FUJITSU

FUJITSU Software
Interstage Studio

ユーザーズガイド

B1WD-3158-02Z0(00)
2013年6月

まえがき

Interstage Studioは、Java EE 5およびJava EE 6のWebアプリケーションやEnterprise JavaBeans (EJB)などの開発に対応したJava統合開発環境です。現在広く使われているオープンソースの開発環境Eclipseをベースにしており、業界標準の操作性で開発を行うことができます。J2EE1.4のアプリケーションの開発も可能ですが、Java EEを推奨します。

本書の目的

本書は、ワークベンチが提供する機能について、開発方法、操作手順、開発時の留意事項を説明しています。

本書の読者

本書は、ワークベンチを使って、各種アプリケーションを開発する人のために必要な事項を説明しています。
本書は、読者がJava EEのWebアプリケーションやEJBなどに関する基本的な知識があることを前提としています。

本書の表記について

本書は以下の表記方法で記述しています。

- ボタン名、メニュー名、ダイアログボックス名、コンテキストメニューを[ ]で示します。
  例) コンテキストメニューから[追加] > [テンプレート]を選択します。
- ファイル、項目、関数などの名前は「 」で示します。
- 本製品がアプリ画面(Windows 8およびWindows Server 2012の場合)およびスタートメニュー(その他のWindowsの場合)に作成するグループ名を[Interstage Studio Vxx]と表記します。実際のグループ名はソフトウェア説明書で確認し、読み替えてお読みください。
- インストールフォルダの読み替えについてはソフトウェア説明書の"フォルダ構成とファイル"を参照してください。
  例)
  <ワークベンチのインストールフォルダ>はソフトウェア説明書の"フォルダ構成とファイル"の対応表で示す各機能のインストールフォルダです。

ワークベンチ名称

本書では、ワークベンチで共通の説明は「ワークベンチ」と表記します。ワークベンチを区別して説明する場合には、製品インストール時に標準でインストールされるワークベンチを「標準のワークベンチ」、Java EE 6開発機能を選択した時にインストールされるワークベンチを「Java EE 6ワークベンチ」と表記します。

輸出管理規制について

本ドキュメントを輸出または提供する場合は、外国為替、外国貿易法、米国輸出管理関連法規など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

- Microsoft、Active Directory、ActiveX、Excel、Internet Explorer、MS-DOS、MSDN、Visual Basic、Visual C++、Visual Studio、Windows、Windows NT、Windows Server、Win32 は、米国およびその他の国における 米国Microsoft Corporationの商標または登録商標です。
- OracleとJavaは、Oracle Corporation およびその子会社、関連会社の米国およびその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。
- その他の記載されている商標および登録商標については、一般に各社の商標または登録商標です。

出版年月および版数

出版年月および版数	マニュアルコード
2013年 6月　第2版	B1WD-3156-02Z0(00)/B1WD-3156-02Z2(00)
2012年 8月　初版	B1WD-3156-01Z0(00)/B1WD-3156-01Z2(00)

マニュアルの読み方

本書の構成および読み方を説明します。

マニュアル構成と読み方

本書は、標準のワークベンチとJava EE 6ワークベンチの開発方法を説明します。マニュアルの構成と読み方に従い、開発するワークベンチにあわせてお読みください。

標準のワークベンチでは、IJServerクラスタの使用例を記載しています。IJServerクラスタとInterstage Java EE DASサービスの両方で利用できる機能があります。機能差や使い分けについては、"Interstage Application Server Java EE運用ガイド"を参照してください。

Java EE 6アプリケーションを開発する場合には、最初に"第9章 Java EE 6アプリケーションを開発する"をお読みください。

操作説明について

WebアプリケーションやEJBなど、開発するアプリケーションの種別ごとに章を分けています。各章の"入門"は、各アプリケーションの開発手順をサンプルを用いて説明しています。"入門"の操作説明は、標準のワークベンチの操作を使って説明しています。Java EE 6ワークベンチで開発する場合には、設定項目が異なる場合があります。

Java EE 6ワークベンチの場合

- ターゲットランタイム/構成

ターゲットランタイム	Interstage Application Server V11.1(Java EE 6)
構成	Interstage Application Server V11.1(Java EE 6)デフォルト構成

- 操作手順や画面項目名について参照するヘルプ

目次	参照先	
	ヘルプ名	タイトル
9.2.1.3 Webアプリケーションの開発	Webツールプラットフォームユーザガイド	Webアプリケーションの作成
9.3.1.3 EJBの開発	Webツールプラットフォームユーザガイド	EJBアプリケーション開発
9.4.1.3 JPAを使用したアプリケーションの開発	Dali Java Persistence Tools User Guide	-
9.5.1.3 Webサービスアプリケーションの開発	Webツールプラットフォームユーザガイド	Webサービスアプリケーションの作成

各章の説明と対応するワークベンチ

各章の記述内容と対応するワークベンチを以下に示します。Java EE 6ワークベンチでアプリケーションを開発する場合には、"記載内容"に標準のワークベンチと異なる点を記載しています。本書の"操作説明について"とあわせてお読みください。

目次タイトル	記載内容	ワークベンチ	
		標準のワークベンチ	Java EE 6 ワークベンチ
第1章 ワークベンチの概要	エディタやビューなどのワークベンチの構成要素、ワークスペースやプロジェクトなどの開発資産の形式、および、ワークベンチの基本的な操作について説明します。	○	○
第2章 Webアプリケーションを開発する	Webアプリケーションの概要および、Webアプリケーションの作成方法について説明します。	○	○
第3章 Enterprise JavaBeans (EJB)を開発する	Enterprise JavaBeansの概要および、Enterprise BeanとEJBクライアントアプリケーションの作成方法について説明します。	○	○

目次タイトル	記載内容	ワークベンチ	
		標準のワークベンチ	Java EE 6 ワークベンチ
第4章 Java Persistence APIを使用したアプリケーションを開発する	Java Persistence APIの概要および、JPAを使用したアプリケーションの作成方法について説明します。	○	○
第5章 Webサービスのアプリケーションを開発する	Webサービスの概要および、Webサービスアプリケーションの作成方法について説明します。	○	○
第6章 Java EE 5アプリケーション共通事項	Java EE 5アプリケーションを作成するうえでの共通事項について説明します。	○	－
第7章 Javaアプリケーションを開発する	Javaアプリケーションの概要および、Javaアプリケーションの作成方法について説明します。	○	○
第8章 データベースを操作する	データベースの概要および、データベースの操作方法について説明します。 本章で説明する機能は、各製品から提供されているJDBCドライバの仕様により、正常に動作しない場合があります。データベースの内容に対する作成、更新、削除などの操作は、各製品から提供されているツールやアプリケーションの利用をお勧めします。	○	○
第9章 Java EE 6アプリケーションを開発する	Java EE 6アプリケーションの概要および、アプリケーションの作成方法から配備まで説明します。	－	○
第10章 Tips	ワークベンチを使用するうえでの便利な機能や、Java EEのアプリケーションプログラミングでの留意事項について説明します。	○	○
付録A サンプルの使用方法	提供しているサンプルについて説明します。	○	○
付録B 旧資産からの移行	旧バージョンのワークベンチで開発した資産の移行について説明します。	○	○
付録C JDK 6を用いた開発を行う手順	Java EE 6ワークベンチでの開発でJDK 6を用いるための手順を説明します。	－	○
付録D J2EE1.4アプリケーションの開発について	J2EE1.4アプリケーションの開発方法について説明します。	○	－
付録E トラブルシューティング	ワークベンチで発生する問題を解決する方法について説明します。	○	○
付録F チュートリアル	アプリケーションの開発手順を例題を使用して説明します。	○	－

○：対応しています。

－：対応していません。

目　次

第1章 ワークベンチの概要

ここでは、Eclipseを使用した経験が無い方のために、ワークベンチを使用するうえで知っておくべき基本的な概念、および、ワークベンチの基本的な操作について説明します。

1.1 ワークベンチの基本概念

ここではワークベンチの構成要素、開発資産の形式、および、ワークベンチで行う主な作業について説明します。

1.1.1 ワークベンチとその構成要素

ワークベンチ

ワークベンチはユーザが開発資産の作成、編集、ビルド、動作確認および、開発資産の管理などの操作を行う場となるものです。

エディタ

ワークベンチには1つのエディタ領域があります。プロジェクトエクスプローラビューなどでファイルを開く操作を行うと、エディタ領域にエディタが表示されます。デフォルトでは、エディタ領域には複数のエディタが重ねて表示されます。エディタ領域を左右や上下に分割して同時に2つ以上のエディタを表示することもできます。

エディタ領域の左端には垂直方向ルーラーが表示されます。垂直方向ルーラーにはエディタ領域に表示されている範囲に付いているマーカが表示されます。例えば、エラーや警告のマーカ、ブレークポイント、検索で一致した箇所を表すマーカなどが表示されます。エディタ領域の右端には概説ルーラーが表示されます。概説ルーラーにはエディタ領域に表示されていない部分も含めてファイル全体でどこにどのようなマーカが付いているかが表示されます。概説ルーラーのマーカをクリックすると、そのマーカの周辺がエディタ領域に表示されます。

ポイント

ワークベンチからは、ユーザの使いたいアプリケーションを外部エディタとして起動することもできます。外部エディタはエディタ領域には表示されず、上記のような支援機能も利用することができません。

ビュー

ビューは開発作業をサポートするものです。例えば以下のようなビューがあります。

- 開発資産を表示・管理するもの（例：プロジェクトエクスプローラビュー、ナビゲータビュー）
- 編集作業をサポートするもの（例：アウトラインビュー、スニペットビュー）
- デバッグ作業をサポートするもの（例：ブレークポイントビュー、変数ビュー）

ビューには独自のツールバーまたはメニューがあります。これらのツールバーおよびメニューで行った操作はそのビュー内の項目にだけ影響を与えます。

注意

1つのワークベンチに同じ種類のビューを複数表示することはできません。

パースペクティブ

パースペクティブとは、ワークベンチに表示されるエディタやビューの初期セットとそのレイアウト、および、表示するメニューやツールバーの項目を定義したものです。特定の作業を行うためのパースペクティブがあらかじめ用意されています。例えば、Java EEパースペクティブはJava EEアプリケーションの開発時に使用するビューが含まれ、デバッグパースペクティブはアプリケーションのデバッグに使用するビューが含まれています。ユーザは作業内容に応じて適切なパースペクティブを使用できます。

1.1.2 開発資産

ワークスペース

ワークスペースとは開発資産およびユーザの作業状態が保存される場所です。開発資産はプロジェクトとしてワークスペースに管理されます。ワークスペースには複数のプロジェクトを作成することができます。その他に、ユーザが行った作業の状態、例えばワークベンチの設定情報や、ソースファイルに設定したブレークポイントの情報などがワークスペースに保存されます。

ワークスペースはデフォルトで次の場所に作成されます。

ワークベンチ

〈ユーザのドキュメントフォルダ〉¥Interstage Studio¥〈製品バージョン〉¥workspace

Java EE 6ワークベンチ

〈ユーザのドキュメントフォルダ〉¥Interstage Studio¥〈製品バージョン〉¥workspace_jee6

ワークスペースは必要に応じて複数作成することが可能です。ユーザはワークベンチの起動時に、使用するワークスペースを選ぶことができます。ワークベンチで一度に開けるワークスペースは1つだけです。

ワークスペースのフォルダ名にIVS文字が含まれていると、ソースファイルの編集やビルド時にエラーが発生するなどの不具合が発生する場合があります。

ワークスペースのフォルダ名にはIVS文字を含めないようにしてください。

ワークベンチの起動後にメニューから[ファイル] > [ワークスペースの切り替え]を選択することで、ワークスペースを切り替えることもできます。

プロジェクト

プロジェクトとは、アプリケーション開発を行うための開発資産の管理単位のことです。アプリケーションのビルドやデバッグはプロジェクト単位に行います。ビルドに必要なクラスパスの設定など、アプリケーション開発に必要な各種設定もプロジェクト単位で行います。

プロジェクトの作成にはプロジェクト生成ウィザードを使用します。Interstage StudioではWebアプリケーション用、EJB用などの各種プロジェクト生成ウィザードを提供しています。これらのウィザードを使用すると、プロジェクト生成時に、アプリケーションの開発に必要な設定が行われます。

プロジェクト配下の資産は、通常のファイルやフォルダとして保存されています。このファイルやフォルダをワークベンチ以外から変更したり、ファイルをシステムエディタで変更した場合は、ワークベンチに対して変更を反映させる必要があります。

例えば、ワークベンチ以外から変更した後に、メニューから[ファイル] > [エクスポート]を実行すると、エラーとなります。このような場合は、メニューから[ファイル] > [更新]を行って変更を反映した後にエクスポートしてください。ただし、ワークスペースフォルダにプロジェクトフォルダをコピーした場合、この方法ではワークベンチに反映できません。プロジェクトをワークベンチに反映するには、メニューから[ファイル] > [インポート]を行ってください。

1.1.3 ワークベンチで行う主な作業

開発資産の作成

ワークベンチでの開発資産の作成は、一般的には、ウィザードでファイルを生成し、エディタでそのファイルを編集するという手順で行います。Interstage Studioでは各種ファイルを生成するウィザードおよび、それらを編集するエディタを提供しています。

ビルド

ビルドとは、一般的にはソースファイルから実行形式ファイルを作成する作業のことを指します。ワークベンチではビルダを呼び出すことでビルドを実行しており、プロジェクト種別によって呼び出すビルダを決定しています。

例えば、Javaのソースファイルをコンパイルしてクラスファイルを作成する作業はビルド作業の1つであり、Javaビルダとして機能が提供されています。

ビルドには以下の2種類があります。

- 差分ビルド: 変更されたソースだけをビルドする方法です。
- クリーンビルド: すべてのビルド結果を一度削除してから開発資産全体を再ビルドする方法です。

通常は差分ビルドが行われます。クリーンビルドを行うにはユーザが明にプロジェクトのクリーンを行う必要があります。

ビルドを行うタイミングにも以下の2種類があります。

- 自動ビルド: ソースファイルが変更され保存されたタイミングで自動的にビルドが行われます。
- 手動ビルド: ユーザがメニューから手動でビルドを実行します。

デフォルト設定では自動ビルドが行われます。このためユーザがビルド作業を意識する必要はありません。

ビルド時にコンパイルエラー、警告、または、その他の情報（これらをまとめて"問題"と呼びます）が出た場合、それらは問題ビューに表示されます。問題ビューに表示された問題をダブルクリックすると、その問題が発生したソースの行をエディタで表示することができます。コンパイルエラーや警告などの問題はエディタ上にも表示されます。例えば、問題があることを示すマーカがエディタ領域の左端の垂直方向ルーラーに表示されたり、問題がある箇所に波線が表示されたりします。

ポイント

開発の規模が大きくなってくると、ソースファイルを保存するたびに自動的にビルドが行われることでユーザの操作が妨げられる場合があります。こういった場合には、手動ビルドにすることで、ビルドのタイミングを制御し、ユーザの操作が妨げられることを少なくできます。

アプリケーションの動作確認

作成したアプリケーションはワークベンチから実行またはデバッグすることができます。

アプリケーションの実行およびデバッグには起動構成を使用します。起動構成とはアプリケーションの実行に必要な各種設定を登録しておくためのものです。例えば、使用するアプリケーションサーバや、Java VMの起動オプションなどが設定できます。一度作成した起動構成は再利用できるので、アプリケーションを繰返し実行およびデバッグするのに便利です。

1.2 ワークベンチの基本操作

ここではワークベンチの基本的な操作について説明します。

1.2.1 開発資産に対する操作

プロジェクトを新規作成する

プロジェクトの新規作成は新規プロジェクトウィザードから行います。新規プロジェクトウィザードを起動するには次のどれかの手順を行ってください。

- ワークベンチのメニューから[ファイル] > [新規] > [プロジェクト]を選択します。
- ワークベンチのツールバーにある[新規]ボタンの右の▼を押して、サブメニューから[プロジェクト]を選択します。
- プロジェクトエクスプローラビューやナビゲータビューのコンテキストメニューから[新規] > [プロジェクト]を選択します。

新規プロジェクトウィザードが起動されたら作成したいプロジェクトを選択して、プロジェクト生成ウィザードを起動してください。

ソースファイルを新規作成する

ソースファイルの新規作成は新規ウィザードから行います。新規ウィザードを起動するには次のどれかの手順を行ってください。

- ワークベンチのメニューから[ファイル] > [新規] > [その他]を選択します。
- ワークベンチのツールバーにある[新規]ボタンをクリックします。
- プロジェクトエクスプローラビューやナビゲータビューのコンテキストメニューから[新規] > [その他]を選択します。

新規ウィザードが起動されたら作成したいものを選択して、ソース生成ウィザードを起動してください。

ポイント

コンテキストメニューの[新規]のサブメニューには良く使われるソース生成ウィザードが一覧表示されており、直接、ソース生成ウィザードを起動することもできます。

プロジェクトの設定(プロパティ)を変更する

プロジェクトエクスプローラビューやナビゲータビューでプロジェクトを選択し、コンテキストメニューから[プロパティ]を選択します。プロジェクトのプロパティダイアログボックスが表示されるので、左側のツリーよりプロパティページを選択し、必要な設定変更を行います。

1.2.2 ワークベンチに対する操作

エディタでファイルを編集する

プロジェクトエクスプローラビューやナビゲータビューでファイルをダブルクリックする、または、ファイルのコンテキストメニューから[開く]を選択すると、そのファイルがエディタで開かれ、編集できます。編集するとエディタのタブの部分に未保存を示す"*"が表示されます。

エディタを選択してファイルを開く

プロジェクトエクスプローラビューやナビゲータビューでファイルを選択し、コンテキストメニューから[アプリケーションから開く]を選択します。サブメニューにそのファイルを開くことのできるエディタの一覧が表示されるので、使用するエディタを選択します。

ビューを表示する

ワークベンチのメニューから[ウィンドウ] > [ビューの表示]を選択すると、ビューを表示することができます。

[ビューの表示]のサブメニューには良く使われるビューが一覧表示されています。表示したいビューがその中にない場合には[その他]を選択してください。[ビューの表示]ダイアログボックスが表示され、すべてのビューの中から表示したいビューを選ぶことができます。

ワークベンチの設定を変更する

ワークベンチのメニューから[ウィンドウ] > [設定]を選択します。ワークベンチの設定ダイアログボックスが表示されるので、左側のツリーより設定ページを選択し、必要な設定変更を行います。

パースペクティブを開く

ワークベンチのメニューから[ウィンドウ] > [パースペクティブを開く]を選択する、または、ワークベンチのツールバーにある[パースペクティブを開く]ボタンを押すと、パースペクティブを開くことができます。

[パースペクティブを開く]のサブメニューには良く使われるパースペクティブが一覧表示されています。開きたいパースペクティブがその中にない場合には[その他]を選択してください。[パースペクティブを開く]ダイアログボックスが表示され、すべてのパースペクティブの中から開きたいものを選ぶことができます。

1.2.3 主な作業に関する操作

自動/手動ビルドを切り替える

ビルドの自動/手動を切り替えるには次のどちらかの手順を行ってください。

- ワークベンチのメニューで[プロジェクト] > [自動的にビルド]を選択します。このメニューはトグル項目になっており、選択のたびにビルドの自動/手動が切り替わります。
- ワークベンチの設定ダイアログボックスの [一般] > [ワークスペース]にある [自動的にビルド]チェックボックスで切り替えます。

プロジェクトを手動でビルドする

ビルドの設定を手動ビルドにした場合、次のどちらかの手順で手動ビルドを行ってください。

- プロジェクトエクスプローラビューやナビゲータビューでプロジェクトを選択し、ワークベンチのメニューで[プロジェクト] > [プロジェクトのビルド]を選択します。
  [プロジェクトのビルド]の代わりに[すべてビルド]を選択すると、ワークスペースにあるすべてのプロジェクトをビルドします。
- 標準のワークベンチの場合、プロジェクトエクスプローラビューやナビゲータビューでプロジェクトを選択し、コンテキストメニューから[プロジェクトのビルド]を選択します。

プロジェクトをクリーンする

ビルド結果を削除するには、ワークベンチのメニューで[プロジェクト] > [クリーン]を選択します。ダイアログボックスが表示されるので、クリーンするプロジェクトを選択します。

第2章 Webアプリケーションを開発する

ここでは、Webアプリケーションの概要および、Java EEアプリケーション実行環境上で動作する、Webアプリケーションの作成方法について説明します。

2.1 概要

WebアプリケーションおよびWebアプリケーションの開発支援機能の概要について説明します。

2.1.1 Webアプリケーションとは

WebアプリケーションはWebブラウザをクライアントとして利用するアプリケーションです。WebサーバによるHTTPを利用したWebの配信機能に、サーブレットコンテナによる動的なコンテンツの利用技術を追加した仕組みがベースとなっています。
ユーザインタフェースとしてのクライアント層や、EJBなどで実装されるビジネスロジックへの橋渡しを行うプレゼンテーション層として利用します。

Webアプリケーションの動作イメージとしては、以下のようになります。

1. WebアプリケーションのURLにアクセスする
2. Webブラウザ上にWebページが表示される
3. HTMLのフォームを使用してデータを入力して、データ(HTTP要求)を送信する
4. サーバ側でデータを処理し、クライアントに応答ページ(HTTP応答)を返信する
5. 2.に戻り、処理が繰り返される

Webアプリケーションを作成するうえで利用される技術を簡単に説明します。

HTTP

HTTP（HyperText Transfer Protocol）は、Webサーバとクライアント（ブラウザなど）間で送受信されるプロトコルです。クライアントからWebサーバへリクエスト（要求）を送信すると、Webサーバはリクエストに応じたレスポンス（応答）を返します。
良く使われるリクエストとして、ページの取得を要求するGETメソッドと、フォームに入力したデータをサーバに転送するためのPOSTメソッドがあります。

HTML

文書の論理構造を記述するために考案された、タグ付けによる文書記述の規約です。インターネットによる情報公開で用いられ、事実上、Webブラウザのための記述言語となり、Webブラウザの進歩とともに規約が拡張されています。

カスケードスタイルシート（CSS）

HTMLのスタイル情報を持つファイルです。このファイルによって、ページのブラウザ上でのレイアウト、デザインを細かく指定することができます。HTML規約の拡張に伴い、HTML内に見た目の情報が記述できるようになりましたが、本来は論理構造を記述することが目的であったことや、サイト内の見た目を統一するのに複数のHTMLへの修正が必要という問題があり、それを改善するため、見た目の情報をスタイルシートとして分離するCSSが考案されました。

JavaScript

HTML文書に、計算した結果を表示させるなどの動作をつけるために作成されたスクリプト言語です。現在ではほとんどすべてのブラウザで動作するようになりました。APIの実装などがブラウザによって異なるため、実装の際には注意が必要です。

サーブレット

Webサーバ上で動的にHTMLドキュメントを生成する技術であり、Webブラウザから入力したデータをサーバ上で処理を行うことができます。また、それまでのWebサーバ上の技術として主流であったCGI(Common Gateway Interface)の問題点である、多重度、セション管理などの問題を解決する技術を提供しています。

サーブレットクラスを作成するには、Servletインタフェースを実装する必要があり、要求オブジェクトからのパラメタの取り出しや応答オブジェクトの作成をメソッドとして実装します。応答オブジェクトの作成では、通常はHTMLドキュメントの出力を行います。しかし、JSPが

考案された現在では、動的HTMLドキュメントの作成はJSPで行うのが一般的であり、サーブレットでは、EJBサービスやJSPへの処理の振り分けなどを行います。

また、サーブレットは初めての要求時に初期化され、インスタンスはそれ以降の要求で共有されます。そのため、サーブレットはスレッドセーフとして作成する必要があります。

JSP（JavaServer Pages）

動的なHTMLコンテンツを作る場合には、サーブレット技術を利用しますが、サーブレットのプログラム内に、サーバで処理するロジックとHTMLコンテンツが混在し、記述性が良くないという問題がありました。そこで、動的コンテンツ(データの埋め込みなど)を出力するために、HTMLファイルにJava記述を可能にしたものがJSPです。

JSPは、運用時にサーバ上でサーブレットクラスに変換されてから実行されます。そのため、実行の基本的な技術はサーブレットと同じになります。

JSTL

JSTL（JSP Standard Tag Library）は、Java Community Process（JCP）によってJSR-052として標準化されたJSPのタグライブラリです。よく利用される共通的な機能をタグライブラリとしてまとめて提供しています。

JSF（JavaServer Faces）

JSF（JavaServer Faces）とは、JSR-127としてJava Community Process (JCP)で策定された仕様で、Webアプリケーションのユーザインタフェースを作成するためのアプリケーションフレームワークです。

2.1.1.1 J2EE1.4からの変更点

Java EEに含まれるWebアプリケーションの仕様はServlet 2.5です。また、JSPの仕様はJSP 2.1です。

Servlet 2.5では主に以下の変更があります。

- アノテーションが利用可能になりました。
- web.xmlの記述形式が変更されました。

また、JSP 2.1では、主に以下の機能が追加されています。

- 式言語として、Unified ELが利用可能になりました。

アノテーション

Servlet 2.5で、アノテーションが利用可能になり、Dependency Injectionが可能になりました。これにより、JNDIのlookup処理をソースに記述する必要がなくなり、参照しているリソースの宣言の記述などもweb.xmlに不要になります。

以下に主なアノテーションを示します。

- @Resource
  サーブレットコンテナに対して、Dependency Injectionを行うことを指示します。web.xmlの <resource-ref>要素や<env-ref>要素、<resource-env-ref>要素と同じ働きをします。

- @EJB
  EJBの参照を宣言します。web.xmlの<ejb-ref>要素や<ejb-local-ref>要素と同じ働きをします。

- @DeclareRoles
  セキュリティロールを指定します。web.xmlの<security-role>要素と同じ働きをします。

- @RunAs
  実行時のロールを指定します。web.xmlの<run-as>要素と同じ働きをします。

web.xmlの変更

web.xmlの記述方法が改善されました。複数のパターンや要素を指定する際の記述方法が変更され、まとめて記述することができるようになりました。この結果、記述量が減り、可読性も良くなりました。

- filter-mappingでワイルドカードを利用できる
  <filter-mapping>の<servlet-name>要素で、ワイルドカード"* "を使えるようになりました。これまでは、すべてのサーブレットを列挙する必要がありました。

- url-patternを複数指定できる
  <servlet-mapping>や<filter-mapping>にて、複数のパターンのマッピングを指定できるようになりました。

Unified ELが利用可能

JSP2.0で導入されたEL（Expression Language、式言語）が、JSFの式言語と統一されました。Unified ELでは、${}の形式の表記と#{}の形式の表記の両方が利用可能です。

2.1.2 Webアプリケーションの開発

Webアプリケーションの開発の概要を以下に示します。

Webアプリケーションの開発の準備

Webアプリケーションを開発するには、まずWebアプリケーションのプロジェクトを作成する必要があります。動的Webプロジェクトを作成し、必要なターゲットランタイムやクラスパスなどのプロジェクト設定を行います。
詳細は、"2.3.1 Webアプリケーションを作成する環境を準備する"を参照してください。

サーブレット/HTML/JSP/JavaScript/CSSの作成

各種ソース生成ウィザード、エディタを用いてWebページを表示するのに必要なリソースを作成します。
詳細は、"2.3.2 サーブレットを作成する"、"2.3.4 JSPファイルを作成する"、"2.3.3 HTMLファイルを作成する"、"2.3.6 JavaScriptを作成する"、"2.3.7 CSSを作成する"を参照してください。

Webアプリケーションのビルド、検証

初期状態では、リソースを保存したときに自動的にビルドが行われます。ビルドでは、コンパイルエラーのほかに様々なバリデータによって資産に問題がないか検証が行われます。
詳細は、"ビルド"または"6.2.6.2 検証"を参照してください。

Webアプリケーションの動作確認

作成したWebアプリケーションの動作確認はサーバビューを用いて行います。
詳細は、"2.3.9 Webアプリケーションの動作を確認する"を参照してください。

Webアプリケーションの配布

Webアプリケーションを配布するためにはWebアプリケーションをアーカイブする必要があります。アーカイブファイルの作成にはエクスポートウィザードを使用します。
詳細は、"2.3.11 Webアプリケーションを運用環境に配布する"を参照してください。

2.2 入門

ここでは、Webアプリケーションを作成する手順を紹介します。

2.2.1 作成するアプリケーション

国の総人口の順位付けリストから、順位を入力して国名と総人口を返し、その結果をWebページに表示するWebアプリケーションを作成します。このアプリケーションでは1～10位までの情報だけとし、0以下または11以上の数字が入力された場合にエラーを出力します。順位の入力項目に文字列が入力された場合もエラーを出力します。

2.2.2 開発の流れ

ここでは、以下のようにWebアプリケーションの開発を進めます。

1. Webアプリケーション用のプロジェクトの作成
   Webアプリケーションを作成するために、まず動的Webアプリケーションプロジェクトを作成します。ウィザードに従って動的Webアプリケーションプロジェクトを作成することで、ビルドに必要なクラスパスの設定などが自動的に行われます。

2. Javaクラスの作成
   アプリケーションで必要な2つのクラスを作成します。

   - データクラスの作成
   - ロジッククラスの作成

3. サーブレットクラスの作成
   以下の手順でサーブレットクラスを作成します。

   - サーブレットクラスのひな型の作成
     ウィザードでサーブレットクラスのひな型を作成します。

   - サーブレットクラスの実装
     Javaエディタでサーブレットクラスの実装を行います。

4. 入出力画面の作成
   アプリケーションの入出力画面を、ウィザードでひな型を作成し、エディタで編集します。

   - 入力画面のひな型の作成
   - 入力画面の編集
   - 出力画面のひな型の作成
   - 出力画面の編集
   - エラー画面のひな型の作成
   - エラー画面の編集

5. アプリケーションの動作確認
   以下の手順でアプリケーションの動作確認を行います。

   - プロジェクトとサーバの関連付け
     アプリケーションをどのサーバに配備するかを設定します。

   - ブレークポイントの設定
     実行時にデバッガでプログラムの動作を確認するため、ブレークポイントを設定します。

   - サーバの起動
     Webブラウザからのリクエストをアプリケーションが受けつけられるように、サーバを起動します。サーバ起動前に配備は自動的に行われます。

   - アプリケーションの実行
     WebブラウザをURLにアクセスすることで動作確認を開始します。

   - アプリケーションのデバッグ
     プログラムをデバッグし、アプリケーションが正常に動作することを確認します。

6. 運用環境へのアプリケーションの配布
   以下の手順で運用環境への配布を行います。

   - アプリケーションのエクスポート
     運用環境へアプリケーションを配布するため、WARファイルを作成します。

   - 運用環境への配布
     Interstage管理コンソールからWARファイルを配備します。

2.2.3 開発手順

以下に、実際にアプリケーションを開発する手順を説明します。

1) Webアプリケーション用のプロジェクトの作成

2) Javaクラスの作成

3) サーブレットクラスの作成

4) 入出力画面の作成

5) アプリケーションの動作確認

6) 運用環境へのアプリケーションの配布

1) Webアプリケーション用のプロジェクトの作成

メニューバーから[ファイル] > [新規] > [プロジェクト]を選択すると、[新規プロジェクト]ウィザードが表示されます。

[新規プロジェクト]ウィザードから[Web] > [動的 Web プロジェクト]を選択して、[次へ]をクリックします。

以下の設定項目を確認、入力してください。

設定項目	設定内容
プロジェクト名	WebSample
ターゲットランタイム	Interstage Application Server V11.1 IJServer Cluster (Java EE)
動的 Web モジュールバージョン	2.5
構成	Interstage Application Server V11.1 IJServer Cluster (Java EE) デフォルト構成
EARにプロジェクトを追加	チェックしない

情報を設定後、[次へ]をクリックしてください。Webモジュールページが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
コンテキストルート	WebSample
コンテンツフォルダ	WebContent
Java ソースフォルダ	src
Deployment Descriptor の生成	チェックする

2) Javaクラスの作成

2-1) データクラスの作成

国名と総人口の情報を保持し、そこから情報を取得するデータクラスを作成します。データクラスの作成は、作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [クラス]を選択します。[新規Javaクラス]ウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
ソースフォルダ	WebSample/src
パッケージ	sample
名前	CountryData

以下のソースファイルが生成されます。

ソースファイル	説明
CountryData.java	データクラス

国名と総人口を保持する処理を実装します。以下の**赤字**の箇所をソースに追加してください。

データクラスの実装(CountryData.java)

```
package sample;

public class CountryData {

    private String countryName;
    private int totalPopulation;

    public CountryData(String name, int total) {
        setCountryName(name);
        setTotalPopulation(total);
    }

    public String getCountryName() {
        return countryName;
    }

    public void setCountryName(String countryName) {
        this.countryName = countryName;
    }

    public int getTotalPopulation() {
        return totalPopulation;
    }

    public void setTotalPopulation(int totalPopulation) {
        this.totalPopulation = totalPopulation;
    }

}
```

ポイント

フィールドを追加した後にクラスを選択している状態で、メニューから[ソース] > [Getter および Setter の生成]を選択することで、getter/setterの追加を行うことができます。

2-2) ロジッククラスの作成

ロジッククラスの作成は、作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [クラス]を選択します。[新規Javaクラス]ウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
ソースフォルダ	WebSample/src
パッケージ	sample
名前	PopulationRanking

以下のソースファイルが生成されます。

ソースファイル	説明
PopulationRanking.java	ロジッククラス

順位を入力して国名と総人口を返す処理を実装します。以下の**赤字**の箇所をソースに追加してください。

ロジッククラスの実装(PopulationRanking.java)

```
package sample;

public class PopulationRanking {

    private CountryData[] countries;

    public PopulationRanking(){

        countries = new CountryData[]{
            new CountryData("中国",1330000000),
            new CountryData("インド",1140000000),
            new CountryData("アメリカ",300000000),
            new CountryData("インドネシア",230000000),
            new CountryData("ブラジル",190000000),
            new CountryData("パキスタン",160000000),
            new CountryData("バングラデシュ",150000000),
            new CountryData("ロシア",140000000),
            new CountryData("ナイジェリア",140000000),
            new CountryData("日本",130000000)
        };

    }

    public CountryData getCountryData(int rank) {
        --rank;
        if (rank < 0 || rank >= countries.length) {
            return null;
        }

        return countries[rank];
    }

}
```

3) サーブレットクラスの作成

3-1) サーブレットクラスのひな型の作成

サーブレットクラスのひな型の作成は、作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [サーブレット]を選択します。[サーブレット作成]ウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[次へ]をクリックしてください。

設定項目	設定内容
Web プロジェクト	WebSample

設定項目	設定内容
ソースフォルダ	¥WebSample¥src
Java パッケージ	sample
クラス名	ServletController
スーパークラス	javax.servlet.http.HttpServlet

以下の設定項目を確認、何も変更せずに、[次へ]をクリックしてください。

設定項目	設定内容
名前	ServletController
URL マッピング	/ServletController

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
修飾子	public
作成するメソッドスタブの選択	スーパークラスからのコンストラクタ 継承された抽象メソッド doPost（doGetのチェックを外す）

以下のソースファイルが生成されます。

ソースファイル	説明
ServletController.java	サーブレットクラス

ポイント

ウィザードで指定した内容でweb.xmlにサーブレットマッピングの定義も追加されています。このアプリケーションでは必要ありませんが、web.xmlの編集については、"2.3.8 web.xmlを編集する"を参照してください。

3-2) サーブレットクラスの実装

作成されたサーブレットクラスの赤字の箇所をソースに追加してください。

サーブレットクラス(ServletController.java)

```
package sample;

import java.io.IOException;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class ServletController
 *
 */
public class ServletController extends HttpServlet {
    private static final long serialVersionUID = 1L;
```

```
    /**
     * @see HttpServlet#HttpServlet()
     */
    public ServletController() {
        super();
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
        // TODO Auto-generated method stub
        request.setCharacterEncoding("Windows-31J");
        ServletContext sc = getServletContext();

        // 呼び出すファイル種別取得
        String mode = request.getParameter("mode");
        if (mode != null && mode.equals("top")) {
            // 入力画面のHTMLファイル呼出し
            RequestDispatcher inRd = getServletContext().getRequestDispatcher("/default.jsp");
            inRd.forward(request, response);
            return;
        }

        int rank;

        // 入力チェック
        try {
            rank = Integer.parseInt(request.getParameter("rank"));
        } catch (NumberFormatException e){
            RequestDispatcher errRd = sc.getRequestDispatcher("/error.jsp");
            errRd.forward(request, response);
            return;
        }

        PopulationRanking pop = new PopulationRanking();
        CountryData country = pop.getCountryData(rank);

        // 取得した値の確認
        if(country == null){
            RequestDispatcher errRd = sc.getRequestDispatcher("/error.jsp");
            errRd.forward(request, response);
            return;
        }

        sc.setAttribute("ranking", country);

        RequestDispatcher outRd = sc.getRequestDispatcher("/result.jsp");
        outRd.forward(request, response);

    }
}
```

ポイント

必要な実装をコーディングした後にJavaエディタ上で右クリックし、コンテキストメニューから[ソース] > [インポートの編成]を行うと、プロジェクトに設定されたクラスパスに従った適切なimport文が挿入できます。

4) 入出力画面の作成

4-1) 入力画面のひな型の作成

入力画面となるJSPファイルのひな型を作成します。JSPの作成は、作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [JSP]を選択します。[新規 JavaServer Page]ウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[次へ]をクリックしてください。

設定項目	設定内容
親フォルダを入力または選択	WebSample/WebContent
ファイル名	default.jsp

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。JSPが表示されます。

設定項目	設定内容
JSP テンプレートの使用	チェックする
名前	新規 JSP ファイル (HTML)

4-2) 入力画面の編集

作成されたJSPファイルの赤字の部分を修正します。JSPファイルの編集については、"2.3.4.2 JSPファイルの編集"を参照してください。

入力画面(default.jsp)

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>世界総人口ランキング</title>
</head>
<body>
<h1>入力画面</h1>
<p>1～10の間で順位を入力してください。</p>
<p></p>
<form action="ServletController" method="post">
    順位：<br>
    <input name="rank" type="text" size="10">位<br>
    <p></p>
    <input type="submit" value="OK">
    <input type="reset" value="クリア">
</form>

</body>
</html>
```

ポイント

JSPファイルはデフォルトでは、JSPエディタに対応付けられていますが、JSPファイルを選択し、コンテキストメニューから[アプリケーションから開く] > [Web ページエディタ]を選択することで、デザインやレイアウトを確認しながら編集できるWebページエディタを使用することもできます。Webページエディタでの編集については、"2.3.5 HTML/JSPファイルをグラフィカルに編集する"を参照してください。

4-3) 出力画面のひな型の作成

入力画面と同様に、ウィザードでJSPファイルを作成します。

以下の値を入力し、[完了]をクリックします。表にない項目は初期値のままでかまいません。

設定項目	設定内容
親フォルダを入力または選択	WebSample/WebContent
ファイル名	result.jsp

4-4) 出力画面の編集

作成されたJSPファイルの赤字の部分を修正します。

出力画面(result.jsp)

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>世界総人口ランキング</title>
</head>
<body>
<h1>出力画面</h1>

総人口ランキング${param.rank}位は「${applicationScope.ranking.countryName}」です。<br>
総人口は${applicationScope.ranking.totalPopulation}人です。
<br>
<hr>
<form action="ServletController" method="post">
    <input type="submit" value="入り口に戻る">
    <input type="hidden" name="mode" value="top">
</form>

</body>
</html>
```

4-5) エラー画面のひな型の作成

入力画面と同様に、ウィザードでJSPファイルを作成します。
以下の値を入力し、[完了]をクリックします。表にない項目は初期値のままでかまいません。

設定項目	設定内容
親フォルダを入力または選択	WebSample/WebContent
ファイル名	error.jsp

4-6) エラー画面の編集

作成されたJSPファイルの赤字の部分を修正します。

エラー画面(error.jsp)

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>世界総人口ランキング</title>
</head>
<body>
```

```
<h1>エラー画面</h1>
指定された範囲内の順位が入力されていないか、またはサーバエラーです。<br>
<br>
<hr>
<form action="ServletController" method="post">
    <input type="submit" value="入り口に戻る">
    <input type="hidden" name="mode" value="top">
</form>

</body>
</html>
```

5) アプリケーションの動作確認

5-1) プロジェクトとサーバの関連付け

サーバビューでサーバを選択し、コンテキストメニューから[プロジェクトの追加および除去]を選択してください。
使用可能なプロジェクトから作成したプロジェクトを選択し、[追加]をクリックしてください。構成プロジェクトに動的Webプロジェクトが追加されていることを確認してください。

以下の設定項目を確認後、[完了]をクリックしてください。

設定項目	設定内容
構成プロジェクト	WebSample
サーバが始動済みの場合、直ちに変更を反映する。	チェックする

ポイント

サーバビューに配備先となるサーバが登録されていない場合は、サーバを追加する必要があります。サーバを追加する方法については、"6.2.7 アプリケーションの動作確認"を参照してください。

5-2) ブレークポイントの設定

サーブレットクラスのdoPost()メソッドの先頭行にブレークポイントを設定します。ブレークポイントは、エディタの左側のルーラー部分をダブルクリックして、設定もしくは解除します。

ポイント

JPSファイルについても同様にJSPエディタでブレークポイントを設定することができます。ただし、ブレークポイントは、JSPタグが記載されている行にしか設定できません。(HTMLタグなどで記載されている部分にブレークポイントを設定することはできません。)

注意

JSPファイルにブレークポイントを設定した場合、異なるフォルダにある同名のJSPファイルでも実行が中断します。

5-3) サーバの起動

サーバビューでサーバを選択し、コンテキストメニューから[デバッグで再始動]を選択してください。

ポイント

配備操作を行う前に[デバッグで再始動]や[接続(デバッグ起動)/ログイン]を行っていても、デバッグはできません。

サーバビューで以下の情報を確認してください。

確認項目	確認内容
サーバの状態	デバッグ
サーバの状況	同期済み
WARファイル（WebSample）の状況	同期済み

5-4) アプリケーションの実行

サーバビューで、配備されているプロジェクトを選択して、コンテキストメニューから[Webブラウザ]を選択します。Webブラウザが起動されて、入力画面が開きます。

設定項目	設定内容
入力画面のURL	http://localhost/WebSample/

[順位:]の入力フィールドに人口ランキングの順位を入力して、[OK]をクリックします。

ポイント

プロジェクトを選択し、コンテキストメニューから[実行] > [サーバで実行]または[デバッグ] > [サーバでデバッグ]を選択することで、サーバの追加、プロジェクトの追加、サーバの起動、クライアントの起動をまとめて行うことができます。
また、クライアントとして使用するWebブラウザは、メニュー [ウィンドウ] > [Web ブラウザ] から変更することができます。

5-5) アプリケーションのデバッグ

アプリケーションが動作し、ブレークポイントで中断されます。変数ビューで変数の値を確認します。メニューから[実行] > [ステップオーバ]を選択し、変数ビューでプログラムの状況を確認します。デバッグについては、"6.2.7.1 デバッグする"を参照してください。

ポイント

変数ビューでは値の確認だけでなく、変更も行うことができます。

デバッガで中断している処理は、メニューから[実行] > [再開]を選択することで再開され、サーバ側での処理が終了し、Webブラウザに出力画面が表示されます。入力した順位の国名と総人口が表示されることを確認してください。

6) 運用環境へのアプリケーションの配布

このアプリケーションを運用環境に配布するには、WARファイルを作成する必要があります。作成したWARファイルは、サーバの配備機能を利用して、サーバに配備します。

6-1) アプリケーションのエクスポート

WARファイルの作成は、エクスポートウィザードから行います。エクスポートウィザードを起動するには、[ファイル] > [エクスポート]を選択してください。[エクスポート]ウィザードから[Web] ＞ [WAR ファイル]を選択してください。

WARエクスポートウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
Web モジュール	動的Webアプリケーションプロジェクトを指定します。
宛先	WARファイルの作成先を指定します。

6-2) 運用環境への配布

運用環境のInterstage管理コンソールにログインし、ローカルにあるWARファイルを指定することにより、リモート環境から運用環境に配備を行うことができます。

2.3 タスク

ここでは、Interstage Studioを用いてWebアプリケーションを開発する方法について、開発作業課題(タスク)ごとに説明します。

- 2.3.1 Webアプリケーションを作成する環境を準備する
- 2.3.2 サーブレットを作成する
- 2.3.3 HTMLファイルを作成する
- 2.3.4 JSPファイルを作成する
- 2.3.5 HTML/JSPファイルをグラフィカルに編集する
- 2.3.6 JavaScriptを作成する
- 2.3.7 CSSを作成する
- 2.3.8 web.xmlを編集する
- 2.3.9 Webアプリケーションの動作を確認する
- 2.3.10 JavaScriptの動作を確認する
- 2.3.11 Webアプリケーションを運用環境に配布する

2.3.1 Webアプリケーションを作成する環境を準備する

Webアプリケーションを作成するには、まずプロジェクトを作成し、必要なライブラリをプロジェクトに設定してビルドできる環境を整えます。

プロジェクトの作成

[新規プロジェクト]ウィザードから[Web] > [動的 Web プロジェクト]を選択し、Webアプリケーションプロジェクトを作成します。
プロジェクトウィザードの共通部分の詳細については、"6.2.2.1 プロジェクトを新規に作成する"を参照してください。

動的Webプロジェクトウィザードの固有の設定は、以下を参考にしてください。

- コンテキストルート
  Webサーバに配備されるときの最上位フォルダを指定します。

- コンテンツフォルダ
  Webアプリケーションとして配備するすべてのリソースを格納するフォルダを指定します。

- Java ソースフォルダ
  プロジェクトのソースを格納するフォルダを指定します。

- Deployment Descriptorの生成
  Webアプリケーションではweb.xmlが必要です。

クラスパスの設定

アプリケーションを作成するのに必要なライブラリなどがある場合には、クラスパスの設定を行う必要があります。クラスパスの設定は、プロジェクトのビルドパスで行います。
ビルドパスの設定の詳細については、"6.2.2.3 クラスパスを設定する"を参照してください。

注意

ビルドの際に削除されるため、WEB-INF/classesに直接クラスファイルを格納しないでください。

注意

WebアプリケーションでJSFを使用する場合には、以下の点に注意してください。

- 動的Webプロジェクトウィザードの[構成]に"JavaServer Faces v1.1 プロジェクト" または "JavaServer Faces v1.2 プロジェクト"を指定してください。
- 動的Webプロジェクトウィザードの[JSF機能]ページでは、以下の手順でライブラリを追加してください。
  1. [新規]をクリックします。
  2. ライブラリ名に"JSF"を入力し、[追加]をクリックして以下のjarファイルを選択して[完了]をクリックします。

     <製品インストールフォルダ>¥APS¥F3FMisjee¥lib¥jsf-impl.jar

2.3.2 サーブレットを作成する

サーブレットを作成するには、サーブレットウィザードでサーブレットクラスのソースファイルを作成し、そこに必要なメソッドを実装します。以下にその方法を説明します。

サーブレットの新規作成

サーブレットは、[新規]ウィザードから[Web] > [サーブレット]を選択し、ウィザードで作成します。ウィザードでの設定は、以下を参考にしてください。

- Web プロジェクト
  サーブレットクラスのソースを格納するプロジェクトを指定します。

- ソースフォルダ
  サーブレットクラスのソースを格納するフォルダを指定します。

- Java パッケージ
  サーブレットクラスのパッケージ名を指定します。

- クラス名
  サーブレットクラスのクラス名を指定します。

- スーパークラス
  継承するクラスを指定します。

- サーブレットの名前、説明、初期化パラメタ、URLマッピング
  サーブレットの定義情報を指定します。ここで指定した内容がWebアプリケーションのdeployment descriptor (web.xml) に挿入されます。

- 修飾子
  サーブレットクラスの修飾子を指定します。

- インタフェース
  実装するインタフェースを指定します。

- 作成するメソッドスタブの選択
  実装するメソッドやスタブを選択します。

ウィザードを実行すると、サーブレットクラスのJavaソースが生成され、web.xmlに必要な記述が追加されます。

サーブレットの編集

ソースを生成した後はdoPost()やdoGet()といったサーブレットのメソッドを実装します。これらのメソッドは、通常以下のような順に処理を実装します。

1. 要求オブジェクトからパラメタの値を取得する
2. 応答オブジェクトにヘッダを設定する
3. 応答オブジェクトから出力ストリームを取得して、応答オブジェクトに出力する

これらメソッドの実装の方法は、通常のJavaクラスに行う場合と変わりありません。Javaエディタを使ってメソッドの実装を行ってください。

2.3.3 HTMLファイルを作成する

HTMLファイルを作成するには、HTMLウィザードでHTMLファイルを作成し、ページのレイアウトはHTMLエディタ等を用いて行います。以下にその方法を説明します。

2.3.3.1 HTMLファイルの新規作成

HTMLファイルは、[新規]ウィザードから[Web] > [HTML]を選択し、ウィザードで作成します。ウィザードでの設定は、以下を参考にしてください。

- 親フォルダを入力または選択
  HTMLファイルを作成するフォルダを選択します。

- ファイル名
  作成するHTMLファイルの名前を指定します。拡張子を省略した場合、拡張子は ".html"になります。

- HTML テンプレートの使用
  テンプレートを使ってHTMLファイルを作成する場合にチェックします。新たにテンプレートを追加することもできます。

ポイント

拡張子を省略したときの拡張子、および、ファイル作成時に埋め込まれるエンコードは、設定ページの[Web] > [HTMLファイル]でカスタマイズすることができます。

2.3.3.2 HTMLファイルの編集

HTMLファイルの編集には、HTMLエディタを用います。HTMLエディタは以下の特徴を持つテキストエディタです。

- 構文の強調表示
  要素名、属性名、属性値、コメントなどをそれぞれ異なる色で強調します。強調の方法は、設定でカスタマイズすることができます。

- 問題の指摘
  記述したタグ、属性名、属性値などに誤りがある場合には、エラーや警告としてマーカや波線で指摘します。

- コンテンツアシスト
  カーソルの位置に応じて、選択可能な要素名、属性名、属性値などの候補がコンテンツアシストリストに表示されます。[Ctrl+Space]キーを押してコンテンツアシストリストを表示し、文字を入力することで候補を絞り込みます。

- ツールチップ表示
  マウスカーソルを要素名や属性名の上に移動し、[F2]キーを押すと、その要素や属性の説明が表示されます。

- ユーザ定義可能なテンプレートとスニペット
  コンテンツアシストで利用可能なテンプレートを登録することができます。
  また、良く利用するコードの断片などをスニペットに登録することで、ドラッグ＆ドロップで簡単にコードを挿入することができます。

- タグの選択
  カーソルの位置に応じて、そのタグが及ぶ範囲をページ左の垂直方向ルーラーにインジケータを表示します。

HTMLエディタで編集するときには、以下のビューを適宜利用します。

- アウトラインビュー
  HTML構文のアウトラインを表示します。要素の追加や属性の追加もこのビューで行うことができます。

- プロパティビュー
  エディタで選択されている要素の属性を表示します。値を変更することもできます。属性値を一覧から選択することも可能です。

- スニペットビュー
  良く利用するコードの断片を登録することで、ドラッグ＆ドロップでコードを簡単にソースに挿入することができます。

タグを入力する

以下のどちらかの方法でタグを入力します。

- コンテンツアシスト
  [Ctrl+Space]キーを押すと、現在のカーソル位置に応じてタグの候補がコンテンツアシストリストに表示されます。コンテンツアシストリストには、構文上挿入可能なタグのほかに、HTMLのテンプレートに登録されているテンプレートも表示されます。

- アウトラインビューを利用
  アウトラインのコンテキストメニューからタグを挿入することができます。アウトラインビューで現在選択されているタグの子、前、後のどれかの位置にタグを挿入できます。

表を挿入する

コンテンツアシストで表を挿入します。テンプレートに表が「table」としてあらかじめ登録されています。

リストを挿入する

コンテンツアシストでリストを挿入します。順序付きリストは、テンプレートの「ol」を使います。順序なしリストは、テンプレートの「ul」を使います。

ポイント

良く使うタグは、テンプレートやスニペットビューに登録しておくと便利です。キーボードによるコンテンツアシストを利用する場合は、テンプレートを利用します。マウス操作でタグを挿入する場合は、スニペットビューを利用します。また、スニペットビューでは、ユーザ定義の変数を使用することができ、挿入時に変数の値を指定することが可能です。
テンプレートの登録についての詳細は、Tipsの"テンプレート"を参照してください。スニペットの登録についての詳細は、Tipsの"スニペット"を参照してください。

属性/属性値を入力する

属性の追加や属性値の変更には、以下の方法があります。

- コンテンツアシストを利用する
  タグの中で、[Ctrl+Space]キーを押すと、そのタグで利用可能な属性の一覧がコンテンツアシストリストに表示されます。追加したい属性をリストより選択します。選択肢の決まっている属性値は、属性値の先頭にキャレットを移動してコンテンツアシストを起動する

と、属性値の候補がコンテンツアシストリストに表示されます。

- プロパティビューを利用する
  エディタでキャレットのある位置のタグの属性一覧が、プロパティビューに表示されます。プロパティビューの値カラムに値を入力すると、その値がタグの属性値に反映されます。

ページをプレビューする

作成したHTMLファイルをプレビューするには、そのファイルをWebブラウザで開きます。

ポイント

編集を行うHTMLエディタとプレビューを行うWebブラウザを縦に並べて表示すると、便利です。エディタを縦に並べて表示するには、エディタのタブをドラッグします。

2.3.3.3 HTMLタグの検証

HTML構文バリデータによって、HTMLタグの検証が行われます。検証の詳細については、"6.2.6.2 検証"を参照してください。

2.3.4 JSPファイルを作成する

JSPファイルを作成するには、JSPウィザードでJSPファイルを作成し、ページのレイアウトや動作などをJSPエディタ等のエディタを用いて行います。以下にその方法を説明します。

2.3.4.1 JSPファイルの新規作成

JSPファイルは、[新規]ウィザードから[Web] > [JSP]を選択し、ウィザードで作成します。ウィザードでの設定は、以下を参考にしてください。

- 親フォルダを入力または選択
  JSPファイルを作成するフォルダを選択します。

- ファイル名
  作成するJSPファイルの名前を指定します。拡張子を省略した場合、拡張子は ".jsp"になります。

- JSP テンプレートの使用
  テンプレートを使ってJSPファイルを作成する場合にチェックします。新たにテンプレートを追加することもできます。

ポイント

- インクルードディレクティブ(<%@ include file="..."%>)などで、他のJSPファイルに取り込まれる目的のファイルで、直接JSPとして呼び出されることの無いファイルは、拡張子をjspにはしません。こういったファイルは拡張子をjspfなどにします。
- 拡張子を省略したときの拡張子、および、ファイル作成時に埋め込まれるエンコードは、設定ページの[Web] > [JSPファイル]でカスタマイズすることができます。

2.3.4.2 JSPファイルの編集

JSPファイルの編集には、JSPエディタを用います。JSPエディタは以下の特徴を持つテキストエディタです。

- 構文の強調表示
  要素名、属性名、属性値、コメントなどをそれぞれ異なる色で強調します。強調の方法は、設定でカスタマイズすることができます。

- 問題の指摘
  記述したタグ、属性名、属性値などに誤りがある場合には、エラーや警告としてマーカや波線で指摘します。

- コンテンツアシスト
  カーソルの位置に応じて、選択可能な要素名、属性名、属性値などの候補がコンテンツアシストリストに表示されます。[Ctrl+Space]キーを押してコンテンツアシストリストを表示し、文字を入力することで候補を絞り込みます。

- ツールチップ表示
  マウスカーソルを要素名や属性名の上に移動し、[F2]キーを押すと、その要素や属性の説明が表示されます。

- ユーザ定義可能なテンプレートとスニペット
  コンテンツアシストで利用可能なテンプレートを登録することができます。また、良く利用するコードの断片などをスニペットに登録することで、ドラッグ＆ドロップで簡単にコードを挿入することができます。

- タグの選択
  カーソルの位置に応じて、そのタグが及ぶ範囲をページ左の垂直方向ルーラーにインジケータを表示します。

JSPエディタで編集するときには、以下のビューを適宜利用します。

- アウトラインビュー
  JSP構文のアウトラインを表示します。要素の追加や属性の追加もこのビューで行うことができます。

- プロパティビュー
  エディタで選択されている要素の属性を表示します。値を変更することもできます。属性値を一覧から選択することも可能です。

- スニペットビュー
  良く利用するコードの断片を登録することで、ドラッグ＆ドロップでコードを簡単にソースに挿入することができます。

JSPエディタの操作は、HTMLエディタと同様です。"2.3.3.2 HTMLファイルの編集"を参照してください。

JSP拡張タグを使う

Webアプリケーションを運用する際に必要となるJSP拡張タグの設定を行えば、コンテンツアシストリストにJSP拡張タグが表示されます。すなわち、以下の準備を行います。

1. JSPタグライブラリの場所を指定する
   以下のどれかの方法でJSPタグライブラリを指定します。
   - web.xmlにtaglibタグを記述し、.tldファイルの場所を指定する
   - /WEB-INF に.tldファイルを置く
   - .tldファイルが/META-INF に格納されているJARファイルを/WEB-INF/libに置く

2. JSPファイルにtaglibディレクティブを記述する
   JSP拡張タグをコンテンツアシストリストに表示するには、JSPファイルにtaglibディレクティブを記述します。
   taglibディレクティブを記述するには、コンテンツアシストでテンプレート「JSP taglib ディレクティブ」を選択します。uri属性の値を入力する際にもコンテンツアシストが利用できます。また、uri属性にtaglibのuriを指定してある場合は、prefix属性の値入力でもコンテンツアシストが利用でき、taglibに指定されているデフォルトのprefixがコンテンツアシストリストに表示されます。

注意

- JSPエディタで以下の条件のJSPファイルを開いて、メニューの[ソース] > [フォーマット]を実行した場合、タグ構成が崩れてエラーとなる場合があります。JavaScriptの処理を記述しているJSPファイルで、[フォーマット]を実行する場合は、Webページエディタを利用してください。
  - タグ内にJavaScriptの動作する属性を記述して、JavaScriptの処理を記述している（例：onclick属性 など）
- JSPファイルにJSP2.1で追加された属性（例：trimDirectiveWhitespaces属性 など）を指定した場合、属性が認識されず問題ビューに警告が表示されますが、動作上は問題ありません。

2.3.4.3 JSPの検証

JSPを検証するバリデータとして、JSP構文バリデータとJSP内容バリデータがあります。検証の詳細は、"6.2.6.2 検証"を参照してください。

注意

JSPタグの開始タグと終了タグの対応が取れているかどうかは、バリデータによって検証されません。このため、問題ビューにはエラーや警告は出力されません。しかし、開始タグと終了タグの対応が取れていないことにより、JSPファイルから変換されたJavaコードのコンパイルでエラーが検出されることがあります。

2.3.5 HTML/JSPファイルをグラフィカルに編集する

HTMLファイルやJSPファイルのデザイン・レイアウトを確認しながらグラフィカルに編集するには、Webページエディタを利用します。

ポイント

Webページエディタでファイルを編集するには、ファイルを選択し、コンテキストメニューから[アプリケーションから開く] > [Web ページエディタ]を選択します。

Webページエディタは以下の特徴を持つエディタです。

グラフィカル編集とテキスト編集

[設計]タブにはデザインページとソースページがあります。
デザインページでは、Webページの出力イメージを直接編集するような感覚で編集作業を行うことができます。
ソースページでは、ソースの内容が構文によって色分けされて表示され、コンテンツアシストなどにより、編集作業を行うことができます。
デザインページやソースページで編集した内容は、すぐにもう一方のページに反映されます。

プレビュー

[プレビュー]タブで、実行時に表示されるWebページのイメージを確認することができます。

Webページエディタで編集するときには、以下のビューを適宜利用します。

パレット ビュー

HTMLやJSPの標準タグやクラスパスに設定されているJSP拡張タグのアイテムが表示されます。パレット ビューからアイテムをドラッグ＆ドロップすることでデザインページに挿入することができます。

アウトラインビュー

HTML/JSP構文のアウトラインを表示します。要素の追加や属性の追加もこのビューで行うことができます。

プロパティビュー

エディタで選択されている要素の属性を表示します。値を変更することもできます。属性値を一覧から選択することも可能です。

ソースページでの操作は、HTMLエディタ/JSPエディタと同様です。"2.3.3.2 HTMLファイルの編集"、"2.3.4.2 JSPファイルの編集"を参照してください。

設計タブの配置を変更する

[設計]タブには、デザインページとソースページがありますが、ツールバーから以下の4つのパターンの表示に切り替えることができます。

- 上下分割表示
- 左右分割表示
- デザインページのみ表示
- ソースページのみ表示

また、デザインページの一部がパレットになっていますが、パレット ビューを表示することで、デザインビューを広く使用することもできます。

スタイルを変更する

デザインページでは、コンテキストメニューの[スタイル]のサブメニューを選択することで、スタイルを変更することができます。

2.3.6 JavaScriptを作成する

JavaScriptファイルを作成するには、JavaScriptウィザードでJavaScriptファイルを作成し、JavaScriptエディタ等を用いてファイルの編集を行います。以下にその方法を説明します。

JavaScriptファイルの新規作成

JavaScriptファイルは、[新規]ウィザードから[JavaScript] > [JavaScriptソースファイル]を選択し、ウィザードで作成します。ウィザードでの設定は、以下を参考にしてください。

- 親フォルダを入力または選択
  JavaScriptファイルを作成するフォルダを選択します。

- ファイル名
  作成するJavaScriptファイルの名前を指定します。拡張子を省略した場合、拡張子は ".js"になります。

JavaScriptファイルの編集

JavaScriptファイルを編集するには、JavaScriptエディタを用います。JavaScriptエディタは、以下の特徴を持つエディタです。

- 構文の強調表示
  キーワード、コメント、文字列リテラルなどをそれぞれ異なる色で強調します。強調の方法は、設定でカスタマイズすることができます。

- コンテンツアシスト
  カーソルの位置に応じて、オブジェクト、メソッド、関数、プロパティなどの候補がコンテンツアシストリストに表示されます。[Ctrl+Space]キーを押してコンテンツアシストリストを表示し、文字を入力することで候補を絞り込みます。

JavaScriptの検証

JavaScriptバリデータによって、JavaScriptファイルの記述内容について検証が行われます。JavaScriptファイルは、JavaScriptソースフォルダとして指定されているフォルダ内に格納される必要があります。

JavaScriptソースフォルダを指定するには、以下の手順で指定してください。

1. プロジェクトエクスプローラで対象となるプロジェクトを選択します。
2. コンテキストメニューの[プロパティ]を選択して、[プロパティ]ダイアログボックスを開きます。
3. [プロパティ]ダイアログボックスの左側のツリービューで[JavaScript] > [JavaScript ライブラリ]を選択します。
4. [JavaScript ライブラリ]ページの[ソース]タブを選択して、JavaScriptファイルの格納先となるフォルダを追加します。

ポイント

JavaScriptプロジェクトの場合は、デフォルトでJavaScriptソースフォルダとなります。

注意

JavaScriptの意味解析に関する検証は実施されません。そのため、[設定]ページやプロジェクトの[プロパティ]ページの[JavaScript] > [バリデータ] > [エラー/警告]ページでの設定が有効にならないものがあります。

2.3.7 CSSを作成する

CSSファイルを作成するには、CSSウィザードでCSSファイルを作成し、CSSエディタ等を用いてファイルの編集を行います。以下にその方法を説明します。

CSSファイルの新規作成

CSSファイルは、[新規]ウィザードから[Web] > [CSS]を選択し、ウィザードで作成します。ウィザードでの設定は、以下を参考にしてください。

- 親フォルダを入力または選択
  CSSファイルを作成するフォルダを選択します。

- ファイル名
  作成するCSSファイルの名前を指定します。拡張子を省略した場合、拡張子は ".css"になります。

- CSS テンプレートの使用
  テンプレートを使ってCSSファイルを作成する場合にチェックします。新たにテンプレートを追加することもできます。

CSSファイルの編集

CSSファイルを編集するには、CSSエディタを用います。CSSエディタは、以下の特徴を持つエディタです。

- 構文の強調表示
  セレクタ、コメント、プロパティ名、プロパティ値などをそれぞれ異なる色で強調します。強調の方法は、設定でカスタマイズすることができます。

- コンテンツアシスト
  カーソルの位置に応じて、セレクタ、プロパティ名、プロパティ値の候補がコンテンツアシストリストに表示されます。[Ctrl+Space]キーを押してコンテンツアシストリストを表示し、文字を入力することで候補を絞り込みます。

ポイント

CSSファイルのデザインを確認するには、そのCSSファイルを適用したHTMLファイルをWebブラウザで参照します。HTMLファイルにCSSファイルを適用するには、以下のようなLINKタグをHTMLファイルに記述します。
<LINK href="default.css" rel="stylesheet" type="text/css">
デザインを確認するためのWebブラウザとCSSエディタをエディタ領域で縦に並べておくと便利です。CSSファイルを変更したときは、保存を行ってからWebブラウザで「最新の情報に更新」を行います。

2.3.8 web.xmlを編集する

web.xmlを編集するには、XMLエディタを用います。XMLエディタには、[設計]タブと[ソース]タブがあります。
[ソース]タブではXMLファイルを直接編集でき、コンテンツアシストでタグの追加が行えます。
[設計]タブでは、XMLの構造がツリー形式で表示され、コンテキストメニューからタグの追加などの操作が行えます。
XMLエディタの詳細については、"XMLファイルを編集する"を参照してください。

ポイント

コンテンツアシストでは、タグを一個ずつ追加していく必要がありますが、コンテキストメニューから[子の追加]を選択してタグの追加を行った場合には、追加したタグの必須の子要素(タグ)も一緒に追加することができます。

2.3.9 Webアプリケーションの動作を確認する

Webアプリケーションの動作を確認するには、Webアプリケーションを実行環境に配備してから実行する必要があります。これらの操作はサーバビューから行います。
操作の詳細については、"6.2.7 アプリケーションの動作確認"を参照してください。

注意

WebアプリケーションをTCP/IPモニタでモニタリングした際に、Webアプリケーションの応答がなくなる場合があります。

以下のどちらかの方法で対処してください。

- Webアプリケーションのリクエスト処理を再度実行する。
- Webブラウザを再度起動してから、再度Webアプリケーションを実行する。

2.3.10 JavaScriptの動作を確認する

JavaScriptのデバッグ支援機能を使用すると、デバッグ用のプリント命令を記述せずにJavaScriptのグローバル変数の値を参照したり変更する事ができます。

注意

JavaScriptのデバッグ支援機能では、中断点を設定してJavaScriptの実行を中断したり、中断時の変数の値を参照あるいは更新したりすることはできません。このようなデバッグが必要な場合には、サードパーティ製のブラウザ組込み型デバッガを使用してください。Internet Explorer 8、9または10の場合にはInternet Explorer開発者ツールが利用できます。詳細はInternet Explorerのヘルプを参照してください。

デバッグを開始するまでの手順

JavaScriptのデバッグ支援機能を使ってデバッグを開始するまでの手順を以下に示します。

1. デバッグ支援用Webアプリケーションの配備
   以下に格納されているデバッグ支援用WebアプリケーションをIJServerクラスタに配備します。

   <Java統合開発環境のインストールフォルダ>¥etc¥jsdebug¥f5drjsdbg.war

2. IJServerクラスタの起動
   デバッグ対象アプリケーションとデバッグ支援用Webアプリケーションが配備されているIJServerクラスタを起動します。

3. Webブラウザからデバッグ支援用Webアプリケーション経由でデバッグ対象Webアプリケーションにアクセスする
   デバッグ支援用Webアプリケーションにアクセスする際にURLパラメタにデバッグ対象WebアプリケーションのURLを指定します。

   例) http://localhost/f5drjsdbg/?url=/(デバッグ対象のコンテキストルート)/index.html

上記を行うとWebブラウザにデバッグ対象のアプリケーション画面とデバッグ支援機能の画面が表示されます。

注意

- デバッグ支援機能とデバッグ対象のアプリケーションは、同一ホストに配置する必要があります。
- デバッグ支援機能の画面はポップアップウィンドウとして表示されます。ブラウザでポップアップウィンドウをブロックする設定にしている場合にはデバッグ支援機能の画面が表示されないことがあります。この場合には、デバッグ対象のアプリケーションを配備したサイトに対してポップアップを許可するようにブラウザを設定してください。

変数の値の参照

以下を参考に変数リストに変数を追加して、変数の値を参照してください。

- 変数名
  変数リストに追加する、あるいは、変数リストから削除する変数名を指定します。変数名をコンマあるいは改行で区切ることで、複数の変数名を指定することができます。
- 変数リスト
  変数名、変数の型および変数の値を以下の形式で表示します。

変数名 (変数の型): 変数の値

変数がオブジェクトの場合には、変数名の前に"+"のアイコンが表示されます。このアイコンをクリックすると、そのオブジェクトのプロパティ名、型、値の情報が階層表示されます。

- 最新の値を表示
  [変数リスト]の値の表示は、アプリケーションでその値が変わっても自動的には更新されません。表示を更新するには[最新の値を表示]を押してください。値が更新された箇所が赤く表示されます。

注意

- 表示可能な変数はグローバル変数のみです。関数の中で定義されたローカル変数を表示することはできません。
- 関数型の変数の値を参照することはできません。なおInternet Explorerをお使いの場合には、関数型の変数であっても変数の型は"object"と表示されます。
- JavaScriptの変数は、その変数が定義されているページでのみ有効です。他のページに表示が切り替わると、以前のページで定義されていた変数は無効になります。
- 変数の値の参照や変更が行えるのは、アプリケーション画面のウィンドウが開いている間のみです。ウィンドウを一旦閉じてしまうと、再度開きなおしても変数の値の参照や変更は行えません。

ポイント

アプリケーション画面がフレーム分割されている場合、[変数名]に"フレーム名.変数名"と指定することで、フレーム内のページで定義されている変数を表示することができます。

変数の値の変更

変数の値を変更するには、[変数リスト]に表示された変数の値をクリックします。すると値の表示部分が入力フィールドに変わります。そこに新しい値を入力して[Enter]キーを押すと、変数の値を変更できます。変更した箇所は緑色で表示されます。

値の入力方法は、入力したい値の種類によって以下のように異なります。

値の種類	入力方法
数値(number)	数値を入力します。
論理値(boolean)	trueあるいはfalseを入力します。
文字列(string)	文字列をダブルクォーテーション(")で囲んで入力します。
未定義値(undefined)	変数の値を未定義にするには、undefinedと入力します。
ヌル値(null)	変数の値をnullにするには、nullと入力します。
オブジェクト(object)	オブジェクト型の変数の値を変更することはできません。オブジェクトが数値型、論理値型あるいは文字列型のプロパティを持つ場合には、そのプロパティに対して値の変更を行うことができます。
関数(function)	関数型の変数の値を変更することはできません。

2.3.11 Webアプリケーションを運用環境に配布する

作成したWebアプリケーションを運用環境に配布するには、アーカイブファイルを作成し、管理コンソールから配備を行う必要があります。

アーカイブファイルの作成

アーカイブファイルの作成は、エクスポートウィザードから行います。
エクスポートウィザードでの操作の詳細については、"6.2.8 運用環境への配布"を参照してください。

管理コンソールでの配備

管理コンソールはリモート環境から操作できます。運用環境の管理コンソールにログインし、ローカルにある配備するファイルを指定することにより、リモート環境から運用環境に配備を行うことができます。
配備や管理コンソールの詳細については、Interstage Application ServerのJava EE実行環境の使用方法を別途ご確認ください。

第3章 Enterprise JavaBeans (EJB)を開発する

ここでは、Enterprise JavaBeans（以降、EJBと略します）の概要および、Java EEアプリケーション実行環境上で動作する、Enterprise BeanとEJBクライアントアプリケーションの作成方法について説明します。

3.1 概要

EJBおよびEJBの開発支援機能の概要について説明します。

3.1.1 EJBとは

EJBは、多階層(3階層)の分散オブジェクト指向に基づいたJavaのためのサーバコンポーネントモデルです。サーバのアプリケーションとして必要なコンポーネントのライフサイクル管理、トランザクション管理などの低レベルのインタフェースを隠蔽し、ビジネスロジックの処理を記述するだけで、より抽象度の高いサーバコンポーネントを作成するためのフレームワークです。

EJBでは、サーバのアプリケーションに必要なライフサイクルなどの各種の管理処理をコンテナ(container)と呼ばれるサーバコンポーネントの入れ物で実現します。コンテナが煩雑な処理を肩代わりします。また、EJBではコンテナ上で動作するサーバコンポーネントをEnterprise Beanと呼びます。

コンテナにEnterprise Beanをインストールし、実行可能な状態にすることを配備と呼びます。EJBでは配備という方式の導入により、ある特定のコンテナに依存しないポータビリティのあるサーバコンポーネントの作成を可能にしています。

EJBでは、以下の機能を規定しています。

- Enterprise Beanインスタンスのライフサイクル管理
- トランザクション管理
- セキュリティ管理
- セション管理
- リソース管理

Enterprise Beanの種類

Enterprise Beanには、大きく分けて、以下の種類があります。

- Session Bean(セションビーン)
  クライアントとの対話処理を行うEnterprise Beanです。主に業務処理で必要となる処理(ビジネスロジック)を記述します。

- Message-driven Bean(メッセージドリブンビーン)
  非同期通信処理を行うためのEnterprise Beanです。

以下に分類とそれぞれの用途について説明します。

EJB2.1までの仕様では、データベースシステムなどのデータを扱うためのEntity Bean(エンティティビーン)があります。しかしEJB 3.0からはEntity Beanの代わりにJava Persistence APIを使うようになっているため、本マニュアルではEntity Beanの説明は省略します。

Session Bean(セションビーン)

Session Bean(セションビーン)とは、アプリケーションの業務ロジック(ビジネスロジック)をサーバに配置したもので、複数のクライアントに対して、ネットワークを介してサービスを提供するものです。
Session Beanでは、他のEnterprise Beanを呼び出したり、トランザクションや処理の流れを制御したり、独自の処理を実装することで業務ロジックを実行します。

Session Beanには、以下の2種類があります。

- Stateful
  クライアントと1対1に対応し、クライアントから呼び出される複数のメソッドにまたがって、トランザクション状態やEnterprise Beanに定

義されている変数の値を保持することができます。そのため、ある機能を提供するうえで複数の手順(メソッド)が必要な場合に使用します。

- Stateless
  複数のメソッドにまたがって、トランザクション状態やEnterprise Beanに定義されている変数の値を保持することができません。そのため、1つのメソッドで完結する機能を提供する場合に使用します。また、複数のクライアントが、同じSession Beanのインスタンスを共有することができるため、サーバの負荷を軽減することができます。

例えば、単なる四則演算を行うような処理をSession Beanで実現するような場合にはStatelessにすべきです。保持しなければならない情報がないので、Statelessにしてサーバの負荷を減らします。一方、オンラインショッピングサイトのショッピングカートをSession Beanで実現する場合、これはStatefulにすべきです。ユーザが買い物かごにどの商品を入れたかを複数のページに渡って保持する必要があるからです。

Message-driven Bean(メッセージドリブンビーン)

Message-driven Beanとは、クライアントから送信されたメッセージに対して非同期に処理を行うための基盤を提供するものです。
Message-driven BeanからSession Beanなどを呼び出すことで、その機能を非同期に実行できます。

Message-driven Beanは、JMSメッセージまたはリソースアダプタのメッセージを受信して処理します。JMSメッセージの場合は、メッセージの処理の方法で以下の2種類に分類できます。

- Point-To-Pointモデル
  送信者から送信された1つのメッセージに対して、特定の受信者が処理を行います。すなわち、メッセージに対して受信者を単独で割り当てる場合に使用します。
  このモデルでは、送信されたメッセージはJMSサーバ上のキューに貯えられます。貯えられたメッセージは特定の受信者に割り当てられて処理されます。

- Publish/Subscribeモデル
  送信者から送信された1つのメッセージに対して、複数の受信者が処理を行います。すなわち、同一のメッセージを複数の受信者に配信する必要がある場合に使用します。
  このモデルでは、送信されたメッセージはJMSサーバ上のトピックに貯えられます。貯えられたメッセージはすべての受信者に割り当てられて各受信者によって処理されます。

3.1.1.1 J2EE1.4からの変更点

Java EEに含まれるEJB仕様はEJB 3.0です。EJB 3.0はJ2EE 1.4までのEJBが複雑であるという声を受けて、開発が簡単にできるように仕様が改善されています。

記述の簡素化

EJB 3.0の大きな特徴の1つは、EJBの記述が簡素化されたことです。

従来のEJB仕様ではEJBの各種定義をejb-jar.xmlというdeployment descriptorに記述する必要がありました。EJB 3.0では、EJBの各種定義はJavaのアノテーションを用いてEnterprise Beanクラスに記述するようになっています。これによりEJBのdeployment descriptorを作成する必要はほとんどの場合なくなりました。

また、EJBの各種定義も必要な場合だけ定義すれば良いように改善されています。これにより開発者が記述しなければならない定義の数も少なくなっています。例えば、Session BeanのクラスにJavaのインタフェースを1つ実装すると、それが自動的にEJBのLocalインタフェースとして扱われます。この場合、どのインタフェースがLocalインタフェースかという情報を開発者が記述する必要はありません。

加えて、Enterprise Beanの実装に最低限必要となるファイルの数も少なくなっています。従来のEJBとEJB3.0でそれぞれEnterprise Beanの開発に必要な要素は次表のとおりです。

表3.1 Enterprise Beanの実装に必須の要素

Enterprise Beanの種類	従来のEJB（～EJB 2.1）	EJB 3.0
Session Bean	・ Enterprise Beanクラス ・ Homeインタフェース ・ Componentインタフェース ・ deployment descriptor	・ Enterprise Beanクラス ・ ビジネスインタフェース

Enterprise Beanの種類	従来のEJB（～EJB 2.1）	EJB 3.0
Message Driven Bean	・ Enterprise Beanクラス ・ deployment descriptor	・ Enterprise Beanクラス

Java Persistence API(JPA)の導入

EJB 3.0のもう1つの大きな特徴はJPAが導入されたことです。JPAはEntity Beanに代わる永続化APIです。他のEnterprise Beanと同様に、JPAの記述もJavaのアノテーションを用いて行うことができます。また、JPAでは、データベースへのアクセスにEJB QLを拡張したクエリ言語が使用できます。

JPAの大きな特徴は、JPAが汎用的な永続化APIであることです。JPAはEJBからだけでなく、Webアプリケーションや Javaアプリケーションから使用することも可能です。

JPAおよびJPAを用いた開発の詳細については"4.1.2 JPAを使用したアプリケーションの開発"を参照してください。

EJB 3.0の主なアノテーション

以下に、EJB 3.0の主なアノテーションについて、簡単な例を示しながら紹介します。

javax.ejb.Stateless/javax.ejb.Statefulアノテーション

クラスをSession Beanとして定義するアノテーションです。以下のプロパティを指定できます。

プロパティ名	解説
name	EJB名を定義します。このプロパティを定義しない場合、Session Beanのクラス名がEJB名になります。
mappedName	アプリケーションサーバ製品が固有のBeanマッピング名を必要とする場合に、このプロパティを使用します。
description	このSession Beanの説明を定義します。

Statelessアノテーションを使用したSession Beanの作成例

```
package sample;

import javax.ejb.Stateless;

@Stateless
public class CalcBean implements Calc {
      public int add(int param1,int param2) {
            return param1 + param2;
      }
}
```

javax.ejb.MessageDrivenアノテーション

クラスをMessage-driven Beanとして定義するアノテーションです。以下のプロパティを指定できます。

プロパティ名	解説
name	EJB名を定義します。このプロパティを定義しない場合、Message-driven Beanのクラス名がEJB名になります。
messageListenerInterface	Beanのメッセージリスナインタフェースを定義します。このプロパティはBeanのクラスがメッセージリスナインタフェースを実装していない場合、または、複数のインタフェースを実装している場合にだけ定義する必要があります。
activationConfig	Message-driven Bean用のプロパティを定義します。
mappedName	アプリケーションサーバ製品が固有のBeanマッピング名を必要とする場合に、このプロパティを使用します。
description	このMessage-driven Beanの説明を定義します。

Message Drivenアノテーションを使用したMessage Driven Beanの作成例

```
package sample;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.MessageListener;
import javax.jms.Message;

@MessageDriven(mappedName = "jms/CalcBean" activationConfig = {
	@ActivationConfigProperty(
		propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge"),
	@ActivationConfigProperty(
		propertyName = "destinationType", propertyValue = "javax.jms.Queue")
})
public class CalcBean implements MessageListener {
	public void onMessage(Message msg) {
		...
	}
}
```

javax.ejb.EJBアノテーション

Enterprise Beanを呼び出す場合にDependency Injectionするためのアノテーションです。以下のプロパティを指定できます。

プロパティ名	解説
name	参照するBeanのJNDI名
beanInterface	参照するBeanのビジネスインタフェース。
beanName	参照するBeanのBean名。
mappedName	アプリケーションサーバ製品が固有のBeanマッピング名を必要とする場合に、このプロパティを使用します。
description	この呼出しの説明を定義します。

EJBアノテーションを使用したEJBクライアントの作成例

```
@EJB
private static Calc calc;

public int callCalc(int param1,int param2) {
	return calc.add(param1,param2);
}
```

3.1.2 EJBの開発

EJBの開発の概要を以下に示します。

EJBの開発の準備

EJBを開発するには、まずEJBのプロジェクトを作成する必要があります。プロジェクトを作成し、必要なターゲットランタイムやクラスパスなどのプロジェクト設定を行います。
詳細は、"3.3.1 EJBを作成する環境を準備する"を参照してください。

Enterprise Beanの作成

Enterprise Bean は、Session Beanウィザードまたは Message-driven Beanウィザードを用いて作成します。
詳細は、"3.3.2 Session Beanを作成する"または"3.3.3 Message-driven Beanを作成する"を参照してください。

データベースとの連携

Enterprise BeanからデータベースにアクセスするにはJava Persistence APIを使用します。
詳細は、"3.3.4 データベースを利用する"を参照してください。

EJBクライアントの作成

EJBを呼び出す方法には、@EJBアノテーションを使用します。
詳細は、"3.3.5 EJBクライアントを作成する"を参照してください。

EJBの動作確認

作成したEnterprise Beanアプリケーションの動作確認はサーバビューを用いて行います。
詳細は、"3.3.6 EJBの動作を確認する"を参照してください。

EJBの配布

EJBを配布するためにはEJBをアーカイブする必要があります。アーカイブファイルの作成にはエクスポートウィザードを使用します。
詳細は、"3.3.7 EJBを運用環境に配布する"を参照してください。

3.2 入門

ここでは、WebアプリケーションからEJBを使用するアプリケーションを作成する手順を紹介します。

3.2.1 作成するアプリケーション

国の総人口の順位付けリストから、順位を入力して国名と総人口を返すSession Beanを作成し、WebアプリケーションからそのSession Beanを呼び出し、その結果をWebページに表示するアプリケーションを作成します。Webアプリケーションの入門で作成したアプリケーションを題材に、ビジネスロジック部分をEJBに置き換えたものになります。

3.2.2 開発の流れ

ここでは、以下のようにEJBアプリケーションの開発を進めます。

1. EJBプロジェクトの作成
   Enterprise Beanを作成するために、まずEJBプロジェクトを作成します。ウィザードに従ってEJBプロジェクトを作成することで、ビルドに必要なクラスパスの設定などが自動的に行われます。

2. Session Beanの作成
   以下の手順でSession Beanを作成します。

   - データクラスの作成
     Session Beanの実装で利用するデータクラスを作成します。

   - Session Beanのひな型の作成
     ウィザードでSession Beanのひな型を作成します。1つのメソッドで完結するアプリケーションのためStateless Session Beanを作成します。

   - Session Beanの実装
     ウィザードで作成されたSession Beanのひな型にメソッドを宣言し、処理を実装します。

3. EARプロジェクトの作成
   EJBクライアントも含めて1つのアプリケーションとして配布するため、EARファイルを作成するためのプロジェクトを作成します。作成済みのEJBプロジェクトはEARプロジェクトの作成時にEARファイルに含めることを指定します。EJBクライアントは未作成のためEJBクライアントのプロジェクト作成時にEARファイルに含めることを指定します。

4. EJBクライアントの作成
以下の手順でEJBクライアントを作成します。

ー クライアントプロジェクトの作成
EJBクライアントとしてWebアプリケーションを作成するため、動的Webプロジェクトを作成します。

ー ビルドパスの設定
WebアプリケーションからEnterprise Beanを呼び出せるようにビルドパスを設定します。

ー サーブレットクラスの作成
Webアプリケーションのリクエストを受け付けるコントローラとしてのサーブレットをウィザードで作成します。

ー 入出力画面の作成
Webアプリケーションの入出力画面をウィザードで作成します。

ー Dependency Injectionの指定
サーブレットからSession Beanのメソッドを呼び出せるようにフィールドにDependency Injectionを指定します。

ー Session Beanの呼出し処理の実装
Dependency Injectionを宣言したフィールドを利用してSession Beanのメソッドを呼び出す処理を実装します。

5. アプリケーションの動作確認
以下の手順でアプリケーションの動作確認を行います。

ー プロジェクトとサーバの関連付け
アプリケーションをどのサーバに配備するかを設定します。アプリケーションはEARとしてまとめられているため、EARプロジェクトをサーバに追加します。

ー ブレークポイントの設定
実行時にデバッガでプログラムの動作を確認するため、ブレークポイントを設定します。

ー サーバの起動
Webブラウザからのアプリケーションに対するリクエストを受けつけられるように、サーバを起動します。サーバ起動前に配備は自動的に行われます。

ー アプリケーションの実行
Webブラウザを起動し、アプリケーションのURLにアクセスすることで動作確認を開始します。

ー アプリケーションのデバッグ
プログラムをデバッグし、アプリケーションが正常に動作することを確認します。

6. 運用環境へのアプリケーションの配布
以下の手順で運用環境への配布を行います。

ー アプリケーションのエクスポート
運用環境へアプリケーションを配布するため、EARファイルを作成します。

ー 運用環境への配布
Interstage管理コンソールからEARファイルを配備します。

3.2.3 開発手順

以下に、実際にアプリケーションを開発する手順を説明します。

1) EJBプロジェクトの作成

2) Session Beanの作成

3) EARプロジェクトの作成

4) EJBクライアントの作成

5) アプリケーションの動作確認

6) 運用環境へのアプリケーションの配布

1) EJBプロジェクトの作成

メニューバーから[ファイル] > [新規] > [プロジェクト]を選択すると、[新規プロジェクト]が表示されます。

[新規プロジェクト]ウィザードから[EJB] > [EJBプロジェクト]を選択して、[次へ]をクリックします。

以下の設定項目を確認、入力してください。

設定項目	設定内容
プロジェクト名	EJBSample
ターゲットランタイム	Interstage Application Server V11.1 IJServer Cluster (Java EE)
EJB モジュール バージョン	3.0
構成	Interstage Application Server V11.1 IJServer Cluster (Java EE) デフォルト構成
EARにプロジェクトを追加	チェックしない

情報を設定後、[次へ]をクリックしてください。EJBモジュールページが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
ソースフォルダ	ejbModule
Deployment Descriptorの生成	チェックしない

ポイント

EJB3.0では、deployment descriptorは必須ではなく、アノテーションのみでEnterprise Beanを作成することができます。

2) Session Beanの作成

2-1) データクラスの作成

国名と総人口の情報を保持し、そこから情報を取得するデータクラスを作成します。データクラスの作成は、作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [クラス]を選択します。[新規Javaクラス]ウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
ソースフォルダ	EJBSample/ejbModule
パッケージ	sample
名前	CountryData

以下のソースファイルが生成されます。

ソースファイル	説明
CountryData.java	データクラス

国名と総人口を保持する処理を実装します。以下の**赤字**の箇所をソースに追加してください。

データクラスの実装(CountryData.java)

```
package sample;

public class CountryData {

    private String countryName;
    private int totalPopulation;

    public CountryData(String name, int total) {
        setCountryName(name);
        setTotalPopulation(total);
    }

    public String getCountryName() {
        return countryName;
    }

    public void setCountryName(String countryName) {
        this.countryName = countryName;
    }

    public int getTotalPopulation() {
        return totalPopulation;
    }

    public void setTotalPopulation(int totalPopulation) {
        this.totalPopulation = totalPopulation;
    }

}
```

ポイント

フィールドを追加した後にクラスを選択している状態で、メニューから[ソース] > [Getter および Setter の生成]を選択することで、getter/setterの追加を行うことができます。

2-2) Session Beanのひな型の作成

Session Beanのひな型の作成は、作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [Session Bean]を選択すると[新規 EJB 3.0 Session Bean]ウィザードが表示されます。

以下の設定項目を確認、入力してください。

設定項目	設定内容
EJB プロジェクト	EJBSample
ソースフォルダ	¥EJBSample¥ejbModule
Java パッケージ	sample
クラス名	PopulationRanking
状態タイプ	Stateless
Remote	チェックしない

設定項目	設定内容
Local	チェックする sample.PopulationRankingLocal

情報を設定後、[次へ]をクリックしてください。Session Beanの情報設定ページが表示されます。

以下の設定項目を確認してください。情報を設定確認後、[完了]をクリックしてください。

設定項目	設定内容
Bean名	PopulationRanking
トランザクションタイプ	コンテナ
インタフェース	sample.PopulationRankingLocal
継承された抽象メソッド	チェックする
スーパークラスからのコンストラクタ	チェックする

[完了]をクリックすると、以下のソースファイルが生成されます。

ソースファイル	説明
PopulationRanking.java	Session Beanクラス
PopulationRankingLocal.java	ビジネスインタフェース

2-3) Session Beanの実装

順位を入力して国名と総人口を返す処理をSession Beanのメソッドとして実装します。以下の赤字の箇所をソースに追加してください。

ビジネスインタフェースの実装(PopulationRankingLocal.java)

```
package sample;

import javax.ejb.Local;

@Local
public interface PopulationRankingLocal {
    public CountryData getCountryData(int rank);
}
```

Session Beanクラスの実装(PopulationRanking.java)

```
package sample;

import javax.ejb.Stateless;

/**
 * Session Bean implementation class PopulationRanking
 */
@Stateless
public class PopulationRanking implements PopulationRankingLocal {

    /**
     * Default constructor.
     */
    public PopulationRanking() {
    }

    private CountryData countries[] = new CountryData[] {
       new CountryData("中国",1330000000),
```

```
        new CountryData("インド",1140000000),
        new CountryData("アメリカ",300000000),
        new CountryData("インドネシア",230000000),
        new CountryData("ブラジル",190000000),
        new CountryData("パキスタン",160000000),
        new CountryData("バングラデシュ",150000000),
        new CountryData("ロシア",140000000),
        new CountryData("ナイジェリア",140000000),
        new CountryData("日本",130000000)
    };

    public CountryData getCountryData(int rank) {
        --rank;
        if (rank < 0 || rank >= countries.length) {
            return null;
        }
        return countries[rank];
    }
}
```

3) EARプロジェクトの作成

メニューバーから[ファイル] > [新規] > [プロジェクト]を選択すると、[新規プロジェクト]が表示されます。

[新規プロジェクト]ウィザードから[Java EE] > [エンタープライズアプリケーションプロジェクト]を選択して、[次へ]をクリックします。

以下の設定項目を確認、入力してください。

設定項目	設定内容
プロジェクト名	EJBEARSample
ターゲットランタイム	Interstage Application Server V11.1 IJServer Cluster (Java EE)
EAR バージョン	5.0
構成	Interstage Application Server V11.1 IJServer Cluster (Java EE) デフォルト構成

情報を設定後、[次へ]をクリックしてください。EARに追加するモジュールページが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
Java EE モジュール依存関係	EJBSampleをチェックする
コンテンツフォルダ	EarContent
Deployment Descriptorの生成	チェックしない

4) EJBクライアントの作成

本来であればWebアプリケーションをクライアントとして作成するための手順が必要ですが、ここではサンプルをインポートしてEJBクライアントとして重要なポイントを確認していきます。Webアプリケーションの作成についての詳細な手順を知りたい場合は、Webアプリケーションの"2.2 入門"を参照してください。

サンプルのインポートは次の手順で行います。
メニューから[ファイル] > [インポート]を選択します。表示されたインポートウィザードで[一般] > [既存プロジェクトをワークスペースへ]を選択します。[アーカイブファイルの選択]をチェックし、[参照]ボタンを押して、以下のファイルを選択します。

<ワークベンチのインストールフォルダ>¥sample¥EJBSample.zip

[プロジェクト]から[EJBClientSample]を選択し、[完了]をクリックしてください。
EJBClientSampleプロジェクトのインポート後に、EJBEARSampleプロジェクトを選択して、右クリックでコンテキストメニューから[プロパティ]を選択します。プロパティダイアログボックスの[Java EEモジュール依存関係]を選択し、EJBClientSampleをチェックしてEARにEJBクライアントを追加します。

4-1) クライアントプロジェクトの作成

EJBクライアントとして動的Webプロジェクトを作成します。サンプルのインポートで代替済みのため、詳細は省略します。設定項目は以下を参考にしてください。

設定項目	設定内容
プロジェクト名	EJBClientSample
ターゲットランタイム	Interstage Application Server V11.1 IJServer Cluster (Java EE)
構成	Interstage Application Server V11.1 IJServer Cluster (Java EE) デフォルト構成
動的 Web モジュールバージョン	2.5
EARにプロジェクトを追加	チェックする
EARプロジェクト名	EJBEARSample

4-2) ビルドパスの設定

ビジネスインタフェース(PopulationRanking)を参照するためのビルドパス(クラスパス)を設定します。インポートしたサンプルプロジェクトでは設定済みです。
設定内容を確認するには、プロジェクト(EJBClientSample)を選択して、右クリックでコンテキストメニューから[プロパティ]を選択します。プロパティダイアログボックスの[Java EEモジュール依存関係]を選択して、[Java EEモジュール]タブを表示します。

以下がチェックされていることを確認します。

設定項目	設定内容
JAR/モジュール	EJBSample.jar

4-3) サーブレットクラスの作成

サーブレットクラスを作成します。インポートしたサンプルプロジェクトでは以下のような設定項目で作成済みです。

設定項目	設定内容
Java パッケージ	sample
クラス名	ServletController
スーパークラス	javax.servlet.http.HttpServlet
作成するメソッドスタブの選択	スーパークラスからのコンストラクタ 継承された抽象メソッド doPost(doGetのチェックを外す)

4-4) 入出力画面の作成

入出力画面のJSPを作成します。インポートしたサンプルプロジェクトでは以下のようなファイルを作成済みです。

入力画面(default.jsp)

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>世界総人口ランキング</title>
</head>
<body>
<h1>入力画面</h1>
<p>1～10の間で順位を入力してください。</p>
<p></p>
```

```
<form action="ServletController" method="post">
    順位：<br>
    <input name="rank" type="text" size="10">位<br>
    <p></p>
    <input type="submit" value="OK">
    <input type="reset" value="クリア">
</form>

</body>
</html>
```

出力画面(result.jsp)

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>世界総人口ランキング</title>
</head>
<body>
<h1>出力画面</h1>

総人口ランキング${param.rank}位は「${applicationScope.ranking.countryName}」です。<br>
総人口は${applicationScope.ranking.totalPopulation}人です。
<br>
<hr>
<form action="ServletController" method="post">
    <input type="submit" value="入り口に戻る">
    <input type="hidden" name="mode" value="top">
</form>

</body>
</html>
```

エラー画面(error.jsp)

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>世界総人口ランキング</title>
</head>
<body>

<h1>エラー画面</h1>
指定された範囲内の順位が入力されていないか、またはサーバエラーです。<br>
<br>
<hr>
<form action="ServletController" method="post">
    <input type="submit" value="入り口に戻る">
    <input type="hidden" name="mode" value="top">
</form>

</body>
</html>
```

4-5) Dependency Injectionの指定

Session Beanを呼び出すための実装をサーブレットクラスに行います。インポートしたサンプルプロジェクトのサーブレットソースには実装済みです。
ビジネスインタフェースのフィールドを作成して、以下のようにEJBアノテーションを設定し、Dependency Injectionの指定を行います。

EJBアノテーションの定義

```
@EJB
private PopulationRankingLocal business;
```

ポイント

J2EE1.4までのlookupメソッドの代わりに、Java EEではDependency Injectionを利用することで、オブジェクトを取得して使用することができます。(オブジェクトは実行時に自動的にJava EEコンテナによってフィールドに設定されます。)

4-6) Session Beanの呼出し処理の実装

Session Beanのメソッドを呼び出すには、EJBアノテーションを定義したビジネスインタフェース型のフィールドを利用してそのままメソッド呼出しを行うだけです。
インポートしたサンプルプロジェクトのサーブレットソースには実装済みです。

Session Beanの呼出しメソッドの定義

```
CountryData country = business.getCountryData(rank);
```

Session Bean呼出しの実装を行ったサーブレットクラスは以下のようになります。(インポートしたプロジェクトのサーブレットソースは、この形式になっています。)
赤字の部分が実際には追加、変更する箇所になります。

サーブレットクラス(ServletController.java)

```
package sample;

import java.io.IOException;

import javax.ejb.EJB;
import javax.servlet.RequestDispatcher;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class ServletController
 */
public class ServletController extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @EJB
    private PopulationRankingLocal business;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ServletController() {
        super();
```

```
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
        // TODO Auto-generated method stub
        request.setCharacterEncoding("Windows-31J");
        ServletContext sc = getServletContext();

        // 呼び出すファイル種別取得
        String mode = request.getParameter("mode");
        if (mode != null && mode.equals("top")) {
            // 入力画面のHTMLファイル呼出し
            RequestDispatcher inRd = getServletContext().getRequestDispatcher("/default.jsp");
            inRd.forward(request, response);
            return;
        }

        int rank;

        // 入力チェック
        try {
            rank = Integer.parseInt(request.getParameter("rank"));
        } catch (NumberFormatException e) {
            RequestDispatcher errRd = sc.getRequestDispatcher("/error.jsp");
            errRd.forward(request, response);
            return;
        }

        CountryData country = business.getCountryData(rank);

        //取得した値の確認
        if (country == null) {
            RequestDispatcher errRd = sc.getRequestDispatcher("/error.jsp");
            errRd.forward(request, response);
            return;
        }

        sc.setAttribute("ranking", country);

        RequestDispatcher outRd = sc.getRequestDispatcher("/result.jsp");
        outRd.forward(request, response);
    }
}
```

ポイント

必要な実装をコーディングした後にJavaエディタ上で右クリックし、コンテキストメニューから[ソース] > [インポートの編成]を行うと、プロジェクトに設定されたクラスパスに従った適切なimport文が挿入できます。

5) アプリケーションの動作確認

5-1) プロジェクトとサーバの関連付け

サーバビューでサーバを選択し、コンテキストメニューから[プロジェクトの追加および除去]を選択してください。
使用可能なプロジェクトから作成したEARプロジェクトを選択し、[追加]をクリックしてください。構成プロジェクトにEARプロジェクトが追加されていることを確認してください。

下の情報を確認後、[完了]をクリックしてください。

設定項目	設定内容
構成プロジェクト	EJBEARSample

ポイント

サーバビューに配備先となるサーバが登録されていない場合は、サーバを追加する必要があります。サーバを追加する方法については、"6.2.7 アプリケーションの動作確認"を参照してください。

5-2) ブレークポイントの設定

Session BeanクラスのgetCountryData()メソッドの先頭行にブレークポイントを設定します。ブレークポイントは、エディタの左側のルーラー部分をダブルクリックして、設定もしくは解除します。

5-3) サーバの起動

サーバビューでサーバを選択し、コンテキストメニューから[デバッグ]または[デバッグで再起動]または[接続(デバッグ起動)/ログイン]を選択してください。

ポイント

デバッガを使用せず単純にアプリケーションを起動したい場合には、コンテキストメニューから[開始]を選択します。

サーバが起動する前に自動的にEARファイルが配備されます。

サーバビューで以下の情報を確認してください。

確認項目	確認内容
サーバの状態	始動済み
サーバの状況	同期済み
EARファイル（EJBEARSample）の状況	同期済み
EARファイル配下のEJB-JAR（EJBSample）ファイルの状況	同期済み
EARファイル配下のWARファイル(EJBClientSample)の状況	同期済み

5-4) アプリケーションの実行

サーバビューで、配備されているWebプロジェクト（EJBClientSample）を選択して、コンテキストメニューから[Webブラウザ]を選択します。Webブラウザが起動されて、入力画面が開きます。

設定項目	設定内容
入力画面のURL	http://localhost/EJBClientSample/

[順位:]の入力フィールドに人口ランキングの順位を入力して、[OK]をクリックします。

ポイント

プロジェクトを選択し、コンテキストメニューから[実行] > [サーバで実行]または[デバッグ] > [サーバでデバッグ]を選択することで、サーバの追加、プロジェクトの追加、サーバの起動、クライアントの起動をまとめて行うことができます。
また、クライアントとして使用するWebブラウザは、メニュー [ウィンドウ] > [Web ブラウザ] から変更することができます。

5-5) アプリケーションのデバッグ

アプリケーションが動作し、ブレークポイントで中断されます。変数ビューの表示を確認します。メニューから[実行] > [ステップオーバ]を選択し、変数ビューでプログラムの状況を確認します。デバッグについては、"6.2.7.1 デバッグする"を参照してください。

ポイント

変数ビューでは値の確認だけでなく、変更も行うことができます。

デバッガで中断している処理は、メニューから[実行] > [再開]を選択することで再開され、サーバ側での処理が終了し、Webブラウザに出力画面が表示されます。入力した順位の国名と総人口が表示されることを確認してください。

6) 運用環境へのアプリケーションの配布

このアプリケーションを運用環境に配布するには、EARファイルを作成する必要があります。作成したEARファイルは、サーバの配備機能を利用して、サーバに配備します。

6-1) アプリケーションのエクスポート

EARファイルの作成は、エクスポートウィザードから行います。エクスポートウィザードを起動するには、[ファイル] > [エクスポート]を選択してください。[エクスポート]ウィザードから[Java EE] > [EARファイル]を選択してください。

EARエクスポートウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
EARアプリケーション	EJBEARSampleを指定します。
宛先	EARファイルの作成先を指定します。

6-2) 運用環境への配布

運用環境の管理コンソールにログインし、ローカルにあるEARファイルを指定することにより、リモート環境から運用環境に配備を行うことができます。

3.3 タスク

ここでは、Interstage Studioを用いてEJBアプリケーションを開発する方法について、開発作業課題(タスク)ごとに説明します。

- 3.3.1 EJBを作成する環境を準備する
- 3.3.2 Session Beanを作成する
- 3.3.3 Message-driven Beanを作成する
- 3.3.4 データベースを利用する
- 3.3.5 EJBクライアントを作成する
- 3.3.6 EJBの動作を確認する
- 3.3.7 EJBを運用環境に配布する

3.3.1 EJBを作成する環境を準備する

EJBを作成するには、まずプロジェクトを作成し、必要なライブラリをプロジェクトに設定してビルドできる環境を整えます。

プロジェクトの作成

[新規プロジェクト]ウィザードから[EJB] > [EJBプロジェクト]を選択し、EJBプロジェクトを作成します。
プロジェクトウィザードの共通部分の詳細については、"6.2.2.1 プロジェクトを新規に作成する"を参照してください。

EJBプロジェクトウィザードの固有の設定は、以下を参考にしてください。

- ソースフォルダ
  プロジェクトのソースを格納するフォルダを指定します。

- Deployment Descriptorの生成
  EJB3.0では、deployment descriptorは必須ではありません。
  アノテーションを利用したDependency Injectionだけでなく、JNDIのlookup処理を行う場合や、environment entryを使用する場合など、必要に応じて選択してください。

クラスパスの設定

アプリケーションを作成するのに必要なライブラリなどがある場合には、クラスパスの設定を行う必要があります。クラスパスの設定は、プロジェクトのビルドパスで行います。
ビルドパスの設定の詳細については、"6.2.2.3 クラスパスを設定する"を参照してください。

3.3.2 Session Beanを作成する

Session Beanを作成するには、EJB 3.0 Session BeanウィザードでSession Beanクラスのソースファイルを作成し、そこにビジネスメソッドを実装します。以下にその方法を説明します。

EJB 3.0 Session Beanウィザードによる Session Beanクラスの作成

Session Beanは、[新規]ウィザードから[EJB] > [Session Bean]を選択し、ウィザードで作成します。ウィザードでの設定は、以下を参考にしてください。

- EJB プロジェクト
  Session Beanを生成する、EJBプロジェクトを指定します。

- ソースフォルダ
  Session Beanのソースを格納するフォルダを指定します。

- Java パッケージ
  Session Beanクラスとビジネスインタフェースのパッケージ名を指定します。

- クラス名
  Session BeanのBeanクラス名を指定します。

- 状態タイプ
  作成したいSession Beanの種別に応じて、"Stateful "または"Stateless "を選択します。

- ビジネスインタフェースの作成
  生成するビジネスインタフェースを"Remote"もしくは"Local"を選択して、生成するインタフェース名を指定します。

- Bean名
  Session BeanのEJB名を指定します。

- 対応付け名
  Session Beanを対応付けるJNDI名を指定します。

- トランザクションタイプ
  トランザクションタイプを"コンテナ"または"Bean"を選択します。

- インタフェース
  Session Beanで使用するインタフェースを指定します。

- Home / Component インタフェース (EJB 2.x)
  EJB2.x形式のインタフェースを生成する場合は、"Local"もしくは"Remote"を選択して、生成するインタフェース名を指定します。

ウィザードを実行すると、Session Beanクラスとビジネスインタフェースのソースが生成されます。

ビジネスメソッドの実装

ソースを生成したあとはSession Beanにビジネスメソッドを実装します。ビジネスメソッドの実装手順は以下のとおりです。

1. ビジネスインタフェースにメソッドの宣言を記述する。
2. Session Beanのクラスにそのメソッドの実装を記述する。

ビジネスインタフェースに宣言したメソッドが自動的にSession Beanのビジネスメソッドとして扱われます。

これらメソッドの宣言および実装の方法は、通常のJavaインタフェース、Javaクラスに行う場合と変わりありません。Javaエディタを使ってメソッドの宣言および実装を行ってください。

3.3.3 Message-driven Beanを作成する

Message-driven Beanを作成するには、Enterprise BeanウィザードでMessage-driven Beanクラスのソースファイルを作成し、メッセージリスナメソッド（onMessage()）の実装を行います。以下にその方法を説明します。

EJB 3.0 Message-driven Beanウィザードによる Message-driven Beanクラスの作成

Message-driven Beanは、[新規]ウィザードから[EJB] > [Message-driven Bean]を選択し、ウィザードで作成します。ウィザードでの設定は、以下を参考にしてください。

- EJB プロジェクト
  Message-driven Beanを生成する、EJBプロジェクトを指定します。

- ソースフォルダ
  Message-driven Beanのソースを格納するフォルダを指定します。

- Java パッケージ
  Message-driven Beanクラスのパッケージ名を指定します。

- クラス名
  Message-driven BeanのBeanクラス名を指定します。

- Destination名
  JMSリソースのJNDI名を指定します。

- JMS
  JMS メッセージを利用する場合は、チェックします。チェックした場合は、"Destination タイプ"を設定します。

- Destinationタイプ
  "JMS"をチェックした場合に、Message-driven BeanのDestinationタイプを"キュー"または"トピック"から選択します。

- Bean 名
  Message-driven BeanのEJB名を指定します。

- トランザクションタイプ
  トランザクションタイプを"コンテナ"または"Bean"を選択します。

- インタフェース
  Message-driven Beanで使用するインタフェースを指定します。

ウィザードを実行すると、Message-driven BeanクラスのJavaソースが生成されます。

メッセージリスナメソッド(onMessage())の実装

ウィザードが生成したMessage-driven BeanクラスのJavaソースには、空のonMessage()メソッドが定義されています。受信したメッセージに対する処理をこの中に実装してください。

3.3.4 データベースを利用する

EJB 3.0ではデータベースを扱う仕組みとして、Entity Beanの代わりにJava Persistence API (JPA)が導入されました。
JPAを用いたデータベースの利用の方法については、"4.1.2 JPAを使用したアプリケーションの開発"を参照してください。

3.3.5 EJBクライアントを作成する

ここでは、Session Beanのビジネスメソッドを呼び出すクライアント、および、Message-driven Beanにメッセージを送信するクライアントの作成について説明します。

3.3.5.1 Session Beanを呼び出すクライアントを作成する

Session Beanを呼び出すクライアントの作成に特別な定義ファイルは必要ありません。クライアントとなるJavaクラス(サーブレットなど)に、Session Beanのビジネスインタフェースを取得する処理と、それを用いてビジネスメソッドを呼び出す処理を記述します。

ビジネスインタフェースの取得

Dependency Injectionによって簡単にビジネスインタフェースを取得することができます。
ビジネスインタフェースを取得する場合には、以下のように@EJBアノテーションを使用します。

EJBアノテーションの使用例

```
@EJB
private Calc calc;
```

記述例	説明
ejb/Calc	Session Beanが対応付けられているJNDI名を指定します。
Calc	ビジネスインタフェースです。(ローカルインタフェースです。)
Calc#add(int, int)	ビジネスメソッドです。

注意

Dependency Injectionは、サーブレットやEJBなどJava EEコンテナで管理されているJava EEコンポーネントでしか使用することができません。それ以外のクラスでは、JNDIのlookupによってオブジェクトを取得します。
詳細は、"10.2.2 JNDIのlookupによるオブジェクトの取得について"を参照してください。

ビジネスメソッドの呼び出し

ビジネスメソッドを呼び出すには、取得したビジネスインタフェースを利用してメソッドを呼び出します。

ビジネスメソッドの呼び出し例

```
int result = calc.add(100,200);
```

3.3.5.2 Message-driven Beanにメッセージを送信するクライアントを作成する

Message-driven Beanにメッセージを送信するクライアントの作成に特別な定義ファイルは必要ありません。クライアントとなるJavaクラス(サーブレットなど)に、Message-driven BeanのJMSコネクションファクトリなどを取得する処理と、それを用いてメッセージを送信する処理を記述します。

ここではPoint-To-PointモデルのMessage-driven Beanにメッセージを送信する場合について説明します。

JMSコネクションファクトリおよびJMS Destinationの取得

Dependency Injectionによって簡単にJMSコネクションファクトリやJMS Destinationを取得することができます。
JMSコネクションファクトリやJMS Destinationを取得する場合には、以下のように@Resourceアノテーションを使用します。

Resourceアノテーションの使用例

```
@Resource(name="jms/QueueCF001")
private QueueConnectionFactory qcf;
@Resource(name="jms/Queue")
private Queue queue;
```

記述例	説明
jms/QueueCF001	JMSコネクションファクトリ名
jms/Queue	JMS Destination名

注意

Dependency Injectionは、サーブレットやEJBなどJava EEコンテナで管理されているJava EEコンポーネントでしか使用することができません。それ以外のクラスでは、JNDIのlookupによってオブジェクトを取得します。
詳細は、"10.2.2 JNDIのlookupによるオブジェクトの取得について"を参照してください。

メッセージの送信

JMSコネクションファクトリおよびJMS Destinationを取得したあとのメッセージの送信手順はEJB 2.1までのものと同じです。以下にその例を示します。

メッセージ送信の例

```
// connection, session, senderの作成
QueueConnection connection = qcf.createQueueConnection();
QueueSession session = connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
QueueSender sender = session.createSender(queue);
// テキストメッセージの送信
TextMessage msg = session.createTestMessage();
msg.setText("Hello World!");
sender.send(msg);
// クローズ
connection.close();
```

3.3.6 EJBの動作を確認する

EJBの動作を確認するには、EJBを実行環境に配備してから実行する必要があります。これらの操作はサーバビューから行います。
操作の詳細については、"6.2.7 アプリケーションの動作確認"を参照してください。

3.3.7 EJBを運用環境に配布する

作成したEJBを運用環境に配布するには、アーカイブファイルを作成し、管理コンソールから配備を行う必要があります。

アーカイブファイルの作成

アーカイブファイルの作成は、エクスポートウィザードから行います。
エクスポートウィザードでの操作の詳細については、"6.2.8 運用環境への配布"を参照してください。

管理コンソールでの配備

管理コンソールはリモート環境から操作できます。運用環境の管理コンソールにログインし、ローカルにある配備するファイルを指定することにより、リモート環境から運用環境に配備を行うことができます。
配備や管理コンソールの詳細については、Interstage Application ServerのJava EE実行環境の使用方法を別途ご確認ください。

第4章 Java Persistence APIを使用したアプリケーションを開発する

ここでは、Java Persistence API(以降、JPAと略します)の概要とJPAを使用したアプリケーションの作成方法について説明します。

4.1 概要

JPAとJPA開発支援機能の概要について説明します。

4.1.1 JPAとは

Java Persistence API(JPA)は、EJB 2.1のEntity Beanの仕様をEJB3.0から独立した1つの仕様にまとめられものです。JPAは、リレーショナルデータベース(RDB)をJavaのアプリケーションから操作するために、オブジェクト/リレーション(O/R)マッピングの仕組みを提供します。O/Rマッピングは、Javaの永続クラスとRDBのテーブルをマッピングすることで、データの永続化を容易に行えるようにしたものです。JPAはデータベースをJavaのオブジェクトとして扱えるため、データベースにSQL文を発行する場面を少なくすることができます。

以下に、JPAを理解するうえで必要な用語について簡単に説明します。

- 永続ユニット
  persistence.xmlで定義される永続化するための論理的な集合です。複数の永続クラス(エンティティクラスなど)やO/Rマッピングファイルによって定義されます。

- Entity
  Entityは、データベースのテーブルと対応付けられるJavaクラスです。このインスタンスがデータベースの1レコードに相当します。

- 永続コンテキスト
  エンティティクラスのインスタンスの集合です。JPAでは永続コンテキストに対して操作を行い、トランザクションの区切りでデータベースと同期がとられます。

- Java Persistence Query Language(JPQL)
  データベースのSQLにあたる問い合わせ言語です。

- Entityマネージャ
  EntityやJPQLを使用して永続コンテキストを操作するためインタフェースです。

- persistence.xml
  エンティティクラス、O/Rマッピングファイル、トランザクションのタイプなどの構成情報を永続ユニットに定義できます。

- orm.xml
  O/Rマッピングファイルです。O/Rマッピングファイルはエンティティとデータベースとの対応付けを記述します。アノテーションでも記述することができるため必須ではありません。

4.1.1.1 EJB2.1からの変更点

EJB2.1のEntity BeanとJPAのEntityでは、以下の点が大きく異なります。

- Entityを構成するファイル
  EJB2.1のEntity Beanでは、規約に基づいてEntity Beanクラス、HomeインタフェースおよびComponentインタフェースを作成して、deployment descriptorにEntity Beanを定義する必要がありました。
  JPAのEntityでは、Javaオブジェクトとして操作するためのクラスを作成して、そのクラスに@Entityアノテーションなどでデータベースとのマッピング情報を定義するだけです。

- Entityの呼び出し方法
  Java EEではJPAのEntityに限らず、Dependency Injectionを使用して従来のlookup処理を簡単に記述することができます。

以下に、JPAで使用する主なアノテーションについて、簡単な例を紹介します。

javax.persistence.Entityアノテーション

クラスをEntityとして定義するアノテーションです。以下の要素を指定できます。

要素名	解説
name	Entityの名前。省略するとクラスの名前がEntityの名前になります。

Entityアノテーションの使用した例

```
package sample;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Employee {
          @Id
      private int     id;
      private String  name;
      private String  address;
      private String  tel;
}
```

javax.persistence.PersistenceContextアノテーション

EntityはEntityマネージャで管理されています。このEntityマネージャをDependency Injectionするためのアノテーションです。以下の要素を指定できます。

要素名	解説
name	コンテキストを参照する環境でEntityマネージャにアクセスする場合の名前です。Dependency Injectionの場合は使用しません。
properties	コンテナや永続プロバイダにプロパティを指定する場合に使用します。
type	トランザクションのタイプを指定します。
unitName	永続ユニットの名前。JNDIでアクセスできる名前を指定します。

PersistenceContextアノテーションを使用した例

```
@PersistenceContext
EntityManager em;

public Employee getEmployeeByPrimaryKey (int id) {
    return em.find(Employee.class, id);
}
```

4.1.2 JPAを使用したアプリケーションの開発

JPAを使用したアプリケーションを開発する概要を以下に示します。

JPAを使用したアプリケーションの開発の準備

JPAを使用したプログラムは、一般的なライブラリとして扱うことができます。また、JPAを使用したプログラムをWebアプリケーションやEJBアプリケーションの中に含めることもできます。WebアプリケーションやEJBアプリケーションの中に含める場合は、専用のプロジェクトを作成する必要はありません。ライブラリにする場合は、JPAプロジェクトを作成します。
詳細は、"4.3.1 JPAを使用したアプリケーションを作成する環境を準備する"を参照してください。

永続ユニットの開発

永続ユニットを開発するには、エンティティクラスの作成やXMLファイルの編集を行う必要があります。
JPAの仕様に沿って開発が行われているかを検証する機能も提供されています。
詳細は、"4.3.2 永続ユニットを開発する"を参照してください。

エンティティクラスの作成

エンティティクラスを作成するには、Javaクラスとデータベースのマッピングをするか、データベースのテーブルから作成します。
詳細は、"4.3.3 エンティティクラスを作成する"と"4.3.3.4 テーブルからエンティティクラスを生成する"を参照してください。

JPAでデータベースの操作

JPAでは、Entityマネージャでエンティティを操作することが、データベースを操作することになります。
詳細は、"4.3.4 JPAでデータベースを操作する"を参照してください。

JPAを使用したアプリケーションの動作確認

作成したアプリケーションの動作確認はサーバビューを用いて行います。
詳細は、"4.3.5 JPAを使用したアプリケーションの動作を確認する"を参照してください。

JPAを使用したアプリケーションの配布

JPAを使用したアプリケーションを配布するには、アプリケーションをアーカイブする必要があります。アプリケーションのアーカイブファイルを作成するため、エクスポートウィザードを提供しています。
詳細は、"4.3.6 JPAを使用したアプリケーションを運用環境に配布する"を参照してください。

4.2 入門

ここでは、JPAを使用したアプリケーションを作成するための手順を紹介します。

4.2.1 作成するアプリケーション

国名と総人口のデータを格納したデータベースから、データを取得するJPAのライブラリを作成します。そして、このライブラリを呼び出すWebアプリケーションも作成します。Webアプリケーションは、人口ランキングの順位を入力する画面と、入力した順位の国名と総人口を出力する画面を用意します。
データベースは"8.2 入門"で作成したものを使用します。また、Webアプリケーションは"2.2 入門"で作成したものを使用します。

4.2.2 開発の流れ

ここでは、以下のようにJPAを使用したアプリケーションの開発を進めます。

1. データベースの準備
   JPAアプリケーションのためのデータベースの環境を準備します。

2. 永続ユニットの作成

   - JPAプロジェクトの作成
     JPAアプリケーションを作成するために、まずJPAプロジェクトを作成します。ウィザードに従ってJPAプロジェクトを作成することで、ビルドに必要なクラスパスの設定などが自動的に行われます。

   - persistence.xmlの編集
     永続ユニットで使用するデータソースをpersistence.xmlに記述します。

3. Entityの作成
以下の手順でEntityを作成します。

- エンティティクラスの作成
ウィザードでエンティティクラスを作成します。

- Entityアノテーションの設定
JPA構造ビューとJPA詳細ビューを使用し、作成したクラスにEntityアノテーションを設定します。

- テーブルとの関連付け
JPA構造ビューとJPA詳細ビューを使用し、Entityとデータベーステーブルの関連付けを行います。

- フィールドの追加
Entityに永続項目となるフィールドを追加します。

- カラムとの関連付け
JPA構造ビューとJPA詳細ビューを使用し、永続フィールドとデータベースの関連付けを行います。

- getメソッドの追加
永続フィールドにアクセスするためのgetメソッドを追加します。

4. ロジッククラスの作成

- クラスの作成
ウィザードでJavaクラスを作成します。

- ロジッククラスの実装
ロジッククラスにEntityを使用したデータベースアクセス処理の実装を行います。

5. EARプロジェクトの作成
クライアントも含めて1つのアプリケーションとして配布するため、EARファイルを作成するためのプロジェクトを作成します。作成済みのJPAプロジェクトはEARプロジェクトの作成時にEARファイルに含めることを指定します。クライアントは未作成のためクライアントのプロジェクト作成時にEARファイルに含めることを指定します。

6. Webアプリケーションの作成
以下の手順でクライアントとなるWebアプリケーションを作成します。

- 動的Webプロジェクトの作成
クライアントとしてWebアプリケーションを作成するため、動的Webプロジェクトを作成します。

- ビルドパスの設定
WebアプリケーションからJPAプロジェクトのロジッククラスを呼び出せるようにビルドパスを設定します。

- サーブレットクラスの作成
Webアプリケーションのリクエストを受け付けるコントローラとしてのサーブレットをウィザードで作成します。

- Dependency Injectionの指定
ロジッククラスではEntityを使用したデータベースアクセス処理を行うためEntityManagerFactoryが必要です。そのオブジェクトを取得するためにフィールドにDependency Injectionを指定します。

ー サーブレットクラスの実装
ロジッククラスの呼び出しを含めたサーブレットの実装処理を行います。

ー 入出力画面の作成
Webアプリケーションの入出力画面をウィザードで作成します。

7. Interstage Application ServerのJDBC設定
永続ユニットで指定したデータソースをJava EEコンテナで使用可能にするために以下の手順で環境を設定します。

ー Interstage管理コンソールの起動
サーバビューからInterstage管理コンソールを起動します。

ー JDBC Connection Poolの作成
Interstage管理コンソールでJDBC Connection Poolを作成します。

ー JDBC Resourceの作成
Interstage管理コンソールでJDBC Resourceを作成します。

ー JDBCドライバのクラスパスの設定
実行時にJava EEコンテナからJDBCドライバが参照できるようにクラスパスの設定を行います。

8. アプリケーションの動作確認
以下の手順でアプリケーションの動作確認を行います。

ー プロジェクトとサーバの関連付け
アプリケーションをどのサーバに配備するかを設定します。アプリケーションはEARとしてまとめられているため、EARプロジェクトをサーバに追加します。

ー ブレークポイントの設定
実行時にデバッガでプログラムの動作を確認するため、ブレークポイントを設定します。

ー サーバの起動
Webブラウザからのアプリケーションに対するリクエストを受けつけられるように、サーバを起動します。サーバ起動前に配備は自動的に行われます。

ー アプリケーションの実行
Webブラウザを起動し、アプリケーションのURLにアクセスすることで動作確認を開始します。

ー アプリケーションのデバッグ
プログラムをデバッグし、アプリケーションが正常に動作することを確認します。

9. 運用環境へのアプリケーションの配布
以下の手順で運用環境への配布を行います。

ー アプリケーションのエクスポート
運用環境へアプリケーションを配布するため、EARファイルを作成します。

ー 運用環境への配布
Interstage管理コンソールからEARファイルを配備します。

4.2.3 開発手順

以下に、実際にアプリケーションを開発する手順を説明します。

1) データベースの準備

2) 永続ユニットの作成

3) Entityの作成

4) ロジッククラスの作成

5) EARプロジェクトの作成

6) Webアプリケーションの作成

7) Interstage Application ServerのJDBC設定

8) アプリケーションの動作確認

9) 運用環境へのアプリケーションの配布

1) データベースの準備

データベースはSymfoware Serverを使用します。国名と総人口のデータを格納するデータベースのテーブルを作成し、アプリケーションの実行に必要なデータを挿入します。また、ワークベンチからデータベースに接続できる環境を整える必要があります。
これらについては、"8.2 入門"で説明していますので、そちらを参照してください。

2) 永続ユニットの作成

2-1) JPAプロジェクトの作成

メニューバーから[ファイル] > [新規] > [プロジェクト]を選択すると、[新規プロジェクト]ウィザードが表示されます。

[新規プロジェクト]ウィザードから[JPA] > [JPA プロジェクト]を選択して、[次へ]をクリックします。

以下の設定項目を確認、入力してください。

設定項目	設定内容
プロジェクト名	JPASample
ターゲットランタイム	Interstage Application Server V11.1 IJServer Cluster (Java EE)
構成	Interstage Application Server V11.1 IJServer Cluster (Java EE) デフォルト構成

情報を設定後、[次へ]をクリックしてください。[JPA ファセット]ページが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
プラットフォーム	汎用
接続	Symfoware_Studio
接続からデフォルトスキーマを上書きする	チェックしない
サーバランタイムが提供する実装を使用	チェックする
注釈されたクラスを自動的に検出する	チェックする
orm.xml の作成	チェックしない

ポイント

[接続]には、データベースに接続している接続プロファイルを指定することで、データベースのテーブルやカラムの情報をJPA詳細ビューなどで利用することができます。

2-2) persistence.xmlファイルの編集

Entityに関連付けるデータベースのJDBC Resourceをpersistence.xmlに記述します。
作成したプロジェクト(JPASample)を選択して、[src] > [META-INF] > [persistence.xml]をダブルクリックして永続化 XMLエディタで開きます。
永続化 XMLエディタの[接続]タブを選択して、JDBC Resourceの定義を以下のように設定します。

設定項目	設定内容
トランザクションタイプ	JTA
JTA データソース名	jdbc/Symfo

3) Entityの作成

データベースのテーブルとマッピングするEntityを作成します。

ポイント

以下では、JPA構造ビューとJPA詳細ビューの使用方法を説明するため、Javaクラスを作成して、それにデータベースを対応付ける方法を説明していますが、データベースのテーブルからEntityを生成することもできます。詳細は、"4.3.3.4 テーブルからエンティティクラスを生成する"を参照してください。

3-1)エンティティの作成

作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [その他]を選択すると、[新規]ウィザードが表示されます。

[新規]ウィザードから[JPA] > [エンティティ]を選択して、[次へ]をクリックします。

以下の設定項目を確認、入力してください。

設定項目	設定内容
プロジェクト	JPASample
ソースフォルダ	¥JPASample¥src
Java パッケージ	sample
クラス名	CountryData
継承	エンティティ
XMLのエンティティマッピングに追加	チェックしない

情報を設定後、[次へ]をクリックしてください。[エンティティプロパティ]ページが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
エンティティ名	CountryData
デフォルトを使用	チェックしない
テーブル名	C_DATA
エンティティフィールド	以下のフィールドを追加します
アクセスタイプ	フィールドベース

キー	名前	型
チェックする	id	int
チェックしない	countryName	java.lang.String

キー	名前	型
チェックしない	totalPopulation	int

3-2) テーブルとの関連付け

EntityのCountryDataクラスとデータベースのC_DATAテーブルを関連付けます。[JPA構造]ビューでCountryDataクラスが選択されていることを確認します。[JPA詳細]ビューで[テーブル]グループの[名前]に"C_DATA"が設定されていることを確認します。

ポイント

データベースに接続していない場合にはコンボボックスに候補は表示されませんが、テーブル名を直接指定することはできます。

以下の設定項目を確認、選択してください。

設定項目	設定内容
名前	デフォルト (CountryData)
テーブル:名前	C_DATA
テーブル:カタログ	デフォルト ()
テーブル:スキーマ	STUDIO

3-3) カラムとの関連付け

エンティティフィールドとテーブルのカラムを関連付けます。

各フィールドは、以下の表のようにカラムと対応付けます。

フィールド	対応付け	カラム:名前
id	Id	ID
countryName	基本(デフォルト)	C_NAME
totalPopulation	基本(デフォルト)	T_POP

これまでの手順で以下のようなソースが作成されます。

CountryDataクラス

```
package sample;

import java.io.Serializable;
import java.lang.String;
import javax.persistence.*;

/**
 * Entity implementation class for Entity: CountryData
 *
 */
@Entity
@Table(name="C_DATA", schema = "STUDIO")
public class CountryData implements Serializable {

    @Id
    @Column(name="ID")
    private int id;
    @Column(name="C_NAME")
    private String countryName;
    @Column(name="T_POP")
    private int totalPopulation;
```

```
    private static final long serialVersionUID = 1L;

    public CountryData() {
        super();
    }
    public int getId() {
        return this.id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getCountryName() {
        return this.countryName;
    }
    public void setCountryName(String countryName) {
        this.countryName = countryName;
    }
    public int getTotalPopulation() {
        return this.totalPopulation;
    }
    public void setTotalPopulation(int totalPopulation) {
        this.totalPopulation = totalPopulation;
    }

}
```

4) ロジッククラスの作成

データベースへの問合せなどを行うPopulationRankingクラスを作成します。

4-1) クラスの作成

作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [クラス]を選択します。[新規Javaクラス]ウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
ソースフォルダ	JPASample/src
パッケージ	sample
名前	PopulationRanking

4-2) ロジッククラスの実装

Entityは、javax.persistence.EntityManagerクラスにより管理されています。このクラスを作成するjavax.persistence.EntityManagerFactoryクラスを引数にしたコンストラクタを追加します。PopulationRankingクラスはWebアプリケーションのサーブレットクラスで使用されるので、この引数はサーブレットクラスから渡されます。以下のコードをPopulationRankingクラスに追加します。

コンストラクタの追加

```
package sample;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;

public class PopulationRanking {
    private EntityManager em;

    public PopulationRanking(EntityManagerFactory factory) {
        this.em = factory.createEntityManager();
```

```
    }
}
```

ポイント

必要な実装をコーディングした後にJavaエディタ上で右クリックし、コンテキストメニューから[ソース] > [インポートの編成]を行うと、プロジェクトに設定されたクラスパスに従った適切なimport文が挿入できます。

C_DATAテーブルからIDに対する総人口ランキングを取得するメソッドを追加します。C_DATAテーブルとマッピングしているCountryDataクラスを取得するには、EntityManagerインスタンスのfindメソッドを使用します。テーブルのデータが総人口の多い順番に格納されているので、順位とIDは同じ値になります。

以下のコードをPopulationRankingクラスに追加します。

総人口ランキングを取得するメソッドの追加

```
public CountryData getCountryData(int rank) {
    if (rank < 1 || rank > 10) {
        return null;
    }
    CountryData country = em.find(CountryData.class, rank);
    return country;
}
```

5) EARプロジェクトの作成

メニューバーから[ファイル] > [新規] > [プロジェクト]を選択すると、[新規プロジェクト]が表示されます。

[新規プロジェクト]ウィザードから[Java EE] > [エンタープライズアプリケーションプロジェクト]を選択して、[次へ]をクリックします。

以下の設定項目を確認、入力してください。

設定項目	設定内容
プロジェクト名	JPAEARSample
ターゲットランタイム	Interstage Application Server V11.1 IJServer Cluster (Java EE)
EAR バージョン	5.0
構成	Interstage Application Server V11.1 IJServer Cluster (Java EE) デフォルト構成

情報を設定後、[次へ]をクリックしてください。[エンタープライズアプリケーション]ページが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
Java EE モジュール依存関係	JPASample
コンテンツフォルダ	EarContent
Deployment Descriptorの生成	チェックしない

6) Webアプリケーションの作成

本来であればWebアプリケーションをクライアントとして作成するための手順が必要ですが、ここではサンプルをインポートしてJPAを使用する場合に重要なポイントを確認していきます。Webアプリケーションの作成についての詳細な手順を知りたい場合は、Webアプリケーションの"2.2 入門"を参照してください。

サンプルのインポートは次の手順で行います。
メニューから[ファイル] > [インポート]を選択します。表示されたインポートウィザードで[一般] > [既存プロジェクトをワークスペースへ]を選択します。[アーカイブファイルの選択]をチェックし、[参照]ボタンを押して、以下のファイルを選択します。

<ワークベンチのインストールフォルダ>¥sample¥JPASample.zip

[プロジェクト]から[JPAClientSample]を選択し、[完了]をクリックしてください。
JPAClientSampleプロジェクトのインポート後に、JPAEARSampleプロジェクトを選択して、右クリックでコンテキストメニューから[プロパティ]を選択します。プロパティダイアログボックスの[Java EEモジュール依存関係]を選択し、JPAClientSampleをチェックしてEARにクライアントを追加します。

6-1) 動的Webプロジェクトの作成

クライアントとして動的Webプロジェクトを作成します。サンプルのインポートで代替済みのため、詳細は省略します。設定項目は以下を参考にしてください。

設定項目	設定内容
プロジェクト名	JPAClientSample
ターゲットランタイム	Interstage Application Server V11.1 IJServer Cluster (Java EE)
構成	Interstage Application Server V11.1 IJServer Cluster (Java EE) デフォルト構成
EARにプロジェクトを追加	チェックする
EARプロジェクト名	JPAEARSample

6-2) ビルドパスの設定

エンティティクラス（CountryData）やロジッククラス（PopulationRanking）を参照するためのビルドパス（クラスパス）を設定します。インポートしたサンプルプロジェクトでは設定済みです。
設定内容を確認するには、作成したプロジェクト（JPAClientSample）を選択して、右クリックでコンテキストメニューから[プロパティ]を選択します。プロパティダイアログボックスの[Java EEモジュール依存関係]を選択して、[Java EEモジュール]タブを表示します。

以下がチェックされていることを確認します。

設定項目	設定内容
JAR/モジュール	JPASample.jar

6-3) サーブレットクラスの作成

サーブレットクラスを作成します。インポートしたサンプルプロジェクトでは以下のような設定項目で作成済みです。

設定項目	設定内容
Java パッケージ	sample
クラス名	ServletController
スーパークラス	javax.servlet.http.HttpServlet
作成するメソッドスタブの選択	スーパークラスからのコンストラクタ 継承された抽象メソッド doPost（doGetのチェックを外す）

6-4) Dependency Injectionの指定

ロジッククラスを呼び出すための実装をサーブレットクラスに行います。インポートしたサンプルプロジェクトのサーブレットソースには実装済みです。
PopulationRankingクラスを作成するためには、コンストラクタにjavax.persistence.EntityManagerFactoryが必要になります。EntityManagerFactoryのフィールドを作成して、以下のようにPersistenceUnitアノテーションを設定し、Dependency Injectionの指定を行います。

PersistenceUnitアノテーションの定義

```
@PersistenceUnit
private EntityManagerFactory emf;
```

ポイント

J2EE1.4までのlookupメソッドの代わりに、Java EEではDependency Injectionを利用することで、オブジェクトを取得して使用することができます。(オブジェクトは実行時に自動的にJava EEコンテナによってフィールドに設定されます。)

6-5) サーブレットクラスの実装

以下のようにロジッククラスのインスタンスを作成し、メソッドを呼び出します。インポートしたサンプルプロジェクトのサーブレットソースには実装済みです。

Entityの呼び出しメソッドの定義

```
PopulationRanking ranking = new PopulationRanking(emf);
CountryData country = ranking.getCountryData(rank);
```

実装を行ったサーブレットクラスは以下のようになります。(インポートしたプロジェクトのサーブレットソースは、この形式になっています。)
赤字の部分が追加した定義になります。

サーブレットクラス(ServletController.java)

```
package sample;

import java.io.IOException;

import javax.persistence.EntityManagerFactory;
import javax.persistence.PersistenceUnit;
import javax.servlet.RequestDispatcher;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


/**
 * Servlet implementation class ServletController
 */
public class ServletController extends javax.servlet.http.HttpServlet {
    private static final long serialVersionUID = 1L;

    //@PersistenceUnit
    @PersistenceUnit
    private EntityManagerFactory emf;

   /**
    * @see HttpServlet#HttpServlet()
    */
    public ServletController() {
        super();
        // TODO Auto-generated constructor stub
    }

   /**
    * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
    */
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
        // TODO Auto-generated method stub
        request.setCharacterEncoding("Windows-31J");
```

```
        ServletContext sc = getServletContext();

        // 呼び出すファイル種別取得
        String mode = request.getParameter("mode");
        if (mode != null && mode.equals("top")) {
            // 入力画面のHTMLファイル呼出し
            RequestDispatcher inRd = getServletContext().getRequestDispatcher("/default.jsp");
            inRd.forward(request, response);
            return;
        }

        int rank;

        // 入力チェック
        try{
            rank = Integer.parseInt(request.getParameter("rank"));
        } catch (NumberFormatException e){
            RequestDispatcher errRd = sc.getRequestDispatcher("/error.jsp");
            errRd.forward(request, response);
            return;
        }

        PopulationRanking ranking = new PopulationRanking(emf);
        CountryData country = ranking.getCountryData(rank);

        //取得した値の確認
        if(country == null){
            RequestDispatcher errRd = sc.getRequestDispatcher("/error.jsp");
            errRd.forward(request, response);
            return;
        }

        sc.setAttribute("ranking", country);

        RequestDispatcher outRd = sc.getRequestDispatcher("/result.jsp");
        outRd.forward(request, response);
    }
}
```

ポイント

必要な実装をコーディングした後にJavaエディタ上で右クリックし、コンテキストメニューから[ソース] > [インポートの編成]を行うと、プロジェクトに設定されたクラスパスに従った適切なimport文が挿入できます。

6-6) 入出力画面の作成

入出力画面のJSPを作成します。インポートしたサンプルプロジェクトでは以下のようなファイルを作成済みです。

入力画面(default.jsp)

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>世界総人口ランキング</title>
</head>
<body>
<h1>入力画面</h1>
```

```
<p>1～10の間で順位を入力してください。</p>
<p></p>
<form action="ServletController" method="post">
    順位：<br>
    <input name="rank" type="text" size="10">位<br>
    <p></p>
    <input type="submit" value="OK">
    <input type="reset" value="クリア">
</form>

</body>
</html>
```

出力画面(result.jsp)

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>世界総人口ランキング</title>
</head>
<body>
<h1>出力画面</h1>

総人口ランキング${param.rank}位は「${applicationScope.ranking.countryName}」です。<br>
総人口は${applicationScope.ranking.totalPopulation}人です。
<br>
<hr>
<form action="ServletController" method="post">
    <input type="submit" value="入り口に戻る">
    <input type="hidden" name="mode" value="top">
</form>

</body>
</html>
```

エラー画面(error.jsp)

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>世界総人口ランキング</title>
</head>
<body>

<h1>エラー画面</h1>
指定された範囲内の順位が入力されていないか、またはサーバエラーです。<br>
<br>
<hr>
<form action="ServletController" method="post">
    <input type="submit" value="入り口に戻る">
    <input type="hidden" name="mode" value="top">
</form>

