Web コンポーネントの プログラミング

Forte™ for Java™ プログラミングシリーズ

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

Part No. 816-2849-01
2001 年 10 月 Revision A

Copyright © 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303, U.S.A. All rights reserved.

本製品に採用されているテクノロジに関する知的財産権は Sun Microsystems, Inc. が保有しています。特に、これらの知的財産権には、ウェブサイト http://www.sun.com/patents にリスト表示されている米国特許、または米国および他の国へ出願中の特許が含まれている可能性があります。

本製品は、本製品やドキュメントの使用、コピー、配布、および逆コンパイルを規制するライセンス規定に従って配布されます。本製品のいかなる部分も、その形態および方法を問わず、Sun およびそのライセンサーの事前の書面による許可なく複製することを禁じます。

フォントテクノロジを含むサードパーティ製のソフトウェアの著作権およびライセンスは、Sun のサプライヤが保有しています。PointBase ソフトウェアは社内開発での使用のみを目的としており、商用で使用する場合には別途 PointBase からライセンスを取得する必要があります。

Sun、Sun Microsystems、Sun のロゴ、Forte、Java、Jini、Jiro、Solaris、iPlanet、および NetBeans は、米国および他の各国における Sun Microsystems, Inc. の商標または登録商標です。

SPARC は SPARC International, Inc. の米国および他の各国における商標または登録商標であり、同社とのライセンス契約のもとで使用されています。SPARC の商標を使用した製品は Sun Microsystems, Inc. が開発したアーキテクチャに基づいています。

連邦政府による取得：市販ソフトウェア -- 米国政府機関による使用は、標準のライセンス条項に従うものとします。

原典：　*Building Web Components*
Part No: 816-1410
Revision A

© 2001 by Sun Microsystems, Inc.

目次

図目次

表目次

はじめに

『Web コンポーネントのプログラミング』では、Java™ 2 Platform, Enterprise Edition (J2EE™) Web コンポーネントを使用して Web アプリケーションを作成するユーザーに必要な情報を提供します。Forte™ for Java™ プログラミングシリーズの一部として、このマニュアルでは、J2EE のコンテキストにおける Web アプリケーションの開発と、Java サーブレットや JSP (JavaServer Pages™) などのサポート技術に焦点を絞ります。

特に、このマニュアルでは、Web アプリケーションが JSP ページ、Java サーブレット、JSP タグライブラリ、およびサポートするクラスとファイルを使用する方法について説明します。これらの Web アプリケーションは、データベースなどの持続データを使用することがあります。これらは、すべての機能が Web コンテナで管理される、独立したアプリケーションの場合もあります。または、ユーザーインタフェースを提供する一方で、ビジネスロジックの実行や持続データへのアクセスなど、その他のサービスについて J2EE Enterprise JavaBeans (EJB™) コンテナのコンポーネントに依存する場合もあります。

対象読者

このマニュアルの対象読者は、アプリケーションコードを記述する Web アプリケーション開発者、またはアプリケーションと対話する方法を指定し、インタフェースコンポーネントを選択し、それらをレイアウトする Web アプリケーション設計者です (特に注意がない場合、Web アプリケーションとは J2EE Web アプリケーションを指します)。Web アプリケーション開発者は、Webアプリケーション設計者と同一である場合と同一でない場合があります。いずれの場合も、このマニュアルでは、読者が Java プログラム、JSP ページプログラミング、および HTML コーディングの一般的な知識

を有することを前提としています。このマニュアルの情報は、Web コンポーネントベースのアプリケーション作成に関与するテクニカルライター、グラフィックアーティスト、プロダクションおよびマーケティング専門家、および試験業者にも役立ちます。

このマニュアルで扱われる項目

『Web コンポーネントのプログラミング』には、次の情報が記載されています。

第 1 章は、Web アプリケーションのコンポーネント作成に使用される J2EE 技術の概要を説明します。

第 2 章は、Forte for Java IDE を使用して Web アプリケーションをプログラミングするプロセスを説明します。

第 3 章は、Forte for Java IDE を使用して Web アプリケーションをテスト実行、デバッグ、および配備するプロセスを説明します。

付録 A は、Dreamweaver™ テンプレートと JSP ページを使用して作業する方法について説明します。

付録 B は、Forte for Java IDE を使用してカスタムタグライブラリをプログラミングする方法について説明する簡単なチュートリアルを提供します。

用語集は、このマニュアルで使用される重要な用語を定義します。用語集で扱われる用語は、マニュアル内でゴシック体で表記されています。

このマニュアルで扱われない項目

このマニュアルでは、アプリケーションの設計方法について詳しく説明していません。また、EJB コンポーネントの開発方法についても深く掘り下げていません。参考文献については、xii ページの「関連情報」を参照してください。

お読みになる前に

IDE に組み込まれている「CDShopCart」に必ず目をとおしてください。このチュートリアルでは、Forte for Java IDE のツールを使用して簡単な Web アプリケーションを構築する手順が説明されています。

表記上の規則

表 P-1　表記上の規則

字体または記号	意味	例
`AaBbCc123`	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	`.login` ファイルを編集します。 `ls -a` を使用してすべてのファイルを表示します。 `system%`
`AaBbCc123`	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	`system% su` `Password:`
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには `rm` *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	`sun% grep '^#define \` `  XV_VERSION_STRING'`

関連情報

次の参考文献は、Web アプリケーションの設計と実装に関連するトピックで入手できます。

- Alur、Deepak、Crupi、John、Malks、Dan、Core J2EE Patterns、Sun Microsystems Press、Prentice Hall、2001。Web アプリケーションのアーキテクチャとモデルについて詳しく解説しています。一般的な問題に対する J2EE ベースの解決策など、前後関係で発生する問題への再帰的な解決策が示されており、Java 設計者と Sun Java Center に蓄積された経験が反映されています。
- Kassem、Nicholas、Designing Enterprise Applications With the Java 2 Platform, Enterprise Edition (The Java Series)、Addison-Wesley、2001。Sun Enterprise Team のさまざまなメンバーが記述した本書は、Java を使用して分散型 Web アプリケーションを構築するための公式な青写真と言えます。

Java 2 プラットフォームの公式ドキュメント

次のドキュメントは、このマニュアルの基底となる技術の詳細なバックグラウンドを説明しています。

- *Sun BluePrints™ Design Guidelines for J2EE*
  `http://www.java.sun.com/j2ee/blueprints`
- *Java 2 Platform Enterprise Edition Specification*
  `http://www.java.sun.com/products`
- *Java™ Servlet Specification, v2.2*
  `http://www.java.sun.com/products/servlet/index.html`
- *JavaServer Pages™ Specification, v1.1*
  `http://www.java.sun.com/products/jsp/index.html`

Forte for Java プログラミングシリーズ

Forte for Java では Acrobat Reader (PDF) 形式とオンラインヘルプで構成されるマニュアルセットを提供します。この節では、これらのマニュアルについて説明します。

次のマニュアルは、Forte for Java Web サイトからダウンロードできます。

- Forte for Java プログラミングシリーズ
  - 『Web コンポーネントのプログラミング』

    JSP ページ、サーブレット、タグライブラリ、およびサポートするクラスとファイルを使用する J2EE Web モジュールとして Web アプリケーションを構築する方法を説明します。
  - 『持続プログラミング』

    JDBC (Java Database Connectivity)、TP (Transparent Persistence) など、Forte for Java IDE が提供するさまざまな持続プログラミングモデルのサポートについて説明します。

役に立つ Web サイト

- The Source for Java Technology は、Web コンポーネント技術について豊富な情報を提供します。たとえば、製品と API、Developer Connection へのアクセス、ドキュメントとトレーニング、オンラインサポート、コミュニティによる議論、業界ニュース、市場ソリューション、ケーススタディなどです。
  `http://java.sun.com` から参照できます。
- JSP Insider は、設計情報、読み物、コード、他の Web サイトへのリンク、ニュース、書評、JSP に関する議論などを盛り込んだ JavaServer Page の Web サイトです。`http://www.jspinsider.com` から参照できます。
- The JSP Resource Index は、JSP チュートリアル、スクリプト、コード、ジョブリストを検索するのに役立ちます。`http://www.JSPin.com` から参照できます。
- The Jakarta Project は、オープンかつ協力的な方法論で開発された Java プラットフォームベースの商用サーバーソリューションを提供します。Jakarta は、Jakarta `Taglibs` および受賞経験もある Tomcat 3.2 Server など多数のサブプロジェクトで構成されるプロジェクトの総称です。`http://jakarta.apache.org` から参照できます。

オンラインヘルプ

オンラインヘルプは、Forte for Java 開発環境内から参照できます。ヘルプキー (Solaris オペレーティング環境では Help キー、Windows および Linux 環境では F1 キー) を押すか、「ヘルプ」 > 「内容」を選択します。ヘルプの項目と検索機能が表示されます。

Javadoc

Javadoc のマニュアルは、Forte for Java の多くのモジュールに用意されており、IDE の中で参照できます。このマニュアルの使用方法については、リリースノートを参照してください。IDE を起動すると、エクスプローラの Javadoc 区画の中でこの Javadoc マニュアルを参照できます。

Sun のマニュアルの注文方法

Sun の製品マニュアルは、Fatbrain.com インターネットブックストアを通じて米国 Sun Microsystems, Inc. に直接注文できます。Fatbrain.com の Sun Documentation Center へは次の URL でアクセスできます。

`http://www.fatbrain.com/documentation/sun`

ご意見の送付先

Sun のマニュアルについてのご意見やご要望をお寄せください。今後のマニュアル作成の参考にさせていただきます。次のアドレスまで電子メールをお送りください。

`docfeedback@sun.com`

電子メールのタイトルに、対象マニュアルの Part No.（このマニュアルの場合は 816-2849-01）を明記してください。

第1章

J2EE Web コンポーネントの基礎

この章は、Web アプリケーションで使用される次の主要な J2EE テクノロジについて概要を示します。

- Web コンテナ
- Web モジュール
- サーブレット
- JSP ページ
- カスタムタグライブラリ

J2EE アーキテクチャについて

J2EE 仕様は、多数のコンポーネントタイプ、およびこれらのコンポーネントに対する実行時環境を含む、広範囲なアーキテクチャを定義します。この仕様では、Web コンテナ、EJB コンテナ、およびアプリケーションクライアントコンテナの 3 つの実行時環境を定義しています。

さらに、コンポーネントタイプは、実行するコンテナに対応するカテゴリに分類されます。つまり、Web コンポーネント、EJB コンポーネント、アプリケーションクライアントコンポーネントです。

この章では、Web コンテナとその Web コンポーネントについてのみ、重要なコンセプトを説明します。また、Web コンポーネントには直接管理されないが、論理的には Web アプリケーションの一部となり、Web コンポーネントとともに配備される、クラスおよびファイルのサポートについても考察します。

Web コンポーネント

Web コンポーネントは、サーバーサイドの J2EE コンポーネントです。それらは、Web コンテナによって管理され、直接 Web コンテナと通信します。Web コンポーネントは、Web コンテナを介して HTTP 要求を受け取り、それらを処理し、HTTP 応答を返すことができます。J2EE プラットフォームは、サーブレットと JSP ページの 2 種類の Web コンポーネントを定義します。

Web コンテナ

Web コンテナは、Web アプリケーションの Web コンポーネントの実行をサポートする実行時サービスを提供します。これらのサービスには以下のものが含まれます。

- ライフサイクル管理
- ネットワークサービス (要求および応答の送信)
- 要求の復号化と応答のフォーマット化
- JSP ページのサーブレットへの変換と処理
- セキュリティ、並行性、トランザクション、配備を提供する J2EE サービスおよび通信 API へのアクセス

Web コンテナは、Web サーバーからのクライアント要求をアプリケーションの Web コンポーネントへ転送し、そのクライアントに対する Web コンポーネントからの応答を Web サーバーへ転送します。Web コンテナは一般的に、Web サーバープロセス (Web サーバープラグインとして) または J2EE アプリケーションサーバープロセスで実行されます。

Web モジュール

Web モジュールは、J2EE アプリケーションでの配備と使用が可能な Web リソースの最小単位です。Web モジュールは、Web アーカイブ (WAR) ファイルとしてパッケージし、配備することができます。WAR ファイルの形式は、JAR ファイルの形式と同一です。ただし、WAR ファイルのコンテンツおよび使用方法は、JAR ファイルのコンテンツおよび使用方法と異なるため、WAR ファイル名は、`.war` 拡張子を使用します。

注 - J2EE Web モジュールは、*Java Servlet Specification version 2.2* で定義されているように 「Web アプリケーション」に対応します。Forte™ for Java™ IDE では、ともに配備される複数の Web モジュールは、**Web モジュールグループ** と呼ばれています。詳細については、50 ページの「Web モジュールグループを作成し実行する」 を参照してください。

Web モジュールの構造

Web モジュールには、次の要素が含まれます。

- オプションで JAR ファイルとしてパッケージされる、サーブレットに対する Java クラスファイル、およびそれらが依存するクラス
- JSP ページおよびそれらのヘルパー Java クラス
- JSP タグライブラリ (通常、JAR ファイルとしてパッケージされている)
- 静的ドキュメント (HTML、イメージ、音声ファイルなど)
- アプレットおよびそれらのクラスファイル

Web モジュールには、必ず次のものが含まれます。

- **Web 配備記述子** (`web.xml` ファイル)

Web モジュールは、リソースを格納するため、階層構造を使用しています。この構造は、ファイルシステムとして、配備時に表すことができます。次の図は、Web モジュールの階層を示します。

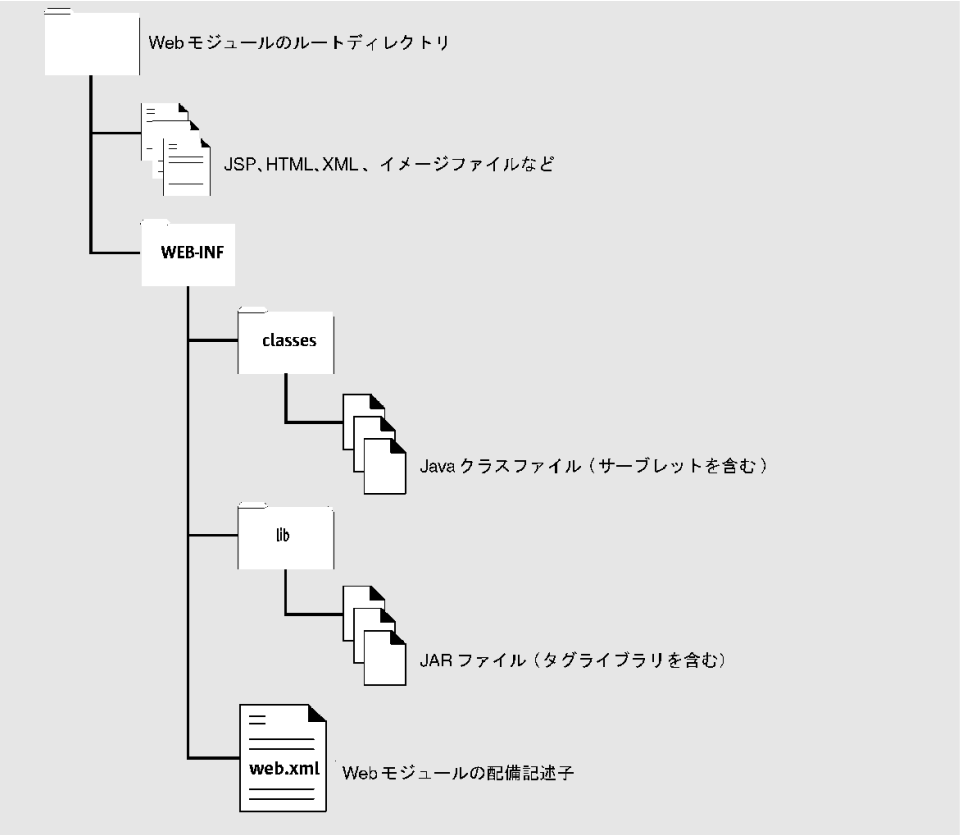

図 1-1　Web モジュール階層構造

Web モジュールの実行時表現

Web モジュールは実行時に、`ServletContext` インタフェースを実装するオブジェクトで表されます。`ServletContext` インスタンスは、Web コンポーネントに対し、Web モジュール内で提供されるリソースへのアクセスを提供します。たとえば、Web コンポーネントは、イベントを記録し、リソースへの URL 参照を入手し、Web モジュールの別の Web コンポーネントが使用することのできる属性を設定し、保存することができます。

`ServletContext` インスタンスは、分散型でない Web モジュール内で一意であり、Web モジュール内のすべての Web コンポーネントで共有されます。このオブジェクトは、`application` インスタンス変数 (この変数は常に提供されます。宣言する必要はありません) として JSP ページで暗黙のうちに利用できます。

ServletContext インスタンス (およびそれが表す Web モジュール) は、Web サーバー内の固有のパスに対応付けられます。たとえば、
http://www.myStore.com/productList に対応付けたとします。この場合、コンテキストパスとして知られている、/productList 要求パスで始まるすべての要求は、この ServletContext インスタンスへ配信されます。

サーブレット

厳密に言うと、**サーブレット**とは、javax.servlet.Servlet を実装する Java クラスのことです。ただし、通常のサーブレットは、
javax.servlet.http.HttpServlet のサブクラスです。

サーブレットは、Web コンテナ内で実行され、Web サーバーおよび Web 対応のアプリケーションサーバーの機能を拡大するために使用されます。サーブレット API により、プログラマはサーブレットコード内で HTTP 要求を使用し、Java オブジェクトとして HTTP 応答を生成することができます。またこれらのオブジェクトを処理する多数の有用なメソッドを提供できます。たとえば、単純なメソッド呼び出しを介して、要求および応答パラメータを検出し、設定することができます。さらに、Java オブジェクトを介して HTTP クッキーを使用し、ユーザーセッションを管理することができます。

サーブレットは、一般に、HTML 形式で生成された要求への応答で動的コンテンツを生成するなどのサービスを提供し、それを実行するため、データソースを使用することがあります。さらに、サーブレットが追跡する状態に応じて、特定の Web リソースへのアクセスを可能、不可能にすることにより、アプリケーションフローを制御するためにも使用されます。別の一般的なサーブレットの使用方法としては、ユーザーセッションの追跡があります。たとえば、ユーザーのショッピングカートへ項目を追加または削除するために使われます。

JSP ページ

JSP ページは、実行前に Web コンテナでサーブレットへ動的に変換される、テキストベースの Web コンポーネントです。

このマニュアルでは、次の用語を使用します。

- **JSP ファイル**; 開発者が作成、編集する JSP テキストベースのソースファイル
- **JSP 実装クラス**; JSP ファイルの変換により、Web コンテナが作成する Java クラス
- **JSP ページ**; 上記 2 つのコンセプトの両方を含む論理用語で、それらを区別することが重要でない、または望ましくない場合に使用されます。

ユーザー側から見ると、JSP ページは、サーブレットクラスと同じです。ロジック主体型ではなく、表現およびドキュメント主体型の方法によって、HTTP 要求を処理し、HTTP 応答を生成する方法を記述するからです。物理的には、JSP ページは、サーブレットを裏返したようなものです。サーブレットソースファイルは一般に、HTML が組み込まれたプログラミングコードであるのに対し、JSP ファイルは一般に、プログラミングコードが組み込まれた HTML です。

