BEA WebLogic Server™

BEA WebLogic Express™

WebLogic HTTP サーブレット プログラマーズ ガイド

WebLogic Server バージョン 6.1
マニュアルの日付：2001 年 12 月 19 日

著作権

限定的権利条項

商標または登録商標

WebLogic HTTP サーブレット プログラマーズ ガイド

パート番号	マニュアルの日付	ソフトウェアのバージョン
	2001 年 12 月 19 日	WebLogic Server バージョン 6.1

目次

4. 管理とコンフィグレーション

索引

このマニュアルの内容

このマニュアルでは、WebLogic HTTP サーブレットをプログラミングおよびデプロイする方法について説明します。

このマニュアルの構成は次のとおりです。

- 第 1 章「HTTP サーブレットの概要」では、Hypertext Transfer Protocol（HTTP）サーブレットのプログラミングについて概説し、HTTP サーブレットを WebLogic Server で使用する方法について説明します。
- 第 2 章「プログラミングの概要」では、基本的な HTTP サーブレットのプログラミングについて説明します。
- 第 3 章「プログラミング タスク」では、WebLogic Server 環境での HTTP サーブレットの記述方法について説明します。
- 第 4 章「管理とコンフィグレーション」では、WebLogic Server 環境での HTTP サーブレットの記述方法について説明します。

対象読者

このマニュアルは、HTTP サーブレットと Sun Microsystems の Java 2 Platform, Enterprise Edition（J2EE）を使用して e- コマース アプリケーションを構築するアプリケーション開発者を対象としています。Web テクノロジ、オブジェクト指向プログラミング手法、および Java プログラミング言語に読者が精通していることを前提として書かれています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA の Web サイトで入手できます。BEA のホーム ページで [製品のドキュメント] をクリックします。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用すると、Web ブラウザからこのマニュアルのメイン トピックを一度に 1 つずつ印刷できます。

このマニュアルの PDF 版は、Web サイトで入手できます。PDF を Adobe Acrobat Reader で開くと、マニュアルの全体（または一部分）を書籍の形式で印刷できます。PDF を表示するには、WebLogic Server ドキュメントのホーム ページを開き、[ドキュメントのダウンロード] をクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader は、Adobe の Web サイト（http://www.adobe.co.jp）から無料で入手できます。

関連情報

- javax.servlet パッケージ
  （`http://java.sun.com/products/servlet/2.2/javadoc/javax/servlet/package-summary.html`）
- javax.servlet.http パッケージ
  （`http://java.sun.com/products/servlet/2.2/javadoc/javax/servlet/http/package-summary.html`）
- サーブレット 2.2 仕様
  （`http://java.sun.com/products/servlet/download.html#specs`）

- 『Web アプリケーションのアセンブルとコンフィグレーション』（http://edocs.beasys.co.jp/e-docs/wls61/webapp/index.html）
- 「Web アプリケーションのデプロイメント記述子の記述」（http://edocs.beasys.co.jp/e-docs/wls61/webapp/webappdeployment.html）

サポート情報

BEA のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで docsupport-jp@bea.com までお送りください。寄せられた意見については、ドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、ご使用のソフトウェア名とバージョン名、およびマニュアルのタイトルと作成日付をお書き添えください。本バージョンの BEA WebLogic Server について不明な点がある場合、または BEA WebLogic Server のインストールおよび動作に問題がある場合は、BEA WebSUPPORT（www.bea.com）を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

カスタマ サポートでは以下の情報をお尋ねしますので、お問い合わせの際はあらかじめご用意ください。

- お名前、電子メール アドレス、電話番号、ファクス番号
- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
〔Ctrl〕+ 〔Tab〕	同時に押すキーを示す。
斜体	強調または本のタイトルを示す。

等幅テキスト	コード サンプル、コマンドとそのオプション、Java クラス、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。
	例 :
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

斜体の等幅テキスト	コード内の変数を示す。 例 : `String *CustomerName*;`
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例 : LPT1 BEA_HOME OR
{ }	構文内の複数の選択肢を示す。

[ ]	構文内の任意指定の項目を示す。
	例 :
	`java utils.MulticastTest -n *name* -a *address*`
	`[-p *portnumber*] [-t *timeout*] [-s *send*]`

