Web コンポーネントのプログラミング

Sun™ ONE Studio 5 プログラミングシリーズ

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No. 817-3291-10
2003 年 7 月, Revision A

原典： *Building Web Components*
Part No: 817-2334-10
Revision A

目次

はじめに

このマニュアルは、Java™ 2 Platform, Enterprise Edition (J2EE™) の Web コンポーネントを使用して Web アプリケーションの作成に携わる方すべてにとって非常に重要な情報を説明しています。このマニュアルは、Sun™ ONE Studio 5 プログラミングシリーズを構成しています。このマニュアルでは J2EE およびそのサポートテクノロジの元での Web アプリケーションの開発を重点的に取り上げます。サポートテクノロジとは Java サーブレットおよび JavaServer Pages™ (JSP™) などです。

このマニュアルでは Web アプリケーションを概説し、その構造について提案します。また、Web アプリケーション開発の作業の流れについて説明します。さらに設計の実践を提案し、スケーラブルで保守の容易な Web アプリケーションのための構造とフレームワークを提示します。JSP ページとサーブレットの作成、コーディング、テスト、デバッグ、および配備について議論しながら、Sun ONE Studio 5 統合開発環境 (IDE) の中にこうした概念をあてはめていきます。

このマニュアルではとくに、JSP ページ、Java サーブレット、JSP タグライブラリ、サポートするクラスおよびファイルについて Web アプリケーションによる典型的な使用方法を説明します。Web アプリケーションは持続的データ、たとえばデータベースを使用することがあります。これらは Web コンテナが管理する機能を含む独立したアプリケーションであってもかまいません。あるいは、J2EE Enterprise JavaBeans (EJB™) コンテナの中のコンポーネントに依存しながら他のサービスに対しユーザーインタフェースを提供することもあります。こうしたサービスには、ビジネスロジックの実行や持続的データへのアクセスが含まれる場合があります。

このマニュアルで説明しているプログラム例は、実際に作成することができます。作業環境については、以下の Web サイトにあるリリースノートを参照してください。

`http://sun.co.jp/software/sundev/jde/documentation`

使用するプラットフォームによっては、このマニュアルに掲載している画面イメージと異なることがあります。ほとんどの手順で Sun ONE Studio 5 ソフトウェアのユーザーインタフェースを使用しますが、場合によっては、コマンド行にコマンドを入力する必要があります。その場合は、Microsoft Windows の「コマンドプロンプトウィンドウ」で次の構文を入力します。

```
c:\>cd MyWorkSpace\MyPackage
```

UNIX プラットフォームまたは Linux プラットフォームの場合は、次のように入力します。

```
% cd MyWorkSpace/MyPackage
```

お読みになる前に

このマニュアルでは、Web アプリケーション開発者および Web アプリケーション設計者を読者として想定しています。Web アプリケーション開発者とは、アプリケーションコードを書く人です。Web アプリケーション設計者とは、ユーザーによるアプリケーションの操作方法を指定し、またインタフェースコンポーネントを選択してそれをさまざまなビューに整理する人です。とくに明記しない限り、「Web アプリケーション」という用語は 「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が Web アプリケーション設計者も兼ねる場合とそうでない場合とがあります。いずれの場合も、読者が Java プログラミング、JSP ページのプログラミング、および HTML のコーディングについて一般的な知識をもっていることを前提としています。このマニュアルの内容は、Web コンポーネントに基づくアプリケーションの作成に関わる専門家にとっても有用でしょう。専門家とはテクニカルライター、グラフィックアーティスト、生産やマーケティングのスペシャリスト、テスターなどです。このマニュアルでは IDE を使用して Web アプリケーション開発のワークフローを簡単にする方法を紹介します。そしてこの生産性向上のためのツールを使う状況を示します。

Web アプリケーションの開発は、従来の Java アプリケーションの開発とは明らかに違います。Web アプリケーションの開発には、次のようなさまざまなテクノロジを理解していることが必要です。このマニュアルではこれらの知識があることを前提としています。

- HTML 構文
- XML 構文
- Java™ プログラミング言語
- Java Servlet 構文
- JavaServer Pages™ 構文
- HTTP プロトコル
- Web サーバーの概念
- セキュリティの問題

このマニュアルは読者の方が現在のスキルを活用できるようにします。さらに IDE を使用した Web アプリケーションの構築を生産的なものにするのに役立つ参考資料を紹介します。

また、次に示すような J2EE の概念に関する一般的な知識が必要です。

- Java 2 Platform, Enterprise Edition Blueprints
  `http://java.sun.com/j2ee/blueprints`
- Java 2 Platform, Enterprise Edition Specification
  `http://java.sun.com/j2ee/download.html#platformspec`
- The J2EE Tutorial
  `http://java.sun.com/j2ee/tutorial`
- Java Servlet Specification Version 2.3
  `http://java.sun.com/products/servlet/download.html#specs`
- JavaServer Pages Specification Version 1.2
  `http://java.sun.com/products/jsp/download.html#specs`
- JavaServer Pages Standard Tag Library 1.0 Specification
  `http://www.jcp.org/aboutJava/communityprocess/review/jsr052/index.html`

さらに、Java API for XML-Based RPC (JAX-RPC) の詳細に精通していると役立ちます。詳細は、以下を参照してください。

`http://java.sun.com/xml/jaxrpc`

『Sun ONE Studio 5, Web アプリケーションチュートリアル』は「Developer Resources」Web サイトからダウンロードできます。このチュートリアルにアクセスするには、IDE のメインメニューで「ヘルプ」->「学習」->「例 / チュートリアル」の順に選択します。

『J2EE Tutorial』には Web アプリケーションの開発プロセスについて説明があります。次の Web サイトをご覧下さい。

```
http://java.sun.com/j2ee/tutorial/
```

Web Services Developer パックの「Web Apps」チュートリアルには背景の説明があり、有用です。このマニュアルをお読みになる際の参考として役立つはずです。次の Web サイトからこのチュートリアルをダウンロードできます。

```
http://jdc.sun.co.jp/j2ee/tutorial/
```

注 - Sun では、本マニュアルに掲載した第三者の Web サイトのご利用に関しましては責任はなく、保証するものでもありません。また、これらのサイトあるいはリソースに関する、あるいはこれらのサイト、リソースから利用可能であるコンテンツ、広告、製品、あるいは資料に関して一切の責任を負いません。Sun は、これらのサイトあるいはリソースに関する、あるいはこれらのサイトから利用可能であるコンテンツ、製品、サービスのご利用あるいは信頼によって、あるいはそれに関連して発生するいかなる損害、損失、申し立てに対する一切の責任を負いません。

内容の紹介

このマニュアルには以下の内容が含まれています。

第 1 章は Web アプリケーションを概説し、Web アプリケーションの開発中に開発者が出会う問題を説明します。また、IDE を利用してこうした問題に対処する方法を説明します。

第 2 章は Web アプリケーションの構造について説明します。Web アプリケーションのコンポーネント構築で使用される J2EE の主要テクノロジの概要が含まれます。

第 3 章は Web アプリケーションの便利なデザインパターンとフレームワークについて概説します。

第 4 章は IDE を使用して Web アプリケーションを開発する方法について記述します。

第 5 章は IDE を使用して Web アプリケーションを実行、デバッグ、および配備する方法を詳しく説明します。

用語集は、 このマニュアルで使用されている重要な用語を定義します。マニュアル中の用語集で取り上げられている用語は、ゴシック体で表記されています。

このマニュアルで取り上げないこと

このマニュアルは IDE を生産性向上のツールとして利用できるようにするために十分な情報を提供することを目的としています。けれども、チュートリアルとして書かれてはいません。このマニュアルは包括的な参考書でも、Web アプリケーションのあらゆる設計を取り扱うものでもありませんし、ビジュアルなデザインガイドでもありません。また、J2EE Web 層の開発を重点的に取り上げたり、EJB コンポーネントの開発方法を詳述したりはしていません。推奨する参考資料については、xiii ページの「参考資料」を参照してください。Web アプリケーションの開発に関するチュートリアルの参考情報については、x ページの「お読みになる前に」を参照してください。

参考資料

ここではこのマニュアルを読む際に役立つ参考資料を紹介します。

次の本は Web アプリケーションの設計と導入に関するテーマを取り上げたものです。

『J2EE パターン : 明暗を分ける設計の戦略』中野明彦・佐野祐一郎・宮田泰宏・土屋聡一朗 共著、ピアソン・エデュケーション、2002 年 (原書 : Prentice Hall「Core J2EE Patterns」Deepak.Alur / John.Crupi / Dan.Malks)。この本は、Web アプリケーションのアーキテクチャとモデルに関する優れた書物であり、コンテキストにおいて発生する問題の解決方法を、J2EE ベースによる答えも含めて示します。Java 技術者および Sun Java Center による経験を収集し、それを反映しています。

有用な Web サイト

Web アプリケーションテクノロジを取り扱った Web サイトを以下に紹介します。

- 「The Source for Java Technology」には Web コンポーネントテクノロジに関する情報が豊富にあります。製品、API、Developer Connection へのアクセス、ドキュメンテーションとトレーニング、オンラインサポートなどのテーマを取り上げています。また、コミュニティの議論、業界ニュース、マーケットプレースのためのソリューション、ケーススタディなどもあります。
  `http://java.sun.com`
- 「JSP Insider」は JavaServer Page の Web サイトで、設計に関する情報や論説、コード、他の Web サイトへのリンク、ニュース記事、書評が掲載されています。
  `http://www.jspinsider.com`
- 「JSP Resource Index」はチュートリアル、スクリプト、また求人情報を捜すサイトです。
  `http://www.JSPin.com`
- 「Jakarta Project」は、公開し共同作業で作る方法で開発した、商用レベルの品質の Java プラットフォームベースのサーバーソリューションを提供します。Jakarta とは多数のサブプロジェクトをまとめたプロジェクト名であり、Jakarta タグライブラリや Tomcat サーバーもこれに含まれます。
  `http://jakarta.apache.org`
- 「TheServerSide.com」 は、J2EE のニュースソースであり、開発者のコミュニティです。
  `http://www.theserverside.com`

書体と記号について

書体または記号	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、コーディング例。	.cvspass ファイルを編集します。 DIR を使用してすべてのファイルを表示します。 Search is complete.
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表わします。	> **login** Password:
AaBbCc123 または **ゴシック**	コマンド行の可変部分。実際の名前または実際の値と置き換えてください。	削除するには **DEL** *filename* と入力します。 rm **ファイル名** と入します。
『 』	参照する書名を示します。	『Solaris ユーザーマニュアル』
「 」	参照する章、節、または、強調する語を示します。	第 6 章「データの管理」を参照してください。 これらは、「クラス」オプションと呼ばれます。
\	枠で囲まれたコード例で、テキストがページ行幅を越える場合、バックスラッシュは、継続を示します。	machinename% grep '^#define \ XV_VERSION_STRING'
➤	階層メニューのサブメニューを選択することを示します。	作成: 「返信」➤「送信者へ」

シェルプロンプトについて

シェル	プロンプト
UNIX の C シェル	`machine_name%`
UNIX の Bourne シェルと Korn シェル	`machine_name$`
スーパーユーザー (シェルの種類を問わない)	#

関連マニュアル

Sun ONE Studio 5 のマニュアルは、Acrobat Reader (PDF) ファイル、リリースノート、オンラインヘルプ、サンプルアプリケーションの `readme` ファイル、Javadoc™ 文書の形式で提供しています。

オンラインで入手可能なマニュアル

以下に紹介するマニュアルは、Sun ONE Studio 開発者リソースポータルのドキュメントサイト (http://sun.co.jp/software/sundev/jde/documentation/) および `docs.sun.com`™ (http://docs.sun.com) から入手できます。

`docs.sun.com` Web サイトでは、サンのマニュアルをインターネットを通じて閲覧、印刷、購入することができます。サイト内でマニュアルを見つけられない場合には、製品と一緒にローカルシステムまたはローカルネットワークにインストールされているマニュアルインデックスを参照してください。

- リリースノート (HTML 形式)

  Sun ONE Studio 5 の Edition ごとに用意されています。このリリースでの変更情報と技術上の注意事項を説明しています。

  - 『Sun ONE Studio 5, Standard Edition リリースノート』

- インストールガイド (PDF 形式)

  対応プラットフォームへの Sun ONE Studio 5 統合開発環境 (IDE) のインストール手順を説明しています。さらに、システム要件、アップグレード方法、アプリケーションサーバーの情報、コマンド行での操作、インストールされるサブディレクトリ、Javadoc の設定、データベースの統合、アップデートセンターの使用方法などが含まれます。

  - 『Sun ONE Studio 5, Standard Edition インストールガイド』
  - 『Sun ONE Studio 4, Mobile Edition インストールガイド』

- Sun ONE Studio 5 プログラミングシリーズ (PDF 形式)

  Sun ONE Studio 5 の各機能を使用して、優れた J2EE アプリケーションを開発するための方法を詳細に説明しています。

  - 『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、タグライブラリを使用し、クラスやファイルをサポートする Web アプリケーションを J2EE Web モジュールとして構築する方法を説明しています。

  - 『J2EE アプリケーションのプログラミング』

    EJB モジュールや Web モジュールを J2EE にアセンブルする方法を説明しています。また、J2EE アプリケーションの配備や実行についても説明しています。

  - 『Enterprise JavaBeans コンポーネントのプログラミング』

    Sun ONE Studio 5 の EJB ビルダーウィザードや、他の IDE コンポーネントを使用し、EJB コンポーネント (コンテナ管理や Bean 管理の持続性の機能を持つセッション Bean やエンティティ Bean、メッセージ駆動型 Bean) を作成する方法を説明しています。

  - 『Web サービスのプログラミング』

    Sun ONE Studio 5 IDE を使用して Web サービスを構築したり、UDDI レジストリを経由して第三者に Web サービスを利用させたり、また、ローカル Web サービスや UDDI レジストリから Web サービスクライアントを生成する方法などを説明しています。

  - 『Java DataBase Connectivity の使用』

    Sun ONE Studio 5 IDE の JDBC 生産性向上ツールを使用し、JDBC アプリケーションを作成する方法について説明しています。

- Sun ONE Studio 5 チュートリアル (PDF 形式)

  Sun ONE Studio 5, Standard Edition の主な機能の活用方法を紹介しています。

  - 『Sun ONE Studio 5 Web アプリケーションチュートリアル』

    簡単な J2EE Web アプリケーションの構築方法を順を追って解説します。

  - 『Sun ONE Studio 5 J2EE アプリケーションチュートリアル』

    EJB コンポーネントと Web サービス技術を使用したアプリケーションの構築方法を順を追って解説します。

  - 『Sun ONE Studio 4, Mobile Edition チュートリアル』

    携帯やPDA 端末などの無線機器を対象とした簡単なアプリケーションの構築方法を順を追って解説します。このアプリケーションは Java 2 Platform, Micro Edition (J2ME™ プラットフォーム) に準拠し、Mobile Information Device Profile (MIDP) と Connected, Limited Device Configuration (CLDC) を満たすものです。

  チュートリアルアプリケーションは、以下のサイトからもアクセスできます。
  `http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`

オンラインヘルプ

オンラインヘルプは、Sun ONE Studio 5 IDE から参照できます。ヘルプを開くには、ヘルプキー (Windows および Linux 環境では F1 キー、Solaris オペレーティング環境では Help 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。

プログラム例

Sun ONE Studio 5 の機能を紹介したプログラム例とチュートリアルアプリケーションを、以下の Sun ONE Studio 開発者リソースのポータルサイトからダウンロードすることができます。

`http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`

このチュートリアルで使用するアプリケーションも上記サイトに収録されています。

Javadoc

Javadoc 形式のマニュアルは、Sun ONE Studio 5 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。

技術サポートへの問い合わせ

製品についての技術的なご質問がございましたら、以下のサイトからお問い合わせください (このマニュアルで回答されていないものに限ります)。

`http://sun.co.jp/service/contacting`

第1章

Web アプリケーションの開発における問題への取り組み

この章では、Web アプリケーションについて概説し、従来の開発で使用されていたスタンドアロンのデスクトップアプリケーションとの違いについて説明します。また、Sun ONE Studio 5 IDE を使用してこれらのアプリケーションを構築する方法についても説明します。

Web アプリケーションとは

Web アプリケーションは複数の Web コンポーネントの集合です。通常、Web アプリケーションは **Web ブラウザ**に表示されるインタフェースを介してエンドユーザに機能を提供します。Web アプリケーションの例には、電子ショッピングやオークションサイトなどがあります。Web アプリケーションはクライアントサーバーモデルに基づいています。このモデルでは、**クライアント**は Web ブラウザで、**Web サーバー**はリモートで動作するフィーチャセットです。

Web アプリケーションの最も簡単な形式では、ブラウザがクライアントとなります。ブラウザは、Web サーバーに対して要求を送ります。Web サーバーは要求を受け取ると、この要求をサーブレットコンテナ内で動作する Web アプリケーションに渡します。Web アプリケーション内の JSP とサーブレットが要求を処理して応答を生成します。サーバーは、ブラウザに応答を送ります。

図 1-1 に、フィルタ、サーブレット、JSP ページに対する応答の関係、およびサンプル Web アプリケーションのサーブレットと JSP ページに対するセッションデータの関係を示します。このサンプルアプリケーションはフロントコントローラを使用します。フロントコントローラについては第 3 章を参照してください。

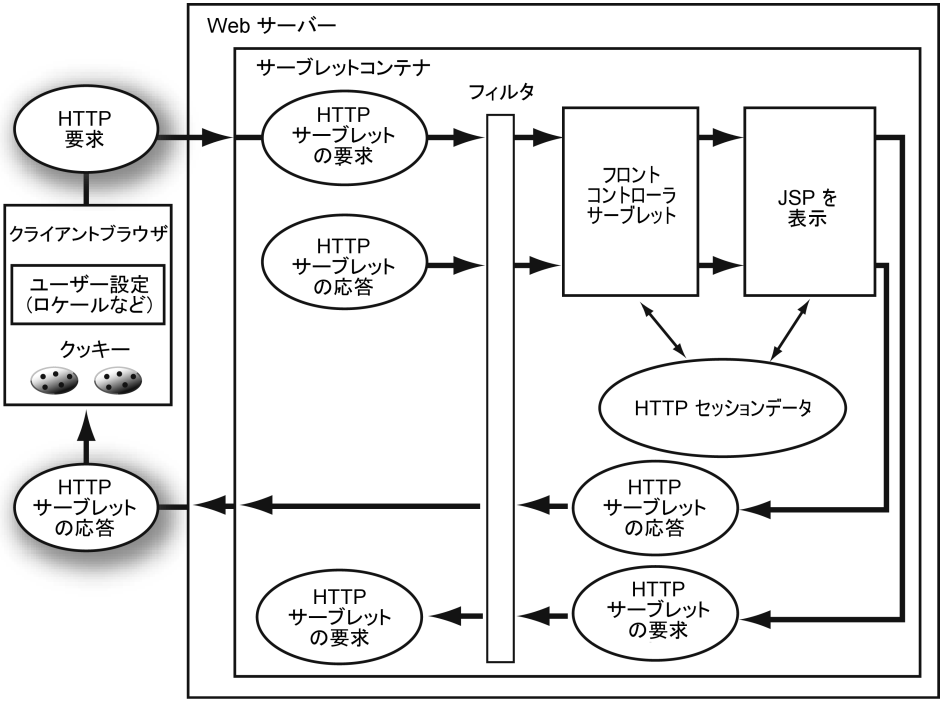

図 1-1　　Web アプリケーションでの要求と応答

Web アプリケーションは相互に関連するWeb コンポーネントから構成されます。これには、ともに機能する JSP ページ、サーブレット、およびタグライブラリが含まれています。Web アプリケーションで使用される複数のリソースは、配備記述子ファイルを使用して連携させる必要があります。この配備記述子ファイルには、Web アプリケーションを説明するメタ情報が含まれています。**サーブレットコンテナ**は JSP ファイルをサーブレットに変換し、Web サーバーで実行できるようにします。このコンポーネントについての詳細は、 第 2 章を参照してください。

Web アプリケーションの開発における対処

Web アプリケーションは、主に次の 2 つの点においてスタンドアロンアプリケーションと大きな違いがあります。

- Web コンポーネント間では直接対話せず、 サーブレットコンテナとクライアントブラウザがコンポーネント間のリンクを仲介します。

- データ表示、データフロー、データ処理は、クライアントのブラウザ、サーブレットコンテナ、およびそれぞれの Web コンポーネントに分散されます。Web コンポーネントが相互リンクを維持してデータを共有することを可能にする仕組みについては、このマニュアル全体で説明されています。

図 1-1 に示すように、Web サーバーが処理の中心となります。情報の格納やデータベースへのアクセスは、すべてサーバーで行われます。

プロジェクトを Web アプリケーションとして実装すると、配備と保守が簡単になります。アプリケーションはサーバー上に存在するため、ユーザーは自身のコンピュータにソフトウェアをインストールする必要はありません。また、アップグレードは全クライアント上で実行するのではなく、サーバー管理者によって一ヵ所で行われます。

また、Web アプリケーションのアーキテクチャでは、再利用可能なコンポーネントをより簡単に作成することができます。Web アプリケーションのコンポーネントは、スタンドアロンのデスクトップアプリケーションのコンポーネントに比べてゆるやかに結合されています。

Web アプリケーションの開発の相違点

スタンドアロンのアプリケーションでは、コンポーネントは互いに直接対話できます。たとえば、データオブジェクトはそれぞれ 1 つの型を持っており、それがメソッド間で渡されます。また、データ表示、データフロー、およびデータ処理はすべてアプリケーション自身が管理します。たとえば 1 つのウィザードの中では、それぞれの区画はアプリケーションの一部であり、アプリケーションデータへ直接アクセスすることができます。開発者は、データおよびアプリケーションの機能に集中するだけで、実行時の環境、つまり仮想マシンについては理解する必要はありません。

これに対して、Web アプリケーションでは、スタンドアロンの場合よりも Web コンポーネントがさらにカプセル化されています。Web コンポーネントは、サーブレットコンテナを介して互いに対話します。データは文字列として渡されます。このため、情報は、クライアントのブラウザ、Web サーバー、およびサーブレットコンテナの協同プロセスの間で安全に渡されます。Web アプリケーションを作成する場合は、開発者はさまざまな実行環境について理解する必要があります。テストは、複数の Web サーバーのブラウザで繰り返して行います。実際のファイルの場所と構造は、Web アプリケーションを配備するための手順によって異なります。通常、Web アプリケーションをサーバーに配備して実行する処理には、時間がかかります。

サーバーでの実行の一元化の意味

Web アプリケーションにおける実行は、すべてクライアントブラウザではなくサーバーで行われます。そのため、Web アプリケーションはこの特徴に対処するために、HTTP プロトコルを利用しています。

図 1-1 で示したように、データ表示、データフロー、データ処理は、ブラウザ、サーブレットコンテナ、およびそれぞれの Web コンポーネントに分散されます。このような場合に、コンパイル時間、実行時間、およびタスクのデバッグにどのような影響を及ぼすかについては、7 ページの「Web アプリケーションの一般的なエラー」を参照してください。

Web サーバーが分散している状況では、複数のマシン上で作動している Web サーバーに、サーブレットコンテナを分散させる必要があります。また、さまざまなサーバーインスタンスに遭遇する 1 つのクライアントブラウザの複数の要求を処理できるよう、セッション情報を共有する必要があります。

IDE によるサポート

生産性を向上させて Web アプリケーションの実装でメリットを得るために、IDE は、Web アプリケーション開発者特有の問題に対処しています。この節では、IDE が提供する完全な Web コンポーネントのサポート、実行のサポート、デバッグと監視のツール、配備のサポート、およびオープンな実行環境について説明します。

完全な Web コンポーネントのサポート

IDE には、Web コンポーネントのフルセットをサポートするために次の特徴があります。

- **JSP ページ、サーブレット、タグライブラリ、およびタグライブラリのハンドラ用のエディタ。**これらの機能には、構文の色別表示とチェック機能、コードの補完、およびコンパイル機能を持ったソースエディタも含まれています。詳細については、49 ページの「JSP ファイルの変更」、56 ページの「サーブレットの変更」および 62 ページの「タグライブラリから JSP ページへのカスタムアクションの挿入」を参照してください。
- **配備記述子ファイルの自動編集と手動編集。**配備記述子は `web.xml` ファイルとも呼ばれます。詳細については、78 ページの「プロパティシートを使用して `web.xml` ファイルを編集する」および 80 ページの「IDE を使用してソースエディタで `web.xml` ファイルを編集する」を参照してください。
- **Web モジュール構造内での開発。**実行するすべてのタスクは、「Servlet 2.3 Specification」に記載されているコンテキストで実行されます。詳細については、45 ページの「IDE での Web モジュール」を参照してください。
- **JSP ページ用に生成されたサーブレットコードの表示。**サーブレットのデバッグ中に、JSP ページコードとサーブレットコードの現在の位置を参照できます。詳細については、94 ページの「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」を参照してください。

- **タグライブラリの作成が容易。**詳細については、64 ページの「タグライブラリおよびタグの作成」を参照してください。
- **既存のタグライブラリを Web アプリケーションに組み込むためのサポート。**詳細については、61 ページの「既存のタグライブラリの使用」を参照してください。

実行のサポート

IDE は、次の方法で Web アプリケーションの実行をサポートします。

- Web アプリケーションを構築、配備、および実行するためのワンクリック実行
- さまざまなクライアントブラウザを使用したテスト対話
- さまざまなサーバーでのテスト実行

詳細については、81 ページの「Web モジュールのテスト実行」および 85 ページの「外部サーバーでの実行」を参照してください。

デバッグツールと監視ツール

Web アプリケーションのデータ管理は、アプリケーションのコンポーネント間に分散されます。コンパイル時に検出された多数のバグを Web アプリケーションでは実行時に追跡する必要があります。このため、スタンドアロンアプリケーションにおいては Web アプリケーションをデバッグするときには、実行時のテストにより重点をおくことになります。IDE には、JSP ページ、サーブレット、およびヘルパークラスに対するソースレベルのデバッグ機能があります。Web サーバーで受け取った HTTP 要求と生成した HTTP 要求がデバッグ対象となります。これらの機能によって、Web アプリケーションのデバッグと監視をサポートします。詳細については、85 ページの「Web アプリケーションのデバッグ」を参照してください。

Web アプリケーションはコンポーネント間でデータを共有するのに、従来のアプリケーションとは異なる方法を使用します。Web アプリケーションのコンポーネント (JSP やサーブレットなど)は、互いに直接通信することはできません。その代わりにコンポーネントは、ServletContext、ServletRequest、HTTPSession、および PageContext オブジェクトとともに格納されている属性 (名前/値の組み合わせ)を使ってほかのコンポーネントとデータを共有します。

図 1-1 に示したように、データは、クライアントブラウザから Web アプリケーションへ HTTP 要求として送信されます。次に、HTTP 応答を介して情報が返されます。要求がクライアントからサーブレットコンテナに挿入されるときに、応答が作成されます。要求と応答は、その要求の処理が完了するまで有効になります。

HTTP 要求は、クライアントブラウザで作成されるメッセージです。この要求には、ロケール情報やクライアントブラウザのクッキーなどのユーザー設定が付いたヘッダーが含まれています。Web サーバーは、要求をサーブレットコンテナへ渡します。図 1-1 に示されたサンプル Web アプリケーションの場合は、要求はフィルタ、

フロントコントローラサーブレット、JSP ファイルを介し、そして再度フィルタを介して渡されます。これらの場所で、その他の情報を取得することも、(可能な場合は) 修正することもできます。JSP ファイルとフィルタの処理が完了すると、HTTP 要求が完了します。フロントコントローラの詳細については、34 ページの「フロントコントローラ」を参照してください。