JSP ページのライフサイクル

JSP ページは、実行時環境 (Web コンテナ) で処理され、次に HTTP 要求の処理を実行し、HTTP 応答を生成します。このフェーズに含まれる処理は、JSP 変換、JSP インスタンス化、要求処理、JSP 破棄です。

変換

JSP ページの変換とは、Web コンテナが JSP ファイルをサーブレットクラスへ変換するプロセスを指します。このプロセスの細部は、実装によって異なります。リファレンス実装では、JSP ファイルは Java サーブレットソースファイルに変換された後、クラスファイルにコンパイルされます。

Web コンテナは、最初に要求を受け取った時点で JSP ファイルを変換します。同じ JSP ページに対する次の要求では、Web コンテナは、通常このフェーズをバイパスします。ただし、JSP 実装クラスの日付が JSP ファイルの日付より古い場合は、変換が発生する場合があります。

インスタンス化

Web コンテナは特定の JSP ページに対する要求を受け取ると、まず対応する JSP インスタンスを検出しようとします。それが検出されない場合、インスタンス化を行います (この処理の一部として、実装クラスがまだ存在しない場合、JSP ファイルを変換します)。次に、JSP ファイルの `jspInit` メソッドに対応する、インスタンスの `jspInit` メソッドを呼び出します。`jspInit` メソッドを使用すると、JSP ページが要求するリソースを準備することができます。

要求処理

JSP ページは、Web コンテナからクライアント要求を受け取り、プログラムされたロジックに従って要求を処理し、応答をコンテナへ送ります。デフォルトでは、各要求は自身のスレッドで実行されます。

破棄

Web コンテナは、JSP インスタンスを破棄することにより、リソースを再生できます。これを実行する前に、JSP ファイルの `jspDestroy` メソッドに対応する、インスタンスの `jspDestroy` メソッドを呼び出します。`jspDestroy` メソッドを使用すると、必要のないリソースを閉じることができます。

Web コンテナは一般的に、JSP インスタンスが要求を受け取らなくても持続できる時間を制限する方法を備えています。ユーザーが指定した制限を超えると、Web コンテナは、`jspDestroy` メソッドを呼び出します。

JSP ページでのコード構成

JSP ページは、テンプレートデータおよび要素を含むことができます。**要素**は、Web コンテナで認識される構造であり、動的機能を提供します。**テンプレートデータ**は、HTML や XML コードなどの認識されない構造です。これらは、逐時 HTTP 応答へと渡されます。テンプレートデータは一般に、静的コンテンツを提供し、動的データをフォーマットするために使用されます。HTML は、そのまま渡されるため、Web ページ設計者にとって、表現コンテンツのコーディングは、きわめて自然です。

JSP 要素は、指令要素、アクション要素、スクリプト要素の 3 つのカテゴリに分類されます。

指令要素

指令要素は、特定の要求に関係しない JSP ページについてのグローバルな宣言情報を提供します。たとえば、指令を使用して、パッケージをページへインポートします。また指令を使用して、ページを現在の HTTP セッションへ関連付けます。指令は、変換時に処理されます。指令は、出力を HTTP 応答オブジェクトへ書き込みません (HTTP 応答オブジェクトへ書き込まれた出力は、生成された Web ページにテキストとして表示されます)。

指令は、`<%@` と `%>` 記号のあいだに置かれます。たとえば、次の `page` 指令は、`java.util` パッケージをインポートし、JSP ページを現在の HTTP セッションへ関連付けます。

```
<%@ page import="java.util.*" session="true"%>
```

JSP 仕様は、`page`、`include`、および `taglib` 指令を定義しています。

アクション要素

アクション要素は、Java コードを記述することなく、Java オブジェクトを使用する方法を提供する XML 型のタグです。たとえば、アクションを使用して、オブジェクトを検出、インスタンス化し、オブジェクトのプロパティを入手し、設定することができます。アクションは、要求時に処理されます。一部のアクションは、HTTP 応答オブジェクトに出力を書き込みます。

アクションは XML 構文を使用するため、Web ページ設計者に、動的データにアクセスするための使い慣れたパラダイムを提供します。(それ自体がコードアクションでない場合でも、Web ページ設計者は、コードアクションを含むファイルを処理できるよう、アクションを理解する必要があります。したがって、Web ページへ出力を生成するアクションに対し、HTML フォーマット化を提供しなければならない場合もあります)。またアクションは、Java コードと異なり、潜在的にツールにとって分析しやすくなっています。

標準アクションは、JSP 仕様で定義され、Web コンテナで実装されるアクションです。標準アクションは、`forward`、`include`、`useBean`、`getProperty`、`setProperty`、`param`、および `plugin`です。

JSP 仕様はさらに、標準アクションで提供されない機能を提供するため、**カスタムアクション**の開発をサポートします。タグライブラリ記述子 (TLD) と呼ばれる XML ドキュメントでカスタムアクションを定義し、それらを JavaBeans™ コンポーネントで

実装します。TLD および実装する beans は、概念的には、**タグライブラリ**と呼ばれる 1 つのコンポーネントです。詳細については、14 ページの「JSP カスタムタグライブラリ」 を参照してください。

アクションは、< および /> 記号のあいだに置かれます。次の例は、header.jsp と名付けられた JSP ページを現在の JSP ページへ挿入するのに使用される include アクションを示しています。

```
<jsp:include page="header.jsp" flush="true"/>
```

例では、コロンの前の接頭辞 (jsp) によって、これが標準アクションであることが示されています。コロンの後の文字列、この場合 include は、アクション名です。名前と値のペア (page="header.jsp" と flush="true") は、アクションの属性です。

アクションの中には、ボディを持つものもあります。つまり、別のアクション、スクリプト要素、テンプレートデータを囲む、始まりと終わりのタグを持ちます。たとえば、次のコードにおいて、useBean アクションは、application スコープ内の参照 cBean によって取得できるオブジェクトを配置し、さらに cBean と呼ばれるスクリプト変数を介してローカルで取得できるようにします (スコープの詳細については、11 ページの「スコープと暗黙オブジェクト」 を参照してください)。オブジェクトが検出されない場合、指定された Expns.CBean クラスを使用して、アクションがインスタンス化し、ローカルに提供します。アクションのボディに含まれる 2 つのメソッド呼び出し (getConnected と getEngine) は、アクションが Expns.CBean クラスをインスタンス化する場合のみ、呼び出されます。アクションが既に存在するインスタンスを検出すると、2 つのメソッドは呼び出されません。

```
<jsp:useBean id="cBean" scope="application" class="Expns.CBean">
<%
    cBean.getConnected();
    cBean.getEngine();
%>
</jsp:useBean>
```

スクリプト要素

スクリプト要素によって、JSP ファイルに Java コードを組み込むことができます。この要素を使用して、ロジックをプログラミングしたり、HTTP 応答オブジェクトへ出力を書き込むことができます。スクリプト要素には、宣言、スクリプトレット、式、という構文的に異なる 3 つの種類があります。

宣言では、変数の宣言と初期化、オブジェクトのインスタンス化、メソッドの宣言を行います。宣言は、変換時に処理され、HTTP 応答オブジェクトへ出力を書き込みません。宣言は、`<%!` および `%>` 記号のあいだに置かれます。次の例は、2 つの`String` 変数を宣言して、初期化しています。

```
<%!
    String name = null;
    String title = null;
%>
```

スクリプトレットでは、すべての有効な Java コードを入力することができます。宣言要素で宣言された変数とメソッドは、同じ JSP ファイルのスクリプトレットで使用できます。Java 文は、あるスクリプトレットから始まり、別のスクリプトレットで終わることができます (たとえば、HTML コードで分散される場合)。スクリプトレットは、要求時に処理され、HTTP 応答オブジェクトへ出力を書き込みます (書き込みをコード化した場合)。スクリプトレットは、`<%` および `%>` 記号のあいだに置かれます。

次のスクリプトレット例は、2 つのスクリプトレットにわたる Java の if 文で、それらの間に存在する HTML コードのフラグメントを条件付けるために使用されています。`if` 文が `true` に評価された場合のみ、HTML コードは HTTP 応答に含まれます。

```
<% if (name.equals("Elvis Presley")){
%>
<p>Let's hear it for Elvis!
<%  title = "King";
}
%>
```

式要素では、有効で完全なすべての Java 式を入力することができます。Web コンテナは、要求時に、式要素を String へ変換します。次に、String は、HTTP 応答オブジェクトへ書き込まれます。式は、<%= および %> 記号のあいだに置かれます。

次の例は、動的データの一部を HTML 文字列へ挿入します。

```
<p>Hail the <%= title %>!
```

スコープと暗黙オブジェクト

JSP ページでオブジェクトをインスタンス化した場合は、アプリケーションの他のオブジェクトへ提供したいと考えるでしょう。また、アプリケーションのすべてのオブジェクトに提供したり、提供範囲をアプリケーションのオブジェクトのいくつかのサブセットに限定したい場合もあるでしょう。たとえば、現在のユーザーの HTTP セッションに関連したオブジェクトにのみ、提供したい場合です。

オブジェクトの提供範囲を制御するため、JSP 仕様は、オブジェクトへの参照を置くことができるスコープの数を定義しています。スコープには、page、request、session、application があります。実行時に、これらのスコープは、 表 1-1 に示されているように、Java オブジェクトとして実装されます。

表 1-1　JSP ページのスコープ

スコープ	オブジェクトタイプ	説明
page	`javax.servlet.jsp.PageContext`	現在の JSP ページを表します。このオブジェクトは、現在のページまたは `include` 指令に含まれるページ (ただし、`include` アクションに含まれるページではありません。指令はページ変換時に実行され、含められたページは同じ JSP 実装クラスへ連結されるためです) の JSP 要素にのみ、提供されます。
request	`javax.servlet.ServletRequest`	現在の HTTP 要求を表します。このオブジェクトは、現在の HTTP 要求で実行される JSP ページおよびサーブレットにのみ、提供されます。たとえば、ある JSP ページが他のページへ転送されると (`forward` アクションを使用)、両方のページは同じ `ServletRequest` オブジェクトへアクセスします。
session	`javax.servlet.http.HttpSession`	現在のユーザーの HTTP セッションを表します。このオブジェクトは、現在のユーザーの HTTP セッションと関連付けられた要求で実行される JSP ページおよびサーブレットにのみ、提供されます。
application	`javax.servlet.ServletContext`	実行時 Web モジュールを表します。このオブジェクトは、Web モジュールにあるすべての JSP ページおよびサーブレットに提供されます。

useBean アクションを使用して、これらのスコープの 1 つでオブジェクトを検出、または提供することができます。このアクションでは、bean インスタンスの提供範囲を指定するために、scope 属性を使用します。次に例を示します。

```
<jsp:useBean id="myCart" scope="session" class="Cart">
```

スコープ (およびそれらが表すオブジェクト) はさらに、ページが自動的にインスタンス化するスクリプト変数を介して、ページのスクリプト要素へ暗黙的に提供されます。これらのスクリプト変数は、スコープと同じ名前の、page、request、session、および application を使用します。

たとえば、次のスクリプトレットは、暗黙の request 変数を使用して、前例の useBean アクションでインスタンス化した Cart bean を生成します。次に、session 変数を使用して、ページの他のスクリプト要素、または同じユーザーセッションの他のページに提供される session スコープに Cart bean を置きます。session および request 変数はインスタンス化されていません。

```
<%
 CartLineItem lineItem = new CartLineItem();
 lineItem.setID(request.getParameter("cdId"));
 lineItem.setCDTitle(request.getParameter("cdTitle"));
 lineItem.setPrice(request.getParameter("cdPrice"));
 myCart.lineItems.addElement(lineItem);
 session.putValue("myLineItems", myCart.getLineItems());
%>
```

注 - デフォルトでは、JSP ページは、session スコープへアクセスします。ただし、ページの page 指令で、値が false に設定された session 属性を指定すると、そのページは現在の HTTP セッションと関連付けられないため、session スコープを使用できず、session 暗黙変数を参照できません。アプリケーションの一部には、セッションデータ (ユーザーがログインしなくてもよいサイトに関するバックグラウンド情報) を必要としないものがあります。ユーザーがアプリケーションのそのような部分でのみ存続する場合、ユーザーセッションを作成するオーバーヘッドを回避できます。

たとえば、次の page 指令を含むページにおいては、上記の useBean アクションおよびスクリプトレットコード例は、不正となります。

```
<%@ page session="false" %>
```

JSP カスタムタグライブラリ

8 ページの「アクション要素」で説明されているように、JSP 仕様は、独自のカスタムアクションを作成することにより標準のアクションセットを拡張する方法を定義しています。カスタムアクションを作成することにより、アプリケーション内で機能単位のコードをモジュール化、カプセル化することができ、コードを更に再利用可能にします。適切な設計により、書式とロジックを明確に切り離すことができ、結果的に、JSP ページで使用される Java コードの量を削減することができます。

カスタムアクションも、一般に**カスタムタグ**と呼ばれます。ただし、カスタムアクションという語は一般に、JSP ページで使用されるコード構造を示すのに対し、カスタムタグという語は一般に、カスタムアクションの機能を実装するコードを示します。

タグライブラリは、関連するカスタムタグの集まりです。タグライブラリは、ライブラリ中のタグを記述する XML ドキュメントであるタグライブラリ記述子 (TLD)、およびタグライブラリの機能を実装するタグハンドラで構成されます。**タグハンドラ**は、1 つのタグに対する機能を実装する bean です。TLD は、実装するタグハンドラに各タグをマップします。図 1-2 に、タグライブラリのアーキテクチャを示します。

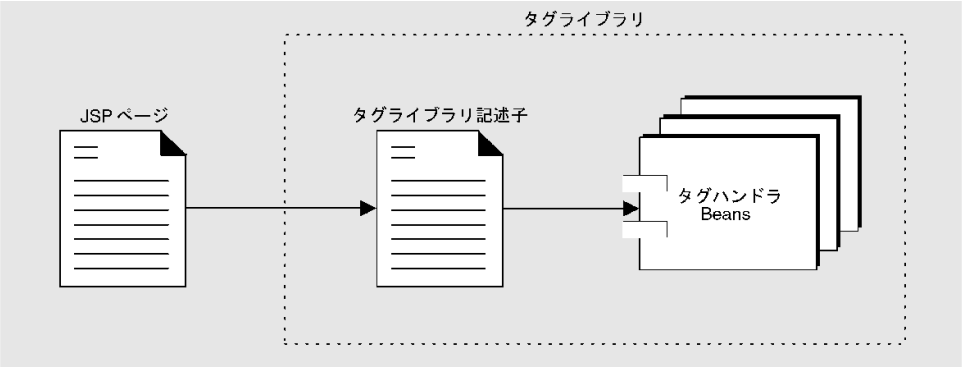

図 1-2　タグライブラリのアーキテクチャ

タグライブラリは、通常 JAR ファイルとしてパッケージされ、ページの `taglib` 指令を介して JSP ページへ提供されます。独自のタグライブラリを開発することも、ベンダーからそれらを入手することもできます (たとえば、ベンダーの Web コンテナの実装の一部として提供される場合)。カスタムアクションとタグライブラリの詳細については、23 ページの「カスタムタグライブラリの開発」 および 付録 B を参照してください。

タグライブラリ記述子

タグライブラリ記述子 (TLD) は、タグライブラリを定義する XML ドキュメントです。Web コンテナは、タグライブラリの TLD を使用することにより、`taglib` 指令を介してそのタグライブラリを参照する JSP ページ上のカスタムアクションを解釈します。TLD の先頭部分で、バージョン番号および意図する Web コンテナのバージョン番号など、全体としてタグライブラリの詳細を定義します。TLD の下の部分で、ライブラリでの各タグを定義します。

Forte for Java IDE の使用により、XML コードを書くことなく、TLD を作成し、編集することができます。IDE が提供するタグライブラリテンプレートから TLD を作成します。TLD を作成した後は、メニューコマンド、TLD およびその要素のカスタマイザウィンドウを介してエクスプローラから TLD を編集することができます。TLD ファイルは決して削除または変更しないでください。たとえば、エクスプローラで TLD を選択し、コンテキストメニューから「タグを追加」を選択することにより、TLD でタグを定義することができます。次に、タグを選択し、コンテキストメニューから「属性を追加」を選択することにより、そのタグの属性を定義できます。これらのアクションは、デフォルト値を持つタグおよび属性要素を作成します。このような要素を作成すると、適切なカスタマイザダイアログで、それらを編集することができます。TLD の作成と編集の詳細については、25 ページの「カスタムタグとタグハンドラの開発」 を参照してください。

タグハンドラ

タグハンドラは、カスタムアクションの機能を実装する bean です。カスタムアクションとタグハンドラは 1 対 1 で対応しています。このトピックの詳細については、32 ページの「タグライブラリによるカスタムアクションの挿入」 を参照してください。

Forte for Java の組み込みタグライブラリ

タグライブラリの開発のサポートに加え、Forte for Java IDE を使用すると、サードパーティ製のタグライブラリをインポートしたり、自分の JSP ページからタグライブラリを表示および編集できます。Forte for Java は、いくつかの組み込みタグライブラリを提供しています。これらの組み込みタグライブラリを使用すると、次の利点があります。

- JDBC または Transparent Persistence (TP) を使用して、データソースにアクセスし、操作を実行できます。
- 次の項目の内部で行とフィールドを反復できます。
  - JDBC の `ResultSet`
  - `Vector`、`Collection`、`List`、`Iterator`、または `Enumeration` のオブジェクトとそれらのフィールド
  - Java 配列の要素 (要素がオブジェクトである場合はそれらのフィールド)
- JSP ページの一部を条件付けることができます (if/else ロジックを使用)。

組み込みタグライブラリの詳細については、JSP およびサーブレットモジュールのオンラインヘルプを参照してください。

クラス、bean とその他のファイルのサポート

Web コンポーネントは一般に、サポートする機能を提供するため、追加のクラス、beans、HTML ファイル、その他のファイルを必要とします。たとえば、サーブレットは、サポートする bean に対し、画面フロー管理やセッション制御などの複雑なタスクを委託することができます。さらにサーブレットは、bean を使用して、EJB やデータベースなどの遠隔リソースにアクセスしたり、それらのリソースへの呼び出しによって戻された結果をキャッシュすることができます。さらに、JSP ページと HTML ファイルは、多くの場合イメージファイル、音声ファイル、およびビデオファイルを参照します。

第2章

Web アプリケーションのプログラム

J2EE Web アプリケーションは 1 つまたは複数の Web モジュールで構成されます。この章は、Forte for Java IDE を使用して Web モジュールをプログラムする方法について概説します。全体的な観点から、アプリケーションの作成段階で実行するタスクを順序付けし、個々のプログラミングタスクの詳細について考察します。

Web アプリケーションの構成、実行、デバッグ、および配備の説明については、第 3 章を参照してください。

Web モジュールの開発プロセスに関するあらゆる面の詳細については、『CDShopCart チュートリアル』を参照してください。

Web モジュールプログラミングのワークフロー

この節では、Forte for Java IDE を使用する Web モジュールのプログラミングに関するワークフローの概要を説明します。この概要では、コーディングやテスト方法を反復して説明しません。ここでは、主な開発タスクをリストし、これらのタスクを実行する論理的な順序を示します。各タスクは、そのタスクについての詳しい情報を提供するこの章の後ろの節への参照です。さらに JSP サーブレットモジュールのオンラインヘルプでもこれらのタスクについて情報を提供します。