</body>
</html>
```

7) Interstage Application ServerのJDBC設定

Java EEコンテナからSymfowareデータベースを使用できるように、Interstage Application Serverの管理コンソールでJDBCの設定をします。

Interstage管理コンソールの起動の仕方および使用方法については、Interstage Application ServerのJava EE実行環境の使用方法を別途ご確認ください。

7-1) JDBC Console Poolの作成

SymfowareのJDBC Connection Poolを作成します。左のペインから[リソース] > [JDBC] > [接続プール]を選択すると、[接続プール]ページが表示されます。

[新規...]をクリックして、以下の設定項目を入力してください。

設定項目	設定内容
名前	SymfoPool
リソースタイプ	javax.sql.ConnectionPoolDataSource
データベースベンダー	空白

情報を設定後、[次へ]をクリックしてください。

次のページでは、以下の設定項目を入力してください。

設定項目	設定内容
データソースクラス名	com.fujitsu.symfoware.jdbc2.SYMConnectionPoolDataSource

最後の行にある[追加プロパティ]で、プロパティを追加します。プロパティの追加は[プロパティを追加]をクリックします。以下の設定項目の入力が完了したら、[完了]をクリックします。

設定項目	設定内容
User	データベースにアクセスするためのユーザIDを指定します
Password	ユーザ名に対するパスワードを指定します
PortNumber	2050
networkProtocol	symford
DatabaseName	COUNTRYDATA
serverName	myhost

7-2) JDBC Resourceの作成

作成したJDBC Console Poolに対応するJDBC Resourceを作成します。左のペインから[リソース] > [JDBC] > [JDBCリソース]を選択すると、[JDBCリソース]ページが表示されます。[新規...]をクリックすると、[新しい JDBC リソース]ページが表示されます。

以下の設定項目を入力してください。

設定項目	設定内容
JNDI 名	jdbc/Symfo
プール名	SymfoPool
ターゲット(選択したターゲット)	MyDebugJEE

7-3) JDBCドライバのクラスパスの設定

Symfoware JDBCドライバをクラスパスに設定します。左のペインから[設定] > [MyDebugJEE-config] > [JVM 設定]を選択すると、[JVM一般設定]ページが表示されます。[パス設定]タブをクリックして、以下の設定をします。

以下の設定項目を入力してください。

設定項目	設定内容
クラスパスのサフィックス	C:/SFWCLNT/JDBC/fjjdbc/lib/fjsymjdbc2.jar
ネイティブライブラリのサフィックス	C:/SFWCLNT/JDBC/fjjdbc/bin

8) アプリケーションの動作確認

8-1) プロジェクトとサーバの関連付け

サーバビューでサーバを選択し、コンテキストメニューから[プロジェクトの追加および除去]を選択してください。
使用可能なプロジェクトから作成したEARプロジェクトを選択し、[追加]をクリックしてください。構成プロジェクトにEARプロジェクトが追加されていることを確認してください。

以下の情報を確認後、[完了]をクリックしてください。

設定項目	設定内容
構成プロジェクト	JPAEARSample

ポイント

サーバビューに配備先となるサーバが登録されていない場合は、サーバを追加する必要があります。サーバを追加する方法については、"6.2.7 アプリケーションの動作確認"を参照してください。

8-2) ブレークポイントの設定

ロジッククラスのgetCountryData()メソッドの先頭行にブレークポイントを設定します。ブレークポイントは、エディタの左側のルーラー部分をダブルクリックして、設定もしくは解除します。

8-3) サーバの起動

サーバビューでサーバを選択し、コンテキストメニューから[デバッグ]を選択してください。

ポイント

デバッガを使用せず単純にアプリケーションを起動したい場合には、コンテキストメニューから[開始]を選択します。

サーバが起動する前に自動的にEARファイルが配備されます。

サーバビューで以下の情報を確認してください。

確認項目	確認内容
サーバの状態	始動済み
サーバの状況	同期済み
EARファイル(JPAEARSample)の状況	同期済み
EARファイル配下のJARファイル(JPASample)の状況	同期済み
EARファイル配下のWARファイル(JPAClientSample)の状況	同期済み

8-4) アプリケーションの実行

サーバビューで、配備されているWebプロジェクト（JPAClientSample）を選択して、コンテキストメニューから[Webブラウザ]を選択します。Webブラウザが起動されて、入力画面が開きます。

設定項目	設定内容
入力画面のURL	http://localhost/JPAClientSample/

[順位:]の入力フィールドに人口ランキングの順位を入力して、[OK]をクリックします。

ポイント

プロジェクトを選択し、コンテキストメニューから[実行] > [サーバで実行]または[デバッグ] > [サーバでデバッグ]を選択することで、サーバの追加、プロジェクトの追加、サーバの起動、クライアントの起動をまとめて行うことができます。
また、クライアントとして使用するWebブラウザは、メニュー [ウィンドウ] > [Web ブラウザ] から変更することができます。

8-5) アプリケーションのデバッグ

アプリケーションが動作し、ブレークポイントで中断されます。変数ビューの表示を確認します。メニューから[実行] > [ステップオーバ]を選択し、変数ビューでプログラムの状況を確認します。デバッグについては、"6.2.7.1 デバッグする"を参照してください。

ポイント

変数ビューでは値の確認だけでなく、変更も行うことができます。

デバッガで中断している処理は、メニューから[実行] > [再開]を選択することで再開され、サーバ側での処理が終了し、Webブラウザに出力画面が表示されます。入力した順位の国名と総人口が表示されることを確認してください。

9) 運用環境へのアプリケーションの配布

アプリケーションを運用環境に配布するには、EARファイルを作成する必要があります。作成したEARファイルは、サーバの配備機能を利用して、サーバに配備します。

9-1) アプリケーションのエクスポート

EARファイルの作成は、エクスポートウィザードから行います。エクスポートウィザードを起動するには、[ファイル] > [エクスポート]を選択してください。[エクスポート]ウィザードから[Java EE] > [EARファイル]を選択してください。

EARファイルエクスポートウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
EARアプリケーション	エンタープライズアプリケーションプロジェクトを指定します。
宛先	EARファイルの作成先を指定します。

9-2) 運用環境に配備する

運用環境のInterstage管理コンソールにログインし、ローカルにあるEARファイルを指定することにより、リモート環境から運用環境に配備を行うことができます。

4.3 タスク

ここでは、Interstage StudioでJPAを使用したアプリケーションを開発する方法について、開発作業課題(タスク)ごとに説明します。

- 4.3.1 JPAを使用したアプリケーションを作成する環境を準備する
- 4.3.2 永続ユニットを開発する
- 4.3.3 エンティティクラスを作成する
- 4.3.4 JPAでデータベースを操作する

- 4.3.5 JPAを使用したアプリケーションの動作を確認する
- 4.3.6 JPAを使用したアプリケーションを運用環境に配布する

4.3.1 JPAを使用したアプリケーションを作成する環境を準備する

JPAを使用したアプリケーションを作成するには、提供する形態により以下の方法があります。

- ライブラリ
  JPAを使用したプログラムを部品として提供する場合は、この方法を選択します。

- EJBやWebアプリケーションに組み込む
  EJBやWebアプリケーションだけで利用し、JPAを使用したプログラムのインタフェースを厳密に規定しない場合は、この方法を選択します。

提供する形態が決定したら、プロジェクトを作成し、必要なライブラリをプロジェクトに設定してビルドできる環境を準備します。

ポイント

JPAを使用したアプリケーションを開発する場合は、データベースの接続環境を構築しておくとデータベースのテーブルからエンティティクラスを生成するなどの支援機能を使用できます。
データベースの接続環境の構築については、"8.3.1 データベースに接続する"を参照してください。構築したデータベースの接続情報は、JPAに関する情報の設定で利用します。詳細は、"4.3.1.1 JPAファセットの設定"を参考にしてください。

プロジェクトの作成

JPAを使用するアプリケーションに対応したプロジェクトをプロジェクトウィザードで作成します。このプロジェクトウィザードには[プロジェクトファセット]ページがあります。このページで必ず[Java 永続化]プロジェクトファセットを選択してください。

以下にプロジェクトの説明を示します。

- Webアプリケーション
  WebアプリケーションでJPAを使用する場合は、動的Webプロジェクトを作成します。[新規プロジェクト]ウィザードから[Web] > [動的Webプロジェクト]を選択します。
  動的Webプロジェクトウィザードの詳細については、"2.3.1 Webアプリケーションを作成する環境を準備する"を参照してください。

- EJBアプリケーション
  EJBアプリケーションでJPAを使用する場合は、EJBプロジェクトを作成します。[新規プロジェクト]ウィザードから[EJB] > [EJBプロジェクト]を選択します。
  EJBプロジェクトウィザードの詳細については、"3.3.1 EJBを作成する環境を準備する"を参照してください。

- ライブラリ
  ライブラリとしてJPAを使用するプログラムを提供する場合は、JPAプロジェクトを作成します。[新規プロジェクト]ウィザードから[JPA] > [JPAプロジェクト]を選択します。
  プロジェクトウィザードの共通部分の詳細については、"6.2.2.1 プロジェクトを新規に作成する"を参照してください。

[Java 永続化]プロジェクトファセットを選択した場合は、JPAに関する情報を設定することができます。JPAに関する情報の設定については、"4.3.1.1 JPAファセットの設定"を参考にしてください。

ポイント

既存のEJBプロジェクトや動的WebプロジェクトでJPAを使用したい場合は、プロジェクトプロパティの[プロジェクトファセット]から[Java 永続化]を追加し、JPAに関する情報を設定します。
JPAに関する情報の設定については、"4.3.1.1 JPAファセットの設定"を参考にしてください。

クラスパスの設定

アプリケーションを作成するのに必要なライブラリなどがある場合には、クラスパスの設定を行う必要があります。クラスパスの設定は、プロジェクトのビルドパスで行います。
ビルドパスの設定の詳細については、"6.2.2.3 クラスパスを設定する"を参照してください。

4.3.1.1 JPAファセットの設定

[Java 永続化]プロジェクトファセットをプロジェクトに追加する場合には、プロジェクトウィザードやファセット追加ウィザードで、以下のJPAに関する情報の設定が行えます。

- 接続
  データベースの接続をプロジェクトに設定できます。ただし、データベースの接続を設定しなくても、JPAを使用するアプリケーションは作成できます。
  データベースの接続については、"8.3.1 データベースに接続する"を参照してください。

- JPA 実装
  JPAの実装クラスを指定できます。サーバランタイムが提供する実装を使用するか、実装ライブラリを使用するかを選択します。

- 永続クラスの管理
  persistence.xmlにエンティティクラスを記述する管理をするか、persistence.xmlにエンティティクラスを記述せずアノテーションのクラスが自動的に検出されるようにするかを選択します。

- orm.xmlの作成
  O/Rマッピングファイルを作成するか作成しないかを選択します。

ポイント

接続と永続クラスの管理については、プロジェクトプロパティの[JPA]から指定することもできます。

4.3.2 永続ユニットを開発する

永続ユニットを開発するには、persistence.xml、複数の永続クラス(エンティティクラスなど)およびO/Rマッピングファイルを定義します。
persistence.xmlには、永続ユニットで使用するデータソース、明示的に宣言する永続クラスやO/Rマッピングファイルを記述します。
O/Rマッピングファイルは、アノテーションを使用しない永続クラスの情報を記述するか、アノテーションでJavaソースに記述した情報を上書きするために使用します。

ポイント

プロジェクト作成時に作成されるorm.xmlは、persistence.xmlに明示的に宣言する必要が無いO/Rマッピングファイルです。
persistence.xmlでの永続クラスの宣言もアノテーションを用いた開発を行っていれば、異なるJARファイルにクラスが格納されている場合などを除いて通常は必要ありません。

上記のような仕様のため、永続ユニットの開発はアノテーションを利用した永続クラスの作成とpersistence.xmlやO/Rマッピングファイルの編集になります。
アノテーションを利用した永続クラスの作成方法については、"4.3.3 エンティティクラスを作成する"を参照してください。

persistence.xmlの編集

persistence.xmlを編集するには、永続化 XMLエディタを用います。
永続化 XMLエディタには、[一般]、[接続]、[プロパティ]タブと[ソース]タブがあります。[ソース]タブではXMLファイルを直接編集でき、コンテンツアシストでタグの追加が行えます。
[一般]、[接続]、[プロパティ]タブでは、XMLの内容をGUIで編集できます。
例えば、永続ユニットで使用するデータソースを宣言するためには、以下のようにpersistence.xmlを記述します。

persistence.xmlの使用例

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/
persistence_1_0.xsd">
    <persistence-unit name="JPASample">
        <jta-data-source>jdbc/Symfo</jta-data-source>
    </persistence-unit>
</persistence>
```

O/Rマッピングファイルの編集

O/Rマッピングファイルの編集は、XMLエディタで行います。

また、XMLエディタで開いている状態で、[JPA構造]ビューでエンティティマッピングを選択すると、[JPA詳細]ビューでいくつかの要素について設定することができます。設定の詳細は、以下を参考にしてください。

- パッケージ
  XMLファイルに記述されているマッピング情報で、完全修飾名で記述されていないクラスに適用されるパッケージを指定します。

- スキーマ、カタログ
  デフォルトで使用されるスキーマやカタログを指定します。

- アクセス
  永続プロバイダがEntityにフィールドもしくはプロパティのどちらでアクセスするかのデフォルトを指定します。

ポイント

永続ユニット配下にある要素は、アノテーションで記述されている永続クラスに適用されます。それ以外の項目は、O/Rマッピングファイル(orm.xml)に記述されている永続クラスに適用されます。

JPAアプリケーションとしての検証

JPAアプリケーションバリデータによって、永続ユニットの検証が行われます。
検証の詳細については、"6.2.6.2 検証"を参照してください。

4.3.3 エンティティクラスを作成する

エンティティクラスを作成するには、新規 JPA エンティティウィザードを使用することで、@Entityアノテーションなどが設定されたクラスを生成することができます。

ポイント

データベースのテーブルからエンティティクラスを生成することもできます。
詳細は、"4.3.3.4 テーブルからエンティティクラスを生成する"を参照してください。

JPAパースペクティブを開く

はじめにJPAパースペクティブを開きます。

エンティティクラスの作成

エンティティクラスの作成は、[新規]ウィザードから[JPA] > [エンティティ]を選択し、ウィザードで作成します。ウィザードでの設定は、以下を参考にしてください。

- プロジェクト
  エンティティクラスを生成するプロジェクトを指定します。

- ソースフォルダ
  エンティティクラスのソースを格納するフォルダを指定します。

- Java パッケージ
  エンティティクラスのパッケージ名を指定します。

- クラス名
  エンティティクラス名を指定します。

- 継承
  エンティティクラスが継承するクラスを"エンティティ"か"マップドスーパークラス"から選択します。

- XMLエンティティマッピング
  O/Rマッピングファイルにエンティティを追加する場合は、"XMLのエンティティマッピングに追加"をチェックします。

- エンティティ名
  エンティティ名を指定します。

- テーブル名
  既存のテーブルと関連付ける場合は、"デフォルトを使用"のチェックをはずして、テーブル名を指定します。

- エンティティフィールド
  エンティティフィールドを追加します。フィールド名と型を指定して追加します。主キーとするフィールドは、"キー"をチェックします。

- アクセスタイプ
  エンティティのアクセスタイプを"フィールドベース"か"プロパティベース"かを指定します。

アノテーションの設定

[JPA構造]ビューをアクティブにして、作成したエンティティクラスを選択します。[JPA詳細]ビューの [タイプ]で表示されているマッピングタイプをクリックします。[マッピングタイプの選択]ダイアログボックスでマッピングタイプを選択すると、エンティティクラスにアノテーションが設定されます。
Entityは、継承や埋め込みなどを利用し複雑なデータ構成にも対応したものも作成できます。
詳細は、"4.3.3.1 データ構成を考慮してエンティティを作成する"を参照してください。

ポイント

Javaクラスをエンティティにする場合は、JPA構造ビューにJavaクラスが表示されないため、Javaソースに直接アノテーション定義を追加してください。

エンティティクラスとデータベースのマッピング

エンティティクラスとデータベースとをマッピングする場合は、[JPA詳細]ビューの[テーブル]グループにある[名前]コンボボックスで、データベースのテーブル名を指定します。クラス名と同じ名前のテーブルとマッピングする場合は、デフォルトのままでかまいません。

エンティティクラスのフィールドとデータベースのカラムもマッピングできます。エンティティクラスにフィールドを作成して、そのフィールドを[JPA構造]ビューで選択します。[JPA詳細]ビューの[カラム]グループにある[名前]コンボボックスで、データベースのテーブル名を指定します。フィールド名と同じ名前のカラムとマッピングする場合は、デフォルトのままでかまいません。

ポイント

プロジェクトに設定したデータベースに接続している場合は、[JPA詳細]ビューのコンボボックスにマッピングできる候補が一覧されます。

データベースとのマッピングは、単純なテーブルやカラムとの対応付け以外も行うことができます。
データベースとのマッピングの詳細は、"4.3.3.2 エンティティとデータベースをマッピングする"を参照してください。
また、Entity間の関係についてもデータベースの情報と絡めてマッピングすることができます。
詳細は、"4.3.3.3 エンティティ間の関係を定義する"を参照してください。

ポイント

データベースとのマッピングはアノテーションだけでなく、O/Rマッピングファイルでも行うことができます。複数の環境で使用する場合などアプリケーションの可搬性を高めたい場合は、O/Rマッピングファイルを利用してください。

4.3.3.1 データ構成を考慮してエンティティを作成する

エンティティはエンティティを継承して作成することができます。エンティティ以外にも共通の永続化項目を特殊なスーパークラスで管理し、それを継承して作成することもできます。そのスーパークラスを作成するには、@MappedSuperclassアノテーションを設定します。
継承してエンティティを作成した場合、継承している永続化項目の属性を上書きして変更することや、継承している各エンティティのデータベーステーブルへのマッピング方法を指定することができます。これらには、@AttributeOverrideアノテーションや@Inheritanceアノテーションを使用します。

また、関連する永続項目をまとめて特殊なクラスとして管理し、エンティティのフィールドの型として使用することで、エンティティに特殊なクラスを埋め込むこともできます。その特殊なクラスを作成するには、@Embeddableアノテーションを設定し、エンティティに埋め込む場合には@Embeddedもしくは@EmbeddedIdアノテーションをエンティティのフィールドに設定します。
埋め込みを行った場合、埋め込んだクラスの永続項目の属性を上書きして変更することもできます。

MappedSuperclassアノテーションの設定

[JPA構造]ビューでスーパークラスを選択します。[JPA詳細]ビューの[タイプ]で表示されているマッピングタイプをクリックします。[マッピングタイプの選択]ダイアログボックスで"Mapped Superclass"を選択すると、@MappedSuperclassアノテーションが設定されます。

Javaクラスの場合は、クラスファイルをJavaエディタで開き、直接アノテーションを定義してください。

AttributeOverrideアノテーションの設定

[JPA構造]ビューでエンティティクラスを選択します。[JPA詳細]ビューの[属性の上書き]で、上書きしたい永続項目を選択し、属性の上書きを行います。

Inheritanceアノテーションの設定

[JPA構造]ビューで階層のルートのエンティティクラスを選択します。[JPA詳細]ビューの[継承]で、エンティティのデータベーステーブルへのマッピング方法を指定します。
設定については、以下を参考にしてください。

- 方法
  テーブルのマッピング方法について以下から選択します。

  - 単一テーブル
    継承関係にあるすべてのエンティティを1つのテーブルにマッピングします。この方式の場合、データベースのテーブルの各行が継承関係にあるエンティティクラスのどのデータかを示すためのカラム(識別カラム)がテーブルに必要になります。

  - JOINED
    各エンティティクラス単位にテーブルにマッピングします。各テーブルにはそのエンティティクラス固有の永続項目(継承している永続項目を含まない)に対応するカラムと結合カラムを定義します。結合カラムは、階層のルートのエンティティに対応するテーブル(主テーブル)の主キーに対応付けるためのカラムです。

ー クラス毎にテーブル
各エンティティクラス単位にテーブルにマッピングし、そのエンティティクラスの永続項目 (継承している永続項目を含む)に対応するカラムを定義します。

・ 識別カラム、識別タイプ、識別値
方法として"単一テーブル"を選択した場合に指定します。識別値は、継承関係にあるエンティティクラスを識別するための値を設定します。
これらの項目は、@DiscriminatorColumn、@DiscriminatorValueアノテーションに相当します。

・ 主キー結合カラム
方法として"JOINED"を選択した場合に指定します。現在のエンティティの主キーのカラムと主テーブルの主キーのカラムから参照されるカラムをペアで指定します。
この項目は、@PrimaryKeyJoinColumnアノテーションに相当します。

Embeddableアノテーションの設定

[JPA構造]ビューでJavaクラスを選択します。[JPA詳細]ビューの[タイプ]で表示されているマッピングタイプをクリックします。[マッピングタイプの選択]ダイアログボックスで"Embeddable"を選択すると、@Embeddableアノテーションが設定されます。

Javaクラスの場合は、クラスファイルをJavaエディタで開き、直接アノテーションを定義してください。

EmbeddedもしくはEmbeddedIdアノテーションの設定

[JPA構造]ビューでエンティティクラスの@Embeddableアノテーションを設定したクラス型の永続項目を選択します。[JPA詳細]ビューの[属性]で表示されているマッピングタイプをクリックします。[マッピングタイプの選択]ダイアログボックスで"Embedded"もしくは"Embedded Id"を選択すると、永続項目に@Embeddedアノテーションもしくは@EmbeddedIdアノテーションが設定されます。
"Embedded" を選択した場合、[JPA詳細]ビューの[属性の上書き]で、@Embeddableアノテーションを設定したクラスの属性を上書きして変更することができます。

4.3.3.2 エンティティとデータベースをマッピングする

エンティティとデータベーステーブルをマッピングするには、@Tableアノテーションを使用します。また、エンティティを1つ以上のデータベーステーブルとマッピングすることもでき、その場合には、@SecondaryTableアノテーションを使用します。

エンティティとテーブルをマッピングした場合には、永続項目とカラムは名前で対応付けられますが、永続項目単位にデータベースのカラムにマッピングすることもできます。また、主キーになる永続項目を宣言する必要があります。

マッピングするカラムの情報以外にも、データ読み込み(フェッチ)の方法を指定したり、型に関するオプションを指定したり、永続化しない項目やOptimistic Lockingに使用する項目を指定することもできます。

Tableアノテーションの設定

[JPA構造]ビューでエンティティクラスを選択します。[JPA詳細]ビューの[テーブル]で、テーブル名、カタログ、スキーマを指定します。

SecondaryTableアノテーションの設定

[JPA構造]ビューでエンティティクラスを選択します。[JPA詳細]ビューの[二次テーブル]で、主テーブル(@Tableアノテーションで指定したもの)以外のテーブルを追加します。

[主キー結合カラム]は、主テーブルの主キーと二次テーブルとして追加したテーブルのどのカラムを対応付けるかを指定します。

永続項目とカラムのマッピング

[JPA構造]ビューでエンティティクラスの永続項目を選択します。[JPA詳細]ビューの[カラム]で、カラム名、テーブルを指定します。
[挿入可能]、[更新可能]は、その永続項目に対応するカラムを挿入や更新対象とするかを指定する項目です。

主キーの設定

[JPA構造]ビューでエンティティクラスの主キーになる永続項目を選択します。[JPA詳細]ビューの[属性]で表示されているマッピングタイプで"Id"を選択します。すると、フィールドに@Idアノテーションが設定されます。

プログラム内で主キーの値を明示的に設定せずに、主キーの値を自動的に生成する場合には、上記の[JPA詳細]ビューの[主キーの生成]で行います。設定については、以下を参考にしてください。

この項目は、@GeneratedValue、@TableGenerator、@SequenceGeneratorアノテーションに相当します。

- 方法
  主キーの生成方法について以下から選択します。

  ー 自動
  実行環境がデータベースに適切な生成方法を選択します。

  ー 識別
  データベースのIDENTITYカラムを使用します。

  ー シーケンス
  データベースのシーケンスを使用します。

  ー テーブル
  データベースのテーブルを使用します。

- ジェネレータ名
  テーブルジェネレータやシーケンスジェネレータで指定した名前を指定します。

- テーブルジェネレータ
  名前はジェネレータ名に指定する名前です。値カラムはジェネレータで生成した値を格納するカラムです。主キーカラムの値は、このテーブルが複数の主キーの生成に使われる場合に、どのジェネレータのものか識別するための値を指定します。(主キーカラムと値カラムにこの値と最後に生成された主キーの値がペアで格納されます。)

- シーケンスジェネレータ
  名前はジェネレータ名に指定する名前です。シーケンスには、データベースのシーケンス名を指定します。

フェッチ方法の指定

[JPA構造]ビューで永続項目を選択している状態で、[JPA詳細]ビューの[フェッチ]でフェッチ方法の指定を行います。
設定については、以下を参考にしてください。

- EAGER
  Entityが取得されるときにフェッチされなければいけない場合に指定します。

- LAZY
  最初に永続項目にアクセスされるときにフェッチしてもいい場合に指定します。

日時型に関するオプションの指定

永続項目の型が日時関連の場合には、Date(日付)、Time(時間)、Timestamp(日時)のどれに対応付けるかを指定できます。これには、@Temporalアノテーションを使用します。

@Temporalアノテーションを設定するには、[JPA構造]ビューで日時関連の型の永続項目を選択し、[JPA詳細]ビューの[日時型]でコンボボックスから対応付ける方法を選択します。

列挙型に関するオプションの指定

永続項目が列挙型の場合には、値をOrdinal(序数)とString(文字列)のどちらで扱うかを指定できます。これには、@Enumeratedアノテーションを使用します。

@Enumeratedアノテーションを設定するには、[JPA構造]ビューで列挙型の永続項目を選択し、[JPA詳細]ビューの[列挙型]でコンボボックスから扱い方を選択します。

巨大オブジェクト型に関するオプションの指定

永続項目の型が、データベースの巨大オブジェクト型(LOB)に対応付けられる型の場合に巨大オブジェクトとして扱うかを指定できます。これには、@Lobアノテーションを使用します。

@Lobアノテーションを設定するには、[JPA構造]ビューで巨大オブジェクト型に対応付けられる永続項目を選択し、[JPA詳細]ビューの[Lob]をチェックします。

永続化しない項目の指定

エンティティクラスに宣言されるフィールドやプロパティは、デフォルトで永続化対象として扱われてしまいます。永続化しない項目については、@Transientアノテーションを設定する必要があります。

@Transientアノテーションを設定するには、[JPA構造]ビューで永続化しない項目を選択し、[JPA詳細]ビューの[属性]で表示されているマッピングタイプで"Transient"を選択します。

Optimistic Lockingを適用する項目の指定

データを読み込んだ時と更新するときで値が変更されていないかをチェックするような排他方式をOptimistic Lockingといいます。この方式を適用する項目については、@Versionアノテーションを設定する必要があります。

@Versionアノテーションを設定するには、[JPA構造]ビューで項目を選択し、[JPA詳細]ビューの[属性]で表示されているマッピングタイプで"Version"を選択します。

4.3.3.3 エンティティ間の関係を定義する

エンティティ間の関係には、1対1、多対1、1対多、多対多の関係があります。関係には方向性もあり、どちらからも参照できる双方向と、片側からしか参照できない単方向があります。
エンティティ間の関係を定義するには、参照される側のEntity型やそのコレクション型のフィールドやプロパティをエンティティに定義し、関係の種別に合わせて@OneToOne、@ManyToOne、@OneToMany、@ManyToManyアノテーションを使用します。

1対1の関係もしくは多対1の関係を定義する

参照する側のエンティティに参照される側のEntity型のフィールドもしくはプロパティを追加します。[JPA構造]ビューで追加した項目を選択し、[JPA詳細]ビューの[属性]で表示されているマッピングタイプで"One to One"もしくは"Many to One"を選択し、関係の定義を行います。関係の定義については、以下を参考にしてください。

- 対象のEntity
  フィールドもしくはプロパティの型から判別可能なため指定する必要はありません。

- フェッチ
  参照先のEntityをどのタイミングでフェッチするかを以下から選択します。

  - EAGER
    エンティティが取得されるときにフェッチされなければいけない場合に指定します。

  - LAZY
    最初にアクセスされるときにフェッチしてもいい場合に指定します。

- 所有側
  関係が双方向の場合に、被所有者側の定義から所有側の永続項目を指定します。
  1対1の関係の場合には、外部キーを持つ側が所有側になります。
  多対1の関係の場合には、多側が所有側になるため、この領域を指定することはできません。

- カスケード
  参照先のエンティティにカスケードする永続化処理(メソッド)を選択します。例えば、エンティティ自身が削除された場合に参照先まで削除を実施するかなどを考慮して選択します。

- 結合カラム
  1対1もしくは多対1の関係の場合には、外部キーを使って関係を定義します。そのため、結合カラムとしては、外部キーのカラムの名前とそれによって参照される参照先のカラム名などを指定します。
  この項目は、@JoinColumnアノテーションに相当します。

1対多の関係もしくは多対多の関係を定義する

参照する側のエンティティに参照される側のエンティティのコレクション型のフィールドもしくはプロパティを追加します。[JPA構造]ビューで追加した項目を選択し、[JPA詳細]ビューの[属性]で表示されているマッピングタイプで"One to Many"もしくは"Many to Many"を選択し、関係の定義を行います。
関係の定義については、以下を参考にしてください。(1対1もしくは多対1と同一のものは省略しています。)

- 対象のEntity
  コレクション型を宣言する場合に総称を使用していれば、フィールドもしくはプロパティの型から判別可能なため指定する必要はありません。総称を使用していない場合は、参照先のEntity型を指定します。

- 所有側
  1対多の関係の場合には多側が所有側になるため、双方向の場合にはこの領域を指定します。
  多対多の関係で双方向の場合には、どちらかを所有側に決めて、被所有側でこの領域を指定します。

- カスケード
  参照先のエンティティにカスケードする永続化処理(メソッド)を選択します。例えば、エンティティ自身が削除された場合に参照先まで削除を実施するかなどを考慮して選択します。

- 順序
  フェッチする場合の参照先のエンティティの順序を指定します。
  この項目は、@OrderByアノテーションに相当します。

- 結合テーブル
  1対多もしくは多対多の関係の場合には、結合テーブルを使って関係を定義します。結合テーブルには、所有者側と被所有者側に対応するカラムが必要で、それぞれについてカラムの名前とそのカラムから参照されるエンティティに対応するテーブルのカラムを指定します。
  この項目は、@JoinTableアノテーションに相当します。

4.3.3.4 テーブルからエンティティクラスを生成する

データベースの接続をプロジェクトに設定している場合は、以下の手順でデータベースのテーブルからエンティティクラスを生成することができます。

1. [データソースエクスプローラ]ビューでプロジェクトに設定したデータベースに接続します。
   データベースに接続する詳細については、"8.3.1 データベースに接続する"を参照してください。
2. プロジェクトを右クリックして、[JPAツール] > [Entityの生成]を選択します。
3. 表示された[Entityの生成]ダイアログボックスで、テーブルを選択して、ソースフォルダやパッケージを入力します。

4.3.4 JPAでデータベースを操作する

JPAでは、Entityマネージャでエンティティを操作することが、データベースを操作することになります。そのため、Entityマネージャが必要になります。取得したEntityマネージャでEntityを操作することで、データベースの検索、挿入、更新、削除に相当する操作を行うことができます。

ポイント

厳密には、Entityマネージャによる操作は永続コンテキスト(メモリ内部に構築されたEntityの集まりのようなイメージ)に対して行われ、データベースとの同期はトランザクションの区切りのタイミングで行われます。

Entityマネージャの取得

Entityマネージャを取得する場合は、以下のように@PersistenceContextアノテーションを使用します。

PersistenceContextアノテーションの使用例

```
@PersistenceContext
private EntityManager em;
```

Entityマネージャによる永続コンテキストの操作

Entityマネージャによる永続コンテキストを操作する例を、以下に示します。

Entityの主キーによる検索

EMPLOYEEテーブルからID(主キー)を指定して対応する従業員を検索する例を示します。Entityマネージャのfindメソッドにエンティティクラスと主キーの値を指定して検索します。

Entityの主キーによる検索例

```
public Employee getEmployeeByPrimaryKey (int id) {
    return em.find(Employee.class, id);
}
```

EntityのJava Persistence Query Languageによる検索

EMPLOYEEテーブルから名前が一致する従業員を検索する例を示します。EntityマネージャのcreateQueryメソッドでクエリを作成してパラメタを設定してから実行し、検索結果を取得しています。

EntityのJava Persistence Query Languageによる検索例

```
public Collection<Employee> getEmployeeByName (String name) {
    Query query = em.createQuery("SELECT employee FROM Employee employee WHERE employee.name = :name");
    query.setParameter("name" ,name);
    return query.getResultList();
}
```

Entityの挿入

EMPLOYEEテーブルに新しい従業員を追加する例を示します。エンティティクラス(Employee)のインスタンスを作成し、Entityマネージャのpersistメソッドで永続コンテキストに追加します。

Entityの挿入例

```
public void insertEmployee (int id, String name, String address, String tel) {
    Employee newEmployee = new Employee(id, name, address, tel);
    em.persist(newEmployee);
}
```

Entityの更新

EMPLOYEEテーブルからID(主キー)を指定して対応する従業員を検索し、住所と電話番号を変更する例を示します。Entityマネージャのfindメソッドで検索し、エンティティクラスの値を変更します。

Entityの更新例

```
public void updateEmployee (int id, String address, String tel) {
    Employee employee = em.find(Employee.class, id);
    employee.setAddress(address);
    employee.setTel(tel);
}
```

ポイント

永続コンテキスト内での変更のため、Entityのインスタンスを変更するだけで自動的に変更が検出されます。
永続コンテキスト外のインスタンスを更新した場合(例えば、上述したgetEmployeeByPrimaryKeyメソッドの復帰値のインスタンスを更新するような場合)にはEntityマネージャのmergeメソッドを呼び出し、永続コンテキストに変更を通知する必要があります。

Entityの削除

EMPLOYEEテーブルからID(主キー)を指定して対応する従業員を検索し、その従業員の情報を削除する例を示します。Entityマネージャのfindメソッドで検索したエンティティクラスのインスタンスを引数に指定してremoveメソッドを呼び出し、永続コンテキストから削除します。

Entityの主キーによる削除例

```
public void deleteEmployee (int id) {
    Employee employee = em.find(Employee.class, id);
    em.remove(employee);
}
```

4.3.5 JPAを使用したアプリケーションの動作を確認する

JPAを使用したアプリケーションの動作を確認するには、アプリケーションを実行環境に配備してから実行する必要があります。これらの操作はサーバビューから行います。
操作の詳細については、"6.2.7 アプリケーションの動作確認"を参照してください。

4.3.6 JPAを使用したアプリケーションを運用環境に配布する

作成したJava EEアプリケーションを運用環境に配布するには、アーカイブファイルを作成し、管理コンソールから配備を行う必要があります。

アーカイブファイルの作成

アーカイブファイルの作成は、エクスポートウィザードから行います。
エクスポートウィザードでの操作の詳細については、"6.2.8 運用環境への配布"を参照してください。

管理コンソールでの配備

管理コンソールはリモート環境から操作できます。運用環境の管理コンソールにログインし、ローカルにある配備するファイルを指定することにより、リモート環境から運用環境に配備を行うことができます。
配備や管理コンソールの詳細については、Interstage Application ServerのJava EE実行環境の使用方法を別途ご確認ください。

第5章 Webサービスのアプリケーションを開発する

ここでは、Webサービスの概要とJava EEアプリケーション実行環境上で動作する、Webサービスアプリケーションおよびクライアントアプリケーションの作成方法について説明します。

5.1 概要

WebサービスとWebサービス開発支援機能の概要について説明します。

5.1.1 Webサービスとは

Webサービスは、ネットワークを通じてシステムの機能をサービスとして公開する技術で、既存システムの有効活用やソフトウェアの再利用を促進する方法として注目を集めています。
これらを実現するために、Webサービスは、HTTPを下位プロトコルに利用するSOAPを使用して通信を行い、公開するサービスのインタフェースはWSDLで定義することが標準となっています。
また、Webサービスは相互接続性が重要であり、制約や、Webサービスで使用する仕様(SOAP/WSDL/UDDIなど)の曖昧な点の明確化などが、WS-I Basic Profileに規定されています。

実際にWebサービスを提供するには、以下の規約に基づいてサービスエンドポイントインタフェースを決定し、サービスエンドポイントを提供します。

- SOAP
  ネットワーク経由でオブジェクト間通信を行うプロトコルです。XMLを用いて記述するデータ構造が既定されていますが、データを転送するプロトコルは既定されておらず、HTTPやSMTPなど様々なプロトコルにのせることができます。ただし、一般的には、HTTPを利用することが多いようです。

- JAX-WS
  アプリケーションのリモート呼び出し(Remote Procedure Call)をJavaで実現するために規定されたJAX-RPCの後継です。Webサービスで公開する機能をJavaで記述する場合には、この規約の範囲内で作成します。

- WSDL
  Webサービスの仕様をユーザに公開するための記述方法です。XMLで記述します。

- WS-I Basic Profile 1.1
  相互接続性向上の観点からWebサービスの標準であるSOAP/WSDL/UDDIなどの仕様について詳細化・制約付加を規定した規約です。ワークベンチでWebサービスアプリケーションを作成する過程において、通常は規約の中身について意識する必要はありません。WSDLを直接作成して公開するような場合は、規約を意識する必要があります。

- WS-I Attachments Profile 1.0
  添付ファイルを利用したときにWebサービスの相互接続性向上のため、SOAP Messages with Attachmentsをベースにして、SOAPやWSDL定義の仕様の明確化や使用範囲の限定を定めた規約です。WebサービスをWS-I Attachments Profileに準拠させることにより、添付ファイル付きSOAPメッセージを送受信する際のシステム間での相互運用性が向上します。

ポイント

開発方法には、サービスエンドポイントインタフェースをWSDLで設計するトップダウン開発と、サービスエンドポイントとなるJavaクラスを作成し、その結果としてサービスエンドポイントインタフェースが決定するボトムアップ開発があります。

5.1.1.1 J2EE1.4からの変更点

J2EE1.4のWebサービスとJava EEのWebサービスでは以下の点で仕様が異なります。

- JAX-RPCとJAX-WS
  J2EE1.4ではJAX-RPCに準拠していましたが、Java EEではJAX-WSに準拠したWebサービスの開発を推奨しています。

・ Webサービスを構成するファイル
J2EE1.4ではSEI(サービスエンドポイントインタフェース)、実装クラス、WSDL、マッピングファイル、deployment descriptorなどWebサービスを作成するために多数のファイルを必要としましたが、Java EEでは、Javaクラスに@WebServiceアノテーションを宣言するだけでWebサービスを作成することができます。

・ Webサービスの呼び出し方法
Java EEではWebサービスに限らず、Dependency Injectionを用いて従来のlookup処理を簡単に記述することができます。

以下に、Webサービス開発時に使用する主なアノテーションについて、簡単な例を示しながらJava EEでのWebサービスについて紹介します。

javax.jws.WebServiceアノテーション

クラスをWebサービスとして定義するアノテーションです。以下のプロパティを指定できます。

プロパティ名	解説
endpointInterface	クラスのメソッドをすべて公開したくないような場合にサービスエンドポイントインタフェースを定義し、それを宣言するために使用します。
name	WSDLのportTypeの名前
portName	WSDLのportの名前
serviceName	WSDLのサービスの名前
targetNamespace	WSDLの名前空間
wsdlLocation	定義済みのWSDLに基づいてWebサービスを公開する場合に使用します。

WebServiceアノテーションを使用したWebサービスの作成例

```
package sample;

import javax.jws.WebService;

@WebService(endpointInterface="sample.Calc")
public class CalcImpl implements Calc {
      public int add(int param1,int param2) {
            return param1 + param2;
      }
}
```

javax.xml.ws.WebServiceRefアノテーション

Webサービスを呼び出す場合に、Dependency Injectionするためのアノテーションです。以下のプロパティを指定できます。

プロパティ名	解説
mappedName	マッピングされるリソース名
name	JNDI名
type	リソースのJava型
value	サービスクラス 通常は、javax.xml.ws.Serviceを継承します。
wsdlLocation	WSDLの位置を指定します。

WebServiceRefアノテーションを使用したWebサービスクライアントの作成例

```
@WebServiceRef(name="service/Calc")
private CalcService service;

public int callCalc(int param1,int param2) {
```

```
        Calc port = service.getCalcPort();
            return port.add(param1,param2);
}
```

5.1.2 Webサービスの開発

Webサービスの開発の概要を以下に示します。

Webサービスの開発の準備

Webサービスを開発するには、Webアプリケーションとして作成する方法とEJBアプリケーションとして作成する方法があります。それぞれの方法に合わせてプロジェクトを作成し、必要なターゲットランタイムやクラスパスなどのプロジェクト設定を行います。
詳細は、"5.3.1 Webサービスを作成する環境を準備する"を参照してください。

Webサービスの作成

Webサービスを作成する方法としては、ボトムアップ的な方法とトップダウン的な方法があります。

ボトムアップ的な方法では、Javaクラスウィザードでwebサービスを作成します。トップダウン的な方法では、WSDLウィザード、WSDLエディタ、WSDLバリデータを利用してWSDLを作成したあとで、ウィザードでサービスエンドポイントインタフェースを作成し、それを実装するクラスをJavaクラスウィザードで作成します。
詳細は、"5.3.2 Webサービスを作成する"を参照してください。
EJBアプリケーションとして提供する場合には、"5.3.3 Stateless Session BeanをWebサービス化する"を参照してください。

Webサービスクライアントの作成

Webサービスを呼び出すために必要なサービスエンドポイントインタフェースなどをWSDLから作成するウィザードを提供しています。
詳細は、"5.3.6 Webサービスクライアントを作成する"を参照してください。

Webサービスの検証

Java EEのWebサービスの仕様に沿って開発が行われているか、また、そのアプリケーションがInterstage Application Server上で動作するかを検証するInterstage Java EE検証機能を提供しています。
詳細は、"6.2.6.3 Interstage Java EE検証"を参照してください。

Webサービスの動作確認

作成したアプリケーションの動作確認はサーバビューを用いて行います。また、Webサービスの呼び出しが行えるWebサービスエクスプローラや、Webサービスのメッセージの確認が行えるTCP/IPモニタを提供しています。
詳細は、"5.3.7 Webサービスの動作を確認する"を参照してください。

Webサービスの配布

Webサービスを配布するためにはアプリケーションをアーカイブする必要があります。作成したアプリケーションのアーカイブファイルを作成するため、エクスポートウィザードを提供しています。
詳細は、"5.3.8 Webサービスを運用環境に配布する"を参照してください。

5.2 入門

ここでは、WebアプリケーションからWebサービスを呼び出すアプリケーションを作成する手順を紹介します。

5.2.1 作成するアプリケーション

国の総人口の順位付けリストから、順位を入力して国名と総人口を返すWebサービスアプリケーションを作成し、Webアプリケーションからそのwebサービスを呼び出し、その結果をWebページに表示するアプリケーションを作成します。
ここでは、Webサービスのクライアントアプリケーションとしてwebアプリケーションの入門で作成したアプリケーションを利用します。

5.2.2 開発の流れ

ここでは、以下のようにWebサービスアプリケーションの開発を進めます。

1. Webサービス用のプロジェクトの作成
   Webサービスアプリケーションを作成するために、まず動的Webアプリケーションプロジェクトを作成します。ウィザードに従って動的Webアプリケーションプロジェクトを作成することで、ビルドに必要なクラスパスの設定などが自動的に行われます。

2. Webサービスの作成
   以下の手順でWebサービスを作成します。

   ー データクラスの作成
   Webサービスの実装で利用するデータクラスを作成します。

   ー Webサービスのひな型の作成
   ウィザードでJavaクラスを作成します。

   ー WebServiceアノテーションの宣言
   ウィザードで作成されたJavaクラスにアノテーションを宣言します。

   ー Webサービスの実装
   アノテーションを宣言したクラスにメソッドを宣言し処理を実装します。

3. EARプロジェクトの作成
   クライアントも含めて1つのアプリケーションとして配布するため、EARファイルを作成するためのプロジェクトを作成します。作成済みの動的WebプロジェクトはEARプロジェクトの作成時にEARファイルに含めることを指定します。クライアントは未作成のためクライアントのプロジェクト作成時にEARファイルに含めることを指定します。

4. WSDL取得の準備
   以下の手順でクライアント作成に必要なWSDLの取得の準備を行います。

   ー EARプロジェクトとサーバの関連付け
   サーバに配備することでWSDLが取得可能となるためサーバとの関連付けを行います。

   ー サーバへの配備
   実際にサーバに配備することでサーバからWSDLを取得可能にします。

   ー WSDLのURLの確認
   WSDLを取得するためのURLを確認します。

5. Webサービスクライアントの作成
   以下の手順でクライアントとなるWebアプリケーションを作成します。

   ー クライアントプロジェクトの作成
   クライアントとしてWebアプリケーションを作成するため、動的Webプロジェクトを作成します。

   ー WSDLからサービスエンドポイントインタフェースの作成
   ウィザードを使用してWSDLからサービスエンドポイントインタフェースなどの必要なファイルを作成します。

   ー サーブレットクラスの作成
   Webアプリケーションのリクエストを受け付けるコントローラとしてのサーブレットをウィザードで作成します。

- 入出力画面の作成
  Webアプリケーションの入出力画面をウィザードで作成します。

- Dependency Injectionの指定
  サーブレットからWebサービスのメソッドを呼び出せるようにフィールドにDependency Injectionを指定します。

- Webサービスの呼び出し処理の実装
  Dependency Injectionを宣言したフィールドを利用してWebサービスのメソッドを呼び出す処理を実装します。

6. アプリケーションの動作確認
   以下の手順でアプリケーションの動作確認を行います。

   - プロジェクトとサーバの関連付け
     アプリケーションをどのサーバに配備するかを設定します。WSDLを取得するために設定済みのため、ここでは何もしません。

   - ブレークポイントの設定
     実行時にデバッガでプログラムの動作を確認するため、ブレークポイントを設定します。

   - サーバの起動
     Webブラウザからのアプリケーションに対するリクエストを受けつけられるように、サーバを起動します。サーバ起動前に配備は自動的に行われます。

   - アプリケーションの実行
     Webブラウザを起動し、アプリケーションのURLにアクセスすることで動作確認を開始します。

   - アプリケーションのデバッグ
     プログラムをデバッグし、アプリケーションが正常に動作することを確認します。

7. 運用環境へのアプリケーションの配布
   以下の手順で運用環境への配布を行います。

   - アプリケーションのエクスポート
     運用環境へアプリケーションを配布するため、EARファイルを作成します。

   - 運用環境への配布
     Interstage管理コンソールからEARファイルを配備します。

5.2.3 開発手順

以下に、実際にアプリケーションを開発する手順を説明します。

1) Webサービス用のプロジェクトの作成

2) Webサービスの作成

3) EARプロジェクトの作成

4) WSDL取得の準備

5) Webサービスクライアントの作成

6) アプリケーションの動作確認

7) 運用環境へのアプリケーションの配布

1) Webサービス用のプロジェクトの作成

メニューバーから[ファイル] > [新規] > [プロジェクト]を選択すると、[新規プロジェクト]が表示されます。

[新規プロジェクト]ウィザードから[Web] > [動的Webプロジェクト]を選択して、[次へ]をクリックします。

以下の設定項目を確認、入力してください。

設定項目	設定内容
プロジェクト名	WebServiceSample
ターゲットランタイム	Interstage Application Server V11.1 IJServer Cluster (Java EE)
動的 Web モジュールバージョン	2.5
構成	Interstage Application Server V11.1 IJServer Cluster (Java EE) デフォルト構成
EARにプロジェクトを追加	チェックしない

情報を設定後、[次へ]をクリックしてください。[Webモジュール]ページが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
コンテキストルート	WebServiceSample
コンテンツフォルダ	WebContent
Javaソースフォルダ	src
Deployment Descriptorの生成	チェックする

2) Webサービスの作成

2-1) データクラスの作成

国名と総人口の情報を保持し、そこから情報を取得するデータクラスを作成します。データクラスの作成は、作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [クラス]を選択します。[新規Javaクラス]ウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
ソースフォルダ	WebServiceSample/src
パッケージ	sample
名前	CountryData
スーパークラスからのコンストラクタ	チェックする

以下のソースファイルが生成されます。

ソースファイル	説明
CountryData.java	データクラス

国名と総人口を保持する処理を実装します。以下の**赤字**の箇所をソースに追加してください。

データクラスの実装(CountryData.java)

```
package sample;

public class CountryData {

    private String countryName;
    private int totalPopulation;
```

```
    public CountryData() {
    }

    public CountryData(String name, int total) {
        setCountryName(name);
        setTotalPopulation(total);
    }

    public String getCountryName() {
        return countryName;
    }

    public void setCountryName(String countryName) {
        this.countryName = countryName;
    }

    public int getTotalPopulation() {
        return totalPopulation;
    }

    public void setTotalPopulation(int totalPopulation) {
        this.totalPopulation = totalPopulation;
    }

}
```

ポイント

フィールドを追加した後にクラスを選択している状態で、メニューから[ソース] > [Getter および Setter の生成]を選択することで、getter/setterの追加を行うことができます。

2-2) Webサービスのひな型の作成

WebサービスはJavaクラスの作成から行います。作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [クラス]を選択します。特別なJavaクラスを作成する必要はなく、ソースフォルダ、パッケージ、名前を指定してクラスを作成します。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
ソースフォルダ	WebServiceSample/src
パッケージ	sample
名前	PopulationRanking

2-3) WebServiceアノテーションの宣言

作成したJavaクラスにWebServiceアノテーションを以下のように定義します。**赤字**の部分がWebServiceアノテーションの定義になります。

WebServiceアノテーションの定義

```
package sample;

import javax.jws.WebService;

@WebService
public class PopulationRanking {
}
```

2-4) Webサービスの実装

Webサービスとして公開したい機能のメソッドを実装します。以下の赤字の箇所をソースに追加してください。

Webサービスとして公開するメソッドの実装

```
package sample;

import javax.jws.WebService;

@WebService
public class PopulationRanking {

    private CountryData[] countries;

    public PopulationRanking(){
        countries = new CountryData[]{
             new CountryData("中国",1330000000),
             new CountryData("インド",1140000000),
             new CountryData("アメリカ",300000000),
             new CountryData("インドネシア",230000000),
             new CountryData("ブラジル",190000000),
             new CountryData("パキスタン",160000000),
             new CountryData("バングラデシュ",150000000),
             new CountryData("ロシア",140000000),
             new CountryData("ナイジェリア",140000000),
             new CountryData("日本",130000000)
        };
    }

    public CountryData getCountryData(int rank){

        --rank;
        if(9 < rank || rank < 0 ){
            return null;
        }

        return countries[rank];
    }
}
```

3) EARプロジェクトの作成

メニューバーから[ファイル] > [新規] > [プロジェクト]を選択すると、[新規プロジェクト]ウィザードが表示されます。

新規プロジェクトウィザードから[Java EE] > [エンタープライズアプリケーションプロジェクト]を選択して、[次へ]をクリックします。

以下の設定項目を確認、入力してください。

設定項目	設定内容
プロジェクト名	WebServiceEARSample
ターゲットランタイム	Interstage Application Server V11.1 IJServer Cluster (Java EE)
EARバージョン	5.0
構成	Interstage Application Server V11.1 IJServer Cluster (Java EE) デフォルト構成

情報を設定後、[次へ]をクリックしてください。EARに追加するモジュールページが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
Java EE モジュール依存関係	WebServiceSample
コンテンツフォルダ	EarContent
Deployment Descriptorの生成	チェックしない

4) WSDL取得の準備

4-1) EARプロジェクトとサーバの関連付け

サーバビューでサーバを選択し、コンテキストメニューから[プロジェクトの追加および除去]を選択してください。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
構成プロジェクト名	WebServiceEARSample

4-2) サーバへの配備

サーバビューでサーバを選択し、コンテキストメニューから[開始]を選択してください。

サーバが始動すると自動的にEARファイルが配備されます。

サーバビューで以下の情報を確認してください。

確認項目	確認内容
サーバの状態	始動済み
サーバの状況	同期済み
EARファイル（WebServiceEARSample）の状況	同期済み
EARファイル配下のWARファイル(WebServiceSample)の状況	同期済み

4-3) WSDLのURLの確認

WSDLのURLの確認にはInterstage Application Serverの管理コンソールを使用します。

サーバビューでサーバを選択し、コンテキストメニューから[管理コンソール]を選択して、Interstage管理コンソールを起動します。

Interstage管理コンソールの使用方法については、Interstage Application ServerのJava EE実行環境の使用方法を別途ご確認ください。

管理コンソールの左側にあるツリーから、[Webサービス] > [Webサービスクラス名(PopulationRanking)]をクリックしてください。次に、右の画面に表示される [WSDL]の[表示]ボタンをクリックしてください。Interstage Application Serverによって作成されたWSDLが表示されます。

表示されたWSDLのURLを確認してください。このURLをクライアント作成時に利用します。

WSDLのURL

```
http://localhost/WebServiceSample/PopulationRankingService?wsdl
```

5) Webサービスクライアントの作成

本来であればWebアプリケーションをクライアントとして作成するための手順が必要ですが、ここではサンプルをインポートしてWebサービスクライアントとして重要なポイントを確認していきます。Webアプリケーションの作成についての詳細な手順を知りたい場合は、Webアプリケーションの"2.2 入門"を参照してください。

サンプルのインポートは次の手順で行います。
メニューから[ファイル] > [インポート]を選択します。表示されたインポートウィザードで[一般] > [既存プロジェクトをワークスペースへ]を選択します。[アーカイブファイルの選択]をチェックし、[参照]ボタンを押して、以下のファイルを選択します。

<ワークベンチのインストールフォルダ>¥sample¥WebServiceSample.zip

[プロジェクト]から[WebServiceClientSample]を選択し、[完了]をクリックしてください。
WebServiceClientSampleプロジェクトのインポート後に、WebServiceEARSampleプロジェクトを選択して、右クリックでコンテキストメニューから[プロパティ]を選択します。プロパティダイアログボックスの[Java EEモジュール依存関係]を選択し、WebServiceClientSampleをチェックしてEARにWebサービスクライアントを追加します。

5-1) クライアントプロジェクトの作成

Webサービスクライアントとして動的Webプロジェクトを作成します。サンプルのインポートで代替済みのため、詳細は省略します。設定項目は以下を参考にしてください。

設定項目	設定内容
プロジェクト名	WebServiceClientSample
ターゲットランタイム	Interstage Application Server V11.1 IJServer Cluster (Java EE)
動的 Web モジュールバージョン	2.5
構成	Interstage Application Server V11.1 IJServer Cluster (Java EE) デフォルト構成
EARにプロジェクトを追加	チェックする
EARプロジェクト名	WebServiceEARSample

5-2) WSDLからサービスエンドポイントインタフェースの作成

WSDLからWebサービスを呼び出すために必要なサービスエンドポイントインタフェースなどのファイルを作成する必要があります。インポートしたサンプルプロジェクトでは以下のように作成済みです。

[新規]ウィザードから[Webサービス] > [Webサービス(JAX-WS)]を選択し、Webサービス(JAX-WS)ウィザードでファイルを作成します。ファイルをインポートするソースフォルダと4)でInterstage Application Serverから取得したWSDLファイルのURLを設定します。

WebサービスをEARとして配備する場合、サービスエンドポイントインタフェースのパッケージを変更する必要があります。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
ソースフォルダ	WebServiceClientSample/src
WSDLファイル	Interstage Application Serverから取得したURLを設定します
自動生成されるソースのパッケージをWSDLで指定されているパッケージ以外に変更する	チェックする
パッケージ	stub

ファイル名リストにある以下のファイルがソースフォルダ配下に作成されていることを確認してください。

ファイル名	備考
PopulationRanking.java	サービスエンドポイントインタフェース
PopulationRankingService.java	サービスインタフェース
CountryData.java	データ定義クラス
GetCountryData.java	データバインディングクラス
GetCountryDataResponse.java	データバインディングクラス
ObjectFactory.java	オブジェクト生成クラス
package-info.java	パッケージ情報クラス

ポイント

WSDL定義の参照などのために、Webサービス(JAX-WS)ウィザードで指定したWSDLファイルおよび、WSDLファイルが参照するXML Schemaファイルは、参照先から自動的に取得され、プロジェクト配下の"wsdl"フォルダにコピーされます。

5-3) サーブレットクラスの作成

サーブレットクラスを作成します。インポートしたサンプルプロジェクトでは以下のような設定項目で作成済みです。

設定項目	設定内容
Java パッケージ	sample
クラス名	ServletController
スーパークラス	javax.servlet.http.HttpServlet
作成するメソッドスタブの選択	スーパークラスからのコンストラクタ 継承された抽象メソッド doPost(doGetのチェックを外す)

5-4) 入出力画面の作成

入出力画面のJSPを作成します。インポートしたサンプルプロジェクトでは以下のようなファイルを作成済みです。

入力画面(default.jsp)

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>世界総人口ランキング</title>
</head>
<body>
<h1>入力画面</h1>
<p>1～10の間で順位を入力してください。</p>
<p></p>
<form action="ServletController" method="post">
    順位：<br>
    <input name="rank" type="text" size="10">位<br>
    <p></p>
    <input type="submit" value="OK">
    <input type="reset" value="クリア">
</form>

</body>
</html>
```

出力画面(result.jsp)

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>世界総人口ランキング</title>
</head>
<body>
<h1>出力画面</h1>
```

```
総人口ランキング${param.rank}位は「${applicationScope.ranking.countryName}」です。<br>
総人口は${applicationScope.ranking.totalPopulation}人です。
<br>
<hr>
<form action="ServletController" method="post">
    <input type="submit" value="入り口に戻る">
    <input type="hidden" name="mode" value="top">
</form>

</body>
</html>
```

エラー画面(error.jsp)

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>世界総人口ランキング</title>
</head>
<body>

<h1>エラー画面</h1>
指定された範囲内の順位が入力されていないか、またはサーバエラーです。<br>
<br>
<hr>
<form action="ServletController" method="post">
    <input type="submit" value="入り口に戻る">
    <input type="hidden" name="mode" value="top">
</form>

</body>
</html>
```

5-5) Dependency Injectionの指定

Webサービスを呼び出すための実装をサーブレットクラスに行います。インポートしたサンプルプロジェクトのサーブレットソースには実装済みです。
サービスインタフェースのフィールドを作成して、以下のようにWebServiceRefアノテーションを設定し、Dependency Injectionの指定を行います。

WebServiceRefアノテーションの定義

```
@WebServiceRef(name="service/PopulationRanking")
private PopulationRankingService service;
```

ポイント

J2EE1.4までのlookupメソッドの代わりに、Java EEではDependency Injectionを利用することで、オブジェクトを取得して使用することができます。(オブジェクトは実行時に自動的にJava EEコンテナによってフィールドに設定されます。)

5-6) Webサービスの呼び出し処理の実装

Webサービスの呼び出しは、以下のようにサービスエンドポイントインタフェースにオブジェクトを取得し、インタフェース経由でメソッドを呼び出します。作成したサーブレットクラスに以下のようにWebサービスの呼び出し処理を定義します。インポートしたサンプルプロジェクトのサーブレットソースには実装済みです。

Webサービスの呼び出しメソッドの定義

```
PopulationRanking business = service.getPopulationRankingPort();
CountryData country = business.getCountryData(rank);
```

Webサービスの呼出しの実装を行ったサーブレットクラスは以下のようになります。(インポートしたプロジェクトのサーブレットソースは、この形式になっています。)

赤字の部分が実際には追加、変更する箇所になります。

サーブレットクラス(ServletController.java)

```
package sample;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletContext;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.WebServiceRef;

import stub.CountryData;
import stub.PopulationRanking;
import stub.PopulationRankingService;

/**
 * Servlet implementation class ServletController
 */
public class ServletController extends HttpServlet {

    static final long serialVersionUID = 1L;

    @WebServiceRef(name="service/PopulationRanking")
    private PopulationRankingService service;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ServletController() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{

        request.setCharacterEncoding("Windows-31J");
        ServletContext sc = getServletContext();

        // 呼び出すファイル種別取得
        String mode = request.getParameter("mode");
        if (mode != null && mode.equals("top")) {
            // 入力画面のHTMLファイル呼出し
            RequestDispatcher inRd = getServletContext().getRequestDispatcher("/default.jsp");
            inRd.forward(request, response);
            return;
        }
```

```
        int rank;

        // 入力チェック
        try{
            rank = Integer.parseInt(request.getParameter("rank"));
        } catch (NumberFormatException e){
            RequestDispatcher errRd = sc.getRequestDispatcher("/error.jsp");
            errRd.forward(request, response);
            return;
        }

        PopulationRanking business = service.getPopulationRankingPort();
        CountryData country = business.getCountryData(rank);

        //取得した値の確認
        if(country == null){
            RequestDispatcher errRd = sc.getRequestDispatcher("/error.jsp");
            errRd.forward(request, response);
            return;
        }

        sc.setAttribute("ranking", country);

        RequestDispatcher outRd = sc.getRequestDispatcher("/result.jsp");
        outRd.forward(request, response);
    }
}
```

6) アプリケーションの動作確認

6-1) プロジェクトとサーバの関連付け

クライアント作成の準備として実施しているため、実施する必要はありません。

6-2) ブレークポイントの設定

Webサービスクラスの getCountryData()メソッドの先頭行にブレークポイントを設定します。ブレークポイントは、エディタの左側のルーラー部分をダブルクリックして、設定もしくは解除します。

6-3) サーバの起動

サーバビューでサーバを選択し、コンテキストメニューから[デバッグ]を選択してください。

ポイント

- デバッガを使用せず単純にアプリケーションを起動したい場合には、コンテキストメニューから[開始]を選択します。
- サーバビューに配備先となるサーバが登録されていない場合は、サーバを追加する必要があります。サーバを追加する方法については、"6.2.7 アプリケーションの動作確認"を参照してください。

サーバが始動する前に自動的にEARファイルが配備されます。

サーバビューで以下の情報を確認してください。

確認項目	確認内容
サーバの状態	始動済み
サーバの状況	同期済み
EARファイル（WebServiceEARSample）の状況	同期済み

確認項目	確認内容
EARファイル配下のWARファイル（WebServiceSample）の状況	同期済み
EARファイル配下のWARファイル(WebServiceClientSample)の状況	同期済み

6-4) アプリケーションの実行

サーバビューで、配備されているWebプロジェクト（WebServiceClientSample）を選択して、コンテキストメニューから[Webブラウザ]を選択します。Webブラウザが起動されて、入力画面が開きます。

設定項目	設定内容
入力画面のURL	http://localhost/WebServiceClientSample/

[順位:]の入力フィールドに人口ランキングの順位を入力して、[OK]をクリックします。

ポイント

プロジェクトを選択し、コンテキストメニューから[実行] > [サーバで実行]または[デバッグ] > [サーバでデバッグ]を選択することで、サーバの追加、プロジェクトの追加、サーバの起動、クライアントの起動をまとめて行うことができます。
また、クライアントとして使用するWebブラウザは、メニュー [ウィンドウ] > [Web ブラウザ] から変更することができます。

6-5) アプリケーションのデバッグ

アプリケーションが動作し、ブレークポイントで中断されます。変数ビューの表示を確認します。メニューから[実行] > [ステップオーバ]を選択し、変数ビューでプログラムの状況を確認します。
デバッグについては、"6.2.7.1 デバッグする"を参照してください。

ポイント

変数ビューでは値の確認だけでなく、変更も行うことができます。

デバッガで中断している処理は、メニューから[実行] > [再開]を選択することで再開され、サーバ側での処理が終了し、Webブラウザに出力画面が表示されます。入力した順位の国名と総人口が表示されることを確認してください。

7) 運用環境へのアプリケーションの配布

このアプリケーションを運用環境に配布するには、EARファイルを作成する必要があります。作成したEARファイルは、サーバの配備機能を利用して、サーバに配備します。

7-1) アプリケーションのエクスポート

EARファイルの作成は、エクスポートウィザードから行います。エクスポートウィザードを起動するには、[ファイル] > [エクスポート]を選択してください。[エクスポート]ウィザードから[Java EE] > [EARファイル]を選択してください。

EARファイルエクスポートウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
EAR アプリケーション	エンタープライズアプリケーションプロジェクトを指定します。
宛先	EARファイルの作成先を指定します。

7-2) 運用環境への配布

運用環境のInterstage管理コンソールにログインし、ローカルにあるEARファイルを指定することにより、リモート環境から運用環境に配備を行うことができます。

5.3 タスク

ここでは、Interstage Studioを用いてWebサービスアプリケーションを開発する方法について、開発作業課題(タスク)ごとに説明します。

- 5.3.1 Webサービスを作成する環境を準備する
- 5.3.2 Webサービスを作成する
- 5.3.3 Stateless Session BeanをWebサービス化する
- 5.3.4 WSDLを作成する
- 5.3.5 WSDLからサービスエンドポイントインタフェースを作成する
- 5.3.6 Webサービスクライアントを作成する
- 5.3.7 Webサービスの動作を確認する
- 5.3.8 Webサービスを運用環境に配布する

5.3.1 Webサービスを作成する環境を準備する

Webサービスは、以下のようにWebアプリケーションとして作成する方法とEJBアプリケーションとして作成する方法があります。

- WebアプリケーションでWebサービスを提供する
  簡単にWebサービスを作成したい場合や、Webサービスとしてのインタフェースを厳密に定義する必要があり、WSDLでインタフェース設計を行う必要がある場合には、この方法を選択します。
  詳細は、"5.3.2 Webサービスを作成する"を参照してください。

- EJBアプリケーションでWebサービスを提供する
  提供する機能をEJBとしても利用したい場合や、アプリの構成としてEJBモジュールとして提供した方が開発や保守がしやすいなどの場合に、この方法を選択します。
  詳細は、"5.3.3 Stateless Session BeanをWebサービス化する"を参照してください。

提供する形態が決定したら、プロジェクトを作成し、必要なライブラリをプロジェクトに設定してビルドできる環境を整えます。

プロジェクトの作成

WebアプリケーションとしてWebサービスを提供する場合は、[新規プロジェクト]ウィザードから[Web] > [動的Webプロジェクト]を選択し、動的Webプロジェクトを作成します。
EJBアプリケーションとしてWebサービスを提供する場合は、[新規プロジェクト]ウィザードから[EJB] > [EJBプロジェクト]を選択し、EJBプロジェクトを作成します。

詳細については、"2.3.1 Webアプリケーションを作成する環境を準備する"または"3.3.1 EJBを作成する環境を準備する"を参照してください。

クラスパスの設定

アプリケーションを作成するのに必要なライブラリなどがある場合には、クラスパスの設定を行う必要があります。クラスパスの設定は、プロジェクトのビルドパスで行います。
ビルドパスの設定の詳細については、"6.2.2.3 クラスパスを設定する"を参照してください。

5.3.2 Webサービスを作成する

WebアプリケーションとしてWebサービスを作成するには、以下のアプローチがあります。

- JavaクラスをWebサービス化する
  最も簡単にWebサービスを作成することができます。単純にWebサービスを作成したいだけで、ほかに制約などがなければ、この方法で作成するのが簡単です。

- WSDLからWebサービスを作成する
  Webサービスとしてのインタフェースを厳密に定義する必要があり、WSDLでインタフェース設計を行う必要がある場合などに、この方法を選択します。

5.3.2.1 JavaクラスをWebサービス化する

JavaクラスをWebサービス化するための前提として、Javaクラスを作成する必要があります。もちろん、既存のJavaクラスをWebサービス化することもできます。
その後は、Javaクラスに@WebServiceアノテーションを宣言し、公開したいサービスを実装すれば、Webサービス化できます。

Javaクラスの作成

Javaクラスの作成は、[新規]ウィザードから[クラス]を選択します。特別なJavaクラスを作成する必要はなく、パッケージ、名前を指定してクラスを作成します。

WebServiceアノテーションの宣言

作成したJavaクラスに@WebServiceアノテーションを以下のように記述します。

WebServiceアノテーションの使用例

```
package sample;

import javax.jws.WebService;

@WebService
public class Calc {
	public int add(int param1,int param2) {
		return param1 + param2;
	}
}
```

サービスの実装

Webサービスとして公開したい機能のメソッドを定義し、メソッドを実装します。

5.3.2.2 WSDLからWebサービスを作成する

WSDLからWebサービスを作成するための前提として、WSDLを作成する必要があります。もちろん、既存のWSDLを利用してもかまいません。
その後は、WSDLに則したWebサービスを作成し、宣言されているメソッドの実装をすればWebサービスを作成できます。

WSDLの作成

WSDLは、[新規]ウィザードから[Webサービス] > [WSDL]を選択し、ウィザードで作成することができます。また、専用のエディタで編集することができます。
WSDLの作成、編集の詳細については、"5.3.4 WSDLを作成する"を参照してください。

WSDLに則したWebサービスの作成

WSDLに則したWebサービスを作成するためには、まず、WSDLからサービスエンドポイントインタフェースを作成する必要があります。
サービスエンドポイントインタフェースの作成は、[新規]ウィザードから[Webサービス] > [Webサービス(JAX-WS)]を選択して行うことができます。
ウィザードでの作成の詳細については、"5.3.5 WSDLからサービスエンドポイントインタフェースを作成する"を参照してください。

サービスエンドポイントインタフェースが作成できたら、次にサービスエンドポイントインタフェースを実装するクラスをJavaクラスウィザードで作成します。
[新規]ウィザードから[クラス]を選択します。サービスエンドポイントインタフェースを実装するJavaクラスを作成するため、以下を指定して生成します。

- パッケージ、名前
  作成するWebサービスのパッケージ、クラス名を指定します。

- インタフェース
  サービスエンドポイントインタフェースを指定します。

作成したJavaクラスに@WebServiceアノテーションを以下のように記述します。

WebServiceアノテーションの使用例

```
package sample;

import javax.jws.WebService;

@WebService(endpointInterface="sample.Calc", wsdlLocation="CalcService.wsdl")
public class CalcImpl implements Calc {
	public int add(int param1,int param2) {
		return param1 + param2;
	}
}
```

メソッド(サービス)の実装

宣言されているメソッドの実装をします。

5.3.3 Stateless Session BeanをWebサービス化する

Stateless Session BeanをWebサービス化するための前提として、EJBプロジェクトにStateless Session Beanを作成する必要があります。
もちろん、EJB3.0の仕様で記述されている既存のStateless Session BeanをWebサービス化することもできます。
その後は、Stateless Session Beanに@WebServiceアノテーションを宣言すれば、Webサービス化できます。

Stateless Session Beanの作成

Stateless Session BeanはEJBプロジェクトに作成します。
作成方法の詳細については、"EJB作成の準備"、"3.3.2 Session Beanを作成する"を参照してください。

WebServiceアノテーションの宣言

Stateless Session Beanに@WebServiceアノテーションを以下のように記述します。

WebServiceアノテーションの使用例

```
package sample;

import javax.jws.WebService;
import javax.ejb.Stateless;

@WebService
@Stateless
public class Calc {
	public int add(int param1,int param2) {
		return param1 + param2;
	}
}
```

5.3.4 WSDLを作成する

WSDLを作成するために、新規ウィザードと専用のエディタが提供されています。また、WSDLバリデータが用意されており、保存したときなどにWSDLファイルの妥当性がチェックされます。

WSDLウィザード

WSDLは、[新規]ウィザードから[Webサービス] > [WSDL]を選択し、ウィザードで作成することができます。ウィザードでの設定は、以下を参考にしてください。

・ ターゲット名前空間と接頭部
WSDLで使用する名前空間とそのプレフィックスを指定します。

・ プロトコルとSOAPバインディングオプション
WS-I Basic Profile に準拠した方式とする場合には、プロトコルとしてSOAPを指定し、SOAPバインディングオプションとして文書リテラルまたはRPCリテラルを指定します。
文書リテラルの方がプログラム通信に適しており、特に理由が無い場合は、より広く使用されている文書リテラルを選択することをお勧めします。
エンコードされたRPCは、WS-I Basic Profile 1.0 が策定される前に広く利用されていた方式です。以前のSOAPサービスを利用したシステムとの接続など、エンコードされたRPCでの接続が必要である場合などに選択します。

WSDLエディタ

WSDLエディタでは、[設計]タブと[ソース]タブがあります。

[ソース]タブでは、WSDLをXMLファイルとして直接編集できます。
[設計]タブでは、コンテキストメニュー、プロパティビュー、アウトラインビューを利用し、以下のWSDLを構成する要素をグラフィカルに編集することができます。

・ サービス
追加すると初期状態でポートが追加されます。

・ ポート
サービスを選択し、ポートを追加することができます。追加する際にはバインディングやプロトコルを指定します。
ポートを選択して、バインディングの設定を行えます。既存のバインディングを利用することもできますし、設定を行うときに新規バインディングを作成することもできます。

・ バインディング
ポートに関係なく、先にバインディングを作成することもできます。
バインディングを選択して、ポートタイプの設定を行えます。既存のポートタイプを利用することもできますし、設定を行うときに新規にポートタイプを作成することもできます。

・ ポートタイプ
バインディングに関係なく、先にポートタイプを作成することもできます。
ポートタイプを作成すると初期状態でオペレーションが1つ追加されています。

・ オペレーション
ポートタイプを選択して、オペレーションを追加することができます。また、オペレーションを選択してfaultを追加できます。
input,output,faultに対してメッセージを設定することができます。既存のメッセージを利用することもできますし、設定を行うときに新規にメッセージを作成することもできます。

・ メッセージ
オペレーションに関係なく、先にメッセージを作成することもできます。
メッセージを作成すると初期状態でパーツが1つ追加されています。

・ パーツ
メッセージを選択してパーツを追加することができます。
パーツを選択して、タイプや要素の設定が行えます。既存のものを利用することもできますし、設定を行うときに新規に作成することもできます。

・ インポート
インポートはアウトラインビューから追加できます。

- タイプ
  パーツの設定から行うこともできますし、アウトラインビューでタイプの配下のスキーマを開き、スキーマを編集することもできます。

WSDLバリデータ

WSDLがWS-I Basic Profile 1.1に準拠しているかをチェックすることができます。妥当性をチェックした結果は、問題ビューに表示されます。また、エディタ編集時にはエディタ上にマークが付きます。バリデータの詳細は、"6.2.6.2 検証"を参照してください。

注意

Interstage Studioで提供しているWebサービス関連のウィザードでは、上記の設定ページにあるWSDLの検証オプション([WSDL ファイルを使用するウィザードで WSDL 検証を実行するかどうか])を有効にしても、WSDL検証を行いません。

5.3.5 WSDLからサービスエンドポイントインタフェースを作成する

Webサービスを呼び出す場合や、WSDLからWebサービスを作成する場合にはサービスエンドポイントインタフェースが必要です。サービスエンドポイントインタフェースはWSDLから作成します。

サービスエンドポイントインタフェースの作成

サービスエンドポイントインタフェースなどは、[新規]ウィザードから[Webサービス] > [Webサービス(JAX-WS)]を選択し、ウィザードで作成することができます。
Webサービス(JAX-WS)ウィザードでは、以下を参考にしてください。

- WSDL
  WSDLファイルを指定します。URLまたはローカルファイルの位置を指定できます。
  Webサービスを呼び出すクライアントを作成する場合で、WSDLを開発資産として保持する必要がないなら、公開されているWSDLのURLを指定することもできます。逆にWSDLからWebサービスを作成する場合で、WSDLを開発資産として保持する場合には、ローカルファイルを指定することもできます。

ポイント

Webサービス(JAX-WS)ウィザードで指定したWSDLファイルおよび、WSDLファイルが参照するXML Schemaファイルは、参照先から自動的に取得され、プロジェクト配下の"wsdl"フォルダにコピーされます。コピーされたWSDLファイルは、参照用のため、Webサービス(JAX-WS)ウィザードで定義しないでください。

注意

以下の場合に、Webサービス(JAX-WS)ウィザードを実行して取得し、プロジェクト内にコピーされたWSDLファイルについて、WSDLの検証エラーが発生することがあります。

- Webサービス(JAX-WS)ウィザードを実行してWSDLファイルを取得した後に、Webサービスアプリケーションを更新(サービスの追加など)し、再度Webサービス(JAX-WS)ウィザードを実行して、更新されたWSDLファイルを取得した場合

上記の場合に、初めのWSDLファイル取得時に、WSDLファイルで参照されているXML Schemaファイルがワークベンチのキャッシュに登録されます。再度Webサービス(JAX-WS)ウィザードを実行して、更新されたWSDLファイルを取得しても、WSDLの検証ではキャッシュされたXML Schemaを用いて検証が行われるため、WSDLファイルとXML Schemaが不整合となり、検証エラーが発生するものです。

エラーが発生した場合は、以下の手順で回避できます。

1. ワークベンチのメニューから[ウィンドウ] > [設定]を実行して設定ダイアログボックスを開きます。
2. [設定]ダイアログボックスで、[一般] > [ネットワーク接続] > [キャッシュ]を選択して、[キャッシュ]ページを開きます。
3. [キャッシュエントリ]からWSDLファイルで参照しているXML Schemaファイルを選択して、[除去]をクリックします。
4. ワークベンチのメニューから[プロジェクト] > [クリーン]を実行します。

5. [クリーン]ダイアログボックスで、問題が発生しているプロジェクトを選択して、[OK]をクリックします。

5.3.6 Webサービスクライアントを作成する

Webサービスを呼び出すクライアントを作成するには、サービスエンドポイントインタフェースやサービスインタフェースが必要です。それらの必要なファイルはWSDLから作成することができます。
必要なファイルが揃えば、あとはDependency Injectionを定義するなどしてインタフェースにオブジェクトを対応付けて、インタフェース経由でメソッドを呼び出すことができます。

必要なファイルの作成

サービスエンドポイントインタフェース等は、[新規]ウィザードから[Webサービス] > [Webサービス(JAX-WS)]を選択し、ウィザードで作成することができます。
作成方法の詳細については、"5.3.5 WSDLからサービスエンドポイントインタフェースを作成する"を参照してください。

インタフェースへのオブジェクトの取得

Dependency Injectionによって簡単にオブジェクトを取得することができます。
Webサービスを呼び出す場合には、以下のように@WebServiceRefアノテーションを使用します。

WebServiceRefアノテーションの使用例

```
@WebServiceRef(name="service/Calc")
private CalcService service;
```

記述例	説明
service/Calc	サービスが対応付けられている名前を指定します。
CalcService	サービスインタフェースです。

注意

Dependency Injectionは、サーブレットやEJBなどJava EEコンテナで管理されているJava EEコンポーネントでしか使用することができません。
それ以外のクラスでは、JNDIのlookupによってオブジェクトを取得します。
詳細は、"10.2.2 JNDIのlookupによるオブジェクトの取得について"を参照してください。

Webサービスの呼び出し

Webサービスの呼び出しは、以下のようにサービスエンドポイントインタフェースにオブジェクトを取得し、インタフェース経由でメソッドを呼び出します。

Webサービスの呼び出し例

```
Calc port = service.getCalcPort();
int result = port.add(100,200);
```

説明項目	説明
Calc	サービスエンドポイントインタフェースです。
service	サービスインタフェース用のフィールドで、Dependency InjectionまたはJNDIのlookupでオブジェクトを取得したものが設定されています。
port.add(100,200)	サービスエンドポイントインタフェースで公開されているメソッドを呼び出しています。Calcインタフェースでは、addメソッドが公開されています。

5.3.7 Webサービスの動作を確認する

Webサービスの動作を確認するには、Webサービスアプリケーションを実行環境に配備してから実行する必要があります。これらの操作はサーバビューから行います。
操作の詳細については、"6.2.7 アプリケーションの動作確認"を参照してください。

また、動作を確認するための支援機能として以下があります。

- Webサービスエクスプローラ
  Webサービスとして公開されているメソッドの引数や復帰値がプリミティブ型などの簡単なものであれば、Webサービスを呼び出すクライアントを作成しなくても、WebサービスエクスプローラからWebサービスを呼び出し、その結果を確認することができます。
  詳細は、"5.3.7.1 WebサービスクライアントとしてWebサービスエクスプローラを使用する"を参照してください。

- TCP/IPモニタ
  TCP/IPモニタを利用することで、WebサービスクライアントとWebサービスとのSOAPデータのやり取りをモニタリングすることが可能です。
  詳細は、"5.3.7.2 TCP/IPモニタでWebサービスのメッセージを確認する"を参照してください。

5.3.7.1 WebサービスクライアントとしてWebサービスエクスプローラを使用する

Webサービスとして公開されているメソッドの引数や復帰値がプリミティブ型などの簡単なものであれば、WebサービスエクスプローラからWebサービスを呼び出し、その結果を確認することができます。

Webサービスエクスプローラの起動

Webサービスエクスプローラは、メニューから[実行] > [Webサービスエクスプローラの起動]を選択して起動します。

WSDLページの表示

Webサービスエクスプローラには、UDDI、WSIL、WSDLのページがあり、それぞれを右肩のボタンで切り替えます。WSDLページを表示するには[WSDLページ]ボタンを押してください。

Webサービスの呼び出し

WebサービスエクスプローラのWSDLページでは以下の手順でWebサービスを呼び出すことができます。

1. 左側のナビゲータでWSDLメインを選択します。
2. 右上のアクションで、WSDLのURLを指定し、[実行]ボタンを押します。
3. 左側のナビゲータにWSDLの構造が表示されるので、動作を確認したいオペレーションを選択します。
4. 右上のアクションで、オペレーションの引数を入力し、[実行]ボタンを押します。
5. 右下の状況にWebサービスを呼び出した結果が表示されます。

ポイント

右上のアクションや右下の状況でソースを選択すると、Webサービスのメッセージを直接確認することができます。

注意

WebサービスエクスプローラのUDDIページでは、UDDIレジストリを参照することができますが、Interstage Application Serverでは、レジストリ機能をサポートしていないため、通常は利用しません。

5.3.7.2 TCP/IPモニタでWebサービスのメッセージを確認する

TCP/IPモニタをWebサービスとWebサービスクライアントの間に挟むようにすることで、SOAPデータのやり取りをモニタリングすることが可能です。

TCP/IPモニタの設定

TCP/IPモニタの設定は、設定ページの[実行/デバッグ] > [TCP/IPモニタ]から行います。
設定については、以下を参考に行ってください。

- ローカルモニタポート
  クライアントからTCP/IPモニタにアクセスするポート番号を指定します。

- ホスト名
  サーバが動作しているホスト名を指定します。

- ポート
  サーバが動作しているポート番号を指定します。

TCP/IPモニタの起動

TCP/IPモニタを起動するには、[TCP/IPモニタ]設定ページで、モニタを選択して、開始ボタンを押します。

クライアント側の接続URLの変更

クライアント側は、TCP/IPモニタを通してWebサービスにアクセスする必要があるため、接続URLを変更する必要があります。
@WebServiceRefアノテーションを利用している場合には、以下のようにwsdlLocationを利用し、TCP/IPモニタのURLを指定します。

WebServiceRefアノテーションの使用例

```
@WebServiceRef(wsdlLocation="http://localhost:8888/websv/CalculateService")
private CalcService service;
```

TCP/IPモニタでのメッセージ確認

TCP/IPモニタが起動されている状態で、クライアントからTCP/IPモニタにメッセージを送信すると、その要求メッセージとそれに対する応答メッセージがTCP/IPモニタに表示されます。
メッセージをクリアするには、TCP/IPモニタの右肩のクリアボタンを押します。

WS-Iメッセージログファイルの検証

TCP/IPモニタでモニタリングしたメッセージがWS-I Basic Profileとして妥当かどうかを、以下の手順でチェックすることができます。

1. TCP/IPモニタで送信リクエストを選択します。
2. TCP/IPモニタの右肩にある、WS-Iメッセージログファイルの検証ボタンを押します。
3. 表示されたウィザードで、メッセージの保存先を指定します。
4. 必要に応じて、WSDL文書のインクルードを指定します。
5. WS-Iメッセージバリデータでメッセージの妥当性がチェックされます。

バリデータの詳細は、"6.2.6.2 検証"を参照してください。

5.3.8 Webサービスを運用環境に配布する

作成したJava EEアプリケーションを運用環境に配布するには、アーカイブファイルを作成し、管理コンソールから配備を行う必要があります。

アーカイブファイルの作成

アーカイブファイルの作成は、エクスポートウィザードから行います。
エクスポートウィザードでの操作の詳細については、"6.2.8 運用環境への配布"を参照してください。

管理コンソールでの配備

管理コンソールはリモート環境から操作できます。運用環境の管理コンソールにログインし、ローカルにある配備するファイルを指定することにより、リモート環境から運用環境に配備を行うことができます。
配備や管理コンソールの詳細については、Interstage Application ServerのJava EE実行環境の使用方法を別途ご確認ください。

注意

Webサービスクライアントは、URLでWebサービスにアクセスしています。Webサービスを運用環境に配備する場合には、Webサービスクライアント側で、アクセスするURLを変更する必要があります。URLの変更の詳細については、Interstage Application ServerのJava EE実行環境の使用方法を別途ご確認ください。

第6章 Java EE 5アプリケーション共通事項

ここでは、Java EE 5アプリケーション実行環境上で動作するアプリケーションを作成するうえでの共通事項について説明します。

6.1 概要

Java EE 5の概要とJava EE 5に共通する開発機能の概要について説明します。

6.1.1 Java EE 5とは

Java EE 5とは、Java Platform, Enterprise Edition 5の略で、旧Sun(現オラクル)が提唱する業務システム向けのJavaプラットフォームに関する規約です。Javaの標準機能セットであるJava SEに業務システムの開発に使用するServletやEJBなどのサーバアプリケーション向けのセットが追加されています。アプリケーションサーバはこの規約に従って、コンポーネント化、各種サービス、通信方法などをJava EE 5プラットフォームとして実装し、再利用可能なコンポーネントの組み合わせで多階層アプリケーションを作成するテクノロジを提供しています。

Java EE 5では、EAR(Enterprise ARchive)ファイルとしてアプリケーションを提供します。EARファイルには、WebアプリケーションやEJBなどのJava EE 5モジュールをアーカイブして組み込みます。EARファイル内のモジュールはお互いに参照関係を設定することができ、EAR化されたJava EE 5アプリケーションは、Java EE 5プラットフォームに配備することによりクラスパスなどの設定を行わなくても運用が可能となります。

J2EE1.4からの変更点

EJBやWebサービスアプリケーションを中心にアプリケーションの開発方法が大幅に変更されています。また、アノテーションを利用したDependency Injectionが利用可能になるなど、リソースやオブジェクト参照方法も大幅に変更されています。EARファイルとしては、deployment descriptor(application.xml)が必須ではなくなっています。
これにより、アプリケーションの開発が簡単にできるようになりました。

6.1.2 Java EE 5アプリケーションの開発

Java EE 5アプリケーション開発の概要を以下に示します。

Java EE 5アプリケーションの開発の準備

Java EE 5アプリケーションを開発するには、まずエンタープライズアプリケーションプロジェクトを作成する必要があります。プロジェクトを作成し、Java EE 5モジュールをプロジェクトに追加します。詳細は、"6.2.2.2 エンタープライズアプリケーションを作成する"を参照してください。

Java EE 5アプリケーションの検証

Java EE 5の仕様に沿ってアプリケーションが開発されているか検証することができます。詳細は、"6.2.6.3 Interstage Java EE検証"を参照してください。

Java EE 5アプリケーションの動作確認

作成したJava EE 5アプリケーションの動作確認はサーバビューを用いて行います。
詳細は、"6.2.7 アプリケーションの動作確認"を参照してください。

Java EE 5アプリケーションの配布

Java EE 5アプリケーションファイルを配布するためにはEARファイルにアーカイブする必要があります。アーカイブファイルの作成にはエクスポートウィザードを使用します。
詳細は、"6.2.8 運用環境への配布"を参照してください。

6.2 タスク

ここでは、Interstage Studioを用いてJava EE 5アプリケーションを開発する際の共通事項について、開発作業課題(タスク)ごとに説明します。

6.2.1 アプリケーションの動作確認を行う配備先の準備

アプリケーションの動作確認を行う配備先にはIJServerクラスタとInterstage Java EE DASサービスの2つがあります。それぞれの利点および注意点について以下に説明します。環境や要望に応じた配備先で動作確認を実施します。

注意

IJServerクラスタとInterstage Java EE DASサービスの両方を動作確認に使用した場合、以下の注意が必要となるため、お勧めしません。どちらか片方を使用して動作確認を実施してください。

- Interstage Java EE DASサービスはIJServerクラスタを起動するための必須サービスになっています。このため、Interstage Java EE DASサービスの停止/再起動などが、IJServerクラスタの動作にも影響します。

- IJServerクラスタの利点
  - リモートマシンでの動作確認が可能であり、運用環境に近い状態での動作確認ができます。
  - Interstage基盤サービスが起動していない場合、操作の初回にサービスを起動するためのユーザアカウント制御ダイアログボックスが表示されますが、それ以降のタイミングでは表示されません。

注意

IJServerクラスタを使用する場合、"6.2.1.1 IJServerクラスタ(MyDebugJEE)を作成する"を参照してIJServerクラスタを作成してください。

- Interstage Java EE DASサービスの利点
  - Interstage Application Server機能をインストールした場合、Interstage Java EE DASサービスは作成済であり、IJServerクラスタのように作成する手間がありません。
  - Interstage Java EE Node AgentサービスやIJServerクラスタを起動する必要がないため、メモリ使用量を抑えた動作確認ができます。

注意

- ご使用のオペレーティングシステムの機能により、サーバビューの[接続(デバッグ起動)/ログイン]、[接続/ログイン]、[停止]などの操作タイミングでユーザアカウント制御のダイアログボックスが表示されます。
- リモートマシンのInterstage Java EE DASサービスでの動作確認はできません。
- Interstage Java EE DASサービスを使用する場合にはIJServerクラスタ選択時に"server"を選択してください。

6.2.1.1 IJServerクラスタ(MyDebugJEE)を作成する

IJServerクラスタを作成する場合、通常はInterstage Java EE管理コンソールを使用しますが、以下のコマンドを実行することで、デバッグ用のIJServerクラスタを作成することができます。詳細は"Interstage Application Server Java EE運用ガイド"を参照してください。

```
asadmin create-cluster --newinstances MyDebugJEEInstance --user %Admin_User% --passwordfile %File_Name% MyDebugJEE
```

オプションの可変情報については、以下を参照してください。

可変情報	説明
%Admin_User%	管理ユーザID名を指定します。
%File_Name%	管理パスワードが記述されたファイルを指定する。 ・ パスワードファイルのファイルセパレータに使用可能な文字は、"/"または"¥¥"とする 例）c:/password.txt ・ パスワードファイルの書式は以下とする AS_ADMIN_PASSWORD=パスワード

注意

コマンドを実行する場合には、以下が必要になります。

- Interstage Application Server機能のインストール先のF3FMisjeeフォルダ配下のbinにパスが設定されている。
  例）c:¥Interstage¥APS¥F3FMisjee¥bin
- Interstage基盤サービスが起動されている。

6.2.2 アプリケーション作成のための準備

アプリケーションを作成するためには、開発するモジュールに応じてプロジェクトを作成する必要があります。
また、複数のモジュールで構成されるようなアプリケーションについては、エンタープライズアプリケーションとしてまとめることができます。

6.2.2.1 プロジェクトを新規に作成する

プロジェクトウィザードは、メニューから[ファイル] > [新規] > [プロジェクト]を選択して、新規プロジェクトウィザードを起動し、以下のようにモジュールに応じてプロジェクトウィザードを選択します。

カテゴリ	プロジェクト	説明
EJB	EJBプロジェクト	EJBのモジュールを作成する場合に使用します。
Java EE	アプリケーションクライアントプロジェクト	アプリケーションクライアントのモジュールを作成する場合に使用します。
	エンタープライズアプリケーションプロジェクト	EARファイルを作成する場合に使用します。EARファイルは、EJB、Webアプリケーションおよびライブラリをまとめることができます。
	ユーティリティプロジェクト	複数のモジュールで共用するライブラリを作成する場合に使用します。
JPA	JPAプロジェクト	JPAを使用したライブラリを作成する場合に使用します。
Web	動的Webプロジェクト	Webアプリケーションのモジュールを作成する場合に使用します。

プロジェクトウィザードでの指定内容については、以下を参考にしてください。

- プロジェクト名
  生成するプロジェクト名を指定します。

- プロジェクトコンテンツ
  プロジェクト資産の格納先を指定します。デフォルトでは、ワークスペースフォルダ配下になります。

- ターゲットランタイム
  Java EEのアプリケーションを動作させるランタイムを選択します。これによりランタイムのライブラリがクラスパスに設定されます。Interstage Application ServerのJava EEコンテナ(IJServerクラスタ)の場合は、[Interstage Application Server V11.1 IJServer Cluster (Java EE)]を選択します。

- モジュールバージョン
  EJBやWebアプリケーションなどの場合に、モジュールのバージョンを指定します。

- 構成
  作成するモジュールやライブラリが、準拠する規約とそのバージョンを設定します。ウィザードやエディタなどの支援機能が、規約とバージョンに準拠するように動作します。Interstage Application ServerのJava EEコンテナ(IJServerクラスタ)の場合は、[Interstage Application Server V11.1 IJServer Cluster (Java EE) デフォルト構成]を選択します。

- EARメンバシップ
  モジュールやライブラリをEARファイルにまとめる場合は、エンタープライズアプリケーションのプロジェクトを選択します。エンタープライズアプリケーションをあとで作成する場合は、ここで選択する必要はありません。

ポイント

構成で、Interstage Application Server V11.1 IJServer Cluster (Java EE) デフォルト構成を指定すると、プロジェクト作成時にInterstage Application Server固有のdeployment descriptorを生成します。このファイルについてはInterstage Application ServerのJava EE実行環境の仕様を別途ご確認ください。

注意

- [EARメンバシップ]の[EARにプロジェクトを追加]をチェックしないでEJBプロジェクトを作成すると、Session Beanウィザードのビジネスインタフェースは EJBプロジェクトに作成されます。
- [EARメンバシップ]の[EARにプロジェクトを追加]をチェックしてEJBプロジェクトを作成すると、Session Beanウィザードのビジネスインタフェースは EJBクライアントプロジェクトに作成されます。
- [EARメンバシップ]の[EARにプロジェクトを追加]をチェック、[クライアントのインタフェースとクラスを保持するためのEJBクライアントJARモジュールを作成]をチェックしないでEJBプロジェクトを作成すると、Session Beanウィザードのビジネスインタフェースは EJBプロジェクトに作成されます。

6.2.2.2 エンタープライズアプリケーションを作成する

エンタープライズアプリケーションプロジェクトでは、モジュールやライブラリをまとめてEARファイルを作成することができます。

エンタープライズアプリケーションプロジェクトウィザード固有の指定内容については、以下を参考にしてください。

- コンテンツフォルダ
  プロジェクトとして作成しているJava EE モジュール以外で、EARファイルに含めたいファイルを格納するためのフォルダの名前を指定します。

エンタープライズアプリケーションプロジェクト作成後、EARファイルに追加するJava EE モジュールを変更したい場合には、プロジェクトの[Java EE モジュール依存関係]プロパティで編集します。

ポイント

- EARファイルにすると、アプリケーションをまとめて配備できるだけでなく、EARファイル内のJava EEモジュールの依存関係を定義でき、運用環境でのクラスパスの設定などの作業を軽減できます。
- エンタープライズアプリケーションにアプリケーションクライアントを含めるとエンタープライズアプリケーションに含まれるモジュールやライブラリをクライアントスタブとしてInterstage Java EE管理コンソールからダウンロードすることができます。EJBアプリケーション開発とEJBクライアント開発を別環境で行う場合、EJBクライアント開発に必要なEJBのインタフェースを含むjarをこの機能を利用してダウンロード可能です。

6.2.2.3 クラスパスを設定する

Javaアプリケーションをビルドするためには、プロジェクトにビルドパス(クラスパス)を設定します。Java EEのモジュールやライブラリは、以下のどれかの方法でビルドパスを設定します。

ターゲットランタイム

Java EEのアプリケーションを動作させるランタイムを選択すると、そのライブラリがビルドパスに追加されます。これはプロジェクトの[ターゲットランタイム]プロパティで設定することができます。
ビルドパスにランタイムの名前でライブラリが追加されます。

エンタープライズアプリケーションのマニフェストクラスパス

EARファイルにまとめたモジュールやライブラリは、相互にクラスを参照できます。このマニフェストクラスパスは[Java EE モジュール依存関係]プロパティで設定できます。それぞれのプロジェクトの[Java EE モジュール依存関係]プロパティで参照する対象を選択できます。
ビルドパスのEARライブラリに追加されます。

WebアプリケーションのWEB-INF/lib

Webアプリケーションのモジュールは、WEB-INF/lib にライブラリを配置できます。この指定を[Java EE モジュール依存関係]プロパティで設定できます。動的Webプロジェクトの[Java EE モジュール依存関係]プロパティで参照する対象を選択できます。
ビルドパスのWebアプリケーションライブラリに追加されます。

Javaのビルドパス

プロジェクトの[Javaのビルドパス]プロパティでビルドパスを設定できます。この設定で追加したライブラリは、運用環境には反映されません。このため、アプリケーションサーバに配備する前に、運用環境のクラスパスの設定をする必要があります。

ポイント

エンタープライズアプリケーションプロジェクトの[Java EE モジュール依存関係]プロパティで動的Webプロジェクトを選択した場合は、動的Webプロジェクトの[Java EE モジュール依存関係]プロパティに[Java EE モジュール]タブと[Webライブラリ]タブの2つが表示されます。[Java EE モジュール]タブでは、エンタープライズアプリケーションのマニフェストクラスパスが設定できます。[Webライブラリ]タブでは、WebアプリケーションのWEB-INF/libが設定できます。

6.2.2.4 開発資産の無いJava EEモジュールを利用した開発をする

Java EEアプリケーションを開発する場合、他者が開発したJava EEモジュールを利用して開発する場合があります。
エンタープライズアプリケーションプロジェクトでEARファイルを作成する場合、Java EEモジュールをインポートし、Java EEモジュールのプロジェクトを作成する必要があります。
ワークベンチのメニューから[ファイル] > [インポート]を選択し、[インポート]ウィザードからインポートするファイル形式を選択します。

ポイント

インポートするJava EEモジュールにソースファイルが含まれている場合は、インポートウィザードで作成したプロジェクトのソースフォルダにソースファイルが格納されます。インポートするJava EEモジュールにソースファイルが含まれていない場合は、インポートウィザードで作成したプロジェクトにImportedClassesフォルダが作成され、ImportedClassesフォルダにクラスファイルが格納されます。

6.2.3 Javaクラスおよびインタフェースを作成する

Javaのクラスおよびインタフェースの実装は、どちらも拡張子.javaのソースファイルに記述します。これをJavaソースファイルと呼びます。ここではJavaのクラスおよびインタフェースのソースファイルの作成方法、および、その編集方法を説明します。

Javaクラスのソースファイルを新規作成する

Javaクラスのソースファイルを新規に作成するには、[新規]ウィザードから[Java] > [クラス]を選択します。新規Javaクラスウィザードが起動されるので、パッケージ名、クラス名、スーパークラス、実装するインタフェースなどを指定してJavaクラスを作成してください。

Javaインタフェースのソースファイルを新規作成する

Javaインタフェースのソースファイルを新規に作成するには、[新規]ウィザードから[Java] > [インタフェース]を選択します。新規Javaインタフェースウィザードが起動されるので、パッケージ名、クラス名、拡張するインタフェースなどを指定してJavaインタフェースを作成してください。

Javaソースファイルを編集する

Javaのソースファイルは Javaエディタを使って編集します。プロジェクトエクスプローラやナビゲータビューでJavaソースファイルを開く操作を行うと、そのファイルがJavaエディタで開かれます。Javaエディタでは以下のような編集支援を提供しています。

- 入力支援
  エディタ上で[Ctrl+Space]キーを押すと、その箇所に入力される要素の候補が表示されます。例えばクラス名の先頭数文字を入力してから[Ctrl+Space]キーを押すと、その文字列で始まるクラス名の一覧が表示されます。他にもメソッド名や変数名なども入力支援の候補となります。

- インポートの編成
  ソース内で他のクラスやインタフェースを参照する場合には、そのクラスの完全修飾名（パッケージ名を含んだ名前）をimport文に書く必要があります。Javaエディタではそのimport文の作成を支援します。[ソース] > [インポートの編成]を実行すると、ソース内に不足しているimport文が追加され、不要なimport文が削除されます。

- リファクタリング
  リファクタリングとは、プログラムの振る舞いを変えることなくソースコードの内部構造を変更することです。プログラムの構造をより簡潔で分かりやすいものに整理することで、あとの仕様変更にも柔軟に対応できるようにするのが目的です。例えばクラス名、メソッド名、変数名などを変更した場合、他のソースファイルでそれを参照している箇所も同時に変更します。他にもスーパークラスやメソッドの抽出、ファクトリやパラメタオブジェクトの導入など、多くのリファクタリングテクニックを支援しています。

- クイックフィックス
  Javaエディタではソース内のコンパイルエラーや警告となる箇所に波線が表示されます。この波線上で[Ctrl+1]キーを押すと、エラーや警告を修正方法の候補が表示されます。例えば変数名を間違えて記述した場合、正しい変数名に修正する、または、その変数の宣言を新たに追加する、といった候補が表示されます。候補のどれかを選ぶとその修正が適用されます。

6.2.4 XMLファイルを作成する

ここでは、XMLファイルの作成方法、および、その編集方法を説明します。

XMLファイルを作成する

XMLファイルを新規に作成する場合は、[ファイル]メニューの[新規]から[その他] > [XML] > [XML]を選択します。[新規XMLファイル]ウィザードが起動されますので、ウィザードページにしたがって、ファイル名、作成方法などXMLファイルの作成に必要な情報を指定してください。
XMLファイルの作成方法は、[新規XMLファイル]ウィザードの[XMLファイルの作成]ページで以下から選択してください。

- DTDファイルからXMLファイルを作成
  ワークスペースまたはXMLカタログから選択したDTDファイルの定義をもとにXMLファイルを作成します。

・ XMLスキーマファイルからXMLファイルを作成
ワークスペースまたはXMLカタログから選択したXMLスキーマファイルの定義をもとにXMLファイルを作成します。

・ XMLテンプレートからXMLファイルを作成
登録されたテンプレートをもとにXMLファイルを作成します。

XMLファイルを編集する

XMLファイルはXMLエディタを使って編集します。プロジェクトエクスプローラやナビゲータビューでXMLファイルを開く操作を行うと、そのファイルがXMLエディタで開かれます。XMLエディタでは以下のような編集支援を提供しています。

・ 入力支援
エディタ上で[Ctrl+Space]キーを押すと、その箇所に入力できる値の候補が表示されます。
編集中のXMLでDTDおよびXMLスキーマを指定している場合は、その指定にしたがって、入力できる要素名、属性名、属性値が候補として表示されます。DTDやXMLスキーマを指定していない場合は、編集中のXMLから抽出した要素名、属性名、属性値が候補として表示されます。

・ 設計ビューによる編集
[設計]ビューを使用して、XMLのデータおよび属性の値をグリッド形式のエディタで編集することができます。また、[設計]ビューのコンテキストメニューを使用して以下の編集操作が可能です。

  ー 処理命令のコンテキストメニューで、[処理命令の編集]を選択して処理命令の属性を編集することができます。

  ー DOCTYPEのコンテキストメニューで、[DOCTYPEの編集]を選択してDOCTYPEの属性を編集することができます。

  ー 要素のコンテキストメニューで、[子の追加]を選択して子要素のさらに子要素まで含めたツリー形式で要素を追加することができます。

・ クリーンアップ
[ソース]メニューの[文書のクリーンアップ]を選択すると、編集中のファイルに対して以下の操作を一括して適用できます。フォーマットの形式は、[ウィンドウ]メニューの[設定]で[XML]>[XMLファイル]>[エディタ]にしたがいます。

  ー 空要素タグの圧縮

  ー 必須属性の挿入

  ー 欠落タグの挿入

  ー 属性値を引用符で囲む

  ー ソースのフォーマット

  ー 行区切り文字の変換

・ クイックフィックス
XMLエディタではソース内のコンパイルエラーや警告となる箇所に波線が表示されます。この波線上で[Ctrl+1]キーを押すと、エラーや警告を修正方法の候補が表示されます。閉じタグがない場合には、次の子要素の前に終了タグを挿入する、または、要素の最後に終了タグを挿入する、といった候補が表示されます。候補から修正を選択するとその修正が適用されます。

6.2.5 プロパティファイルを作成する

ここでは、プロパティファイルの作成方法、および、その編集方法を説明します。

プロパティファイルを作成する

プロパティファイルを新規に作成する場合は、[ファイル]メニューの[新規] > [ファイル]を選択します。[新規ファイル]ウィザードが起動されるので、ウィザードページにしたがって、親フォルダとファイル名を指定します。ファイルの拡張子には".properties"を指定してください。

プロパティファイルを編集する

プロパティファイルは、プロパティファイルエディタまたはプロパティファイルエディタ(ShiftJIS)を使って編集します。プロジェクトエクスプローラやナビゲータビューでプロパティファイルを開く操作を行うと、標準ではプロパティファイルエディタ(ShiftJIS)で開かれます。

プロパティファイルエディタ(ShiftJIS)とは

"プロパティファイルエディタ(ShiftJIS)"は、Unicodeエスケープ形式のファイルを読み込み、日本語で表示し、日本語での編集機能を提供します。また、保存時は自動的にUnicodeエスケープ形式に変換します。
たとえば、"富士通"という文字列は次のように表示し、保存します。

エディタの表示	ファイルの内容
COMPANY_NAME = 富士通	COMPANY_NAME = ¥u5bcc¥u58eb¥u901a

ポイント

値の先頭に空白を含めたい場合は、空白を"¥ "と記述してください。
エディタのテキスト表示に使用するフォントの指定により、"¥"は"＼"で表示されます。

注意

- プロパティファイルエディタ(ShiftJIS)の変換機能は、"Javaプロパティファイル"コンテンツタイプに定義されているファイルタイプにだけ有効です。コンテンツタイプに合致しないファイルを開いた場合は、Unicodeエスケープ形式の文字列は日本語に変換されません。
製品インストール直後は、"Javaプロパティファイル"コンテンツタイプには"*.properties"が登録されています。"Javaプロパティファイル"コンテンツタイプにファイルタイプを追加する手順は以下のとおりです。

  1. メニューバーの[ウィンドウ] > [設定]を選択します。[設定]ダイアログボックスが表示されます。
  2. [一般] > [コンテンツタイプ]を選択して、[コンテンツタイプ]ページを表示します。
  3. [コンテンツタイプ]から、[テキスト] > [Javaプロパティファイル]を選択します。
  4. [ファイルの関連付け]の[追加]をクリックします。
  5. [新規ファイルタイプ]ダイアログボックスが表示されるので、[ファイルタイプ]に追加したいファイルタイプを指定し、[OK]をクリックします。

- プロパティファイルエディタ(ShiftJIS)で表示中のファイルを、テキストエディタや比較エディタなどの他のエディタで表示した場合、日本語に変換して表示されます。Unicodeエスケープ形式のまま表示したい場合は、該当のファイルを開いているすべてのエディタを閉じてから、目的のエディタで開き直してください。
- プロパティファイルエディタ(ShiftJIS)で、1行の長さが1000文字を超えるようなファイルを編集する際に、エディタの動作が遅くなることがあります。
このような場合、複数の行に分割して1行の長さを短くしてください。行末にバックスラッシュ(¥)を記述することにより、次の行を継続行とすることができます。なお、継続行の先頭にある空白は無視されます。

プロパティファイルエディタ(ShiftJIS)のオプション

プロパティファイルエディタ(ShiftJIS)のオプションは、[ウィンドウ] > [設定] > [Java] > [プロパティファイルエディタ]で設定します。

プロパティファイルについて

プロパティファイルは、Javaプログラムで使用する定数やファイルパスといった文字列を、"キー=値"という形式で記述したファイルです。文字列を外部化すると、Javaプログラムを再ビルドすることなく値を変更することが可能となり、汎用性の高いプログラムを作成することや、国際化されたプログラムを作成することができます。

プロパティファイルはUnicodeエスケープ形式で記述する必要があるため、日本語は¥uXXXXというエスケープシーケンスによって表現した文字列となります。XXXX は、UTF-16コード単位の値を表す4桁の16進数字です。通常はShift_JISコードで一度作成したプロパティファイルを、native2asciiツールを使用して¥uXXXX形式に変換します。

プロパティファイルの詳細は、java.util.PropertiesクラスのAPI仕様で確認してください。
Javaプログラムの国際化については、java.util.ResourceBundleクラスのAPI仕様で確認してください。
JDTには、文字列の外部化をガイドするツール、外部化されていない文字列を検索するツールが用意されています。
詳細は、"Java開発ユーザガイド"の"ストリングの外部化"を参照してください。

6.2.6 問題の検出と修正

JavaファイルやJSPファイルなどのソースファイルの中に、エラーや警告などの問題がある場合は、その部分にマーカが設定されます。そのため、問題は、エディタのルーラー上のアイコンやソースが強調表示されることで確認できます。また、問題ビューにも一覧表示されます。マーカの情報として問題の内容が表示されているので、必要に応じて修正してください。

エラーや警告などの問題は、さまざまな機能で検出されます。問題を検出する機能を以下に示します。

- Javaコンパイラ
  Javaの仕様に違反しているコーディングミスだけでなく、Javaの仕様として問題は無いがトラブルの原因になる可能性がある記述をチェックすることができます。

- 検証
  ファイルやアプリケーション単位に規約などに準拠しているかをチェックします。

- Interstage Java EE検証
  Java EE アプリケーションとしての妥当性を検査します。

- FindBugs
  バグの存在する可能性があるコードを指摘します。

6.2.6.1 Javaコンパイラ

Javaの仕様に違反しているコーディングミスだけでなく、Javaの仕様として問題は無いがトラブルの原因になる可能性がある記述をチェックすることができます。

以下にチェックする項目の概要を示します。

- コードスタイル
- 潜在的なプログラミングの問題
- 名前のシャドーイングおよび競合
- 使用すべきではない制限されたAPI
- 不要なコード
- 総称型
- 注釈

上記のカテゴリに属する細かいチェック項目があり、そのチェックレベルをカスタマイズすることができます。カスタマイズは、設定ページの[Java] > [コンパイラ]、またはその配下の設定ページから行います。(プロジェクト固有の設定を行う場合には、プロジェクトプロパティの[Javaコンパイラ]で行います。)

6.2.6.2 検証

以下のようにファイルやアプリケーションの妥当性を検証するバリデータが提供されており、ファイル単位やプロジェクト単位にチェックすることができます。

バリデータ	チェック内容
HTML構文バリデータ	HTMLファイルの基本構文をチェックします。
JavaScript 構文検証	HTML/JSPファイルに記述したJavaScriptの構文をチェックします。
JPAバリデータ	JPAアプリケーションとしての妥当性をチェックします。

バリデータ	チェック内容
JSP構文バリデータ	JSPファイルをJavaコードに変換してから、そのJavaコードにコンパイルエラーが無いかを検査することでJSPファイルの構文をチェックします。
JSP内容バリデータ	JSPのELやディレクティブなどをチェックします。
XMLバリデータ	XMLファイルが整形式であること、および妥当であることをチェックします。

バリデータは以下を契機に動作します。

- ファイルの保存時
  自動ビルドがONの場合にビルダとして実行されます。

- エディタの編集時
  ファイル単位にチェック可能なバリデータは、エディタで編集中にもチェックを行います。

- メニューからの実行時
  リソース(ファイルやプロジェクトなど)を選択して、コンテキストメニューから[検証]を選択すると実行されます。

検証の設定

検証の動作をカスタマイズすることができます。カスタマイズは、設定ページの[検証]で行います。(プロジェクト固有の設定を行う場合には、プロジェクトプロパティの[検証]で行います。)
カスタマイズは以下を参考に行ってください。

- すべてのバリデータを中断
  すべてのバリデータを一時的に実行しない場合に指定します。

- 手操作/ビルド
  メニューから実行する場合(手操作)と、ビルドの場合に有効にするバリデータを指定することができます。

6.2.6.3 Interstage Java EE検証

以下のように、Java EE アプリケーションとしての妥当性をアーカイブファイル単位に検証することができます。

- 配備前(デフォルト)
  動作確認などのために、Interstage StudioからInterstage Application Serverに配備する前に妥当性をチェックします。

- メニューからの実行
  プロジェクトを選択して、コンテキストメニューから[Interstage Java EE検証] > [実行]を選択することで妥当性をチェックします。アプリケーションの作成途中などにチェックしたい場合に利用してください。

- ビルド
  ビルド時にInterstage Java EE検証機能を実行することができます。ただし、自動ビルドがONの場合にはファイルの保存のたびに、Java EE アプリケーションとしてアーカイブファイル全体のチェックが行われてしまい効率的ではありません。
  そのため、個々のJava EEモジュールの開発時に開発者が利用することはお勧めしません。アプリケーションを一括してビルドするなどの際に、まとめて検証も実行するために利用してください。

ポイント

エンタープライズアプリケーションプロジェクトをチェックすると中身のモジュールまでチェックされてしまいます。ワークスペースでエンタープライズアプリケーションとその中身のモジュールを両方作成しており、ビルド時にチェックする場合には、どちらかのみを選択してチェックが重複しないようにしてください。

Interstage Java EE検証の設定

Interstage Java EE検証の動作をカスタマイズすることができます。カスタマイズは、設定ページの[Interstage Java EE検証]で行います。
(プロジェクト固有の設定を行う場合には、プロジェクトプロパティの[Interstage Java EE 検証]で行います。)
カスタマイズは以下を参考に行ってください。

- JSPをチェックする
  検証のバリデータと重複し、検証バリデータの方が問題箇所を特定しやすいためデフォルトではチェックを行っていません。ただ、Java EEアプリケーションの検証では、Interstage Application ServerのJSPコンパイラを使ってチェックするため、Interstage Application Serverで動作するアプリケーションを厳密にチェックしたい場合には、HTML/JSP構文バリデータの代わりに使用することもできます。

- 配備前にチェックする
  サーバビューから配備する前に妥当性をチェックします。アプリケーションとしての構成がほぼ決定し、実装部分の障害修正のみの段階に入った場合など、動作確認前に妥当性を検査する必要性が低くなってきた場合にはチェックをはずしてください。

- 差分ビルドではチェックしない
  ビルド時にチェックを行う場合には、デフォルトでは差分ビルド時はチェックしないようになっています。差分ビルド時にも、アーカイブファイル単位にチェックが毎回動作してもいいという場合にはチェックをはずしてください。

6.2.6.4 FindBugs

以下のように、バグの存在する可能性があるコードを指摘するFindBugsを使用することができます。

- メニューからの実行
  リソース(プロジェクトやファイル)を選択して、コンテキストメニューから[FindBugs] > [バグをスキャン]を選択することでFindBugsを実行します。

- ビルド
  ビルド時にFindBugsを実行することができます。(デフォルトではビルド時にFindBugsを実行する設定になっていません。)

ポイント

複数プロジェクトを選択して、コンテキストメニューから[FindBugs] > ["自動的に実行"を設定]を選択することで、まとめてビルダとして実行するように設定することもできます。

検出された問題は、以下の形式で問題ビューに表示されます。

〈警告優先度〉〈バグカテゴリ〉〈バグパターン〉：〈メッセージ〉

- 警告優先度
  High、Medium、Lowの頭文字で表されます。[レポートする最小の優先度]で、検出レベルをカスタマイズすることもできます。

- バグカテゴリ
  バグカテゴリの省略形(一文字の英字)で表されます。チェックするかをバグカテゴリ単位に指定することができます。バグカテゴリと省略形の対応付けについては、"チェック内容"を参照してください。

- バグパターン
  バグパターンの省略形(数文字の英数字)で表されます。フィルタファイルに記載することでチェックの動作をカスタマイズすることができます。

また、以下のように、FindBugsで検出された問題を表示するためのビューやパースペクティブも用意されています。

- FindBugsパースペクティブ
  検出されたバグになる可能性があるコードを修正するためのパースペクティブです。バグ詳細ビュー、バグツリー ビュー、バグのユーザ注釈ビューなどで構成されています。

- バグ詳細ビュー
  検出された問題の詳細情報を表示します。
  詳細情報としては、警告優先度、バグカテゴリ、問題箇所のクラス、メソッド名、フィールド、ソースファイル名および行番号、問題の概要と詳細内容が表示されます。

- バグツリー ビュー
  検出された問題をツリー形式で表示します。
  バグツリー ビューでは、プロジェクト単位にタブが割り当てられており、検出された問題は、バグパターン単位にタブの中にツリー形式で表示されます。バグツリー ビュー上で選択した問題の詳細が、バグ詳細ビューに表示されます。また、バグツリー ビューの問題をダブルクリックすると、問題ビューと同様に問題が発生している箇所をエディタ領域に表示します。

- バグのユーザ注釈ビュー
  FindBugsで検出された個々の問題について、ユーザ注釈を記述することができます。
  注釈としては、分類の選択、任意のコメントを指定できます。また、問題が検出された日時もビューに表示されます。

チェック内容

FindBugsでは、以下のようなチェックを行っています。

バグカテゴリ	省略形	概要
実行効率	P	必ずしも間違いではないが、実行効率が悪くなる可能性のあるもの。例えば、以下のようなものを検出します。 ・ URLクラスの非効率な使い方 ・ Stringクラスの非効率な使い方 ・ valueOfメソッドの非効率な使い方 ・ ボクシング/アンボクシングの非効率な使い方 ・ 使用されていないフィールド、メソッド宣言
正確性	C	明らかなコーディングミスによって、開発者の意図しないコードになっているもの。例えば、以下のようなものを検出します。 ・ equalsメソッドの誤使用 ・ 明らかな無限ループ ・ フィールドやメソッドなど命名に関する問題 ・ nullポインタ
国際化	I	国際化、ロケールの扱いに対するコードに欠陥があるもの。例えば、以下のようなものを検出します。 ・ Localeを引数に持つメソッドを利用していない
マルチスレッド環境での正確性	M	スレッド、ロック、volatileの扱いに対するコードに欠陥があるもの。
良くない習慣	B	推奨される、あるいは本質的なコーディング規範から逸脱しているもの。例えば、以下のようなものを検出します。 ・ 例外の扱い方 ・ 文字列の不正な比較 ・ finalizerの誤使用 ・ equalsメソッドの使い方 ・ フィールドやメソッドなどのまぎらわしい名前 ・ 直列化の問題

バグカテゴリ	省略形	概要
脆弱性を持つコード	V	信頼されていないコードからの攻撃に対して脆弱性を持つコード。例えば、以下のようなものを検出します。 ・ 内部表現が流出する可能性があるもの ・ privateやfinalなど修飾子に関するもの
回避可能	D	誤解を与えやすいコード、変則的、間違いを招きやすいコード。例えば、以下のようなものを検出します。 ・ 変数への値の設定に関するもの ・ キャストに関するもの

上記のバグカテゴリ単位にチェックする/しないをカスタマイズすることができます。

FindBugsの設定

FindBugsの動作をカスタマイズすることができます。カスタマイズは、設定ページの[FindBugs]で行います。(プロジェクト固有の設定を行う場合には、プロジェクトプロパティの[FindBugs]で行います。)

カスタマイズは以下を参考に行ってください。

- レポートする最小の優先度
  指定した検出レベル以上の問題が検出されます。

- 除外フィルタファイル
  追加したフィルタファイルの設定でバグ検出を除外します。

プロジェクト固有の設定項目については、以下を参考に行ってください。

- FindBugsを自動的に実行
  ビルド時にFindBugsを実行する場合に指定します。

ポイント

複数プロジェクトを選択して、コンテキストメニューから[FindBugs] > ["自動的に実行"を設定]を選択することで、まとめてビルダとして実行するように設定することもできます。

FindBugsのフィルタファイルの書式

FindBugsでは、チェックした内容を問題として扱うかをフィルタファイルで制御することができます。例えば、FindBugsによって問題として検出されたが、コードを確認したところ問題ないことが判明した場合には、除外フィルタファイルを設定することで、以降に問題として検出されないようにすることができます。

以下にフィルタファイルで使用するタグについて説明します。

タグ	属性	説明
FindBugsFilter	―	フィルタファイルのトップレベルの要素です。
Match	―	クラスを特定するための要素です。
	class	クラス名を指定します。
	classregex	クラス名を正規表現で指定します。
BugCode	―	バグパターンの省略形を指定するための要素です。バグパターンの省略形は問題メッセージの3番目に表示されるアルファベットの組合せです。
	name	バグパターンの省略形をコンマで区切って指定します。
Priority	―	優先度を指定するための要素です。

タグ	属性	説明
	value	以下の値を1つ指定します。 1:高い 2:普通 3:低い
Method	－	メソッドを指定するための要素です。params属性とreturns属性は必須ではありませんが、指定する場合は両方指定する必要があります。
	name	メソッド名を指定します。
	params	引数の型の完全修飾名をコンマで区切って指定します。
	returns	復帰値の型の完全修飾名をコンマで区切って指定します。
Or	－	OR論理演算子の役目をする要素です。

実際の除外フィルタファイルの例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<FindBugsFilter>
    <!-- ClassNotToBeAnalyzedクラスに関する問題を検出しないようにします -->
    <Match class="com.foobar.ClassNotToBeAnalyzed" />

    <!-- ClassWithSomeBugsMatchedクラスに関する以下の問題を検出しないようにします
      DE : 例外を見落としている、または無視しています。
      UrF: 参照しないフィールドがあります。
     -->
    <Match class="com.foobar.ClassWithSomeBugsMatched">
      <BugCode name="DE,UrF" />
    </Match>

    <!-- すべてのクラスに対してSQLの問題を検出しないようにします -->
    <Match classregex=".*" >
      <BugCode name="SQL" />
    </Match>

    <!-- AnotherClassクラスのnonOverloadedMethod, frob, blatメソッドに対してフィールドの二重チェックの問題を検出しないようにします -->
    <Match class="com.foobar.AnotherClass">
      <Or>
        <Method name="nonOverloadedMethod" />
        <Method name="frob" params="int,java.lang.String" returns="void" />
        <Method name="blat" params="" returns="boolean" />
      </Or>
      <BugCode name="DC" />
    </Match>

   <!-- MyClassクラスのsomeMethodに対して優先度2(普通)の意味の無いローカル変数への代入の問題を検出しないようにします -->
   <Match class="com.foobar.MyClass">
     <Method name="someMethod"/>
     <BugCode name="DLS "/>
     <Priority value="2"/>
   </Match>

</FindBugsFilter>
```

6.2.7 アプリケーションの動作確認

Java EEアプリケーションを動作確認するには、Java EEアプリケーションをサーバに配備し、サーバを起動します。これらの操作はサーバビューで行います。サーバを起動すると、Java EEアプリケーションは、クライアントからの呼び出しを待ちます。クライアントからJava EEアプリケーションを呼び出すことで動作確認を行います。
また、プログラムの実行を中断してコードを1行ずつ実行したり、変数の値を確認したりするには、デバッグを行います。

サーバを操作するための準備をする

サーバの起動や停止といった操作をワークベンチで行うためには、サーバビューに操作対象となるサーバを追加する必要があります。また、サーバに配備するアプリケーションを指定するなどの準備が必要になります。

サーバを追加する

動作確認を行うアプリケーションの配備先となるサーバをサーバビューに追加します。サーバの追加には新規サーバウィザードを使用します。サーバビューを右クリックし、コンテキストメニューから[新規] > [サーバ] を選択してください。ウィザードの設定項目については以下の内容を参考にしてください。

- サーバのホスト名
  サーバがあるホストの名前を指定します。ローカルマシンの場合は"localhost"を指定します。

- サーバのタイプ
  サーバのタイプを選びます。InterstageのJava EEコンテナの場合は[FUJITSU LIMITED] > [Interstage Application Server V11.1 IJServerクラスタ (Java EE)]を選択してください。

- サーバ名
  サーバ名をカスタマイズできますが、Interstage Studioでは、ここでの変更は反映されず、以下の形式となります。
  「IJServerクラスタ名 [サーバタイプ名] (ホスト名)」

- サーバランタイム環境
  サーバのタイプに対応するランタイム設定を指定します。InterstageのJava EEコンテナの場合は[Interstage Application Server V11.1 IJServer Cluster (Java EE)]を選択してください。

Interstage Application Serverの場合は、続けて以下の設定項目が表示されます。

- ターゲットとの接続にHTTPS通信を使用する
  SSL暗号化通信を使用する場合にチェックします。サーバの設定に合わせる必要があります。
  サーバの設定はInterstage Java EE管理コンソールのHTTPリスナーのセキュリティプロパティの値を参照してください。

- Interstage JMX サービスのポート番号
  管理コンソールのポート番号に指定する値と同じ値を指定してください。ポート番号を指定したら[ログイン]をクリックして、サーバにログインできるか確認してください。ログインできると[次へ]が有効になります。管理者権限のないユーザの場合は、認証ダイアログボックスが表示されるので、管理者権限を持つユーザIDおよびパスワードを設定してください。

- 管理コンソールのポート番号
  Interstage Java EE管理コンソールのポート番号を指定します。デフォルトのポート番号は12001です。サーバの設定に合わせる必要があります。
  サーバの設定はInterstage Java EE管理コンソールの管理サーバポートの値を参照してください。

- IJServerクラスタの選択
  サーバビューで操作するIJServerクラスタを選択します。

- HTTPポート番号、デバッグポート番号
  デフォルト値が表示されるので、必要に応じて変更してください。

- ワークスペース終了時にIJServerクラスタを停止する
  ローカルマシンにあるIJServerクラスタの場合にはこのチェックボックスが選択可能になります（IJServerクラスタとしてserverを選択された場合は、選択可能になりません）。チェックするとワークスペース終了時にそのIJServerクラスタを停止する処理が行われます。なお、リモートマシンにあるIJServerクラスタについてはワークスペース終了時の停止処理は行われません。このため、ウィザードでリモートマシンを指定した場合にはこのチェックボックスは選択不可となります。

ポイント

- ローカルマシンでは動作確認にInterstage Java EE DASサービスを使用することができます。Interstage Java EE DASサービスを使用する場合にはIJServerクラスタ選択時に"server"を選択してください。
- サーバの追加は、新規ウィザードからも行うことができます。

注意

- Interstage Java EE DASサービスとIJServerクラスタの利点、注意点については"6.2.1 アプリケーションの動作確認を行う配備先の準備"を参照してください。
- IJServer(J2EE)の場合には、[ターゲットとの接続にHTTPS通信を使用する]ではなく、[管理コンソールへの接続にHTTPS通信を使用する]になり、指定した場合には管理コンソールへの接続のみがSSL通信の対象となります。
- ウィザードの入力項目は指定したホスト、IJServerクラスタの種別によってマスクされることがあります。
- サーバのタイプにInterstage Application Server V9.3以前の値を指定した場合は、以下の値を指定してください。
  - Interstage JMX サービスのポート番号
    Interstage Java EE管理コンソールのポート番号を指定します。デフォルトのポート番号は8919です。
  - 管理コンソールのポート番号
    Interstage 管理コンソールのポート番号を指定します。デフォルトのポート番号は12000です。

サーバに配備するプロジェクトを指定する

サーバに配備するプロジェクトの指定は、プロジェクトをサーバビューのサーバに追加することによって行います。サーバに配備するプロジェクトを追加するには、プロジェクトの追加および除去ウィザードを使用します。プロジェクトの追加および除去ウィザードを起動するには、サーバビューでサーバを選択し、コンテキストメニューから[プロジェクトの追加および除去]を選択してください。
新規サーバウィザードの最後のページでプロジェクトを追加することもできます。

ポイント

手動でアプリケーションを配備するには、サーバを選択し、コンテキストメニューより[公開]を選択します。[クリーン]を用いると、配備されているアプリケーションを一度すべて破棄し、配備しなおします。アプリケーションがサーバと同期している場合は、状況に「同期済み」と表示されます。同期していない場合は、「再公開」と表示されます。

サーバを起動する

サーバビューでサーバを選択し、コンテキストメニューから[開始]または[デバッグ]を選択します。サーバを起動するときにアプリケーションを自動的に配備することができます。デフォルトでは、サーバ起動時に自動的に配備する設定になっています。

ポイント

- IJServerクラスタへの接続がまだ行われていない場合には、サーバビューでのIJServerクラスタの[状態]欄に何も表示されず、起動や停止といった操作が行えません。そのような場合にはサーバビューでそのIJServerクラスタを選択し、コンテキストメニューから[接続/ログイン]を選択してIJServerクラスタへの接続を行ってください。
- IJServerクラスタへの接続がされている状態で、サーバのInterstage基盤サービスが停止された場合は、IJServerクラスタへの接続が切断されるため、サーバビューでのIJServerクラスタの操作に失敗します。操作に失敗した場合は、サーバビューでIJServerクラスタを選択し、コンテキストメニューから[最新表示]を選択してIJServerクラスタの状態を更新してから、コンテキストメニューの[接続/ログイン]を選択してIJServerクラスタへの再接続を行ってください。
- プロジェクトを選択し、コンテキストメニューから[実行] > [サーバで実行]または[デバッグ] > [サーバでデバッグ]を選択することで、サーバの追加、プロジェクトの追加、サーバの起動、クライアントの起動をまとめて行うことができます。

- サーバ起動時に自動的に配備しないようにするには、設定ページの[サーバ] > [起動]を選択し、[サーバの始動時に自動的に公開]をオフにします。

注意

サーバをデバッグモードで起動している状態で、プロジェクトのコンテキストメニューから[実行] > [サーバで実行]または[デバッグ] > [サーバでデバッグ]を選択する場合は、以下の点に注意してください。

- [実行] > [サーバで実行]を選択する場合はデバッグモードを解除する必要があります。実行前にサーバビューでサーバを再始動または停止してください。
- [デバッグ] > [サーバでデバッグ]を選択すると、[サーバでデバッグ]ダイアログボックスが表示されます。[現行モードで続行]を選び、[OK]を選択してください。

サーバを停止する

サーバを停止するには、サーバビューでサーバを選択して、ツールバーの[停止]を選択します。

6.2.7.1 デバッグする

ここでは、プログラムにおける論理エラーの検出に用いられるデバッグについて説明します。
アプリケーションをデバッグするには、ブレークポイントを設定して、起動されたプログラムを中断し、コードを1行ずつ実行し、変数の内容を確認することによって行います。

ブレークポイント

ブレークポイントを設定した行でプログラムの実行が中断します。実行が中断されると、デバッグパースペクティブを開くかどうかの確認ダイアログボックスが表示されます。デバッグパースペクティブは、デバッグする際に利用するデバッグビュー、変数ビュー、ブレークポイントビューなどがレイアウトされています。

- ブレークポイントを設定・解除する
  ブレークポイントの設定と解除は、エディタのルーラー部分をダブルクリックして切り替えます。また、追加されているブレークポイントの確認や、ブレークポイントの削除をブレークポイントビューで行うこともできます。

- ブレークポイントを無効にする
  ブレークポイントを一時的に無効にするには、ルーラーのコンテキストメニューで[ブレークポイントを使用不可にする]を選択するか、ブレークポイントビューでチェックを外します。また、すべてのブレークポイントを一時的にスキップするようにするには、ブレークポイントビューのツールバー、または、メニューバーの[実行]で、[すべてのブレークポイントをスキップ]を選択します。

注意

JSPファイルにブレークポイントを設定した場合、異なるフォルダにある同名のJSPファイルでも実行が中断します。

実行制御

実行を中断したプログラムを再度実行する方法には、ステップオーバ、ステップイン、ステップリターン、再開などがあります。これらのコマンドは、デバッグビューでスタックフレームを選択し、デバッグビューのツールバー、コンテキストメニュー、または、メニューバーの[実行]から行います。

- ステップオーバ
  現在選択されている行が実行され、次の実行可能行で中断されます。

- ステップイン
  現在選択されている行の、次に実行する必要のある式が呼び出され、呼び出されたメソッドの次の実行可能行で実行が中断されます。

- ステップリターン
  現在のメソッドの次の return ステートメントが実行されるまで再開され、次の実行可能行で中断されます。

- 再開
  ブレークポイントに到達するまで実行を継続します。

変数の値の確認と変更

スタックフレームを選択すると、そのスタックフレームで可視になっている変数を変数ビューに表示できます。
変数ビューには、プリミティブ型の値が表示されます。複雑な変数は、それを展開してそのメンバを表示すれば検査できます。値の変更は、コンテキストメニューの[値の変更]から行います。

ポイント

変数の値を参照するなどのデバッグを行うには、コンパイルされたクラスファイルにデバッグ情報を追加する必要があります。デフォルトでは、デバッグに必要なデバッグ情報はクラスファイルに追加する設定になっています。運用環境に配布する場合などで、デバッグを行う必要が無い場合はデバッグ情報を追加せずに、クラスファイルのサイズを小さくすることができます。

例外がスローされたときなどに出力するスタックトレースでは、ソースファイル名の表示と行番号の表示に、デバッグ情報を使います。行番号属性とソースファイル名はクラスファイルに追加することをお勧めします。

6.2.7.2 管理コンソールを起動する

サーバビューのコンテキストメニューで[管理コンソール]を選択すると、エディタ領域に管理コンソールが表示されます。
管理コンソールの表示には、内部Webビューアが利用されます。内部Webビューアは、Webサーバ上のページを表示するために使用される組み込みのブラウザです。
内部Webビューアには、アドレスバーとツールバーが用意されています。

アドレスバー

アドレスバーには、表示するページのURLを入力する[アドレス]フィールドと以下のボタンが表示されます。

ボタン	コマンド	説明
	移動	入力したURLのページに移動します。

ツールバー

ツールバーには、以下のボタンが表示されます。

ボタン	コマンド	説明
	戻る	内部Webビューアで直前に表示していたページに戻ります。
	進む	内部Webビューアで[戻る]をクリックする前に表示していたページを再び表示します。
	停止	内部Webビューアで表示するページのロードを停止します。
	最新表示	内部Webビューアで表示しているページを再ロードします。
	文字のサイズ	内部Webビューアで表示する文字の大きさを変更します。文字の大きさは、[最大]、[大]、[中]、[小]、[最小]の5種類の中から選択できます。

注意

- 内部Webビューアで管理コンソールを表示する場合、アドレスバーは表示されません。
- サーバのInterstage基盤サービスが停止されている場合には、コンテキストメニューの[管理コンソール]を選択しても、管理コンソールの表示に失敗します。その場合は、IJServerクラスタへの接続が切断されているため、IJServerクラスタへ再接続する必要があり

ます。再接続するには、サーバビューでIJServerクラスタを選択し、コンテキストメニューから[最新表示]を選択してIJServerクラスタの状態を更新してから、コンテキストメニューの[接続/ログイン]を選択してください。再接続が完了してから、再度、[管理コンソール]を実行してください。

6.2.8 運用環境への配布

Java EEアプリケーションを運用環境に配布するには、アーカイブファイルを作成する必要があります。作成したアーカイブファイルは、サーバの配備機能を利用して、サーバに配備します。

アーカイブファイルの作成

アーカイブファイルを作成するにはエクスポートウィザードを利用します。

エクスポート

1. エクスポートウィザードの起動
   [ファイル] > [エクスポート]を選択します。
2. 設定項目の入力
   - エクスポート先

プロジェクト	エクスポート先
EJBプロジェクト	[EJB] > [EJB JARファイル]
エンタープライズアプリケーションプロジェクト	[Java EE] > [EARファイル]
アプリケーションクライアントプロジェクト	エンタープライズアプリケーションプロジェクトで作成するアプリケーションに含まれるため、EARとしてエクスポートします。単体でエクスポートする場合には以下を使用します。 [Java EE] > [アプリケーションクライアントJARファイル]
動的Webプロジェクト	[Web] > [WARファイル]
JPAプロジェクト	エンタープライズアプリケーションプロジェクトまたは動的Webプロジェクトで作成するアプリケーションに含まれるため、EARまたはWARとしてエクスポートします。単体でエクスポートする場合には以下を使用します。 [Java] > [JARファイル]

   - モジュール
     エクスポートするプロジェクトを指定します。
   - 宛先
     アーカイブファイルの作成先を指定します。

ポイント

- エンタープライズアプリケーションプロジェクトをエクスポートすると、それに含まれている各プロジェクトのアーカイブファイルを含んだEARファイルが作成されるため、個々にアーカイブファイルを作成する必要はありません。
- ビルドパスに含めているJARファイルのうち、EARに入れないJARファイルは、サーバのクラスパスに設定します。

サーバへの配備

サーバへの配備は、サーバの配備機能を利用します。
配備の詳細については、Interstage Application ServerのJava EE実行環境の使用方法を別途ご確認ください。

6.2.9 Entity Beanを使用したJava EEアプリケーションを開発する

Java EEアプリケーションの開発としては、EJB 3.0から使用可能となったJava Persistence APIの利用を推奨しています。EJB 2.1までの仕様として使われていたEntity Bean(エンティティビーン)を移行する場合、"D.6.1 J2EEコンテナからJava EEコンテナへの移行"を参考に、CMP拡張情報ファイル、DB定義情報などを移行してください。

Java EEアプリケーションのCMP2.0リレーション定義の情報を修正する場合、"D.6.1 J2EEコンテナからJava EEコンテナへの移行"およびInterstage Application Serverのマニュアルを参照してください。

第7章 Javaアプリケーションを開発する

ここでは、Javaアプリケーションの概要および、Javaアプリケーションの作成方法について説明します。

7.1 概要

Javaアプリケーションの概要について説明します。

7.1.1 Javaアプリケーションとは

Javaアプリケーションとは、Javaプラットフォームで動作するアプリケーションです。JavaアプリケーションはJavaプログラミング言語を使用して開発することができます。

Interstage Studioでは、アプリケーションの開発に必要なJavaのビルドパスやビルダが設定されたJavaアプリケーションプロジェクトが用意されています。Javaアプリケーションプロジェクトでは以下のアプリケーションを開発することができます。

クライアントアプリケーション

クライアントアプリケーションとは、javaコマンドやjavawコマンドを使用して、単独で実行できるプログラムです。

アプレット

アプレットとは、HTMLに埋め込むことにより、Webブラウザ上で実行できるプログラムです。HTMLは静的な表現だけですが、アプレットは動的な表現をすることができます。

ライブラリ

ライブラリとは、複数のアプリケーションで使用される機能を部品としてまとめたプログラムです。ライブラリは単独で実行できないため、クライアントアプリケーションやアプレットから使用されます。また、WebアプリケーションやEJBなどのJ2EEのアプリケーションからも使用されます。

フォーム

フォームは大別するとフレーム、ダイアログ、パネルの3種類があり、それぞれの種別ごとに継承フォームが用意されています。

JavaBeans

JavaBeansとは、Javaのクラスを部品化する規約です。JavaBeansは、規約に沿った部品の一般名称でもあります。部品化した個々の部品はBeanと呼びます。

ポイント

コンテナやBeanの種類に、重量コンポーネント(AWT)と軽量コンポーネント(Swing)の二つがあります。Javaフォームやアプレットの作成では、できるだけ重量コンポーネントと軽量コンポーネントの利用を統一することを推奨します。重量コンポーネントと軽量コンポーネントを混在すると、重ね合わせの問題(必ず重量コンポーネントが上に表示される)やメニューのBeanの下への潜り込みなど、問題が発生する可能性があります。

7.1.2 Javaアプリケーションの開発

Javaアプリケーションの開発の概要を以下に示します。

Javaアプリケーションの開発の準備

Javaアプリケーションを開発するには、まずJavaアプリケーションプロジェクトを作成し、必要なクラスパスなどのプロジェクト設定を行います。詳細は、"7.3.1 Javaアプリケーションを作成する環境を準備する"を参照してください。

クラス/アプレット/フォーム/JavaBeansの作成

各種ソース生成ウィザード、エディタを用いてJavaアプリケーションに必要なリソースを作成します。
詳細は、"7.3.2 クラスを作成する"、"7.3.3 アプレットを作成する"、"7.3.4 フォームを作成する"、"7.3.5 JavaBeansを作成する"を参照してください。

Javaアプリケーションのビルド、検証

初期状態では、リソースを保存したときに自動ビルドが行われます。ビルドでは、コンパイルエラーのほかに様々なバリデータによって資産に問題がないか検証が行われます。
詳細は、"ビルド"または"6.2.6.2 検証"を参照してください。

Javaアプリケーションの動作確認

作成したJavaアプリケーションの実行およびデバッグを行います。
詳細は、"7.3.11 Javaアプリケーションの動作を確認する"を参照してください。

Javaアプリケーションの配布

運用環境に実行資産を配布します。
詳細は、"7.3.12 Javaアプリケーションを運用環境に配布する"を参照してください。

7.2 入門

ここでは、アプレットを使用したJavaアプリケーションを作成する手順を紹介します。

その他、複数画面の遷移を行うアプレットの作成やJavaBeansの開発についてのチュートリアルがあります。詳細は"F.1 Javaアプリケーション"を参照してください。

7.2.1 作成するアプレット

国の総人口の順位付けリストから、順位を入力して国名と総人口を返し、その結果をメッセージボックスに表示するアプレットを作成します。

7.2.2 開発の流れ

ここでは、以下のようにアプレットの開発を進めます。

1. プロジェクトの作成
   アプレットを作成するために、まずJavaアプリケーションプロジェクトを作成します。ウィザードに従ってJavaアプリケーションプロジェクトを作成することで、ビルドに必要なクラスパスの設定などが自動的に行われます。

2. Javaクラスの作成
   アプレットで必要な2つのクラスを作成します。

   - データクラスの作成
   - ロジッククラスの作成

3. 入力画面の作成
   入力画面となるアプレットを作成し、グラフィカルエディタで編集します。

   - 入力画面のひな型の作成
   - 入力画面の編集

4. アプレットの動作確認
   以下の手順でアプリケーションの動作確認を行います。

   - ブレークポイントの設定
     実行時にデバッガでプログラムの動作を確認するため、ブレークポイントを設定します。

   - アプレットの実行
     アプレットビューアを起動し、アプレットの動作確認を開始します。

   - アプレットのデバッグ
     プログラムをデバッグし、アプレットが正常に動作することを確認します。

5. 運用環境へのアプレットの配布
   以下の手順で運用環境への配布を行います。

   - アプレットのエクスポート
     運用環境へアプレットを配布するため、JARファイルを作成します。

   - HTMLファイルの作成
     Webブラウザで表示するHTMLファイルを作成し、アプレットの定義を記述します。

   - 運用環境への配布
     HTMLファイルおよびJARファイルを配備します。

7.2.3 開発手順

以下に、実際にアプレットを開発する手順を説明します。

1) プロジェクトの作成

2) Javaクラスの作成

3) 入力画面の作成

4) アプレットの動作確認

5) 運用環境へのアプリケーションの配布

1) プロジェクトの作成

メニューバーから[ファイル] > [新規] > [プロジェクト]を選択すると、[新規プロジェクト]ウィザードが表示されます。

[新規プロジェクト]ウィザードから[Java] > [Javaアプリケーションプロジェクト]を選択して、[次へ]をクリックします。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
プロジェクト名	AppletSample
内容	ワークスペース内に新規プロジェクトを作成
JRE	デフォルト JRE の使用
プロジェクトをワーキングセットに追加	チェックしない