表記法	適用

|	構文の中で相互に排他的な選択肢を区切る。
	例 :
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`password {application} {source}`

...	コマンドラインで以下のいずれかを示す。 ■ 引数を複数回繰り返すことができる。 ■ 任意指定の引数が省略されている。 ■ パラメータや値などの情報を追加入力できる。
. . .	コード サンプルまたは構文で項目が省略されていることを示す。

1　HTTP サーブレットの概要

以下の節では、Hypertext Transfer Protocol（HTTP）サーブレットのプログラミングについて概説し、HTTP サーブレットを WebLogic Server で使用する方法について説明します。

- サーブレットとは
- サーブレットの特長
- サーブレットのデプロイメントの概要
- サーブレットと J2EE
- HTTP サーブレット API リファレンス

サーブレットとは

サーブレットとは、Java に対応したサーバで実行される Java クラスです。HTTP サーブレットは、HTTP リクエストを処理し、通常は HTML ページの形式で HTTP 応答を送信する特殊なサーブレットです。WebLogic HTTP サーブレットの最も一般的な使い方は、クライアントサイドのプレゼンテーションに標準的な Web ブラウザを使い、WebLogic Server ではサーバサイドのプロセスとしてビジネス ロジックを処理する、対話型アプリケーションを作成することです。WebLogic HTTP サーブレットは、データベース、エンタープライズ JavaBeans、メッセージング API、HTTP セッションなどの WebLogic Server の機能にアクセスできます。

WebLogic Server は、Sun Microsystems の サーブレット 2.3 最終草案バージョン 1 仕様で定義されている HTTP サーブレットを完全にサポートしています。HTTP サーブレット形式は、Java 2 Enterprise Edition（J2EE）規格の不可欠な部分です。

サーブレットの特長

- HTML フォームを使用してエンドユーザの入力を取得し、その入力に応答する HTML ページを表示する、動的な Web ページを作成できます。たとえば、オンライン ショッピング カート、金融サービス、パーソナライズされたコンテンツなどに使用できます。
- オンライン会議などのコラボレーション システムを作成できます。
- WebLogic Server で実行されるサーブレットは、さまざまな API やサービスにアクセスできます。次に例を示します。
  - セッション トラッキング ― Web サイトで複数の Web ページにわたるユーザの動きを追跡できます。この機能は、ショッピング カートを使用する e- コマース サイトなどの Web サイトをサポートします。WebLogic Server はデータベースへのセッション永続性をサポートしており、サーバのダウン タイム中のフェイルオーバ、およびクラスタ化されたサーバ間のセッションの共有を提供します。詳細については、3-12 ページの「サーブレットからのセッション トラッキング」を参照してください。
  - JDBC ドライバ（BEA のドライバを含む）― JDBC ドライバにより、基本的なデータベース アクセスが提供されます。WebLogic Server の多層 JDBC 実装により、接続プール、サーバサイドのデータ キャッシュ、およびトランザクションを利用できます。詳細については、3-26 ページの「データベースへのアクセス」を参照してください。
  - セキュリティ ― 認証用の ALC やセキュアな通信を実現するセキュア ソケット レイヤ（SSL）の使用など、さまざまなタイプのセキュリティをサーブレットに適用できます。
  - エンタープライズ JavaBeans ― サーブレットでエンタープライズ JavaBean（EJB）を使用して、セッション、データベースのデータ、その他の機能をカプセル化できます。
  - Java Messaging Service（JMS）― JMS を使用して、他のサーブレットや Java プログラムとメッセージを交換できます。
  - Java JDK API ― サーブレットでは、標準的な Java JDK API を使用できます。
  - リクエストの転送 ― 他のサーブレットやリソースへリクエストを転送できます。

- J2EE 準拠のサーブレット エンジン用に作成されたサーブレットであれば、WebLogic Server に簡単にデプロイできます。
- サーブレットと JavaServer Pages（JSP）を組み合わせてアプリケーションを作成できます。

サーブレットのデプロイメントの概要

- HTTP サーブレットのプログラマは、JavaSoft の標準 API である `javax.servlet.http` を利用して対話型のアプリケーションを作成します。
- HTTP サーブレットは HTTP ヘッダを読み取り、HTML コードを書き出してブラウザ クライアントへ応答を送り出すことができます。
- サーブレットは、Web アプリケーションの一部として WebLogic Server にデプロイされます。Web アプリケーションとは、サーブレット クラス、JavaServer Pages（JSP）、静的な HTML ページ、画像、セキュリティなどのアプリケーション コンポーネントをグループ化したものです。詳細については、4-1 ページの「管理とコンフィグレーション」を参照してください。