HTTP 応答は、サーブレットコンテナで作成されるメッセージです。このメッセージには、クッキー、ヘッダー、出力など、最終的にはクライアントブラウザへ返される属性が含まれています。応答処理は、フロントコントローラサーブレット、JSP ファイル、およびフィルタでも発生します。要求の情報にアクセスし、(可能な場合は) 修正することができます。また、応答に情報を追加することもできます。要求が完了すると、Web サーバーは、サーブレットコンテナから Web サーバーを介して応答を渡し、最終的にはクライアントブラウザへ渡します。詳細については、56 ページの「サーブレットによって生成された HTTP 応答」を参照してください。

特定のクライアントからの複数の要求は、互いに関連付けることができます。このような関連付けは、**HTTP セッション**を使用して行います。HTTP セッションは、クライアントブラウザと Web サーバー間の複数の要求を対象とした会話です。各セッションには、各自のセッションデータがあります。JSP ファイルとサーブレットは、要求と応答の情報だけでなく、セッションデータにもアクセスできます。セッションデータはセッションスコープ内にあります。つまり、セッションに関連付けられた要求をコンポーネントが処理する場合のみ、コンポーネントはセッションのデータにアクセスできます。詳細については、21 ページの「HTTP セッション」を参照してください。

サーブレットフィルタは Web コンポーネントの一種で、これを使用して、要求されたサーブレットまたは JSP ファイルを処理する前後に、HTTPRequest と HTTPResponse オブジェクトを検査および修正することができます。フィルタの詳しい使用方法については、60 ページの「HTTP 要求および応答の処理」を参照してください。

従来のアプリケーションでは、オブジェクト間のインタフェースに対応するためにソースレベルのデバッグを使用しました。このテクニックは、Web アプリケーションの開発でも有効です。ただし、コンポーネントのインタフェースレベルで機能するデバッグツールも必要になります。デバッグツールでは、クライアントと Web サーバー間で HTTP メッセージが渡され、コンポーネントとエラーが分離されます。通常は、コード検査によってここでエラーを検出できます。詳細については、7 ページの「Web アプリケーションの一般的なエラー」を参照してください。HTTP 要求を追跡するための IDE のツールの詳細は、88 ページの「HTTP モニターの起動」を参照してください。

配備のサポート

Web アプリケーションは複数のリソースで構成されているため、配備記述子にはさまざまな Web コンポーネントが登録され、関連付けられています。前述のように、Web アプリケーションのリソースを連係させる処理のほとんどは、IDE で自動的に行われます。

IDE には、次の機能を持つ配備のサポートが用意されています。

- **配備記述子の編集。**Web アプリケーションは、プロパティベースのエディタを使用するか、または XML ファイルを直接編集して関連付けし、登録する必要があります。詳細については、76 ページの「Web モジュール配備記述子の構成」を参照してください。
- **WAR ファイルのパッケージ作成。**J2EE に準拠した任意のサーバーに配備できるよう、リソースをパッケージ化できます。詳細については、94 ページの「Web アプリケーションのパッケージ化」を参照してください。

オープンな実行環境の統合

IDE は、複数のサーバーで Web アプリケーションを実行するためのサポートを提供しています。具体的には、API ベースのメカニズムを使用して IDE のソースから直接配備を行い、サードパーティ製のサーバーとの統合を実現しています。詳細については、81 ページの「Web モジュールのテスト実行」および 85 ページの「外部サーバーでの実行」を参照してください。

Web アプリケーションの一般的なエラー

Web アプリケーションでは、問題は、適切なデータを取得しないコンポーネントから発生します。また、要求が特定の状態のサーバーに挿入されない場合にも問題が発生します。Web アプリケーションの一般的なソースエラーには、コーディングのエラーと配備のエラーも含まれます。

コーディングエラーには、一般的に次のものがあります。

- リンク名のミススペル
- パラメータ名または値のミススペルまたは欠落
- 不正なクッキー値またはドメイン
- 初期化されていないセッション属性のミススペル
- クライアントから送信されない予期されたエラー
- 初期化パラメータの欠落またはミススペル

配備エラーには、一般的に次のものがあります。

- 配備されていないコンポーネント
- 配備されていないライブラリ

- ミススペル、または配備記述子から除外されたサーブレット URL マッピング

Web アプリケーションデータは、ServletContext、ServletRequest、HTTPSession、および PageContext オブジェクトとともに格納されている属性 (名前/値の組み合わせ)を使って設定する必要があります。このため、コンポーネント間でデータをやりとりするときにエラーが発生することがあります。Web アプリケーションのコンポーネント内でのプログラミングロジックに対応するには、ソースレベルのデバッグが便利です。ただし、ソースレベルのデバッグでは、コンポーネント間のデータのやりとりを参照できません。たとえば、あるコンポーネントが title 属性の値を設定して、別のコンポーネントが title 属性の値を取得しようとすることがあります。このタイプのエラーは、コンパイル時には検出されません。

エラーによって実行時に例外が発生した場合は、原因を見つけるためにブレークポイントを設定する場所を容易に決定できます。ただし、空白ページやデータの欠落といった問題が発生した場合は、問題の原因となっているコンポーネントを容易に見極めることができるとは限りません。問題は、フロントコントローラがデータを受け取っていないということや、フロントコントローラがデータを適切に渡していない、といったことかも知れません。また、セッションで必要なデータが指定されていない、といった理由も考えられます。問題の原因となるコンポーネントを判断できない場合は、ソースレベルのデバッグは行き当たりばったりのやり方になります。問題の原因を突き止めるには、要求を追跡してセッション属性をチェックすることが必要です。

Web アプリケーションのエラー個所を特定するために、IDE のデバッグツールを使って次の処理を行うことができます。

- コンポーネント間でのデータフローを追跡する
- 特定の状態でサーバーまたはセッションに入る
- Web リソースの生成された出力を 1 行ずつステップ実行する

Web アプリケーションのデバッグにおける対処

Web アプリケーションのデバッグには、データを共有しているコンポーネントの識別、コードのデバッグ、および配備プロセスとの連携などが含まれます。IDE には、これらの処理をサポートするために次の機能が用意されています。

- **HTTP モニター。**要求に関する情報を、Web アプリケーションおよびそのコンポーネントに記録します。詳細については、88 ページの「HTTP モニターの起動」を参照してください。
- **ソースコードデバッガ。**コンポーネントのデバッグを行います。指定されたコンポーネントが JSP ページの場合は、JSP ファイル、および生成されたサーブレットコードを並べて表示することができます。この表示機能によって、サーバーで行われるプロセスがわかりやすくなります。詳細については、91 ページの「ソースレベルのデバッグ」を参照してください。

- **アプリケーションを制御し、さまざまな Web サーバーへ配備するための一般的なインタフェース。**IDE における配備の識別の問題をサポートします。85 ページの「外部サーバーでの実行」を参照してください。

HTTP モニターによる要求の表示

組み込み **HTTP モニター**を使用して、現在の要求を表示し、後で再生するためにこれらを記録することができます。現在の要求は現在の IDE セッションが終了するまで有効です。保存されている要求は、ユーザーが明示的にその要求を削除するまで IDE のセッション間で有効になります。

HTTP モニターを使用して、データフローのシーケンス内でエラーが発生した場所を特定できます。多くの場合には、問題の原因は JSP ファイルの処理ではなく、入力パラメータの指定にあります。

HTTP モニターには、クライアントブラウザと Web サーバー間でやりとりされたすべてのメッセージが、関連するコンテキスト情報付きで表示されます。この機能は、矛盾した個所を特定するのに役立ちます。また、HTTP モニターを使用して、コンポーネント間で誤って渡された情報を検出することもできます。詳細については、88 ページの「モニターのデータレコードの表示」を参照してください。

HTTP モニターは、要求の入クッキーと出クッキーの両方を記録して、情報をより簡単に表示できるようにします。

IDE 機能の要約

表 1-1 に、この章で説明した Web 開発の主なニーズに対する IDE の機能をまとめています。

表 1-1 開発者のニーズに対する IDE の取り組み

開発者のニーズ	機能の説明	参照個所
クイックスタート	**作成ウィザード** JSP ページ、サーブレット、フィルタ、リスナー、Web モジュール、Web モジュールグループ、タグライブラリ、および HTML ファイルを作成するためのテンプレートを提供する	46 ページの「Web モジュールの作成」 48 ページの「JSP ページの作成」, 54 ページの「サーブレットの作成」, 59 ページの「フィルタの作成」 64 ページの「独自のタグライブラリの開発」
完全な Web コンポーネントのサポート	**JSP エディタ** JSP ソースコードの編集は HTML の編集に類似している。IDE は HTML タグと JSP タグをサポートしている	49 ページの「JSP ファイルの変更」
効率よい Web アプリケーション開発	**コード補完** JSP ファイル、サーブレット、およびタグライブラリで使用できる補完にアクセスできる	49 ページの「JSP ファイルの変更」と 56 ページの「サーブレットの変更」.
コードの再利用、開発者とページ設計者の分離	**タグライブラリの編集** タグライブラリ記述子とタグライブラリカスタマイザを提供し、タグを効率よく開発できるようにする	61 ページの「既存のタグライブラリの使用」 64 ページの「独自のタグライブラリの開発」.
データフローを表示するためのメカニズム	**HTTP モニター** サーブレットエンジン内での JSP ファイルとサーブレットの実行に関する情報を収集する。JSP ファイルとサーブレットのデバッグに関する作業を整備する。	88 ページの「HTTP モニターの起動」

表 1-1　開発者のニーズに対する IDE の取り組み (続き)

開発者のニーズ	機能の説明	参照個所
JSP ファイルと生成されたサーブレットの比較	**JSP とサーブレットのデバッグ** JSP ファイル、およびそれらの生成されたサーブレットを並べて表示する。	94 ページの「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」
最新のサーバーと仕様のサポート	**J2EE 1.3 準拠** セキュリティ、同時性、トランザクション、および配備を提供する J2EE サービスや通信 API へアクセスする	14 ページの「サーブレットコンテナと Web コンポーネント」
異なるサーバーで配備およびテストするための共通のインタフェース	**配備記述子の編集** 配備記述子の要素を `web.xml` ファイルのプロパティシートで表示および設定する。または、`web.xml` ファイルをソースエディタで開いて手動で編集する	76 ページの「Web モジュール配備記述子の構成」
拡張性	**Enterprise Edition との統合** 既存の EJB モジュールまたは Web モジュールから J2EE アプリケーションを作成する。あるいは、エクスプローラファイルシステムまたはパッケージノードからアプリケーションを作成する	98 ページの「J2EE アプリケーションに Web モジュールをインクルードする」
Web サービスへの簡単なアクセス	**Web サーバープラグイン** コンピュータにインストールされている Web サーバーと統合できるよう、Sun ONE Application Server 7 プラグインと Tomcat プラグインを設定する	80 ページの「統合サーバーの使用」

第2章

Web モジュールの構造

この章では、Web モジュールの構造について説明します。具体的には次の内容が含まれています。

- サーブレットコンテナ
- JSP ページ
- サーブレット
- サーブレットフィルタ
- タグライブラリ
- リスナー

この章では、サーブレットコンテナとその Web コンポーネントについてだけ、基本的な概念を説明します。ここでは、クラスとファイルを直接管理するのではなく、サーブレットコンテナを介してサポートする機能についても説明します。これらのクラスとファイルは Web モジュールの論理的な部分で、Web コンポーネントとともに配備されます。

Web モジュールはサーブレットコンテナ内で実行されます。次に、サーブレットコンテナは Web サーバーに含まれます。Web モジュールには、プレゼンテーション、コントローラ、およびモデルの要素が含まれています。プレゼンテーション要素は、表示要素とも呼ばれ、ユーザーが参照および対話する物理的なページです。これは JSP ページまたは HTML ファイルのいずれかで構成されます。サーブレットなどのコントローラ要素は、ユーザーに表示する内容およびユーザーが対話する方法を制御します。モデル要素には、プレゼンテーション要素とコントローラ要素で使用されるデータが含まれています。データは Java ライブラリまたは他のリソースに含まれている場合もあります。これらのリソースには、GIF イメージ、HTML ファイルなどがあります。

Web サーバー

Web サーバーは、インターネット、イントラネット、またはエクストラネット上のクライアントが Web リソースのリポジトリにアクセスするためのメカニズムを提供します。これらのリソースには、HTML ページ、CGI (Common Gateway Interface) スクリプト、イメージなどがあります。Web サーバーのメカニズムには、次のものが含まれています。

- HTTP および他のプロトコルのサポート
- サーブレットや CGI スクリプトなど、サーバー側プログラムの実行 (特定の機能を実行する)
- サーブレットコンテナのサポート (Web サーバーを利用して HTTP メッセージを処理する)
- 同じベンダーから 1 つまたは複数のサーブレットコンテナをホスティングする

HTTP 要求は、クライアントブラウザで作成されるメッセージです。この HTTP 要求には、属性、およびそのクライアントブラウザのクッキーが含まれています。Web サーバーは、要求をサーブレットコンテナへ渡します。要求処理は、サーブレット、JSP ファイル、およびサーブレットフィルタでも発生します。この処理が完了すると HTTP 要求が完了します。

HTTP 応答はサーブレットコンテナで生成されるメッセージで、この中には最終的にクライアントブラウザへ返されるクッキー、ヘッダー、および出力が含まれています。応答処理は、サーブレット、JSP ファイル、およびサーブレットフィルタでも発生します。Web サーバーは、サーブレットコンテナからクライアントブラウザへ応答を渡します。

サーブレットコンテナと Web コンポーネント

Web サーバーには、**サーブレットコンテナ**が用意されています。サーブレットコンテナは、Web モジュールの Web コンポーネントの実行をサポートするための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
- 配備

- 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

サーブレットコンテナは、クライアントの要求を、Web サーバーからモジュールの Web コンポーネントへ渡します。また、クライアントにバインドされている応答を、Web コンポーネントから Web サーバーへ返します。サーブレットコンテナは一般的に、Web サーバープロセスで (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。

サーブレットコンテナには、要求と応答の送信、要求のネットワークサービス、要求の逆符号化、および応答のフォーマット化を行うネットワークサービスも用意されています。すべてのサーブレットコンテナは、要求および応答のプロトコルとして HTTP (Hypertext Transfer Protocol) をサポートしています。また、その他にも HTTPS (Hypertext Transfer Protocol Secure Sockets) などの要求 - 応答プロトコルが用意されています。

「分散型のサーブレットコンテナ」では、分散可能としてタグ付けされている Web モジュールを実行することができます。サーバーは同じホストまたは異なるホストで実行されている複数の Java 仮想マシンに Web モジュールを分散して実行する場合があり、モジュールはこうした分散型の実行に適切に対応する必要があります。この場合には、Web モジュール内でオブジェクトのスコープが拡張されます。複数のサーバー間で共通のセッション情報の同期化を管理するには、一定量のオーバーヘッドが含まれます。このオーバーヘッドには、パフォーマンスに不利な条件や、記憶域に関する検討事項が含まれています。詳細については、21 ページの「HTTP セッション」を参照してください。

「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。管理はサーブレットコンテナによって行われ、直接サーブレットコンテナと通信します。Web コンポーネントは、サーブレットコンテナを介して HTTP 要求を受け取り、それらを処理して、HTTP 応答を返すことができます。J2EE プラットフォームは、2 種類の Web コンポーネント、サーブレットと JSP ページを定義しています。

サーブレットコンテキスト

Web モジュールの**サーブレットコンテキスト**は、サーブレットを実行する Web モジュールのサーブレットビューが含まれているオブジェクトです。サーブレットコンテキストは、サーブレットがサーブレットコンテナと通信する際に使用する方法を定義します。たとえば、次の処理を行うための方法です。

- 結果のディスパッチ
- イベントの記録
- リソースに対する URL 参照の取得
- コンテキスト内の他のサーブレットが使用可能な属性の設定と保存

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。サーブレットコンテキストは、Web コンポーネントに対して、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。1 つの Java 仮想マシンの 1 つの Web モジュールについて、1 つのサーブレットコンテキストが存在します。

ServletContext インスタンスは、分散型でない Web モジュール内では一意です。また、このインスタンスは、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、JSP ページ内でapplication インスタンス変数として暗黙的に使用できます。この変数は常に使用できます。宣言する必要はありません。

ServletContext インスタンス (およびこのインスタンスが表す 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。この場合には、/productList コンテキスト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。

Web モジュール

Web モジュールは、1 つ以上の Web リソースから構成される単位で、J2EE Web アプリケーションとして配備できます。この Web リソースは、Web コンポーネントや画像などの静的 Web コンテンツファイルになります。Web モジュールを使って、Web ベースの製品カタログアプリケーションを実装できます。このようなプログラムには、具体的に次のものがあります。

- ユーザーに製品を表示する JSP ページ
- カタログ内のナビゲーションを制御するサーブレット
- データベースから情報を取得する Bean

Web モジュールには、必要なものが自身に完備されています。通常、Web モジュールを Web サーバーへ配備するには、1 つのルートディレクトリのコンテンツのみが必要です。

詳細については、45 ページの「IDE での Web モジュール」を参照してください。

JSP ページ

JSP ページは、Web ブラウザに表示されるページなどクライアントの表現を記述するもので、これによって Web モジュールがエンドユーザーと対話することができます。JSP ページはサーブレットコンテナ内でサーブレットクラスに変換されます。

JSP ページには、HTTP 要求をどのように処理するか、また HTTP 応答をどのように生成するかについて記述されています。JSP ページの構文は HTML に類似しており、Java コードに用意されている機能よりも、表現とドキュメントの問題に重点をおいています。サーブレットの詳細については、25 ページの「サーブレット」を参照してください。IDE 内の JSP ページのサポートについては、48 ページの「JSP ページの作成」を参照してください。

JSP ページのライフサイクル

JSP ページは、実行環境 (つまりサーブレットコンテナ) で生成されます。サーブレットコンテナは、要求と応答の管理、作成、対応付け、および削除を管理します。さらに、適切な Web コンポーネントを有効にすることによって、要求と応答の処理を調整します。JSP ページは、HTTP 要求、および生成された HTTP 応答に関する処理を行います。このフェーズに含まれる処理には、JSP ページ変換、インスタンス化、要求処理、および破棄が含まれます。

変換とインスタンス化

JSP ページの変換とは、サーブレットコンテナが 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表します。詳細なプロセスは、サーブレットコンテナによって異なります。Web サーバーまたはサーブレットコンテナでは、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ます。そして、その後でクラスファイルにコンパイルされます。

サーブレットコンテナは、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。サーブレットコンテナは通常、同じ JSP ページに対する次の要求に対して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 ファイルの日付以前の場合は、変換が行われることがあります。このため、JSP ページは Web サーバーを再起動しないで配備し直すことが可能になります。変換プロセスは、サーブレットコンテナで管理されます。

JSP ページはサーブレットに変換された後で、`jspInit` メソッドを呼び出してインスタンス化されます。JSP ページで要求するリソースを準備するには、通常 `jspInit` メソッドを使用します。

要求処理

最も単純なケースでは、JSP ページは、サーブレットコンテナからクライアント要求を受け取ります。そしてプログラムされたロジックに従って要求を処理して、応答をコンテナへ送信します。デフォルトでは、それぞれの要求は自身のスレッドで実行されます。要求処理には、サーブレット、フィルタなどのほかのコンポーネント、またはクライアントからの要求を転送するほかの JSP ファイルを含めることもできます。

破棄

通常、サーブレットコンテナには、JSP インスタンスが要求を受け取らない場合に保持される期間を制限する方法が用意されています。ユーザーが指定した制限時間が経過すると、サーブレットコンテナは JSP インスタンスを破棄してリソースを要求し直すことができます。これを実行する前に、サーブレットコンテナは、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` メソッドを使用して、不要なリソースをクローズすることができます。

JSP ページでのコード構成

JSP ページには、テンプレートデータと要素を含めることができます。**テンプレートデータ**は、HTML や XML コードのように、JSP 以外の構造で構成されるもので、HTTP 応答へ逐次に渡されます。テンプレートデータは一般的に、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場合に使用します。HTML は、そのままのイメージで渡されるため、Web ページの設計者はプレゼンテーションコンテンツをそのままの形でーディングすることができます。**要素**は、サーブレットコンテナによって認識される構造で、動的な機能を提供します。

JSP 要素は、指令要素、アクション要素、およびスクリプト要素の 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。指令は、変換時に処理されます。

指令は、`<%@` と `%>` 記号のあいだに指定します。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 ページを現在のエラーページへ関連付けます。

```
<%@page import="java.util.*" errorPage="showError.jsp" %>
```

表 2-1 に、JSP 仕様で定義されている指令について説明します。

表 2-1 JSP の指令

JSP の指令	説明
`page`	クラスのインポート、セッションへの参加の設定、エラーページの選択を行う
`taglib`	JSP ファイル内でタグを使用できるようにタグライブラリを特定する
`include`	JSP ファイルの他のファイルをインクルードする

アクション要素

「アクション要素」は XML スタイルのタグで、これによって、Java コードを記述しないで Java オブジェクトを使用することができます。たとえば、アクションを使用して、オブジェクトの検出、インスタンス化、およびオブジェクトのプロパティの取得と設定を行うことができます。アクションは、「要求時」、つまりサーブレットコンテナが要求を受け取ったときに処理されます。いくつかのアクションは、HTTP 応答に出力を書き込みます。

「標準アクション」は、サーブレットコンテナで実装されます。

表 2-2 に、JSP 仕様で定義されている JSP の標準アクションについて説明します。

表 2-2 JSP の標準アクション

標準アクション	説明
`forward`	要求の処理を、JSP ページ、サーブレット、HTML ページなどの他のリソースへ即時に渡す
`include`	JSP ファイルの他のファイルをインクルードする。インクルードされたファイル名は要求時に算出することができる
`useBean`	JSP ファイルからアクセスできる Bean を特定する
`getProperty`	`useBean` アクションを介して、JSP ファイルに関連付けられている Bean のプロパティ値を取得する
`setProperty`	`useBean` アクションを介して、JSP ファイルに関連付けられている Bean のプロパティ値を設定する
`plugin`	クライアントブラウザに Java プラグインをロードできるようにする。必要な場合には、組み込みアプレットまたは JavaBeans コンポーネントを実行する

JSP 仕様では、「カスタムアクション」の開発についてもサポートしており、標準アクションで提供されない機能を提供しています。カスタムアクションは、タグライブラリを作成またはインポートして実装します。詳細については、28 ページの「タグライブラリ」を参照してください。

アクションは XML 構文を使用するので、Web ページ設計者は使い慣れているパラダイムを使用して動的データにアクセスすることができます。Web ページの設計者は、アクションそのものをコーディングしない場合もあります。しかし、アクションが含まれているファイルを操作できるよう、アクションについて十分に理解する必要があります。Web ページの設計者は、Web ページへの出力を生成するアクションに対して、HTML テンプレートを提供しなければならない場合もあります。

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 応答へ出力を書き込んだりできます。スクリプト要素は Web サーバーで実行されます。クライアントに送信された応答ページには、スクリプト要素コードの結果だけが表示されます。これに対して JavaScript は、サーバーを介してルーティングされ、処理を行うためにクライアントに返されます。

スクリプト要素には、宣言、式、およびスクリプトレットという構文的に異なる 3 つのタイプがあります。

宣言は、変数の宣言と初期化、オブジェクトのインスタンス化、およびメソッドの宣言を行います。宣言は、変換時に処理され、HTTP 応答へ出力を書き込みません。宣言は、`<%!` と `%>` 記号で囲んで指定します。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;
%>
```

「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。サーブレットコンテナは、式要素を要求時に `String` へ変換します。次に、`String` は、HTTP 応答へ書き込まれます。式は、`<%=` と `%>` 記号で囲んで指定します。

次に、動的データの一部を HTML 文字列へ挿入する例を示します。

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

スクリプトレットは、データを表示用に操作する場合に便利です。スクリプトレットを使用して、有効な Java コードを入力できます。ただし、スクリプトレットの管理は難しいため、スクリプトレットでビジネスロジックを実行することは推奨しません。ビジネスロジックは、Bean やタグハンドラなどの再利用可能な Java クラス内でカプセル化することをお勧めします。

スクリプトレットを使用して、新しいコードのプロトタイプ作成やテストを開始することができます。フィーチャセットが要件を満たしていることを確認したうえで、Bean またはタグライブラリにコードを移動します。複雑なビューロジックについては、39 ページの「ビュー作成ヘルパー」を参照してください。タグライブラリの詳しい作成方法については、64 ページの「独自のタグライブラリの開発」を参照してください。

本稼動用の Web モジュールではスクリプトレットを使用しないでください。開発者がコードを管理し、Web ページ設計者が Web モジュールの外観について管理している場合には、このことは特に重要です。

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 (HTML コードで分散される場合などには)、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。スクリプトレットは、要求時に処理され、(そのようにコーディングしている場合は) HTTP 応答へ出力を書き込みます。

次のスクリプトレット例では、2 つのスクリプトレットにわたる Java の `if` 文を示しています。`if` 文が `true` であると評価された場合のみ、HTTP 応答に HTML コードがインクルードされます。スクリプトレットは、`<%` と `%>` 記号で囲んで指定します。

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

HTTP セッション

HTTP セッションは、クライアントブラウザとサーバー間の会話を表す多数の要求を関連付ける、サーブレット API のメカニズムです。

ユーザーが JSP ページを要求すると、セッションが確立されていない場合には新しいセッションが自動的に作成されます。Web モジュール内のページに対するこれ以降の要求は、通常、このセッションに関連付けられます。page 指令のセッション属性が false に設定されている場合は、要求はセッションに関連付けられません。