2) Javaクラスの作成

2-1) データクラスの作成

国名と総人口の情報を保持し、そこから情報を取得するデータクラスを作成します。データクラスの作成は、作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [クラス]を選択します。[新規Javaクラス]ウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
ソースフォルダ	AppletSample/src
パッケージ	sample
名前	CountryData

以下のソースファイルが生成されます。

ソースファイル	説明
CountryData.java	データクラス

国名と総人口を保持する処理を実装します。以下の**赤字**の箇所をソースに追加してください。

データクラスの実装(CountryData.java)

```
package sample;

public class CountryData {

    private String countryName;
    private int totalPopulation;

    public CountryData(String name, int total) {
        setCountryName(name);
        setTotalPopulation(total);
    }

    public String getCountryName() {
        return countryName;
    }

    public void setCountryName(String countryName) {
        this.countryName = countryName;
    }

    public int getTotalPopulation() {
        return totalPopulation;
    }

    public void setTotalPopulation(int totalPopulation) {
        this.totalPopulation = totalPopulation;
    }

}
```

ポイント

フィールドを追加した後にクラスを選択している状態で、メニューから[ソース] > [Getter および Setter の生成]を選択することで、getter/setterの追加を行うことができます。

2-2) ロジッククラスの作成

ロジッククラスの作成は、作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [クラス]を選択します。[新規Javaクラス]ウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
ソースフォルダ	AppletSample/src
パッケージ	sample
名前	PopulationRanking

以下のソースファイルが生成されます。

ソースファイル	説明
PopulationRanking.java	ロジッククラス

順位を入力して国名と総人口を返す処理を実装します。以下の赤字の箇所をソースに追加してください。

ロジッククラスの実装(PopulationRanking.java)

```
package sample;

public class PopulationRanking {

    private CountryData[] countries;

    public PopulationRanking(){

        countries = new CountryData[]{
            new CountryData("中国",1330000000),
            new CountryData("インド",1140000000),
            new CountryData("アメリカ",300000000),
            new CountryData("インドネシア",230000000),
            new CountryData("ブラジル",190000000),
            new CountryData("パキスタン",160000000),
            new CountryData("バングラデシュ",150000000),
            new CountryData("ロシア",140000000),
            new CountryData("ナイジェリア",140000000),
            new CountryData("日本",130000000)
        };

    }

    public CountryData getCountryData(int rank) {
        --rank;
        if (rank < 0 || rank >= countries.length) {
            return null;
        }

        return countries[rank];
    }

}
```

3) 入力画面の作成

3-1) 入力画面のひな型の作成

入力画面となるアプレットを作成します。アプレットの作成は、作成したプロジェクトを選択して、右クリックでコンテキストメニューから[新規] > [その他]を選択します。ウィザードの一覧から[Java] > [GUI] > [アプレット]を選択し、[次へ]をクリックします。[新規アプレット]ウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[次へ]をクリックしてください。

設定項目	設定内容
ソースフォルダ	AppletSample/src
パッケージ	sample

[アプレットの新規作成]ダイアログボックスが表示されます。Javaタブにある[アプレット]を選択し、[OK]をクリックします。アプレットウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[次へ]をクリックしてください。

設定項目	設定内容
パッケージ名	sample
アプレット名	PopulationRankingApplet
基底クラス	com.fujitsu.jbk.gui.JFApplet
矩形	(80,170,400,400)
フォント	Dialog,12
前景色	黒
背景色	白

HTMLの作成では、以下の設定項目を確認、入力してください。以下の情報を設定後、[次へ]をクリックしてください。

設定項目	設定内容
HTMLを生成する	チェックします
JBKプラグイン用HTMLを生成する	チェックします
ページタイトル	世界総人口ランキング
アプレットの幅	400
アプレットの高さ	400

このアプレットでは、パラメタを使用しません。パラメタ情報で"myItem0"を選択し、[削除]をクリックします。パラメタの削除後、[作成]をクリックしてください。

3-2) 入力画面の編集

グラフィカルエディタを使用して、作成されたPopulationRankingApplet.javaファイルを編集します。グラフィカルエディタの編集については、"7.3.6 Javaフォームを編集する"を参照してください。

以下の手順で作成したアプレットにBeanを貼り付けます。

1. プロパティウィンドウの上側に表示されているオブジェクトの一覧から、作成したアプレットを選択します。
2. プロパティウィンドウの下側に表示されているプロパティの一覧で、アプレットのプロパティを、以下の値に設定します。プロパティの設定値部分をマウスでクリックまたはダブルクリックすることにより、プロパティの種類に応じた設定を行うことができます。

プロパティ名	設定値
レイアウトマネージャ	<なし>

3. Javaフォーム定義ウィンドウのオブジェクトパレットで[AWT]をクリックします。
4. オブジェクトパレットからAWTラベルをクリックし選択します。

5. アプレットに配置します。貼り付けたい位置でマウスの左ボタンを押します。そのままマウスをドラッグし、適当な大きさのところでマウスの左ボタンを離すことにより、Beanを貼り付けることができます。
6. 同様に、AWTラベル×2個（先の手順で貼り付けたBeanを含め計3個）、AWTテキストフィールド、AWTボタンを貼り付けます。
7. プロパティウィンドウで貼り付けたBeanのプロパティを、以下の値に設定します。プロパティウィンドウの[標準]/[固有]タブを切り替えて、変更するプロパティを選択し、設定値を変更します。

label1 [AWTラベル]

プロパティ名	設定値
矩形	(24,24,256,24)
テキスト	1～10の間で順位を入力してください。

label2 [AWTラベル]

プロパティ名	設定値
矩形	(24,80,48,24)
テキスト	順位:

textField1 [AWTテキストフィールド]

プロパティ名	設定値
矩形	(80,80,120,24)
テキスト	

label3 [AWTラベル]

プロパティ名	設定値
矩形	(208,80,48,24)
テキスト	位

button1 [AWTボタン]

プロパティ名	設定値
矩形	(40,136,200,32)
ラベル	OK

3-3) 入力画面のイベント処理の記述

グラフィカルエディタのアプレット画面でButton1を選択し、コンテキストメニューから[イベント処理記述] > [action] > [actionPerformed]を選択します。PopulationRankingApplet.javaファイルに「button1_action_actionPerformed」イベント処理が追加されます。以下の**赤字**の箇所をソースに追加してください。

button1の「action_actionPerformed」イベント処理(PopulationRankingApplet.java)

```
private void button1_action_actionPerformed$(java.awt.event.ActionEvent e) {
        if (!defaultEventProc$(e)) {
            // ここにイベント発生時の処理を記述します。
            // 入力値の取得
            int rank;
            try {
                rank = Integer.parseInt(textField1.getText());
            } catch (NumberFormatException ex) {
                rank = 0;
            }
```

```
            // 入力のチェック
            if (rank < 1 || rank > 10) {
                javax.swing.JOptionPane.showMessageDialog(
                        this,
                        "指定された範囲内の順位が入力されていません。",
                        "エラー",
                        javax.swing.JOptionPane.ERROR_MESSAGE);
                return;
            }

            // 結果の出力
            PopulationRanking pop = new PopulationRanking();
            CountryData country = pop.getCountryData(rank);
            StringBuilder message = new StringBuilder();
            message.append("総人口ランキング");
            message.append(rank);
            message.append("位は「");
            message.append(country.getCountryName());
            message.append("」です。");
            message.append(System.getProperty("line.separator"));
            message.append("総人口は");
            message.append(country.getTotalPopulation());
            message.append("人です。");
            javax.swing.JOptionPane.showMessageDialog(
                    this,
                    message,
                    "結果",
                    javax.swing.JOptionPane.INFORMATION_MESSAGE);
        }
    }
```

4) アプレットの動作確認

4-1) ブレークポイントの設定

アプレットクラスのbutton1_action_actionPerformed$()メソッドの先頭行にブレークポイントを設定します。ブレークポイントは、エディタの左側のルーラー部分をダブルクリックして、設定もしくは解除します。

4-2) アプレットの実行

パッケージエクスプローラビューで、PopulationRankingApplet.javaファイルを選択して、コンテキストメニューから[デバッグ] > [Javaアプレット]を選択します。アプレットビューアが起動されて、入力画面が開きます。

[順位:]の入力フィールドに人口ランキングの順位を入力して、[OK]をクリックします。

アプレットが動作し、ブレークポイントで中断されます。変数ビューで変数の値を確認します。メニューから[実行] > [ステップオーバ]を選択し、変数ビューでプログラムの状況を確認します。デバッグについては、"6.2.7.1 デバッグする"を参照してください。

ポイント

[Javaアプレット]起動構成を未作成の場合は、[実行] > [デバッグの構成]から作成してください。

変数ビューでは値の確認だけでなく、変更も行うことができます。

デバッガで中断している処理は、メニューから[実行] > [再開]を選択することで再開され、メッセージボックスに結果が表示されます。入力した順位の国名と総人口が表示されることを確認してください。

5) 運用環境へのアプリケーションの配布

このアプレットを運用環境に配布するには、HTMLファイルおよびJARファイルを作成する必要があります。

5-1) アプレットのエクスポート

JARファイルの作成は、エクスポートウィザードから行います。エクスポートウィザードを起動するには、[ファイル] > [エクスポート]を選択してください。[エクスポート]ウィザードから[Java] > [JAR ファイル]を選択してください。

JARエクスポートウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
エクスポートするリソースの選択	AppletSample/src/sampleフォルダにある以下のファイルのみチェックします。 ・ CountryData.java ・ PopulationRanking.java ・ PopulationRankingApplet.java その他のファイルが選択されている場合は、そのファイルのチェックを外します。
生成されたクラスファイルとリソースをエクスポート	チェックします。
JAR ファイル	AppletSample¥AppletSample.jar
JAR ファイルの内容を圧縮	チェックします。

5-2) HTMLファイルの作成

HTMLファイルを作成し、アプレットの定義を記述します。このアプレットでは、プロジェクトの作成で自動生成されたひな型のHTMLファイルをそのまま使用します。プロジェクト内に自動生成されたPopulationRankingApplet-JBKPlugin.htmlをエディタで開き、以下の定義が記述されていることを確認してください。

属性名	設定値
code	sample.PopulationRankingApplet.class
archive	AppletSample.jar

HTMLファイルを新たに作成する場合は、"J Business Kit オンラインマニュアル"を参照してください。

5-3) 運用環境への配布

作成した以下のファイルを運用環境のWebサーバ上の同じフォルダにコピーします。

- PopulationRankingApplet-JBKPlugin.html
- AppletSample.jar

ポイント

HTMLファイル内に記述されているアプレット定義にcodebase属性を追加することにより、HTMLファイルとアプレットを別の場所に配備することができます。詳細については、"J Business Kit オンラインマニュアル"を参照してください。

7.3 タスク

ここでは、Interstage Studioを用いてJavaアプリケーションを開発する方法について、開発作業課題(タスク)ごとに説明します。

7.3.1 Javaアプリケーションを作成する環境を準備する

Javaアプリケーションを作成するには、まずプロジェクトを作成してビルドできる環境を整えます。

プロジェクトの作成

[新規]ウィザードから[Java] > [Javaアプリケーションプロジェクト]を選択し、Javaアプリケーションプロジェクトを作成します。

クラスパスの設定

アプリケーションを作成するのに必要なライブラリなどがある場合には、クラスパスの設定を行う必要があります。クラスパスの設定は、プロジェクトの[Javaのビルドパス]プロパティで行います。

7.3.2 クラスを作成する

Javaクラスファイルを作成するには、JavaクラスウィザードでJavaクラスファイルを作成し、Javaエディタなどを用いて編集します。以下にその方法を説明します。

クラスファイルの新規作成

クラスファイルは、新規ウィザードから[クラス]を選択し、ウィザードで作成します。

クラスファイルの編集

クラスファイルの編集には、Javaエディタを用います。Javaエディタは以下の特徴を持つテキストエディタです。

- 構文の強調表示
  色とフォントスタイル（太字）で、キーワードを強調表示します。
- コンテンツ/コードアシスト
  コードの作成時に、コンテンツアシスト(コードアシストとも呼ばれる)を表示します。
- コードフォーマット
  コードフォーマッタの設定に従い、ソースを整形します。
- インポート支援
  プロジェクトに設定されたクラスパスに従った適切なimport文が挿入できます。
- Javaフォームの変更禁止ソースを保護
  Javaフォームの画面(Bean)の情報やイベント処理を呼び出す部分のソースが変更されないよう、保護します。
- BeanInfoの定義
  JavaBeansのBeanInfoを定義し、BeanInfoの情報をコメントとして付加します。

Javaエディタの詳細については、"Java開発ユーザガイド"の"概念"の"Javaエディタ"を参照してください。

7.3.3 アプレットを作成する

アプレットソースを作成するには、アプレットウィザードでアプレットソースを作成し、グラフィカルエディタまたはJavaエディタを用いて編集します。以下にその方法を説明します。

アプレットの新規作成

java.applet.Applet/javax.swing.JAppletおよびその継承クラスを基底クラスとしたJavaフォームです。

[新規]ウィザードから[Java] > [GUI] > [アプレット]を選択し、ウィザードで作成します。

ウィザードでの設定は、以下を参考にしてください。

- パッケージ名
  生成されるソースのパッケージ名を指定します。

- アプレット名
  アプレットソースのクラス名を指定します。

- 設定
  アプレットの矩形、フォント、前景色および背景色を指定します。

- 派生クラス情報
  基底クラスを指定します。

- HTMLの作成、JBKプラグイン用HTMLを生成する
  アプレットまたはJBKプラグイン用のアプレットを起動するためのHTMLの生成を指定したり、HTMLのページタイトル、HTMLに記述されるアプレットの幅および高さを指定します。

- パラメタ情報
  自動生成するHTMLファイルに反映するパラメタ情報です。アプレットのパラメタは名前と値を組にして、複数組指定できます。

注意

アプレットの基底クラスに、ほかのプロジェクトのクラスやライブラリのクラスを指定した場合は、自動生成されたひな型HTMLの修正が必要です。これは、アプレットを実行するためには基底クラスが必要になるからです。

例えば、アプレットのMyAppletクラスが"project1.jar"の中にあり、基底クラスがbaseclass.jarの中にある場合は、appletタグのarchive属性に"baseclass.jar"を追加します。

<applet code=MyApplet.class width=400 height=400 archive="project1.jar, baseclass.jar">

</applet>

アプレットのビルド時にabstractメソッドが記載されていない旨のエラーメッセージが表示される場合があります。このような場合は、Javaエディタなどを起動し、エラーメッセージで表示されたabstractメソッドを記述してください。

7.3.3.1 アプレット情報

項目	説明
ソースフォルダ	アプレットソースを格納するソースフォルダを選択します。
パッケージ	生成されるクラスのパッケージ名を指定する場合は、パッケージ名を入力します。

7.3.4 フォームを作成する

フォームを作成するには、フォームウィザードでソースを作成し、グラフィカルエディタまたはJavaエディタを用いて編集します。以下にその方法を説明します。

7.3.4.1 フレーム、ダイアログ、パネルを作成する

フォームには以下の種類があります。フォームは任意のフレーム/ダイアログボックス/パネルクラスを基底クラスとして構成されています。基底クラスはクラスパス上にあることが前提となります。

フォームは、[新規]ウィザードから[Java] > [GUI] > [フォーム]を選択し、ウィザードで作成します。

項目	内容
フレーム	java.awt.Frame/javax.swing.JFrame およびその継承クラスを基底クラスとしたJavaフォームです。 プルダウンメニューやタイトルバーなどで構成される一般的なウィンドウのフレーム(枠)をもつフォームです。
ダイアログ	java.awt.Dialog/javax.swing.JDialog およびその継承クラスを基底クラスとしたJavaフォームです。 見かけは、フレームと似ていますが、モーダルにできる、タイトルバーのアイコンを指定できないなど、フレームとの機能差があります。
パネル	java.awt.Container/java.awt.Window/java.awt.Panel/javax.swing.JPanel およびその継承クラスを基底クラスとしたJavaフォームです。 ほかのフォームで扱う基底クラスは、パネルとはなりません。

ポイント

自分で作成したクラスなど、任意のクラスを継承したい場合はフレーム、ダイアログ、パネルを選択します。

注意

フレーム、ダイアログ、パネルのビルド時にabstractメソッドが記載されていない旨のエラーメッセージが表示される場合があります。このような場合は、Javaエディタなどを起動し、エラーメッセージで表示されたabstractメソッドを記述してください。

7.3.4.2 画面制御パネル、画面制御タブパネル、ウィンドウ制御、オブジェクト制御を作成する

画面制御パネル、画面制御タブパネル、ウィンドウ制御、および、オブジェクト制御を新規作成します。

[新規]ウィザードから[Java] > [GUI] > [フォーム]を選択し、ウィザードで作成します。

各ウィザード、編集の相違点は以下です。

	画面制御パネル	画面制御タブパネル	オブジェクト制御	ウィンドウ制御
基底クラス	JFLCardPanel	JFCTabbedPanel	JFCObjectLoaderManager	JFCWindowLoaderManager
制御されるオブジェクトの基底クラス	JFLEntryPanel	JFLEntryPanel	java.lang.Object	JFCFrameまたは JFCDialog
制御されるオブジェクトの編集ウィンドウ上の名称	画面名	画面名	オブジェクト名	ウィンドウ名
編集開始コメント	//@@JFLCardPanel createPanels Method start	//@@JFCTabbedPanel createPanels Method start	//@@JFCObjectLoaderManager createObjects Method start	//@@JFCWindowLoaderManager createWindows Method start
編集終了コメント	//@@JFLCardPanel createPanels Method end	//@@JFCTabbedPanel createPanels Method end	//@@JFCObjectLoader	//@@JFCWindowLoaderManager

	画面制御パネル	画面制御タブパネル	オブジェクト制御	ウィンドウ制御
			Manager createObjects Method end	createWindows Method end

注意

各クラスのパッケージ名は断りのない限りcom.fujitsu.jbk.gui.ctrlです。編集開始コメントと編集終了コメントの間が編集ツールの編集対象となります。

オブジェクト制御ウィザードはコンストラクタ上からaddメソッドを呼び出すようにcreateObjectsメソッドを生成します。ただし、既存の資産を指定して作成した場合には生成されません。

オブジェクト制御編集には制御される側のオブジェクトの作成および編集機能はありません。

ウィンドウ制御編集で、制御される側のクラスとしてダイアログを指定した場合には、生成および削除のタイミングではなく、モーダル/モードレスを指定します。フレームを指定した場合は、他のクラスと同様に、生成タイミングと削除タイミングを指定します。ウィンドウ制御編集でプロジェクト資産上から資産を選択した場合には、自動的にダイアログとフレームに判別されます。

7.3.4.3 Javaフォーム情報

項目	説明
ソースフォルダ	Javaフォームソースを格納するソースフォルダを選択します。
パッケージ	生成されるクラスのパッケージ名を指定する場合は、パッケージ名を入力します。

7.3.5 JavaBeansを作成する

JavaBeansにはいくつかの種類が存在し、それぞれ作成方法が異なっています。

- 単体Bean
  テキストフィールドやボタンのような単一のコンポーネントから構成されるBeanです。既存の単体Beanを継承し、付加機能を付け、新しい単体Beanとして作成します。
- 複合Bean
  複数のBeanで構成するBeanです。Javaフォーム定義で、フォームのレイアウトを作成する要領でパネル(JPanelまたはPanel)に対して、Beanを貼り付け作成します。
- 不可視Bean
  基底クラスにビューを持たないクラスを指定した場合は不可視Beanとなります。不可視Beanはビューを持たないため、一般のJavaソースと同様にJavaエディタで編集します。

単体Beanと不可視Beanは、新規ウィザードから[Java] > [GUI] > [JavaBeans]を選択し、ウィザードで作成します。

ウィザードでの設定は、以下を参考にしてください。

- パッケージ名
  生成されるソースのパッケージ名を指定します。

- Bean名
  Beanのクラス名を指定します。

- BeanInfo作成
  BeanInfo情報を自動的に作成します。継承元クラスの親のプロパティやメソッド、イベントをデフォルトのBeanInfo情報として生成します。チェックしなかった場合は空のBeanInfo情報を生成します。

- 設定
  Beanの矩形、フォント、前景色および背景色を指定します。

- 派生クラス情報
  基底クラスを指定します。

複合Beanは、フォーム(パネル)を作成後、ワークベンチのメニューバーから[編集] > [BeanInfoの定義] > [BeanInfo定義]からBeanInfoの定義を行うことで作成します。BeanInfo定義の詳細については、"7.3.8 BeanInfoを定義する"を参照してください。

7.3.5.1 JavaBeans情報

項目	説明
ソースフォルダ	Beanソースを格納するソースフォルダを選択します。
パッケージ	生成されるクラスのパッケージ名を指定する場合は、パッケージ名を入力します。

7.3.6 Javaフォームを編集する

Javaフォームの編集

Javaフォームを編集するために、Javaフォーム定義では以下の操作を行うことができます。

- Beanを設定する
  オブジェクトパレットからBeanを選択および配置し、属性を設定します。
- 排他選択グループを定義する
  Javaフォーム定義のメニューバーから[ツール] > [排他選択グループ定義]を選択します。
- Javaフォームに装飾する
  Javaフォームを線や四角などの描画で装飾します。

Javaフォーム定義の画面構成

Javaフォーム定義の画面は、以下5つのウィンドウ、エディタおよびビューから構成されています。これらの画面を使用してJavaフォームの編集を行います。

- Javaフォーム定義ウィンドウ
  Javaフォームの保存やBeanのコピーなどの編集をすることができます。
- プロパティウィンドウ
  Beanの一覧を表示し、プロパティの参照や変更をすることができます。
- Javaフォームウィンドウ
  Beanをマウス操作でJavaフォーム上に配置し、レイアウトの編集をすることができます。
- Javaエディタ
  Beanのイベント処理やユーザ定義メソッドの記述をすることができます。
- Beanリストビュー
  Beanの一覧を表示し、メソッドやイベントの挿入をすることができます。

注意

JavaエディタでJavaフォームを開いた場合、背景色の異なる部分のソースは編集できません。この部分のソースは、画面(Bean)の情報やイベント処理を呼び出すソースです。また、先頭の"// Graphical Editor Form"と"//@@Form Design Information start"、"//@@Form Design Information end"で囲まれた部分についても、変更を禁止しています。これらの部分を編集すると、グラフィカルエディタで開けなくなります。

7.3.6.1 Beanを設定する

Beanの新規配置、サイズの変更、移動および複写などの編集操作はマウスを使って行います。

- Beanの新規配置
  Beanの新規配置は、Javaフォーム定義からドラッグ＆ドロップで簡単に行うことができます。

- サイズや色などの変更
  Beanのプロパティを設定することで、サイズや色、フォントなどを変更することができます。

7.3.6.2 Beanを配置する

JavaアプリケーションでBeanを配置する方法として、レイアウトマネージャがあります。レイアウトマネージャを使用すると、画面のサイズ変更に応じて自動的にレイアウトが補正されますが、レイアウトマネージャのサポートする画面形式によって制約を受けます。

画面の拡張性の観点からは画面のサイズ変更に対応することが望ましいため、Javaアプリケーションの画面設計では、レイアウトマネージャの使用を推奨します。

レイアウトの特性

- FlowLayout
  Beanを横並びに配置します。
- BorderLayout
  上下左右にBeanを配置します。
- GridLayout
  縦横サイズがそれぞれ均等なグリッド状にBeanを配置します。
- GridBagLayout
  グリッド状にBeanを配置するのは「GridLayout」と同様ですが、Beanのサイジング規則を細かく設定することが可能です。
- CardLayout
  パネルを切り替えるために使用します。

ポイント

Beanの描画が重なる場合は、重量コンポーネントが軽量コンポーネントの上に描画されます。また、各コンポーネント同士の描画が重なる場合は、コンテナへの追加順に従って決定されます。追加順は、コンポーネントツリー上の順番です。レイアウトマネージャの指定が「なし」以外の場合は、同じコンポーネント同士で描画が重なることはありません。

7.3.6.3 Beanのプロパティを参照/設定する

Beanのプロパティには以下の種類があります。

- 標準プロパティ
- 固有プロパティ

標準プロパティの一覧

JavaフォームおよびBeanの標準的プロパティです。標準プロパティの一覧は以下になります。

和名プロパティ	英名プロパティ
Bean名	beanName
レイアウトマネージャ	layout
矩形	bounds
境界線種別	border
配置条件	constraints
重なり／並び順	addIndex
表示	visible
背景色	background
前面色	foreground

和名プロパティ	英名プロパティ
フォント	font
AWT識別名	AWTName
ツールチップ文字列	toolTipText
アテンション	attention

注意

boundsプロパティはウィンドウのクライアント域の幅と高さの値を表示しますが、[実行イメージで編集]が選択されている場合は、ウィンドウの幅と高さの値を表示します。

固有プロパティ

固有プロパティは、各種Beanが提供するBeanInfo情報に指定された情報をもとに表示します。

固有プロパティが読み取り専用プロパティ、書き込み専用プロパティ、および、不可視属性いずれかの場合は、表示を行いません。

また、環境設定のオプションのプロパティで[専門家向けのプロパティを表示する]が選択されていない場合は、一部のプロパティは表示を行いません。

注意

固有プロパティのプロパティ名の先頭に「*」が付いている項目は、[設定値]を初期値から変更することはできません。

プロパティシートの情報

プロパティの参照および設定は、プロパティウィンドウのプロパティシートで行います。プロパティシートには以下の情報が表示されます。

- 標準タブ
  Beanの標準プロパティを表示します。
- 固有タブ
  Beanの固有プロパティを表示します。
- プロパティ名
  プロパティの名称を表示します。
- 設定値
  設定中のプロパティの値を表示します。この値を書き換えると、Beanのプロパティを変更することができます。プロパティの種類によって、一覧からの値選択、数値のスピンボタンでの選択が行えます。
- 説明文
  プロパティシートの下部には、選択中のプロパティの説明文を表示します。

プロパティシートの機能

プロパティウィンドウ内のボタンには、以下の機能があります。

- Beanカスタマイザ表示
  選択中のBeanがカスタマイザを提供している場合、カスタマイザを表示します。
- 一覧から選択
  選択中のプロパティが、プロパティエディタを提供している場合、プロパティエディタを表示します。
- null値を設定
  選択中のプロパティにnullを設定します。Java言語の基本データ型ではないプロパティに対して設定できます。

- 初期値に戻す
  プロパティの値を標準値に戻します。
- 元に戻す
  プロパティの値を1つ前の状態に戻します。

ポイント

Javaフォーム定義には、スポット編集機能というBeanの主要なプロパティをフォーム上で編集する機能があります。スポット機能を使用するには、Javaフォーム定義のメニューバーから[環境設定] > [オプション] > [フォーム]タブで[スポット編集機能を有効にする]をチェックします。

選択しているBean上でマウスをクリックするか、[Ctrl+E]キーを押すとスポット編集状態になり、Beanの上部にスポット編集中であることを示すラベルが表示されます。

スポット編集を終了するには[Enter]キー、複数行入力可能スポット編集の場合は[Ctrl+Enter]キーを押します。

7.3.6.4 Bean関係を作成する

Javaフォーム定義では、Javaフォーム、アプレットおよびJavaプログラムにおいてBean関係を対話形式で作成し、そのBean関係を可視化することができます。

Bean関係の作成

以下の方法で[Bean関係の作成ウィザード]を起動することができます。ウィザードの指示にしたがって、ソースBeanのイベントおよびターゲットBeanのメソッド/プロパティを設定すると、指定したBeanのイベント処理記述内に、Bean関係のソースが作成されます。

- メニュー起動
  Javaフォーム定義のメニューバーから[ツール] > [Bean関係の作成ウィザード]を選択します。
- ワイヤリング起動

  1. Javaフォームのイベント処理を行うBean(ソースBean)上でマウスの右ボタンをクリックします。
  2. マウスの右ボタンをクリックした状態で、そのままドラッグします。ドラッグ中は、マウスの右ボタンをクリックした位置からマウスカーソルがある位置まで、線(ワイヤ)が表示されます。
  3. ドラッグした状態でマウスを移動させ、メソッド呼出し/プロパティ取得/プロパティ設定を行うBean(ターゲットBean)上でマウスの右ボタンを離します。

Bean関係の編集

[Bean関係の作成ウィザード]に設定されている値を編集し[作成]をクリックすると、可視化されたBean関係(矢印線)に対応するイベント処理記述内のBean関係のソースが更新されます。

ウィザードは以下の方法で起動することができます。

- Javaフォームの可視化されたBean関係(矢印線)上でマウスの左ボタンをダブルクリック。
- Javaフォームの可視化されたBean関係(矢印線)上でマウスを右クリックしコンテキストメニューを表示。[ウィザードで編集]を選択。
- Javaフォーム定義のメニューバーから[ツール] > [Bean関係の確認]を選択。[Bean関係の確認]画面のBean関係一覧から編集するBean関係を選択し、[ウィザードで編集]をクリック。

Bean関係の削除

Javaフォームの可視化されたBean関係(矢印線)上でマウスの右ボタンをクリックし、コンテキストメニューから[削除]をクリックすることで、Bean関係を削除できます。

ポイント

Bean関係の削除は、Bean関係の確認画面からも行うことができます。

Bean関係の可視化

Bean関係を確認するには、Javaフォーム定義のメニューバーから[ツール] > [Bean関係の確認]を選択し、Bean関係の確認画面を起動します。

[Bean関係の確認]画面には、すべてのBean関係の一覧が表示され、Bean関係の新規作成、編集、削除を行うことができます。

Bean関係の解析

Bean関係は自動的に解析されますが、Javaフォーム定義のメニューバーから[編集] > [Bean関係再解析]を選択して任意で解析を行うことも可能です。

7.3.6.5 メニューを定義する

メニューの定義

プルダウンメニューを使う場合は、Javaフォーム定義のメニューバーから[ツール] > [メニュー定義]を選択し、メニュー定義を起動します。メニュー定義では、メニューの構成やメニューに表示する文字列などの情報を定義します。

メニューの操作

Javaフォームに定義したメニューのオブジェクトはJavaフォームのクラスフィールドとなります。通常これらのフィールドを使用してメニューを直接操作する必要はありませんが、特別な処理を行いたい場合にはクラスフィールドを通してオブジェクトを操作できます。

メニューオブジェクトを操作する例を以下に示します。

例

"APPLY"アテンションが発生したら、メニューオブジェクト「cancel」をマスクします。

```
public boolean processAttention_APPLY() {
  cancel.setEnabled(false);
  return true;
}
```

アテンション

メニュー操作に対する動作を記述するためにはアテンションを使用します。アテンションはメニューの操作のみならず、画面全般の操作に対して付けた名前で、メニュー、ボタン、キーなどの操作に対する動作の記述に使用します。

アテンションを発生させるためにはアクションイベントの定義が必須なため、一般のアクションイベントと違いがありません。そのため手間が増加するケースもあるため、アテンションの使用は推奨しません。

7.3.6.6 排他選択グループを定義する

排他選択グループとは、排他選択型Bean(ラジオボタン)をグループ化し、自動的に選択状態を設定する機能です。同一の排他選択グループ内で1つのBeanだけが選択状態に設定され、ほかのBeanは非選択状態に設定されます。

排他選択グループの定義方法

1. 排他選択グループの設定ダイアログボックスの表示
   Javaフォーム定義のメニューバーから[ツール] > [排他選択グループ定義]を選択し、排他選択グループ定義を起動します。
2. 排他選択グループの新規作成
   [JBK用グループ追加]、[AWT用グループ追加]または[Swing用グループ追加]をクリックして新規に選択グループを作成します。
3. グループメンバの設定
   [グループ一覧]リストにある作成したグループ名を選択して、[メンバ設定]をクリックすると、[排他選択グループ内のメンバ設定]画面が表示されます。ここで、グループのメンバを設定します。

注意

デフォルトの排他選択グループ(グループ名:__cdDefaultJFButtonGroup、__cdDefaultCheckboxGroupおよび__cdDefaultButtonGroup)があらかじめ定義されています。排他選択型BeanをJavaフォームに貼った場合、デフォルトの排他選択グループに自動的に追加されます。デフォルトの排他選択グループを削除することはできません。

排他選択グループの直接操作

特別な処理を行いたい場合にはクラスフィールドを通してオブジェクトを操作することができます。

[クラスフィールド名]
排他選択グループ定義ダイアログボックスでグループを追加した際に「グループオブジェクト名」で指定した名前。

[クラスフィールド型]

作成した排他選択グループの種類	排他選択グループオブジェクトのクラスフィールド型
AWT用グループ	java.awt.CheckboxGroup
Swing用グループ	javax.swing.ButtonGroup

例

"APPLY"アテンションが発生したら、AWT用排他選択グループ「select」内で排他選択されているオブジェクトの背景色を赤に設定します。

```
public boolean processAttention_APPLY() {
  select.getSelectedCheckbox().setBackground(java.awt.Color.red);
  return true;
}
```

7.3.7 画面制御パネル、ウィンドウ制御パネルを編集する

7.3.7.1 画面制御パネル編集

既存の資産を使用して新規作成

画面制御パネルを既存の資産を使用して開くには、[新規]ウィザードから[Java] > [GUI] > [フォーム]を選択し、以下の手順で行います。

1. [新規フォーム]ウィザードでソースフォルダとパッケージを指定します。
2. [Javaフォームの新規作成]ダイアログボックス > [Java]タブ > [画面制御パネル]アイコンを選択し、[OK]をクリックします。
3. [既存の資産を画面制御パネル編集で編集できる形式に変換する]を選択し、[次へ]をクリックします。
4. [変換元ソース]欄にJavaソース名を入力して、[変換]をクリックします。
5. または、[選択]をクリックして変換元Javaソースを選択します。
6. [作成]をクリックします。画面制御パネル編集機能の画面が起動し、変換元のソースのバックアップが作成されます。

画面制御パネルの保存

編集中の画面制御パネルを保存するためには、画面制御パネルの[ファイル]メニューを使用します。

画面制御パネルの編集

以下の項目では設定する画面部品の順序や名称、生成/削除のタイミングなどを編集することができます。詳しくは"JBK GUIライブラリユーザーズガイド"の"画面操作ライブラリ"を参照してください。

項目	説明
画面名	登録する画面名です。
クラス名	登録するクラス名です。
生成タイミング	画面部品を生成するタイミングです。 ・ 初期化時:JFCPanelLoader.BUFFERED_INSTANCEに対応しています。 ・ バックグラウンド:JFCPanelLoader.BACKGROUND_CREATIONに対応しています。 ・ 表示時:JFCPanelLoader.DELAYED_CREATIONに対応しています。
削除タイミング	画面部品を削除するタイミングです。 ・ 非表示時保持:JFCPanelLoader.BUFFERED_INSTANCEに対応しています。 ・ 保持:JFCPanelLoader.PERMANENT_INSTANCEに対応しています。 ・ 非表示時:JFCPanelLoader.TEMPORARY_INSTANCEに対応しています。

7.3.7.2 ウィンドウ制御パネル編集

ウィンドウ制御パネルの保存

編集中のウィンドウ制御パネルを保存するためには、ウィンドウ制御パネルの[ファイル]メニューを使用します。

ウィンドウ制御パネルの編集

以下の項目では設定する画面部品の順序や名称、生成/削除のタイミングなどを編集することができます。詳しくは"JBK GUIライブラリユーザーズガイド"の"画面操作ライブラリ"を参照してください。

項目	説明
種別	フレーム、またはダイアログです。
ウィンドウ名	登録するウィンドウ名です。
クラス名	登録するクラス名です。
生成タイミング(種別がフレームの場合のみ)	ウィンドウ部品を生成するタイミングです。 ・ 初期化時:JFCPanelLoader.BUFFERED_INSTANCEに対応しています。 ・ バックグラウンド:JFCPanelLoader.BACKGROUND_CREATIONに対応しています。 ・ 表示時:JFCPanelLoader.DELAYED_CREATIONに対応しています。
削除タイミング(種別がフレームの場合のみ)	ウィンドウ部品を削除するタイミングです。 ・ 非表示時保持:JFCPanelLoader.BUFFERED_INSTANCEに対応しています。 ・ 保持:JFCPanelLoader.PERMANENT_INSTANCEに対応しています。 ・ 非表示時:JFCPanelLoader.TEMPORARY_INSTANCEに対応しています。
モーダル値(種別がフレームの場合のみ)	ダイアログ表示時のモーダル/モードレスの切り替えです。

7.3.8 BeanInfoを定義する

BeanInfoは、Beanをコンポーネントとして利用するプログラムに対するインタフェース情報です。BeanInfoの情報を定義するにはBeanInfo定義を使います。Beanのファイルを開き、ワークベンチのメニューバーから[編集] > [BeanInfoの定義] > [BeanInfo定義]を選択して、BeanInfo定義を呼び出します。

ポイント

BeanInfo定義のメニューは、Beanのファイルを開いたJavaエディタにフォーカスが当たっていないと有効になりません。パッケージエクスプローラ等、Javaエディタ以外がフォーカスされていないか確認してメニューを開いてください。

BeanInfoの定義については以下があります。

- プロパティの定義
  Beanのプロパティを追加するには[プロパティ追加]を押して表示される[プロパティの追加]ダイアログボックスで情報を入力します。

  ー 定義フィールド
  Beanのプロパティの中で特殊な情報は以下があります。

項目	説明
編集クラス名	プロパティを編集するための専用クラス名をパッケージ名付きで指定します。編集クラス名は省略することができます。
内部使用のために不可視とする	一般の利用者は使用しないプロパティとする場合に選択します。
専門家向けの機能とする	専門家向けのプロパティとする場合に選択します。
属性情報	プロパティに対する簡単な説明を記述できます。

ポイント

[内部使用のために不可視とする]、[専門家向けの機能とする]、[属性情報]については、プロパティ、メソッド、イベントの各定義フィールドで共通となります。

- メソッドの定義
  メソッドを有効にするには、ツリー表示域でメソッド名を選択し、[有効]をクリックします。継承元クラスのメソッドをBeanのメソッドとする場合は、[extendsクラスのpublicメソッドを表示]をチェック状態にし、ツリー表示域で継承元クラスのメソッド名を選択し、[有効]をクリックします。
  Beanのメソッドを無効にするには、ツリー表示域でメソッド名を選択し、[無効]をクリックします。

- イベントの定義
  Beanのイベントを追加するには、[イベント追加]をクリックします。続いて表示される[イベントの追加]ダイアログボックスで必要な情報を入力し、[OK]をクリックします。
  基底クラスのイベントをBeanのイベントにするには、ツリー表示域に表示されているイベント名を選択し、[有効]をクリックします。
  Beanのイベントを無効にするには、ツリー表示域で、イベント名を選択し、[無効]をクリックします。
  イベント・オブジェクト・クラスならびにイベント・リスナ・インタフェースのソースを生成するときは、ワークベンチのメニューバーから[編集] > [BeanInfoの定義] > [イベントソースの生成]を選択します。生成されたクラスの修正は、通常のJavaソースとしてJavaエディタを利用します。

注意

BeanInfo定義を利用して生成されたプロパティ、メソッド、イベントなどを無効にしてもソース上では削除されません。削除する場合は、BeanInfo定義を終了し、Javaエディタの機能で削除してください。

7.3.9 ユーザインタフェースの処理を記述する

ユーザインタフェースの処理を作成する方法について説明します。この説明の対象はフォーム、アプレットおよびJavaBeansの可視Beanになります。

7.3.9.1 ユーザインタフェースを持つアプリケーションの処理を記述する

ユーザインタフェースを持つアプリケーションの例として、フレームを表示するアプリケーションの処理を説明します。

処理には、以下のものがあります。

- メインプログラム:アプリケーションの初期化やフォームの表示
- 初期化処理:フォームの初期化処理
- イベント処理:ユーザ操作などによって発生するイベントに対する処理

フレームを表示するメインプログラムの例

```
public class MyJavaApp {
    //コンストラクタ
    public MyJavaApp() {
    }
    //フレームを表示するメソッド
    public void run() {
        //フレームのインスタンスを作成
        Frame1 form = new Frame1();
        //フレームのインスタンスのsetVisibleメソッドを使い、表示を有効にする。
        form.setVisible(true);
    }
    //メイン処理
    public static void main(String[] args) {
        // MyJavaAppクラスのインスタンスを作成
        MyJavaApp object = new MyJavaApp();
        // MyJavaAppクラスのインスタンスのrunメソッドを呼び出す。
        object.run();
    }
}
```

7.3.9.2 フォームを操作する

フォームには、フレーム、ダイアログ、パネルがあります。

ここではフレームを例に、処理の記述について説明します。

フレームの操作

操作の例として、フレームの操作について説明します。

- 情報の共有
  メインプログラムとJavaフォーム間、Javaフォーム同士でデータを共有する場合は、メインプログラムにデータを定義し、各フォームでそのデータを参照するか、コンストラクタを追加し、共有したいデータをパラメタとして渡すようにします。
- フレームの表示方法
  フレーム表示方法として例を示します。

```
アプリケーションまたは既存のフレームからFrame1というフレームを呼び出す場合
  Frame1  form  =  new  Frame1();   // フレームのインスタンスを作成します。
  form.init();                      // フレームを初期化します。
  form.setVisible(true);            // フレームを表示します。
```

- フレームのクローズ方法
  フレームをクローズする場合は、「this.dispose();」を記述します。

7.3.9.3 Beanを操作する

ここでは、Beanを操作する方法について説明します。

Javaフォームに貼ったBeanはJavaフォームクラスのクラスフィールドとして生成されます。クラスフィールドはJavaフォーム定義時に「Bean名」プロパティに指定した名前になります。

Beanの操作はBeanのプロパティを設定、参照したり、Beanのメソッドを呼び出したりすることで行います。

プロパティの参照および設定

プロパティの参照および設定の例を以下に示します。

テキストフィールドBean「textField1」がフォーカスを得たときに発生するfocus_focusGainedイベントで「背景色(background)」プロパティを「青」に変更します。また、フォーカスを失ったときに発生するfocus_focusLostイベントで「背景色(background)」プロパティを「白」に戻します。

```
[「TextField1」の処理]
public void textField1_focus_focusGained(FocusEvent e) {
  if (!defaultEventProc(e)) {
    textField1.setBackground(java.awt.Color.blue);
  }
}
public void textField1_focus_focusLost(FocusEvent e) {
  if (!defaultEventProc(e)) {
    textField1.setBackground(java.awt.Color.white);
  }
}
```

メソッドの呼び出し

メソッド呼出しの例を以下に示します。

プッシュボタン「button1」を押すとテキストフィールド「textField1」に入力されている文字をリストボックス「list1」に追加します。

```
[「button1」の処理]
public void button1_action_actionPerformed(ActionEvent e) {
  if (!defaultEventProc(e)) {
    list1.addItem(textField1.getText());
  }
}
```

7.3.9.4 イベント処理を記述する

イベント処理の追加方法には以下3つの方法があります。

- Javaエディタで直接イベント処理を記述
  Javaエディタを使用して、イベントごとに直接メソッドを記述します。
- Bean関係の作成ウィザードでイベント処理を生成
  Bean関係の作成ウィザードを使用すると実際にプログラミングすることなく、対話的に処理を生成することができます。詳細は"7.3.6.4 Bean関係を作成する"を参照してください。
- Beanリストビューでイベント処理を挿入
  BeanリストビューはJavaフォーム定義で編集しているJavaフォームとBeanの一覧を表示し、メソッドやイベントを挿入する機能を持ちます。Beanリストビューには以下の機能があります。
  - Javaフォーム定義で編集しているBeanを一覧表示
    Javaフォームで編集しているBeanをツリー形式で表示することができます。
  - 各Beanのプロパティ、メソッド、イベントを一覧表示
    プロパティ、メソッド、イベントを展開して、各Beanのプロパティなどを一覧表示することができます。
  - ソースにメソッドを挿入
    [メソッド挿入]ダイアログボックスを使用して、ソースに簡単にメソッドを挿入することができます。
  - ソースにイベントを挿入
    ソースに簡単にイベントを挿入することができます。

7.3.10 Javaアプリケーションの環境設定を行う

7.3.10.1 Beanの登録

作成したBeanをJavaフォーム定義に登録することで、J Business Kitなどと同様にJavaフォームに貼り付けて使用できるようになります。
Beanの登録は、Javaフォーム定義のメニューバーから[環境設定] > [Beanの登録]ダイアログボックスから以下の手順で行います。

1. [パレットの編集]タブにある[Beanの追加]をクリックすると、[Beanの追加]ダイアログボックスが表示されます。
2. JARファイル名またはBeanのクラス名を入力します。指定したBeanが「パレットに表示可能なBean」内に追加されます。
3. 「パレットに表示可能なBean」に追加されたBeanを、「パレットに表示されているBean」に追加します。

7.3.10.2 ウィザードのカスタマイズ機能

Javaフォーム、アプレットおよびJavaBeansのウィザードでは、以下のカスタマイズ機能があります。Javaフォーム定義のメニューバーから[環境設定] > [ウィザードの編集]でカスタマイズします。

- ウィザードの編集
  ウィザードアイコンの配置、タブ、名称およびアイコンを設定できます。
- ユーザ設定フォームウィザードフォームの基底クラスがあらかじめ決まっている場合は、[ウィザードの追加／編集]ダイアログボックスで基底クラスを固定としたカスタムウィザードを作成できます。
  [使用可能なウィザード]リストで[ユーザ設定パネル]、[ユーザ設定フレーム]、[ユーザ設定ダイアログ]または[ユーザ設定アプレット]を選択します。

注意

ユーザ設定パネルで、java.applet.Appletやjava.awt.Frameを指定することはできません。これらを指定する場合はユーザ設定フレームや、ユーザ設定アプレットを使います。

7.3.10.3 オプションの設定

Javaフォーム定義では、Javaフォーム定義のメニューバーから[環境設定] > [オプション]で各種オプションを設定することが可能です。その中でも特殊な項目について以下に記します。

- [フォーム]タブ

項目	説明
グリッドに合わせる	新規に貼り付けたBeanを、グリッド境界に合わせて位置調整するかどうかを指定します。Beanを移動したときには、Beanの左上がグリッド境界に合致するように位置調整されます。
推奨サイズで貼る	フォーム上にBeanを新規に作成するとき、マウスで指定した矩形にかかわらず「推奨サイズ」で作成します。

- [プロパティ]タブ

項目	説明
専門家向けのプロパティを表示する	プロパティシートに専門家向けのプロパティを表示します。
設定	[フォントの選択]ダイアログボックスを起動します。

注意

プロパティオプションのフォントは、サイズおよびスタイルは設定できません。

- [エディタ]タブ

項目	説明
Bean削除時にイベント記述を削除する	Bean削除時に、削除するBeanに対応するイベント記述をソース上から削除します。
Bean削除時にターゲットBean関係を削除する	Bean削除時に、削除するBeanがターゲットBeanとなっているBean関係をソース上から削除します。

項目	説明
Bean変名時にソースを無条件に置換する	Bean変名時に、ソース上のBean名文字列を置換します。

注意

[Bean削除時にイベント記述を削除する]、[Bean削除時にターゲットBean関係を削除する]を設定した場合、元に戻す操作でイベント処理記述、Bean関係を元に戻すことはできません。Beanを削除するときは注意してください。

[Bean変名時にソースを無条件に置換する]を設定した場合、ソース上のBean名に相当する文字列を無条件に置換するため、意図しない置換をすることがあります。Bean名を変名するときは注意してください。

- [環境]タブ

項目	説明
基底フォームの検索パス一覧	基底フォームの検索パス一覧を表示します。基底フォームを読み込む場合に、この検索パス一覧の順番で検索を行います。

- [ウィザード]タブ
  Javaフォームの新規作成時にウィザードが生成するフォームの情報および各プロパティの初期値を設定します。

項目	説明
クラス情報	Swing/AWTの初期値を設定します。
アクティブフォームより取得	現在表示中のアクティブなフォームより各項目の値を取得します。

- [Bean関係の可視化]タブ
  Bean関係の可視化方法と可視化するときの色を設定します。

項目	説明
Bean関係の可視化方法	Bean関係の可視化方法を選択します。[詳細設定]ボタンによって、可視化方法の詳細設定を行うことができます。 ・ すべてのBeanに関するBean関係を可視化する:すべてのBeanに関するBean関係を可視化するとき選択します。 ・ 最後に選択したBeanに関するBean関係を可視化する:最後に選択したBeanに関するBean関係を可視化するとき選択します。 ・ すべてのBean関係を可視化しない:すべてのBean関係を可視化しないときに選択します。
色	Bean関係を可視化するときの色を設定します。[設定]ボタンをクリックすると[色の選択]ダイアログボックスが起動します。 ・ Bean関係を可視化するときの色:Bean関係を可視化するときの色を設定します。 ・ パラメタとして設定されるBean関係を可視化するときの色:パラメタとして設定されているBean関係を可視化するときの色を設定します。パラメタとして設定されているBean関係とは以下の例の「jTextField1.getText()」の部分を示します。 jLabel1.setText(jTextField1.getText()); ・ 可視化されたBean関係を選択したときの色:可視化されたBean関係を選択したときの色を設定します。

- [Bean関係の可視化方法の詳細設定]ダイアログボックス
  Beanの可視化方法の詳細を設定します。

項目	説明
指定したイベントに関するBean関係のみを可視化する	特定のイベントに関するBean関係を可視化したいとき選択します。 標準のイベント:Bean関係を可視化したいイベント名を選択します。 その他のイベント:標準のイベントに無いイベント名をコンマ、セミコロン、空白で区切って指定します。
パラメタとして設定されるBean関係を可視化する	パラメタとして設定されているBean関係を可視化したいとき選択します。パラメタとして設定されているBean関係とは以下の例の「jTextField1.getText()」の部分を示します。 jLabel1.setText(jTextField1.getText());

- [その他]タブ
  その他の項目を設定します。

項目	説明
非推奨機能を表示する	非推奨となっている以下の項目が有効となります。過去の互換性保持のために例外的に使用する機能であり、将来的に削除される可能性があるため、使用は推奨されません。 ・ クラス非継承ウィザード、および、それらの項目を表示するタブ(アプレット、フレーム、ダイアログ、パネル) ・ Beanのシリアライズ ・ Javaフォームの装飾 ・ Beanの状態の保存、および、復元 ・ フォーカス移動順定義

7.3.10.4 フォーム一覧

Javaフォーム定義で開いているフォーム一覧を表示します。リスト上で選択されているフォームをダブルクリックするか、[前面に表示]ボタンを押すとそのフォームを最前面に表示します。

7.3.11 Javaアプリケーションの動作を確認する

開発したJavaアプリケーションに応じて以下の方法でデバッグします。[デバッグ構成]ダイアログボックスはワークベンチのメニューバーから[実行] > [デバッグ構成]を選択することで開くことができます。

- [Javaアプリケーション]起動構成を使用する。
  作成したJavaアプリケーションをデバッグする場合、起動構成はJavaアプリケーションを使用します。[デバッグ構成]ダイアログボックスでは、[Javaアプリケーション]を選択します。

- [Javaアプレット]起動構成を使用する。
  Javaアプレット起動構成を使用して、アプレットをデバッグすることができます。デバッグが開始されると、アプレットビューアが起動されデバッグ操作が可能です。[デバッグ構成]ダイアログボックスでは、[Javaアプレット]を選択します。

  注意

  Swingベースのアプレット(javax.swing.JAppletクラスを継承しているアプレット)をデバッグすると、デバッグ終了時にjavax.swing.TimerQueueクラスがクラスエディタで開かれます。この動作を抑止するには、ワークベンチのメニューバーから[ウィンドウ] > [設定] > [Java] > [デバッグ]を選択し、[キャッチされない例外で実行を中断]の選択を解除します。

- [リモートJavaアプリケーション]起動構成を使用する。
  アプリケーションをデバッグモードで起動し、リモートJavaアプリケーション起動構成を使うことにより、アプリケーションをリモートデバッグすることができます。[デバッグ構成]ダイアログボックスでは、[リモートJavaアプリケーション]を選択します。

- JavaBeansのデバッグ
  ワークベンチのウィザードでJavaBeansを生成した場合は、テストドライバとしてmainメソッドが用意されています。生成直後はコメントアウトされています。mainメソッドを有効にすることにより、Javaアプリケーションとして簡単にJavaBeansをデバッグすることができます。

- アプレットをブラウザに表示させてのデバッグ
  アプレットをInternet Explorerなどのブラウザ上で動作させてデバッグする場合にはJBKプラグインを使ってリモートデバッグします。JBKランタイムをインストールしたフォルダ配下のclassesフォルダにあるjbkplugin.propertiesに、以下のように記述を追加することにより、デバッグモードでアプレットを動作させることができます。

```
(例)
jbk.plugin.vmoption=-Xrs -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=nnnn
```

参考

nnnnは、デバッガと被デバッグプログラムが通信するために使用するポート番号です。jbk.plugin.vmoption のパラメタに指定した値を、リモートデバッグする際にデバッガ側でデバッグするための通信に使うポート番号として設定する必要があります。

ポイント

作成した起動構成は、デバッグ時に毎回作成する必要はありません。同じ起動構成でデバッグする場合は、[デバッグ構成]ダイアログボックスから作成した構成ファイルを選択し、[デバッグ]をクリックしてデバッグを開始してください。

7.3.12 Javaアプリケーションを運用環境に配布する

Javaプロジェクトを運用環境向けに配布する際は、JARエクスポート機能によりJARを作成する必要があります。

JARファイルのパッケージ(JARエクスポート)

JARファイルを作成するにはエクスポートウィザードを使用します。エクスポートウィザードを起動するには、[ファイル] > [エクスポート]を選択してください。

アプレットの配備

アプレットの配備に使用するHTMLファイルに、JARファイルを記載する必要があります。

7.3.13 JDK 6を使用してJavaアプリケーションを開発する場合

Java EE 6ワークベンチでは、JDK 6を使用してJavaアプリケーションを開発することができます。

ここでは、Java EE 6ワークベンチでのJavaアプリケーションを開発する手順についてワークベンチの開発方法と異なる点を中心に説明します。

7.3.13.1 Javaアプリケーションを作成する環境を準備する

プロジェクトの作成

[新規]ウィザードから[Java] > [Javaアプリケーションプロジェクト]を選択し、Javaアプリケーションプロジェクトを作成します。

[JRE]には[JRE 実行環境を使用]を指定して[JavaSE-1.6]を選択します。

クラスパスの設定

JBKライブラリなど必要なライブラリを追加する場合は、プロジェクトのビルドパスを設定する必要があります。

ビルドパスの設定の詳細については、"9.6.1.4 クラスパスを設定する"を参照してください。

7.3.14 留意事項

アプレットの留意事項

アプレットを開発、運用する場合の留意事項について以下に説明します。

- Webブラウザの選択
  Webブラウザによって、アプレットが画面で表示されるサイズなどに若干の違いがあります。また、Webブラウザのオプション設定によって、アプレットの動作を変更できるものもあります。ブラウザのJavaScriptやHTMLのサポートレベルにも注意する必要があります。

- Javaプラグイン
  Javaプラグインとは、Webブラウザの提供するJavaVM(実行環境)を利用しないで、外付けのJavaVM(たとえば、JDKのJavaVM)を利用して、アプレットを実行する仕組みです。Javaプラグインを利用する場合、クライアントマシンにプラグインをインストールする必要があります。

- サーバアクセスの制限
  悪意をもつアプレットによる被害を防ぐために、アプレットからはローカルファイルやほかのサーバへのアクセスが禁止されています。アプレットから、データベースのサーバと通信をする機能を使う場合には、アプレットが存在するWebサーバにデータベースのサーバ環境を作成する必要があります。
  Webサーバの負荷を軽減するためには、Webサーバ上にサーブレットやEnterprise JavaBeans(以降では、EJBと略します)などのサーバアプリケーションを置き、アプレットはサーバアプリケーションと通信し、サーバアプリケーションから別のサーバ(データベースサーバなど)にアクセスするようにします。

- EJBの利用
  アプレットからは、Webサーバ上のEJBを利用することができます。
  アプレットの開発時には、Enterprise Beanのスタブをプロジェクトに追加し、プロジェクトのJARファイルに結合します。また運用時には、EJBで使用するライブラリをサーバにインストールする必要があります。

- データベースの利用
  アプレットからサーバ上のデータベースを利用する場合は、クライアントマシンにデータベースに合ったJDBCドライバをインストールする必要があります。

Beanの留意事項

Bean固有の留意事項について、以下に説明します。

- 分割ペイン(JSplitPane)
  分割ペインには、分割部分にパネルを貼る必要があります。そのため、分割ペインを貼ったときに自動でパネルが貼られます。このパネルは削除することはできません。

- スクロール(JScrollPane)
  Swingのリストボックス(JList)などは、その内容がBeanの表示領域を超えてもスクロールバーが自動的に付きません。スクロールバーを付けるためには、まずスクロールペイン(JScrollPane)を貼って、その上にSwingのリストボックス(JList)などを貼ります。これで、スクロールバーが必要なときに表示されるようになります。なお、スクロールペイン(JScrollPane)には、Beanを1つしか貼ることができません。
  AWTのBeanは、スクロールペイン(JScrollPane)に貼ってもこのスクロール動作はできません。AWTのBeanは、それ自身がスクロール機能をもつので、スクロールペインを利用する必要はありません。

- 重量コンポーネントとパネル
  重量コンポーネントを何かに貼る場合は、その貼り付け先も重量コンポーネントにすることを推奨します。たとえば、AWTボタンやAWTテキストフィールドをパネルに貼る場合、そのパネルもAWTパネルを利用します。

- 重量コンポーネントと軽量コンポーネント
  重量コンポーネントのBeanと軽量コンポーネントのBeanを重ねると、必ず重量コンポーネントのBeanが上に表示されます。

- 不可視Bean
  GUIをもたない不可視Beanの新規貼り付けでは、可視Beanと同様にコンテナやJavaフォーム上で貼り付け操作を行いますが、貼り付けたあとはJavaフォーム上に表示されず、コンポーネントツリー上にグレー表示されます。プロパティの操作は、コンポーネントツリー上でBeanを選択して行います。

- タブ(JTabbedPane)
  タブは、複数のページで構成されています。以下にページの追加、削除およびページタイトルの指定の操作について説明します。

  ー ページの追加
    タブに直接貼り付けたBeanが、タブ上で1つのページになります。一般には、1つのページは複数のBeanで構成します。そのため、各ページには、最初にパネル(JPanel)を貼り(パネルは内部に複数のBeanを貼ることができる)、その上に必要な複数のBeanを貼ります。このとき、パネルのレイアウトマネージャは必要に応じて選択します。なお、タブを新規に貼ったときに、この

パネルを貼ったページが複数自動生成されます。
ページを実際に追加するには、パネルをオブジェクトパレットから選択し、タブストリップ(ページ切替え時にクリックするページのタイトルが表示されている箇所)の位置からドラッグを開始します。Beanの貼り付けのドラッグをタブストリップの位置から開始した場合が、ページの追加操作になります。タブストリップの位置がタブの下や横にある場合も同様に、その下や横に付いたタブストリップの位置からドラッグを開始するとページが追加されます。なお、ページが一切ない場合は、どの位置からドラッグしてもページ追加となります。

- ページの削除
タブのページを削除するには、ページに貼ってあるパネルを選択して、Beanの削除操作をします。

- ページタイトルの指定
ページタイトルを指定するには、指定するページのパネルを選択し、パネルの標準プロパティ「配置条件」にタイトル文字列を指定します。

- ページ順序の変更
ページの並びを変更するには、変更するページのパネルを選択し、親コンテナへの追加順を指定します。

・ JDKのバージョン変更に伴うBeanのデザイン差異
JDKのバージョンを変更すると、Beanのデザインに差異が生じることがあります。

Javaフォーム機能の留意事項

Javaフォーム機能の留意事項について、以下に説明します。

・ 変更禁止ソースをパッケージエクスプローラ、アウトライン、ソースメニューなどから編集しないでください。
注）変更禁止ソースとは、ソース先頭の"// Graphical Editor Form"と"//@@Form Design Information start"、"//@@Form Design Information end"で囲まれた部分のソースです。

・ 変更禁止ソースの背景色は変更することができません。

・ Javaフォーム定義はプロジェクトごとに起動されます。

・ Javaフォーム、アプレット、JavaBeansのソースはリファクタリングできません。

・ ナビゲータビューなどからファイルの名前を変更しないでください。Javaフォームソースを別名保存するときは、必ずJavaフォーム定義から行ってください。変更した場合には、元の名前に戻してから開き直してください。

・ Javaフォームのクラス宣言などが破壊されて解析できない場合には、解析できない部分のソースを復元します。復元前のソースはコメントアウトされずにそのまま残ります。

・ グラフィカルエディタは、リンクファイルまたはリンクフォルダに存在するファイルを開くことができません。

・ Javaフォーム、アプレット、JavaBeansのファイルをグラフィカルエディタ以外のエディタで編集した場合、ローカルヒストリ機能などを利用して、編集前の状態に戻してからグラフィカルエディタで開き直してください。

・ Beanリストビューの"リファレンス"メニューは標準ワークベンチでは表示されません。APIリファレンスは各マニュアルを参照してください。

・ プロパティウィンドウのヘルプ表示機能について。マニュアルが存在しない項目を選択してもヘルプボタンが無効になりません。また、固有プロパティのJBKに関する項目は表示されないことがあります。これらは制限となります。

・ 対応文字コードはShift_JIS(MS932)となります。

画面制御パネル編集の留意事項

画面制御パネル編集の留意事項について、以下に説明します。

・ 画面制御パネルのファイルをリファクタリングで名前変更する場合、ファイルが編集中であっても、その時点のファイルの内容でリファクタリング処理が行われます。リファクタリングの結果は編集中のソースには反映されません。また、編集中のソースを保存すると、リファクタリングで名前変更する前のファイル名で保存されます。

共通留意事項

共通の留意事項について、以下に説明します。

・ グラフィカルエディタで編集中のファイルまたはプロジェクトを移動、名前変更、削除した場合には、グラフィカルエディタで編集を継続することができません。

- ソースにUNIXやMacなどのWindows以外の行区切り文字を含めないでください。
- 複数起動されたワークベンチからグラフィカルエディタで編集中のファイルを同時に編集しないでください。
- グラフィカルエディタで編集できるソースは、クラス宣言やメソッド宣言に以下の構文を使用していないものとなります。
  - 総称
  - 可変引数
  - 注釈(メタデータ)

第8章 データベースを操作する

ここでは、データベースを操作する機能について説明します。

注意

- 本章で説明する機能は、各製品から提供されているJDBCドライバの仕様により、正常に動作しない場合があります。データベースの内容に対する作成、更新、削除などの操作は、各製品から提供されているツールやアプリケーションの利用をお勧めします。
- HA Database Ready(NativeSQL)ではデータベースの操作が一部制限されます。詳細については、"FUJITSU Integrated System HA Database Ready 業務開発ガイド(Native SQL編)"を参照してください。

8.1 概要

データベースとデータベースを操作する機能の概要について説明します。

8.1.1 データベースとは

データベースとは、複数の利用者がデータを共有できるように、データへのアクセスを管理するソフトウェアです。データベースは階層型、ネットワーク型、オブジェクト指向などのデータモデルがありますが、最も普及しているのはリレーショナルデータベースです。リレーショナルデータベースは、データを行と列の2次元のテーブルで管理して、テーブルとテーブルとの間をリレーション(関係)で管理します。

ワークベンチでは、このリレーショナルデータベースを操作する機能を提供します。

リレーショナルデータベースに関連する用語を以下に説明します。

SQL

リレーショナルデータベースを操作できる言語です。SQLには、表の作成、変更、削除などのデータ定義文(DDL)、データの更新、削除、検索などのデータ操作文(DML)、データの操作、確定、取消などのデータ制御文(DCL)があります。
SQLは標準規格ですが、ベンダによって準拠の状況はさまざまです。

スキーマ

テーブルやビューなどの要素の構成を表すものです。

ビュー

1つまたは複数のテーブルから列を抽出した仮想的なテーブルです。

インデックス

データベースのデータ検索の効率を高めるためのものです。

制約

テーブルに入力できるデータの条件を定義できるものです。主キーや外部キーなどの制約があります。

8.1.2 データベースを操作する機能

ワークベンチには、データベースを操作する機能が含まれています。

データベースへの接続

データベースに接続するためには、接続プロファイルを作成します。接続するとデータベースの内容を参照することができます。
詳細は、"8.3.1 データベースに接続する"を参照してください。

データベースの内容を参照

データベースに定義しているテーブルやビューなどの要素を参照できます。データベースの内容は[データソースエクスプローラ]ビューで参照できます。
詳細は、"8.3.2 データベースの内容を参照する"を参照してください。

SQLの実行

SQLファイルに記述したSQLを実行することができます。実行した結果は[SQLの結果]ビューで確認できます。
詳細は、"8.3.3 SQLを実行する"を参照してください。

テーブルの作成

標準のワークベンチではテーブルを作成するSQL文をダイアログボックスで作成することができます。このSQL文を実行すると、テーブルが作成できます。
詳細は、"8.3.4 テーブルを作成する"を参照してください。

テーブルの削除

標準のワークベンチではテーブルを削除するSQL文を作成することができます。このSQL文を実行すると、テーブルが削除できます。
詳細は、"8.3.5 テーブルを削除する"を参照してください。

テーブルのデータの更新

標準のワークベンチではテーブルのデータをテーブルデータエディタで編集することができます。
詳細は、"8.3.6 テーブルのデータを更新する"を参照してください。

8.1.3 制限事項

データベースを操作する機能では、マニュアルに記載された事項のうち、使用を制限しているものがあります。

項番	制限事項
1	利用するJDK/JREのバージョンに合わせたJDBCドライバを利用してください。

8.2 入門

ここでは、ワークベンチでデータベースを操作する手順を紹介します。

8.2.1 作成するデータベース

データベースに国の総人口を格納したテーブルを作成します。テーブルのカラムはID、国名および総人口で、IDを主キーにします。また、データは総人口の多い順番に格納します。このテーブルを作成するための手順を紹介します。

8.2.2 開発の流れ

ここでは、以下のようにデータベースの操作を進めます。

1. データベースの準備

   - データベースの環境作成
     データベースのサーバが必要です。ここで使用するデータベースのサーバ側の環境について説明します。

   - JDBCドライバのインストール
     ワークベンチからデータベースにアクセスするためには、JDBCドライバのインストールが必要です。

2. データベースへの接続
   以下の手順でデータベースへの接続を行います。

   ー パースペクティブの切り替え
     データベース開発用のパースペクティブに切り替えます。

   ー 接続プロファイルの作成
     準備したデータベースの環境用の接続プロファイルを作成します。

   ー データベースとの接続
     定義した接続プロファイルでデータベースに接続します。

3. テーブルの作成
   以下の手順でデータベーステーブルの作成を行います。

   ー パースペクティブの切り替え
     SQLファイルを作成するためにパースペクティブを切り替えます。

   ー プロジェクトの作成
     SQLファイルを格納するためのプロジェクトを作成します。

   ー SQLファイルの作成
     ウィザードでSQLファイルを作成します。

   ー SQLファイルの編集
     SQLファイルにテーブルを作成するSQL文を記述します。

   ー SQLの実行
     SQLファイルに記述されているSQL文を実行します。

4. テーブルの確認

   ー パースペクティブの切り替え
     データベースの定義情報を確認するためにパースペクティブを切り替えます。

   ー テーブル定義の確認
     作成したテーブルの定義情報をデータソースエクスプローラビューで確認します。

   ー テーブルデータの参照
     テーブルデータエディタで、テーブルのデータを確認します。

8.2.3 開発手順

以下に、実際にデータベースのテーブルを作成する手順を説明します。

1) データベースの準備

2) データベースへの接続

3) テーブルの作成

4) テーブルの確認

1) データベースの準備

国名と総人口のデータを格納するデータベースを作成します。データベースはSymfoware Serverを使用します。

1-1) データベースの環境作成

Symfoware Serverに以下の設定項目のような環境を作成します。設定項目と異なる環境の場合は、その設定項目は以降の説明では読み替えてください。設定方法の詳細はSymfoware Serverのマニュアルを参照してください。

項目	内容
サーバ名	myhost
RDBシステム	rdbsys1
データベース	COUNTRYDATA
データベーススペース	COUNTRYDATA
リモート接続のポート番号	2050
ユーザ名	データベースにアクセスできるユーザ名
パスワード	ユーザ名に対するパスワード
スキーマ	STUDIO

1-2) JDBCドライバのインストール

ワークベンチからSymfoware Serverに接続するために、Symfoware JDBCドライバをインストールします。インストール方法についてはSymfoware Serverのマニュアルを参照してください。

2) データベースへの接続

2-1) パースペクティブの切り替え

メニューバーから[ウィンドウ] > [パースペクティブを開く] > [その他]を選択すると、[パースペクティブを開く]ダイアログボックスが表示されます。[パースペクティブを開く]ダイアログボックスから[データベース開発]を選択して、[OK]をクリックします。

2-2) 接続プロファイルの作成

データソースエクスプローラビューから[データベース]を選択して、右クリックします。コンテキストメニューから[新規]を選択すると、[新規接続プロファイル]ウィザードが表示されます。

[ウィザード選択]ページの[接続プロファイルタイプ]リストから[Symfoware]を選択します。以下の情報を設定後、[次へ]をクリックしてください。

設定項目	設定内容
名前	Symfoware_Studio

[ドライバおよび接続の詳細の指定]ページが表示されます。

[ドライバ]の右にある(新しいドライバ定義)をクリックすると、[新規ドライバ定義]ダイアログボックスが表示されます。[名前/種類]タブの[使用可能なドライバテンプレート]リストから[データベース] > [Symfoware Server JDBC ドライバ]を選択します。その他の項目は以下となるように確認、入力してください。以下の情報を設定後、[OK]をクリックしてください。

[名前/種類]タブ

設定項目	設定内容
ドライバ名	Symfoware Server JDBC ドライバ

[JARの一覧]タブ

設定項目	設定内容
ドライバファイル	C:¥SFWCLNT¥JDBC¥fjjdbc¥lib¥fjsymjdbc2.jar （パス名はJDBCドライバのインストールフォルダに合わせて変更してください）

[ドライバおよび接続の詳細の指定]ページの[プロパティ]グループの項目を以下のように設定します。[次へ]をクリックすると[要約]ページが表示されます。

[プロパティ]グループの[一般]タブ

設定項目	設定内容
データベース名	COUNTRYDATA
ホスト	myhost
ポート番号	2050
ユーザ名	データベースに接続するためのユーザ名
パスワード	ユーザ名に対するパスワード

[要約]ページでは、入力した情報を確認することができます。

以下の情報を確認後、[完了]をクリックしてください。

確認項目	確認内容
名前	Symfoware_Studio
説明	なし
開始時に自動接続	false
終了時に自動接続	true
データベース	COUNTRYDATA
ホスト	myhost
ポート番号	2050
ユーザ名	データベースに接続するためのユーザID
パスワードの保存	false
URL	jdbc:symford://myhost:2050/COUNTRYDATA

2-3) データベースとの接続

デフォルトでは、[新規接続プロファイル]ウィザードの[ドライバおよび接続の詳細の指定]ページで[ウィザード完了時に接続]がチェックされています。この状態ではウィザード完了時に自動的にデータベースへ接続されます。

手動で接続する場合には、データソースエクスプローラビューから作成した接続プロファイル[Symfoware_Studio]を選択します。右クリックするとコンテキストメニューが表示されるので、コンテキストメニューから[接続]を選択します。

3) テーブルの作成

3-1) パースペクティブの切り替え

[パースペクティブを開く]ダイアログボックスで[リソース]を選択して、[OK]をクリックします。

3-2) プロジェクトの作成

SQLファイルを置くためのプロジェクトを作成します。既存のプロジェクトにSQLファイルを置く場合は、プロジェクトを作成する必要はありません。メニューバーから[ファイル] > [新規] > [プロジェクト]を選択すると、[新規プロジェクト]ウィザードが表示されます。新規プロジェクトウィザードから[一般] > [プロジェクト]を選択して、[次へ]をクリックします。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
プロジェクト名	DBSample

3-3) SQLファイルの作成

テーブルを作成するSQLファイルをプロジェクトの直下に作成します。プロジェクトエクスプローラビューで[DBSample]プロジェクトを選択して、右クリックします。コンテキストメニューから[新規] > [その他]を選択します。[新規]ウィザードから[SQL開発] > [SQLファイル]を選択して、[次へ]を選択すると、[新規SQLファイル]ウィザードが表示されます。

以下の設定項目を確認、入力してください。以下の情報を設定後、[完了]をクリックしてください。

設定項目	設定内容
ファイル名	countrydata.sql
データベースサーバタイプ	Symfoware_9
接続プロファイル名	Symfoware_Studio
データベース名	COUNTRYDATA

3-4) SQLファイルの編集

作成したSQLファイルに以下のようにテーブル定義、インデックス定義、データ挿入を記述します。

SQLファイルの定義

```
CREATE TABLE STUDIO.C_DATA (
ID      INTEGER  PRIMARY KEY  NOT NULL,
C_NAME   VARCHAR(80),
T_POP    NUMERIC(10)
);

CREATE INDEX STUDIO.C_DATA.ID KEY(ID);

INSERT INTO STUDIO.C_DATA VALUES(1, '中国', 1330000000);
INSERT INTO STUDIO.C_DATA VALUES(2, 'インド', 1140000000);
INSERT INTO STUDIO.C_DATA VALUES(3, 'アメリカ', 300000000);
INSERT INTO STUDIO.C_DATA VALUES(4, 'インドネシア', 230000000);
INSERT INTO STUDIO.C_DATA VALUES(5, 'ブラジル', 190000000);
INSERT INTO STUDIO.C_DATA VALUES(6, 'パキスタン', 160000000);
INSERT INTO STUDIO.C_DATA VALUES(7, 'バングラデシュ', 150000000);
INSERT INTO STUDIO.C_DATA VALUES(8, 'ロシア', 140000000);
INSERT INTO STUDIO.C_DATA VALUES(9, 'ナイジェリア', 140000000);
INSERT INTO STUDIO.C_DATA VALUES(10, '日本', 130000000);
```

3-5) SQLの実行

SQLファイルエディタで作成したSQLファイルを開いて、エディタ領域上で右クリックします。コンテキストメニューから[すべて実行]を選択します。

SQLの実行結果は、[SQLの結果]ビューで確認します。

4) テーブルの確認

4-1) パースペクティブの切り替え

[パースペクティブを開く]ダイアログボックスで[データベース開発]を選択して、[OK]をクリックします。

4-2) テーブル定義の確認

作成したテーブルはデータソースエクスプローラビューで確認できます。内容を更新するには、接続プロファイル[Symfoware_Studio]を選択して、右クリックします。コンテキストメニューから[更新]を選択します。

4-3) テーブルのデータ参照

作成したテーブルのデータをテーブルデータエディタで参照することができます。テーブルデータエディタは、データソースエクスプローラビューで[データベース] > [Symfoware_Studio] > [COUNTRYDATA] > [スキーマ] > [STUDIO] > [テーブル] > [C_DATA]を選択して、右クリックでコンテキストメニューを表示します。コンテキストメニューの[データ] > [編集]を選択することで起動します。

ポイント

エディタのセルを編集することで、値を変更することができます。またセルを選択して、コンテキストメニューから行の削除や行の追加などの編集が可能です。

8.3 タスク

ここでは、ワークベンチでデータベースを操作する方法について、目的(タスク)ごとに説明します。

8.3.1 データベースに接続する

データベースに接続したい場合は、接続プロファイルを作成します。以下に接続プロファイルを作成してからデータベースに接続する方法を説明します。

データベース開発パースペクティブを開く

[データソースエクスプローラ]ビューを表示するため、[データベース開発]パースペクティブを開きます。

接続プロファイルの作成

[データソースエクスプローラ]ビューにある[データベース]を選択しコンテキストメニューから [新規]メニューを選択します。[新規接続プロファイル]ウィザードが表示されます。このウィザードでデータベースと接続するための設定を行います。ウィザードの指定内容については、以下を参考にしてください。

注意

Symfoware Server(Openインタフェース) V12と接続する場合、[新規ドライバ定義]ダイアログボックスの[名前/種類]タブでは、"Symfoware Server(Openインタフェース) 12 JDBC ドライバ"を指定してください。バージョンに表示されている"9(Openインタフェース)"は、JDBCドライバのバージョンになります。

- 接続プロファイルタイプ
  接続したいデータベースに合わせた接続プロファイルタイプを選択します。

- ドライバ
  ドロップダウンから定義されているドライバを選択します。定義されていない場合には、右にある  (新しいドライバ定義)をクリック

し、以下のように定義を追加します。各データベースのJDBCドライバの情報は、"8.3.7.1 JDBCドライバ"を参照してください。

1. [新規ドライバ定義]ダイアログボックスの[名前/種類]タブで、接続に使用するドライバのテンプレートを選択します。
2. [JARの一覧]タブではドライバファイルにJDBCドライバのファイルが格納されている場所を指定します。(暫定的に追加されているドライバファイルがある場合には、正しいパスを指定します。)
3. [プロパティ]タブではデフォルトとして使用されるプロパティを必要であれば変更します。
4. [OK]をクリックして、[新規ドライバ定義]ダイアログボックスを閉じます。

- 接続プロファイルのプロパティ
  選択したドライバ種別に応じて接続プロファイルのプロパティを設定します。各データベースの接続プロパティのプロファイル情報は、"8.3.7.2 接続プロファイルのプロパティ"を参照してください。

- 接続のテスト
  ドライバとプロパティを設定して[接続のテスト]をクリックすると、データベースとの接続テストが行えます。

データベースの接続

[データソースエクスプローラ]ビューにある[データベース]の配下にある接続プロファイルを選択して、コンテキストメニューから[接続]を選択します。
データベースに接続し、データベースの内容が表示されます。

8.3.2 データベースの内容を参照する

データベースに定義しているテーブルやビューなどの要素を、[データソースエクスプローラ]ビューで参照することができます。

データベースの接続

データベースに接続します。データベースの接続の詳細については、"8.3.1 データベースに接続する"を参照してください。

データベースの内容を参照

[データソースエクスプローラ]ビューにある接続プロファイルの配下に、データベースの内容がツリー構造で表示されます。データベースに定義している以下の要素を参照することができます。

- スキーマ
- テーブル
- 列
- インデックス
- 制約
- ビュー

注意

データベースにスキーマが存在していて、かつ、そのスキーマに属するテーブルやビューが存在しない場合は、スキーマの要素がツリー構造に表示されません。

データベースの要素を操作

[データソースエクスプローラ]ビューは、データベースの要素を操作する起点になっています。例えば、テーブルのデータを編集したい場合は、テーブルを選択して、コンテキストメニューから[データ] > [編集]を選択します。

8.3.3 SQLを実行する

データベースにSQLを発行するには、SQLファイルを作成して、そのSQL文を実行します。

SQLファイルの作成

[新規]ウィザードから[SQL開発] > [SQLファイル]を選択します。ウィザードの指定内容については、以下を参考にしてください。

- データベースサーバタイプ
  データベースの製品とバージョンの組合せを選択します。

- 接続プロファイル名
  データベースサーバタイプの接続プロファイルがリストに一覧されるので、その中から選択します。

- データベース名
  接続プロファイルで接続しているデータベースがリストに一覧されるので、その中から選択します。

- [作成]ボタン
  接続プロファイルを作成できます。接続プロファイルの詳細については、"8.3.1 データベースに接続する"を参照してください。

ポイント

データベースサーバタイプ、接続プロファイル名およびデータベース名を選択しなくても、SQLファイルは作成できます。ただし、それらを設定しない場合には、エディタの支援機能で使えないものがあります。

SQLファイルの編集

SQLファイルをSQLファイルエディタで開きます。SQLファイルにSQL文を記述します。
データベースに接続している場合には、コンテンツアシストによりテーブルやカラムを入力することができます。

ポイント

SQL文とSQL文の区切りは、セミコロン(;)を指定します。この区切りがSQL文により解析できない場合があります。このような場合は、SQL文の最後を改行して、次の行の先頭にセミコロンを指定してください。または、SQL文を1行で記述してください。

SQLファイルの実行

SQLファイルエディタにある[接続プロファイル]グループを確認します。型、名前、データベースのコンボボックスに値が設定されていてかつ、状況が接続であることを確認してください。
SQLファイルエディタのコンテキストメニューから次のどれかを選択します。SQLファイルに記述されたSQL文が実行され、結果が[SQLの結果]ビューに表示されます。

- [すべて実行]
  すべてのテキストを;(セミコロン)やGOのデリミタで分割して、SQL文を実行します。

- [選択したテキストの実行]
  選択したテキストを;(セミコロン)やGOのデリミタで分割して、SQL文を実行します。

- [選択したテキストを1つのステートメントとして実行]
  選択したテキストを1つのSQL文として実行します。

- [カレントテキストの実行]
  現在のカーソル位置にあるSQL文を実行します。

ポイント

[カレントテキストの実行]の動作は、ワークベンチの設定画面の[データ管理] > [SQL開発] > [SQLエディタ]にある[カレントテキストの実行]の設定に従います。デフォルトの設定は[カレント行の実行]になっています。

- デリミタの間にあるSQLの実行
  現在のカーソル位置で前後のデリミタの間にあるSQLを実行します。

- カレント行の実行
  カーソルがある行のテキストを実行します。

- 空白行の間にあるSQLの実行
  現在のカーソル位置で前後の空白行の間にあるSQLを実行します。

8.3.4 テーブルを作成する

標準のワークベンチではテーブルを作成するSQL文をダイアログボックスで作成することができます。

データベースの接続

データベースに接続します。データベースの接続の詳細については、"8.3.1 データベースに接続する"を参照してください。

SQL文の生成

テーブルを作成するSQL文を生成します。[データソースエクスプローラ]ビューにある[テーブル]を選択してコンテキストメニューから[新規テーブル]を選択します。ダイアログボックスの指定内容については、以下を参考にしてください。

- オプション
  選択したオプションに対応したSQL(DDL)文が生成されます。

- 列
  テーブル名を設定します。[列の追加]ボタンで列を追加してから、列の名前やデータ型を設定します。

- 主キー
  列の中から主キーを選択します。また、主キーに名前を設定します。

SQLファイルの実行

SQLファイルエディタにSQL文が生成されます。SQLファイルエディタでSQL文を実行します。SQLファイルの実行は、"8.3.3 SQLを実行する"を参照してください。

SQLを実行すると、結果が[SQLの結果]ビューに表示されます。[データソースエクスプローラ]ビューを更新して、作成したテーブルを確認することができます。

8.3.5 テーブルを削除する

標準のワークベンチではテーブルを削除するSQL文を作成することができます。

データベースの接続

データベースに接続します。データベースの接続の詳細については、"8.3.1 データベースに接続する"を参照してください。

SQL文の生成

テーブルを削除するSQL文を生成します。[データソースエクスプローラ]ビューにある[テーブル] の配下で表示したいテーブルを選択してコンテキストメニューから[削除]を選択します。

SQLファイルの実行

SQLファイルエディタにSQL文が生成されます。SQLファイルエディタでSQL文を実行します。SQLファイルの実行は、"8.3.3 SQLを実行する"を参照してください。

SQLを実行すると、結果が[SQLの結果]ビューに表示されます。[データソースエクスプローラ]ビューを更新して、削除したテーブルを確認することができます。

8.3.6 テーブルのデータを更新する

標準のワークベンチではテーブルのデータを編集できます。各データベースで編集可能なデータ型は、"8.3.7.3 編集可能なデータ型"を参照してください。

データベースの接続

データベースに接続します。データベースの接続の詳細については、"8.3.1 データベースに接続する"を参照してください。

テーブルのデータを編集

以下の手順でテーブルのデータを編集します。

1. [データソースエクスプローラ]ビューにある[テーブル]の配下で表示したいテーブルを選択して、コンテキストメニューから [データ] > [編集]を選択します。
2. テーブルデータエディタでデータを編集します。データの値を変更したり、コンテキストメニューで行を挿入したりできます。
3. メニューにある[ファイル] > [保存]でデータを保存します。データの保存に成功したかを、[SQLの結果]ビューで確認できます。

ポイント

行の挿入はできるが、行の削除や更新ができない場合があります。
行の削除や更新の場合は、SQL文のDELETEやUPDATEに、対象の行を特定するWHERE句が付けられます。テーブルに主キーが存在する場合は、このWHERE句に主キーの列を指定します。しかし、テーブルに主キーがない場合は、このWHERE句にすべての列を指定します。このため、テーブルの列がWHERE句に指定できないデータ型の場合にエラーとなり、行の削除や更新に失敗します。

8.3.6.1 データを編集する

テーブルデータエディタは、テーブルのデータを編集できます。このエディタで編集できるデータを以下に示します。

- 文字
  列が文字を格納するデータ型の場合は、文字でデータを設定できます。

- 数値
  列が数値を格納するデータ型の場合は、数値でデータを設定できます。

- 浮動小数点
  列が浮動小数点を格納するデータ型の場合は、浮動小数点でデータを設定できます。

- 日時
  列が日時を格納するデータ型の場合は、次の書式でデータを設定できます。DATE型は"yyyy-mm-dd"の形式で設定します。TIME型は"hh:mm:ss"の形式で設定します。TIMESTAMP型は"yyyy-mm-dd hh:mm:ss.ffffffff"の形式で設定します。

- バイナリ
  列がバイナリを格納するデータ型の場合は、2桁の16進数でデータを設定できます。

ポイント

編集できないデータの場合は、null以外の値を設定することができません。

8.3.6.2 データの抽出とロード

テーブルのデータをファイルに抽出したり、ファイルのデータをテーブルにロードしたりすることができます。

テーブルのデータを抽出

テーブルのデータをファイルに保存することができます。以下の手順でデータの抽出を行います。

1. [データソースエクスプローラ]ビューにある[テーブル]の配下で表示したいテーブルを選択して、コンテキストメニューから [データ] > [抽出]を選択します。
2. [データの抽出]ダイアログボックスで、保存するファイルとフォーマットを指定します。このファイルは、UTF-8でエンコードされたテキストファイルです。

テーブルのデータをロード

ファイルのデータをテーブルにロードすることができます。以下の手順でデータのロードを行います。

1. [データソースエクスプローラ]ビューにある[テーブル]の配下で表示したいテーブルを選択して、コンテキストメニューから [データ] > [ロード]を選択します。
2. [データのロード]ダイアログボックスで、ロードするファイルとフォーマットを指定します。このファイルは、UTF-8でエンコードされたテキストファイルである必要があります。

注意

Java EE 6ワークベンチにおいて、Symfoware ServerのNCHAR型、NCHAR VARYING型のデータをロードする場合、ロードするファイルのエンコードは、ワークスペースのテキストファイルエンコードと同一である必要があります。

テキストファイルエンコードはJava EE 6ワークベンチの[ウィンドウ] > [設定] > [一般] > [ワークスペース]から確認できます。

8.3.7 サポートするデータベースの情報

この機能がサポートするデータベースに関連する情報を示します。詳細は各データベースのマニュアルを参照してください。

- JDBCドライバ
- 接続プロファイルのプロパティ
- 編集可能なデータ型

8.3.7.1 JDBCドライバ

データベースとの接続は、JDBCドライバのDriverManagerクラスを利用します。各データベースの情報を以下に示します。

注意

1つのドライバファイルに対して複数のドライバ定義をしないでください。1つのドライバ定義はデータベースへの接続に成功しますが、別のドライバ定義はデータベースへの接続に失敗します。

Symfoware Server

Symfoware Serverに接続する場合は、ドライバタイプをネイティブブリッジ(タイプ2)の接続形態にしてください。

リモートアクセス(RDB2_TCP連携)にする場合のドライバ定義を以下に示します。

プロパティ	設定
ドライバファイル (標準のワークベンチ)	[JDBC2.Xドライバ] インストールフォルダ¥fjjdbc¥lib¥fjsymjdbc2.jar [JDBC4.Xドライバ] インストールフォルダ¥fjjdbc¥lib¥fjsymjdbc4.jar
ドライバファイル (Java EE 6ワークベンチ)	[JDBC4.Xドライバ] インストールフォルダ¥fjjdbc¥lib¥fjsymjdbc4.jar
ドライバクラス	com.fujitsu.symfoware.jdbc.SYMDriver
接続URL	jdbc:symford://hostname:2050/dbname

Oracle

Oracle Thinドライバにする場合のドライバ定義を以下に示します。

プロパティ	設定
ドライバファイル (標準のワークベンチ)	[Oracle Database 11g JDBCドライバ] JDBCドライバ格納先ディレクトリ¥ojdbc6.jar JDBCドライバ格納先ディレクトリ¥orai18n.jar
ドライバファイル (Java EE 6ワークベンチ)	[Oracle Database 11g Release 2 JDBCドライバ] JDBCドライバ格納先ディレクトリ¥ojdbc6.jar JDBCドライバ格納先ディレクトリ¥orai18n.jar
ドライバクラス	oracle.jdbc.OracleDriver
接続URL	jdbc:oracle:thin:@server:1521:db

SQL Server

Microsoft(R) JDBCドライバはMicrosoft(R) SQL Server(TM)には同梱されていません。Microsoft Corporationのホームページより、SQL Server(TM) 2005 JDBC Driver 1.2以降をダウンロードし、インストールして使用してください。

ドライバ定義の例を以下に示します。

プロパティ	設定
ドライバファイル (標準のワークベンチ)	インストールフォルダ¥sqljdbc_<version>¥<location>¥sqljdbc4.jar <version>: Microsoft(R) SQL Server(TM) 2005 JDBC Driver 1.2の場合は"1.2" <version>: Microsoft(R) SQL Server(TM) JDBC Driver 2.0の場合は"2.0" <version>: Microsoft(R) SQL Server(TM) JDBC Driver 3.0の場合は"3.0" <version>: Microsoft(R) JDBC Driver 4.0 for SQL Server(TM)の場合は"4.0" <location>: 日本語版の場合は"jpn"、英語版の場合は"enu"
ドライバファイル (Java EE 6ワークベンチ)	インストールフォルダ¥sqljdbc_<version>¥<location>¥sqljdbc4.jar <version>: Microsoft(R) SQL Server(TM) JDBC Driver 3.0の場合は"3.0" <version>: Microsoft(R) JDBC Driver 4.0 for SQL Server(TM)の場合は"4.0" <location>: 日本語版の場合は"jpn"、英語版の場合は"enu"
ドライバクラス	com.microsoft.sqlserver.jdbc.SQLServerDriver
接続URL	jdbc:sqlserver://localhost:1433;databaseName=pubs

PowerGres Plus

接続する場合は、PowerGres Plusに同梱されているドライバをローカル環境にコピーして使用してください。

ドライバ定義の例を以下に示します。

プロパティ	設定
ドライバファイル (標準のワークベンチ)	以下のファイルをローカル環境にコピーして使用します。 [PowerGres Plusインストール先のドライバ位置] /usr/local/pgsqlplus/share/java/postgresql_jdbc4.jar
ドライバファイル (Java EE 6ワークベンチ)	以下のファイルをローカル環境にコピーして使用します。 [PowerGres Plusインストール先のドライバ位置] /usr/local/pgsqlplus/share/java/postgresql_jdbc4.jar
ドライバクラス	org.postgresql.Driver
接続URL	jdbc:postgresql://hostname:5432/<データベース名>

Derby(Java DB)

Derby(Java DB)に接続する場合は、本製品に同梱されているドライバを使用できます。標準のワークベンチからはDerby(Java DB) V10.4に、Java EE 6ワークベンチからはDerby(Java DB) V10.8に接続できます。

ドライバ定義の例を以下に示します。

プロパティ	設定
ドライバファイル (標準のワークベンチ)	[Derby 10.4 用 Derby 組み込み JDBC ドライバ] <Interstage Studioインストールフォルダ>¥APS¥F3FMisjee¥javadb¥lib¥derby.jar [Derby クライアント JDBC ドライバ] <Interstage Studioインストールフォルダ>¥APS¥F3FMisjee¥javadb¥lib¥derbyclient.jar
ドライバファイル (Java EE 6ワークベンチ)	[Derby 10.8 用 Derby 組み込み JDBC ドライバ] <Interstage Studioインストールフォルダ>¥APS¥F3FMisje6¥javadb¥lib¥derby.jar [Derby クライアント JDBC ドライバ] <Interstage Studioインストールフォルダ>¥APS¥F3FMisje6¥javadb¥lib¥derbyclient.jar
ドライバクラス	[Derby 10.4/10.8 用 Derby 組み込み JDBC ドライバ] org.apache.derby.jdbc.EmbeddedDriver [Derby クライアント JDBC ドライバ] org.apache.derby.jdbc.ClientDriver
接続URL	jdbc:derby:<データベースロケーション>;create=true

HA Database Ready(NativeSQL)

HA Database Ready(NativeSQL)に接続する場合は、ドライバタイプをネイティブブリッジ(タイプ2)の接続形態にしてください。

リモートアクセス(RDB2_TCP連携)にする場合のドライバ定義を以下に示します。

プロパティ	設定
ドライバファイル (標準のワークベンチ)	[JDBC2.Xドライバ] クライアントのインストールフォルダ¥fjjdbc¥lib¥fjsymjdbc2.jar [JDBC4.Xドライバ] クライアントのインストールフォルダ¥fjjdbc¥lib¥fjsymjdbc4.jar
ドライバファイル (Java EE 6ワークベンチ)	[JDBC4.Xドライバ] クライアントのインストールフォルダ¥fjjdbc¥lib¥fjsymjdbc4.jar
ドライバクラス	com.fujitsu.symfoware.jdbc.SYMDriver

プロパティ	設定
接続URL	jdbc:symford://hostname:26551/dbname

HA Database Ready(OpenSQL)

HA Database Ready(OpenSQL)に接続する場合は、HA Database Ready(OpenSQL)に同梱されているドライバをローカル環境にコピーして使用してください。

ドライバ定義の例を以下に示します。

プロパティ	設定
ドライバファイル (標準のワークベンチ)	[JDBC4.Xドライバ] クライアントのインストールフォルダ¥OICL32¥JDBC¥lib¥postgresql-jdbc4.jar ※64bitの場合はOICL64フォルダ配下
ドライバファイル (Java EE 6ワークベンチ)	[JDBC4.Xドライバ] クライアントのインストールフォルダ¥OICL32¥JDBC¥lib¥postgresql-jdbc4.jar ※64bitの場合はOICL64フォルダ配下
ドライバクラス	org.postgresql.Driver
接続URL	jdbc:postgresql://hostname:5432/<データベース名>

8.3.7.2 接続プロファイルのプロパティ

新規接続プロファイル作成時に設定するプロパティの内容について以下に示します。

Symfoware Server

プロパティ	内容
データベース	データベース名
ホスト	データベースのサーバ名あるいはIPアドレス
ポート番号	データベース接続に使用するポート番号
ユーザ名	データベース接続に使用するユーザ名
パスワード	データベース接続に使用するパスワード

Oracle

プロパティ	内容
SID	データベースのSID
ホスト	データベースのサーバ名あるいはIPアドレス
ポート番号	データベース接続に使用するポート番号
ユーザ名	データベース接続に使用するユーザ名
パスワード	データベース接続に使用するパスワード

SQL Server

プロパティ	内容
データベース	データベース名
ホスト	データベースのサーバ名あるいはIPアドレス
ポート番号	データベース接続に使用するポート番号

プロパティ	内容
ユーザ統合認証	データベース接続にWindows認証を用いる場合にチェックします
ユーザ名	データベース接続に使用するユーザ名。ユーザ統合認証を使用しない場合に用います
パスワード	データベース接続に使用するパスワード。ユーザ統合認証を使用しない場合に用います

注意

SQL Serverの接続プロファイルで[ユーザ統合認証]を用いるには、SQL ServerのJDBCドライバの中で提供されているsqljdbc_auth.dll(x86版)が必要になります。sqljdbc_auth.dll(x86版)を格納してあるフォルダ（例: SQL Server JDBCドライバのインストールフォルダ¥sqljdbc_1.2¥jpn¥auth¥x86）をあらかじめ環境変数PATHに追加しておいてください。

PowerGres Plus

プロパティ	内容
データベース	データベース名
URL	データベース接続に使用するURL
ユーザ名	データベース接続に使用するユーザ名
パスワード	データベース接続に使用するパスワード

Derby(Java DB)

プロパティ	内容
データベースロケーション	データベースを配置するフォルダ名
ユーザ名	データベース接続に使用するユーザ名（省略可能）
パスワード	データベース接続に使用するパスワード（省略可能）

HA Database Ready(NativeSQL)

プロパティ	内容
データベース	データベース名
ホスト	データベースのサーバ名あるいはIPアドレス
ポート番号	データベース接続に使用するポート番号
ユーザ名	データベース接続に使用するユーザ名
パスワード	データベース接続に使用するパスワード

HA Database Ready(OpenSQL)

プロパティ	内容
データベース	データベース名
URL	データベース接続に使用するURL
ユーザ名	データベース接続に使用するユーザ名
パスワード	データベース接続に使用するパスワード

8.3.7.3 編集可能なデータ型

テーブルデータエディタ、データの抽出およびデータのロードで、編集可能なデータ型を以下に示します。

Symfoware Server

データ型	説明
CHARACTER	固定長の文字列型
CHARACTER VARYING	可変長の文字列型
NATIONAL CHARACTER	固定長の各国語文字列型
NATIONAL CHARACTER VARYING	可変長の各国語文字列型
NUMERIC	真数型
DECIMAL	真数型
INTEGER	真数型
SMALLINT	真数型
REAL	概数型
DOUBLE PRECISION	概数型
DATE	日付の日時型
TIME	時刻の日時型
TIMESTAMP	日時型
BINARY LARGE OBJECT	バイナリ型

Oracle

データ型	説明
CHAR	固定長の文字列型
VARCHAR2	可変長の文字列型
NCHAR	固定長の各国語文字列型
NVARCHAR2	可変長の各国語文字列型
NUMBER	数値データ型
FLOAT	浮動小数点データ型
LONG	可変長の文字列型
LONG RAW	可変長のバイナリ型
RAW	可変長のバイナリ型
DATE	日時型、ただし時刻は編集できない。
BLOB	バイナリ型
CLOB	文字列型
NCLOB	各国語文字列型

注意

BLOB、CLOB、LONG、LONG RAWおよびNCLOB型の列があり、かつ、主キーの列がないテーブルは、データの更新やデータの削除ができません。

SQL Server

データ型	説明
bit	真理値

データ型	説明
bigint	int型
int	int型
smallint	int型
tinyint	int型
decimal	decimal型
numeric	numeric型
money	money型
smallmoney	smallmoney型
float	概数型
real	概数型
datetime	datetime型
smalldatetime	smalldatetime型
char	固定長の文字列型
varchar	可変長の文字列型
nchar	固定長の各国語文字列型
nvarchar	可変長の各国語文字列型
binary	固定長のバイナリ型
varbinary	可変長のバイナリ型
uniqueidentifier	GUID

PowerGres Plus

データ型	説明
smallint	狭範囲の整数
integer	通常使用する整数
bigint	広範囲整数
decimal	ユーザ指定精度、正確
numeric	ユーザ指定精度、正確
real	可変精度、不正確
double precision	可変精度、不正確
serial	自動増分整数
bigserial	広範囲自動増分整数
character varying,varchar	上限付き可変長文字列
character,char	空白でパッドされた固定長文字列
text	制限なし可変長文字列
bytea	可変長のバイナリ文字列
date	日付のみ
timestamp	日付と時刻両方
timestamp with time zone	日付と時刻両方、時間帯付き
time	その日の時刻のみ

データ型	説明
time with time zone	その日の時刻のみ、時間帯付き
boolean	boolean型
oid	オブジェクト識別子

Derby(Java DB)

データ型	説明
CHAR	固定長の文字列型
CHAR FOR BIT DATA	固定長のビット列型
VARCHAR	可変長の文字列型
VARCHAR FOR BIT DATA	可変長のビット列型
NUMERIC	真数型
DECIMAL	真数型
INTEGER	真数型
SMALLINT	真数型
BIGINT	真数型
REAL	概数型
DOUBLE PRECISION,DOUBLE	概数型
DATE	日付の日時型
TIME	時刻の日時型
TIMESTAMP	日時型

HA Database Ready(NativeSQL)

データ型	説明
CHARACTER	固定長の文字列型
CHARACTER VARYING	可変長の文字列型
NATIONAL CHARACTER	固定長の各国語文字列型
NATIONAL CHARACTER VARYING	可変長の各国語文字列型
NUMERIC	真数型
DECIMAL	真数型
INTEGER	真数型
SMALLINT	真数型
REAL	概数型
DOUBLE PRECISION	概数型
DATE	日付の日時型
TIME	時刻の日時型
TIMESTAMP	日時型
BINARY LARGE OBJECT	バイナリ型

HA Database Ready(OpenSQL)

データ型	説明
smallint	狭範囲の整数
integer	通常使用する整数
bigint	広範囲整数
decimal	ユーザ指定精度、正確
numeric	ユーザ指定精度、正確
real	可変精度、不正確
double precision	可変精度、不正確
serial	自動増分整数
bigserial	広範囲自動増分整数
character varying,varchar	上限付き可変長文字列
character,char	空白でパッドされた固定長文字列
text	制限なし可変長文字列
bytea	可変長のバイナリ文字列
date	日付のみ
timestamp	日付と時刻両方
timestamp with time zone	日付と時刻両方、時間帯付き
time	その日の時刻のみ
time with time zone	その日の時刻のみ、時間帯付き
boolean	boolean型
oid	オブジェクト識別子

注意

Java EE 6ワークベンチでは、decimal型、numeric型のテーブルに対する操作はできません。

8.3.8 JDBC処理をDBアクセスクラスウィザードで自動生成する

標準のワークベンチではDBアクセスクラスウィザードを使うことにより、JDBCによるDBアクセス処理をDBアクセスクラスとして自動生成することができます。

DBアクセスクラスでは、ウィザードで指定された情報にもとづいて検索、挿入、削除、更新の処理がメソッドとして、処理対象となるデータベースのカラムはクラスのフィールドとして展開されます。そのため、DBアクセスクラスを使う場合には、クラスのフィールドに対して値の設定、取得を行いながらメソッドを呼び出すことでDBアクセス処理を行うことになります。

DBアクセスクラスの作成

DBアクセスクラスは、[新規]ウィザードから[Java] > [ソース] > [DBアクセスクラス]を選択し、ウィザードで作成します。ウィザードでの設定は、以下を参考にしてください。

- ソースフォルダ
  DBアクセスクラスのソースを格納するフォルダを指定します。

- パッケージ
  DBアクセスクラスのパッケージ名を指定します。

- 名前
  DBアクセスクラス名を指定します。

- マニュアルコミット用のメソッドを生成
  DBアクセスクラスにマニュアルコミット用のメソッドを生成するかどうかを指定します。チェックした場合、トランザクションを制御するためのcommitメソッドとrollbackメソッドを生成します。

- DBアクセスによる例外処理をキャッチ
  DBアクセスメソッド内で例外(SQLExceptionなど)をキャッチするかどうかを指定します。

- 接続リストボックス
  作成済みのDB接続一覧が表示され、この中から1つ選択します。

- 接続の追加
  新規DB接続ウィザードを開きます。

- 接続
  接続リストボックスで選択されたDBに接続します。

- [DB情報]ページ
  [追加]、[編集]、[削除]ボタンを使用してDBアクセスメソッド定義に関する情報を指定します。

- [DBアクセスメソッド]ダイアログボックス
  DBアクセスメソッドの名前、戻り値の型、引数などを指定します。
  [データベースオブジェクト]には接続しているデータベースのスキーマ、表および列名の一覧が表示されます。処理対象とする表、列を指定します。
  [アクセスタイプ]にはメソッドで行うデータベースアクセスタイプを選択します。
  [条件]にはデータベースにアクセスする際に使用する条件を指定します。条件には、SQL文のWHERE節以降の文字列を以下の例のように指定します。条件を動的に変更にするための"?"の代わりにDBアクセスメソッドのパラメタ名の前後を"?"で挟んで指定します。
  (例) WHERE NAME = ?param1? AND ID = ?param2?

- [フィールドの情報定義]ページ
  DBアクセスメソッドの定義内容を確認または変更します。
  [型]およびフィールド名は変更可能です。
  [取得メソッド]および[設定メソッド]は、[getter/setterメソッドを生成する]がチェックされた場合に有効となります。

参考

JDBCドライバ2.xを使用する場合は、自動生成されたクラスのconnectメソッドを、同じソース内に記述してある以下のコメント内容と入れ替える必要があります。

```
// JDBC2.x接続用のconnectメソッドです。
// 既存のconnectメソッドと入れ替え、必要項目を入力する事でデータソースによるアクセスが実行できます。
//public void connect(java.lang.String userName, java.lang.String passWord) throws javax.naming.NamingException,
java.sql.SQLException {
//    //データベースに接続します。
//    java.util.Hashtable env = new java.util.Hashtable();
//    env.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY, "{初期コンテキストファクトリ}");
//    env.put(javax.naming.Context.PROVIDER_URL, "{プロバイダURL}");
//    javax.naming.InitialContext ctx = new javax.naming.InitialContext(env);
//    javax.sql.DataSource ds = (javax.sql.DataSource)ctx.lookup("jdbc/{データソース名}");
//    con = ds.getConnection(userName, passWord);
```

```
//     con.setAutoCommit(true);
//}
```

また、コメント内容との入れ替えの際には、以下のデータベースに接続するための項目を記述し直します。

項目名	記述例
初期コンテキストファクトリ	com.fujitsu.symfoware.jdbc2.jndisp.SYMContextFactory (あらかじめプロジェクトのビルドパスにJDBCドライバを登録)
プロバイダURL	SYM://接続先アドレス:ポート番号
データソース名	JDBC/データソース登録ツールで登録した名前

第9章 Java EE 6アプリケーションを開発する

ここでは、Java EE 6ワークベンチを使用した、Java EE 6アプリケーション開発方法について説明します。

"マニュアルの読み方"に操作説明についての読み方を記載していますので、あわせてお読みください。

9.1 概要

Java EE 6アプリケーション開発の概要について説明します。

9.1.1 Java EE 6とは

Java EE 6とは、Java Platform, Enterprise Edition 6の略で、旧Sun(現オラクル)が提唱する業務システム向けのJavaプラットフォームに関する規約です。Javaの標準機能セットであるJava SEに業務システムの開発に使用するServletやEJBなどのサーバアプリケーション向けのセットが追加されています。アプリケーションサーバはこの規約に従って、コンポーネント化、各種サービス、通信方法などをJava EE 6プラットフォームとして実装し、再利用可能なコンポーネントの組み合わせで多階層アプリケーションを作成するテクノロジを提供しています。

Java EE 6の詳細情報については、以下の規約を参照してください。
JSR 316: Java(TM) Platform, Enterprise Edition 6 (Java EE 6) Specification

注意

- JAX-RPCやEJB 2.xのEntity Beanなど、古くて使用されなくなったAPIや代替機能が用意されているAPIに関しては規約からの削除が検討されています。これらのAPIを使用している場合には、Java EE 5またはJava EE 6 APIへの移行を御検討ください。
- Java EE 6ワークベンチのJava EE 6アプリケーション開発機能の一部は、アプリケーションの移植性を最大限に向上させるために、オープンソースソフトウェアであるGlassFish plugin v3.1.2をベースにしています。JAX-RSについてはJava EE 6標準の機能範囲をサポートします。GlassFish plugin固有実装による機能はサポートしません。

9.1.2 Java EE 6アプリケーション開発

Java EE 6アプリケーション開発の流れを以下に示します。

Java EE 6アプリケーション開発の準備

Java EE 6アプリケーションを開発するには、アプリケーション作成の準備を行った後、エンタープライズアプリケーションプロジェクトを作成します。プロジェクトを作成し、Java EE 6モジュールをプロジェクトに追加します。詳細は、"9.6.1 アプリケーション作成のための準備"および"6.2.2.2 エンタープライズアプリケーションを作成する"を参照してください。

Java EE 6アプリケーションの開発

Webアプリケーション、EJBなど、Java EE 6アプリケーションの開発を行います。
詳細は"9.2 Webアプリケーションを開発する"、"9.3 Enterprise JavaBeans(EJB)を開発する"、"9.4 Java Persistence APIを使用したアプリケーションを開発する"、"9.5 Webサービスアプリケーションを開発する"を参照してください。

Java EE 6アプリケーションの動作確認

作成したJava EE 6アプリケーションの動作確認はサーバビューを利用して行います。
詳細は、"6.2.7 アプリケーションの動作確認"を参照してください。

Java EE 6アプリケーションの配布

Java EE 6アプリケーションファイルを配布するためにはEARファイルにアーカイブする必要があります。アーカイブファイルの作成にはエクスポートウィザードを使用します。
詳細は、"6.2.8 運用環境への配布"を参照してください。

注意

Java EE 6アプリケーション運用環境には管理コンソール機能が提供されていません。設定や配備などはコマンドを使用します。

9.2 Webアプリケーションを開発する

ここでは、Java EE 6アプリケーション運用環境上で動作する、Webアプリケーションの概要および作成方法について説明します。

9.2.1 概要

9.2.1.1 Webアプリケーションとは

Webアプリケーションの概要については、"2.1.1 Webアプリケーションとは"を参照してください。

9.2.1.2 Java EE 5からの変更点

Java EE 6に含まれるWebアプリケーションの仕様はServlet 3.0です。

Servlet 3.0では主に以下の機能が追加されています。

- アノテーションを使ったサーブレットやフィルタ等の設定記述
  アノテーションを使って、クラスにサーブレット、サーブレットフィルタ等の設定を直接記入することができます。web.xmlを利用しない事も可能です。

- フレームワーク毎の設定記述
  Webフラグメント記述子を使って、フレームワーク毎に設定を記述し、jarファイルに含めることでプラガビリティを向上しています。

- ファイルアップロード
  マルチパートデータに対応し、簡単にファイルアップロードを実現できます。

- 非同期処理
  非同期プロセスを別スレッドで起動することができるようになり、プッシュ型アプリケーションを開発できます。

アノテーションを使ったサーブレットやフィルタ等の設定記述

Servlet 3.0で、アノテーションを使って、サーブレットやサーブレットフィルタ等の設定を記述する事が可能になりました。これにより、web.xmlの設定をアノテーションで定義した内容で上書きしたり、web.xmlを利用しない事も可能です。以下に主なサーブレット設定用のアノテーションを示します。

- @WebServlet
  サーブレットのクラス定義を行います。web.xmlの<servlet>要素や<servlet-mapping>要素と同じ働きをします。

- @WebFilter
  サーブレットフィルタの定義を行います。web.xmlの<filter>要素や<filter-mapping>要素と同じ働きをします。

- @WebListner
  リスナークラスの定義を行います。web.xmlの<listener>要素と同じ働きをします。

サーブレット、サーブレットフィルタ、リスナのインタフェースを実装したクラスにこれらのアノテーションを設定することで、サーブレット設定を定義できます。

ポイント

Servlet 3.0の詳細情報については、以下の規約を参照してください。

- JSR-315: Java(TM) Servlet 3.0 Specification

JSP 2.2の詳細情報については、以下の規約を参照してください。

- JSR-245: JavaServer(TM) Pages 2.2

JSF 2.1の詳細情報については、以下の規約を参照してください。

- JSR-314: JavaServer Faces 2.1

9.2.1.3 Webアプリケーションの開発

Webアプリケーションの開発の流れや開発手順については、"2.2.2 開発の流れ"および "2.2.3 開発手順"を参照してください。なお、アプリケーションを作成するための準備については、"9.6.1 アプリケーション作成のための準備"を参照してください。

注意

WebアプリケーションでJSFを使用する場合には、動的Webプロジェクトウィザードの[構成]に"JavaServer Faces v2.0 プロジェクト"を指定してください。

9.3 Enterprise JavaBeans(EJB)を開発する

ここでは、Java EE 6アプリケーション運用環境上で動作する、EJBの概要および作成方法について説明します。

9.3.1 概要

9.3.1.1 EJBとは

EJBの概要については、"3.1.1 EJBとは"を参照してください。

9.3.1.2 Java EE 5からの変更点

Java EE 6に含まれるEJB仕様はEJB 3.1です。EJB 3.1は開発の更なる簡易化に向けて以下のような仕様が改善されています。

- EJB-JARファイルの作成が不要
  従来、EJBを利用するためには、EJB-JARファイルを作成する必要がありましたが、EJB 3.1では作成する必要がなくなりました。
  アノテーションによりコンポーネント定義されているクラスは、WEB-INF/classesディレクトリ内に、あるいはWEB-INF/libディレクトリ内のjarファイルとしてパッケージされる事でEJBコンポーネントになります。

- ローカルビジネスインタフェースの省略
  従来、ローカルからEnterprise Beanを利用するためには、ローカルビジネスインタフェースを定義する必要がありましたが、EJB 3.1ではローカルビジネスインタフェースの定義を省略することができるようになりました。

使用例 (従来の定義)

ローカルビジネスインタフェースの定義：

```
public interface Calc {
    public int add(int param1, int param2);
}
```

EJBクラスの定義：

```
@Stateless
public class CalcBean implements Calc {
    public int add(int param1, int param2) {
        return param1 + param2;
    }
}
```

```
クライアントからの使用：
@EJB Calc a;

    a.add(1,1);
```

使用例 (EJB 3.1での定義)

```
EJBクラスの定義：
@Stateless
public class CalcBean {
    public int add(int param1, int param2) {
        return param1 + param2;
    }
}

クライアントからの使用：
@EJB CalcBean a;

    a.add(1,1);
```

ポイント

EJB 3.1の詳細情報については、以下の規約を参照してください。

- JSR 318: Enterprise JavaBeans(TM) 3.1

9.3.1.3 EJBの開発

EJBの開発の流れや開発手順については、"3.2.2 開発の流れ"および "3.2.3 開発手順"を参照してください。なお、アプリケーションを作成するための準備については、"9.6.1 アプリケーション作成のための準備"を参照してください。

注意

標準のワークベンチとJava EE 6ワークベンチでは、ウィザードの名前が異なる場合があります。
例:[Session Bean]ウィザードが、[Session Bean (EJB 3.x)]ウィザードに変更されています。

9.4 Java Persistence APIを使用したアプリケーションを開発する

ここでは、Java EE 6アプリケーション運用環境上で動作する、JPAを使用したアプリケーションの概要および作成方法について説明します。

9.4.1 概要

9.4.1.1 JPAとは

JPAの概要については、"4.1.1 JPAとは"を参照してください。

9.4.1.2 Java EE 5からの変更点

Java EE 6に含まれるJPAの仕様はJPA 2.0です。JPA 2.0では以下のような機能が追加されました。

- 埋め込み可能なオブジェクトのコレクションのマッピング機能
  アノテーションを使って、埋め込み可能オブジェクトのコレクションをマッピングできます。

- Java Persistence Query Language(JPQL)の拡張
  CASE式が利用可能になるなど、新しい演算子が利用できます。

- クライテリアAPI
  クエリの制御を、Javaオブジェクトで制御するAPIを利用できます。

ポイント

JPA 2.0の詳細情報については、以下の規約を参照してください。

- JSR 317: Java(TM) Persistence 2.0

9.4.1.3 JPAを使用したアプリケーションの開発

JPAを使用したアプリケーションの開発の流れや開発手順については、"4.2.2 開発の流れ"および "4.2.3 開発手順"を参照してください。なお、アプリケーションを作成するための準備については、"9.6.1 アプリケーション作成のための準備"を参照してください。

注意

- Java EE 6ワークベンチで接続プロファイルを作成する場合、[新規]ウィザードから[接続プロファイル] > [接続プロファイル]を選択してください。
- Java EE 6ワークベンチで"第4章 Java Persistence APIを使用したアプリケーションを開発する"の"開発手順"を行う場合、以下の差分があります。
  "2-1) JPAプロジェクトの作成"では[JPAプロジェクト]ウィザードの[プラットフォーム]では"Generic 2.0"を、JPA実装には"ライブラリ構成を無効にする"を指定します。
  "6-2) ビルドパスの設定"ではEARプロジェクトを選択して、コンテキストメニューから[プロパティ]を選択して、[ビルドパス]で確認します。
- JDBC接続プールの作成にはcreate-jdbc-connection-poolコマンドを、JDBCリソースの作成にはcreate-jdbc-resourceコマンドを使用します。JDBCドライバのクラスパスの設定など、Interstage Application ServerのJDBC設定の詳細については、"Interstage Application Server Java EE運用ガイド(Java EE 6編)"の"データベースの環境設定"を参照してください。

9.5 Webサービスアプリケーションを開発する

ここでは、Java EE 6アプリケーション運用環境上で動作する、Webサービスアプリケーションの概要および作成方法について説明します。

9.5.1 概要

9.5.1.1 Webサービスとは

Webサービスの概要については、"5.1.1 Webサービスとは"を参照してください。

9.5.1.2 Java EE 5からの変更点

Java EE 6には新たにJAX-RS1.1が含まれます。

JAX-RS1.1を利用することで、Representational State Transfer(REST) 型のWebアプリケーションを簡単に開発できます。

JAX-RS1.1ではアノテーションを利用することで、Javaリソースのパスを指定したり、JavaメソッドをHTTPリクエストメソッドにバインドすることが簡単にできます。

アノテーション

アノテーションを利用し、REST型のWebアプリケーションとして動作するクラスを定義することができます。

代表的なアノテーションを以下に挙げます。

- @Path
  ルートリソースやサブリソースのJavaメソッドのパスを記述するために使われます。

- @GET
  HTTPリクエストメソッド"GET"を指定するアノテーションです。ルートリソースまたはサブリソースのJavaメソッドをHTTPリクエストメソッド"GET"にバインドできます。

- @POST
  HTTPリクエストメソッド"POST"を指定するアノテーションです。ルートリソースまたはサブリソースのJavaメソッドをHTTPリクエストメソッド"POST"にバインドできます。

- @Produces
  ルートリソースやサブリソースのJavaメソッドが生成し、クライアントに返却することができるMIMEメディアタイプを示します。

- @Consumes
  ルートリソースやサブリソースのJavaメソッドが、クライアントから受け入れることができるMIMEメディアタイプを示します。

アノテーションを利用したREST型Webアプリケーションの作成例

```
@Path(value=""/root"")
public class RootResource {
    @GET
    public String getName() {
        return ""name"";
    }
}
```

ポイント

JAX-RS 1.1の詳細情報については、以下の規約を参照してください。

- JSR-311: JAX-RS: The Java(TM) API for RESTful Web Services

JAX-WS 2.2の詳細情報については、以下の規約を参照してください。

- JSR 224: Java(TM) API for XML-Based Web Services (JAX-WS) 2.2

9.5.1.3 Webサービスアプリケーションの開発

Webサービスアプリケーションの開発の流れや開発手順については、"5.2.2 開発の流れ"および"5.2.3 開発手順"を参照してください。
なお、アプリケーションを作成するための準備については、"9.6.1 アプリケーション作成のための準備"を参照してください。

注意

Java EE 6ワークベンチでは、ijwsimportコマンドを使用してWSDLからサービスエンドポイントインタフェースを生成します。

1. コマンドプロンプトを開き、作業フォルダを作成します。
   例：mkdir temp
2. 作業フォルダに移動します。
   例：cd temp
3. ソース出力先フォルダを作成します。
   例：mkdir src
4. ijwsimportコマンドを使用して、作業フォルダにWebサービスクライアントに必要なソースを生成します。
   ijwsimport -p <パッケージ名> -s <ソース出力先フォルダ> -keep <WSDLのURL>
   例： ijwsimport -p stub -s src -keep http://localhost:28282/WebServiceSample6/PopulationRankingService?wsdl

5. Webサービスクライアントプロジェクトのソースフォルダ配下に、4.の<ソース出力先フォルダ>のソースファイルをパッケージフォルダごとコピーします。WSDLファイルは不要です。
6. Webサービスクライアントプロジェクトを選択して、[F5]キーを押します。
7. 作業フォルダを削除します。

上記5.および6.で追加したソースファイルでエラーが検出される場合には、引き続いて以下の手順でビルドパスの設定を行ってください。

1. Webサービスクライアントプロジェクトを選択して、右クリックでコンテキストメニューから[プロパティ]を選択します。
2. [Javaのビルドパス] > [ライブラリ]タブ > [外部JARの追加]で以下に存在するJARファイルを追加します。
   <製品インストールフォルダ>¥APS¥F3FMisje6¥glassfish¥modules¥endorsed
   <製品インストールフォルダ>¥APS¥F3FMisje6¥glassfish¥lib¥endorsed
3. 続けて[順序およびエクスポート]タブで、2.で追加したJARファイルを[JREシステムライブラリ]よりも上に移動して[OK]ボタンを押してください。

ijwsimportコマンドやWSDLのURLについては、"Interstage Application Server Java EE運用ガイド(Java EE 6編)"の"Webサービスアプリケーションの開発"の"配備とWSDLの取得・保管"を参照してください。

ポイント

JAX-RSアプリケーションを作成する場合には、"Interstage Application Server Java EE運用ガイド(Java EE 6編)"の"JAX-RSアプリケーションの作成方法"を参照してください。

9.6 Java EE 6アプリケーション共通事項

ここでは、Interstage Studioを用いてJava EE 6アプリケーションを開発する際の共通事項について、開発作業課題(タスク)ごとに説明します。

9.6.1 アプリケーション作成のための準備

アプリケーションを作成するためには、開発するモジュールに応じてプロジェクトを作成する必要があります。
また、複数のモジュールで構成されるようなアプリケーションについては、エンタープライズアプリケーションとしてまとめることができます。

9.6.1.1 サーバを操作するための準備をする

Java EE 6ワークベンチでサーバの起動や停止などの操作を行うためには、サーバビューに操作対象となるサーバを追加する必要があります。サーバビューに追加する際に指定するランタイムは、Java EE 6アプリケーションを開発する際のウィザードでターゲットランタイムとして指定しますので、Java EE 6アプリケーション開発前にサーバを追加する事をお勧めします。

サーバを追加する

動作確認を行うアプリケーションの配備先となるサーバをサーバビューに追加します。サーバの追加には新規サーバウィザードを使用します。サーバビューを右クリックし、コンテキストメニューから[新規] > [サーバ] を選択してください。ウィザードの設定項目については以下の内容を参考にしてください。

- サーバのタイプ
  サーバのタイプを選択します。InterstageのJava EE 6運用環境の場合は[FUJITSU LIMITED] > [Interstage Application Server V11.1(Java EE 6)]を選択してください。

- サーバのホスト名
  サーバがあるホストの名前を指定します。ローカルマシンの場合は"localhost"を指定します。

- サーバ名
  サーバ名を指定します。サーバビューでは、ここで指定した名前が表示されます。

- サーバランタイム環境
  サーバのタイプに対応するランタイム設定を指定します。InterstageのJava EE 6運用環境の場合は[Interstage Application Server V11.1 IJServer Cluster(Java EE)]を選択してください。

Interstage Application Serverの場合は、続けて以下の設定項目が表示されます。

- 再配備間のセッションを維持する
  再配備時に、セッションを継続して保持する場合に指定します。

- 運用管理用HTTPリスナーポート
  運用管理用HTTPリスナーポートを指定します。デフォルトのポート番号は12011です。サーバの設定に合わせる必要があります。

- HTTPリスナーポート
  デフォルトのポート番号は28282です。サーバの設定に合わせる必要があります。リモートサーバに接続する場合には、サーバのHTTPSリスナーポートを指定してください。

- デバッグポート番号
  デフォルト値が表示されるので、必要に応じて変更してください。

- HTTPポート番号
  接続確認に使用するHTTPサーバのポート番号を指定します。デフォルト値が表示されるので、サーバの設定に合わせて変更してください。

  ー Webサーバコネクタ(Interstage HTTP Server 2.2用)を利用する場合

    Webサーバ(Interstage HTTP Server 2.2)のポート番号を指定してください。

  ー Webサーバコネクタ(Interstage HTTP Server 2.2用)を利用しない場合

    - 配備先にInterstage Java EE 6 DASサービスを指定する場合、Java EE 6運用環境のHTTPリスナーポートまたはHTTPSリスナーポート番号と同じ値を指定してください。

    - 配備先にIJServerクラスタを指定する場合、IJServerクラスタのHTTPリスナーポートまたはHTTPSリスナーポート番号と同じ値を指定してください。

ポート番号を指定したら[ログイン]をクリックして、サーバにログインできるか確認してください。管理者名、管理者パスワードには、Interstage Java EE 6運用環境の管理者ユーザ、管理者パスワードを指定します。ログインできると[次へ]が有効になります。

ポイント

- サーバの追加は、新規ウィザードからも行うことができます。
- 管理者名、管理者パスワードなど、Interstage Application ServerのJava EE 6運用環境に関する詳細は、"Interstage Application Server Java EE運用ガイド(Java EE 6編)"を参照してください。
- 追加したサーバの情報は、サーバページ([サーバ]ビューで対象のサーバをダブルクリック)で変更できます。

注意

- サーバのホスト名に"localhost"以外を指定した場合は、リモートサーバと見なします。リモートサーバへの接続はhttps通信になります。
- ウィザードの入力項目はホストに指定した内容によって異なります。
- サーバビューでのサーバ操作とコマンドによるサーバ操作を併用すると、配備などの際にエラーになる場合があります。

9.6.1.2 プロジェクトを新規に作成する

プロジェクトウィザードは、メニューから[ファイル] > [新規] > [プロジェクト]を選択して、[新規プロジェクト]ウィザードを起動し、以下のようにモジュールに応じてプロジェクトウィザードを選択します。

カテゴリ	プロジェクト	説明
EJB	EJBプロジェクト	EJBのモジュールを作成する場合に使用します。
Java EE	アプリケーションクライアントプロジェクト	アプリケーションクライアントのモジュールを作成する場合に使用します。
	エンタープライズアプリケーションプロジェクト	EARファイルを作成する場合に使用します。EARファイルは、EJB、Webアプリケーションおよびライブラリをまとめることができます。
	ユーティリティプロジェクト	複数のモジュールで共用するライブラリを作成する場合に使用します。
JPA	JPAプロジェクト	JPAを使用したライブラリを作成する場合に使用します。
Web	動的Webプロジェクト	Webアプリケーションのモジュールを作成する場合に使用します。

プロジェクトウィザードでの指定内容については、以下を参考にしてください。

- プロジェクト名
  生成するプロジェクト名を指定します。

- プロジェクトコンテンツ
  プロジェクト資産の格納先を指定します。デフォルトでは、ワークスペースフォルダ配下になります。

- ターゲットランタイム
  Java EE 6のアプリケーションを動作させるランタイムを選択します。これによりランタイムのライブラリがクラスパスに設定されます。
  Interstage Application ServerのJava EE 6運用環境の場合は、[Interstage Application Server V11.1(Java EE 6)]を選択します。

- モジュールバージョン
  EJBやWebアプリケーションなどの場合に、モジュールのバージョンを指定します。

- 構成
  作成するモジュールやライブラリが、準拠する規約とそのバージョンを設定します。ウィザードやエディタなどの支援機能が、規約とバージョンに準拠するように動作します。Interstage Application ServerのJava EE 6コンテナの場合は、[Interstage Application

Server V11.1(Java EE 6) デフォルト構成]を選択します。

- EARメンバシップ
  モジュールやライブラリをEARファイルにまとめる場合は、エンタープライズアプリケーションのプロジェクトを選択します。エンタープライズアプリケーションをあとで作成する場合は、ここで選択する必要はありません。

- ワーキングセット
  プロジェクトをワーキングセットに追加する場合は、ワーキングセットを選択します。

注意

- [EARメンバシップ]の[EARにプロジェクトを追加]をチェックしないでEJBプロジェクトを作成すると、Session Beanウィザードのビジネスインタフェースは EJBプロジェクトに作成されます。
- [EARメンバシップ]の[EARにプロジェクトを追加]をチェックしてEJBプロジェクトを作成すると、Session Beanウィザードのビジネスインタフェースは EJBクライアントプロジェクトに作成されます。
- [EARメンバシップ]の[EARにプロジェクトを追加]をチェック、[クライアントのインタフェースとクラスを保持するためのEJBクライアントJARモジュールを作成]をチェックしないでEJBプロジェクトを作成すると、Session Beanウィザードのビジネスインタフェースは EJBプロジェクトに作成されます。

9.6.1.3 エンタープライズアプリケーションを作成する

エンタープライズアプリケーションプロジェクトでは、モジュールやライブラリをまとめてEARファイルを作成することができます。

エンタープライズアプリケーションプロジェクトウィザード固有の指定内容については、以下を参考にしてください。

- コンテンツフォルダ
  プロジェクトとして作成しているJava EE モジュール以外で、EARファイルに含めたいファイルを格納するためのフォルダの名前を指定します。

エンタープライズアプリケーションプロジェクト作成後、EARファイルに追加するJava EE モジュールを変更したい場合には、プロジェクトの[ビルドパス]プロパティで編集します。

ポイント

- EARファイルにすると、アプリケーションをまとめて配備できるだけでなく、EARファイル内のJava EEモジュールの依存関係を定義でき、運用環境でのクラスパスの設定などの作業を軽減できます。
- エンタープライズアプリケーションにアプリケーションクライアントを含めるとエンタープライズアプリケーションに含まれるモジュールやライブラリをクライアントスタブとしてダウンロードすることができます。詳細は"Interstage Application Server Java EE運用ガイド(Java EE 6編)"の"Java EEアプリケーションクライアントの運用操作"の"クライアントスタブJARファイルをダウンロード"を参照してください。

9.6.1.4 クラスパスを設定する

Javaアプリケーションをビルドするためには、プロジェクトにビルドパス(クラスパス)を設定します。Java EE 6のモジュールやライブラリは、以下のどれかの方法でビルドパスを設定します。

ターゲットランタイム

Java EE 6のアプリケーションを動作させるランタイムを選択すると、そのライブラリがビルドパスに追加されます。これはプロジェクトの[ターゲットランタイム]プロパティで設定することができます。
ビルドパスにランタイムの名前でライブラリが追加されます。

エンタープライズアプリケーションのビルドパス

パッケージ(EAR/WARファイルなど)にまとめたモジュールやライブラリは、パッケージ内のクラスを参照できます。この指定はEARプロジェクトの[ビルドパス] プロパティで指定します。また、動的WebプロジェクトやEJBプロジェクトなどの[ビルドパス] プロパティの[デプロイメントアセンブリ]タブでも同様の設定ができます。

Javaのビルドパス

プロジェクトの[Javaのビルドパス]プロパティでビルドパスを設定できます。この設定で追加したライブラリは、運用環境には反映されません。このため、アプリケーションサーバに配備する前に、運用環境のクラスパスの設定をする必要があります。

9.6.1.5 開発資産の無いJava EE 6モジュールを利用した開発をする

Java EE 6アプリケーションを開発する場合、他者が開発したJava EE 6モジュールを利用して開発する場合があります。
エンタープライズアプリケーションプロジェクトでEARファイルを作成する場合、Java EE 6モジュールをインポートし、Java EE 6モジュールのプロジェクトを作成する必要があります。
ワークベンチのメニューから[ファイル] > [インポート]を選択し、[インポート]ウィザードからインポートするファイル形式を選択します。

ポイント

インポートするJava EE 6モジュールにソースファイルが含まれている場合は、インポートウィザードで作成したプロジェクトのソースフォルダにソースファイルが格納されます。インポートするJava EE 6モジュールにソースファイルが含まれていない場合は、インポートウィザードで作成したプロジェクトにImportedClassesフォルダが作成され、ImportedClassesフォルダにクラスファイルが格納されます。

9.6.2 Javaクラスおよびインタフェースを作成する

"6.2.3 Javaクラスおよびインタフェースを作成する"を参照してください。

9.6.3 XMLファイルを作成する

"6.2.4 XMLファイルを作成する"を参照してください。

9.6.4 プロパティファイルを作成する

"6.2.5 プロパティファイルを作成する"を参照してください。

9.6.5 問題の検出と修正

"6.2.6 問題の検出と修正"を参照してください。

注意

標準のワークベンチとJava EE 6ワークベンチでは、提供されている検証機能に差があります。

9.6.6 アプリケーションの動作確認

Java EE 6アプリケーションを動作確認するには、Java EE 6アプリケーションをサーバに配備し、サーバを起動します。これらの操作はサーバビューで行います。サーバを起動すると、Java EE 6アプリケーションは、クライアントからの呼び出しを待ちます。クライアントからJava EE 6アプリケーションを呼び出すことで動作確認を行います。
また、プログラムの実行を中断してコードを1行ずつ実行したり、変数の値を確認したりするには、デバッグを行います。

注意

- 以下の環境でアプリケーションの配備操作を行うと、ワークスペースフォルダ配下に配備用のアーカイブファイル(EAR/WARなど)を作成します。空きディスク容量をご用意ください。

  ー リモートサーバに配備する場合。

ー サーバページで[JARアーカイブを作成してから配備する]がチェックされているlocalhostに配備する場合。

- アプリケーションの動作確認に、Webサーバ(Interstage HTTP Server 2.2)とWebサーバコネクタ(Interstage HTTP Server 2.2用)を利用する場合、"Interstage Application Server Java EE運用ガイド(Java EE 6編)"の"Webサーバ連携時の制限事項"を参照してください。
  Webサーバ連携の設定時にasadminコマンドを使用できない場合には、以下の操作が必要です。

  ー アプリケーションを配備した後、wscadminコマンドのadd-application-refサブコマンドで振り分け先のアプリケーションを追加する必要があります。

  ー アプリケーションを配備解除した後、wscadminコマンドのdelete-application-refサブコマンドで振り分け先のアプリケーションを削除する必要があります。

サーバに配備するプロジェクトを指定する

サーバに配備するプロジェクトの指定は、プロジェクトをサーバビューのサーバに追加することによって行います。サーバに配備するプロジェクトを追加するには、プロジェクトの追加および除去ウィザードを使用します。プロジェクトの追加および除去ウィザードを起動するには、サーバビューでサーバを選択し、コンテキストメニューから[プロジェクトの追加および除去]を選択してください。
新規サーバウィザードの最後のページでプロジェクトを追加することもできます。

ポイント

手動でアプリケーションを配備するには、サーバを選択し、コンテキストメニューより[公開]を選択します。[クリーン]を用いると、配備されているアプリケーションを一度すべて破棄し、配備しなおします。アプリケーションがサーバと同期している場合は、状況に「同期済み」と表示されます。同期していない場合は、「再公開」と表示されます。

サーバを起動する

サーバビューでサーバを選択し、コンテキストメニューから[開始]または[デバッグ]を選択します。サーバを起動するときにアプリケーションを自動的に配備することができます。デフォルトでは、サーバ起動時に自動的に配備する設定になっています。

ポイント

- プロジェクトを選択し、コンテキストメニューから[実行] > [サーバで実行]または[デバッグ] > [サーバでデバッグ]を選択することで、サーバの追加、プロジェクトの追加、サーバの起動、クライアントの起動をまとめて行うことができます。
- サーバ起動時に自動的に配備しないようにするには、設定ページの[サーバ] > [起動]を選択し、[サーバの始動時に自動的に公開]をオフにします。

サーバを停止する

サーバを停止するには、サーバビューでサーバを選択して、コンテキストメニューから[停止]を選択します。

9.6.6.1 デバッグする

ここでは、プログラムにおける論理エラーの検出に用いられるデバッグについて説明します。
アプリケーションをデバッグするには、ブレークポイントを設定して、起動されたプログラムを中断し、コードを1行ずつ実行し、変数の内容を確認することによって行います。

ブレークポイント

ブレークポイントを設定した行でプログラムの実行が中断します。実行が中断されると、デバッグパースペクティブを開くかどうかの確認ダイアログボックスが表示されます。デバッグパースペクティブは、デバッグする際に利用するデバッグビュー、変数ビュー、ブレークポイントビューなどがレイアウトされています。

- ブレークポイントを設定・解除する
  ブレークポイントの設定と解除は、エディタのルーラー部分をダブルクリックして切り替えます。また、追加されているブレークポイントの確認や、ブレークポイントの削除をブレークポイントビューで行うこともできます。

- ブレークポイントを無効にする
  ブレークポイントを一時的に無効にするには、ルーラーのコンテキストメニューで[ブレークポイントを使用不可にする]を選択するか、ブレークポイントビューでチェックを外します。また、すべてのブレークポイントを一時的にスキップするようにするには、ブレークポイントビューのツールバー、または、メニューバーの[実行]で、[すべてのブレークポイントをスキップ]を選択します。

JSPファイルにブレークポイントを設定した場合、異なるフォルダにある同名のJSPファイルでも実行が中断します。

実行制御

実行を中断したプログラムを再度実行する方法には、ステップオーバ、ステップイン、ステップリターン、再開などがあります。これらのコマンドは、デバッグビューでスタックフレームを選択し、デバッグビューのツールバー、コンテキストメニュー、または、メニューバーの[実行]から行います。

- ステップオーバ
  現在選択されている行が実行され、次の実行可能行で中断されます。

- ステップイン
  現在選択されている行の、次に実行する必要のある式が呼び出され、呼び出されたメソッドの次の実行可能行で実行が中断されます。

- ステップリターン
  現在のメソッドの次の return ステートメントが実行されるまで再開され、次の実行可能行で中断されます。

- 再開
  ブレークポイントに到達するまで実行を継続します。

変数の値の確認と変更

スタックフレームを選択すると、そのスタックフレームで可視になっている変数を変数ビューに表示できます。
変数ビューには、プリミティブ型の値が表示されます。複雑な変数は、それを展開してそのメンバを表示すれば検査できます。値の変更は、コンテキストメニューの[値の変更]から行います。

変数の値を参照するなどのデバッグを行うには、コンパイルされたクラスファイルにデバッグ情報を追加する必要があります。デフォルトでは、デバッグに必要なデバッグ情報はクラスファイルに追加する設定になっています。運用環境に配布する場合などで、デバッグを行う必要が無い場合はデバッグ情報を追加せずに、クラスファイルのサイズを小さくすることができます。

例外がスローされたときなどに出力するスタックトレースでは、ソースファイル名の表示と行番号の表示に、デバッグ情報を使います。行番号属性とソースファイル名はクラスファイルに追加することをお勧めします。

9.6.7 運用環境への配布

Java EE 6アプリケーションを運用環境に配布するには、アーカイブファイルを作成する必要があります。作成したアーカイブファイルは、配備機能を利用して、サーバに配備します。

アーカイブファイルの作成

アーカイブファイルを作成するにはエクスポートウィザードを利用します。

エクスポート

1. エクスポートウィザードの起動
   [ファイル] > [エクスポート]を選択します。

2. 設定項目の入力

- エクスポート先

プロジェクト	エクスポート先
EJBプロジェクト	[EJB] > [EJB JARファイル]
エンタープライズアプリケーションプロジェクト	[Java EE] > [EARファイル]
アプリケーションクライアントプロジェクト	エンタープライズアプリケーションプロジェクトで作成するアプリケーションに含まれるため、EARとしてエクスポートします。単体でエクスポートする場合には以下を使用します。 [Java EE] > [アプリケーションクライアントJARファイル]
動的Webプロジェクト	[Web] > [WARファイル]
JPAプロジェクト	エンタープライズアプリケーションプロジェクトまたは動的Webプロジェクトで作成するアプリケーションに含まれるため、EARまたはWARとしてエクスポートします。単体でエクスポートする場合には以下を使用します。 [Java] > [JARファイル]

- モジュール
  エクスポートするプロジェクトを指定します。

- 宛先
  アーカイブファイルの作成先を指定します。

ポイント

- エンタープライズアプリケーションプロジェクトをエクスポートすると、それに含まれている各プロジェクトのアーカイブファイルを含んだEARファイルが作成されるため、個々にアーカイブファイルを作成する必要はありません。
- ビルドパスに含めているJARファイルのうち、EARに入れないJARファイルは、サーバのクラスパスに設定します。

サーバへの配備

サーバへの配備は、サーバの配備機能を利用します。
配備の詳細については、"Interstage Application Server Java EE運用ガイド(Java EE 6編)"の"アプリケーションの配備"を参照してください。