Forte for Java IDE を使用して Web モジュールをプログラムする手順は次のとおりです。

1. Web モジュールを作成します (18 ページの「Web モジュールの作成」)。
2. 使用する予定の Web コンポーネントを作成します。このプロセスには次の作業が含まれる場合もあります。

- Web モジュールに必要とされる JSP ページの作成（ 22 ページの「JSP ページの作成」を参照)。
- Web モジュールに必要とされるサーブレット、クラス、beans の作成またはインポート

サーブレット、クラス、beans が JAR ファイルとしてパッケージされていない場合は、Web モジュールの `/WEB-INF/classes` (`/WEB-INF/`クラス) ディレクトリに置きます。JAR でパッケージされている場合は、`/WEB-INF/lib` ディレクトリに置きます (23 ページの「サーブレット、クラス、Beans の作成」 を参照)。

TP を使用して開発したクラスは、この規則の例外です。Web モジュールが必要とするあらゆる**持続可能クラス**は、Web モジュールの外で開発する必要があります。**持続可能クラス**を JAR ファイルとしてパッケージ化して、パッケージを Web モジュールの `/WEB-INF/lib` ディレクトリに置きます。これで、アプリケーションをテストしたり、WAR ファイルとしてパッケージ化することができます。

TP についての詳細は、Forte for Java プログラミングシリーズの 『持続プログラミング』を参照してください。Web モジュールのパッケージ化についての詳細は、67 ページの「Web モジュールのパッケージと配備」を参照してください。

- JSP ページが必要とするあらゆるタグライブラリの作成または追加

通常、その目的に特化した別の Web モジュールおよびファイルシステム内で、タグライブラリを開発します。次にそれを JAR ファイルとしてパッケージ化し、依存する JSP ページを含む Web モジュールの `lib` ディレクトリに置きます (23 ページの「カスタムタグライブラリの開発」を参照)。

Web モジュールの作成

Web モジュールは、J2EE **配備**構造になっています。Forte for Java IDE を使用して Webアプリケーションを開発すると、IDE が必要な Web モジュール構造を作成します。Web モジュール構造を強制することで、IDE は Web モジュールを確実に WAR ファイルとしてパッケージ化できるようにし、しかも大多数のサーバの配備に必要な配備記述子 (`web.xml` ファイル) が Web モジュールに必ず含まれるようにします。また、*Java Servlet Specification, v2.2* および *JavaServer Pages Specification, v1.1* では、JSP ページは Web モジュール内で実行されるよう要求しています。

Forte for Java IDE での Web モジュール

Forte for Java IDE では、Web モジュールは、WAR ファイルの構造 (この構造の詳細については、3 ページの「Web モジュールの構造」を参照) に適合するマウント済みのファイルシステムとしてエクスプローラウィンドウに表示されます。

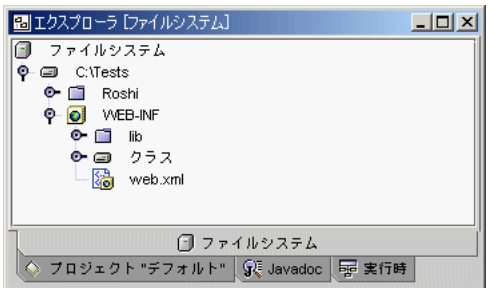

図 2-1　エクスプローラにマウントされた Web モジュール

他のファイルシステムにマウントするのと同様に、Web モジュールをエクスプローラにマウントします (ファイルシステムのマウントの詳細については、オンラインヘルプを参照)。ただし、Web モジュールそのものをマウントする必要があります。Web モジュールそのものでなく、Web モジュールを含むディレクトリをマウントすると (つまり、マウントされたファイルシステムのサブディレクトリとなるように Web モジュールをマウントすると)、IDE は、Web モジュールを正しく認識しません。この状態では、Web モジュールと通常関連付けられたいくつかの操作を実行できません。

Web モジュールは、プログラミング上では、オブジェクトタイプでないにもかかわらず、エクスプローラではオブジェクトタイプとして処理されます。これは、たとえば、Web モジュールのオブジェクトタイプは、そのプロパティウィンドウで設定できる属性、および Web モジュールのコンテキストメニューで利用できる関連コマンドのセットを備えています。また、エクスプローラのその他のオブジェクトタイプと同様、テンプレートから Web モジュールを作成することができます。

Web モジュールは、次の 2 つの方法のいずれかで作成することができます。

- 新規のディレクトリとして Web モジュールを作成する
- 既存のディレクトリを Web モジュールに変換する

注 - Web モジュールの構造に適合する既存のディレクトリ構造がある場合は、それを変換することなく、Web モジュールとしてエクスプローラでマウントし、使用することができます。IDE は、そのようなディレクトリを、その構造によって Web モジュールとして認識します。

▼ 新規のディレクトリを Web モジュールとして作成する

1. メインメニューバーから、「ファイル」>「新規」を選択します。

   テンプレートウィザードから新規作成が開きます。

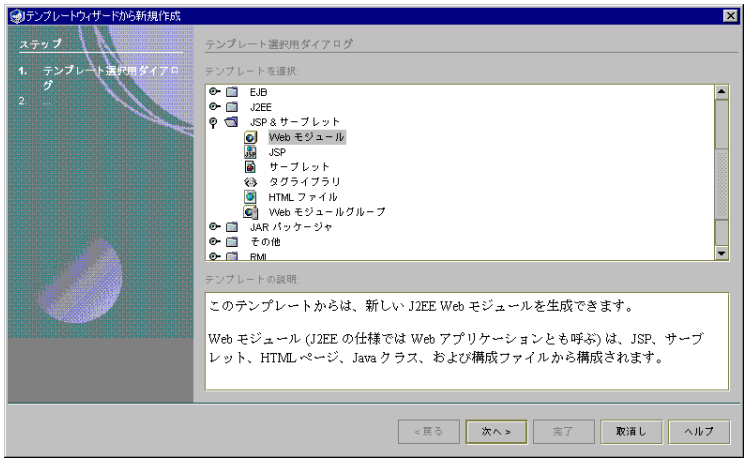

   図 2-2　テンプレートウィザードから新規作成

2. 「JSP & サーブレット」テンプレートカテゴリから、「Web モジュール」テンプレートを選択し、「次へ」をクリックします。

   ドキュメントベース区画が表示されます。

3. 「…」ボタンをクリックします。

   ファイル選択用の「Web モジュールディレクトリ」ダイアログが開きます。

4. 新規のディレクトリを作成する場所に移動し、「フォルダの新規作成」アイコンをクリックします。

   新規フォルダディレクトリは、選択した場所に物理的に作成されるので注意してください。開発ディレクトリが選択した場所でない限り、新規フォルダディレクトリは開発ディレクトリの下に作成されません。

5. 新規フォルダが作成されます (「新しいフォルダ」または「名称未設定フォルダ」と名付けられていますが、それは選択されていません。スクロールして見つける必要があります)。

6. 新規フォルダディレクトリを選択し、新しいフォルダのタイトルをクリックして、編集可能にします。

7. フォルダの名前を入力し、Enter キーを押します。

   入力する名前にはスペースを入れてはなりません。

8. 「ファイル名」フィールドにフォルダの新しい名前が表示されていることを確認します (新しい名前になっていない場合は、別のフォルダを選択し、新規フォルダを選択し直してください)。

9. フォルダを選択し直してから (まだ選択していない場合)、「追加」をクリックします。

   ドキュメントベース区画にフォーカスが戻ります。

10. ドキュメントベース区画の「ディレクトリ」フィールドに選択したディレクトリが表示されていることを確認したら、「完了」をクリックします。

    Web モジュールは、エクスプローラ内のファイルシステム区画またはプロジェクト区画で作成され、マウントされます。

▼ 既存のディレクトリを Web モジュールに変換する

1. メインメニューバーから、「ファイル」>「ファイルシステムをマウント」を選択して、変換するファイルシステムのルートディレクトリをマウントします。

2. エクスプローラで、新たにマウントされたファイルシステムを選択します。

3. メインメニューバーから、「ツール」>「ファイルシステムを Web モジュールに変換」を選択します。

   ディレクトリは Web モジュールに変換されます。ディレクトリは移動していませんが、IDE が Web モジュールのディレクトリとして認識するようになります。`WEB-INF/lib` ディレクトリに `.jar` ファイルを含んでいる Web モジュールがマウントされると、`lib` ディレクトリ内の一部の `.jar` ファイルもマウントされます。既存のディレクトリを Web モジュールに変換する詳細については、オンラインヘルプを参照してください。

注 - WEB-INF/lib ディレクトリ内の .jar ファイルの横に対応する jarContent ファイルがある場合、IDE は .jar ファイルのソースクラスがマウントされていて、それ以外に .jar はマウントされていないとみなします。

JSP ページの作成

JSP ページは、次の 2 つの方法のいずれかで作成することができます。

- テンプレート選択ダイアログを使用する
- Dreamweaver テンプレートから生成する。このトピックの詳細については、付録 A を参照してください。

▼ JSP ページを作成する

1. エクスプローラで、JSP ページを作成するファイルシステムのルートディレクトリを選択します。

   ほとんどの場合、Web モジュールのルートディレクトリまたはルートディレクトリ内に作成したサブディレクトリに JSP ページを作成する必要があります。JSP ページを WEB-INF ディレクトリ (またはそのサブディレクトリのいずれか) に置くと、クライアントのブラウザから直接アクセスできなくなります。ただし、jsp:include または jsp:forward のようなサーブレットからのリソースとしてであればアクセスできます。この機能は、チェックアウト手続きの途中のページなど、特定の順序またはセキュリティの制約に従ってアクセスする必要のある JSP ページへのアクセスを制御するために、フロントコントローラアーキテクチャで頻繁に使用されます。

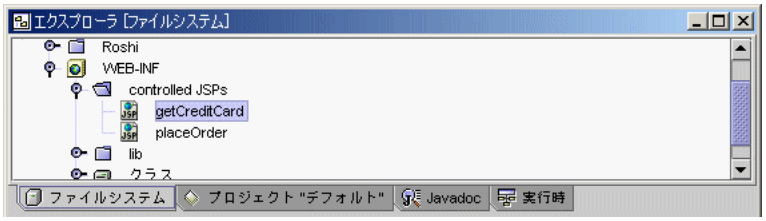

図 2-3　Web モジュール構造内の JSP の正しい位置

2. ディレクトリのコンテキストメニューから、「新規」 > 「JSP & サーブレット」 > 「JSP」を選択します。
3. ウィザードの「名前」フィールドで、JSP ページの名前を入力し、「完了」をクリックします。

   JSP ページが作成され、ソースエディタに表示されます。

サーブレット、クラス、Beans の作成

他のオブジェクトタイプ同様、IDE は、サーブレット、クラス、beans を作成するためのテンプレートを提供します。この節では、サーブレットの作成方法について説明します。クラスと beans の作成方法の詳細については、オンラインヘルプを参照してください。Web モジュールの `WEB-INF/`クラスディレクトリでサーブレット、クラス、beans を作成する必要があります。Web モジュールが エクスプローラでマウントされると、このディレクトリは、IDE の内部クラスパスに入れられます。

▼ サーブレットを作成する

1. エクスプローラで、`WEB-INF/classes` (`WEB-INF/`クラス) ディレクトリを選択します。
2. コンテキストメニューから、「新規」>「JSP & サーブレット」>「サーブレット」を選択します。
3. ウィザードの「名前」フィールドで、サーブレットの名前を入力し、「完了」をクリックします。

   サーブレットが作成され、ソースエディタに表示されます。

カスタムタグライブラリの開発

Web モジュールを作成して、必要な JSP ページ、サーブレット、クラス、および beansを追加したら、JSP ページが依存しているカスタムタグライブラリを開発できます。必要なタグライブラリがすでに存在していて、Web モジュールに追加するだけになっている場合もあります。タグライブラリを Web モジュールに追加する手順の詳細

については、38 ページの「カスタムタグライブラリのパッケージと配備」を参照してください。タグライブラリの例を作成して使用する方法については、付録 B を参照してください。

▼ カスタムタグライブラリを作成する

1. エクスプローラで、タグライブラリを作成する Web モジュールのルートディレクトリを選択します。

   新規または既存の Web モジュールまたはファイルシステムにタグライブラリを作成できます。

2. Web モジュールまたはファイルシステムのルートディレクトリを右クリックします。
3. コンテキストメニューから、「新規」>「JSP & サーブレット」>「タグライブラリ」を選択します。
4. ウィザードの「名前」フィールドで、タグライブラリの名前を入力し、「完了」をクリックします。

   この操作によって、TLD ファイルが作成されます。カスタムタグライブラリのプロパティを定義する「タグライブラリカスタマイザ」ダイアログで、確認してください。

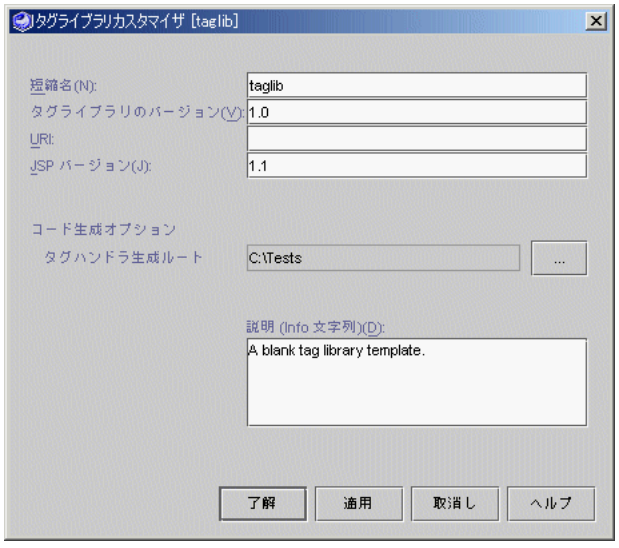

図 2-4　タグライブラリカスタマイザ

タグライブラリの名前、バージョン、URI を指定するだけでなく、コード生成オプションを設定し、タグライブラリの機能に関する説明を入力できます。タグライブラリカスタマイザのプロパティの詳細については、オンラインヘルプを参照してください。

▼ タグライブラリをカスタマイズする

1. エクスプローラでタグライブラリカスタマイザをまだ開いていない場合は、カスタマイズするタグライブラリのタグライブラリ記述子を右クリックします。

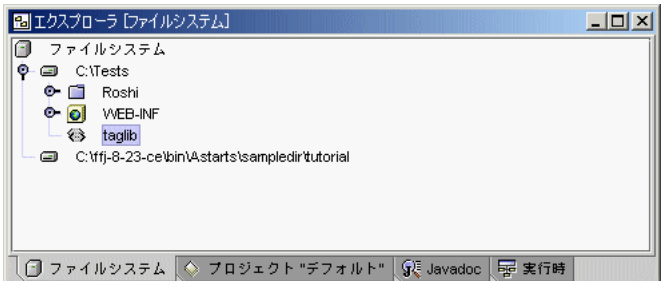

図 2-5　エクスプローラの TLD

2. コンテキストメニューから「カスタマイズ」を選択します。

   もう 1 つの選択方法として、カスタマイズするタグライブラリをダブルクリックします。タグライブラリカスタマイザの内容に、選択したタグライブラリのプロパティが反映されています。タグライブラリカスタマイザが表示されると、タグライブラリカスタマイザでの選択が追跡され、選択したタグライブラリのプロパティを表示します。

3. タグライブラリカスタマイザで、タグライブラリを編集します。

4. 「了解」をクリックし、変更内容が有効になるようにタグライブラリカスタマイザを終了するか、「適用」をクリックし、タグライブラリカスタマイザを終了しないで変更内容を適用してください。

   タグライブラリのプロパティを指定すると、タグを追加したりカスタマイズできます。

カスタムタグとタグハンドラの開発

この節では、カスタムタグを開発し、タグハンドラを生成する方法について説明します。

▼ タグを追加してカスタマイズする

カスタムタグは、Java コードの本体であるタグシグニチャーとタグハンドラで構成されます。IDE は「タグカスタマイザ」ダイアログで指定した仕様に基づいてタグハンドラのスケルトンコードを生成します。コードが生成されたら、タグハンドラコードを直接編集し、タグの機能を実装するロジックを挿入します。

1. エクスプローラで、タグを追加するタグライブラリ記述子を右クリックします。
2. コンテキストメニューから「タグを追加」を選択します。

   「新しいタグを追加」ダイアログが表示されます。

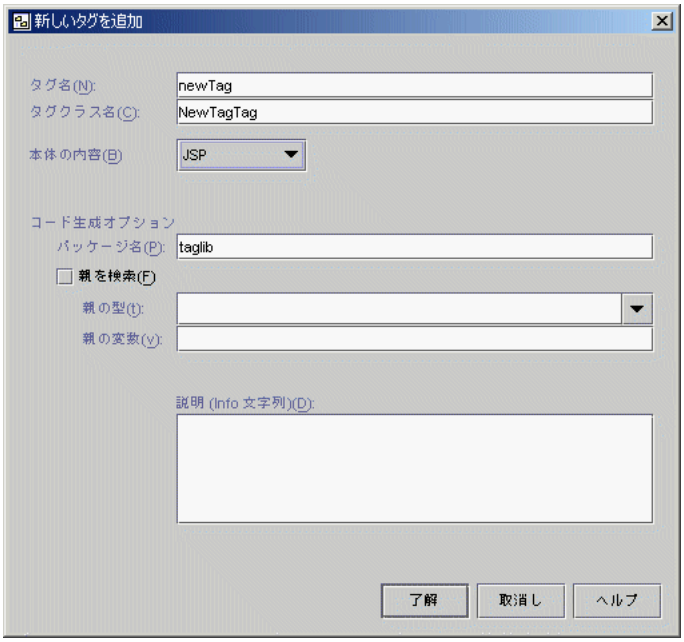

図 2-6　「新しいタグを追加」ダイアログ

3. 「タグ名」テキストフィールドで、新しいタグに一意のタグ要素名を入力します。

   タグ要素名は、Java 言語のネーミング標準規格に従う必要があります。たとえば、名前を数字で始めることはできません。スペースを入れたり、!、#、または + などの特殊文字を入れることもできません。複数バイトのタグ名は使用できますが、複数のタグ名の入っているタグライブラリは、一部の Web サーバーではうまく実行できない場合があります。

4. 「タグクラス名」フィールドに、新規のタグのクラス名を入力します。

   タグクラス名は、Java 言語のネーミング標準規格に従う必要があり、しかも Java クラス名が有効でなければなりません。

5. 「本体の内容」コンボボックスから、タグの本体に表示されるコンテンツのタイプを選択します。

   指定できるオプションについては、34 ページの「カスタムアクションの本体の処理方法を指定する」を参照してください。

6. 生成されたタグハンドラクラスの Java パッケージの名前を指定します。

   デフォルト値はタグライブラリの短縮名です。

7. 生成されたタグハンドラに、親 (つまり、指定した) タグを検索するためのコードが入っていることを示すには、「親を検索」チェックボックスをクリックします。

   タイプは「親の型」プロパティによって決定されます。親のタグハンドラのインスタンス (検索された場合) は、「親の変数」プロパティで指定された変数に入れられます。「親を検索」をオンにすると、「親の型」と「親の変数」の各プロパティが有効になります。デフォルト値はオフになります。「親を検索」をオンにした場合は、必ず「親の型」と「親の変数」の値を入力してください。