```
<%@page contentType="text/html" session="false" %>
```

タイムアウトになったとき、または Web モジュールが明示的に無効になったときにセッションは終了します。セッションの timeout 値は、Web モジュールの配備記述子ファイルに設定されています。JSP ページまたはサーブレットクラスは、invalidate メソッドを使用してセッションを無効にします。セッションのタイムアウト値を設定する詳しい方法については、76 ページの「Web モジュール配備記述子の構成」を参照してください。invalidate メソッドの詳しい使用方法は、49 ページの「セッションの作成と無効化」を参照してください。

一般的な JSP ページは 1 つのセッションに関連付けられます。ただし、JSP ページには、クライアントに特有ではない動的なページも含まれています。このようなページを生成して、多数のセッションで共有できます。セッションに関連付けられていない JSP ページを使用すると、システムリソースに対する Web モジュールの要件が減少します。

分散環境では、セッションを開始した Web サーバーにのみ、特別なセッション情報が含まれています。複数の要求が、全体の負荷に応じてさまざまな Web サーバーへ対応付けられた場合には、セッション情報が複製されます。このような場合には、Web サーバーは共有セッション情報にアクセスし、同期をとる必要があります。Web サーバーで共有のセッション情報を管理することは可能です。ただし、この同期の必要性によって時間とリソースが消費され、システムの負荷全体が高くなります。

HTTP モニターは、ページがセッションに関連付けられているかどうかを示します。また、HTTP モニターは、HTTP 要求について有効な情報を提供します。IDE の HTTP モニターの詳細については、88 ページの「モニターのデータレコードの表示」を参照してください。

スコープと暗黙オブジェクト

JSP ページのオブジェクトをインスタンス化した場合は、それをアプリケーションのほかのオブジェクトで使用できるようにしたいと考えるでしょう。このとき、アプリケーションのすべてのオブジェクトがこのオブジェクトにアクセスできるようにしたり、逆にこのオブジェクトを使用できるオブジェクトを限定したりする場合があります。たとえば、このオブジェクトを、ユーザーの現在の HTTP セッションに関連付けられているオブジェクトでのみ使用できるようにする場合です。JSP 仕様は、オブジェクトに対する参照を設定できるスコープを定義します。

これらのスコープでは、オブジェクトを利用できるかどうかを制御します。このオブジェクトに対する参照は、これらのスコープの任意の場所に設定できます。通常は、1 つのスコープには 1 つのオブジェクトのみを定義します。このオブジェクトは、選択されたスコープのすべてのサブセットで使用できます。つまり、page スコープのオブジェクトは、request、session、および application のスコープ内のオブジェクトで使用することができます。

Web モジュールのスコープの概念は、スタンドアロンアプリケーションにおける従来の概念とは異なります。Web モジュールでは、モジュールのさまざまなコンポーネントに対するオブジェクトの可用性を**スコープ**と呼びます。スコープには、page、request、session、および application があります。スタンドアロンアプリケーションでは、コードブロックにおける変数またはオブジェクトの可用性をスコープと呼びます。

表 2-3 に示されているように、これらのスコープは実行時に Java オブジェクトとして実装されます。

表 2-3　JSP ページのスコープ

スコープ	説明	オブジェクトタイプ
page	現在の JSP ページを表す。このオブジェクトは、現在のページ、または include 指令に含まれているページの JSP 要素でだけ使用できる。このオブジェクトは、include アクションに含まれているページでは使用できない。指令は変換時にページで実行され、これに含まれているページは、同じ JSP 実装クラスに連結される	javax.servlet.jsp.PageContext
request	現在の HTTP 要求を表す。このオブジェクトは、現在の HTTP 要求で実行中の JSP ページおよびサーブレットでだけ使用できる。たとえば forward アクションを使用して、ある JSP ページが他のページへ転送されると、両方のページで同じ ServletRequest オブジェクトにアクセスする	javax.servlet.ServletRequest
session	ユーザーの現在の HTTP セッションを表す。このオブジェクトは、ユーザーの現在の HTTP セッションに関連付けられている要求で実行中の JSP ページおよびサーブレットでだけ使用できる	javax.servlet.http.HttpSession
application	実行時 のWeb モジュールを表す。このオブジェクトは、Web モジュール内のすべての JSP ページとサーブレットで使用できる	javax.servlet.ServletContext

useBean アクションを使用して、Bean を検出したり、またはこれらのスコープのいずれかで使用できるようにしたりすることができます。このアクションでは、scope 属性を使用して、Bean インスタンスが使用できるかどうかを指定します。たとえば次のように指定します。

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

スコープ、およびそのスコープが表すオブジェクトは、ページのスクリプト要素で暗黙的に使用できるようになります。スコープとオブジェクトは、ページを自動的にインスタンス化するスクリプティング変数を使用します。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、ページがセッションに参加していない場合は、session スコープは使用できません。また、JSP ページは暗黙の session 変数を参照できません。ページがセッションに参加していない場合は、page 指令の session 属性が false に設定されています。Web モジュールの中には、セッションデータを必要としない部分があります。この例としては、ユーザーのログインが不要なサイトのバックグラウンド情報があげられます。ユーザーが、そのような部分のみを使用する場合は、ユーザーセッションを作成するときのオーバーヘッドを回避できます。

IDE のコード補完機能を使用して、ほかのスコープ対象のオブジェクトで使用できるメソッドを表示できます。詳細については、49 ページの「JSP ファイルの変更」を参照してください。

スコープ対象のオブジェクトは、サーブレット、タグ、ハンドラ、およびスクリプトレットから設定して使用することができます。スコープ内のオブジェクトを設定するには、対象のスコープオブジェクトで addAttribute メソッドを使用します。スコープ内のオブジェクトを取得するには、対象のスコープオブジェクトで getAttribute メソッドを使用します。有効なスコープについては、表 2-3 を参照してください。

サーブレット

サーブレットは、サーブレットコンテナ内で実行される Java クラスです。これらは次の処理で使用されます。

- Web サーバーおよび Web 対応アプリケーションサーバーの機能を拡張する
- 動的なコンテンツを生成する
- 要求ー応答パラダイムを使用して Web クライアントと対話する

IDE におけるサーブレット開発のサポートについては、54 ページの「サーブレットの作成」を参照してください。

一般的に、サーブレットはフロントコントローラおよびディスパッチャとして使用され、Web モジュールを介してナビゲーションを制御します。また、アプリケーションフローの制御でも使用されます。サーブレットでは、追跡する状態に応じて、特定の Web リソースへのアクセの有効、無効を切り替えることができます。

フロントコントローラのデザインパターンの使用については、34 ページの「フロントコントローラ」を参照してください。

サーブレットのライフサイクル

サーブレットのライフサイクルは、サーブレットのロード、インスタンス化、および初期化をどのように行うかを定義します。また、サーブレットのライフサイクルは、サーブレットがどのようにクライアントの要求を処理し、サービスから消去するかについて説明します。

ロードとインスタンス化

サーブレットは、ネットワークサービスを提供するコンテナ内で実行されます。これらのサービスには、要求と応答の送信、要求のデコード、応答の書式化などがあります。すべてのサーブレットコンテナは、要求および応答のプロトコルとして HTTP をサポートしています。また、HTTPS など、その他の要求－応答プロトコルも用意されています。

サーブレットコンテナでは、サーブレットのロードとインスタンス化を行います。オプションとして、サーブレットコンテナが起動されたときに読み込みとインスタンス化が行われるよう指定できます。「起動時に読み込み」が指定されているサーブレットは、サーブレットコンテナの起動時に読み込まれます。詳細については、55 ページの「起動時読み込み」を参照してください。読み込みが完了すると、コンテナは `Servlet` クラスをインスタンス化して使用できるようにします。

初期化

サーブレットがクライアントの要求を処理するには、サーブレットコンテナによってサーブレットを初期化する必要があります。初期化を行うと、サーブレットは次の処理が可能になります。

- 持続的な設定データを読み込む
- 次の 2 つのタイプのリソースを初期化する
  - 1 つのインスタンスが必要なリソース
  - データベース接続など初期化に時間のかかるリソース
- アプリケーション固有の起動アクティビティを実行する

要求の処理

サーブレットが初期化されると、サーブレットコンテナはサーブレットに対して要求をルーティングできます。サーブレットは要求を処理して、それに対する応答を作成します。サーブレットは、これらのオブジェクトを HTTPServlet インタフェースの service メソッドのパラメータとして渡します。service メソッドは、サーブレットコンテナで呼び出されます。このメソッドでは、サーブレットの init メソッドが初期化された後で、サーブレットが要求に応答します。

通常、サーブレットは、複数の要求を同時に処理できるマルチスレッドのサーブレットコンテナ内で実行されます。開発者は、ファイルやネットワーク接続などの共有リソースに対するアクセスを同期する必要があります。また、サーブレットのクラス変数とインスタンス変数に対するアクセスについても必ず同期化します。サーブレットコンテナは、必要に応じてサーブレットに対して要求をシリアライズできます。

同時に処理されるそれぞれの要求に対して、1 つの新しいサーブレットインスタンスが作成されます。つまり、多数の要求が同時に到着した場合は、新しい多数のスレッドが作成されます。サーバーがオーバーロードしないように、サーブレットコンテナはサーブレットの最大数を制限できます。この制限を設定するには、以下を参照してください。

- Sun ONE Application Server 7 の場合は、http://docs.sun.com で『Sun ONE Application Server 7 管理者用設定ファイルリファレンス』を参照してください。
- Tomcat の場合は、http://jakarta.apache.org/tomcat/tomcat-4.0-doc/config で「Configuration reference」を参照してください。

破棄

サーブレットコンテナは、サービスからサーブレットを削除することを決定すると、Servlet インタフェースの destroy メソッドを呼び出します。このメソッドでは、サーブレットがリソースを解放し、持続的な状態を保存します。

destroy メソッドが呼び出されると、コンテナは、このサーブレットインスタンスに対してほかの要求をルーティングできなくなり、service メソッドの実行中のスレッドは実行を完了することができます。destroy メソッドが終了すると、コンテナは、ガベージコレクションに対してサーブレットインスタンスを解放します。

ほかの Java クラスとは異なり、サーブレットの有効期間は時間、日、または週で設定できます。そのため、データベース接続などのリソースを必要なときだけ使用できるようにするために、これらのリソースの使用と作成を管理する必要があります。

タグライブラリ

タグライブラリはカスタムアクションの集合です。図 2-1 に示すように、タグライブラリは、タグハンドラとタグライブラリ記述子ファイルで構成されます。カスタムアクション、つまりタグは、機能が含まれているタグハンドラ Bean として実装されます。タグライブラリ記述子 (TLD) は、ライブラリの各タグを関連するタグハンドラにマップする XML ドキュメントです。TLD は、ライブラリのタグに関連付けられるパラメータとスクリプティング変数について記述します。

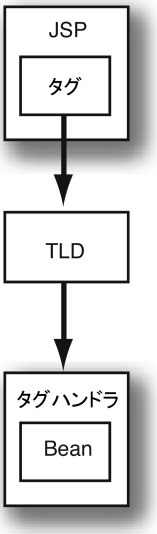

図 2-1　　タグライブラリの構造

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ており、`taglib` 指令を介して JSP ファイルへ提供されます。IDE は、JSTL (Java Server Pages™ Standard Tag Library) などの既存のタグライブラリ、および他社製のライブラリの使用をサポートしています。また、独自のタグライブラリを作成することもできます。IDE におけるタグライブラリのサポートの詳細は、60 ページの「タグライブラリの使用」を参照してください。

JSP Standard Tag Library

JSTL (JSP Standard Tag Library) のタグを使用して、Web モジュールで使用できるアクションセットを拡張できます。JSTL ベースのタグを使用すると、Web モジュールにおけるコンポーネントの移植性が高くなります。また、JSP ファイルにおける管理を難しくする Java コードをスクリプトレットで必要とすることが減ります。

たとえば、さまざまなベンダーの反復タグを使用してリストを反復する代わりに、JSTL タグを使用することができます。JSTL タグは、すべての Web 環境で同じように機能します。このように「標準化されている」ということは、1 つのタグについて習得し、それを複数の JSP コンテナで使用することを意味します。また、仕様によっては、コンテナが標準タグを認識し、それらの実装を最適化することができます。

JSTL では、ページの開発を簡単にするために式言語の概念を導入しています。この概念には、テスト目的での言語の実験的なバージョンが含まれています。JSTL は、既存のカスタムタグと JSTL タグを統合するためのフレームワークも提供しています。

JSTL タグのチュートリアルと説明は、
`http://java.sun.com/webservices/docs/1.1/tutorial/doc/JSTL.html`
で Java Server Pages Standard Tag Library を参照してください。

JSTL および関連する仕様をダウンロードするには、次の Web サイトにアクセスしてください。
`http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html`

IDE によるタグライブラリの編集機能と管理機能を使用すると、JSTL タグを簡単に使用することができます。詳細については、61 ページの「既存のタグライブラリの使用」を参照してください。

サーブレットフィルタ

サーブレットフィルタは、このマニュアルでは「フィルタ」とも呼ばれており、サーブレットへの要求、およびサーブレットからの応答を修正する Java クラスです。フィルタを使用して、以下の機能を実行することができます。

- **認証。**あるユーザーが、限定された Web リソースのみにアクセスすることを保証する。
- **Web アプリケーションユーザーのロギングと監査。**ユーザーが Web リソースにアクセスするたびに追跡を行い、アクセスをログファイルに記録する。
- **ローカリゼーション。**ユーザーのロケールに対して適切なリソースを選択する。
- **データの圧縮。**サーブレットとのやりとりの過程においてデータを圧縮または解凍する。

- **XML コンテンツのスタイル変換。**Web リソースを表示する前に XML コンテンツを変換する。

フィルタは、Java Servlet Specification のバージョン 2.3 に定義されています。サーブレットコンテキストは要求を受け取ると、その要求に関連付けられたフィルタをすべて呼び出してから、要求を処理リソースに渡します。要求が転送されるなどの理由で、要求が内部に割り当てられている場合は、フィルタは呼び出されません。この動作によって、フィルタは門番のような役割として重要なポイントとなります。たとえば、以下の処理でフィルタを使用できます。

- **ロギング。**あるユーザーにとって着信要求のプロパティが関係ある場合は、フィルタによって要求を抽出し、それをファイルに書き込むことができる。
- **要求に関連するデータの変更または追加。**元の要求と応答をラッパーで置き換えることができる。これによって、着信要求が JSP ページまたはサーブレットで処理される前に、この要求を変更することができる。
- **リソースグループで必要な前処理の実行。**たとえば、ユーザープロファイルがセッションに読み込まれているかどうか確認できる。

これに対して、`RequestDispatcher` API を使用してサーブレットまたは JSP ページから要求を送信する場合には、ロギングまたは前処理が必要になります。このような場合は、代わりにアクションをサーブレットとして実装する必要があります。

サーブレットコンテキストは要求を受け取ると、リソースへのパスに一致するフィルタがあるかどうかを決定します。いずれかのフィルタが検出されると、サーブレットコンテキストは `FilterChain` オブジェクトを作成します。このように、一致するフィルタは、配備記述子の中で宣言されている順序ですべて連結してチェーンが形成されます。サーブレットコンテキストは、最初のフィルタで `doFilter()` メソッドを呼び出します。各フィルタは、チェーン内の次のフィルタを呼び出し、次のフィルタに対して制御を行うということが順に繰り返されます。チェーン内の最後のフィルタが `doFilter` メソッドを呼び出したときに、要求は JSP ページまたはサーブレットによって処理されます。要求は、画像などの静的なコンテンツ内で処理することも可能です。このプロセスが完了すると、最後のフィルタに制御が返されます。最後のフィルタの `doFilter` メソッドが完了すると、最後から 2 番目のフィルタに制御が戻される、というように順に制御が戻されます。

アクションにおけるフィルタの例として、IDE の HTTP モニターがあります。HTTP モニターは、ほかの Web モジュールリソースが要求を処理する前後に、フィルタを使用して要求とサーブレットコンテキストに関するデータを収集します。フィルタは、リプライ要求の処理も行います。この処理は、着信要求をラッパーで置き換えることによって実行されます。フィルタは、次に元の要求データをラッパーに挿入します。このため、HTTP モニターのフィルタはアプリケーション定義のフィルタよりも前に呼び出されるよう、チェーンの先頭に定義しておく必要があります。通常、フィルタは各 Web モジュールの配備記述子で宣言されます。

IDE におけるサーブレットフィルタのサポートについては、59 ページの「フィルタの作成」を参照してください。

フィルタのライフサイクル

フィルタのライフサイクルによって、フィルタのロード、インスタンス化、および初期化をどのように行うかが定義されます。また、フィルタのライフサイクルは、フィルタがどのようにクライアントの要求を処理し、サービスから消去されるかについて説明します。

ロードとインスタンス化

サーブレットコンテナは Web リソースにアクセスする前に、Web リソースに適用されるフィルタのリストを特定します。サーブレットコンテナは、リスト上の各フィルタに対して適切なクラスのフィルタがインスタンス化されるようにします。また、サーブレットコンテナは、各フィルタに対して`FilterConfig` メソッドを呼び出します。

初期化

コンテナは要求を受け取ると、チェーン内でフィルタリストの先頭にあるフィルタに対して `doFilter` メソッドを呼び出します。次にコンテナは、サーブレットの要求、応答、およびサーブレットが使用するフィルタチェーンを通過します。フィルタチェーンは配備記述子で定義されています。

フィルタに対する `doFilter` メソッドは、要求のヘッダーを調査します。また、要求や応答のヘッダーを修正することも、要求や応答のオブジェクトをラッピングすることによってデータを変更することも可能です。

フィルタは、次にフィルタチェーン内の次のエンティティを呼び出します。このエンティティはフィルタの場合もあります。要求は、フィルタチェーン内で最後のフィルタに到達するまで順に処理されます。最後のフィルタは、関連する Web リソース、サーブレットや JSP ページなどを呼び出します。

破棄

コンテナは、サービスからフィルタを削除する前に、対象のフィルタで `destroy` メソッドを呼び出します。コンテナはほかのすべてのリソースを解放してクリーンアップ処理を実行します。

リスナー

アプリケーションイベントリスナーは、Servlet 2.3 Specification で新しく定義されたものです。これらはサーブレットイベントリスナーのインタフェースを実装するクラスです。アプリケーションイベントリスナーは、Web モジュールの配備時に Web モジュール内でインスタンス化されて、登録されます。

リスナークラスには、Web モジュール内のセッションを追跡する方法が用意されています。これは、セッションが無効になった理由を調べる場合に便利です。たとえば、コンテナでセッションがタイムアウトになった、またはモジュールのいずれかが `invalidate` メソッドを呼び出したとします。このような場合には、リスナーと `HTTPSession` API メソッドを使用して区別することができます。

サーブレットイベントリスナーは、次のオブジェクトの状態変化についてのイベント通知をサポートしています。

- **サーブレットコンテキストオブジェクト。**モジュールに対して仮想マシンレベルでリソースを管理する場合に有効。
- **HTTP セッションオブジェクト。**状態を管理する場合に有効。あるモジュールについて同じクライアントの一連の要求に関連付けられているリソースを処理する場合にも有効。

ライフサイクル内で発生する変化を監視するには、複数のリスナーを使用します。これらのリスナーは、サーブレットコンテキストおよび HTTP セッションオブジェクトの属性を追跡する場合にも便利です。

- 「サーブレットコンテキストイベント」には、次のものがあります。
  - **ライフサイクル。**サーブレットコンテキストが作成された直後で最初の要求をサービスすることができる、または停止の準備ができている。
  - **属性の変更。**サーブレットコンテキストの属性が追加、削除、またはリプレースされた。
- 「HTTP セッションイベント」には、次のものがあります。
  - **ライフサイクル。**HTTP セッションが作成されたか無効になった、またはタイムアウトした。
  - **属性の変更。**HTTP セッションの属性が追加、削除、またはリプレースされた。

第3章

デザインパターンとフレームワーク

この章では、Web アプリケーションのアーキテクチャに使用すると便利なデザインパターンに関する用語と一般的な概念について説明します。一般的に使用されるいくつかのパターンについて概説し、Web アプリケーションの開発プロセスを簡略化するフレームワークを紹介します。

デザインパターン

デザインパターンは、ソフトウェアの設計時に起こる一般的な問題をアーキテクチャによって解決するソリューションです。これらのパターンは、開発者の経験と洞察から自然発生的に生まれてきたものです。この節では、Web アプリケーションの開発するときに有効であろうと考えられる基本パターンをいくつか紹介します。

これらのパターンに関する情報は、Java Developer Connection 内の次の Web サイトにも掲載されています。