第10章　Tips

ここでは、知っておくと便利な内容をツール利用編、プログラミングテクニック編、Eclipseプラグイン利用編、およびシンクライアント環境編に分けて説明します。

10.1 ツール利用編

ここでは、ワークベンチの便利な機能について、以下のカテゴリに分類して紹介します。

- 設定
  環境設定、ワークベンチのカスタマイズなどについて紹介します。

- エディタ
  エディタを利用するうえで知っていると便利な機能を紹介します。

- コーディング支援
  コーディングするうえで知っていると便利な機能を紹介します。

- 検索
  検索に関連する便利な機能を紹介します。

- ビルドおよびデバッグ
  ビルドおよびデバッグに関連する便利な機能を紹介します。

- ヘルプ
  ヘルプに関連する便利な機能を紹介します。

- その他
  ワークベンチの基本的な操作、javadoc、リファクタリング、JUnitなどに関連する便利な機能を紹介します。

10.1.1 設定

プロキシの使用

プロキシを使用する環境でワークベンチを使用する場合には、設定ページの[一般] > [ネットワーク接続]で設定を行う必要があります。

キーバインディング

場面に応じて特定のキーストロークやキーシーケンスを特定のコマンドに割り当てることができ、その割り当て方法を、設定ページの[一般] > [キー]でカスタマイズすることができます。
以下を参考にカスタマイズしてください。

- スキーム
  キーバインディングのセットです。デフォルトとEmacsがあります。

- バインディング
  実際のキーストロークやキーシーケンスをキーボードからタイプすることで入力されます。

- 場合
  キーバインディングを設定する場面を選択してください。

色とフォント

ワークベンチ内で使用されるフォントおよび画面の色については、設定ページの[一般] > [外観] > [色とフォント]で設定を行うことができます。
ツリービューより変更したいアイテムを選択することにより、フォントや色に応じた変更ボタンが表示されますので、そこから変更してください。

文字コード

ファイルの文字コードは、ファイル内に文字コードの記述があるもの(HTML、JSP、XMLなど)は、その文字コードで判断されますが、Javaソースなど文字コードを含まないものについては、以下の設定の内容で決まります。

- リソースのプロパティ
  選択しているリソースのプロパティの[リソース]の[テキストファイルエンコード]で、そのファイル、または、そのフォルダ配下のファイルの文字コードを指定することができます。

- コンテンツタイプのデフォルトエンコード
  設定ページの[一般] > [コンテンツタイプ]で、コンテンツタイプ単位にデフォルトエンコードを指定できます。ワークスペースのデフォルトは、[一般] > [ワークスペース]で指定します。

HTML、JSPなどは、以下の設定ページで、新規ウィザードで埋め込む文字コードの初期値を指定できます。

ファイルタイプ	設定ページ
CSS	[Web] > [CSSファイル]
HTML	[Web] > [HTMLファイル]
JSP	[Web] > [JSPファイル]
XMLファイル	[XML] > [XMLファイル]

デフォルトでは見えない機能を有効にする

Interstage Studioでは、一般的なユーザが使用しない機能を見えないようにしています。例えば、独自のEclipseプラグインなどを開発するためのプラグイン開発機能などがそれにあたります。
それらの機能を有効にするには、設定ページの[一般] > [機能]から以下のように行います。

1. [拡張]をクリックして、[高度な機能設定]ダイアログボックスを表示します。
2. 有効にしたい機能を選択し、[OK]をクリックします。

10.1.2 エディタ

デフォルトエディタの変更

エディタでファイルを開く際に、コンテキストメニューから[アプリケーションから開く]を選択して使用するエディタを選択できます。
設定ページの[一般] > [エディタ] > [ファイルの関連付け]から、選択可能なエディタを追加したり、そのファイルタイプのデフォルトエディタを指定したりすることができます。

エディタ領域の最大化

通常のパースペクティブではエディタ領域はワークベンチの中央に表示されています。コーディング中などエディタ領域を広くしたい場合は、エディタのファイル名が表示されているタブをダブルクリックすることで、最大化したり元に戻したりすることができます。

コンテンツアシスト

[Ctrl+Space]キーを押すと、その場面に応じて入力候補が表示されます。入力候補を絞り込むために、最初の数文字を入力し、その時点での[Ctrl+Space]キーで候補を表示することもできます。また、候補が表示されている状態で最初の数文字を入力して表示する候補を絞り込むこともできます。

対応する括弧の検索

Javaエディタでは、左大括弧または右大括弧を選択して、メニューから[ナビゲート] > [ジャンプ] > [対応する大括弧]を選択することで対応する括弧を検索することができます。
また、左大括弧の後か右大括弧の前でダブルクリックして、これら 2つの括弧で囲まれているテキストを選択することもできます。

クイックDiff

テキストエディタではクイックDiff機能により、最後に保存した状態や資産管理されているファイルとの差をエディタの左端に色分けして表示します。
マウスカーソルを変更箇所に持っていくことにより、変更内容が吹き出し確認できます。

スペルチェック

テキストエディタのスペルチェック機能をカスタマイズする場合には、設定ページの[一般] > [エディタ] > [テキストエディタ] > [スペル]より行います。

フォーマット

以下のファイルタイプではフォーマット機能が提供されており、設定ページでフォーマット形式をカスタマイズすることができます。

ファイルタイプ	設定ページ
Antスクリプト	[Ant] > [エディタ] > [フォーマッタ]
Java	[Java] > [コードスタイル] > [フォーマッタ]
CSS	[Web] > [CSS ファイル] > [エディタ]
HTML	[Web] > [HTML ファイル] > [エディタ]
JSP	[Web] > [JSP ファイル] > [エディタ]
XMLファイル	[XML] > [XML ファイル] > [エディタ]

フォーマットは、エディタで開いたり、ビューでファイルを選択したりした状態で、メニューより[ソース] > [フォーマット]を選択して実行します。

クリーンアップ

以下のエディタでは、フォーマットより強力なクリーンアップ機能が提供されており、エディタでファイルを開いた状態で以下のメニューから選択することで使用することができます。

エディタ	メニュー	代表的な付加機能
Javaエディタ	[ソース] > [クリーンアップ]	staticメンバアクセス方法変更 未使用importの削除 @Overrideや@Deprecatedの追加
HTML/JSPエディタ	[ソース] > [文書のクリーンアップ]	タグ/属性の大文字小文字の統一 必須属性や欠落タグの補完 行区切り文字の変換
CSSエディタ	[ソース] > [文書のクリーンアップ]	大文字小文字の統一
XMLエディタ	[ソース] > [文書のクリーンアップ]	空要素タグの圧縮 必須属性、欠落タグの挿入 行区切り文字の変換

構文の色の指定

以下のファイルタイプのエディタでは、構文の色を設定ページでカスタマイズすることができます。

ファイルタイプ	設定ページ
Antスクリプト	[Ant] > [エディタ]の[構文]タブ

ファイルタイプ	設定ページ
Java	[Java] > [エディタ] > [構文の色の指定]
CSS	[Web] > [CSS ファイル] > [エディタ] > [構文の色の指定]
DTD	[XML] > [DTD ファイル] > [構文の色の指定]
HTML	[Web] > [HTML ファイル] > [エディタ] > [構文の色の指定]
JavaScript	[JavaScript] > [エディタ] > [構文の色の指定]
JSP	[Web] > [JSP ファイル] > [エディタ] > [構文の色の指定]
XMLファイル	[XML] > [XML ファイル] > [エディタ] > [構文の色の指定]

保存時に自動的に実行する動作の設定

Javaエディタには、設定ページの[Java] > [エディタ] > [保存時アクション]より、保存時に自動的に実行する動作を指定できます。
フォーマットや、インポートの編成などを行うことができます。

エディタとビューの同期

Javaエディタで編集しているファイルのパッケージエクスプローラなどでの位置を確認したい場合には、コンテキストメニューより[表示] > [<位置を確認したいビュー>]を選択することで、ビュー上で、編集しているファイルをアクティブにすることができます。

また、ビューによっては右上のメニューから[エディタにリンク]を設定することで、編集しているファイルとビューの選択を同期させることができます。

エディタで開いているファイルの切り替え

エディタで多くのファイルを開いている場合には、すべてのファイル名がエディタ領域のタブには表示されません。エディタタブ表示領域の右端の数字は表示されていないファイルの数を表しており、ここをクリックすることで一覧が表示されます。さらに、そこからファイルを選択することで、編集するファイルを切り替えることができます。

参照していた箇所に戻る機能

エディタで複数のファイルを参照したり編集したりしている際に、直前の参照していた箇所に戻りたい場合があります。
そのような場合には、メニューから[ナビゲート] > [戻る]、[進む]、[最後の編集位置]などを利用すると便利です。これらの機能はツールバーからも利用することができます。

10.1.3 コーディング支援

try/catchブロックの追加

Javaで例外を処理しなければいけない場合に、以下のように簡単にtry/catchブロックを追加することができます。

1. Javaエディタ上で例外を処理したい範囲を選択します。
2. メニューより、[ソース] > [囲む] > [try/catchブロック]を選択します。

Getter/Setterの生成

エディタでJavaクラスを編集している際に、コンテキストメニューから[ソース] > [GetterおよびSetterの生成]を選択することで、フィールドを指定してそのGetterおよびSetterを生成することができます。

import文の追加

Javaでパッケージ名を指定せずにクラス名だけを使ってコーディングすると、import文を追加するまでエディタ上では解決できない型としてエラーが表示されます。
その状態で、コンテキストメニューから[ソース] > [インポートの編成]を選択すると、自動的に必要なimport文を追加することができます。

テンプレート

何度も利用するコードの断片をテンプレートとして保存しておき、コンテンツアシストを利用して編集中のファイルに挿入したり、新規ウィザードでファイルを作成するときに利用したりすることができます。

テンプレートの編集は、以下の設定ページで行います。

ファイルタイプ	設定ページ
Antスクリプト	[Ant] > [エディタ] > [テンプレート]
Java	[Java] > [エディタ] > [テンプレート]
SQLファイル	[SQL 開発] > [SQL エディタ] > [テンプレート]
CSS	[Web] > [CSS ファイル] > [エディタ] > [CSSテンプレート]
DTD	[XML] > [DTD ファイル] > [DTDテンプレート]
HTML	[Web] > [HTML ファイル] > [エディタ] > [HTMLテンプレート]
JSP	[Web] > [JSP ファイル] > [エディタ] > [JSPテンプレート]
XMLファイル	[XML] > [XML ファイル] > [エディタ] > [XMLテンプレート]

テンプレートでは、ワークベンチで前もって用意されている変数(ユーザ、日付、選択リソースなど)を利用することができます。

スニペット

以下のように、コーディングで利用する断片をスニペットに登録しておくと、マウスを利用してドラッグ＆ドロップすることで簡単に挿入することができます。

1. スニペットビューを表示します。
2. コンテキストメニューより[カスタマイズ]を選択します。
3. [パレットのカスタマイズ]ダイアログボックスで、名前、変数、テンプレートパターンを記述します。

変数が設定されているテンプレートを挿入した場合には、変数の値を挿入時に指定することができます。
なお、あらかじめ定義されているカテゴリJSPに対して、テンプレートを追加、削除、変更することはできません。

ファイルコメント、タイプコメント

設定ページの[Java] > [コードスタイル] > [コードテンプレート]で、ファイルコメントやタイプコメントを設定することができます。これにより、Javaクラスを作成する際にコメントを統一することができます。

変数名の提案

フィールドやローカル変数を宣言する際に、型名を入力して空白を入力したあとで、[Ctrl+Space]キーを入力するとフィールド名や変数名の候補が表示されます。

変数、returnの強調表示

フィールドやローカル変数名を選択すると、そのフィールドやローカル変数を使用しているところが強調表示されます。
また、メソッドで復帰値の型を選択した場合には、return位置が強調表示されます。

TODOコメント

後からしなければいけないことなどをソース上にTODOコメントで記述しておくと、ビルド時にその情報が収集されてタスクビューに一覧表示されます。タスクビューからはコメントが記述されている位置にジャンプすることができます。

この機能はJava、XML、HTML、JSPなどで使用できます。以下の設定ページでTODO以外のタスクタグを追加することで、いろいろな用途に利用することができます。

ファイルタイプ	設定ページ
Java	[Java] > [コンパイラ] > [タスクタグ]
XML、HTML、JSPなど	[一般] > [エディタ] > [構造化テキストエディタ] > [タスクタグ]

ストリングの外部化

以下のように、Javaソースに直接記述されている文字列定数を抽出して、プロパティファイルにまとめて管理することができます。

1. ビューからパッケージやソースフォルダなどを選択します。
2. メニューから[ソース] > [ストリングの外部化]を選択します。
3. [ストリングの外部化]で対象とするソースを選択し、[外部化]ボタンを押します。
4. ストリングの外部化ウィザードが起動されるので、外部化する際のプロパティファイル、キー、アクセスクラスなどを指定してストリングの外部化を行います。

10.1.4 検索

Javaの宣言を開く

Javaでコーディングしているときに、型、変数、メソッドなどの宣言を確認したい場合があります。そのような場合には、宣言を確認したいものをコード上で選択し、コンテキストメニューから[宣言を開く]を選択することで、宣言を確認することができます。

Javaの参照箇所を検索

Javaでコーディングをしているときに、影響箇所を調べたい場合があります。そのような場合には、影響箇所を調べたいものをコード上で選択し、コンテキストメニューから[参照] > [<検索したい範囲>]を選択することで、検索ビューに参照箇所の一覧を表示することができます。

検索範囲の絞り込み

検索を行う場合に以下のように検索範囲を指定することができます。

- ワークスペース
  すべての資産を対象にします。

- 選択されたリソース
  ビューで選択したリソースの配下を対象にします。

- エンクロージングプロジェクト
  ビューで選択したリソースが所属するプロジェクトを対象にします。

- ワーキングセット
  ワーキングセットを設定することで対象をカスタマイズすることができます。

検索結果の履歴

検索結果は、検索ビューに表示されます。検索ビューのツールバーには、[直前の検索を表示]ボタンがあり、検索ヒストリの中から選択することで、検索ビューに表示される内容を切り替えることができます。

プロパティファイルの検索

プロパティファイル(*.properties)はUnicode参照文字で記述するため、通常のテキスト検索では日本語などのマルチバイト文字をそのまま検索することができません。たとえば、テキスト検索で“富士通”という文字列を検索する場合、Unicode参照文字に変換した“¥u5bcc¥u58eb¥u901a”という文字列で検索する必要があります。
「プロパティ検索」を使用すると、マルチバイト文字のままで検索を行うことができます。
プロパティファイルの検索を開始するには、[検索] > [検索]を選択し、表示された[検索]ダイアログボックスの[プロパティ検索]タブを選択します。

各フィールドの指定方法は、[ファイル検索]と同じです。"ワークベンチユーザガイド"の"ファイル検索"で確認してください。

注意

- [ファイル検索]の検索結果にプロパティファイルが含まれるとき、[検索]ビューからプロパティファイルを「プロパティファイルエディタ(ShiftJIS)」で開くと、検索文字列が正しく強調表示されていない場合があります。これは、ファイル上での文字のカウント方法とエディタ上での文字のカウント方法が異なるために発生します。
  [編集] > [検索/置換]を選択して[検索/置換]ダイアログボックスに検索文字列を再指定してファイル内検索を行うか、[プロパティ検索]を使用して検索をやり直してください。
- 直接テキスト検索アクション(メニューバーの[検索] > [テキスト]配下の各コマンド)では、プロパティファイル内の日本語文字列を検索することはできません。「プロパティ検索」を利用してください。

10.1.5 ビルドおよびデバッグ

問題ビューのフィルタ機能

問題ビューに多くの問題が表示されている場合には、ビューの右上のメニューより[フィルタ]を選択することで、表示内容をカスタマイズすることができます。
フィルタには、以下のような機能があります。

- 範囲指定
  例えば、[同一プロジェクト内の任意の要素]を指定することで、選択しているリソースを含むプロジェクトに限定することができます。
- 重度の指定
  エラー、警告、情報のどれを表示するかを指定することができます。
- 問題のタイプの指定
  ビルダやバリデータなどの問題のタイプを指定して、表示内容を限定することができます。
- その他
  特定の記述を含む/含まないを指定して、表示内容を限定することができます。

ビルド時のAntの呼び出し

Antスクリプトを記述して、以下のように設定することでビルド時にAntを呼び出すことができます。

1. プロジェクトのプロパティから[ビルダ]を選択します。
2. [新規]をクリックし、[構成タイプの選択]ダイアログボックスで[Antビルダ]を選択します。
3. [メイン]の[ビルドファイル]に、使用するAntスクリプトファイルを指定します。
4. Antスクリプトでワークスペース内のファイルを更新する場合には、[更新]タブでAntスクリプト実行後に最新化を行いたい資産を指定します。
5. [ターゲット]タブでは、手動ビルド、自動ビルドなどビルドの仕方に合わせてターゲットを設定することができます。
6. [プロパティ]タブでは、Antの実行に必要なプロパティを指定できます。

ポイント

Antスクリプトを選択して、コンテキストメニューから[デバッグ] > [Antビルド]を選択することで、Antスクリプトのデバッグをすることもできます。

クラスパスの追加

ワークスペース内にないJARファイルをクラスパスに追加する場合は、以下のどちらかの方法を使用すると環境依存の部分を吸収することができ、プロジェクト資産の可搬性が向上します。

クラスパス変数

設定ページの[Java] > [ビルドパス] > [クラスパス変数]で、名前とパスを対応付けて管理します。
クラスパスへの追加の際は、プロジェクトプロパティの[Javaのビルドパス]の[ライブラリ]タブの[変数の追加]ボタンから行い、表示されたダイアログボックスでクラスパス変数を選択し、[拡張]ボタンから、クラスパス変数に対応付けられているパスの配下のリソースを選択します。

ユーザライブラリ

設定ページの[Java] > [ビルドパス] > [ユーザライブラリ]で、名前と複数のJARファイルを対応付けて管理します。ライブラリ群を指定する場合に利用すると便利です。
指定したライブラリのクラスパスへの追加は、プロジェクトプロパティの[Javaのビルドパス]の[ライブラリ]タブの[ライブラリの追加]ボタンから行います。

ワークベンチを起動せずに、コマンドラインよりビルドする

ワークベンチを起動せずに、コマンドラインよりプロジェクトをビルドするには、ツール isstudiobld.exe を使用します。ツール isstudiobld.exe は、以下の場所にあります。

〈ワークベンチのインストールフォルダ〉¥eclipse¥isstudiobld.exe

以下にツール isstudiobld の使い方を説明します。

形式	
isstudiobld -data <workspace> [オプション] [<target> [<target2>...]] 注) [ ]は省略可能です。< >は該当する値を指定します。	
パラメタ	
-data <workspace>	ワークスペースフォルダを指定します。
<target>	実行するAntのターゲットを指定します。
オプション	
-f <buildfile>	ビルドファイル(Antスクリプト)を指定します。 省略した場合は、以下の場所にあるビルドファイルを使用します。 <インストールフォルダ>¥IDE¥1101¥etc¥build¥buildAll.xml
-verbose または -v	詳細を表示します。
-D<property>=<value>	プロパティを指定します。
-propertyfile <name>	プロパティを指定されたファイルからロードします。
-vm <JDKのインストールフォルダ>¥jre¥bin	ビルドファイル(Antスクリプト)を実行するJDKのjava.exeが存在するフォルダを指定します。 省略した場合は、環境変数PATHに指定されたJDKが使用されます。

注意

- プロジェクトをビルドするには eclipse.incrementalBuild Antタスクを使います。
- eclipse.incrementalBuild Antタスク内では、プロジェクトに指定されている[Javaコンパイラ]や[Javaのビルドパス]の情報を元にビルドを行います。
  eclipse.incrementalBuild以外のAntタスクでは、-vmオプションで指定したJDKを使用します。

eclipse.incrementalBuild Antタスクの使い方は以下です。

属性	説明
kind	ビルドの種類。incremetal, full, cleanのいずれかを指定します。デフォルト値はincremental。

属性	説明
project	ビルドするプロジェクトを指定します。省略した場合はワークスペースをビルドします。

例: ワークスペースをフルビルドする

```
<eclipse.incrementalBuild kind="full" />
```

例: プロジェクト project1 をクリーンする

```
<eclipse.incrementalBuild project="project1" kind="clean" />
```

ブレークポイントの一時的な無効化

ブレークポイントを多数設定していると余分な箇所での実行中断が多くなり、デバッグ効率が悪くなることがあります。

このような場合に、ブレークポイントを解除しないで一時的に無効にすることができます。ブレークポイントビューの右上のメニューから[グループ] > [<グループ化の方法>]を選択すると、ブレークポイントがグループ化されるので、グループ化されたブレークポイントをまとめて無効化することができます。
また、ブレークポイントビューのツールバー上から[すべてのブレークポイントをスキップ]を選択することで、すべてのブレークポイントを無効にすることもできます。

例外ブレークポイント

例外が発生したときにプログラムの実行を中断したい場合があります。このような場合には、例外ブレークポイントを利用します。メニューから[実行] > [Java例外ブレークポイントの追加]を用いて例外クラスを指定することで、その例外が発生した際に実行を中断するように設定することができます。

選択メソッドにステップイン

デバッグ時に、例えば

addValue(obj.getName(), obj.getValue());

のような文をステップイン実行する際に、単にステップイン(F5)を実行すると以下の順番でステップインが行われます。

1. obj.getName()にステップイン
2. obj.getValue()にステップイン
3. addValue()にステップイン

もしaddValue()だけにステップインしたいのであれば、エディタ上で"addValue"を選択し、メニューから[選択項目にステップイン]を選択します。そうすると1.と2.は行われず、即座に3.が行われます。

式の評価

デバッグ中にオブジェクトの内容の確認だけでなく、メソッドを実行してみたい場合があります。
そのような場合には、表示ビューまたは変数ビューの詳細ペインに式を入力して選択し、コンテキストメニューから、[表示]、[インスペクション]、または[実行]を選択します。

ビルドやデバッグに使用するJavaのバージョンを指定するには

ビルドやデバッグに使用するJavaのバージョンは、プロジェクトのビルドパスに指定しているJREシステムライブラリによって決まります。プロジェクトのJREシステムライブラリを構成することでJavaのバージョンを指定できます。

JREシステムライブラリでJavaのバージョンを指定する方法には、以下の種類があります。

- ワークスペースのデフォルトJRE
  ワークスペースのデフォルトJREを使用します。ワークスペースのデフォルトJREは、メニューバー[ウィンドウ] > [設定]で表示される設定ダイアログボックスの[Java] > [インストール済みのJRE]ページでチェックされているインストール済みのJREです。ワークスペースのデフォルトJREを変更すると、各プロジェクトで使用されるJREシステムライブラリも変更されます。
- 代替JRE
  Javaのバージョンをプロジェクトごとに指定する場合に使用します。インストール済みのJREの中から、ビルドやデバッグに使用するJavaを選択します。

プロジェクトのJREシステムライブラリを構成するには、以下の手順で行います。

1. パッケージエクスプローラビューなどからプロジェクトを選択します。
2. コンテキストメニューから[プロパティ]を選択するか、メニューバーから[ファイル] > [プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. 左のペインで[Javaのビルドパス]を選択します。[Javaのビルドパス]ページが表示されます。
4. [ライブラリ]タブを選択します。
5. 一覧[ビルドパス上のJARおよびクラスフォルダ]から、[JREシステムライブラリ]を選択し、[編集]をクリックします。[ライブラリの編集]ダイアログボックスが表示されます。
6. システムライブラリから[ワークスペースのデフォルトJRE]か[代替JRE]を選択します。[代替JRE]を選択した場合は、コンボボックスで利用するインストール済みJREを選択します。

インストール済みのJREにJDKを追加するには、以下の手順で行います。

1. ワークベンチのメニューから[ウィンドウ] > [設定]を選択します。
2. [設定]ダイアログボックスの左のペインで[Java] > [インストール済みのJRE]を選択します。
3. 右の[インストール済みのJRE]画面で[追加]をクリックします。
4. [JREの型]画面では"標準VM"を選択して[次へ]をクリックします。
5. [JRE定義]画面の[JREホーム]に、JDKのインストールフォルダを指定します。[JRE名]に選択したフォルダ名に応じた名前が表示され、[JREシステムライブラリ]に指定したJDKのライブラリのJARファイルの一覧が表示されます。[完了]をクリックして画面を閉じます。

注意

- システムライブラリの選択項目として[実行環境]がありますが、Interstage Studioでは使用できません。
- コンパイラ準拠レベルは、JREシステムライブラリで指定されているJavaのバージョンより大きくしないでください。

ポイント

Javaコンパイルでは、JREシステムライブラリのほかに、クラスファイルの互換性やソースの互換性などのコンパイラ準拠レベルを指定できます。コンパイラ準拠レベルは、プロジェクトのプロパティの[Javaコンパイラ]ページ、もしくは、ワークスペースの設定の[Java] > [コンパイラ]ページで指定できます。

スタックトレースビュー

スタックトレースビューを使用することで、FJVMログまたはスレッドダンプツールで採取したスレッドダンプを読み込んで、そこに記録されているスレッドとスタックトレースの一覧を階層表示することができます。

スタックトレースビューは以下の手順で表示できます。

1. メニューバーから[ウィンドウ] > [ビューの表示] >[その他]を選択します。
2. [ビューの表示]ダイアログボックスで[Java] > [スタックトレース]を選択します。

参照

FJVMログの詳細については、"Interstage Application Server チューニングガイド"の"JDK/JREのチューニング"を参照してください。

スレッドダンプツールの詳細については、"Interstage Application Serverトラブルシューティング集"の"Javaツール機能"の"スレッドダンプツール"を参照してください。

注意

- Java EE 6ワークベンチではスタックトレースビューは使用できません。
- スタックトレースビューで表示可能なスレッドダンプのファイルは、スレッドダンプツールのオプション "-f"（スレッドダンプの出力先の指定）を指定して出力されたファイルになります。標準出力やアプリケーションサーバのログファイルに記録されたスレッドダンプは表示できません。

スタックトレースビューには、以下の機能があります。

- FJVMログ/スレッドダンプを開く
  ビューに表示するFJVMログファイルまたはスレッドダンプを開きます。開いたファイルの内容はコンソールビューにも表示されます。
- ファイルを閉じる
  ビューに現在表示されているファイルを閉じます。
- 選択されたファイルの表示
  ファイルを複数個開いている場合に、どのファイルを表示するかを選択します。
- コンソールの表示
  ビューに現在表示されているファイルの内容をコンソールビューに表示します。
- ソート
  スレッドをソート表示します。ボタンを押すたびにソートのオン/オフが切り替わります。
  ソートがオンの場合には、先に通常のスレッドがスレッド名で昇順に表示され、次にシステムスレッドがスレッド名で昇順に表示されます。ソートがオフの場合には、ファイルに記録されている順番にスレッドが表示されます。
- システムスレッドの表示
  システムスレッドを表示します。コマンドを選択するたびにシステムスレッドの表示/非表示が切り替わります。
- 修飾名の表示
  スタックトレースやモニタのエレメントに表示されるクラス名に修飾名（パッケージ名）を付けて表示します。
- モニタの表示
  モニタを表示します。スレッドダンプでデッドロック状態のスレッドやモニタがある場合には、それらは強調表示され、アイコンもデッドロック状態のものに変わります。
  コマンドを選択するたびにモニタの表示/非表示が切り替わります。
- エンコードの設定
  FJVMログファイルやスレッドダンプファイルを読み込む際に使用するエンコードを指定します。デフォルトはUTF-8です。ファイルがUTF-8以外のエンコードで記録されている場合にだけこの設定を行ってください。

スレッドダンプにモニタの情報が記録されている場合には、[モニタの表示]をオンにすると以下のエレメントが表示されます。

エレメント	アイコン		説明
所有するモニタ			スレッドの子エレメントとして表示され、そのスレッドが所有しているモニタを表します。アイコンの右側にはモニタとなっているオブジェクトのクラス名とIDが表示されます。所有しているモニタが複数ある場合には、このエレメントが複数個表示されます。 モニタがデッドロック状態にある場合には右側のアイコンが表示されます。
モニタを待機中のスレッド			所有するモニタの子エレメントとして表示され、そのモニタを待機中のスレッドを表します。アイコンの右側にはスレッド名と、システムスレッドか否かが表示されます。 待機中のスレッドが複数ある場合には、このエレメントが複数個表示されます。 スレッドがデッドロック状態にある場合には右側のアイコンが表示されます。
待機中のモニタ			スレッドの子エレメントとして表示され、そのスレッドが待機中のモニタを表します。アイコンの右側にはモニタとなっているオブジェクトのクラス名とIDが表示されます。 モニタがデッドロック状態にある場合には右側のアイコンが表示されます。
モニタを所有するスレッド			待機中のモニタの子エレメントとして表示され、そのモニタを所有しているスレッドを表します。アイコンの右側にはスレッド名と、システムスレッドか否かが表示され

エレメント	アイコン	説明
		ます。 スレッドがデッドロック状態にある場合には右側のアイコンが表示されます。
モニタのロック		スタックトレースの間に表示され、どの処理の間にモニタがロックされたかを表します。アイコンの右側にはモニタとなっているオブジェクトのクラス名とIDが表示されます。

10.1.6 ヘルプ

ヘルプにドキュメントを追加する

開発作業中に参照するお客様固有のドキュメントを、ヘルプシステムに追加することができます。ドキュメントの追加は[ウィンドウ]メニューの[設定]で[ヘルプ]>[ドキュメント]を選択して行います。

ドキュメントの追加および編集で表示される[ユーザドキュメント]画面には、次の値を指定します。

項目		設定値
ドキュメント名		ドキュメントの名称を指定します。
ドキュメントURL		ドキュメントがアーカイブされていない場合に選択します。
	ロケーションパス	ドキュメントの初期ページを、URL形式で指定します。ローカルファイルの場合は、"file:〜"で指定します。
アーカイブ内のドキュメント		ドキュメントがZIPファイルまたはJARファイルにアーカイブされている場合に選択します。
	アーカイブパス	アーカイブファイルへのパスを指定します。
	アーカイブ内のパス	アーカイブファイル内の初期ページを指定します。
初期ページのリンクを自動認識して目次を作成		このオプションを選択すると、ロケーションパスまたはアーカイブ内のパスで指定されたHTMLページ内のリンクを抽出し、目次を作成します。 アーカイブされていないドキュメントの場合は、ロケーションパスが"file:〜"で開始している場合にだけ有効です。
同一フォルダ内のドキュメントも検索		このオプションを選択すると、ドキュメント内の下記のHTMLページを検索の対象として追加することができます。 - *.html, *.htm - *.xhtml, *.xhtm ただし、javadocの場合は次のHTMLページだけを検索の対象として追加します。 - overview-summary.html - package-summary.html このオプションを選択しない場合は、目次に登録されているHTMLページだけが検索の対象となります。

注意

ロケーションパス、アーカイブパスおよびアーカイブ内のパスには、日本語文字を含まないように指定してください。日本語文字を含むパスを指定した場合、HTMLページが正しく表示されません。

Infopopヘルプ

ダイアログボックスやウィザードなどの使い方がわかない場合には、そのGUIを表示している状態で[F1]キーを押します。そうすることで、そのGUIと対応付けられているヘルプが表示されます。

10.1.7 その他

ワークベンチを複数表示する

ビューやエディタで異なる箇所を同時に参照したい場合やパースペクティブを切り替えるのが面倒な場合には、メニューから[ウィンドウ] > [新規ウィンドウ]を選択することで、ワークベンチを複数表示することができます。

ビューの表示方法

エディタと同様にタイトルバーをダブルクリックすることで、ビューを最大化し、一時的に見やすくすることができますが、それ以外にも以下のような方法でビューを好みに合わせて表示することができます。

- ビューの切り離し
  ビューのタイトルバーで右クリックして、コンテキストメニューの[切り離し]を選ぶと、ワークベンチのウィンドウからビューを切り離して表示できます。
  こうすることで、切り離されたビューは任意の場所に任意の大きさで前面に表示することができます。ビューをワークベンチのウィンドウ内に戻すには、コンテキストメニューで再度[切り離し]を選びます。

- 高速ビュー
  ビューのタイトルバーで右クリックして、コンテキストメニューの[高速ビュー]を選ぶと、ビューの表示/非表示をワークベンチのステータスバー上のボタンから切り替えることができます。高速ビュー表示を止めるには、コンテキストメニューで再度[高速ビュー]を選びます。

設定、プロパティ画面でのフィルタ

プロジェクトプロパティや設定ダイアログボックスの左上にキーワードを入力することで、左端に表示するページを絞り込むことができます。

リファクタリング(名前変更)

変数、フィールド、メソッド、クラス、パッケージなどの名前を変更したい場合には、以下のようにリファクタリング機能を使用すると呼び出し箇所も含めて必要な修正を一度に行うことができます。

1. エディタまたはビューで変更したい対象を選択します。
2. メニューから[リファクタリング] > [名前変更]を選択します。
3. 表示される内容に従って名前を変更します。

差分を確認する

プロジェクトエクスプローラビューなどで比較するリソースを選択し、コンテキストメニューから[比較] > [相互]を選択することで、フォルダやファイル単位に差分を確認することができます。

javadoc

javadocビューでは、選択しているクラス、メソッド、フィールドなどの要素のjavadocを表示することができます。また、エディタ上で、その要素上にマウスカーソルを持っていくと、吹き出しでjavadocを確認することもできます。
表示対象となるjavadocは、ソースにjavadocが記述してあるものか、JARファイルのプロパティの[Javadocロケーション]が記述してあるものになります。
また、後者のjavadocについては、メニューから[ナビゲート] > [外部Javadocを開く]を選択することで、ブラウザでjavadocを表示することもできます。

ポイント

javadocドキュメントを作成したい場合には、エクスポートウィザードの[Java] > [Javadoc]で作成できます。

10.2 プログラミングテクニック編

ここでは、アプリケーション開発時の留意事項などを紹介します。

10.2.1 Webアプリケーションにおける文字コードの考慮について

要求オブジェクトからパラメタの値を取得する場合には、入力フォームの文字コードを意識した処理をする必要があります。そのため、Webアプリケーションでは文字コードを統一しておくことをお勧めします。
HTML、JSPおよび、サーブレットについては、以下のように文字コードを設定します。

Webコンポーネント	設定方法	記述例
HTML	metaタグのContent-Type（charset）の値で指定します。	<meta http-equiv="Content-Type" content="text/html; charset=Windows-31J" />
JSP	pageディレクティブのcontentType属性でHTTPヘッダのContent-Type(charset)の値を指定します。	<%@ page language="java" contentType="text/html; charset=Windows-31J"%>
サーブレット	HttpServletResponseインタフェースのsetContentTypeメソッドでHTTPヘッダのContent-Type(charset)の値を指定します。	res.setContentType("text/html; charset=Windows-31J");

ポイント

- JSPでは、ファイル保存時の文字コードを別に指定できます。pageディレクティブのpageEncoding属性に記述します。
- HTMLとJSPの新規ウィザードでは、ファイル生成時に文字コードを埋め込むための方法を提供しています。埋め込む文字コードについては、ワークスペース単位に指定でき、それぞれ設定ダイアログボックスの以下で指定します。
  HTML: [Web] > [HTMLファイル]
  JSP: [Web] > [JSPファイル]

フィルタで文字コードを考慮する場合は、たとえば、以下のクラスを作成し、web.xmlでフィルタマッピングを定義する必要があります。この例では、web.xmlの初期パラメタに文字コードを指定し、このフィルタクラスを利用する環境によってカスタマイズできるようになっています。そのため、初期パラメタについても定義する必要があります。

フィルタでの文字コードの考慮例

```
package filter;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class flt_request implements Filter {
  private FilterConfig config=null;
  public void init(FilterConfig conf){
    this.config=conf;
  }
  public void destroy(){}
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
 throws ServletException, IOException {
    request.setCharacterEncoding(config.getInitParameter("encoding"));
    chain.doFilter(request,response);
  }
}
```

注意

setCharacterEncodingの文字コード指定は、POSTメソッドには有効ですが、GETメソッドがパラメタを渡すURL部分には適用されないため、GETメソッドでは有効になりません。

10.2.2 JNDIのlookupによるオブジェクトの取得について

Java EEでは、Dependency Injectionによってリソースやオブジェクトを取得できますが、Dependency Injectionでは以下のような問題があります。

- Java EEコンテナで管理されるコンポーネントのみで使用可能
  Dependency Injectionは、Java EEコンテナによって依存リソース/オブジェクトの注入が行われます。そのため、Java EEコンテナによって管理されるコンポーネントのみで使用可能であり、任意のクラスで使用できる方法ではありません。

- プロパティはコードへ直接記述
  Dependency Injectionの際にアノテーションにプロパティを記述する場合、その記述はソースに直接記述することになります。そのため、プロパティを変更するとJavaソースのコンパイルが必要になります。
  lookupの場合には、参照名とJNDI名の対応付けはdeployment descriptorで変更できるため、実際の環境を意識せずに使用することができます。

上記のような問題を解決するためには、従来と同様にJNDIのlookupによってリソースやオブジェクトの取得を行います。
以下に、いくつか例を示します。

EJBのローカルインタフェースのlookup

以下のようにEJBのローカルインタフェースをlookupします。

lookupの例

```
InitialContext ic = new InitialContext();
Calc calc = (Calc)ic.lookup("java:comp/env/Calc");
```

lookupのためのdeployment descriptorの記述例

```
<ejb-local-ref>
    <ejb-local-ref-name>Calc</ejb-local-ref-name >
    <local>sample.Calc</local>
    <ejb-link>ejb1.jar#CalcBean</ejb-link>
</ejb-local-ref >
```

JMS Destinationのlookup

以下のようにJMS Destinationをlookupします。

lookupの例

```
InitialContext ic = new InitialContext();
Queue queue = (Queue)ic.lookup("java:comp/env/jms/myQueue");
```

lookupのためのdeployment descriptorの記述例

```
<message-destination-ref>
    <message-destination-ref-name>jms/myQueue</message-destination-ref-name>
    <message-destination-type>javax.jms.Queue</message-destination-type>
    <message-destination-usage>Produces</message-destination-usage>
    <message-destination-link>myDestination</message-destination-link>
</message-destination-ref >
...
<message-destination>
    <message-destination-name>myDestination</message-destination-name>
    <mapped-name>testQueue</mapped-name>
</message-destination>
```

JDBCリソースのlookup

以下のようにJDBCリソースをlookupします。

lookupの例

```
InitialContext ic = new InitialContext();
DataSource symfo = (DataSource)ic.lookup("java:comp/env/jdbc/Symfo");
```

lookupのためのdeployment descriptorの記述例

```
<resource-ref>
    <res-ref-name>jdbc/Symfo</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <mapped-name>SymfoTestServer</mapped-name>
</resource-ref>
```

10.3 Eclipseプラグイン利用編

ここでは、Eclipseプラグインの利用方法および留意事項について説明します。

10.3.1 プラグインのインストール・アンインストールの手順

Eclipseプラグインを利用する場合、以下の手順でインストール・アンインストールしてください。

インストールフォルダ内のdropinsフォルダにプラグインを格納することでワークベンチにプラグインをインストールできます。

ポイント

インストールフォルダは、標準インストールの場合は <ワークベンチのインストールフォルダ>¥eclipse になります。

インストール手順

1. ワークベンチを終了します。
2. インストールするプラグインをインストールフォルダ内のdropinsフォルダに格納します。
   プラグインの格納方法は、以下の格納方法がサポートされています。

dropinsフォルダにプラグインを格納

```
eclipse/
  dropins/
    com.fujitsu.*_1.0.0.jar
    com.fujitsu.*_1.0.0/
      plugin.xml
      tools.jar
      ... etc ...
    ...
```

dropinsフォルダにeclipseフォルダを格納

```
eclipse/
  dropins/
    eclipse/
      features/
      plugins/
```

dropinsフォルダにコンポーネントごとのフォルダを格納

```
eclipse/
  dropins/
    gef/
      eclipse/
        features/
        plugins/
    emf/
      eclipse/
        features/
        plugins/
    ... etc ...
```

dropinsフォルダにlinkファイルを格納

```
eclipse/
  dropins/
    fujitsu.link
```

3. 上記2の操作を行った直後のみ、[ファイル名を指定して実行]や[コマンドプロンプト]から、ワークベンチを"-clean"オプション付きで起動します。
   標準インストールの場合は、"<ワークベンチのインストールフォルダ>¥eclipse¥isstudio.exe -clean" で起動します。
4. [ヘルプ] > [Interstage Studioについて] > [フィーチャーの詳細]もしくは[プラグインの詳細]で、インストールしたフィーチャーもしくはプラグインが表示されていればインストール完了です。

アンインストール手順

1. ワークベンチを終了します。
2. インストールフォルダ内のdropinsフォルダに格納したプラグインを削除します。
3. 上記2の操作を行った直後のみ、[ファイル名を指定して実行]や[コマンドプロンプト]から、ワークベンチを"-clean"オプション付きで起動します。
   標準インストールの場合は"<ワークベンチのインストールフォルダ>¥eclipse¥isstudio.exe -clean" で起動します。
4. [ヘルプ] > [Interstage Studioについて] > [フィーチャーの詳細]もしくは[プラグインの詳細]で、インストールしたフィーチャーもしくはプラグインが表示されていなければアンインストール完了です。

注意

プラグインの格納方法でlinkファイルを使用した場合は、linkファイルに指定したプラグインも削除してください。

10.3.2 Eclipseプラグインをインストールする際の留意事項

- インストールするEclipseプラグインに関するドキュメントを参照して、インストールするEclipseプラグインが、ワークベンチのEclipseのバージョンと互換性があることを確認してください。
- インストールしたEclipseプラグインの動作確認はお客様自身で行ってください。
- Eclipseプラグインをインストールした際の動作は保証していません。
- Eclipseプラグインをインストールしたことに起因する問題についてはサポートを受けられません。
- 以下の表のEclipseプラグインに対して富士通で修正(バグ修正、機能追加)を行っているので、Eclipse Foundationで公開されているEclipseと同一のものではありません。

  ＜標準のワークベンチ＞

プラグイン	バージョン
Eclipse	3.4.1
GEF	3.4.1

プラグイン	バージョン
EMF	2.4.1
WTP	3.0.1
DTP	1.6.1

＜Java EE 6ワークベンチ＞

プラグイン	バージョン
Eclipse	3.6.2
GEF	3.6.2
EMF	2.6.1
WTP	3.2.3
DTP	1.8.2

- 基本的には、Eclipse FoundationのEclipseと互換性がありますが、富士通が行った修正がインストールするEclipseプラグインに影響する可能性もあります。

10.4 シンクライアント環境編

ここでは、シンクライアント環境で使用する場合の留意事項について説明します。

- ワークスペースはユーザごとに作成する必要があります。

  他のユーザとワークスペースを共有することはできません。必ずユーザごとに1つ以上のワークスペースを作成してください。

- リンクされたフォルダ、ファイルやワークスペース外のプロジェクトは参照のみ使用できます。

  リンクされたフォルダ、ファイルやワークスペース外のプロジェクトで、他のユーザと同時に作業することはできません。参照することのみ可能です。

- 環境設定はワークスペースごとに保存されます。

  [ウィンドウ] > [設定]で表示される設定ページで行う設定は、他のユーザと共有することができません。同じ設定をする必要がある場合には個別に設定してください。

- IJServerクラスタはユーザごとに作成する必要があります。

  他のユーザと同じIJServerクラスタを使用することはできません。ユーザごとに必ず別のIJServerクラスタを作成してください。Interstage Java EE DASサービスを使用することはできません。

- Interstage基盤サービス操作ツールを用いてサービスを停止することは推奨されません。

  どうしてもサービスを停止する必要がある場合には、他のユーザがInterstageやInterstage管理コンソールを使用していないことを確認してから停止してください。

注意

Windows Server 2003/Windows Server 2008のターミナルサービスまたはWindows Server 2012のリモートデスクトップサービスで、開発環境製品やクライアント運用環境製品を使用する場合、ターミナルサービスまたはリモートデスクトップサービスを同時に利用するクライアント台数分のライセンスの購入が必要です。

10.5 IPv6環境での開発編

ここでは、IPv6環境で使用する場合の使用方法および留意事項について説明します。

注意

本製品は、IPv6/IPv4デュアルスタックのみサポートしています。IPv4を無効にした場合の運用はサポートしておりません。

(注)以下の機能はIPv4環境だけをサポートしています。

- ワークベンチの[サーバ]ビュー機能
- デバッグ機能全般

10.5.1 IPv6アドレスの指定方法

ワークベンチおよびJava EE 6ワークベンチにおいて、IPv6アドレスを使用する場合、以下のhostsファイルにIPv6ユニキャストアドレスとホスト名の宣言を追加します。宣言を追加したうえで、ホスト名を指定してください。

(Windowsインストールフォルダ)¥system32¥drivers¥etc¥hosts

10.5.2 IPv6環境でのアプリケーション開発における留意事項

IPv6環境でアプリケーション開発を行う場合、以下の留意事項があります。

Java EE 5、J2EE1.4アプリケーションの開発

- "6.2.1 アプリケーションの動作確認を行う配備先の準備"でIJServerクラスタやIJServerなどを作成する場合は、localhostに作成してください。
- 動作確認にはlocalhostのサーバを使用します。[新規サーバ]ウィザードなど、サーバを追加する際に指定する[サーバのホスト名]には"localhost"を指定してください。
- Webサービス(JAX-WS)ウィザードの[WSDLファイル]に指定するURLには、"localhost"に存在するサーバのURLを指定してください。
- 運用環境への配布は、ftpやフォルダ共有などの機能を使用して、サーバにアーカイブファイルを転送後、Interstage Application Serverの配備機能を利用して配備してください。

アプリケーションのデバッグ

- デバッグされるプログラムのマシンは、"localhost"またはIPv4アドレスで指定する必要があります。

付録A　サンプルの使用方法

ここでは、提供しているサンプルについて説明します。

サンプルの格納先

標準のワークベンチおよびJava EE 6ワークベンチのサンプルアプリケーションは、以下の場所に格納されています。

<標準のワークベンチのインストールフォルダ>¥sample

<Java EE 6ワークベンチのインストールフォルダ>¥sample

サンプルの一覧

ファイル名	プロジェクト名	備考
WebSample.zip	WebSample	"Webアプリケーションを開発する"の"2.2 入門"で作成したアプリケーションです。
EJBSample.zip	EJBSample	"Enterprise JavaBeans(EJB)開発する"の"3.2 入門"で作成したアプリケーションです。
	EJBClientSample	
	EJBEARSample	
JPASample.zip	JPASample	"Java Persistence APIを使用したアプリケーションを開発する"の"4.2 入門"で作成したアプリケーションです。
	JPAClientSample	
	JPAEARSample	
WebServiceSample.zip	WebServiceSample	"Webサービスのアプリケーションを開発する"の"5.2 入門"で作成したアプリケーションです。
	WebServiceClientSample	
	WebServiceEARSample	
AppletSample.zip	AppletSample	"Javaアプリケーションを開発する"の"7.2 入門"で作成したアプリケーションです。

Java EE 6ワークベンチのサンプルは、ファイル名およびプロジェクト名の末尾に"6"が付いています。

サンプルの利用手順

サンプルは以下の手順でワークスペースにインポートします。

1. ワークベンチを起動します。
2. メニューから[ファイル] > [インポート]を選択します。
3. インポートソースの選択で[一般] > [既存プロジェクトをワークスペースへ]を選択し、[次へ]をクリックします。
4. 次画面で[アーカイブファイルの選択]を選択し、[参照]をクリックしてインポートするアーカイブファイルを指定します。
5. [終了]をクリックします。

注意

- Java EE 6ワークベンチのAppletSample6.zip以外の場合、インポート前に"9.6.1.1 サーバを操作するための準備をする"でInterstage Application Server V11.1(Java EE 6)ランタイムを登録する必要があります。
- 本製品を"C:¥Interstage"以外にインストールしている環境で、Java EE 6ワークベンチのWebサービスのサンプル(WebServiceSample6.zip)を利用する場合、サンプルのインポート後に"9.5.1.3 Webサービスアプリケーションの開発"の注意を参考に、WebServiceClientSample6プロジェクトのビルドパスの再設定を行って下さい。

付録B　旧資産からの移行

ここでは、旧バージョン(Interstage Apworks、Interstage Studio)のワークベンチで開発した資産の移行について説明します。

移行の手順

旧バージョン・レベルで作成したワークスペースやプロジェクトを利用するには、以下の手順に従ってください。移行に関する注意点については次節以降を参照してください。

旧バージョン・レベルで作成したワークスペースを利用する

ワークベンチの起動ダイアログボックスの環境設定で旧バージョン・レベルのワークスペースを指定して、ワークベンチを起動します。ワークスペースおよび、その中に格納されているプロジェクトがワークベンチで利用可能となります。

旧バージョン・レベルで作成したプロジェクトを利用する

ワークスペース全体ではなくワークスペース内の個々のプロジェクトを移行したい場合には、以下の手順でインポートすることができます。

1. インポートするプロジェクトを、エクスプローラなどを用いてワークスペースフォルダにコピーします。
2. メニューバーから[ファイル] > [インポート]を選択し、[一般] > [既存プロジェクトをワークスペースへ]を選択します。
3. [プロジェクトのインポート]画面で、[ルートフォルダの選択]に1.でプロジェクトをコピーしたワークスペースフォルダを指定します。
4. [プロジェクト]の欄にインポート可能なプロジェクトの一覧が表示されます。インポートしたいプロジェクトがチェックされていることを確認し、[完了]をクリックします。

コンポーネントデザイナで作成したプロジェクトを利用する

コンポーネントデザイナのプロジェクトを移行したい場合には、以下の手順でインポートすることができます。

1. メニューバーから[ファイル] > [インポート]を選択し、[その他] > [既存のコンポーネントデザイナのプロジェクトをワークスペースへ]を選択します。
2. [インポートするプロジェクト]にインポートするプロジェクトファイルのパスを指定します。
3. [プロジェクト名]にプロジェクト名が表示されていることを確認し、[完了]をクリックします。

ポイント

いくつかのワークスペースおよびプロジェクト設定は[旧バージョンのワークスペース/プロジェクトの更新]コマンドを用いて自動的に更新することができます。詳しくは"B.6 ワークスペースおよびプロジェクトの自動更新"を参照してください。

注意

"B.6 ワークスペースおよびプロジェクトの自動更新"を行わない場合、ビルドを行うと問題ビューに"旧バージョンのプロジェクトであるためビルドできません。旧バージョンのプロジェクトの更新を行ってください"というメッセージが表示される場合があります。"B.6 ワークスペースおよびプロジェクトの自動更新"を行ったあとは正しくビルドできるようになります。

B.1　V10までの資産の移行に関する注意点

ここではV10の資産を移行する際の注意点を説明します。

注意

本バージョンのJava EE 6ワークベンチに資産を移行できるのは、旧版のJava EE 6ワークベンチで作成された資産、およびJavaプロジェクトとJavaアプリケーションプロジェクトの資産です。

B.1.1 ワークスペースの移行に関する注意点

V10のワークスペースを本バージョンのワークベンチで利用する場合には、以下の注意点を参照してください。

テンプレートの移行について

V10の互換ワークベンチで提供していたテンプレートビューは、提供していません。V10の互換ワークベンチのワークスペースを本バージョンのワークベンチで開いた場合、V10の互換ワークベンチで利用していたテンプレートビューのテンプレートをワークベンチで利用できます。利用できるのは、拡張子がjava, xml, html, htm, jsp, js, css, dtd, sql のファイルに対するテンプレートです。これらのテンプレートは、各対応するエディタのテンプレートとして利用できます。

V10の互換ワークベンチのテンプレートビューで利用していたテンプレートを、各エディタのテンプレートとして利用するには、テンプレートを移行する必要があります。以下にテンプレートの移行手順を示します。

1. メニューバーから[プロジェクト] > [旧バージョンのワークスペース/プロジェクトの更新]を選択します。
2. [旧バージョンのワークスペース/プロジェクトの更新]ダイアログボックスが表示されます。ワークスペースの更新を実施していない場合は、ダイアログボックスの左側に「ワークスペース」が表示されるので、「ワークスペース」をチェックして[OK]をクリックします。テンプレートの移行に必要なファイルが生成されます。
3. 作業用の任意のプロジェクトに、次のファイルをコピーします。
<ワークスペースフォルダ>¥.metadata¥templateBackups¥studioTemplates.xml
4. 3でコピーしたXMLファイルをXMLエディタで開きます。開いたxmlファイルは、以下のようなツリー構造になっています。

```
<syntaxtemplate>
  + <root>
    + <category>
      + <template>
```

<template>要素のname属性がテンプレートの名前です。description属性はテンプレートの説明です。context属性がコンテキストです。<template>要素の子テキストがテンプレートです。

5. 移行の必要のないテンプレートを削除します。削除するには、XMLエディタの設計ページにて、<template>要素を削除します。
6. Java以外のテンプレートは、コンテキストが設定されていない場合があります。その場合、Context属性の属性値は、空文字列になっています。Context属性に適切なコンテキストを入力してください。ファイル種別を変更しないのであれば、インポート後にコンテキストを変更できます。この場合、インポートしたいファイル種別ごとに以下の値を設定します。

ファイルの種類	Context属性の属性値
Java	Java
XML	xml_all
CSS	css_all
DTD	dtd_new
HTML	html_all
JavaScript	javaScript
JSP	jsp_all
SQL	org.eclipse.datatools.sqltools.editor.template.sql.generic

7. XMLエディタのソースページを開き、変数名を変更します。V10の互換ワークベンチでは、${xxx}の形式の変数名に制限はなく、日本語文字などを使うことができました。本バージョンのワークベンチでは変数名には英数字とアンダーバー"_"だけを利用できます。変数名に英数字とアンダーバー"_"以外の文字を利用している場合は移行することはできません。
8. テンプレートは、各対応するエディタのテンプレートの設定からインポートします。インポートでは、そのエディタに関連したコンテキストのテンプレートだけがインポートされます。

J2EE関連のテンプレートの移行について

J2EEの以下のテンプレートについては、テンプレート定義ファイルを用意しています。

コンテキスト	名前	説明
Java	EJB Homeの参照	EJB Homeの参照処理を行う
	EJB Local Homeの参照	EJB Local Homeの参照処理を行う
	MessageProducer	イベントチャンネルにメッセージを送信する
	Point-To-Point	キューにメッセージを送信する
	Publish/Subscribe	トピックにメッセージを送信する

必要に応じて定義ファイルをインポートして適用してください。定義ファイルの適用手順について以下に示します。

1. メニューから[ウィンドウ] > [設定]を選択します。
2. [設定]ダイアログボックスが表示されるので、[Java] > [エディタ] > [テンプレート]を選択します。
3. [テンプレート]ページが表示されるので、[インポート]をクリックします。
4. [テンプレートのインポート]ダイアログボックスが表示されるので、以下のファイルを指定して、テンプレート定義をインポートします。

```
<ワークベンチのインストールフォルダ>¥etc¥templates¥templates_ejb_jms.xml
```

スニペットビューのタグについて

V10の互換ワークベンチのワークスペースを本バージョンのワークベンチで開いた場合、スニペットビューに定義されているタグや属性で、ワークベンチで利用できないものがあります。利用できないタグや属性をスニペットビューで表示しないようにするには、以下の定義ファイルを削除する必要があります。

```
<ユーザのドキュメントフォルダ>¥Interstage Studio¥workspace¥.metadata¥.plugins¥org.eclipse.wst.common.snippets¥user.xml
```

注意

V10の互換ワークベンチのスニペットビューで個別タグを登録していた場合は、上記の定義ファイルを削除すると個別タグの情報も削除されるため、再度スニペットビューのカスタマイズ機能を利用して、登録してください。

B.1.2 プロジェクトの移行に関する注意点

WAR/EJB-JAR/EARファイル作成に関する注意点

ワークベンチでは、WAR/EJB-JAR/EARファイルをビルド時に作成しません。WAR/EJB-JAR/EARファイルはエクスポートウィザードを用いて作成します。V10の互換ワークベンチのプロジェクトに対して"B.6 ワークスペースおよびプロジェクトの自動更新"をおこなった場合、WAR/EJB-JAR/EARファイルを作成するためのJ2EEパッケージャビルダは削除されます。代わりに、これらのアーカイブファイルを作成するためのANTスクリプトファイルがプロジェクトに作成されます。

生成するアーカイブファイルの種類	ANTスクリプトファイル
WARファイル	buildWAR.xml
EJB-JARファイル	buildEJB-JAR.xml
EARファイル	buildEAR.xml

ポイント

EARファイルを作成するためのANTスクリプトファイル buildEAR.xmlを実行する際には、ワークスペースと同じJREで実行する必要があります。ANTスクリプトファイルをワークスペースと同じJREで実行するには、以下の設定を行います。

1. buildEAR.xmlファイルを選択し、コンテキストメニューの[実行] > [Antビルド...]を選択します。[構成の編集]ダイアログボックスが開きます。

2. [JRE]タブをクリックし、JREページを開きます。
3. [ワークスペースと同じJREで実行]をチェックします。
4. [適用]をクリックします。

ビルド時にWAR/EJB-JAR/EARファイルを作成するには、これらアーカイブファイルを作成するためのANTスクリプトファイルをビルダに追加します。以下にその手順を示します。

1. プロジェクトのプロパティを表示します。
2. プロジェクトのプロパティ画面において、左のペインで[ビルダ]を選択します。ビルダページが表示されます。
3. ビルダページの右ペインで[新規]ボタンをクリックします。[構成タイプの選択]ダイアログボックスが開きます。
4. [構成タイプの選択]ダイアログボックスの一覧より[Antビルダ]を選択し、[OK]をクリックします。[構成の編集]ダイアログボックスが開きます。
5. [名前]にこの処理の名前を入れます。
6. [メイン]タブの[ビルドファイル]の[ワークスペースの参照]ボタンをクリックします。[ロケーションの選択]ダイアログボックスが開きます。
7. [ロケーションの選択]ダイアログボックスで、アーカイブファイルを作成するためのANTスクリプトファイルを選択し、[OK]をクリックします。
8. EARファイルを作成する場合は、ANTスクリプトファイルをワークスペースと同じJREで実行するための設定を行います。[JRE]タブで[ワークスペースと同じJREで実行]をクリックします。
9. [OK]をクリックします。プロジェクトのプロパティ画面に戻ります。
10. 追加した項目がビルダの一番最後になっていることを確認します。追加した項目が一番最後に無い場合は、[上へ]ボタンおよび[下へ]ボタンを使ってビルダの一番最後に項目を移動します。

注意

ANTスクリプトファイルを実行してアーカイブファイルを作成・更新しても、ワークベンチ上ではリソースが更新されません。コンテキストメニューの[更新]を選択するなどして、リソースの変更をワークベンチに反映してください。

参考

EARファイルを作成するためのANTスクリプトファイルbuildEAR.xmlでは、直接EARファイルを作成していません。EARファイルを実際に作成しているのは、ANTスクリプトファイル buildEAR2.xmlです。ANTスクリプトファイルbuildEAR.xmlは、プロジェクトの構成に合わせてbuildEAR2.xmlを更新して、buildEAR2.xmlを呼び出しています。プロジェクトの構成に変更が無い場合は、buildEAR.xmlを呼び出す必要はありません。buildEAR2.xmlを実行するだけでEARファイルを作成することができます。

IJServer起動構成

V10の互換ワークベンチで提供していたIJServer起動構成は、提供していません。

本バージョンのワークベンチでは、「Interstage Application Server」起動構成を提供していますが、Webアプリケーションをクライアントアプリケーションとして実行する場合は、サーバビューで、配備されているWebアプリケーションプロジェクトを選択して、コンテキストメニューから[Webブラウザ]を選択することで、Webブラウザを起動して、Webアプリケーションの実行を行うことができます。

B.2 V9までの資産の移行に関する注意点

ここではV9までの資産を移行する際の注意点は、V10とほぼ同じです。

V9のワークベンチが、V10では互換ワークベンチとして提供されている点、V9.2のJava EEワークベンチが、V10の標準のワークベンチとして提供されている点に注意して、以下を参照してください。

- B.1 V10までの資産の移行に関する注意点

B.3 V8までの資産の移行に関する注意点

ここではV8までの資産を移行する際の注意点を説明します。V8までの資産を移行する際には以下も合わせて参照してください。

- B.2 V9までの資産の移行に関する注意点

B.3.1 プロジェクトの移行に関する注意点

V8までのワークスペースを本バージョンのワークベンチで利用する場合および、V8までのプロジェクトをインポートして利用する場合には、以下の注意点を参照してください。

JARパッケージビルドツールの設定

V8までのJARパッケージビルドツールは提供されなくなりました。EJB JARファイルを作成するには、EJB JARエクスポートウィザードを利用します。

JARパッケージビルドツールでは、JARに含めるリソースを1つ1つ指定することができました。しかし、EJB JARのエクスポートウィザードには、この指定が引き継がれません。JARパッケージビルドツールでJARファイルに含めるリソースを指定していた場合には、以下のようにリソースの配置を変更してください。

- JARファイルに含めたくないリソースは、ソースフォルダ以外の場所に移動してください。また、JARファイルに含めたいリソースは、ソースフォルダに移動してください。EJB JARのエクスポートウィザードはソースフォルダにあるリソースをJARファイルにアーカイブします。
- マニフェストファイルを独自に作成する場合には、そのファイルを(ソースフォルダ)/META-INF/MANIFEST.MFとして格納してください。

IDLコンパイラビルドツール

V8までのIDLコンパイラビルドツールはJavaプロジェクトでは提供されなくなりました。V8までのプロジェクトでIDLコンパイラビルドツールを使用していた場合、IDLコンパイルがビルド時に行われなくなります。ビルド時にIDLコンパイルが必要な場合は、IDLコンパイラ(idlc.exe)を外部ビルドツールとしてビルダに追加してください。

B.4 V7までの資産の移行に関する注意点

ここではV7までの資産を移行する際の固有の注意点を説明します。V7までの資産を移行する際には以下も合わせて参照してください。

- B.3 V8までの資産の移行に関する注意点

B.4.1 ワークスペースの移行に関する注意点

V7までのワークスペースを本バージョンのワークベンチで利用する場合には、以下の注意点および、"B.4.2 プロジェクトの移行に関する注意点"の両方を参照してください。

ワークスペースの更新について

ワークベンチの起動ダイアログボックスの環境設定でV7までのワークスペースを指定すると、そのワークスペースを開くときに、ワークスペースを更新するかどうかを確認するメッセージボックスが表示されます。[はい]を選択するとワークスペースの内部設定情報が更新され、本バージョンのワークベンチで使用可能になります。ワークスペースを更新したくない場合には[いいえ]を選択してください。ワークベンチはワークスペースを更新せずにそのまま終了します。

パースペクティブ設定

V7までのワークスペースを本バージョンのワークベンチで使用した場合、初めに開くパースペクティブはそのワークスペースで最後に使用していたものではなく、本バージョンにおけるデフォルトパースペクティブになります。必要に応じてパースペクティブを開き直してください。また、パースペクティブのカスタマイズ設定を行っていた場合にはその設定はすべて初期化されます。必要に応じて再度カスタマイズ設定を行ってください。

自動ビルドの有効化

本バージョンのワークベンチでは自動ビルドのデフォルト設定がオンに変更になり、リソースが保存されるとインクリメンタルビルドが実行されます。ただしV7までのデフォルト設定(オフ)で使用していたワークスペースを本バージョンのワークベンチで開いた場合には、V7までの設定(オフ)がそのまま引き継がれます。自動ビルドをオンにするには、メニューバーから[ウィンドウ] > [設定]を選択し、[設定]ダイアログボックスの[一般] > [ワークスペース]の[自動的にビルド]をチェックしてください。

エディタの設定

フォント、外観色、マージン、行番号の表示、強調表示など、V7までで行っていたエディタの設定は本バージョンのワークベンチには引き継がれません。必要に応じて再度設定し直してください。

Javaアプレット起動構成

V7までで作成していたJavaアプレット起動構成は本バージョンのワークベンチには引き継がれません。ワークスペースを開いてからJavaアプレット起動構成を再度作成し直してください。なお新しいJavaアプレット起動構成では、HTMLファイルを指定する代わりに[パラメタ]タブの[幅]および[高さ]にアプレットの表示サイズを指定してください。

JDBCドライバ名と接続先URLの履歴

Enterprise Beanウィザードでは、JDBCドライバを使ってデータベースにアクセスした際に、そのJDBCドライバ名と接続先URLを履歴として保持し、次にウィザードを使用したときに同じ値を再利用できるようになっていますが、V7までで保持していた履歴情報は本バージョンのワークベンチには引き継がれません。これらのウィザードを使用する際には新たにJDBCドライバ名と接続先URLを入力し直してください。

B.4.2 プロジェクトの移行に関する注意点

V7までのワークスペースを本バージョンのワークベンチで利用する場合および、V7までのプロジェクトをインポートして利用する場合には、以下の注意点を参照してください。

JUnitのクラスパスの設定

JUnitのJARファイルの参照方法が変更になりました。V7までのプロジェクトでクラスパスに"ECLIPSE_HOME/plugins/org.junit_3.7.0/junit.jar"を設定している場合には、新しいクラスパスに変更する必要があります。従来のjunit.jarのクラスパス設定を削除し、代わりにJUnitライブラリをクラスパスに設定してください。JUnitライブラリはJUnitのJARファイルを参照するために導入されたライブラリです。

なお"B.6 ワークスペースおよびプロジェクトの自動更新"を使用することにより、JUnitのクラスパスを自動的に変更することができます。

EARファイル生成(earbuild.xml)

V7までのプロジェクトでearbuild.xmlを用いたEARファイルの生成機能を利用していた場合、そのままでは本バージョンのワークベンチでEARファイルを生成することができません。

"B.6 ワークスペースおよびプロジェクトの自動更新"を使用することにより、EARファイルの生成機能を自動的に更新することができます。

Javaコンパイラビルドツールの設定

V7までのプロジェクトをインポートする場合、Javaコンパイラビルドツールの設定は引き継がれません。V7までのワークスペースを利用する場合にもこれらの設定は引き継がれません。デフォルト以外の設定にしている場合は再度設定し直してください。

V7までのSOAPアプリケーション

ワークベンチでは、JAX-WSに準拠したWebサービス機能が標準のWebサービス実行環境となっています。V7までのSOAPサービスのRPC方式を使用したアプリケーションについては、使用可能な型などの範囲が一部異なりますが、アプリケーションの実装部分については新しいJava EEのWebサービスにも流用可能です。流用する場合には以下のように作業を行ってください。なお、V7までのWSDLファイルは相互接続性などの観点から流用を推奨しません。その他の流用時の注意点については"Interstage Application Server 移行ガイド"を参照してください。

1. 実装処理の移植
   既存の実装を流用し、データ型の変更や実行環境の違いなどを考慮して移植を行ってください。

2. 実装クラスのWebサービス化
   実装クラスを@WebServiceアノテーションを追記して、Webサービス化します。
   Webサービスの開発については、"第5章 Webサービスのアプリケーションを開発する"を参照していください。

Webアプリケーションプロジェクトの利用

Webアプリケーションプロジェクトを利用する場合は、以下の注意が必要です。

- プロジェクトのビルドパスにContextRoot/WEB-INF/lib フォルダ内のJARファイルを追加している場合には、そのJARファイルをビルドパスから削除してください。JARファイルを削除してもビルドは正常に行えます。ただし、Javaエディタを利用する場合、以下の注意が必要です。
  - JARファイルで定義されているクラスは、問題として強調表示されます。
  - JARファイルで定義されているクラスは、コンテンツアシストが動作しません。
- Interstage Application Server V6.0以前のサーブレットコンテナでは、web.xmlにサーブレットマッピングを定義していなくてもサーブレットを呼び出すことができました。サーブレットを使っている場合は、web.xmlにサーブレットマッピングの定義を追加してください。詳細については、"Interstage Application Server J2EE ユーザーズガイド(旧版互換)"を参照してください。

Tomcat起動構成

Tomcat起動構成は標準では利用できなくなりました。Webアプリケーションをデバッグする場合は、代わりにサーバビューを用いたデバッグを行うことを推奨します。

B.5 バージョン共通の注意点

ここでは、製品の特定のバージョン・レベルに依らない注意点を説明します。

B.5.1 Javaの移行に関する注意点

Javaの移行に関する注意点を説明します。

Javaのバージョン

プロジェクトのJavaのバージョンは以下で指定されています。

- JavaビルドパスのJREシステムライブラリ
- プロジェクトファセット
- Javaコンパイラのコンパイラ準拠レベル

標準のワークベンチでプロジェクトの自動更新を行った場合、基本的には以下のように設定されます。使用するJavaのバージョンが異なる場合には個別に修正する必要があります。

移行前のJavaのバージョン	移行後のJavaのバージョン
6	6
5以前	6

注意

標準のワークベンチの場合、移行元のワークスペースやプロジェクトの設定次第では、上記の変換を行わない場合があります。移行後は必ずJavaのバージョンを確認し、プロジェクトファセットの設定と矛盾がないようにしてください。

例:[Javaビルドパス]の[JREシステムライブラリ]の設定が[ワークスペースのデフォルトJRE]の場合、[JREシステムライブラリ]は移行先のワークスペースの設定に従います。

事前定義ライブラリ

使用可能な事前定義ライブラリを以下に示します。

事前定義ライブラリ名	備考
EARライブラリ	
JBKライブラリ	旧互換ライブラリは未サポート
JREシステムライブラリ	
JSFライブラリ	
JUnit	
Webアプリケーションライブラリ	
サーバランタイム	J2EEライブラリはプロジェクトの自動更新時にサーバランタイムに変更されます。 旧版のサーバランタイムやターゲットランタイムは最新のバージョンに変更されます。
プラグイン依存関係	
ユーザライブラリ	
接続ドライバ定義	
Fujitsu XMLライブラリ	非推奨機能のため新規追加は行えません。

旧バージョンで提供していた上記以外の事前定義ライブラリについては、最新バージョンでは機能を提供していないためプロジェクトの自動更新時に削除されます。その結果、最新バージョンで機能を提供していない事前定義ライブラリを使用している場合には、コンパイルエラーが発生しますので代替手段への実装変更を行う必要があります。

アーカイブファイル

ワークベンチでは、基本的にはアーカイブファイルが無くてもデバッグ可能なため、ビルド時にアーカイブファイルを自動的に作成する処理をデフォルトでは行っていません。

運用環境への配布のためにアーカイブファイルを作成する必要がある場合には、"6.2.8 運用環境への配布"を参考にしてください。

ポイント

プロジェクトの自動更新を行うことで、アーカイブファイルを作成するためのAntスクリプトが作成されます。

そのAntスクリプトをプロジェクトのプロパティの[ビルダ]からAntビルドとして登録することで、ビルド時に自動的にアーカイブファイルを作成することができます。

B.5.2 J2EEアプリケーションの移行に関する注意点

J2EEアプリケーションの移行に関する注意点を説明します。

旧バージョンのInterstage Application Serverで運用するJ2EEアプリケーションを開発する

J2EEアプリケーションのプロジェクトを移行した場合、サーバランタイムはデフォルトでは「Interstage Application Server V11.1 IJServer(J2EE)」になっています。旧バージョンのInterstage Application Serverで運用するJ2EEアプリケーションのプロジェクトの場合には、プロジェクトのプロパティの[ターゲットランタイム]でサーバランタイムの変更を行う必要があります。

注意

旧バージョンのInterstage Application Serverで運用するJ2EEアプリケーションを開発する場合には、インストール時にアプリケーションサーバをインストールしないでください。その代わりとして、開発したいバージョンのInterstage Application Serverのクライアントパッケージをインストールし、デバッグについてはリモート環境のサーバを利用して行う必要があります。

実行環境をIJServer(J2EE)からIJServerクラスタに変更する

旧バージョンのInterstage Application Serverで運用していたJ2EEアプリケーションを、IJServerクラスタで動作させる場合には、プロジェクトのプロパティの[ターゲットランタイム]でサーバランタイムの変更を行う必要があります。

注意

以下のJ2EEアプリケーションについては注意事項があります。

- Enterprise Bean (Container-managed Persistence)

  CMP拡張情報ファイルはIJServer(J2EE)で動作させるためのファイルです。IJServerクラスタで動作させるためには、Interstage Application Serverのマニュアルを参考にファイルをIJServer(クラスタ)用に変更してください。

- Webサービス

  IJServer(J2EE)で運用していたWebサービス(JAX-RPC)は、IJServerクラスタで実行することができません。詳細はInterstage Application Serverのマニュアルを参照してください。

J2EEアプリケーションからJava EEアプリケーションへの移行

J2EEアプリケーションをJava EEアプリケーションに移行するには、各アプリケーションの変更点を参照してアプリケーションを更新してください。

- Webアプリケーション:"2.1.1.1 J2EE1.4からの変更点"
- Enterprise JavaBeans(EJB):"3.1.1.1 J2EE1.4からの変更点"
- Java Persistence API:"4.1.1.1 EJB2.1からの変更点"
- Webサービス:"5.1.1.1 J2EE1.4からの変更点"

EJB2.0からEJB2.1への移行

EJB2.0に準拠した資産をEJB2.1準拠に移行するには、以下の手順に従ってください。

注意

EJB2.1に移行した場合は、Interstage Application ServerではリモートEJBアプリケーションにアクセスすることはできません。

1) プロジェクト新規作成

EJBプロジェクトウィザードで新規にEJBプロジェクトを作成します。このとき、ウィザードの[EJB モジュール バージョン]で[2.1]を選択してください。

2) 旧資産のインポート

インポートウィザードを使用して旧資産をインポートします。
メニューバーから[ファイル] > [インポート]を選択して、インポートウィザードを起動します。[インポートソースの選択]で[一般] > [ファイルシステム]を選択して、旧資産の必要な資産(ソースファイル、deployment descriptorなど)をインポートしてください。deployment descriptorはソースフォルダのMETA-INFフォルダ配下に格納してください。deployment descriptorは、EJBプロジェクト作成時に生成されますが、インポート時に既存資産のdeployment descriptorを上書きしてください。

ポイント

以下のフォルダはインポート対象から外してください。

- .externalToolBuilders
- .settings
- bin
- src

・ distribute

3) deployment descriptorの更新

deployment descriptorをEJB2.0からEJB2.1形式に変更します。変更は、deployment descriptorをXMLエディタで開き、XMLを直接変更する必要があります。

・ deployment descriptorのスキーマ定義をDTDからXML Schemaに変更する

以下のようにXML Schemaを定義します。

－ **EJB2.0**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN" "http://java.sun.com/dtd/
ejb-jar_2_0.dtd">
<ejb-jar>
```

－ **EJB2.1**

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">
</ejb-jar>
```

・ タグ構成の変更

EJB2.0からEJB2.1でタグ構成、値が変更になったタグを修正します。

－ <small-icon>、<large-icon>タグ

アイコンの指定タグ（<small-icon>、<large-icon>）が、<icon>タグ配下に変更になったので、<icon>タグ配下に変更します。

EJB2.0

```
<small-icon>icon/small.gif</small-icon>
<large-icon>icon/large.gif</large-icon>
```

EJB2.1

```
<icon>
    <small-icon>icon/small.gif</small-icon>
    <large-icon>icon/large.gif</large-icon>
</icon>
```

－ <reentrant>タグ

<reentrant>タグの設定値（True or False）が、すべて小文字に変更になったので、小文字（true or false）に修正します。

EJB2.0

```
<prim-key-class>sample.EventPrimaryKey</prim-key-class>
<reentrant>False</reentrant>
<cmp-version>2.x</cmp-version>
```

EJB2.1

```
<prim-key-class>sample.EventPrimaryKey</prim-key-class>
<reentrant>false</reentrant>
<cmp-version>2.x</cmp-version>
```

－ <security-role-ref>、<security-identity>タグ

<security-role-ref>、<security-identity>タグの出現箇所が参照EJBタグ（<ejb-ref>、<ejb-local-ref>）と参照リソースタグ（<resource-ref>、<resource-env-ref>）の間から、参照リソースタグの後ろに変更になったので、タグの挿入位置を変更します。

EJB2.0

```
<ejb-local-ref>
    省略
</ejb-local-ref>
<security-role-ref>
    <role-name>Security1</role-name>
    <role-link>Security1</role-link>
</security-role-ref>
<security-identity>
    <run-as>
        <role-name>Security1</role-name>
    </run-as>
</security-identity>
<resource-ref>
    省略
</resource-ref>
```

EJB2.1

```
<ejb-local-ref>
    省略
</ejb-local-ref>
<resource-ref>
    省略
</resource-ref>
<security-role-ref>
    <role-name>Security1</role-name>
    <role-link>Security1</role-link>
</security-role-ref>
<security-identity>
    <run-as>
        <role-name>Security1</role-name>
    </run-as>
</security-identity>
```

・ Message-driven Bean関連タグの変更

Message-driven Beanの以下のタグが削除されたため、新規に追加された<activation-config>タグでプロパティ名および値として定義しなおします。

― <message-selector>

― <acknowledge-mode>

― <message-driven-destination>

― <destination-type>

― <subscription-durability>

プロパティの指定方法は以下のようになります。なお、<message-driven-destination>タグは値を持ちません。

EJB2.0タグ	EJB2.1タグ	
	プロパティ名	値
message-selector	messageSelector	任意の文字列
acknowledge-mode	acknowledgeMode	以下のどちらかを指定します。 ・ AUTO_ACKNOWLEDGE ・ DUPS_OK_ACKNOWLEDGE
destination-type	destinationType	以下のどちらかを指定します。 ・ javax.jms.Queue

EJB2.0タグ	EJB2.1タグ	
	プロパティ名	値
		・ javax.jms.Topic
subscription-durability	subscriptionDurability	以下のどちらかを指定します。 ・ Durable ・ NonDurable

設定例を以下に示します。

```
<message-driven>
    省略
    <activation-config>
        <activation-config-property>
            <activation-config-property-name> messageSelector </activation-config-property-name>
            <activation-config-property-value> JMSType = 'car' AND color = 'blue'</activation-config-property-value>
        </activation-config-property>
        <activation-config-property>
            <activation-config-property-name> destinationType </activation-config-property-name>
            <activation-config-property-value> javax.jms.Topic </activation-config-property-value>
        </activation-config-property>
        <activation-config-property>
            <activation-config-property-name> subscriptionDurability </activation-config-property-name>
            <activation-config-property-value> NonDurable </activation-config-property-value>
        </activation-config-property>
    </activation-config>
</message-driven>
```

4) Javaソースの修正

EJB2.0からEJB2.1への移行ではソースの修正の必要はありません。

EJB1.0またはEJB1.1からEJB2.0への移行

EJB1.0またはEJB1.1に準拠した資産をEJB2.0準拠に移行するには、以下の手順に従ってください。

1) プロジェクト新規作成と旧資産のインポート

ワークベンチで作成していない資産については、以下の処理を行う必要があります。

a. EJBプロジェクトを新規に作成します。
   [EJB モジュール バージョン]の指定では、[2.0]を指定します。

b. インポートウィザードを使い旧資産をインポートします。

メニューバーから[ファイル] > [インポート]を選択すると、インポートウィザードが選択できますので、ウィザードを選択し、必要な資産のインポートを行ってください。EJB1.0または、EJB1.1からEJB2.0に移行する場合には、deployment descriptorは、インポートしません。

2) deployment descriptorの更新

プロジェクト作成時に生成されたdeployment descriptorをXMLエディタで開き、XMLを直接変更する必要があります。必要な情報を設定して、deployment descriptorを更新します。

3) Javaソースの修正

EJB1.1からEJB2.0への移行ではソースの修正は必要ありません。EJB1.0からの移行の場合は以下の処理について適宜修正してください。

- lookup処理
- ビジネスメソッド呼出し処理
- UserTransaction処理
- Enterprise Bean Environment利用処理

J2EEアプリケーションクライアント1.3からJ2EEアプリケーションクライアント1.4への移行

J2EEアプリケーションクライアント1.3に準拠した資産をJ2EEアプリケーションクライアント1.4準拠に移行するには、以下の手順に従ってください。

1) プロジェクト新規作成

アプリケーションクライアントプロジェクトウィザードで新規にアプリケーションクライアントプロジェクトを作成します。[アプリケーションクライアントモジュールバージョン]の指定では、[1.4]を指定します。

2) 旧資産のインポート

インポートウィザードを使用して旧資産をインポートします。
メニューバーから[ファイル] > [インポート]を選択して、インポートウィザードを起動します。[インポートソースの選択]で[一般] > [ファイルシステム]を選択して、旧資産の必要な資産(ソースファイル、deployment descriptorなど)をインポートしてください。deployment descriptorはソースフォルダのMETA-INFフォルダ配下に格納してください。deployment descriptorは、アプリケーションクライアントプロジェクト作成時に生成されますが、インポート時に既存資産のdeployment descriptorを上書きしてください。

3) deployment descriptorの更新

deployment descriptorをJ2EEアプリケーションクライアント1.3からJ2EEアプリケーションクライアント1.4形式に変更します。変更は、deployment descriptorをXMLエディタで開き、XMLを直接変更する必要があります。

- deployment descriptorのスキーマ定義をDTDからXML Schemaに変更する

  以下のようにXML Schemaを定義します。

  ー J2EEアプリケーションクライアント1.3

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application-client PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application Client 1.3//EN" "http://
java.sun.com/dtd/application-client_1_3.dtd">
<application-client>
```

  ー J2EEアプリケーションクライアント1.4

```
<?xml version="1.0" encoding="UTF-8"?>
<application-client id="Application-client_ID" version="1.4"
xmlns=http://java.sun.com/xml/ns/j2ee
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/application-client_1_4.xsd">
</application-client>
```

- タグ構成の変更

  J2EEアプリケーションクライアント1.3からJ2EEアプリケーションクライアント1.4でタグ構成、値が変更になったタグを修正します。

  ー <small-icon>、<large-icon>タグ

  アイコンの指定タグ(<small-icon>、<large-icon>)が、<icon>タグ配下に変更になったので、<icon>タグ配下に変更します。

  J2EEアプリケーションクライアント1.3

```
<small-icon>icon/small.gif</small-icon>
<large-icon>icon/large.gif</large-icon>
```

  J2EEアプリケーションクライアント1.4

```
<icon>
    <small-icon>icon/small.gif</small-icon>
    <large-icon>icon/large.gif</large-icon>
</icon>
```

Webアプリケーションで JSFを使用する場合

WebアプリケーションでJSFを使用する場合には、資産移行後に以下の手順でライブラリを追加してください。

1. プロジェクトを選択し、コンテキストメニューから[プロパティ] > [Javaのビルドパス] > [ライブラリ]タブ > [ライブラリの追加]をクリックします。
2. [JSFライブラリ]を選択して[次へ]をクリックします。
3. [追加]をクリックします。
4. ライブラリ名に"JSF"を入力し、[追加]をクリックして以下のjarファイルを選択します。
   <製品インストールフォルダ>¥APS¥F3FMisjee¥lib¥jsf-impl.jar
5. [JSF実装]をチェックして[完了]をクリックします。
6. 追加したJSFライブラリをチェックして[完了]をクリックします。

B.5.3 JDK 6/7 に移行する際の注意点

JDK 6およびJDK 7に移行する際の非互換や問題点については、以下のマニュアルを参照してください。

- JDK 1.3からJDK 5.0への移行

  "Javaプラットフォーム移行ガイド(バージョン1.3から5.0へ)"

  このガイドは、次の箇所に格納しています。

  製品DVD:¥PDF¥jmig¥jm_white_paper_r6a-jp.pdf

- JDK 5.0からJDK 6への移行

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

- JDK 6からJDK 7への移行

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

注) 上記のURLはオラクル社が管理・維持しているWebサイトです。予告なく変更されることがあります。

B.5.4 コンポーネントデザイナの資産の移行に関する注意点

ここではコンポーネントデザイナの資産を移行する際の固有の注意点を説明します。

移行可能なプロジェクト

Interstage Apworks V7.0L10、および、それ以前のバージョンのInterstage Apworksで作成したコンポーネントデザイナのプロジェクトを、ワークベンチのインポート機能を使用して移行することができます。
以下のコンポーネントデザイナのプロジェクトを移行できます。

移行可能なプロジェクト
Pure Javaアプリケーション 注1)
JavaBeans 注1)
アプレット 注1)
Enterprise JavaBeans 注1) 注2)
Webアプリケーション 注1) 注3)
Webアプリケーション(Apcoordinator) 注1) 注4)

注1)ホスト画面連携フレームワーク、ホスト画面連携フレームワーク(Apcoordinator)、SOAPサーバアプリケーション(Apcoordinator)を使用したアプリケーションは移行できません。
注2)EJB規約バージョンが1.0のEnterprise JavaBeansは移行できません。
注3)XML表示入力アプリケーション、拡張タグライブラリアプリケーションは移行できません。
注4)携帯端末対応Apcoordinatorは移行できません。

注意

- ワークベンチでサポートしていない以下のプロジェクト種別、および、アプリケーション種別はインポートできません。
  - iアプリ
  - MIDPアプリケーション
  - EJB規約バージョンが1.0 のEnterprise JavaBeansプロジェクト
  - ホスト画面連携フレームワークを使用したアプリケーション
  - SOAPサーバアプリケーション(Apcoordinator)
  - XML表示入力アプリケーション
  - 拡張タグライブラリアプリケーション
  - 携帯端末対応Apcoordinator
  - AIM連携アプリケーション
  - 標準アプリケーション
  - フロントGUI(エミュレータ)
  - フロントGUI(MeFt)
  - フロントGUI(ACM)
  - ActiveXコントロール
  - ワークフローアプリケーション(FlowACT)
  - ActiveXコントロール(FlowACT)
  - CAAプロジェクト(Pure Java)
  - CAAプロジェクト(Enterprise JavaBeans)
  - COBOLアプリケーション
  - CORBAサーバ
  - CORBAクライアント
  - コンポーネントトランザクションアプリケーション
  - 電子フォームアプリケーション(Apcoordinator)
  - 電子フォームアプリケーション
- コンポーネントデザイナで設定しているクラスパスは、すべてインポートされたプロジェクトのビルドパスに追加されます。いずれかのクラスパスが存在しない場合はビルド時にエラーになります。ビルドパスはインポート後に適切に変更してください。
- コンポーネントデザイナで作成した資産格納庫は移行できません。

複数プロジェクトを一括してインポートする

コンポーネントデザイナのサブプロジェクトを利用したインポート用の作業プロジェクトを作成することで、複数のプロジェクトを一括してインポートすることができます。以下に複数のプロジェクトを一括してインポートする手順を示します。

1. コンポーネントデザイナで、他のプロジェクトの親プロジェクトとなるインポート用の作業プロジェクトを作成します。
2. インポートするプロジェクトのプロジェクトフォルダをエクスプローラで開き、プロジェクトファイル「プロジェクト名.prj」をドラッグし、コンポーネントデザイナのプロジェクト表示域に表示されているインポート用の作業プロジェクトのプロジェクトファイル(インポート用の作業プロジェクト名がworkの場合、work.prj)にドロップします。
   [ファイル追加]ダイアログボックスが表示されますので、[OK]をクリックします。
   インポートするプロジェクトがインポート用の作業プロジェクトのサブプロジェクトとして追加されます。

3. インポートするコンポーネントデザイナのすべてのプロジェクトに対し、2.の操作を行います。
4. コンポーネントデザイナを終了します。
5. ワークベンチを起動し、メニューバーから[ファイル] > [インポート]を選択し、[既存のコンポーネントデザイナのプロジェクトをワークスペースへ]を選択します。
6. [インポートするプロジェクト]に1.で作成したインポート用の作業プロジェクトのプロジェクトファイルを指定します。
7. [サブプロジェクトのインポート]をチェックし、[終了]をクリックすると、インポート用の作業プロジェクトとそのサブプロジェクトすべてがインポートされます。
8. インポート完了後、インポートされたインポート用の作業プロジェクトをワークベンチから削除します。

ビルドパスの確認

インポート後のプロジェクトには、そのプロジェクトの種類で標準的に使用される事前定義ライブラリが、ビルドパスに追加されます。標準で追加されない事前定義ライブラリを使用する場合は、ビルドパスにその事前定義ライブラリを追加してください。

コンポーネントデザイナで設定していたクラスパスは、すべてインポートされたプロジェクトのビルドパスに追加されます。事前定義ライブラリで解決されるJARファイルがビルドパスに直接指定されている場合、そのJARファイルはビルドパスに指定する必要はありません。そのようなJARファイルはビルドパスより削除してください。また、インストールフォルダの違いでJARファイルが見つからないものがあります。Javaコンパイルで参照されないJARファイルや事前定義ライブラリで解決されるJARファイルは、ビルドパスより削除してください。個別に追加したライブラリは、適切な場所にライブラリがあるか確認してください。

ファイル配置の確認

インポート時には、プロジェクトの種類、ファイルの種類、ファイルの設定によってファイルの配置場所が決まります。インポート時の標準的な配置では、Interstage Studioで開発するうえでは適切ではない場所に配置されるファイルもあります。インポート後は、各ファイルが正しい配置となっているか確認し、適切な配置となっていないファイルは適切な配置に移動してください。

以下に、インポート機能がファイルを配置する規則を示します。

- Javaソースファイル(*.java)は、ソースフォルダ配下に配置されます。
- プロジェクトのプロパティの[構築]タブで、リソース結合対象としてチェックしているファイルは、ソースフォルダ配下に、元のフォルダ構成のまま配置されます。
- プロジェクトのプロパティの[構築]タブで、リソース結合対象としてチェックしていないファイルは、プロジェクトフォルダ配下に、元のフォルダ構成のまま配置されます。
- SPTファイル(*.spt)は、プロジェクトフォルダに配置されます。
- インポートするプロジェクトがWebアプリケーションおよびWebアプリケーション(Apcoordinator)の場合、HTMLファイルとJSPファイルは、ContextRootフォルダ配下に元のフォルダ構成のまま配置されます。ただし、一部のJSPファイルは、ContextRoot/pagesフォルダに配置されるものがあります。
  それ以外のプロジェクト種別の場合、HTMLファイルはプロジェクトフォルダ配下に元のフォルダ構成のまま配置されます。
- インポートするプロジェクトがWebアプリケーションおよびWebアプリケーション(Apcoordinator)の場合、WEB-INFフォルダにあるファイルは、ContextRoot/WEB-INFフォルダに配置されます。
- インポートするプロジェクトがEnterprise JavaBeansの場合、ejb-jar.xmlはソースフォルダ配下のMETA-INFフォルダに配置されます。
- application-client.xmlは、ソースフォルダ配下のMETA-INFフォルダに配置されます。
- コンポーネントデザイナの構築オプションで[Beanのシリアライズファイル(*.ser)をリソースとして扱う]がチェックされている場合は、プロジェクトフォルダ直下にあるすべての .serファイルがソースフォルダに配置されます。

Webアプリケーションや、Webアプリケーション(Apcoordinator)では、HTMLファイルとJSPファイル、および、WEB-INFフォルダ内にあるファイルだけがContextRootフォルダ配下に配置されます。その他のファイルは、リソース結合対象かどうかによってソースフォルダ配下やプロジェクトフォルダ配下に元のフォルダ構成のまま配置されます。
HTMLやJSPから参照するリソースファイルを確認し、適切な配置となるようにファイルを移動してください。

リソース結合対象となっているファイルは、基本的にソースフォルダ配下に配置されます。このため、テキストファイルなどのビルドの対象としないファイルもソースフォルダ配下に配置される場合があります。ビルドの対象としないファイルは、ソースフォルダ配下から適切な位置に移動してください。

B.5.5 アプレットやJavaフォームの移行に関する注意点

アプレットやJavaフォームの移行に関する注意点を説明します。

フォーム拡張機能が使用されたアプレットやJavaフォームを移行する

フォーム拡張機能が使用されたアプレットやJavaフォームは、そのまま移行することができません。これは、フォーム拡張機能がJDK/JRE 6およびJDK/JRE 7をサポートしていないためです。

フォーム拡張機能が使用されたアプレットやJavaフォームを移行したい場合は、AWTやSwingを使用したアプレットやJavaフォームに変更してください。フォーム拡張機能が使用されたアプレットやJavaフォームであるかを判断するには、以下のクラスが基底クラスであるかを確認します。移行したい場合は、基底クラスを以下のクラスに変更します。

移行前の基底クラス	移行後の基底クラス
com.fujitsu.apworks.compod.ui.CDApplet	java.applet.Applet
com.fujitsu.apworks.compod.ui.CDJApplet	javax.swing.JApplet
com.fujitsu.apworks.compod.ui.CDFrame	java.awt.Frame
com.fujitsu.apworks.compod.ui.CDJFrame	javax.swing.JFrame
com.fujitsu.apworks.compod.ui.CDDialog	java.awt.Dialog
com.fujitsu.apworks.compod.ui.CDJDialog	javax.swing.JDialog
com.fujitsu.apworks.compod.ui.CDPanel	java.awt.Panel
com.fujitsu.apworks.compod.ui.CDJPanel	javax.swing.JPanel

注意

フォーム拡張機能が提供する以下の機能は、移行することができません。機能に依存する処理は、代替の処理に変更していただくか、処理を削除してください。

- フォーカス移動順定義
- ステータスバー

注意

派生フォーム機能を使って作成したアプレットやJavaフォームは、移行することができません。派生元のソースファイルとマージするなどして、継承関係を解除してください。

なお、対象のファイルが派生フォーム機能を使っているかどうかは、以下の手順でソースファイルを確認することでわかります。ご不明な場合は当社技術員にお問い合わせください。

1. ソースファイル内の以下のコメント部分を確認します。

   //@@Form Design Information start
   //@@Inherits From クラス名 8桁の数値

2. 上記のクラス名が表の"移行前の基底クラス"以外のものが対象です。

3. 8桁の数値を16進に変換し、3バイト目を確認します。

   01になっているものが派生フォームです。

   例1://@@Inherits From クラス名 16843267
   16843267 (10進) -> 0x01 01 02 03 (16進)
   -> 派生フォームです

例2://@@Inherits From クラス名 16973315
16973315 (10進) -> 0x01 02 FE 03 (16進)
-> 非派生フォームです

移行する一例として、com.fujitsu.apworks.compod.ui.CDFrameクラスを基底クラスにしたJavaフォームを、AWTのjava.awt.Frameクラスを基底クラスにしたJavaフォームに変更します。この手順を以下に示します。

1. Javaフォームの新規作成

   java.awt.Frameクラスを基底クラスにしたJavaフォームを新規作成します。
   移行前のJavaフォームと同じプロジェクトに、移行後のJavaフォームを作成します。ワークベンチのメニューバーから[ファイル] > [新規] > [その他]を選択します。[新規]ウィザードで[Java] > [GUI] > [フォーム]を選択します。[Javaフォーム情報]ページで情報を入力します。[Javaフォームの新規作成]ダイアログボックスで[JDK]タブの[フレーム]を選択します。[フレーム]ダイアログボックスでは、[Javaフォーム名]フィールドに移行前のJavaフォームと異なる名前を入力して、[基底クラス]コンボボックスに"java.awt.Frame"を入力します。[作成]ボタンをクリックすると、Javaフォームが新規作成されます。

2. Beanのコピー

   移行前のJavaフォームと移行後のJavaフォームを、グラフィカルエディタで開きます。移行前のJavaフォームに貼り付けられているすべてのBeanを、移行後のJavaフォームにコピーします。

3. 各種の定義をする

   以下の定義はコピーすることができないため、移行前のJavaフォームと同じ定義を移行後のJavaフォームに定義します。ただし、あとでアテンション処理のメソッドはコピーするため、アテンション処理定義をする必要はありません。

   - メニュー定義
   - 排他選択グループ定義
   - アテンション定義

4. コンストラクタのコピー

   移行前のJavaフォームには、com.fujitsu.apworks.compod.ui.CDApplicationContextクラスを引数にしたコンストラクタがあります。このクラスはフォーム拡張機能の一部のため、移行後のJavaフォームにコピーできません。このため、コンストラクタの処理だけを、移行前のJavaフォームから移行後のJavaフォームにコピーします。
   コンストラクタの処理は、Javaエディタでコピーします。

5. ソースのコピー

   コンストラクタ以外のソースを、移行前のJavaフォームから移行後のJavaフォームにコピーします。ただし、変更禁止ソースはコピーしないでください。変更禁止ソースとは、ソース先頭の"// Graphical Editor Form"と"//@@Form Design Information start"、"//@@Form Design Information end"で囲まれた部分のソースです。
   ソースは、Javaエディタでコピーします。

6. メソッド名の変更

   移行後のJavaフォームはクラス継承フォームのため、イベント処理とアテンション処理のメソッド名を変更します。
   コピーしたソースの中に、以下の形式のメソッドが存在するかを確認します。これがイベント処理とアテンション処理のメソッドです。

   - public void "Bean名"_"イベントリスナクラス名"_"イベントリスナのメソッド名"("イベントクラス名")
   - public boolean processAttention_"アテンション名"()

   このメソッドを以下の形式のように、メソッド名の最後に$を付けます。

   - public void "Bean名"_"イベントリスナクラス名"_"イベントリスナのメソッド名"$("イベントクラス名")
   - public boolean processAttention_"アテンション名"$()

   Javaフォームを保存すると、Beanとイベント処理のメソッドや、アテンションとアテンション処理のメソッドを関連付けるコードが生成されます。

7. Javaフォーム名の変更

   移行後のJavaフォーム名を移行前のJavaフォームと同じものにします。
   移行前のJavaフォームを削除するか、移行後のJavaフォームを別のプロジェクトにコピーします。移行後のJavaフォームをグラフィカルエディタで開きます。ワークベンチのメニューバーから[ファイル] > [別名保存]を選択して、Javaフォームを別名で保存します。別名にする前のJavaフォームは削除してください。

B.6 ワークスペースおよびプロジェクトの自動更新

ワークスペースおよびプロジェクトの移行に関する注意点のうち、以下のものはワークベンチのコマンドを用いて自動的に更新することができます。

- JUnitのクラスパスの設定
- EARファイル生成(earbuild.xml)

これらの設定を自動的に変換するには、以下の手順に従ってください。

1. "移行の手順"に従ってワークスペースまたはプロジェクトを本バージョンのワークベンチで利用可能にします。
2. メニューバーから[プロジェクト] > [旧バージョンのワークスペース/プロジェクトの更新]を選択します。
3. [旧バージョンのワークスペース/プロジェクトの更新]ダイアログボックスが表示されます。ダイアログボックスの左側に設定の更新が必要なワークスペースおよびプロジェクトが表示されるので、更新するものをチェックして[OK]をクリックします。
4. ワークスペースおよびプロジェクト設定の更新が行われます。更新の状況はコンソールビューに表示されますので、更新が正常に行われたことを確認してください。

注意

- 閉じているプロジェクトがある場合、そのプロジェクトは自動更新の対象になりません。更新したいプロジェクトを開いた状態にしてから自動更新を実施してください。
- 自動更新が行われた後に自動ビルドが実行された場合、クラスパスが正しく設定されているにもかかわらずパッケージエクスプローラでプロジェクトにエラーマークが付くことがあります。このような場合にはプロジェクトを一度クリーンして再度ビルドするとエラーマークが消えます。
- 以下の条件にすべて合致するプロジェクトは、自動更新を行ってもプロジェクトのプロパティのビルダに、"ビルダ(com.fujitsu.apworks.apdesinger.java.packagerbuilder)が欠落"と表示されてしまう場合があります。その場合、もう一度自動更新を行ってください。

  ― V9のワークベンチ、またはV10の互換ワークベンチで開発した資産である。

  ― プロジェクトのビルダに[JARパッケージャビルダ]が設定されている。

  ― プロジェクトのビルドパスに[Interstage J2EE]ライブラリが設定されている。

  なお、"ビルダ(com.fujitsu.apworks.apdesinger.java.packagerbuilder)が欠落"と表示されてしまっていても、ビルドエラーなどの不具合は発生しません。

付録C　JDK 6を用いた開発を行う手順

ここでは、Java EE 6ワークベンチでの開発でJDK 6を用いるための手順を説明します。

JDK 6を用いたJava EE 6ワークベンチの起動

Java EE 6ワークベンチをJDK 6を用いて起動するには、以下の手順に従ってください。

1. コマンドプロンプトを開き、Java EE 6開発機能のインストールフォルダに移動します。

   [Java EE 6開発機能のインストールフォルダ]

   ```
   <製品のインストールフォルダ>¥IDE¥1101_WB36¥eclipse
   ```

2. isstudio.exeを以下のように-vmオプションをつけて実行します。

   ```
   > isstudio.exe -vm (JDK 6のインストールフォルダ)¥jre¥bin
   ```

3. [ワークスペースの選択]画面では、新しいワークスペースフォルダを選択することを推奨します。
   もし、既存のワークスペースを選択した場合には、次の"既存のワークスペースの設定変更"の手順に従ってください。

既存のワークスペースの設定変更

Java EE 6ワークベンチ起動時に既存のワークスペースを選択した場合、以下の手順に従ってJDK 6を使用するための設定変更を行ってください。

1. [インストール済みのJRE]にJDK 6を追加します。
   a. ワークベンチのメニューから[ウィンドウ] > [設定]を選択します。
   b. [設定]ダイアログボックスの左のペインで[Java] > [インストール済みのJRE]を選択します。
   c. 右の[インストール済みのJRE]画面で[追加]をクリックします。
   d. [JREの型]画面では"標準VM"を選択して[次へ]をクリックします。
   e. [JRE定義]画面では[JREホーム]に、JDK 6のインストールフォルダを指定します。
      [JRE名]に"JDK6"と表示され、[JREシステムライブラリ]にJDK 6のライブラリのJARファイル一覧が表示されます。[完了]をクリックして画面を閉じます。
   f. [インストール済みのJRE]の一覧に"JDK6"が追加されます。
      "JDK6"のチェックボックスにチェックを付け、JDK 6をデフォルトのJDKにします。[OK]をクリックしてダイアログボックスを閉じます。
2. 既存のプロジェクトが使用するJDKをJDK 6に変更します。
   a. 変更するプロジェクトをプロジェクトエクスプローラなどで選択し、ポップアップメニューから[プロパティ]を選択します。
   b. [プロパティ]ダイアログボックスの左のペインで[Javaのビルドパス]を選択します。
      右に[Javaのビルドパス]ページが表示されるので[ライブラリ]タブを選択します。
   c. 一覧の中から"JREシステムライブラリ[JavaSE-1.7]"を選択し、[編集]をクリックします。
   d. [JREシステムライブラリ]画面で"実行環境"を"JavaSE-1.6(JDK6)"に変更して[完了]をクリックします。
   e. [OK]をクリックして[プロパティ]ダイアログボックスを閉じます。

JDK 6を用いたJava EE 6実行環境の初期化

JDK 6を利用して開発したアプリケーションをJava EE 6実行環境に配備して動作確認する場合はJava EE 6実行環境をJDK 6用に初期化する必要があります。初期化後は作成していたInterstage Java EE 6 DASサービスや、配備していたアプリケーションはすべて削除されます。必要に応じて初期化後にInterstage Java EE 6 DASサービスの再作成、アプリケーションの再配備を行ってください。

初期化は以下の手順で実施します。

表C.1 初期化の方法

No.	操作手順	参照ドキュメント
1	アプリケーションサーバの資源をバックアップします。 注) 本製品に同梱のInterstage Application ServerでJ2EEおよびJava EE 5の機能を利用している場合、必要に応じて作業してください。	"Interstage Application Server 運用ガイド(基本編)" > "資源のバックアップ/移出"
2	アプリケーションサーバをアンインストールします。	"Interstage Studio インストールガイド" > "インストール作業" > "上書きインストール"
3	APSフォルダ配下の必要なファイルを退避した後に、APSフォルダを削除します。 (例:製品のインストール先を"C:¥Interstage" とした場合は、C:¥Interstage¥APSフォルダ)	ー
4	アプリケーションサーバをインストールします。[Java EE 6機能で使用するJDKの選択]画面ではJDK6を選択します。	"Interstage Studio インストールガイド" > "インストール作業" > "上書きインストール"
5	バックアップした資源を復旧します。 注) 本製品に同梱のInterstage Application ServerでJ2EEおよびJava EE 5の機能を利用している場合、必要に応じて作業してください。	"Interstage Application Server 運用ガイド(基本編)" > "資源のリストア/移入"

ポイント

JDK 6を用いた開発を行った後に、JDK 7を用いた開発を行う場合は、以下の手順で設定を戻してください。

- Java EE 6ワークベンチの起動は通常どおり、アプリ画面(Windows 8およびWindows Server 2012の場合)およびスタートメニュー(その他のWindowsの場合)から行ってください。
- 既存のワークスペースをJDK 6用に設定変更していた場合は、JDK 7用に設定を戻してください。
  - ー [インストール済みJRE]の一覧では、"JDK7"のチェックボックスにチェックを付け、JDK 7をデフォルトのJDKにします。
  - ー 既存のプロジェクトでは、[JREシステムライブラリ]画面で"代替JRE"を"JDK7"に変更します。
  - ー 既存のプロジェクトは[Javaコンパイラ]画面で[コンパイラ準拠レベル]を[1.7]に変更します。
- Java EE 6実行環境をJDK 7用に初期化してください。操作方法は、"初期化の方法"と同様です。アプリケーションサーバをインストールする時に[Java EE 6機能で使用するJDKの選択]画面でJDK7を選択してください。

付録D　J2EE1.4アプリケーションの開発について

ここでは、J2EE1.4アプリケーションの開発方法について説明します。

ポイント

Interstage Studio V11では、Java EE5のアプリケーションの開発を推奨しています。

D.1　Webアプリケーションを開発する

J2EE1.4のWebアプリケーションの開発について、Java EEのWebアプリケーションの開発方法と異なる点を中心に説明します。

D.1.1　Webアプリケーションを作成する環境を準備する

Webアプリケーションを作成する準備については、"2.3.1 Webアプリケーションを作成する環境を準備する"を参照してください。

プロジェクトの作成

J2EE1.4のWebアプリケーションを開発する場合、動的Webプロジェクトウィザードで指定するターゲットランタイムは[Interstage Application Server V11.1 IJServer (J2EE)]を選択します。

注意

動的Webモジュールバージョンが2.2および2.3であるプロジェクトの新規作成はサポート対象外です。

D.1.2　サーブレットを作成する

サーブレットの作成および編集については、"2.3.2 サーブレットを作成する"を参照してください。

D.1.3　HTMLファイルを作成する

HTMLファイルの作成および編集については、"2.3.3 HTMLファイルを作成する"を参照してください。

D.1.4　JSPファイルを作成する

JSPファイルの作成および編集については、"2.3.4 JSPファイルを作成する"を参照してください。

D.1.5　HTML/JSPファイルをグラフィカルに編集する

HTML/JSPファイルのグラフィカルな編集については、"2.3.5 HTML/JSPファイルをグラフィカルに編集する"を参照してください。

D.1.6　JavaScriptを作成する

JavaScriptの作成および編集については、"2.3.6 JavaScriptを作成する"を参照してください。

D.1.7　CSSを作成する

CSSファイルの作成および編集については、"2.3.7 CSSを作成する"を参照してください。

D.1.8　web.xmlを編集する

web.xmlの編集については、"2.3.8 web.xmlを編集する"を参照してください。

D.1.9 Webアプリケーションの動作を確認する

Webアプリケーションの動作の確認については、"D.4.7 アプリケーションの動作確認"を参照してください。

D.1.10 Webアプリケーションを運用環境に配布する

Webアプリケーションの運用環境への配布は、アーカイブを作成する必要があります。詳細は"6.2.8 運用環境への配布"を参照してください。

D.2 Enterprise JavaBeans (EJB)を開発する

J2EE1.4のEJBの開発について、Java EEのEJBの開発方法と異なる点を中心に説明します。

D.2.1 EJBを作成する環境を準備する

EJBを作成する準備については、"3.3.1 EJBを作成する環境を準備する"を参照してください。

プロジェクトの作成

J2EEのEJBを開発する場合、EJBプロジェクトウィザードで指定するターゲットランタイムは[Interstage Application Server V11.1 IJServer (J2EE)]を選択します。

注意

EJBモジュールバージョンが1.1であるプロジェクトの新規作成はサポート対象外です。

D.2.2 Enterprise Beanを作成する

Enterprise Beanの生成

EJB2.1に準拠したEnterprise Beanを追加、生成するには、Enterprise Beanウィザードで作成します。

Enterprise Beanウィザードは、以下のEnterprise Beanを作成することができます。

- Stateless Session Bean
- Stateful Session Bean
- EJB2.x Container-managed persistence Entity Bean
- Bean-managed persistence Entity Bean
- Point-To-Point model Message-driven Bean
- Publish/Subscribe model Message-driven Bean

注意

Enterprise Beanは単一のソースフォルダに作成してください。
deployment descriptorファイルはソースフォルダ配下に格納されるため、1つのプロジェクトの複数のソースフォルダにEnterprise Beanを作成するとdeployment descriptorが複数作成されてしまいます。この状態でJAR形式にアーカイブすると、同じ名前のファイルが存在することが原因でエラーになります。

Enterprise Beanは、[新規]ウィザードから[EJB] > [J2EE] > [Enterprise Bean]を選択し、ウィザードで作成後、生成されたJavaソース、deployment descriptorなどを編集します。
Enterprise Beanウィザードでは、Enterprise Bean生成に関する基本情報やEnterprise Bean種別などを指定します。

Stateless Session Beanの作成

Session Beanは、クライアントとの対話処理を行うEnterprise Beanです。主に業務処理で必要となる処理(ビジネスロジック)を記述します。

Stateless Session Beanは、複数のメソッドにまたがって、トランザクション状態やEnterprise Beanに定義されている変数の値を保持することができません。そのため、1つのメソッドで完結する機能を提供する場合に使用します。また、複数のクライアントが、同じSession Beanのインスタンスを共有することができるため、サーバの負荷を軽減することができます。

Enterprise Beanウィザードでは、以下の情報を指定します。

1. Enterprise Beanの基本情報の指定を行います。
   パッケージ名、クラス名等を入力します。
2. Session Beanオプションの指定を行います。
   トランザクション管理種別を指定します。
3. ejbCreateメソッド定義の指定を行います。
   Enterprise Beanを初期化するためのメソッドを定義します。
   初期状態でパラメタなしのejbCreateメソッドが定義されています。ejbCreateメソッドでスローする例外を追加する処理を行いたい場合だけ、初期値を編集する必要があります。
4. ビジネスメソッド定義の指定を行います。
   ビジネスメソッドを追加します。Stateless Session Beanの場合は、処理が完結する単位でメソッドを作成する必要があります。

Stateful Session Beanの作成

Session Beanは、クライアントとの対話処理を行うEnterprise Beanです。主に業務処理で必要となる処理(ビジネスロジック)を記述します。

Stateful Session Beanは、クライアントと1対1に対応し、クライアントから呼び出される複数のメソッドにまたがって、トランザクション状態やEnterprise Beanに定義されている変数の値を保持することができます。そのため、ある機能を提供するうえで複数の手順(メソッド)が必要な場合に使用します。

Enterprise Beanウィザードでは、以下の情報を指定します。

1. Enterprise Beanの基本情報の指定を行います。
   パッケージ名、クラス名を入力します。
2. Session Beanオプションの指定を行います。
   トランザクション管理種別と、SessionSynchronizationインタフェースを実装するかどうかを指定します。
3. ejbCreateメソッド定義の指定を行います。
   Enterprise Beanを初期化するためのメソッドを定義します。
   初期値として引数なしのejbCreateメソッドが定義されています。必要に応じて、編集、追加してください。
4. ビジネスメソッド定義の指定を行います。
   ビジネスメソッドを追加します。

Bean-managed persistence Entity Beanの作成

Entity Beanは、データベースなどの永続データをJavaのオブジェクトにマッピングし、複数のクライアントからアクセスできるようにしたものです。

Bean-managed persistenceでは、Entity BeanのソースにSQL文などの永続データへのアクセス処理を記述します。データアクセス処理を直接記述するため、データベース種別に応じた、きめ細かいアクセス制御ができます。

Enterprise Beanウィザードでは、JDBCを使ってデータベースにアクセスする処理のひな型を生成することができます。

Enterprise Beanウィザードでは、以下の情報を指定します。

1. Enterprise Beanの基本情報の指定を行います。
   パッケージ名、クラス名を入力します。

2. Entity Beanオプションの指定を行います。
   データベースアクセスに関する以下の情報などを指定します。

   ー リエントラントにする
   Entity Beanが再帰的に呼び出されることを可能とするかどうかを指定します。

   ー データベースアクセス処理を生成する
   Enterprise Beanソース上にデータベースアクセス処理のひな型を生成するかどうかを指定します。

   ー SQL文中の表明を変更可能にする
   SQL文中の表名を環境プロパティ「TableName」の値を使うようにソースを生成します。スキーマやテーブルが運用環境によって変わる可能性がある場合に指定してください。

   ー Entity検索処理の最適化
   通常のEJB規約に準拠したBean-managed persistenceのデータベースアクセス処理のひな型では、検索時にPrimary Keyの取得と永続化フィールドの取得のタイミングが異なるため2度DBに検索しにいきます。このオプションを指定することによりPrimary Key取得時に永続化フィールドの情報も同時に取得し、メモリ上に貯えておくようにソースを生成します。

   ー Recordクラスでデータの取得/設定を行う
   Entity Beanの永続化フィールドのデータを、Recordクラスを使用して一度に取得/設定を行うメソッドを生成します。

   ー データソース名
   永続化する際に使用するデータソース名を指定します。

3. 永続化フィールド定義の指定を行います。
   永続化フィールドを定義します。データベースを参照して永続化フィールドを追加することができます。

4. ejbCreateメソッド定義の指定を行います。
   データベースに永続化データを挿入するためのメソッドを定義します。
   初期値として、永続化フィールドまたはレコードクラスを引数に持つejbCreateメソッドが定義されています。必要に応じて、編集、追加してください。

5. Finderメソッド定義の指定を行います。
   永続化データを検索するためのメソッドを定義します。
   初期値として、Primary Keyクラスによる検索を行うためのejbFindByPrimaryKeyメソッドが定義されています。必要に応じて、追加してください。

6. Homeメソッド定義の指定を行います。
   EJB2.1の場合に使用できる機能です。Entity Beanのインスタンスに依存しない処理(例えば、件数のカウントなど)を行うためのメソッドを定義します。必要に応じて、追加してください。

7. ビジネスメソッド定義の指定を行います。
   ビジネスメソッドを追加します。

Container-managed persistence Entity Beanの作成

Entity Beanは、データベースなどの永続データをJavaのオブジェクトにマッピングし、複数のクライアントからアクセスできるようにしたものです。

Container-managed persistenceでは、Entity Beanのソースには永続データへのアクセス処理を記述せず、データベースとのマッピング処理や検索処理をカスタマイズ情報として定義するものです。永続データへのアクセス処理は、カスタマイズ情報をもとにコンテナが行います。そのため、異なるデータソース(データベース種別など)への移植性(ポータビリティ)に優れています。

Enterprise Beanウィザードでは、Interstage EJBサービスのカスタマイズ情報であるCMP拡張情報ファイルを生成することができます。

Enterprise Beanウィザードでは、以下の情報を指定します。

1. Enterprise Beanの基本情報の指定を行います。
   パッケージ名、クラス名を入力します。

2. Entity Beanオプションの指定を行います。
   データベースアクセスに関する以下の情報などを指定します。

   ー リエントラントにする
   Entity Beanが再帰的に呼び出されることを可能とするか

ー Recordクラスでデータの取得/設定を行う
Entity Beanの永続化フィールドのデータを、Recordクラスを使用して一度に取得/設定を行うメソッドを生成します。

ー データソース名
永続化する際に使用するデータソース名を指定します。

ポイント

データソース名はアプリケーションのターゲットランタイムがJ2EEコンテナかJava EEコンテナかによって書き込まれるファイルが変わります。

- J2EEの場合：FJCMP_XXX.xmlの<fujitsu-cmp2x-mapping-definition>/ <datasource-name>
- Java EEの場合：sun-ejb-jar.xmlの<sun-ejb-jar>/ <enterprise-beans>/ <cmp-resource>/ <jndi-name>

3. 永続化フィールド定義の指定を行います。
永続化フィールドを定義します。データベースを参照して永続化フィールドを追加することができます。
4. ejbCreateメソッド定義の指定を行います。
データベースに永続化データを挿入するためのメソッドを定義します。
5. 初期値として、永続化フィールドまたはレコードクラスを引数に持つejbCreateメソッドが定義されています。必要に応じて、編集、追加してください。
6. Finderメソッド定義の指定を行います。
永続化データを検索するためのメソッドを定義します。
7. 初期値として、Primary Keyクラスによる検索を行うためのfindByPrimaryKeyメソッドが定義されています。必要に応じて、追加してください。
8. Homeメソッド定義の指定を行います。
EJB2.1の場合に使用できる機能です。Entity Beanのインスタンスに依存しない処理(例えば、件数のカウントなど)を行うためのメソッドを定義します。必要に応じて、追加してください。
9. ejbSelectメソッド定義の指定を行います。
EJB2.1の場合に使用できる機能です。Entity Beanのメソッド内で行う検索処理のためのメソッドを定義します。必要に応じて、追加してください。
10. ビジネスメソッド定義の指定を行います。
ビジネスメソッドを追加します。

ポイント

CMP Entity BeanではJ2EEコンテナ向けかJava EEコンテナ向けかによって最終的な成果物に以下の違いがあります。

項目	JavaEEコンテナ	J2EEコンテナ
CMP拡張情報ファイル	sun-cmp-mappings.xml	FJCMP_XXXX.xml
DB定義情報	sun-ejb-jar.xml	FJCMP_XXXX.xml
DBテーブル情報	XXX.dbschema (DB接続時に「DBスキーマを利用する」にチェックを入れていた場合)	なし

Point-To-Point model Message-driven Beanの作成

Message-driven Beanは、非同期通信処理を行うためのEnterprise Beanです。

Point-To-Point model Message-driven Beanでは、送信者から送信された1つのメッセージに対して、ただ1人の受信者が処理を行います。すなわち、メッセージに対して受信者を単独で割り当てる場合に使用します。

Enterprise Beanウィザードでは、以下の情報を指定します。

1. Enterprise Beanの基本情報の指定を行います。
   パッケージ名、クラス名を入力します。

2. Message-driven Beanオプションの指定を行います。
   トランザクション管理種別と、必要に応じてメッセージセレクタを指定します。

Publish/Subscribe model Message-driven Beanの作成

Message-driven Beanは、非同期通信処理を行うためのEnterprise Beanです。

Publish/Subscribe model Message-driven Beanでは、送信者から送信された1つのメッセージに対して、複数の受信者が処理を行います。すなわち、同一のメッセージを複数の受信者に配信する必要がある場合に使用します。

Enterprise Beanウィザードでは、以下の情報を指定します。

1. Enterprise Beanの基本情報の指定を行います。
   パッケージ名、クラス名を入力します。

2. Message-driven Beanオプションの指定を行います。
   トランザクション管理種別、サブスクライバの永続性、必要に応じてメッセージセレクタを指定します。

D.2.3 CMP Entity Beanを開発する

CMP2.0リレーションを定義する

EJB2.1仕様のContainer-managed persistence Entity Beanのリレーションは、CMP2.0 リレーション定義ウィザードで定義します。このウィザードはEntity Beanと、別のEntity Beanの関係を設定するために使用します。このためCMP2.0リレーション 定義ウィザードを使用する前に、同一プロジェクト配下に関係を定義するEJB2.1 Container-managed persistence Entity Beanを作成しておく必要があります。

CMP2.0リレーション定義ウィザードは、[新規]ウィザードから[EJB] > [J2EE] > [CMP2.0リレーション定義]を選択します。以下の設定項目を確認、入力してください。

deployment descriptorの指定

リレーション定義を行うEntity Beanのdeployment descriptorファイルを指定します。

項目	説明
ejb-jar.xmlファイル	ejb-jar.xmlのパスを指定します。

CMP2.0 リレーション情報の指定

Entity Bean間の関係を指定します。

項目	説明
追加	リレーション情報を新規に追加します。 [追加]をクリックすると、"リレーションの追加"画面が表示されます。
削除	一覧で選択しているリレーション情報を削除します。
編集	一覧で選択しているリレーション情報を編集します。 [編集]をクリックすると、"リレーションの編集"画面が表示されます。
Home/Componentインタフェースにリレーション情報を反映する	CMRフィールドのアクセスメソッドをLocalインタフェースに反映したい場合に指定します。 Home/Componentインタフェース生成機能を呼び出すことにより、ウィザードによって定義されたCMRフィールドのアクセスメソッドをLocalインタフェースに反映します。

リレーションの追加/編集

Entity BeanとEntity Beanの関係を指定します。
便宜上、関係を定義するEnterprise Beanの片方をEnterprise Bean A、もう片方をEnterprise Bean Bとしています。

項目	説明
EJBリレーション名	Entity Bean間のリレーション名を指定します。EJBリレーション名は省略可能です。
多重度	Entity Bean間の多重度を指定します。多重度とは、リレーションを構成するEnterprise Beanの対応関係で、1対1、1対多、多対多の関係が存在します。 1対多については、A、Bのどちらでも1側として定義できるため、One:One、One:Many、Many:One、Many:Manyの4種類が指定可能です。
EJBリレーションロール名	リレーションを構成するEntity Beanのリレーションロール名を指定します。EJBリレーションロール名は省略可能です。
Enterprise Bean名	リレーションを構成するEntity BeanのEnterprise Bean名を指定します。
Bean A/Bを参照するフィールド名	相手のEnterprise Beanを参照するCMRフィールド名を指定します。どちらか一方は必ず指定する必要があります。
CMRフィールド型	リレーションの相手となるEnterprise Bean を参照するCMRフィールドの型を指定します。多重度がOne:Many、Many:OneまたはMany:Manyで、相手側が多数の場合にjava.util.Collectionまたはjava.util.Setのどちらかを指定できます。
Bean Aが削除された時に削除、Bean Bが削除された時に削除	リレーション関係にあるEnterprise Beanの片方が削除された場合に、もう片方を削除するかどうかを指定します。
説明	このリレーション定義に関する説明を記述します。説明は省略可能です。
マッピングAタブ	Enterprise Bean Aに指定したEnterprise Beanと対応付けるデータベース情報を指定します。多重度がOne:Oneの場合に、相手のEnterprise BeanのPrimary keyを参照するデータベースのカラムを指定する必要があります。 [マッピングA]タブを選択すると、"マッピングA"画面が表示されます。
マッピングBタブ	Enterprise Bean Aに指定したEnterprise Beanと対応付けるデータベース情報を指定します。多重度がOne:Oneの場合に、相手のEnterprise BeanのPrimary keyを参照するデータベースのカラムを指定する必要があります。 [マッピングB]タブを選択すると、"マッピングB"画面が表示されます。
Joinテーブル	多重度がOne:Many、Many:One、Many:Manyの場合に必要なJoinテーブルの情報を指定します。 [Joinテーブル]タブを選択すると、"Joinテーブル"画面が表示されます。

マッピングA、マッピングBの指定

データベースの表または列をEnterprise Beanのフィールドにマッピングするための情報を指定します。
この画面で指定された情報はCMP拡張情報ファイルに保存されます。

項目	説明
データソース名	対応するEnterprise Beanのフィールドを永続化するデータベースのデータソース名を指定します。
スキーマ名	対応するEnterprise Beanのフィールドを永続化するデータベースのスキーマ名を指定します。
テーブル名	対応するEnterprise Beanのフィールドを永続化するデータベースのテーブル名を指定します。
PK	Primary keyの場合にチェックします。
DBカラム名	永続化フィールドに対応付けるデータベースのカラム名を指定します。
結合先Enterprise Bean名、結合先フィールド名	DBカラム名に対応付ける永続化フィールドのEnterprise Bean名とフィールド名を指定します。多重度がOne:Oneの場合には、相手のEnterprise BeanのPrimary keyの永続化フィールドと対応付けるDBカラムが必要です。
追加	行を新規に追加します。

項目	説明
削除	一覧で選択している行を削除します。
DB参照	データベースの表または列を参照して行を追加します。[DB参照]をクリックすると、ログイン画面が表示されます。

Joinテーブルの指定

多重度がOne:Many、Many:One、Many:Manyの場合に作成されるJoinテーブルの情報を指定します。
この画面で指定された情報は、Enterprise Bean AのCMP拡張情報ファイルに保存されます。

項目	説明
スキーマ名	テーブルとテーブルを関係付けるためのJoinテーブルのスキーマ名を指定します。
Joinテーブル名	テーブルとテーブルを関係付けるためのJoinテーブルのテーブル名を指定します。
PK	JoinテーブルのPrimary key項目をチェックします。多重度のMany側のEnterprise BeanのPrimary key項目がJoinテーブルのPrimary key項目になります。そのため、多重度がMany:Manyの場合には、すべての項目がPrimary keyになります。
DBカラム名	データベースのカラム名を指定します。
結合先Enterprise Bean名、結合先フィールド名	DBカラム名に対応付ける永続化フィールドのEnterprise Bean名とフィールド名を指定します。リレーションを定義する両方のEnterprise BeanのPrimary key項目と対応付けるDBカラムが必要です。
追加	行を新規に追加します。
削除	一覧で選択している行を削除します。
DB参照	データベースの表または列を参照して行を追加します。[DB参照]をクリックすると、ログイン画面が表示されます。

リレーション定義の詳細については、"Interstage Application Server J2EE ユーザーズガイド(旧版互換)"を参照してください。

deployment descriptor(ejb-jar.xml)の編集

deployment descriptor(ejb-jar.xml)の編集は、XMLエディタを用います。編集については、"XMLファイルを編集する"を参照してください。

CMP拡張情報ファイルを作成する

Enterprise Beanクラスのソースを自分で作成した場合など、何らかの理由で、Enterprise Beanに対応するCMP拡張情報ファイルが存在しない場合、CMP拡張情報ファイルを単体で生成することができます。

CMP拡張情報ファイルは、[新規]ウィザードから[EJB] > [J2EE] > [CMP拡張情報]を選択し、ウィザードで作成します。ウィザードの設定は以下を参考にしてください。

項目	説明
ejb-jar.xmlファイル	ejb-jar.xmlのパスを指定します。
Enterprise Bean名	ejb-jar.xmlのパスを指定すると、定義されているContainer-managed persistence Entity Beanの一覧が表示されますので、CMP拡張情報ファイルを生成したいEnterprise Beanを選択します。

注意

Java EEコンテナ向けのアプリケーションではCMP拡張情報ウィザードは動作しません。

ポイント

CMP拡張情報ファイルは、指定されたejb-jar.xmlが存在するソースフォルダのルートに生成されます。

CMP拡張情報ファイルの編集

CMP拡張情報ファイルを編集するには、CMP拡張情報エディタを利用します。CMP拡張情報エディタは、CMF(Container-managed Field)とデータベースとの対応付けや、finderメソッドの検索条件を編集するためのエディタです。

CMP拡張情報ファイルの編集は、[プロジェクトエクスプローラ]ビューなどから、FJCMP_(Enterprise Beanクラス名).xmlを選択し、コンテキストメニューで[アプリケーションから開く] > [CMP拡張情報エディタ]を選択します。一度、編集するとエディタが対応付けられるため、ダブルクリックやポップアップメニューの[開く]でもCMP拡張情報エディタを起動することができます。

ポイント

CMP拡張情報エディタでは、XMLをそのまま表示する機能を提供していません。XMLソースを直接編集する場合は、XMLエディタを利用してください。

D.2.4 EJBテストクライアントを作成する

作成済みのEnterprise Beanから動作確認を行うためのクライアント(以降では、テストクライアントと呼びます)を生成し、各ビジネスメソッドの動作確認を行います。テストクライアントの生成方法およびテストの方法について説明します。ここでは、EJBテストクライアントをWebアプリケーションとして作成する場合を説明します。

ポイント

EJBにリモート接続するEJBテストクライアントを作成する場合は、Javaアプリケーションプロジェクトを作成して、EJBテストクライアントウィザードの[クライアント種別]で"Web"以外を指定してください。

LocalHome/Localインタフェースを含まないEJBのEJBテストクライアントでは、[クライアント種別]で"Web"を選択できません。

プロジェクトの作成

[新規]ウィザードから[Web] > [動的Webプロジェクト]を選択し、テストクライアント用のWebアプリケーションプロジェクトを作成します。

作成したWebアプリケーションプロジェクトを選択してコンテキストメニューの[プロパティ] > [Javaのビルドパス] > [プロジェクト]タブを選択し、テスト対象となるEnterprise Beanのプロジェクトを追加します。

EJBテストクライアントソースの作成

EJBテストクライアントソースは、[新規]ウィザードから[EJB] > [J2EE] > [EJBテストクライアント]を選択し、ウィザードで作成します。

ウィザードでの設定は、以下を参考にしてください。ここでは、Webアプリケーションとして、EJBテストクライアントを作成するため、[クライアント種別]では、"Web"を選択します。

- ソースフォルダ
  テストクライアントのソースを格納するフォルダを指定します。
- パッケージ
  テストクライアントのパッケージ名を指定します。生成するすべてのクラスで使用されます。
- 名前
  テストクライアントのメインクラス名を指定します。
- EJB JARファイル/EJBプロジェクト
  動作確認を行うEnterprise Beanが含まれているEJB JARファイルのパスまたはプロジェクトを指定します。動作確認を行うEnterprise Beanは、Home/RemoteインタフェースまたはLocalHome/Localインタフェースを持つ必要があります。Message-driven Beanの場合は、インタフェースを持つ必要はありません。

・ EJBテストクライアント詳細情報
以下を参照してください。

項目		説明
Enterprise Beanテストクライアント生成リスト		EJB JARファイル内に含まれているEnterprise Beanの一覧が表示されます。[クライアント種別]の指定によって一覧に表示されるEnterprise Beanが切り替わります。 個々のEnterprise Beanについてテストクライアントを生成するかどうかはリストの左端にあるチェックボックスで指定します。
	Enterprise Bean名	EJB JARファイル内のdeployment descriptorに記述されているEnterprise Bean名が表示されます。 [クライアント種別]の指定条件によって表示されるEnterprise Bean一覧が以下のように変わります。 ・ [EJB],[J2EE1.3]を指定した場合 ー Home/Remoteインタフェースを持つEnterprise Bean(Session/Entity Beanの場合) ー Message-driven Bean ・ [Web]を指定した場合 ー LocalHome/Localインタフェースを持つEnterprise Bean(Session/Entity Beanの場合) ー Message-driven Bean
	クラス名	Enterprise Beanに対応するテストクライアントのクラス名を指定します。 Message-driven Beanの場合はEnterprise Bean単位にソースを生成しないため、指定する必要はありません。
	lookup ID	Session BeanやEntity Beanの場合は、Homeインタフェースを取得するためのlookup識別子を指定します。 Message-driven Beanの場合はDestinationを取得するためのlookup識別子を指定します。
クライアント種別		クライアント種別を指定します。
	EJB	deployment descriptorは作成しません。そのため、単純にEnterprise Beanを呼び出すだけで、クライアントからのトランザクション制御などは行えません。
	J2EE1.3	deployment descriptorは作成しません。そのため、単純にEnterprise Beanを呼び出すだけで、クライアントからのトランザクション制御などは行えません。
	Web	Enterprise Beanを呼び出して結果を表示するServletクラスを生成します。 [ソースフォルダ]に指定したプロジェクトがWebアプリケーションの場合に選択可能となります。

ポイント

テストクライアントを開発するためには、テストするEnterprise Beanに含まれる以下のクラスが必要です。

・ [クライアント種別]が"EJB"または"J2EE1.3"の場合

ー Homeインタフェース、Remoteインタフェースクラス(必須)

・ [クライアント種別]が"Web"の場合

ー Localインタフェース、LocalHomeインタフェース(必須)

- 共通条件

  ー Enterprise Beanの呼出しでパラメタや復帰値として受け渡す、ユーザ定義クラス(これらのクラスを使っている場合)

注意

EJB2.1アプリケーションはリモートでアクセスできないため、EJBテストクライアントウィザードの[クライアント種別]で[Web]を指定して、作成されたWebアプリケーションからLocalインタフェース経由でEJB2.1アプリケーションの動作確認をしてください。

Stateless Session BeanをWebサービス化したEnterprise Beanのテストクライアントの生成はできません。

クライアント種別でJ2EE1.3を選択した場合、EJBテストクライアントのプロジェクトにJ2EE1.3形式のdeployment descriptorが生成されますが、このプロジェクトをエンタープライズアプリケーションプロジェクトに追加することはできません。

生成ファイルの編集

必要に応じて、ウィザードによって生成されたファイルを編集してください。

- doGetメソッドに展開されたビジネスメソッドの呼び出し順序は、解析順序となっているため、正しい順序に並び替えてください。
- メソッドパラメタに設定する値を確認してください。
- 配列の要素の値を取得したい場合は、出力処理を追加してください。

実行

テストするEJBアプリケーションと作成したWebアプリケーションをIJServerの種別が[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]のJ2EE実行環境に配備し、起動します。Webブラウザよりアプリケーションの動作確認をしてください。

D.2.5 EJBクライアントを作成する

EJBクライアントアプリケーションを作成するための準備、EJBクライアントアプリケーションの生成支援機能について説明します。

Session BeanまたはEntity Beanクライアントアプリケーションを作成するための準備

Session BeanやEntity Beanを呼び出すアプリケーションを開発するためには、呼び出すEnterprise Beanの以下のクラスが必要です。

- Homeインタフェース、RemoteインタフェースクラスまたはLocalHomeインタフェース、Localインタフェースクラス
- Enterprise Beanの呼出しでパラメタや復帰値として受け渡す、ユーザ定義クラス(これらのクラスを使っている場合)

Enterprise Beanを呼び出すアプリケーションを作成するためには、上記に加え、クライアント配布物をクラスパスに設定する必要があります。クライアント配布物の詳細は、"クライアント配布物"を参照してください。

ポイント

EJBアプリケーションのテストを実施する場合は、以下の手順でサーバランタイムにInterstage Application Server V11.1 IJServer(J2EE)を設定する必要があります。

- プロジェクトを選択し、コンテキストメニューから[プロパティ] > [Javaのビルドパス] > [ライブラリ]タブ > [ライブラリの追加]をクリックし、[サーバランタイム]で[Interstage Application Server V11.1 IJServer(J2EE)]を指定します。

クライアント種別でJ2EE1.3を選択した場合、EJBテストクライアントのプロジェクトにJ2EE1.3形式のdeployment descriptorが生成されますが、このプロジェクトをエンタープライズアプリケーションプロジェクトに追加することはできません。

D.2.6 EJBの動作を確認する

EJBの動作の確認については、"D.4.7 アプリケーションの動作確認"を参照してください。

注意

EJB2.1のアプリケーションは、以下のJ2EE実行環境に配備してください。それ以外のJ2EE実行環境に配備すると配備に失敗します。

- 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

D.2.7 EJBを運用環境に配布する

EJBの運用環境への配布は、アーカイブを作成する必要があります。詳細は"6.2.8 運用環境への配布"を参照してください。

D.2.8 Statless Session BeanをWebサービス化する

Stateless Session BeanをWebサービス化するための手順について説明します。

(1) Stateless Session Beanの作成

プロジェクトの作成

プロジェクトの作成に関しては"3.3.1 EJBを作成する環境を準備する"を参照してください。

このとき、[EJBモジュールバージョン]は[2.1]を選択します。

Enterprise Beanの作成

[新規Enterprise Bean]ウィザードでEnterprise Beanを作成します。

1. Enterprise Bean種別
   [Enterprise Bean種別]では、[Stateless Session Bean]を選択します。
2. インタフェース生成オプション
   Stateless Session Beanを作成する場合は、[LocalHome/Localインタフェースを生成する]を選択してください。Webサービスからだけ呼び出されるStateless Session Beanの場合はLocalHome/Localインタフェースは不要ですが、ここでは、[LocalHome/Localインタフェースを生成する]を選択してください。

   注意

   Interstage Application Serverで運用するEJB2.1アプリケーションは、Home/Remoteインタフェース経由でのリモートアクセスをサポートしていません。
3. ビジネスメソッド定義

   Webサービスで公開するメソッドを定義します。ウィザード終了後に生成されたEnterprise Beanクラスに定義したメソッドを実装します。

サービスエンドポイントインタフェースの作成

生成したEnterprise Beanクラスからサービスエンドポイントインタフェースを作成します。

サービスポイントインタフェースでは以下の条件を満たすようにしてください。

- java.rmi.Remoteインタフェースを継承する
- メソッドはjava.rmi.RemoteExceptionをthrowする
- メソッドの引数と復帰値に指定できるJavaデータ型には制限があります。詳細は、"Interstage Application Server J2EE ユーザーズガイド(旧版互換)"を参照ください。
- メソッドのオーバーロードをしない
- メソッドの引数と復帰値としてEJBObjectやEJBLocalObjectを含まない
- メソッドの引数と復帰値に使用する配列型の要素や、構造体型・Bean型のメンバに以下は定義しない
  - Localインタフェース
  - Remoteインタフェース

- LocalHomeインタフェース
- Homeインタフェース
- Timersインタフェース
- Timersハンドルインタフェース
- Collection型のCMPのfinderメソッドの復帰値

ポイント

インタフェースの生成は[インタフェースの抽出]機能を利用して生成できます。

1. プロジェクトエクスプローラで生成したEnterprise Beanクラスを選択します。
2. コンテキストメニューから[リファクタリング] > [インタフェースの抽出]を選択します。
3. [インタフェースの抽出]ダイアログボックスの[インタフェース内で宣言されるメンバ]で公開するメソッドを選択します。
4. 生成されたインタフェースをサービスエンドポイントインタフェースの条件を満たすように修正します。

(2) Stateless Session BeanのWebサービス化

Stateless Session BeanをWebサービス化するには、以下の対応が必要となります。

- deployment descriptor(ejb-jar.xml)へのサービスエンドポイントインタフェース情報の定義
- Webサービスの公開に必要となる以下のWebサービス関連定義ファイルの作成
  - WSDLファイル
  - <WSDLファイル名>_mapping.xml
  - webservices.xml

Webサービスウィザードを利用することで、サービスエンドポイントインタフェースから、上記のファイルの更新、生成を自動で行うことができます。

WebサービスウィザードによるStateless Session BeanのWebサービス化

WebサービスウィザードでStateless Session BeanをWebサービス化します。ウィザードの詳細については"Webサービスに必要なファイルの生成"を参照してください。

1. ソースフォルダ
   [ソースフォルダ]には、Webサービス化したいEnterprise JavaBeansプロジェクトのソースフォルダを指定します。Enterprise JavaBeansバージョンが"2.1"でないプロジェクトは指定できません。
2. サービスエンドポイントインタフェース名
   作成したサービスエンドポイントインタフェース名を指定します。

ポイント

Stateless Session BeanのWebサービス化では、Enterprise Beanクラスが実装クラスとなるため、実装クラス名を設定する必要はありません。

注意

以下のEnterprise JavaBeansをWebサービス化しようとした場合は、ウィザードでエラーとなります。

- 同一のサービスエンドポイントインタフェースを複数のEnterprise Beanクラスで実装している場合
- 同じEnterprise Beanクラスを別のEJB名で登録してある場合

(3) EARファイルにパッケージング

WebサービスStateless Session Beanを配備する場合は、モジュールは必ずEARファイルにパッケージングします。

(4) 配備

作成したEARファイルをInterstage Application ServerのJ2EE実行環境に配備します。

注意

WebサービスStateless Session Beanを含むEARファイルは、以下のJ2EE実行環境に配備してください。それ以外のJ2EE実行環境に配備すると失敗します。

- 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

ポイント

WebサービスStateless Session BeanをIJServerクラスタに配備すると、自動的にEJB SOAPルータと呼ばれるWebアプリケーションが組み込まれます。

WebサービスではHTTPプロトコルでリクエストが送受信されるため、Webサービス化されたStateless Session Beanを運用する場合にはSOAP(HTTP)からのリクエストを送受信するためのServletが必要になります。このServletのことをEJB SOAPルータと呼びます。

EJB SOAPルータはWebサービス化されたStateless Session Beanを含むEJB JARファイルに対して1つずつ作成されます。

D.3 Webサービスアプリケーションを開発する

J2EE1.4のWebサービスの開発について、Java EEのEJBの開発方法と異なる点を中心に説明します。

D.3.1 Webサービスアプリケーションを作成する環境を準備する

ここでは、WebサービスをWebアプリケーションで提供する場合について説明します。
EJBアプリケーションでWebサービスを提供する場合は、"D.2.8 Statless Session BeanをWebサービス化する"を参照してください。

プロジェクトの作成

プロジェクトの作成については、"2.3.1 Webアプリケーションを作成する環境を準備する"を参照してください。
動的Webプロジェクトウィザードで指定するターゲットランタイムは[Interstage Application Server V11.1 IJServer (J2EE)]を選択します。

ポイント

Webサービスアプリケーションを作成する場合は、必要なライブラリを以下の手順で追加する必要があります。

1. プロジェクトを選択し、コンテキストメニューから[プロパティ] > [Javaのビルドパス] > [ライブラリ]タブ > [外部JARの追加]で以下に存在するJARファイルを追加します。
   <製品インストールフォルダ>¥APS¥J2EE¥lib¥isws-saaj-api.jar
2. 続けて[順序およびエクスポート]タブで、1.で追加したJARファイルを[JREシステムライブラリ]よりも上に移動して[OK]ボタンを押してください。

D.3.2 Webサービスを作成する

サービスエンドポイントインタフェースを作成後、WebサービスウィザードでWebサービスアプリケーションに必要なファイルを作成します。

サービスエンドポイントインタフェースの作成

Webサービスとして公開するサービスのインタフェースをJavaで記述します。

[新規]ウィザードから[インタフェース]を選択し、サービスエンドポイントインタフェースを作成します。

サービスエンドポイントインタフェースは、以下を満たす必要があります。

- java.rmi.Remoteを継承する
- 定義するメソッドはjava.rmi.RemoteExceptionをthrows句に宣言する

注意

メソッドを定義する場合には、使用可能なデータ型の範囲で行うようにしてください。使用可能なデータ型の詳細については、"Interstage Application Server J2EE ユーザーズガイド(旧版互換)"を参照してください。

Webサービスに必要なファイルの生成

サービスエンドポイントインタフェースから、Webサービスアプリケーションに必要なファイルを生成します。[新規]ウィザードから[Webサービス] > [IJServer] > [Webサービス(JAX-RPC)]を選択し、ウィザードで生成します。ウィザードでの設定は、以下を参考にしてください。

[サービスエンドポイント関連情報]ページ

項目		説明
ソースフォルダ		実装テンプレートクラスの生成先フォルダを指定します。 WSDLファイルやdeployment descriptorなどは、指定したソースフォルダが属するプロジェクトのContextRoot配下に生成されます。 WebアプリケーションプロジェクトおよびEnterprise JavaBeansバージョンが"2.1"のEnterprise JavaBeansプロジェクト以外のプロジェクト内のソースフォルダは指定できません。
サービスエンドポイントインタフェース名		Webサービスで公開するサービスエンドポイントインタフェース名を完全修飾名で指定します。 指定するサービスエンドポイントインタフェースは、以下の規則を満たしている必要があります。 ・ java.rmi.Remoteを継承していること ・ メソッドのthrows句にjava.rmi.RemoteExceptionが設定されていること ・ メソッドのオーバーロードがされていないこと
実装テンプレートクラス		実装テンプレートクラスの情報を指定します。 [ソースフォルダ]に指定したプロジェクトがEnterprise JavaBeansプロジェクトの場合は指定できません。
	パッケージ	作成する実装テンプレートのパッケージを指定します。 デフォルトでは、サービスエンドポイントインタフェースのパッケージ名が表示されます。サービスエンドポイントインタフェースとパッケージを同じにしたくない場合は変更してください。
	実装テンプレートクラス名	作成する実装テンプレートクラス名を指定します。 デフォルトでは、サービスエンドポイントインタフェース名＋"SOAPBindingImpl"が表示されます。
	上書き生成しない	既存の実装テンプレートクラスを上書きしたくない場合に、選択します。

[WSDLファイル関連情報]ページ

項目	説明
WSDLファイル名	生成するWSDLファイルがContextRoot配下の相対パスで表示されます。

項目	説明
style属性/use属性	WSDLに記述されるstyle属性とuse属性を指定します。以下から選択可能です。 ・ DOCUMENT/LITERAL ・ RPC/ENCODED ・ RPC/LITERAL WS-I Basic Profile 1.0 に準拠した方式とする場合には、DOCUMENT/LITERALまたはRPC/LITERALを指定します。 DOCUMENT/LITERALの方がプログラム通信に適しており、特に理由が無い場合は、より広く使用されているDOCUMENT/LITERALを選択することをお勧めします。 RPC/ENCODEDは、WS-I Basic Profile 1.0 が策定される前に広く利用されていた方式です。以前のSOAPサービスを利用したシステムとの接続など、RPC/ENCODEDでの接続が必要である場合などに選択します。
Location情報	Webサービスにアクセスするためのエンドポイントを指定します。 この値は運用環境に依存する値です。ローカル環境でクライアントと組み合わせて動作確認する場合のために、デフォルトの値を設定しています。
添付ファイル	Webサービスで添付ファイルを扱うアプリケーションを開発する際のWSDLの生成方法を指定します。以下から選択可能です。 ・ WS-I Attachments Profile 1.0準拠のWSDLファイル生成 ・ 固有データ型を使用したWSDLファイル生成 添付ファイルを利用して、WS-I Attachments Profile 1.0 準拠した方式のWSDLファイルを生成する場合は、サービスエンドポイントインタフェースでjavax.activation.DataHandlerクラスを利用して、WS-I Attachments Profile 1.0準拠のWSDLファイル生成を指定します。 WS-I Attachments Profile 1.0準拠のWSDLファイル生成を指定した場合に添付ファイルで使用する以下のクラスを使うと生成エラーとなります。 ・ java.awt.Image ・ javax.mail.internet.MimeMultipart ・ javax.xml.transform.Source javax.activation.DataHandlerクラス以外のクラスを使用する場合は、バイナリからJavaオブジェクトへの変換が行われ、オリジナルのファイルデータが完全に保持されないため、javax.activation.DataHandlerクラスを利用することを推奨します。

注意

Webサービスウィザードを使用するには、Interstage Application Server機能またはInterstage Application Serverクライアントパッケージをインストールしておく必要があります。
Webサービス(JAX-RPC)ウィザードでは、クラスファイル(.class)を解析し情報取得を行うため、ウィザードを起動する前にサービスエンドポイントインタフェースのコンパイルが行われている必要があります。

ポイント

- ワークベンチの初期状態では、ファイルの保存時に自動的にビルドが行われるようにオプションが設定されています。その場合、サービスエンドポイントインタフェースの保存が行われ、コンパイルエラーが発生していなければ問題ありません。
- Webサービスウィザードを起動する際に、プロジェクトエクスプローラでサービスエンドポイントインタフェースを選択すると、ウィザードのサービスエンドポイントインタフェース名など入力項目が初期値として設定されています。

以下にWebサービスウィザードで生成されるファイルを示します。

生成ファイル	ファイル名	内容
Webサービスエンドポイント	<サービスエンドポイントインタフェース名>SOAPBindingImpl.java	サービスエンドポイントインタフェースの実装クラスです。このクラスにWebサービスの実装を記述します。 ウィザードで指定したパッケージ、名前でソースフォルダ配下に生成されます。
WSDLファイル	<サービスエンドポイントインタフェース名>.wsdl	Webサービスのインタフェース定義ファイルです。 <コンテンツフォルダ>/WEB-INF/wsdl配下に生成されます。
deployment descriptor	webservices.xml	Webサービスに関する配備情報が記述されているファイルです。 <コンテンツフォルダ>/WEB-INF配下に生成されます。(既存のファイルがある場合には情報が追記されます。)
	web.xml	WARファイルの配備情報が記述されているファイルです。 <コンテンツフォルダ>/WEB-INF配下に生成されます。(既存のファイルがある場合には情報が追記されます。)
その他のファイル	<WSDLファイル名>_mapping.xml	Webサービスの実行に必要な自動生成ファイルです。 <コンテンツフォルダ>/WEB-INF配下に生成されます。

ポイント

ファイル生成時のJava型からWSDLのXML型への交換規則については、"Interstage Application Server J2EE ユーザーズガイド(旧版互換)"を参照してください。

WSDLファイルの編集

WSDLファイルの編集は、WSDLエディタを利用します。WSDLエディタについては、"WSDLエディタ"を参照してください。

WSDLの妥当性を検証する

WSDLの妥当性を検証するために、以下のバリデータが用意されています。

- WSDLバリデータ

これをビルド時に実行する場合は、[検証]ビルダを選択した上で、[検証]プロパティでバリデータを実行するように設定してください。通常は設定されています。

上記で設定されているバリデータについては、ビルド時以外にもリソースを選択し、コンテキストメニューから[検証]を選択することで実行することができます。

deployment descriptorの編集

以下のdeployment descriptorの編集には、XMLエディタを利用します。XMLエディタについては、"XMLファイルを編集する"を参照してください。

- webservices.xml
- web.xml(WebサービスをWebアプリケーションとして提供する場合)
- ejb-jar.xml(WebサービスをEJBアプリケーションとして提供する場合)

Webサービスアプリケーションの実装

Webサービスアプリケーションを実装します。ウィザードで生成されたWebサービスエンドポイントにWebサービスの実装コードを記述します。

その他の生成されたファイルは、通常は編集する必要はありません。編集する場合は、ファイルの内容を理解したうえで修正を行ってください。各ファイルの詳細は、"Interstage Application Server J2EE ユーザーズガイド(旧版互換)"を参照してください。

ポイント

サービスエンドポイントインタフェースでHolderクラスを使用している場合、そのパラメタはinout型として扱われます。パラメタをout型にしたい場合には、WSDLファイルを編集する必要があります。

D.3.3 Stateless Session BeanをWebサービス化する

"D.2.8 Statless Session BeanをWebサービス化する"を参照してください

D.3.4 WSDLからサービスエンドポイントインタフェースを作成する

すでにインタフェースが決まっており、WSDLファイルが存在する場合には、Interstage Application Serverから提供されているコマンド(iswsgen)を使用し、Webサービスアプリケーションを作成することができます。

ただし、そのコマンドではdeployment descriptorは自動生成されないため、deployment descriptorはユーザが作成する必要があります。

コマンドやコマンドを使っての開発方法の詳細については、"Interstage Application Server J2EE ユーザーズガイド(旧版互換)"または"Interstage Application Server 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。

D.3.5 Webサービスクライアントを作成する

Webサービスクライアントアプリケーションを作成する手順について説明します。

ポイント

Webサービスクライアントを作成する場合は、必要なライブラリを以下の手順で追加する必要があります。

1. プロジェクトを選択し、コンテキストメニューから[プロパティ] > [Javaのビルドパス] > [ライブラリ]タブ > [ライブラリの追加]をクリックし、[サーバランタイム]で[Interstage Application Server V11.1 IJServer(J2EE)]を指定して[完了]ボタンを押します。
2. 続けて[外部JARの追加]で以下に存在するJARファイルを追加します。
   <製品インストールフォルダ>¥APS¥J2EE¥lib¥isws-saaj-api.jar
3. 続けて[順序およびエクスポート]タブで、2.で追加したJARファイルを[JREシステムライブラリ]よりも上に移動して[OK]ボタンを押してください。

Webサービスのインタフェース情報(WSDLファイル)の取得

Webサービスクライアントを作成するために、WSDLファイルを取得します。

Interstage Application ServerのWebサービスの場合、Interstage管理コンソールの左側のツリーから、[Webサービス] > [Webサービスクラス名]をクリックします。[Webサービス]ページの[一般]タブを選択して、[Webサービス]ページの[WSDL]により、WSDLファイルのURLが取得できます。

Webサービスの開発資産がある場合には、開発資産のWSDLを利用することもできます。

ポイント

WSDLから生成されるファイルには、WSDLファイルに記述されているlocation情報(接続先のWebサービスのURL)が出力されます。運用環境を変更する場合などのため、アプリケーションを再構築することなく、環境設定で接続先を変更することは可能ですが、利用するWebサービスのlocation情報が記述されているWSDLファイルを使うと手間がかかりません。

Interstage管理コンソールより取得できる公開用WSDLファイルは、取得時に配備先の環境に合わせてWSDLファイルのlocation情報が更新されており、これを使用すると接続先についてユーザが意識する必要はありません。

ただし、Webサービスアプリケーションの開発も同時に行うような場合には、デバッグ環境のlocation情報をWebサービスウィザードで指定しておくことで、配備などの作業を行わずに、開発資産のWSDLファイルをそのまま使うことができ、Webサービスアプリケーションと Webサービスクライアントの開発を平行して行うことができます。

WebサービスクライアントがJ2EEアプリケーションで、サービスインタフェースをJNDIのlookupを利用して取得する場合は、WSDLファイルに記述されているlocation情報を意識する必要はありません。

スタブの生成

Webサービスのインタフェースを記述したWSDLファイルから、Webサービスにアクセスするために必要なJavaクラスを生成することができます。[新規]ウィザードから[Webサービス] > [IJServer] > [Webサービスクライアント(JAX-RPC)]を選択し、ウィザードで生成します。ウィザードでの設定は、以下を参考にしてください。

- ソースフォルダ
  Javaソースファイルの生成先フォルダを指定します。
- WSDLファイル
  アクセスしたいWebサービスのインタフェース情報が定義されたWSDLファイルを指定します。
- ユーザ定義型クラスを生成する
  サービスエンドポイントインタフェースで利用しているユーザ定義型クラスを生成するかどうかを指定します。

注意

Webサービスクライアント(JAX-RPC)ウィザードを使用するには、Interstage Application Server機能またはInterstage Application Server クライアントパッケージをインストールしておく必要があります。

以下にWebサービスクライアント(JAX-RPC)ウィザードで生成されるファイルを示します。

生成ファイル	ファイル名	内容
サービスエンドポイントインタフェース	<WSDLのportType名>.java	利用するWebサービスエンドポイントのインタフェース定義をJavaのインタフェースで記述したものです。
サービスインタフェース	<WSDLのservice名>.java	Webサービスに含まれるWebサービスエンドポイントのスタブを取得するためのインタフェースです。
ユーザ定義型クラス	xxxxx.java	ユーザ固有の型をWebサービスで利用している場合に生成されます。
Holderクラス	xxxxxHolder.java	パラメタがoutやinout型の場合で、ユーザ固有の型や配列など標準のHolderクラスがない場合に生成されます。
その他のクラス	_isws_+XXXXX+.java	Webサービスクライアントの実行に必要なクラスです。アプリケーションでこれらのクラスを意識して使用する必要はありません。

ポイント

ファイル生成時のWSDLのXML型からJava型の変換規則については、"Interstage Application Server J2EE ユーザーズガイド(旧版互換)"を参照してください。

注意

Webサービスアプリケーションも作成している場合、Webサービスアプリケーション時に作成したサービスエンドポイントインタフェースと、Webサービスクライアント時にWSDLファイルから生成したサービスエンドポイントインタフェースは同一であるとは限りません。そのため、Webサービスクライアントの開発ではWSDLファイルから生成したものを使用してください。

Webサービスクライアントの開発

生成されたスタブなどを利用してWebサービスにアクセスするクライアントアプリケーションを作成します。

Webサービスにアクセスするには、スタブを取得して、スタブのメソッドを呼び出します。

スタブを取得するには、以下の方法があります。

- JNDIを使用してServiceオブジェクトをlookupする方法
- ServiceFactoryを使用する方法

JNDIを使用してServiceオブジェクトをlookupしたスタブの取得

Webアプリケーション、EJBアプリケーションまたはJ2EEアプリケーションクライアントからWebサービスを呼び出す場合、JNDIを使用してServiceオブジェクトをlookupすることができます。

deployment descriptorの編集

Webサービスクライアントアプリケーションから JNDIを使用してServiceオブジェクトをlookupする場合、クライアントアプリケーションの形態に応じて以下のdeployment descriptorにservice reference記述を定義する必要があります。

クライアントアプリケーションの形態	deployment descriptor
Webアプリケーション	web.xml
EJBアプリケーション	ejb-jar.xml
J2EEアプリケーションクライアント	application-client.xml

Serviceオブジェクトを取得してスタブからWebサービスにアクセス

1. InitialContextオブジェクトを生成する
   InitialContextオブジェクトを新規に生成します。
2. Serviceオブジェクトを取得する
   取得したInitialContextオブジェクトのlookupメソッドを使用して、Serviceオブジェクトを取得します。
   lookupの引数には、以下の文字列を指定します。
   java:comp/env/[deployment descriptorの<service-ref-name>に指定した値]
3. スタブオブジェクトを取得する
   取得したServiceオブジェクトのメソッドを使用し、スタブオブジェクト(サービスポイントインタフェースを実装したクラスのインスタンス)を取得します。
4. スタブオブジェクトのメソッドを呼び出す
   取得したスタブオブジェクトのメソッドを呼び出し、Webサービスにアクセスします。

注意

JNDIを利用してServiceオブジェクトをlookupする場合は、単一のInitialContextオブジェクトを複数のスレッドで使用することはできません。

ポイント

J2EEアプリケーションクライアントからJNDIを使用する場合は、JNDIサービスプロバイダの環境設定をする必要があります。JNDIサービスプロバイダの環境設定については、"Interstage Application Server J2EE ユーザーズガイド(旧版互換)"を参照してください。

ServiceFactoryを使用したスタブの取得

1. ServiceFactoryオブジェクトを取得する
   JAX-PRCで提供されているjavax.xml.rpc.ServiceFactoryクラスのnewInstanceメソッドを呼び出し、ServiceFactoryオブジェクトを取得します。

2. Serviceオブジェクトを取得する
   取得したServiceFactoryオブジェクトのloadService(java.lang.Class)メソッドを使用し、Serviceオブジェクト(サービスインタフェースを実装したクラスのインスタンス)を取得します。

3. スタブオブジェクトを取得する
   取得したServiceオブジェクトのメソッドを使用し、スタブオブジェクト(サービスエンドポイントインタフェースを実装したクラスのインスタンス)を取得します。

4. スタブオブジェクトのメソッドを呼び出す
   取得したスタブオブジェクトのメソッドを呼び出し、Webサービスにアクセスします。

使用するメソッドの詳細は、"Interstage Application Server J2EE ユーザーズガイド(旧版互換)"およびWebサービス関連のjavadocを参照してください。

D.3.6 Webサービスの動作を確認する

Webサービスの動作の確認については、"D.4.7 アプリケーションの動作確認"を参照してください。

D.3.7 Webサービスを運用環境に配布する

Webサービスの運用環境への配布は、アーカイブを作成する必要があります。詳細は"6.2.8 運用環境への配布"を参照してください。

D.4 J2EEアプリケーション共通事項

J2EE1.4アプリケーション実行環境上で動作するアプリケーションを作成するうえでの共通事項をJava EEアプリケーションの開発方法と異なる点を中心に説明します。

D.4.1 アプリケーションの動作確認を行う配備先の準備

アプリケーションの動作確認を行う配備先となるIJServerは以下の方法で作成します。

IJServer(MyDebug/My1VMDebug)の作成

IJServerを作成するには、通常は、Interstage管理コンソールを使用しますが、ここでは、ローカルデバッグ向けのIJServer用の定義ファイルを利用して、コマンドで作成する方法を示します。

Interstage管理コンソールからIJServerを作成する方法については、"Interstage Application Server J2EE ユーザーズガイド(旧版互換)"を参照してください。

1. アプリ画面(Windows 8およびWindows Server 2012の場合)およびスタートメニュー(その他のWindowsの場合)から [Interstage Studio Vxx] > [Interstage基盤サービス操作ツール]を選択します。

2. ツール画面の[localhostで使用するデバッグ環境]の[J2EE実行環境を使用する]がチェックされているかを確認します。チェックされていなければチェックしてください。

3. [サービスの情報]の[状態]を確認します。サービスが"停止"または"停止(一部)"の場合には[必須サービスのみ起動状態にする]をクリックしてサービスを起動してください。

4. 操作が完了したら[閉じる]をクリックしてツール画面を閉じてください。

5. J2EEの環境設定／定義操作コマンド（isj2eeadmin）を実行します。

   IJServer定義ファイルを引数にして、以下のコマンドを実行します。
   isj2eeadminコマンドの詳細については、"Interstage Application Server 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。

