WebOTX アプリケーション開発ガイド

WebOTX アプリケーション開発ガイド

バージョン: 7.1

版数: 第 2 版

リリース: 2010 年 1 月

目次

3.J2EE WTP

本章では、WebOTX Developer の機能を使いこなすための詳細な説明を行います。また、WebOTX が提供する API の利用方法について説明します。

3.1.Web アプリケーション

Developer’s Studio の Web アプリケーション開発環境では、Web アプリケーション開発の生産性を向上させる機能を提供しています。

3.1.1.Web アプリケーションの開発

Web アプリケーションとは

Web アプリケーションは、複数のコンポーネントの集まりです。一般的に Web アプリケーションは以下のコンポーネントを 1 つまたは複数組み合わせて作成されます。

- Servlet
- JSP
- 標準的な JavaBean とユーティリティクラス
- 静的な HTML、DHTML、XHTML、XML や類似ページ
- マルチメディアファイル(イメージファイル、音声ファイルなど)
- クライアントサイドのアプレット、スタイルシート、JavaScript ファイル
- テキスト
- メタ情報(web.xml など)

Web アプリケーションのディレクトリ構造

Web アプリケーションは、通常、ディレクトリの階層構造を構成し、ほとんどの Web コンテナがサーブレット仕様で推奨されているディレクトリ階層構造に準拠しています。階層内のルートディレクトリが Web アプリケーションのドキュメントルート(コンテキストルート)となり、この配下にコンポーネントを配置します。

WTP でプロジェクトを作成する場合、コンテキストルートの名前やコンテキストディレクトリを指定することができます。デフォルトでは、コンテキストルートの名前:”プロジェクト名”、コンテキストディレクトリ:”WebContent” になります。

Web アプリケーションのディレクトリ階層内には、次の表の特殊なディレクトリがあります。

ディレクトリ名	説明
WEB-INF	Web アプリケーションに関連するメタ情報のレポジトリです。このディレクトリ配下のリソースは、クライアント(WEB ブラウザ)からアクセスすることはできません。Web アプリケーション内の Servlet や Java クラスからはアクセス可能です。
WEB-INF/classes	Servlet クラスはユーティリティクラスやリソースプロパティファイル(*.properties)などを配置します。クラスが Java パッケージ内に存在する場合、classes 配下にはパッケージ名に対応するサブディレクトリが存在しなければなりません。
WEB-INF/lib	Web アプリケーション用のクラスローダでロードしたい JAR ファイルを配置します。ここに配置した JAR ファイル中のクラスファイルを Web アプリケーションで参照することができます。

上記のディレクトリの他に、特殊なファイルとして Web アプリケーションの配備記述子である web.xml ファイルがあります。一般的に、web.xml ファイルは WEB-INF ディレクトリ配下に配置されます。

WAR ファイル

MEMO

Web コンテナとは、J2EE の Web 層で動作するアプリケーション(Servlet、JSP)を制御・管理・実行するコンテナを指します。

Web アプリケーションは、WAR(Web Archive)ファイルとしてパッケージ化することができます。WAR ファイルは、.war という拡張子でなければなりません。WAR ファイルとしてパッケージする際には、コンテキストルートからの相対的な Web アプリケーションのディレクトリ構造を維持しなければなりません。

MVC モデル 2

Web アプリケーションの開発モデルとして、オブジェクト指向プログラミング言語のひとつ「Smalltalk」に使用されていたアプリケーション・アーキテクチャ「MVC」が改めて注目されています。MVC を Web アプリケーションに応用したものを「MVC モデル 2」とよびます。Web アプリケーションにおける MVC の Model、View、Control の役割はそれぞれ、以下のようになります。

- Model：EJB などによる業務ロジックを記述する部分
- View：HTML、JSP による Web デザインを記述する部分
- Control：Servlet によるクライアントからの要求と業務処理の連携制御を記述する部分

MVC のメリットとしては、次のことが考えられます。

- 「Model」、「View」、「Control」ごとに独立性を与えられる
- Web デザイナは Web デザインに、プログラマはロジック・プログラミングに注力できる
- プログラム・HTML の可読性が高まり、画面の修正・プログラムの修正を独立して行うことができる
- コンポーネントとして、ビジネスロジックの再利用性を高めることができる

このメリットを含み、開発(設計・コーディング・テスト)を効率化するための指針を具体化し、アプリケーション構築時に生じる問題を解決する手段を提供したものを、MVC ベースの Web アプリケーション・フレームワークとしてとらえることができます。Web アプリケーション・フレームワークには、JSF(JavaServer Faces)や多くの Web システムで採用されている Struts フレームワークなどがあります。

Web アプリケーションの定義

Web アプリケーションの定義ファイルとして、配備記述子である web.xml ファイルがあります。このファイルは Web コンテナに配備する際に必要となります。定義する内容には以下のものがあります。

- ServletContext オブジェクトの初期パラメータ
- セッション・コンフィグレーション
- Servlet 宣言
- Servlet マッピング
- アプリケーション・ライフサイクル・リスナ
- Filter 宣言と Filter マッピング
- MIME タイプ・マッピング
- Welcome ファイルのリスト
- エラーページ
- ロケールとエンコードマッピング
- セキュリティ
- EJB の参照
- JSP 関連の設定

Servlet2.4 仕様から web.xml が XML スキーマベースになったことにより、特定のスキーマを指定することで新しい要素を追加することができます。web.xml のルート要素である web-app 要素の記述は以下のようになります。(赤字は、WTP で作成したプロジェクトの web.xml で追加されます)

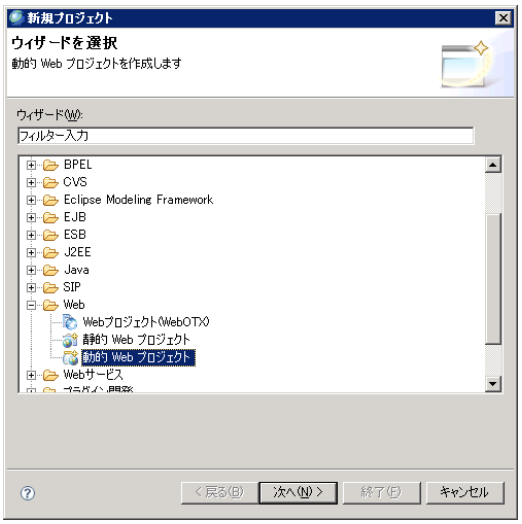
```
<web-app id="WebApp_ID" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee web-app_2_4.xsd"
    version="2.4">
      (ここに上述した設定内容に対応する定義を記述します。)
</web-app>
```

動的 Web プロジェクト・ウィザードの活用

Web プロジェクト・ウィザードは、Web アプリケーションを開発するためのプロジェクトを作成します。

ファイル(F) | **新規**(N) | **プロジェクト**(R)を選択して、**新規プロジェクト**ダイアログを表示します。

新規プロジェクト画面の **Web** 配下の **動的 Web プロジェクト**を選択して、[**次へ**]をクリックします。

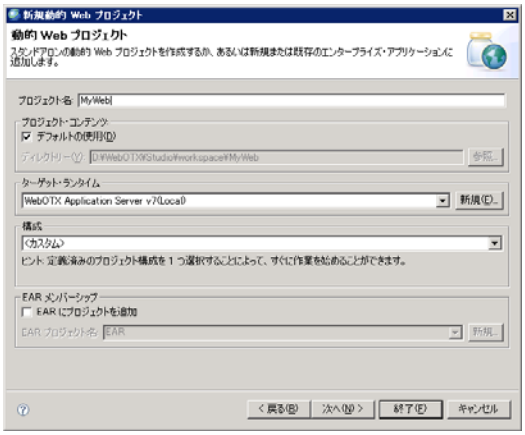


プロジェクト名にプロジェクトの名前を入力します。

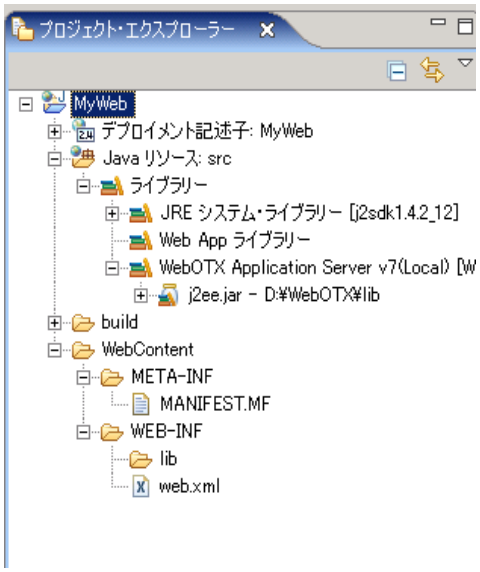
プロジェクトコンテンツでは、プロジェクトの保管先をデフォルト以外の場所に変更することができます。

プロジェクト名を入力すれば、[**終了**]ボタンをクリックすることで **Web プロジェクト**を作成することができます。

[**次へ**]ボタンをクリックすると**プロジェクト・ファセットページ**が表示されます。



プロジェクト名を入力後、[**終了**]ボタンで**動的 Web プロジェクト**を作成した場合、次の図に示されるディレクトリ構成で作成します。



作成されるディレクトリ構成について、次の表に説明します。

項目	説明

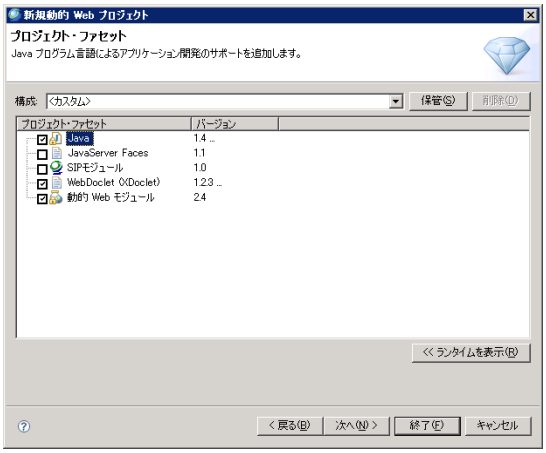
プロジェクトルート(MyWeb)	プロジェクトのルートディレクトリです。この配下で作業をします。任意の HTML・JSP ファイルやイメージファイル・ディレクトリなどを作成して配置します。
src	Web プロジェクトのソースフォルダです。Web アプリケーションで利用する Java ファイルやサーブレット Java ファイルをパッケージ構成で格納します。
JRE システム・ライブラリー WebOTX v6 Runtime j2ee.jar Web アプリケーション・ライブラリー	プロジェクトが参照するライブラリです。
WebContent	コンテキストディレクトリです。プロジェクト作成時に、任意のディレクトリに変更することができます。
MATA-INF/MANIFEST.MF	マニフェストファイルです。特に変更は行いません。
WEB-INF	Web アプリケーションの隠蔽ディレクトリです。この配下にあるリソースは、クライアント(WEB ブラウザ)からアクセスすることはできません。Web アプリケーション・プログラムが利用する定義ファイルなどを格納します。
build/classes	Web プロジェクトの出力フォルダです。ソースフォルダ中のソースファイルのコンパイルで生成されるファイルが格納されます。
WEB-INF/lib	開発する Web アプリケーションで利用するライブラリ(JAR ファイル)を格納します。
WEB-INF/web.xml	Web アプリケーションの配備記述子です。

動的 Web プロジェクトで**プロジェクト名**を入力後、[**次へ**]ボタンをクリックした場合、**プロジェクト・ファセットページ**が表示されます。

プロジェクト・ファセットページでは、作成する **Web プロジェクト**の以下の設定を行います。

Project Facet の **WebDoclet(XDoclet)**を選択して、[**終了**]をクリックします。

[**次へ**]ボタンをクリックすると **Web Module ページ**が表示されます。

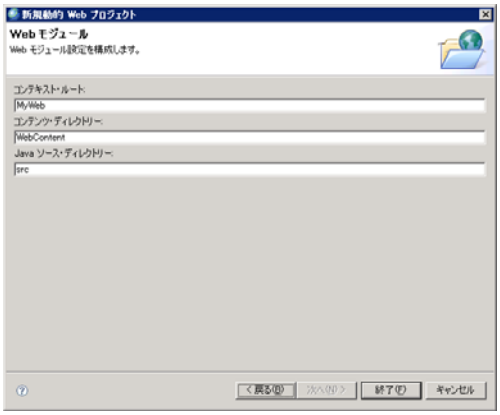


項目	説明
動的 Web モジュール	必須。プロジェクトが Dynamic Web Module として配備されることを可能にします。
Java	必須。Java のサポートを可能にします。
WebDoclect(XDoclet)	任意。WebOTX サーバに必要な配備記述子(nec-web.xml)が作成されます。