```
http://developer.java.sun.com/developer/restricted/patterns/
J2EEPatternsAtAGlance.html
```

ここでは、特に Web アプリケーションの構築に関連するパターンを紹介します。詳細な説明はこの後の節で行います。

- **フロントコントローラ。**受信する要求の処理を調整します。詳細については、34 ページの「フロントコントローラ」を参照してください。
  - **ディスパッチャ。**ユーザーが表示するビューの制御方法を記述する、フロントコントローラパターンのサブパターンです。詳細については、36 ページの「ディスパッチャ」を参照してください。
  - **ビューヘルパー。**この場合は、処理機能、データアクセス、ビジネスロジックなどのビジネスルールをカプセル化する、フロントコントローラパターンのサブパターンです。詳細については、37 ページの「ヘルパー」を参照してください

- **複合ビュー。**「テンプレート」とも呼ばれます。複数のサブコンポーネントから 1 つの集約的ビューを作成します。詳細については、38 ページの「複合ビュー」を参照してください。

Web アプリケーションでのデザインパターンの使用に関する詳細な解説は、Deepak、Crupi、Malks 著の『Core J2EE Patterns』を参照してください。このマニュアルについての詳細は xiii ページの「参考資料」を参照してください。

フロントコントローラ

フロントコントローラは、受信するユーザー要求のルーティングを行います。また、Web アプリケーション内のナビゲーションを強制することもできます。ユーザーが Web アプリケーション内を自由に表示できる部分にいる間は、フロントコントローラは適切なページへの要求を単純に中継します。たとえば、電子商取引アプリケーションでは、顧客は商品カタログのいろいろな部分を表示して回ります。しかし、制御されている部分に入ったら、ユーザーは特定のパスに従ってアプリケーション内を移動しなければなりません。そのような部分では、フロントコントローラは、受信する要求について妥当性検査を行い、その結果に従って適切なルートに振り分けることができます。たとえば、顧客がショッピングカートに入っている商品を購入する場合を例にとります。その顧客が購入を正しく完了するには、特定のルートに従う必要があります。

フロントコントローラは、Web アプリケーション内の複数のリソースへの要求が通過する単一のエントリポイントを提供します。1 つのフロントコントローラでアプリケーションへのすべての要求を処理できます。アプリケーションの一部分への要求を、複数のフロントコントローラでそれぞれ処理することもできます。通常、フロントコントローラはサーブレットとして実装され、次のタスクに一般的に使用されます。

- ページフローやナビゲーションの制御
- モデルデータへのアクセスとそのデータの管理
- ビジネス処理の処理
- ユーザープロファイルなどの GUI 表示に関連するデータへのアクセス

フロントコントローラを使用すると、複数のリソースで同じ処理を必要とする場合に、特に、JSP ページ内のコードの重複を減らすことができます。たとえば、ユーザーのプロファイルがすでに見つかっているかどうかを確認したり、製品 ID に対応するデータを取得したりします。

すべてのクライアント要求がフロントコントローラを通るようにすると、Web アプリケーションをより効果的に維持制御することができます。ビューの選択やテンプレートの作成などの機能を、このデザインパターンにより一元化することができます。これらの機能は、フロントコントローラによってすべてのページやビューに一貫して適用されます。したがって、これらの機能の動作を変更する必要がある場合は、フロントコントローラとそのヘルパークラスを変更するだけですみます。アプリケーションの中でそれらが占めるのは比較的小さな部分です。

図 3-1 のような 2 層型の Web アプリケーションでは、ユーザー要求をフロントコントローラで処理する方法をお勧めします。この場合、ユーザーに対して表示するプレゼンテーション要素や、選択されたプレゼンテーションに使用するデータも、コントローラが決定します。この方法は、各ユーザー要求を個別のビューにマッピングしなければならない従来のアプローチと対照的です。

フロントコントローラで要求を、直接ビューに振り分ける必要はありません。フロントコントローラを連鎖させて、たとえば、1 つのフロントコントローラだけがユーザープロファイル情報にアクセスするようにします。フロントコントローラはこのプロファイルを別のフロントコントローラに転送することができます。

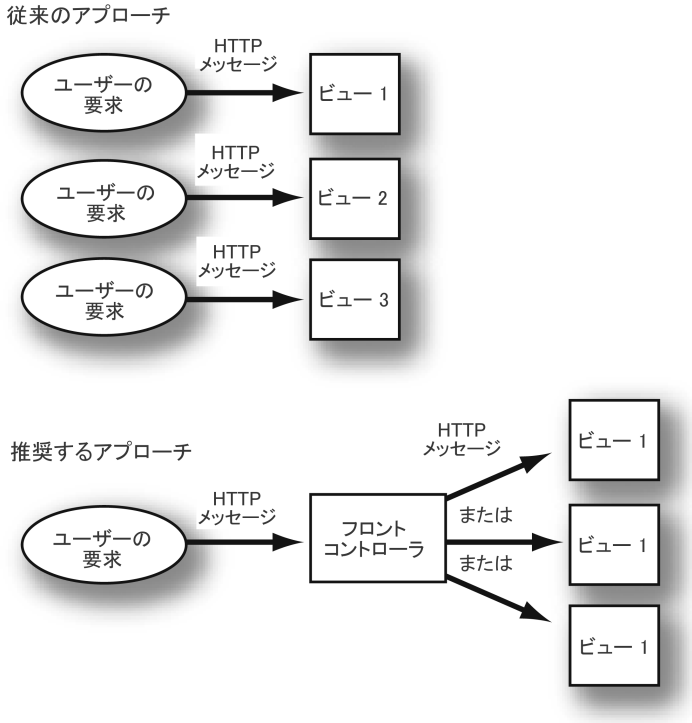

図 3-1　フロントコントローラによるユーザービューの決定

IDE を使用してサーブレットをフロントコントローラとして作成する方法についての詳細は、57 ページの「フロントコントローラとしてのサーブレットの使用」を参照してください。

ディスパッチャ

通常、フロントコントローラでユーザーのナビゲーションを調整するときには、この目的のための**ディスパッチャ**サブパターンが使用されます。図 3-2 に示すように、フロントコントローラが要求を処理します。ユーザーの要求は、たとえば、電子商取引アプリケーションのショッピングカートに入っている商品をチェックアウトすることです。

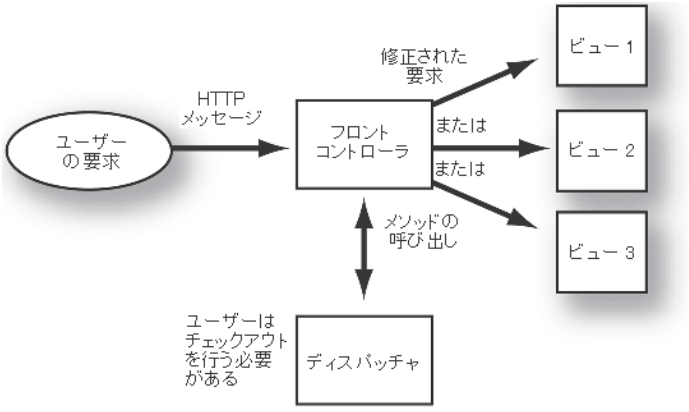

図 3-2　　フロントコントローラの機能のディスパッチ

ディスパッチャコードは、フロントコントローラサーブレット内にある場合と別のクラスにある場合があります。実際には、ディスパッチャがフロントコントローラに要求の転送先を指示します。フロントコントローラのデザインパターンでは、ユーザーに対して表示するビューを制御する動作が、ディスパッチャによってカプセル化されます。

ビューマッパー

クライアントの種類によって Web リソースが異なる場合は、**ビューマッパー**を使用してディスパッチャの機構を補助することができます。ここでのクライアントとは、Web ブラウザ、PDA (携帯情報端末)、携帯電話などです。たとえば、波や潮に関する情報を取得する Web アプリケーションを開発中であるとします。この場合、この情報をデスクトップのパーソナルコンピュータから表示したいというユーザーがいれば、携帯電話から表示したいというユーザーもいるでしょう。ビューマッパーを使用すると、常に同じ JSP ページにディスパッチするのではなく、クライアントの種類に応じて異なる JSP ページを送信することができます。

1. たとえば、Web アプリケーションは、外から要求を受信したときにそれをフロントコントローラサーブレットにルーティングします。

2. フロントコントローラはヘルパー Bean を使用して適切なデータを取得します。

3. また、ビューマッパーを使用して、クライアントに応じた適切なビューの種類を決定します。

4. ビューマッパーからの入力に基づいて、ディスパッチャからフロントコントローラにビュー情報が返されます。

5. これを受けて、アプリケーションは、図 3-3 に示すように、ユーザーが使用しているクライアント用の特定のビューに要求を転送します。

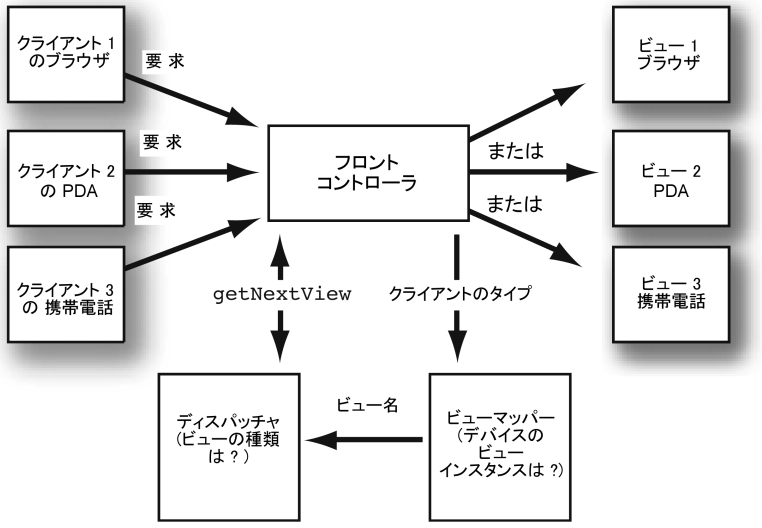

図 3-3 ビューマッパーの使用

開発中の潮と波のアプリケーションを使用するユーザーが PDA と携帯電話のどちらに情報を表示したいと希望しているのか、最初はわからないかもしれません。そのような場合は、ビューマッパーを使って代替オブジェクトまたはオブジェクトファミリを作成することができます。

ビューマッパーを使用すると、異なるデバイスだけでなく、異なるロケールや異なるビューにも情報をリダイレクトすることができます。

ヘルパー

フロントコントローラサーブレットは、すぐに大きくなって扱いにくくなってしまいます。このため、ヘルパークラスを使って機能別に分割し、アプリケーションの構築や維持の作業を行いやすくする必要があります。ヘルパークラスには、次のようなタスクをカプセル化することができます。

- ファイル、別の Web サイト、または Web サービスからのコンテンツの取得
- ユーザーによって入力された情報の妥当性検査

- フロントコントローラがビジネスロジックの処理を委任する必要がある場合は、図 3-4 に示すように、この目的にヘルパーを使用できる
- データ処理

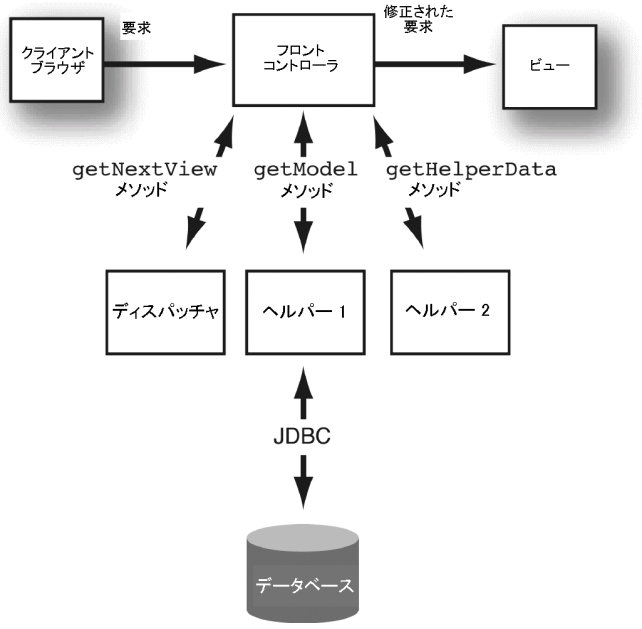

図 3-4 ヘルパーによるビジネスロジックの処理の委任

プレゼンテーション層のデータの処理と取得を行うヘルパーは、ビューヘルパーと呼ばれます。ビューヘルパーによってビジネスロジックがプレゼンテーションロジックから分離されるため、アプリケーションを容易に管理することが可能になります。

ヘルパーは、通常の Java クラスとして実装できます。詳細については、53 ページの「追加のクラスまたは Bean の使用」を参照してください。

複合ビュー

複合ビューは、複数のコンポーネントビューから 1 つの集約的ビューを作成するデザインパターンです。複合ビューには、ページ内の複数の動的なモジュール部分を含めることができます。このデザインパターンは、Web アプリケーション設計で多数のサブビューから 1 つのビューを作成するときに適してします。複雑な Web ページは、さまざまなリソースから派生したコンテンツで構成されていることがよくあります。ページのレイアウトは、異なるサブビューのコンテンツごとに管理されています。たとえば、ナビゲーション、検索、特集記事、ヘッドラインなどのサブビューを

持つビューがあるとします。含まれるビューは、より大きな全体の一部分を成すサブビューです。含まれるビューもまた、図 3-5 に示すように、複数のサブビューから構成されているかもしれません。

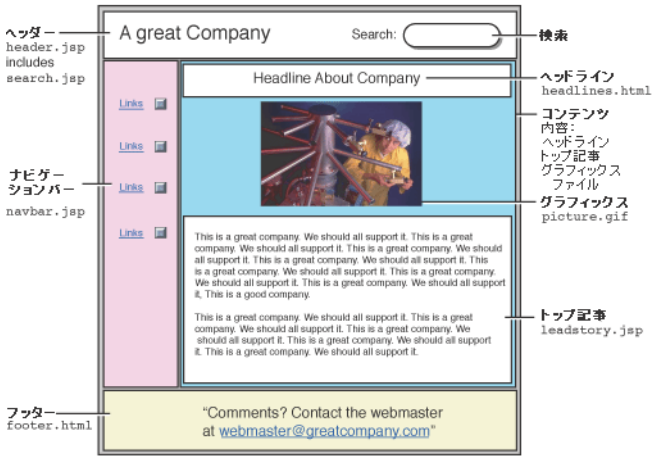

図 3-5　　複合ビューによるレイアウトから切り離したコンテンツ管理

複合ビューを作成する場合は、静的なコンテンツと動的なコンテンツを含めることができます。静的なコンテンツは、たとえば HTML ファイルから構成されています。動的なコンテンツとは JSP ページなどです。JSP 変換時および実行時にもコンテンツを含めることができます。

IDE を使用して複合ビューパターンを実装する方法については、51 ページの「複合ビューテンプレートの作成」を参照してください。

ビュー作成ヘルパー

通常、Web ページは、時がたつうちに再利用やメンテナンスを行う必要が生じてきます。データを受信したままの状態で表示する必要がある場合は、**ビュー作成ヘルパー**を使用します。その情報はテーブルであるかもしれませんし、リンクのセットであるかもしれません。ビュー作成ヘルパーには任意の Bean または Java クラスを使用できます。ただし、JSP ページでの表示用に特別に作成されているので、通常、ビュー作成ヘルパーにはタグハンドラークラスが使用されます。

ビュー作成ヘルパークラスを使用すると、特定の表示機能に関連する Java コードを JSP ファイルやフロントコントローラサーブレットに直接配置する必要がありません。たとえば、Web アプリケーションにカタログ検索が含まれているとします。カタログ検索は、表示される結果が決まっています。このような場合は、図 3-6 のように、動作を JSP タグにカプセル化します。

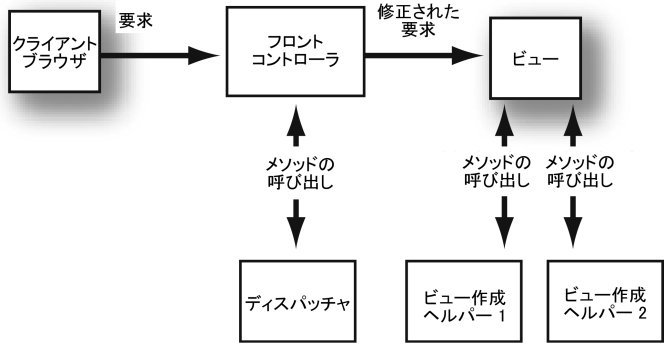

図 3-6　　ビュー作成ヘルパーの使用

また同様に、ビュー内のデータのフォーマットにもこのロジックを使用します。ビュー作成ヘルパーは、JSP ファイル内の他の Bean と同じ方法で取得および使用することができます。IDE での Bean の使用方法についての詳細は、53 ページの「追加のクラスまたは Bean の使用」を参照してください。

Model オブジェクト

Model オブジェクトは、Web アプリケーション内のアプリケーションデータをカプセル化する Java オブジェクトです。たとえば、ショップカート式の電子商取引アプリケーションでは、ユーザープロファイルを Model オブジェクトとして抽象化することができます。このオブジェクトには、たとえば、ユーザーの名前、電子メールアドレス、電話番号などがデータとして含まれています。

こうした情報はデータベースに格納し、アプリケーションがネットワーク要求を実行してデータを取得するのが一般的です。ネットワーク要求はリソースを消費するため、アプリケーションは同じデータに対する要求の数を最小限に抑える必要があります。このため、セッションデータを注意して設計する必要があります。JDBC を使用してデータベースにアクセスする方法についての詳細は、Sun ONE Studio 5 プログラミングシリーズの『Java DataBase Connectivity の使用』を参照してください。

Model オブジェクトをフロントコントローラから JSP ファイルに渡すには、次の 2 つの方法があります。

- 要求属性を使用する
- Web アプリケーションのセッションデータ内に配置する

  製品データをアプリケーションに 1 つの要求の間だけ保存するのではなく、もっと長い期間保存する必要がある場合は、その情報をアプリケーションのセッション内に配置することができます。

通常、単一のユーザーからの要求はすべて、セッション情報が保存されている単一のサーバーに送られます。負荷均衡型システムでは、1 人のユーザーの要求が異なるサーバーにルーティングされる可能性があります。このため、特定ユーザーのセッション情報をサーバー間で共有する必要があります。このような状況で、Web アプリケーションがセッションデータへのアクセスの同期化を行わなければならない場合は、処理速度が低下することがあります。

負荷均衡型システムにアプリケーションを配備する場合、セッションデータを配信するのは非効率的です。この場合は、ユーザーに送信される、ページの隠しパラメータとしてデータを格納できます。その後データは、そのページから送られた後続の要求とともにサーバーに送信されます。

フレームワーク

フレームワークは、新しいアプリケーションを構築する際の設計とコーディングの処理を簡略化するためのデザインパターン、API、実行時実装のセットです。フレームワークには Struts、Sun ONE Application Framework、および JavaServer Faces などがあります。これらについてはこの後の節で説明します。

フレームワークの設計者は、既存のアプリケーションから共通する機能を取り出して、適切なデザインパターンを使ってそれらを実装しています。フレームワークを使用すると、開発者は開発中のアプリケーションに必要な独自の機能のセットだけに集中することができます。

通常、Web アプリケーションのフレームワークでは、次のような機能のサポートを提供しています。

- アプリケーション内のページ間のナビゲーション
- エンドユーザーからの入力を受け入れるフォームの構築
- エンドユーザーに対して情報を表示するためのビューの作成
- JavaBeans およびエンタープタイズ JavaBeans による処理やデータへのアクセスの提供
- データベースシステムから情報を直接取り込むためのデータベースアクセスの提供
- JNDI (Java Naming and Directory Interface) や LDAP (Lightweight Directory Access Protocol) などのディレクトリサービスの使用
- 認証機構を使用したセキュリティ保護されたアクセスとセキュリティ保護されていないアクセスの提供

Struts

「Struts」は、Jakarta プロジェクトによるオープンソースフレームワークです。Java サーブレット API および JSP テクノロジを使った Web アプリケーションの構築を目的に設計されています。Struts パッケージは、再利用可能なコンポーネントの統合セットを提供します。このセットには、コントローラサーブレット、JSP カスタムタグライブラリ、ユーティリティクラスなどが含まれます。これらは、任意の Web ベース接続に適用できるユーザーインタフェースを作成するためのコンポーネントです。Struts は、ワークフローの標準化や、簡単で再利用可能な Web アプリケーションの構築を容易にします。Struts フレームワークについての詳細は、`http://jakarta.apache.org/struts/` を参照してください。

JavaServer Faces

JavaServer Faces では、JSP タグと Java クラスの標準セットを定義することを提案しています。このセットのねらいは、Java Web アプリケーションのグラフィカルユーザーインタフェース (GUI) の構築を簡略化することにあります。JavaSerer Faces では、複雑な HTML フォームやその他の共通の GUI 要素を定義します。また、ツールや Sun 以外のコンポーネントのベンダーは、JSP ページおよびサーブレットの単一のコンポーネントフレームワークに焦点を置くことができます。このフレームワークは、従来の GUI ツールキットの開発者と Web ベースの GUI の開発者のギャップを埋めることを目的としています。GUI コンポーネント、コンポーネントの状態、レンダリングおよび入力処理のための使い慣れた API を提供することも目的の 1 つです。国際化と入力の基本的な妥当性を検査するための総合的サポートが提案されています。このため、開発者は、国際化と入力の妥当性検査を初回リリースからサポートしなければなりません。JavaServer Faces についての詳細は、`http://www.jcp.org/jsr/detail/127.jsp` を参照してください。

第4章

Web アプリケーションの開発

この章では、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、アプリケーションを作成するために必要なさまざまな作業について体系的に説明します。また、さらに、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 5 章を参照してください。

IDE を使って Web アプリケーションを実際に構築する前に、『Sun ONE Studio 5 Web アプリケーションチュートリアル』でひと通り学習しておくことをお勧めします。

開発ワークフロー

この節では、IDE を使用した Web モジュール開発の一般的なワークフローの概要を説明します。新しい Web アプリケーションを開発するとき、反復形式で作業を進めていくのは多くの場合有意義なことです。ここでは、開発のための主な作業について説明します。各作業の詳細については、この章の各タスクに関する節で説明します。JSP/サーブレットモジュールの IDE オンラインヘルプにも、これらのタスクの情報があります。

IDE を使用して Web モジュールを開発する手順は、次のとおりです。

1. エクスプローラウィンドウで、Web モジュールの**ドキュメントルートノード**を作成します。詳細については、46 ページの「Web モジュールの作成」を参照してください。
2. 使用する Web コンポーネントを作成します。このプロセスには、たとえば次のような作業が含まれます。

- エンドユーザーに対して表示するフォームやその他の動的な Web ページに使用する JSP ページの作成。詳細については、48 ページの「JSP ページの作成」を参照してください。
- Web モジュールに必要なサーブレット、Bean、フィルタ、およびユーティリティクラスの作成。サーブレットを使用すると、Web モジュールのフローを制御したり、外部のリソースにアクセスすることができます。フィルタを使用すると、Web アプリケーション内のサーブレットや JSP ページにアクセスしようとしているユーザーを認証することができます。詳細については、54 ページの「サーブレットの作成」および59 ページの「フィルタの作成」を参照してください。

  サーブレット、フィルタ、Bean、およびユーティリティクラスは、Web モジュールの `WEB-INF/classes` ディレクトリに置かれます。ただし、JAR ファイルとしてパッケージ化したクラスは、`WEB-INF/lib` ディレクトリに置かれます。

  IDE にはこれらすべてのオブジェクトのテンプレートが用意されており、新規ウィザードで使用することができます。単純フィルタと拡張フィルタのテンプレートや 4 種類のリスナーのテンプレートも用意されています。その他のオブジェクトにはそれぞれ 1 つのテンプレートがあります。詳細については、Core IDE オンラインヘルプの「新しいファイルの作成」を参照してください。
- Web モジュールに必要なサーブレット、Bean、およびその他のクラスのインポート。詳細については、47 ページの「既存の Web モジュールのインポート」を参照してください。
- JSTL などの既存のタグライブラリのインポート。詳細については、61 ページの「既存のタグライブラリの使用」を参照してください。
- JSP ページに必要なその他のカプセル化された機能のカスタムタグライブラリの開発。

  タグライブラリは、それを使用する Web モジュールの中で開発することをお勧めします。タグの開発を終了したら、タグライブラリのソースファイルをメンテナンス用に別のファイルシステムに移動します。この時点で、このライブラリを 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 `WEB-INF/lib` ディレクトリに置きます。詳細については、64 ページの「独自のタグライブラリの開発」を参照してください。

3. 配備記述子ファイルを編集して、Web モジュールとそのコンポーネントを構成します。詳細については、76 ページの「Web モジュール配備記述子の構成」を参照してください。

IDE での Web モジュール

Web モジュールは J2EE の**配備**構造です。Java Servlet 2.3 Specification および JSP 1.2 Specification では、JSP ページを必ず Web モジュール内で実行するように規定しています。IDE を使用して Web アプリケーションを開発すると、必要な Web モジュール構造が自動的に作成されます。IDE では、Web モジュール構造を強制することによって、Web モジュールを確実にパッケージ化して配布できるようにしています。また IDE は、サーブレットコンテナへの配備に必要な配備記述子情報 (`web.xml` ファイル) が Web モジュールに必ず含まれるようにしています。

注 – IDE で JSP ページやサーブレットを実行またはデバッグするには、それらを Web モジュール内に置く必要があります。JSP ページとサーブレットは、いずれも必ず Web モジュール内から実行しなければなりません。この動作は、IDE の以前の一部のバージョンと異なります。

各 J2EE Web モジュールは、Java Servlet Specification バージョン 2.3 で定義されているように 1 つの Web アプリケーションに対応しています。IDE では、**Web モジュールグループ**と呼ばれる構造を使って、複数の **Web モジュール**を一緒に配備することができます。詳細については、JSP/サーブレットのオンラインヘルプの「Web モジュールグループの作成」および「Web モジュールの実行」を参照してください。

Web モジュールは、J2EE アプリケーション内に配備して使用できる Web リソースの最小単位です。J2EE 仕様に定義されているように、**サーブレットコンテキスト**に対応しています。Web モジュールは、通常、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。ただし、使用する Web サーバーによっては、パッケージ化しなくても Web モジュールを配備できる場合があります。WAR ファイルの形式は JAR ファイルの形式と同一です。しかし、WAR ファイルの内容と用途は JAR ファイルと異なるので、WAR ファイル名には `.war` 拡張子を使用します。

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。開発時には、この構造は Web モジュールのルートディレクトリにマウントされた、次のファイルとフォルダを持つファイルシステムとして表現されます。

- JSP ページ、HTML ファイル、GIF などのイメージファイル、その他これらにはエンドユーザーが直接アクセスすることができます。
- `WEB-INF` フォルダ。このフォルダの中には次のものがあります。
  - `classes` フォルダ。サーブレット、フィルタ、リスナーなどの Java クラスファイルが入っています

- `lib` フォルダ。タグライブラリ、JDBC (Java Database Connectivity) ドライバ、その他の Java クラスライブラリなどの JAR ファイルが入っています。Web アプリケーションの機能の多くは、これらのファイルによって提供されます
- サーブレットやフィルタクラスを使って表示することだけできて、エンドユーザーが直接アクセスすることはできない JSP ページ
- `web.xml` ファイル。これは Web モジュールの配備記述子です
- タグライブラリ記述子ファイル

Web モジュールの作成

IDE から Web モジュールを作成するための最初の手順は、モジュールの**ドキュメントルートノード**を作成することです。新規ウィザードを使用して、指定したファイルシステムに Web モジュールを作成します。ウィザードでの手順が完了すると、Web モジュールのドキュメントルートディレクトリがエクスプローラにノードとして表示されます。ノードを展開すると、ノードの内容が表示されます。この内容には、Web モジュールの構造が反映されています。

- WEB-INF ノードには、次のものが含まれています。
  - `classes` ディレクトリ
  - `lib` ディレクトリ
  - `web.xml` ファイル。このマニュアルでは配備記述子と呼ばれています。

Web モジュールのディレクトリ構造は、IDE のエクスプローラウィンドウ内のオブジェクトとして扱われます。そして Web モジュールのドキュメントルートディレクトリにマウントされたファイルシステム内に表示されます。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。Web モジュールオブジェクトは、WEB-INF ノード のアイコンで表示されます。

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 モジュールのドキュメントルートノードを作成できます。

注 – 他のファイルシステムをマウントする場合とまったく同じ方法で、エクスプローラに Web モジュールをマウントします。ファイルシステムのマウントについては、Core IDE ヘルプセットの「ファイルシステムのマウント」を参照してください。ただし、Web モジュールは、必ずルートディレクトリにマウントします。`WEB-INF` ディレクトリがあるのがルートディレクトリです。Web モジュール自体ではなく、Web モジュールを含んでいるディレクトリをマウントすると、Web モジュールが正しく認識されません。この場合、マウントしたファイルシステムのサブディレクトリが Web モジュールのドキュメントルートとなり、Web モジュールに通常関連付けられる操作の一部を実行することができません。実行できない操作は、たとえば、Web モジュールまたはそのコンポーネントの実行や配備などです。

既存の Web モジュールのインポート

IDE では、外部で作成した Web モジュールの開発を継続して行うことができます。既存の Web モジュールをインポートするには、Web モジュールの配布方法に応じて次の 2 つの方法のいずれかを使用します。

- ルートディレクトリにある Web モジュールをインポートする場合は、ファイルシステムのエクスプローラで Web モジュールのルートディレクトリをマウントします。
- WAR ファイル内の Web モジュールをインポートする場合は、WAR ファイルがあるディレクトリをマウントします。WAR ファイルノードに対して「Web モジュールとしてアンパック」アクションを使用し、Web モジュールディレクトリを作成します。

「ファイルシステムをマウント」ダイアログで、インポートしたい Web モジュールの場所を指定します。Web モジュールを含む CVS ファイルシステムをマウントすることもできます。

詳細については、JSP/サーブレットのオンラインヘルプの「既存の Web モジュールのマウント」を参照してください。

IDE にマウントする Web モジュールは、構造化されたディレクトリ形式または Web アーカイブ形式 (WAR) に従っている必要があります。これらの形式については、『Java Servlet 2.3 Specification』を参照してください。

インポートした WAR ファイルを実行するには、まずそのファイルをアンパックして Web モジュールとしてマウントする必要があります。エクスプローラの「ファイルシステム」タブから WAR ファイルをアンパックしてマウントします。関連する WAR ファイルを含むディレクトリをまだマウントしていない場合は、これをマウントします。アンパックする WAR ファイルのアイコンを右クリックします。次に、コンテキストメニューから「Web モジュールとしてアンパック」を選択します。「WAR アンパックの場所の選択」ダイアログで、アンパックしたファイルを保存するディレクトリを指定します。「ここでアンパック」をクリックします。指定したディレクトリに WAR ファイルがアンパックされます。ディレクトリがマウントされ

て、エクスプローラの「ファイルシステム」タブに Web モジュールとして表示されます。WAR ファイルに Java ソースファイルが含まれていない場合は、WAR ファイルをアンパックするとその中のサーブレットおよび JavaBeans コンポーネントは編集できなくなります。

詳細については、JSP/サーブレットのオンラインヘルプの「WAR ファイルのアンパックとマウント」を参照してください。

JSP ページの作成

JSP ページは、エンドユーザーに対して情報を表示するために Web アプリケーション内で使用されます。エンドユーザーが入力したデータをサーバーに送信することもできます。たとえば、フォームを使用したデータの表示や変更に使用されます。

IDE で JSP ファイルを作成および管理する方法は、ほかのファイル形式の場合と同様です。JSP ページを実行するには、正しくマウントされた Web モジュール内に JSP ページを配置する必要があります。

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。

- 新規ウィザードの使用。詳細については、JSP/サーブレットのオンラインヘルプの「JSPファイルやサーブレットファイルの作成」を参照してください。
- Dreamweaver テンプレートに基づいた生成。このテクニックの詳細については、JSP/サーブレットのオンラインヘルプの「Dreamweaver ファイルの取り扱い」を参照してください。

JSP ページは、Web モジュールのルートディレクトリか、このルートディレクトリのサブディレクトリで作成することができます。

`WEB-INF` ディレクトリまたはそのサブディレクトリに配置された JSP ページに、クライアントブラウザから直接アクセスすることはできません。ただし、サーブレットからリソースとしてアクセスすることはできます。通常、この機能はフロントコントローラデザインパターンとともに使用されます。この機能を使用して、特定の順番で表示する必要がある JSP ページへのアクセスを制御することができます。セキュリティ制限に基づいてアクセスを制御することもできます。アクセス制御が必要な JSP ページの例には、たとえばチェックアウト手順の中のページがあります。フロントコントローラデザインパターンの詳細については、34 ページの「フロントコントローラ」および57 ページの「フロントコントローラとしてのサーブレットの使用」を参照してください。

Dreamweaver バージョン 3.0 以前の `.dwt` を使用することができます。IDE では、`.dwt` テンプレートから JSP ページを生成するだけでなく、ソースエディタで `.dwt` ファイルを開いたり編集したりすることができます。選択したエディタで `.dwt` テンプレートが開かれるように、IDE を構成することも可能です。

page 指令の使用

page 指令を使用すると、特定の要求に関連しない JSP ページに関するグローバル宣言情報を指定することができます。たとえば、指令を使って次のことができます。

- セッション内のページの参加を設定する
- パッケージなどのクラスをページにインポートする
- エラーページを選択する

page 指令の詳細については、18 ページの「指令要素」を参照してください。

セッションの作成と無効化

セッションがまだ存在しない場合は、JSP ページを実行するとセッションが作成されます。その後のセッション関連ページには、セッションが無効化されるまでそのセッションが引き続き使用されます。無効化は、タイムアウトまたは明示的な呼び出しによって発生します。JSP スクリプトレット内で次の呼び出しを使用できます

```
session.invalidate();
```

次のコードを使用して、サーブレット内からセッションを無効化することもできます。

```
request.getSession().invalidate();
```

HTTP セッションの詳細については、21 ページの「HTTP セッション」を参照してください。

JSP ファイルの変更

IDE では、JSP タグのコード補完などの JSP 構文のサポートを提供しています。たとえば、Ctrl-Spacebar を押すとコード補完ボックスが表示されます。コード補完の詳細については、Core IDE オンラインヘルプの「HTML タグの補完」を参照してください。

JSP ソースコードの編集方法は、HTML の編集方法と似ています。このため、IDE では HTML タグと JSP タグの両方のサポートを提供しています。詳細については、JSP/サーブレットのオンラインヘルプの「JSPファイルやサーブレットファイルの編集」を参照してください。

IDE では、変更された JSP ページを含む Web モジュールを実行すると、アクセスした時点の JSP ファイルが保存されてコンパイルし直されます。詳細については、オンラインヘルプの「JSPファイルやサーブレットファイルの実行」を参照してください。

スクリプトレットとは?

スクリプトレットは、有効な Java コードを JSP ページに入力できるスクリプト要素です。宣言要素で宣言された変数とメソッドは、同じ JSP ファイル内の他のスクリプトレットで使用できます。スクリプト要素の詳細については、20 ページの「スクリプト要素」を参照してください。

スクリプトレットを使用するとき

過去には、多くの JSP 開発ガイドがスクリプトレットの使用について重点的に解説していました。現在では、スクリプトレットの使用を避けることが推奨されています。スクリプトレットを使用すると、JSP ページがわかりにくくてメンテナンスしづらいものになってしまいます。JSP ページの作成や変更を行うコンテンツ開発者は一般的に Java コードに慣れていません。このため、スクリプトレットコードに悪影響を及ぼすツールを使用する可能性があります。Java コードでは、JSP ファイル内で処理を実行しなければならない場合があります。たとえば、多数のチャンクに分割された大量の情報のテーブルや表示をフォーマットする場合がそうです。このような場合には、カスタムタグライブラリの機能の使用を考えてください。61 ページの「既存のタグライブラリの使用」を参照してください。または、プロパティに JSP タグでアクセスできるビュー作成ヘルパー Bean を作成することもできます。詳細については、39 ページの「ビュー作成ヘルパー」および53 ページの「追加のクラスまたは Bean の使用」を参照してください。

スクリプトレットの使用は、制限された環境で有効で、IDE によって完全にサポートされています。たとえば、後に Bean またはタグハンドラにカプセル化することを予定している新機能の初期のプロトタイプを作成する場合がそうです。

JSP の include の使用

モジュール形式のコンポーネントから複雑なページを簡単に作成するには、JSP ページに他のページを取り込みます。JSP ページに JSP ページと HTMLファイルまたは XML ファイルを取り込むことができます。これらのページを取り込むには、`<jsp:include>` アクションまたは `<%@include%>` 指令を使用します。変換および実行時を含む JSP のライフサイクルの詳細については、17 ページの「JSP ページのライフサイクル」を参照してください。JSP ページに複合ビューを取り込む方法の例については、51 ページの「複合ビューテンプレートの作成」を参照してください。

<jsp:include> アクションの使用

コードの中で JSP の include アクションを使用すると、取り込まれたファイルを使用した完全なページが要求時に構築されます。アクションを使用している JSP ページは、独自の実装クラスを持っています。これによって、ページ内の他の部分の処理が影響を受けたり、応答の HTTP ヘッダーが変更されることはありません。ページのコンテンツが動的で、そのページを含む JSP ファイルを変換した後に変更される可能性が高い場合は、<jsp:include> アクションを使用します。取り込むべきページが要求時までわからない場合も、<jsp:include> アクションを使用することをお勧めします。このような使い方が推奨されるのは、page 属性をたとえば次のような式によって設定することができるからです。

```
<jsp:include page="<%= dynamicPageName %>" flush="true">
```

次に、include アクションを使ってファイルを取り込む例を示します。