```
isstart
isj2eeadmin ijserver -a -f Java統合開発環境のインストールフォルダ¥etc¥ijserver¥ijserver_studio_debug.xml
```

注意

IJServer定義ファイル内の<Location>タグには、参照モジュールの情報が設定されています。通常は、インストール時にインストール先フォルダの情報から値が設定されますが、パス構成に誤りがある場合は、適宜インストール環境に合わせて、設定を変更してください。

D.4.2 アプリケーションを作成する環境を準備する

アプリケーションを作成するためには、開発するモジュールに応じてプロジェクトを作成する必要があります。

また、複数のモジュールで構成されるようなアプリケーションについては、エンタープライズアプリケーションとしてまとめることができます。

アプリケーションの作成の準備については、"6.2.2 アプリケーション作成のための準備"を参照してください。

プロジェクトの作成

J2EE1.4のエンタープライズアプリケーションを開発する場合、[Java EE] > [エンタープライズアプリケーションプロジェクト]を選択し、EARアプリケーションプロジェクトウィザードで指定するターゲットランタイムに[Interstage Application Server V11.1 IJServer (J2EE)]を選択します。

注意

EARバージョンが1.3以前のプロジェクトの新規作成はサポート対象外です。

D.4.3 Javaクラスおよびインタフェースを作成する

Javaクラスおよびインタフェースの作成については、"6.2.3 Javaクラスおよびインタフェースを作成する"を参照してください。

D.4.4 XMLファイルを作成する

XMLファイルの作成については、"6.2.4 XMLファイルを作成する"を参照してください。

D.4.5 プロパティファイルを作成する

プロパティファイルの作成については、"6.2.5 プロパティファイルを作成する"を参照してください。

D.4.6 問題の検出と修正

問題の検出と修正については、"6.2.6 問題の検出と修正"を参照してください。

D.4.7 アプリケーションの動作確認

各J2EE1.4アプリケーションをInterstage Application ServerのJ2EEコンテナ（IJServer）で動作確認する方法を説明します。

J2EEアプリケーションを動作確認するには、J2EEアプリケーションをサーバに配備して、サーバを起動します。これらの操作はサーバビューで行います。クライアントからJ2EEアプリケーションを呼び出すことで動作確認を行います。また、プログラムの実行を中断してコードを1行ずつ実行したり、変数の値を確認するには、デバッグを行います。

サーバを操作するための準備

サーバの起動や停止といった操作を行うために、サーバビューに操作対象とするサーバを追加します。

サーバを追加する

動作確認を行うアプリケーションの配備先となるサーバビューをサーバに追加します。サーバの追加には、新規サーバウィザードを使用します。サーバビューでコンテキストメニューから[新規] > [サーバ]を選択します。ウィザードの設定項目については、以下を参考にしてください。

設定項目	設定内容
サーバのホスト名	サーバがあるホストの名前を指定します。ローカルマシンの場合は"localhost"を指定します。
サーバのタイプ	サーバのタイプを選択します。対象とするバージョンのサーバタイプを選択します。InterstageのV11のJ2EEコンテナの場合は、[FUJITSU LIMITED] > [Interstage Application Server V11.1 IJServer (J2EE)]を選択します。
サーバ名	サーバ名をカスタマイズできますが、Interstage Studioでは、ここでの変更は反映されず、以下の形式となります。 「IJServer名 [サーバタイプ名] (ホスト名)」
サーバランタイム環境	サーバのタイプに対応するランタイム設定を指定します。サーバタイプとサーバランタイムは1対1で関連付いているため、特に変更する必要はありません。InterstageのV11のJ2EEコンテナの場合は、[Interstage Application Server V11.1 IJServer (J2EE)]を選択します。 サーバランタイム環境が登録されていない場合は、[サーバランタイム環境]の項目は表示されずに、[Interstage Application Server]ページが表示されます。サーバランタイム環境は自動的に登録されるので、[次へ]ボタンを押してください。
管理コンソールへの接続にHTTPS通信を使用する	Interstage管理コンソールを表示する際に、SSL暗号化通信を使用する場合にチェックします。サーバの設定に合わせる必要があります。 Interstage管理コンソールの運用に、SSL暗号化通信を使用するかどうかの設定は、Interstageインストール時に"運用形態の選択"で選択することができます。
Interstage管理コンソールのポート番号	Interstage管理コンソールのポート番号を指定します。通常は変更する必要はありません。デフォルトのポート番号は12000です。サーバ環境に合わせて変更してください。
Interstage JMX サービスのポート番号	Interstage JMXサービスが、Interstage管理コンソールの要求を受け付けるポート番号を指定します。デフォルトのポート番号は12200です。サーバ環境に合わせて変更してください。ホストがリモート環境の場合に設定可能です。
ログイン	サーバへの接続を行います。既に接続済みもしくは、ボタンを押してログインに成功するとボタンは押せない状態となります。 接続先のサーバがlocalhost以外の場合は、認証ダイアログボックスが表示されるので、ユーザ名とパスワードを指定します。ただし、localhostの場合でも、管理者権限のないユーザの場合は、認証ダイアログボックスが表示されるので、管理者権限を持つユーザ名および、パスワードを設定してください。
ユーザ名	サーバ接続に使用するユーザ名を指定します。ここで指定したユーザ名でアプリケーションサーバの認証が行われます。認証に使用するユーザ名によっては、操作が制限されます。認証と制限される操作の詳細は、"Interstage Application Server 運用ガイド(基本編)"を参照してください。
パスワード	認証に使用するパスワードを指定します。
IJServerクラスタ	接続したサーバに定義されているIJServerクラスタの一覧が表示されるので、対象とするIJServerクラスタを選択します。
HTTPポート番号	サーバのHTTPポート番号を指定します。サーバ環境に合わせて変更してください。
プロジェクトの追加および除去	サーバに配備するプロジェクトを指定します。使用可能なプロジェクトにプロジェクトの一覧が表示されるので、配備するプロジェクトを構成プロジェクトに追加します。 プロジェクトの追加は、サーバ追加後にもコンテキストメニューから実施することができます。

注意

- EJB2.1のアプリケーションおよび、Webサービスアプリケーションが動作可能なIJServerの種別はWebアプリケーションとEJBアプリケーションを同一JavaVMで運用のみとなります。
- [サーバのタイプ]で旧版のサーバタイプを選択する場合は、サーバタイプで選択したバージョンのInterstage Application Serverのクライアント機能がインストールされている必要があります。また、接続するサーバにはリモート環境にあるサーバを指定してください。

プロジェクトとサーバの関連付け

サーバに配備するプロジェクトの指定は、プロジェクトをサーバビューのサーバに追加することによって行います。サーバに配備するプロジェクトを追加するには、プロジェクトの追加および除去ウィザードを使用します。プロジェクトの追加および除去ウィザードを起動するには、サーバビューでサーバを選択し、コンテキストメニューから[プロジェクトの追加および除去]を選択してください。

ダイアログボックスの設定項目については、以下を参考にしてください。

設定項目	設定内容
プロジェクトの追加および除去	サーバに配備するプロジェクトを指定します。使用可能なプロジェクトにプロジェクトの一覧が表示されるので、配備するプロジェクトを構成プロジェクトに追加します。

注意

サーバに既に別環境などで配備済みのアプリケーションが存在する場合は、サーバビューでは、プロジェクト名ではなく、モジュール名で表示されます。既存で配備済みのアプリケーションがあり、そのプロジェクトをインポートして配備した場合、サーバビュー上は「プロジェクト名」と「アプリケーション名」の両方が表示される状態となりますが、APSへの配備モジュールは同一のものとなります。

サーバの起動

サーバビューでサーバを選択し、コンテキストメニューから[開始]または[デバッグ]を選択します。サーバを起動するときにアプリケーションを自動的に配備することができます。

ポイント

IJServerクラスタへの接続がまだ行われていない場合には、サーバビューでのIJServerクラスタの[状態]欄に何も表示されず、起動や停止といった操作が行えません。その場合はサーバビューでそのIJServerクラスタを選択し、コンテキストメニューから[接続/ログイン]を選択してIJServerクラスタへの接続を行ってください。

サーバを停止するには、サーバビューでサーバを選択して、ツールバーの[停止]を選択します。

注意

IJServerクラスタへ接続されている状態で、サーバのInterstage基盤サービスが停止された場合は、IJServerクラスタへの接続が切断されるため、サーバビューでのIJServerクラスタの操作に失敗します。操作に失敗した場合は、サーバビューでIJServerクラスタを選択し、コンテキストメニューから[最新表示]を選択してIJServerクラスタの状態を更新してから、コンテキストメニューの[接続/ログイン]を選択してIJServerクラスタへの再接続を行ってください。

アプリケーションの実行

サーバに配備したJ2EEアプリケーションの動作を確認するには、J2EEアプリケーションを呼び出すクライアントアプリケーションを実行します。

クライアントアプリケーションが、Webアプリケーションの場合は、Webブラウザを起動し、アプリケーションのURLにアクセスすることで動作確認を行います。

ポイント

Webプロジェクトの場合は、サーバビューで、配備されているWebプロジェクトを選択して、コンテキストメニューから[Webブラウザ]を選択すると、URLを内部的に組み立ててWebブラウザを起動します。

アプリケーションのデバッグ

アプリケーションをデバッグするには、サーバビューでサーバを選択し、コンテキストメニューから[デバッグ]を選択します。ブレークポイントを設定して、起動されたプログラムを中断し、コードを1行ずつ実行し、変数の内容を確認することによって行います。

デバッグの詳細については、"6.2.7.1 デバッグする"を参照してください。

D.5 J2EE1.4アプリケーション留意事項

D.5.1 EJB関連

クライアント配布物

Enterprise Beanを呼び出すアプリケーションを作成するためには、Enterprise Beanの Homeインタフェース/Remoteインタフェースのコンパイル済みクラスをクラスパスに設定する必要があります。このコンパイル済みクラスをクライアント配布物と呼びます。

クライアント配布物は、以下のマニュアルを参考に取得してください。

- "Interstage Application Server J2EE ユーザーズガイド(旧版互換)"の"EJBを参照する場合の環境設定"の"クライアント配布物の設定方法"

java.naming.factory.initialプロパティの設定

クライアントアプリケーションからEnterprise Beanを呼び出す場合には、以下の組み合わせで、javaコマンドの-Dオプションなどを使用してシステムプロパティを設定する必要があります。

クライアント種別	キー	値	備考
EJBクライアント	java.naming.factory.initial	com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient	
J2EEアプリケーションクライアント		com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient	J2EEアプリケーションクライアントのdeployment descriptorに参照するEJBの情報を指定する必要があります。

Webサービス化した場合はEARにする

Stateless Session BeanをWebサービス化した場合は、EARファイルで配備する必要があります。[新規 EARアプリケーションプロジェクト]ウィザードを実行して、エンタープライズアプリケーションプロジェクトを作成します。このとき、エンタープライズアプリケーションに追加するJ2EEモジュールとして、Enterprise JavaBeansプロジェクトを選択します。

リモートサーバを使う場合のネーミングサービスの環境変更

ネーミングサービスにアクセスするクラスの実装を指定するため、以下のように環境プロパティ(java.naming.factory.initial)に実装クラス名を指定する必要があります。

EJBクライアントの場合はEnterprise Beanを呼び出す以外(トランザクション制御やリソースアクセス)は行えません。

クライアント種別	値
EJBクライアント	com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient
J2EEアプリケーションクライアント	com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient

設定する方法には以下があります。

- jndi.propertiesファイル
- FJjndi.propertiesファイル
- javax.naming.InitialContext(Hashtable environment)引数

・ アプリケーション起動時のjavaコマンドラインでの引数(-D)

実際にクライアントアプリケーション実行時にエラーが発生する場合は、環境が正しく設定されていない可能性があります。"Interstage Studio ユーザーズガイド"の"トラブルシューティング"やInterstage Application Serverのマニュアルなどを参考に環境を再確認してください。

クライアント製品の場合、ネーミングサービス接続先サーバの指定

Interstage Application Serverのクライアント機能がインストールされている状態で、実際にリモート環境のEnterprise Beanを呼び出す場合には、ネーミングサービスにアクセスするために、ネーミングサービスの接続先情報を指定する必要があります。以下のファイルにサーバ情報を記述してください。

Interstage Application Serverインストールフォルダ¥odwin¥etc¥inithost

1VMはリモート環境からEnterprise Beanを呼び出せない

配備先のIJServerの種別が[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]の場合には、リモート環境からEnterprise Beanを呼び出すことができません。クライアントから呼び出すためには、[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]もしくは[EJBアプリケーションのみ運用]に配備する必要があります。

EJB2.1のアプリケーションは、リモート環境からアクセスできない

Interstage Application Serverで運用するEJB2.1のアプリケーションは、リモート環境からアクセスすることができないため、Webアプリケーションからローカルにアクセスするようにしてください。

JMSコネクションファクトリとDestination

クライアントーサーバで、Message-driven Beanを動作させるためには、JMSコネクションファクトリとDestinationをクライアント側でも作成する必要があります。クライアント側で作成する場合にはJMS運用コマンドを使って作成してください。

クライアント側では、Interstage Application ServerのJMS運用コマンドを使用し、以下のように登録処理を行います。

・ JMSコネクションファクトリの登録

デフォルトのJMSコネクションファクトリを使用する場合は登録の必要はありません。
デフォルト以外を使用する場合はjmsmkfactコマンドを使用します。
jmsmkfactコマンドの詳細については、"Interstage Application Server 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。

・ Destination定義の登録

Destination定義の登録は、jmsmkdstコマンドを使用します。
jmsmkdstコマンドの詳細については、"Interstage Application Server 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。

D.5.2 Webサービスアプリケーション関連

アプリサーバが必須

Interstage StudioでWebサービスを開発するには、Interstage StudioのInterstage Application Server機能をインストールするか、もしくはInterstage Application Serverクライアントパッケージをインストールしておく必要があります。

WSDL location情報

WSDLから生成されるファイルには、WSDLファイルに記述されているlocation情報(接続先のWebサービスのURL)が出力されます。運用環境を変更する場合などのため、アプリケーションを再構築することなく、環境設定で接続先を変更することは可能ですが、利用するWebサービスのlocation情報が記述されているWSDLファイルを使うと手間がかかりません。

Interstage管理コンソールより取得できる公開用WSDLファイルは、取得時に配備先の環境に合わせてWSDLファイルのlocation情報が更新されており、これを使用すると接続先についてユーザが意識する必要はありません。

ただし、Webサービスアプリケーションの開発も同時に行うような場合には、デバッグ環境のlocation情報をWebサービスウィザードで指定しておくことで、配備などの作業を行わずに、開発資産のWSDLファイルをそのまま使うことができ、WebサービスアプリケーションとWebサービスクライアントの開発を平行して行うことができます。

WebサービスクライアントがJ2EEアプリケーションで、サービスインタフェースをJNDIのlookupを利用して取得する場合は、WSDLファイルに記述されているlocation情報を意識する必要はありません。

相互接続先

Webサービスの相互接続性を向上させる目的で、WS-I Basic Profileというガイドラインが作成されています。このガイドラインでは、主に、Webサービスで交換されるメッセージの形式やWSDLの記述についての指針が記述されています。

旧資産の移行

以前のバージョンでは、メッセージ交換方式としてMessaging方式とRPC方式がありました。

RPC方式については、使用可能な型などの範囲が全く同じではありませんが、アプリケーションの実装部分については流用可能です。

既存のWSDLファイルの利用については、相互接続性などの観点からおすすめできないため、旧開発資産を流用する場合には以下のように作業を行ってください。流用時の注意点については、"Interstage Application Server 移行ガイド"を参照してください。

1. サービスエンドポイントインタフェースの作成

   以前に使用していたインタフェースがある場合には、それがサービスエンドポイントインタフェースの規約の範囲であればそのまま使用できます。規約の範囲外のデータ型を使用している場合には、使用可能なデータ型の範囲でサービスエンドポイントインタフェースを作成しなおしてください。
   インタフェースが無い場合には、公開している機能に合うように使用可能なデータ型の範囲でサービスエンドポイントインタフェースを新規に作成してください。

2. Webサービスウィザードによるファイル生成

   サービスエンドポイントインタフェースからWebサービスに必要なファイルを生成します。

3. 実装処理の移植

   既存の実装を流用し、データ型の変更や実行環境の違いなどを考慮して移植を行います。

Webサービスが動作するのは同一VMタイプのJ2EE実行環境のみ

Webサービスは、[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]タイプのJ2EE実行環境上でのみ動作します。

D.5.3 J2EEアプリケーション共通

J2EEモジュール開発からJ2EEアプリケーション開発移行時のJ2EEモジュールの配備解除

J2EEモジュールの開発が完了し、J2EEアプリケーションの開発に移る場合には、J2EEモジュールの管理方法の違いにより配備時にエラーが起こる可能性があるため、配備したJ2EEモジュールを配備解除する必要があります。

Interstage高速配備用のEARファイルを作成する場合には、作成時にEARファイルの配備が内部的に行われるため、EARファイルの作成前にJ2EEモジュールの配備解除を行っておく必要があります。

D.6 互換に関する情報

D.6.1 J2EEコンテナからJava EEコンテナへの移行

ここでは、J2EEコンテナ向けJ2EE1.4アプリケーションをJava EEコンテナ向けに変更する方法を説明します。

注意

以下のアプリケーションはJava EEコンテナ向けに移植することはできません。

- Webサービス(JAX-RPC)
- Webサービスクライアント(JAX-RPC)

Webアプリケーション

以下の手順でJava EEコンテナ上で動作可能となります。

1. プロジェクトのコンテキストメニューからプロパティを選択します。
2. [ターゲットランタイム]ページで[全てのランタイムを表示]にチェックを入れます。
3. 表示されたランタイムの中からJava EEのランタイムを選択し、[OK]をクリックします。

EJBアプリケーション

CMPを含まないEJBアプリケーションの移行処理

CMPを含まないEJBアプリケーションについては以下の手順でJava EEコンテナ上で動作可能となります。

1. プロジェクトのコンテキストメニューからプロパティを選択します。
2. [ターゲットランタイム]ページで[全てのランタイムを表示]にチェックを入れます。
3. 表示されたランタイムの中からJava EEのランタイムを選択し、[OK]をクリックします。

CMP2.0の移行処理

CMP2.0についてJava EEコンテナとJ2EEコンテナでは以下の違いがあります。

項目	JavaEEコンテナ	J2EEコンテナ
ターゲットランタイム	JavaEEコンテナ	J2EEコンテナ
CMP拡張情報ファイル	sun-cmp-mappings.xml	FJCMP_XXXX.xml
DB定義情報	sun-ejb-jar.xml	FJCMP_XXXX.xml
DBテーブル情報	XXX.dbschema	なし。

このため、CMP2.0をJava EEコンテナで利用するためにはsun-cmp-mappings.xml、sun-ejb-jar.xml、XXX.dbschemaを新規に生成する必要があります。これらのファイルの生成やターゲットランタイムの変更は、[CMP2.0 Java EE環境への更新]ウィザードから行います。[新規]ウィザードから[EJB] > [J2EE] > [CMP2.0 Java EE環境への更新]を選択してください。

以下の設定項目を確認、入力してください。

項目	説明
プロジェクト選択	移行対象のプロジェクトを選択します。 表示されるプロジェクトはCMPを含むEJBプロジェクトです。
DBスキーマを利用する	DBスキーマ利用の有無を指定します。
接続リストボックス	作成済みのDB接続一覧が表示され、この中から1つ選択します。
接続の追加	[新規接続プロファイル]ウィザードを開きます。
接続	接続リストボックスで選択されたDBに接続します。
JNDI名	JavaEEコンテナで利用するDB接続のためのJNDI名を指定します。

情報を設定後、[次へ]をクリックし、以下の項目を設定後、[完了]をクリックしてください。

項目	説明
プロジェクト名	選択したプロジェクトの一覧が表示されます。
EJB名	選択したプロジェクトに含まれるCMPの一覧が表示されます。
テーブル名	CMPに関連付くテーブル情報が表示されます。
スキーマ名	前ページで接続されたDBの情報から、テーブルの利用するスキーマを選択します。 CMP間にリレーションの関連が存在する場合、関連のある全てのCMPのスキーマ情報が変更されます。

注意

CMP2.0 Java EE環境への更新ウィザードはCMP1.1のマイグレーションには対応していません。
CMP2.0、CMP1.1の混在プロジェクトであった場合はCMP2.0のみ移行処理を行い、CMP1.1については移行処理を行いません。
CMP1.1の移行については"CMP1.1の移行処理"を参照してください。

CMP1.1の移行処理

CMP1.1の移行については、以下の作業を行う必要があります。

1. FJCMP_XXX.xmlからsun-cmp-mapping.xmlへの情報の移行

   CMP1.1のFJCMP_XXX.xmlに記述されている情報をsun-cmp-mapping.xmlへ記述します。

   各項目の関連については以下のとおりとなります。

sun-cmp-mappings.xml		FJCMP_XXX.xml
sun-cmp-mappings		
sun-cmp-mapping+		
schema		fujitsu-cmp-definition/schema-name
entity-mapping+	←	fujitsu-cmp-definition
ejb-name		EJB名(ejb-jar.xml/ejb-name)
table-name	←	fujitsu-cmp-definition /table-name
cmp-field-mapping+	←	fujitsu-cmp-definition /field-map
field-name	←	fujitsu-cmp-definition /field-map/field-map-entry/field-name
column-name+	←	fujitsu-cmp-definition /field-map/field-map-entry/dbcolumn-name

   その他の詳細な内容については、Interstage Application Serverのマニュアルを参照してください。

2. sun-ejb-jar.xmlへのJDOQLクエリの記述

   SQLクエリの仕様が変更されるため、sun-ejb-jar.xmlにSQLクエリ情報をJDOQL形式にて記述します。

   sun-ejb-jar.xmlの<ejb>/<ejb-name>、<ejb>/<cmp>/<one-one-finders>/<finder>

   この項目にJDOQLに乗っ取った記載方式で<method-name>、<query-params>、<query-filter>、<query-ordering>を記述します。

   finderメソッドの記述サンプルは以下となります。

```
<ejb>
    <ejb-name>TestCMP11</ejb-name>
    <jndi-name>jdbc/ORACLE</jndi-name>
    <cmp>
        <one-one-finders>
            <finder>
                <method-name>testMethod</method-name>
                <query-params>int param1</query-params>
                <query-filter> id &lt; param1</query-filter>
            </finder>
        </one-one-finders>
    </cmp>
</ejb>
```

   詳細な内容については、Interstage Application Serverのマニュアルを参照してください。

3. DBスキーマファイルの生成

   以下のコマンドを利用してDBスキーマファイルを生成します。

   <Interstage Studioインストールディレクトリ>¥APS¥F3FMisjee¥bin¥capture-schema.bat

コマンドを実行するにあたり以下の情報を指定する必要があります。

必要項目	値
DBユーザ名	データベース接続ユーザ名
DBユーザパスワード	データベースパスワード
DB接続先URL	JDBC接続先のURL
JDBCドライバ名	利用するJDBCドライバのドライバ名
出力先ファイルパス	{ワークスペースディレクトリ}/{プロジェクト}/{ソースディレクトリ}/{sun-cmp-mappings/sun-cmp-mapping/schema}.dbschema
利用スキーマ名	fujitsu-cmp-definition/schema-name
利用テーブル名	fujitsu-cmp-definition/table-name

コマンドは以下の形式で実行します。

```
capture-schema -username {DBユーザ名} -password {DBユーザパスワード}
 -dburl {DB接続先URL} -driver {JDBCドライバ名} -out {出力ファイルパス}
[-schemaname {利用スキーマ名}] [-table {利用テーブル名}]*
```

Earアプリケーション

以下の手順でJava EEコンテナ上で動作可能となります。

1. プロジェクトのコンテキストメニューからプロパティを選択します。
2. [ターゲットランタイム]ページで[全てのランタイムを表示]にチェックを入れます。
3. 表示されたランタイムの中からJava EEのランタイムを選択し、[OK]をクリックします。

付録E　トラブルシューティング

ここでは、ワークベンチで発生する問題を解決する方法を説明します。

E.1　ワークベンチ一般に関する問題

ファイル、フォルダ、プロジェクトが削除できない。

【現象】

ワークベンチからの操作によって、ファイルシステムのパス長の最大値を超えるパスのファイルが誤って生成される場合があり、ワークベンチやエクスプローラからそのファイル、および、そのファイルを含んでいるフォルダ、プロジェクトの削除が失敗する場合があります。

【対処】

以下の方法で、ファイルシステムのパス長の最大値を超えるパスのファイルおよびフォルダを削除してください。

```
java -classpath <ワークベンチのインストールフォルダ> ¥bin¥f5drprfc.jar RemoveFolderContents フォルダ名
```

本コマンドは、指定されたフォルダ内にあるファイルおよびフォルダを、1つずつ削除するかどうか問い合わせた上で削除します。指定されたフォルダ自身の削除は行いません。

[Windowsセキュリティの重要な警告]ダイアログボックスが表示される。

【現象】

以下の場合に[Windowsセキュリティの重要な警告]ダイアログボックスが表示されることがあります。

- アプリケーションのデバッグを実行した場合
- WebアプリケーションあるいはApcoordinatorアプリケーションを、Tomcat起動構成を用いて実行した場合
- ワークベンチのメニューから[ヘルプ] > [ヘルプ目次]を選択した場合

[Windowsセキュリティの重要な警告]ダイアログボックスには以下のように表示されます。

```
   コンピュータを保護するため、このプログラムの機能の一部が
Windowsファイアウォールでブロックされています。
   このプログラムをブロックし続けますか？

     名前(N):    Java(TM) 2 Platform Standard Edition binary
     発行元(P):  FUJITSU LIMITED
```

【対処】

[Windowsセキュリティの重要な警告]ダイアログボックスで、[ブロックする]、[ブロックを解除する]、あるいは、[後で確認する]のいずれかのボタンを押してダイアログボックスを閉じてください。どのボタンを押しても問題なく使えるようになります。

[Windowsセキュリティの重要な警告]ダイアログボックスに表示されるメッセージはOSによって異なります。ここでは、Windows XP Service Pack 2をもとに記述しています。

メモリ不足エラーが発生し、ワークベンチが終了する場合がある。

【現象】

ワークベンチ上でメモリを大量に使用するような処理を行った場合、以下のエラーメッセージが表示されてワークベンチが終了することがあります。

「メモリ不足エラーが発生しました。この種のエラーが今後発生しないようにするには Readme の"Running Eclipse"セクションを参照してください。」

【対処】

このような場合には、以下の記事を参考にヒープ領域を増やしてください。

<ワークベンチのインストールフォルダ>¥eclipse¥isstudio_ja.htm

Interstage管理コンソールまたはInterstage Java EE管理コンソールが起動しない。

【現象】

ローカルにインストールしたアプリケーションサーバのサービスは標準では停止状態です。停止状態のため、Interstage管理コンソールまたはInterstage Java EE管理コンソールが起動されません。

【対処】

ローカルのInterstage管理コンソールまたはInterstage Java EE管理コンソールを使用する場合には、Interstage基盤サービス操作ツールでアプリケーションサーバのサービスを起動してください。

注意

ワークベンチで発生する問題に関する注意

以下で発生した問題に対して当社のサポートを受けることはできません。

- 標準のワークベンチでデフォルトでは見えないようにしている機能

  設定ページの[一般] > [機能] > [拡張]で表示される機能のうち、デフォルトではチェックされていない"プラグイン開発"、"EMF開発"、"サーバランタイムの非推奨機能"などがデフォルトでは見えない機能です。

- Java EE 6ワークベンチの以下の機能
  - CVS機能
  - EMF(Eclipse Modeling Framework)機能
  - プラグイン開発機能
  - EJBのXDoclet関連機能
  - 静的Webプロジェクト関連機能
  - WebサービスのAxis関連機能
  - Interstage Application Server以外のサーバ機能
  - ソフトウェア説明書の"関連ソフトウェア"に記載しているデータベース製品以外に関するデータベース機能

E.2 Javaに関する問題

ウィザードなどで生成されたソースにおいて総称型に関する警告エラーが表示される。

【現象】

ウィザードなどで生成されたソース、またはサンプルをインポートしてビルドした際、Java 5で導入された総称型に関する警告エラーが表示される場合があります。

【対処】

総称型の警告を表示したくない場合は、[ウィンドウ]メニュー > [設定] > [Java] > [コンパイラ] > [エラー/警告]の[総称型]の各値を[無視]に変更してお使い下さい。

E.3 データベースに関する問題

ビューのフィルタを設定しても、すべてのビューがデータソースエクスプローラビューに表示される

【現象】

データソースエクスプローラビューの「ビュー」項目のプロパティで[フィルタを使用不可にする]のチェックを外してフィルタを有効にしても、すべてのビューがデータソースエクスプローラビューに表示されます。

【対処】

データソースエクスプローラビューの「ビュー」項目のプロパティでは、ビューのフィルタリングを行うことはできません。ビューをフィルタリングしたい場合は、そのビューが属しているスキーマの「テーブル」項目のプロパティでフィルタを指定してください。「テーブル」項目のフィルタ設定が、テーブルとビューの両方に適用されます。

Symfoware Serverのテーブルを削除できない場合がある

【現象】

Symfoware Serverのテーブルをテーブルデータエディタで編集したあとに、テーブルを削除するSQL文(DROP TABLE)を実行すると失敗します。テーブルの削除に失敗した場合は、SQL結果ビューに「JYP3913E 表"テーブルの名前"を他の利用者が占有しています」と表示されます。

【対処】

データソースエクスプローラビューでデータベースとの接続を切断してから、再びデータベースに接続してください。それから、テーブルを削除するSQL文(DROP TABLE)を実行してください。

Symfoware Serverのインデックスが異なる名前(DSO名)で表示される

【現象】

データソースエクスプローラビューでは、データベースに定義されているインデックスの一覧を表示することができます。Symfoware Serverに接続している場合は、インデックスの名前にインデックス名ではなくDSO名が表示されます。

テーブルの[削除]コンテキストメニューを選択すると、テーブルを削除するSQL文(DROP TABLE)が生成されます。また、テーブルにインデックスが定義されている場合は、インデックスを削除するSQL文(DROP INDEX)も生成されます。Symfoware Serverに接続している場合は、このインデックスを削除するSQL文に、インデックス名ではなくDSO名が指定されてしまいます。このため、インデックスを削除するSQL文を実行すると失敗する場合があります。

【対処】

インデックスを削除するSQL文にインデックス名を指定して、SQL文を実行してください。または、DSOを削除するSQL文(DROP DSO)とDSIを削除するSQL文(DROP DSI)を実行して、インデックスを削除してください。インデックスのDSO名はDSI名と同じ名前です。

E.4 JavaScriptに関する問題

JavaScriptの構文解析でエラーとなる場合がある

【現象】

JavaScriptの構文解析では、予約語や正規表現のリテラル形式が構文解析で正しく扱われない場合があり、その結果として不当なエラーが検出される場合があります。

【対処】

JavaScriptの構文解析を実行しないようにしてください。以下に手順を説明します。

- JavaScript バリデータを実行しないようにする場合

  1. プロジェクトエクスプローラで対象となるプロジェクトを選択します。
  2. コンテキストメニューの[プロパティ]を選択して、[プロパティ]ダイアログボックスを開きます。
  3. [プロパティ]ダイアログボックスの左側のツリービューで[ビルダ]を選択します。
  4. [ビルダ]ページで、[JavaScriptバリデータ]のチェックを外します。

- JavaScript 構文検証を実行しないようにする場合

  1. プロジェクトエクスプローラで対象となるプロジェクトを選択して、コンテキストメニューの[プロパティ]を選択して、[プロパティ]ダイアログボックスを表示します。または、ワークベンチのメニューの[ウィンドウ]>[設定]を選択して、[設定]ダイアログボックスを表示します。

2. [プロパティ]ページ、または、[設定]ページの左側のツリーから[検証]を選択します。

3. [検証]ページで、[JavaScript 構文検証]の[ビルド]のチェックを外します。

ポイント

問題ビューから問題を削除するには、JavaScriptの構文解析を実行しないようにする前に自動的にビルドを行わない状態でプロジェクトのクリーンを実行する必要があります。

注意

JSPエディタで、<script>～</script>タグ内のJavaScriptコードにJSPタグを記述すると、問題ビューにJavaScriptの構文エラーが表示されることがあります。

JSPタグはJSPの実行時に動的に変換されるものですが、エディタでのJavaScript構文チェック時にはJSPタグを変換することができないため、その部分が構文エラーと見なされてしまうものです。

このような場合にはそのままビルドや実行の操作を行ってください。構文エラーが出てもビルドや実行の操作は可能です。

E.5 エディタに関する問題

JavaエディタあるいはAntエディタでブレークポイントの操作ができない。

【現象】

JavaエディタあるいはAntエディタで、以下の場合にエディタ上でブレークポイントを削除したり使用不可にしたりする操作ができなくなります。

- ブレークポイントが設定してある行より前の位置で行の追加や削除を行い、ファイルを保存せずにブレークポイントを操作する。

【対処】

以下のどちらかの方法で対処してください。

- ブレークポイントビューからブレークポイントを操作する。
- ファイルを保存してからブレークポイントを操作する。

CSSエディタでサイズの大きなファイルを扱うと動作が遅くなる場合がある。

【現象】

CSSエディタで、1000行を超えるようなサイズの大きなファイルを扱う場合、かつ以下の操作をした場合、CSSエディタの動作が遅くなることがあります。

- 一度に多くの行を削除する。
- 一度に多くの場所を編集前の状態に戻す。
  例) 文書全体をフォーマットしたあとで元に戻す操作をする。

【対処】

多くの行を削除するような操作をする場合には、テキストエディタを使用してください。また、フォーマットなど、文書全体に影響がある操作をする場合には、操作前にファイルを保存してください。元に戻す場合には、変更を破棄してファイルを一旦閉じ、開き直してください。

E.6 IJServerクラスタ、Interstage Java EE DASサービスに関する問題

IJServerクラスタのデバッグ起動に失敗する。

【現象】

IJServerクラスタをデバッグ起動した場合、起動に失敗し、以下のエラーダイアログボックスが表示されます。

タイトル:問題が発生

内容:'サーバを始動中 - {0}' に問題が発生しました。

サーバの起動に失敗しました。

詳細:サーバの起動に失敗しました。

OM2014: All server instances in cluster {1} were not started.OM1051: Server Instance unable to start: NAME={2}

【可変情報】

{0}:サーバビューで表示されているサーバ名

{1}:IJServerクラスタ名

{2}:IJServerクラスタを作成する際に指定したインスタンス名

【原因】

サーバビューにIJServerクラスタを登録する際に指定した[デバッグポート番号]のポートが既に使用されている事が原因です。

【対処】

以下のどちらかの方法で対処してください。

- [デバッグポート番号]に指定したポートを使用している他アプリケーションを終了して、再操作する。
- サーバビューで一度IJServerクラスタを削除して、再度[新規サーバ]ウィザードを使用して他と重複していない[デバッグポート番号]を指定した後に再操作する。

Interstage Java EE DASサービスのプロジェクトの操作に失敗する。

【現象】

Interstage Java EE DASサービスに対してプロジェクトの追加、削除などの操作を実施した場合、操作に失敗し、以下のようなエラーダイアログボックスが表示されます。

タイトル:「問題が発生」または「複数の問題が発生しました」

内容:公開が失敗しました。

詳細:公開が失敗しました。

モジュールの公開解除に失敗しました: {0}

OM2997: Unable to connect to admin-server at given host: [{1}] and port: [{2}]. Please check if this server is up and running and that the host and port provided are correct.

【可変情報】

{0}:操作対象のプロジェクト名

{1}:ホスト名

{2}:サーバ登録時に指定したInterstage JMXサービスのポート番号

【原因】

以下のどちらかが原因です。

- Interstage Java EE DASサービスが停止している状態でプロジェクトを操作した。
- サーバビューにInterstage Java EE DASサービスを登録する際に指定した[Interstage JMXサービスのポート番号]に誤りがある。

【対処】

Interstage JMXサービスのポート番号に誤りがない事を確認し、サーバビューで[接続/ログイン]または[接続(デバッグ起動)/ログイン]を行った後に再操作してください。

[新規サーバ]ウィザードで「Unexpected end of file from server」が表示され、ログインに失敗する。

【現象】

[新規サーバ]ウィザードの[新規Interstage Application Server]ページで、[ログイン]ボタンを押すと、以下のエラーメッセージが表示され、ログインに失敗します。

内容:次の理由によりログインに失敗しました:Unexpected end of file from server

【原因】

[新規サーバ]ウィザードの[新規サーバの定義]ページの[サーバのホスト名]に指定したホストにインストールされているInterstage Application ServerではHTTP接続を許可していないが、[新規Interstage Application Server]ページの[ターゲットとの接続にHTTPS通信を使用する]がチェックされていない。

【対処】

接続先のInterstage Application Serverの通信設定を確認し、必要であれば変更する。または、[新規Interstage Application Server]ページの[ターゲットとの接続にHTTPS通信を使用する]をチェックして[ログイン]ボタンを押してください。

E.7 サーバの動作確認に関する問題

EJB、Webアプリケーションなどの動作確認を行うと"Bad version number in .class"のエラーが表示される。

【現象】

EJBやWebアプリケーションの動作確認時にエラーメッセージが表示されます。

以下はWebアプリケーションの動作確認の例ですが、EJBではサーバのログに同様のエラーが出力されます。

```
HTTP Status 500 -
type
    Exception report
message
description
   The server encountered an internal error () that prevented it from > fulfilling this request.
exception
   javax.servlet.ServletException: JSVLT52352: サーブレットインスタンスを割り当て中のエラーです
root cause
    java.lang.UnsupportedClassVersionError: Bad version number in .class > file
```

【原因】

ビルドに使用する[Javaコンパイラ]の[JDK準拠レベル]が、動作確認を行うサーバのJDK/JREのバージョンよりも新しい事が原因です。
例:JDK準拠レベルが1.6で、サーバのJDK/JREが1.5。

【対処】

[Javaコンパイラ]の[JDK準拠レベル]が、動作確認を行うサーバのJDK/JREのバージョンと同じか、古くなるような環境でビルドして再度動作確認してください。

付録F　チュートリアル

ここでは、アプリケーションの開発手順を例題を使用して説明します。

F.1　Javaアプリケーション

Javaアプリケーションプロジェクトは、以下のアプリケーションを開発することができます。

- クライアントアプリケーション
- アプレット
- JavaBeans

F.2　クライアントアプリケーション

簡単なアプリケーションの作成を通して、クライアントアプリケーションの開発手順を説明します。

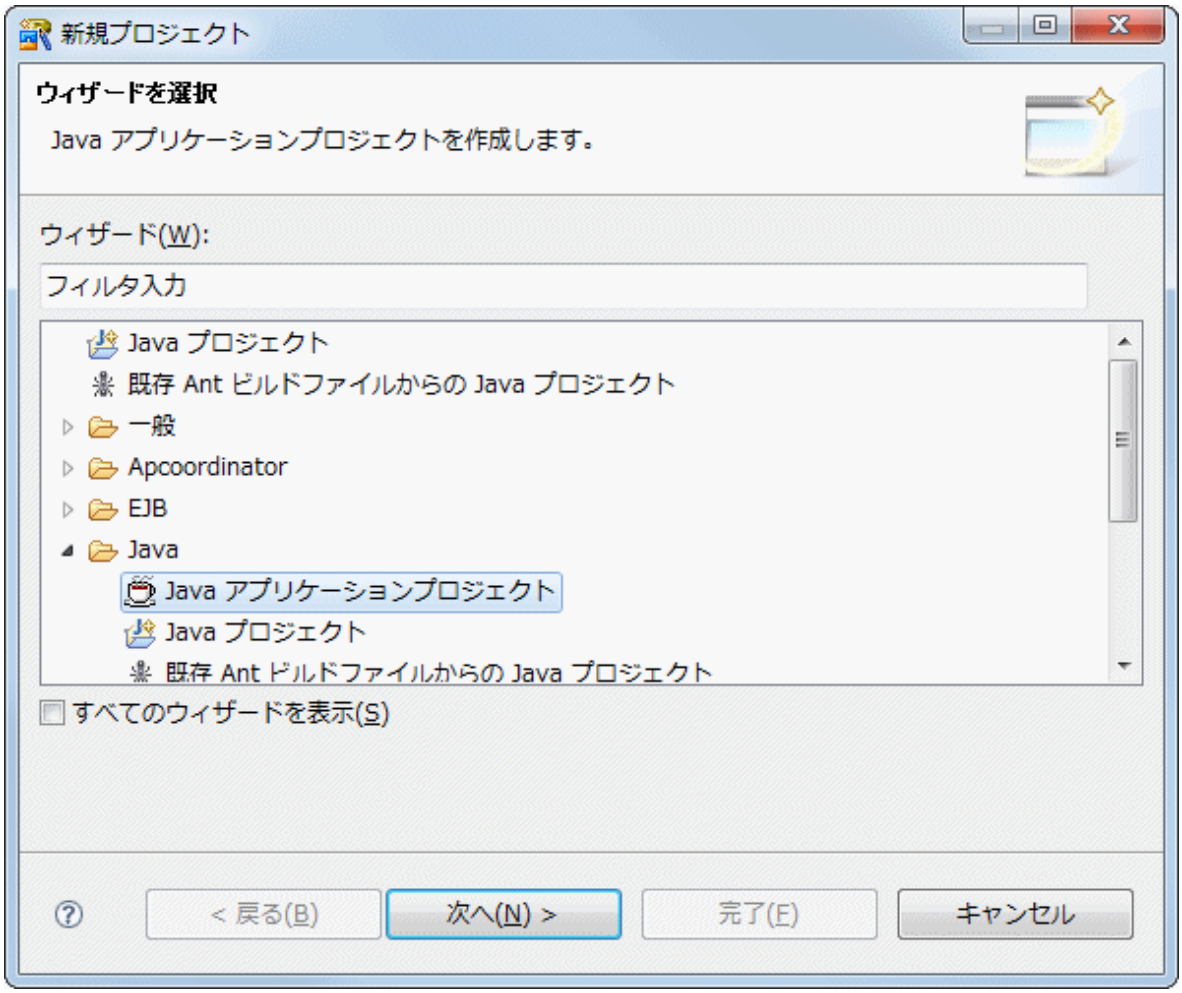
F.2.1　Lesson1 カレンダー

JBK(J Business Kit)のカレンダーBeanが使用されたアプリケーションの開発を通して、プロジェクトの新規作成からアプリケーションの実行までの手順を説明します。

Javaアプリケーションプロジェクトの作成

1. ワークベンチを起動します。
2. メニューバーから[ファイル] > [新規] > [プロジェクト]を選択します。

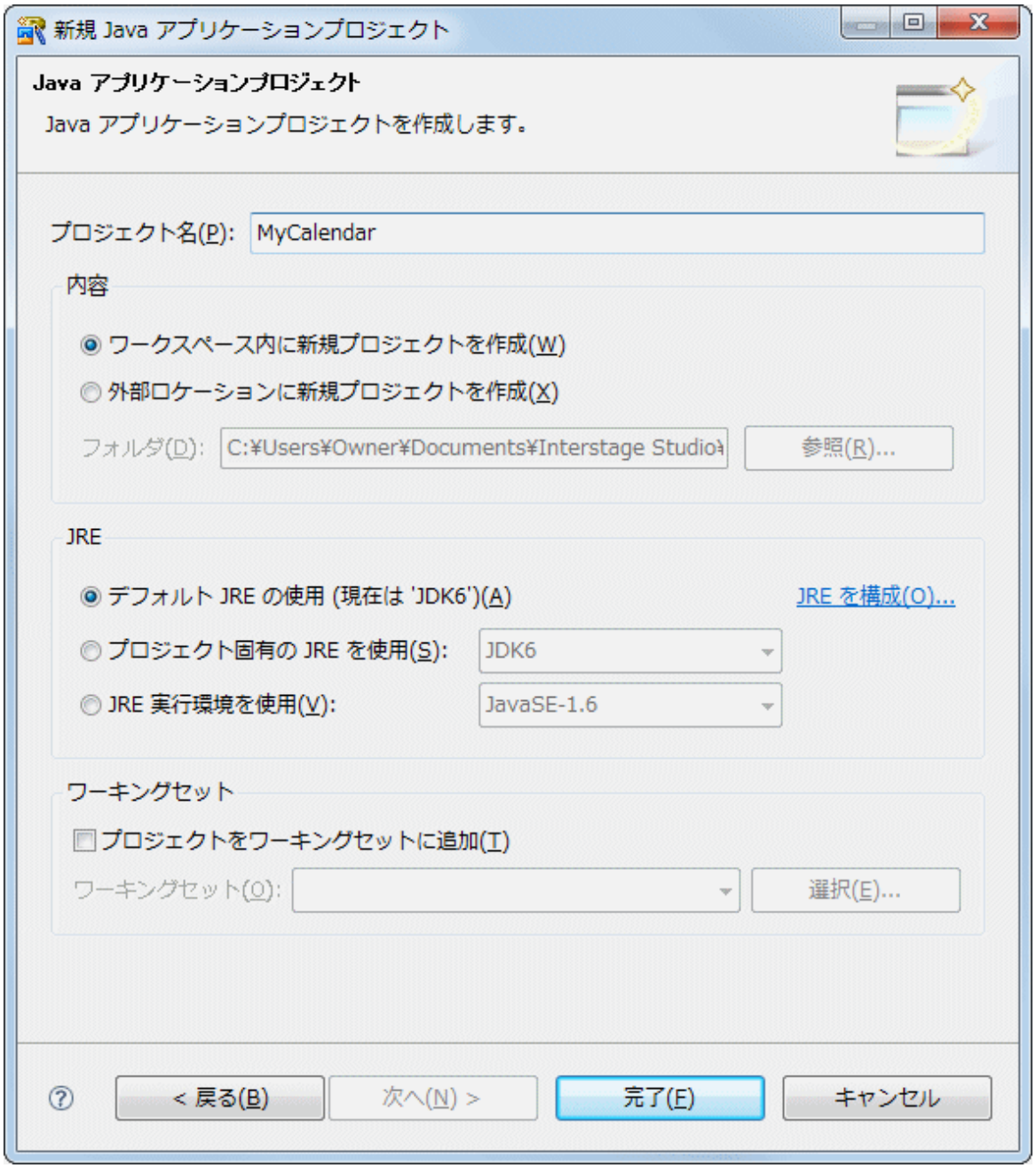
3. [新規プロジェクト]ウィザードが表示されます。
ツリーから[Javaアプリケーションプロジェクト]を選択します。



[次へ]をクリックします。

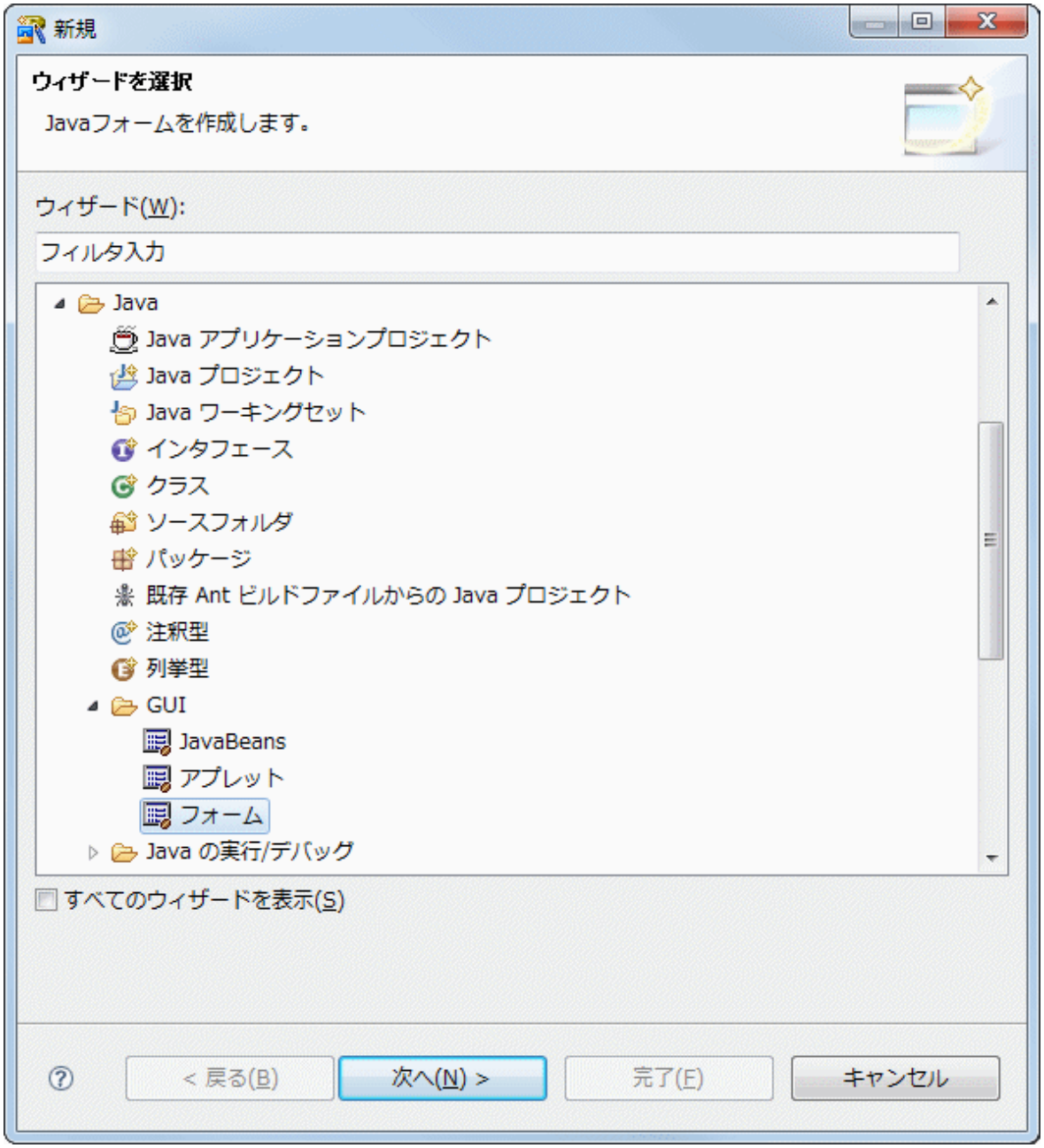
4. [Javaアプリケーションプロジェクト]ページが表示されます。
以下のようにプロジェクトの情報を入力します。

設定項目	設定内容
プロジェクト名	MyCalendar



[完了]をクリックします。
この操作で新規プロジェクトが生成されます。

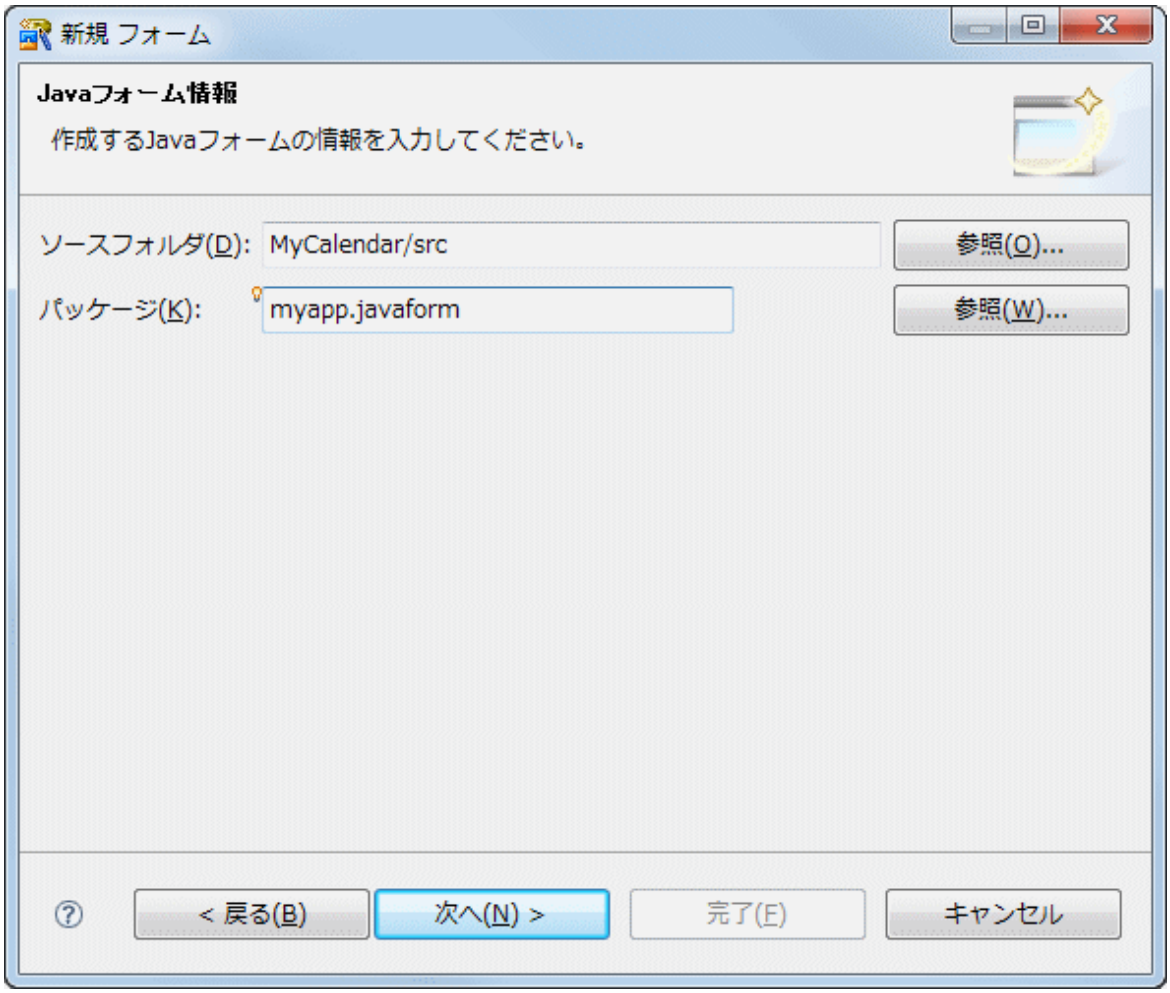
5. フレームを作成します。
   ワークベンチの[ファイル] > [新規] > [その他]を選択します。[新規]ウィザードが表示されます。
   ツリーから[Java] > [GUI] > [フォーム]を選択します。



   [次へ]をクリックします。

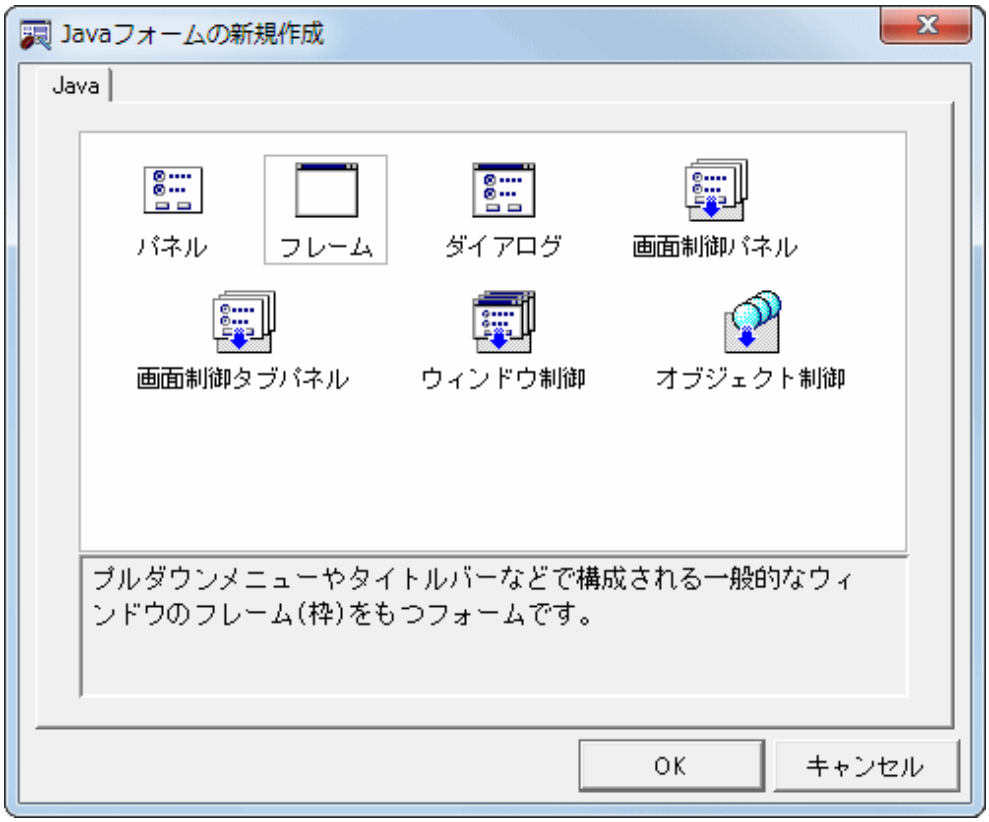
6. [Javaフォーム情報]ページが表示されます。
   このページでは、以下の情報を入力します。

設定項目	設定内容
ソースフォルダ	MyCalendar/src
パッケージ	myapp.javaform


新規 フォーム
Javaフォーム情報
作成するJavaフォームの情報を入力してください。
ソースフォルダ(D):
MyCalendar/src
参照(O)...
パッケージ(K):
myapp.javaform
参照(W)...
< 戻る(B)
次へ(N) >
完了(F)
キャンセル

[次へ]をクリックします。

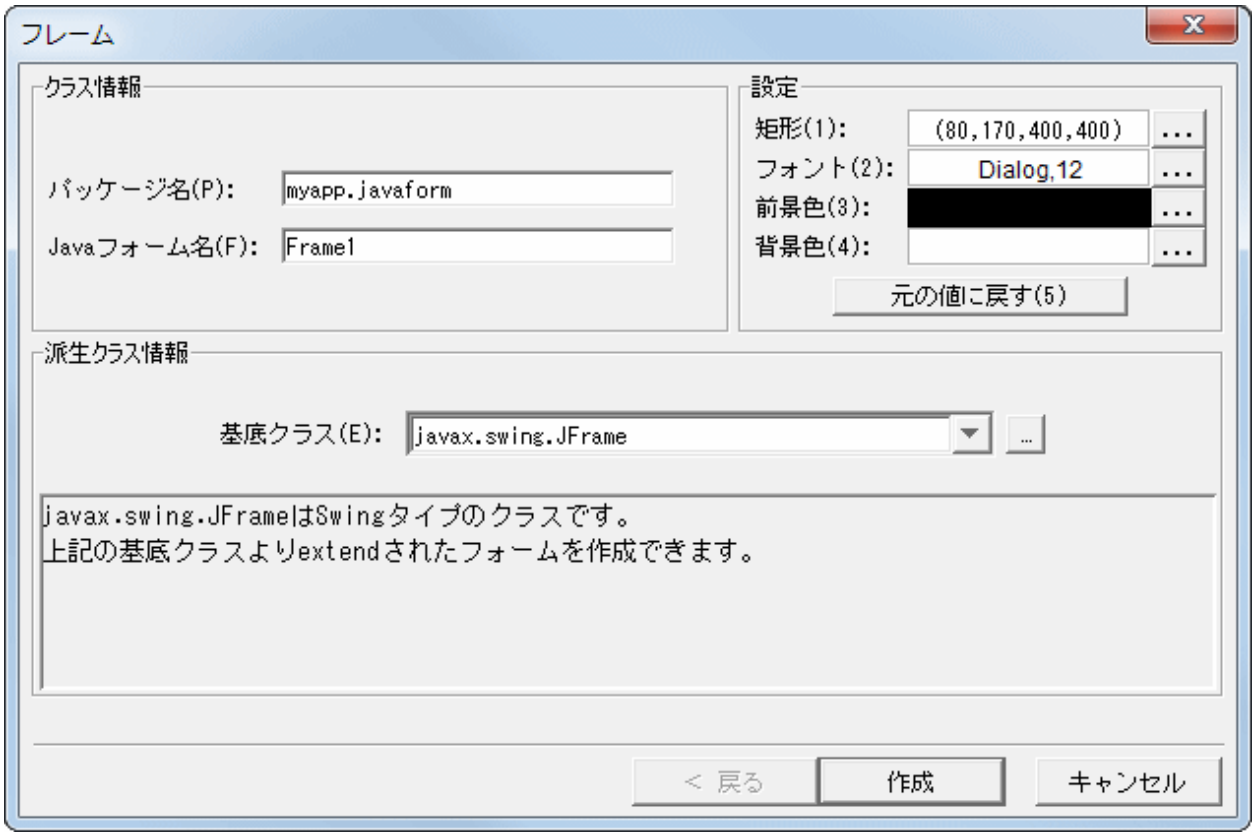
7. [Javaフォームの新規作成]ダイアログボックスが表示されます。
[フレーム]を選択します。



[OK]をクリックします。

8. [フレーム]ダイアログボックスが表示されます。
このページでは、作成するフレームの情報を指定します。
以下の情報を入力します。

設定項目	設定内容
パッケージ名	myapp.javaform
Javaフォーム名	Frame1
基底クラス	javax.swing.JFrame



[作成]をクリックします。

ポイント

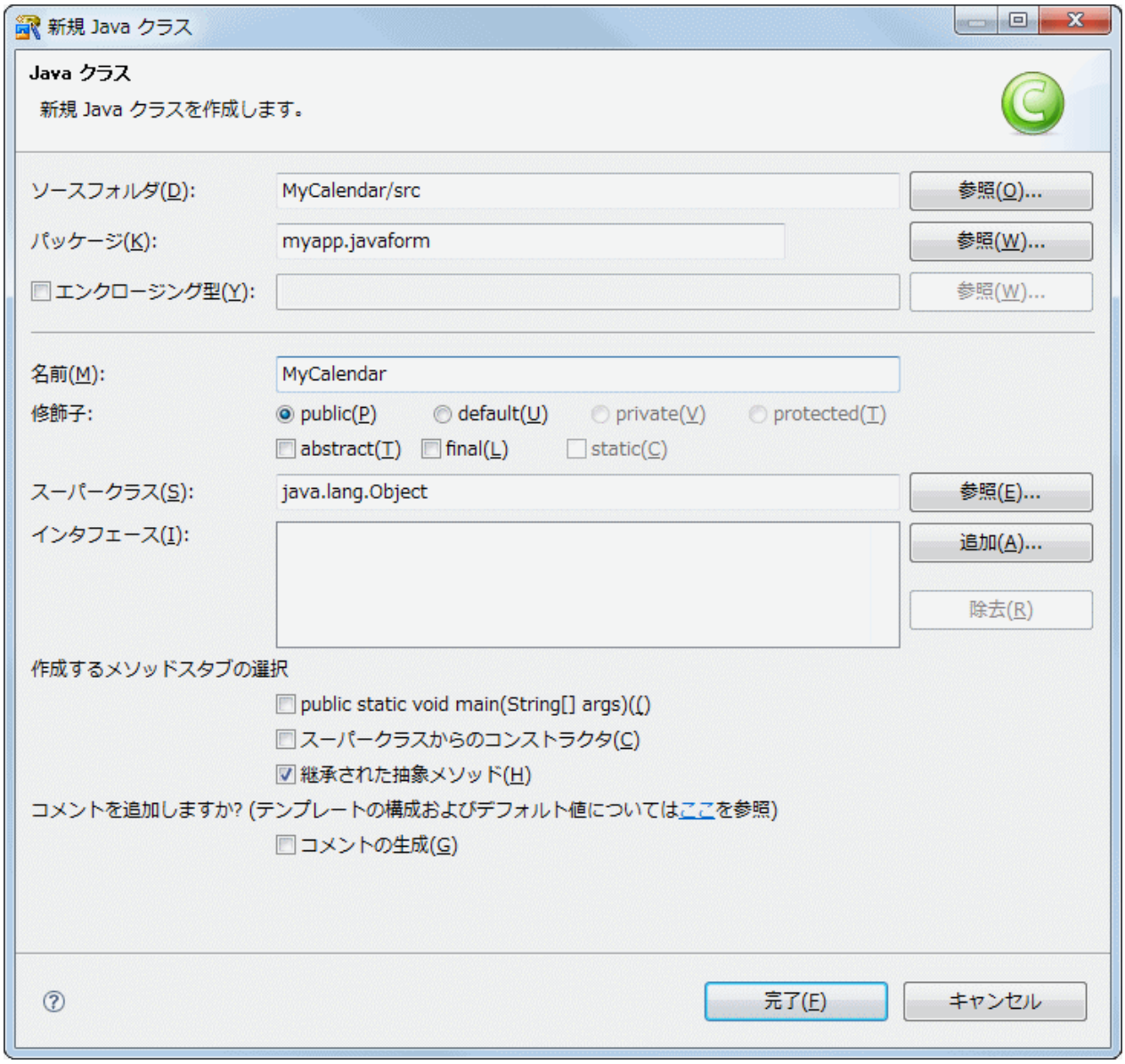
重量コンポーネントと軽量コンポーネント

コンテナやBeanの種類に、重量コンポーネント(AWT)と軽量コンポーネント(Swing)の二つがあります。Javaフォームやアプレットの作成では、できるだけ重量コンポーネントと軽量コンポーネントの利用を統一することを推奨します。重量コンポーネントと軽量コンポーネントを混在すると、重ね合わせの問題(必ず重量コンポーネントが上に表示される)やメニューのBeanの下への潜り込みなど、問題が発生する可能性があります。

9. 次に、フォームのクラスとフォームを表示する実行可能なクラスを作成します。
メニューバーから[ファイル] > [新規] > [クラス]を選択します。

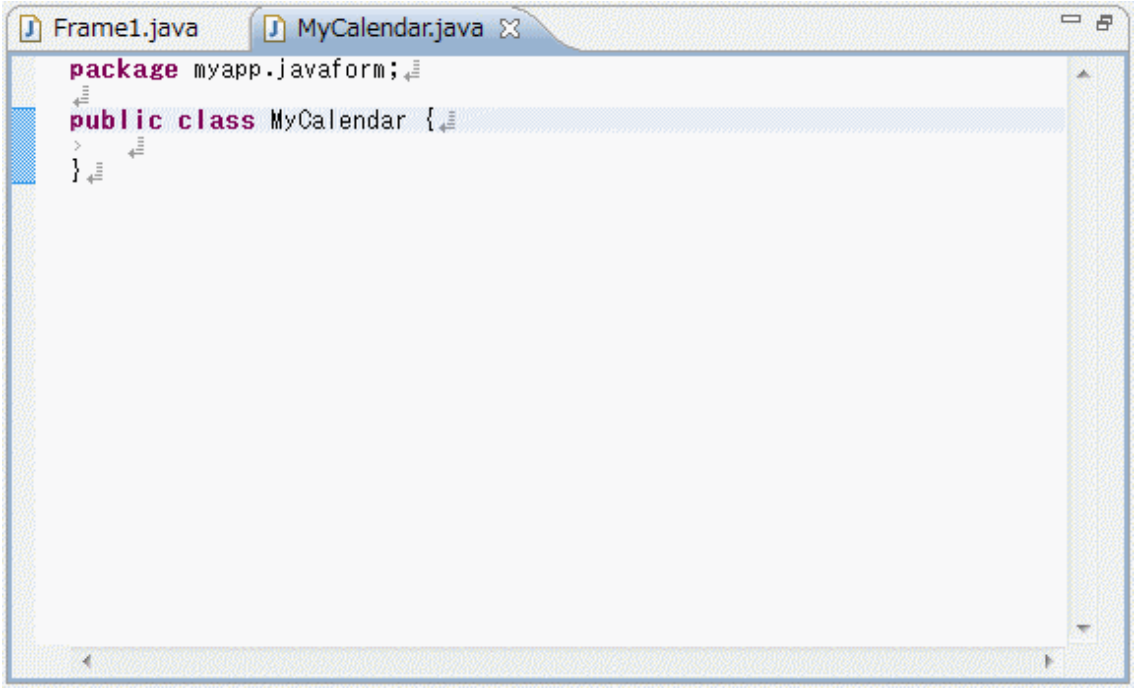
10. [Javaクラス]ページが表示されます。
このページでは、作成するJavaクラスの基本情報を指定します。
以下の情報を入力します。

設定項目	設定内容
ソースフォルダ	MyCalendar/src
パッケージ	myapp.javaform
名前	MyCalendar
継承された抽象メソッド	チェックする


新規 Java クラス
Java クラス
新規 Java クラスを作成します。
ソースフォルダ(D):
MyCalendar/src
参照(O)...
パッケージ(K):
myapp.javaform
参照(W)...
エンクロージング型(Y):
参照(W)...
名前(M):
MyCalendar
修飾子:
public(P)
default(U)
private(V)
protected(T)
abstract(T)
final(L)
static(C)
スーパークラス(S):
java.lang.Object
参照(E)...
インタフェース(I):
追加(A)...
除去(R)
作成するメソッドスタブの選択
public static void main(String[] args)(()
スーパークラスからのコンストラクタ(C)
継承された抽象メソッド(H)
コメントを追加しますか? (テンプレートの構成およびデフォルト値についてはここを参照)
コメントの生成(G)
完了(F)
キャンセル

[完了]をクリックします。

11. [Javaエディタ]に作成されたソースが表示されます。



作成されたソースを修正します。赤字の部分を追加します。

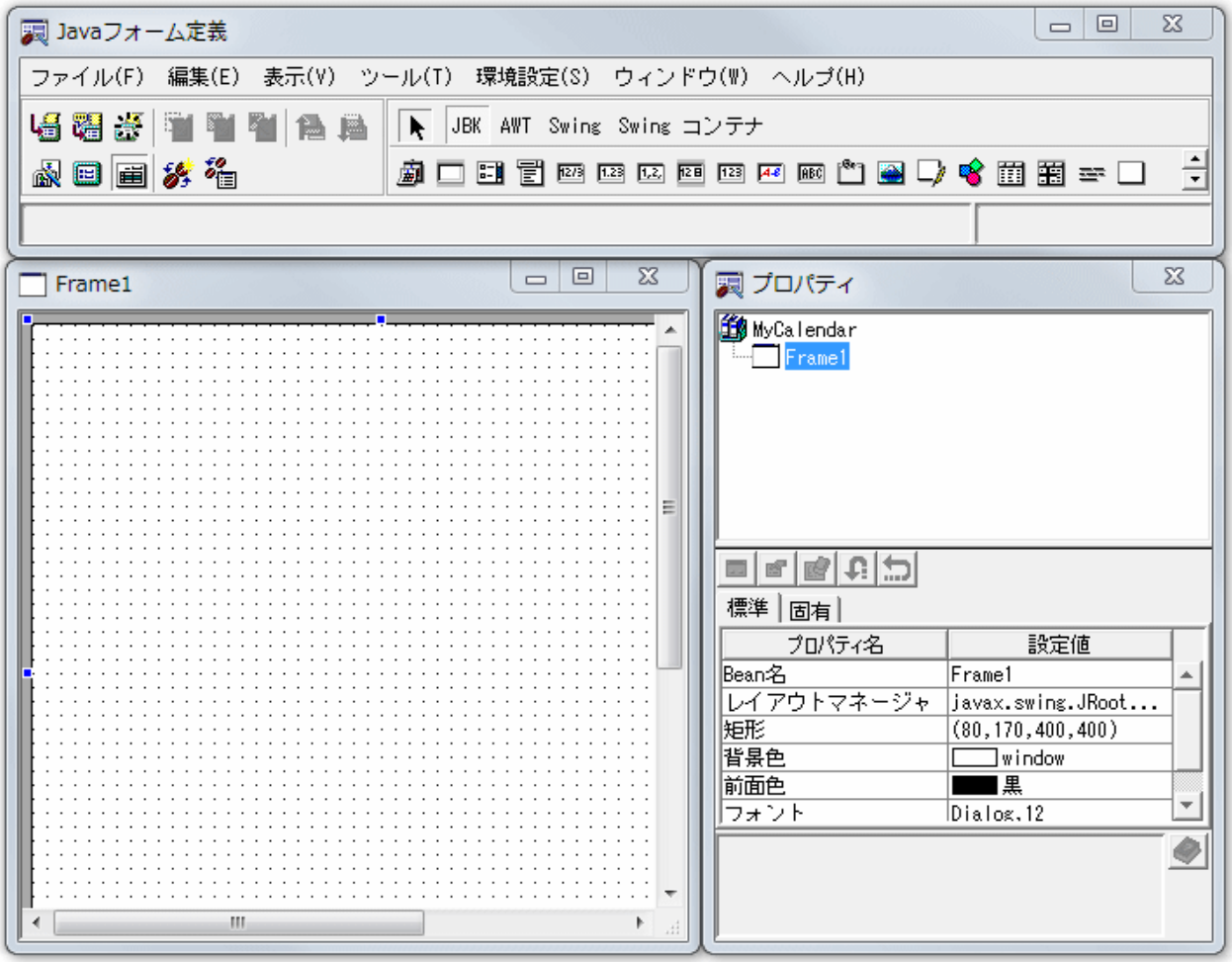
```
//コンストラクタ
public MyCalendar() {
}

public void run() {
    //新規フォームプロジェクトの作成
    Frame1 form = new Frame1();
    //フォームの表示
    form.setVisible(true);
}
//メイン処理
public static void main(String[] args) {
    // MyCalendarクラスのインスタンスを作成
    MyCalendar object = new MyCalendar();
    // MyCalendarクラスのインスタンスのrunメソッドを呼び出す
    object.run();
}
```

12. 以上の操作で、フォームのクラスとフォームを表示する実行可能なクラスが作成されます。

Javaフォームの定義

1. 画面の形やレイアウトをJavaのクラスで定義したものが、Javaフォームです。
   このJavaフォームを編集して完成させます。

2. Javaフォーム定義を使用して、Beanの配置、プロパティ(属性)の設定、処理手続きを記述し、フォームを完成させます。



3. Frame1のプロパティを変更します。
プロパティの参照および設定はプロパティウィンドウ内の「プロパティシート」で行います。
プロパティウィンドウが表示されていない場合は、Javaフォーム定義のメニューバーから[表示] > [プロパティ]を選択し、プロパティウィンドウを表示します。
プロパティシートには選択中のBeanのプロパティが一覧表示されます。選択中のBeanがない場合は、Javaフォームのプロパティが表示されます。
Frame1のプロパティを以下のように指定します。

プロパティ名	値
レイアウトマネージャ	＜なし＞
背景色	ライトグレイ

Beanの情報は、標準プロパティと固有プロパティに分かれて表示されています。
[固有]タブをクリックし、以下のように指定します。

プロパティ名	値
ウィンドウタイトル	カレンダー

カレンダーBeanの配置

1. ユーザとウィンドウの間で入出力を制御するのが、プッシュボタンやリストボックスなどのBeanです。
   ここでは、JBKのカレンダーBeanをJavaフォームに配置します。
   Beanの配置はマウスを使って行います。マウスを使って、Beanの新規配置、サイズの変更、移動などの編集操作を行います。
   オブジェクトパレットの[JBK]が選択されていない場合は、[JBK]をクリックします。

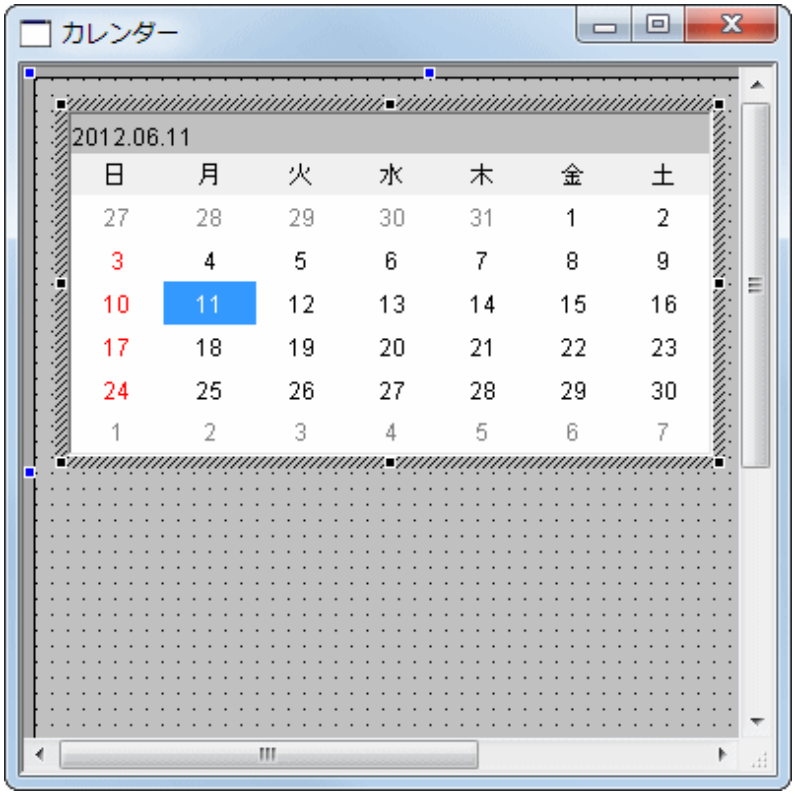
ポイント

オブジェクトパレット

オブジェクトパレットは、フォームにコンポーネント(Beanやコントロール)を配置するときに使うパレットです。パレット上の絵をクリックすることで、コンポーネントが選択できます。

2. オブジェクトパレットの[カレンダー]をクリックし選択します。

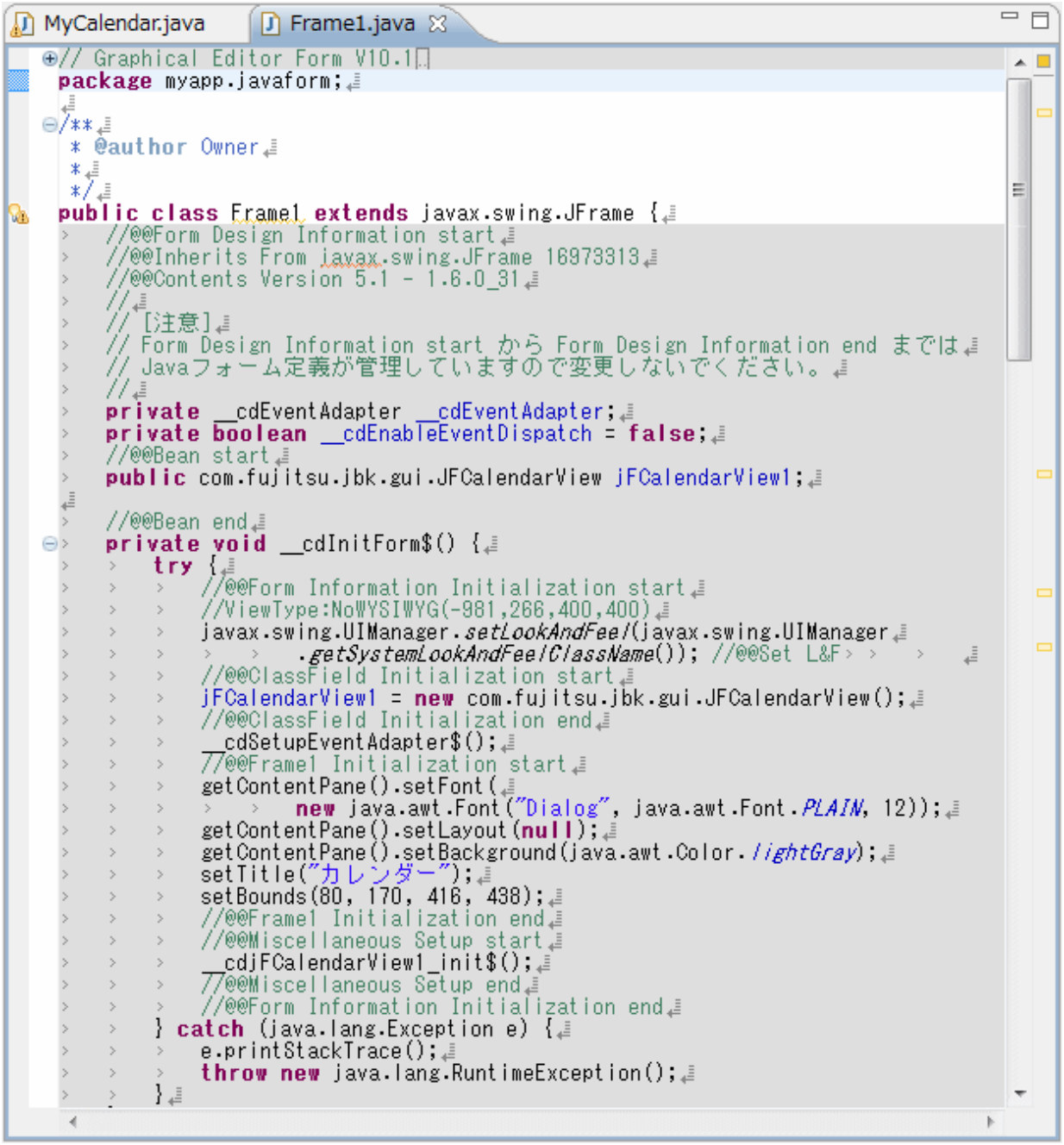
3. Javaフォーム上に配置します。
   編集対象のJavaフォームの貼り付けたい位置でマウスの左ボタンを押します。
   マウスをドラッグし、適当な大きさのところでマウスの左ボタンを離すことにより、フォームにBeanを貼り付けることができます。



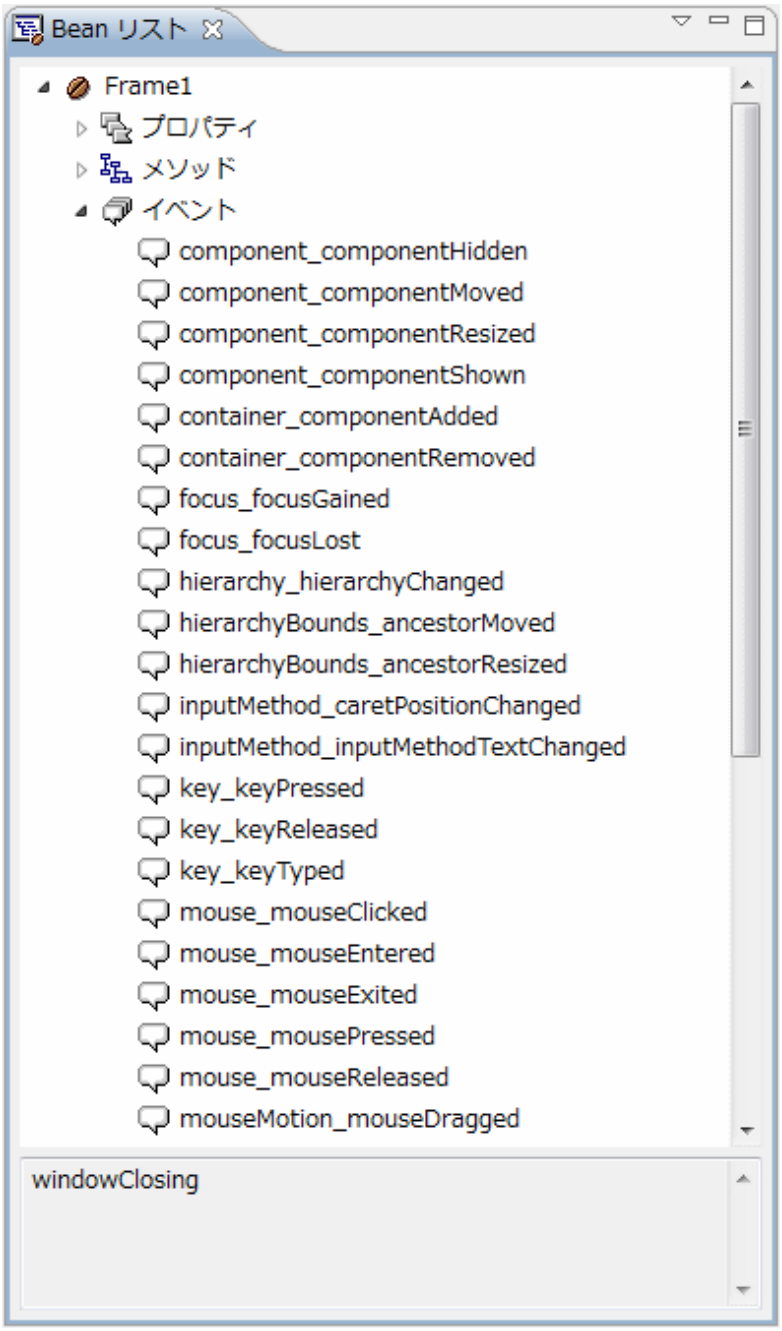
イベント処理の記述

1. マウスのクリックやキーボードからキーを入力するという操作に対応して発生する事象を、イベントといいます。
   イベントに対して記述された処理手続きを、イベント処理と呼びます。
   イベント処理は、Javaエディタを使用して記述します。

Javaフォーム定義のメニューバーから[表示] > [Javaエディタ]を選択し、Javaエディタをアクティブにします。Javaフォーム上や貼り付けたBean上でマウスをダブルクリックすることにより、Javaエディタをアクティブにすることもできます。

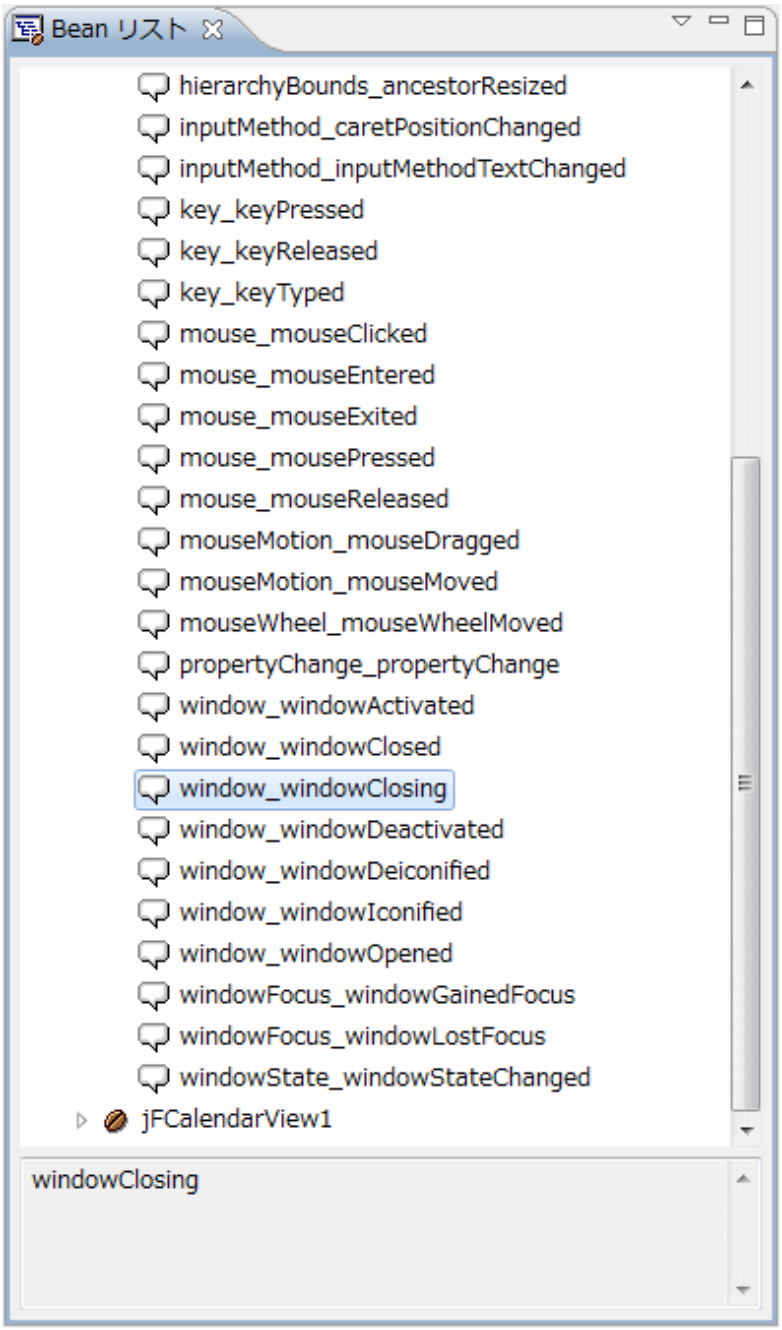


2. イベントが発生したときの処理を記述するため、[Beanリスト]ビューを表示します。
メニューバーから[ウィンドウ] > [ビューの表示] > [その他]を選択します。[ビューの表示]ダイアログボックスが表示されます。[Java] > [Beanリスト]を選択し、[OK]をクリックします。



3. フレームが閉じた場合に、アプリケーションを終了するように処理を記述します。
フレームが閉じた場合は、windowイベントが発生します。windowイベントに対応した「window_windowClosing」イベントに処理を記述します。

[Beanリスト]ビューにある[Frame1] > [イベント] > [window_windowClosing]をダブルクリックします。
イベント処理の作成確認画面が表示されるので、「はい」をクリックします。



4. 「Frame1_window_windowClosing」イベントに赤字の部分を記述します。

```
public void Frame1_window_windowClosing(java.awt.event.WindowEvent e) {
    if(!defaultEventProc(e)) {
        // ここにイベント発生時の処理を記述します。
        System.exit(0);
    }
}
```

ポイント

Javaフォーム定義が管理する変更禁止ソースについて

Javaフォームは、画面(Bean)の情報やイベント処理を呼び出すソースを含んでいます。このソースは、Javaフォーム定義が管理しているため変更禁止です。
青字のコメントで囲われた部分が、Javaフォーム定義が管理する変更禁止ソースになります。

```
public class Frame1 extends javax.swing.JFrame {
    //@@Form Design Information start

      ～Javaフォーム定義が管理する変更禁止ソース～

    //@@Form Design Information end
```

5. Javaフォームを保存します。
   ワークベンチのメニューバーから[ファイル] > [保存]を選択します。または、Javaフォーム定義のメニューバーから[ファイル] > [上書き保存]を選択します。
   ファイルの保存はワークベンチとJavaフォーム定義のどちらからでも行うことができます。
   Javaフォーム定義のメニューバーから [ファイル] > [終了]を選択し、Javaフォームを終了します。

ビルド

1. ビルドは、Javaファイルなどのリソースファイルを保存すると自動的に実行されます。このため、ビルドの手順は必要ありません。
   ビルドが正常に行われると「MyCalendar.jar」ファイルが作成されます。
   手動でビルドの設定にしている場合は、以下の手順で操作します。

2. ビルドするプロジェクトを選択します。
   ワークベンチの[パッケージエクスプローラ]ビューにある[MyCalendar]プロジェクトをクリックします。

3. ビルドを実行します。
   ワークベンチのメニューバーから[プロジェクト] > [プロジェクトのビルド]を選択します。

4. 結果を確認します。
   ビルドが終了するとエラーや警告などの情報を[問題]ビューに表示します。

ポイント

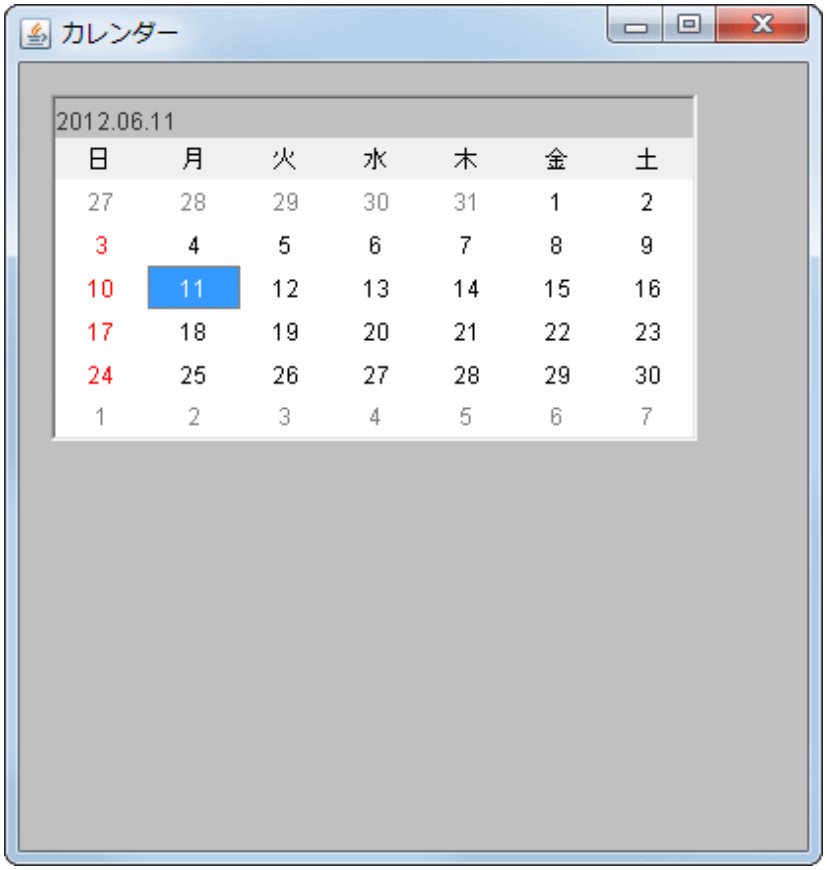
自動ビルドの設定について

自動的にビルドと手動でビルドとは、次の方法で変更できます。
メニューバーから[プロジェクト] > [自動的にビルド]を選択します。

実行

1. 実行するファイル(クラス)を選択します。ワークベンチの[パッケージエクスプローラ]ビューにある[MyCalendar] > [src] > [myapp.javaform] > [MyCalendar.java]をクリックします。

2. メニューバーから[実行] > [実行] > [Javaアプリケーション]を選択します。アプリケーションが起動し、Javaフォームが表示されます。



3. カレンダーBeanの動作を確認します。
   カレンダーBeanは、カレンダーの表示と日付の選択を行う機能を持っています。
   マウスでクリックすると、クリックした日付を選択します。
   カーソルキーや、[PageUp]キー、[PageDown]キーを押すことにより、日/週/月単位に選択表示を変更することができます。

4. タイトルバーのクローズボタン([X]ボタン)をクリックして、アプリケーションを終了します。

F.2.2 Lesson2 入出力フィールド

Lesson1で作成したJavaフォームにJBKの入出力フィールドBeanを追加し、現在の日付からカレンダーで選択された日までの日数を表示するようにします。また、入出力フィールドに入力された日数にしたがってカレンダーの表示も切り替わるように変更します。
入出力フィールドBeanには以下の種類があります。

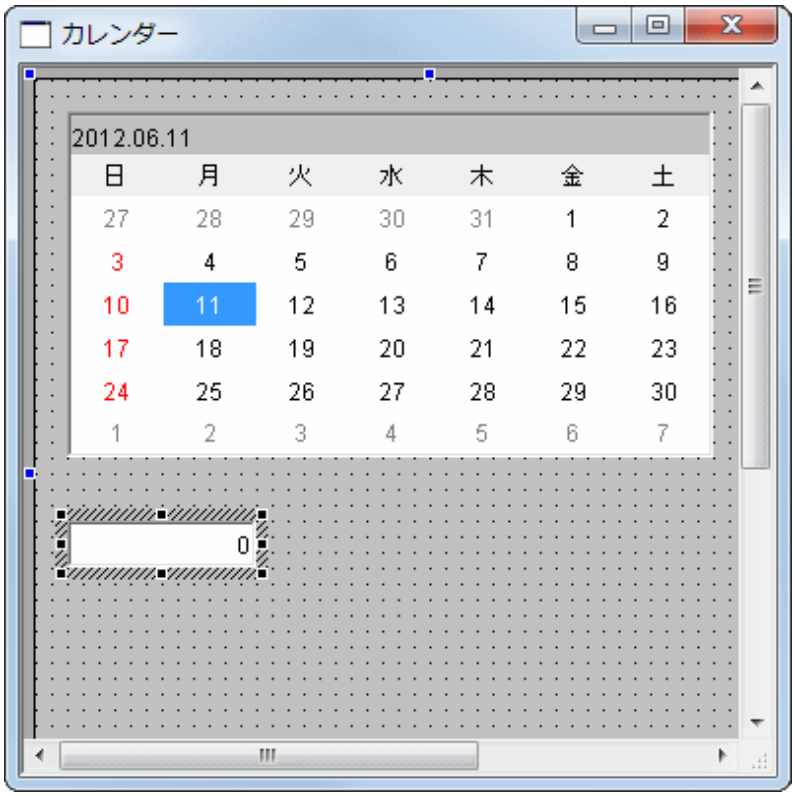
- 日付／時間フィールド(com.fujitsu.jbk.gui.JFFieldDate)
- 実数フィールド(com.fujitsu.jbk.gui.JFFieldDouble)
- 埋め込み文字列フィールド(com.fujitsu.jbk.gui.JFFieldFilled)
- 整数フィールド(com.fujitsu.jbk.gui.JFFieldLong)
- 文字列フィールド(com.fujitsu.jbk.gui.JFFieldString)

ここでは、数値の入出力用に使用したいので整数フィールドBeanを使用します。

整数フィールドBeanの配置

1. Lesson1で作成したプロジェクト「MyCalendar」をワークベンチで開きます。
   [Frame1.java]をJavaフォーム定義で開きます。ワークベンチの[パッケージエクスプローラ]ビューにある[MyCalendar] > [src] > [myapp.javaform] > [Frame1.java]を右クリックしてコンテキストメニューを表示します。そして[アプリケーションから開く] > [グラフィカルエディタ]を選択します。

2. オブジェクトパレットの[JBK]が選択されていない場合は、[JBK]をクリックします。
3. オブジェクトパレットの[整数フィールド]をクリックし選択します。
4. マウスを使ってJavaフォーム上でドラッグして配置します。



イベント処理の記述

1. イベント処理は、Javaエディタを使用して記述します。
   Javaフォーム定義のメニューバーから[表示] > [Javaエディタ]を選択し、Javaエディタをアクティブにします。Javaフォーム上や貼り付けたBean上でマウスをダブルクリックすることにより、Javaエディタをアクティブにすることもできます。
2. JDKのDateクラスを使用するので、importキーワードを追加します。
   DateクラスはJDKの「java.util」パッケージに含まれているので、以下のように記述します。

```
import java.util.Date;
```

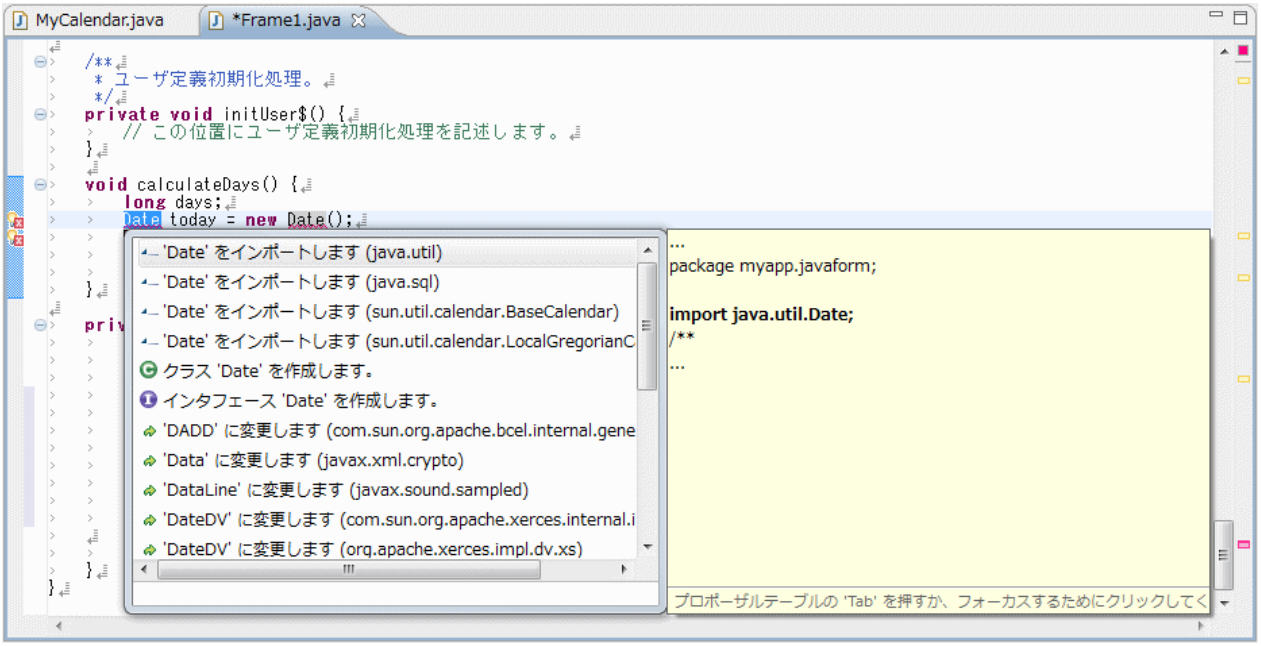
3. 現在の日付からカレンダーBeanで選択されている日付までの日数を計算し、整数フィールドBeanに表示する処理を記述します。
   この処理は、他のイベント処理からも呼び出せるようにFrame1クラスのメソッドとして作成します。
   赤字の部分を記述します。

```
/*
 * ユーザ定義初期化処理。
 */
protected void initUser$() {
    // この位置にユーザ定義初期化処理を記述します。
}

void calculateDays() {
    long days;
    Date today = new Date();
    Date selectday = jFCalendarView1.getDate().getTime();
    days = selectday.getTime() / 86400000 - today.getTime() / 86400000;
    jFFieldLong1.setValue(days);
}
```

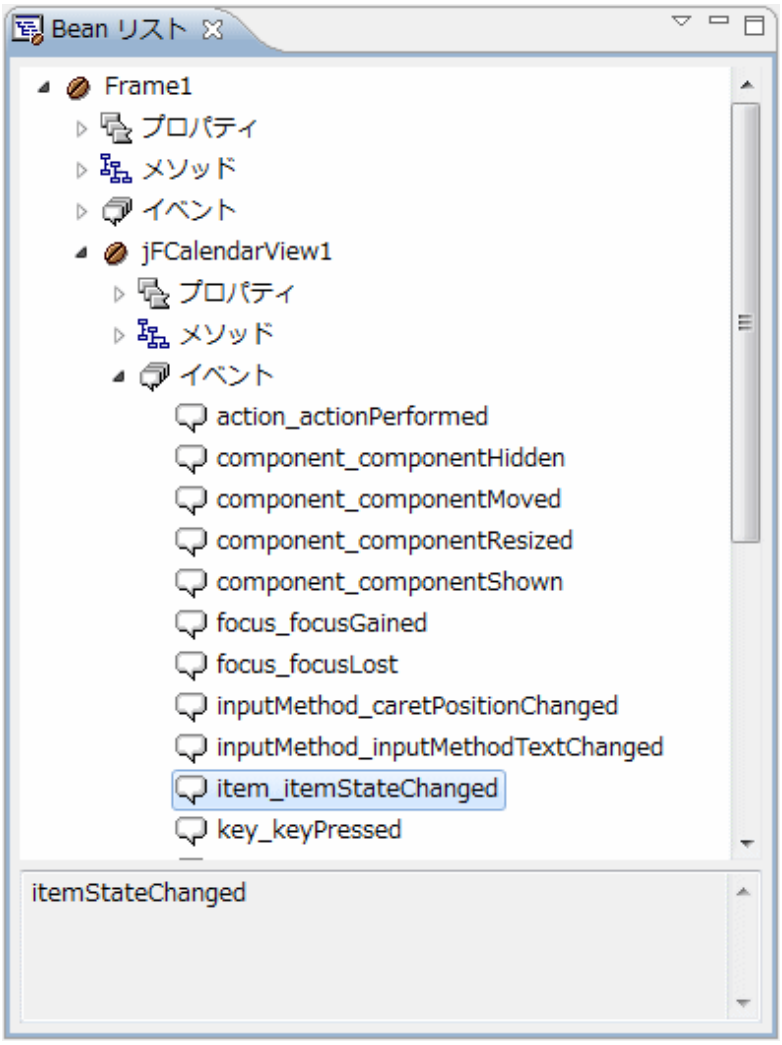
ポイント

Javaエディタのコンテンツアシストについて

本文(任意)Javaエディタにはimportキーワードの不足やクラス、メソッドなどの綴りの間違いを表示して、修正候補を表示する機能があります。使用しているクラスのimportキーワードが不足している場合には、問題のあるクラスに波線が引かれ、垂直ルーラに電球のアイコンが表示されます。この電球アイコンをクリックするとimportできるパッケージや修正の候補が表示されます。この変換候補から[java.util.Dateのインポート]を選択することでもimportキーワードを追加することができます。



4. イベントが発生したときの処理を記述するため、[Beanリスト]ビューを表示します。
メニューバーから[ウィンドウ] > [ビューの表示] > [その他]を選択します。[ビューの表示]ダイアログボックスが表示されます。[Java] > [Beanリスト]を選択し、[OK]をクリックします。



5. カレンダーの日付が選択された場合に、calculateDaysメソッドを呼び出すように処理を記述します。
日付が選択された場合はitemイベントが発生します。itemイベントに対応した「item_itemStateChanged」イベントに処理を記述します。
[Beanリスト]ビューにある[Frame1] > [jFCalendarView1] > [イベント] > [item_itemStateChanged]をダブルクリックします。
イベント処理の作成確認画面が表示されるので、「はい」をクリックします。

6. 「jFCalendarView1_item_itemStateChanged」イベントに赤字の部分を記述します。

```
public void jFCalendarView1_item_itemStateChanged(java.awt.event.ItemEvent e) {
    if(!defaultEventProc(e)) {
        // ここにイベント発生時の処理を記述します.
        calculateDays();
    }
}
```

7. 整数フィールドに値が入力された場合は、入力された値を現在の日付からの経過日数としてカレンダーの選択を変更するように処理を記述します。
整数フィールド上で[Enter]キーを押した場合は actionイベントが発生します。actionイベントに対応した「action_actionPerformed」イベントに処理を記述します。
[Beanリスト]ビューにある[Frame1] > [jFFiledLong1] > [イベント] > [action_actionPerformed]をダブルクリックします。
イベント処理の作成確認画面が表示された場合は、[はい]をクリックします。

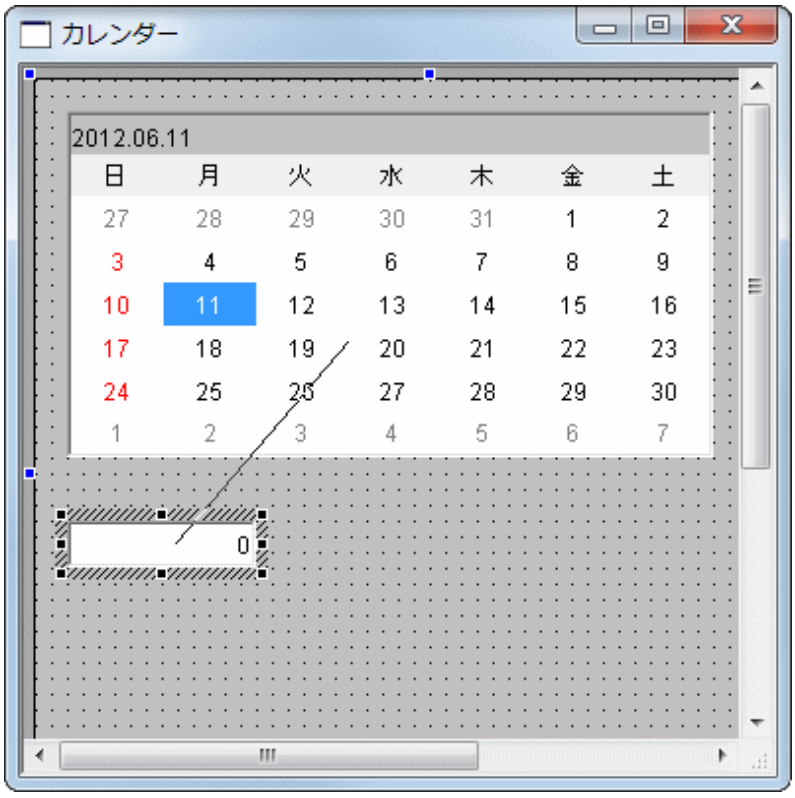
8. 「jFFiledLong1_action_actionPerformed」イベントに**赤字**の部分を記述します。