プロジェクト・ファセットで、[**次へ**]ボタンをクリックした場合、**Web Module ページ**が表示されます。

Web モジュールページでは、作成する **Web プロジェクト**の以下の設定を行います。

変更が必要な場合のみ値を変更して、[**終了**]をクリックします。

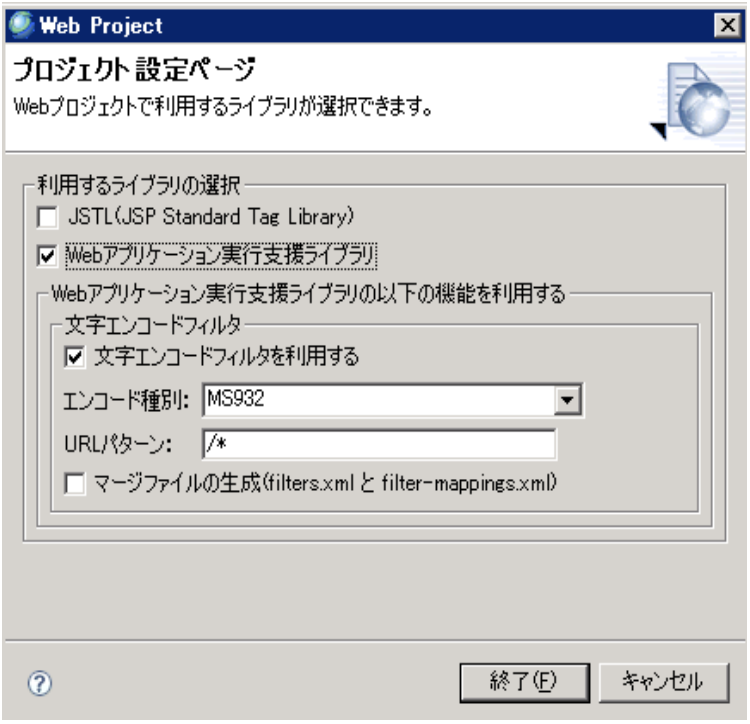


項目	説明
コンテキスト・ルート	コンテキストルートの変更が可能です。 デフォルトはプロジェクト名が入っています。
コンテンツ・ディレクトリー	コンテントディレクトリの変更が可能です。 デフォルトは WebContent。
Java ソース・ディレクトリー	Java のソースディレクトリの指定ができます。

実行支援ライブラリの活用

パッケージエクスプローラ以外の場合はファイルメニューの「**ウィンドウ**」→「**パースペクティブを開く**」より「**Java**」を選択し、パッケージエクスプローラを表示します。

利用したいプロジェクト上で、**右クリック**し「**Web プロジェクト**」→「**支援ライブラリの追加**」を選択します。



プロジェクト設定ページでは、Web プロジェクトが利用するライブラリの選択を行うことができます。

■ 利用するライブラリの選択

開発する Web アプリケーションで利用するライブラリを選択します。JAR ファイルや TLD ファイルをプロジェクトの所定の位置に配置します。

プロジェクト設定ページで選択できる項目の詳細について、次の表に説明します。

項目	説明
JSTL(JSP 標準タグライブラリ)	Jakarta プロジェクトの Taglibs サブプロジェクトで実装されている Standard Taglib(JSTL のオープンソース実装)の JAR ファイルと TLD ファイルをプロジェクトに配置します。
Web アプリケーション実行支援ライブラリ	Web アプリケーションを開発する際に汎用的に利用できるライブラリです。JAR ファイルと TLD ファイルをプロジェクトに配置します。プロジェクト作成時に文字エンコードフィルタの利用選択ができます。

MEMO

文字エンコードフィルタを利用する場合、**エンコード種別**と**URL パターン**の指定は必須です。

3.1.2.Servlet の開発

Servlet とは

Servlet は、Servlet コンテナにより管理される動的コンテンツを生成する Java テクノロジをベースとした Web コンポーネントです。他の Java テクノロジをベースとしたコンポーネントと同様に、Servlet はプラットフォームに依存しない Java クラスです。この Java クラスは、Java テクノロジをサポートする Web サーバにより動的にロードされ実行されるプラットフォーム中間バイトコードです。Servlet エンジンとも呼ばれる、Servlet コンテナとは、Servlet 機能を提供する Web サーバ拡張です。Servlet は、Servlet コンテナにより実装される request/response パラダイムを経由する Web クライアントと相互作用します。

簡潔にいうと、Servlet とは、ブラウザによって発せられたリクエストに応答し、レスポンスを組み立てるための Java クラスです。

Servlet コンテナとは

Servlet コンテナは、MIME ベースにデコードしたり、MIME ベースにフォーマットされるリクエストとレスポンスのネットワーク・サービスを提供する Web サーバまたはアプリケーション・サーバの一部です。また、Servlet コンテナは、Servlet のライフサイクルを制御・管理します。

Servlet コンテナはホスト・Web サーバに組み込むか、あるいはそのサーバのネイティブ拡張 API によって Web サーバへのアドオン・コンポーネントとしてインストールすることができます。また、Servlet コンテナは、Web が使用可能なアプリケーションサーバへ組み込むことができるか、あるいは、インストールすることができます。

すべての Servlet コンテナはリクエストとレスポンス用のプロトコルとして HTTP をサポートしています。しかし、HTTPS(SSL の上の HTTP)のような追加のリクエスト/レスポンスペースのプロトコルがサポートされる可能性もあります。コンテナが実装する HTTP のバージョンは HTTP/1.0 および HTTP/1.1 です。コンテナが RFC2616(HTTP/1.1)に記述されるキャッシングメカニズムを保有する場合、コンテナは Servlet にリクエストを配達する前にクライアントからのリクエストを修正するかもしれないし、あるいはクライアントにレスポンスを送る前に Servlet によって生産されたレスポンスを修正するかもしれないか、RFC2616 に従い、Servlet にそれらを配達せずに、レスポンスを返却するかもしれません。

Servlet コンテナは、Servlet が実行される環境に対してセキュリティ制限を配置するかもしれません。Java 2 プラットフォーム、スタンダード・エディション(J2SE™、v.1.3 以上)あるいは Java 2 プラットフォーム エンタープライズ・エディション(J2EE™、v.1.3 以上)環境では、これらの制限が Java 2 プラットフォームによって定義されるパーミッション・アーキテクチャを使用して配置されるかもしれません。

Servlet のインタフェース

Servlet プログラムを作成する上で必要なインタフェースやクラスを説明します。

javax.servlet.Servlet インタフェース

Servlet は直接的、もしくは、間接的に必ず javax.servlet.Servlet インタフェースを実装している必要があります。javax.servlet.Servlet インタフェースには、Servlet プログラムが必ず実装していなければいけないメソッドが定義されています。

javax.servlet.GenericServlet クラス

javax.servlet.Servlet インタフェースを実装したクラスで、プロトコルに依存しない Servlet を作成する際に使用します。

javax.servlet.http.HttpServlet クラス

javax.servlet.GenericServlet クラスを継承したクラスで、HTTP プロトコルベースの Servlet プログラムを作成する際に使用します。通常このクラスを継承して Servlet を作成します。HttpServlet クラスは抽象クラスのため、このクラスを継承した Servlet プログラムは必ず 1 つのメソッドをオーバーライドする必要があります。

Servlet のライフサイクル

Servlet コンテナは、次の動作により Servlet のライフサイクルを管理します。

- ロードとインタンス化
- 初期化
- リクエストのハンドリング
- サービスの停止

ロードとインスタンス化

各ベンダ製品により多少異なりますが、Servlet コンテナは、以下のような契機で、Servlet をロードしてインスタンス化します。

- Servlet コンテナ起動時、または、Web アプリケーション起動時(配備時など)
- クライアントから Servlet に対して、初めてリクエストが発生したとき

明示的に、前者の契機で Servlet をロードしたい場合は、**web.xml** の Servlet 定義に **load-on-startup** 要素を定義することで可能です。

初期化

Servlet に初めてリクエストが行われたとき、Servlet コンテナは最初に Servlet の初期化を行います。その際に **init()**メソッドが呼ばれます。Servlet コンテナは、**init()**メソッドを 1 回だけ呼び出します。**init()**メソッドでは、事前設定が必要な処理などを記述します。具体的には、次のような処理を記述することが考えられます。

- データベースへの接続手続き
- 永続的なコンフィグレーション･データの読み込み
- **web.xml** の **init-param** 要素で定義した値を **ServletConfig** パラメータから取得

Servlet の初期化の終了後、**ServletConfig** は **getServletConfig()**メソッドで取得できます。

リクエストのハンドリング

Servlet の初期化が正常に行われた後、Servlet コンテナはクライアントからのリクエストをハンドリングします。リクエストは、**ServletRequest** 型のオブジェクトとして表現されます。レスポンスは、**ServletResponse** 型のオブジェクトとして表現されます。HTTP プロトコル通信の場合には、**HttpServletRequest** 型と **HttpServletResponse** 型が Servlet コンテナにより提供されます。**HttpServlet** クラスを継承して Servlet を作成する場合、**HttpServlet** の次のメソッドをオーバーライドします。

戻り型	メソッド	説明
protected void	doGet(HttpServletRequest, HttpServletResponse)	リクエストの HTTP メソッドが GET メソッドの場合、Servlet コンテナが呼び出すメソッドです。
protected void	doPost(HttpServletRequest, HttpServletResponse)	リクエストの HTTP メソッドが POST メソッドの場合、Servlet コンテナが呼び出すメソッドです。
protected void	doDelete(HttpServletRequest, HttpServletResponse)	リクエストの HTTP メソッドが DELETE メソッドの場合、Servlet コンテナが呼び出すメソッドです。
protected void	doHead(HttpServletRequest, HttpServletResponse)	リクエストの HTTP メソッドが HEAD メソッドの場合、Servlet コンテナが呼び出すメソッドです。
protected void	doPut(HttpServletRequest, HttpServletResponse)	リクエストの HTTP メソッドが PUT メソッドの場合、Servlet コンテナが呼び出すメソッドです。

一般的に、**doGet()**メソッド、もしくは、**doPost()**メソッドをオーバーライドします。

その他のサーブレットメソッド

前述のリクエストのハンドリングのほかに、サブクラス化を前提とする抽象クラスとして Servlet のライフサイクルを維持するリソースを管理したい場合に使用される init メソッド および destroy メソッド、Servlet が自身の情報を提供する目的で使用する getServletInfo メソッドなどがあります。

戻り型	メソッド	説明
public void	init()	Servlet コンテナが呼び出すメソッドです。 この Servlet がサービスを開始できる状態になったことを

		示します。
public String	getServletInfo()	作者、バージョン、著作権といった Servlet に関する情報を返します。 デフォルトではこのメソッドは空の文字列を返すだけです。 このメソッドをオーバライドして意味のある値が返るようにしてください。
public void	destroy()	Servlet コンテナが呼び出すメソッドです。この Servlet がサービス提供を停止するときに呼び出されます。

サービスの停止

Servlet コンテナのリソースが少なくなり、一定時間 Servlet にアクセスがない場合や、Servlet コンテナを終了する場合などに、Servlet コンテナは Servlet のサービスを停止します。その際に **destroy()**メソッドが呼ばれます。destroy()メソッドでは、終了処理などを記述します。具体的には、次のような処理を記述することが考えられます。

- データベースの切断処理
- Servlet が利用していたリソース(例えば、メモリ、ファイルハンドラ、スレッド)の解放処理

Servlet とスレッド

通常の Servlet は、複数のユーザからの同時アクセスがある場合、複数のスレッドとして並列処理を行います。このとき、各スレッドは同じ Servlet インスタンスの service()メソッドに同時にアクセスします。これは、Servlet がスレッドセーフにプログラミングされていることを前提とした動作です。つまり、Servlet はスレッドセーフを意識して開発することが重要です。