8. 親タグのクラスタイプを指定するには、クラス名を入力するか、「親の型」コンボボックスからどれかを選択してください。

9. 「親の変数」テキストフィールドに、親に使用される変数名を入力します。

   変数名は、必ず有効な Java 言語の変数名にしてください。

10. 「説明 (Info 文字列)」テキストボックスに、タグに関する説明を入力します。

11. 「了解」をクリックして、変更内容が有効になるようにし、「新しいタグを追加」ダイアログを終了します。

    新規のタグが TLD に追加され、「タグカスタマイザ」ダイアログが表示されます。タグカスタマイザを編集して「了解」をクリックすると、変更内容が有効になり、タグカスタマイザを終了できます。

▼ 既存のタグをカスタマイズする手順

1. エクスプローラで、カスタマイズするタグ要素を右クリックします。

2. コンテキストメニューから「カスタマイズ」を選択します。

3. 「タグカスタマイザ」で、タグを編集します。

   タグカスタマイザのプロパティについては、26 ページの「タグを追加してカスタマイズする」とオンラインヘルプで説明しています。

4. 「了解」をクリックし、変更内容が有効になるようにタグカスタマイザを終了するか、「適用」をクリックし、タグカスタマイザを終了しないで変更内容を適用してください。

▼ タグ属性を追加してカスタマイズする

「新しいタグ属性を追加」ダイアログと「タグ属性カスタマイザ」ダイアログを使用して、既存のタグ属性を編集します。

1. エクスプローラで、属性を追加するタグ要素を右クリックします。
2. コンテキストメニューから「タグ属性を追加」を選択します。
3. 「新しいタグ属性を追加」ダイアログで、属性を指定します。

図 2-7　「新しいタグ属性を追加」ダイアログ

「新しいタグ属性を追加」ダイアログで、新しいタグ属性の各種プロパティを指定できます。新しいタグ属性 (とタグ属性カスタマイザ) のプロパティの詳細については、オンラインヘルプを参照してください。

4. 「了解」をクリックします。

   新しいタグ属性がタグに追加され、タグ属性カスタマイザが表示されます。タグ属性カスタマイザを編集して「了解」をクリックすると、変更内容が有効になり、タグ属性カスタマイザを終了できます。

▼ 既存のタグの属性をカスタマイズする

1. エクスプローラで、属性を右クリックし、コンテキストメニューから「カスタマイズ」を選択します。
2. タグ属性カスタマイザで、属性のプロパティを編集します。

   タグ属性カスタマイザで、タグ属性の各種プロパティを指定できます。タグ属性カスタマイザのプロパティの詳細については、オンラインヘルプを参照してください。

3. 「了解」をクリックし、変更内容が有効になるようにタグ属性カスタマイザを終了するか、「適用」をクリックし、タグ属性カスタマイザを終了しないで変更内容を適用してください。

   タグ属性カスタマイザの内容に、選択した属性のプロパティが反映されています。タグ属性カスタマイザが表示されると、タグ属性カスタマイザでの選択が追跡され、選択したタグライブラリのプロパティを表示します。

4. 選択した場合は、タグハンドラを生成します。

   手順については、36 ページの「タグハンドラを生成する」を参照してください。

▼ スクリプティング変数を追加してカスタマイズする

スクリプティング変数は、タグが JSP ページにエクスポートする値のことです。この値は、スクリプトレットまたは式で使用することができます。詳細については、付録 B を参照してください。

新しいスクリプティング変数を作成するには「新しいタグスクリプティング変数の追加」ダイアログを使用し、スクリプティング変数のプロパティを編集するには「タグスクリプティング変数カスタマイザ」ダイアログを使用してください

1. エクスプローラで、スクリプティング変数を追加するタグ要素を右クリックします。

2. コンテキストメニューから「Scripting 変数を追加」を選択します。

3. 「新しいタグスクリプティング変数の追加」ダイアログで、スクリプティング変数のプロパティを指定します。

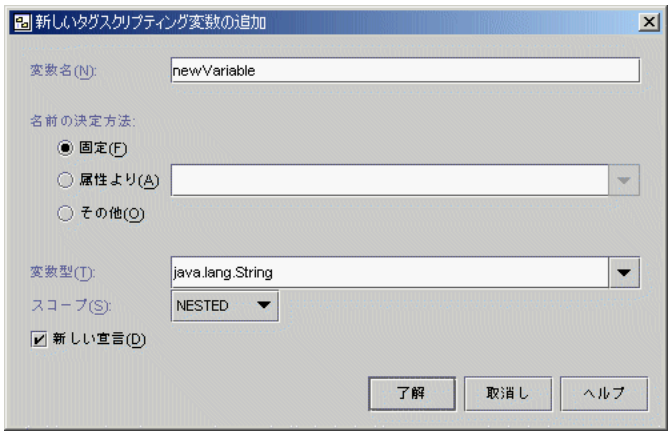

図 2-8　「新しいタグスクリプティング変数の追加」ダイアログ

「新しいタグスクリプティング変数の追加」ダイアログでは、新しいスクリプティング変数に各種プロパティを指定できます。「新しいタグスクリプティング変数の追加」ダイアログ (および「タグスクリプティング変数カスタマイザ」ダイアログ) のプロパティの詳細については、オンラインヘルプを参照してください。

4. 「了解」をクリックし、変更内容が有効になるように「新しいタグスクリプティング変数の追加」ダイアログを終了します。

新しいスクリプティング変数が追加され、タグスクリプティング変数カスタマイザが表示されます。タグスクリプティング変数カスタマイザを編集して「了解」をクリックすると、変更内容が有効になり、タグスクリプティング変数カスタマイザを終了できます。

▼ 既存のタグスクリプティング変数をカスタマイズする

1. エクスプローラで、スクリプティング変数を右クリックして、コンテキストメニューから「カスタマイズ」を選択します。
2. タグスクリプティング変数カスタマイザで、スクリプティング変数のプロパティを編集します。

   タグスクリプティング変数カスタマイザで、スクリプティング変数の各種プロパティを指定できます。タグスクリプティング変数カスタマイザのプロパティの詳細については、オンラインヘルプを参照してください。
3. 「了解」をクリックして、変更内容が有効になるようにタグスクリプティング変数カスタマイザを終了するか、「適用」をクリックし、タグスクリプティング変数カスタマイザを終了しないで変更内容を適用してください。

   タグスクリプティング変数カスタマイザの内容に、選択したスクリプティング変数のプロパティが反映されています。タグスクリプティング変数カスタマイザが表示されると、タグスクリプティング変数カスタマイザでの選択が追跡され、選択したスクリプティング変数のプロパティを表示します。
4. 選択した場合は、次の節の説明にあるように、タグハンドラを生成します。

タグハンドラの生成

タグライブラリを開発する際は、カスタムアクションで必要な機能を実装するためにコードをタグハンドラのクラスに追加します。新しい属性とスクリプティング変数を定義する際は、対応するクラスメンバーとインタフェースが作成されるようにタグハンドラを生成する必要があります。

タグハンドラを生成する場合は次の 2 つのオプションがあります。

- タグハンドラの生成。前回ハンドラを生成した後に変更したハンドラのみ生成することができます。これによって、最新の作業を簡単にチェックすることができます。前回生成した後に変更したハンドラのタグの名前には、図 2-9 にあるとおり、エクスプローラで (G) が付きます。
- すべてのタグハンドラの生成。必要かどうかにかかわらず、すべてのタグハンドラが生成されるようにすることができます。これによって、タグライブラリ内のすべてのハンドラのバージョンが更新されるため、ライブラリに入っているタグの数によっては時間がかかる場合もあります。

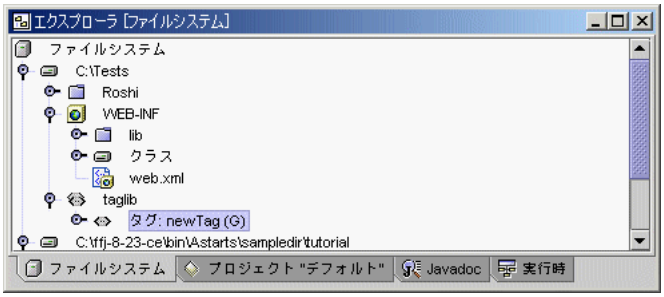

図 2-9　前回生成した後にハンドラが変更されたタグ

タグライブラリによるカスタムアクションの挿入

JSP ページのカスタムアクションをコード化することで、タグライブラリの機能を使用します。カスタムアクションでタグライブラリにアクセスするために、JSP ページは、`taglib` 指令でタグライブラリを宣言する必要があります。

次に例を示します。

```
<%@taglib uri="/WEB-INF/lib/myTagLib.jar" prefix="mt" %>
```

`taglib` 指令の `uri` 属性は、タグライブラリ記述子 (TLD)、または上記の例で示した、TLD とタグハンドラ beans の両方を含む JAR ファイルのいずれかを参照します。`taglib` 指令は、タグライブラリを使用するすべてのカスタムアクションの前に置く必要があります。

上記の例の `uri` 属性は、ハードコード化された相対パスを Web モジュールのルートに指定します (先頭のスラッシュは、Web モジュールルートを示します)。ただし、アプリケーションの配布後に構成することができる、より抽象的な方法でこの属性を指定することもできます。これを行なうためには、Web モジュール配備記述子 (`web.xml` ファイル) で `taglib` 要素を作成する必要があります。作成した後に、URI を TLD またはタグライブラリ JAR ファイルの物理位置へマップするようにこの `taglib` 要素を構成します。

たとえば、次の taglib 要素は、/WEB-INF/tlds/myTagLib.tld に置かれた TLD を URI myTags を介してアクセスできるようにします。

```
<taglib>
    <taglib-uri>myTags</taglib-uri>
    <taglib-location>/WEB-INF/tlds/myTagLib.tld</taglib-location>
</taglib>
```

IDE がこのマッピング手順をどのように容易にしているかという例については、23 ページの「カスタムタグライブラリの開発」を参照してください。

上記のマッピングを宣言すれば、JSP ページに次の taglib 指令を置くことによって、タグライブラリが JSP ページにアクセスできるようになります。

```
<%@taglib uri="myTags" prefix="mt" %>
```

上記の例では、タグライブラリを使用する最初のカスタムアクションの前に、taglib 指令を置く必要があります。

taglib 指令がタグライブラリ JAR ファイルではなく TLD ファイルを参照する場合 (タブライブラリの開発中はその可能性があります)、TLD がタグハンドラのクラス名を指定し、タグハンドラが自分の クラスパスにあることを確認する必要があります。IDE では、タグハンドラを生成すると、これら両方のタスクが実行されます。

JSP ページでコード化されたカスタムアクションからタグライブラリを参照するための識別子を指定するには、taglib 指令の prefix 属性を使用します。たとえば、次のカスタムアクション (前の taglib 指令と同じ JSP ページにあると仮定) は、接頭辞 mt を使用して、タグライブラリを参照します。文字列 table で、このカスタムタグを処理するタグハンドラを指定します。

```
<mt:table results="productDS"/>
```

タグ名 (この場合、table) とタグハンドラ bean 間のマッピングは、TLD ファイルで指定されます。このマッピングは、エクスプローラでタグのコンテキストメニューからアクセスできる「タグカスタマイザ」ダイアログで編集することができます。

カスタムアクションは、オブジェクトを作成し、スクリプティング変数としてそれらを JSP ページで提供することができます。スクリプティング変数は、JSP ページの他のアクションまたはスクリプティング要素で使用することができます。

タグハンドラとカスタムアクション

カスタムアクションとタグハンドラは、1 対 1 で対応しています。

カスタムアクションの本体

カスタムアクションは、原則として、本体を持つことができます。つまり、他のアクション、スクリプティング要素、プレーンテキストを囲む始まりと終わりのタグを持つことができます。

たとえば、下記の例にあるカスタムアクションでは、本体がプレーンテキストで構成されています。

```
<mt:convertToTable>
type distance / a 30,000 / g 5,500 / z 200
</mt:convertToTable>
```

特定のカスタムアクションに本体があるかどうかは、TLD での定義方法によって異なります。

▼ カスタムアクションの本体の処理方法を指定する

「タグカスタマイザ」ダイアログの「本体の内容」では、本体の処理方法を指定することができます (このダイアログは、カスタムアクションのタグハンドラのコンテキストメニューから表示できます)。図 2-10 に示すように、`JSP`、`empty`、または `tagdependent` の 1 つを選択することができます。

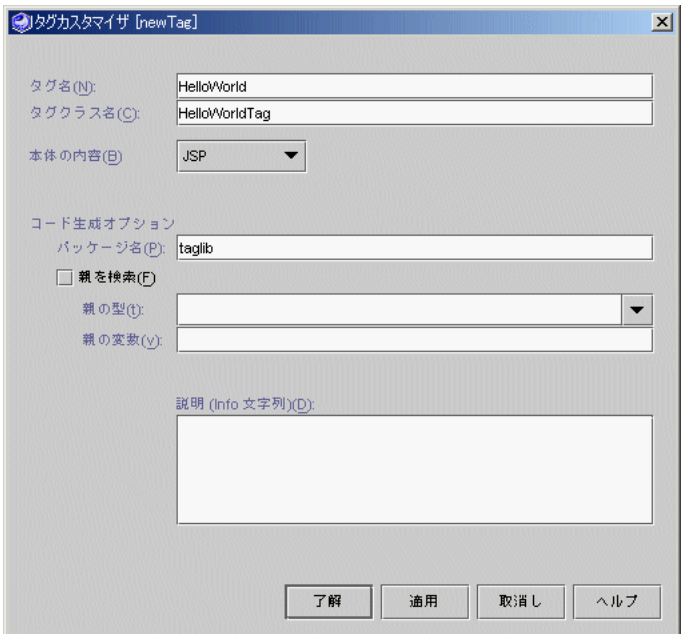

図 2-10 「タグカスタマイザ」ダイアログ

表 2-1 では、各選択肢の意味を説明しています。

表 2-1 「タグカスタマイザ」ダイアログの「本体の内容」選択肢の意味

本体の内容 選択肢	意味
JSP	本体の内容は任意です。Web コンテナは JSP 要素を評価した後、本体をタグハンドラへパスします。タグハンドラは、本体を処理し、プログラミングロジックに従って `out` オブジェクトへ出力を書き込みます。
empty	本体の内容は許可されません。
tagdependent	本体の内容は任意です。Web コンテナは JSP 要素を評価しませんが、本体をタグハンドラへパスします。タグハンドラは、本体を処理し、プログラミングロジックに従って `out` オブジェクトへ出力を書き込みます。

すべてのタグハンドラは、javax.servlet.jsp.tagext.Tag を実装します。本体を受け入れない、または処理しないタグハンドラは、このインタフェースだけを実装する必要があります。本体を処理するタグハンドラも、javax.servlet.jsp.tagext.BodyTag を実装する必要があります。このインタフェースは、この処理を行なうための追加メソッドを提供します。

▼ タグハンドラを生成する

1. エクスプローラで、生成するハンドラが含まれるタグライブラリ記述子を右クリックします。
2. コンテキストメニューから、「タグハンドラを生成」または「すべてのタグハンドラを生成」のいずれか適切な方を選択してください。

   生成されたコードがタグライブラリのタグハンドラ生成ルートの、タグカスタマイザで設定したコード生成オプションのパッケージ名を持つディレクトリに表示されます。パッケージ名がブランクの場合は、タグライブラリのタグハンドラ生成ルートに表示されます。

生成されるタグハンドラ

前の節で説明しているとおり、タグハンドラは、TLD からタグハンドラを生成します。これらの生成されたタグハンドラは、TLD で定義された、対応するカスタムアクションに適切なインタフェースを実装します (Tag インタフェース、または Tag と BodyTag インタフェースの両方)。さらに、タグハンドラの要求とするクラスメンバー (フィールド、メソッド、プロパティ) はすべて生成されます。クラスメンバーの正確なリストは、各自の TLD によって異なります。ただし、タグハンドラが実装するインタフェースで要求されたメソッドは常に含まれます。

生成された固有のクラスメンバーは、タグハンドラが実装するインタフェースに依存し、さらに TLD で宣言した属性およびスクリプティング変数に依存します。たとえば、myAttribute と名付けられた属性を宣言すると、myAttribute と名付けられたプロパティがタグハンドラで生成されます。

生成されるメソッド

表 2-2 は、タグハンドラを生成するときに IDE が作成するメソッドを示しています。プロパティを取得し、設定するメソッドについては示されていません。一部のメソッドにはアスタリスク (*) が付いていますが、これは Tag インタフェースや BodyTag

インタフェースの一部であることを示しています。これらのメソッドは、テンプレートデザインパターンに従って定義されたヘルパーメソッドであるその他のメソッドを呼び出します。

クラスは、Tag インタフェースや BodyTag インタフェースすべてのメソッドを実装できるそれぞれの TagSupport ヘルパークラスや BodyTagSupport ヘルパークラスを拡張して生成されるため、Tag インタフェースや BodyTag インタフェースのすべてのメソッドが生成されるわけではありません。オーバーライドする必要のあるメソッドのみが生成されます。Tag インタフェースや BodyTag インタフェースのその他のすべてのメソッドをオーバーライドする必要がある場合は、TagHandler ファイルにそれらを取り込んでください。

表 2-2　タグハンドラで生成されるメソッド

インタフェース	メソッド
Tag	*doEndTag
	*doStartTag
	otherDoEndTagOperations
	otherDoStartTagOperations
	shouldEvaluateRestOfPageAfterEndTag
	theBodyShouldBeEvaluated
	theBodyShouldBeEvaluatedAgain
BodyTag	Tag インタフェース用に生成されるすべてのメソッドおよび次のメソッド
	*doAfterBody
	writeTagBodyContent

タグハンドラの再生成

タグライブラリを開発する場合は、プログラミングロジックをタグハンドラに追加することで、カスタムアクションで必要な機能を提供します。開発段階で、TLD にその他の属性またはスクリプティング変数を追加しなければならない場合もあります。この場合、対応するクラスメンバーが作成されるよう、タグハンドラを生成し直す必要があります。これを実行すると、タグハンドラのメソッドは、再生成されるものもありますが、そのままのものもあります。

IDE は、doStartTag メソッド、doEndTag メソッド、doAfterBody メソッドを再生成します。タグハンドラを生成し直すと、変更が上書きされるため、ソースエディタでは、これらのメソッドを編集することができません。

再生成されるメソッドを編集する代わりに、これらの再生成されたメソッドが呼び出すメソッドにカスタムコードを置きます。たとえば、doStartTag メソッドは、otherDoStartTagOperations メソッドと theBodyShouldBeEvaluated メソッドを呼び出します。JSP 仕様は、タグの本体が評価される前に、タグの最初で実行される必要のある処理に対し、doStartTag メソッドを使用する必要があると示しています。