サーブレットと J2EE

Java 2 Platform, Enterprise Edition の一部であるサーブレット 2.2 仕様は、サーブレット API の実装と、エンタープライズ アプリケーションでのサーブレットのデプロイ方法を定義しています。WebLogic Server など J2EE 準拠のサーバでサーブレットをデプロイするには、エンタープライズ アプリケーションを構成するサーブレットなどのリソースを Web アプリケーションという 1 つの単位にパッケージ化します。Web アプリケーションでは、リソースを格納する特定のディレクトリ構造と、これらのリソースが対話する方法や、クライアントによるアプリケーションへのアクセス方法を定義する、デプロイメント記述子を利用します。また、Web アプリケーションは `.war` ファイルと呼ばれるアーカイブ ファイルとしてデプロイすることもできます。

Web アプリケーションの作成の詳細については、『Web アプリケーションのアセンブルとコンフィグレーション』を参照してください。サーブレットの管理およびデプロイメントに関する問題の概要は、4-1 ページの「管理とコンフィグレーション」を参照してください。

注意： サーブレット 2.3 仕様は、J2EE 1.3 仕様の一部です。J2EE 1.3 機能の使用については、1-4 ページの「WebLogic Server 6.1 と J2EE 1.2 および J2EE 1.3」を参照してください。

WebLogic Server 6.1 と J2EE 1.2 および J2EE 1.3

BEA WebLogic Server 6.1 は、高度な J2EE 1.3 の機能を実装する最初の e- コマース トランザクション プラットフォームです。J2EE のルールに準拠するために、BEA Systems では 2 つの別個のダウンロードを用意しています。1 つは J2EE 1.3 の機能が有効になっているもの、1 つは J2EE 1.2 の機能に制限されているものです。いずれのダウンロードもコンテナは内容は同じですが、利用可能な API だけ異なります。

J2EE 1.2 の機能に加えて J2EE 1.3 の機能を備える WebLogic Server 6.1

このダウンロードでは、WebLogic Server はデフォルトで J2EE 1.3 の機能を使用して動作します。それらの機能には、EJB 2.0、JSP 1.2、サーブレット 2.3、および J2EE コネクタ アーキテクチャ 1.0 が含まれます。J2EE 1.3 の機能を有効にして、WebLogic Server 6.1 を実行しても、J2EE 1.2 のアプリケーションはそのままフルサポートされます。WebLogic Server 6.1 は、サーブレット 2.3 の最終草案 1 をサポートしていますが、以下の例外があります。

<run-as> 要素はサポートされていません。

最終草案 1 は、以下の点が最終バージョンと異なっています。

1. WebLogic Server がサポートしている Filter インタフェースには、以下のメソッドが含まれています。

`doFilter(ServletRequest,ServletResponse,FilterChain)`

`getFilterConfig()`: この Filter に対する FilterConfig を返します。

`setFilterConfig(Fil-terConfig)`: Filter をインスタンス化して Filter-Config に渡すとき、コンテナはこのメソッドを呼び出します。

2. 以下のリスナ クラスは名前が変更されています。影響を受けるリスナは以下のとおりです。

旧 : `javax.servlet.ServletContextAttributesListener;`

新 : `javax.servlet.ServletContextAttributeListener;`

旧 : `javax.servlet.http.HttpSessionAttributesListener;`

新 : `javax.servlet.http.HttpSessionAttributeListener;`

3. `HttpSession` に `getServletContext()` メソッドが追加されています。

4. `ServletContext.getResourcePaths()` メソッドが、ディレクトリ引数に追加されています。
5. ステータス コード 307 (一時的なリダイレクト) が、`HttpServletResponse` に追加されています。

J2EE 1.2 認定の WebLogic Server 6.1

このダウンロードでは、WebLogic Server はデフォルトで J2EE 1.3 機能が無効な状態で動作し、J2EE 1.2 の仕様と規定に完全に準拠します。