この問題の一つの解として、javax.servlet.SingleThreadModel インタフェースを利用する方法が考えられます。これは Servlet のインスタンスとスレッドが 1 対 1 にマッピングされるモデルで、ブラウザからのアクセス数に合わせて Servlet のインスタンスを複数用意することにより、Servlet のインスタンス変数をスレッドセーフとすることができます。しかし、SingleThreadModel インタフェースには、以下のような問題があるため、利用することは望ましくありません。

- Servlet のインスタンスとスレッドを 1 対 1 とするため、アクセスの頻繁な Servlet を SingleThreadModel インタフェースで実装した場合、コストパフォーマンスが劣化してしまう。
- Servlet2.4 仕様では非推奨となっているため、今後、なくなってしまう可能性がある

スレッドセーフな Servlet プログラミングの考え方は、マルチスレッド環境での Java プログラミングと同様です。ポイントとしては、以下のものが挙げられます。

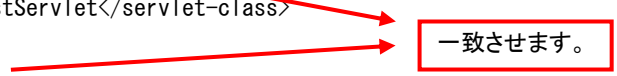
- 設計段階において、共有リソース(インスタンス変数・クラス変数などのメモリデータやファイル、データベースコネクションなど)の扱いに対する規約化を行う
- 共有リソースを扱う場合、共有リソースへのアクセスの同期化
- 同時接続による負荷試験など、スレッドセーフを意識したテスト手法の確立

マルチスレッドの問題は開発するアプリケーションの要件により、さまざまなケースが考えられます。スレッドセーフを意識したプログラミング・開発を心がけましょう。

Servlet の定義

作成した Servlet は web.xml に定義をします。Servlet の定義は web-app 要素の子要素である servlet 要素と servlet-mapping 要素で定義します。servlet 要素では、Servlet のクラスと Servlet の名前(web.xml 内で一意)を定義して、servlet-mapping 要素では servlet 要素で定義した Servlet と URL のマッピングを定義します。以下に Servlet の定義例を記載します。

```
<servlet>
    <servlet-name>TestServlet</servlet-name>
    <servlet-class>servlet.TestServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>TestServlet</servlet-name>
    <url-pattern>/TestServletUri</url-pattern>
```



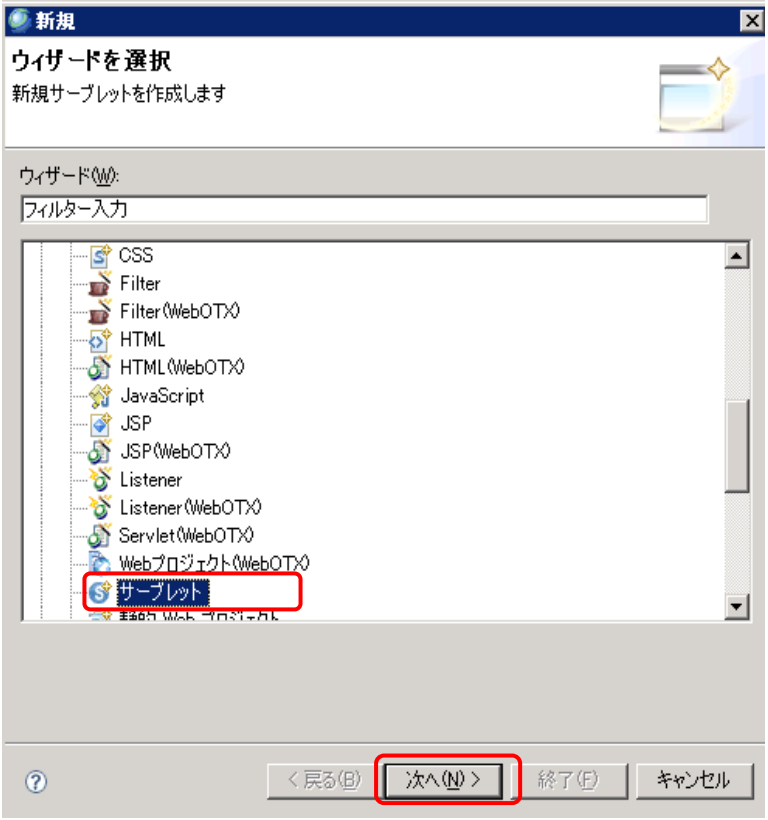
```
</servlet-mapping>
```

この記述例の意味は、『servlet.TestServlet』を『TestServlet』という名前で定義して、『/TestServlet』の URL でアクセスされると TestServlet にリクエストを渡すということです。

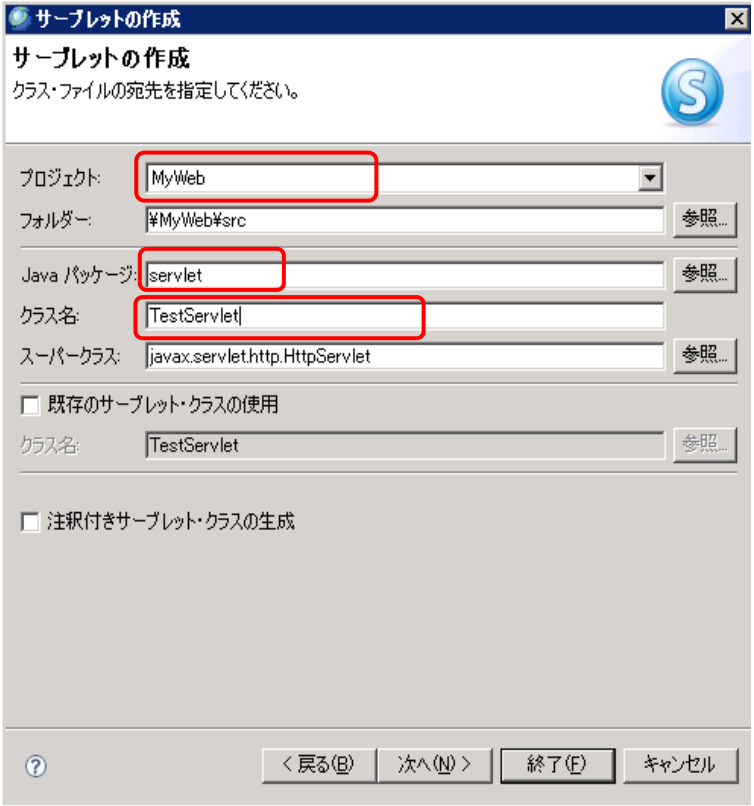
サーブレットウィザードの活用

サーブレットウィザードは、Servlet を作成して、さらに、Web プロジェクト中の web.xml ファイルに作成する Servlet の定義を追加することができます。

ファイル(F) | **新規**(N) | **その他**(O)...を選択して、**新規**画面を表示します。



新規画面の Web 配下の**サーブレット**ウィザードを選択して、[**次へ**]ボタンをクリックします。



フォルダーに Web プロジェクトのソースフォルダーが設定されていることを確認します。

クラス名では、作成する Servlet のクラス名を入力します。

Java パッケージは、作成する Servlet が属するパッケージを入力します。

スーパークラスには、初期値で HttpServlet クラスが設定されています。

ここまでで[**終了**]ボタンをクリックして Servlet クラスを作成した場合も、**web.xml** には記述の追加が行われます。

[**次へ**]ボタンをクリックすると、**サーブレット・デプロイメント記述子固有情報を入力する**画面が表示されます。

デプロイメント記述子固有情報入力画面では、**web.xml** に記述する Servlet の配備記述を入力します。次の表に入力項目の説明をします。

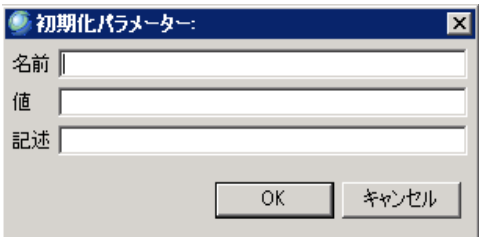
名称	説明
名前	servlet 要素と servlet-mapping 要素の子要素である servlet-name 要素の値となります。
記述	servlet 要素の子要素である desplay-name 要素の値となります。
URL マッピング	servlet-mapping 要素の子要素である url-pattern 要素の値となります。
初期化パラメータ	servlet 要素の子要素である init-param 要素の定義を編集できます。

初期化パラメータの右端にある[**追加**]ボタンをクリックすると**初期化パラメータの設定**ダイアログが表示されます。

名前の入力値が param-name 要素の値になります。

値の入力値が param-value 要素の値になります。

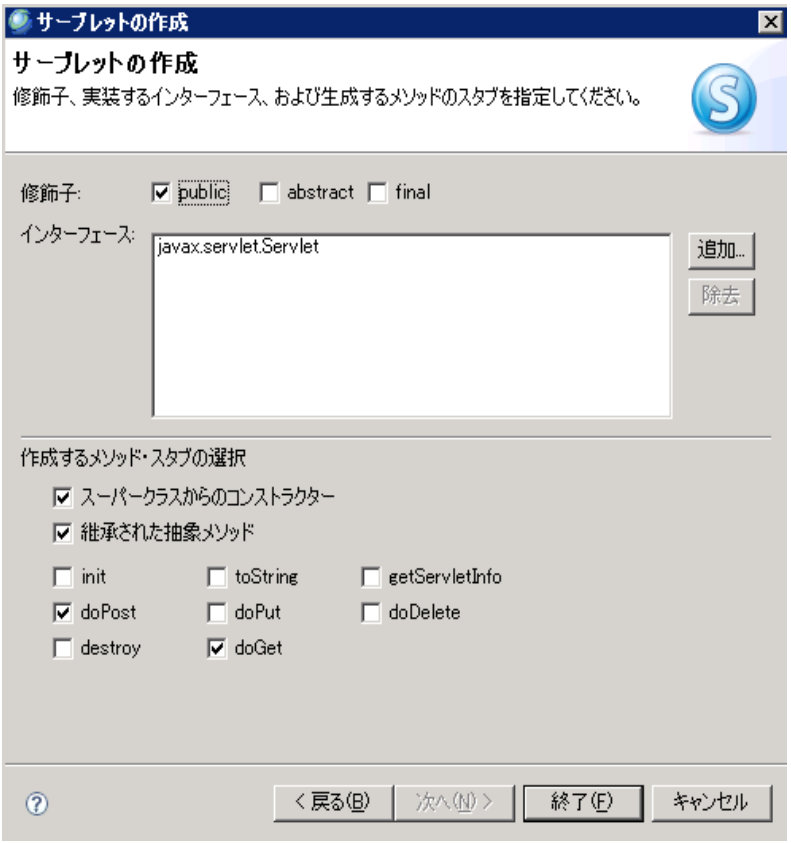
記述の入力値が description 要素の値になります。



[**次へ**]ボタンをクリックすると、以下の画面が表示されます。

MEMO

Servlet 名と **URL** は必須項目です。



本画面では、サーブレットの修飾子、実装するインターフェース、および生成するメソッドのスタブを指定することができます。次の表に入力項目の説明をします。

名称	説明
メソッド・スタブの作成	作成するメソッドのスタブクラスを選択します。メソッドの詳細は、前述の**リクエストのハンドリング**および**その他のサーブレットメソッド**を参照してください。

その他は eclipse 標準のクラスウィザードと同様です。

3.1.3.JSP ファイル開発

JSP とは

JSP(JavaServer Pages)は、HTML 要素などで表現される静的コンテンツと JSP 要素で表現される動的コンテンツの 2 つのタイプを含むテキストドキュメントです。JSP は、通常の Java のソースコードと違ってコンパイルする必要がありません。JSP のソースコードをサーバ(Web コンテナ)に配置するだけで実行可能になります。この秘密は、Web コンテナのアーキテクチャにあります。Web コンテナは、JSP ファイルにリクエストが発生した時点でそれを Java のソースコード(Servlet)に変換して(JSP コンパイル)、さらにそれを Java コンパイルした上で実行します。