さらに doStartTag メソッドは、int 型の値を戻すことにより、本体が評価される必要があるかどうかを示します。Forte for Java では、タグの最初で実行する必要のある処理に対し、otherDoStartTagOperations メソッドを使用し、本体が評価される必要があるかどうかを決定する Boolean を戻すために theBodyShouldBeEvaluated メソッドを使用します。これらの 2 つのメソッド中のコードは、再生成による影響を受けません。

次の表は、再生成されるメソッド、および編集できるメソッドを示しています。

表 2-3　タグハンドラで編集可能なメソッド

編集できないメソッド	カスタムコードを置くメソッド
doEndTag	otherDoEndTagOperations shouldEvaluateRestOfPageAfterEndTag
doStartTag	otherDoStartTagOperations theBodyShouldBeEvaluated
doAfterBody	writeTagBodyContent theBodyShouldBeEvaluatedAgain

カスタムタグライブラリのパッケージと配備

JSP ページにカスタムタグライブラリを参照させるためには、そのタグライブラリは JSP ページを含んでいる Web モジュール内になければなりません。Web モジュールで使用するためにタグライブラリを配備するには、まず JAR ファイルとしてパッケージにする必要があります。パッケージにすると、タグライブラリを次の 4 つの方法で Web モジュールに追加できます。

- タグライブラリをタグライブラリリポジトリに入れて、必要な Web モジュールに追加します。Forte for Java 組み込みカスタムタグライブラリは、すでにタグライブラリのリポジトリに常駐し、どの Web モジュールにも追加できることに注意してください。組み込みタグライブラリの詳細については、オンラインヘルプを参照してください。
- JAR ファイルをファイルシステムから Web モジュールに追加します。
- JAR ファイルを、適切な `WEB-INF/lib` ディレクトリにカット&ペーストし、`Taglib` 要素を配備記述子で変更し (配備記述子の編集の詳細については、46 ページの「Web モジュール配備記述子の構成」を参照)、希望するタグライブラリを含んでいる JAR ファイルにマップします。
- タグライブラリを所定の位置でテストします。このメソッドは、開発段階でタグライブラリをテストする場合に便利です。

タブライブラリ JAR ファイルを Web モジュールに追加すると、エクスプローラのファイルシステム区画にもマウントされる (開発ディレクトリがまだマウントされていない場合) ことに注意してください。

▼ タグライブラリを JAR ファイルとしてパッケージ化する

1. エクスプローラで、パッケージ化するタグライブラリ記述子を右クリックします。
2. コンテキストメニューから、「タグライブラリ JAR の作成」を選択します。

   このアクションは、`.jar` 拡張子の付いたファイルを作成します。また、関連する `jarContent` (recipe) ファイルも作成します。このファイルは、クラスやパッケージをライブラリに追加する際に使用されます。

注 - タグハンドラが生成されコンパイルされていることを確認してからタグライブラリ JAR ファイルを作成してください。

▼ タグライブラリリポジトリを使用してタグライブラリを配備する

1. エクスプローラで、該当するカスタムタグライブラリ JAR ファイルのアイコンを右クリックします。

2. 「ツール」>「タグライブラリをリポジトリに追加」を選択します。

   タグライブラリがタグライブラリリポジトリから利用できるようになり、Web モジュールに追加できます。

3. エクスプローラで、タグライブラリを追加する Web モジュールの WEB-INF ノードを右クリックします。

4. コンテキストメニューから、「JSP タグライブラリを追加」を選択します。次に「タグライブラリリポジトリ内を検索」を選択します。

5. 「JSP タグライブラリ・リポジトリブラウザ」で、希望するタグライブラリを選択します。

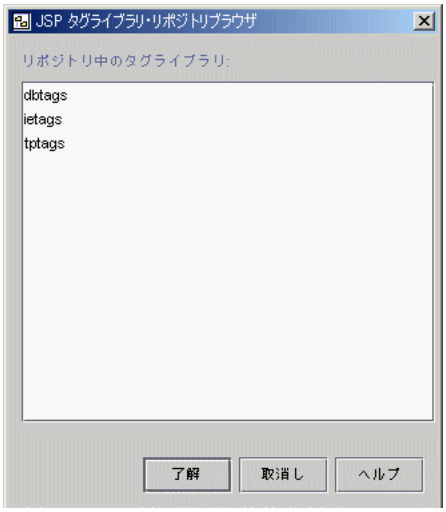

図 2-11 JSP タグライブラリリポジトリブラウ

6. 「了解」をクリックしてください。

▼ JAR ファイルをファイルシステムから追加してタグライブラリを配備する

1. エクスプローラで、タグライブラリを追加する Web モジュールの WEB-INF ノードを右クリックします。

2. コンテキストメニューから、「JSP タグライブラリを追加」を選択します。次に「ファイルシステム内を検索」を選択します。

3. 「JSP タグライブラリブラウザ」で、該当する JAR ファイルを表示し、選択してください。

図 2-12 ファイルシステム内の JSP タグライブラリブラウザ

4. 「了解」をクリックします。

「JSP タグライブラリを追加」メニューを使用してタグライブラリを Web モジュールに追加すると、Web モジュールの配備記述子は自動的に更新され、タグライブラリのエントリが取り込まれます。このエントリは、JSP ページがタグを取得するために使用できる URI にタグライブラリをマップします。

▼ JAR ファイルをコピー＆ペーストしてタグライブラリを配備する

1. エクスプローラで、カットする JAR ファイルを右クリックします。
2. コンテキストメニューから、「コピー」または「カット」を選択します。
3. JAR ファイルをペーストしたい `WEB-INF/lib` ディレクトリを右クリックします。
4. コンテキストメニューから、「ペースト」>「コピー」を選択します。

次に、希望するタグライブラリの入っている JAR ファイルに `Taglib` 要素がマップされるように、配備記述子で Taglib 要素を変更しなければならない場合があります。この手続きは、通常は必要ありません。IDE ではたいてい `Taglib` 要素を自動的に追加します。正しい情報がファイルにあることを確認するには、ステップ 5 からステップ 8 の手順を行なってください。

5. エクスプローラで、`web.xml` ファイル (配備記述子) を右クリックし、「プロパティ」を選択します。「タグライブラリ」プロパティのフィールドをクリックします。

タグライブラリ
フィールド

図 2-13 配備記述子の「タグライブラリ」フィールド

6. 「…」ボタンをクリックして、「プロパティエディタ:タグライブラリ」ダイアログを表示します。

図 2-14 「プロパティエディタ:タグライブラリ」ダイアログ

7. プロパティエディタで、「編集」をクリックして「編集 タグライブラリ」ダイアログを表示します。希望した タグライブラリ の場所が `/WEB-INF/lib/<yourTagLib>.jar`. であることを確認できます。

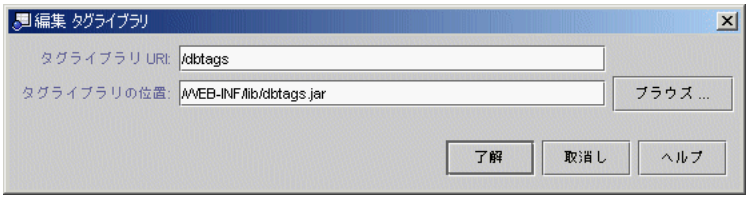

図 2-15 「編集 タグライブラリ」ダイアログ

8. 必要に応じて、タグライブラリ URI、タグライブラリの位置を編集するか、「ブラウズ」ボタンをクリックして、新しいタグライブラリの場所を選択してから、ダイアログを閉じます。

▼ 所定の位置でタグライブラリをテストする

1. タグライブラリがまだ Web モジュールにない場合は、21 ページの「既存のディレクトリを Web モジュールに変換する」に示す手順を使用して、タグライブラリの入っているファイルシステムを変換してください。

   `.tld` ファイルと生成されコンパイルされた Java のタグハンドラクラスを所定の場所に置きます。

2. 配備記述子の `Taglibs` 要素を `/<yourTagLib>.tld` に変更します。

   `Taglib` 要素の変更に関する詳細については、上記手順のステップ 5 を参照してください。

3. JSP ページを作成し、参照を新しいタグに追加します。

4. JSP を実行します。

第3章

Webアプリケーションの実行、デバッグ、および配備

この章は、アプリケーションのプログラム方法についてすでに考慮されており、実行、デバッグ、および配備を開始する準備ができていることを前提としています。

この章では、Forte for Java IDE を使用して Web モジュールを実行、デバッグ、および配備する方法について概説します。全体的な観点から、アプリケーションのテストおよび修正段階で実行するタスクを順序付けし、個々のプログラミングタスクの詳細について考察します。

Web アプリケーションと、その JSP ページ、サーブレット、および beans を含む標準的な Web コンポーネントの作成プロセスの説明については、第 2 章を参照してください。

Web モジュールの開発プロセスに関するあらゆる面の詳細については、『CD ShopCart チュートリアル』を参照してください。

Web モジュールの実行とデバッグのワークフロー

この節では、Forte for Java IDE を使用する Web モジュールの実行とデバッグに関するワークフローの概要を説明します。この概要では、コーディングやテスト方法を反復して説明しません。ここでは、主な開発タスクをリストし、これらのタスクを実行する論理的な順序を示します。各タスクは、そのタスクについての詳しい情報を提供するこの章の後ろの節への参照です。さらに JSP サーブレットモジュールのオンラインヘルプでもこれらのタスクについて情報を提供しています。

Forte for Java IDE を使用して Web モジュールの実行とデバッグを行なう手順は次のとおりです。

1. Web モジュールを構成します (46 ページの「Web モジュール配備記述子の構成」を参照)。アプリケーションを正しく実行するためには、この配備記述子の構成が必要です。

2. IDE でアプリケーションをテスト実行します (アプリケーションのテスト実行については 49 ページの「Web モジュールのテスト」を参照)。

   - アプリケーションをテスト実行するには、組み込み Tomcat サーバーまたは iPlanet Application Server のどちらかを選択します。どちらの場合も、データベースドライバが使用できることを確認してください。
   - JDBC™ データベースドライバを必要とする Web モジュールの場合は、Forte for Java インストールディレクトリの `lib/ext` ディレクトリに Web モジュールをコピーします。
   - このディレクトリにドライバを配置すると、Forte for Java の内部クラスパスにディレクトリが追加され、自分のデータベースを使用してアプリケーションがテストできるようになります。システムクラスパス変数にデータベースドライバを追加しても、このステップを実行したことにはなりません。`lib/ext` ディレクトリにドライバを追加する必要があります。

3. JSP ページ、サーブレット、および Web モジュールをデバッグし、オプションで HTTP モニターを使用してレコードデータをモニターします。HTTP モニターの使用についての詳細は、58 ページの「Web サーバー上でのデータフローのモニター」を参照してください。ソースレベルのデバッグについての詳細は、54 ページの「JSP とサーブレットでのソースレベルデバッガの使用」を参照してください。

4. Web モジュールを WAR ファイルとしてパッケージ化して配備します (67 ページの「Web モジュールのパッケージと配備」 を参照)。

Web モジュール配備記述子の構成

すべての Web モジュールには、Web モジュールの `WEB-INF` ディレクトリにある `web.xml` と名付けられた XML ファイル形式の**配備記述子**が含まれます。配備記述子は、Web モジュールの配備環境、つまり Web コンテナに次のような構成情報を提供します。

- `ServletContext` オブジェクトに対する初期化パラメータ (Web モジュールの実行時表現)

- サーブレットと JSP ページの定義、およびそれらの URI へのマッピング
- URI への タグライブラリのマッピング
- MIME タイプのマッピング
- セッションのタイムアウト間隔
- 開始ファイルのリスト
- エラーコードおよび例外のリソースへのマッピング
- セキュリティ構成

IDE では次の 2 種類の方法で配備記述子を構成することができます。

- エクスプローラで配備記述子の要素をブラウズし、そのプロパティエディタによりそれらを編集する方法。この方法については、オンラインヘルプで詳しく説明しています。配備記述子の各プロパティに関する詳細については、オンラインヘルプを参照してください。
- ソースエディタで配備記述子ファイル (`web.xml`) を開き、それを手動で編集する方法。

▼ プロパティエディタを使用して web.xml を編集する

1. エクスプローラで Web モジュールディレクトリを探し、その WEB-INF/ サブディレクトリを開きます。
2. web.xml ノードを右クリックし、「プロパティ」を選択します。

   web.xml のプロパティシートが表示されます。

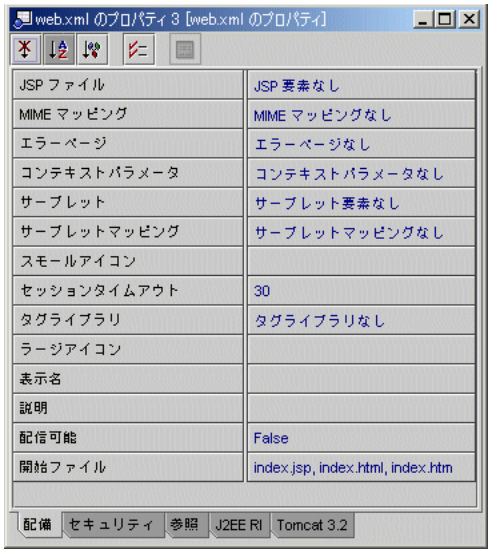

図 3-1　web.xml プロパティシート

3. web.xml プロパティシートから、編集する情報のカテゴリ (たとえば「サーブレットマッピング」) を選択します。
4. 対象となるカテゴリの値フィールドにある「…」ボタンをクリックし、プロパティエディタを表示します。
5. プロパティエディタで変更を行います。
   - 変更が完了したら、web.xml プロパティシートを閉じます。

▼ ソースエディタで web.xml ファイルを編集する

上級ユーザーであれば、web.xml アイコンをダブルクリックして、web.xml ファイルを直接編集することができます。また、外部テキストエディタを使用しても編集可能です。変更を保存すると、IDE がファイルを自動的に構文解析し、エラーがあればXML パーサー区画の出力ウィンドウに表示します。

Web モジュールのテスト

単一の Web モジュールを実行時に内部 Tomcat 3.2 Web Sever に配備することによって、IDE でこのモジュールをテストすることができます。IDE は構成の設定作業を行い、開発サイクル中にテストを繰り返し簡単に行なえるようにします。

いくつかの Web モジュールを 1 つのグループとして実行するには、50 ページの「Web モジュールグループを作成し実行する」で説明するように、最初に **Web モジュールグループ**を作成する必要があります。Web モジュールグループとは、Web モジュールだけを含む J2EE アプリケーションと同様 (同一ではない) の、IDE 固有のオブジェクトのことです。

Web モジュールと Web モジュールグループを実行する際には、HTTP モニターを使用してレコードデータのフローを調べることができます。詳細については、58 ページの「Web サーバー上でのデータフローのモニター」を参照してください。

Web モジュールの実行プロパティを設定するには、Web モジュールの WEB-INF ディレクトリ (エクスプローラの「ファイルシステム」タブにある) を右クリックして、コンテキストメニューで「プロパティ」を選択します。

注 - iPlanet Application Server に配備する Web モジュールや Web モジュールグループを構成するには、追加のステップを実行する必要があります。詳細については、iPlanet Application Server プラグインのオンラインヘルプを参照してください。

▼ 単一 Web モジュールを実行する

1. エクスプローラの「ファイルシステム」タブにある WEB-INF ノードをクリックして、該当する Web モジュールを選択し、コンテキストメニューから「すべてを構築」を選択します。

   Web モジュールを構築すると、確実に Web モジュール内のすべてのファイルが保存され、すべてのクラスおよびコンポーネントがコンパイルされます。

2. 実行する Web モジュールの WEB-INF ノードを右クリックします。

3. コンテキストメニューから「実行」または「実行 (再読み込み強制)」を選択します。

   Web モジュールが実行され、選択したデフォルトブラウザに表示されます。問題が発生した場合は、実行時に表示される出力ウィンドウで詳細を調べてください。

クラスパスの構成

サーブレットを実行する場合の、クラスパスの Web モジュール要素の順序は以下のとおりです。

1. `WEB-INF/classes`
2. `WEB-INF/lib` 内にあるすべての JAR ファイル
3. Web モジュールのルートディレクトリ
4. 残りの IDE の マウント済みファイルシステム

▼ Web モジュールグループを作成し実行する

Web モジュールグループを作成して実行するには、以下の 3 つのメインタスクを実行します。

1. Web モジュールグループを作成します。
2. Web モジュールグループの一部として読み込むそれぞれの Web モジュールについて、URL マッピングを設定します。
3. Web モジュールグループのターゲットサーバーを指定します。

▼ Web モジュールグループを作成する

1. メインウィンドウで「ファイル」>「新規」を選択し、テンプレートウィザードから新規作成を開きます。
2. 「JSP & サーブレット」テンプレートカテゴリから、「Web モジュールグループ」テンプレートを選択し、「次へ」をクリックします。
3. Web モジュールグループのファイルを特定しやすいように名前を入力し、ファイルの作成場所を選択します。

   Web モジュールグループファイルは、配備のためのパッケージに含まれてしまう場合があるので、Web モジュールディレクトリの中には置かないでください。
4. 「完了」をクリックし、Web モジュールグループファイルを作成し、ウィンドウを閉じます。

▼ Web モジュールグループの一部として読み込むそれぞれの Web モジュールについて URL マッピングを設定する

1. エクスプローラの「ファイルシステム」タブで、Web モジュールグループファイルを右クリックし、コンテキストメニューから「Web モジュールを追加」を選択します。
2. 「Web モジュールを追加」ダイアログで、追加する Web モジュールの名前をリストから選択します。

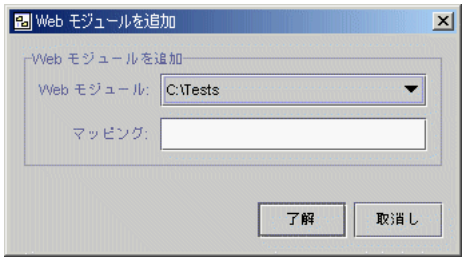

図 3-2　「Web モジュールを追加」ダイアログ

3. Web モジュールを識別するマッピング名を入力します。

   この Web モジュールからファイルを実行する場合は、マッピング済みの名前をファイル名の前の URI に挿入します。

4. 一緒に実行するすべての Web モジュールをすべて追加するまで、ステップ 2 と 3 を繰り返します。
5. 「了解」をクリックして設定を保存し、ダイアログを閉じます。

▼ Web モジュールグループのターゲットサーバーを指定する

1. エクスプローラの「ファイルシステム」タブで、Web モジュールグループファイルを右クリックし、「プロパティ」を選択します。
2. Web モジュールグループのプロパティシート上で、「ターゲットサーバー」プロパティをクリックして値フィールドを編集可能にし、「…」ボタンをクリックしてターゲットサーバーのプロパティエディタを表示します。