HTTP サーブレット API リファレンス

WebLogic Server は、Java サーブレット 2.2 API の `javax.servlet.http` パッケージをサポートしています。このパッケージについては、Sun Microsystems から、さらに次のドキュメントが提供されています。

- API ドキュメント
  - javax.servlet パッケージ
  - javax.servlet.http パッケージ
- サーブレット 2.2 仕様

2　プログラミングの概要

以下の節では、基本的な HTTP サーブレットのプログラミングについて説明します。

- 単純な HTTP サーブレットの記述
- 高度な機能
- HelloWorldServlet サンプルの全文

単純な HTTP サーブレットの記述

この節では、`Hello World` というメッセージを出力する単純な HTTP サーブレットを記述する手順について説明します。これらの手順を示すサンプル コード（`HelloWorldServlet`）の全文がこの節の最後にあります。JDBC、RMI、JMS など各種の J2EE および WebLogic Server サービスをサーブレットで使用する方法の詳細については、このマニュアルで後述します。

1. 以下の適切なパッケージおよびクラスをインポートします。

   ```
   import javax.servlet.*;
   import javax.servlet.http.*;
   import java.io.*;
   ```

2. `javax.servlet.http.HttpServlet` を拡張します。次に例を示します。

   ```
   public class HelloWorldServlet extends HttpServlet{
   ```

3. `service()` メソッドを実装します。

   サーブレットの主要な機能は、Web ブラウザからの HTTP リクエストを受け取って、HTTP 応答を返すことです。この処理は、サーブレットの `service()` メソッドによって行われます。サービス メソッドには、出力を生成する応答オブジェクトと、クライアントからのデータを受け取るリクエスト オブジェクトがあります。

これ以外にも、doPost() メソッドや doGet() メソッドを実装したサーブレットのサンプルを見たことがあるかもしれません。これらのメソッドは、POST または GET リクエストにのみ応答するものです。service() メソッドを実装しておけば、1 つのメソッドですべてのリクエスト タイプを処理できます（ただし、service() メソッドを実装した場合、このメソッドの最初で super.service() を呼び出さない限り、doPost() メソッドや doGet() メソッドを実装することはできません）。HTTP サーブレットの仕様ではこれ以外に、他のリクエスト タイプの処理に使用されるメソッドも解説していますが、全部をまとめて、サービス メソッドと総称しています。

サービス メソッドはすべて、同じパラメータ引数を取ります。HttpServletRequest は、リクエストの情報を提供し、HttpServletResponse は HTTP クライアントに応答する際にサーブレットによって使用されます。サービス メソッドは次のようになります。

```
public void service(HttpServletRequest req,
       HttpServletResponse res) throws IOException
{
```

4. 次のように、コンテンツ タイプを設定します。

   ```
   res.setContentType("text/html");
   ```

5. 次のように、出力に使用する java.io.PrintWriter オブジェクトへの参照を取得します。

   ```
   PrintWriter out = res.getWriter();
   ```

6. 次の例に示すように、PrintWriter オブジェクトに対し println() メソッドを使用して、HTML を生成します。

   ```
   out.println("<html><head><title>Hello World!</title></head>");
   out.println("<body><h1>Hello World!</h1></body></html>");
     }
   }
   ```

7. サーブレットを次のようにコンパイルします。

   a. 開発環境シェルの設定を、クラスパスとパスを正しく指定して行います。

   b. サーブレットの Java ソース コードが格納されているディレクトリから、サーブレットが格納されている Web アプリケーションの WEB-INF\classes ディレクトリに、サーブレットをコンパイルします。次に例を示します。

      ```
      javac -d /myWebApplication/WEB-INF/classes myServlet.java
      ```

8. WebLogic Server にホストが配置される Web アプリケーションの一部として、サーブレットをデプロイします。サーブレットのデプロイメントの概要については、4-1 ページの「管理とコンフィグレーション」を参照してください。
9. ブラウザからサーブレットを呼び出します。

   サーブレットの呼び出しに使用する URL は、(a) そのサーブレットが含まれる Web アプリケーション名、(b) Web アプリケーションのデプロイメント記述子にマップされるサーブレット名によって決まります。リクエスト パラメータも、サーブレットを呼び出す URL に含まれることがあります。