```
<jsp:include page="/foo.jsp"/>
```

<%@include%> 指令の使用

指令を使って指定した JSP include は、それを含む JSP ファイルの実装クラスに変換時に取り込まれます。したがって、これらによってページ内の他の部分が影響を受けたり、応答の HTTP ヘッダーが変更される可能性があります。コンテンツが静的で変更される可能性が低いページを取り込む場合は、<%@include%> 指令を使用してください。

次に、この指令を使ってファイルを取り込む例を示します。

```
<%@include file="bar.html"%>
```

複合ビューテンプレートの作成

次のコードサンプルでは、埋め込まれた HTML テーブル内で include アクションを使用しているテンプレート JSP ページが作成されます。ヘッダー、フッター、およびナビゲーションバーのセルには、常に同じ JSP ファイル (header.jsp、

footer.jsp、および navbar.jsp) が取り込まれます。メインのセルに取り込まれるコンテンツの JSP ファイルは、usePage 属性を使ってテンプレートに渡されます。

```
<table width="100%" border="0" cellspacing="0" cellpadding="0">
<colgroup span="2">
<col width="140">
<col>
</colgroup>
  <tr>
    <td colspan="2">
     <jsp:include page="header.jsp" flush="true"/>
    </td>
  </tr>
  <tr>
    <td align="left" valign="top" bgcolor="#dddddd">
      <jsp:include page="navbar.jsp" flush="true"/>
    </td>
    <td align="left" valign="top">
       <table width="100%" cellpadding="10px" border="0">
        <tr><td>
        <jsp:include page="<%= usePage%>" flush="true"/>
        </font></td></tr>
       </table>
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <jsp:include page="footer.jsp" flush="true"/>
    </td>
  </tr>
</table>
```

通常、テンプレートでは、次の 2 つの項目の開始と終了を宣言します。

- HTML ドキュメント
- 全体的なグリッドレイアウトを定義するテーブル

たとえば、開始タグと終了タグ (終了タグが必要な場合) を持つ完全な HTML 要素が記述されるように、取り込まれるファイルを設計します。

もう 1 つの方法は、特別に設計されたテンプレートタグを使用する方法です。Struts フレームワークに、テンプレートタグの例を見つけることができます。詳細については、以下のサイトにある Struts のユーザーガイドを参照してください。
http://jakarta.apache.org/struts/userGuide/struts-template.html

追加のクラスまたは Bean の使用

Web モジュールでは、Bean を使用して、データベースやフラットファイル (データを含む可能性があるプレーンテキストファイル) などの外部のリソースにアクセスすることができます。また、追加のユーティリティクラスを使用すると、サーブレット、Bean、または JSP スクリプトレットからアクセス可能なメソッドを使って Java 機能を実行することができます。Java クラスを JSP ページから使用できるようにするためには、パッケージに含める必要があることに注意してください。

IDE には、Web アプリケーション内に Bean 作成するためのテンプレートが用意されています。新規ウィザードで Bean のカテゴリをクリックし、Java Bean 項目を選択します。詳細については、Core IDE オンラインヘルプの「新しいファイルの作成」を参照してください。

JSP ページで使用する Bean を指定するには、`<jsp:useBean>` アクションを使用します。Bean に関するプロパティの取得には、`<jsp:getProperty>` アクションを、設定には `<jsp:setProperty>` アクションを使用します。

JSP ファイル内の値を使って Bean のすべての値を設定するには、`setProperty` アクションの `property` 属性に `*` を設定し、`value` 属性を削除します。Bean 内のプロパティの名前と要求パラメータの名前は一致している必要があります。通常は、各要求パラメータが HTML フォーム内の要素の 1 つに対応しています。

プロパティの値が空の場合、Bean 内の属性は変更されません。これはたとえば、ユーザーがフィールドに何も入力しなかったり、値をクリアした場合です。このような状況が発生する可能性がある場合は、次の例に示すように各値を明示的に名前で設定します。`*` は使用しないでください。

次に、JSP ページで Bean を使用する例を示します。

```
<jsp:useBean id="myFoo" scope="session" class="com.sun.FooBean"/>
```

次に、Bean を使用し、プロパティを設定する例を示します。

```
<jsp:useBean id="myBar" scope="page" class="com.sun.BarBean">
<jsp:setProperty id="myBar" property="blah" value="0"/>
</jsp:useBean>
```

次に、プロパティを取得する例を示します。

```
<jsp:getProperty id="myFoo" property="fooProp"/>
```

サーブレットの作成

サーブレットは、必ず Web モジュールの `WEB-INF/classes` ディレクトリ内か、そのディレクトリ内に作成するパッケージの中に作成してください。J2SE 1.4.0 以降では、別のクラスや JSP ページから Java クラスを参照する場合はその Java クラスはパッケージ内にあることが必要とされるため、サーブレットはパッケージに含めることが推奨されます。

エクスプローラで Web モジュールをマウントすると、`WEB-INF/classes` ディレクトリは IDE の内部クラスパスに含まれることに注意してください。

IDE では、新規ウィザードでサーブレットを作成するときに、新しいサーブレットの配備エントリを構成できます。この作業を行うには、ウイザードの「配備構成」パネルを使用します。IDE の外部で作成したサーブレットを Web アプリケーションに追加することもできます。ただし、このサーブレットは単独では実行できません。サーブレットを配備記述子に追加して、Web モジュール内の他の Web リソースと調整する必要があります。詳細については、JSP/サーブレットのオンラインヘルプの「JSP ファイルやサーブレットファイルの作成」を参照してください。

IDE で他の Java クラスや JSP ページに提供されているコード補完を、サーブレットでも利用できます。コード補完を使用して、使用可能なメソッドや値を知ることができます。詳細については、オンラインヘルプの「JSPファイルやサーブレットファイルの編集」を参照してください。

サーブレットのコンパイルは、ソースエディタのコンテキストメニューまたはエクスプローラのノードから実行できます。ソースエディタでサーブレットを開いたまま F9 キーを押すと、コンパイルが実行されます。IDE の外部で作成したサーブレットは、通常の Java クラスとして処理されます。サーブレット機能を使用するには、このクラスをサーブレットとして処理するように IDE に指示する必要があります。詳細については、オンラインヘルプの「サーブレットを Web モジュールに追加する」を参照してください。

配備記述子でのサーブレットの宣言

再利用可能なサーブレットを作成する場合は、配備記述子エントリが役に立ちます。配備記述子には、配備時にサーブレットコードをコンパイルし直さなくても変更できるデータが含まれています。サーブレットの機能を特定の配備状況に合わせて調整するための初期化パラメータを定義することができます。たとえば、サーブレットの `init` パラメータに、通貨、日付、時刻のフォーマットを指定することができます。

サーブレットをアプリケーションで使用するには、それらを宣言する必要があります。前述のように、この宣言は、新規ウィザードでのサーブレット作成手順に含まれています。ただし、それ以外の方法で Web モジュールに追加したサーブレットについては、配備記述子に宣言を追加する必要があります。

サーブレットエントリ

サーブレットエントリには、サーブレットを実装するクラスの名前か、JSP ページへのパスのいずれかが含まれています。Web アプリケーション内でサーブレットを識別する一意の名前も含まれています。

サーブレットマッピング

サーブレットにアクセスしたり、サーブレットで要求を受信するためには、サーブレットエントリとともに、1 つまたは複数のサーブレットマッピングが必要です。JSP ページをマッピングすることもできます。ただし、サーブレットの場合と異なり、JSP ページでは、マッピングしなくても要求を処理することができます。サーブレットマッピングは、名前付きのサーブレットまたは JSP ページを URL パターンと照合します。特定の条件を満たす場合に、サーブレットまたは JSP ページは起動されます。要求 URI のサーバー識別子の後に続く部分とコンテキストパスは URL パターン文字列に一致している必要があります。コンテキストパスとは、Web モジュールをサーバーに配備したときにその Web モジュールに関連付けられたパスです。URL パターンには、一定の文字列を指定することも、ワイルドカード文字を含めることもできます。`/catalog/*` の URL マッピングは、`/catalog/` が先頭に付く任意の要求パスに一致します。

起動時読み込み

サーブレットまたは JSP ページが「起動時に読み込み」として指定されている場合を除いて、サーブレットコンテナではいつでもリソースを初期化することができます。

サーブレットを起動時に読み込むと、コンテナが Web アプリケーションを起動したときにサーブレットがインスタンス化されて、その `init()` メソッドが呼び出されます。同様に、JSP ページを起動時に読み込んだ場合、JSP ページは起動中にコンパイルされて初期化されます。また、JSP ページは、後に変更されるたびにコンパイルされて初期化されます。

サーブレットを起動時に読み込むと、複数のリソースによって使用されるリソースをそのサーブレットで設定することができます。たとえば、サーブレットでサーブレットコンテキストにパラメータを追加することができます。サーブレットが起動時に読み込まれるように構成することによって、そのパラメータは、ほかのリソースがアクセスしようとしたときにいつでもアクセスできるようになります。起動時読み込みの値が負であるか、または設定されていない場合、コンテナは自由にサーブレットを読み込むことができます。この値が正の整数の場合、コンテナは起動時にサーブレットを初期化します。また、コンテナは、低い値を持つサーブレットを先に読み込み、高い値を持つサーブレットを後から読み込みます。

サーブレットエントリの追加

配備記述子にサーブレットを追加するには、web.xml ファイルを右クリックし、「プロパティ」を選択します。「プロパティ」ウィンドウの「配備」パネルで、「サーブレット」を選択します。「サーブレット」プロパティエディタでは、サーブレットエントリを追加、編集、または削除することができます。詳細については、JSP/サーブレットのオンラインヘルプの「「サーブレット」のプロパティエディタ」を参照してください。

サーブレットエントリの表示と変更

選択したサーブレットの配備記述子エントリを表示または変更するには、サーブレッ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。次に、「配備エントリ」フィールドをクリックしてサーブレットプロパティを編集します。

サーブレットの変更

IDE でのサーブレットの変更方法は、ほかの Java クラスを変更する場合と同様です。サーブレットや配備記述子を変更した場合は、Web モジュール全体を配備し直す必要があります。Sun ONE Application Server 7 を使用する場合に、サーバーインスタンスがすでに実行中の場合は、インスタンスはモジュールの全コンポーネントを動的に再度読み込みます。内部 Tomcat サーバーまたは Tomcat 4.0 サーバーを使用する場合は、Web モジュールのコンテキストメニューから「実行 (強制再読み込み)」を選択して Web モジュールを保存、再コンパイル、配備し、サーバーを再起動してからモジュールを実行できます。コンパイルし直されるのはサーブレットのみであることに注意してください。Web モジュールを実行またはデバッグするまで JSP ファイルはコンパイルし直されません。

サーブレットによって生成された HTTP 応答

サーブレットで HTTP 応答に書き込みを行うことができます。サーブレットからの HTML 出力の簡単な例として、新規ウィザード内にある IDEのサーブレットテンプレートの processRequest メソッドがあります。

コード例 4-1　　サーブレットからの HTML の出力

```
response.setContentType("text/html");
java.io.PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<head>");
out.println("<title>Hello World Servlet</title>");
```

コード例 4-1　　サーブレットからの HTML の出力 (続き)

```
response.setContentType("text/html");
out.println("</head>");
out.println("<body>");
out.println("Hello, World!");
out.println("</body>");
out.println("</html>");
out.close();
```

必ずしもサーブレットで HTML を出力しなくてもかまいません。代わりに、サーブレットを使って `HTTPRequest` オブジェクトおよび `HTTPResponse` オブジェクトを変更することができます。たとえば、フロントコントローラサーブレットは、要求を別のサーブレットまたは JSP ファイルに転送し、その応答オブジェクトに対して書き込みを行います。

フロントコントローラとしてのサーブレットの使用

フロントコントローラを構築する手順は、次のとおりです。

1. IDE のサーブレットテンプレートを使用して新しいサーブレットを作成します。メインメニューバーの「ファイル」メニューから、「新規」を選択し、「JSP & サーブレット」を選択して、「サーブレット」を選択します。

2. 新規サーブレットウィザードの「配備構成」パネルで、獲得したい要求 URL に一致する URL パターンを指定します。たとえば、`/ShowProducts/*` を指定します。

   サーブレットコンテナは、次の 2 つの場合にサーブレットへの要求パスの照合を試みます。まず、コンテナが最初に要求を受信したときに照合が試みられます。また、`forward` または `include` によって要求が内部的にディスパッチし直されたときに、ふたたび照合が試みられます。パス `/*` を使用して別のリソースへの転送を行うサーブレットをマッピングすることはできません。そのようなマッピングを行うと、サーブレットの再帰呼び出しが発生します。

3. サーブレットが作成されたら、`processRequest` メソッドに処理コードを挿入して、要求が適切なページに転送されるようにします。たとえば、次のコードサンプルは、以下に示されている要求の処理方法を示しています。

```
http://my.company.com/ShowProducts?product=stuffedbear
```

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
java.io.IOException {
    String sendTo;
    if (request.getQueryString().equals("product=stuffedbear"))
        sendTo = "/WEB-INF/showStuffedBearInfo.jsp";
    else sendTo = "/WEB-INF/noProductSpecified.jsp";

    RequestDispatcher sendPage =
        getServletContext().getRequestDispatcher(sendTo);
    sendPage.forward(request, response);
}
```

この場合、フロントコントローラサーブレットの `ShowProducts` は、照会文字列 `product=stuffedbear` を使用して適切なビューを選択します。そのビューは、ぬいぐるみの熊に関する製品説明のページで構成されていると考えられます。フロントコントローラサーブレットは、要求をその JSP ページに転送します。

アクセスするページの順番をユーザーではなくアプリケーションによって制御する場合は、フロントコントローラを使ってページフローを指示します。この場合の例としては、チェックアウト処理を実装する一連のページがあります。ショッピングカート内の製品の購入手続きを行っている間は、ユーザーは、手順の順番に従わずにチェックアウトのページをブックマークしたり、元に戻ったりすることはできません。このシナリオを実装するには、Web モジュールの `WEB-INF` ディレクトリ内にチェックアウト用の JSP ページを配置します。ブラウザからの要求がこの場所に直接アクセスすることはできません。この場所にアクセスするには、これらのページのための `RequestDispatcher` API を使用している別のリソースを介する必要があります。この状況が前のコード例に示されています。この例では、フロントコントローラサーブレットを作成して、ページの取得方法と使用方法を制御することができます。フロントコントローラはチェックアウト処理中の入力を処理し、ユーザーに対して次に表示するページを決定します。

フロントコントローラデザインパターンの詳細については、34 ページの「フロントコントローラ」を参照してください。

追加のクラスまたは Bean の使用

サーブレットでは、ほかの Java クラスと同じ方法で追加のクラスや Bean を取得することができます。サーブレットで使用するには、クラスや Bean をパッケージに含める必要があることに注意してください。IDE では、新規ウィザードを使って簡単に Bean を作成できます。詳細については、Core IDE オンラインヘルプの「新しいファイルの作成」を参照してください。

フィルタの作成

新規ウィザードでは、フィルタを作成して Web アプリケーションに追加することができます。追加方法は、JSP ファイルやサーブレットの作成方法と同様です。サーブレットの場合と同様に、新規ウィザードのチェックボックスを使用して、デフォルト設定による配備記述子要素を生成します。詳細については、JSP/サーブレットのオンラインヘルプの「フィルタの作成」を参照してください。

配備記述子でのフィルタの宣言

フィルタを宣言するには、配備記述子内のフィルタ要素を使用して次の項目を定義します。

- フィルタ名。フィルタをサーブレットまたは URL にマッピングするために使用されます。
- フィルタクラス。コンテナで、フィルタタイプを識別するために使用されます。
- フィルタの初期化パラメータ。`init-params` と呼ばれます。

また、次の項目も定義することができます。

- 大きいアイコンと小さいアイコン
- テキストによる説明
- ツール操作の表示名

詳細については、JSP/サーブレットのオンラインヘルプの「配備記述子へのファイルの追加」および「フィルタ」プロパティエディタを参照してください。

Web モジュールからフィルタを使用するには、`<filter>` タグを使って配備記述子でこれを宣言する必要があります。フィルタエントリには、そのフィルタを実装するクラスの名前 (`filter-class`) とフィルタの一意の名前が含まれます。フィルタの初期化パラメータを指定することもできます。

フィルタで要求を受信するには、サーブレットのマッピングと同様に、配備記述子に `<filter-mapping>` タグの形で 1 つまたは複数のマッピングを追加する必要があります。サーブレットの論理名によってフィルタを特定のサーブレットにマッピングできます。また、URL パターンを使って、サーブレットや静的なコンテンツのグループにフィルタをマッピングすることもできます。

フィルタマッピング要素には、Web モジュールのために定義されているフィルタマッピングの数が示されます。フィルタマッピングには、特定のフィルタで処理すべき URL パターンを指定します。追加のフィルタマッピングを指定するには、「フィルタマッピング」プロパティエディタを使用します。詳細については、JSP/サーブレットのオンラインヘルプの「Web モジュールの配備プロパティの編集」および「フィルタマッピング」プロパティエディタを参照してください。

サーブレットコンテナは、Web アプリケーションに対する要求を受信すると、フィルタチェーンを構築します。このチェーンには、要求 URL およびコンテキストパスと URL マッピングが一致するすべてのフィルタが、配備記述子内でのフィルタ順に並んでいます。コンテキストパスとは、Web モジュールをサーバーに配備したときにその Web モジュールに関連付けられたパスです。要求がコンテナに入ってくると、このチェーンに含まれるフィルタが起動されます。サーブレットと異なり、フィルタチェーンは内部的なディスパッチ用には構築されません。したがって、フィルタを `/*` にマッピングして、要求がコンテナに入ってくるたびに呼び出されるようにすることができます。

HTTP 要求および応答の処理

フィルタを使用して HTTP 要求や HTTP 応答を変更することができます。フィルタは、認証に使うことができます。たとえば、JSP ファイルまたはサーブレットが要求されたときに、要求された Web コンポーネントの表示がエンドユーザーに許可されているかどうかをフィルタで判断することができます。セッション情報にはエンドユーザー名が含まれています。この名前がブランクの場合、フィルタは要求をログインページにルーティングします。名前がブランクでなくても、ページを表示するための承認がない場合は、別の結果が生じます。フィルタは、必要な承認について説明するページに要求をルーティングします。フィルタを使用して、特定のクライアントに応答が送信されるたびにログエントリを作成することもできます。このエントリは、要求とセッションの情報に基づいて作成されます。

例については、次の場所にある Java Pet Store の `SignOnFilter` クラスを参照してください。

`http://java.sun.com/blueprints/code/jps13/src/`

タグライブラリの使用

前述のように、タグライブラリは、タグライブラリの機能セットを実装するタグハンドラクラスのセットと、ライブラリ内のタグを記述し、各タグをタグハンドラにマッピングするタグライブラリ記述子 (TLD) から構成されています。独自のカスタムアクションやカスタムタグを作成することによって、アクションの標準セットを拡張できます。これによって、アプリケーション内のコードを機能単位でモジュール化およびカプセル化することができ、コードの再利用可能性が広がります。適切に設計すれば、フォーマットからロジックを明確に切り離すことができます。この切り離しにより、JSP ページに埋め込まれた Java コードを使用する必要がなくなります。

IDE では次のことをサポートしています。

- **既存のタグライブラリの Web アプリケーションでの使用。**JSP タグライブラリリポジトリは、JSTL (JavaServer Pages Standard Tag Library) を提供します。詳細については、61 ページの「既存のタグライブラリの使用」および62 ページの「タグライブラリから JSP ページへのカスタムアクションの挿入」を参照してください。
- **新しいタグライブラリの開発。**タグライブラリエディタを使用して、独自のタグライブラリを簡単に作成および変更できます。詳細については、64 ページの「独自のタグライブラリの開発」を参照してください。
- **タグライブラリのパッケージ化と配備。**カスタムタグライブラリを JAR としてパッケージ化し、それを Web モジュールに追加して配備することができます。詳細については、96 ページの「カスタムタグライブラリのパッケージ化と配備」を参照してください。

既存のタグライブラリの使用

IDE では、JSP タグライブラリリポジトリを使用して、既存のカスタムタグライブラリを簡単に管理することができます。JSP タグライブラリリポジトリには、初期状態で Apache Jakarta グループの JSTL (JavaServer Pages Standard Tag Library) 1.0 が含まれています。

JSP タグライブラリリポジトリには、独自に作成したタグライブラリや外部のソースからダウンロードしたタグライブラリを追加することができます。「ツール」メニューから「JSP タグライブラリリポジトリ」を選択します。表示された「JSP タグライブラリリポジトリ」ダイアログで、リポジトリにタグライブラリを追加します。

Web モジュールへのタグライブラリの追加

アプリケーションでタグライブラリを使用するには、まずタグライブラリを Web モジュールに追加する必要があります。

Web モジュールの WEB-INF ノードの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。タグライブラリリポジトリまたはファイルシステムでタグライブラリを検索します。一部のタグライブラリは、単一の JAR ファイルとして配布されています。その他のタグライブラリは、依存する追加の JAR ファイルを含んでいる場合があります。

JSP タグライブラリリポジトリから `standard` を選択した場合は、JSTL に含まれるすべてのタグライブラリが `WEB-INF/lib` ディレクトリに追加されます。JSTL の詳細については、28 ページの「タグライブラリ」を参照してください。

外部のソースのタグライブラリの使用

外部のソースのタグライブラリを JSP ファイルに追加して使用するには、関連する JAR ファイルを JSP タグライブラリリポジトリに追加します。リポジトリでは、Web モジュールに追加できるようにタグを保存することができます。リポジトリに配置することによって、ほかのカスタムタグライブラリは、Web モジュールにいつでも追加できるようになります。

メインメニューバーの「ツール」メニューから「JSP タグライブラリリポジトリ」を選択します。「JSP タグライブラリリポジトリ」ダイアログで「追加」をクリックし、希望するライブラリを選択します。

詳細については、オンラインヘルプの「カスタムタグライブラリをリポジトリに追加する」を参照してください。

タグライブラリ記述子

TLD (タグライブラリ記述子) は、タグライブラリを定義する XML ドキュメントです。サーブレットコンテナでは、特定の JSP ページに対するカスタムアクションを解釈するために、タグライブラリの TLD を使用します。JSP ページは `taglib` 指令によってタグライブラリを参照します。TLD は最上位レベルでタグライブラリ全体の属性を定義します。これらのプロパティには、タグライブラリのバージョン番号や、対象とするサーブレットコンテナのバージョン番号などがあります。下位レベルでは、TLD はライブラリ内の各タグを定義します。

IDE を使用すると、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。作成した TLD は、エクスプローラからメニューコマンドを使って編集することができます。カスタマイザウィンドウと要素を使って TLD を編集することもできます。

タグライブラリから JSP ページへのカスタムアクションの挿入

カスタムアクションは一般にカスタムタグとも呼ばれます。ただし、「カスタムアクション」という用語は、一般的には、JSP ページで使用されるコード構造を意味します。**カスタムタグ**という用語は、一般的には、カスタムアクションの機能を実装するコードを意味します。

タグライブラリの機能を使用するには、JSP ページ内のカスタムアクションをコード化します。カスタムアクションでタグライブラリを使用できるようにするには、JSP ページで `taglib` 指令を使って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 prefix="mt" uri="/WEB-INF/lib/myTagLib.jar" %>
```

taglib 指令の uri 属性は、TLD またはパッケージ化されたタグライブラリの JAR ファイルを参照します。JAR ファイルには、TLD とタグハンドラ Bean が両方含まれています。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。または、Web モジュールの配備記述子で uri を指定することもできます。

前の例の uri 属性は、Web モジュールのルートへの相対的なパスを指定しています。先頭のスラッシュは、Web モジュールのルートを表します。

<%@taglib> 指令内の標準タグライブラリにはコード補完が機能します。たとえば、前の例では、uri という属性名の後に " を入力するだけで十分です。

uri まで入力したら、次に =" を入力し、Ctrl-Spacebar を押します。この Web コンテキストで利用可能な URI のリストが表示されます。

JSP 1.2 仕様では新たに、一部のタグライブラリで、JAR ファイル内に複数の TLD を含めることができるようになっています。それらのタグライブラリを取得して使用するには、WEB-INF/lib ディレクトリに JAR ファイルを置き、taglib 指令の uri 属性に、必要な TLD の URI を使用します。

JAR ファイル内の TLD ファイルの URI を確認するには、WEB-INF/lib ディレクトリに JAR を置きます。これによって、JAR ファイルは、「ファイルシステム」タブの Web モジュールの下にマウントされます。マウントされた JAR ファイルをエクスプローラで開き、META-INF ディレクトリを選択すると、そこに TLD ファイルが入っています。TLD ファイルをダブルクリックすると、カスタマイザに URI が表示されます。

たとえば、JSTL の standard.jar ファイルのコア TLD の taglib 指令は次のようになります。

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
```

taglib 指令は、JSP ページ内の、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置く必要があります。

タグライブラリの開発時には、taglib 指令がタグライブラリの JAR ファイルではなく TLD ファイルを参照するようにします。IDE によってタグハンドラのクラス名が TLD に挿入されます。タグハンドラもまた、Web モジュールのクラスパス内になければなりません。WEB-INF/classes ディレクトリにタグハンドラを配置します。

taglib 指令の prefix 属性を使用して識別子を指定します。JSP ページ内に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ときには、この識別子を使用します。前の例では、taglib 指令とカスタムアクション (指定したタグライブラリ内で定義されている) を同じ JSP ページ内に置く必要があります。このアクションは、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 はタグの名前です。

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

タグ名 (この場合、table) とタグハンドラ Bean のマッピングは、TLD ファイルで指定されます。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」ウィンドウで、このマッピングを編集します。

独自のタグライブラリの開発

JSTL や、Struts フレームワークに含まれているタグライブラリのほかにも、多数の便利なタグライブラリが入手可能です。Struts フレームワークの詳細については、42 ページの「Struts」を参照してください。また、カスタムタグライブラリ専門の Web サイト http://jsptags.com/ もソースとして利用できます。

多くの場合は、ビジネスロジックを独自のカスタムタグにカプセル化すると便利です。これによって、Web モジュール内の JSP ページでビジネスロジックを簡単に取得して使用できるようになります。タグのフォーマットは Web デザイナにとって扱いやすいものです。このため、カスタムタグライブラリを作成することによって、これらの機能を簡単に JSP ページに挿入することができます。

この節では、IDE がカスタムタグライブラリの開発のために提供しているサポートについて説明します。このサポートには、次のものが含まれます。

- タグライブラリの作成と仕様
- タグの作成と仕様
- タグ属性の作成と仕様
- スクリプティング変数の作成と仕様
- タグハンドラの生成

タグライブラリおよびタグの作成

前述のように、タグライブラリは、タグライブラリの機能セットを実装するタグハンドラクラスのセットと、ライブラリ内のタグを記述し、各タグをタグハンドラにマッピングする TLD から構成されています。新しいカスタムタグライブラリを作成すると、実際には新しい TLD ファイルが生成されます。

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。タグライブラリを作成する 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。コンテキストメニューから、「新規」を選択し、「JSP & サーブレット」を選択して、「タグライブラリ」を選択します。この操作によって TLD ファイルが作成されます。タグライブラリカスタマイザを使用して、カスタムタグライブラリのプロパティを定義します。

タグライブラリカスタマイザで、タグライブラリの短縮名、表示名、タグライブラリバージョン、および URI を指定します。タグハンドラ生成ルートを Web モジュールの `WEB-INF/classes` ディレクトリに設定します。生成したタグハンドラクラスは、このディレクトリに配置されます。コード生成オプションを設定したり、タグライブラリの機能に関する説明的な情報を入力することもできます。タグライブラリカスタマイザに表示されるプロパティの詳細については、JSP/サーブレットのオンラインヘルプの「タグライブラリのカスタマイズ」を参照してください。