JSP 要素

JSP は、HTML をベースに記述します。JSP 固有の記述には次の表の 7 種類があります。

名前		記述イメージ	説明
ディレクティブ		<%@ . . . %>	Web コンテナに JSP ページの情報を指示します。
コメント		<%-- . . . %>	コメントを記述します。
スクリプティング	スクリプトレット	<% . . . %>	記述される Java コードを実行します。
	式	<%= . . . %>	式を文字列として記述します。
	宣言	<%! . . . %>	変数の定義など Java コードに関する宣言をします。
アクション		<jsp:include. . . > など	JSP 内でオブジェクトを利用、編集や生成したりします。
暗黙オブジェクト		request、response など	JSP 内の Java プログラムで利用できるオブジェクトです。
EL(Expression Language)		${param.value}など	シンプルな記述でテンプレートデータや属性値を表現できます。JSP2.0 仕様より導入されました。

ディレクティブ

ディレクティブには、page、taglib、include の 3 つがあります。まず、page ディレクティブは、そのページに依存する設定を Web コンテナに知らせます。

page ディレクティブのシンタックスは次のようになります。

```
<%@ page (page ディレクティブ属性) %>
page ディレクティブ属性 ::= { language=" scriptingLanguage" }
                             { extends=" className" }
                             { import=" importList" }
                             { session=" true|false" }
                             { buffer=" none|sizekb" }
                             { autoFlush=" true|false" }
                             { isThreadSafe=" true|false" }
                             { info=" info_text" }
                             { errorPage=" error_url" }
                             { isErrorPage=" true|false" }
                             { contentType=" ctinfo" }
                             { pageEncoding=" peinfo" }
                             { isELIgnored=" true|false" }
```

通常、1 つの宣言に複数の属性を定義するのではなく、1つのインポート宣言で1つの page ディレクティブ、文字エンコード関連で 1 つの page ディレクティブといった感じに分けて記述したりします。

page ディレクティブに設定できる属性は次のようになります。

属性	説明
language	スクリプトレットで使用するスクリプト言語を定義します。デフォルトで使用できる言語は“java”です。
extends	JSP サーブレットが継承するスーパークラスを指定します。これは、JSP ファイルを JSP サーブレットに変換するときに使用されます。Java 言語の extends に相当します。通常、デフォルト以外のものを使用するケースは多くありません。
import	宣言、スクリプトレット、式で使用するパッケージをインポートします。デフォルトでインポートされるパッケージは、”java.lang.*”と”javax.servlet.*”と”javax.servlet.jsp.*”と”javax.servlet.http.*”です。
session	HTTP セッションを使用するかを true/false で定義します。true とする場合、暗黙オブジェクト”session”が利用できます。デフォルトの定義は”true”です。
buffer	クライアントに送信するデータをバッファする際に、そのバッファの容量を指定します。容量は kb で指定します。バッファしない場合は none を指定します。サイズを小さくすると、サーバのメモリ負荷が少なくなり、ク

MEMO

IANA とはインターネット上で利用されるアドレス資源(IP アドレス、ドメイン名、プロトコル番号など)の標準化や割り当てを行なっていた組織でした。1998 年 10 月、インターネット資源の管理・調整を行なう国際的な非営利法人 ICANN が設立さ

	ライアントは素早く受信をすることができます。デフォルトは、最低 8kb です。
autoFlush	バッファリングされたデータを自動的にフラッシュするかどうかを true/false で定義します。true の場合、バッファがいっぱいになると自動的にフラッシュされて、Web ブラウザにデータが送信されます。false の場合、バッファがいっぱいになってもフラッシュせずに例外を上げて Web ブラウザにエラーページを表示させます。false を指定する場合は buffer 属性に適切な値を定義しなければなりません。また、buffer="none"を定義する場合、autoFlush="false"を定義することはできません。デフォルトの定義は true です。
isThreadSafe	JSP ファイルを JSP コンパイルされて生成される Servlet が、複数のクライアントからのリクエストに対して同時に処理できるかどうかを true/false で定義できます。デフォルトは true です。
info	JSP ファイルの作成者やバージョンなどの情報を記述したいときに利用します。この情報は Servlet インタフェースの getServletInfo()メソッドで取得できます。デフォルトは空の文字列が設定されています。
isErrorPage	この JSP がエラーページかどうかを true/false で定義します。true の場合、JSP ページ内でエラーへの参照となる暗黙オブジェクト、"exception"が利用できます。false の場合、"exception"は利用できません。デフォルトは false です。
errorPage	JSP ファイル内で対応していない例外が発生した場合に転送するエラーページを定義します。autoFlush="true"でバッファがフラッシュされた後に例外が発生した場合、定義されるエラーページに転送することはできません。buffer 属性が"none"の場合で例外が発生する前に JSP の出力があると、エラーページに転送することができません。
contentType	JSP のレスポンスの MIME タイプと文字エンコード方式を定義します。定義できる MIME タイプについては、IANA(アイアナ)の『MIME Media Type』を参照してください。定義できる文字エンコード方式については、IANA の『CHARACTER SETS』を参照してください。
pageEncoding	JSP ファイルの文字エンコード方式を定義します。JSP が Servlet に変換される際に利用されます。定義できる文字エンコード方式については、IANA の『CHARACTER SETS』を参照してください。
isELIgnored	JSP2.0 から利用できるようになった式言語(EL)を JSP ファイルで利用するかを true/false で定義します。true の場合、EL 式は Web コンテナに無視されます。false の場合、EL 式が Web コンテナに認識されます。

以下に簡単な利用例を記述します。

- 利用例 1:スクリプト言語を"java"、MIME タイプに"text/html"、文字エンコード方式に"Windows-31J"を使用した例です。

```
<%@ page language="java"
        contentType="text/html; charset=Windows-31J"
        pageEncoding="Windows-31J" %>
```

- 利用例 2:バッファを"32k バイト"、autoFlush を"true"に設定した例です。

```
<%@ page buffer="32kb" autoFlush="true" %>
```

- 利用例 3:インポート宣言の例です。カンマ区切りで複数のパッケージやクラスを宣言できます。

```
<%@ page import="java.util.*, java.io.*" %>
```

- 利用例 4:インポート宣言の例です。パッケージやクラスで 1 つの宣言をしています。

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

次に、taglib ディレクティブについて説明します。taglib ディレクティブは、JSP で利用するカスタムタグライ

れて、2000 年 2 月には、ICANN、南カリフォルニア大学、アメリカ政府の三者合意により、IANA が行なっていた各種資源の管理はICANNに移管されました。現在では、IANA は ICANN における資源管理・調整機能の名称として使われています。

ブラリの宣言です。

taglib ディレクティブのシンタックスは次のようになります。

```
<%@ taglib ( uri=”タグライブラリ URI” | tagdir=”タグライブラリディレクトリ”)
             prefix=”カスタムタグの接頭辞” %>
```

taglib ディレクティブに設定できる属性は次のようになります。

属性	説明
uri	prefix 属性の定義値に対応するタグライブラリ記述子(TLD ファイル)の絶対 URI、相対 URI を定義します。
prefix	JSP ファイル内に記述するカスタムタグライブラリの接頭辞を定義します。
tagdir	uri 属性の代わりとして、Web アプリケーション内のタグライブラリ記述子の格納ディレクトリを定義することができます。”WEB-INF/tags”で始まらない定義値は Web コンテナによりエラーにされます。

最後に include ディレクティブについて説明します。include ディレクティブは、外部のテキストファイルや JSP ファイルをインクルードするときに利用します。

include ディレクティブのシンタックスは次のようになります。

```
<%@ include file="インクルードファイルの相対 URL" %>
```

スクリプティング

スクリプティングには、スクリプトレット・式・宣言の 3 つがあります。以下にそれぞれの使用例を説明します。

- スクリプトレットの使用例:時間帯(午前か午後)によって表示する文字を切り替えます。

```
<% if (Calendar.getInstance().get(Calendar.AM_PM) == Calendar.AM) {%>
    おはよう
<% } else { %>
    こんにちは
<% } %>
```

- 式の使用例:年月日を文字列で表示します。

```
<%= (new java.util.Date()).toString() %>
```

- 宣言の使用例:int 型の変数『i』の宣言をしている例と int 型の変数『j』の値により何らかの文字列を返却するメソッド『getIString』を宣言しています。

```
<%! int i = 0; %>
```

```
<%! public String getIString(int j) {
      if (i<3) return( “...” );
       ...
    }
%>
```

アクション

JSP で利用できる主なアクションには次のようなものがあります。

アクション	説明

<jsp:useBean>	JavaBeans をインスタンス化して特定のスコープに登録を行い、JSP ページから使用可能にするアクションです。
<jsp:setProperty>	<jsp:useBean>などで生成したインスタンスに対して、値の設定を行うアクションタグです。
<jsp:getProperty>	<jsp:useBean>などで生成したインスタンスに対して、取得・表示を行うアクションタグです。
<jsp:include>	リソース(ページ)のインクルードをするアクションです。
<jsp:forward>	リクエストを転送するアクションです。
<jsp:param>	キーと値のペアの情報を扱うアクションです。このアクションを利用して<jsp:include>や<jsp:forward>に値を渡すことができます。
<jsp:plugin>	Java プラグインをダウンロードしてアプレットや JavaBeans コンポーネントを実行する HTML を生成します。
<jsp:params>	<jsp:plugin>の一部として利用するアクションです。アプレットに渡すパラメータなどを指定します。
<jsp:fallback>	<jsp:plugin>の一部として利用するアクションです。ブラウザがサポートしていないタグがある場合に表示するテキストを指定します。

暗黙オブジェクト

JSP で利用できる暗黙オブジェクトには次のようなものがあります。

オブジェクト	スコープ	説明
request	request	HTTP リクエスト(javax.servlet.http.HttpServletRequest)のオブジェクトです。
response	page	HTTP レスポンス(javax.servlet.http.HttpServletResponse)のオブジェクトです。
pageContext	page	この JSP ページ(javax.servlet.jsp.PageContext)のオブジェクトです。
session	session	HTTP セッション(javax.servlet.http.HttpSession)オブジェクトです。
application	application	Servlet コンテキスト(javax.servlet.ServletContext)のオブジェクトです。
out	page	出力用ストリーム(javax.servlet.jsp.JspWriter)のオブジェクトです。
config	page	Servlet の設定(javax.servlet.ServletConfig)のオブジェクトです。
page	page	このページを実装するクラスのインスタンス(java.lang.Object)です。

スコープという用語が出てきましたが、スコープとはデータの有効範囲のことを言います。Web アプリケーションでは以下のスコープがあります。

スコープ	説明
request	リクエスト間でデータを共有したい場合に使用します。
session	HTTP セッション間でデータを共有したい場合に使用します。これにより、異なるページ間でブラウザを閉じるまで、もしくはセッションが解放されるまでデータを共有することができます。
application	Web アプリケーション間でデータを共有したい場合に使用します。Web アプリケーション間とは、サーブレットコンテナに設定される Web アプリケーション内のことを言います。
page	ひとつの JSP ファイル中でデータを共有したい場合に使用します。

EL(Expression Language)

EL(Expression Language)は JSP2.0 で追加された機能です。式の結果を出力するために使用され、アクションタグに属性の値として与えることもできます。ただ、EL 単独では高度なロジックを書くことはできません。JSTL などのカスタムタグと組み合わせることにより、JSP ページの中から極力スクリプトレットを排除し、コードの簡素化をすることができます。

- EL とスクリプティングの式の相違:EL では、変数名を記述することによりプロパティアクセスができます。

```
従来の式の例
        <%= o.getDta() %>
EL の例
        ${ o.data }
```