   一般的には、サーブレットの URL は以下のパターンに従います。

   `http://host:port/webApplicationName/mappedServletName?parameter`

   URL の各要素は次のように定義します。

   - *host* は、WebLogic Server が稼動しているマシンの名前。
   - *port* は、上記マシンが HTTP リクエストをリスンしているポート。
   - *webApplicationName* は、サーブレットが含まれる Web アプリケーションの名前。
   - parameters は、サーブレットで使用できるブラウザから送信された情報が含まれる、1 つまたは複数の名前と値の組み合わせ。

   たとえば、`examplesWebApp` にデプロイされ、自分のマシンで稼動している WebLogic Server から提供される `HelloWorldServlet` （このマニュアルで使われているサンプル）を Web ブラウザで呼び出す場合、次のような URL を入力します。

   `http://localhost:7001/examplesWebApp/HelloWorldServlet`

   URL の *host*:*port* 部分は、WebLogic Sever にマップされている DNS 名に置き換えることもできます。

高度な機能

上記の手順で、基本的なサーブレットが作成できます。サーブレットでは、さらに高度な機能を使用することもできます。

- HTML 形式のデータ処理 — HTTP サーブレットでは、ブラウザ クライアントからの HTML フォームによるデータを受け取り、処理できます。
  - 3-6 ページの「クライアント入力の取得」
- アプリケーションの設計 — HTTP サーブレットでは、さまざまな方法でアプリケーションを設計できます。サーブレットの記述に関する詳細は、以下の節を参照してください。
  - 3-4 ページの「HTTP 応答の提供」
  - 3-30 ページの「HTTP サーブレットにおけるスレッドの問題」
  - 3-31 ページの「別のリソースへのリクエストのディスパッチ」
- サーブレット初期化 — サーブレット初期化時に、データを初期化する、初期化引数を受け取るなどのアクションを実行する必要がある場合は、`init()` メソッドをオーバーライドできます。
  - 3-1 ページの「サーブレットの初期化」
- サーブレットにおけるセッションおよび永続性の使用 — セッションと永続性により、HTTP セッション中、および HTTP セッション間でユーザを追跡できます。セッション管理には、クッキーの使用も含まれます。詳細については、以下の節を参照してください。
  - 3-12 ページの「サーブレットからのセッション トラッキング」
  - 3-22 ページの「サーブレットでのクッキーの使い方」
  - 3-22 ページの「セッションの永続性のコンフィグレーション」
- サーブレットにおける WebLogic サービスの使用 — WebLogic Server が提供するさまざまなサービスや API を Web アプリケーションで使用できます。サービスには、Java Database Connectivity（JDBC）ドライバ、JDBC データベース接続プール、Java Messaging Service（JMS）、Enterprise JavaBeans（EJB）、Remote Method Invocation（RMI）などがあります。詳細については、以下の節を参照してください。
  - 3-26 ページの「HTTP サーブレットからの WebLogic サービスの使い方」

- 4-6 ページの「サーブレットのセキュリティ」
- 3-26 ページの「データベースへのアクセス」

HelloWorldServlet サンプルの全文

この節では、前述の手順で使用した Java ソース コード サンプルの全文を示します。このサンプルは HTTP リクエストに応答する簡単なサーブレットです。このマニュアルの後半では、このサンプルを拡張することにより、HTTP パラメータ、クッキー、およびセッション トラッキングの使い方を説明します。

コード リスト 2-1　HelloWorldServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HelloWorldServlet extends HttpServlet {
  public void service(HttpServletRequest req,
                      HttpServletResponse res)
      throws IOException
  {
    // 先にコンテンツ タイプを設定する
    res.setContentType("text/html");
    // ここで HTML を挿入する PrintWriter を取得する
    PrintWriter out = res.getWriter();