タグライブラリカスタマイザの内容には、選択されているタグライブラリのプロパティが反映されています。タグライブラリのプロパティを指定したら、タグを追加したりカスタマイズすることができます。

カスタムタグライブラリの作成と使用の詳細については、`http://java.sun.com/products/jsp` で入手可能な JavaServer Pages Specification, Version 1.2 を参照してください。

IDE を使用して作成したタグライブラリは、Web モジュールに直接追加することをお勧めします。必ず Web モジュールの `WEB-INF` ディレクトリにタグライブラリ TLD を作成してください。また、`classes` ディレクトリにあるパッケージ内にタグハンドラを生成する必要があります。Web モジュールを配備する準備ができたら、タグライブラリをパッケージ化します。次に、`classes` ディレクトリ内のタグハンドラパッケージを JAR ファイルで置き換えます。さらにこの JAR ファイルを `lib` ディレクトリに配置します。パッケージ化の詳細については、96 ページの「カスタムタグライブラリのパッケージ化と配備」を参照してください。

タグの追加とカスタマイズ

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。

エクスプローラでタグを作成することができます。タグを追加する TLD を表すノードを右クリックします。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」ダイアログでタグを編集します。詳細については、JSP/サーブレットのオンラインヘルプで「タグのカスタマイズ」の「タグカスタマイザのフィールド」の節を参照してください。

エクスプローラで既存のタグを編集することもできます。カスタマイズするタグを右クリックします。コンテキストメニューから、「カスタマイズ」を選択します。「タグカスタマイザ」ダイアログでタグを編集します。詳細については、JSP/サーブレットのオンラインヘルプで「タグのカスタマイズ」の「タグカスタマイザのフィールド」の節を参照してください。

「タグカスタマイザ」ダイアログにはいくつかのタブがあります。「一般」タブには、TLD に挿入される値が表示されます。「コード生成」タブには、タグ用に生成されたタグハンドラに関するオプションが表示されます。

タグの本体で発生する内容の種類を選択する必要があります。指定できるオプションについては、66 ページの「カスタムアクションの本体の処理方法の指定」を参照してください。

また、生成されたタグハンドラクラスの Java パッケージの名前も指定する必要があります。デフォルト値はタグライブラリの短縮名です。

生成されたタグハンドラに、親タグ (つまり、そのタグを含んでいるタグ) を検索するためのコードを含める場合は、「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が見つかった場合、そのインスタンスは、「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です。「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。詳細については、オンラインヘルプで「タグのカスタマイズ」の「タグカスタマイザのフィールド」の節を参照してください。

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。カスタムアクションは、ほかのアクション、スクリプト要素、プレーンテキストを囲む開始タグと終了タグを持っています。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>
```

カスタムアクションの本体の処理方法の指定

本体の処理方法を指定するには、「タグカスタマイザ」ダイアログの「本体の内容」フィールドを使用します。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このウィンドウを表示します。その後、`JSP`、`empty`、または `tagdependent` の中から値を 1 つ選択します。

表 4-1 では、各選択肢の意味を説明しています。

表 4-1 「タグカスタマイザ」ダイアログの「本体の内容」フィールドの選択肢の意味

「本体の内容」フィールド	意味
JSP	本体の内容は任意です。サーブレットコンテナは、「JSP 要素を評価した後、」本体をタグハンドラに渡します。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 オブジェクトに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。サーブレットコンテナは、JSP 要素を評価せずに、本体をタグハンドラに渡します。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 オブジェクトに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。本体がある場合、本体は通常の JSP 処理によって単純に出力に渡されます。本体を処理するタグハンドラは、javax.servlet.jsp.tagext.BodyTag も実装する必要があります。このインタフェースは、この処理を行うための追加メソッドを提供します。

タグ属性の追加とカスタマイズ

タグ属性は、タグに関連付けられたパラメータです。これらのパラメータは、タグの処理中に使用する値を指示または提供します。

タグ属性を作成するには、「新しいタグ属性を追加」ダイアログを使用します。「タグ属性カスタマイザ」ダイアログを使用すると、既存のタグ属性を編集することができます。

「新しいタグ属性を追加」ダイアログでは、新しいタグ属性の各種プロパティを指定します。「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」ダイアログの詳細については、オンラインヘルプの「タグ属性のカスタマイズ」を参照してください。

新しいタグ属性がタグに追加されると、タグ属性カスタマイザが表示されます。このカスタマイザで属性を編集することができます。.

属性を作成するときに考慮しなければならないフィールドが 3 つあります。

- **「必須属性」チェックボックス。**これを選択した場合は、タグが呼び出されるたびにその属性に対する引数を指定する必要があります。デフォルトでは、False に設定されています。

- **「要求時に評価される値」ラジオボタン。**これを選択した場合は、属性の値を要求時に動的に計算することができます。デフォルトでは、`True` に設定されています。この値は、次に説明する「JSP 変換時に評価される値」属性と互いに排他的な関係にあります。
- **「JSP 変換時に評価される値」ラジオボタン。**これを選択した場合、属性の値は静的で、変換時に決定されます。デフォルトでは、`False` に設定されています。この値は、前に説明した「要求時に評価される値」属性と互いに排他的な関係にあります。

指定できる属性プロパティの種類の詳細については、オンラインヘルプで「タグ属性のカスタマイズ」の「タグ属性の情報」の節を参照してください。

スクリプティング変数の追加とカスタマイズ

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、式またはスクリプトレットで使用することができます。

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」ダイアログで、新しいスクリプティング変数の各種プロパティを指定します。スクリプティング変数の作成時には、「変数型」フィールドが特に重要です。次に、コンボボックスのリストから標準の型を選択するか、Java クラス名を入力します。

タグスクリプティング変数カスタマイザを使用すると、既存のスクリプティング変数のプロパティを編集することができます。スクリプティング変数を編集するには、エクスプローラウィンドウで変数のノードを右クリックし、コンテキストメニューから「カスタマイズ」を選択します。

「新しいタグスクリプティング変数の追加」ダイアログとタグスクリプティング変数カスタマイザの詳細については、オンラインヘルプの「スクリプト変数のカスタマイズ」を参照してください。スクリプティング変数のプロパティの詳細については、オンラインヘルプで「スクリプト変数のカスタマイズ」の「タグスクリプティング変数の情報」の節を参照してください。

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。タグハンドラクラスは、IDE によって生成されます。この生成には、タグカスタマイザに設定されているプロパティと、タグに追加したタグ属性およびスクリプティング変数が含まれます。これにタグを追加し、属性およびスクリプティング変数を追加または変更して、タグライブラリをカスタマイズします。このプロセスの間にタグハンドラを再生成することができます。この作業によって、タグハンドラのコードに加えた編集を失なうことなく、変更内容を反映することができます。詳細については、オンラインヘルプの「タグハンドラの生成」を参照してください。

生成されるタグハンドラ

生成されたコードは、タグライブラリの「タグハンドラ生成ルート」で指定されているパッケージディレクトリに入れられます。この属性は、タグカスタマイザで設定するコード生成プロパティです。この値がブランクの場合のデフォルトのタグハンドラ生成ルートは、タグライブラリを含むディレクトリです。

前述のように、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、対応するカスタムアクションに適したインタフェースを実装します。さらに、タグハンドラに必要なクラスメンバー (フィールド、メソッド、プロパティなど) 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 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に必要なメソッドは必ず含まれています。

生成されるクラスメンバーは、TLD で宣言した属性およびスクリプティング変数によって異なります。たとえば、`myAttribute` と呼ばれる属性を宣言すると、`myAttribute` と呼ばれるプロパティがタグハンドラ内に生成されます。

生成されるメソッド

表 4-2 は、タグハンドラを生成したときに IDE によって作成されるメソッドを示しています。プロパティを取得および設定するためのメソッドについては示されていません。アスタリスク (*) が付いているメソッドは、そのメソッドが `Tag` インタフェースおよび `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、ほかのメソッド、つまりヘルパーメソッドを呼び出します。

`Tag` インタフェースおよび `BodyTag` インタフェースに含まれるメソッドがすべて生成されるわけではありません。タグハンドラクラスは、`TagSupport` ヘルパークラスまたは `BodyTagSupport` ヘルパークラスを拡張して生成します。これらのヘルパークラスは、それぞれのインタフェース内のすべてのメソッドを実装します。生成されるのは、オーバーライドする必要のあるメソッドだけです。`Tag` インタフェースや `BodyTag` インタフェースのその他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 4-2　タグハンドラに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には、カスタムアクションに必要な機能を提供するためのプログラミングロジックをタグハンドラクラスに追加します。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。それらを追加した場合は、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します。このとき、タグハンドラのメソッドのうち、再生成されるものと再生成されずにそのまま残るものがあります。

IDE によって再生成されるメソッドは、doStartTag、doEndTag、および doAfterBody です。これらのメソッドをソースエディタで編集することはできません。このような制限があるのは、変更を行っても、タグハンドラを生成し直したときにオーバーライトされてしまうからです。

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。

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。IDE では、タグの最初に実行する必要がある処理には、otherDoStartTagOperations メソッドを使用します。また、本体を評価する必要があるかどうかを示す論理値を戻すために 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。

表 4-3 に、再生成されるメソッドと編集できるメソッドを示します。

表 4-3 タグハンドラの編集できるメソッド

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`

タグライブラリのテスト

IDE ファイルシステム内でも、既存の Web モジュールのファイルシステム内でも、タグライブラリを作成することができます。まだ Web モジュールの中にないタグライブラリをテストする場合は、以下の手順に従ってください。

1. タグライブラリのファイルシステムのルートを右クリックし、「ツール」メニューから「ファイルシステムを Web モジュールに変換」を選択します。`.tld` ファイルと、生成されてコンパイルされた Java タグハンドラクラスは、そのまま残しておきます。
2. *your-tag-lib*`.tld` の配備記述子内にタグライブラリ要素を追加します。
3. JSP ページを作成し、参照を新しいタグに追加します。
4. Web モジュールを配備して実行します。

タグライブラリのテストを終了したら、それを JAR ファイルとしてパッケージ化し、ほかの Web モジュールに取り込みます。詳細については、JSP/サーブレットのオンラインヘルプの「カスタムタグライブラリのパッケージを配備」を参照してください。

他の Web モジュールでこのタグライブラリを簡単に使用できるようにするには、新しいタグライブラリの JAR ファイルを IDE 内のタグライブラリリポジトリに追加します。詳細については、JSP/サーブレットのオンラインヘルプの「カスタムタグライブラリをリポジトリに追加する」を参照してください。

データベースの使用

Web アプリケーション開発においては、データベースとのやりとりは重要な問題です。JSP ファイルやサーブレットの動的な性質を Web モジュール内で有効に活用するために、データベースに保管された情報が使用されます。したがって、このデータへのアクセスはきわめて重要です。

アプリケーション内でデータベースを利用する場合は、データベースにアクセスするための Bean を作成します。この Bean へのアクセスにはタグハンドラを使用します。

IDE には、JDBC ドライバを提供しているデータベースを表示したり変更するためのいくつかのツールが用意されています。詳細については、『Java DataBase Connectivity の使用』を参照してください。

Web アプリケーション内から JDBC を使用してデータベースにアクセスすると、次のことができます。

1. データベース接続 (接続プールやキャッシュの管理を含む) を初期化するためのメソッドを持つデータアクセス Bean を作成する
2. 照会によって特定の情報にアクセスする
3. 行セット内の情報を表示および更新する

データアクセス Bean の作成例については、次の Web サイトにある「Duke痴 Bakery, Part II, A JDBC Order Entry Prototype - Continued」を参照してください。

`http://developer.java.sun.com/developer/technicalArticles/Database/dukesbakery2/`

これらのデータアクセス Bean は、サーブレットやフィルタクラスから直接使用することができます。また、JSP ファイル内から `jsp:useBean` タグを使用してこれらの Bean を参照することもできます。詳細については、53 ページの「追加のクラスまたは Bean の使用」を参照してください。

IDE は、デフォルトで PointBase データベースと内部の Sun ONE Application Server 7 が連携動作するように構成されます。PointBase データベースの起動については、『Sun ONE Studio 5, Standard Edition インストールガイド』の「IDE でのデータベースの使用」を参照してください。Tomcat サーバーとのデータベース接続の設定については、82 ページの「Web サーバー環境をデータベースアクセス用に設定する」を参照してください。

その他のデータベースを使用したり、その他の Web サーバーに配備する場合は、2 つのオプションがあります。データベースのドライバファイルを次の 2 つの場所のいずれかに追加します。

- Web モジュールの `WEB-INF/lib` ディレクトリ

- Web サーバーの共通の `lib` ディレクトリ。

特定のデータベースを使用するように IDE のデータベースエクスプローラを構成することもできますが、その場合も Web モジュールまたは Web サーバー内に適切なドライバを配置する必要があります。IDE のデータベースエクスプローラを使用して、データベースへの接続とそれに使用する適切なドライバへのアクセスを確認します。

タグを使って照会や表示の作業を簡易化するために、カスタムタグライブラリを Web モジュールに追加することができます。詳細については、64 ページの「独自のタグライブラリの開発」を参照してください。

次の目的のためのカスタムラタグを作成することをお勧めします。

- ユーザーへのデータ表示
- データの照会
- データベース内のデータの更新

これらのタグを Web モジュールに追加すると、JSP ページに照会や表示を簡単に追加できるようになります。また、これによって、ビジネスロジックのコードをエンドユーザーに対する表示から明確に切り離すことができます。IDE で開発した、データベースにアクセスする Web アプリケーションの例については、『J2EE チュートリアル』を参照してください。

これらのタグのタグハンドラクラスは、データアクセス Bean 内のメソッドに直接アクセスすることができます。タグライブラリを使用するには、タグライブラリを JSP ファイルにインポートします。詳細については、60 ページの「タグライブラリの使用」を参照してください。

JSTL 内のデータベースアクセスタグについては、次の Web サイトを参照してください。

`http://jakarta.apache.org/taglibs/doc/standard-doc/Overview.html`

第5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開発方法を検討済みのユーザーを対象としています。また、IDE を使ってアプリケーションを実行、デバッグ、および配備する準備が整っていることを前提としています。この章では、全体的な観点から、アプリケーションのテスト段階と修正段階で実行するタスクを順序付けて説明します。次に、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。

標準コンポーネントからの Web アプリケーションの開発方法については、第 4 章を参照してください。標準コンポーネントには、JSP ページ、サーブレット、フィルタ、および Bean があります。

IDE を使って Web アプリケーションを実際に構築する前に、『Sun ONE Studio 5 Web アプリケーションチュートリアル』でひと通り学習しておくことをお勧めします。このチュートリアルには、Web モジュールの配備プロセスについて、さまざまな面からの詳しく説明しています。

タスクの実行とデバッグ

この節では、IDE を使用した Web モジュールの実行に必要なタスクについて概説します。Web アプリケーションで作業する場合には、処理を繰り返して行わなければならないことがあります。ユーザーは、コンポーネントを修正するたびに、アプリケーションを配備し直して正しく機能していることをチェックします。アプリケーションに新しいコンポーネントを追加するたびに、このプロセスを繰り返します。それぞれのタスクは、この章の後ろの節で詳しく説明しています。JSP/サーブレットモジュールの IDE オンラインヘルプにも、これらのタスクの情報があります。

IDE を使用して Web モジュールを実行する手順は、次のとおりです。

1. コンポーネント (JSP ページ、サーブレットなど) を変更します。このプロセスは、第 4 章でそれぞれのコンポーネントタイプについて説明しています。

2. Web モジュールを設定します。IDE では、配備の詳細な設定まで考慮されています。IDE には、配備記述子ファイルを編集するための、プロパティシートと呼ばれるエディタが用意されています。76 ページの「Web モジュール配備記述子の構成」を参照してください。

3. WEB-INF ノードを右クリックし、コンテキストメニューから「配備」を選択して、Web モジュールを配備します。

4. IDE でアプリケーションをテストします。81 ページの「Web モジュールのテスト実行」を参照してください。

5. 「HTTP モニター」ウィンドウを使用して、HTTP トランザクションのデータフローを追跡します。このモニターを使用して、JSP ページとサーブレットで送信されたデータ、および Webアプリケーション内のデータを調査します。HTTP モニターは、プロセスのデバッグ中に使用して以前の要求に応答することもできます。HTTP モニターの詳細については、86 ページの「HTTP モニターを使用した Web アプリケーションのデバッグ」を参照してください。

6. 必要な場合は、ソースレベルのデバッグを使用して 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ます。詳細については、91 ページの「ソースレベルのデバッグ」を参照してください。

7. Web モジュールを WAR ファイルとしてパッケージ化し、配備します。Web アプリケーションのアーカイブは、Web アプリケーションのすべてのコンポーネントが含まれているファイルです。詳細については、94 ページの「Web アプリケーションのパッケージ化」を参照してください。

Web モジュール配備記述子の構成

配備記述子ファイルは、Web モジュールの `WEB-INF` ディレクトリにあります。このファイルは、Web モジュールの開発環境、つまりサーブレットコンテナに対して必要な設定情報を提供します。配備記述子には、Web アプリケーションで使用するサーブレット、フィルタ、またはタグライブラリを指定できます。また、外部リソースとセキュリティの要件、環境パラメータ、およびほかのコンポーネント固有のパラメータとアプリケーション固有のパラメータを指定する場合もあります。

IDE では、次の 2 つの方法で配備記述子を変更することができます。

- `web.xml` ファイルについて、プロパティシートで配備記述子の要素を検索する
- ソースエディタで `web.xml` ファイルを開いて、それを手動で編集する

Web モジュールには、JSP ファイルとサーブレットのみが含まれている単純なものがあります。内部サーバー上に単純な Web モジュールを配備する場合は、配備記述子を変更する必要はない場合もあります。多くの変更は IDE によって行われるため、手間を省くことができます。ただし、場合によっては、次の配備記述子要素をユーザーが変更する必要があります。

- **開始ファイル一覧。**Web アプリケーションが末尾にスラッシュの付いたパス名の要求を受け取った場合に、サーブレットコンテナが検索するリソース。開始ファイルの初期設定は、`index.jsp`、`index.html`、および `index.htm` です。
- **コンテキストパラメータ。**アプリケーション内の多数のリソースで使用される、(通常は不変の) 共有リソースです。これらのパラメータによって、コードを変更せずにアプリケーションの特定の値を変更することができます。コンテキストパラメータはデータベースアクセス情報を提供することが多いのですが、これは配備プラットフォーム間で異なります。
- **MIME タイプ。**アプリケーションは、ファイルの拡張子を MIME タイプに関連付けることができます。
- **セッションのタイムアウト。**Web コンテキストによって作成されたすべてのセッションに対して、セッションのデフォルトのタイムアウト間隔を分単位で定義します。セッションのタイムアウトがゼロより小さい値に設定されている場合は、コンテナはセッションをタイムアウトしません。通常、この設定は、配備されたアプリケーションでは推奨しません。
- **エラーページ。**アプリケーション内での失敗に応答してエラーページのエントリが作成されます。このエントリは、JSP ページやサーブレットなどの特定のリソースに対してディスパッチするようアプリケーションに命令します。リソースは、例外タイプまたは HTTP エラーコードのいずれかにマップされます。`Error type 404, object not found` のような例があります。
- **セキュリティおよび認証のプロパティ。**ログイン情報を設定し、Web モジュールに定義されているセキュリティの制約またはセキュリティロールの数を参照できるようにします。詳細については、JSP/サーブレットのオンラインヘルプの「Web モジュールのセキュリティプロパティの編集」を参照してください。

配備記述子には、その他にも多数の Web アプリケーションプロパティを指定できます。詳細は、Java Servlet Specification v2.3 を参照してください。

注 – 配備記述子の変更を有効にするには、Web モジュールを配備し直す必要があります。

プロパティシートを使用して web.xml ファイルを編集する

web.xml プロパティシートを使用して、web.xml ファイルを構成するさまざまな要素を編集します。このプロパティシートは、配備記述子エディタとも呼ばれます。これらのプロパティは、次のタブを持つ区画に表示されます。

- **配備。**コンテキストパラメータ、説明情報、表示名、エラーページ、フィルタ、JSP ファイル、および Web モジュールが配布可能かどうかを指定します。Web モジュールの大小のアイコン、リスナー、サーブレット、セッションのタイムアウト、バージョン情報、および開始ファイルも指定できます。
- **セキュリティ。**Web モジュールのログイン設定、セキュリティの制約、およびセキュリティロールのプロパティを指定します。
- **参照。**Web モジュールの EJB ローカル参照、EJB 参照、環境エントリ、リソース環境参照、およびリソース参照を指定します。これらの値は、J2EE アプリケーションサーバー上で Web モジュールをより大きい J2EE アプリケーションの一部として配備する場合に使用します。
- **サーバー固有のタブ。**サーバー固有のパラメータ、マッピング、および設定を指定します。

配備記述子の値は、それぞれの配備の状況によって変更することができます。アプリケーションをコンパイルし直したりコードを変更したりする必要はありません。この機能は、外部サーバーのデータを提供する場合に便利です。たとえば、SMTP (Simple Mail Transfer Protocol) メールサーバーやデータベースサーバーの名前、ユーザー名とパスワード情報、または HTML テンプレートなどの外部リソースへのパスのデータを提供する場合です。

サーブレットとフィルタの登録

新しいサーブレットまたはフィルタの作成時には、「新規」ウィザードによって、重要な情報についての確認が行われ、その情報が配備記述子に挿入されます。たとえば、配備記述子エントリを更新して次の特別な機能を指定したいことがあります。

- サーブレットまたはフィルタの追加マッピング
- サーブレットまたはフィルタの初期化パラメータ
- 外部の配備ツールで使用する大小のアイコン

サーブレットとフィルタの宣言については、54 ページの「配備記述子でのサーブレットの宣言」と 59 ページの「配備記述子でのフィルタの宣言」を参照してください。サーブレットとフィルタの配備記述子エントリの更新については、JSP/サーブレットのオンラインヘルプの「サーブレット」プロパティエディタと「フィルタ」プロパティエディタを参照してください。

タグライブラリの登録

配備記述子にタグライブラリを登録する必要があるかどうかは、タグライブラリをどのようにパッケージするかによって決まります。配備記述子に対して「タグライブラリ」プロパティエディタを使用して、Web モジュール内のタグライブラリについてすべてのエントリを参照できます。

自動登録

タグライブラリの JAR ファイルにデフォルトの TLD (つまり `META-INF/taglib.tld`) が含まれている場合には、JAR ファイルが `WEB-INF/lib` ディレクトリに追加されるときに、配備記述子ファイルにタグライブラリのエントリが作成されます。

タグライブラリの JAR ファイルに複数の TLD ファイルが含まれている場合には、デフォルトの TLD は必要ありません。このような例としては、標準のタグライブラリ (JSTL) があります。この場合には、配備記述子には明示的なタグライブラリのエントリは必要ありません。JSP 変換によって、暗黙のタグライブラリマッピングを作成するために、`WEB-INF/lib` ディレクトリの任意の JAR ファイルの TLD ファイルから URI フィールドが抽出されます。

明示的な登録

すべてのタグライブラリにデフォルトの TLD が指定されているわけではありません。「タグライブラリ」プロパティエディタを使用して、配備記述子内にタグライブラリエントリを明示的に作成することができます。

- タグライブラリには、URI とロケーション用のサブ要素があります。URI は、ライブラリを指す `web.xml` ファイルに対して相対的なものです。
- TLD ファイルのロケーションは、ライブラリに対して相対的になります。

たとえば、次の `taglib` 指令は、`/WEB-INF/tlds/myTagLib.tld` に格納されている TLD に 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>
```

詳細については、96 ページの「カスタムタグライブラリのパッケージ化と配備」を参照してください。

タグライブラリの宣言を削除しても、`WEB-INF/lib` ディレクトリからは対応する JAR ファイルは自動的に削除されません。

Taglib 要素でのデフォルトの URI の指定

「タグライブラリ」プロパティエディタは、主にどのタグライブラリが追加されたかを表すときに使用します。このプロパティエディタを使用して、タグライブラリの URI マッピングを変更することもできます。各エントリは、配備記述子の `taglib` 要素に対応します。指定されたタグライブラリを使用するには、JSP ファイルの最上位の URI マッピングを参照してください。URI 参照によって、カスタムタグライブラリのコード補完を使用できるようになります。詳細については、60 ページの「タグライブラリの使用」を参照してください。

タグライブラリの参照を追加、編集したり、または選択した Web モジュールからタグライブラリの参照を削除したりするには、「タグライブラリ」プロパティエディタを使用します。このプロパティエディタを使用して項目を追加または削除しても、`WEB-INF/lib` ディレクトリのタグライブラリ JAR ファイルには影響ありません。

選択した Web モジュールのコンテキストメニューから「JSP タグライブラリを追加」を選択すると、それぞれのタグライブラリのエントリが配備記述子に作成されます。

IDE を使用してソースエディタで web.xml ファイルを編集する

配備記述子の XML 構文を理解している場合は、`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。変更を保存すると、IDE がファイルを自動的に構文解析します。そしてエラーがあれば、「XML パーサー」区画の「出力」ウィンドウに表示します。また、外部テキストエディタを使用してそのファイルを編集することもできます。

統合サーバーの使用

IDE は、Sun ONE Application Server 7 と Tomcat という 2 つの統合サーバーとともにインストールされます。2 つのサーバーは、エクスプローラの「実行時」タブで、「サーバーレジストリ」の「インストールされているサーバー」ノードの下に表示されます。特に指定しない場合は、JSP ページやサーブレットなどの Web コンポーネントをコンパイルして実行するときに、IDE によって Sun ONE Application Server 7 が使用されます。

IDE を使って以下の処理を行うことができます。

- Web モジュールまたは Web モジュールのグループを配備および削除する
- サーバーのインストールに対して一般的なプロパティを設定する
- サーバーを開始、再開、停止する

■ サーバーの設定ファイルを編集する

統合サーバーの詳細は、オンラインヘルプの「Sun ONE Application Server インテグレーションモジュールのヘルプ」と「Tomcat プラグインヘルプ」を参照してください。

Web モジュールのテスト実行

Web モジュールを実行する前に、まず WEB-INF ノードを右クリックし、コンテキストメニューから「配備」を選択して、Web モジュールを配備する必要があります。IDE は、WEB-INF ノードのプロパティシートの「コンテキストルート」値で指定されているコンテキストに Web モジュールを配備します。コンテキストルートは、`/test` のように先頭文字をスラッシュ (/) とします。このコンテキストルートは、Web モジュールの**ドキュメントルート**の形成に使用されます。たとえば、Web モジュールのコンテキストルートが `/test` の場合は、ドキュメントルートは `http://`*host*`:8080/test` のようになります。

「配備」アクションを開始すると、IDE はファイルを保存し、サーブレットをコンパイルし直してから、Web モジュールを配備します。JSP ファイルは実行時に実際に読み込まれるまでコンパイルされないことに注意してください。

Web モジュールを配備したら、「実行」アクションを使用して IDE 内でモジュールをテスト実行できます。このアクションにより、指定のデフォルトブラウザを使用してクライアントの Web ブラウザからアプリケーションが実行されます。

Web モジュールの実行プロパティは、WEB-INF ノードのプロパティシートの「実行」タブで設定できます。

いくつかの Web モジュールをグループとして実行する場合は、最初に 「Web モジュールグループ」を作成します。詳細については、84 ページの「Web モジュールグループの作成と実行」を参照してください。Web モジュールグループは IDE 固有のオブジェクトです。Web モジュールグループを使用すると、いくつかの Web モジュールを指定して、1 回の処理で 1 つの Web サーバーに配備することができます。グループ内の各モジュールには、一意のコンテキストルートを定義する必要があります。コンテキストルートによって、要求が Web サーバーを介して正しいモジュールに送信されます。

Web モジュールや Web モジュールグループを実行するときは、HTTP モニターを使用できます。HTTP モニターを使用すると、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、86 ページの「HTTP モニターを使用した Web アプリケーションのデバッグ」を参照してください。