EL には次の専用の暗黙オブジェクトがあります。

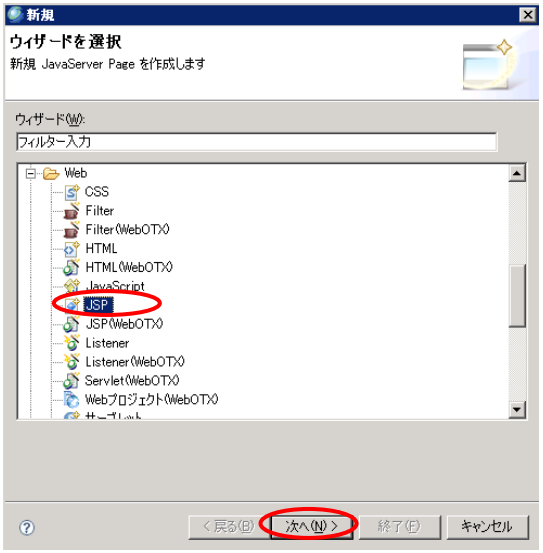
オブジェクト	説明
pageContext	PageContext オブジェクトです。
pageScope	page スコープに格納されたキーと値の Map 型オブジェクトです。
requestScope	request スコープに格納されたキーと値の Map 型オブジェクトです。
sessionScope	session スコープに格納されたキーと値の Map 型オブジェクトです。
applicationScope	application スコープに格納されたキーと値の Map 型オブジェクトです。
param	リクエストパラメータのパラメータ名とパラメータ値を格納する Map 型オブジェクトです。(パラメータ値は、ServletRequest.getParameter("パラメータ名")で取得できる値です。)
paramValues	リクエストパラメータのパラメータ名とパラメータ値(複数)を格納する Map 型オブジェクトです。(パラメータ値は、ServletRequest.getParameterValues("パラメータ名")で取得できる値です。)
header	ヘッダーのキーと値を格納する Map 型オブジェクトです。(値は、ServletRequest.getHeader("キー")で取得できる値です。)
headerValue	ヘッダーのキーと値(複数)を格納する Map 型オブジェクトです。(値は、ServletRequest.getHeaders("キー")で取得できる値です。)
cookie	Cookie のキーと値を格納する Map 型オブジェクトです。
initParam	コンテキストの初期化パラメータのキーと値を格納する Map 型のオブジェクトです。(値は、ServletContext.getInitParameter("キー")で取得できる値です。)

JSP ウィザードの活用

JSP ウィザードの画面について、説明します。

ファイル(F) | **新規(N)** | **その他(O)**を選択して、**新規**ダイアログを表示します。

新規ダイアログで **Web | JSP** を選択して、[**次へ**]をクリックします。

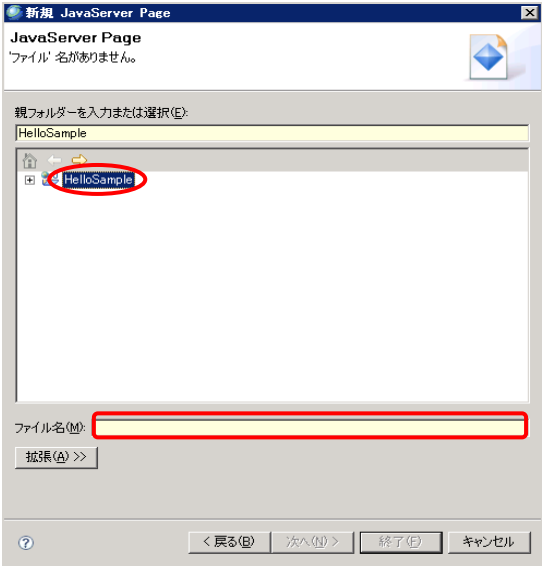


親フォルダ(JSP ファイルの作成場所)をツリーもしくは直接入力により指定します。

このとき WebContent ディレクトリ配下を指定するようにします。

次に**ファイル名(M)**を入力すれば、[**終了**]ボタンをクリックすることで **JSP ファイル**を作成することができます。

[**次へ**]ボタンをクリックすると作成時の **JSP ファイル**の詳細な設定することができる **JSP 詳細**が表示されます。



ファイル名(N)を入力後、[**終了**]ボタンで**JSP ファイル**を作成した場合、次の図に示される JSP ファイルが作成されます。

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
    pageEncoding="windows-31j"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>Insert title here</title>
</head>
<body>

