serverの環境定義”を参照してください。

クライアントの同時接続数

以下のディレクティブに、想定する同時アクセス最大数以上を設定します。(初期値:50)

Windows32/64

ThreadsPerChild

Solaris64 Linux32/64

MaxClients

クライアント送受信タイムアウト時間

Timeoutにクライアントとの間でデータパケットを送受信するときに待機する最長の時間(秒)を指定します。(初期値:600)

◆チューニングの例

想定する同時アクセス最大数が256ユーザのシステムの場合

Windows32/64

Interstage HTTP Server
ThreadsPerChild＝256＋α(注1)
Timeout＝600(注2)

Solaris64 Linux32/64

Interstage HTTP Server
MaxClients＝256＋α(注1)
Timeout＝600(注2)

想定する同時アクセス最大数500ユーザを、5台の認証サーバにて負荷分散するシステムにおける認証サーバ1台あたりのチューニング

Windows32/64

Interstage HTTP Server
ThreadsPerChild＝100＋α(注1)
Timeout＝600(注2)

Solaris64 Linux32/64

Interstage HTTP Server
MaxClients＝100＋α(注1)
Timeout＝600(注2)

注1) システムを安定稼動させるため、αには10～100までの値を設定してください。
注2) 接続しているネットワークのトラフィックが増大し、接続が頻繁に中断される場合には、本時間を増やしてください。

G.3 1台のサーバにリポジトリサーバと認証サーバを構築する場合のチューニング

1台のサーバに、リポジトリサーバと認証サーバを構築する場合のチューニングについて説明します。

■Webサーバ(Interstage HTTP Server)のチューニング

リポジトリサーバ、および認証サーバのチューニングは、Webサーバ(Interstage HTTP Server)の環境定義を変更することにより行います。

詳細については、“付録E Interstage HTTP Serverの環境定義”を参照してください。

クライアントの同時接続数

以下のディレクティブに、想定する同時アクセス最大数×2以上を設定します。(初期値:50)

Windows32/64

ThreadsPerChild

Solaris64 Linux32/64

MaxClients

クライアント送受信タイムアウト時間

Timeoutにクライアントとの間でデータパケットを送受信するときに待機する最長の時間(秒)を指定します。(初期値:600)

◆チューニングの例


例

想定する同時アクセス最大数が256ユーザのシステムの場合

Windows32/64

Interstage HTTP Server
ThreadsPerChild＝256×2＋α(注1)
Timeout＝600(注2)

Solaris64 Linux32/64

Interstage HTTP Server
MaxClients＝256×2＋α(注1)
Timeout＝600(注2)

注1) システムを安定稼動させるため、αには10～100までの値を設定してください。
注2) 接続しているネットワークのトラフィックが増大し、接続が頻繁に中断される場合には、本時間を増やしてください。

■TCP/IPパラメタのチューニング Windows32/64

セション管理の運用を行う場合は、リポジトリサーバ(更新系)のマシンのTCP/IPパラメタのチューニングを行います。(注)

詳細については、“3.3 TCP/IPパラメタのチューニング”を参照してください。

注)リポジトリサーバ(更新系)への同時アクセス最大数が増大し、セション管理サーバとの通信に失敗し、以下のメッセージが出力される場合があります。この場合には、TCP/IPパラメタのチューニングを行ってください。

- sso00114
- sso00119

G.4 業務サーバを構築する場合のチューニング

業務サーバを構築する場合のチューニングについて説明します。
キャッシュサイズ、キャッシュ数、および使用するWebサーバのチューニングを行ってください。

■キャッシュサイズ、キャッシュ数のチューニング

セションの管理を行う運用の場合、認証した利用者の認証情報を業務サーバでキャッシュすることで認可性能を向上させることができます。キャッシュを効果的に行うためにはシステムの同時利用者数、および利用者の認証情報サイズに応じて、キャッシュサイズ、キャッシュ数を適切に設定する必要があります。

キャッシュサイズ、キャッシュ数の設定については、業務サーバのInterstage管理コンソールを使用して、[システム] > [セキュリティ] > [シングル・サインオン] > [業務システム] > [業務システム名] > [環境設定] > [詳細設定[表示]]をクリックし、以下より行います。

- [認証情報のキャッシュ]の[キャッシュサイズ]
- [認証情報のキャッシュ]の[キャッシュ数]

キャッシュサイズ

認証情報のサイズを以下の計算式で見積り、Kバイト単位で設定します。

認証情報サイズ = (150 + DNの文字列長
+ ユーザIDの文字列長
+ 認証方式の文字列長
+ ロールサイズ(注1)
+ 拡張ユーザ情報サイズ(注2)) × 1.4バイト

注1)設定するロール名の文字列長の総和 + 10×ロール数
注2)設定する属性名の文字列長の総和 + 属性値の文字列長の総和 + 10×拡張ユーザ情報数

キャッシュ数

認証情報のキャッシュは、利用者の最終アクセス時からアイドル監視時間が経過するまで保持します。アイドル監視時間内で想定する同時アクセス最大数 + α(注)を設定してください。

注)1人の利用者が、アイドル監視時間内にサインオン、サインオフを繰り返した場合でも、キャッシュ数を消費するため、想定する同時アクセス最大数よりも少し大きめの値を設定してください。

設定したキャッシュサイズ、キャッシュ数を超える利用があった場合も継続して利用可能ですが、システムのログにsso03062、もしくはsso03063のメッセージが出力されます。
認可性能が低下する可能性があるため、メッセージに従い対処してください。