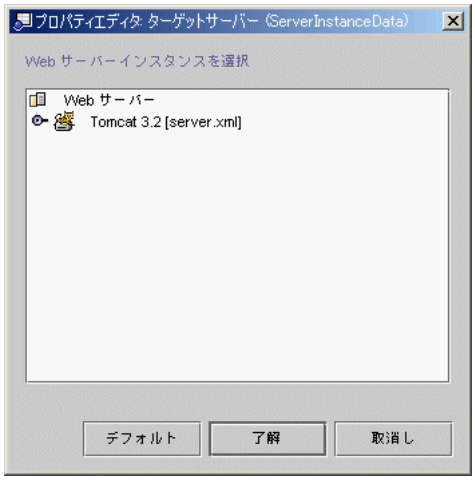

図 3-3　ターゲットサーバーのプロパティエディタ

3. ターゲットサーバーのプロパティエディタでサーバーを選択してから、「了解」をクリックしてプロパティエディタを閉じます。

   Web モジュールの「ターゲットサーバー」プロパティを編集して、個々の Web モジュールにターゲットサーバーを設定することができます。ただし、Web モジュールが Web モジュールグループの一部として実行されている間にこの Web モジュールのコンポーネントが実行される場合、Web モジュールは、Web モジュールグループの「ターゲットサーバー」プロパティによって指定されたサーバー内で実行されます。

注 - ターゲットサーバーの指定は任意です。ターゲットサーバーを指定しない場合は、サーバーレジストリからのデフォルトサーバーが使用されます。

▼ サーバーレジストリでデフォルトサーバーを指定する

1. エクスプローラの「実行時」タブで「サーバーレジストリ」ノードを展開し、「インストールされているサーバー」ノードの下にある、希望するサーバーを表すノードを右クリックします。
2. コンテキストメニューから、「デフォルトとして設定」を選択します。
3. あるいは、エクスプローラの「実行時」タブで、「デフォルトサーバー」ノードの下にある「Web モジュールグループ」ノードを右クリックします。コンテキストメニューから「デフォルトサーバーを設定」を選択します。「デフォルト Web サー

バーを選択」ダイアログで、リストから希望するサーバーを選択して、「了解」をクリックします。どちらの手順を使用しても、「Web モジュールグループ」ノードはデフォルト Web サーバーを示すように変更されます。

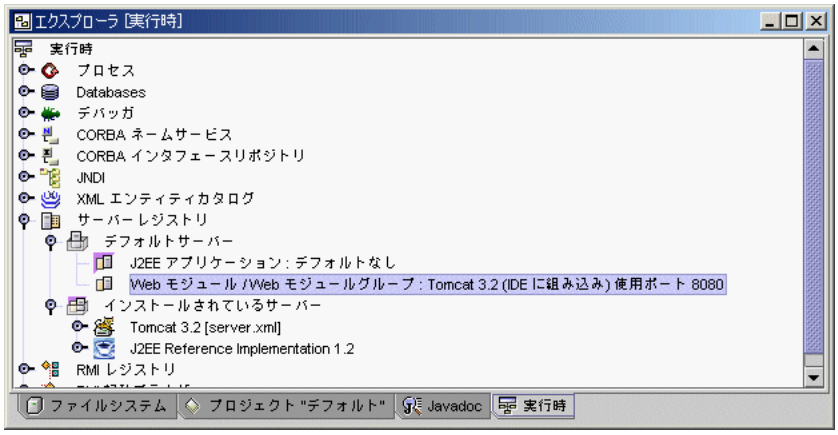

図 3-4　Web モジュールグループのデフォルトサーバー

Web アプリケーションのデバッグ

Forte for Java IDE には、Web アプリケーションをデバッグするツールが 2 つあります。

- JSP とサーブレットの両方に使用できるソースレベルデバッガ
- ソースレベルデバッガを使用する前に、サーブレットリソースの問題を障害追跡するための軽量デバッグコンポーネントである、HTTP モニター

JSP とサーブレットでのソースレベルデバッガの使用

ソースレベルデバッガは、標準的な IDE デバッグ環境にいくつかの拡張機能を追加して構成されています。これらの強化機能を使用すれば、JSP ファイルと生成されたサーブレットファイルとを同時に表示することができます。一方で設定されたブレークポイントが、他方でも自動的に反映されます。(ただし、生成されたサーブレットからブレークポイントを削除しても、対応する JSP ソースファイルからは削除されません。)

取り込まれたすべてのファイルを含む JSP ファイルが、単一のサーブレットファイルにマップされます。具体的に言えば、JSP ファイルの 1 行が、サーブレットファイルの 1 行または複数行にマップされます。ただし、この逆は成り立たず、サーブレットファイルの行の中には、JSP ファイル内のどの単一行にもマップされないものがあります。

▼ デバッガを開始する

- JSP コンパイルを明示的に呼び出すには、メインメニューバーから「構築」>「コンパイル」を選択します。
- コンパイルを自動的にトリガーするには、メインメニューバーから「デバッグ」>「開始」を選択します。
- Web モジュールグループをデバッグするには、Web モジュールグループ内にあるいずれかの Web モジュールから `WEB-INF` ディレクトリを選択し、「デバッグ」>「開始」と選択します。

注 - デフォルトサーバーまたはターゲットサーバーとして iWS または RI を選択した場合、JSP ページや JSP ページから生成されたサーブレットがデバッグできなくなります。Tomcat 3.2 Web Sever (すなわち、組み込み IDE Web サーバー) を使用して、JSP ファイルをデバッグすることだけは可能です。

Forte for Java IDE での標準のデバッグについての詳細は、オンラインヘルプを参照してください。

デバッガを開始すると、「HTTP モニター」ウィンドウが表示されます。デバッグプロセスの補助として HTTP モニターを使用する方法の詳細については、58 ページの「Web サーバー上でのデータフローのモニター」を参照してください。

▼ JSP デバッガオプションを設定する

IDE のグローバルオプションを使用して、JSP デバッグセッションをカスタマイズすることができます。エラーを表示させたいファイルが指定できます (JSP ソースファイルと生成されたサーブレットファイルの両方を開いてデバッグする場合)。デバッグの際に、静的な HTML 行をスキップするかどうかも指定できます。静的な HTML 行とは、JSP 要素やスクリプト言語をまったく含まない行のことです。

1. メインメニューから、「ツール」>「オプション」を選択します。

   「オプション」ウィンドウが開きます。

2. 「オプション」ウィンドウで、「JSP & サーブレット (拡張)」を選択します。「プロパティ」区画が表示されます。

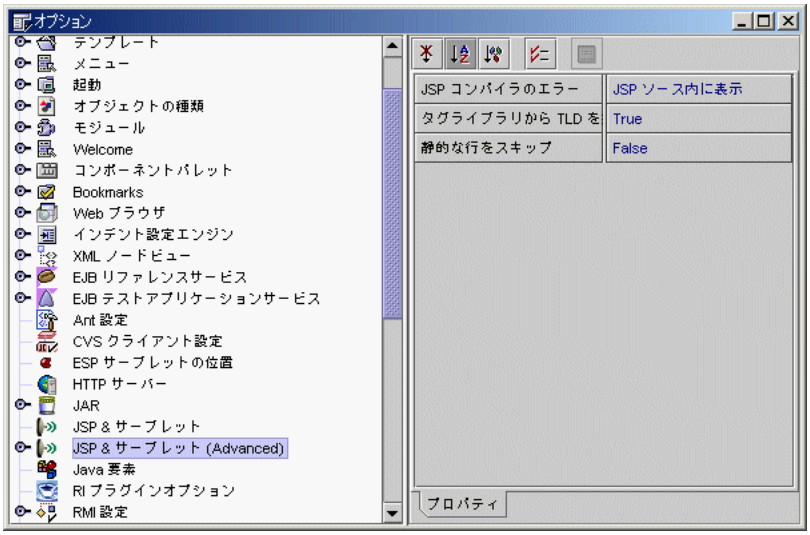

図 3-5　デバッグオプションのプロパティ

3. コードをステップ実行する場合に、JSP ソース内の JSP タグの間にある HTLM 行 (および生成されたサーブレットソース内の対応する行) をスキップするには、「静的な行をスキップ」を「True」に設定します。

4. JSP コンパイラのエラーを JSP ソースファイル内に表示するには、「JSP コンパイラのエラー」を「JSP ソース内に表示」に設定します。JSP コンパイラのエラーをサーブレットソースファイル内に表示するには、「JSP コンパイラのエラー」を「サーブレットソース内に表示」に設定します。

▼ デバッグ時に JSP ファイルとサーブレットファイルの両方を表示する

1. エクスプローラの「ファイルシステム」タブで、デバッグする JSP ソースファイルを選択して右クリックし、コンテキストメニューを表示します。

2. JSP ファイルがまだコンパイルされていない (すなわち、「サーブレットを表示」メニューが選択できない) 場合は、コンテキストメニューから「コンパイル」を選択します。

3. JSP ファイルがコンパイルされたら、コンテキストメニューから「サーブレットを表示」を選択します。

   ソースエディタが開き、生成されたサーブレットコードが表示されます。

4. 「エクスプローラ」ウィンドウで JSP ファイルが選択されたままの状態で、コンテキストメニューから「開く」を選択します。

   デフォルトでは、ソースエディタの新しく追加されたタブ区画に、JSP ソースファイルコードが表示されます。この時点では、サーブレットコードと JSP コードのどちらかが表示されていますが、同時に両方表示することはできません。

5. エディタ内の現在の表示上で右クリックし、コンテキストメニューから「表示をクローン」を選択します。

   このアクションを実行すると、同じコード (JSP) 表示を持つ新しいソースエディタウィンドウが開きます。

6. 元のソースエディタウィンドウで、タブをクリックして別の表示 (サーブレット) に切り替えます。

   2 つのエディタウィンドウが表示され、一方は JSP コードの表示、他方はサーブレットコードの表示になります。これで、一方の表示で変更をマークすると、他方の表示でも変更が反映されたことがわかります。

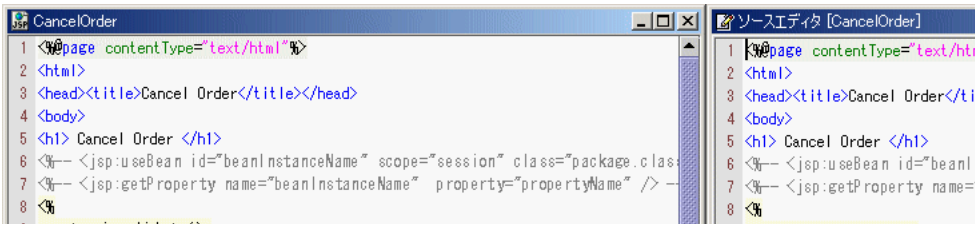

図 3-6　JSP コードとサーブレットコードの表示

注 - デバッグコマンドは、コマンドが発行された時点でフォーカスされている表示 (ファイル) に適用されます。別の表示にコマンドを発行するには、そのウィンドウをクリックしてフォーカスを設定し、コマンドを実行するだけです。

Web サーバー上でのデータフローのモニター

HTTP モニターは、サーブレットエンジンでの JSP ファイルとサーブレットファイルの実行に関するデータを収集する、サーブレットコンポーネントのデバッグツールです。この情報は、Forte for Java IDE 内では利用されません。モニターは、処理される各 HTTP 要求について、着信要求に関するデータ (たとえば `GET` メソッドと `POST` メソッド)、サーバー上で維持されているデータ状態、およびサーブレット環境を記録します。データを表示し、将来のセッションのためにデータを格納し、以前の要求を再実行および編集することができます。HTTP モニターを使用すると、前の節で説明したソースレベルデバッガを使用して実行する前に、どのリソースに問題が含まれているかを知ることができます。

HTTP モニターには、実行サーバーからデータを受信するための内部 Web サーバーが必要です。つまり、HTTP モニターは、IDE の内部 HTTP サービスが実行中でない限りデータを記録しません。内部 HTTP サービスが実行中でない場合、以前に記録されたデータは表示できますが、新しいデータはいっさい表示できません。Web サーバー上でのデータフロー (HTTP モニターのデータレコードまたは IDE のGUI ではトランザクションと呼ばれる) に関する情報は、明示的に要求しない限り、IDE を終了するまで格納されます。

▼ HTTP モニターを表示する

- メインメニューバーから HTTP モニターを開始するには、「デバッグ」>「HTTP モニター」、または「表示」>「HTTP モニター」を選択します。

  デバッグの作業スペースでは、デバッグツールバーの右側にある「HTTP モニター」ボタンをクリックして、HTTP モニターを表示させることもできます。

モニターデータレコードの表示

HTTP モニターは、2 つのパネルで構成されています。HTTP 要求レコードパネルには、保存されているトランザクションを示すツリー表示が含まれます。トランザクションデータの表示パネルには、現在のセッションで記録されたモニターデータ情報に関連付けられたデータが表示されます。

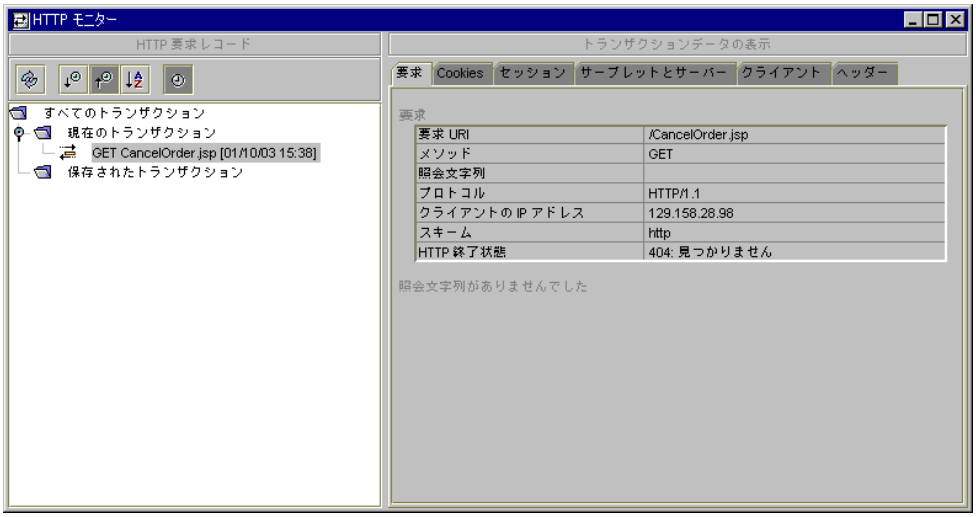

図 3-7　要求トランザクションデータが表示された HTTP モニター

ツリー表示では、「すべてのトランザクション」カテゴリに、「現在のトランザクション」と「保存されたトランザクション」の 2 つのサブカテゴリが含まれます。個々のモニターデータレコードは、このサブカテゴリのいずれかに存在します。「現在のトランザクション」のエントリが使用できるのは、現在の IDE セッション中だけです。現在のモニターデータレコードは、サーバーが再起動されても持続します。現在のモニターデータレコードが消去されるのは、IDE が再起動されたとき、またはユーザーが削除したときだけです。「保存されたトランザクション」のエントリは、ユーザーが削除するまで持続します。すべてのカテゴリにあるモニターデータレコードは、ツリー表示の上にあるボタンを使用して、さまざまな基準に従ってソートすることができます。ソート基準の詳細については、オンラインヘルプを参照してください。

HTTP 要求レコードパネル内のモニターデータレコードが選択されると、そのトランザクションに対応する情報が、トランザクションデータの表示パネル内に表示されます。データ表示パネルは、以下の区画で構成されます。

要求区画

「要求」区画には、図 3-7 に示すように、要求 URI、メソッド、照会文字列、パラメータ、プロトコル、クライアントの IP アドレス、スキーム、および HTTP 終了状態が表示されます。

Cookies 区画

「Cookies」区画には、名前と値を含む、受信クッキーと送信クッキーのリストが表示され、送信クッキーについては、クッキーの有効期限や、クッキーがセキュアプロトコルを必要とするかどうかが表示されます。

セッション区画

「セッション」区画には、要求が処理される前と後の要求に関連付けられた HTTP セッションの状態が表示されます。要求が処理された結果、セッションが作成されたか破壊されたかが表示されます。セッションプロパティには、セッション ID および日付と時刻が含まれます。セッション属性やアクティブでない最大間隔を含む、トランザクション後のアクセスデータも表示されます。

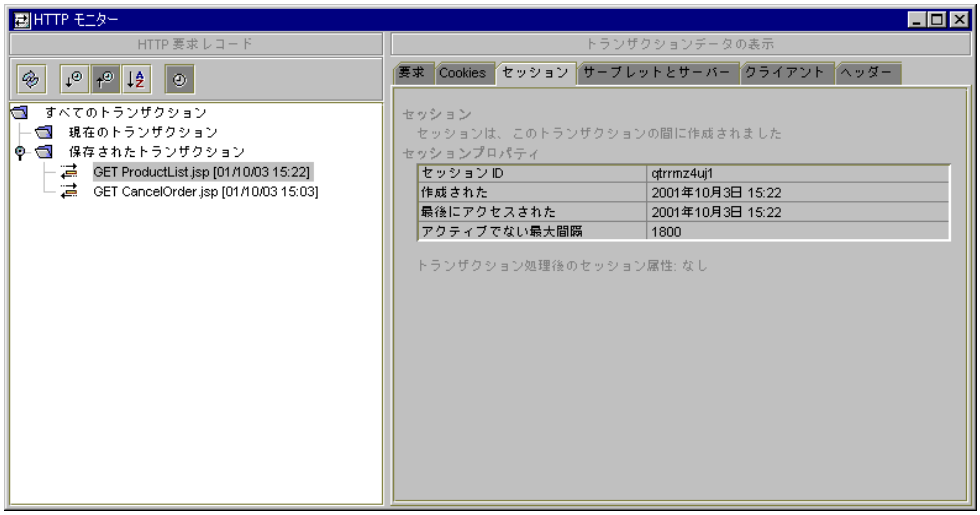

図 3-8　HTTP モニターのセッション区画

サーブレットとサーバー区画

「サーブレットとサーバー」区画には、構成時のサーブレット名、そのクラス名、そのパッケージ名、オプションのサーブレット情報、サーブレットへの相対パス、および変換されたパスが表示されます。サーブレットコンテキスト (すなわち、コンテキストへの絶対パス)、関連する初期化パラメータ、および、Java プラットフォーム、Java バージョン、サーブレットエンジンのホスト名、HTTP サービスのポート番号などのサーブレットエンジンプロパティも含まれます。

クライアント区画

「クライアント」区画には、プロトコル、クライアントの IP アドレス、使用されるソフトウェア、ロケール、および、受け付けられるエンコーディング、ファイル形式、文字セットが表示されます。

ヘッダー区画

「ヘッダー」区画には、要求と共に着信した HTTP ヘッダーが表示されます。ヘッダーは、HTTP クライアント (たとえばブラウザ) によって構築され、通常はクライアントの種類 (ソフトウェアや OS)、言語設定、ブラウザが受け付けるファイル形式などの情報が含まれます。接続情報も提供されます。

▼ HTTP モニターデータレコードを保存する

「すべてのトランザクション」ツリービューの「現在のトランザクション」サブカテゴリに、モニターデータレコードを保存することができます。このアクションを実行すると、将来の IDE セッションで、モニターデータレコードの表示や再実行ができます。

1. 保存するモニターデータレコードを右クリックします。

   Shift キーまたは Ctrl キーを使用すると、一度に複数のトランザクションが選択できます。

2. コンテキストメニューから「保存」を選択します。

   選択されたモニターデータレコードは、「保存されたトランザクション」サブカテゴリに移動 (コピーではない) されます。

HTTP モニターデータレコードの削除

「すべてのトランザクション」ツリービューにある、現在のモニターデータレコード、または「保存されたトランザクション」サブカテゴリに保存されたレコードを削除することができます。