</body>
</html>
```

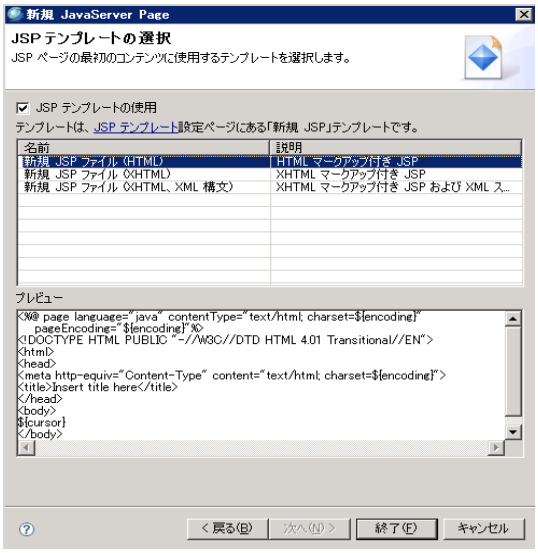
上記の図の **JSP ファイル**が作成されますので、用途に応じて JSP ファイルの編集を行って下さい。

JSP ウィザードで**ファイル名(M)**を入力後、**[次へ]**ボタンをクリックした場合、**JSP テンプレートの選択**が表示されます。

JSP テンプレートの選択では、作成する **JSP ファイル**のテンプレートを選択して JSP ファイルを作成することができます。

新規に JSP のテンプレートの作成して利用することも可能です。

JSP テンプレートの選択で選択できる項目の詳細について、次の表に説明します。



JSP テンプレート

項目	説明
新規 JSP ファイル(HTML)	HTML マークアップつき JSP ファイルを作成します。
新規 JSP ファイル(XHTML)	XHTML マークアップつき JSP ファイルを作成します。
新規 JSP ファイル (XHTML、XML 構文)	XHTML マークアップつき JSP ファイル および XML スタイル構文を作成します。

3.1.4.Filter の開発

Filter とは

Filter は、HTTP リクエストや HTTP レスポンス、ヘッダ情報などの内容を変換することのできる再利用可能なオブジェクトです。Servlet2.3 仕様より導入されています。Filter の利用例として以下のものが挙げられます。

- 認証フィルタ
- ログおよび監査フィルタ
- イメージ変換フィルタ
- データ圧縮フィルタ
- 暗号化フィルタ
- トークン化フィルタ
- リソースアクセスイベントのきっかけとなるフィルタ
- XML コンテンツの変換をする XSL/T フィルタ
- MIME タイプチェーンフィルタ
- キャッシュフィルタ

これら以外にも、要件に応じていろいろなフィルタを開発することができます。

Filter のインタフェース

Filter は javax.servlet.Filter インタフェースを実装して作成します。javax.servlet.Filter インタフェースで実装しなければならないメソッドには以下のものがあります。

メソッド	説明
void init(FilterConfig filterConfig) throws ServletException	Filter がサービス開始される際に Web コンテナによって 1 度だけ呼び出されるメソッドです。
void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws java.io.IOException, ServletException	Filter が適用されるリクエストがあるたびに呼び出されるメソッドです。このメソッドでリクエストやレスポンス、ヘッダ情報を加工するフィルタ処理を実装します。『chain.doFilter()』で次のフィルタ、もしくは、リソースに処理を渡します。
void destroy()	Filter がサービス終了される際に Web コンテナによって呼び出されるメソッドです。

javax.servlet.Filter インタフェースの他に、Filter 関連のインタフェースとして、javax.servlet.FilterChain インタフェースと javax.servlet.FilterConfig インタフェースがあります。

- FilterChain：フィルタされるリクエストの実行チェーンを可視化するために、Web コンテナにより FilterChain インタフェースのオブジェクトが提供されます。Filter はチェーン中の次のフィルタを実行するのに FilterChain を使用したり、もし、Filter がチェーン中で最後のフィルタであるならば、リソースに処理を渡したりするために FilterChain を使用します。

- FilterConfig：Web コンテナは、Filter の初期化処理中に使用する Filter の設定情報を持つ FilterConfig インタフェースのオブジェクトを提供します。このオブジェクトから Filter の初期化パラメータを取得したりできます。

Filter の定義

作成した Filter は、web.xml に定義します。Filter の定義は filter 要素と filter-mapping 要素で定義します。filter 要素では Filter のクラス名と Filter の名前(web.xml 内で一意)を定義して、filter-mapping 要素では filter 要素で定義した Filter をどの URL、もしくは、Servlet にマッピングするかを定義します。以下に Filter の定義例を記載します。

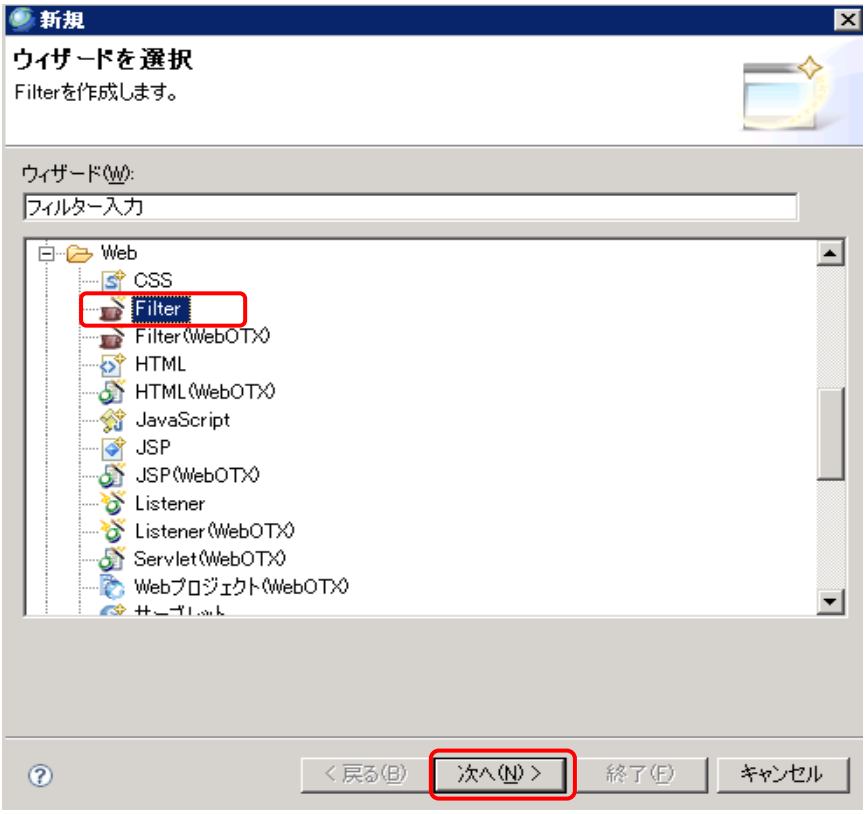
```
<filter>
    <filter-name>TestFilter</filter-name>
    <filter-class>filter.TestFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>TestFilter</filter-name>
    <url-pattern>/TestServlet</url-pattern>
</filter-mapping>
```

この記述例の意味は、『filter.TestFilter』を『TestFilter』という名前で定義して、『/TestServlet』の URL でアクセスされると TestFilter のフィルタ処理を行うということです。

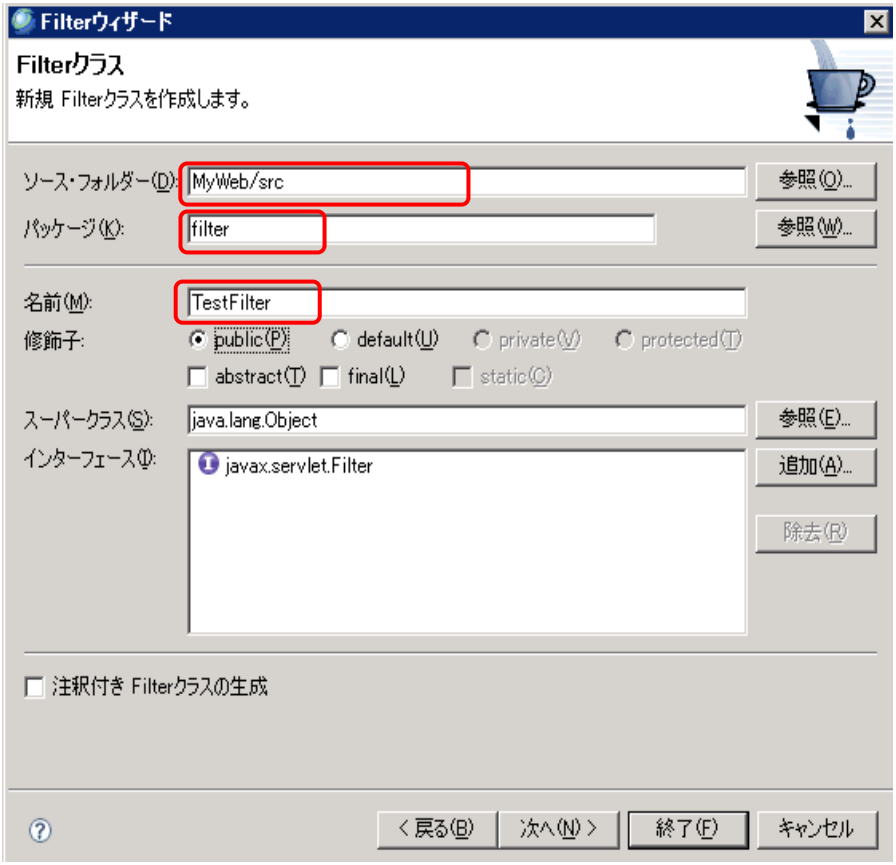
Filter ウィザード

Filter ウィザードは、Filter を作成して、さらに、Web プロジェクト中の web.xml ファイルに作成する Filter の定義を追加することができます。

ファイル(F) | **新規(N)** | **その他(O)...**を選択して、**新規**画面を表示します。



新規画面の **Web** 配下の **Filter** を選択して、[**次へ**]ボタンをクリックします。



ソース・フォルダー(D)に Web プロジェクトのソースフォルダーが設定されていることを確認します。

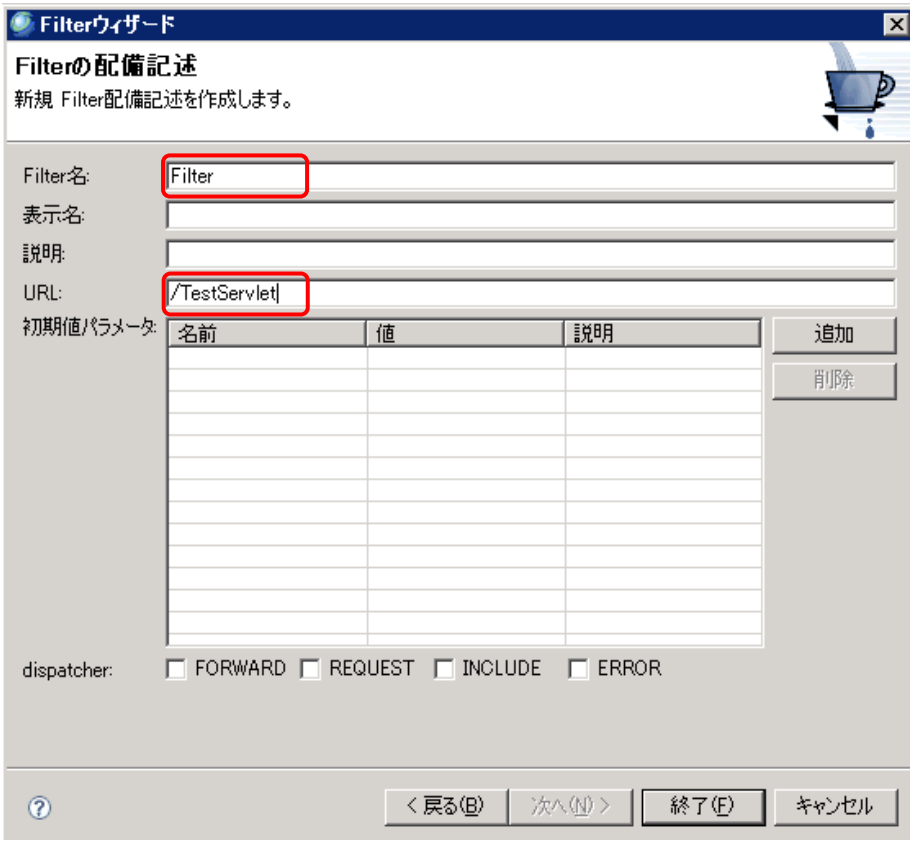
名前(M)では、作成する Filter のクラス名を入力します。

インターフェース(I)でインプリメントするインターフェースを必要に応じて追加します。(デフォルト javax.servlet.Filter)

これら以外の箇所は、eclipse 標準のクラスウィザードと同様です。

ここまでで[**終了**]ボタンをクリックして Filter クラスを作成すると、**web.xml** には記述の追加を行わずに、Filter の作成のみを行います。

[**次へ**]ボタンをクリックすると、**Filter の配備記述**画面が表示されます。



Filter の配備記述画面では、**web.xml** に記述する Filter の配備記述を入力します。次の表に入力項目の説明をします。

名称	説明
Filter 名	filter 要素と filter-mapping 要素の子要素である filter-name 要素の値となります。
表示名	filter 要素の子要素である desplay-name 要素の値となります。
説明	filter 要素の子要素である description 要素の値となります。
URL	filter-mapping 要素の子要素である url-pattern 要素の値となります。
初期化パラメータ	filter 要素の子要素である init-param 要素の定義を編集できます。
dispatcher	filter-mapping 要素の子要素である dispatcher 要素の値となります。

MEMO

Filter 名と **URL** は必須項目です。

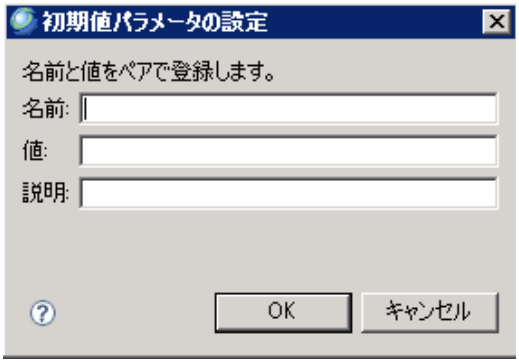
MEMO

dispatcher は **Servlet2.4** 以上で設定可能です。

名前の入力値が param-name 要素の値になります。

値の入力値が param-value 要素の値になります。

説明の入力値が description 要素の値になります。



3.1.5.Listener の開発

Listener とは

Web アプリケーションでは、Web アプリケーション内のイベントをキャッチしてそのイベントが発生した際に処理を行うことのできるリスナという技術が提供されています。イベントの対象として、ServletContext と HttpSession と ServletRequest があります。イベントの種類には次のものがあります。

イベント	説明
コンテキストの生成と破棄	Web コンテナの起動時や Web アプリケーションの配備時など、Web アプリケーションが起動して初期化されたときがコンテキスト(ServletContext)の生成されるときです。Web コンテナの停止などのより，Web アプリケーションが停止してコンテキストが破棄されるときです。
コンテキスト属性の操作	コンテキスト属性を追加、削除、置換したときです。
セッションの生成と破棄	セッションが生成されるときとタイムアウトなどにより破棄されるときです。
セッション属性の操作	セッション属性を追加、削除、置換したときです。
セッションの活性化と非活性化	セッションが活性化、または、非活性化されたときです。
リクエストの発生と終了	Web コンテナでリクエストを受け付けたとき(HttpServletRequest を作成したとき)とリクエストが終了するときです。
リクエスト属性の操作	リクエスト属性を追加、削除、置換したときです。

MEMO

セッションの活性化と非活性化は少し特殊で、分散環境やセッションを永続的に保持する Web コンテナなどで有用です。つまり、利用する環境や Web コンテナを意識する必要があります。

Listener のインタフェース

Lintener は次に上げるインタフェースを実装して作成します。

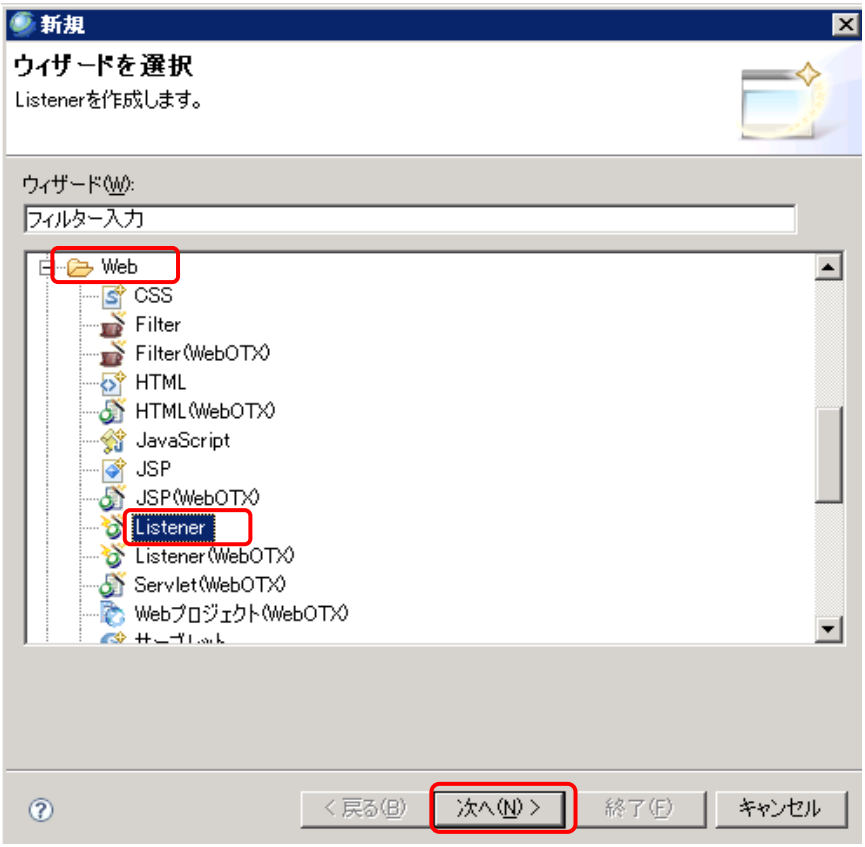
インタフェース	説明
javax.servlet. ServletContextListener	コンテキストの生成と破棄のイベントが発生したときの処理を実装します。contextInitialized(ServletContextEvent sce)メソッドと contextDestroyed(ServletContextEvent sce)メソッドを実装します。
javax.servlet. ServletContextAttributeListener	コンテキスト属性を操作したイベントが発生したときの処理を実装します。attributeAdded(ServletContextAttributeEvent scab)メソッドと attributeRemoved(ServletContextAttributeEvent scab)メソッドと attributeReplaced(ServletContextAttributeEvent scab)メソッドを実装します。
javax.servlet.http.	セッションの生成と破棄のイベントが発生したときの処理を実装します。sessionCreated(HttpSessionEvent se)メソッドと

HttpSessionListener	sessionDestroyed(HttpSessionEvent se)メソッドを実装します。
javax.servlet. HttpSessionAttributeListener	セッション属性の操作のイベントが発生したときの処理を実装します。attributeAdded(HttpSessionBindingEvent se)メソッドとattributeRemoved(HttpSessionBindingEvent se)メソッドとattributeReplaced(HttpSessionBindingEvent se)メソッドを実装します。
javax.servlet. HttpSessionActivationListener	セッションの活性化と非活性化のイベントが発生したときの処理を実装します。sessionDidActivate(HttpSessionEvent se)メソッドとsessionWillPassivate(HttpSessionEvent se)メソッドを実装します。
javax.servlet. HttpSessionBindingListener	valueBound(HttpSessionBindingEvent event)メソッドとvalueUnbound(HttpSessionBindingEvent event)メソッドを実装します。

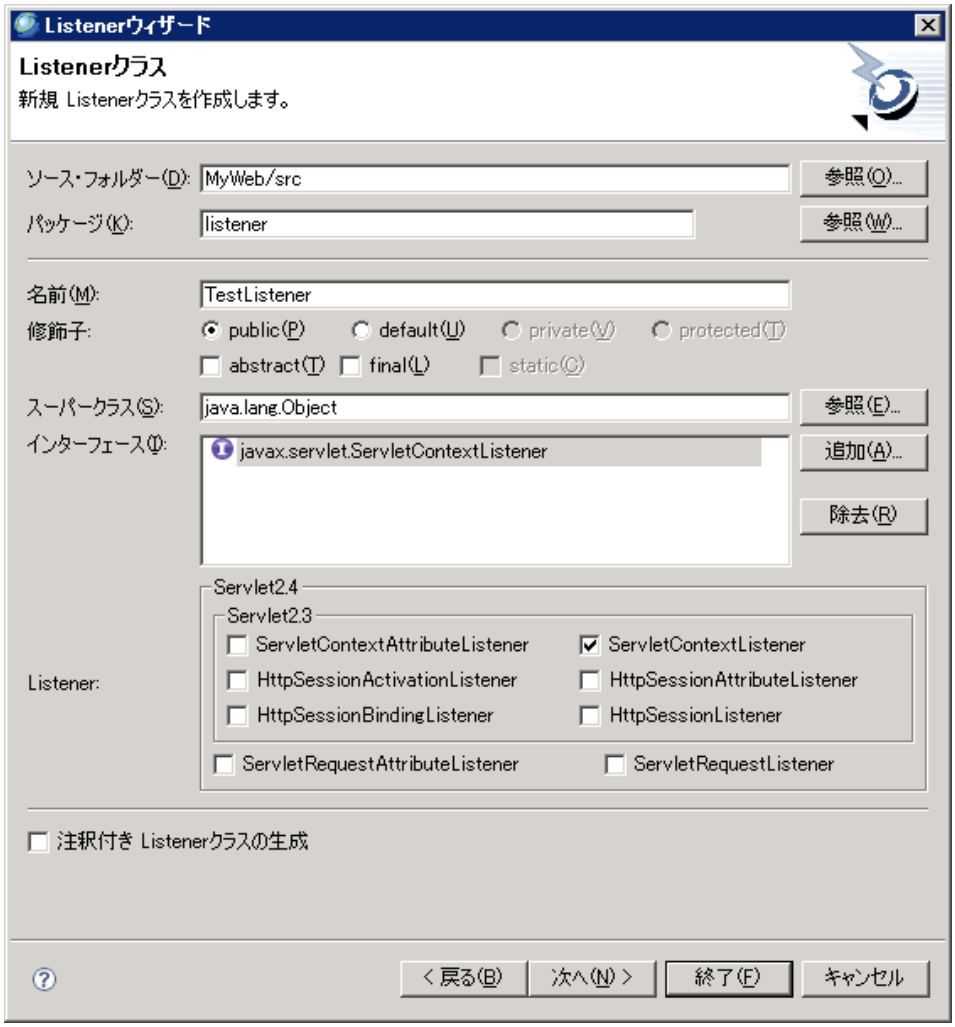
Listener ウィザード

Listener ウィザードは、Listener を作成して、さらに、Web プロジェクト中の web.xml ファイルに作成する Listener の定義を追加することができます。

ファイル(F) | **新規(N)** | **その他(O)...**を選択して、**新規**画面を表示します。



新規画面の **Web** 配下の **Listener** を選択して、[**次へ**]ボタンをクリックします。



ソース・フォルダー(D)に Web プロジェクトのソースフォルダーが設定されていることを確認します。

名前(M)では、作成する Listener のクラス名を入力します。

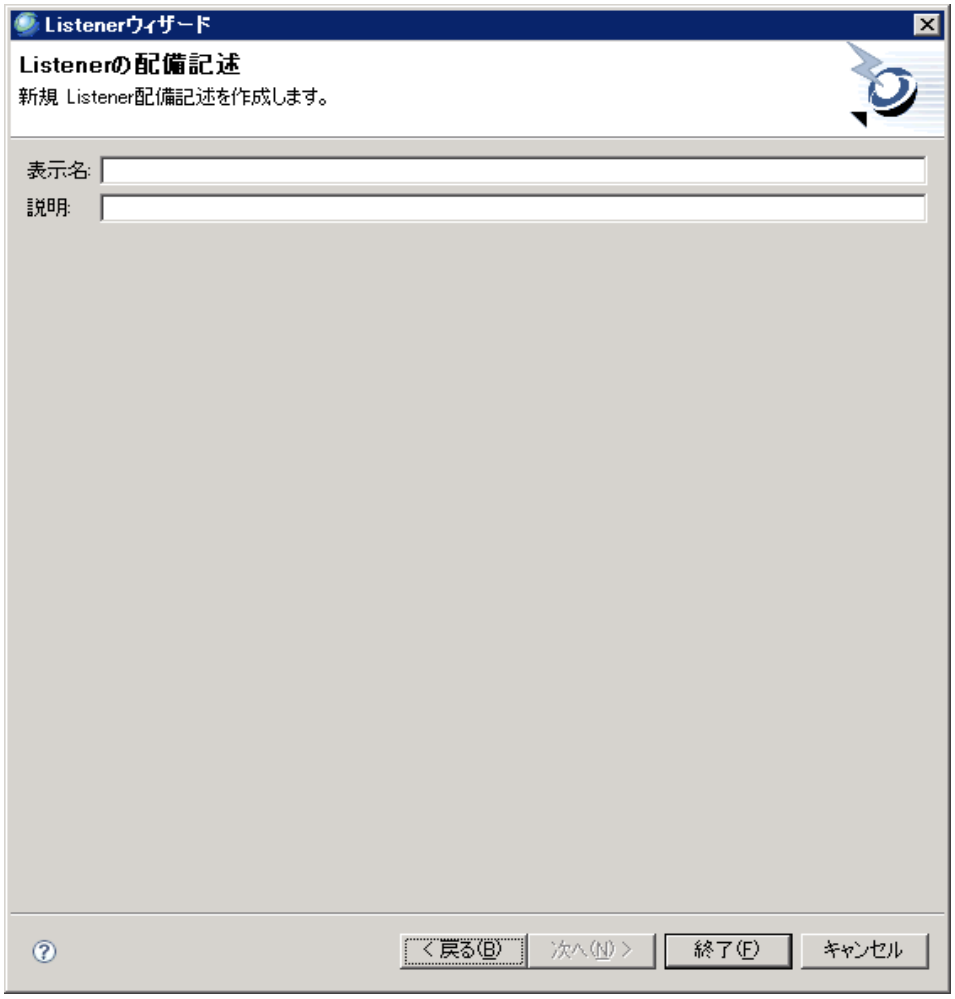
インターフェース(I)でインプリメントするインターフェースを必要に応じて追加します。

Listener：Listener 関連のインターフェースが選択可能です。インプリメントするインターフェースをチェックします。（インターフェース(I)で選択するよりも便利です。）

これら以外の箇所は、eclipse 標準のクラスウィザードと同様です。

ここまでで[**終了**]ボタンをクリックして Listener クラスを作成すると、**web.xml** には記述の追加を行わずに、Filter の作成のみを行います。

[**次へ**]ボタンをクリックすると、**Listener の配備記述**画面が表示されます。



Listener の配備記述画面では、**web.xml** に記述する Listener の配備記述を入力します。次の表に入力項目の説明をします。

名称	説明
表示名	listener 要素の子要素である desplay-name 要素の値となります。
説明	listener 要素の子要素である description 要素の値となります。

MEMO

Servlet2.3 の場合はこの画面では何も設定しないで[**終了**] をクリックしてください。

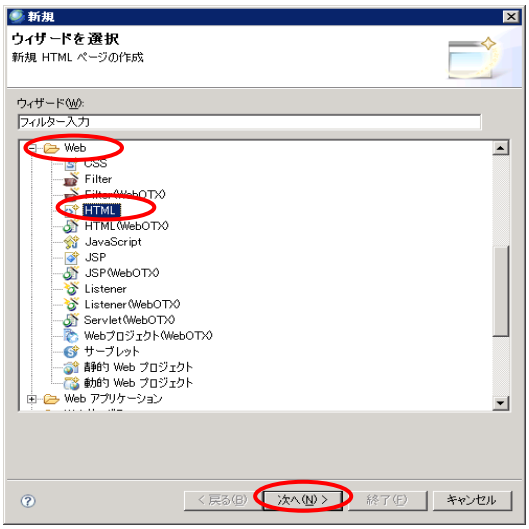
3.1.6.HTML ファイル作成

HTML ウィザード

HTML ウィザードの画面について、説明します。

ファイル(F) | **新規**(N) | **その他**(O)を選択して、**新規**ダイアログを表示します。

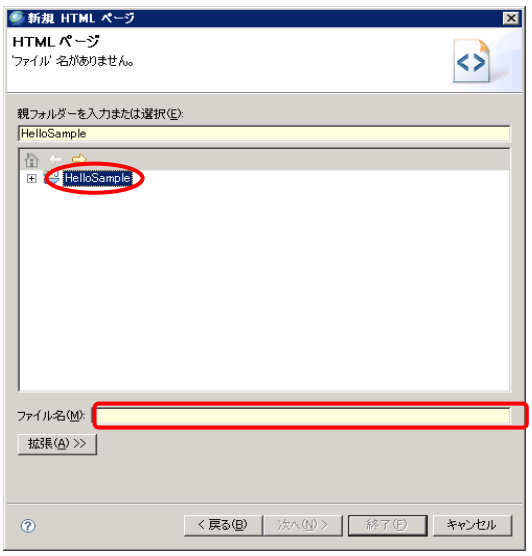
新規ダイアログで **Web** | **HTML** を選択して、[**次へ**]をクリックします。



親フォルダ(HTMLファイルの作成場所)をツリーもしくは直接入力により指定します。

このときWebContentディレクトリ配下を指定するようにします。

次に**ファイル名**(M)を入力すれば、[**終了**]ボタンをクリックすることで **HTML ファイル**を作成することができます。



ファイル名(N)を入力後、[**終了**]ボタンで **HTML ファイル**を作成した場合、次の図に示される HTML ファイルが作成されます。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>ここにタイトルを挿入</title>
</head>
<body>

</body>
</html>
```