◆チューニングの例

利用者の情報が以下の場合を例に説明します。

項目	文字列長	値
DN	55	cn=Fujitsu Tarou,ou=User,ou=interstage,o=fujitsu,dc=com
ユーザID	5	tarou
認証方式	19	basicAuthOrCertAuth
ロール名	5	Admin
ロール名	6	Leader
拡張ユーザ情報(属性名)	4	mail
拡張ユーザ情報(属性値)	20	tarou@jp.fujitsu.com
拡張ユーザ情報(属性名)	14	employeeNumber

拡張ユーザ情報(属性値)	6	100001

ロールサイズ

2つのロールが設定されるため、以下のようになります。
(Admin(5文字) + Leader(6文字)) + 10×ロール数(2個) = 31

拡張ユーザ情報サイズ

2つの属性が設定されるため、以下のようになります。
(mail(4文字) + employeeNumber(14文字)) + (tarou@jp.fujitsu.com(20文字) + 100001(6文字)) + 10×拡張ユーザ情報数(2個) = 64

上記より、認証情報サイズは以下のようになります。

認証情報サイズ = (150 + DNの文字列長(55文字)
+ ユーザIDの文字列長(5文字)
+ 認証方式の文字列長(19文字)
+ ロールサイズ(31文字)
+ 拡張ユーザ情報サイズ(64文字)) × 1.4バイト
= 約454バイト

[キャッシュサイズ]には、上記認証情報サイズを切り上げ、1Kバイトを設定します。

■Webサーバ(Interstage HTTP Server)のチューニング

詳細については、“付録E Interstage HTTP Serverの環境定義”を参照してください。

クライアントの同時接続数

以下のディレクティブに、想定する同時アクセス最大数以上を設定します。(初期値:50)

Windows32/64

ThreadsPerChild

Solaris64 Linux32/64

MaxClients

クライアント送受信タイムアウト時間

Timeoutにクライアントとの間でデータパケットを送受信するときに待機する最長の時間(秒)を指定します。(初期値:600)

■Webサーバ(Interstage HTTP Server 2.2)のチューニング

Interstage HTTP Server 2.2のチューニングについては、以下の定義を設定することによりチューニングを行います。

クライアントの同時接続数

以下のディレクティブに、想定する同時アクセス最大数以上を設定します。(初期値:50)

Windows32/64

ThreadsPerChild
詳細については、“Interstage HTTP Server 2.2 運用ガイド”の“概要”－“Webサーバのプロセス構成(Windows(R))”を参照してください。

Solaris64 Linux32/64

MaxClients
詳細については、“Interstage HTTP Server 2.2 運用ガイド”の“概要”－“Webサーバのプロセス構成(Solaris/Linux)”を参照してください。

クライアント送受信タイムアウト時間

Timeoutにクライアントとの間でデータパケットを送受信するときに待機する最長の時間(秒)を指定します。詳細については、“Interstage HTTP Server 2.2 運用ガイド”の“機能”－“タイムアウト時間”を参照してください。

■Microsoft(R) Internet Information Servicesのチューニング Windows32/64

Microsoft(R) Internet Information Servicesのチューニングについては、“Microsoft(R) Internet Information Services”のプロパティ情報に以下の定義を設定することによりチューニングを行います。

最大接続数

最大接続数フィールドに想定する同時アクセス数以上を設定します。(default:1,000)

EE

付録H　Portable-ORBの環境設定

Portable-ORBの動作環境ファイルは、porbeditenvコマンドを使用して設定します。Portable-ORBを使用する場合は、porbeditenvコマンドを使用して以下の環境設定を行ってください。

- 動作環境情報
- ホスト情報
- セキュリティ
- ネットワーク環境

porbeditenv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“porbeditenv”を参照してください。Portable-ORBの動作環境ファイルの詳細については、“アプリケーション作成ガイド(CORBAサービス編)”の“Portable-ORB動作環境ファイルの指定”を参照してください。

また、運用環境を構築した場合は、万が一に備えて、資源のバックアップを行うことを推奨します。資源のバックアップについては、“運用ガイド（基本編）”の“メンテナンス(資源のバックアップ/他サーバへの資源移行/ホスト情報の変更)”を参照してください。

付録I　RHEL7のunitファイルでの環境定義 Linux32/64

Linux(RHEL7)でオペレーティングシステム起動時に本製品のサービスを自動起動する場合、一部の環境設定は、各サービスが提供する起動用unitファイルに対して、定義内容を追加する必要があります。

unitファイルに追加の環境設定を行う場合は、システム管理者権限で以下の作業を行ってください。

■環境定義用unitファイルの作成

環境定義を追加するための新規のunitファイルを作成して、以下のように記載します。

```
.include /usr/lib/systemd/system/<起動用unitファイル名>

# 以降、追加の定義内容
```

注意

環境定義用unitファイルには、システム管理者だけが編集可能になるようアクセス権限を設定してください。

指定可能な<起動用unitファイル名>は以下です。

- FJSVtd_start.service
- FJSVihs_start.service
- FJSVahs_start.service
- FJSVpcmiisje6_start.service
- FJSVisjmx_start.service

例

環境定義用unitファイルの記載

環境定義用unitファイルの記載例を以下に示します。

```
.include /usr/lib/systemd/system/FJSVisjmx_start.service

[Service]
Environment="ENV1=env1"
```

■環境定義用unitファイルの配置と有効化

作成した環境定義用unitファイルを、ファイル名が<起動用unitファイル名>と同じになるようにして以下のパスに配置します。

```
/etc/systemd/system/<起動用unitファイル名>
```

次に、以下を実行します。

```
/usr/bin/systemctl enable <起動用unitファイル名>
```

上記を実行後、システムを再起動してください。

注意

<起動用unitファイル名>には、フルパスではなくファイル名だけを指定してください。

索 引

[は]

[ま]

[や]

[ら]

[わ]