▼ 保存されたモニターデータレコードまたは個々のモニターデータレコードを削除する

1. 削除するモニターレコードを右クリックします。Shift キーまたは Ctrl キーを使用すると、一度に複数のデータレコードが削除できます。
2. コンテキストメニューから「削除」を選択します。

▼ 現在のモニターデータレコードをすべて削除する

1. 「現在のトランザクション」フォルダを右クリックします。
2. コンテキストメニューから「現在のトランザクションを削除」を選択します。

▼ 保存されたモニターデータレコードをすべて削除する

1. 「保存されたトランザクション」フォルダを右クリックします。
2. コンテキストメニューから「保存されたトランザクションを削除」を選択します。

▼ モニターデータレコードをすべて削除する

1. 「すべてのトランザクション」フォルダを右クリックします。
2. コンテキストメニューから「すべてのトランザクションを削除」を選択します。

要求の再実行

「すべてのトランザクション」ツリービューにある「現在のトランザクション」と「保存されたトランザクション」の両方のサブカテゴリに関連付けられた HTTP 要求を再実行することができます。この実行方法では、元の HTTP 要求を再送信します。

▼ 元のトランザクションをサーバーに再送信する

1. 再実行するトランザクションを示すモニターデータレコードを右クリックします。
2. コンテキストメニューから「再実行」を選択します。

   トランザクションは、選択したブラウザまたはサーバーに表示されます。

▼ 指定したトランザクションを編集して再送信する

1. 編集するトランザクションを示すモニターデータレコードを右クリックします。
2. コンテキストメニューから「編集と再実行」を選択します。

   「編集と再実行」ダイアログが表示されます。このダイアログで、1 つまたは複数のパラメータ、要求、サーバー情報、およびヘッダーを編集した後、要求情報を再送信します。

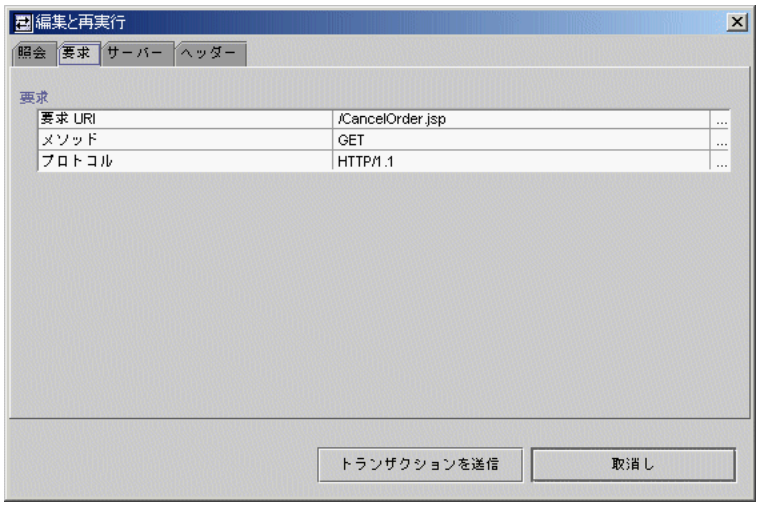

図 3-9　「編集と再実行」ダイアログ

▼ 照会パラメータを編集して再実行する

1. 照会パラメータを追加するには、「照会」タブをクリックしてから「パラメータを追加」ボタンをクリックします。

   「パラメータを追加」ダイアログが表示されます。

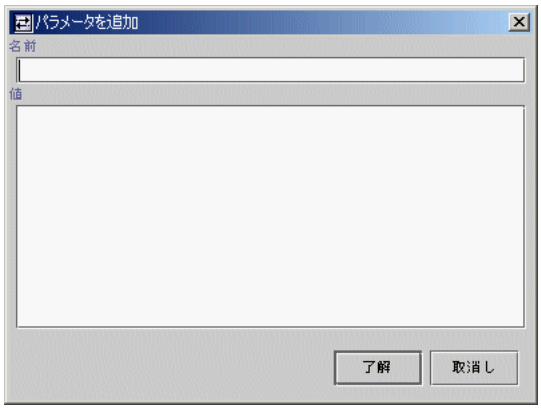

図 3-10 「パラメータを追加」ダイアログ

2. 名前と値を入力し、「了解」をクリックします。

   「編集と再実行」ダイアログの「照会」タブに照会パラメータが表示されます。

3. 照会パラメータを削除するには、パラメータを選択してから「パラメータを削除」をクリックします。

   複数のパラメータを削除するには、Shift キーまたは Ctrl キーを使用してください。

▼ 要求情報を編集して再実行する

1. 「要求 URI」や「プロトコル」などの要求パラメータを編集するには、「要求」タブをクリックして、希望する要求パラメータの「…」ボタンをクリックします。

   「値を編集」ダイアログが表示されます。

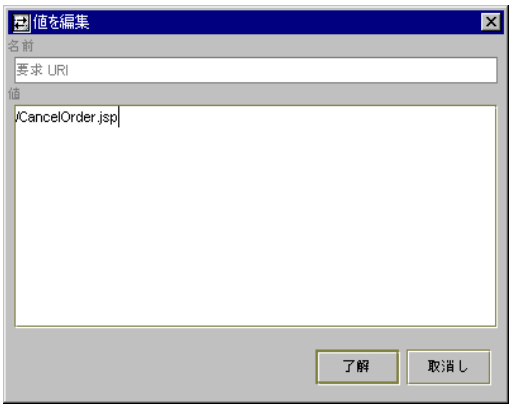

図 3-11 「値を編集」ダイアログ

2. 名前と値を入力し、「了解」をクリックします。

   編集された要求パラメータ値が、「編集と再実行」ダイアログの「要求」タブに表示されます。

3. 要求メソッドを変更するには、「メソッド」フィールドのコンボボックスから、希望するメソッドを選択します。

   たとえば、GET を POST に変更します。

注 - クッキー名はサーバーによって異なるため、再実行する要求に HttpSession ID を持つクッキーが含まれていなければ (たとえば、Tomcat サーバーの JSESSIONID)、処理を実行するコンポーネントが新しい ID を取得します (つまり、新しい HttpSession ID が作成されます)。再実行する要求に HttpSession ID を持つクッキーが含まれていれば、処理を実行するコンポーネントは、タイムアウトしていない限り (タイムアウトしている場合は新しい ID を作成)、現在の属性を備えたその ID の HttpSession を取得します。IDE は、セッションの属性を要求前の属性にリセットしません。この方法により、モニターを使用して複数のセッションを追跡することができます。

▼ サーバー情報を編集して再実行する

1. サーブレットエンジンのホスト名や HTTP サービスのポート番号などのサーバー情報を編集するには、「サーバー」タブをクリックし、希望するサーバーパラメータの隣にある「…」ボタンをクリックします。

   「値を編集」ダイアログが表示されます。

2. 値を入力して「了解」をクリックします。

   編集されたサーバーパラメータ値が、「編集と再実行」ダイアログの「サーバー」タブに表示されます。

▼ ヘッダーを追加および削除して再実行する

1. ヘッダーを追加するには、「ヘッダー」タブをクリックしてから「ヘッダーを追加」ボタンをクリックします。

   「ヘッダーを追加」ダイアログが表示されます。

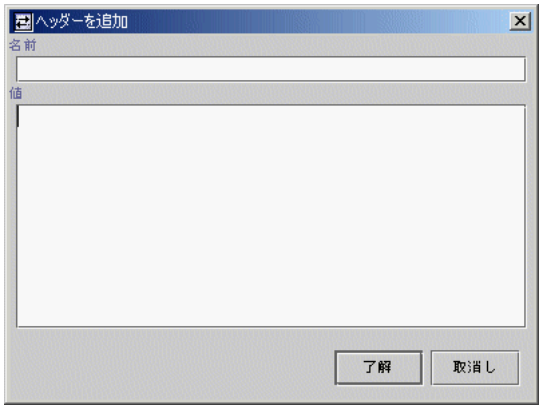

図 3-12 「ヘッダーを追加」ダイアログ

1. 値を入力して「了解」をクリックします。
2. ヘッダーを削除するには、選択してから「ヘッダーを削除」ボタンをクリックします。

   確認ダイアログが表示されます。複数のヘッダーを削除するには、Shift キーまたは Ctrl キーを使用して選択してから、「ヘッダーを削除」ボタンをクリックします。

3. 「Accept」、「Connection」、「Host」、または「User-Agent」などのヘッダーパラメータを編集するには、希望するパラメータの隣にある「…」ボタンをクリックします。(パラメータを直接編集することもできます。)

   「ヘッダーを編集」ダイアログが表示されます。

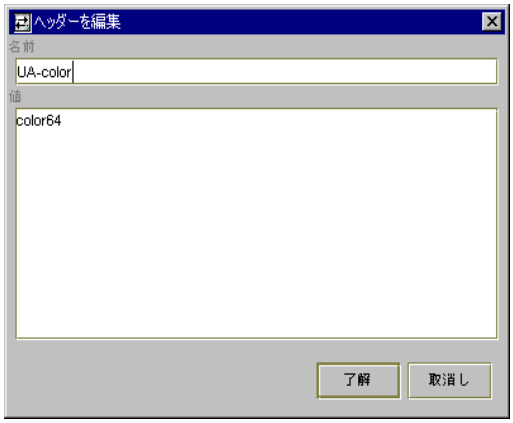

図 3-13 「ヘッダーを編集」ダイアログ

4. 新しい名前と値を入力して、「了解」をクリックします。

   新しい情報が、「編集と再実行」ダイアログの「ヘッダー」タブに反映されます。

Web モジュールのパッケージと配備

Web モジュールを開発する際には、更新が何度も容易に行なえるように、Web モジュールをディレクトリ構造 (つまりパッケージ化されていない形式) から実行するのが一般的です。ただし、Web モジュールの内容の開発準備が整うと、Web モジュールを、転送しやすいように Web ARchive (WAR) 形式のパッケージにすることができます。WAR ファイル形式は、アプリケーションのアーカイブや配備を簡単にするのに役立ちます。J2EE に準拠したすべての Web コンテナは、この形式で Web モジュールを実行することができます。

TP アプリケーションを配備するのに必要なファイルについては、オンラインヘルプ (「透過的な持続性」にある「JAR ファイルの作成」) を参照してください。

注 - Web モジュールに持続可能クラスが含まれる場合は、すべての持続可能クラスが JAR ファイルにパッケージされていること、および Web モジュールの WEB-INF/lib ディレクトリ内に配置されていることを確認してください。JAR コンテンツファイルの作成に使用されるソースは、マウント済みファイルシステムで使用できるものでなければなりません。

▼ Web モジュールを WAR ファイルとしてパッケージ化する

1. エクスプローラで、WEB-INF ノードを右クリックし、コンテキストメニューから「WAR をエクスポート」を選択します。

   コンテキストメニュー上に「WAR ファイルをエクスポート」が表示されていない場合は、パッケージ化する Web モジュールを含むファイルシステムがマウントされているかどうかを調べてください。または、メインメニューバーから「ツール」>「WAR ファイルをエクスポート」を選択します。

2. 「WAR ファイル名を選択」ダイアログで、WAR ファイルの名前を入力します。

3. 「了解」をクリックします。

   WAR ファイルが作成されます。デフォルトでは、Web モジュールファイルシステムの下にあるすべてのファイルが含まれます。

注 - Web モジュールのパッケージ方法は、WEB-INF プロパティシートの「アーカイブ」タブに設定されているプロパティによって異なります。Web モジュールのアーカイブプロパティの編集についての詳細は、オンラインヘルプを参照してください。

これで、選択した外部 Web サーバーに Web モジュールを配備する準備が整いました。ターゲット Web サーバーに関するマニュアルを参照してください。

付録 A

Dreamweaver テンプレートでの作業

Forte for Java IDE では、Macromedia Dreamweaver テンプレートを使用して、次の作業を行なうことができます。

- ソースエディタで Dreamweaver テンプレートを開き、編集することができます。
- 選択したエディタで Dreamweaver テンプレートが開くよう、IDE を構成することができます。
- Dreamweaver テンプレートから JSP ページを生成できます。

ソースエディタで Dreamweaver テンプレートを使用する

IDE は Dreamweaver テンプレートを個々のファイル形式として認識するため (Dreamweaver テンプレートは拡張子 `.dwt` を使用)、Dreamweaver テンプレートをソースエディタで開いて編集することができます。デフォルトでは、エクスプローラでダブルクリックすると、ソースエディタで Dreamweaver テンプレートが開きます。

▼ Dreamweaver アプリケーションで Dreamweaver テンプレートを開くように IDE を構成する

選択した外部エディタで Dreamweaver テンプレートが開くように IDE を構成し直すことができます。

1. 「ツール」 > 「オプション」をクリックします。
2. 「オプション」ウィンドウの左区画で、「JSP & サーブレット (拡張)」と名付けられたノードを開きます。

3. Dreamweaver テンプレートアイコンをクリックします。

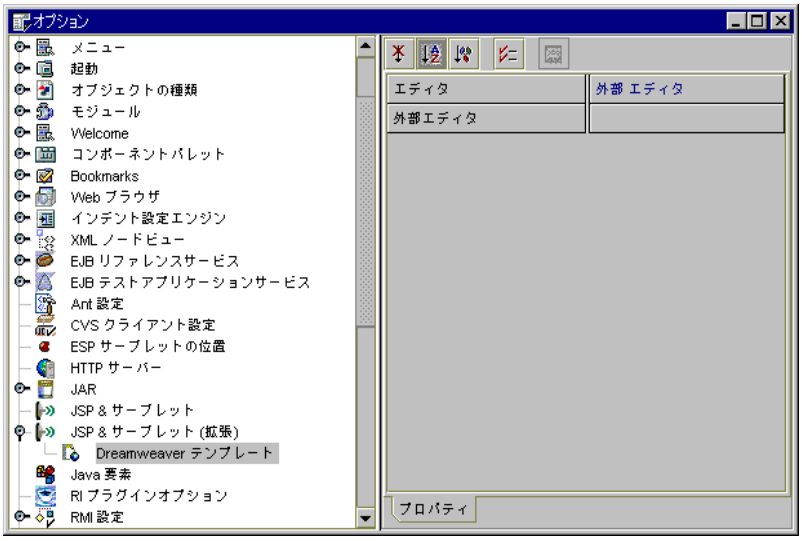

図 A-1　「オプション」ウィンドウの Dreamweaver テンプレートアイコン

4. 右区画で、「エディタ」フィールドをクリックし、コンボボックスから「外部エディタ」を選択します。
5. 「外部エディタ」フィールドをクリックし、「…」ボタンをクリックします。
6. 「フィアルを選択」ダイアログで、実行可能な外部エディタを選択し、「選択」をクリックします。
7. 「オプション」ウィンドウを閉じます。

   Dreamweaver テンプレートをダブルクリックすると、Dreamweaver アプリケーションが開始され、テンプレートが開きます。

▼ Dreamweaver テンプレートから JSP ページを生成する

1. エクスプローラで、Dreamweaver テンプレートを選択します。
2. コンテキストメニューから、「JSP としてテンプレートを保存」を選択します。
3. 「JSP としてテンプレートを保存」ダイアログで、保存先のファイルシステムを選択し、「ファイル名」フィールドに JSP ページの名前を入力し、「了解」をクリックします。

生成された JSP ページは、Dreamweaver テンプレートと同じコンテンツを持ちます。

付録 B

カスタムタグライブラリのチュートリアル

この付録では、簡単なチュートリアルを使用して、次の作業方法を説明します。

- 簡単な「Hello World」タグライブラリを作成し、JSP ページから表示および編集します。
- タグライブラリに属性を追加し、タグハンドラ Bean を再生成します。
- タグライブラリを JAR ファイルとしてパッケージし、JSP ページから表示および編集します。

▼ タグライブラリを作成する

1. 新規の Web モジュールを作成します。詳細については、18 ページの「Web モジュールの作成」を参照してください。

   この操作は、次の図に示すように、Web モジュールディレクトリ構造を作成します。

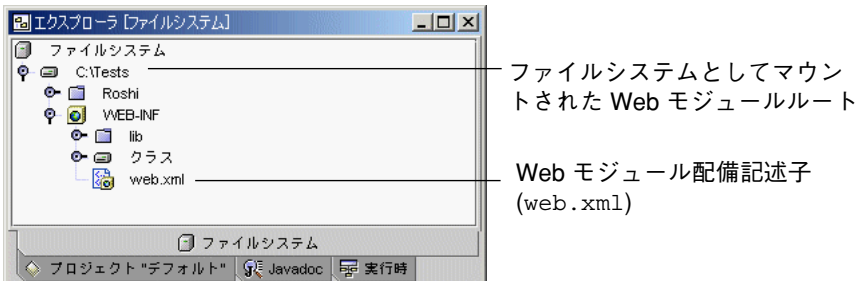

図 B-1　Web モジュールのディレクトリ構造

2. Web モジュールで TLD を作成し、`MyTagLib` と名付けます。

   この操作を実行するには、エクスプローラで Web モジュールのルートディレクトリを右クリックし、コンテキストメニューから、「新規」>「JSP & サーブレット」>「タグライブラリ」を選択します。「テンプレートウィザードから新規作成」の名前フィールドに `MyTagLib` と入力し、「完了」をクリックします。

   この操作は、次の図に示すように、TLD を作成します。

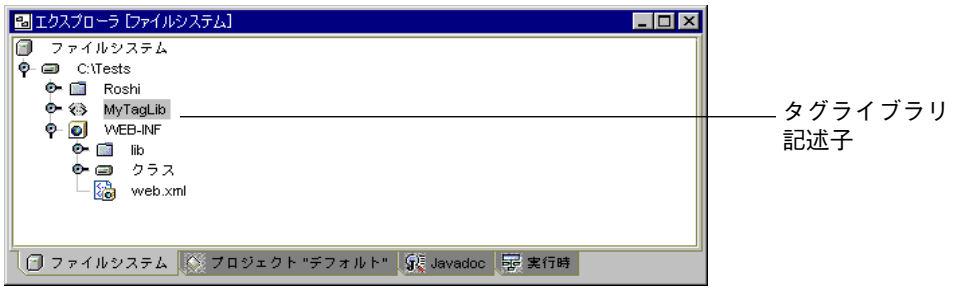

図 B-2　Web モジュールで新たに作成された TLD

3. `HelloWorld` と名付けられたタグ要素を TLD へ追加し、ハンドラクラスとして `HelloWorldTag` を指定します。

   この操作を実行するには、エクスプローラでタグライブラリ記述子 (`MyTagLib` ノード) を右クリックし、コンテキストメニューから「タグを追加」を選択します。「新しいタグを追加」ダイアログで、「タグ名」フィールドに `HelloWorld` と入力します。「タグクラス名」フィールドに **`HelloWorldTag`** と入力します。「了解」をクリックします。