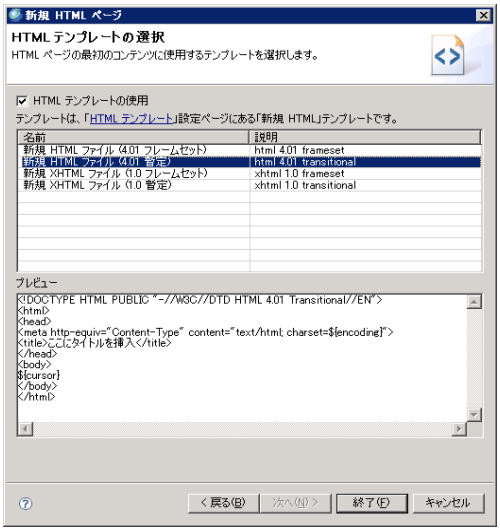
上記の図の HTML ファイルが作成されますので、用途に応じて HTML ファイルの編集を行って下さい。

HTML ウィザードで**ファイル名(M)**を入力後、**[次へ]**ボタンをクリックした場合、**HTML テンプレートの選択**が表示されます。

HTML テンプレートの選択では、作成する**HTML ファイル**のテンプレートを選択して HTML ファイルを作成することができます。

新規に HTML のテンプレートの作成して利用することも可能です。

HTML テンプレートの選択で選択できる項目の詳細について、次の表に説明します。



HTML テンプレート

項目	説明
新規 HTML ファイル (4.01 フレームセット)	HTML 4.01 仕様でフレームセットを利用したい場合の HTML ファイルを作成します。
新規 HTML ファイル(4.01 暫定)	HTML4.01 仕様の標準的な HTML ファイルを作成します。
新規 XHTML ファイル (1.0 フレームセット)	XML をベースとした XHTML ファイルでフレームセットを利用したい場合に作成します。
新規 XHTML ファイル(1.0 暫定)	XML をベースとした標準的な XHTML ファイルを作成します。

3.1.7.従来の Web アプリケーションを WTP プロジェクトへ変換する方法

WebOTX V6.3 以前で作成した Web アプリケーションのプロジェクトを WTP プロジェクトに変換する場合の方法について

WebOTX V6.3 以前の Web アプリケーション開発プラグインで作成したプロジェクトを WTP（動的 Web プロジェクト）プロジェクトに変換する方法について、説明します。