サーバーの構成

Web モジュールを配備または実行すると、IDE によってサーバーの構成ファイルが自動的に更新されます。サーバー構成の詳細は、「Sun ONE Application Server インテグレーションモジュールのヘルプ」の「管理サーバー」と「Tomcat プラグインヘルプ」の「Tomcat サーバー構成ファイルの編集」を参照してください。

Web サーバー環境をデータベースアクセス用に設定する

PointBase は Sun ONE Studio 5 ソフトウェアとともに Sun ONE Application Server 7 ソフトウェアを含むサブディレクトリにインストールされます。IDE と Sun ONE Application Server 7 を別々にインストールすると、サーバーに PointBase ソフトウェアが含まれない場合もあります。サーバーに PointBase ソフトウェアが含まれない場合は、PointBase ソフトウェアをダウンロードして手動でインストールする必要があります。PointBase サーバーの起動とデータベース接続の設定方法の詳細は、『Sun ONE Studio 5, Standard Edition インストールガイド』の「IDE でのデータベースの使用」を参照してください。この章には、Sun ONE Application Server 7 とともにほかの JDBC 対応データベースを使用する方法も記載されています。

Tomcat を使用する場合は、以下のいずれかを行って Tomcat サーブレットコンテナがデータベースドライバを使用できるようにする必要があります。

- Web モジュールの `WEB-INF/lib` ディレクトリにドライバをコピーします。
- ドライバを共有リソースとして内部 Tomcat サーバーに追加するために、*s1studio-install-directory*`/jwsdp/common/lib` ディレクトリにこれをコピーします。このディレクトリのファイルは、IDE のすべてのユーザーで共有されます。
- ドライバを共有リソースとして Tomcat の外部インストールに追加するために、*tomcat-install-directory*`/common/lib` ディレクトリにドライバをコピーします。

  詳細については、Tomcat 4.0 プラグインのオンラインヘルプの「Tomcat インストールプロパティの設定」セクションの「外部実行プロセス」プロパティの説明を参照してください。

注 – Sun ONE Application Server 7 を IDE とともにインストールした場合は、PointBase ドライバは *s1studio-install-directory*`/appserver7/pointbase/client_tools/lib` ディレクトリの `pbclient42RE.jar` ファイルにあります。

システムのクラスパス変数にデータベースドライバを追加しても、Web サーバーがそのデータベースドライバにアクセスできないこともあります。

IDE のデータベースエクスプローラでデータベースドライバを使用して、通信のテストを行うことができます。このテストを行うと、使用しているドライバが Java 言語と IDE の両方で正しく機能することが保証されます。

単一 Web モジュールを実行する

単一の Web モジュールを実行するには、モジュールを右クリックして、コンテキストメニューから「実行」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。この実行により、ブラウザに Web モジュールのルートを要求するよう指示します。また、サーバーはルートディレクトリの開始ファイルを提供しようとします。開始ファイルの設定は、`web.xml` ファイルのプロパティシートの「開始ファイル」プロパティを編集することで行うことができます。サーバーが開始ファイルを見つけられない場合は、ディレクトリの内容を一覧で表示します。

モジュールをまだ配備していない場合、またはモジュールを変更した場合は、まず WEB-INF ノードを右クリックし、コンテキストメニューから「配備」を選択して、モジュールを配備する必要があります。この配備アクションを行うと、IDE によって以下の処理が実行されます。

- 変更されたすべてのファイルが Web モジュールに保存されます。
- クラスファイルがコンパイルし直されます (JSP ページは、ファイルが取得され、Web サーバーで使用されるときにコンパイルし直されます)。
- Web サーバーが動作していない場合は、配備を行うためにサーバーが起動します。Sun ONE Application Server 7 を使用する場合は管理サーバーを実行する必要があることに注意してください。管理サーバーが動作していないと、IDE は Web サーバーを起動することができず、「接続が拒否されました」といったエラーメッセージが表示されます。配備するために IDE で Web サーバーを起動する必要がある場合は、配備処理の完了時に IDE によってサーバーが停止します。
- WEB-INF ノードのプロパティシートの「コンテキストルート」値で指定されているサーバーディレクトリにモジュールが配備されます。たとえば、「コンテキストルート」の値が `/catalog` の場合は、Web モジュールの URL は `http://`*host*`:`*port*`/catalog` になります。

注 – 内部 Tomcat サーバーに配備する場合は、この配備の操作は行う必要はありません。この場合は、実行することによって配備と実行の両方が行われます。

HTTP モニターを使用して、Web モジュールに対する要求、および Web アプリケーションにおけるデータフローを追跡することが可能です。詳細については、86 ページの「HTTP モニターを使用した Web アプリケーションのデバッグ」を参照してください。問題が発生した場合は、実行時に表示される「出力」ウィンドウを確認してください。

クラスパスの構成

サーバーで Web モジュールを実行する場合には、クラスパスが構成されます。クラスパスによって、Web モジュールで必要なすべてのクラスとライブラリを特定できるようになります。デフォルトでは、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、次のとおりです。

1. `WEB-INF/classes`
2. `WEB-INF/lib` にあるすべての JAR ファイル
3. その他のクラスまたはライブラリ (クラスローディングのサーバーの実装によって異なる)

Web モジュールグループの作成と実行

IDE には、複数の Web モジュールを一度に実行できるようにまとめて配備する仕組みがあります。このようにするには、IDE に Web モジュールグループのノードを作成し、そのグループノードに Web モジュールを追加します。次に、配備と実行を行います。

Web モジュールグループを作成して実行するには、主に次の 4 つの処理を行います。

1. マウントされたファイルシステムを右クリックしてコンテキストメニューから「新規」を選択し、「すべてのテンプレート」を選択して、Web モジュールグループオブジェクトを作成します。表示された新規ウィザードで、「JSP & サーブレット」を展開して、「Web モジュールグループ」を選択します。「次へ」をクリックしてウィザードの作業を完了します。マウントされたファイルシステムのルートディレクトリに Web モジュールグループのノードが表示されます。

   Web モジュールグループのノードは、Web モジュールの外部に設定します。このように設定すると、配備用にパッケージ化するときに誤ってこの Web グループがパッケージ化されることがなくなります。

2. Web モジュールグループノードを右クリックして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、Web モジュールグループの一部として読み込まれる各 Web モジュールの URL マッピングを設定します。

3. (省略可能) Web モジュールグループノードのプロパティシートを開き、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。ターゲットサーバーを指定しない場合は、デフォルトの Web サーバーが使用されます。

4. Web モジュールグループノードを右クリックして、「配備」を選択します。IDE によって配備アクションが完了したら、Web モジュールグループノードを右クリックして「実行」を選択します。

これらの処理の詳細については、JSP/サーブレットのオンラインヘルプの「Web モジュールグループの作成」を参照してください。

それぞれの Web モジュールについて 1 つのターゲットサーバーを設定できます。この処理は、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実行します。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ときに、その Web モジュールのコンポーネントが実行されることがあります。このような場合は、Web モジュールが、Web モジュールグループの「ターゲットサーバー」プロパティで指定されているサーバーで作動します。詳細については、JSP/サーブレットのオンラインヘルプの「WEB-INF プロパティの設定」を参照してください。

外部サーバーでの実行

ユーザーが、システム上に Sun ONE Application Server 7 または Tomcat 4.0 Web サーバーの独自のインストールを定義している場合があります。このユーザーの独自のインストールをデフォルト Web サーバーとして設定できます。または、インストールされているサーバーを指定の Web モジュールとして使用するように指定できます。詳細については、JSP/サーブレットのオンラインヘルプの「Web モジュールグループの作成」で、「ターゲットサーバー」プロパティの設定のセクションを参照してください。

外部 Sun ONE Application Server 7 の使用については、『Sun ONE Studio 5, Standard Edition インストールガイド』を参照してください。外部 Tomcat 4.0 Web サーバーを使用する場合は、内部サーバーではなく外部 Tomcat サーバーのインスタンスを使用するように、IDE の Tomcat プラグインモジュールを設定します。詳細については、Tomcat プラグインのオンラインヘルプの「Tomcat 4.0 インストールの追加」を参照してください。

Tomcat Web サーバーとの統合では、次の 2 つの処理モードがあります。

- フルモード。HTTP の監視機能、および JSP とサーブレットのデバッグ機能を使用できる。IDE によって、サーバーの `server.xml` 構成ファイルにいくつかの要素が追加される。
- 最小モード。HTTP の監視機能、および JSP とサーブレットのデバッグ機能は使用できない。配備されたモジュールのコンテキストパスに追加する場合のみ、サーバーの構成ファイルを変更する。

Web アプリケーションのデバッグ

IDE には、Web アプリケーションをデバッグするために次の 2 つのツールがあります。

- HTTP モニター
- ソースレベルのデバッグ機能 (JSP とサーブレットの両方で使用可)

HTTP モニターを使用した Web アプリケーションのデバッグ

HTTP モニターは、ソースレベルのデバッグ機能の代替または補完として使用できる軽量ツールです。HTTP モニターは、サーブレットコンテナでの JSP ファイルとサーブレットの実行に関連するデータを収集するために、IDE によって提供されます。HTTP モニターで生成したデータレコードによって、JSP ファイルとサーブレットのデバッグに伴う処理を効率よく行うことができます。HTTP モニターは、サーバーが受け取るそれぞれの要求に関する情報を記録します。このデータには、受信クッキーと送信クッキー、サーバーで保持されているセッション情報、サーブレット環境の情報、HTTP ヘッダーなどが含まれています。

HTTP モニターでは、Web アプリケーションのコンポーネント間のデータフローが表示されます。HTTP モニターを使って HTTP 要求のプロパティを調べることができます。また、このモニターを使って、JSP ページやサーブレットによって要求が処理されるたびに、サーバー上に保持されているデータを検査したりできます。たとえば、次の状況を検出できます。

- JSP ページまたはサーブレットが、要求でフォームデータを受け取った。この場合には、値が何であるかを特定できます。
- HTTP セッションが要求に関連付けられた。この場合には、JSP ページまたはサーブレットが有効化される前と後の属性がどのように設定されているかを調べることができます。
- JSP ページまたはサーブレットが、なんらかのクッキーを受け取った、または作成した。

テスト実行の際に問題が検出された場合は、HTTP モニターを使用して、問題の原因となる要求に関するデータレコードを表示します。JSP ページやサーブレットが、予想されるデータを受け取っていないことにより問題が発生している場合は、すぐにわかります。このような場合は、問題の原因となっているリソースを特定できます。編集と再実行の機能を使用すると、要求を送信し直す前に調整することが可能です。これによって、別のデータを送信すれば問題が解決できることが確認できます。ほとんどの場合は、この知識があれば、問題の原因となっているコード行を特定することができます。これで問題を解決できない場合は、個々の JSP ページまたはサーブレットを絞り込んで、ソースレベルのデバッガを使用してステップ実行を行います。詳細については、91 ページの「ソースレベルのデバッグ」を参照してください。

問題を解決した後にもう一度 HTTP モニターを使用して、問題の原因となっていた要求を再実行します。このようにすると、ステップを再生するのにブラウザからデータを入力しなければならない場合に、時間を節約することができます。

HTTP モニターでは、記録されている要求データを保存できます。問題が検出できてもすぐに修正する時間がない場合には、関連するレコードを保存し、上記のプロセスを後で実行することも可能です。また、アプリケーションの正常性テストの対象として一連の要求を保存しておくことができます。そして、何らかの変更があるたびにそれを実行することもできます。

要約すると、モニターには次の機能があります。

- データを表示し、将来のセッションのために情報を保存する。以前の要求を再実行したり編集したりする
- ソースレベルのデバッガを使用する前に、不正なデータを送信または受信したコンポーネントを特定する

Web アプリケーションのエラー原因の詳細については、7 ページの「Web アプリケーションの一般的なエラー」を参照してください。ソースレベルのデバッグ機能の詳細については、91 ページの「ソースレベルのデバッグ」を参照してください。

HTTP モニターの配備

Web アプリケーションで HTTP モニターを使用する前に、サーバーがモニターを使用できるように以下の作業を行う必要があります。

1. *IDE-install-directory*/modules/schema2beans.jar と *IDE-install-directory*/modules/ext/httpmonitor.jar を Web モジュールの WEB-INF/lib ディレクトリにコピーします。

1. プロパティシートの「配備」タブで、配備記述子ノード (WEB-INF/web.xml) のプロパティシートを表示します。次に、「フィルタ」の値フィールドをクリックして省略符号 (...) ボタンをクリックし、「フィルタ」ダイアログを表示して「追加」をクリックします。

2. 「追加 フィルタ」ダイアログで、「フィルタの論理名」を「HTTPMonitorFilter」または同様の内容がわかる名前に設定します。

3. 「フィルタクラス」テキストフィールドで、以下を入力します。

   ```
   org.netbeans.modules.web.monitor.server.MonitorFilter
   ```

4. 「フィルタマッピング」の表の下の「追加」をクリックして、「追加 フィルタマッピング」ダイアログを表示し、「了解」をクリックしてデフォルトの /* を受け入れます。

5. 「init パラメータ」の表の下の「追加」をクリックして、「追加 初期パラメータ」ダイアログを表示します。

6. 「初期パラメータ名」テキストフィールドに、netbeans.monitor.ide と入力します。

7. 「初期パラメータ値」テキストフィールドに以下を入力します (変数は実際の値で置き換えます)。

   name-of-host-that-runs-IDE:port-of-internal-HTTP-server

   ポート番号が不明な場合は、エクスプローラウィンドウの「実行時」タブで HTTP サーバーノードのプロパティシートを開きます。「ポート」プロパティにポート番号が表示されます。

8. 「了解」をクリックして「追加 初期パラメータ」ダイアログを閉じ、「了解」をクリックして「追加 フィルタ」ダイアログを閉じます。

本稼動用のサーバーに配備する前に、必ず JAR ファイル、フィルタ、およびフィルタマッピングを削除してください。

注 – IDE の Tomcat サーバープラグインで管理される Tomcat サーバーを使用する場合に、統合モードが「フル」に設定されているときは、これらの作業を行う必要はありません。デフォルトでは、内部 Tomcat サーバーはフル統合モードに設定されています。

HTTP モニターの起動

HTTP モニターは、IDE の最上位レベルの 1 つのウィンドウ、および実行サーバー上で作動する 1 つのサーバー側コンポーネントで構成されます。HTTP モニターを開くには、IDE の「表示」メニューまたは「デバッグ」メニューから「「HTTP モニター」を選択します。詳細については、HTTP モニターモジュールのオンラインヘルプの「Web サーバーのデータフローの監視」を参照してください。

モニターの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。左側の HTTP モニターレコードパネルには、既知のレコードがツリー状に表示されます。右側の HTTP モニターレコードディスプレイパネルには、選択したモニターレコードに関連するデータが表示されます。

ツリー表示では、「すべてのレコード」カテゴリに、「現在のレコード」と「保存されたレコード」という 2 つのサブカテゴリが含まれています。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定義されています。「現在のレコード」のエントリは、現在の IDE セッションでのみ使用できます。現在のモニターデータレコードは、Web サーバーを再起動しても有効です。これらのレコードは、IDE を再起動したとき、またはレコードを削除したときのみクリアされます。「保存されたレコード」のエントリは、ユーザーが削除するまで有効です。

あるリソースが他のリソースに対して要求を割り当てた結果として、データレコードが作成される場合があります。たとえば、サーブレットに対する JSP の転送などは、これに該当します。このような場合には、レコードは、展開可能なノードとしてツリーに表示されます。`forward` アクションまたは `include` アクションによって有効化された他のリソースで、要求がどのように処理されたかを表示するには、このノードを展開します。

すべてのカテゴリの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、HTTP モニターのオンラインヘルプの「HTTP モニター」ツールバーの使用を参照してください。

モニターデータレコードを選択すると、HTTP モニターレコードディスプレイパネルに、対応する情報がそれぞれ表示されます。レコードディスプレイパネルには、以下の区画があります。

- **要求。**要求 のURI、メソッド、照会文字列、パラメータ、送信データ、プロトコル、クライアント IP アドレス、スキーマ、終了ステータス、要求の前後の要求属性の一覧を表示します。
- **Cookie。**受信クッキーと送信クッキーの一覧を表示します。受信クッキーには、クッキー名とクッキー値の情報が含まれています。送信クッキーには、名前、値、有効期間、およびクッキーが送信されるドメインの情報が含まれています。送信「クッキー」には、クッキーが送信されるパス、およびセキュアプロトコルが必要かどうかも含まれています。
- **セッション。**要求の処理の前後に、その要求に関連付けられている HTTP セッションのステータスを表示します。この区画では、要求が処理された結果としてセッションが作成されたかどうかを表します。ID、作成時刻、および最後にアクセスした時刻、非アクティブな状態で保持される最大期間など、セッションのプロパティが表示されます。要求の処理の前後に設定されているセッション属性も表示されます。
- **サーブレット。**設定されたサーブレットの名前、クラス名、パッケージ名、オプションのサーブレット情報を表示します。相対パスと変換されたパスも表示されます。
- **コンテキスト。**Web モジュールから作成されたサーブレットコンテキストの名前、およびそのコンテキストへの絶対パスを表示します。この区画には、要求が処理を開始したときに設定されたコンテキスト属性がすべて表示されます。また、要求に指定された初期化パラメータもすべて表示されます。
- **クライアントおよびサーバー。**HTTP 要求を生成したアプリケーションに関するデータを表示します。このデータには、アプリケーションが使用したプロトコル、IP アドレス、および (情報が提供された場合に) 使用されたアプリケーションが含まれます。この区画には、アプリケーションがサポートするロケール、エンコーディング、文字セット、ファイルフォーマットが表示されます。また、要求を処理したサーブレットエンジンについてのデータも表示されます。このデータには、Java バージョン、プラットフォーム、ホスト名、HTTP サービスのポート番号が含まれます。

- **ヘッダー。**要求に付随されている HTTP ヘッダーを表示します。ヘッダーは、ブラウザなどの HTTP クライアントによって構成されます。ヘッダーの実際の内容は、クライアントによって異なります。通常は、ソフトウェアやオペレーティングシステムなど、クライアントの性質を表す情報が含まれます。また、ブラウザがサポートしている言語設定とファイルフォーマットの情報も含まれます。

要求の再実行

「すべてのレコード」ツリービューの「現在のレコード」と「保存されたレコード」のサブカテゴリに関連付けられている HTTP 要求は、再実行することができます。要求を再実行すると、サーバーは、記録されている要求と同じものを処理します。この応答は、Web モジュールを実行した場合と同じように、IDE のデフォルトの Web ブラウザに表示されます。

この機能によって、エラーの修正をテストするプロセスが簡単になります。たとえば、エラーが検出され、フォームデータを処理するリソースでコードを修正したとします。通常、修正を確認するには、フォームをブラウザに読み込み、データを入力してサブミットします。元の要求レコードで「再実行」を使用すると、データを入力し直する手間がなくなります。また、エラーを再生する前にいくつかの要求をステップ実行する必要がある場合について考えてみます。精算を実装する部分のリソースにエラーがあり、ユーザーは買い物かごに何か入れるまで精算できないとします。このような場合には、記録されているいくつかの要求を実行し直すことができます。再実行によって、Web アプリケーションは、エラーの修正を確認できる状態になります。

詳細については、HTTP モニターのオンラインヘルプの「HTTP モニターで要求を編集および再実行する」を参照してください。

指定した要求の編集と再実行

「編集と再実行」ダイアログでは、要求の情報を送信し直す前に、いくつかのプロパティを編集することができます。これらのプロパティには、要求のパラメータや照会文字列、クッキー、ヘッダーなどが含まれています。HTTP モニターを完全にサポートしている場合には、要求を別のサーバーへ送信するためのオプションも含まれています。

「編集と再実行」ダイアログを使用して、要求を別のサーバーへ送信できます。

「編集と再実行」の機能を使用して、次の処理を行うことができます。

- 別の情報を入力した場合に正常に機能していないかどうか確認する
- ブラウザの設定を変更しないで、クライアントのロケール設定などの項目を変更する

要求を再実行するためのセッションクッキーの指定

要求を再実行すると、デフォルトでは、HTTP モニターはクッキーを送信する場合に、ブラウザが送信しているセッションクッキーを使用します。HTTP モニターは、要求に付随して記録されているセッションクッキーは使用しません。記録されているクッキー、または指定したクッキーを使用できるように、この動作を制御することができます。詳細については、HTTP モニターのオンラインヘルプの「HTTP モニターで要求を編集および再実行する」を参照してください。

ソースレベルのデバッグ

Web アプリケーションの一部として JSP ページが実行される場合には、実際にはサーブレットが実行されます。JSP ページは「生成されたサーブレット」というサーブレットに変換され、この生成されたサーブレットがコンパイルされて実行されます。Sun ONE Application Server 7 または Tomcat サーバーを使用している場合は、元の JSP ソースの行と生成されたサーブレットの行の間のマッピングが IDE で保持されます。この機能によって、次のいずれかから JSP ページをデバッグできます。

- JSP ファイル自身
- JSP ファイルが変換されたサーブレットのソース

ソースレベルのデバッガは、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は、他方に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。

静的にインクルードされているすべてのファイルを含む JSP ファイルは、単一のサーブレットファイルにマップされます (詳細については、51 ページの「<%@include%> 指令の使用」を参照してください)。具体的には、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。サーブレットには、すべての JSP ページに対して生成されており、JSP ソースには明示的に示されないコードも含まれます。

デバッガの使用

IDE のソースレベルのデバッグ機能を使用してアプリケーションをデバッグするには、以下の節で説明するようにブレークポイントを設定し、生成したサーブレットを表示してデバッガを開始します。

ブレークポイントの設定

標準のサーブレットまたはクラスファイルに対するブレークポイントの設定は、他の Java クラスをデバッグする場合と同じです。JSP ページの場合は、JSP ソースファイル、または生成されたサーブレットのいずれかにブレークポイントを設定できます。

JSP ソースファイルにブレークポイントを設定すると、生成されたサーブレットにブレークポイントが自動的に反映されます (これらの 2 つのファイルの間には 1 対 1 の対応はありません)。生成されたサーブレットにブレークポイントを設定すると、1 つの例外を除いては、JSP ページにブレークポイントが反映されます。具体的には、ブレークポイントが、静的にインクルードされている JSP ページから派生したサーブレットに含まれている場合は、ブレークポイントは反映されません。JSP ファイルからブレークポイントを削除すると、生成されたサーブレットの対応するブレークポイントも削除されます。ただし、この逆の場合は削除は行われません。

生成されたサーブレットの表示

エクスプローラで「ファイルシステム」タブをクリックします。JSP ソースファイルを選択します。右クリックしてコンテキストメニューを表示します。JSP ファイルがまだコンパイルされていない (つまり「サーブレットを表示」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。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。JSP ファイルがすでにエディタに表示されている場合は、エディタのコンテキストメニューから、生成されたサーブレットも表示することができます。エディタウィンドウを右クリックし、「サーブレットのコードを表示」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、生成されたサーブレットファイルが表示されます。

JSP ソースを変更しても、生成されたサーブレットは自動的には更新されません。JSP ソースを変更した後は、「サーブレットを表示」または「サーブレットのコードを表示」をもう一度実行して、新しいバージョンを表示します。

注 – JSP ページのコンパイル中にコンパイルエラーが検出され、JSP ソースコードを参照しても問題が明白でない場合があります。このような場合は、生成されたサーブレットを参照すると有効です。これによって、引用符が欠落しているなどの誤りを探すことができます。

デバッガの開始

デバッガを開始するには、デバッグを行うリソースをエクスプローラで選択します。次に、IDE の「デバッグ」メニューから「新規セッション開始」を選択します。別のリソースをデバッグする場合に、デバッガをいったん停止して再起動する必要はありません。Web モジュールが作動している状態で、ブラウザウィンドウのリンクに従うか、またはブラウザのロケーションフィールドに直接 URL を入力します。

そのままでは実行できないヘルパークラスをデバッグする場合は、別のリソース (Web モジュール上の WEB-INF ノードなど) でデバッガを起動する必要があります。

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のいずれかの Web モジュールから WEB-INF ノードを選択します。次に、「デバッグ」メニューから「新規セッション開始」を選択します。

IDE における標準デバッグの詳細については、Core IDE オンラインヘルプの「プログラムのデバッグ」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。デバッグのサポートとして HTTP モニターを使用する方法については、86 ページの「HTTP モニターを使用した Web アプリケーションのデバッグ」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。具体的には次のようにします。

- JSP ソースファイルと生成されたサーブレットファイルを開いた状態で、エラーを表示するファイルのクラスを指定します。「オプション」ウィンドウの「構築」ノードを展開します。「JSP & サーブレット設定」ノードを選択します。次に「プロパティ」区画で、「JSP コンパイルラエラー」オプションに対するコンボボックスから「サーブレットソース内に表示」または「JSP ソース内に表示」を選択します。
- デバッグのときに JSP ソースの JSP タグの間にある静的な HTML 行、および生成されたサーブレットの対応する行をスキップするかどうかを指定します。静的な HTML 行とは、JSP 要素やスクリプト言語が含まれていない行を意味します。IDE のメニューバーの「ツール」メニューから「オプション」を選択します。次に、「デバッグと実行」ノードを展開します。「JSP & サーブレット設定」を選択します。「プロパティ」区画で、「静的な行をスキップ」を `True` に設定します。

詳細については、JSP/サーブレットのオンラインヘルプの「JSP デバッガのオプションの設定」を参照してください。

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

デバッグの際に、JSP ソースファイルと生成されたサーブレットファイルを並べて表示することができます。この機能は、デバッグサイクルで JSP エラーを検出する場合に便利です。両方のファイルで同期してブレークポイントを設定し、ステップ実行を行うことが可能です。

ソースエディタで、生成されたサーブレットを開きます (詳細については、92 ページの「生成されたサーブレットの表示」を参照してください)。エクスプローラで JSP ファイルを選択した状態で、コンテキストメニューの「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 コードのいずれ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。エディタの表示を右クリックし、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。このようにすると、新しいソースエディタが開き、同じ JSP コードが示されます。

これで、2 つのエディタウィンドウが並んで表示されます。一方のウィンドウには JSP コードが表示されます。他方のウィンドウにはサーブレットコードを表示することができます。生成されたサーブレットの変更を参照するには、JSP ファイルをコンパイルします。詳細については、JSP/サーブレットのオンラインヘルプの「デバッグ中に JSP ファイルとサーブレットファイルを表示する」を参照してください。

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。他方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ます。そしてコマンドを実行します。

Web アプリケーションのパッケージ化

通常、開発段階では Web モジュールは展開されている形式で実行されます。これによって頻繁な更新を容易に行うことができます。ただし、本稼動用の環境に配備する準備ができている場合には、Web モジュールを WAR (Web ARchive) ファイルにパッケージ化することができます。このようにすると、再送のプロセスが簡単になります。IDE から WAR ファイルを作成すると、デフォルトでは、Web モジュールのファイルシステム内の拡張子が `.java` 以外のファイルがすべて含まれます。IDE を使用して Web モジュールディレクトリを WAR ファイルにパッケージする方法については、JSP/サーブレットのオンラインヘルプの「Web モジュールのパッケージ化」を参照してください。

IDE では、次の処理を行うことができます。

- Web モジュールから WAR ファイルを作成する
- WAR ファイルのパッケージ化オプションを指定する
- Web モジュールから生成された WAR ファイルの内容を表示する

- WAR ファイルを Web モジュールとしてマウントする

Web モジュールから WAR ファイルを作成する

Web モジュールから WAR ファイルを作成するには、次のいずれかの方法を使用します。

- ファイルシステムエクスプローラで Web モジュールのルートを選択し、「ツール」メニューの「WAR ファイルをエクスポート」を選択する
- Web モジュールの WEB-INF ノードを右クリックし、コンテキストメニューの「WAR ファイルをエクスポート」を選択する