   次の図は、新しく作成されたタグ要素を示します。

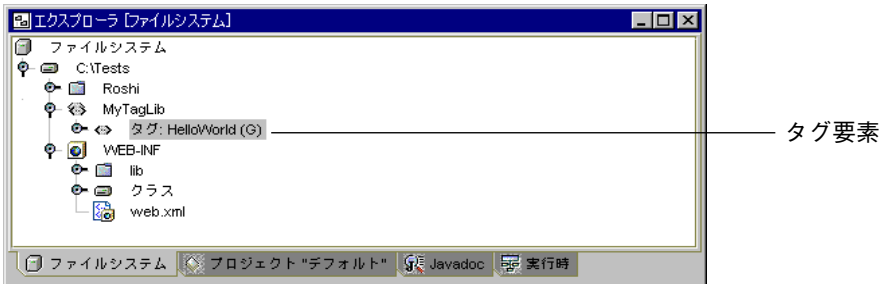

図 B-3　タグ要素

タグ名の後の括弧で囲まれた G は、最後にタグハンドラが生成された後、タグが変更されたことを示します (タグハンドラをまだ生成していないため)。

4. タグハンドラ bean を生成します。

   この操作を実行するには、エクスプローラで TLD を右クリックし、コンテキストメニューから 「タグハンドラを生成」 を選択します。

   この操作により、Web モジュールのルートディレクトリに、`MyTagLib` と名付けられたパッケージが生成されます。このパッケージには、次の図に示すように、タグハンドラ bean `HelloWorldTag` が含まれます。

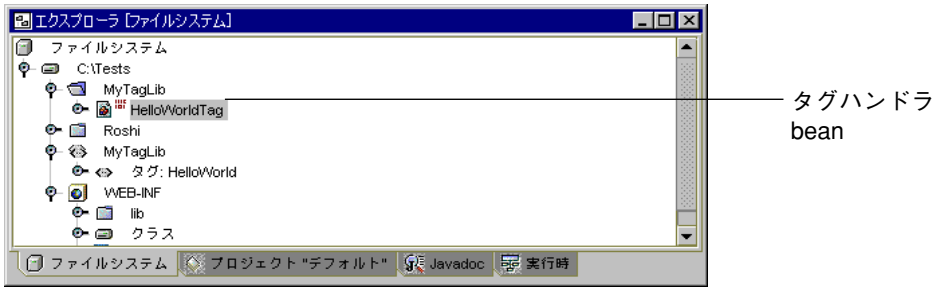

図 B-4　タグハンドラ bean

5. `HelloWorldTag` bean の `otherDoStartTagOperations` メソッドに次のコードを追加して、コンパイルします。

```
try{
 JspWriter out = pageContext.getOut();
 out.println("Hello World");
}
catch (Exception e){
 System.out.println(e);
}
```

6. Web モジュール配備記述子 (`web.xml`) へ `taglib` 要素を追加します。

   この操作を実行するには、`web.xml` のプロパティウィンドウを開き、「タグライブラリ」フィールドを選択し、「…」ボタンをクリックします。タグライブラリのプロパティエディタが表示されます。「追加」をクリックして、「追加 タグライブラリ」ダイアログを表示します。

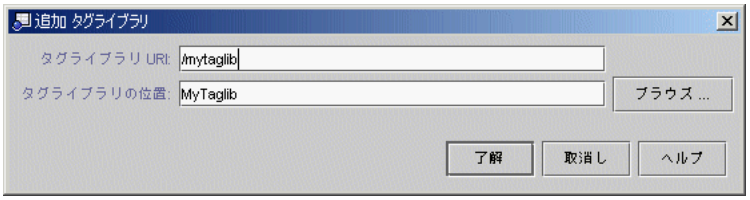

図 B-5　「追加 タグライブラリ」ダイアログ

7. タグライブラリの位置を URI `myTags` へマップします。

   この操作を実行するには、前のステップで作成したタグライブラリ要素に対するプロパティウィンドウを開きます。タグライブラリの位置フィールドを `/MyTagLib.tld` に設定し、タグライブラリ URI フィールドを `myTags` に設定します。

   この操作により、タグライブラリは、URI `myTags` を介して JSP ページにアクセス可能となります。

注 - タグライブラリの位置フィールド (`/MyTagLib.tld`) の前に付くスラッシュは、Web モジュールのルートを示します。

---

8. 新しい JSP ページを作成し、それを `TestCustomTag` と名付けます。

   この操作を実行するには、エクスプローラで Web モジュールのルートディレクトリを右クリックし、コンテキストメニューから、「新規」>「JSP & サーブレット」>「JSP」を選択します。テンプレートウィザードから新規作成の名前フィールドに **`TestCustomTag`** と入力します。「完了」をクリックします。

9. JSP ページの HTML `<body>` タグの後に、次のコードを追加します。

```
<%@taglib uri="myTags" prefix="mt" %><mt:HelloWorld />
```

10. サーバーを再起動し、JSP ページを実行します。

    この操作を実行するには、JSP ページのコンテキストメニューから「実行 (再読み込み強制)」を選択します。

    Web ブラウザは、`Hello World` を読み込むページを表示します。

タグハンドラへの属性の追加

次のチュートリアルは、Web ブラウザで表示される出力の色を制御する属性を「Hello World」タグへ追加する方法を示します。手順の一部として、タグハンドラ bean の再生成方法を学習します。

▼ タグハンドラに属性を追加する

1. `color` と名付けられた属性を `HelloWorld` タグへ追加します。

   この操作を実行するには、エクスプローラで `HelloWorld` タグを右クリックし、コンテキストメニューから「タグ属性を追加」を選択します。「新しいタグ属性を追加」ダイアログが開いたら、「名前」フィールドに **`color`** を入力し、「了解」をクリックします。

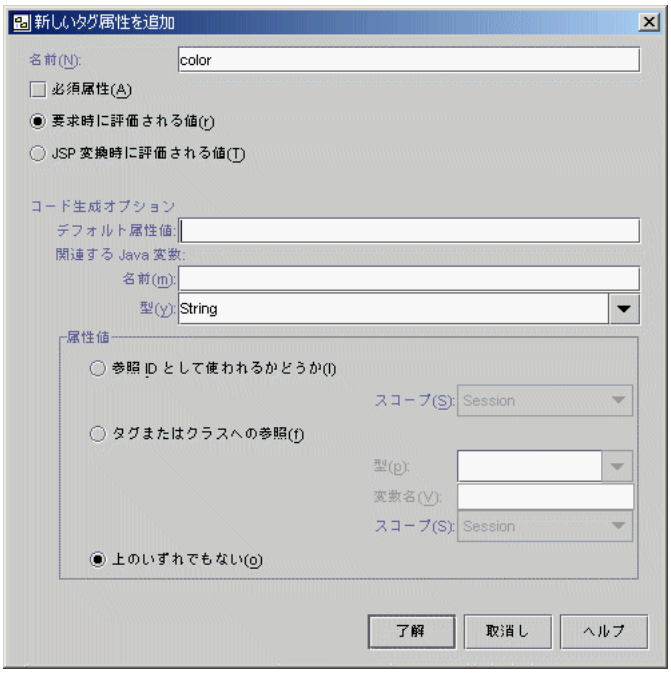

図 B-6　「新しいタグ属性を追加」ダイアログ

2. HelloWorld タグハンドラを再生成します。

   この操作を実行するには、エクスプローラで MyTagLib タブライブラリを右クリックし、コンテキストメニューから「タグハンドラを生成」を選択します。

   この操作により、タグハンドラに color と名付けられたプロパティ、および 3 つの対応するクラスメンバー (color と名付けられたフィールド、getColor および setColor と名付けられたメソッド) が生成されます。

3. 出力するテキストが color 属性に割り当てられた値に従って色付けされるよう、タグハンドラの otherDoStartTagOperations メソッドを変更します。

   この操作を実行するには、次のように、println 文を変更した後、クラスをコンパイルします。

```
out.println("<p><font color=" + getColor() + ">Hello World</font>");
```

4. 出力が赤になるよう、TestCustomTag JSP ページの HelloWorld アクションを変更します。

   この操作を実行するには、次のようにアクションを変更した後、JSP ページをコンパイルします。

```
<mt:HelloWorld color="red"/>
```

5. サーバーを再起動し、JSP ページを実行します。

   この操作を実行するには、JSP ページのコンテキストメニューから「実行 (再読み込み強制)」を選択します。

   Web ブラウザは、Hello World を赤で表示します。

タグライブラリのパッケージと JAR へのアクセス

次の節では、前の節で開発したタグライブラリをパッケージ化し、JSP ページから表示および編集する方法を示します。

▼ タグライブラリをパッケージ化し、アクセスする

1. MyTagLib TLD のコンテキストメニューから、「タグライブラリ JAR の作成」を選択します。

   この操作により、Web モジュールのルートディレクトリに、MyTagLib.jar と名付けられた JAR ファイルが作成されます (ファイル拡張子はエクスプローラでは表示されません)。

2. JAR ファイルのコンテキストメニューにある「カット」と「ペースト」を使用して、JAR ファイルを WEB-INF/lib ディレクトリへ移動します。

図 B-7　WEB-INF/lib ディレクトリ内の JAR ファイル

3. タグライブラリを開発した MyTagLib ディレクトリではなく、 MyTagLib JAR ファイルへマップするよう、配備記述子の Taglib 要素を変更します。

   この操作を実行するには、MyTagLib JAR ファイルを含む WEB-INF ディレクトリの web.xml ファイルを右クリックし、コンテキストメニューから「プロパティ」を選択します。「タグライブラリ」フィールドをクリックし「…」をクリックして、タグライブラリのプロパティエディタを表示します。myTags を選択し、「編集」ボタンをクリックします。次の図が示すように、「編集 タグライブラリ」ダイアログで、「タグライブラリの位置」フィールドに **/WEB-INF/lib/MyTagLib.jar** と入力し、「了解」をクリックし、タグライブラリのプロパティエディタで「了解」をクリックして、「web.xml のプロパティ」ウィンドウを閉じます。

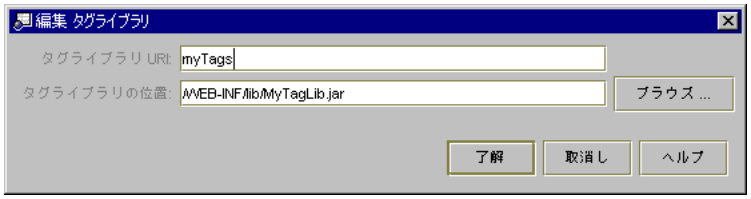

図 B-8　JAR ファイルへの taglib 要素のマッピング

注 - taglib 要素を、JAR ファイルとしてパッケージ化されたタグライブラリへマップする場合、タグライブラリ記述子の位置は指定しません。指定するのは、JAR ファイルの位置だけです。JAR ファイル内のタグライブラリ記述子の位置は、Web コンテナが知らせます。

---

4. サーバーを再起動し、JSP ページを実行します。

   この操作を実行するには、JSP ページのコンテキストメニューから「実行 (再読み込み強制)」を選択します。

   上記のチュートリアル同様、Web ブラウザに、赤で Hello World が表示されます。

用語集

Bean

JavaBeans 仕様に従って作成された再利用可能なソフトウェアコンポーネント。JavaBeans も参照してください。

EJB

Enterprise JavaBeans。オブジェクト指向、分散型、企業レベルのアプリケーションの開発および配備のためのコンポーネントアーキテクチャ。Enterprise JavaBeans アーキテクチャを使用して記述されるアプリケーションは、スケーラブルでトランザクションの処理効率も高く、マルチユーザーによる使用が可能で、なおかつ安全です。JavaBeans および Bean も参照してください。

HTTP

Hypertext Transfer Protocol。World Wide Web 上でファイル (テキスト、イメージ、音声、ビデオなど) の交換を制御するアプリケーションプロトコル。

HTTP モニター

サーブレットエンジン内の JSP ファイルおよび サーブレットファイルの実行に関するデータを収集するメカニズム。JSP オブジェクトまたは サーブレットオブジェクトに関連付けられたトランザクションごとに、モニターは着信要求、サーバーの上で維持されているデータ状態、およびサーブレット環境に関するデータを記録します。

J2EE

Java 2 Platform, Enterprise Edition。企業クラスのサーバー側アプリケーションを構築するために、包括的なアプリケーションプログラミングモデルと一連の互換性テストにより、多数の技術 (enterprise beans、JSP ページ、XML など) を単一のアーキテクチャに結合する Java 2 プラットフォームのエディション。EJB、JSPも参照してください。

J2EE アプリケーション

J2EE プラットフォーム上で動作する J2EE コンポーネント (アプリケーションクライアント、アプレット、HTML ページ、サーブレット、および enterprise beans) で構成されるアプリケーション。J2EE アプリケーションは、通常、複数のコンピューティング階層に分散して設計されます。配備のために、J2EE アプリケーションは、`.ear` (Enterprise Archive) ファイルにパッケージ化されます。J2EE も参照してください。

JAR

Java Archive file。クラス、イメージ、その他のファイルを単一の圧縮ファイルに含めるプラットフォーム独立型のファイル形式であり、ダウンロードを高速化します。

JavaBeans

移植性がありプラットフォーム独立型で再利用可能なコンポーネントモデルを定義するアーキテクチャ。Bean は、このモデルの基本単位です。Bean は、一般的なオペレーティングシステムのネットワークに配備できます。EJB も参照してください。

JDBC

Java Database Connectivity。Java プラットフォームと広範なデータベース間で独立した接続を確立するための業界規格。JDBC インタフェースは、SQL ベースのデータベースアクセスに対し呼び出しレベルの API を提供します。

JSP

Java Sever Pages。 テンプレートデータ、カスタム要素、スクリプト言語、およびクライアントに動的コンテンツを戻すサーバー側の Java オブジェクトを使用する拡張可能な Web 技術。通常、コンテンツは、HTML または XML 要素で構成され、多くの場合、Web ブラウザがクライアントになります。JSP 技術は、サーブレット技術を拡張したものです。静的な Web ペー

ジに対する動的なデータの追加を容易にします。JSP ページは、実行前に Web コンテナによってサーブレットへ動的に変換される、テキストベースの Web コンポーネントです。**サーブレット** も参照してください。

JSP タグライブラリ

動的なコンテンツやプロセスをカプセル化するタグの集合体。JSP ページ内のタグを介して、これらのコンテンツやプロセスを呼び出すことができます。JSP タグライブラリは JSP 仕様の一部で、J2EE 準拠のどのサーバーにも使用できます。**カスタムタグ** も参照してください。

MIME

Multipurpose Internet Mail Extensions。ビデオ、オーディオ、グラフィックスなど、テキスト以外の電子メール添付ファイルを送受信するためのインターネット規格。Web ブラウザは、MIME タイプを使用して、HTML 形式ではないファイルを解釈して表示するアプリケーションを割り当てます。

TLD

タグライブラリ記述子。タグライブラリを記述する XML ファイル。TLD ファイルは、そのタグライブラリを参照する `taglib` 指令を含むページを解釈するために JSP コンテナで使用されます。このファイルには、ライブラリのすべてのドキュメンテーションおよび各タグのドキュメンテーション、JSP コンテナやタグライブラリに関するバージョン情報、およびタグライブラリで定義されている各アクションに関する情報が入っています。TLD ファイルは、カスタムタグライブラリの作成時に生成されます。

TP

Transparent Persistence。Java オブジェクトとして格納されているデータの情報へのアクセスを可能にする Forte for Java IDE モジュール。このモジュールを使用すると、データベースプログラミングから Java プログラミングを切り離すことができます。

URI

Uniform Resource Indicator。ブラウザに表示される URL を構築するためにサーブレットの実行時 (またはデバッグ時) に使用されるプロパティ。URI の構文は次のとおりです。

`http://<web-server-name>:<web-server-port>/<web-module-URI>/<servlet-URI>&<servlet-query-params>`。

`<web-module-URI>` は、通常、コンテキストのルートとして示されます。

WAR

Web Archive file。拡張子 `.war` の付いた特殊な JAR ファイル。Web モジュールを J2EE に準拠したサーバーに配備するには WAR ファイルとしてパッケージ化する必要があります。JAR も参照してください。

Web コンテナ

Web アプリケーションの Web コンポーネントの実行をサポートする実行時サービスを提供します。Web コンテナは、Web サーバーからのクライアント要求をアプリケーションの Web コンポーネントへ転送し、そのクライアントに対する Web コンポーネントからの応答を Web サーバーへ転送します。Web コンテナは一般的に、Web サーバープロセス (Web サーバープラグインとして) または J2EE アプリケーションサーバープロセスで実行されます。

Web コンポーネント

WAR ファイルに含まれる実行可能ファイルであるサーバー側 J2EE コンポーネント。Web コンテナによって管理され、直接 Web コンテナと通信します。Web コンポーネントは、Web コンテナを介して HTTP 要求を受け取り、それらを処理し、Web コンテナを介して HTTP 応答を返すことができます。J2EE プラットフォームは、サーブレットと JSP ページの 2 種類の Web コンポーネントを定義します。JSP、サーブレット、および WAR も参照してください。

Web ブラウザ

ブラウザを参照してください。

Web モジュール

J2EE アプリケーションでの配備と使用が可能な Web リソースの最小単位です。Web モジュールは、Web アーカイブ (WAR) ファイルとしてパッケージ化し、配備することができます。Forte for Java IDE において、ともに配備される複数の Web モジュールは、Web モジュールグループと呼ばれます。WAR ファイルも参照してください。

カスタムタグ

外部ライブラリに含まれているフォーマット情報や処理ロジックを表す、ドキュメント内のテキスト要素。タグを使用すると、Java コードを JSP ページに取り込まないようにすることができます。**JSP タグライブラリ**も参照してください。

クライアント

クライアント/サーバー通信モデルにおいて、計算や格納領域など、リモートサーバーのリソースを要求するプロセス。**サーバー**も参照してください。

サーバー

リソースを管理し、クライアントにサービスを提供するネットワークデバイス。**クライアント**も参照してください。

サーブレット

`javax.servlet` を実装するクラスのことで、通常は `javax.servlet.http.HttpServlet` のサブクラスです。サーブレットは、Web コンテナ内で実行され、Web サーバーおよび Web 対応のアプリケーションサーバーの機能を拡大するために使用されます。サーブレットは、HTML フォームで作成された要求に応答してデータソースから動的コンテンツを生成し、特定の Web リソースへのアクセスを有効/無効にしてアプリケーションフローを制御し、ユーザーセッション (たとえば、ユーザーのショッピングカートからのアイテムの追加/削除など) を追跡します。

持続可能クラス

持続的データストアからのデータを含み、SQL またはデータストア固有のコードを必要としないクラス。ビジネスロジックをこれらの Java プログラミング言語クラスに挿入するには、追加のメソッドを定義し、自動生成されるメソッドを拡張します。

スクリプトレット

すべての有効な Java コードを入力可能にするスクリプト要素。宣言要素で宣言された変数とメソッドは、同じ JSP ページのスクリプトレットへ提供されます。

タグ

カスタムタグを参照してください。

配備

ソフトウェアを操作環境にインストールするプロセス。

配備記述子

Web モジュールの配備方法を記述する XML ファイル。この記述子は、JSP ページとサーブレットの `init` パラメータ、サーブレットラッパー、および JSP ページに利用できるカスタムタグライブラリなどの Web モジュールを構成するコンポーネントを記述します。

ブラウザ

HTML ドキュメントおよびアプレットの表示、ナビゲート、対話式操作をするためのアプリケーション。Web ブラウザとも呼ばれます。

索引

L

M

P

R

S

T

U

よ