変換にあたって、WTP（動的 Web プロジェクト）プロジェクトのファイルを流用するので、既存のプロジェクトが無い場合、あらかじめ、WTP（動的 Web プロジェクト）プロジェクトを作成してください。

手順1
変換前のプロジェクトをインポート後、ナビゲータービューで、.project ファイルを開き、
WTP 関連のネイチャーの追加、プロジェクト名の変更を行い、
以下の **Web アプリケーション開発プラグインのネイチャー**

```
<nature>com.nec.webotx.webap.WebProjectNature</nature>
```

を削除します。
以下が、変更後の.project ファイルの内容です。******** が変更後のプロジェクト名で、その他の太字の箇所が追加された行です。

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>****</name>
    <comment></comment>
    <projects>
    </projects>
    <buildSpec>
        <buildCommand>
          <name>org.eclipse.jdt.core.javabuilder</name>
          <arguments>
          </arguments>
        </buildCommand>
        <buildCommand>
          <name>org.eclipse.wst.common.project.facet.core.builder</name>
          <arguments>
          </arguments>
        </buildCommand>
        <buildCommand>
          <name>org.eclipse.wst.validation.validationbuilder</name>
          <arguments>
          </arguments>
        </buildCommand>
    </buildSpec>
    <natures>
      <nature>org.eclipse.wst.common.project.facet.core.nature</nature>
      <nature>org.eclipse.jdt.core.javanature</nature>
      <nature>org.eclipse.wst.common.modulecore.ModuleCoreNature</nature>
      <nature>org.eclipse.jem.workbench.JavaEMFNature</nature>
    </natures>
</projectDescription>
```

手順 2
動的 Web プロジェクト(WTP プロジェクト)用のファイル構成に変更します。
既存の動的 Web プロジェクト(WTP プロジェクト)のファイル構成を参考に、以下の変更を行います。

①プロジェクト¥WEB-INF¥配下の src フォルダをプロジェクト直下に移動します。

②プロジェクト直下に build¥classes フォルダを追加します。
③プロジェクト直下に WebContent フォルダを追加し、その下に WEB-INF、META-INF フォルダ
および JSP や HTML ファイルを移動します。

手順 3
ナビゲータービューで、.classpath ファイルで、下記のように更新します。

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
  <classpathentry
     kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
  <classpathentry kind="src" path="src"/>
  <classpathentry kind="lib" path="<WebOTX Root>/lib/j2ee.jar"/>
  <classpathentry kind="lib" path="<WebOTX Root>/lib/webc-jasper.jar"/>
  <classpathentry kind="output" path="build/classes"/>
</classpath>
```

※ **<WebOTX Root>**は WebOTX のインストールパス(“C:/WebOTX” など)です。

手順 4
ナビゲータービューで、既存の WTP(動的 Web プロジェクト)から.settings フォルダを、変換する
プロジェクトの直下へコピーします。

.settings フォルダ配下
①org.eclipse.wst.common.component(後述の deploy-name 等の修正が必要)
②org.eclipse.jdt.core.prefs
③org.eclipse.wst.common.project.facet.core.xml
④org.eclipse.jst.common.project.facet.core.prefs

ナビゲータービューで、org.eclipse.wst.common.component ファイルを以下のように
修正します。(****はプロジェクト名に変更します)

```
<?xml version="1.0"encoding="UTF-8"?>
<project-modules id="moduleCoreId">
    <wb-module deploy-name="****">
       <wb-resource source-path="/WebContent" deploy-path="/"/>
       <wb-resource source-path="/src" deploy-path="/WEB-INF/classes"/>
       <property name="context-root" value="****"/>
       <property name="java-output-path" value="/build/classes/"/>
    </wb-module>
</project-modules>
```

手順 5

- Eclipse を再起動してください。

3.1.8.Web アプリケーション実行支援ライブラリ利用について

Web アプリケーション実行支援ライブラリ

Web アプリケーション実行支援ライブラリは、Web アプリケーションで必要とされる機能をライブラリで提供したものです。以下の実行支援ライブラリを利用することにより、作成効率が高まります。

- 文字エンコード機能

Web アプリケーション実行支援ライブラリを利用する前に

Web アプリケーション実行支援ライブラリを利用する前に次のことを確認して下さい。

Web アプリケーション実行支援ライブラリを利用する場合は、プロジェクト中の WEB-INF 配下に otxjsptag.tld ファイル、WEB-INF¥lib 配下に webapsupport.jar ファイルが存在している必要があります。

以下に、Web アプリケーション実行支援ライブラリの各機能の概要と利用手順を示します。

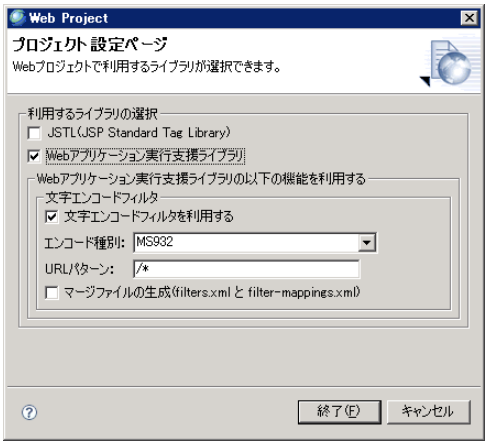
文字エンコード機能

文字エンコード機能を利用することで、HTTP リクエストを Servlet へ引き渡す際の問題により発生する文字化け問題を回避することが可能になります。本機能は、Servlet コンテナのフィルタリング機能を利用して作られています。

手順は以下のとおりです。

手順 1

WTP の**動的 Web プロジェクト**にて、プロジェクトを作成している場合は、**Java パースペクティブ**に切り替えてから、プロジェクトを選択した状態で、右クリックメニュー→「**Web プロジェクト**」→「**支援ライブラリの追加**」を選択し、「**利用するライブラリの選択**」で「**Web アプリケーション実行支援ライブラリ**」のチェックボックスをチェックしてください。



手順 2

Web プロジェクトのプロジェクト設定ページで

「**Web アプリケーション実行支援ライブラリ**」をチェックします。(このとき「**文字エンコードフィルタを利用する**」にチェックが入っていない場合はチェックしてください)

手順 3

「**エンコード種別**」と「**URL パターン**」を必要に応じて変更します。

MEMO

「WTP プロジェクト」の作成方法は、「**動的 Web プロジェクト・ウィザードの活用**」の項目を参照して下さい 。

設定上の注意事項

- 「エンコード種別」には、 MS932、SJIS (または Shift_JIS)、UTF8、EUC-JP 等の中のいずれかを設定して下さい。(例)MS932 ページの日本語の文字化けの対処には MS932 を設定します。)

使用できる文字コードの例

文字コード	説明
MS932	Windows 日本語
SJIS	Shift-JIS、日本語
UTF8	8 ビット Unicode 変換形式
EUC-JP	日本語 EUC

サポートしない文字コードの例(使用しないで下さい)

文字コード	説明
JISAutoDetect	Shift-JIS、EUC-JP、ISO 2022 JP の検出および変換 (Unicode 変換のみ)

- 「URL パターン」にはエンコードを行うページの URL パターンを指定します。すべてエンコードしたい場合は ”/*” と指定します。
- Web プロジェクトの作成後にエンコード種別や URL パターンを変更したい場合は、web.xml エディタを利用して、変更して下さい。(詳しくは、web.xml エディタの項目を参照して下さい。)

3.1.9.Valve の開発

Vavle とは

Web コンテナの内部の処理として、クライアントからのリクエストをキャッチして、処理することができます。Filter と似ていますが、Valve は、Web コンテナ自体に組み込むものである点が異なります。Web コンテナ自体に組み込むため、Valve の処理に不備があると、Web コンテナに影響を与えます。通常の処理であれば、Valve ではなく Filter を利用されることを強く推奨します。

MEMO
リクエストやレスポンスのデータ処理は、Filter を使うことを推奨します。

Valve のインタフェース

Valve は次のクラスをに上げるクラスを継承して作成します。

クラス	説明
org.apache.catalina.valves. ValveBase	ValveBase を継承して、Valve を作成します。 invoke(Request request, Response response)メソッドに Valve の処理を実装します。

ValveBase のインタフェースは次のようになっています。

クラス	説明
public String getInfo()	この Valve の情報を文字列で返却します。作成した Valve の説明を返却することができます。
public Valve getNext()	次に実行する Valve を返却します。処理を実装する必要はありません。
public void invoke(Request request, Response response) throws java.io.IOException, javax.servlet.ServletException	Valve の処理を実装します。
public void setNext(Valve valve)	次に実行する Valve を設定します。処理を実装する必要はありません。

invoke メソッド

invoke メソッドで Valve の処理を実装します。

invoke メソッドのインタフェースは次のようになります。

public void invoke(Request request, Response response)
throws java.io.IOException, javax.servlet.ServletException

ソースのサンプルを次に示します。

```
package org.apache.catalina.valves;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.catalina.Request;
```

```
import org.apache.catalina.Response;
import org.apache.catalina.Container;
import org.apache.catalina.Context;
import org.apache.catalina.Wrapper;

public final class RequestDispatchTestValve extends ValveBase {

    // ----------------------------------------------------------- Constructors

    /**
     * Constructor
     */
    public RequestDispatchTestValve() {
        super();
    }

    // ----------------------------------------------------------- Public Methods

    public void invoke(Request request, Response response)
        throws java.io.IOException., javax.servlet.ServletException {

            // HttpServletRequest 取得
            ServletRequest req = request.getRequest();
            HttpServletRequest hreq = (HttpServletRequest) req;

            // HttpServletRequest からコンテキストパス取得
            String ctxPath = hreq.getContextPath();

            // HttpServletRequest からセッション取得
            HttpSession session = hreq.getSession(false);

            // request からこのリクエストのコンテキスト取得
            Context ctx = request.getContext();
            String ctxName = ctx.getName();

            ……必要な処理の実装

            getNext().invoke(request, response);  //リクエストの実行
    }

}
```

3.1.10.プログラムで認証を行えるAPIの使用方法

クライアントからユーザ名とパスワードを入力し認証を行うかわりに、プログラムで認証を行うことが便利な場合があります。このような場合に利用するAPIの使用方法について説明します。 この API を利用することで、クライアントはBASIC認証もしくはFORMベース認証において、ユーザ名、パスワードを入力することなく認証が取れた状態にすることができます。

プログラムで認証が行えるAPI（ServletAuthenticator）の使用方法について説明します。

Web アプリケーションでの利用方法

Web アプリケーションでの ServletAuthenticator の利用方法を説明します。

ServletAuthenticatorを使用し、プログラムで認証を行うWeb アプリケーションは次のように作成します。

1) 対象となる Web アプリケーションを作成します。

2) BASIC 認証もしくは FORM 認証に関する設定を web.xml および nec-web.xml に行います。

3) ServletAuthenticator を呼び出して認証させるための servlet もしくは JSP を追加します。

MEMO

web.xml および nec-web.xml の設定

ServletAuthenticator を呼び出して認証させるための servlet もしくは JSP には、認証が不要となるように設定する必要があります。

Servlet から、ServletAuthenticator を呼び出して認証する場合

以下のようなコードを、servlet 内の適切な箇所に入れます。

```
import org.apache.catalina.authenticator.ServletAuthenticator;

String username="admin";
String password="adminadmin";

ServletContext application = getServletContext();
ServletAuthenticator sau = new ServletAuthenticator( application );
sau.authenticate(req,res,username,password);
 // req は、 HttpServletRequest オブジェクト、 res は、 HttpServletResponse オブジェクトです
```

JSP から、ServletAuthenticator を呼び出して認証する場合

JSP から、ServletAuthenticatorを呼び出す場合は、下記のような JSP ファイルを作成します。

太字の部分が必要なコードです。

```
<%@ page import="org.apache.catalina.authenticator.ServletAuthenticator" session="true"
contentType="text/html;charset=Shift_JIS" %>

<html>
<head>
<title>サンプル</title>
</head>

<%
```

```
        String username = "admin";
        String password = "adminadmin";

        ServletAuthenticator sau = new ServletAuthenticator( application );
        sau.authenticate(request, response, username, password);
%>
サンプルです<br>
</body>
</html>
```