```
public void jFFieldLong1_action_actionPerformed(java.awt.event.ActionEvent e) {
    if(!defaultEventProc(e)) {
        // ここにイベント発生時の処理を記述します.
        jFCalendarView1.moveToday();
        jFCalendarView1.moveDate((int) jFFieldLong1.getValue());
    }
}
```

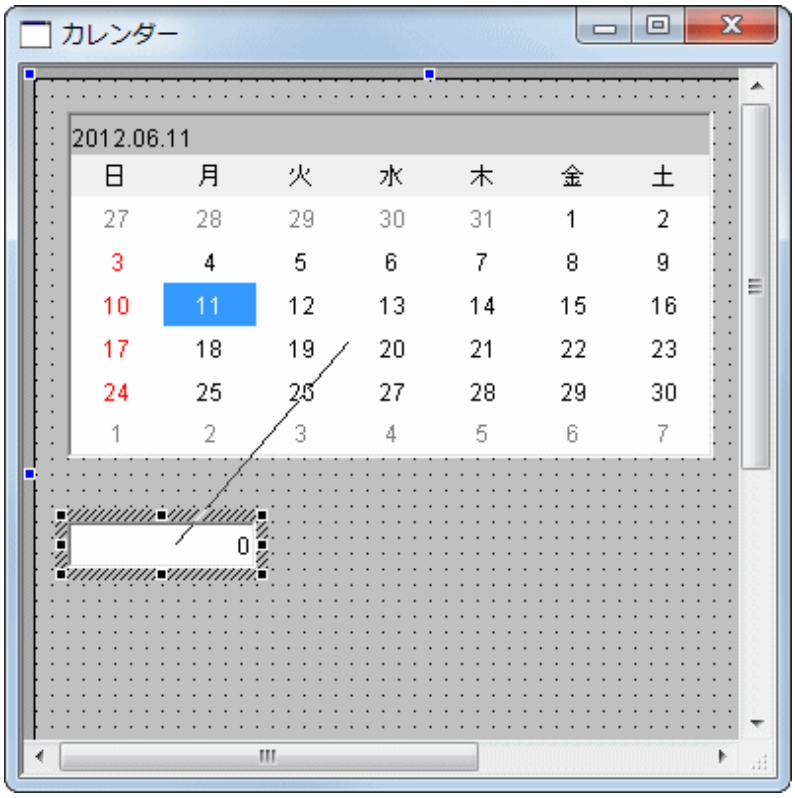
ポイント

JavaBeansのプロパティ・メソッドを使用するソースコードが対話的に作成できます。

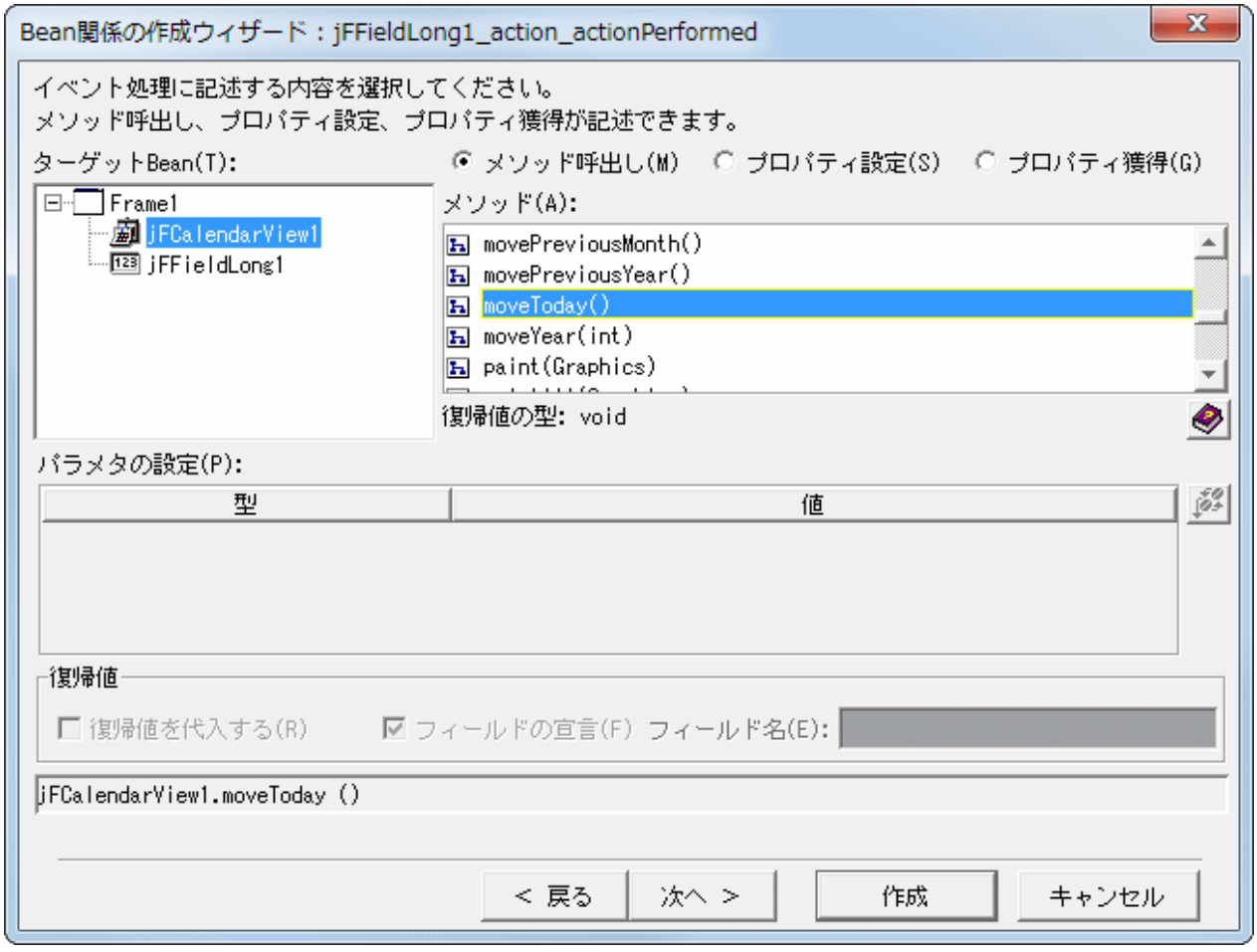
Interstage Studioでは、イベント処理メソッド内でJavaBeansのプロパティやメソッドを使用するソースコードの作成を支援するために「Bean関係の作成機能」を提供しています。Bean関係の作成機能を利用することにより、効率的にイベント処理を開発することができます。
上の例では、**赤字**の部分をJavaエディタで記述しましたが、「**jFCalendarView1.moveToday();**」はイベント処理メソッド内にあるJavaBeansのメソッド呼出しソースコードなのでBean関係の作成機能が使用できます。
まずjFFieldLong1上でマウスの右ボタンをクリックし、そのままjFCalendarView1までドラッグしてからボタンを離します。この操作をワイヤリングといいます。



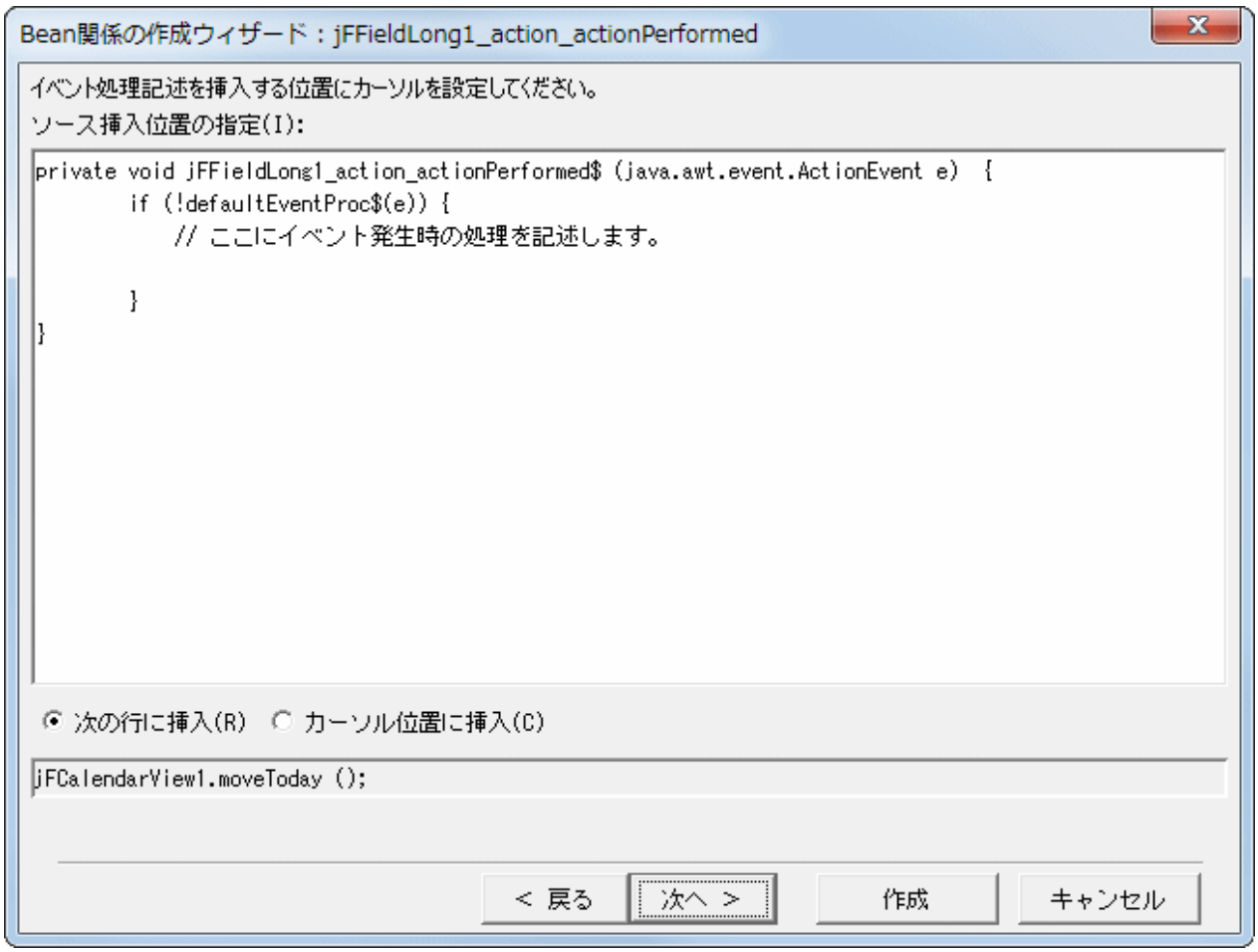
Bean関係の作成ウィザードが表示されます。ソースBeanリストでは、ワイヤリング操作時に始点となったBeanが既に選択されています。ソースイベントリストでactionイベントに対応した[action_actionPerformed]を選択して、[次へ]をクリックします。



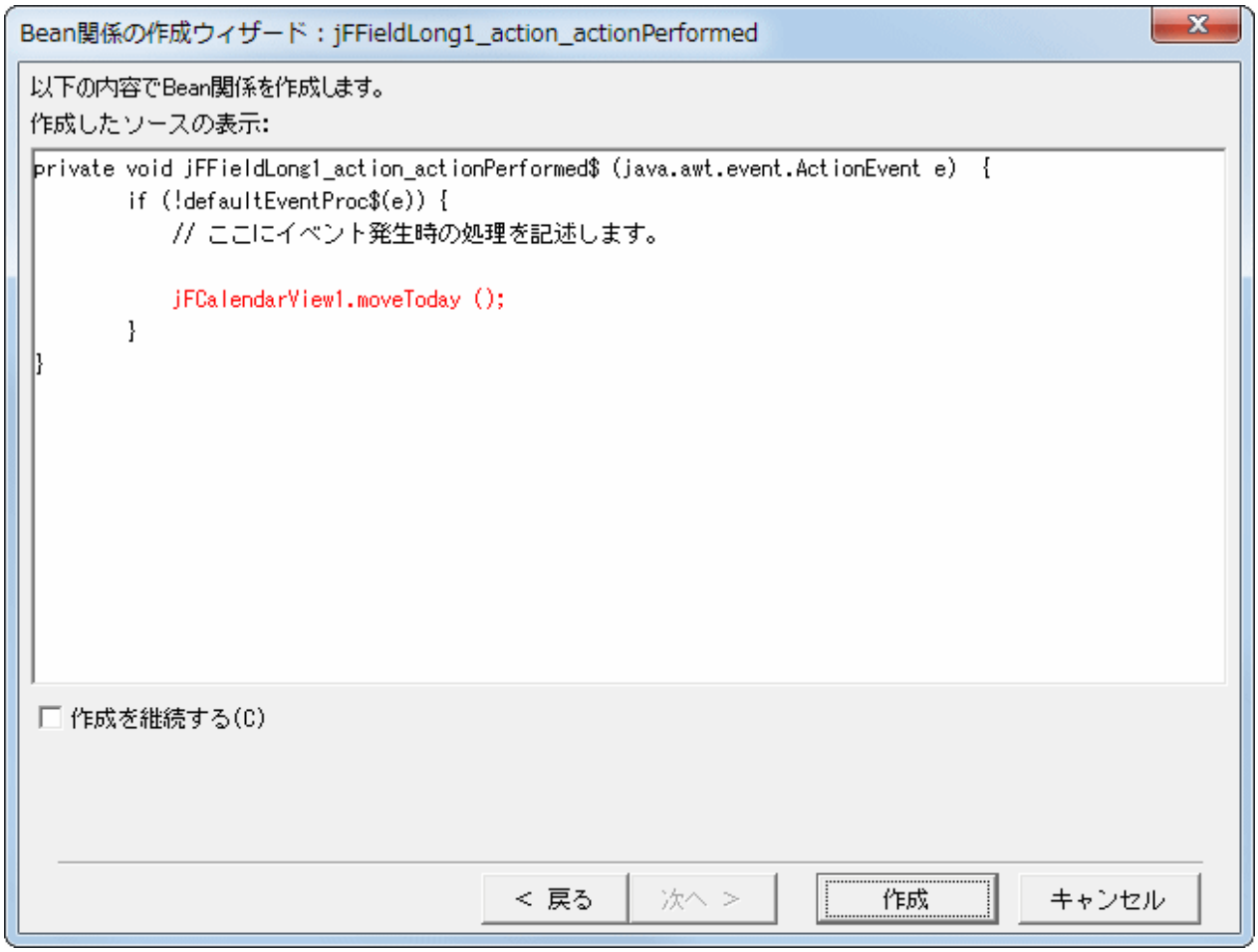
ターゲットBeanリストでは、ワイヤリング操作時に終点となったBeanが既に選択されています。Bean関係の種別として[メソッド呼出し]を選択し、メソッドリストから[moveToday()]を選択します。[次へ]をクリックします。



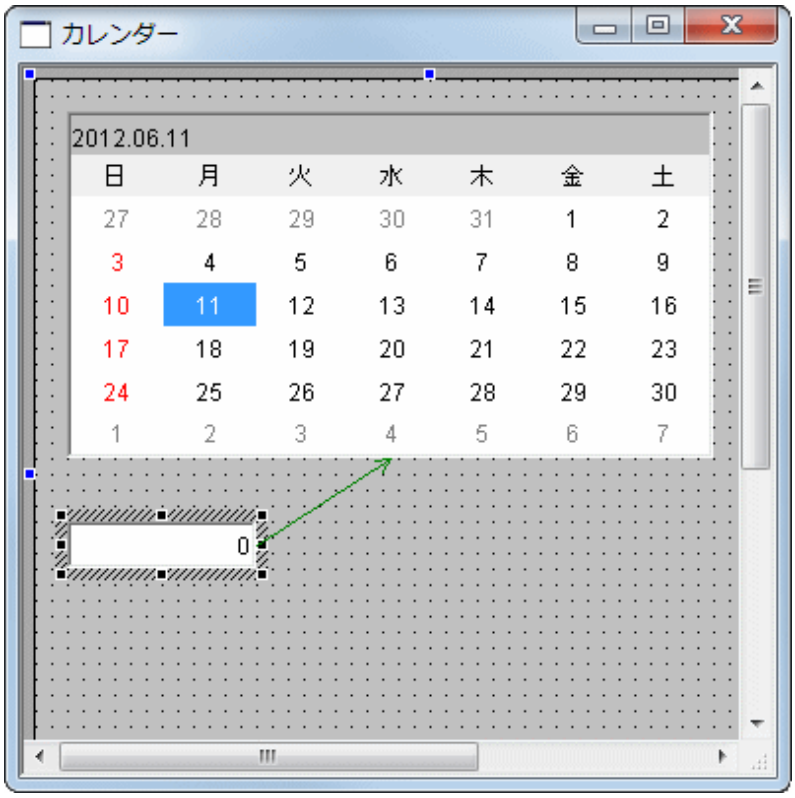
ソースコードの挿入位置を選択し、[次へ]をクリックします。



ソース挿入後のイベント処理メソッド全体のイメージが表示されます。**赤字**の部分がウィザードにより作成されるソースコードです。ソースの内容および挿入位置を確認し、[作成]をクリックします。



イベント処理メソッド内でJavaBeansのプロパティやメソッドを使用するソースコードはBean同士をつないだ線として画面に表示されます。



9. Javaフォームを保存します。
   ワークベンチのメニューバーから[ファイル] > [保存]を選択します。
   Javaフォーム定義のメニューバーから[ファイル] > [終了]を選択し、Javaフォームを終了します。

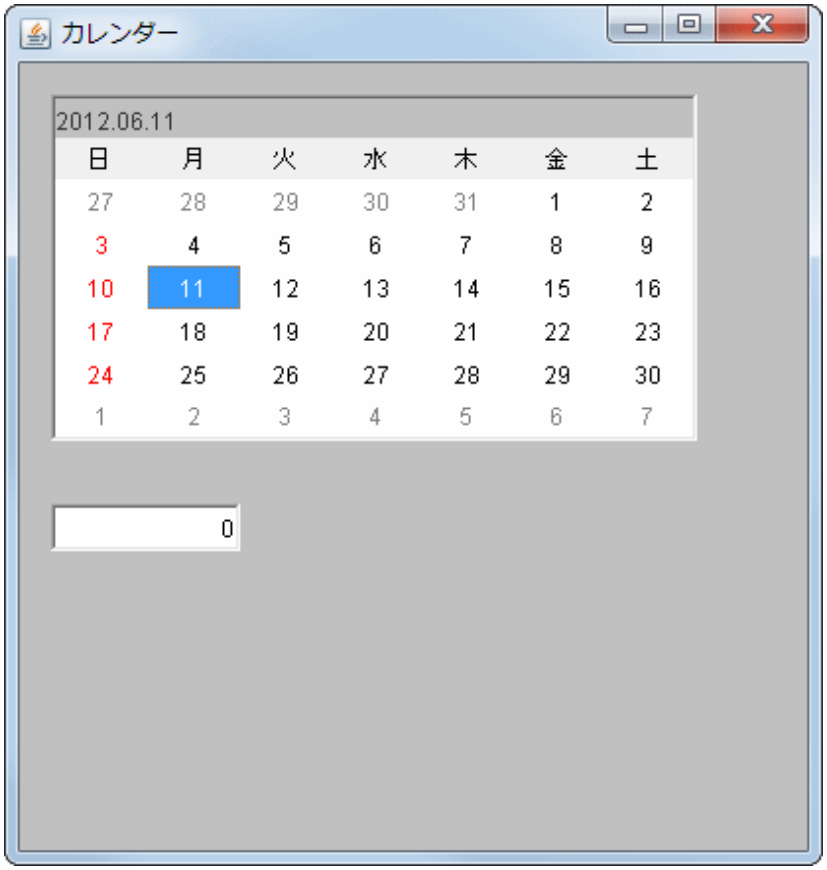
ビルド

1. ビルドは、Javaファイルなどのリソースファイルを保存すると自動的に実行されます。ビルドが正常に行われると「MyCalendar.jar」ファイルが作成されます。

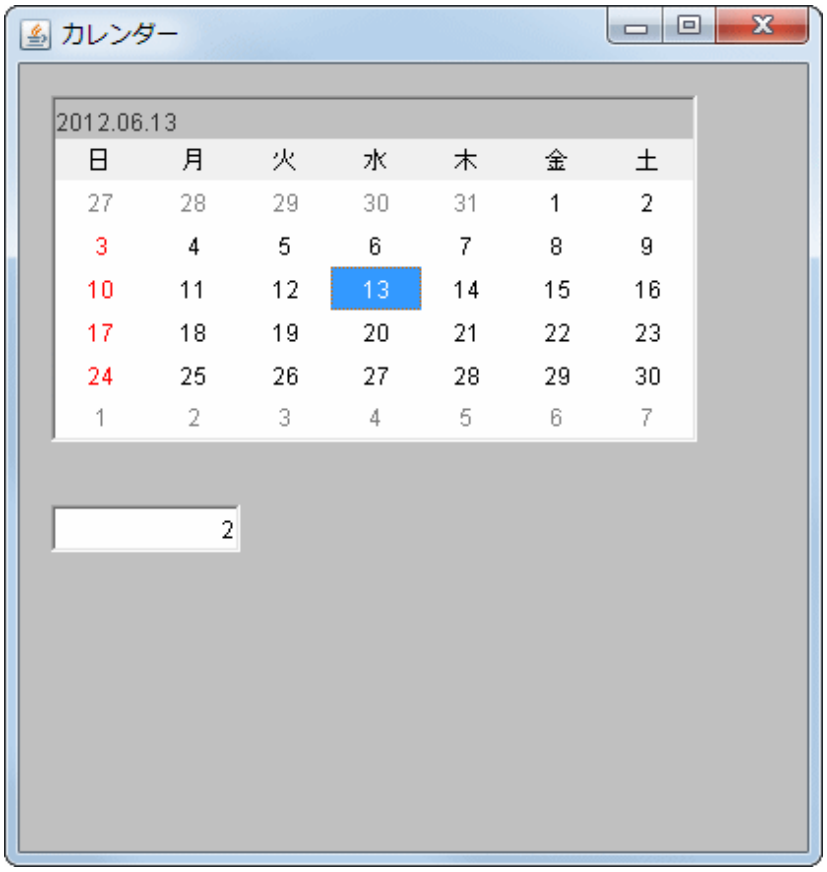
実行

1. 実行するファイル(クラス)を選択します。 ワークベンチの[パッケージエクスプローラ]ビューにある[MyCalendar] > [src] > [myapp.javaform] > [MyCalendar.java]をクリックします。
2. メニューバーから[実行] > [実行] > [Javaアプリケーション]を選択します。
   アプリケーションが起動し、Javaフォームが表示されます。

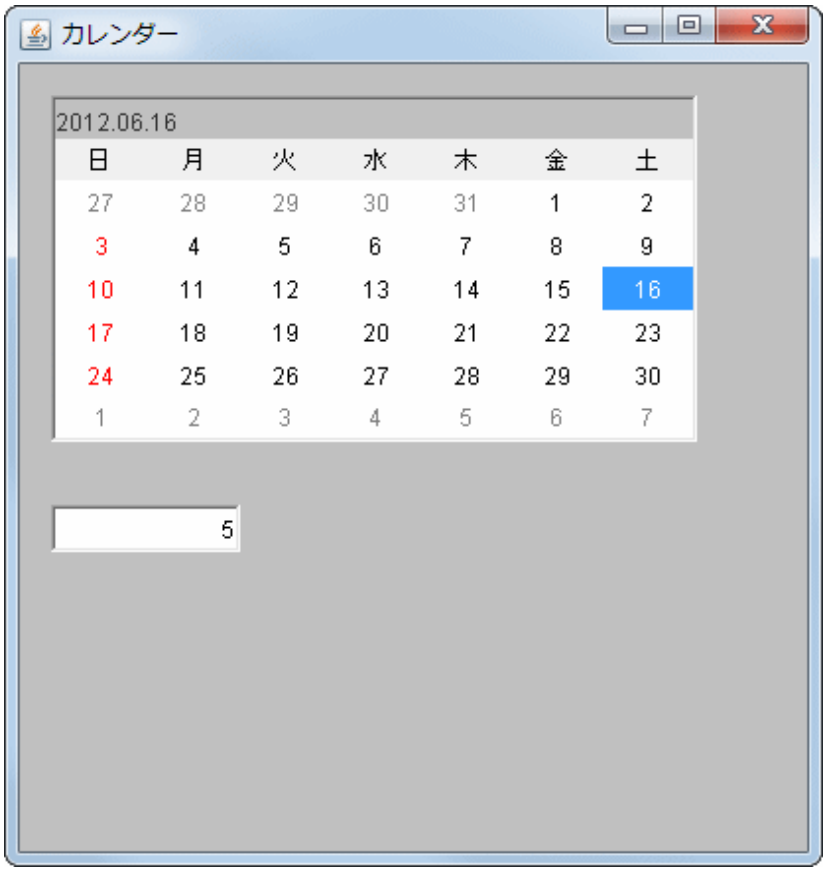
3. 選択されている日付と異なる日付をマウスでクリックします。
   整数フィールドに経過日数が表示されます。



「2日後」を選択します。


カレンダー
2012.06.13
日 月 火 水 木 金 土
27 28 29 30 31 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
1 2 3 4 5 6 7
2

4. 整数フィールドに数字を入力し、[Enter]キーを押します。
   カレンダーの表示が変わることを確認します。
   整数フィールドに「5」を入力します。



5. タイトルバーのクローズボタン([X]ボタン)をクリックして、アプリケーションを終了します。

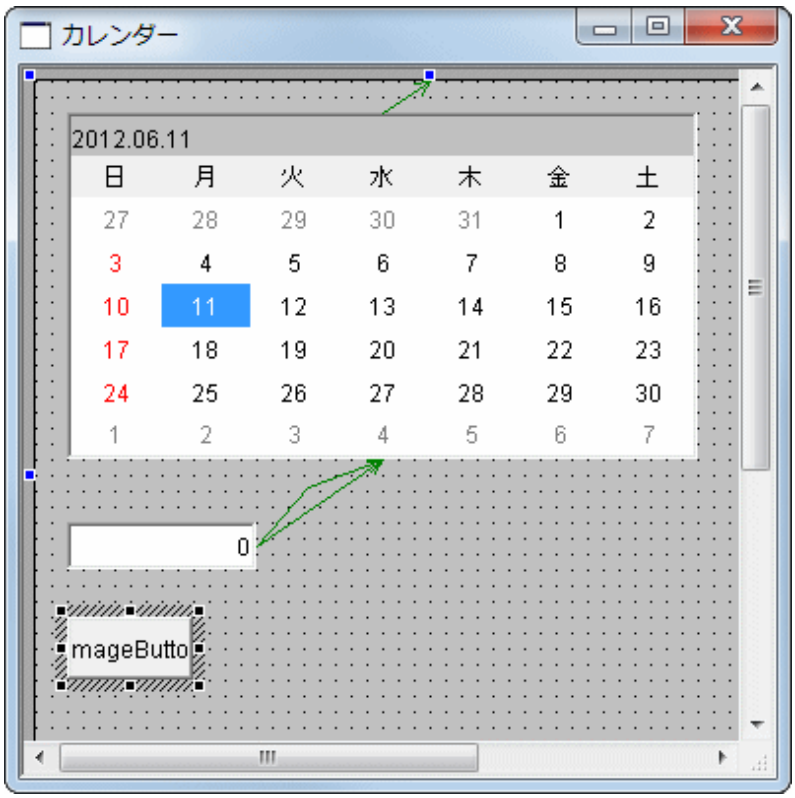
F.2.3 Lesson3 ボタン

Lesson2で作成したJavaフォームにJBKのイメージボタンBeanを追加し、ボタンを押すことによりカレンダーの表示が変更できるようにします。

ボタンBeanの配置

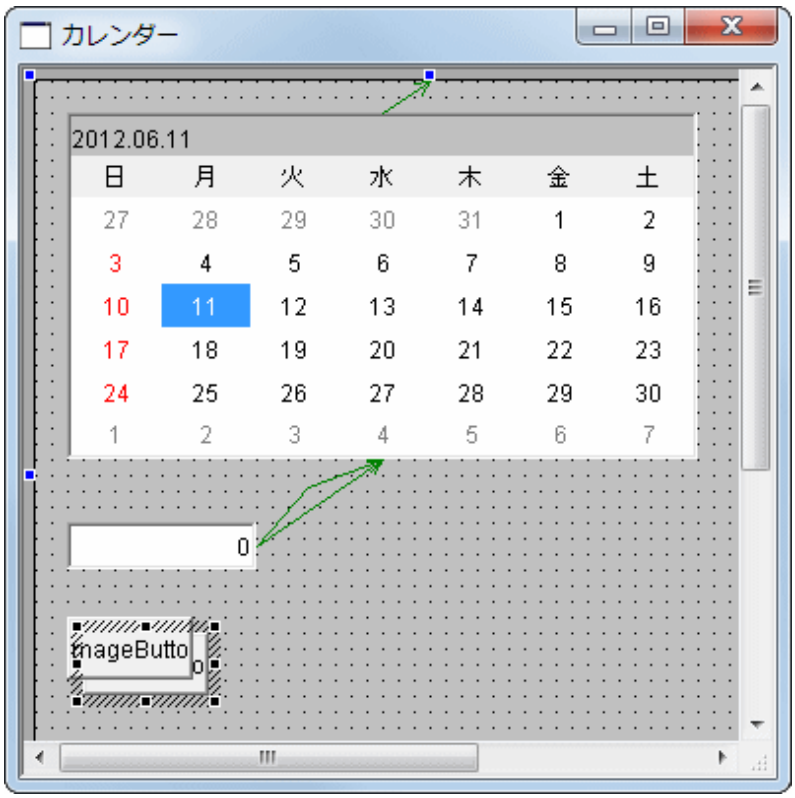
1. Lesson2で作成したプロジェクト「MyCalendar」をワークベンチで開きます。
   [Frame1.java]をJavaフォーム定義で開きます。ワークベンチの[パッケージエクスプローラ]ビューにある[MyCalendar] > [src] > [myapp.javaform] > [Frame1.java]を右クリックしてコンテキストメニューを表示します。そして[アプリケーションから開く] > [グラフィカルエディタ]を選択します。
2. オブジェクトパレットの[JBK]が選択されていない場合は、[JBK]をクリックします。
3. オブジェクトパレットの[イメージボタン]をクリックし選択します。

4. マウスを使ってJavaフォーム上でドラッグして配置します。

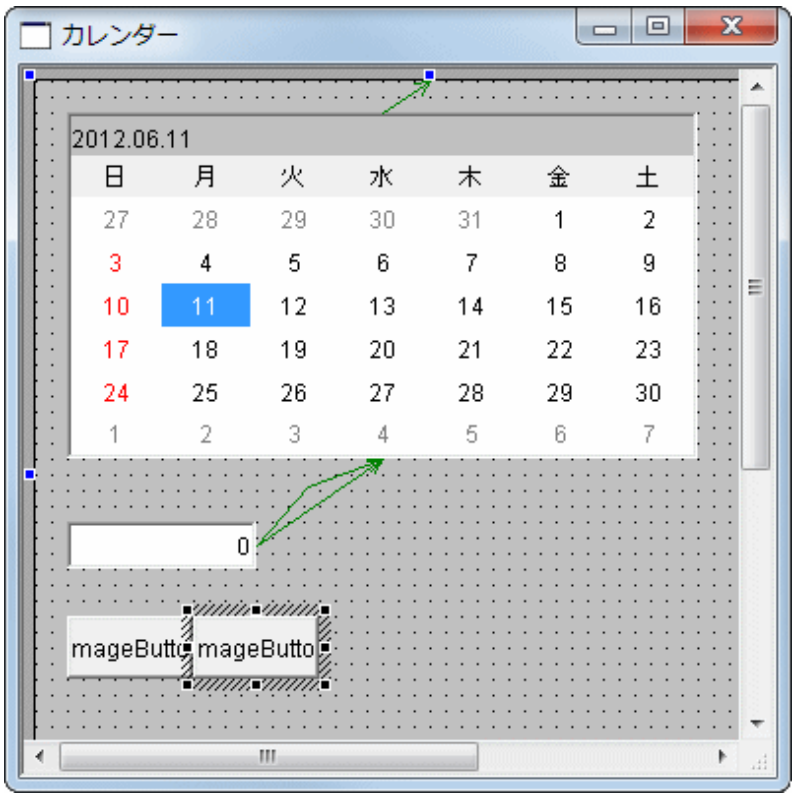


Beanを選択している状態で、Javaフォーム定義のメニューバーから[編集] > [コピー]を選択します。

メニューバーから[編集] > [貼り付け]を選択します。



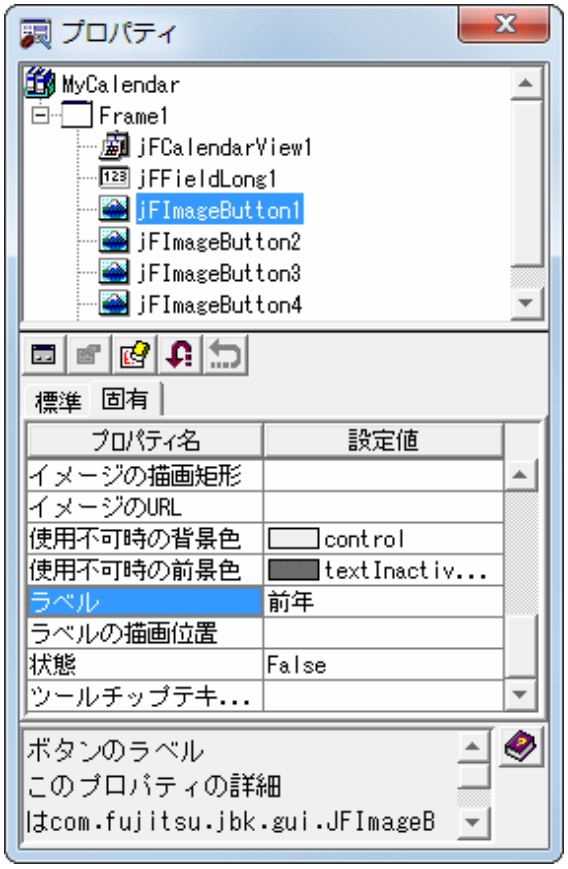
貼り付けたBeanをドラッグして、Beanの位置を変更します。



同様な操作でボタンBeanを全部で5個配置し、下表のようにラベルプロパティの変更、イベント処理を記述します。

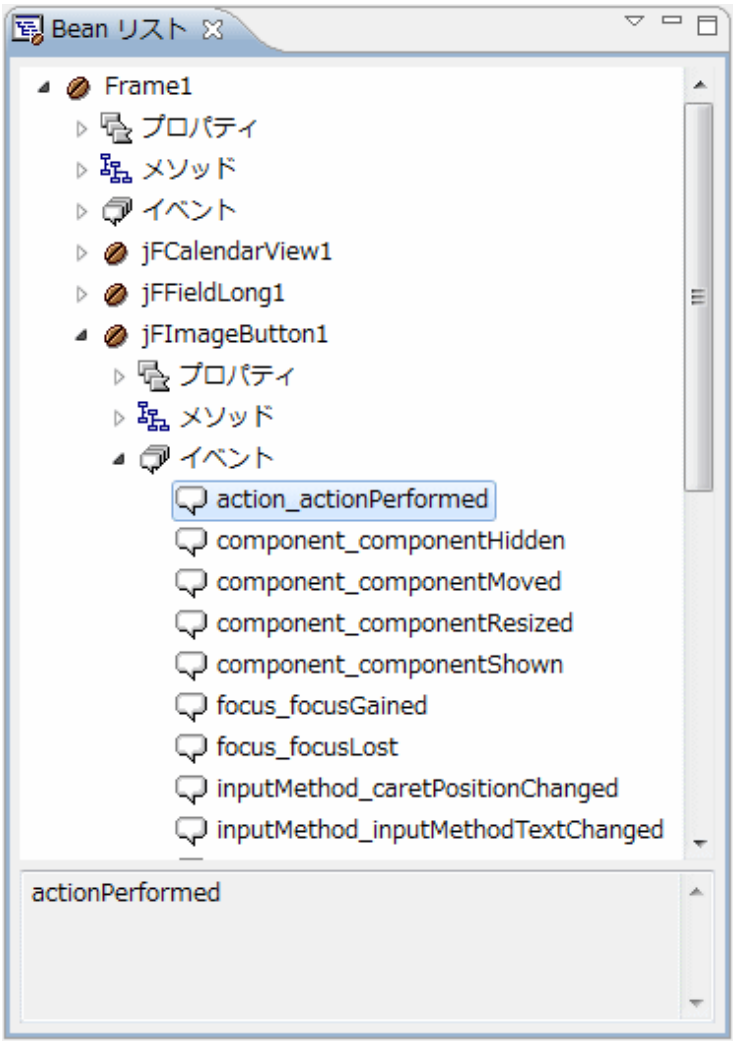
Bean名	ラベルプロパティの値	actionイベント(action_actionPerformed)の処理
jFImageButton1	前年	jFCalendarView1.moveYear(-1); calculateDays();
jFImageButton2	前月	jFCalendarView1.moveMonth(-1); calculateDays();
jFImageButton3	今日	jFCalendarView1.moveToday(); calculateDays();
jFImageButton4	翌月	jFCalendarView1.moveMonth(1); calculateDays();
jFImageButton5	翌年	jFCalendarView1.moveYear(1); calculateDays();

プロパティの変更は、プロパティシートを使って行います。jFImageButton1のラベルプロパティを変更する場合は、以下のようにします。

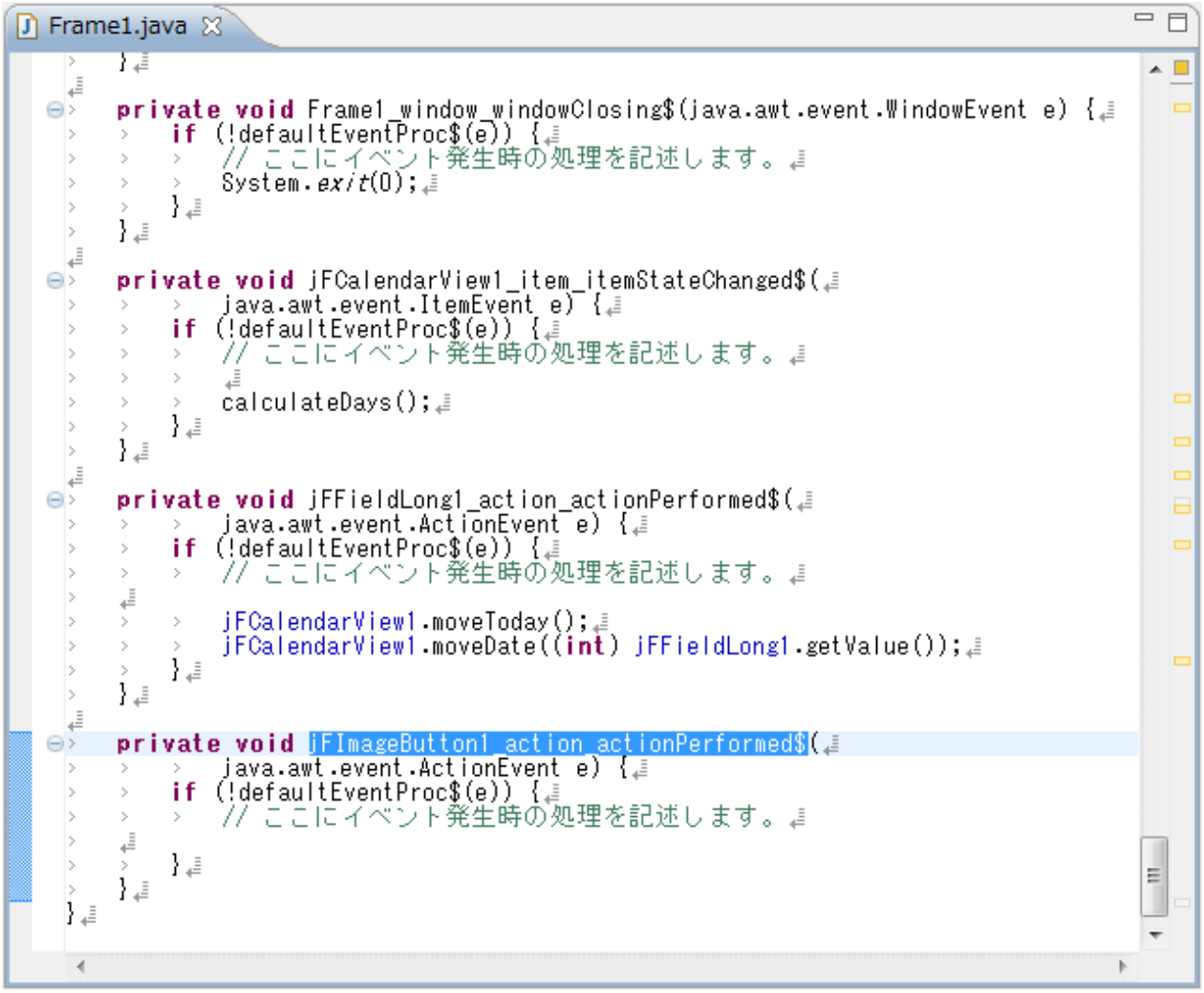


イベント処理の記述は、Javaエディタを使って、それぞれのボタンの「action_actionPerformed」に処理を記述します。
jFImageButton1の「action_actionPerformed」に処理を記述する場合は、[Beanリスト]ビューにある[Frame1] > [jFImageButton1] >

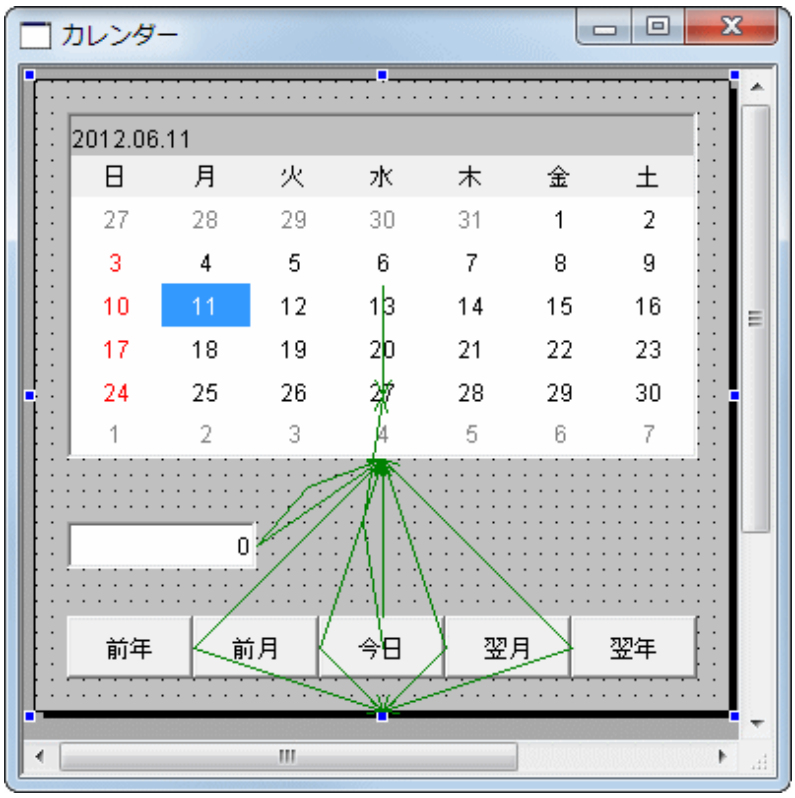
[イベント] > [action_actionPerformed]をダブルクリックします。
イベント処理の作成確認画面が表示された場合は、[はい]をクリックします。



「jFImageButton1_action_actionPerformed」イベントの処理手続きが表示されるので、上の表にしたがって、手続きを入力します。



Beanの配置およびボタンのラベルプロパティを変更したJavaフォームは以下のようになります。



5. Javaフォームを保存します。
   Javaフォーム定義のメニューバーから[ファイル] > [上書き保存]を選択します。
   Javaフォーム定義のメニューバーから[ファイル] > [終了]を選択し、Javaフォームを終了します。

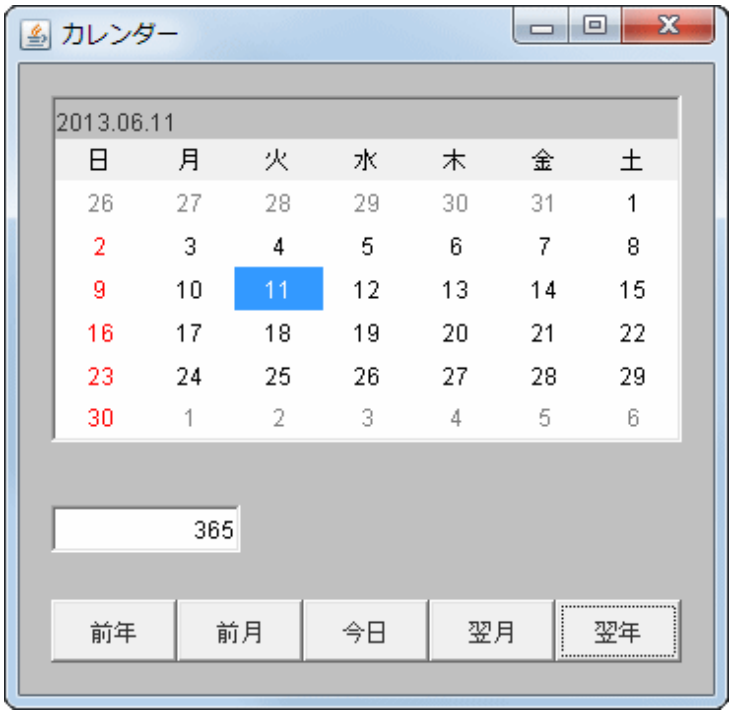
ビルド

1. ビルドは、Javaファイルなどのリソースファイルを保存すると自動的に実行されます。ビルドが正常に行われると「MyCalendar.jar」ファイルが作成されます。

実行

1. 実行するファイル(クラス)を選択します。ワークベンチの[パッケージエクスプローラ]ビューにある[MyCalendar] > [src] > [myapp.javaform] > [MyCalendar.java]をクリックします。
2. メニューバーから[実行] > [実行] > [Javaアプリケーション]を選択します。アプリケーションが起動し、Javaフォームが表示されます。

3. 追加した[前年]、[前月]、[今日]、[翌月]、[翌年]をクリックします。
   押されたボタンに対応して、カレンダーの表示が切り替わります。
   以下は「今日」を表示した状態から「翌年」を押した結果です。



4. タイトルバーのクローズボタン([X]ボタン)をクリックして、アプリケーションを終了します。

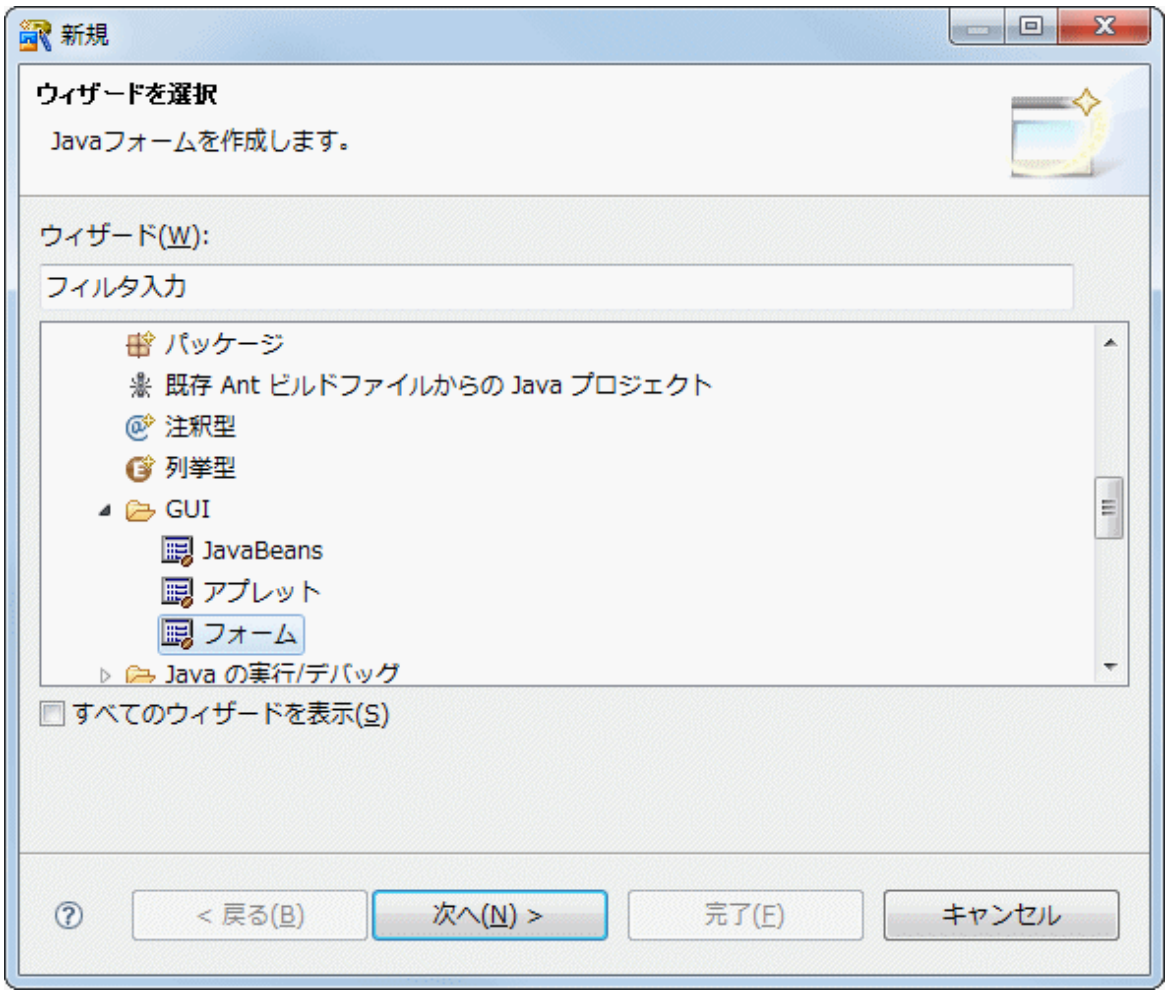
F.2.4 Lesson4 ダイアログ

ダイアログは、情報を入力する画面やメッセージを出力する画面などの補助的な画面として使用します。
カレンダーBeanには、各日付にメモを設定しツールチップで表示する機能があります。日付に対するメモを入力する画面をダイアログを使って実装します。
カレンダーの日付をダブルクリックするとダイアログを表示し、メモの入力ができるようにします。Lesson3で作成したプロジェクトに対しダイアログの追加を行います。

ダイアログの作成

1. Lesson3で作成したプロジェクト「MyCalendar」をワークベンチで開きます。
   メニューバーから[ファイル] > [新規] > [その他]を選択します。

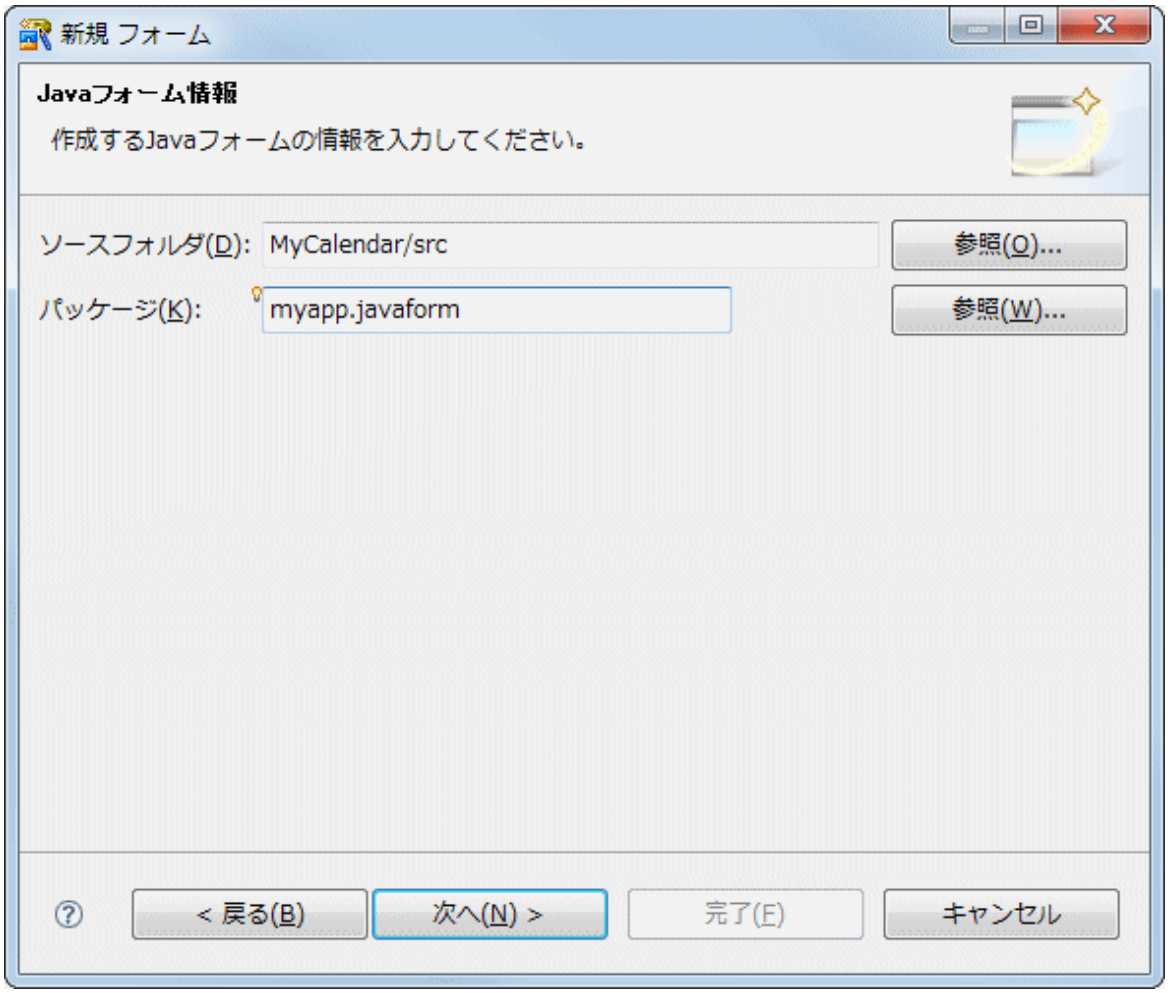
2. [新規]ウィザードが表示されます。ツリーから[Java] > [GUI] > [フォーム]を選択します。



[次へ]をクリックします。

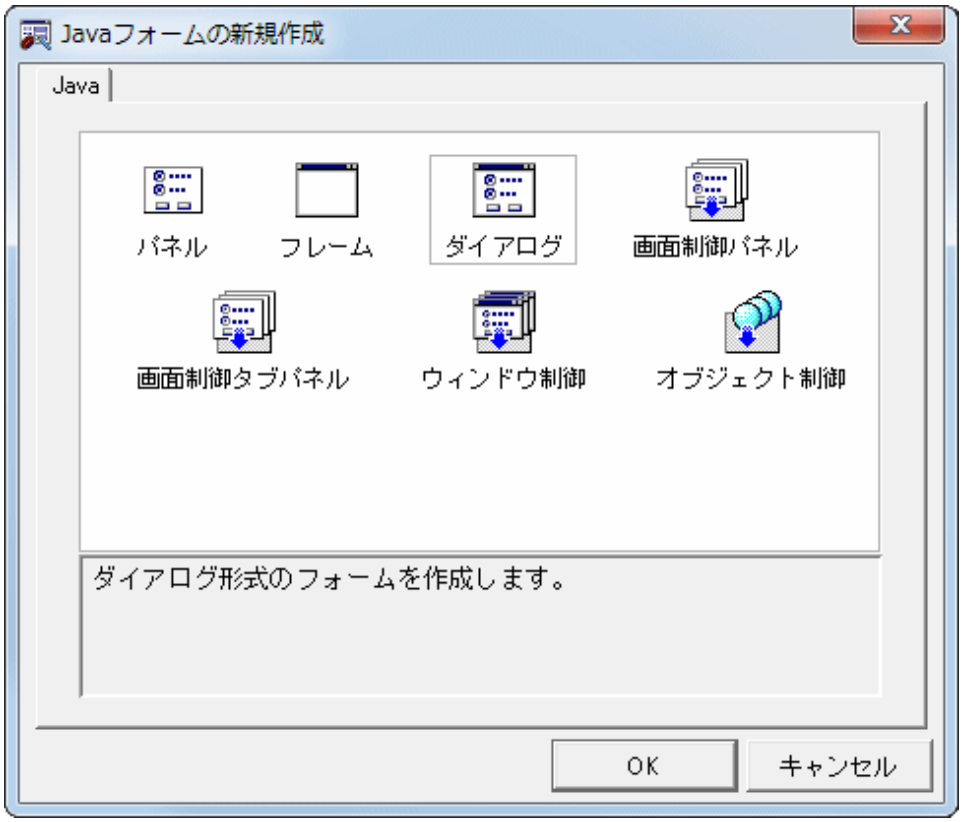
3. [Javaフォーム情報]ページが表示されます。このページでは、以下の情報を入力します。

設定項目	設定内容
パッケージ	myapp.javaform


新規 フォーム
Javaフォーム情報
作成するJavaフォームの情報を入力してください。
ソースフォルダ(D):
MyCalendar/src
参照(O)...
パッケージ(K):
myapp.javaform
参照(W)...
< 戻る(B)
次へ(N) >
完了(F)
キャンセル

[次へ]をクリックします。

4. [Javaフォームの新規作成]ダイアログボックスが表示されます。
[ダイアログ]を選択し、[OK]をクリックします。



ポイント

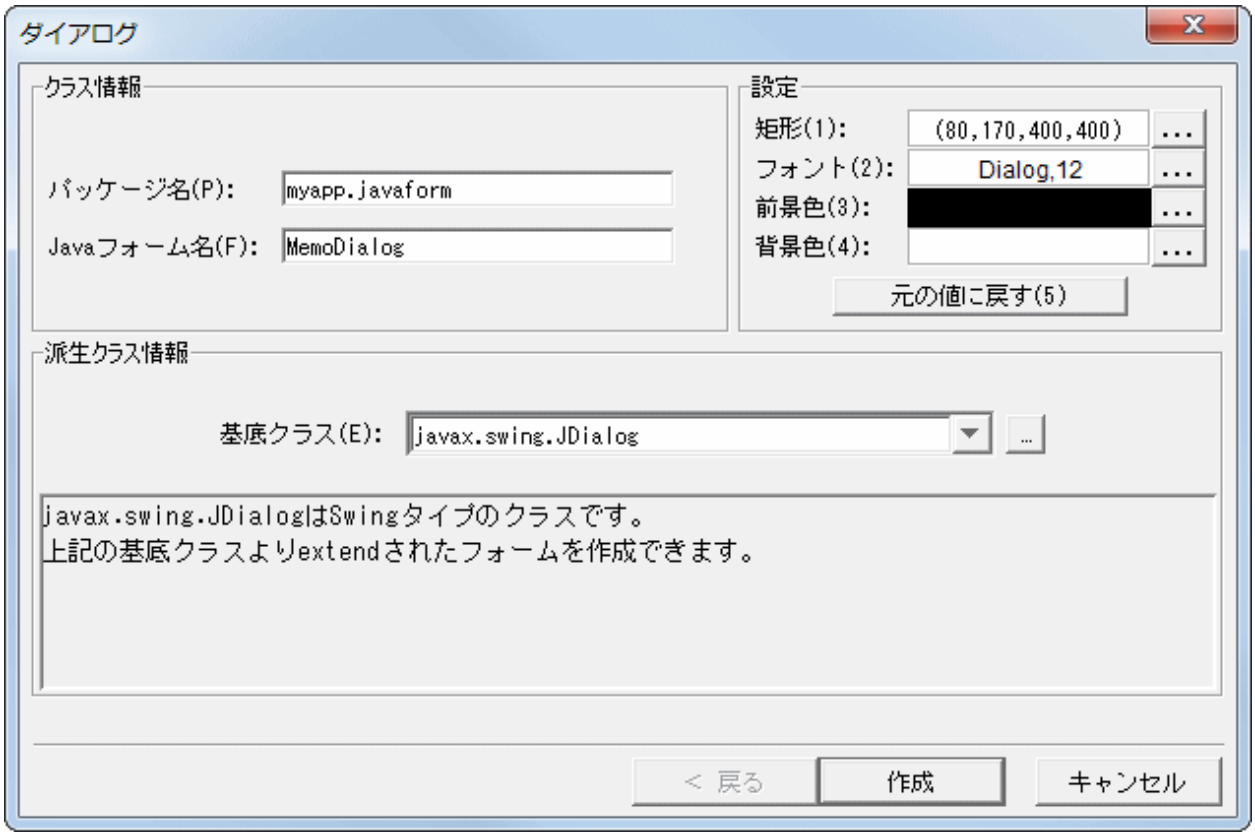
Javaフォームの種類について

Javaフォームには以下の種類があり、それぞれの種別ごとにクラス継承フォームが用意されています。なお、よく使われるフォームのレイアウトはあらかじめ登録されています。利用する場合は、画面の[ウィザード]タブをクリックして、自動生成するフォームの種別を選択します。

- フレーム
  プルダウンメニューやタイトルバーなどで構成される一般的なウィンドウのフレーム(枠)を持つフォームです。アプリケーションの初期画面など基本となる画面として利用します。
- ダイアログ
  見かけは、フレームと似ていますが、モーダルにできる、タイトルバーのアイコンを指定できないなど、フレームとの機能差があります。フレームに対して補助的な位置づけで、一時的に表示してユーザの入出力を行うような画面として利用します。
- パネル
  フレームをもたないフォームです。ほかのフレームやダイアログに貼り付けて利用します。複数のBeanをパネルに貼り付けて、複合的なGUI部品を作成する場合に利用します。

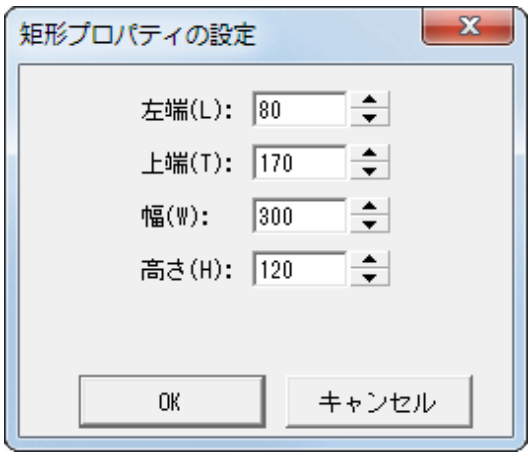
5. ダイアログを新規作成します。
[ダイアログ]ページが表示されます。以下の情報を入力します。

設定項目	設定内容
パッケージ名	myapp.javaform
Javaフォーム名	MemoDialog
基底クラス	javax.swing.JDialog

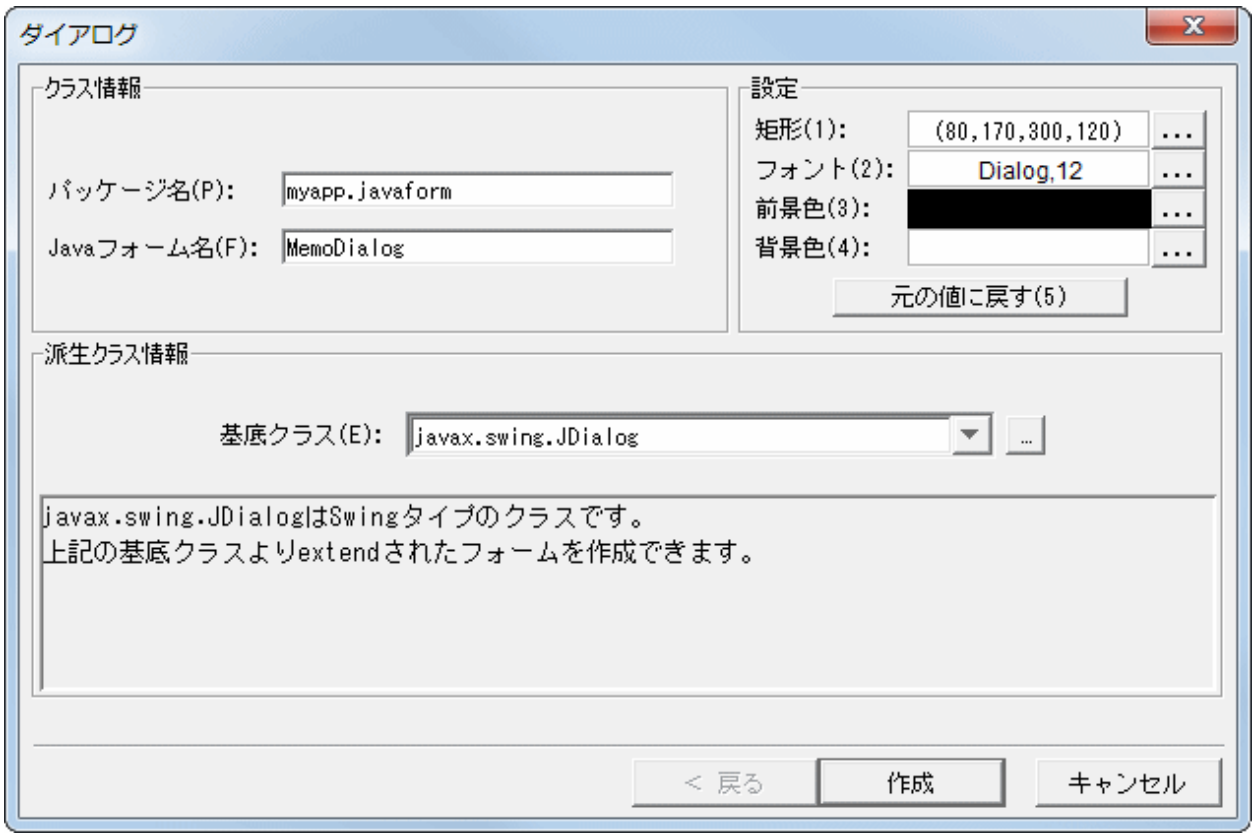


矩形を設定します。
矩形の[...]をクリックします。[矩形プロパティの設定]ダイアログボックスが表示されます。
プロパティを以下のように指定し、[OK]をクリックします。

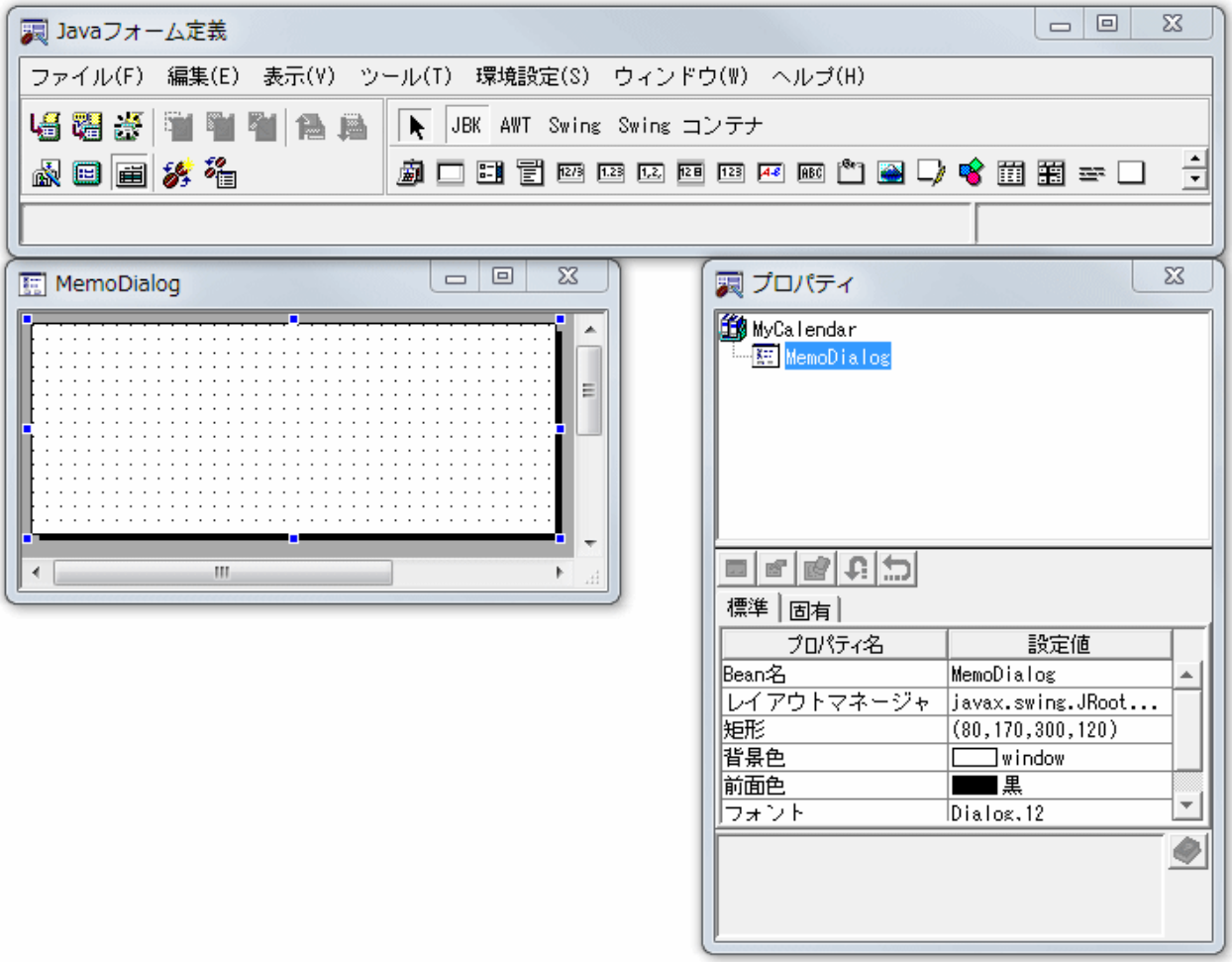
プロパティ名	値
幅	300
高さ	120



[作成]をクリックします。


ダイアログ
クラス情報
パッケージ名(P):
myapp.javaform
Javaフォーム名(F):
MemoDialog
設定
矩形(1):
(80,170,300,120)
フォント(2):
Dialog,12
前景色(3):
背景色(4):
元の値に戻す(5)
派生クラス情報
基底クラス(E):
javax.swing.JDialog
javax.swing.JDialogはSwingタイプのクラスです。
上記の基底クラスよりextendされたフォームを作成できます。
< 戻る
作成
キャンセル

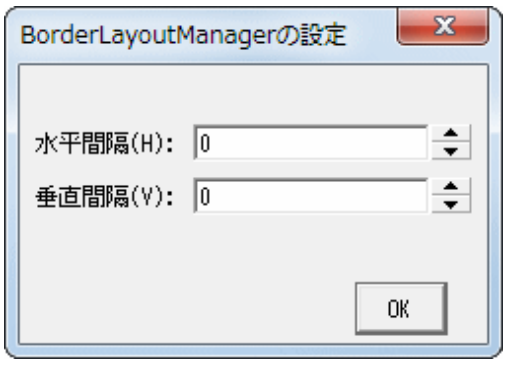
6. Javaフォーム定義が起動されます。



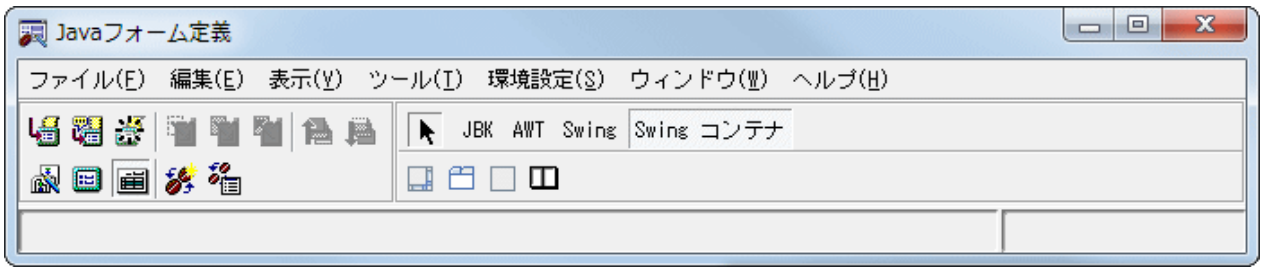
Beanの配置

1. 作成したMemoDialogの[レイアウトマネージャ]プロパティを[BorderLayout]に変更します。

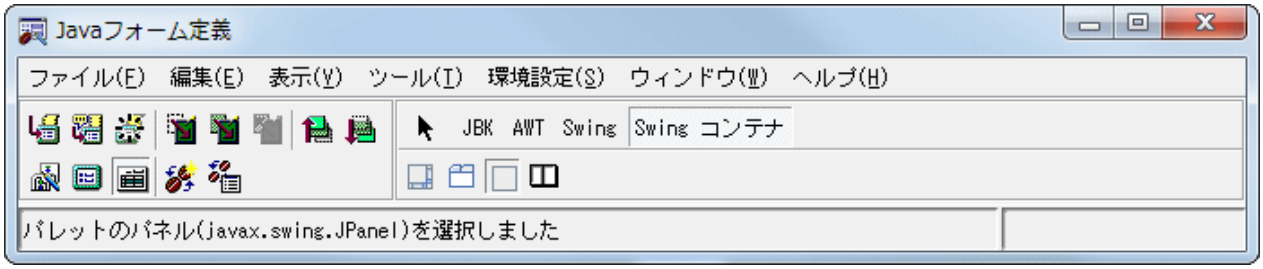
[レイアウトマネージャ]を変更すると、[BorderLayoutManagerの設定]ダイアログボックスが表示されるので[OK]をクリックします。



2. オブジェクトパレットの[Swingコンテナ]が選択されていない場合は、[Swingコンテナ]をクリックします。

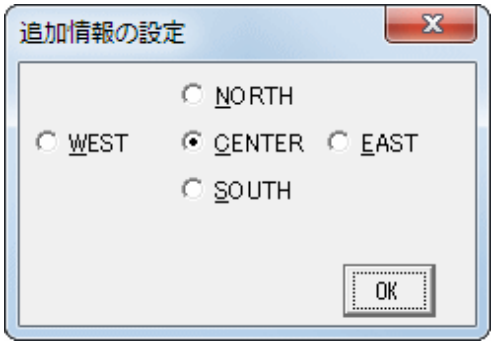


3. オブジェクトパレットの[パネル]をクリックし選択します。

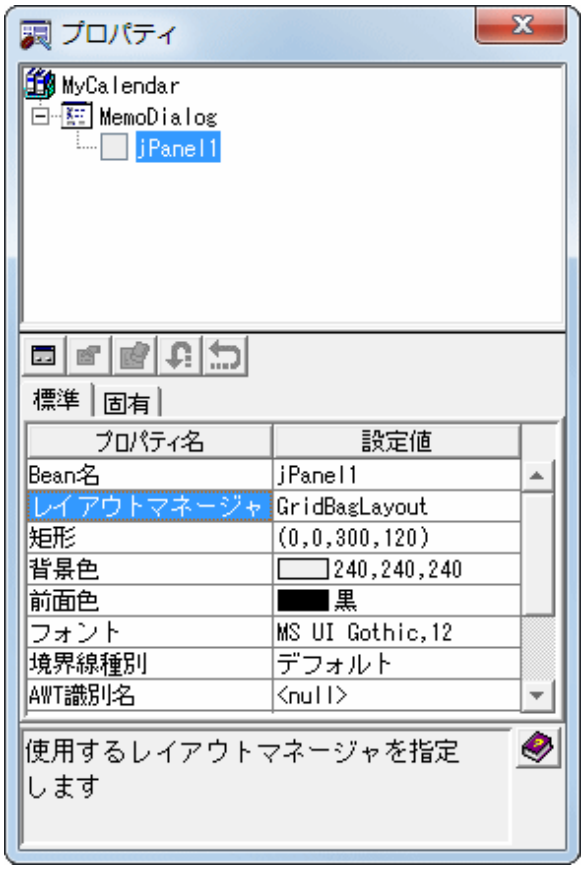


4. マウスを使ってJavaフォーム上でドラッグして配置します。

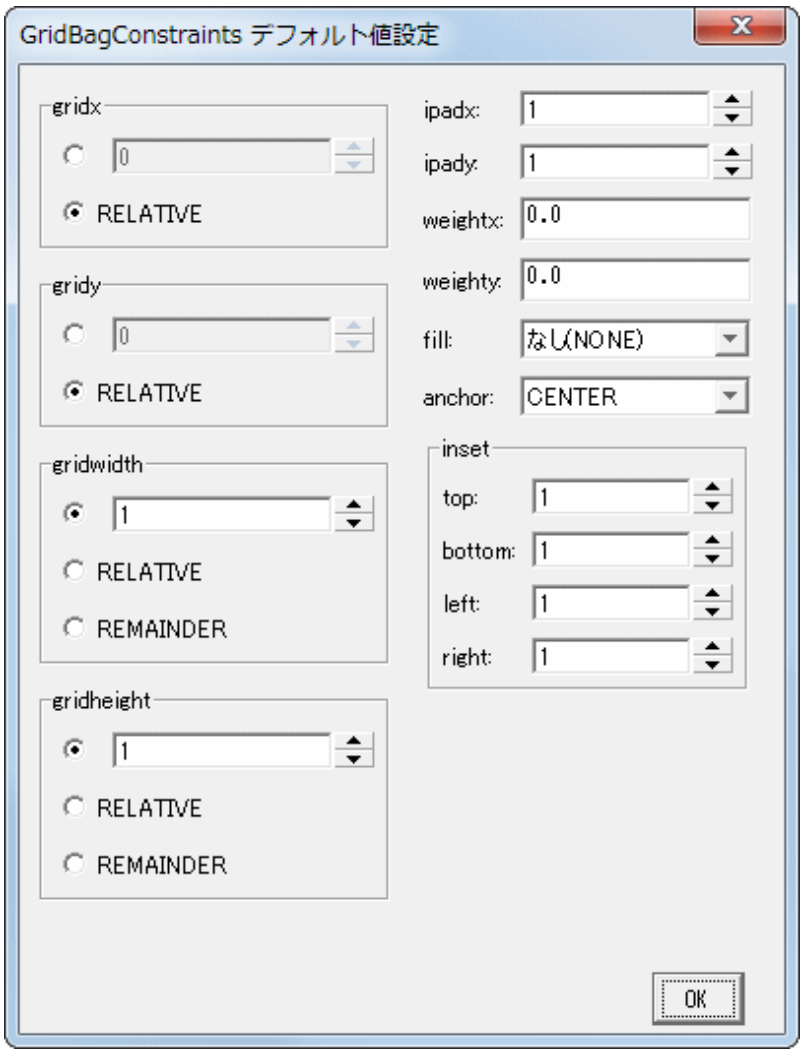
5. [追加情報の設定]では[CENTER]を選択して[OK]をクリックします。



6. 作成したパネルのレイアウトを変更します。
パネルの[レイアウトマネージャ]プロパティを[GridBagLayout]に変更します。

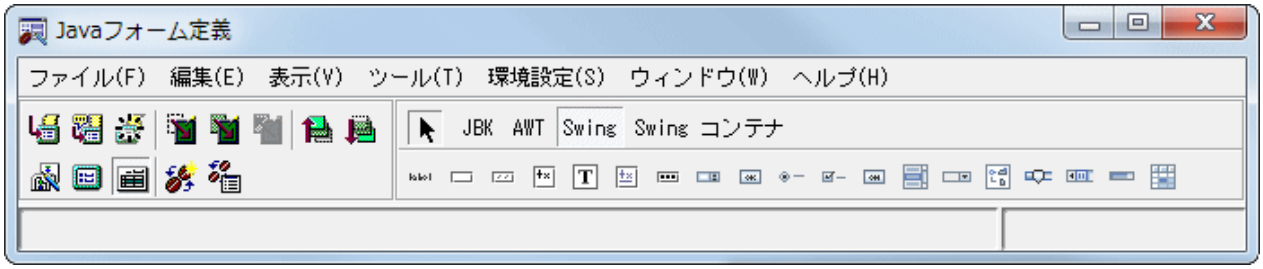


[GridBagConstraintsデフォルト値設定]ダイアログボックスでは、そのまま[OK]をクリックします。

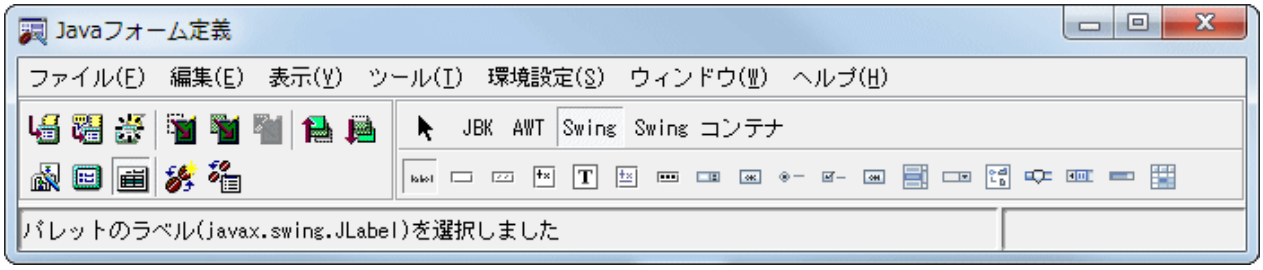


7. 2つ目のパネルを配置します。
   ダイアログの直下に配置するため、先ほど配置したパネルの上でドラッグを行わず、ダイアログの外のグレーの領域でドラッグしてください。
   [追加情報の設定]では[SOUTH]を選択して[OK]をクリックします。

8. ラベルを配置します。
   オブジェクトパレットの[Swing]が選択されていない場合は、[Swing]をクリックします。

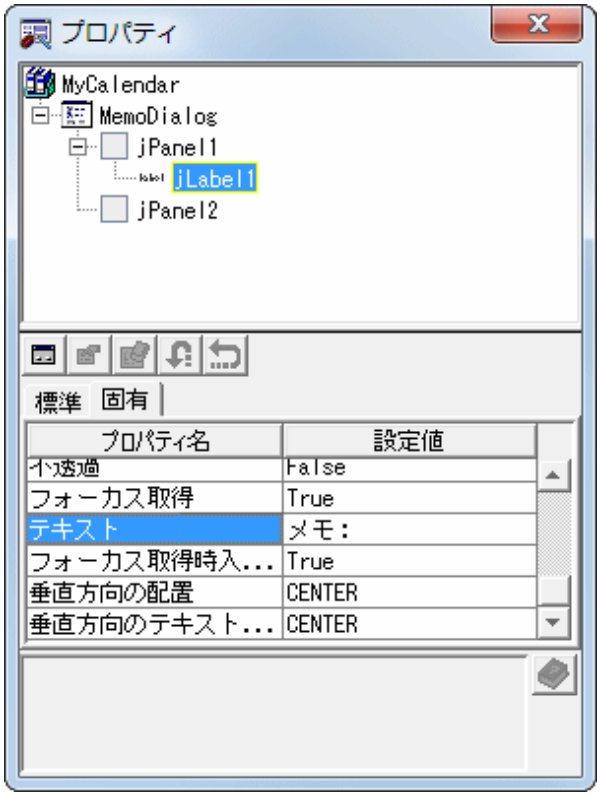


9. オブジェクトパレットの[ラベル]をクリックし選択します。



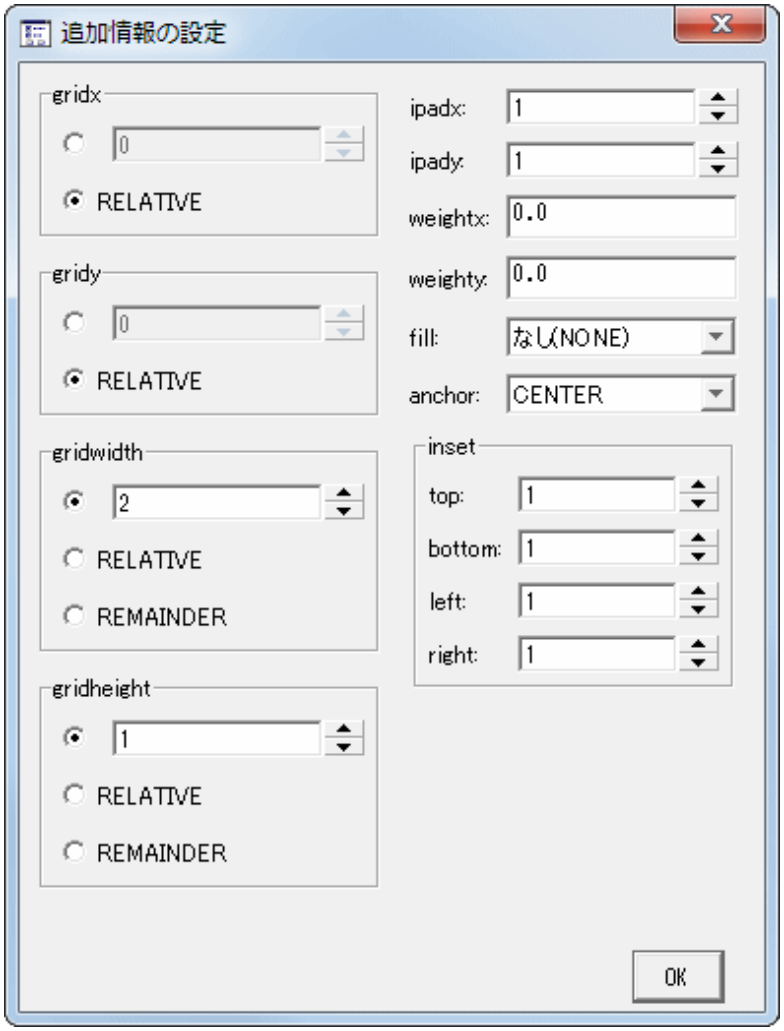
10. 1つ目に作成したパネルの上に配置するよう、作成したダイアログの上側でドラッグします。
[追加情報の設定]では、そのまま[OK]をクリックします。

11. 作成したラベルのテキストを変更します。
作成したラベルを選択し、プロパティウィンドウの[固有]タブを選択します。
[テキスト]プロパティを[メモ:]に変更します。

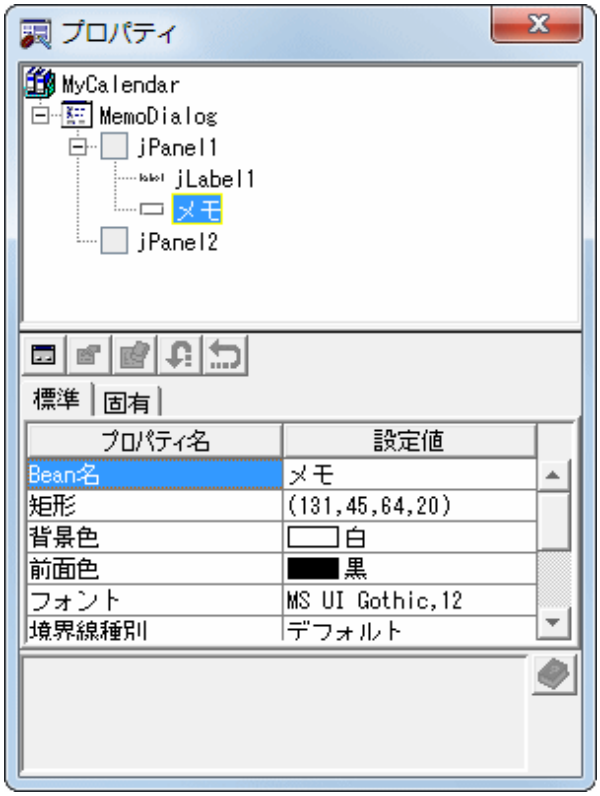


12. テキストフィールドを配置します。
オブジェクトパレットの[テキストフィールド]をクリックし選択します。

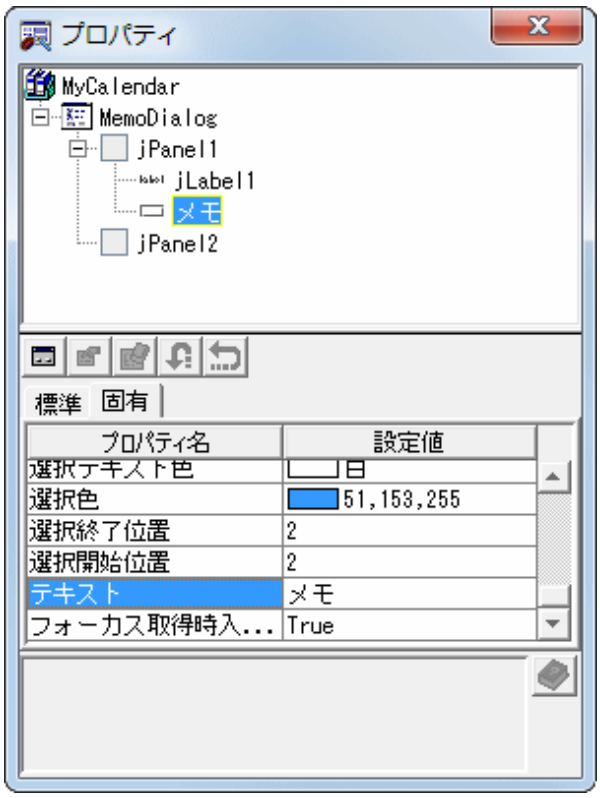
13. 1つ目に作成したパネルの上に配置するよう、作成したダイアロの上側でドラッグします。
[追加情報の設定]では、[gridwidth]を[2]し変更し[OK]をクリックします。



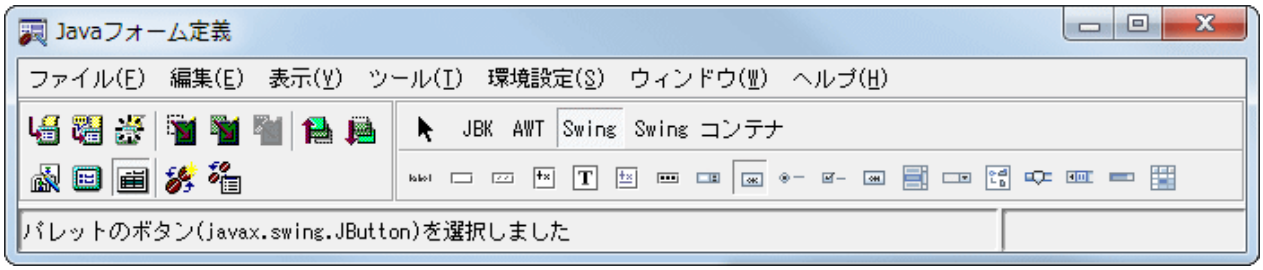
14. 作成したテキストフィールドの[Bean名]を変更します。
作成したテキストフィールドを選択し、プロパティウィンドウの[標準]タブを選択します。
[Bean名]プロパティを[メモ]に変更します。



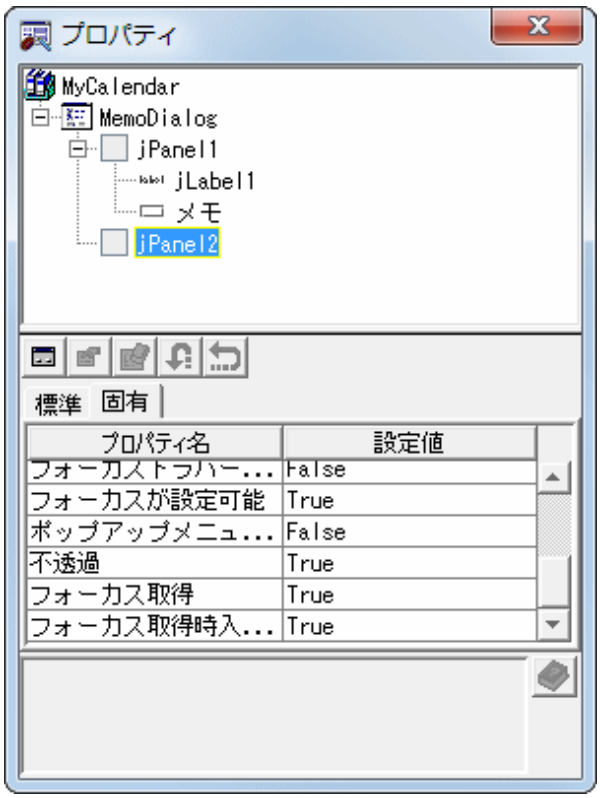
15. 作成したテキストフィールドのテキストを変更します。
作成したテキストフィールドを選択し、プロパティウィンドウの[固有]タブを選択します。
[テキスト]プロパティを[メモ]に変更します。



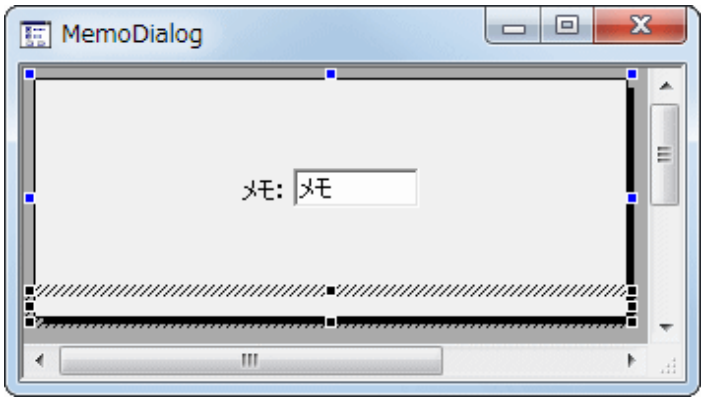
16. ボタンを配置します。
オブジェクトパレットの[ボタン]をクリックし選択します。



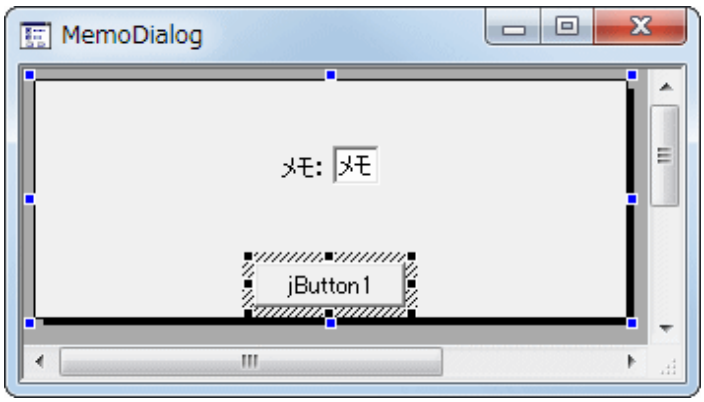
17. 2つ目に作成したパネルの上に配置するよう、プロパティウィンドウのツリーに存在する[jPanel2]を選択します。



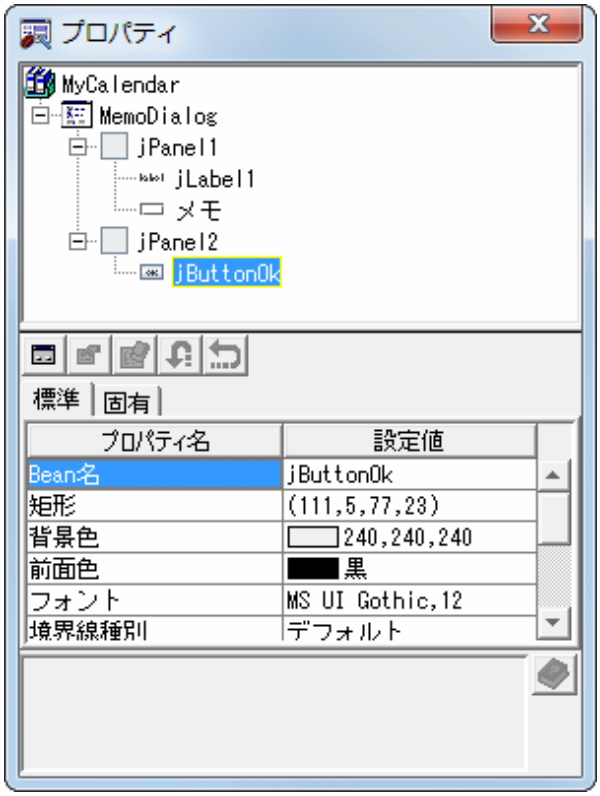
18. 選択すると対象パネルの部分に枠が表示されます。



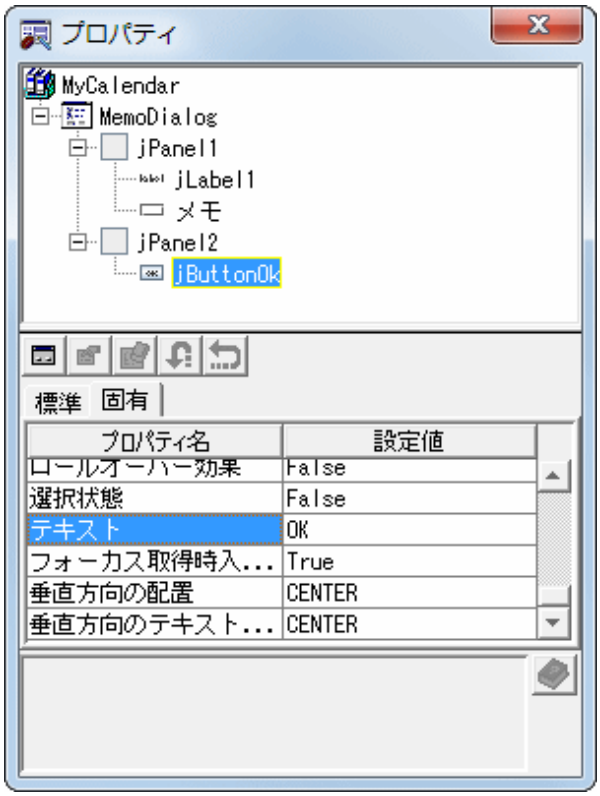
19. 枠の中でドラッグします。



20. 作成したボタンの[Bean名]を変更します。
作成したボタンを選択し、プロパティウィンドウの[標準]タブを選択します。
[Bean名]プロパティを[jButtonOk]に変更します。



21. 作成したボタンの[テキスト]を変更します。
作成したボタンを選択し、プロパティウィンドウの[固有]タブを選択します。
[テキスト]プロパティを[OK]に変更します。

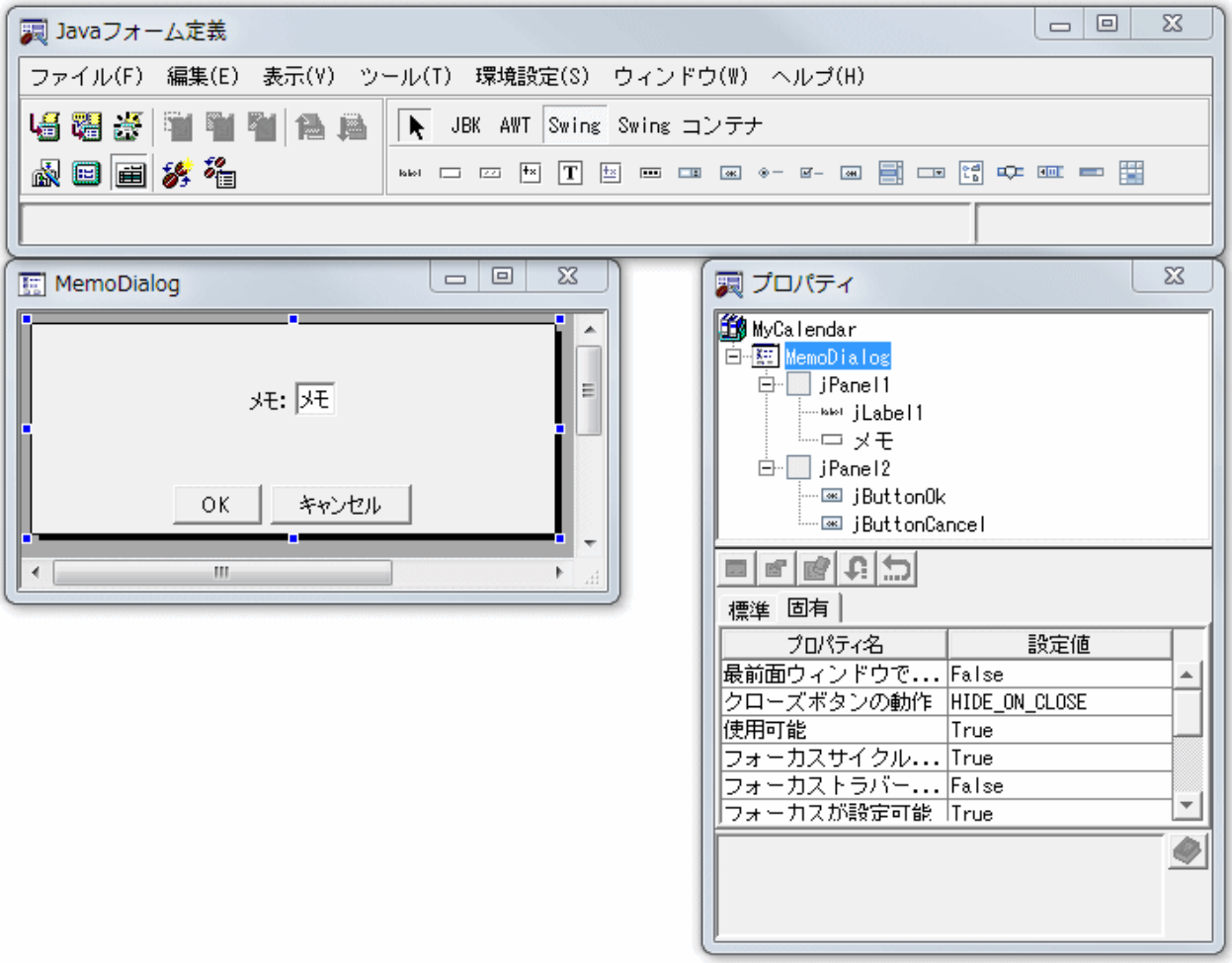


22. 2つ目のボタンを配置します。
    オブジェクトパレットの[ボタン]をクリックし選択します。

23. 先ほどと同様に2つ目に作成したパネルの上に配置するよう、プロパティウィンドウのツリーに存在する[jPanel2]を選択します。
    選択すると対象パネルの部分に枠が表示されます。
    枠の中でドラッグします。

24. 作成したボタンの[Bean名]を変更します。
    作成したボタンを選択し、プロパティウィンドウの[標準]タブを選択します。
    [Bean名]プロパティを[jButtonCancel]に変更します。

25. 作成したボタンの[テキスト]を変更します。
    作成したボタンを選択し、プロパティウィンドウの[固有]タブを選択します。
    [テキスト]プロパティを[キャンセル]に変更します。



26. Javaフォーム定義のメニューバーから[表示] > [Javaエディタ]を選択し、Javaエディタをアクティブにします。

27. JBKのクラスを使用するので、importキーワードを追加します。
    赤字の部分を追加します。

```
import com.fujitsu.jbk.gui.JFCalendarView;
```

28. MemoDialogクラスにフィールドを追加します。
    このフィールドは呼び出し元フレームのカレンダークラスを保存するために使用します。赤字の部分を追加します。

```
//@@Form Design Information end

JFCalendarView cv;
/**
```

```
 * フォームを構築します。
 */
public MemoDialog(java.awt.Frame param0, boolean param1) {
    super(param0, param1);
}
```

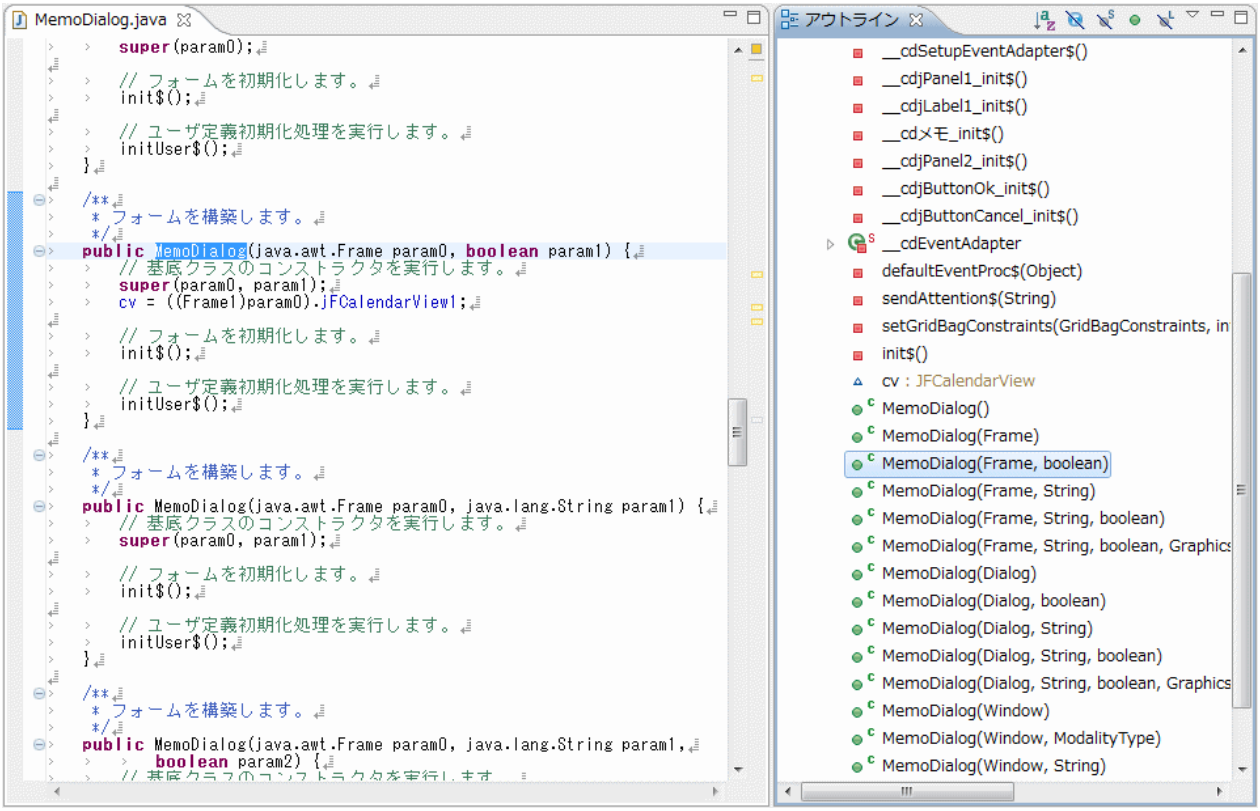
29. MemoDialogクラスのコンストラクタに処理を追加します。
コンストラクタとは、オブジェクトを生成したときに自動的に呼び出されるメソッドで、通常はオブジェクトの初期化処理を行います。クラス名と同名のメソッドがコンストラクタとして扱われます。赤字の部分を追加します。

```
public MemoDialog(java.awt.Frame param0, boolean param1) {
        // 基底クラスのコンストラクタを実行します。
        super(param0, param1);
        cv = ((Frame1) param0).jFCalendarView1;
        // フォームを初期化します。
        init$();
        // ユーザ定義初期化処理を実行します。
        initUser$();
}
```

ポイント

Javaエディタのスクロールについて

Javaエディタの表示を編集したいメソッドやフィールドまでスクロールしたい場合、[アウトライン]ビューを利用すると便利です。[アウトライン]ビューにある[MemoDialog(Frame, boolean)]をクリックすると、Javaエディタの表示がコンストラクタまでスクロールします。

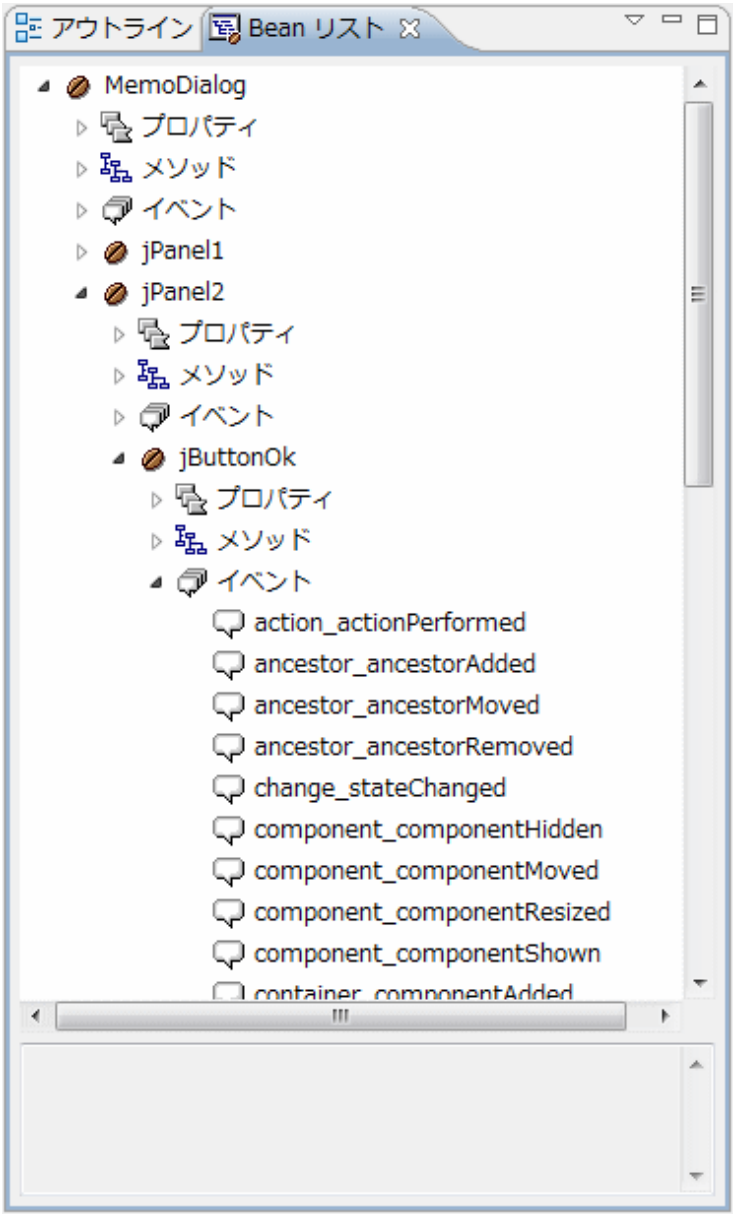


30. MemoDialogクラスのinitUser$メソッドに処理を追加し、カレンダーに設定されているメモを初期表示するようにします。
initUser$()の処理手続きに赤字の部分を追加します。

```
    protected void initUser$() {
        // この位置にユーザ定義初期化処理を記述します.
```

```
        メモ.setText(cv.getMemo(cv.getDate()));
        メモ.setColumns(10);
    }
```

31. [OK]をクリックしたときの処理を記述します。
[Beanリスト]ビューにある[MemoDialog] > [jPanel2] > [jButtonOk] > [イベント] > [action_actionPerformed]をダブルクリックします。
イベント処理の作成確認画面が表示された場合は、[はい]をクリックします。



「jButtonOk_action_actionPerformed」の処理手続きが表示されるので、赤字の部分を記述します。

```
    public void jButtonOk_action_actionPerformed(java.awt.event.ActionEvent e) {
        if(!defaultEventProc(e)) {
            // ここにイベント発生時の処理を記述します.
            cv.setMemo(cv.getDate(), メモ.getText());
            dispose();
        }
    }
```

次に、[キャンセル]をクリックしたときの処理を記述します。
[Beanリスト]ビューにある[MemoDialog] > [jPanel2] > [jButtonCancel] > [イベント] > [action_actionPerformed]をダブルクリックします。

イベント処理の作成確認画面が表示された場合は、[はい]をクリックします。
「jButtonCancel_action_actionPerformed」の処理手続きが表示されるので、赤字の部分を記述します。

```
public void jButtonCancel_action_actionPerformed(java.awt.event.ActionEvent e) {
    if(!defaultEventProc(e)) {
        // ここにイベント発生時の処理を記述します.
        dispose();
    }
}
```

32. Javaフォームを保存します。
ワークベンチのメニューバーから[ファイル] > [保存]を選択します。
Javaフォーム定義のメニューバーから [ファイル] > [終了]を選択し、Javaフォームを終了します。

カレンダーへの処理追加

1. カレンダーの日付をダブルクリックしたときにダイアログを表示し、メモの入力ができるようにします。
[Frame1.java]をJavaフォーム定義で開きます。ワークベンチの[パッケージエクスプローラ]ビューにある[MyCalendar] > [src] > [myapp.javaform] > [Frame1.java]を右クリックしてコンテキストメニューを表示します。そして[アプリケーションから開く] > [グラフィカルエディタ]を選択します。

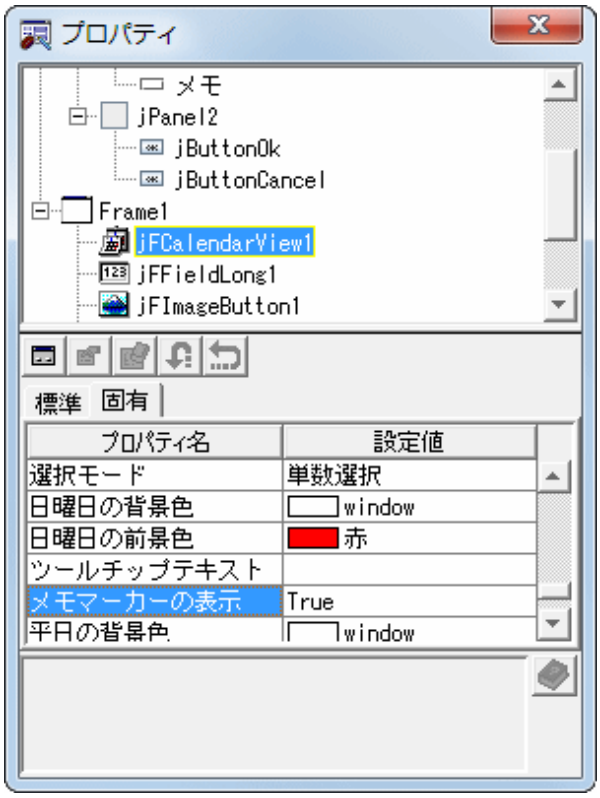
2. Javaフォーム定義のメニューバーから[表示] > [Javaエディタ]を選択し、Javaエディタをアクティブにします。

3. カレンダーの日付をダブルクリックした場合は actionイベントが発生します。actionイベントに処理を記述します。
[Beanリスト]ビューにある[Frame1] > [jFCalendarView1] > [イベント] > [action_actionPerformed]をダブルクリックします。イベント処理の作成確認画面が表示された場合は、[はい]をクリックします。
「jFCalendarView1_action_actionPerformed」の処理手続きが表示されるので、赤字の部分を記述します。