この後で、WAR ファイルの名前を指定します。デフォルトでは、Web モジュールのファイルシステムの下位のファイルがすべて WAR ファイルに含まれます。

詳細については、JSP/サーブレットのオンラインヘルプの「Web モジュールのパッケージ化」を参照してください。

オプションの指定

WEB-INF ノードのプロパティシートの「アーカイブ」タブには、Web モジュールに関するパッケージ化のオプションが表示されます。ここで、Web モジュールの外部のファイルを WAR パッケージに追加することができます。ファイルは、エクスプローラにマウントされている任意のファイルシステムから追加できます。追加できるファイルとして、「すべてのファイル」、「`.java` を除くすべてのファイル (デフォルトのオプション)」、「`.java`、`.jar`、および `.form` を除くすべてのファイル」、のいずれかを選択できます。また、`.*txt` や `Pro.*` などの正規表現も使用できます。定義済のフィルタ、または正規表現のいずれかを選択できますが、フィルタと正規表現の両方は選択できません。フィルタに使用される正規表現は、UNIX の正規表現と同じではないことに注意してください。詳細については、JSP/サーブレットのオンラインヘルプの「Web モジュールのアーカイブプロパティの編集」を参照してください。

内容の表示

「WAR の内容」ウィンドウを使用して、WAR パッケージの内容を表示できます。WAR パッケージに含まれるファイルは、WEB-INF ノードのプロパティシートの「アーカイブ」タブで設定されたプロパティによって決定されます。

「WAR の内容」ウィンドウには、Web モジュールを WAR ファイルとしてエクスポートする際に含まれるファイルが表示されます。また、各ファイルのパスと拡張子の情報も表示されます。

デフォルトでは、この一覧はパス名のアルファベット順にソートされています。ほかの列のヘッダーをクリックすると、一覧が、その列のアルファベット順にソートされます。詳細については、JSP/サーブレットのオンラインヘルプの「WAR ファイルの内容の表示」を参照してください。

カスタムタグライブラリのパッケージ化と配備

JSP ページでカスタムタグライブラリを参照するためには、JSP ページが含まれている Web モジュールの中にタグライブラリを定義する必要があります。本稼動用の Web モジュール内でタグライブラリを使用する場合には、それを JAR ファイルとしてパッケージしておくことをお勧めします。このようにすると、他の Web モジュールでタグライブラリを簡単に使用することができます。タグライブラリは、次の方法で Web モジュールに追加できます。

- 「JSP タグライブラリを追加」メニューを使用して、タグライブラリリポジトリから JAR ファイルを追加する
- 「JSP タグライブラリを追加」メニューを使用して、ファイルシステムから JAR ファイルを追加する
- 別の Web モジュールやファイルシステムから JAR ファイルをコピーしてペーストする

これらのいずれかの方法で Web モジュールにタグライブラリ JAR ファイルを追加すると、IDE によってエクスプローラの「ファイルシステム」区画にもタグライブラリがマウントされます。

タグライブラリを JAR ファイルとしてパッケージ化する

配備が簡単になるように、タグライブラリを JAR ファイルにパッケージします。タグライブラリ JAR を作成するには、タグライブラリのコンテキストメニューで「タグライブラリ JAR の作成」を選択します。タグライブラリ JAR を作成すると、IDE によって `.jar` 拡張子のファイルが作成されます。また、関連する `jarContent` ファイルも作成されます。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します。`jarContent` ファイルは「レシピ」ファイルとも呼ばれます。

新しく作成された JAR ファイルのプロパティについては、Core IDE のオンラインヘルプの「エクスプローラ内の JAR レシピノード」を参照してください。

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

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できます。

タグライブラリをタグライブラリリポジトリから Web モジュールに追加するには、WEB-INF ノードの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を検索」を選択すると、「JSP タグライブラリ - リポジトリブラウザ」ダイアログが表示されます。一覧から 1 つまたは複数の項目を選択して「了解」ボタンをクリックします。選択したタグライブラリに対する適切な JAR ファイルが、配備記述子内の必要なエントリとともに `WEB-INF/lib` ディレクトリに追加されます。

タグライブラリ JAR ファイルは Web モジュールに追加されると、エクスプローラの「ファイルシステム」タブにもマウントされます。

ファイルシステムから JAR ファイルを追加してタグライブラリを配備する

タグライブラリ JAR が、IDE の他の Web モジュールまたはマウントされているファイルシステム内にある場合は、「JSP タグライブラリを追加」>「ファイルシステム内を検索」を使用します。これによって JAR ファイルの場所が特定され、Web モジュールの `WEB-INF/lib` ディレクトリへコピーされます。マウントされている任意のファイルシステムから、タグライブラリ JAR ファイルを Web モジュールへ追加することができます。WEB-INF ノードを右クリックして、「JSP タグライブラリを追加」、「ファイルシステム内を検索」の順に選択します。.必要な場合は、タグライブラリのエントリが含まれるように配備記述子が自動的に更新されます。このエントリを使用して、タグへのアクセスで使用する JSP ページの URI に対してタグライブラリをマップします。

他のモジュールまたはファイルシステムから JAR ファイルをコピーアンドペーストしてタグライブラリを配備する

タグライブラリ JAR が、IDE の他の Web モジュールまたはマウントされているファイルシステムに内にある場合は、エクスプローラの JAR ファイルのノードでコンテキストメニューの「コピー」を選択します。次に Web モジュールの `WEB-INF/lib` ディレクトリに JAR ファイルをペーストします。タグライブラリ JAR ファイルを Web モジュールの `WEB-INF/lib` ディレクトリにコピーすると、配備記述子に `Taglib` 要素が作成されることがあります。ほとんどの場合には、`Taglib` 要素は IDE によって自動的に追加されます。正しい情報が定義されていることを確認することができます。確認するには、JSP/サーブレットのオンラインヘルプの「カスタムタグライブラリのパッケージと配備」の指示に従ってください。

J2EE アプリケーションに Web モジュールをインクルードする

IDE を使用すると、既存の EJB モジュールまたは Web モジュールから J2EE アプリケーションを作成できます。または、エクスプローラファイルシステムまたはパッケージノードからアプリケーションを作成できます。いずれの場合にも、IDE に新しいアプリケーションを作成するように指示すると、新しい J2EE アプリケーションのノードがエクスプローラに表示されます。ここから、アプリケーションにモジュールを追加することができます。

注 – エンタープライズ (J2EE) アプリケーションを作成するには、EJB モジュールとともに Web モジュールをアセンブルする必要があります。

J2EE アプリケーションは、EJB モジュールまたは Webモジュール、あるいはその両方で構成されています。既存の EJB モジュール、またはエクスプローラのファイルシステムから、IDE に J2EE アプリケーションを作成します。J2EE アプリケーションを作成すると、それに対して Web モジュールを追加できます。

Web モジュールを J2EE アプリケーションに追加する詳しい方法については、J2EE アプリケーションアセンブラの オンラインヘルプの「J2EE アプリケーションへの EJB モジュールまたは Web モジュールの追加」を参照してください。

Web モジュールを Web アプリケーションにアセンブルする方法については、『J2EE アプリケーションのプログラミング』を参照してください。

用語集

Bean JavaBeans 仕様に従って記述された、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) オブジェクト指向の分散型エンタープライズアプリケーションを開発および配備するためのコンポーネントアーキテクチャ。エンタープライズ JavaBeans を使用して記述されたアプリケーションは、スケーラブルで、トランザクション機能、マルチユーザー機能、セキュリティ保護機能を備えています。「JavaBeans」および「Bean」も参照してください。

HTTP (Hypertext Transfer Protocol) WWW 上でのファイル交換を制御するアプリケーションプロトコル。ファイルには、テキスト、イメージ、音声、ビデオなどがあります。

HTTPS (Hypertext Transfer Protocol Secure Socket) インターネットおよびイントラネットの環境で広く使用されている、セキュリティ保護された HTTP プロトコル。クライアントとサーバーの間でセキュリティ保護された情報を交換するときに使用します。このプロトコルを使用すると、セキュリティ保護された接続を使用してアプレットや bean を Web ブラウザにダウンロードすることができます。また、それらのアプレットまたは bean で、セキュリティ保護されたサーバー接続を作成することもできます。

HTTP 応答 最終的にクライアントブラウザに届くクッキー、ヘッダー、および出力を含むサーブレットコンテナ内で生成されるメッセージ。`HTTP` メソッドで指定される応答です。このマニュアルでは、通常、これを「応答」と呼びます。この応答は、サーブレットコンテナ内の `HTTPResponse` オブジェクトにカプセル化されます。

HTTP セッション クライアントブラウザと Web サーバーの間で複数の要求にまたがってやりとりされる会話を表すサーブレット API 構造。

HTTP モニター サーブレットエンジン内の JSP ファイルや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ための IDE モジュール。JSP ファイルまたはサーブレットに関連付けられた要求ごとにデータを記録します。記録されるデータには、受信した要求、受信したクッキーと送信したクッキー、およびサーバーによって維持されるセッション情報などがあります。

HTTP 要求 クライアントブラウザの属性およびクッキーを含むクライアントブラウザで作成されるメッセージ。GET メソッドまたは POST メソッドで指定される要求です。このマニュアルでは、通常、これを「要求」と呼びます。この要求は、サーブレットコンテナ内の HTTPRequest オブジェクトにカプセル化されます。

J2EE (Java 2 Platform, Enterprise Edition) 多数の技術を 1 つのアーキテクチャに集結した、Java 2 プラットフォームのエディション。含まれる技術には、Enterprise Bean、JSP ページ、XML などがあります。J2EE は、エンタープライズクラスのサーバー側アプリケーションを開発するための総合的なアプリケーションプログラミングモデルおよび互換性テストスイートを提供します。「EJB」、「JSP テクノロジ」、および「サーブレット」も参照してください。

J2EE Web 階層 J2EE アーキテクチャを構成する 3 つの階層の 1 つ。Web 階層ではプレゼンテーションロジックを作成します。この階層は、HTML クライアントや Web クライアントなどのプレゼンテーションクライアントから応答を受け取ります。また、それに対する適切な要求を行います。この階層の他にクライアント層とビジネス層があります。

J2EE アプリケーション J2EE プラットフォームで実行される J2EE コンポーネントから構成されたアプリケーション。J2EE コンポーネントには、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、Enterprise Bean などがあります。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。配備する際、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」および「J2EE Web 階層」も参照してください。

JAR (Java Archive ファイル) クラス、イメージ、およびその他のファイルを 1 つの圧縮ファイルに含める、プラットフォームに依存しないファイル形式。このファイル形式を使用すると、ダウンロード時間が短縮します。

JavaBeans 移植性がありプラットフォームに依存しない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。「EJB」も参照してください。

JDBC (Java Database Connectivity) Java プラットフォームとさまざまなデータベースの間でデータベースに依存しない接続性を実現するための業界標準規格。JDBC インタフェースは、SQL ベースでデータベースにアクセスするための呼び出しレベルの API を提供します。

JSF J2EE Web 階層アプリケーションの構築を簡略化するアーキテクチャおよび API を定義しようという提案。「J2EE Web 階層」も参照してください。

JSP アクション 暗黙オブジェクトやその他のサーバー側オブジェクト上で動作可能な JSP オブジェクト。新しい記述変数をこれによって定義することもできます。開始タグ、本体、終了タグを持つ要素の XML 構文に従います。本体が空の場合は、空タグの構文も使用できます。タグには接頭辞を付ける必要があります。アクションは、タグによって実装される抽象用語です。

JSP 仕様には「標準」アクションが定義されています。標準アクションは、インポートしなくても JSP ファイルでいつでも使用できます。

「カスタム」アクションは、移植可能なように TLD によって記述されます。カスタムアクションは Java クラスのコレクションであり、`taglib` 指令によって JSP にインポートされます。

JSP 式 有効なスクリプト言語の式を含むスクリプト要素。評価され、`String` に変換されて、暗黙の `out` オブジェクト内に配置されます。

JSP スクリプト要素 JSP 仕様によってタグの構文が定義されている、JSP の宣言、スクリプレット、または式。内容は、JSP ページで使用されているスクリプトの言語に従って記述されます。

JSP タグ JSP 仕様によって定義されているタグ。ドキュメント内のテキスト要素です。これを含むドキュメントは、外部ライブラリに含まれるフォーマット情報または処理ロジックを表します。XML フォーマットで記述されるので、JSP タグはデータではなくマークアップとして識別されます。タグを使用すると、JSP ページに Java コードを含める必要がありません。「JSP タグライブラリ」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ハンドラ (Java クラス) のコレクション。JSP ページ内のタグを使っ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。したがって、すべての JSP エンジンがこれを変換できます。「JSP タグ」および「カスタムタグ」も参照してください。

JSP テクノロジ (JavaServer Pages™) テンプレートデータ、カスタム要素、スクリプトの言語、およびサーバー側の Java オブジェクトを使用してクライアントに動的にコンテンツを返す、拡張可能な Web テクノロジ。通常、返されるコンテンツは HTML 要素または XML 要素で構成されています。多くの場合、クライアントは Web ブラウザです。JSP テクノロジはサーブレットテクノロジの拡張テクノロジです。「JSP ページ」および「サーブレット」も参照してください。

JSP ファイル JSP ページコードの物理的な表現。「JSP ページ」を参照してください。

JSP ページ 実行前にサーブレットコンテナによって動的にサーブレットに変換される、テキストベースの Web コンポーネント。「JSP ファイル」および「サーブレット」も参照してください。

JSP 要素 JSP ページ内の、JSP トランスレータによって認識される部分。JSP アクション、指令、JSP スクリプト要素のいずれかです。

JSTL (JavaServer Pages™ Standard Tag Library) 多数の JSP ページで一般的に使用されるコア機能を単純なタグとしてカプセル化する標準タグライブラリ。JSTL には、構造に関する一般的な作業をサポートする機能が含まれています。含まれている機能には、繰り返しおよび条件付き、XML ドキュメントを操作するためのタグ、国際化タグ、SQL タグなどがあります。

MIME (Multipurpose Internet Mail Extensions) ASCII 以外の電子メール添付ファイル (ビデオ、オーディオ、グラフィックスなど) を送受信するためのインターネット標準規格。Web ブラウザでは、HTML 形式でないファイルをどのように表示したり解釈したりするかを判断する手段としても MIME タイプを使用します。

Model オブジェクト Web アプリケーション内のアプリケーションデータをカプセル化する Java オブジェクト。

Struts Jakarta プロジェクトによるオープンソースフレームワーク。Java サーブレット API および JSP テクノロジを使って Web アプリケーションを構築するためのフレームワークです。Struts パッケージは、再利用可能なコンポーネントの統合セットを提供します。これには、コントローラサーブレット、JSP カスタムタグライブラリ、ユーティリティクラスなどが含まれます。ユーザーインタフェースを構築するためのこれらのコンポーネントは、任意の Web ベース接続に適用できます。「フレームワーク」を参照してください。

Sun ONE Application フレームワーク エンタープライズ Web アプリケーションの開発のために強化された、Sun Microsystems の J2EE Web アプリケーションフレームワーク。このフレームワークには、表示フィールド、アプリケーションイベント、コンポーネント階層、ページ主体型の開発アプローチなどの概念が組み込まれています。「JATO」とも呼ばれます。

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。JSP コンテナでは、TLD ファイルを使用して、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します。TLD ファイルには、ライブラリ全体に関するドキュメンテーションが含まれています。個々のタグに関するドキュメンテーションや、JSP コンテナおよびタグライブラリのバージョン情報も含まれています。また、TLD には、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記述されています。IDE では、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。

URI Uniform Resource Identifier) サーブレットを実行 (またはデバッグ) して、ブラウザに表示する URL を作成するときに使用されるプロパティ。通常、Web アプリケーションへの URI は次の構文に従っています。

```
http://server:port/context path/
local resource identifier ? query string
```

Value オブジェクト ヘルパーなどによって使用されるモデルの中間表現。「Model オブジェクト」および「ヘルパー」も参照してください。

WAR (Web Application Archive) Java クラスライブラリに使用されるパッケージに似た JAR ファイル形式。WAR ファイル形式はサーブレットコンテナにインストールまたは配備されます。通常、WAR には、Web コンポーネントと、Web リソースと呼ばれるその他のファイルが含まれます。サーバー側のユーティリ

ティクラス (データベース bean やショッピングカートなど) も含まれています。Web リソースには、静的な Web コンテンツ (HTML ファイル、イメージファイル、音声ファイル) とクライアント側のクラス (アプレットおよびユーティリティクラス) があります。Web アプリケーションは、WAR ファイルから、または WAR と同じフォーマットに編成されたパッケージ化されていないディレクトリから実行できます。「JAR」も参照してください。

Web アプリケーション このマニュアルでは、「Web モジュール」と交換可能な用語として使用される場合があります。その他の場合は、サーバーセット上にあるすべてのものを意味します。Web アプリケーションは、一般的には、ユーザーが動的な Web ページ上で Web ブラウザを使って特定の一連の作業を実行するために必要なすべての機能を備えたプログラムを意味します。Web アプリケーションの例としては、電子ショッピングモールやオークションサイトなどがあります。Web アプリケーションはクライアントサーバーモデルに基づいています。このモデルでは、クライアントは Web ブラウザであり、サーバーはリモート実行される機能のセットです。Web アプリケーションのコンポーネントセットには、サーブレット、JSP ページ、ユーティリティクラスを含めることができます。また、静的ドキュメント、クライアント側のアプレット、Java クラス、およびそれらの要素すべてを結びつけるメタ情報も含まれています。「Web モジュール」および「Web サーバー」も参照してください。

Web コンテキスト Web アプリケーションのコンポーネントへの識別パス。このパスを使用すると、単一のサーバー上の複数の Web モジュールにアクセスすることができます。「サーブレットコンテキスト」も参照してください。

Web コンテナ 「サーブレットコンテナ」を参照してください。

Web コンポーネント サーブレットおよび JSP の仕様によって定義されているコンポーネント。Web コンポーネントを実行するには、Web モジュールディレクトリまたはアーカイブの一部を Web サーバー上に配備する必要があります。「JSP ページ」および「サーブレット」も参照してください。

Web クライアント 一般的な例は Web ブラウザ (Netscape など) 。ただし、HTTP 要求の送信や HTTP 応答の解釈を専門に行うアプリケーションの場合もあります。「Web ブラウザ」も参照してください。

Web サーバー インターネット、イントラネット、またはエクストラネットの全域にサービスを提供するソフトウェア。Web サーバーは Web サイトをホストし、HTTP およびその他のプロトコルのサポートを提供します。Web サイトでは、CGI スクリプトやサーブレットなどの、特定の機能を実行するサーバー側プログラムが実行されます。J2EE アーキテクチャでは、Web サーバーは Web コンテナにサービスを提供します。たとえば、Web コンテナは、通常、HTTP メッセージの処理を Web サーバーに依存しています。J2EE アーキテクチャでは、Web コンテナが同じベンダーの Web サーバーによってホストされているものと想定しています。このため、この 2 つのエンティティ間の規約を指定していません。Web サーバーはそれぞれ 1 つまたは複数の Web コンテナをホストすることができます。

Web ブラウザ HTML ドキュメントやアプレットを表示するためのアプリケーション。それらの要素の間を移動したり、それらの要素とやりとりすることもできます。Web ブラウザは「ブラウザ」とも呼ばれます。また、ブラウザは「クライアント」と呼ばれることがあります。「クライアント」および「Web サーバー」も参照してください。

Web モジュール J2EE アプリケーション内の Web リソースの、配備や再利用が可能な最小の単位。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て配備することができます。「Web モジュールグループ」、「WAR ファイル」、および「Web アプリケーション」も参照してください。

Web モジュールグループ Sun ONE Studio 4 IDE では、いくつかの Web モジュールが一緒に配備されます。「Web モジュール」、「Web アプリケーション」、および「WAR ファイル」も参照してください。

XML (Extensive Markup Language) 必要なタグの定義によって XML ドキュメント内のデータやテキストを識別できるようにしたマークアップ言語。J2EE 配備記述子は XML で表現されます。

アプリケーションイベントリスナー サーブレットイベントリスナーのインターフェースを実装するクラス。配備時に Web アプリケーション内でインスタンス化されて登録されます。「サーブレットイベントリスナー」も参照してください。

カスタムタグ アプリケーション開発者が追加の Java 機能を提供するために JSP ページ内に作成するタグ。「JSP タグ」も参照してください。

クッキー サーバーが個々のクライアントを追跡するために使用する機構。少量のデータを HTTP 要求および応答のヘッダーに含めて送信します。Web リソースによって応答のヘッダー内に送信クッキーが設定されます。この応答を受信したクライアントでは、クッキーの有効期限が切れるまでクッキーを保管します。クライアントは、クッキーの送信元サーバーに対するすべての HTTP 要求にこのクッキーを含めて送信します。クッキードメインで指定されているサーバーにクッキーを送信することもできます。また、クッキーのパスが指定されていた場合、クライアントは、そのパスに一致するサーバーへの要求にもクッキーを含めて送信します。Web サイトでは、たとえば、複数のブラウザセッション間で特定のユーザー ID を記憶しておく手段としてこの技術を使用します。「HTTP セッション」も参照してください。

クライアント 通信のクライアントサーバーモデルにおいて、リモートサーバーのリソースを要求するプロセス。たとえば、計算や記憶空間がその例です。

コンテキストルート Web クライアントのドキュメントルートにマッピングされる名前。たとえば、Web モジュールのコンテキストルートが `/catalog` の場合は、要求の URL は `http://`*host*`:8081/catalog/index.html` のようになります。

サーバー リソースの管理とクライアントへのサービスの提供を行うネットワークデバイス。J2EE サーバーは Web コンテナまたは EJB コンテナを提供します。「クライアント」および「Web サーバー」も参照してください。

サーバープラグイン Web サーバーのベンダーによって提供される IDE のモジュール。IDE で行うアプリケーションの構成作業およびそのアプリケーションのサーバーへの配備作業を補助します。

サーブレット `javax.servlet` を実装するクラス。通常は、`javax.servlet.http.HttpServlet` のサブクラスになっています。サーブレットを使用すると、Web サーバーや Web 対応のアプリケーションサーバーの機能を拡張することができます。サーブレットはサーブレットコンテナ内で実行されます。通常は、フロントコントローラとして、複雑でない単純な HTTP 応答を生成するために使用されます。「フロントコントローラ」および「HTTP 応答」も参照してください。

サーブレットイベントリスナー サーブレットコンテキストオブジェクトや HTTP セッションオブジェクトの状態の変化を知らせるイベント通知をサポートするクラス。「アプリケーションイベントリスナー」も参照してください。

サーブレットコンテキスト サーブレットが実行されている Web アプリケーションに関するサーブレットのビューを含むオブジェクト。サーブレットコンテキストを使って Web モジュールのリソースを管理することができます。コンテキストを使用すると、サーブレットでさまざまなタスクを実行できます。イベントの記録、リソースへの URL 参照の取得、同じコンテキスト内の他のサーブレットが使用できる属性の設定および保存が可能です。

サーブレットコンテナ ネットワークサービスを提供するコンテナ。ここで、要求や応答の送信、要求の復号化、応答のフォーマット化を行います。すべてのサーブレットコンテナは、要求および応答のためのプロトコルとして HTTP をサポートしています。要求と応答のための追加のプロトコルとして HTTPS をサポートしている場合もあります。この場合、要求はサーブレットによって処理されます。

「分散型」のサーブレットコンテナでは、分散可能としてタグ付けされている Web アプリケーションを実行することができます。アプリケーションは、同じホストまたは異なるホストで実行されている複数の Java 仮想マシンに分散して実行されます。この場合は、Web アプリケーション内のオブジェクトの提供範囲が拡張されます。セッションデータを複数のサーバー間で共有しなければならないので、同期化によるオーバーヘッドが生じます。

サーブレットフィルタ サーブレットが HTTP 要求を受信したときに `HTTPRequest` オブジェクトおよび `HTTPResponse` オブジェクトを検査して変更する、再利用可能なコード。このマニュアルでは、これを「フィルタ」と呼ぶこともあります。

サーブレットマッピング URL パターンとサーブレットの関連付けの定義。受信した要求の URI を調査して要求をサーブレットにマッピングするときに使用されます。「サーブレット」、「URI」、および「HTTP 要求」を参照してください。

スクリプティング変数 タグによって JSP ページにエクスポートされる値。この値は、式またはスクリプトレットで使用することができます。

スクリプト要素 「JSP スクリプト要素」を参照してください。

スクリプトレット 任意の有効な Java コードを JSP ページに入力することができるスクリプト要素。宣言要素で宣言された変数とメソッドは、同じ JSP ファイル内の他のスクリプトレットで使用できます。JSP ページでのスクリプレットの使用はお勧めしません。代わりに、コードをタグまたは bean にカプセル化することをお勧めします。

スコープ Web アプリケーション内の他のオブジェクトに対するオブジェクトの提供範囲の定義。サーブレットおよび JSP の仕様では、`ServletContext` (アプリケーション)、`Session`、`Page` (JSP ページのみ)、および `Request` の 4 つのスコープが定義されています。

セッション 「HTTP セッション」を参照してください。

タグ 「JSP タグ」を参照してください。

タグ属性 タグの処理に使用する値を指示または提供する、タグに関連付けられたパラメータ。

ディスパッチャ ユーザーに対して表示するビューの制御を記述する設計サブパターン。「デザインパターン」、「フロントコントローラ」、「ビューマッパー」も参照してください。

デザインパターン ソフトウェア設計で繰り返し起こる問題をアーキテクチャによって解決するソリューション。デザインパターンは、問題を取り巻く状況や圧力、およびソリューションの結果や効果に対処するための、最適と考えられるさまざまな方法も含んでいます。

ドキュメントルート Web モジュールのルート URL。たとえば、Web モジュールの**コンテキストルート**が `/catalog` の場合は、ドキュメントルートは `http://`*host*`:8080/catalog` のようになります。

配備 ソフトウェアを運用環境にインストールするプロセス。

配備記述子 配備構成情報を記述するファイル。このマニュアルで「配備記述子」というときは、Web モジュールの `WEB-INF` ディレクトリにある `web.xml` という名前のファイルを意味します。配備記述子は、Web モジュールの配備環境、つまりサーブレットコンテナに必要な構成情報を提供します。この情報には、外部リソースの要件やセキュリティおよび環境のパラメータが含まれます。また、コンポーネント固有のパラメータやアプリケーション固有のパラメータも含まれます。

ビュー作成ヘルパー JSP ページ内のデータをさまざまなデザインパターンで表示するために使用されるタグハンドラークラス。

ビューマッパー クライアントのタイプによってリソースが異なる場合に要求の処理を決定するオブジェクト。

フィルタ 「サーブレットフィルタ」を参照してください。

複合ビュー 複数のコンポーネントビューから 1 つの集約的ビューを作成するデザインパターン。複合ビューには、ページ内の複数の動的なモジュール部分を含めることができます。「テンプレート」とも呼ばれます。「デザインパターン」も参照してください。

ブラウザ 「Web ブラウザ」を参照してください。

フレームワーク 設計とコーディングの処理を簡略化するための API。「Struts」および「JSF」も参照してください。

フロントコントローラ Web アプリケーションの一部に関するビジネスロジックを 1 つのオブジェクトに集めて、受信するクライアント要求がすべてそのオブジェクトで処理されるようにしたデザインパターン。クライアント要求を使ってアクセスするリソースには、いくつかの種類があります。ビジネスロジックを実行するヘルパーや委任を、フロントコントローラによって起動することができます。モデルデータの管理、ページフローの制御、および適切なビューへの要求のディスパッチにもフロントコントローラを使用できます。「ヘルパー」および「ディスパッチャ」も参照してください。

ヘルパー 処理機能 (またはビジネスルール) またはデータ取得動作をカプセル化する設計サブパターン。「デザインパターン」および「フロントコントローラ」も参照してください。

リスナー 「アプリケーションイベントリスナー」および「サーブレットイベントリスナー」を参照してください。

索引

I

J

M

P

R

S

T

あ

い

え

か

く

こ

さ

し

す

せ

そ

た

へ

ほ

ま

も

よ

ら

り

れ

ろ