```
public void jFCalendarView1_action_actionPerformed( java.awt.event.ActionEvent e) {
    if( !defaultEventProc(e) ) {
        // ここにイベント発生時の処理を記述します。
        MemoDialog dlg = new MemoDialog(this, false);
        dlg.setVisible(true);
    }
}
```

4. カレンダーBeanのプロパティを変更します。
プロパティシートを使って[固有]タブにある以下のプロパティを変更します。

プロパティ名	値
メモ表示	True
メモマーカーの表示	True



5. Javaフォームを保存します。
   Javaフォーム定義のメニューバーから[ファイル] > [上書き保存]を選択します。
   Javaフォーム定義のメニューバーから [ファイル] > [終了]を選択し、Javaフォームを終了します。

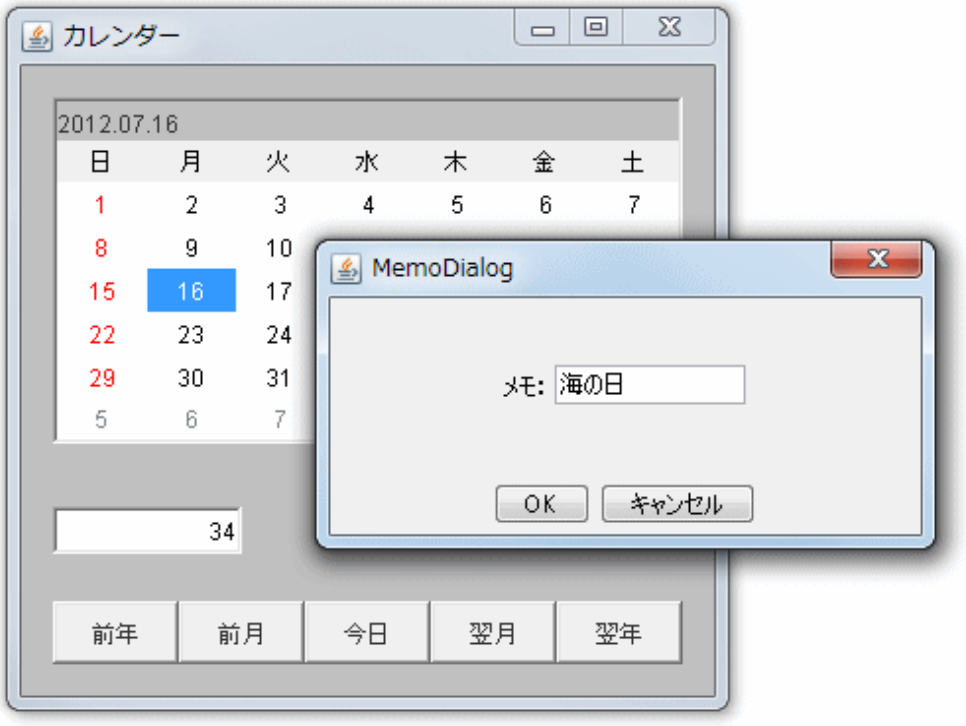
ビルド

1. ビルドは、[プロジェクト]メニューの[自動的にビルド]が有効になっている場合は、Javaファイルなどのリソースファイルを保存すると自動的に実行されます。
   [自動的にビルド]が無効になっている場合は、プロジェクトを選択し右クリックメニューの[プロジェクトのビルド]を選択するか、[プロジェクト]メニューの[プロジェクトのビルド]を選択します。
   ビルドが正常に行われると「MyCalendar.jar」ファイルが作成されます。

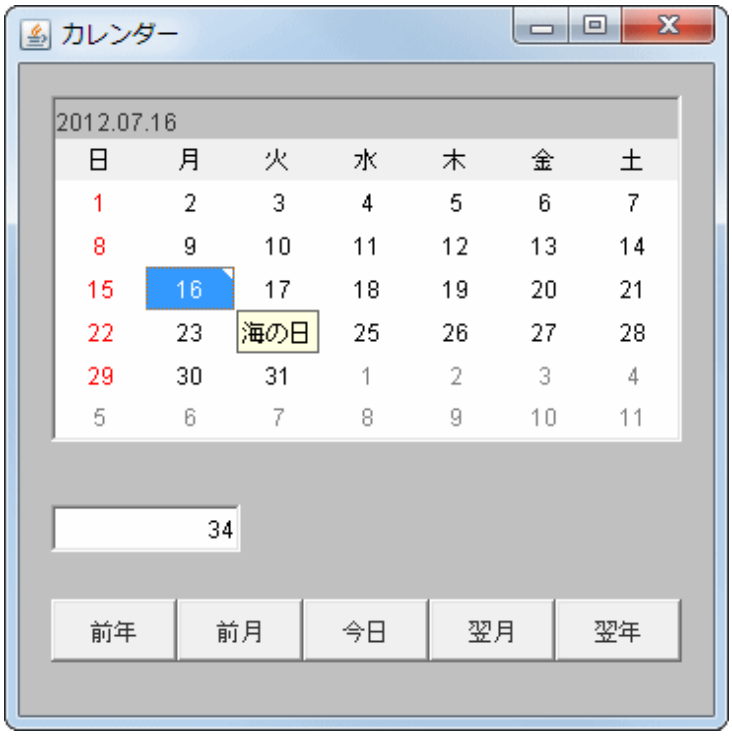
実行

1. 実行するファイル(クラス)を選択します。ワークベンチの[パッケージエクスプローラ]ビューにある[MyCalendar] > [src] > [myapp.javaform] > [MyCalendar.java]をクリックします。
2. メニューバーから[実行] > [実行] > [Javaアプリケーション]を選択します。アプリケーションが起動し、Javaフォームが表示されます。

3. カレンダー上の日付をダブルクリックすると、ダイアログが表示されます。
   メモを入力し、[OK]をクリックします。



4. メモを入力した日付にマーカーが表示されます。
   また、メモが設定された日付の上にカーソルを重ねると、ツールチップにメモが表示されます。



5. タイトルバーのクローズボタン([×]ボタン)をクリックして、アプリケーションを終了します。

F.3　アプレット

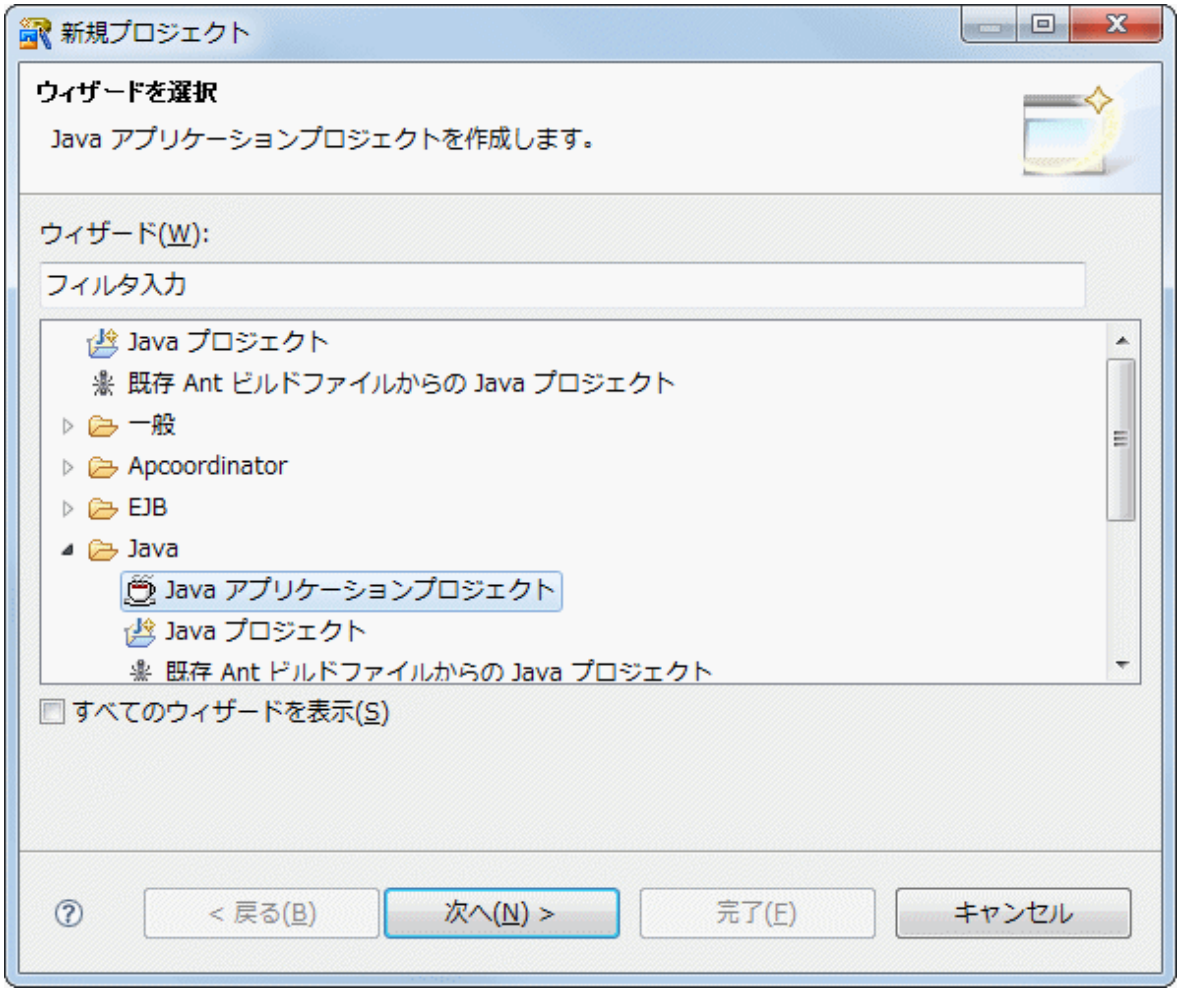
簡単なアプレットや複数画面の遷移を行うアプレットの作成を通して、アプレットの開発手順を説明します。

F.3.1　アプレットの開発手順

アプレットとは、Webブラウザ上に表示されるHTMLに埋め込まれて動作するプログラムです。Interstage Studioでは、Javaアプリケーションのひとつであるアプレットを開発することができます。
"F.2 クライアントアプリケーション"のLesson1～Lesson3で作成したアプリケーションと同じ動作をするアプレットを開発します。

Javaアプリケーションプロジェクトの作成

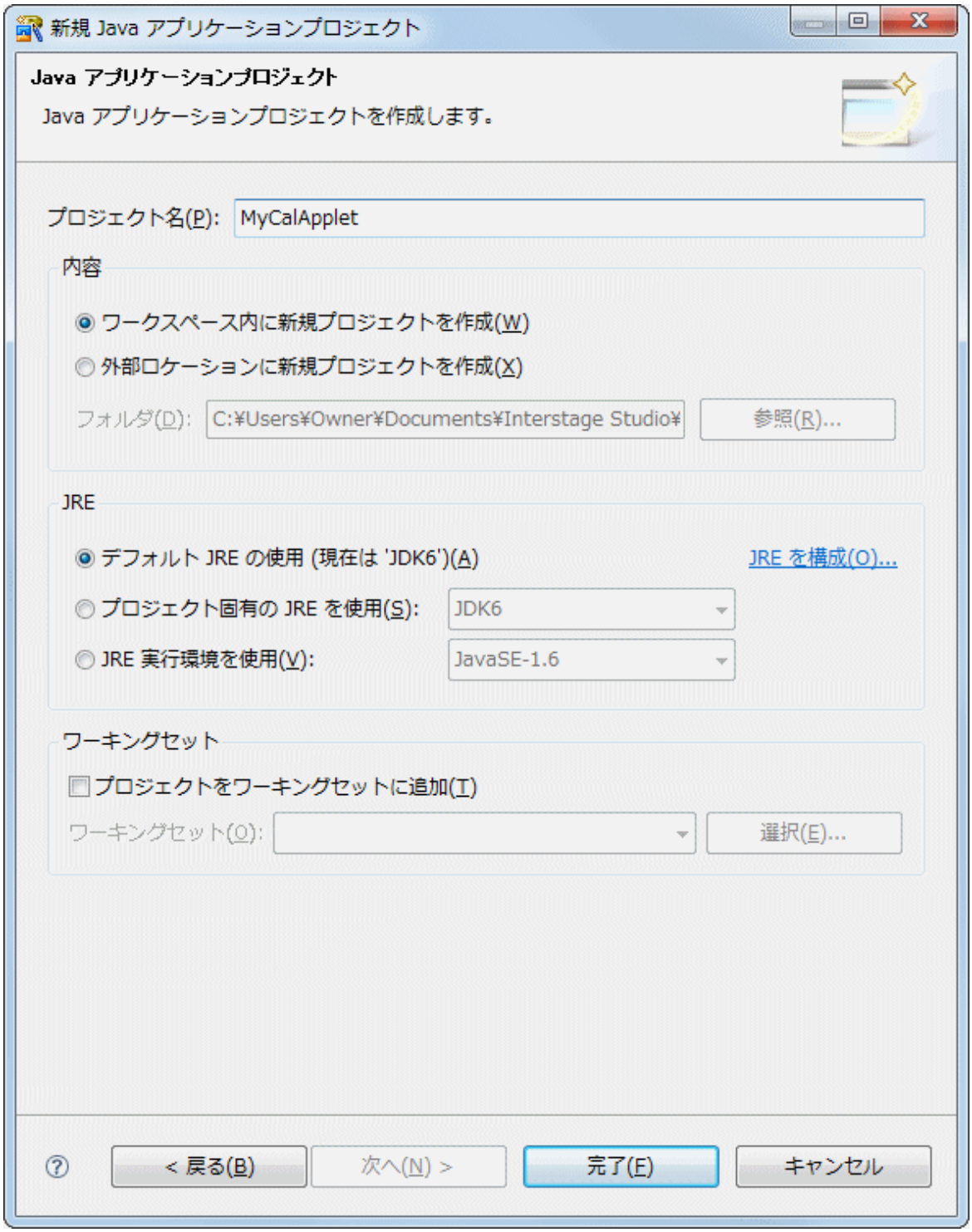
1. ワークベンチを起動します。
2. メニューバーから[ファイル] > [新規] > [プロジェクト]を選択します。
3. [新規プロジェクト]ウィザードが表示されます。ツリーから[Javaアプリケーションプロジェクト]を選択します。



[次へ]をクリックします。

4. [Javaアプリケーションプロジェクト]ページが表示されます。このページで入力した項目に従ってプロジェクトが作成されます。以下のようにプロジェクトの情報を入力します。

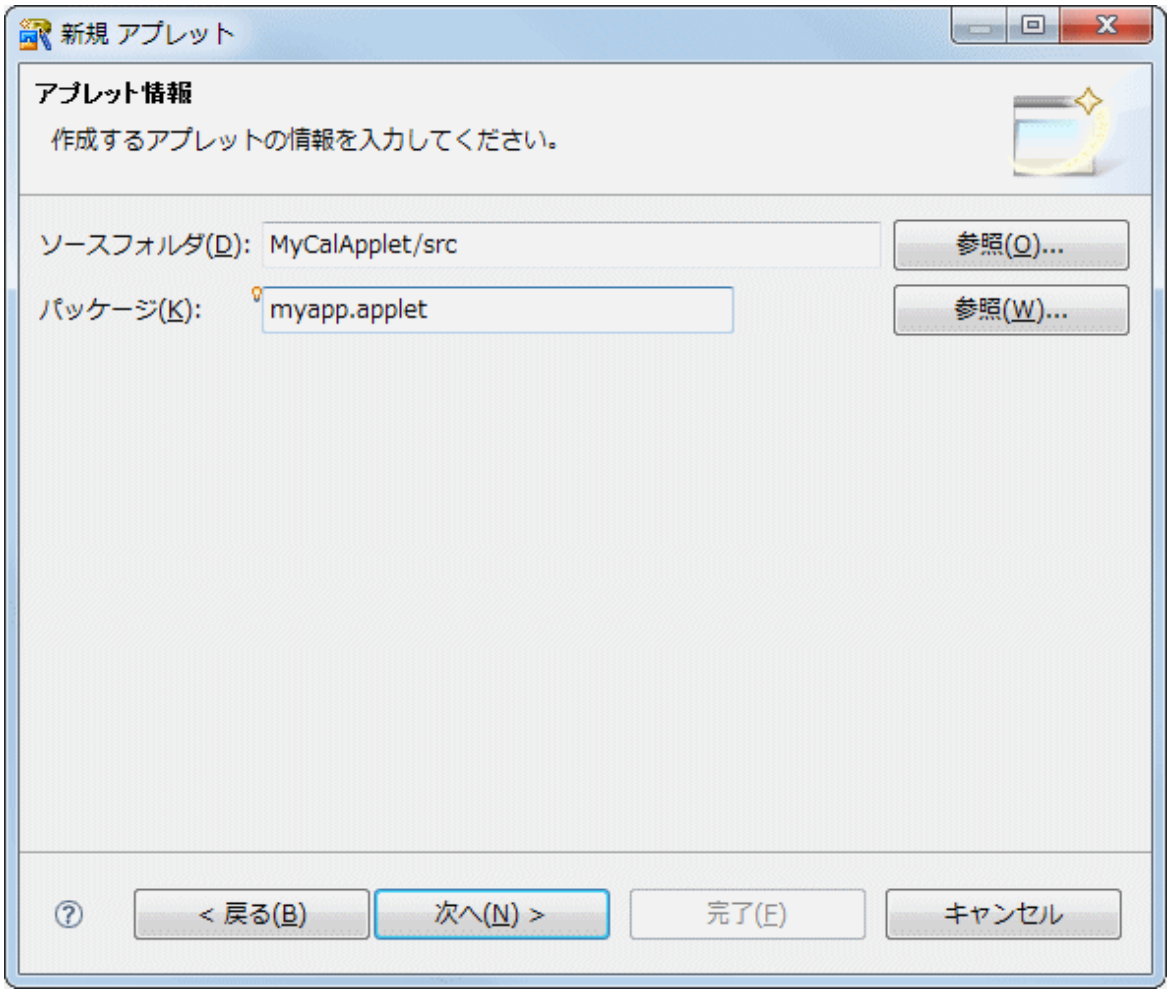
設定項目	設定内容
プロジェクト名	MyCalApplet



[完了]をクリックします。

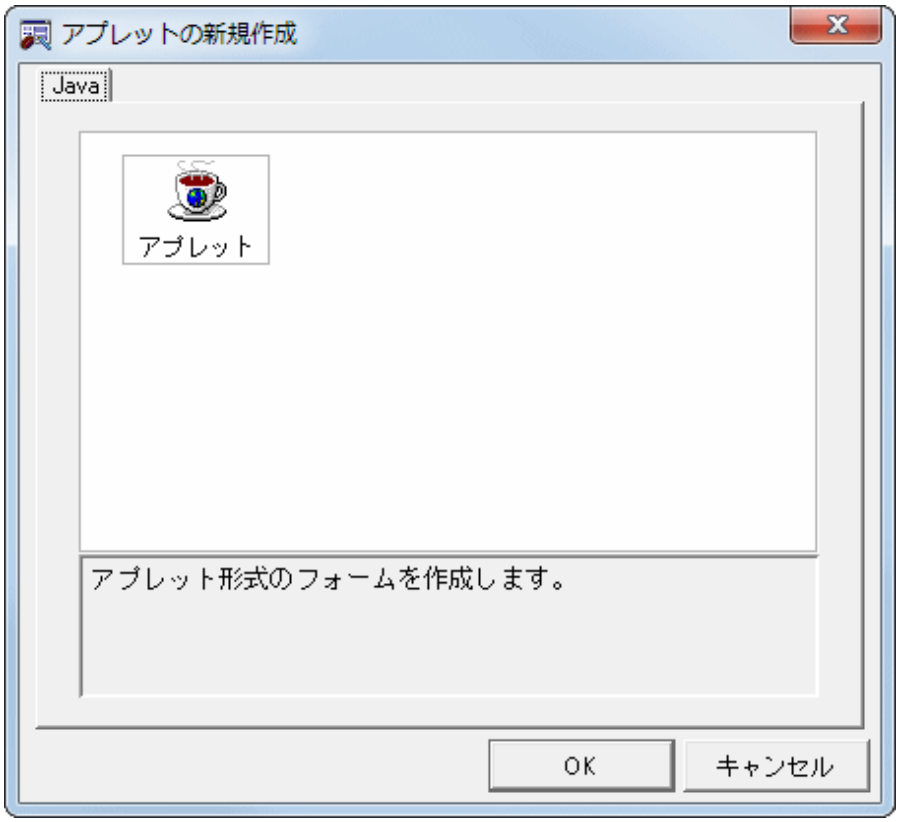
5. 次にアプレットを作成します。ワークベンチのメニューバーから、[ファイル] > [新規] > [その他]を選択します。表示される[新規]ウィザードのツリーから、[Java] > [GUI] > [アプレット]を選択します。
[アプレット情報]ページが表示されます。このページでは、以下の情報を入力します。

設定項目	設定内容
パッケージ	myapp.applet

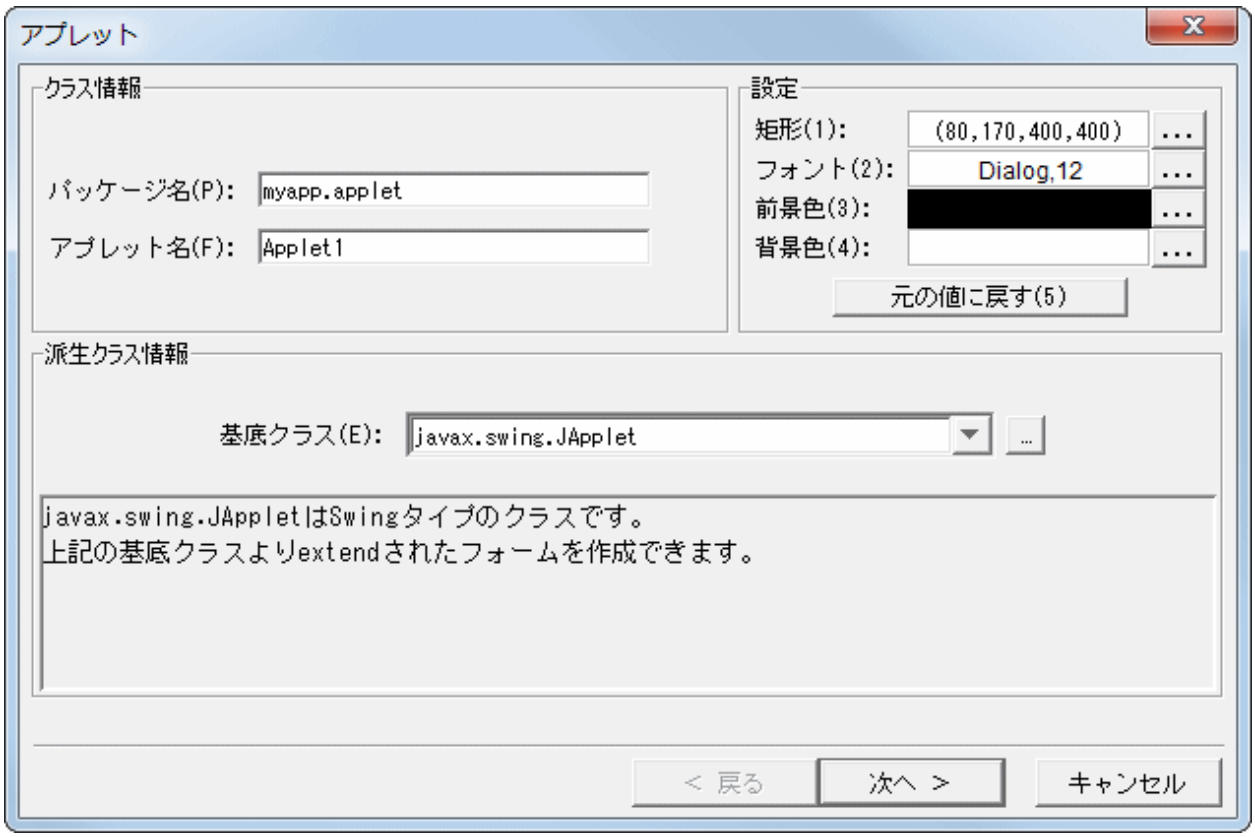

新規 アプレット
アプレット情報
作成するアプレットの情報を入力してください。
ソースフォルダ(D):
MyCalApplet/src
参照(O)...
パッケージ(K):
myapp.applet
参照(W)...
< 戻る(B)
次へ(N) >
完了(F)
キャンセル

[次へ]をクリックします。

6. [アプレットの新規作成]ダイアログボックスが表示されます。
[Java]タブの[アプレット]を選択し、[OK]をクリックします。



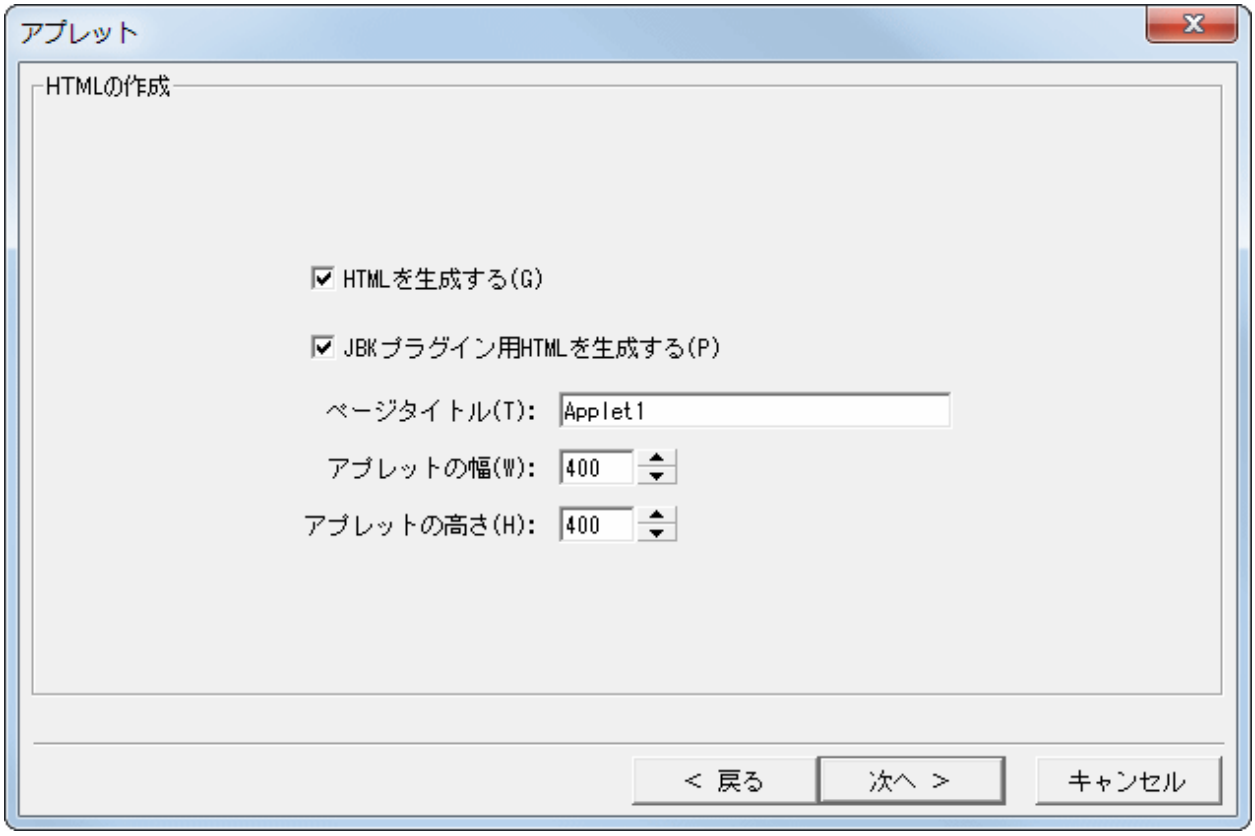
7. アプレットのクラス情報を入力するページが表示されます。アプレット名に「Applet1」と指定します。
基底クラスには[javax.swing.JApplet]を選択します。



[次へ]をクリックします。

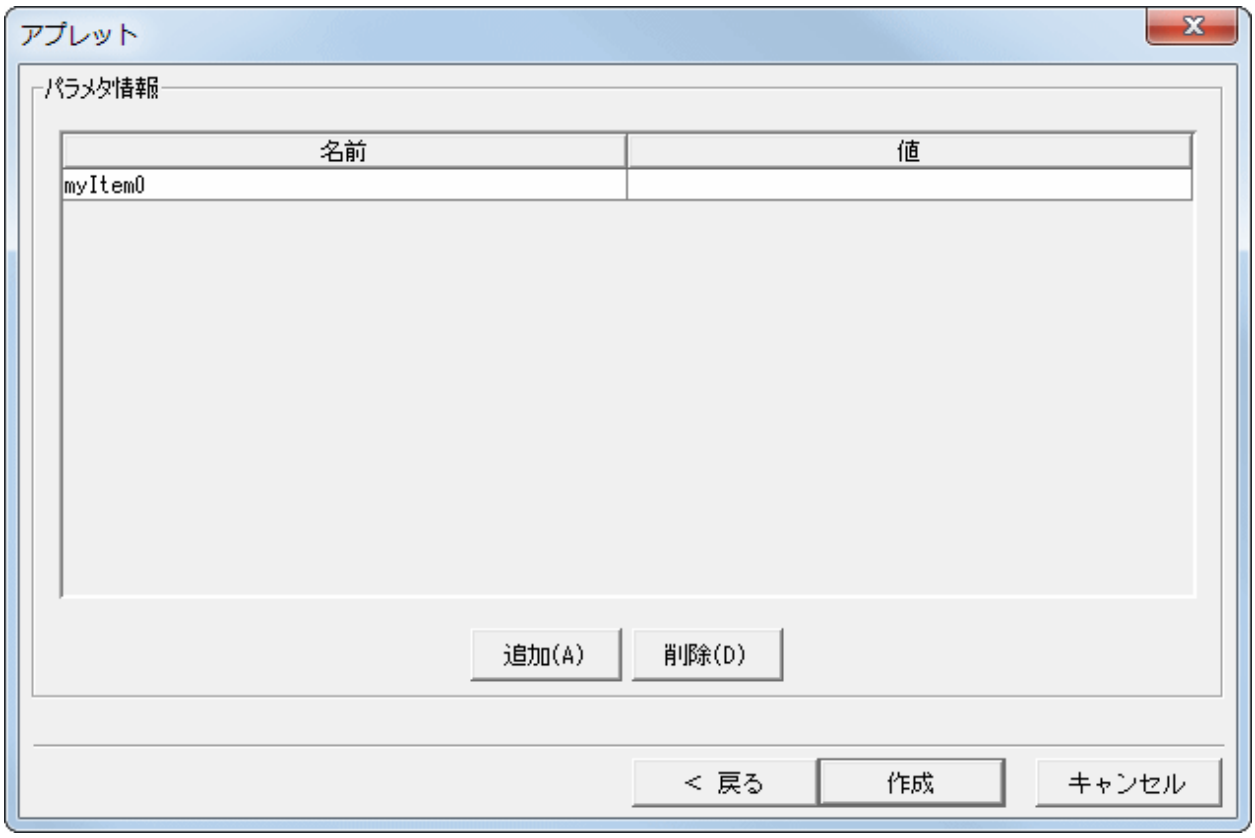
8. [HTMLの作成]ページが表示されます。ここではアプレットを貼るHTMLのひな型を作成することができます。
以下のように指定します。

設定項目	設定内容
HTMLを生成する	チェックする
JBKプラグイン用HTMLを生成する	チェックする
ページタイトル	Applet1
アプレットの幅	400
アプレットの高さ	400

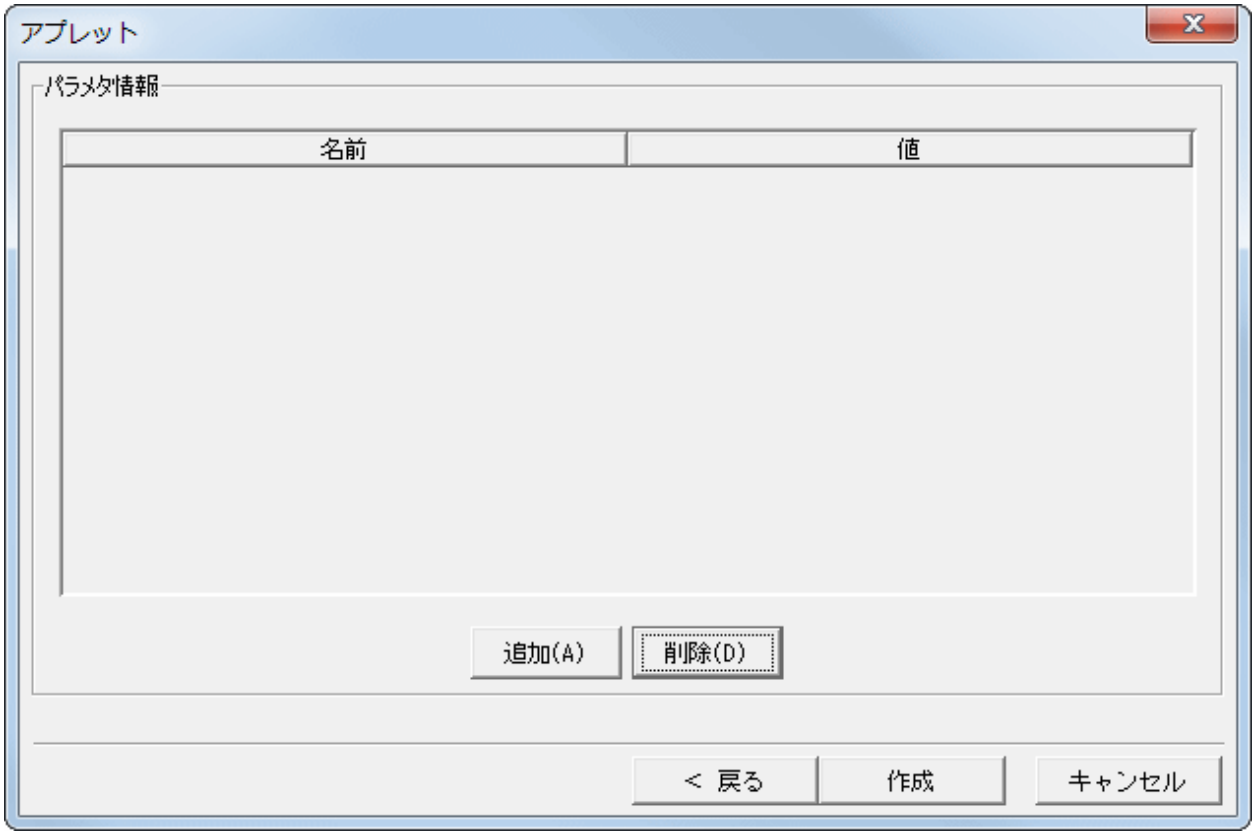


[次へ]をクリックします。

9. [パラメタ情報]ページが表示されます。



10. パラメタは使わないので、削除します。「myItem0」を選択後、[削除]をクリックします。

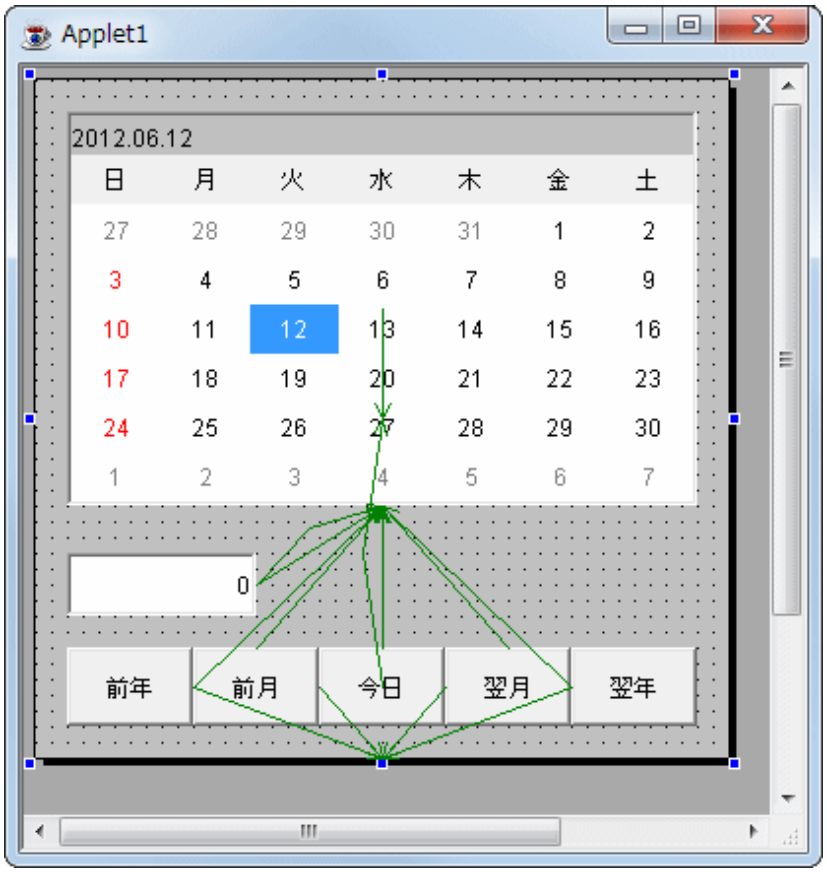


[作成]をクリックします。

11. Javaフォーム定義が起動されます。

アプレットの編集

1. "F.2 クライアントアプリケーション"のLesson1～Lesson3を参考に、アプレットフォームにBeanを配置し、プロパティの設定、イベント処理の記述を行ってください。

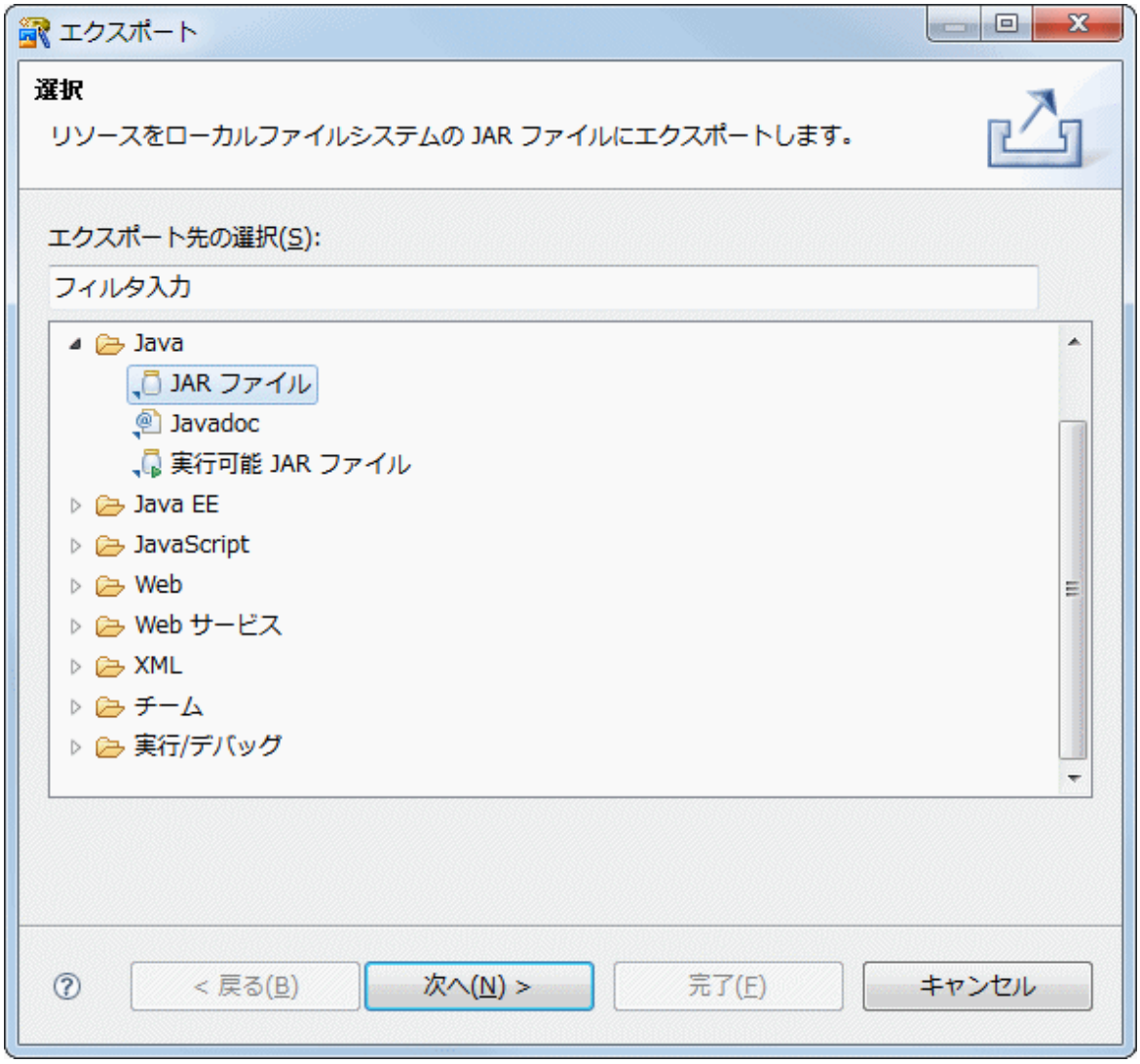


ビルド

1. ビルドは、[プロジェクト]メニューの[自動的にビルド]が有効になっている場合は、Javaファイルなどのリソースファイルを保存すると自動的に実行されます。
   [自動的にビルド]が無効になっている場合は、プロジェクトを選択し右クリックメニューの[プロジェクトのビルド]を選択するか、[プロジェクト]メニューの[プロジェクトのビルド]を選択します。

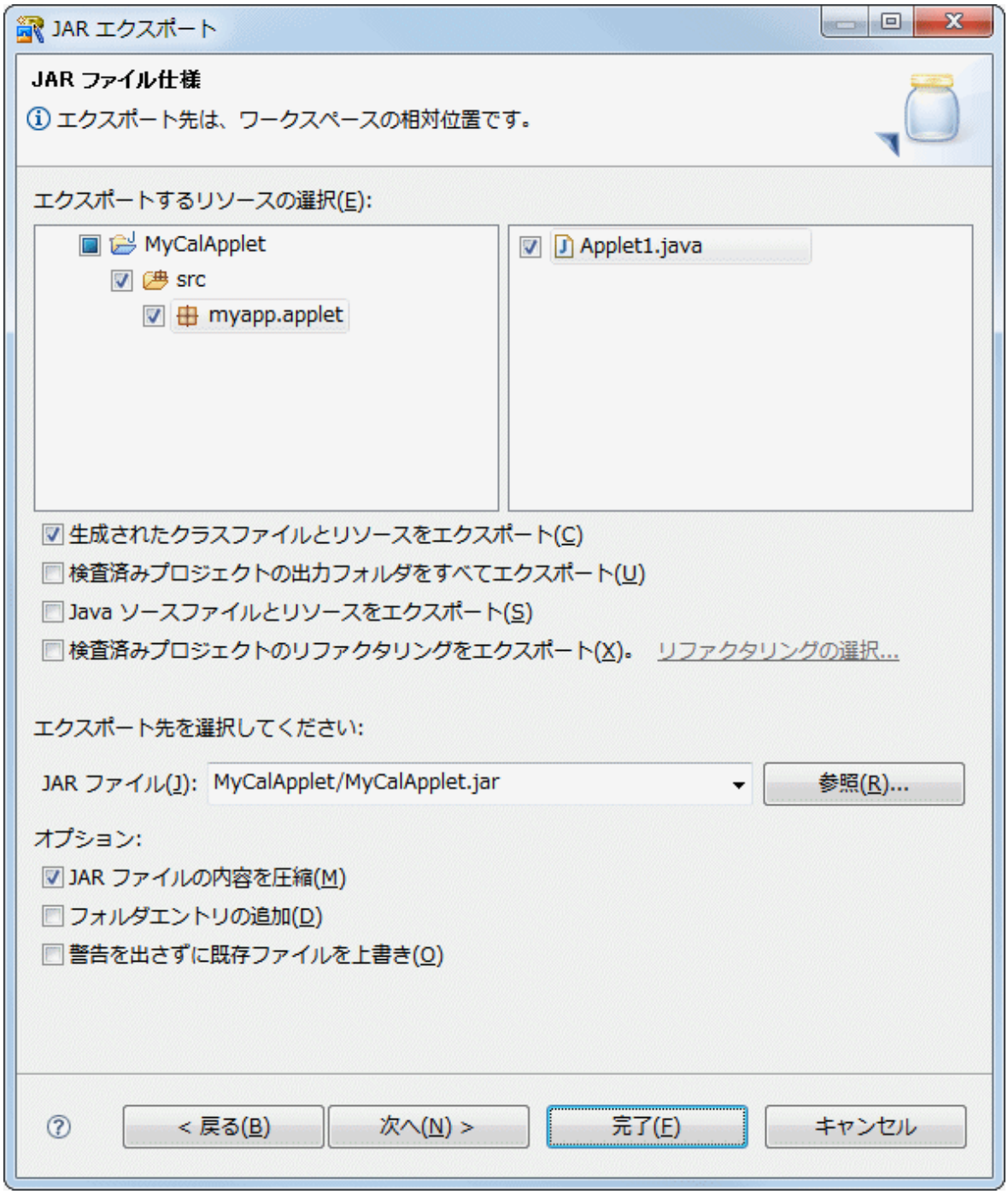
2. JARファイルについては別途必要なため、これを作成します。
   JARファイルの作成は、エクスポートウィザードから行います。

エクスポートウィザードを起動するには、[ファイル] > [エクスポート]を選択します。
エクスポートウィザードから[Java] > [JARファイル]を選択します。

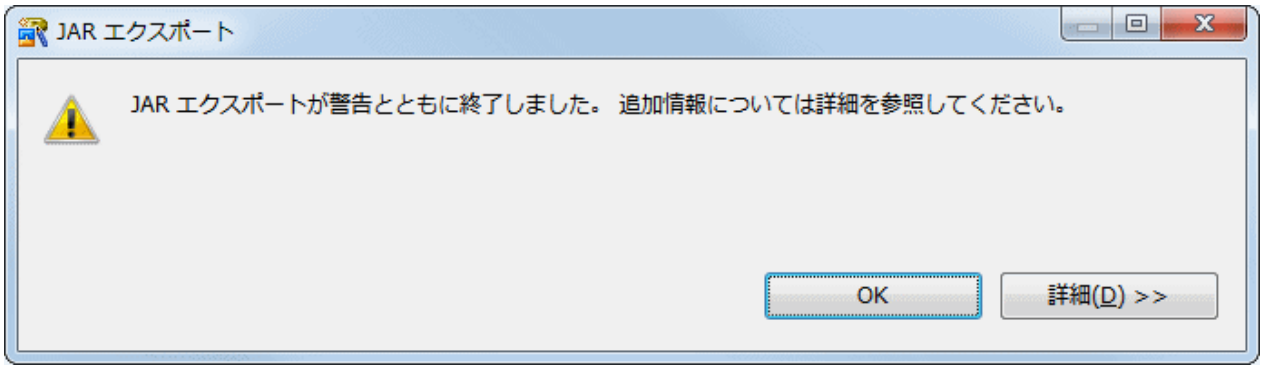


3. JARエクスポートウィザードが表示されます。
[エクスポートするリソースの選択]で[MyCalApplet] > [src] > [myapp.applet]を選択し、以下の設定項目を入力します。
情報を設定後、[完了]をクリックします。

設定項目	設定内容
エクスポートするリソースの選択	Applet1.java
生成されたクラスファイルとリソースをエクスポート	チェックする
JARファイル	MyCalApplet/MyCalApplet.jar
JARファイルの内容を圧縮	チェックする



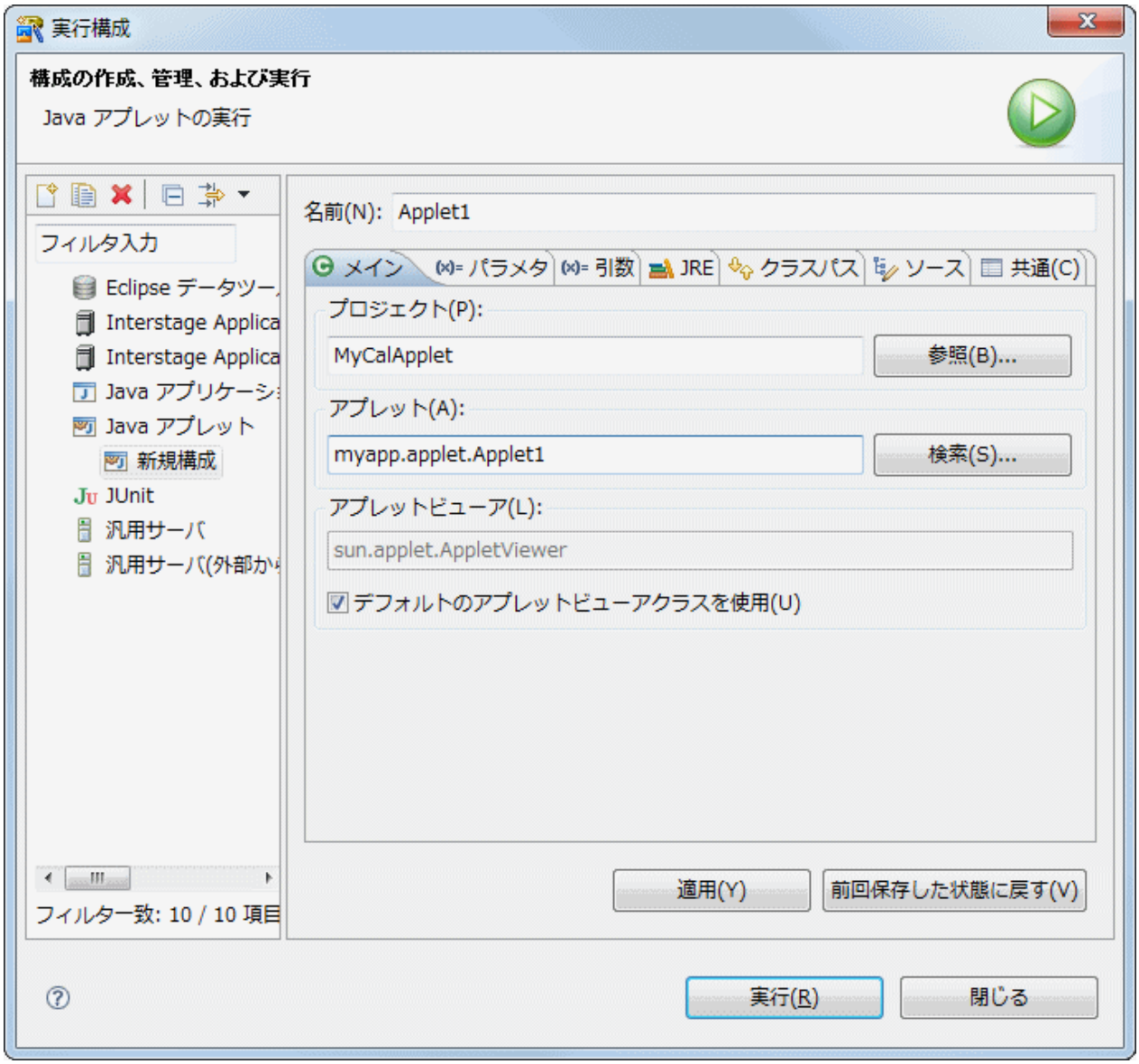
JARエクスポートでは以下のメッセージが出力されますが、問題ありません。



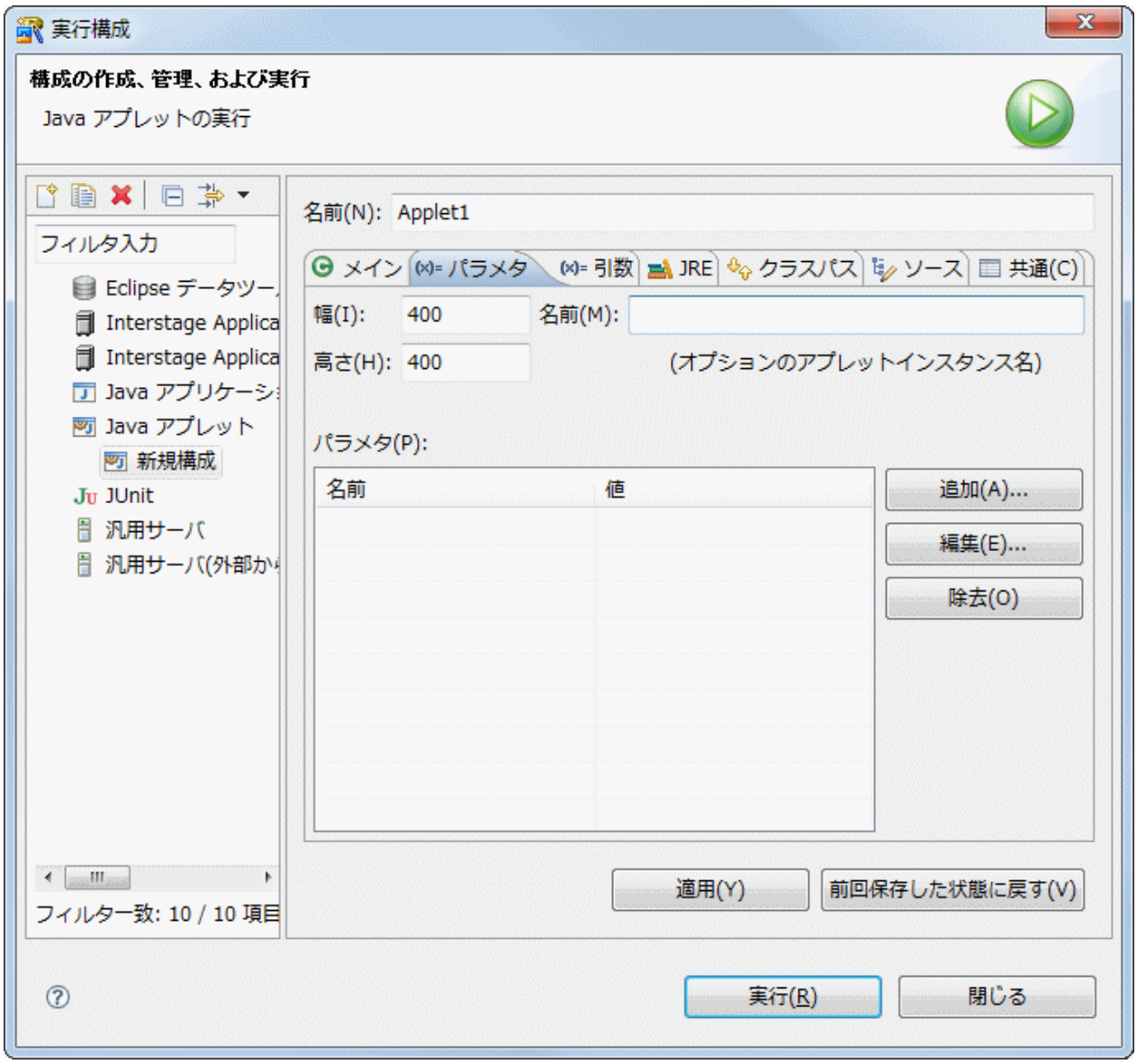
実行

1. 実行するファイル(クラス)を選択します。ワークベンチの[パッケージエクスプローラ]ビューにある[MyCalApplet] > [src] > [myapp.applet] > [Applet1.java]をクリックします。
2. ワークベンチのメニューバーから[実行] > [実行構成...]を選択します。
3. [実行構成]ダイアログボックスが表示されます。
   [構成]ツリーから[Javaアプレット]を選択し、ダブルクリックすると新規に構成が追加されます。
   [メイン]タブで以下を設定します。

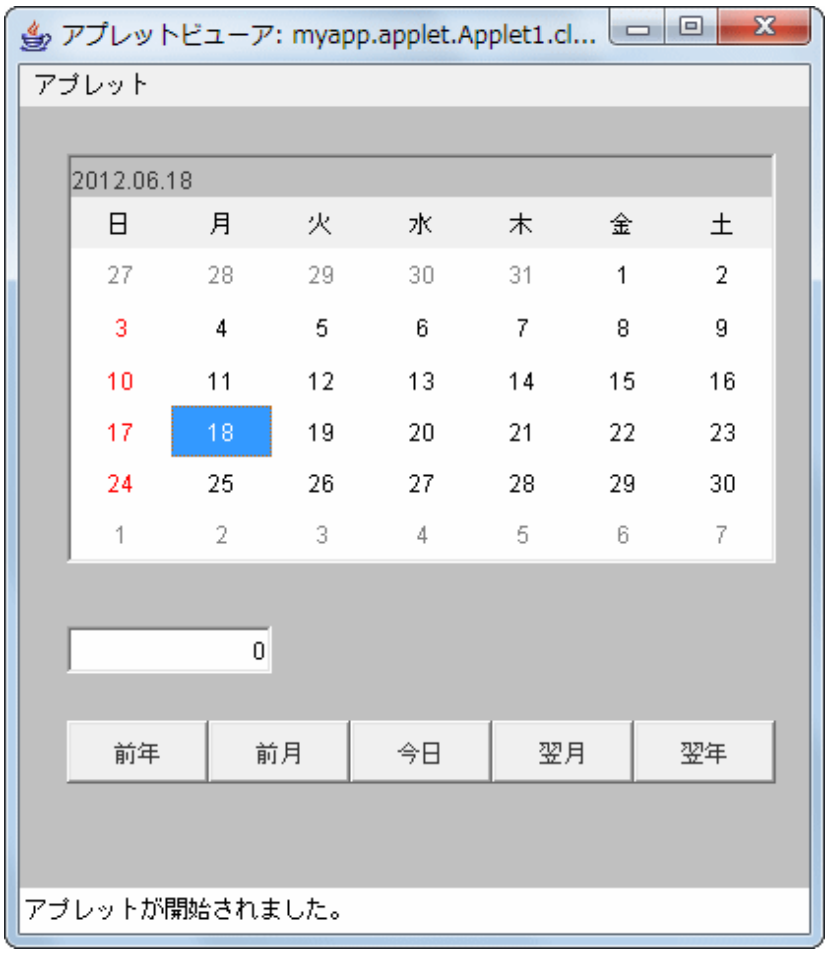
設定項目	設定内容
名前	Applet1
プロジェクト	MyCalApplet
アプレット	myapp.applet.Applet1



4. アプレットの幅と高さを指定します。[パラメタ]タブをクリックして、[幅]と[高さ]にそれぞれ「400」を指定します。

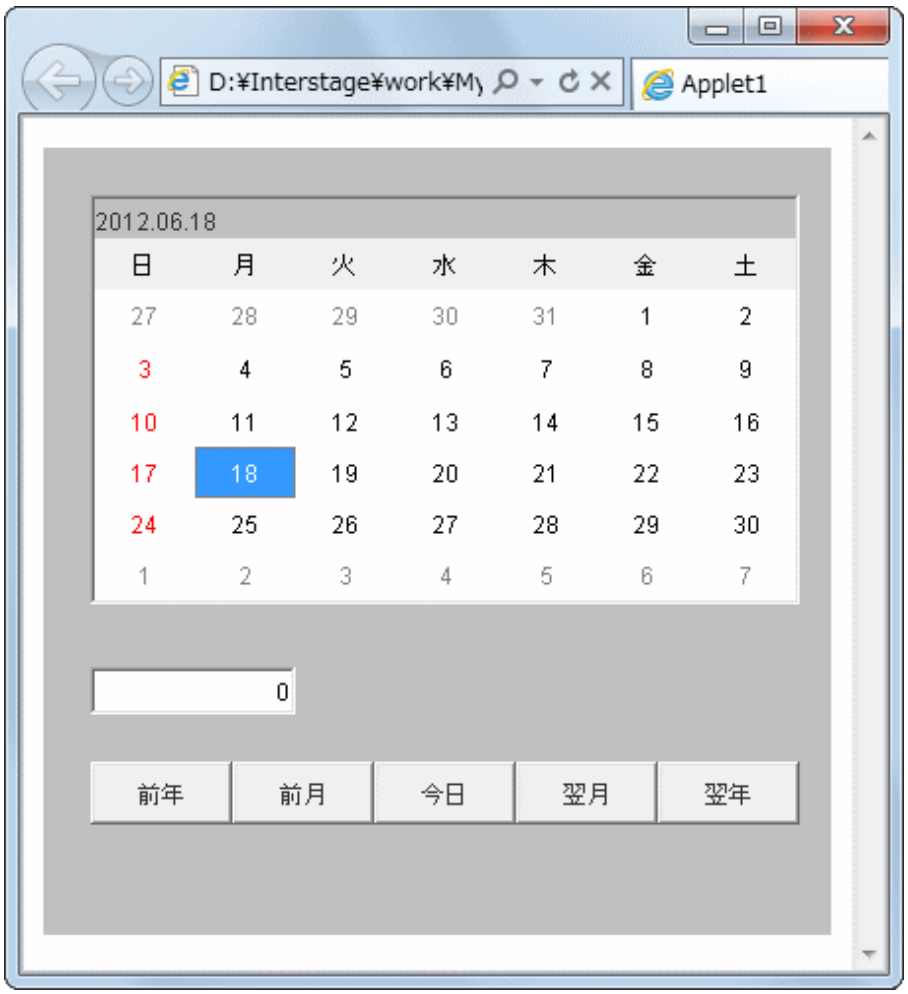


5. [実行]をクリックするとアプレットビューアが起動し、アプレットが表示されます。



6. タイトルバーのクローズボタン([×]ボタン)をクリックして、アプレットビューアを終了します。

7. 実際に、Webブラウザを使って動作確認をしてみます。
   プロジェクトフォルダに格納されている「Applet1-JBKPlugin.html」ファイルをInternet Explorerで開き、同様に動作を確認してみてください。



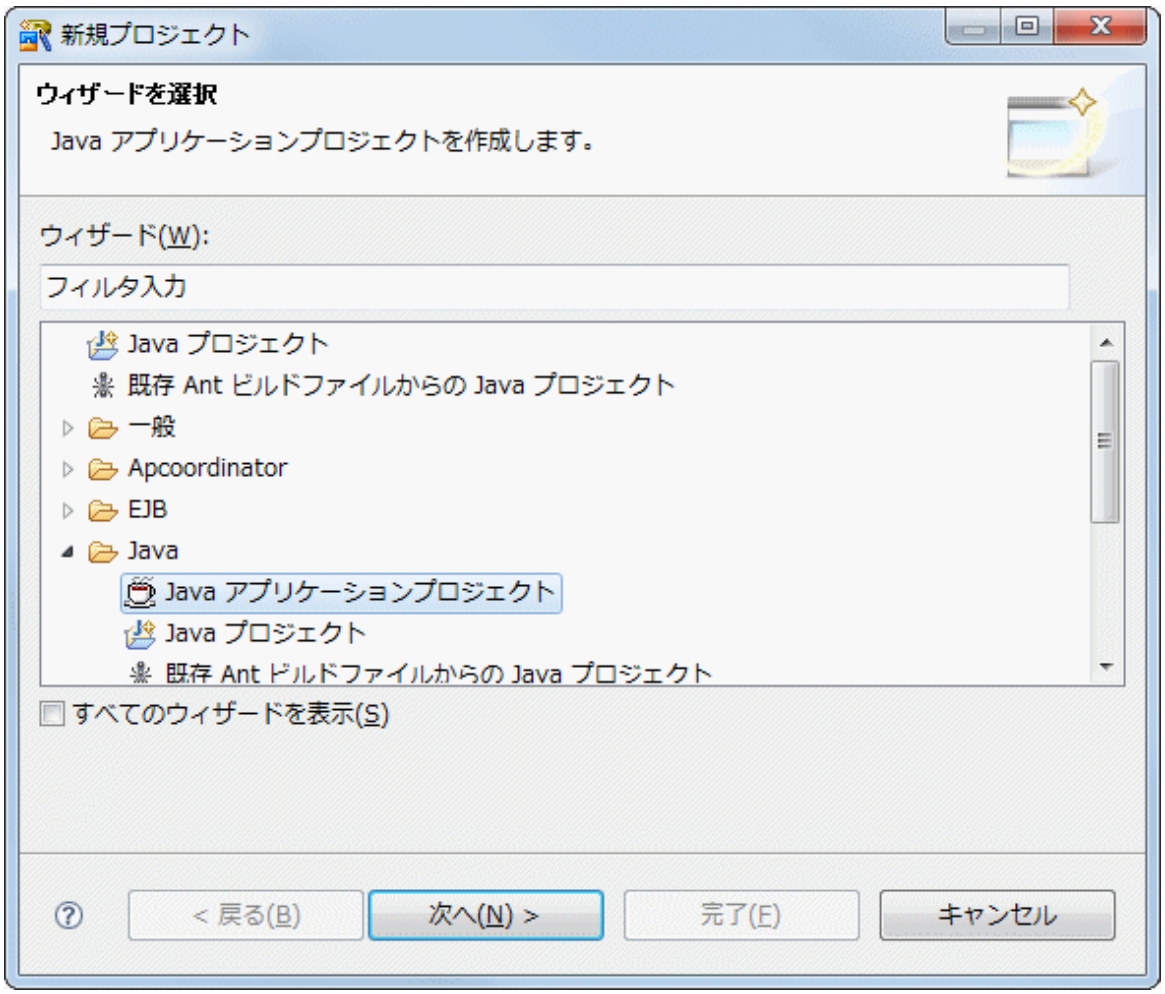
F.3.2　複数の画面を使用するアプレットの開発手順

画面操作ライブラリは、画面遷移が伴うアプリケーションあるいはアプレットにおいて大量画面を管理し、画面の読み込み、表示、削除といった制御機能を持つライブラリです。
ここでは、画面操作ライブラリを使用して、二つの画面間で表示の切り替えを行うアプレットの開発手順を説明します。

Javaアプリケーションプロジェクトの作成

1. ワークベンチを起動します。
2. メニューバーから[ファイル] > [新規] > [プロジェクト]を選択します。

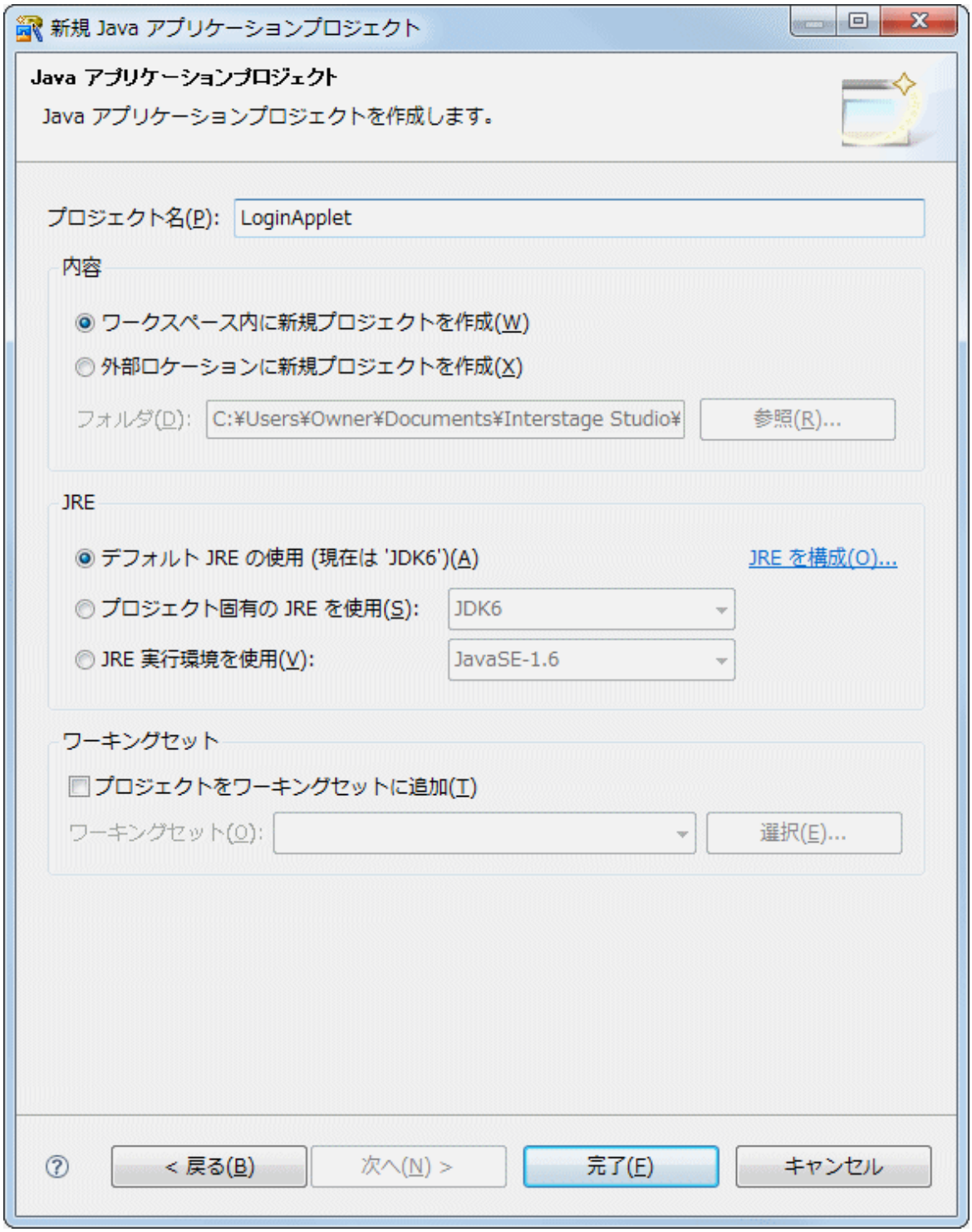
3. [新規プロジェクト]ウィザードが表示されます。ツリーから[Java] > [Javaアプリケーションプロジェクト]を選択します。



[次へ]をクリックします。

4. [Javaアプリケーションプロジェクト]ページが表示されます。このページで入力した項目に従ってプロジェクトが作成されます。以下のように入力します。

設定項目	設定内容
プロジェクト名	LoginApplet

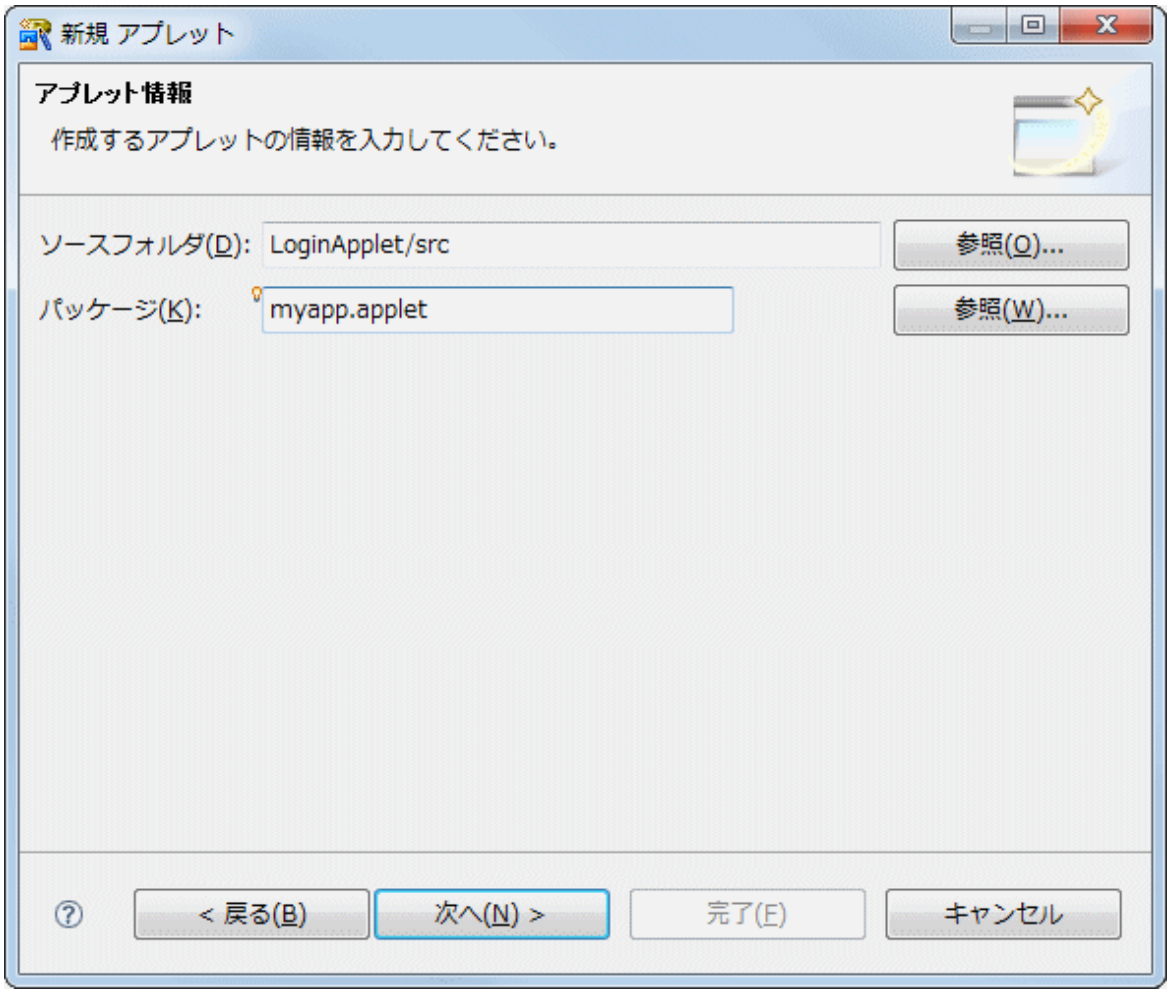


[完了]をクリックします。

5. 次にアプレットを作成します。ワークベンチのメニューバーから、[ファイル] > [新規] > [その他]を選択します。表示される[新規]ウィザードのツリーから、[Java] > [GUI] > [アプレット]を選択します。

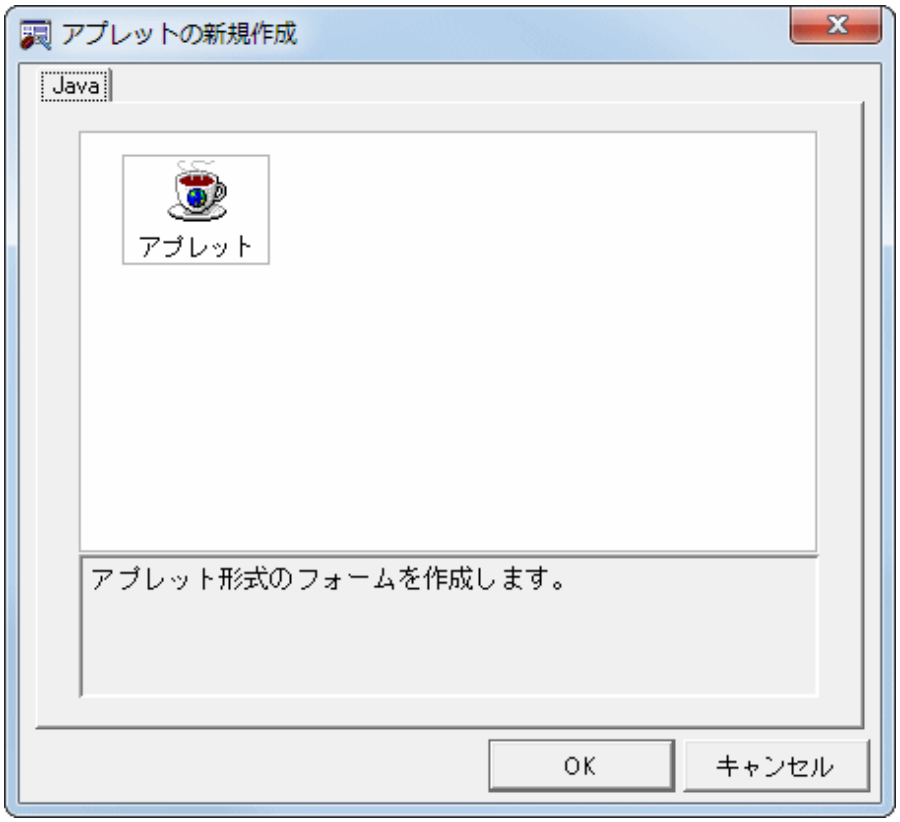
6. [アプレット情報]ページが表示されます。このページでは、以下の情報を入力します。

設定項目	設定内容
パッケージ	myapp.applet

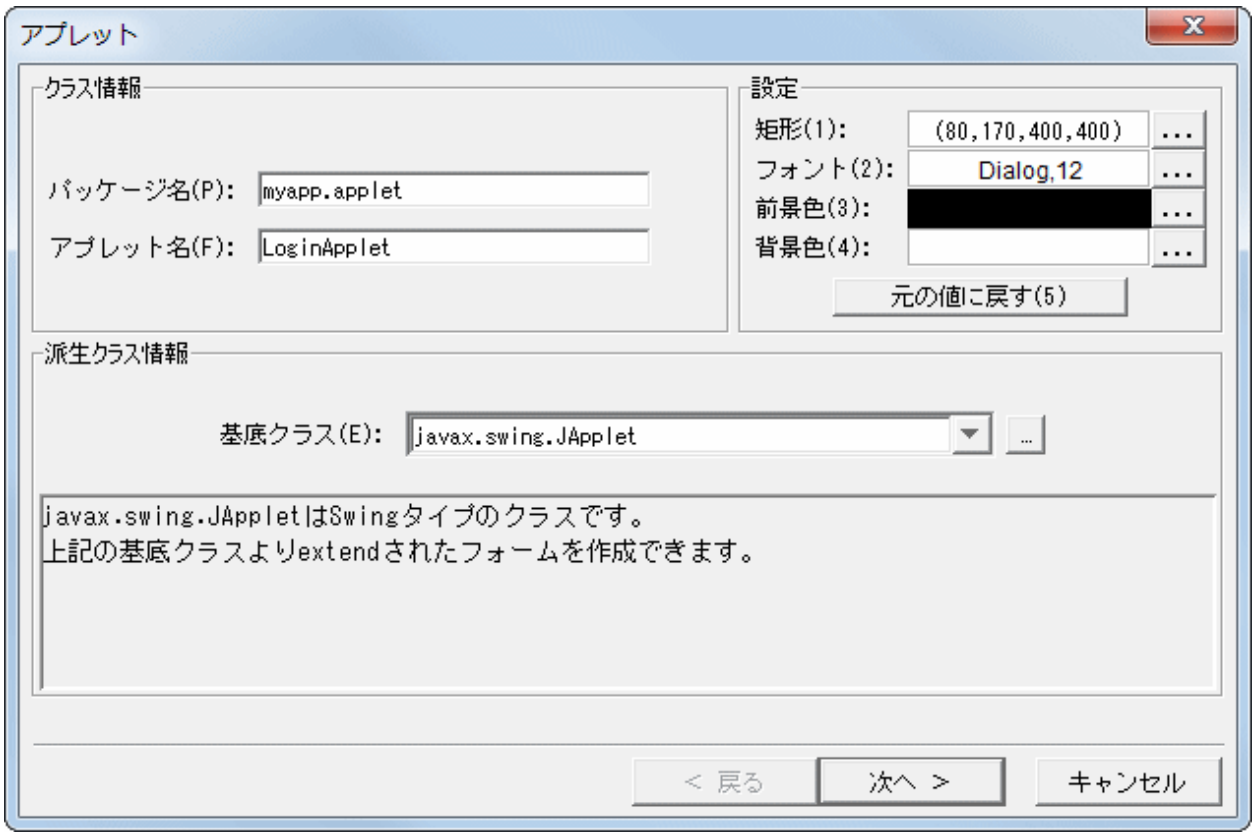

新規 アプレット
アプレット情報
作成するアプレットの情報を入力してください。
ソースフォルダ(D):
LoginApplet/src
参照(O)...
パッケージ(K):
myapp.applet
参照(W)...
< 戻る(B)
次へ(N) >
完了(F)
キャンセル

[次へ]をクリックします。

7. [アプレットの新規作成]ダイアログボックスが表示されます。
[Java]タブの[アプレット]を選択し、[OK]をクリックします。



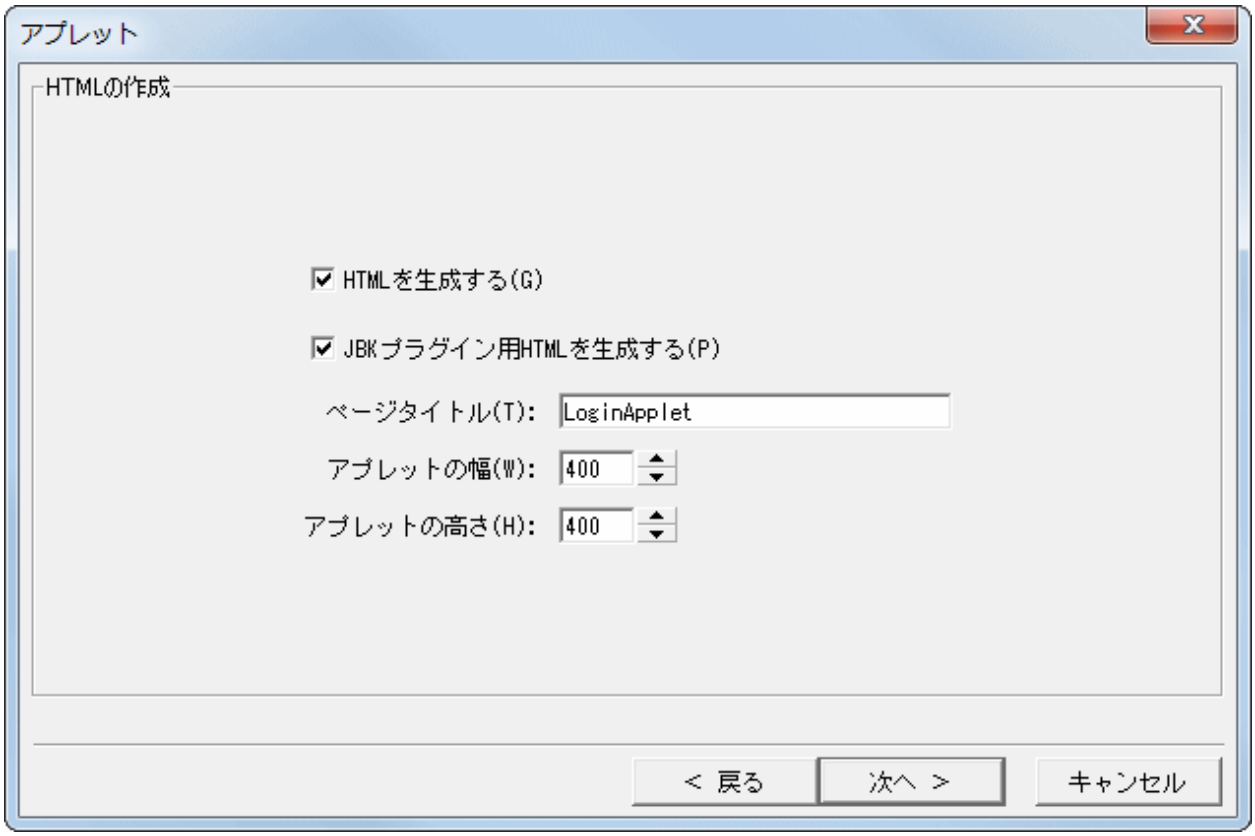
8. アプレットクラスの情報を入力するページが表示されます。
アプレット名に「LoginApplet」と指定します。
基底クラスには[javax.swing.JApplet]を選択します。



[次へ]をクリックします。

9. [HTMLの作成]ページが表示されます。ここではアプレットを貼るHTMLのひな型を作成することができます。
以下のように指定します。

設定項目	設定内容
HTMLを生成する	チェックする
JBKプラグイン用HTMLを生成する	チェックする
ページタイトル	LoginApplet
アプレットの幅	400
アプレットの高さ	400



[次へ]をクリックします。

10. [パラメタ情報]ページが表示されます。
前回と同じように、パラメタは使わないので削除します。
「myItem0」を選択後、[削除]をクリックします。[作成]をクリックします。

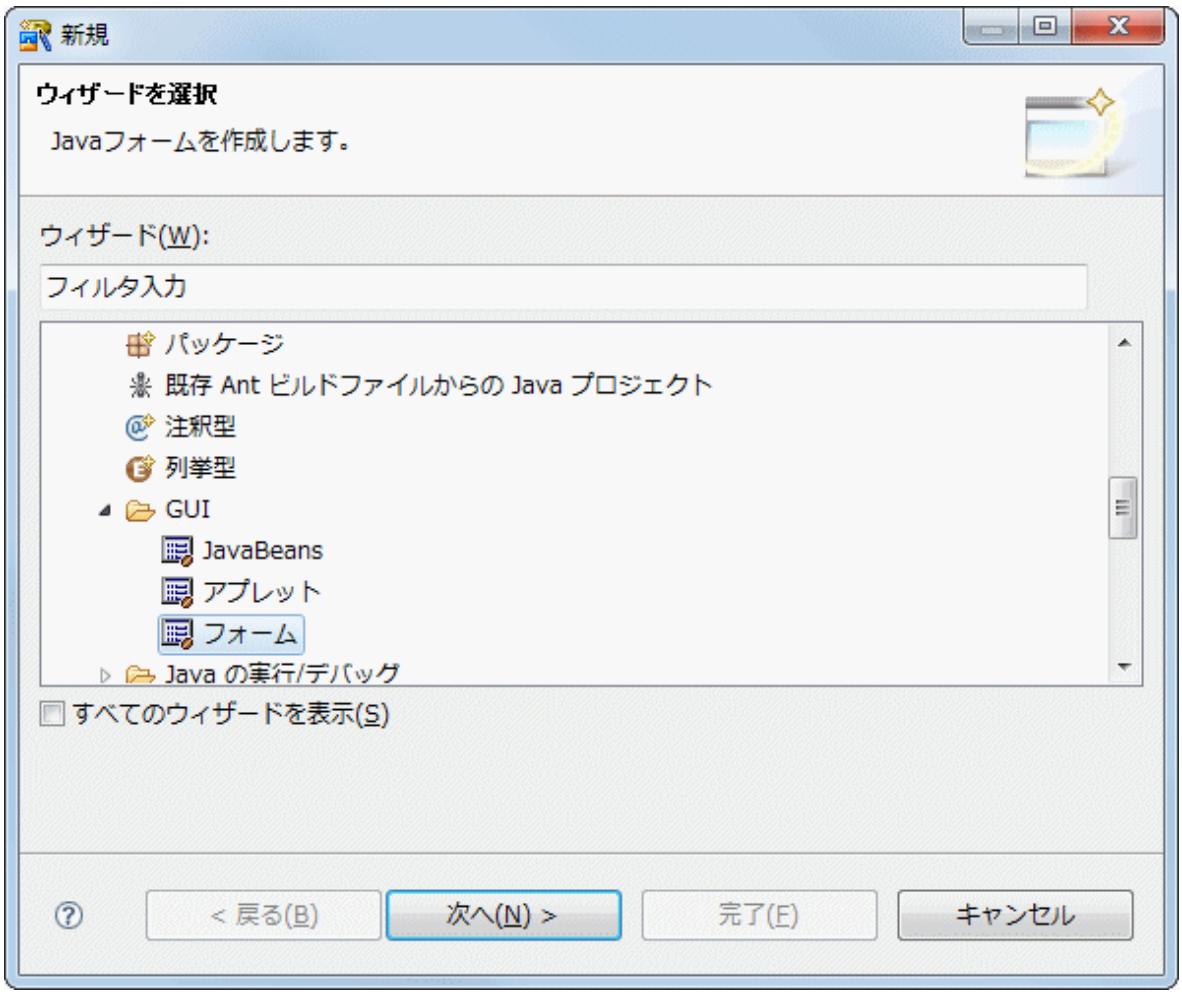
11. Javaフォーム定義が起動されます。
画面操作ライブラリは、アプレットに画面操作クラス(JFLCardPanelの派生クラス)を貼り付けて利用しますが、この作業は後ほど行いますので、Javaフォーム定義は終了します。
Javaフォーム定義の[ファイル]メニューの[終了]を選択して、アプレットフォームを閉じます。

以上の操作で、新規にプロジェクトが作成されます。

画面制御パネルの作成

1. 画面制御パネルを作成します。
ワークベンチのメニューバーから[ファイル] > [新規] > [その他]を選択します。

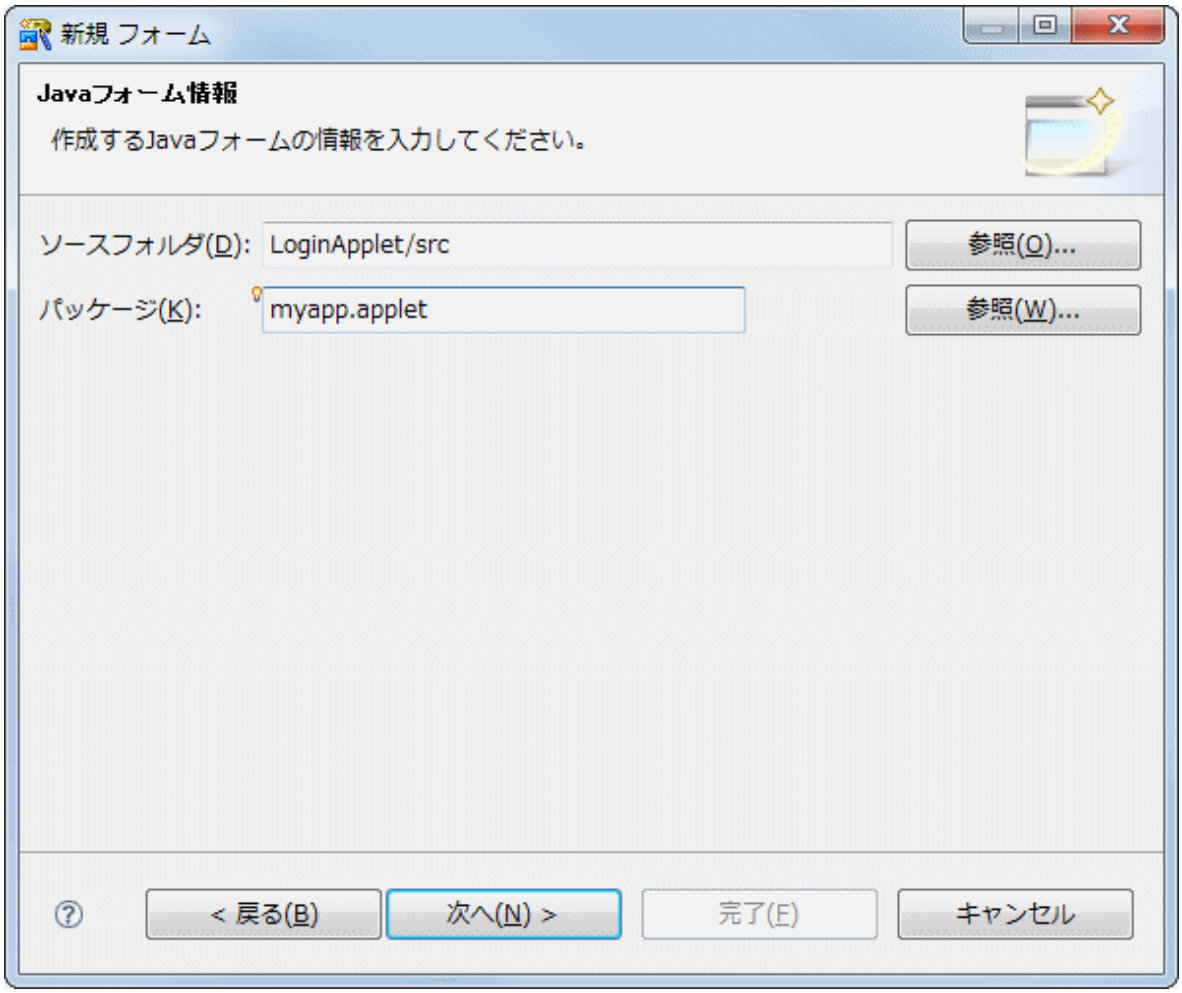
2. [新規]ウィザードが表示されます。ツリーから[Java] > [GUI] > [フォーム]を選択します。



[次へ]をクリックします。

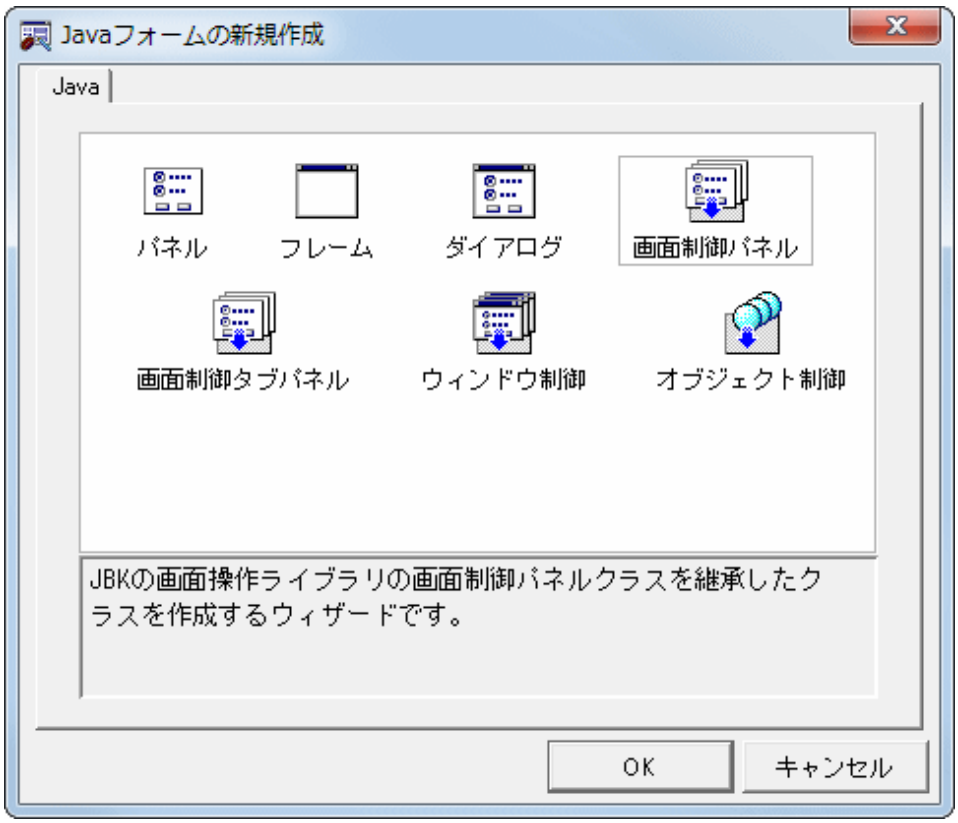
3. [Javaフォーム情報]ページが表示されます。このページでは、以下の情報を入力します。

設定項目	設定内容
パッケージ	myapp.applet

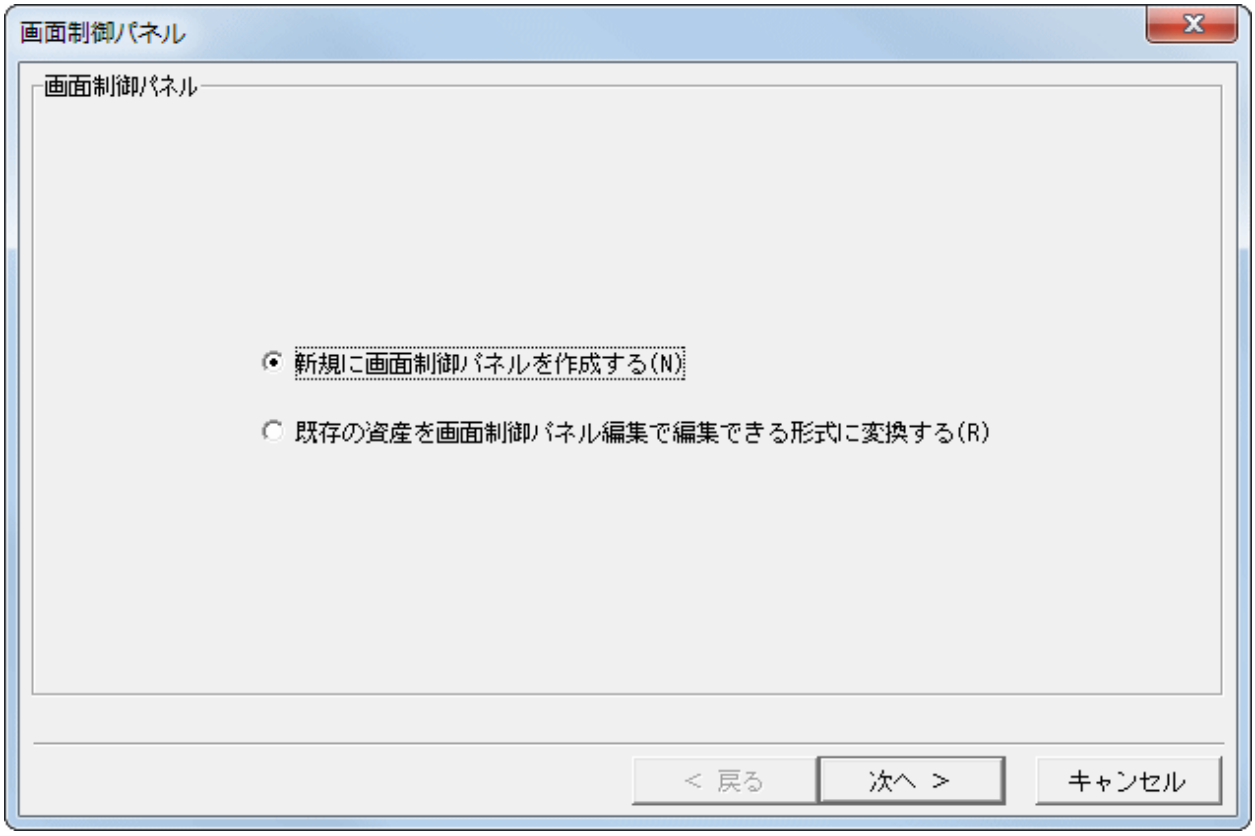

新規 フォーム
Javaフォーム情報
作成するJavaフォームの情報を入力してください。
ソースフォルダ(D):
LoginApplet/src
参照(O)...
パッケージ(K):
myapp.applet
参照(W)...
< 戻る(B)
次へ(N) >
完了(F)
キャンセル

[次へ]をクリックします。

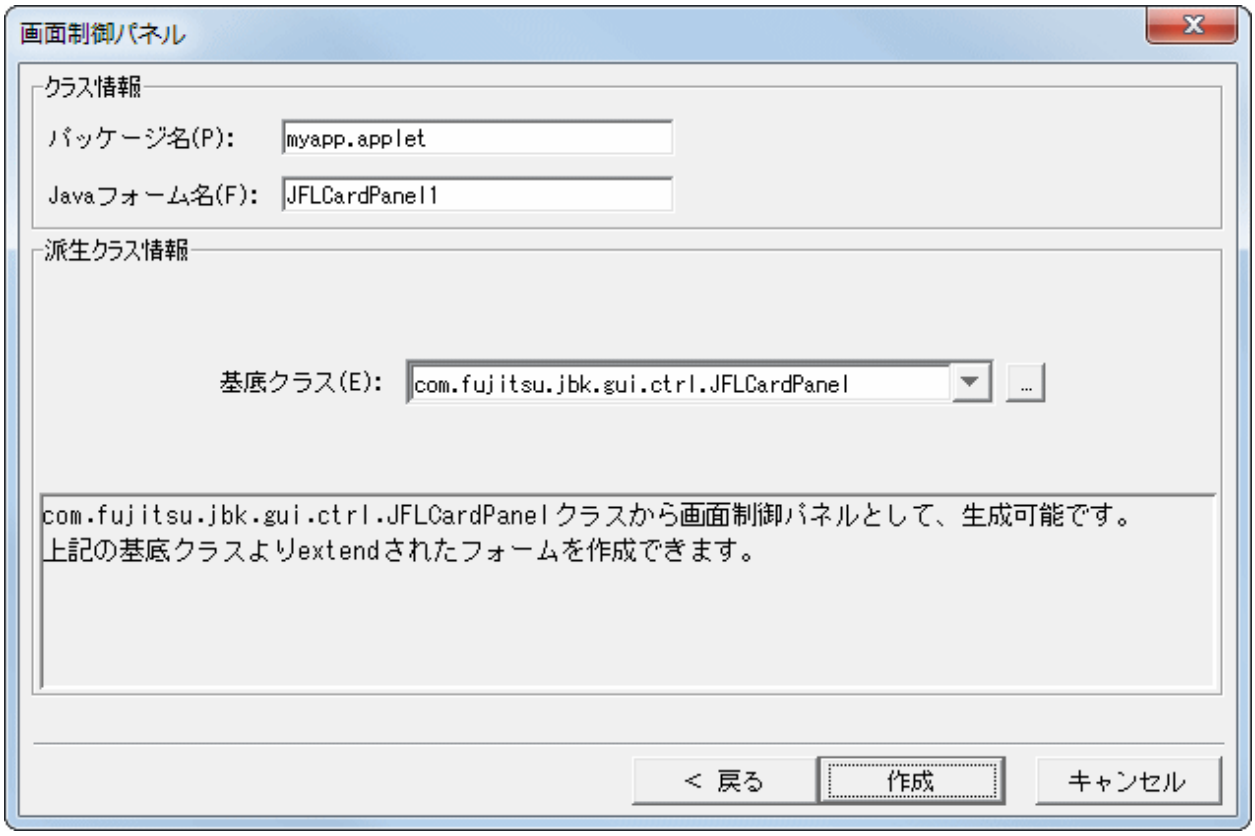
4. [Javaフォームの新規作成]ウィザードが表示されます。[Java]タブの[画面制御パネル]を選択し、[OK]をクリックします。



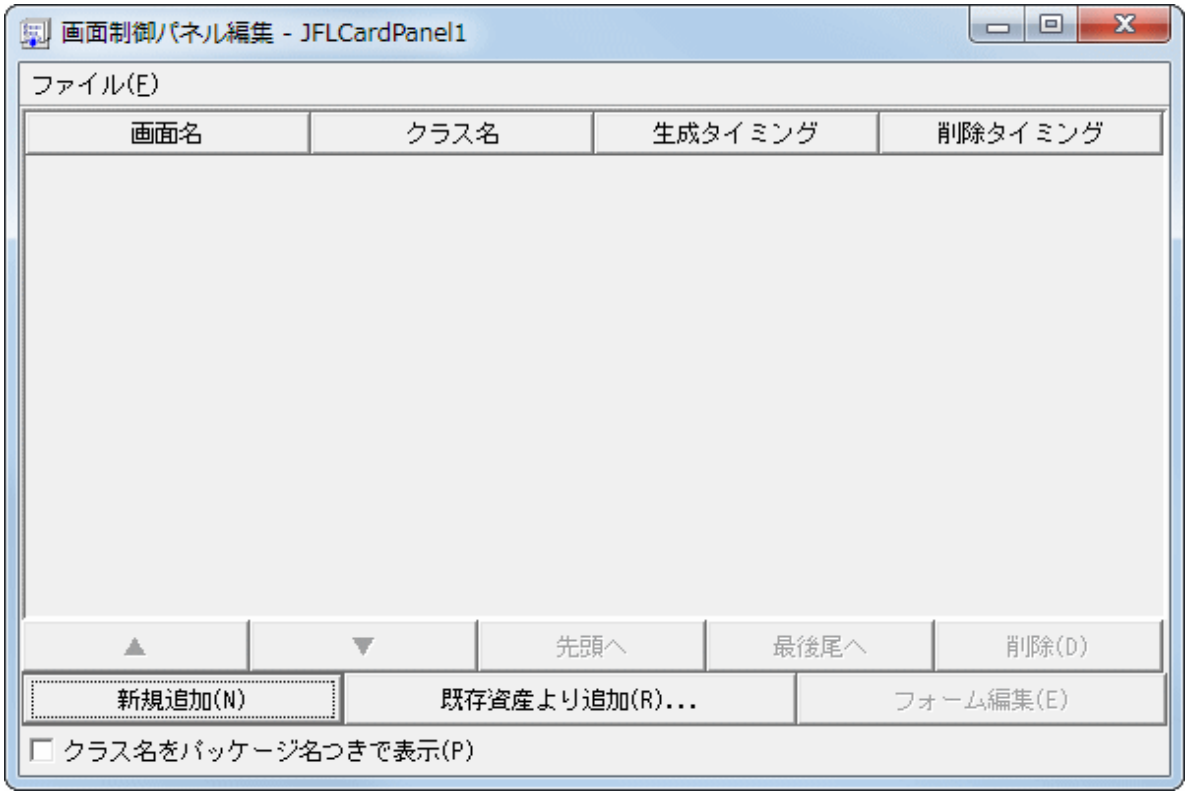
5. ウィザードの最初のページでは新規の資産を作成するか、既存のソースを流用して作成するか選択します。今回は新規に作成しますので、[新規に画面制御パネルを作成する]を選択したら、[次へ]をクリックします。



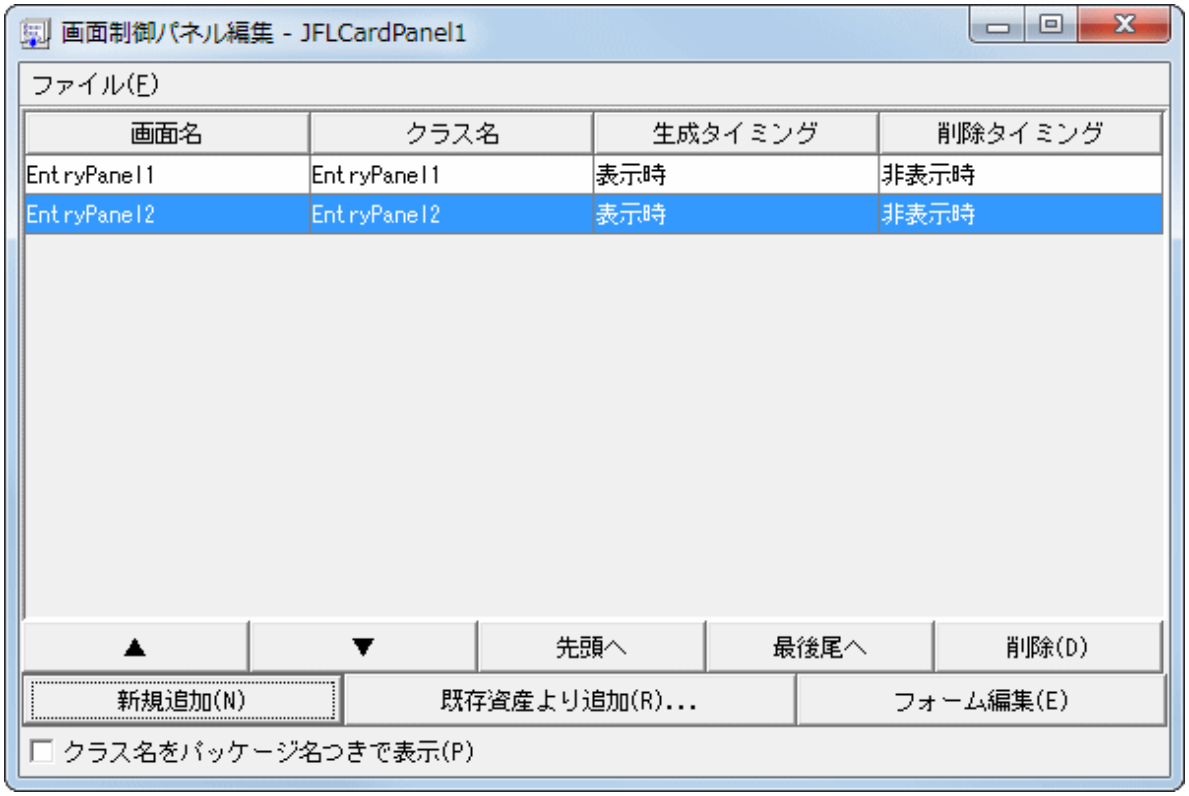
6. クラス情報を入力するページが表示されます。Javaフォーム名に「JFLCardPanel1」、基底クラスに「com.fujitsu.jbk.gui.ctrl.JFLCardPanel」が指定されていることを確認し、[作成]をクリックします。



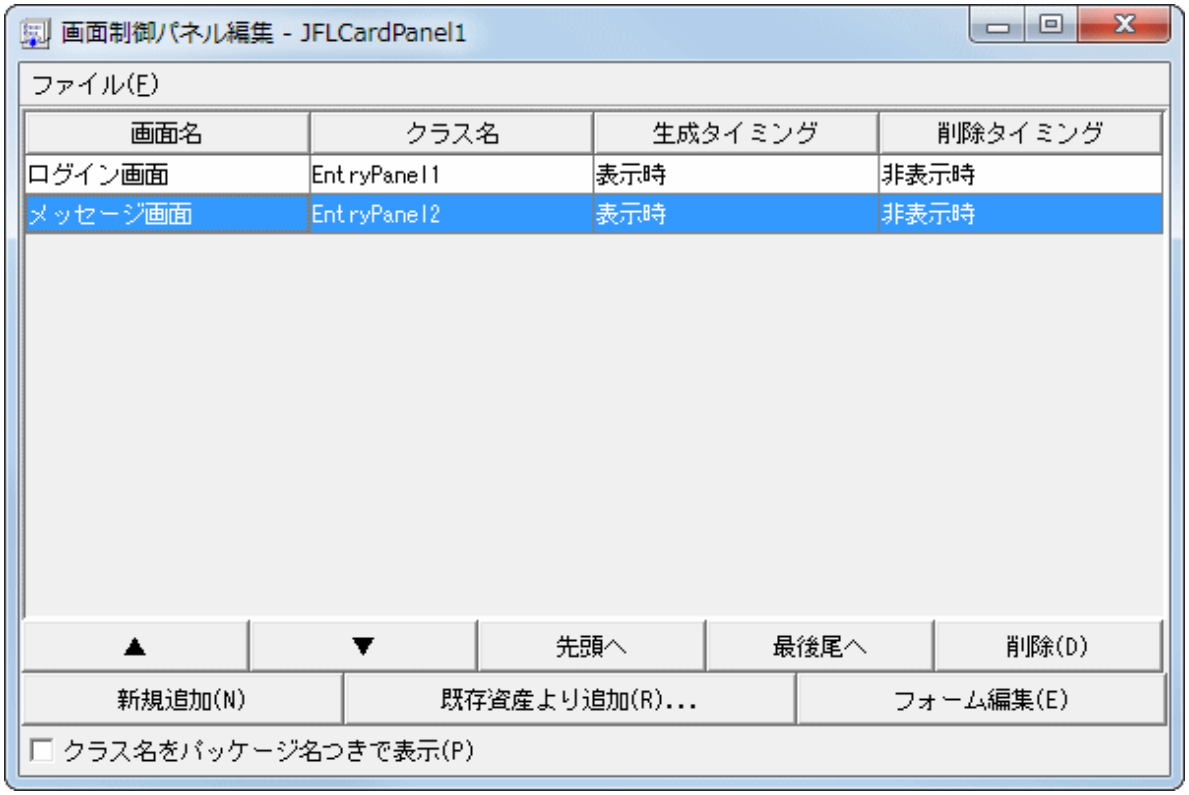
7. 画面制御パネル編集画面が表示されます。この画面では、設定する画面部品の順序や名称、生成/削除のタイミングなどを管理します。今回は、ログイン画面とメッセージ画面の2つの画面を使用します。[新規追加]を2回クリックします。



8. パネルが2枚追加されます。「EntryPanel1」に「ログイン画面」、「EntryPanel2」に「メッセージ画面」を定義することにします。



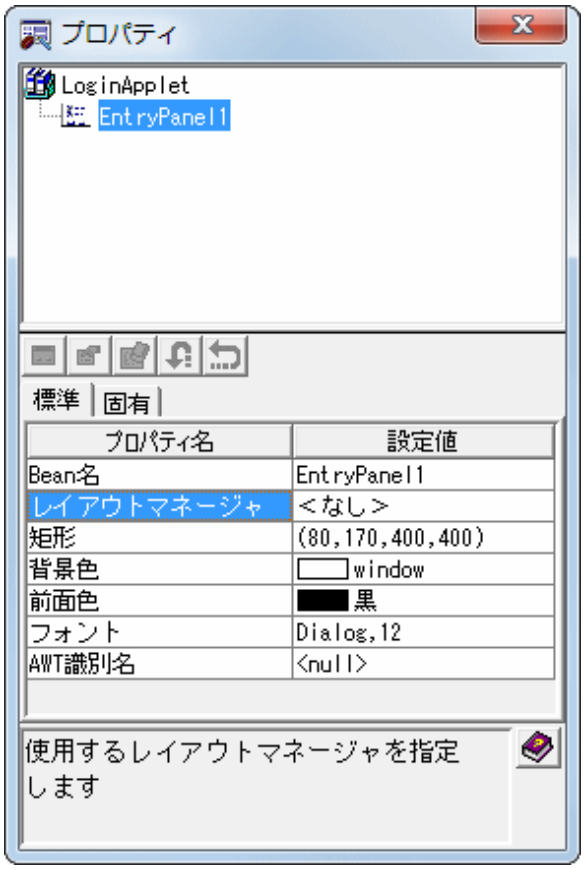
9. 画面名をダブルクリックすると、編集可能な状態になります。「EntryPanel1」を「ログイン画面」、「EntryPanel2」を「メッセージ画面」と変更します。



ログイン画面の作成

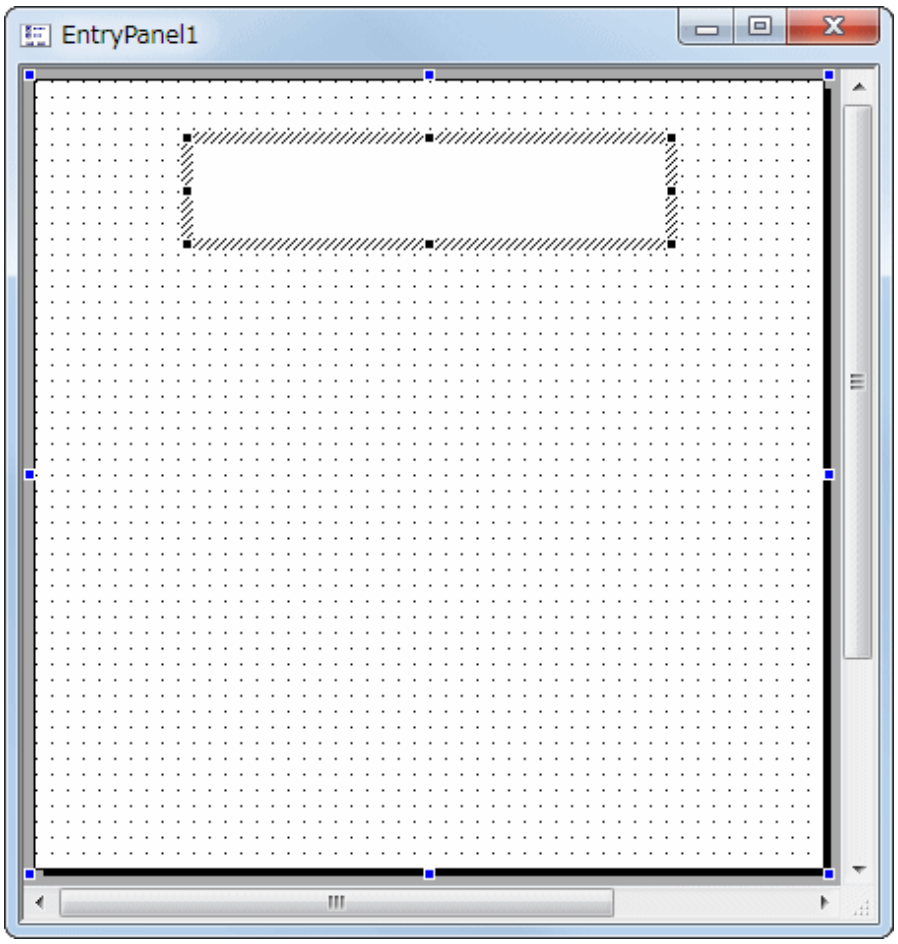
1. ログイン画面を作成します。画面制御パネル編集画面の「ログイン画面」を選択し、[フォーム編集]をクリックします。
   フォームの作成確認画面が表示されますので、[はい]をクリックします。

2. Javaフォーム定義が起動され、EntryPanel1が表示されます。
   初期状態では、EntryPanel1のレイアウトマネージャは「FlowLayout」に設定されています。FlowLayoutはコンポーネントを左から右に順番に配置します。今回はBeanを自由に配置したいので、レイアウトマネージャの設定を解除します。
   プロパティウィンドウが表示されていない場合は、Javaフォーム定義の[表示]メニューから[プロパティ]を選択し、プロパティウィンドウを表示します。
   レイアウトマネージャを「FlowLayout」から「<なし>」に変更します。

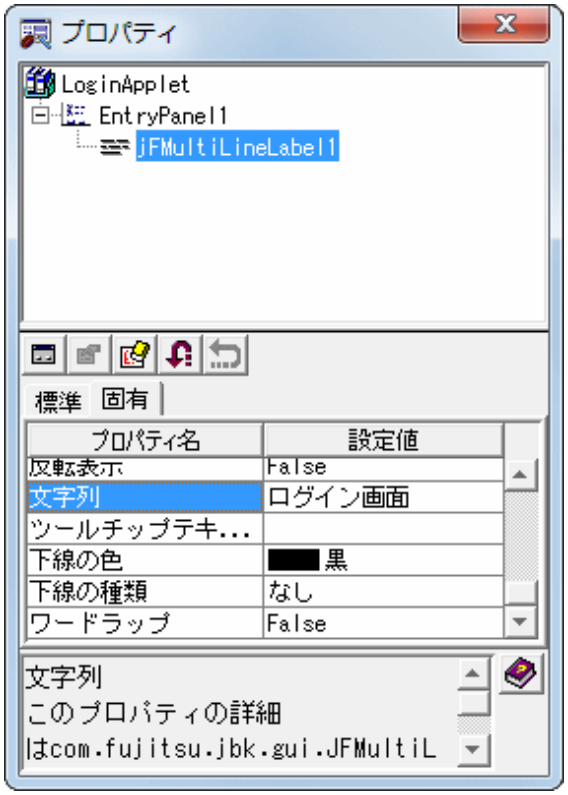


3. Beanを配置します。
   最初に、タイトル文字列「ログイン画面」を配置します。
   オブジェクトパレットの[JBK]が選択されていない場合は、[JBK]をクリックします。

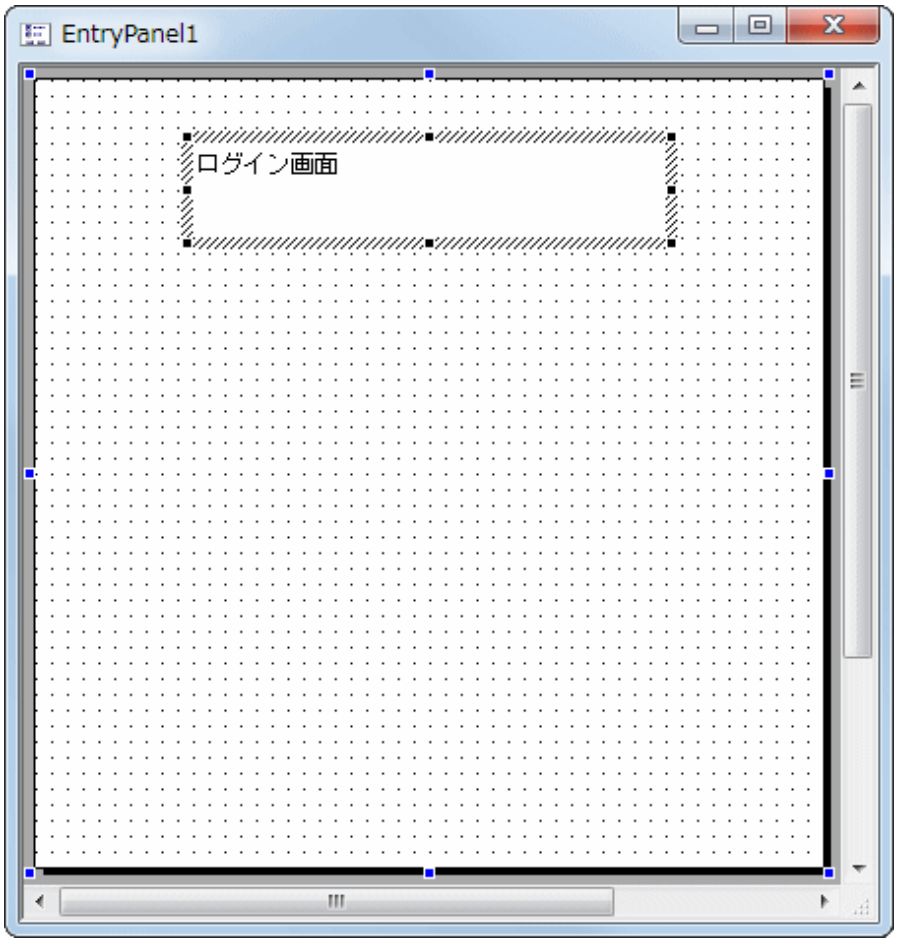
オブジェクトパレットの[複数行ラベル]をクリックし選択します。
Javaフォーム上に配置します。編集対象のJavaフォームの貼り付けたい位置でマウスの左ボタンを押します。
マウスをドラッグし、適当な大きさのところでマウスの左ボタンを離すことにより、フォームにBeanを貼り付けることができます。



表示する文字列を設定します。
配置したBeanが選択されている状態で、プロパティシートの[固有]タブをクリックします。

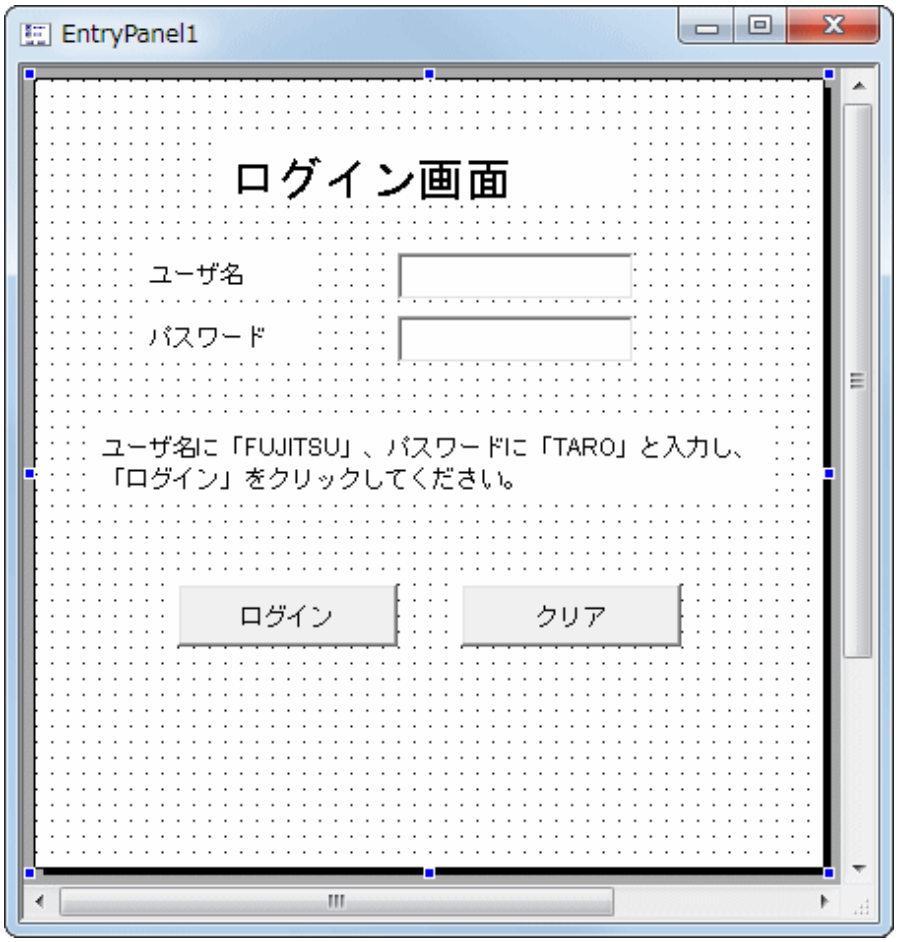


文字列プロパティに「ログイン画面」と入力します。



残りのBeanも配置して、プロパティの設定を行ってください。Beanの矩形(配置位置および大きさ)は、他のBeanと重ならなければ、以下の表のとおりでなくても構いません。

Beanの種類		Bean名	矩形(左端,上端,幅,高さ)	文字列、ラベル	その他
JBK	複数行ラベル	jFMultiLineLabel1	96,32,208,32	ログイン画面	フォント:Dialog,24
JBK	複数行ラベル	jFMultiLineLabel2	56,88,88,24	ユーザ名	
JBK	文字列フィールド	username	184,88,120,24		
JBK	複数行ラベル	jFMultiLineLabel3	56,120,88,24	パスワード	
JBK	文字列フィールド	password	184,120,120,24		エコー文字:*
JBK	複数行ラベル	jFMultiLineLabel4	32,176,344,40	ユーザ名に「FUJITSU」、パスワードに「TARO」と入力し、「ログイン」をクリックしてください。	
JBK	イメージボタン	jFImageButton1	72,256,112,32	ログイン	
JBK	イメージボタン	jFImageButton2	216,256,112,32	クリア	

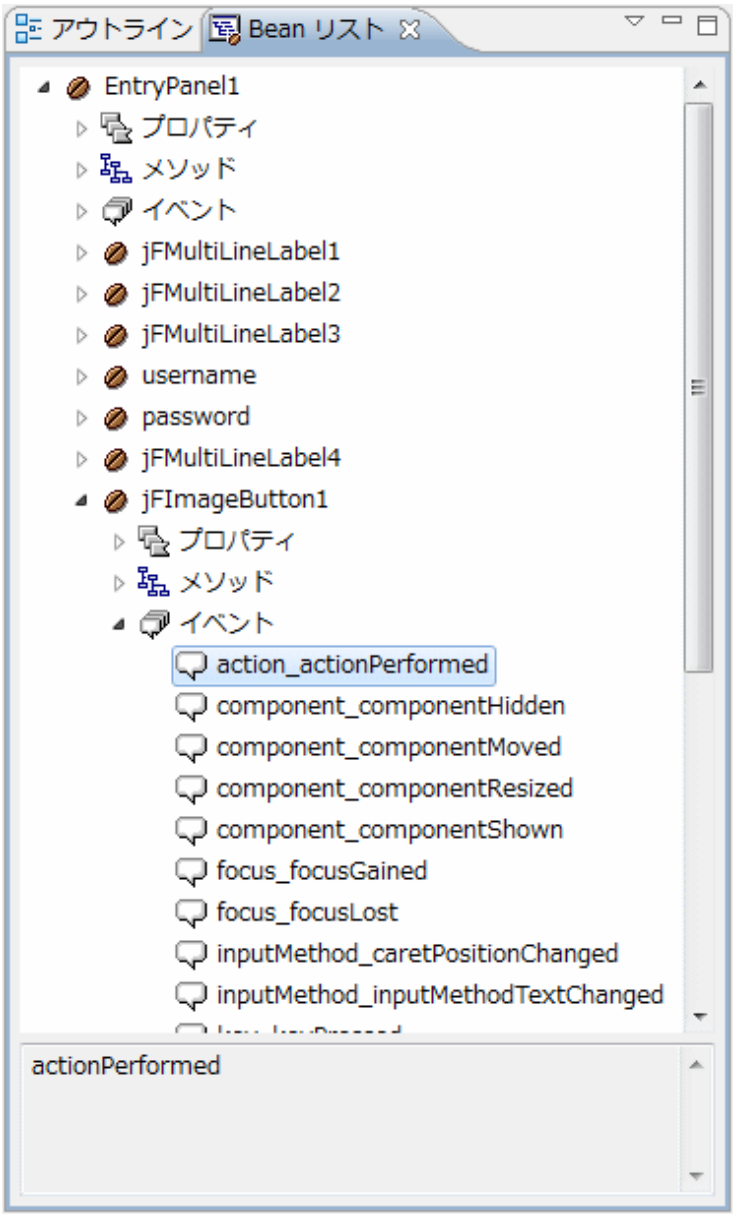


4. イベント処理を記述します。処理はワークベンチ上のJavaエディタで記述します。
   Javaフォーム定義の[表示]メニューの[Javaエディタ]を選択し、Javaエディタをアクティブにします。
   次にイベント処理を記述しますが、イベント処理で「java.util」パッケージに含まれているHashtableクラスを使用しますので、ソースの先頭にimportキーワードを追加します。赤字の部分を追加します。

```
import java.util.Hashtable;
```

5. [ログイン]および[クリア]をクリックしたときに行う処理を記述します。
   ボタンをクリックしたときの処理は、「action_actionPerformed」イベントに処理を記述します。「jFImageButton1」の「action_actionPerformed」に処理を記述する場合は、Beanリストビューで[EntryPanel1] > [jFImageButton1] > [イベント]と展開した後、「action_actionPerformed」を選択し、右クリックして表示されるコンテキストメニューから[挿入]を選択するか、ダブルクリックします。
   Beanリストビューが表示されていない場合には、ワークベンチのメニューバーから[ウィンドウ] > [ビューの表示] > [その他]を選択し、[ビューの表示]ダイアログボックスから[Java]を展開して[Beanリスト]を選択すると表示されます。

イベント処理の作成確認画面が表示された場合は、[はい]をクリックします。イベントメソッドがJavaエディタのソースに追加されます。



追加した「jFImageButton1_action_actionPerformed」メソッドに、[ログイン]をクリックしたときの処理を定義します。JFLEntryPanelのsetDataメソッドを使用して画面データを格納し、setPanelメソッドを使用してメッセージ画面へ遷移するようにします。
赤字の部分を入力します。

```
private void jFImageButton1_action_actionPerformed$(java.awt.event.ActionEvent e) {
    if (!defaultEventProc$(e)) {
        // ここにイベント発生時の処理を記述します。
        Hashtable inputdata = new Hashtable();
        inputdata.put("username", new String(username.getText()));
        inputdata.put("password", new String(password.getText()));
        this.setData(inputdata);
        this.setPanel("メッセージ画面");
    }
}
```

次に、[クリア](jFImageButton2)をクリックしたときの処理を、「jFImageButton2_action_actionPerformed」メソッドに記述します。
Beanリストビューで[jFImageButton2] > [イベント]を展開後、「action_actionPerformed」を選択して右クリックし、表示されるコンテ

キストメニューの[挿入]を選択するか、ダブルクリックします。
イベント処理の作成確認画面が表示された場合は、[はい]をクリックします。
[クリア]をクリックしたときは、二つの文字列フィールドをクリアするように定義します。赤字の部分を入力します。

```
private void jFImageButton2_action_actionPerformed$(java.awt.event.ActionEvent e) {
    if (!defaultEventProc$(e)) {
        // ここにイベント発生時の処理を記述します。
        username.setText("");
        password.setText("");
    }
}
```

6. 以上でログイン画面の定義は終了です。
ワークベンチのメニューバーから[ファイル] > [保存]を選択します。
Javaフォーム定義のメニューバーから [ファイル] > [終了]を選択し、Javaフォームを終了します。

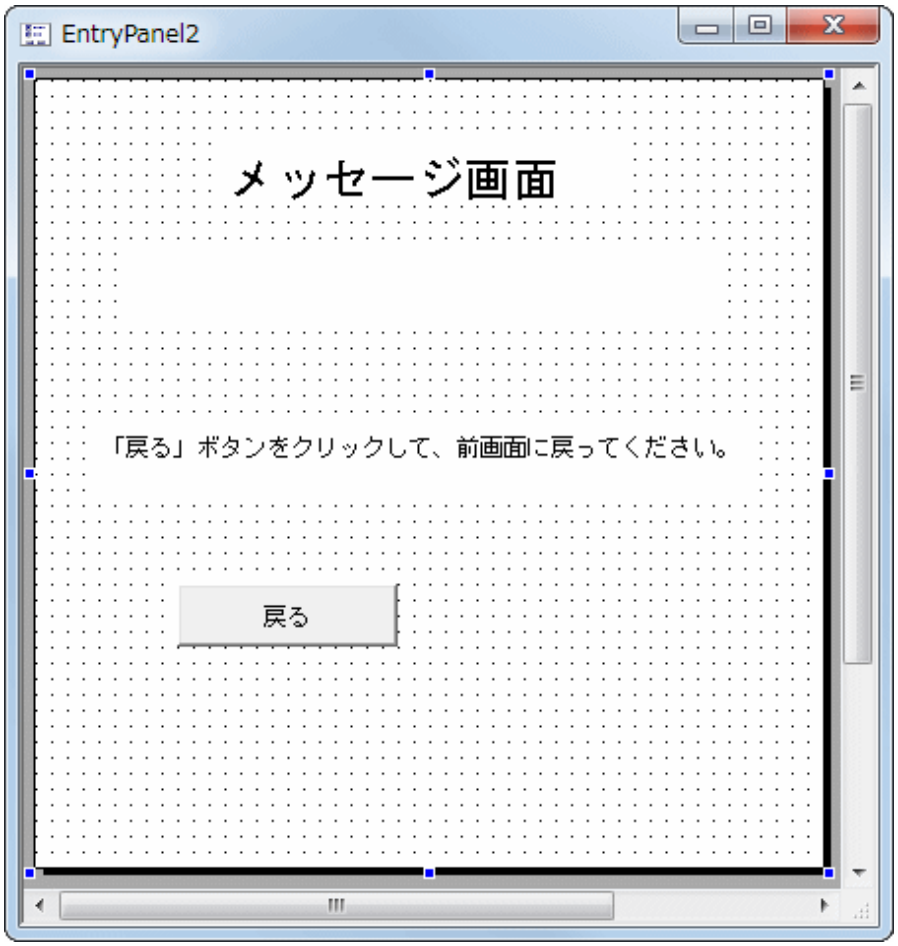
メッセージ画面の作成

1. 次に、メッセージ画面を作成します。画面制御パネル編集画面の「メッセージ画面」を選択し、[フォーム編集]をクリックします。
フォームの作成確認画面が表示されます。[はい]をクリックします。

2. Javaフォーム定義が起動され、EntryPanel2が表示されます。
このパネルの場合もBeanを自由に配置したいので、レイアウトマネージャの設定を解除します。レイアウトマネージャを「FlowLayout」から「<なし>」に変更します。

3. Beanを配置して、プロパティの設定を行ってください。Beanの矩形(配置位置および大きさ)は、他のBeanと重ならなければ、以下の表のとおりでなくても構いません。

Beanの種類		Bean名	矩形(左端,上端,幅,高さ)	文字列	その他
JBK	複数行ラベル	jFMultiLineLabel1	96,32,208,32	メッセージ画面	フォント:Dialog,24
JBK	複数行ラベル	msg	48,88,304,40		
JBK	複数行ラベル	jFMultiLineLabel2	32,176,336,40	「戻る」ボタンをクリックして、前画面に戻ってください。	
JBK	イメージボタン	jFImageButton1	72,256,112,32	戻る	



4. 画面が遷移するときに、ログイン画面の内容を読み取り、メッセージを表示するように手続きを記述します。「java.util」パッケージに含まれているHashtableクラスを使用しますので、ソースの先頭にimportキーワードを追加します。
赤字の部分を追加します。

```
import java.util.Hashtable;
```

次にJFLEntryPanelのgetDataメソッドを利用して、画面に対応した画面データを参照します。
ユーザ名およびパスワードが正しいかチェックしメッセージを複数行ラベルに設定することにします。
以下の処理をクラスの最後に記述します。

```
    public void exposedPanel(String lastPanelName) {
        Hashtable inputdata = (Hashtable) this.getData("ログイン画面");
        String username = (String) inputdata.get("username");
        String password = (String) inputdata.get("password");
        if (username.equals("FUJITSU") && password.equals("TARO")) {
            msg.setText(username + "さん、こんにちは。ログイン成功です。");
        } else {
            msg.setText("ユーザ名またはパスワードが誤っています。");
        }
    }
```

ポイント

Javaフォーム定義が管理する変更禁止ソースについて

Javaフォームは、画面(Bean)の情報やイベント処理を呼び出すソースを含んでいます。このソースは、Javaフォーム定義が管理しているため変更禁止です。
青字のコメントで囲われた部分が、Javaフォーム定義が管理する変更禁止ソースになります。

```
public class EntryPanel2 extends com.fujitsu.jbk.gui.ctrl.JFLEntryPanel {
    //@@Form Design Information start

      ～Javaフォーム定義が管理する変更禁止ソース～

    //@@Form Design Information end
```

次に[戻る]をクリックしたときの処理を「jFImageButton1_action_actionPerformed」メソッドに記述します。
Beanリストビューで[EntryPanel2] > [jFImageButton1] > [イベント]と展開した後、「action_actionPerformed」を選択し、右クリックして表示されるコンテキストメニューから[挿入]を選択するか、ダブルクリックします。
イベント処理の作成確認画面が表示された場合は、[はい]をクリックします。
[戻る]をクリックしたときは、setPanelメソッドを使用してログイン画面へ遷移するように定義します。
赤字の部分を記述します。

```
    private void jFImageButton1_action_actionPerformed$(java.awt.event.ActionEvent e) {
        if (!defaultEventProc$(e)) {
            // ここにイベント発生時の処理を記述します。
            this.setPanel("ログイン画面");
        }
    }
```

5. 以上でメッセージ画面の定義は終了です。
   ワークベンチのメニューバーから[ファイル] > [保存]を選択します。
   Javaフォーム定義のメニューバーから [ファイル] > [終了]を選択し、Javaフォームを終了します。

6. 画面制御パネル編集を終了します。
   画面制御パネル編集のメニューバーから[ファイル] > [終了]を選択します。

画面操作クラスをアプレットに貼り付ける

1. 画面操作クラス(画面制御パネル)は、LoginAppletクラスのinitUserイベントで、addメソッドを使用してアプレットに貼り付けます。
   処理はワークベンチ上のJavaエディタで記述します。
   LoginAppletクラスのinitUser$()メソッド内に赤字の部分を記述します。

```
    protected void initUser$() {
        // この位置にユーザ定義初期化処理を記述します。
        JFLCardPanel1 cardPanel = new JFLCardPanel1();
        cardPanel.setBounds(0,0,400,400);
        getContentPane().add(cardPanel);
    }
```

2. LoginAppletクラスを保存します。ワークベンチのメニューバーから[ファイル] > [保存]を選択します。

ビルド

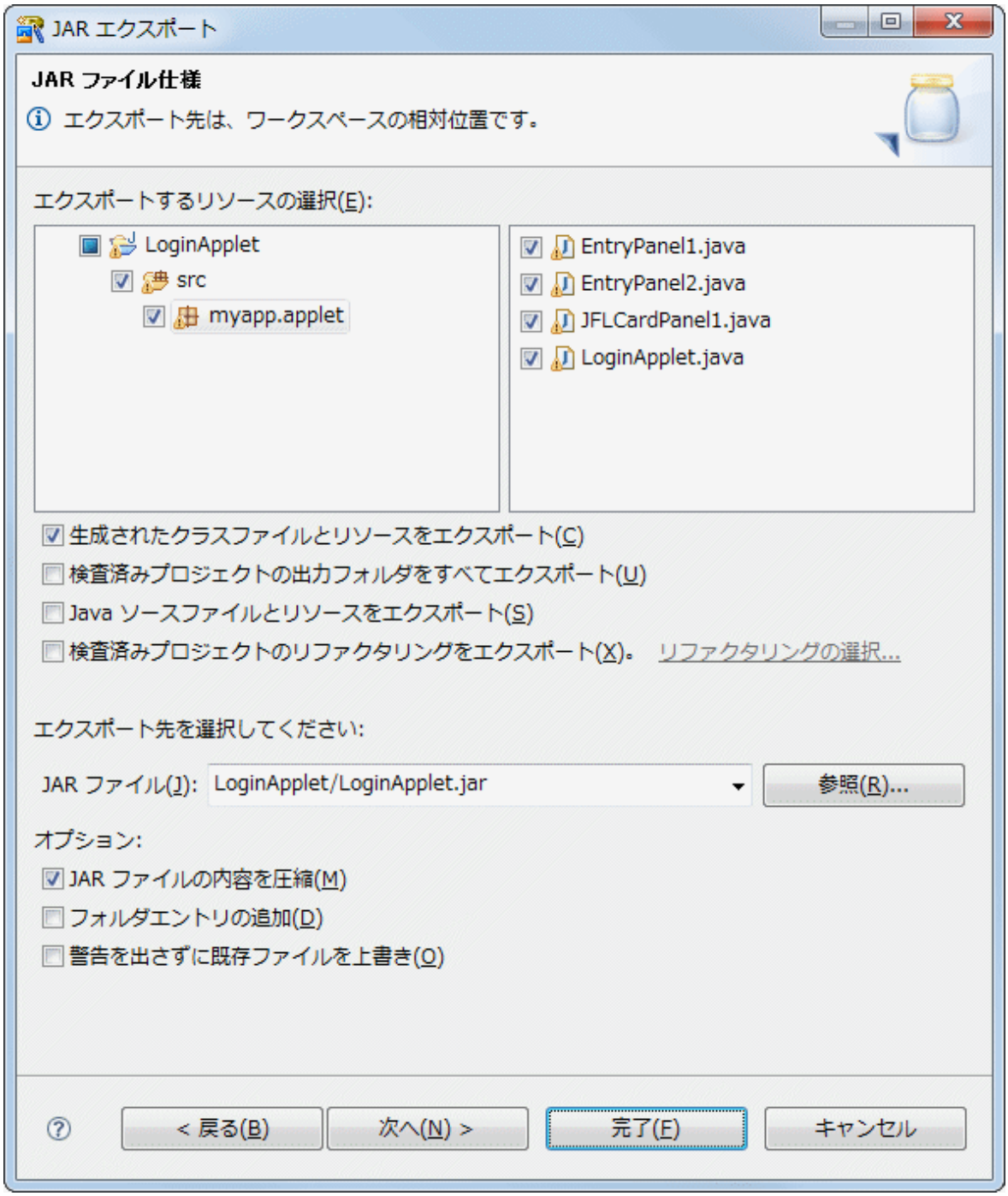
1. ビルドは、[プロジェクト]メニューの[自動的にビルド]が有効になっている場合は、Javaファイルなどのリソースファイルを保存すると自動的に実行されます。
   [自動的にビルド]が無効になっている場合は、プロジェクトを選択し右クリックメニューの[プロジェクトのビルド]を選択するか、[プロジェクト]メニューの[プロジェクトのビルド]を選択します。

2. JARファイルについては別途必要なため、これを作成します。
   JARファイルの作成は、エクスポートウィザードから行います。

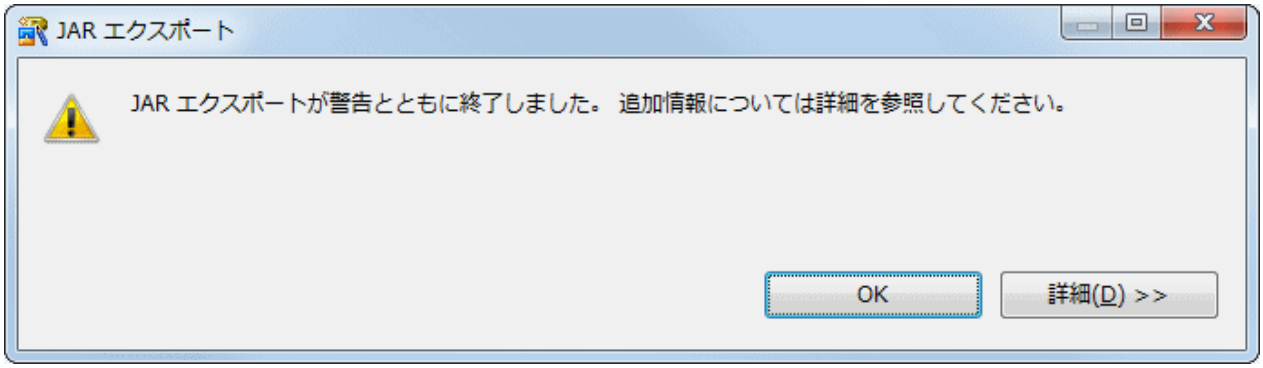
エクスポートウィザードを起動するには、[ファイル] > [エクスポート]を選択します。
エクスポートウィザードから[Java] > [JARファイル]を選択します。

3. JARエクスポートウィザードが表示されます。
[エクスポートするリソースの選択]で[LoginApplet] > [src] > [myapp.applet]を選択し、以下の設定項目を入力します。
情報を設定後、[完了]をクリックします。

設定項目	設定内容
エクスポートするリソースの選択	EntryPanel1.java EntryPanel2.java JFLCardPanel1.java LoginApplet.java
生成されたクラスファイルとリソースをエクスポート	チェックする
JARファイル	LoginApplet/LoginApplet.jar
JARファイルの内容を圧縮	チェックする



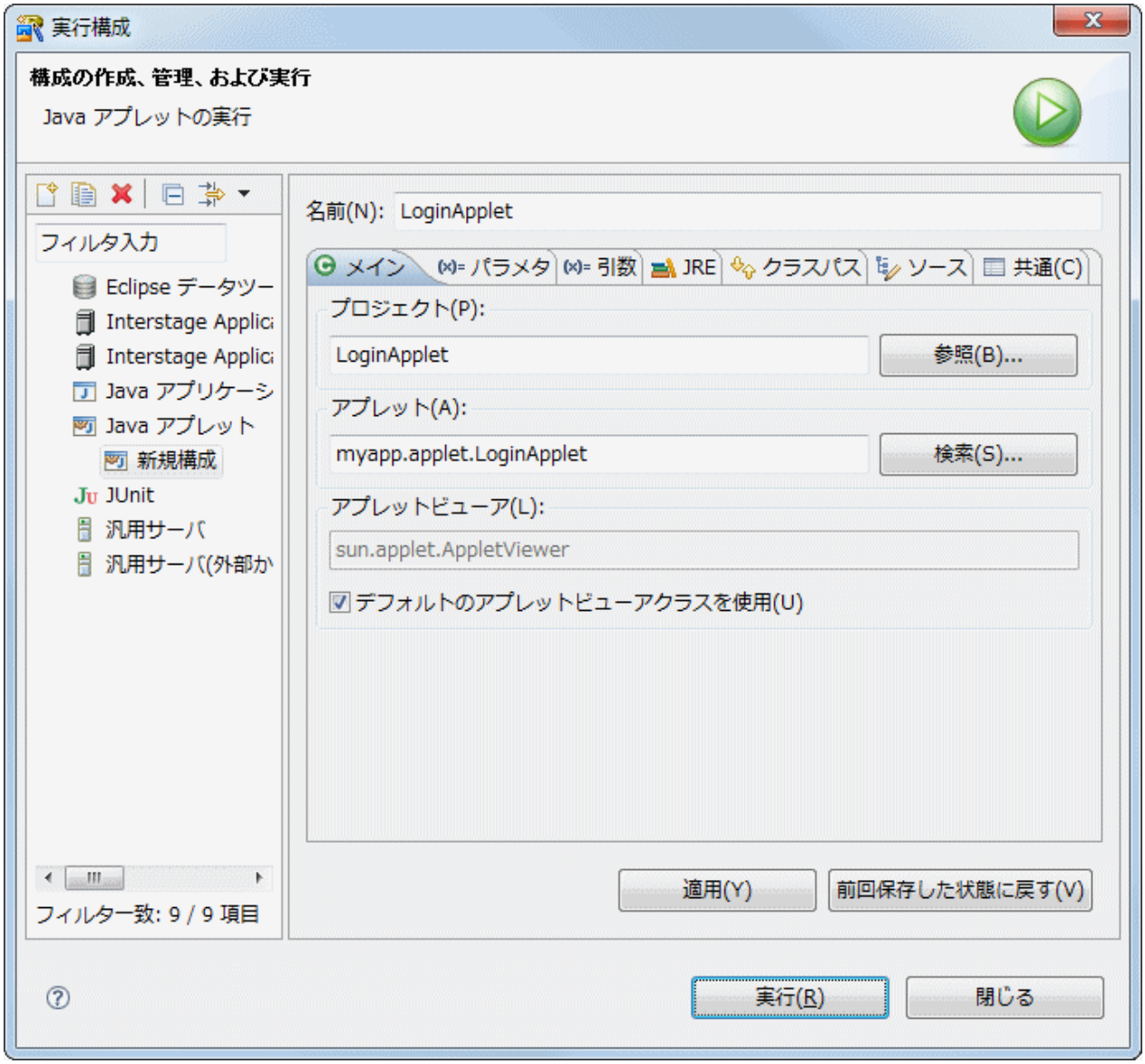
JARエクスポートでは以下のメッセージが出力されますが、問題ありません。



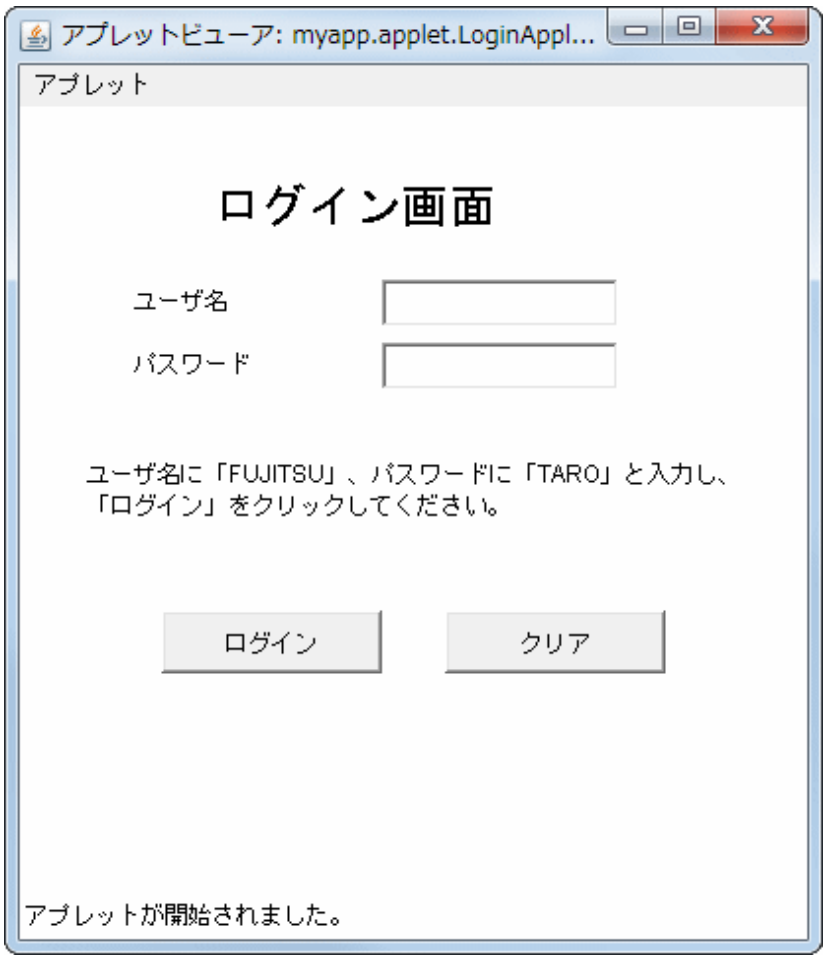
実行

1. 実行するファイル(クラス)を選択します。ワークベンチの[パッケージエクスプローラ]ビューにある[LoginApplet] > [src] > [myapp.applet] > [LoginApplet.java]をクリックします。
2. "F.3.1 アプレットの開発手順"の"実行"と同様に起動構成を作成し、実行します。以下のように設定します。

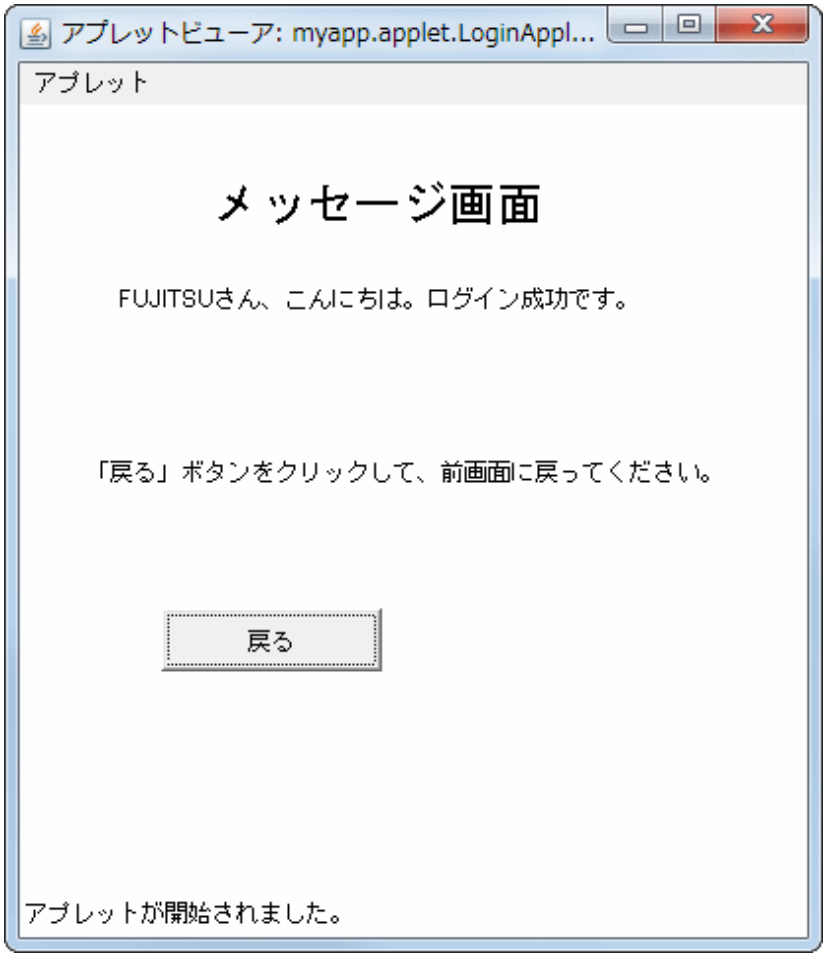
設定項目	設定内容
プロジェクト	LoginApplet
アプレット	myapp.applet.LoginApplet
パラメタ 幅	400
パラメタ 高さ	400



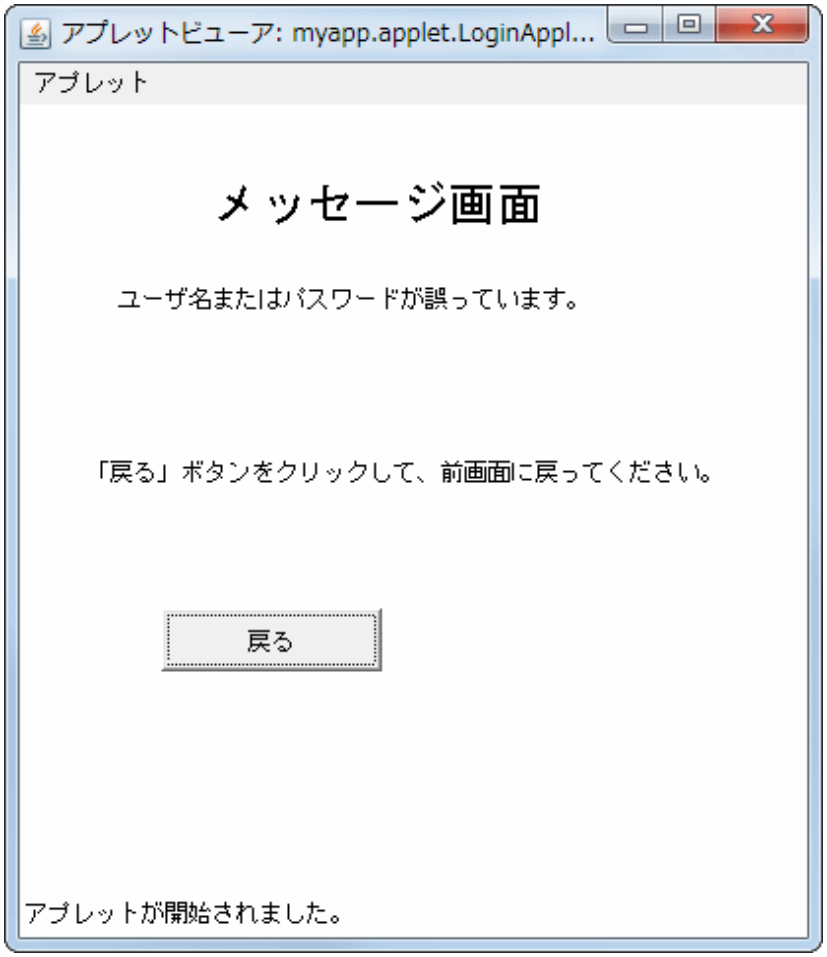
3. [実行]をクリックするとアプレットビューアが起動し、アプレットが実行されます。



4. ユーザ名に「FUJITSU」、パスワードに「TARO」を入力し、[ログイン]をクリックします。
   メッセージ画面が表示されます。

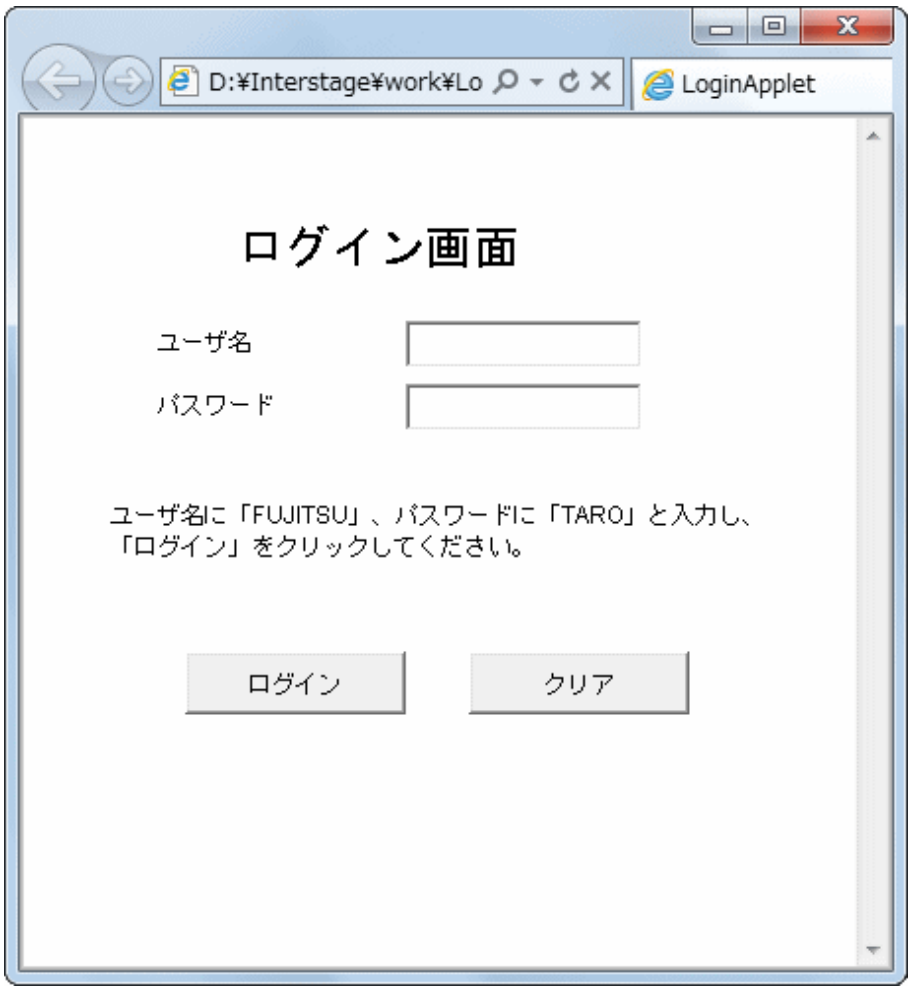


[戻る]をクリックするとことにより、ログイン画面に戻ることができます。
また、ユーザ名に「FUJITSU」、パスワードに「TARO」以外を入力した場合には、以下のメッセージが表示されます。



5. タイトルバーのクローズボタン([×]ボタン)をクリックして、アプレットビューアを終了します。

6. 実際に、Webブラウザを使って動作確認をしてみます。
   プロジェクトフォルダに格納されている「LoginApplet-JBKPlugin.html」ファイルをInternet Explorerで開き、同様に動作を確認してみてください。



F.4 JavaBeans

JBKの整数フィールドBeanとスピンボタンBeanを組み合わせた部品(JavaBeans)の作成を通して、JavaBeansの開発手順を説明します。

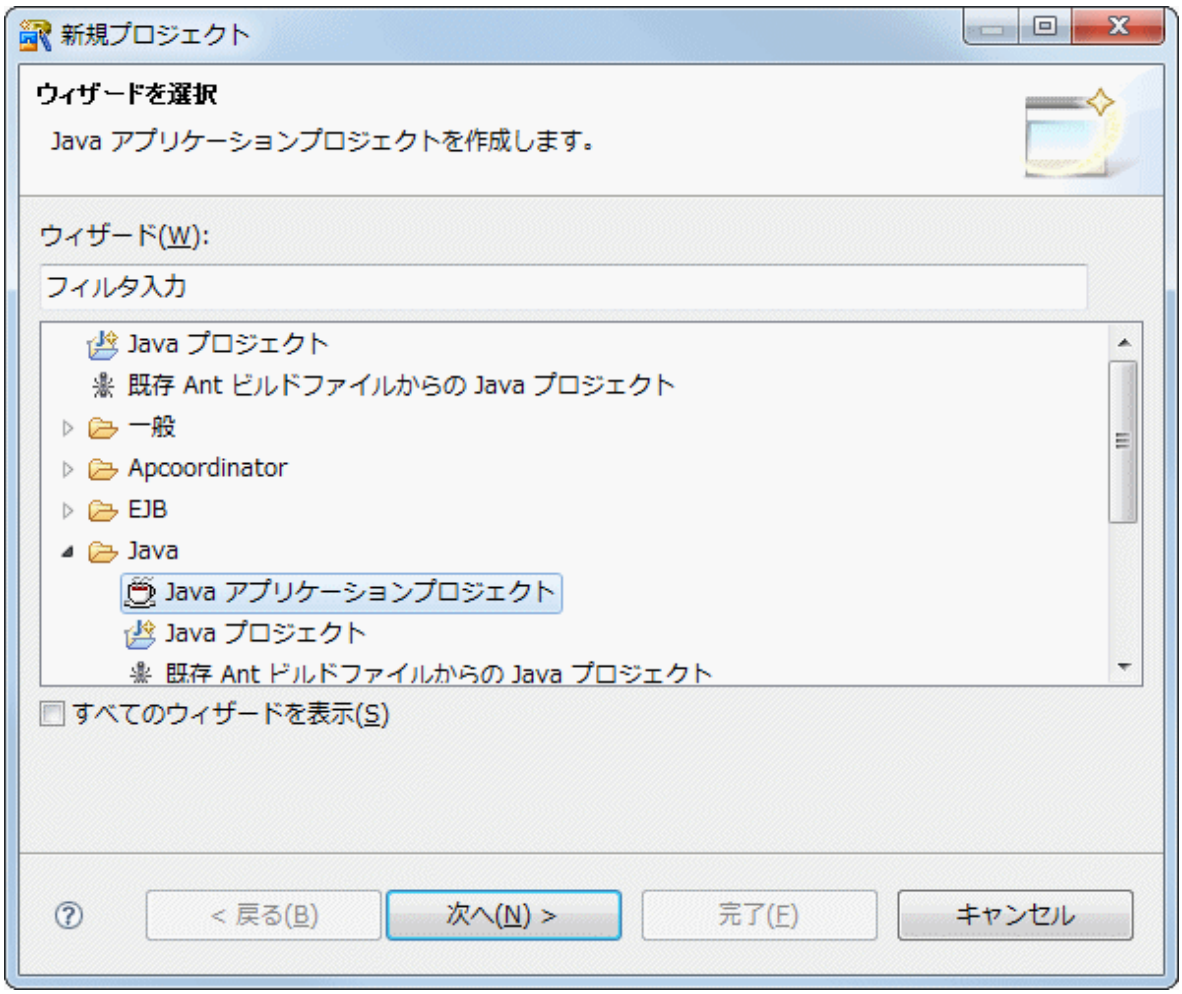
F.4.1 JavaBeansの開発

Interstage Studioでは、Javaアプリケーションのひとつであ るJavaBeansを開発することができます。
JavaBeansとは、Javaのクラスを部品化する規約であり、その部品の一般名称でもあります。部品化した個々の部品をBeanと呼びます。
ここでは、JBKの整数フィールドとスピンボタンを組み合わせた、新しいBeanを開発します。

JavaBeansのプロジェクトの作成

1. ワークベンチを起動します。
2. メニューバーから[ファイル] > [新規] > [プロジェクト]を選択します。

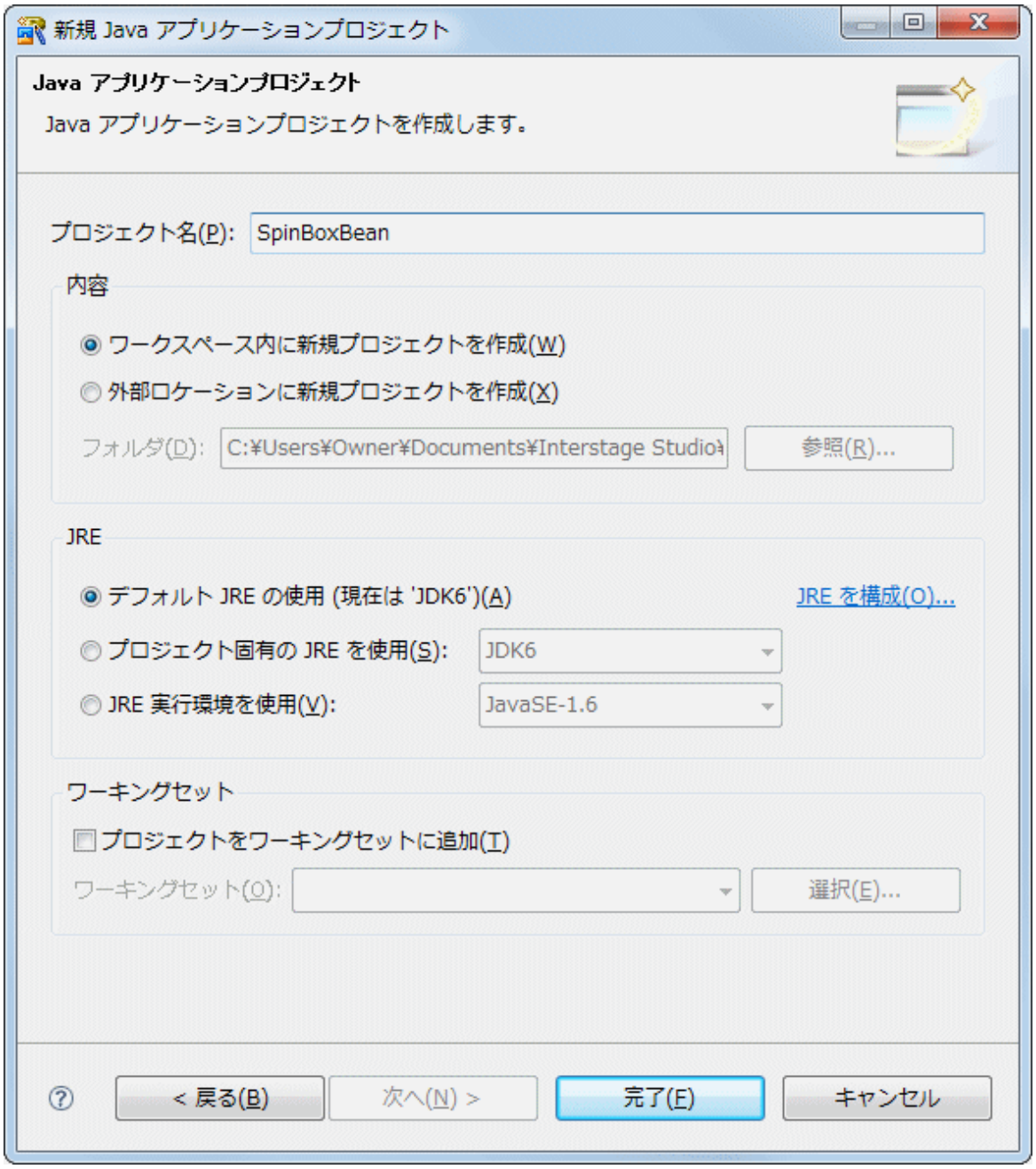
3. [新規プロジェクト]ウィザードが表示されます。ツリーから[Javaアプリケーションプロジェクト]を選択します。



[次へ]をクリックします。

4. [Javaアプリケーションプロジェクト]ページが表示されます。
以下のようにプロジェクトの情報を入力します。

設定項目	設定内容
プロジェクト	SpinBoxBean



[完了]をクリックします。

ポイント

Beanの分類について

Interstage StudioではBeanを以下の2つに分類し、それぞれの定義機能を提供しています。

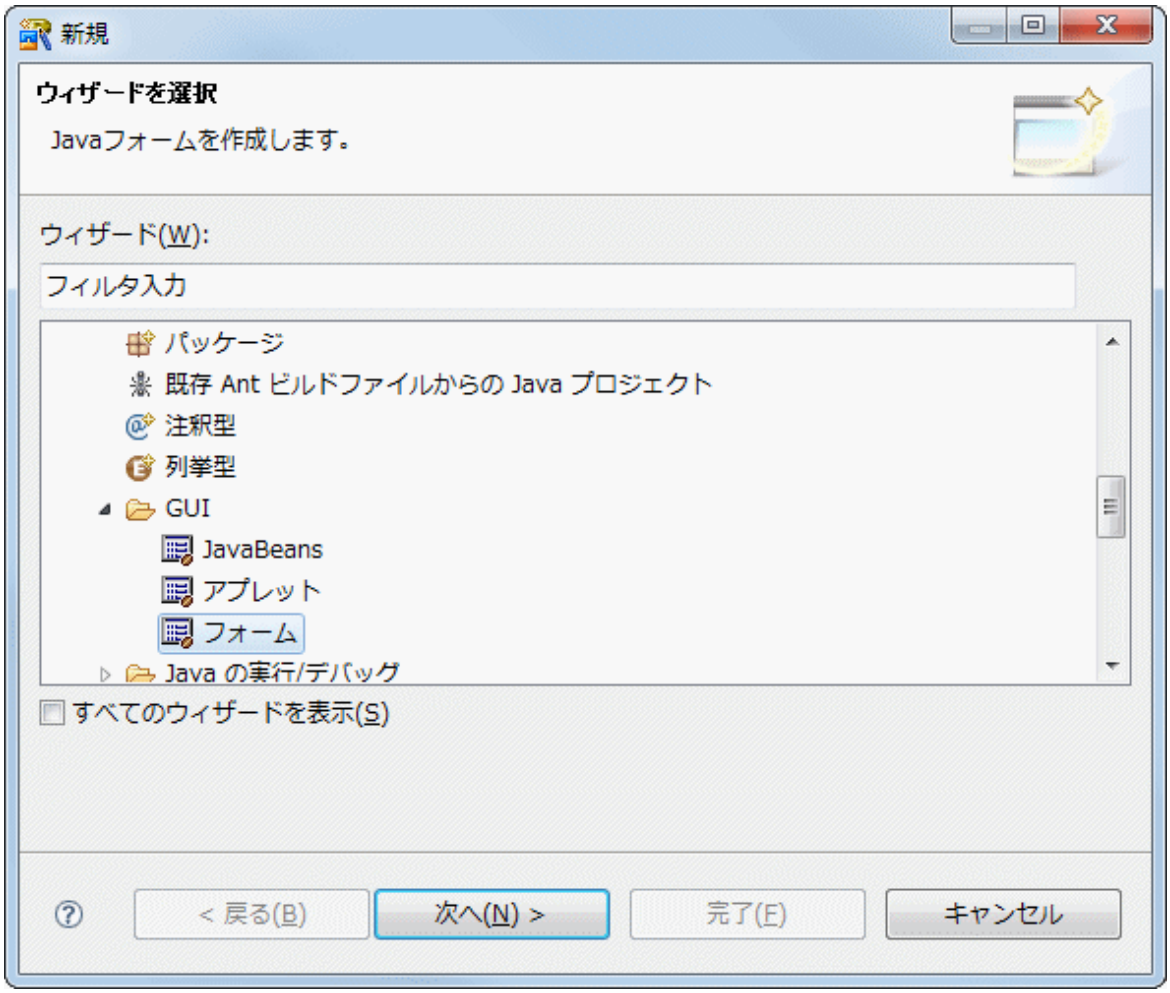
- 単体Bean
  テキストフィールドやボタンのような単機能のBeanです。テキストフィールド(JTextField)など既存の単体Beanを継承し、付加機能を付け、新しい単体Beanとして作成します。
  基底クラスにビューをもたないクラスを指定した場合は、不可視Beanとなります。java.lang.Objectなどを継承します。不可視Beanはビューをもたないため、一般のソースと同様にエディタ上で編集します。

- 複合Bean
  複数のBeanで構成するBeanです。Javaフォーム定義で、フォームのレイアウトを作成する要領でパネル(JPanelまたはPanel)

に対して、Beanを貼り付け作成します。コンテナタイプにより、軽量コンポーネントの場合はJPanel、重量コンポーネントの場合はPanelを使います。

Javaフォームの作成

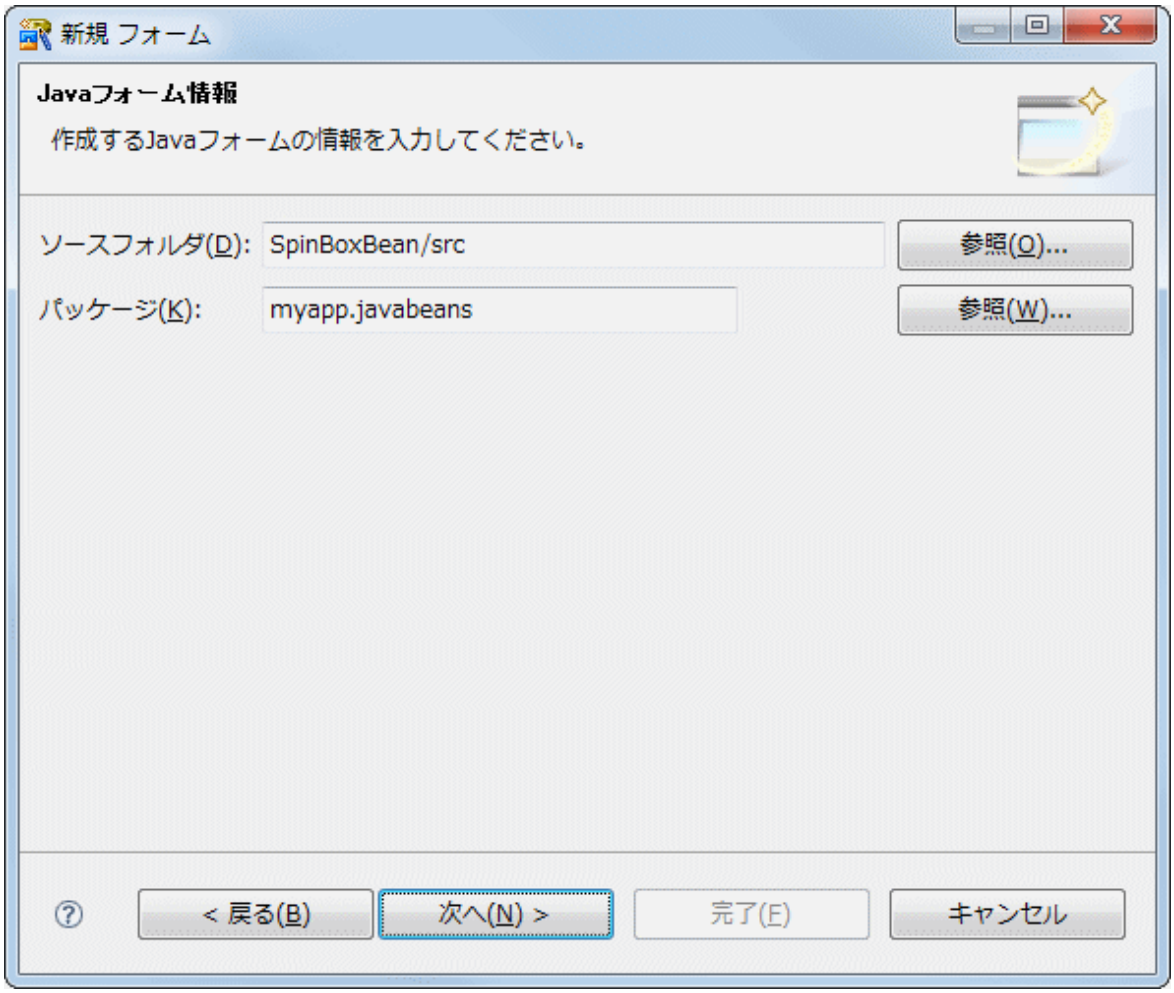
1. 引き続き複合Beanを作成します。ワークベンチのメニューバーから、[ファイル] > [新規] > [その他]を選択します。表示される[新規]ウィザードのツリーから、[Java] > [GUI] > [フォーム]を選択します。
   単体Beanや不可視Beanを作成する場合は[JavaBeans]を選択しますが、今回は複合Beanを作成するので[フォーム]を選択します。



   [次へ]をクリックします。

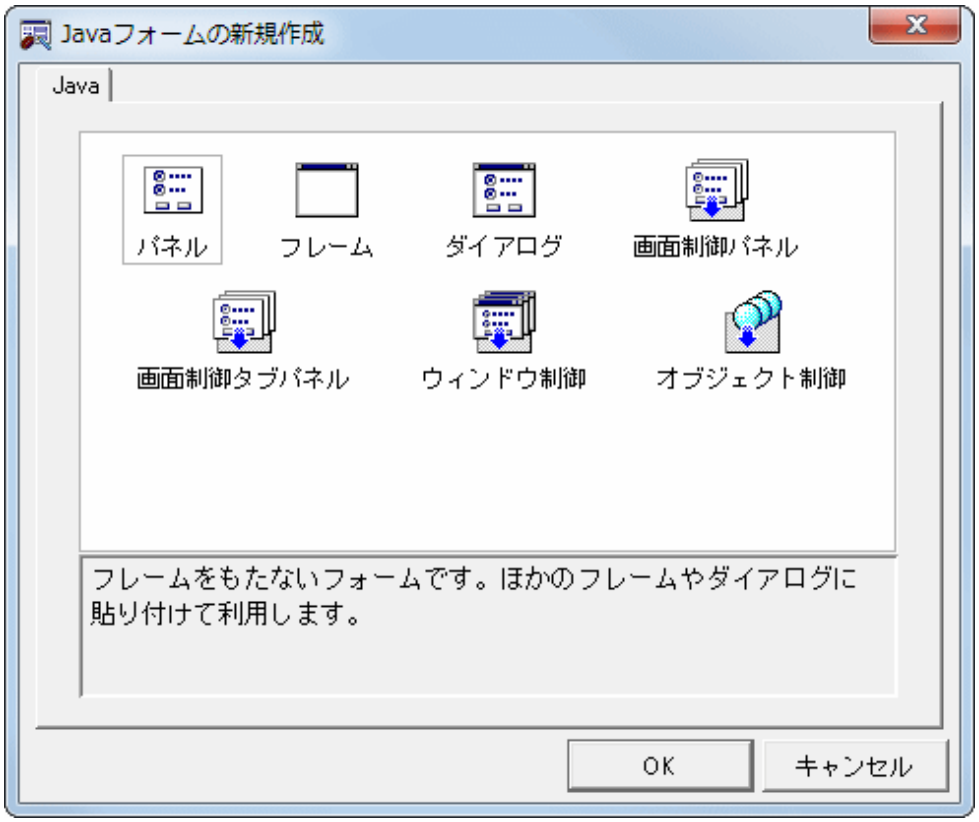
2. [Javaフォーム情報]ページが表示されます。
   このページでは、以下の情報を入力します。

設定項目	設定内容
パッケージ	myapp.javabeans


新規 フォーム
Javaフォーム情報
作成するJavaフォームの情報を入力してください。
ソースフォルダ(D):
SpinBoxBean/src
参照(O)...
パッケージ(K):
myapp.javabeans
参照(W)...
< 戻る(B)
次へ(N) >
完了(F)
キャンセル

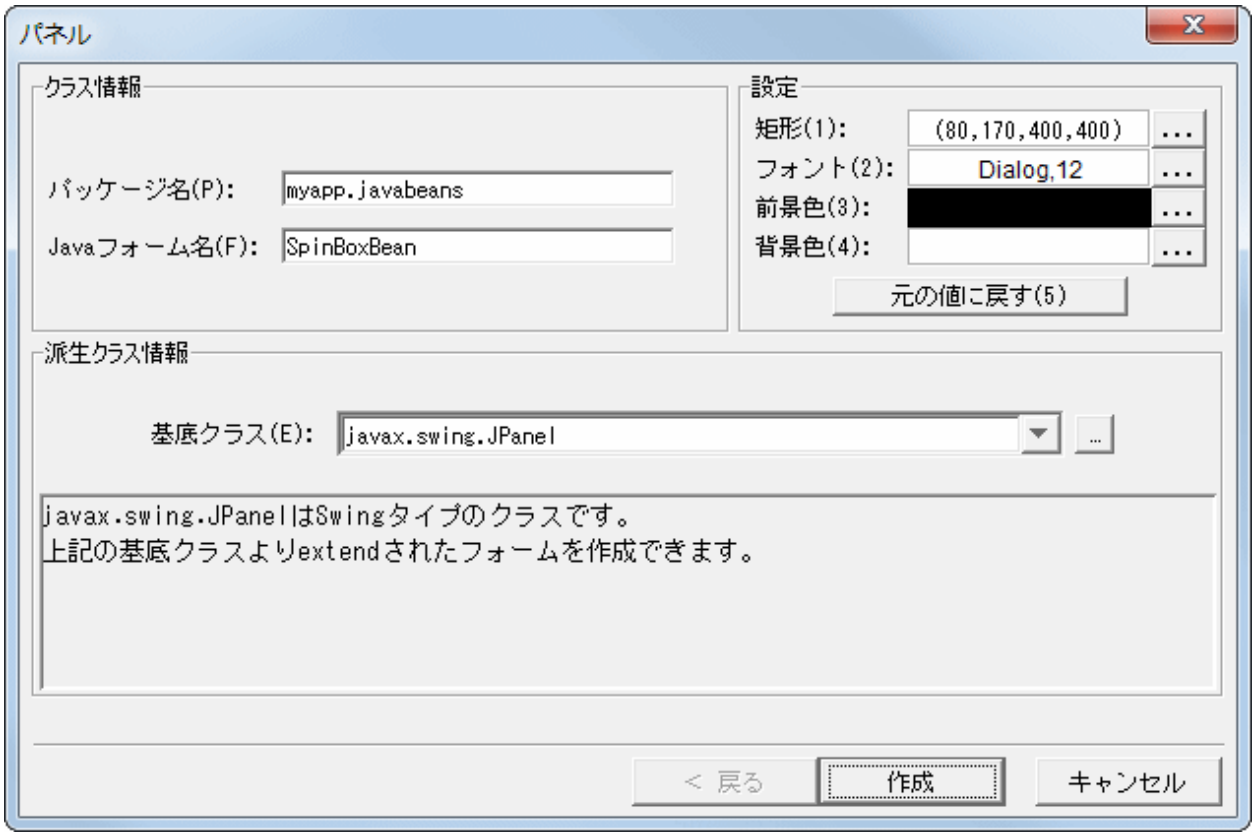
[次へ]をクリックします。

3. [Javaフォームの新規作成]ウィザードが表示されます。
作成するフォームのタイプを選択します。[Java]タブにある[パネル]を選択し、[OK]をクリックします。



4. パネル形式のフォームを新規作成します。
以下の情報を入力します。

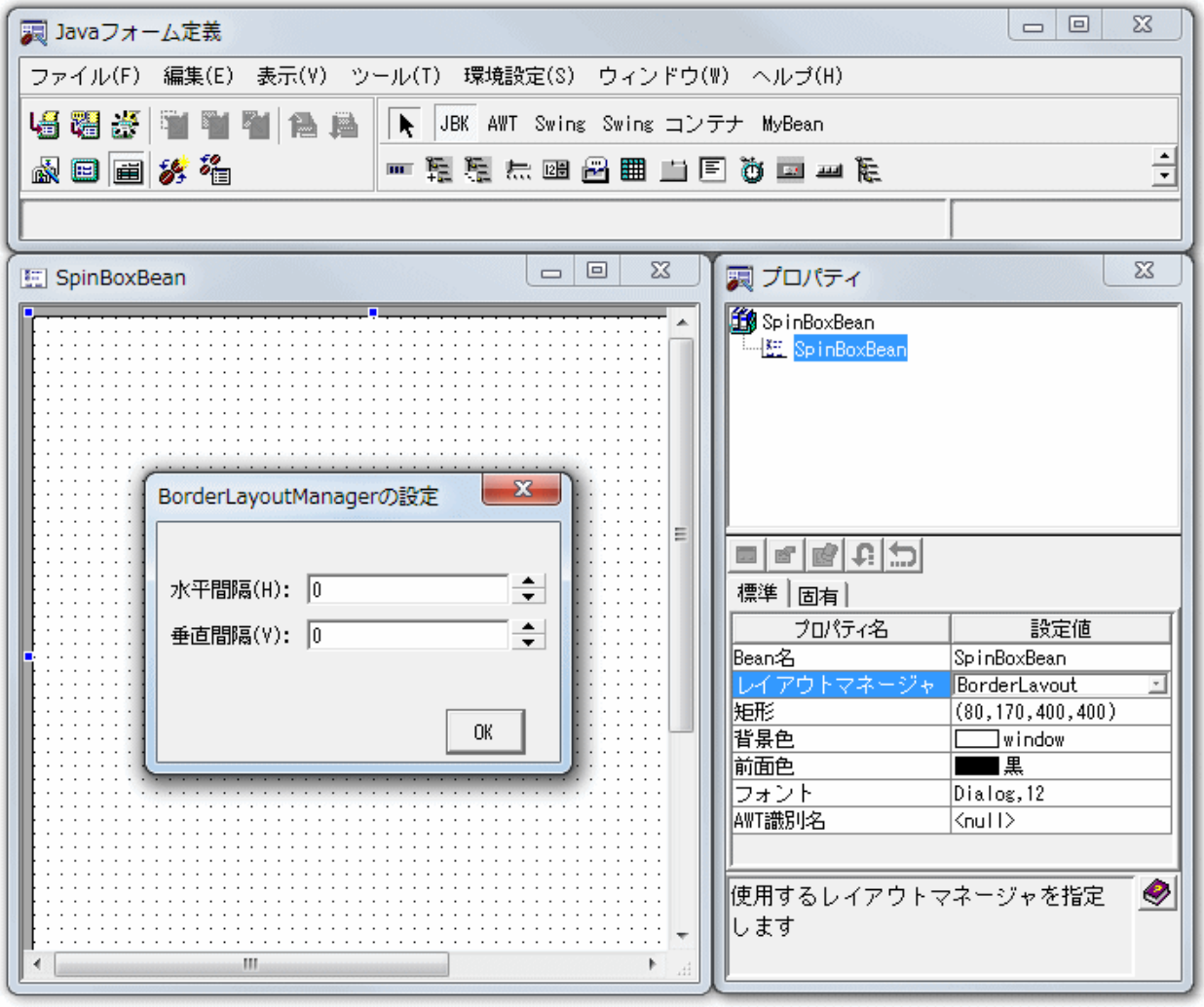
設定項目	設定内容
Javaフォーム名	SpinBoxBean
基底クラス	javax.swing.JPanel



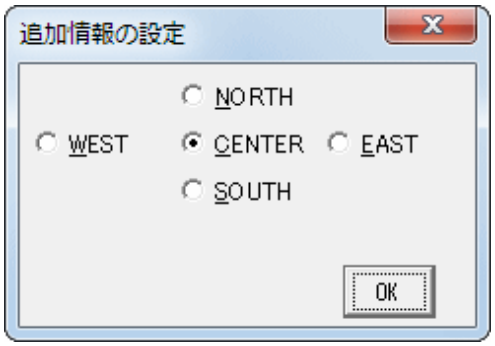
[作成]をクリックします。

5. Javaフォーム定義が起動されます。Javaフォーム定義を使用して、Beanの配置、プロパティ(属性)の設定、処理手続きの記述をし、フォームを完成させます。

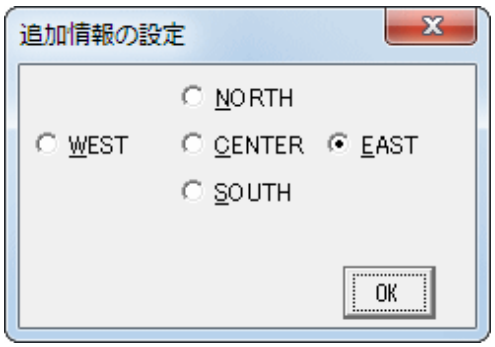
6. フレームのプロパティを変更します。
レイアウトマネージャプロパティに「BorderLayout」を設定します。
水平間隔、垂直間隔はともに0を指定し、[OK]をクリックします。



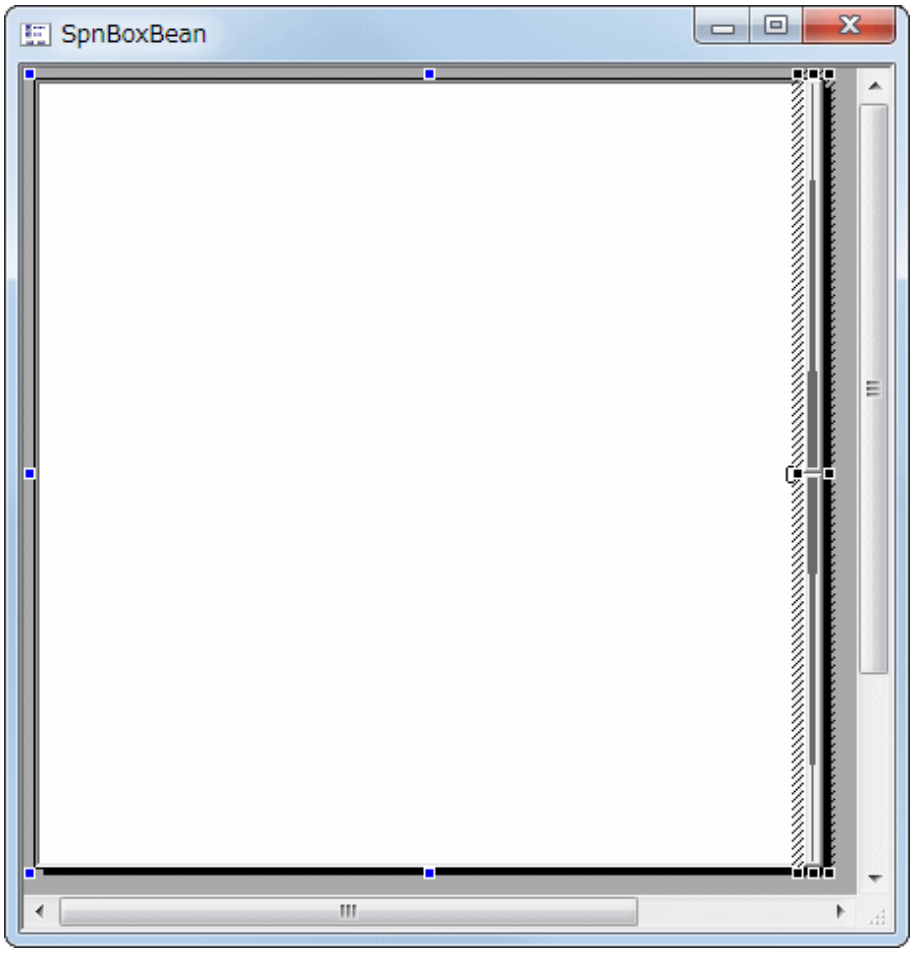
7. 整数フィールドBeanを配置します。
オブジェクトパレットの[JBK]が選択されていない場合は、[JBK]をクリックします。
オブジェクトパレットの[整数フィールド]をクリックして選択します。マウスを使ってJavaフォーム上でドラッグして配置します。
「BorderLayout」は、上下左右にBeanを配置します。上下左右および中央は、「NORTH」、「SOUTH」、「WEST」、「EAST」、「CENTER」と呼ばれます。
[CENTER]を選択し、[OK]をクリックします。



8. スピンボタンBeanを配置します。
   オブジェクトパレットの[スピンボタン]をクリックして選択します。マウスを使ってJavaフォーム上でドラッグして配置します。
   [EAST]を選択し、[OK]をクリックします。



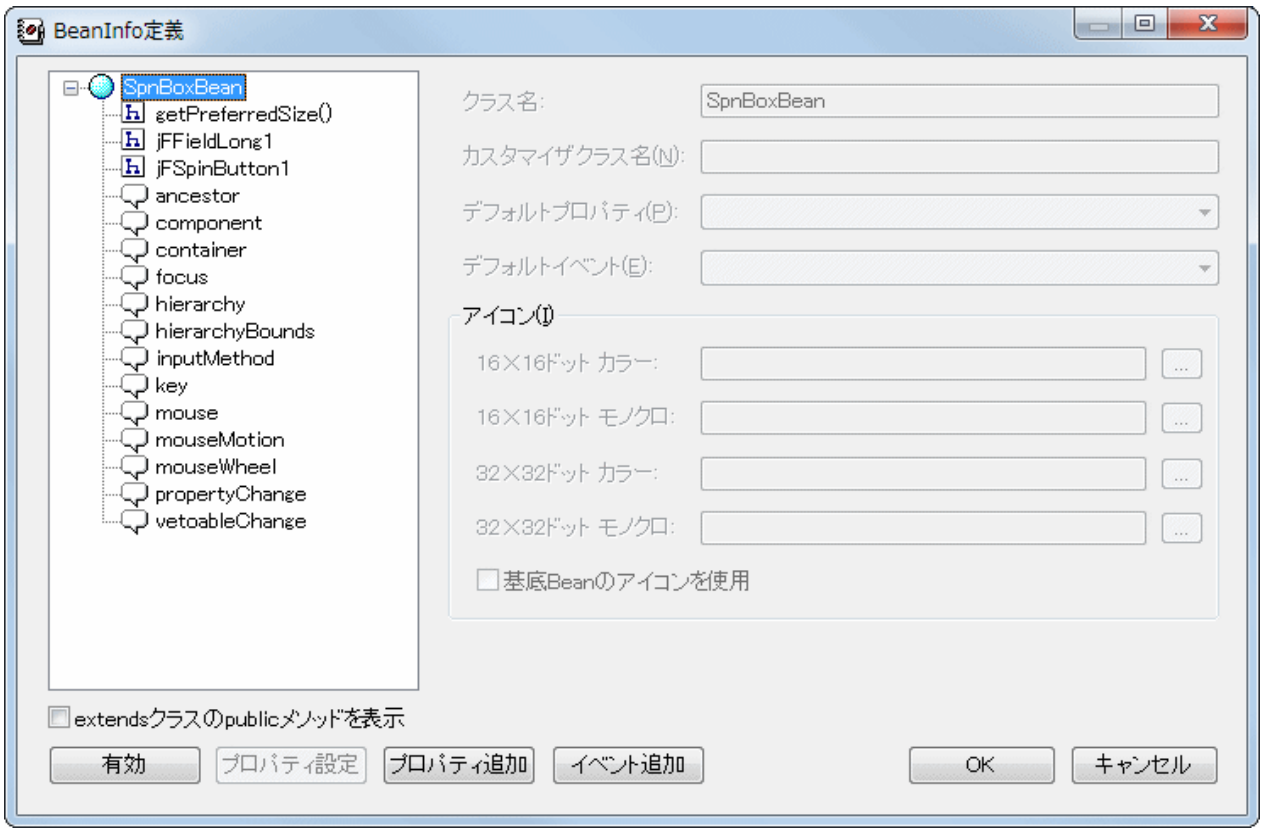
   2つのBeanは以下のように貼り付けられます。



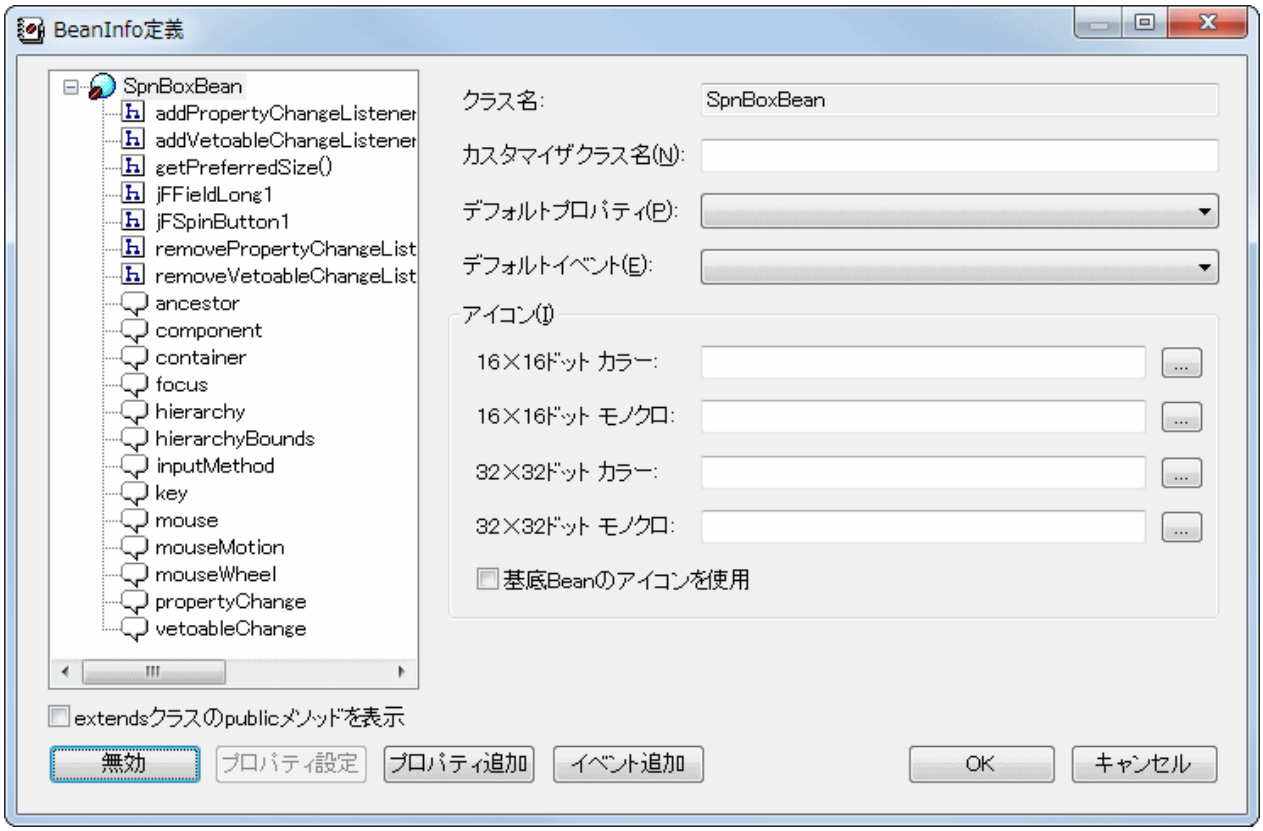
BeanInfoの定義

1. BeanInfoは、Beanをコンポーネントとして利用するプログラムに対する外部インタフェース情報です。BeanInfo定義を使用して、プロパティ、メソッド、イベントなどの情報を定義します。
   BeanInfo定義は、ワークベンチから起動します。Javaフォーム定義のメニューバーから[表示] > [Javaエディタ]を選択し、ワークベンチをアクティブにします。

2. ワークベンチのメニューバーから[編集] > [BeanInfoの定義] > [BeanInfo定義]を選択して、BeanInfo定義を呼び出します。



「SpinBoxBean」が選択されている状態で、[有効]をクリックします。



ポイント

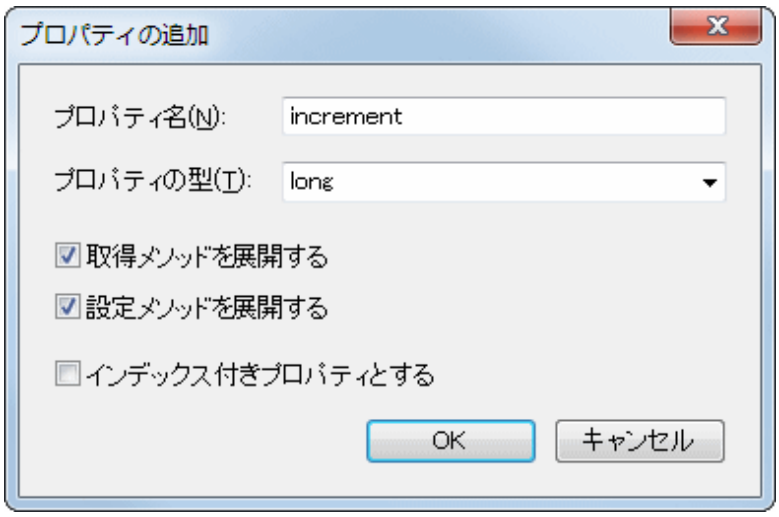
アイコンについて

BeanInfo定義では、作成するBeanのアイコンを指定することができます。
アイコンのイメージファイルは、GIF形式またはJPEG形式で任意のペイントツールであらかじめ作成し、次の方法でプロジェクトに追加してください。ワークベンチのメニューバーから[ファイル] > [インポート]をクリックします。[インポート]ダイアログボックスにある[ファイルシステム]を利用して、イメージファイルをプロジェクトにインポートします。
Interstage Studioでは、Beanのアイコンをオブジェクトパレットやプロパティウィンドウ内のコンポーネントツリーに表示する際に、「16×16ドット カラー」に指定されたイメージファイルを使用します。
アイコンのイメージファイルの指定は、省略することができます。省略した場合は、Interstage Studioが用意しているデフォルトのアイコンとなります。

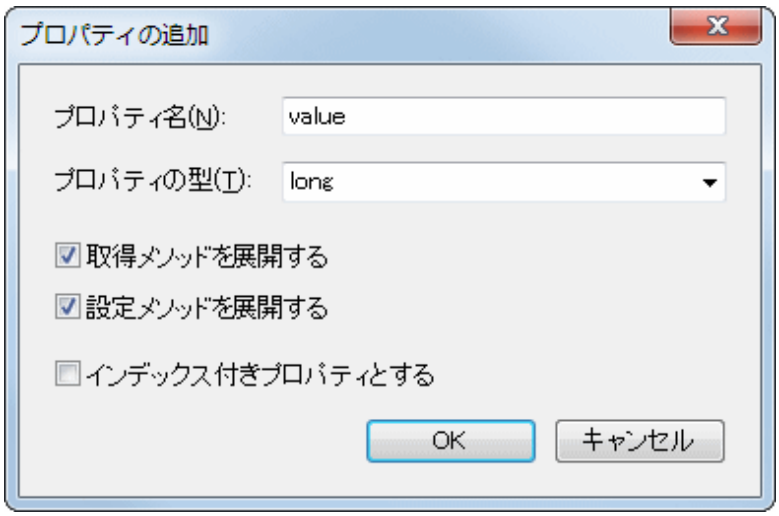
3. 次にプロパティを定義します。
   以下の2つのプロパティを公開することにします。

プロパティ名	型	意味
increment	long	スピンボタンの増分値
value	long	整数フィールドの値

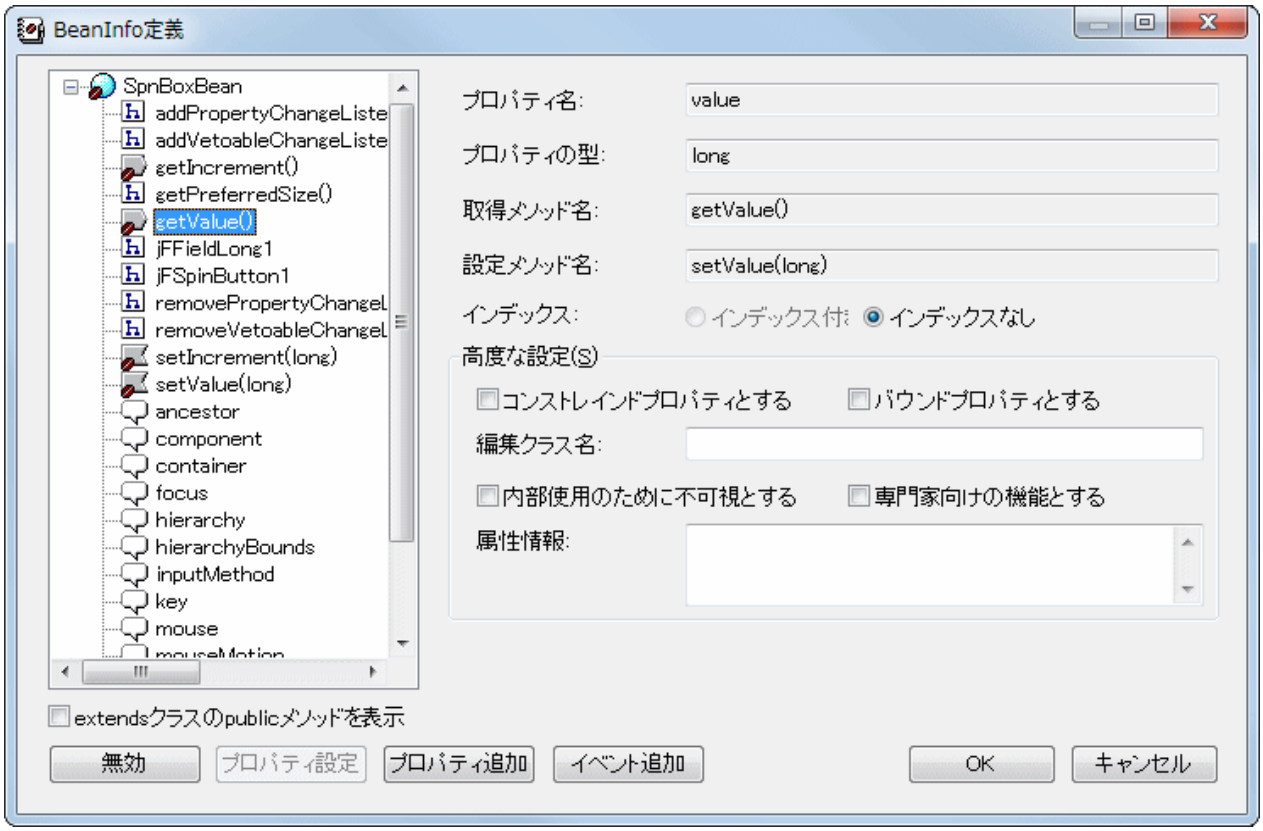
BeanInfo定義画面で、[プロパティ追加]をクリックします。
ダイアログボックスのように指定したら、[OK]をクリックします。



同様に、valueプロパティを追加します。
BeanInfo定義画面で、[プロパティ追加]をクリックします。
ダイアログボックスのように指定したら、[OK]をクリックします。



以下のようにプロパティが追加されます。



4. [OK]をクリックして、BeanInfo定義画面を閉じます。

イベント処理の記述

1. エディタエリアにある[SpinBoxBean.java]タブをクリックして、Javaエディタをアクティブにします。
2. JBKのクラスを使用するので、importキーワードを追加します。
   赤字の部分を追加します。

```
import com.fujitsu.jbk.gui.JFSpinButton;
```

3. 増分値を保持するフィールドをクラスに追加します。
SpinBoxBeanクラスにあるコンストラクタの上に、incrementフィールドを追加します。赤字の部分を追加します。

```
//@@Form Design Information end

long increment = 1;
/**
 * フォームを構築します。
 */
public SpinBoxBean() {
}
```

ポイント

Javaフォーム定義が管理する変更禁止ソースについて

Javaフォームは、画面(Bean)の情報やイベント処理を呼び出すソースを含んでいます。このソースは、Javaフォーム定義が管理しているため変更禁止です。
青字のコメントで囲われた部分が、Javaフォーム定義が管理する変更禁止ソースになります。

```
public class SpinBoxBean extends javax.swing.JPanel {
    //@@Form Design Information start

      ～Javaフォーム定義が管理する変更禁止ソース～

    //@@Form Design Information end
```

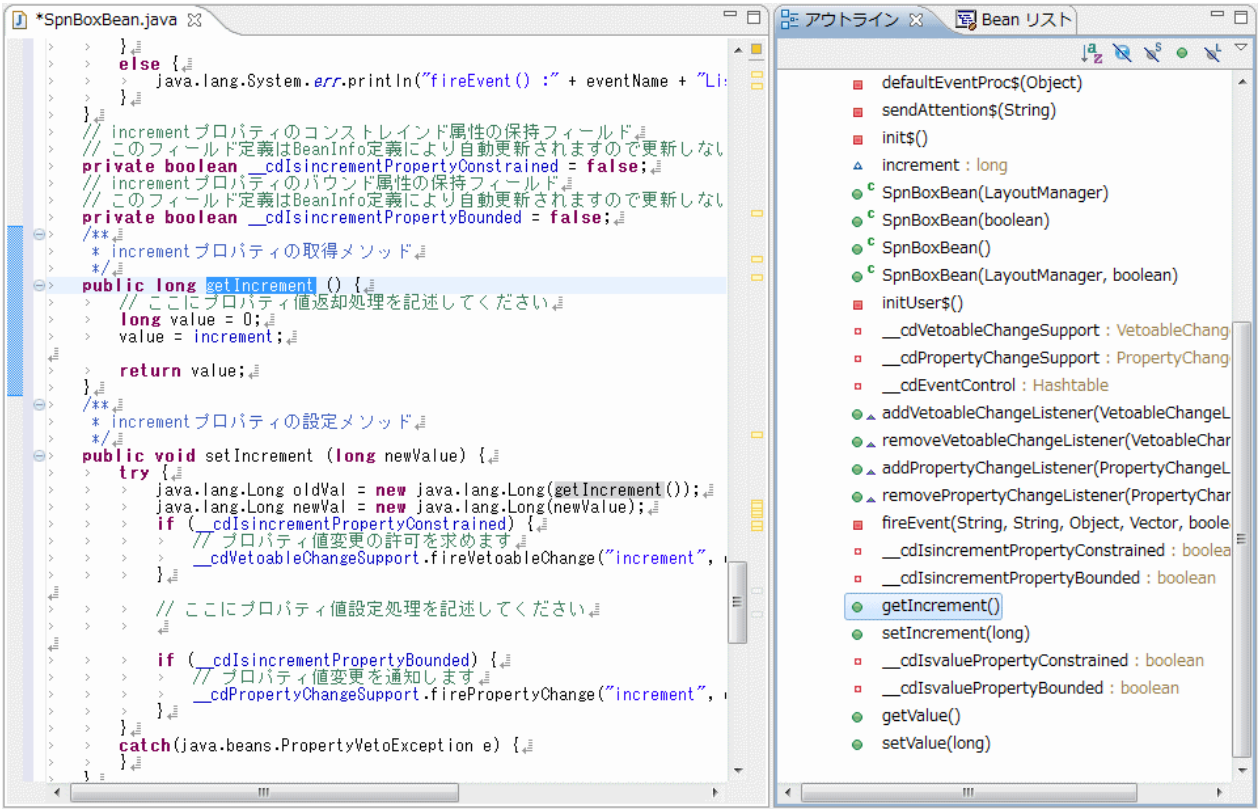
4. getIncrementメソッドの処理を記述します。
incrementプロパティの取得メソッドであるgetIncrementに処理を追加します。赤字の部分を追加します。

```
public long getIncrement() {
    // ここにプロパティ値返却処理を記述してください
    long value = 0;
    value = increment;
    return value;
}
```

ポイント

Javaエディタのスクロールについて

Javaエディタの表示を編集したいメソッドやフィールドまでスクロールしたい場合、[アウトライン]ビューを利用すると便利です。
[アウトライン]ビューにある「getIncrement()」をクリックすると、Javaエディタの表示がgetIncrementメソッドまでスクロールします。



5. setIncrementメソッドの処理を記述します。
incrementプロパティの設定メソッドであるsetIncrementに処理を追加します。赤字の部分を追加します。

```
public void setIncrement(long newValue) {
    try {
        java.lang.Long oldVal = new java.lang.Long(getIncrement());
        java.lang.Long newVal = new java.lang.Long(newValue);
        if( __cdIsincrementPropertyConstrained ) {
            // プロパティ値変更の許可を求めます
            __cdVetoableChangeSupport.fireVetoableChange("increment",oldVal,newVal);
        }

        // ここにプロパティ値設定処理を記述してください
        increment = newValue;

        if( __cdIsincrementPropertyBounded ) {
            // プロパティ値変更を通知します
            __cdPropertyChangeSupport.firePropertyChange("increment",oldVal,newVal);
        }
    }
    catch(java.beans.PropertyVetoException e) {
    }
}
```

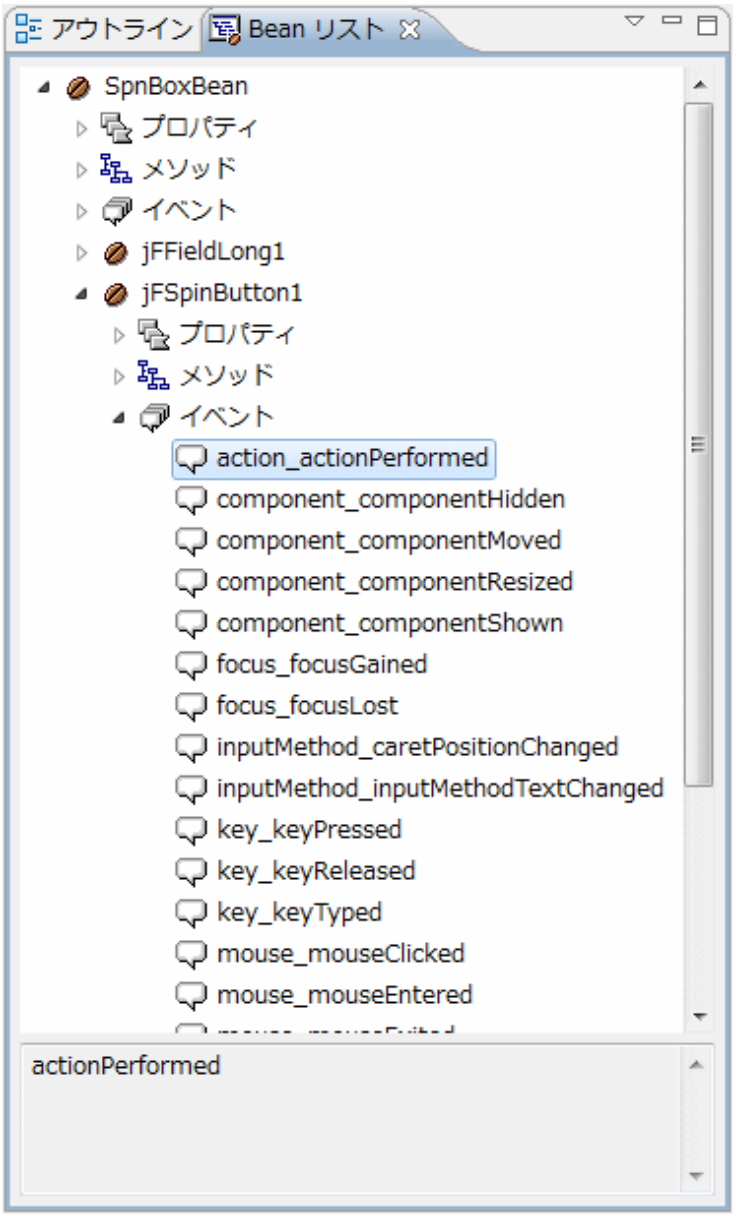
6. getValueメソッドの処理を記述します。
valueプロパティの取得メソッドであるgetValueに処理を追加します。赤字の部分を追加します。

```
public long getValue() {
    // ここにプロパティ値返却処理を記述してください
    long value = 0;
    value = jFFieldLong1.getValue();
    return value;
}
```

7. setValueメソッドの処理を記述します。
valueプロパティの設定メソッドであるsetValueに処理を追加します。赤字の部分を追加します。

```
public void setValue(long newValue) {
    try {
        java.lang.Long oldVal = new java.lang.Long(getValue());
        java.lang.Long newVal = new java.lang.Long(newValue);
        if( __cdIsvaluePropertyConstrained ) {
            // プロパティ値変更の許可を求めます
            __cdVetoableChangeSupport.fireVetoableChange("value",oldVal,newVal);
        }

        // ここにプロパティ値設定処理を記述してください
        jFFieldLong1.setValue(newValue);

        if( __cdIsvaluePropertyBounded ) {
            // プロパティ値変更を通知します
            __cdPropertyChangeSupport.firePropertyChange("value",oldVal,newVal);
        }
    }
    catch(java.beans.PropertyVetoException e) {
    }
}
```

8. イベントが発生したときの処理を記述するため、[Beanリスト]ビューを表示します。
メニューバーから[ウィンドウ] > [ビューの表示] > [その他]を選択します。[ビューの表示]ダイアログボックスが表示されます。[Java] > [Beanリスト]を選択し、[OK]をクリックします。



9. スピンボタンをクリックしたときの処理を記述します。
[Beanリスト]ビューにある[SpinBoxBean] > [jFSpinButton1] > [イベント] > [action_actionPerformed]をダブルクリックします。
イベント処理の作成確認ダイアログが表示されるので、[はい]をクリックします。
「jFSpinButton1_action_actionPerformed」の処理手続きが表示されるので、**赤字**の部分を追加します。

```
    public void jFSpinButton1_action_actionPerformed(java.awt.event.ActionEvent e) {
        if (!defaultEventProc(e)) {
            // ここにイベント発生時の処理を記述します.
            if (e.getActionCommand() == JFSpinButton.UP){
                // 上のボタンが押された
                jFFieldLong1.setValue(jFFieldLong1.getValue() + increment);
            }
            if (e.getActionCommand() == JFSpinButton.DOWN){
                // 下のボタンが押された
                jFFieldLong1.setValue(jFFieldLong1.getValue() - increment);
            }
```

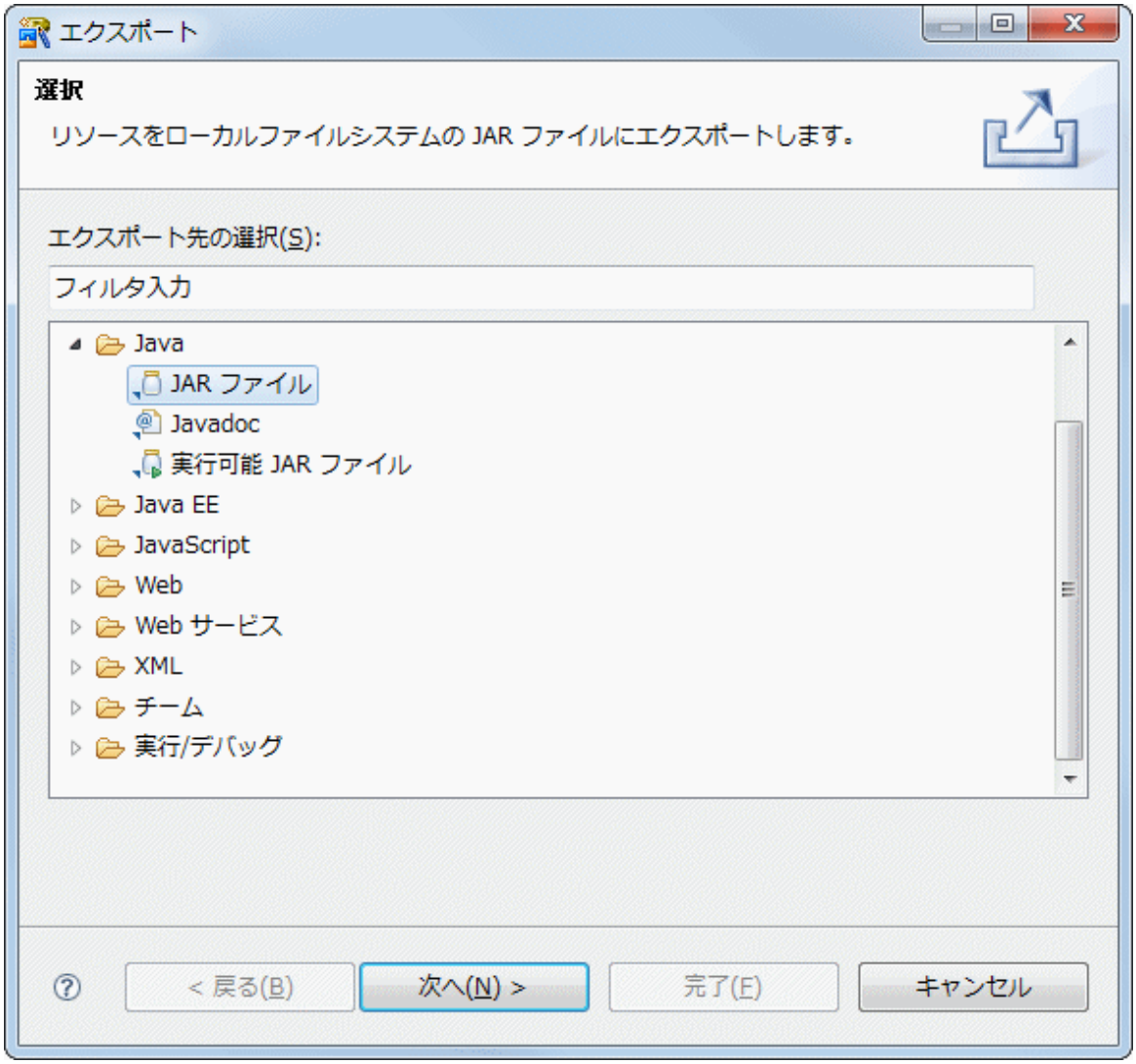
```
        }
    }
```

10. Javaフォームを保存します。
    ワークベンチのメニューバーから[ファイル] > [保存]を選択します。Javaフォーム定義のメニューから [ファイル] > [終了]を選択し、Javaフォームを終了します。

ビルド

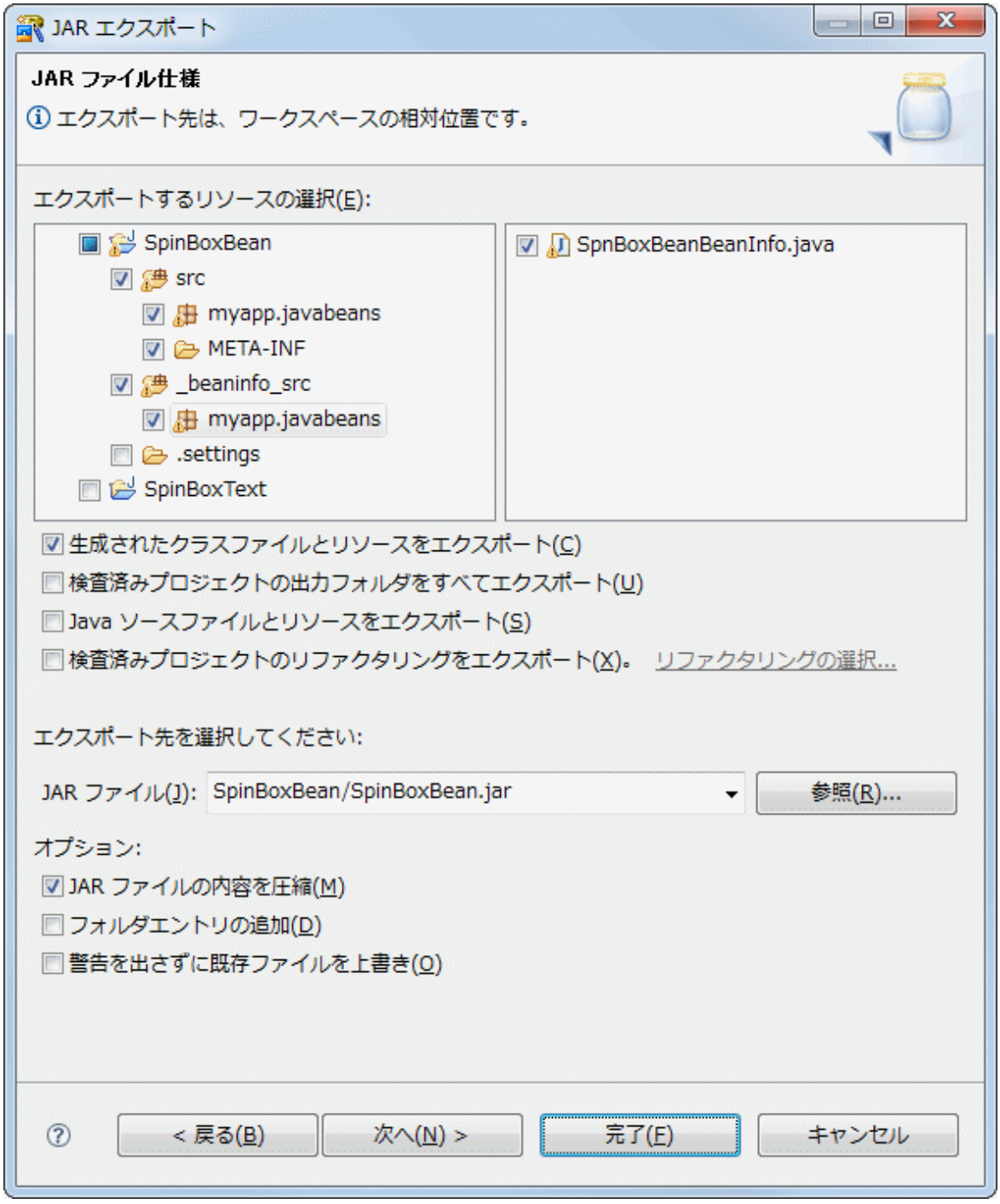
1. ビルドは、[プロジェクト]メニューの[自動的にビルド]が有効になっている場合は、Javaファイルなどのリソースファイルを保存すると自動的に実行されます。
   [自動的にビルド]が無効になっている場合は、プロジェクトを選択し右クリックメニューの[プロジェクトのビルド]を選択するか、[プロジェクト]メニューの[プロジェクトのビルド]を選択します。

2. JARファイルについては別途必要なため、これを作成します。
   JARファイルの作成は、エクスポートウィザードから行います。
   エクスポートウィザードを起動するには、[ファイル] > [エクスポート]を選択します。
   エクスポートウィザードから[Java] > [JARファイル]を選択します。

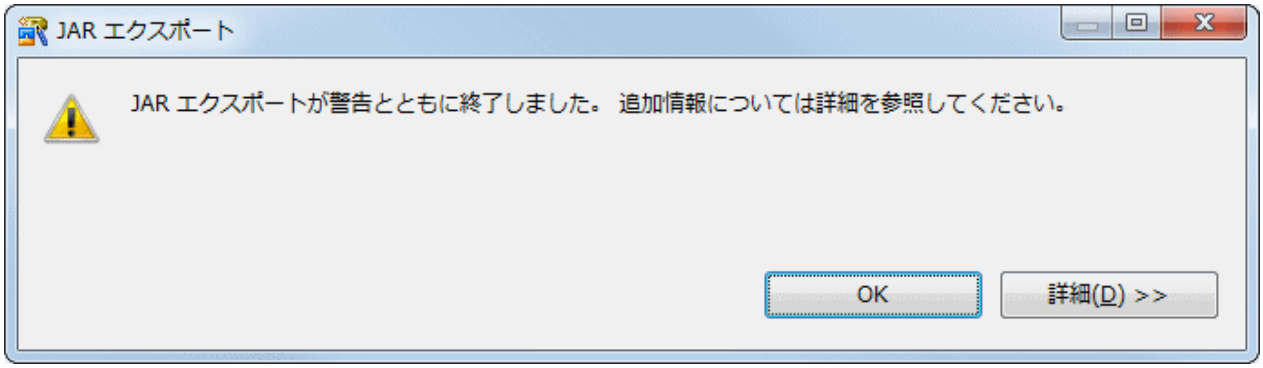


3. JARエクスポートウィザードが表示されます。
   [エクスポートするリソースの選択]で[SpinBoxBean] > [src] > [myapp.javabeans]を選択し、以下の設定項目を入力します。
   情報を設定後、[完了]をクリックします。

設定項目	設定内容
エクスポートするリソースの選択	SpinBoxBean.java SpinBoxBeanInfo.java MENIFEST.MF
生成されたクラスファイルとリソースをエクスポート	チェックする
JARファイル	SpinBoxBean/SpinBoxBean.jar
JARファイルの内容を圧縮	チェックする

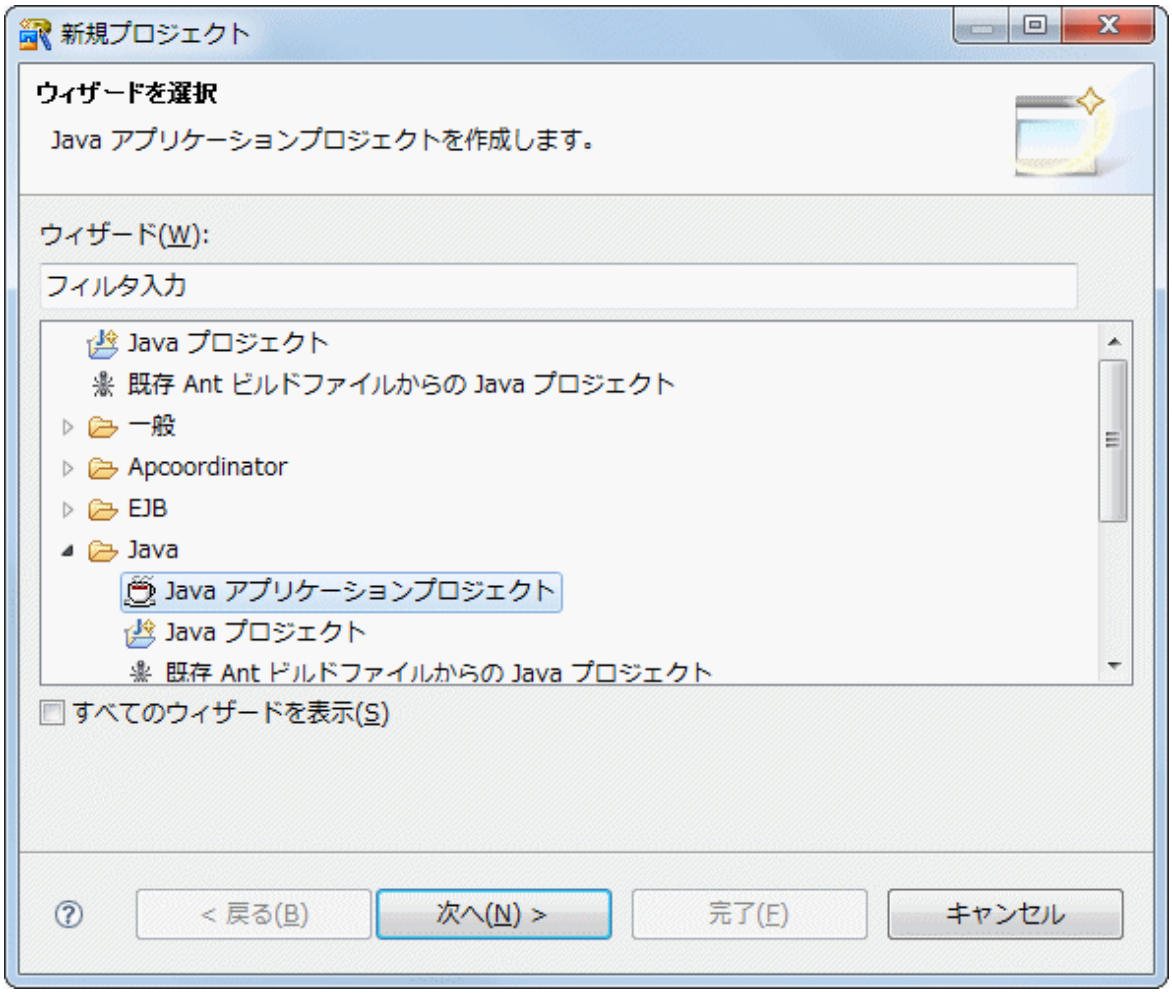


JARエクスポートでは以下のメッセージが出力されますが、問題ありません。



Javaアプリケーションプロジェクトの作成(動作確認を行うプログラムの作成)

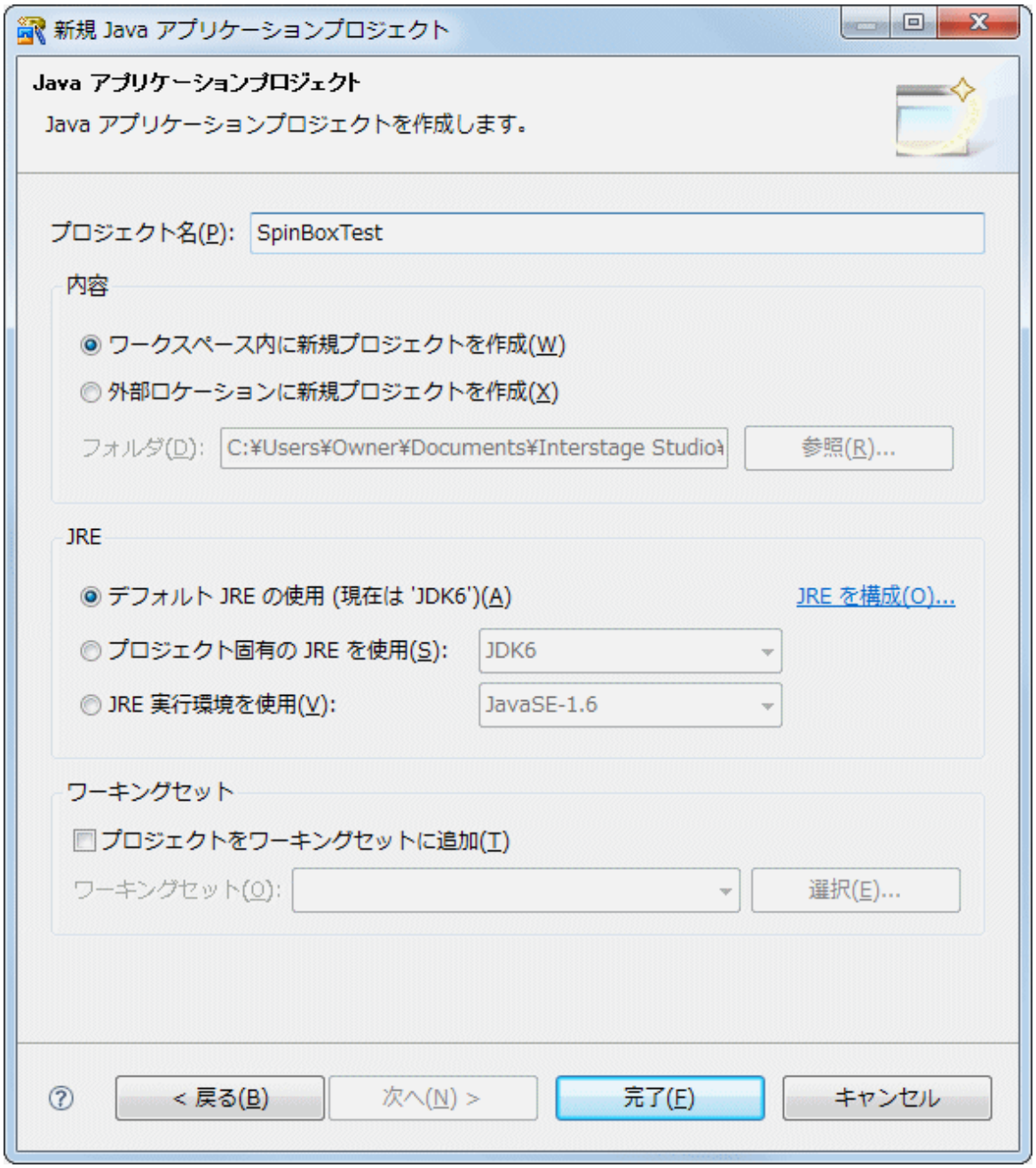
1. 作成したSpinBoxBeanの動作確認を行うために、別のプロジェクトを作成します。
   メニューバーから[ファイル] > [新規] > [プロジェクト]を選択します。
2. [新規プロジェクト]ウィザードが表示されます。ツリーから[Javaアプリケーションプロジェクト]を選択します。



[次へ]をクリックします。

3. [Javaアプリケーションプロジェクト]ページが表示されます。以下のようにプロジェクトの情報を入力します。

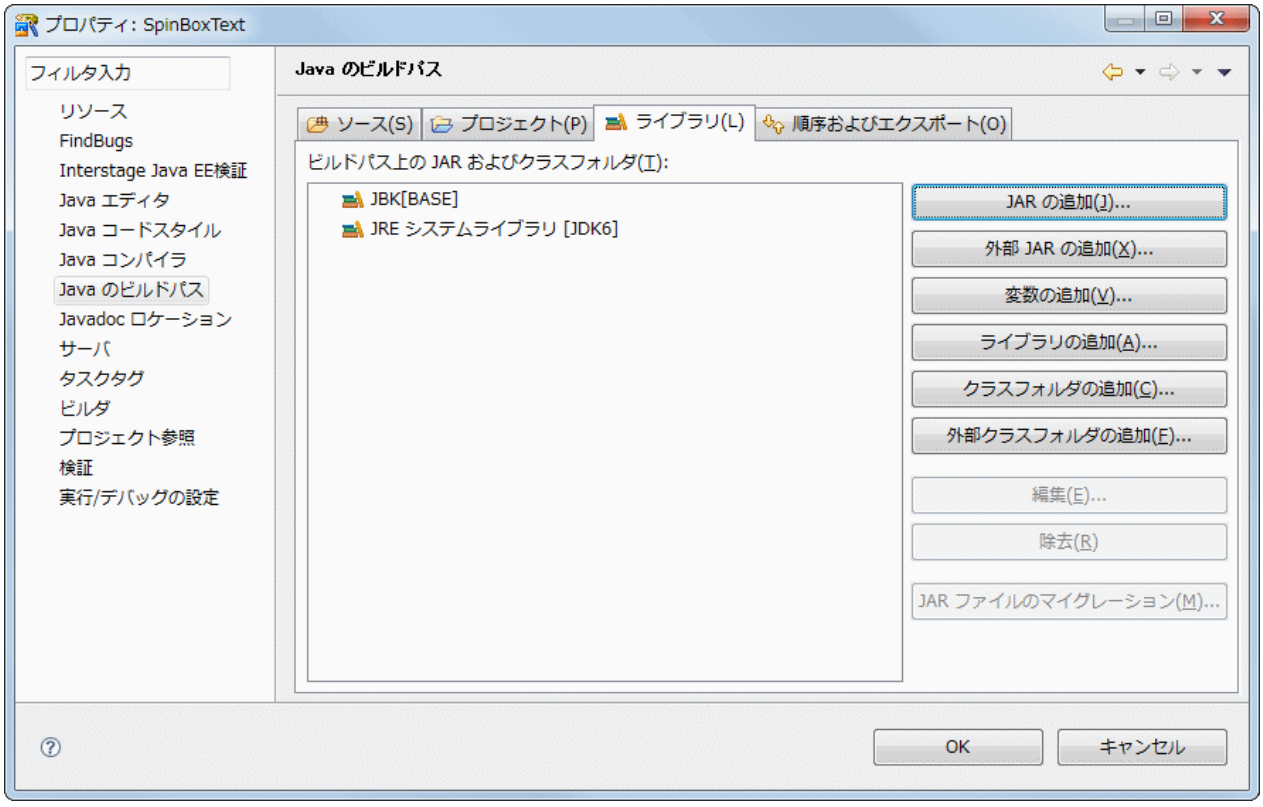
設定項目	設定内容
プロジェクト名	SpinBoxTest



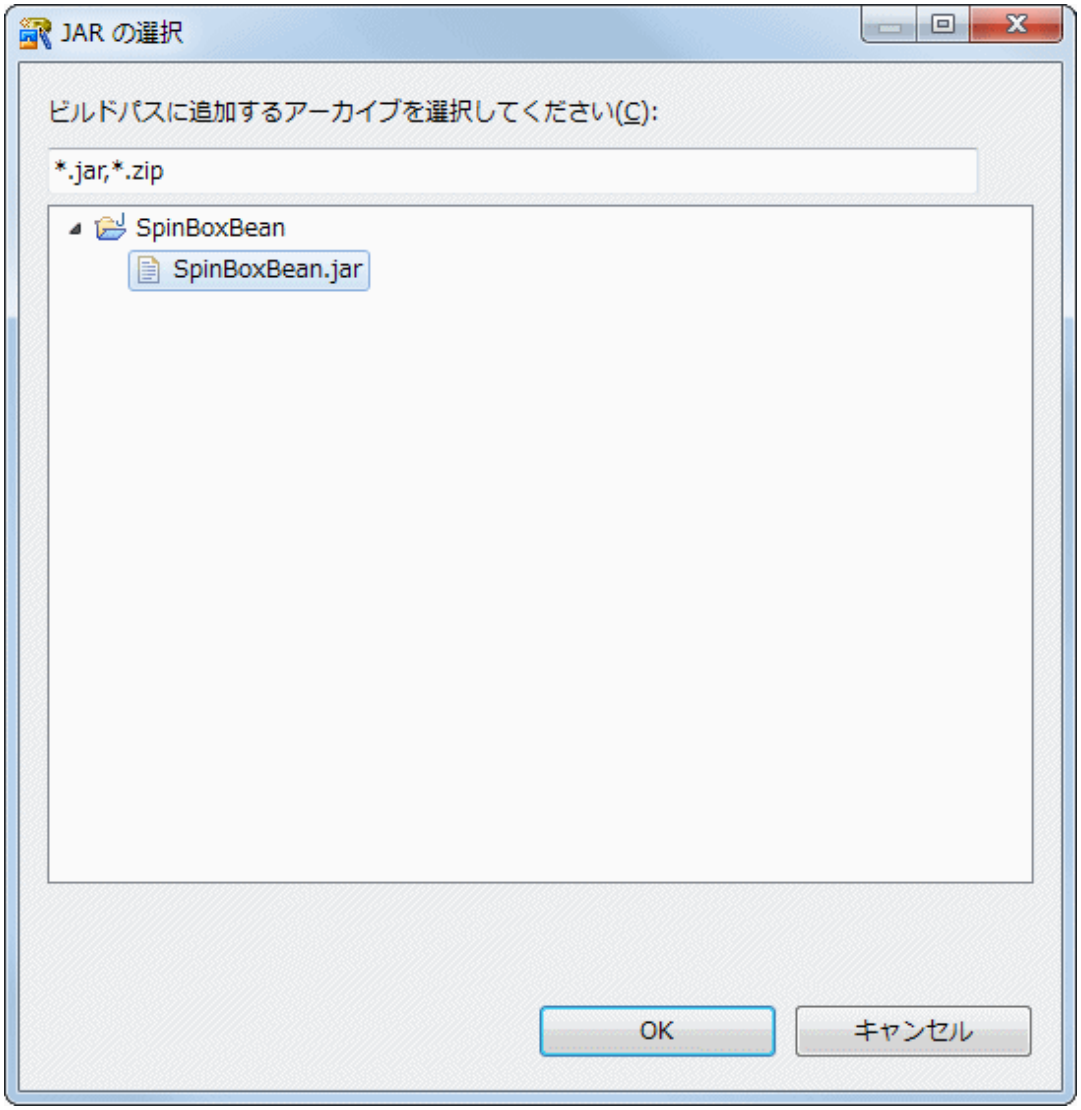
[完了]をクリックします。

4. 生成されたプロジェクトを選択し、右クリックして表示されるコンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されますので、左側のペインのツリーから[Javaのビルドパス]を選択します。[Javaのビルドパス]ページが表示されます。
このプロジェクトでは先程作成したBeanを利用するため、「SpinBoxBean」プロジェクトにある「SpinBoxBean.jar」ファイルをビルド

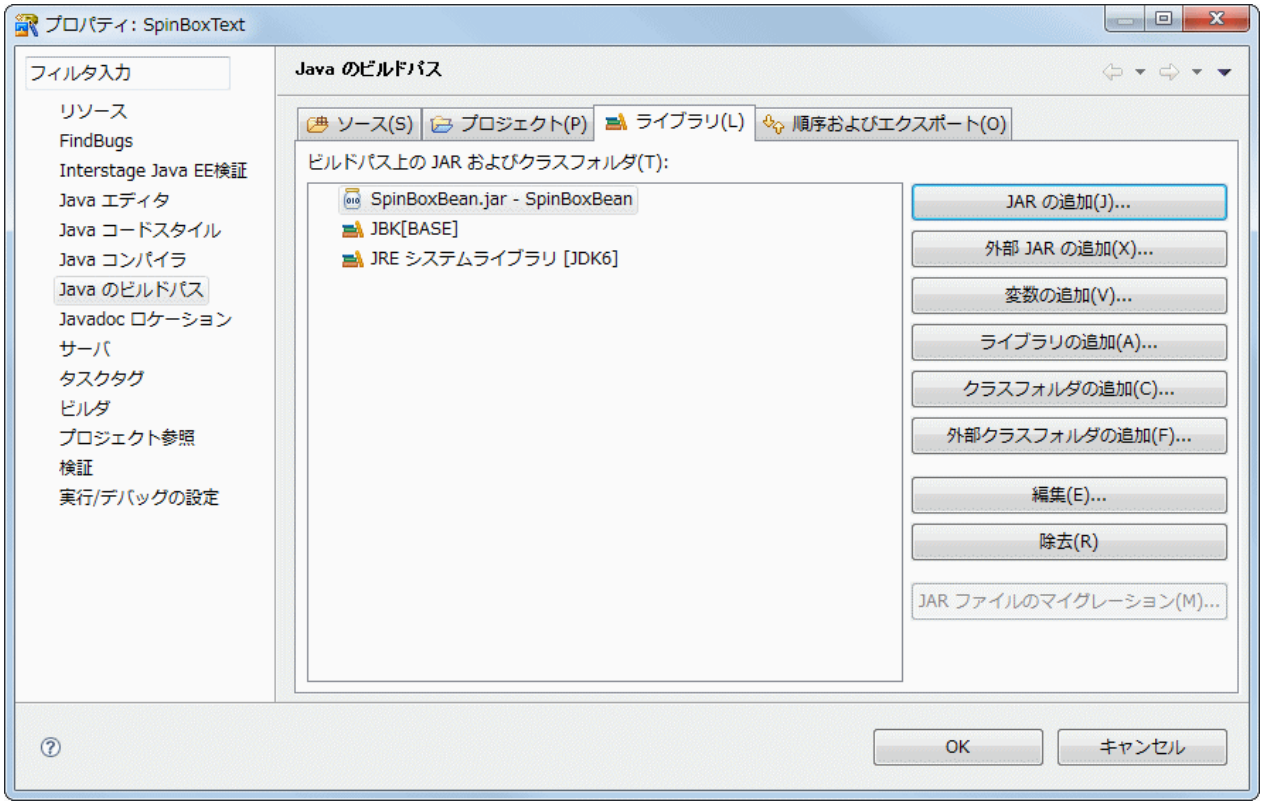
パスに追加します。
[ライブラリ]タブをクリックします。



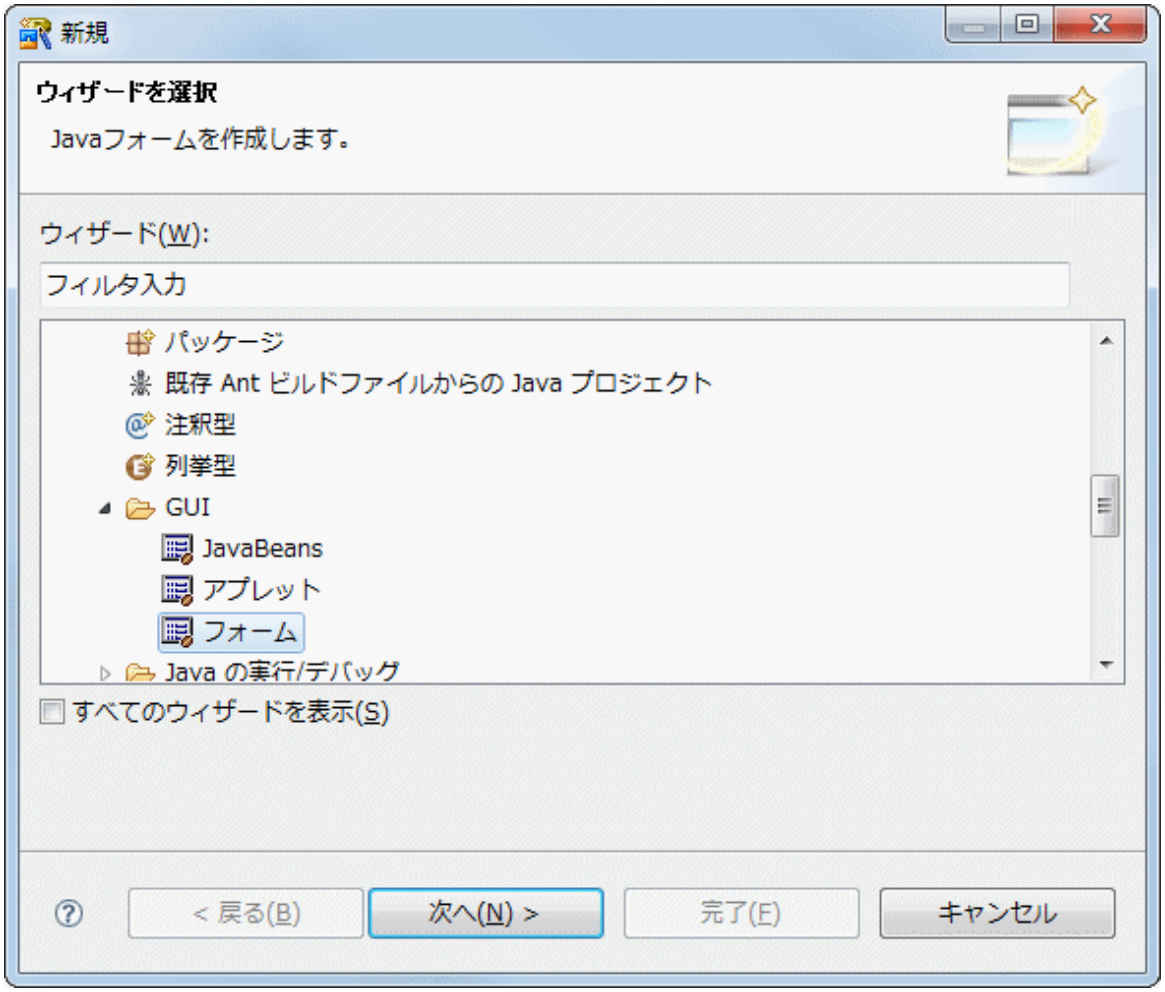
[JARの追加]をクリックすると、[JARの選択]ダイアログボックスが表示されます。[ビルドパスに追加するJARアーカイブを選択してください]にある[SpinBoxBean] > [SpinBoxBean.jar]をクリックします。
[OK]をクリックします。



[SpinBoxBean.jar]が追加されます。
[OK]をクリックします。



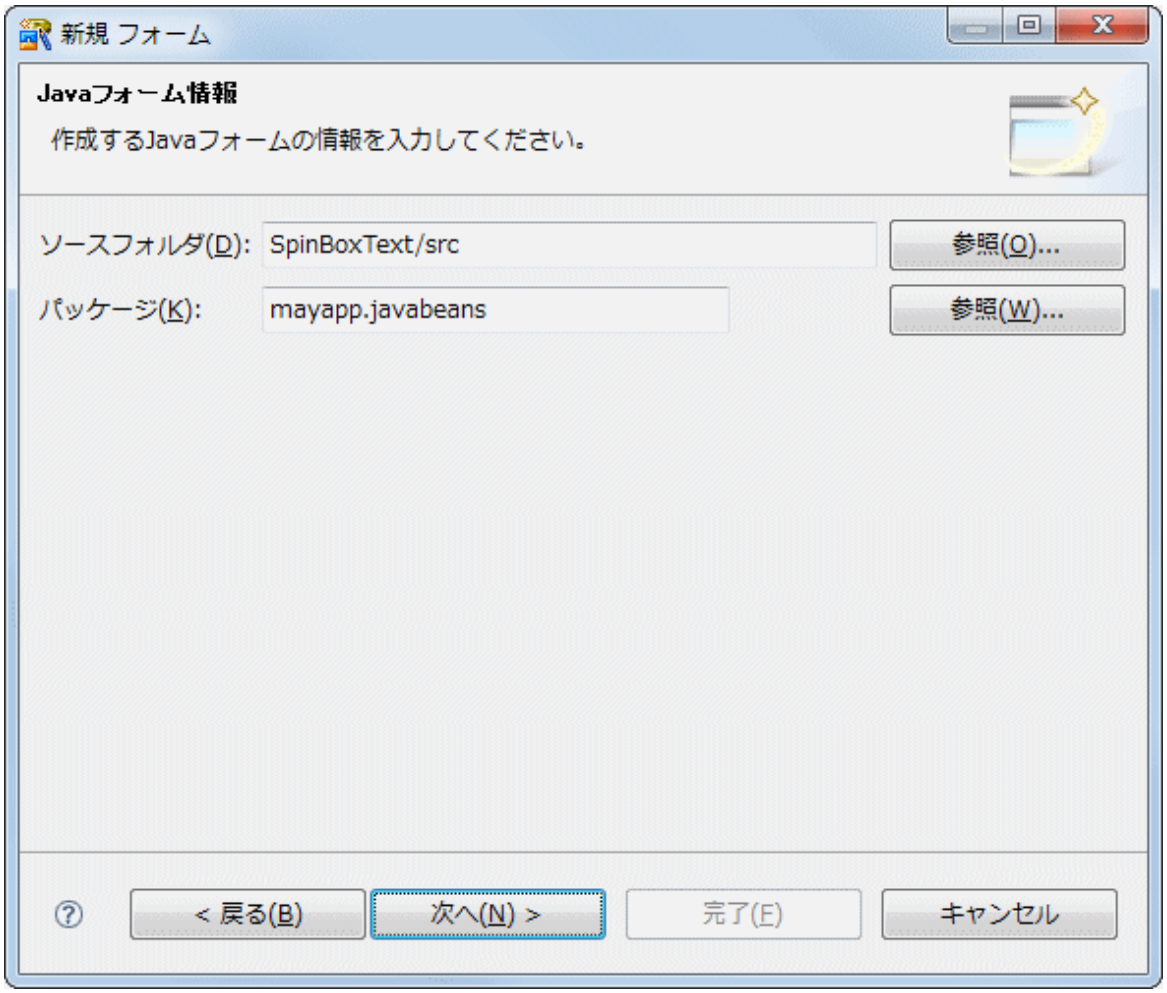
5. 次に、フレームを作成します。
ワークベンチのメニューバーから、[ファイル] > [新規] > [その他]を選択します。
[新規]ウィザードが表示されます。ツリーから[Java] > [GUI] > [フォーム]を選択します。



[次へ]をクリックします。

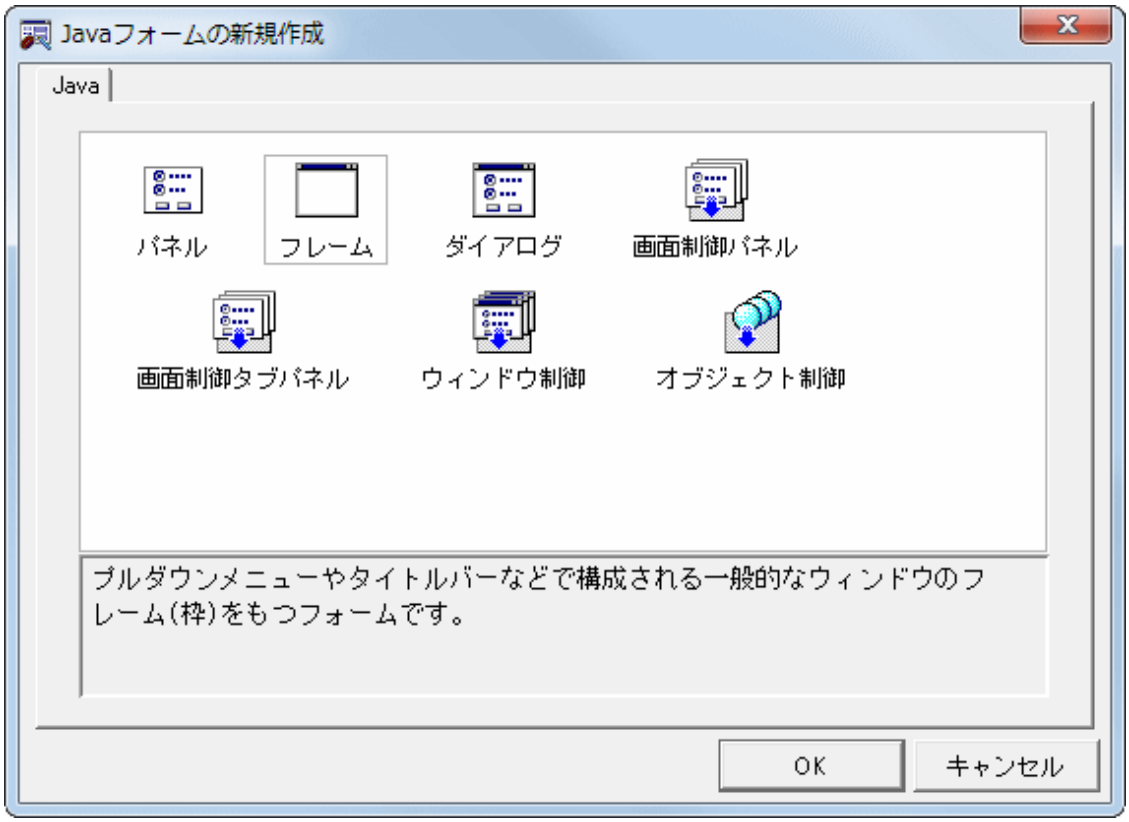
6. [Javaフォーム情報]ページが表示されます。
このページでは、以下の情報を入力します。

設定項目	設定内容
ソースフォルダ	SpinBoxTest/src
パッケージ	mayapp.javabeans


新規 フォーム
Javaフォーム情報
作成するJavaフォームの情報を入力してください。
ソースフォルダ(D):
SpinBoxText/src
参照(O)...
パッケージ(K):
mayapp.javabeans
参照(W)...
< 戻る(B)
次へ(N) >
完了(F)
キャンセル

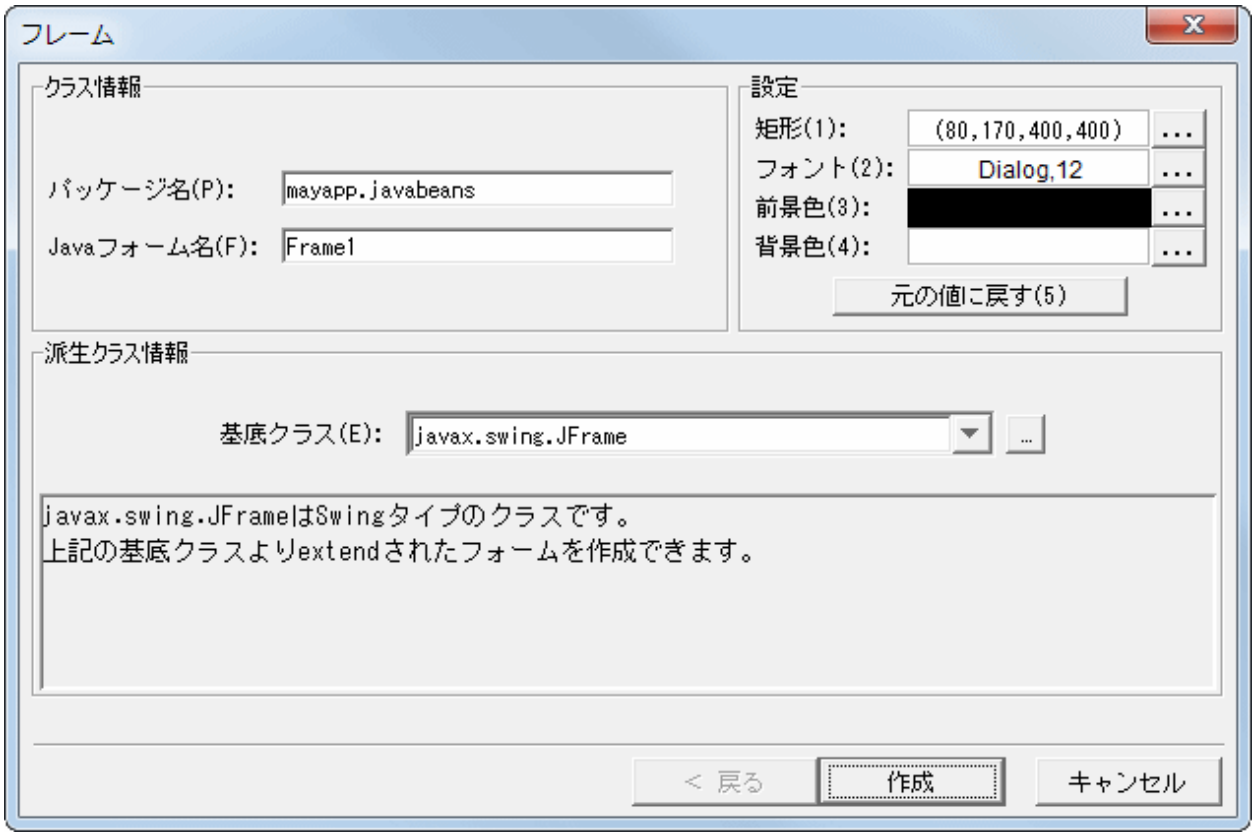
[次へ]をクリックします。

7. [Javaフォームの新規作成]ダイアログが表示されます。
[フレーム]を選択し、[OK]をクリックします。



8. [フレーム]ダイアログが表示されます。
このページでは、作成するフレームの情報を指定します。
以下の情報を入力します。

設定項目	設定内容
パッケージ名	myapp.javabeans
Javaフォーム名	Frame1
基底クラス	Javax.swing.JFrame

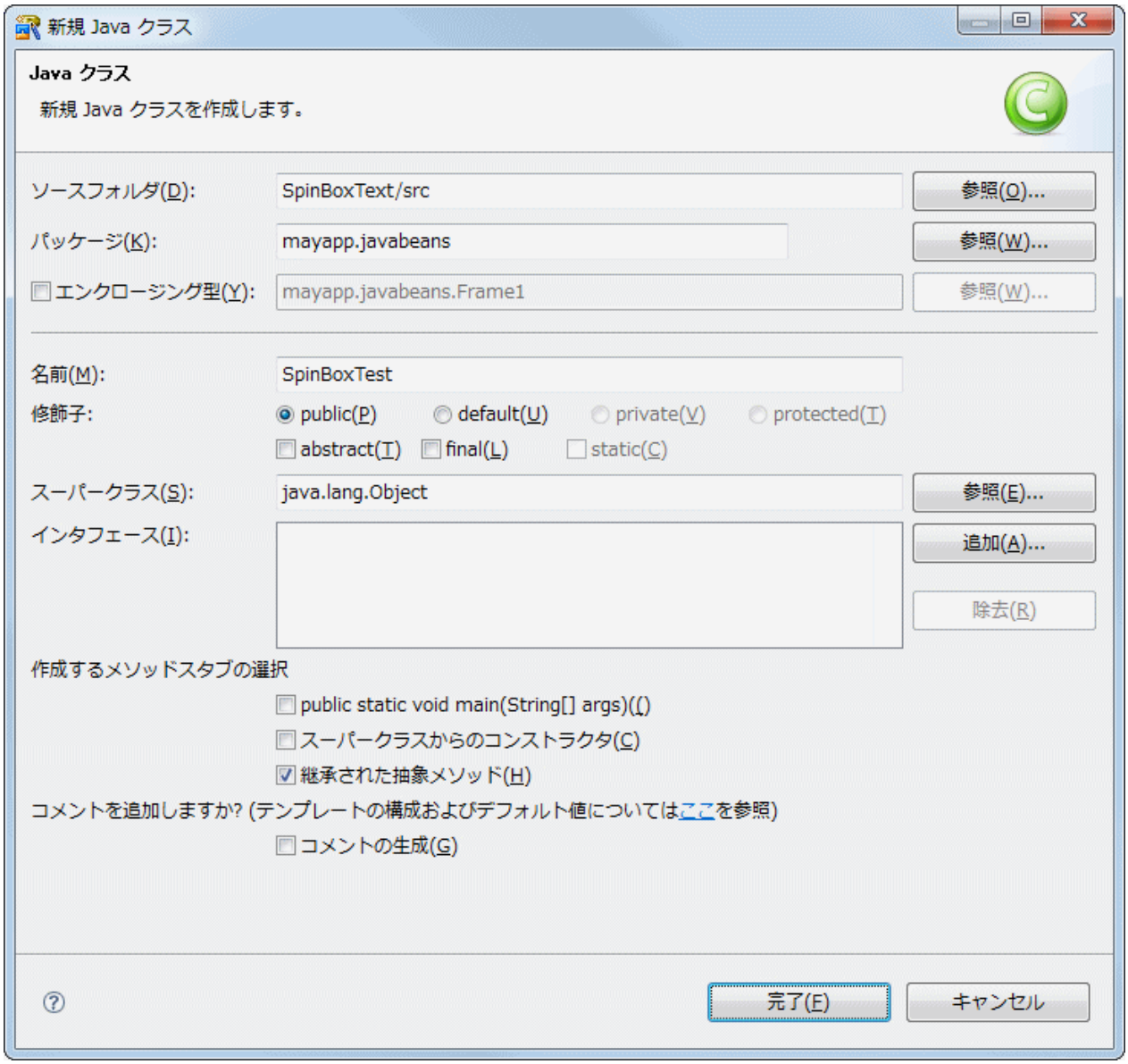


[作成]をクリックします。

9. フォームのクラスとフォームを表示する実行可能なクラスを作成します。
メニューバーから[ファイル] > [新規] > [クラス]を選択します。

10. [Javaクラス]ページが表示されます。
このページでは、作成するJavaクラスの基本情報を指定します。
以下の情報を入力します。

設定項目	設定内容
ソースフォルダ	SpinBoxTest/src
パッケージ	myapp.javabeans
名前	SpinBoxTest
継承された抽象メソッド	チェックする



[完了]をクリックします。

11. [Javaエディタ]に作成されたソースが表示されます。
作成されたソースを修正します。赤字の部分を追加します。

```
package myapp.javabeans;

public class SpinBoxTest {
    //コンストラクタ
    public SpinBoxTest() {
    }

    public void run(String[] args) {
        //新規フォームプロジェクトの作成。
        Frame1 form = new Frame1();
        //フォームの表示。
        form.setVisible(true);
    }

    //メイン処理
    public static void main(String[] args) {
        // SpinBoxTestクラスのインスタンスを作成
```

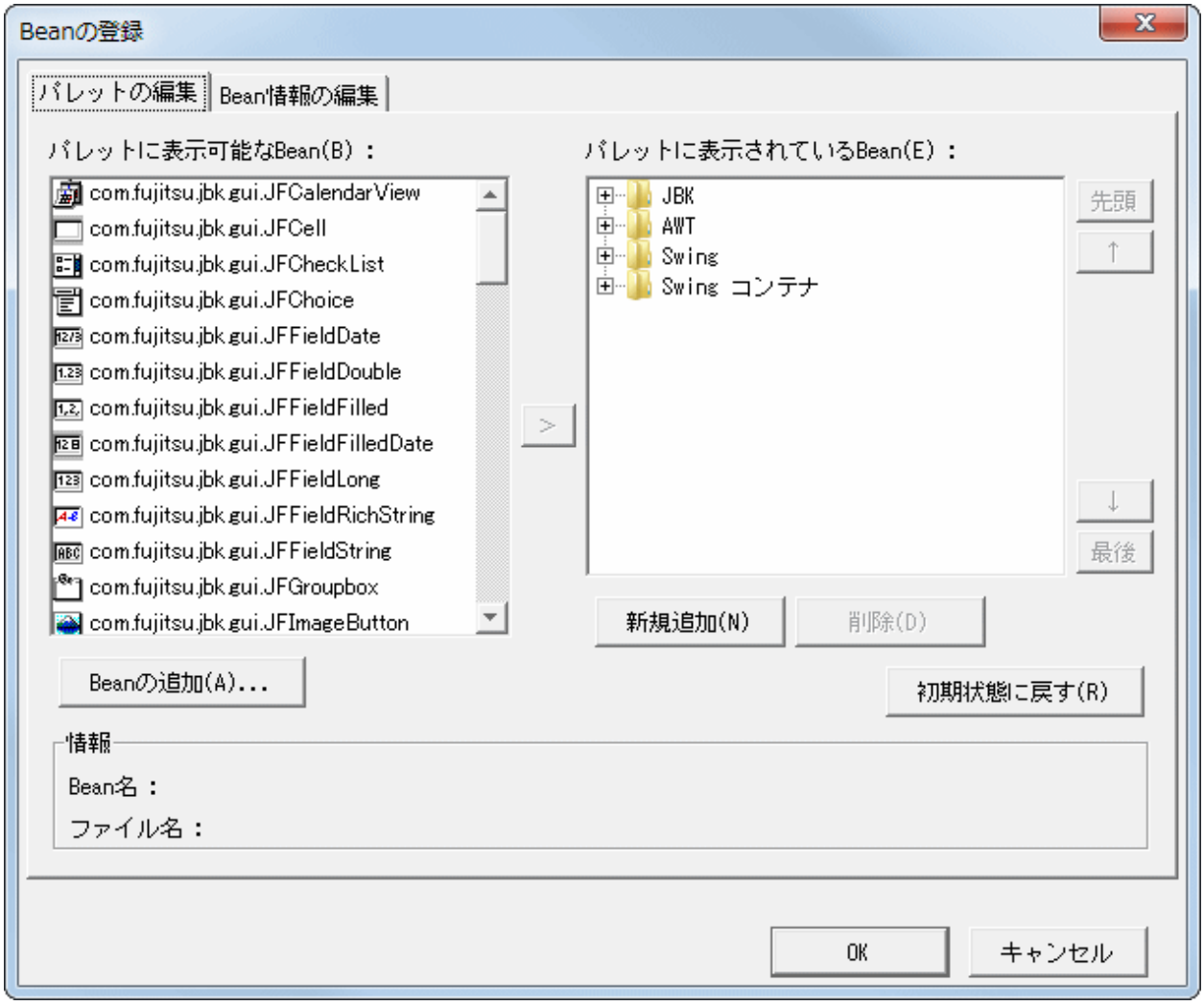
```
        SpinBoxTest object = new SpinBoxTest();
        // SpinBoxTestクラスのインスタンスのrunメソッドを呼び出す。
        object.run(args);
    }
}
```

Beanの登録

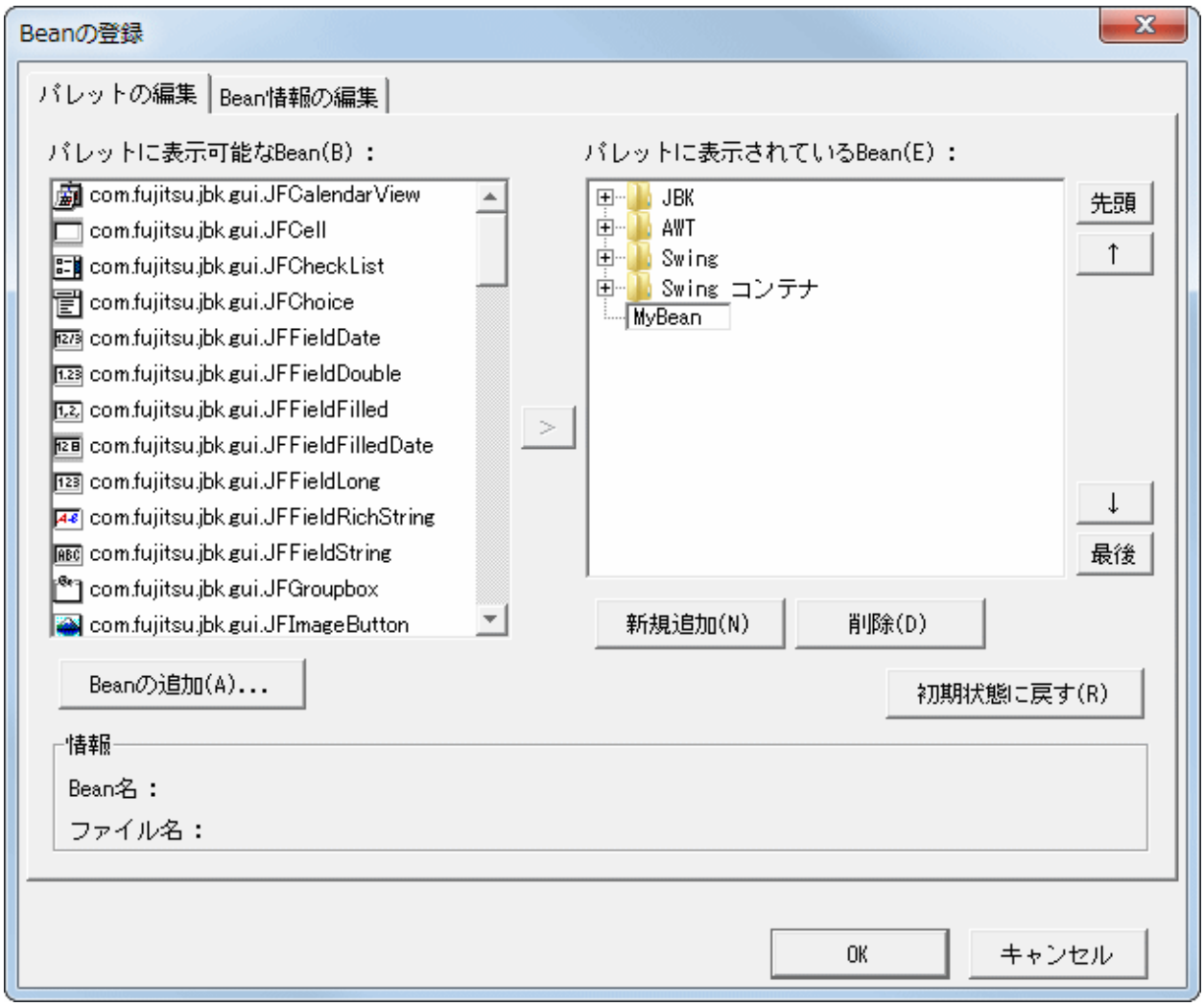
1. [Frame1.java]をJavaフォーム定義で開きます。
   ワークベンチの[パッケージエクスプローラ]ビューにある[SpinBoxTest] > [src] > [myapp.javabeans] > [Frame1.java]を右クリックしてコンテキストメニューを表示します。そして[アプリケーションから開く] > [グラフィカルエディタ]を選択します。

2. 先程作成したBeanを、Javaフォーム定義に登録します。登録することにより新規開発したBeanを、JBKのBeanなどと同様にJavaフォームに貼り付けて使用できるようになります。
   Javaフォーム定義のメニューバーから[環境設定] > [Beanの登録]を選択します。

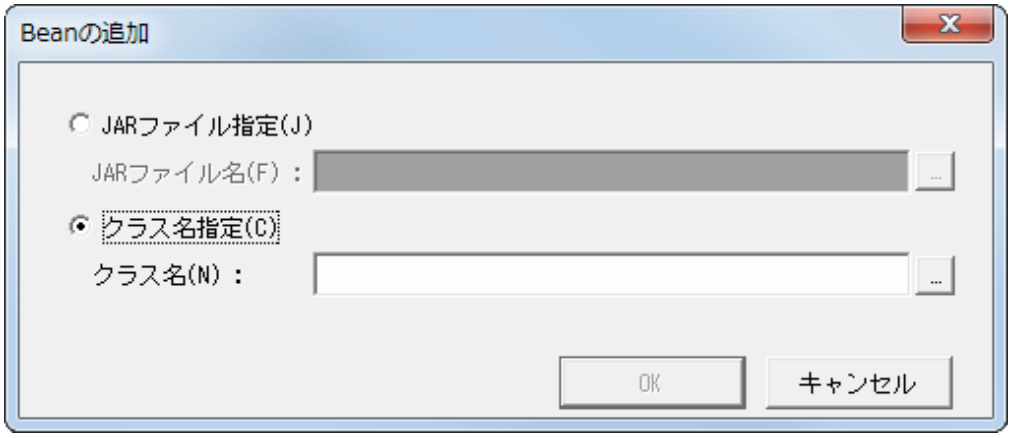
3. [Beanの登録]ダイアログボックスが表示されます。



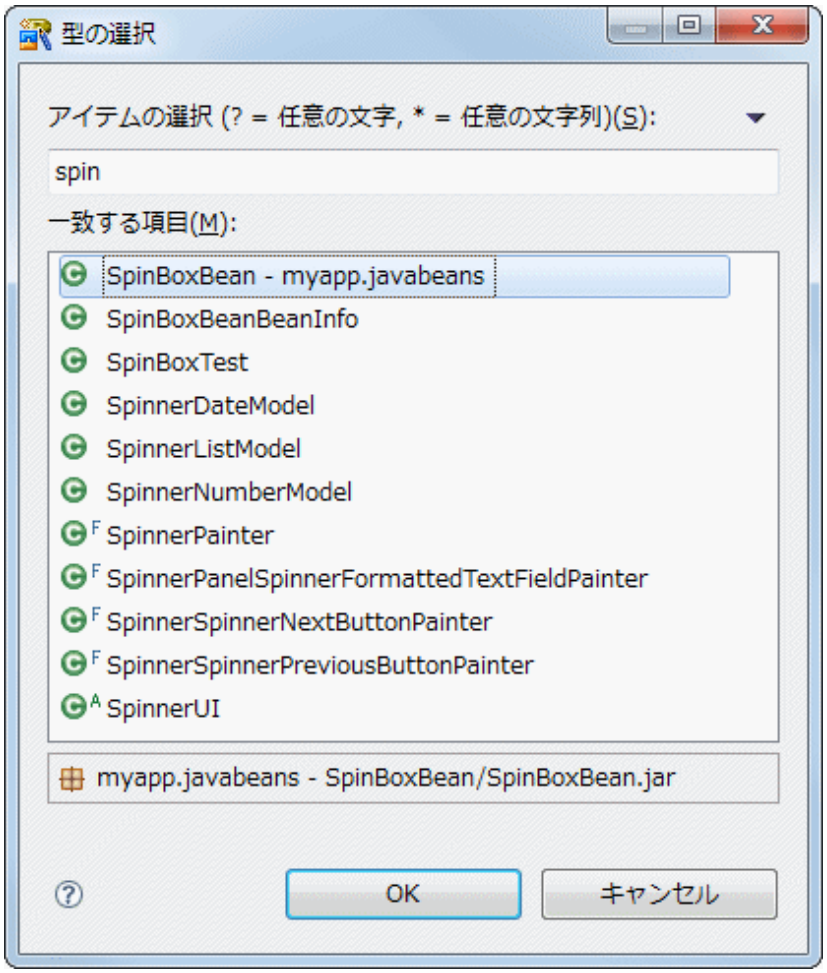
4. [新規追加]をクリックします。「MyBean」と入力します。



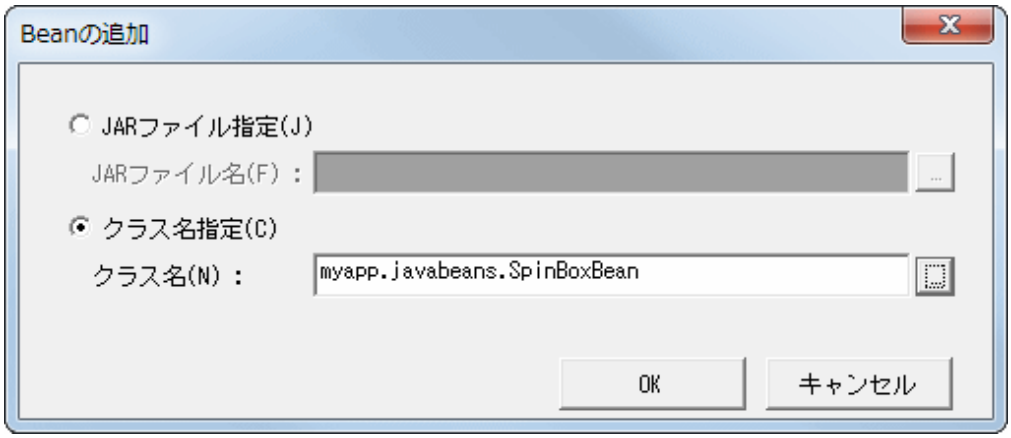
5. 次に、[Beanの追加]をクリックします。[Beanの追加]ダイアログボックスが表示されます。
先程作成した「SpinBoxBean」を指定します。[クラス名指定]を選択し、参照ボタン[...]をクリックします。



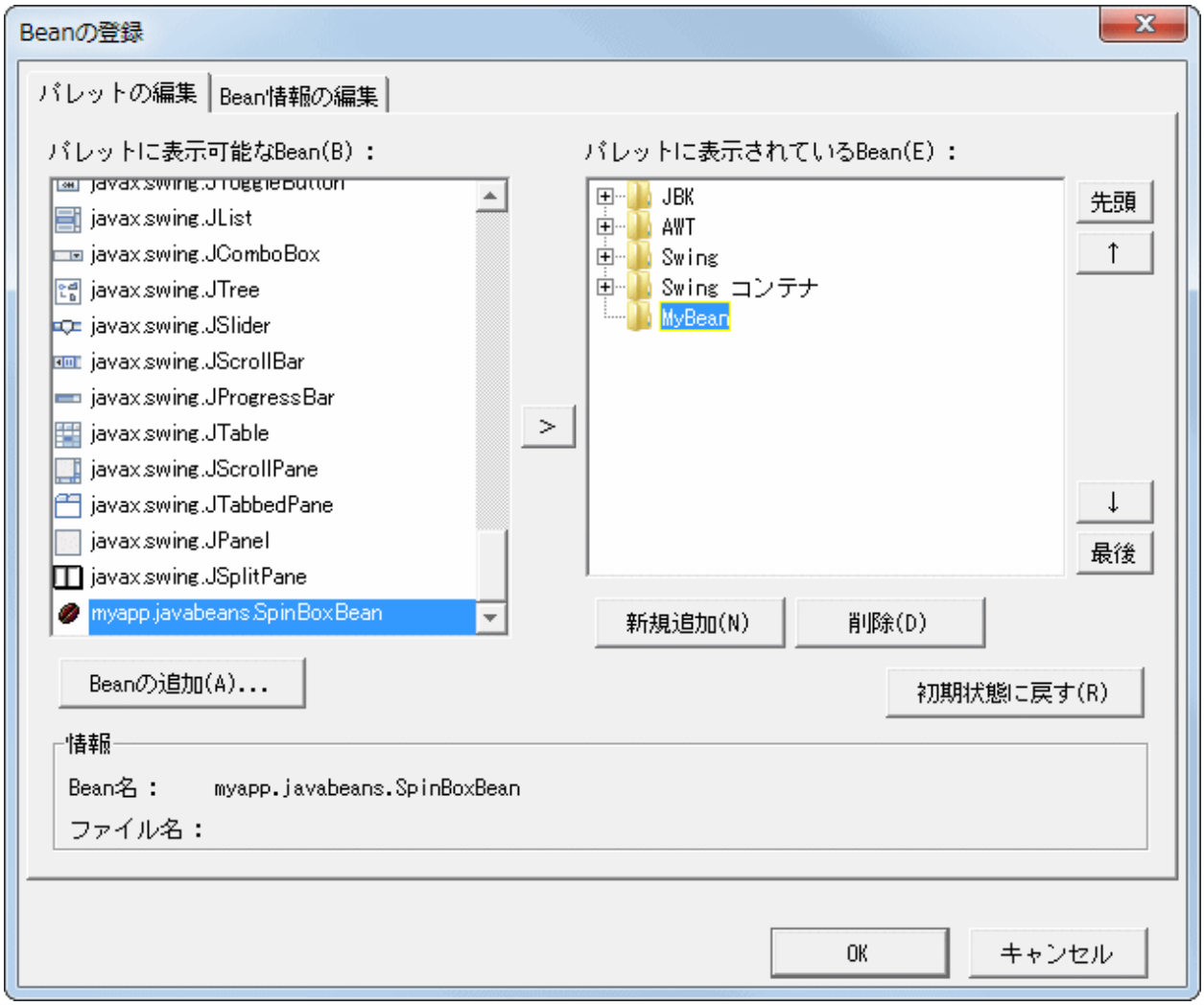
6. [型の選択]ダイアログボックスが表示されます。
[アイテムの選択]エディットボックスに[spin]と入力すると[一致する項目]に[SpinBoxBean]が表示されるので、これを選択します。



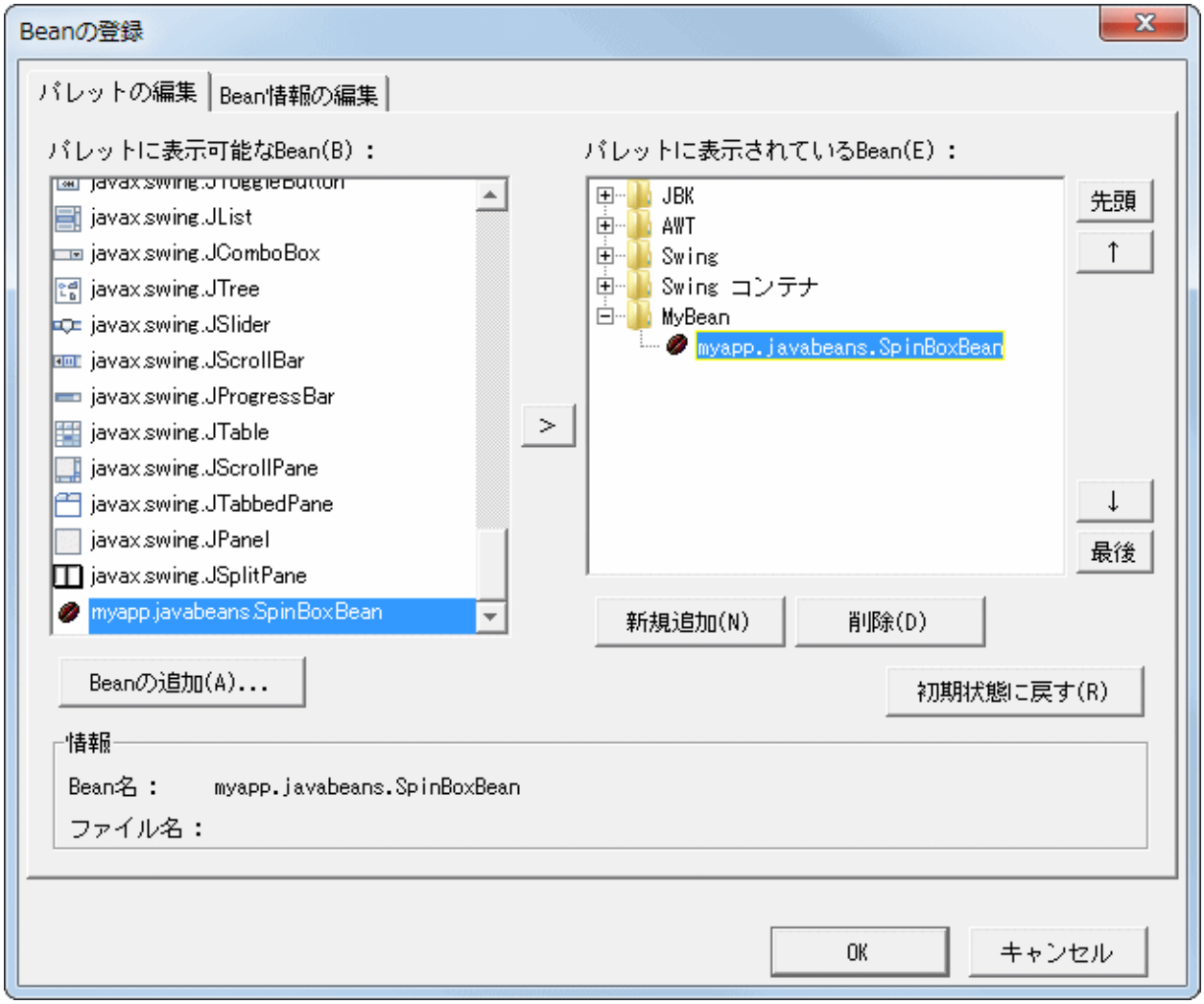
7. [クラス名]に「SpinBoxBean」が設定されます。
[OK]をクリックします。



8. [パレットに表示可能なBean]に追加された「myapp.javabeans.SpinBoxBean」と、[パレットに表示されているBean]の「MyBean」を選択し、[>]ボタンをクリックします。



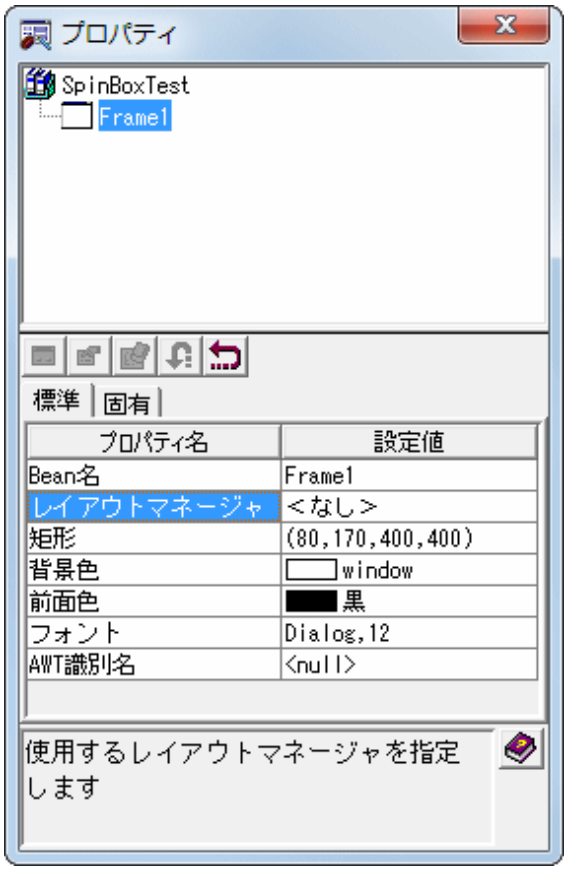
MyBeanの下に、SpinBoxBeanが登録されます。
これで、JavaフォームにSpinBoxBeanを貼り付けることが可能になりました。



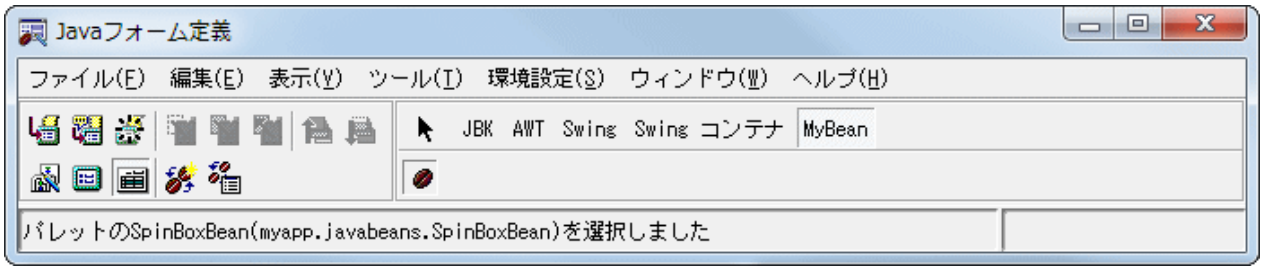
9. [OK]をクリックして、Beanの登録画面を閉じます。

Javaフォームの編集

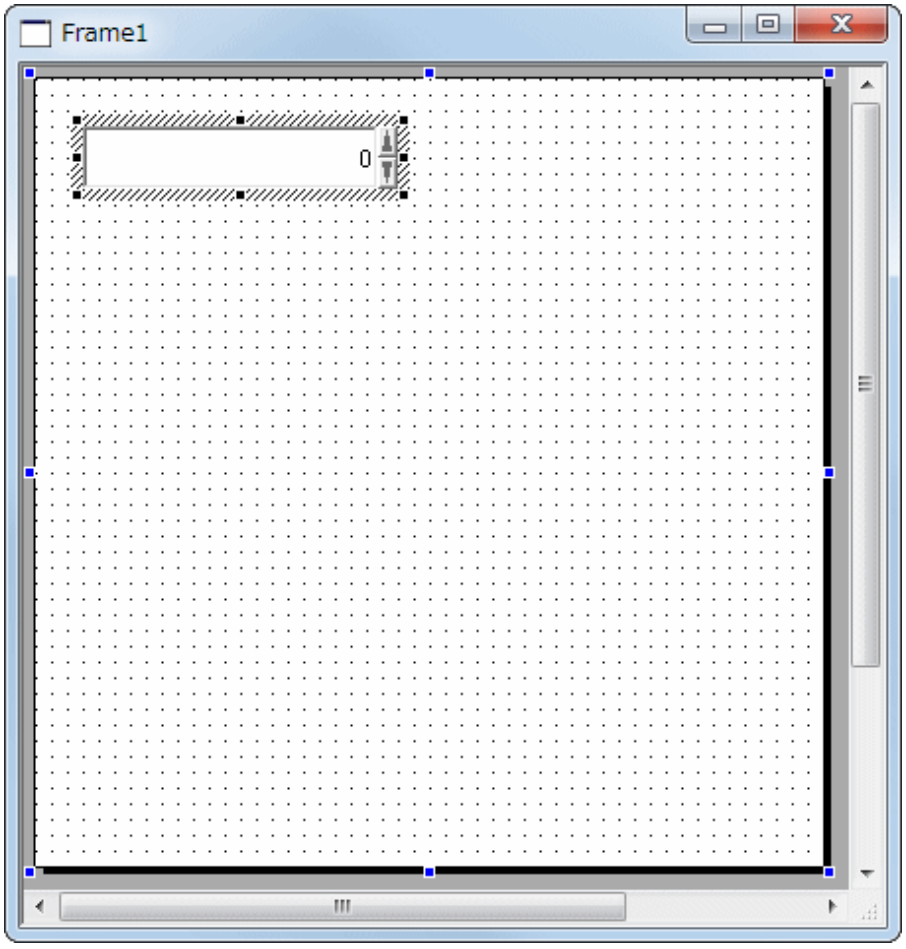
1. [Frame1]のプロパティ[レイアウトマネージャ]を[＜なし＞]にします。



2. SpinBoxBeanを貼り付けます。
   オブジェクトパレットの[MyBean]をクリックして選択してから、[SpinBoxBean]をクリックします。なお、BeanInfo定義でアイコンを指定した場合は、オブジェクトパレットにそのアイコンが表示されます。このチュートリアルではアイコンの指定をしていないので、オブジェクトパレットにデフォルトのアイコンが表示されています。

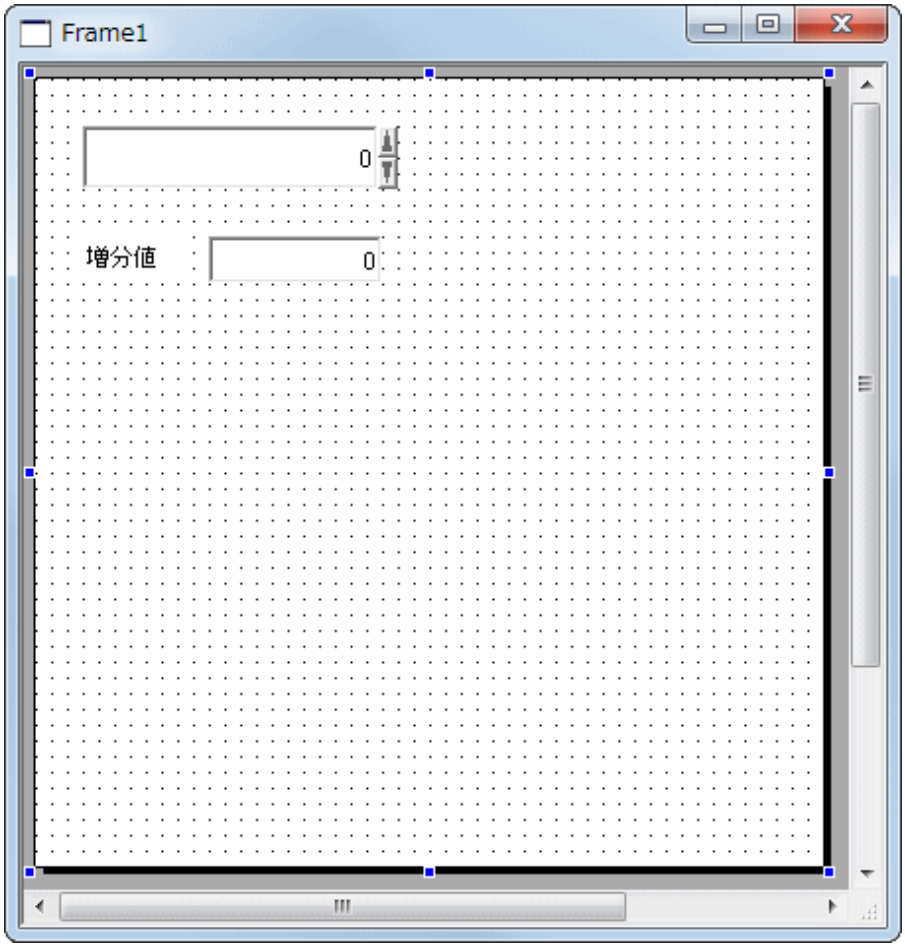


3. マウスを使ってJavaフォーム上でドラッグして配置します。



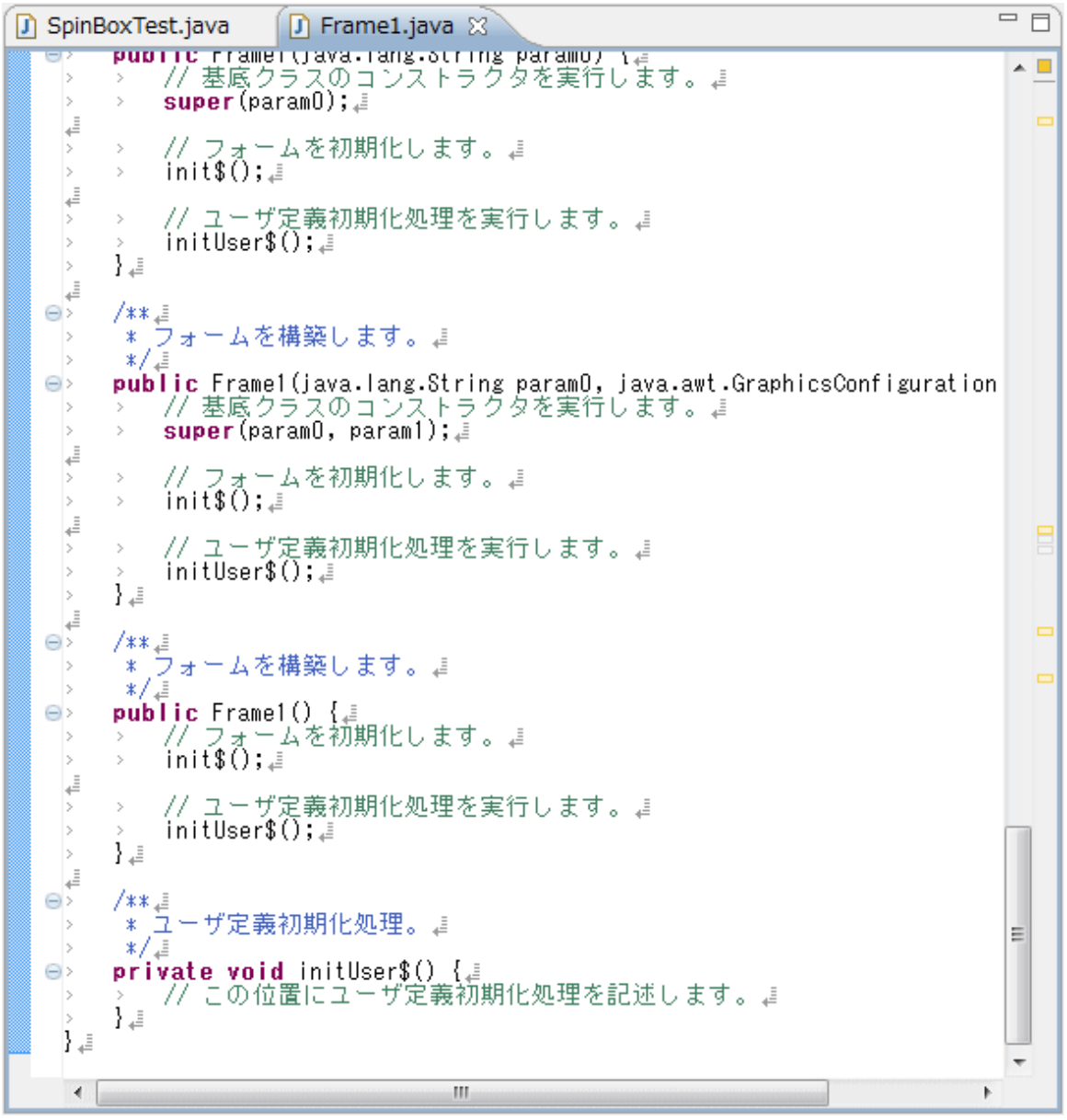
4. JBKの[複数行ラベル]を貼り付け、プロパティシートを使って、文字列プロパティに「増分値」と設定します。

5. JBKの[整数フィールド]を貼り付けます。
   画面は以下のようになります。



イベント処理の記述

1. Frame1のinitUser$メソッドを変更します。ワークベンチのエディタエリアにある[Frame1.java]タブをクリックして、Javaエディタをアクティブにします。



2. initUser$メソッドの処理を記述します。
整数フィールド(jFFieldLong1)に初期値を設定する処理を追加します。赤字の部分を追加します。

```
protected void initUser$() {
    // この位置にユーザ定義初期化処理を記述します.
    jFFieldLong1.setValue(SpinBoxBean1.getIncrement());
}
```

3. 整数フィールド(jFFieldLong1)のactionイベントに処理を記述します。
[Beanリスト]ビューにある[Frame1] > [jFFieldLong1] > [イベント] > [action_actionPerformed]をダブルクリックします。
イベント処理の作成確認画面が表示された場合は、[はい]をクリックします。
「jFFieldLong1_action_actionPerformed」の処理手続きが表示されるので、赤字の部分を追加します。

```
public void jFFieldLong1_action_actionPerformed(java.awt.event.ActionEvent e) {
    if (!defaultEventProc(e)) {
        // ここにイベント発生時の処理を記述します.
```

```
            SpinBoxBean1.setIncrement(jFFieldLong1.getValue());
        }
    }
```

4. フレームが閉じた場合に、アプリケーションを終了するように処理を記述します。
   フレームが閉じた場合は、windowイベントが発生します。windowイベントに対応した「window_windowClosing」イベントに処理を記述します。
   [Beanリスト]ビューにある[Frame1] > [イベント] > [window_windowClosed]をダブルクリックします。
   イベント処理の作成確認画面が表示されるので、「はい」をクリックします。
   「Frame1_window_windowClosed」の処理手続きが表示されるので、赤字の部分を追加します。

```
    public void Frame1_window_windowClosed(java.awt.event.WindowEvent e) {
        if(!defaultEventProc(e)) {
            // ここにイベント発生時の処理を記述します.
            System.exit(0);
        }
    }
```

5. Javaフォームを保存します。
   Javaフォーム定義のメニューから[ファイル] > [上書き保存]を選択します。
   メニューバーから [ファイル] > [終了]を選択し、Javaフォームを終了します。

ビルド

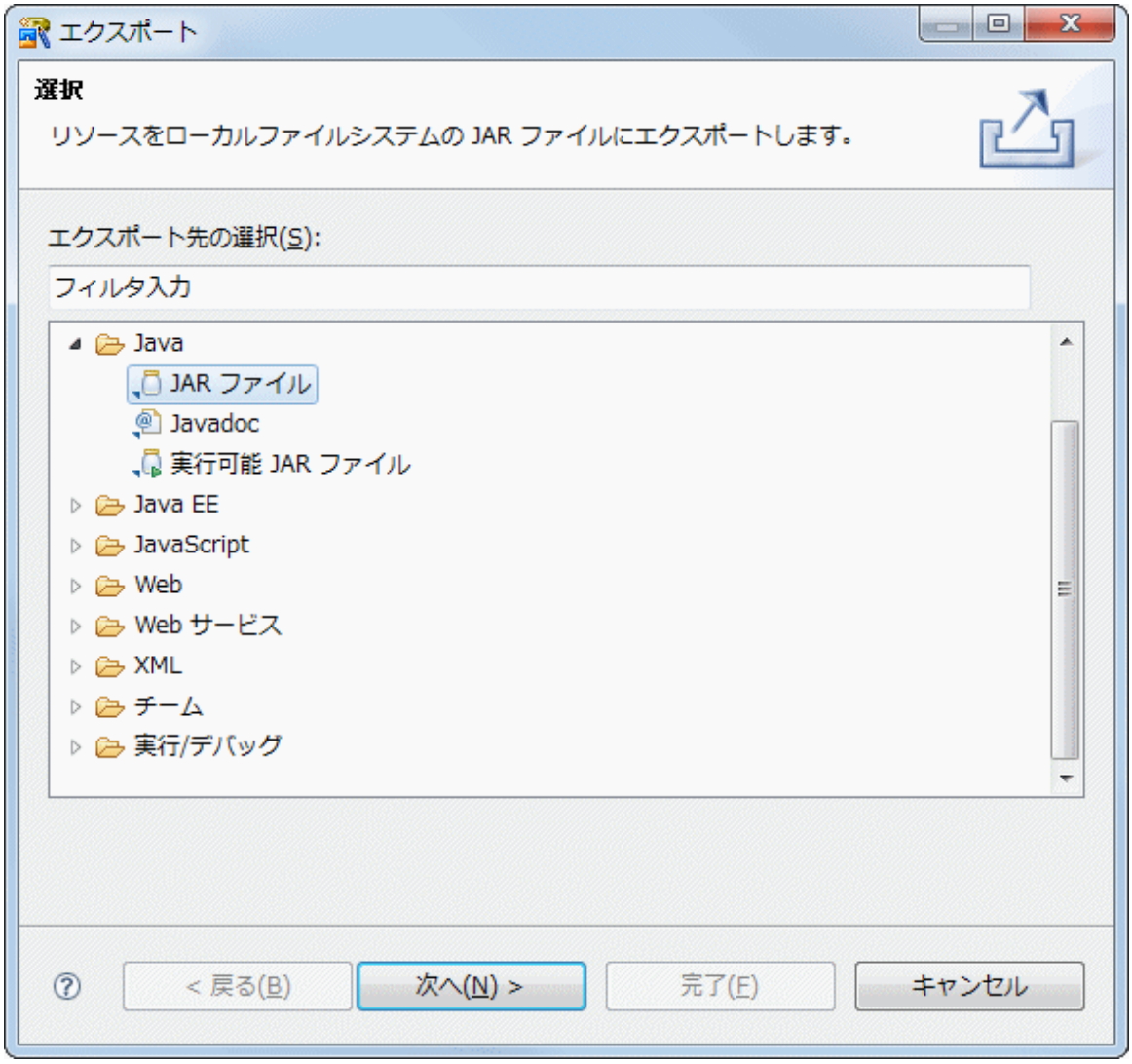
1. ビルドは、[プロジェクト]メニューの[自動的にビルド]が有効になっている場合は、Javaファイルなどのリソースファイルを保存すると自動的に実行されます。
   [自動的にビルド]が無効になっている場合は、プロジェクトを選択し右クリックメニューの[プロジェクトのビルド]を選択するか、[プロジェクト]メニューの[プロジェクトのビルド]を選択します。

SpinBoxBeanプロジェクトのクリーンについて

SpinBoxBeanプロジェクトをクリーンしてビルドすると失敗する場合があります。これは、Javaフォーム定義がSpinBoxBean.jarを使用しているため、このファイルを削除できないからです。この場合は、Javaフォーム定義を終了して、かつ、SpinBoxTestプロジェクトを閉じてください。

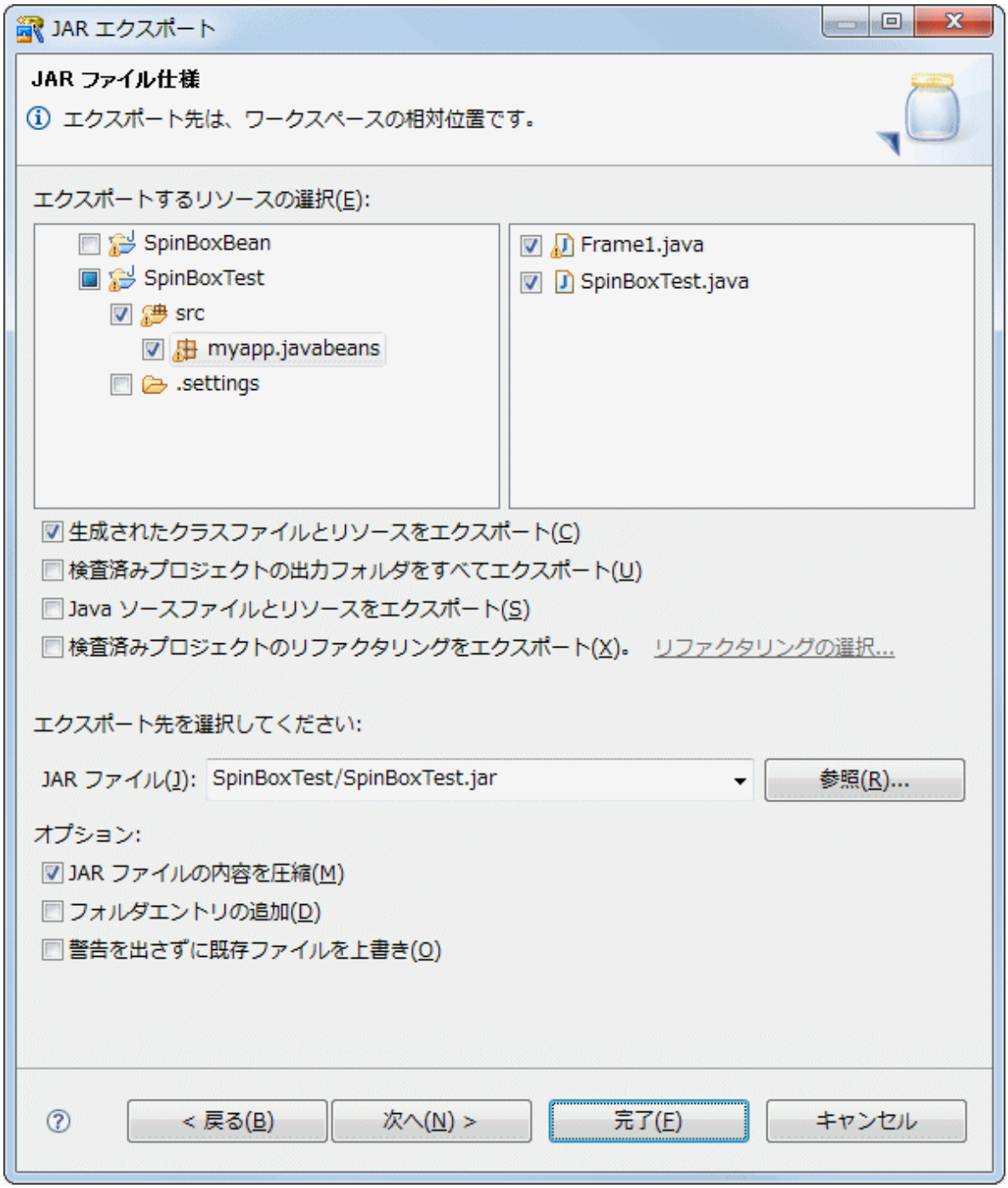
2. JARファイルについては別途必要なため、これを作成します。
   JARファイルの作成は、エクスポートウィザードから行います。

エクスポートウィザードを起動するには、[ファイル] > [エクスポート]を選択します。
エクスポートウィザードから[Java] > [JARファイル]を選択します。

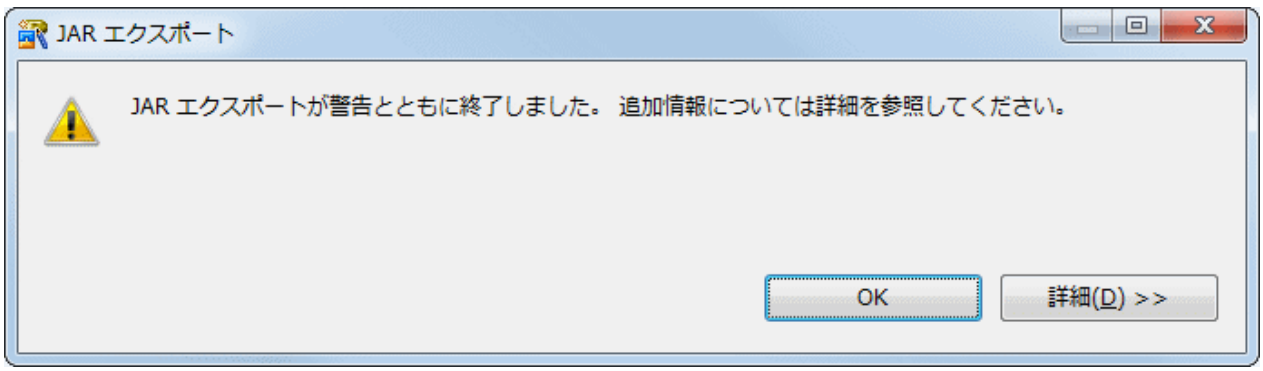


3. JARエクスポートウィザードが表示されます。
[エクスポートするリソースの選択]で[SpinBoxTest] > [src] > [myapp.javabeans]を選択し、以下の設定項目を入力します。
情報を設定後、[完了]をクリックします。

設定項目	設定内容
エクスポートするリソースの選択	SpinBoxTest.java Frame1.java
生成されたクラスファイルとリソースをエクスポート	チェックする
JARファイル	SpinBoxTest/SpinBoxTest.jar
JARファイルの内容を圧縮	チェックする

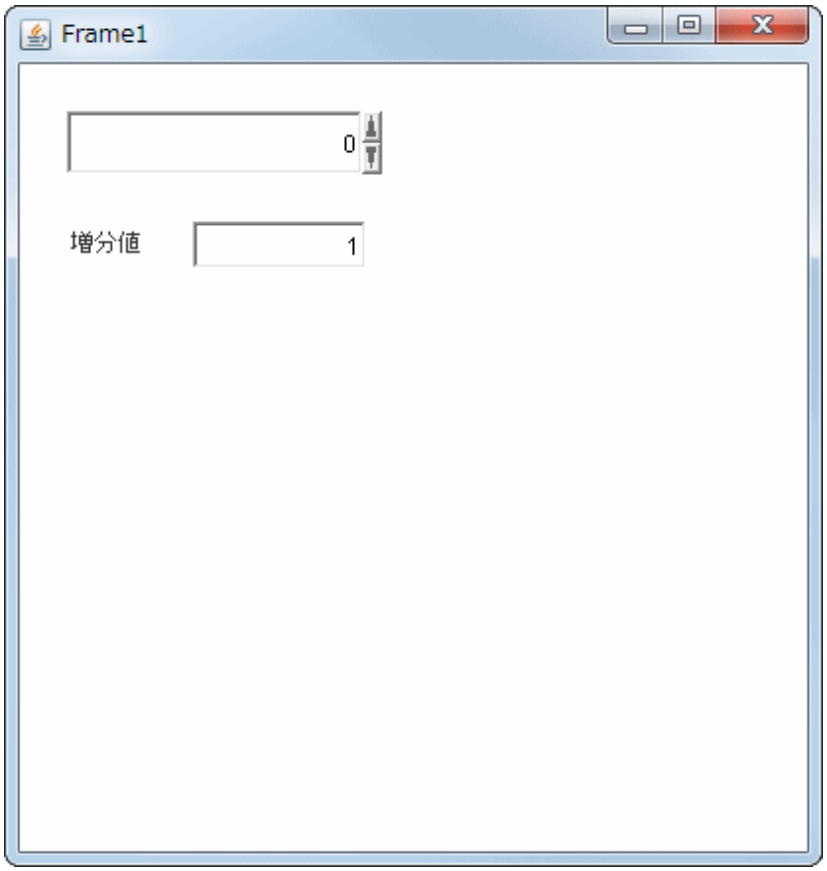


JARエクスポートでは以下のメッセージが出力されますが、問題ありません。

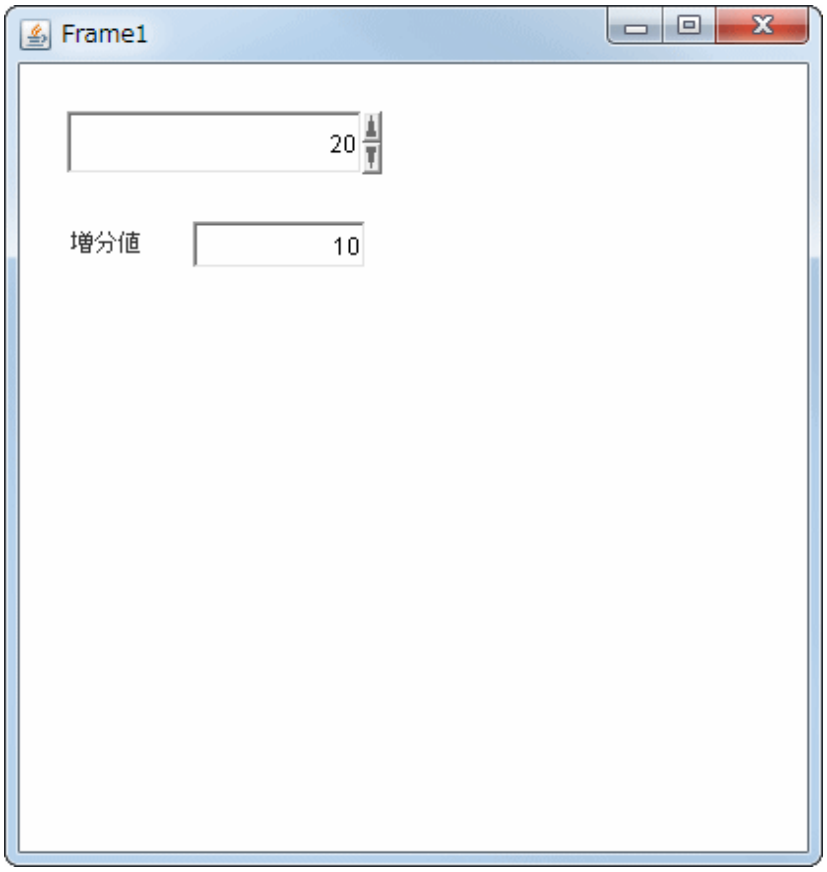


実行

1. 実行するファイル(クラス)を選択します。 ワークベンチの[パッケージエクスプローラ]ビューにある[SpinBoxTest] > [src] > [myapp.javabeans] > [SpinBoxTest.java]をクリックします。
2. メニューバーから[実行] > [実行] > [Javaアプリケーション]を選択します。アプリケーションが起動し、Javaフォームが表示されます。



3. 増分値に「10」を入力し、[Enter]キーを押します。
   スピンボタンを押して、値が10単位に変更されることを確認します。



4. タイトルバーのクローズボタン([×]ボタン)をクリックして、アプリケーションを終了します。

索 引