    out.println("<html><head><title>" +
                "Hello World!</title></head>");
    out.println("<body><h1>Hello World!</h1></body></html>");
  }
}
```

WebLogic Server 配布キットの samples\examples\servlets ディレクトリに、このソース コードと、このマニュアルで使われている全サンプルをコンパイルして実行するための手順説明があります。

3 プログラミング タスク

以下の節では、WebLogic Server 環境で HTTP サーブレットを記述する方法について説明します。

- サーブレットの初期化
- HTTP 応答の提供
- クライアント入力の取得
- サーブレットからのセッション トラッキング
- サーブレットでのクッキーの使い方
- HTTP サーブレットからの WebLogic サービスの使い方
- データベースへのアクセス
- HTTP サーブレットにおけるスレッドの問題
- 別のリソースへのリクエストのディスパッチ

サーブレットの初期化

通常、WebLogic Server によってサーブレットが初期化されるのは、そのサーブレットに対して最初のリクエストが出されたときです。その後、そのサーブレットが変更されると、既存のバージョンのサーブレットに対して `destroy()` メソッドが呼び出されます。変更後のサーブレットにリクエストが送信されると、変更後のサーブレットの `init()` メソッドが実行されます。詳細については、4-7 ページの「サーブレット開発のヒント」を参照してください。

サーブレットが初期化されるとき、WebLogic Server はサーブレットの `init()` メソッドを実行します。サーブレットは一度初期化されると、WebLogic Server を再起動するまで、またはサーブレットが変更された場合はサーブレットが呼び出されるまで、再び初期化されることはありません。`init()` メソッドをオー

バーライドすると、データベース接続の確立など、サーブレットの初期化時に特定のタスクを実行させることができます（3-3 ページの「init() メソッドのオーバーライド」を参照）。

WebLogic Server 起動時のサーブレットの初期化

サーブレットに最初のリクエストが送信された時に WebLogic Server がサーブレットを初期化するのではなく、サーバの起動時に初期化するように、最初に WebLogic Server をコンフィグレーションできます。そのためには、Web アプリケーション デプロイメント記述子の `<load-on-startup>` 要素内にサーブレット クラスを指定します。詳細については、「servlet 要素」を参照してください。

初期化中に HTTP サーブレットにパラメータを渡すには、サーブレットを含む Web アプリケーションでパラメータを定義します。このパラメータを使用すると、サーブレットが初期化されるたびに書き換えることなく値を渡せます。詳細については、「Web アプリケーションのデプロイメント記述子の記述」を参照してください。

たとえば、Web アプリケーション デプロイメント記述子の以下のようなエントリの場合、次の 2 つの初期化パラメータが定義されます。1 つは `Welcome` の値を持つ `greeting`、もう 1 つは `WebLogic Developer` の値を持つ `person` です。

```
<servlet>
  ...
  <init-param>
    <param-name>greeting</param-name>
    <param-value>Welcome</param-value>
    <description>The salutation</description>
  </init-param>
  <init-param>
    <param-name>person</param-name>
    <param-value>WebLogic Developer</param-value>
    <description>name</description>
  </init-param>
</servlet>
```

初期化パラメータを取得するには、親 `javax.servlet.GenericServlet` クラスから `getInitParameter(String name)` メソッドを呼び出します。パラメータ名を渡されると、このメソッドはパラメータの値を文字列で返します。

init() メソッドのオーバーライド

init() メソッドをオーバーライドすることで、初期化時にサーブレットにタスクを実行させることができます。次のコードでは、Web アプリケーション デプロイメント記述子の中で挨拶文と名前を定義する <init-param> タグを読み込んでいます。

```
String defaultGreeting;
String defaultName;

public void init(ServletConfig config)
    throws ServletException {
  if ((defaultGreeting = getInitParameter("greeting")) == null)
    defaultGreeting = "Hello";

  if ((defaultName = getInitParameter("person")) == null)
    defaultName = "World";
}
```

各パラメータの値は、クラス インスタンス変数 defaultGreeting および defaultName に格納されます。最初にパラメータが null 値を持つかどうかをテストして、null 値が返されれば、適切なデフォルト値を提供します。

これで、service() メソッドを使用して、応答の中にこれらの変数を含めることができます。次に例を示します。

```
  out.print("<body><h1>");
  out.println(defaultGreeting + " " + defaultName + "!");
  out.println("</h1></body></html>");
```

WebLogic Server 配布キットの samples\examples\servlets ディレクトリに、完全なソース コードと、HelloWorld2.java というサンプルをコンパイル、インストール、および試行するための手順説明があります。その中で、init() メソッドの使い方を解説します。

サーブレットの init() メソッド では、WebLogic Server がサーブレットをロードする際に必要なあらゆる初期化処理を行います。デフォルトの init() メソッドでは、WebLogic Server に必要な初期処理がすべて行われるので、初期化の際に特別に行わなければならない処理がないかぎり、デフォルトのメソッドをオーバーライドする必要はありません。init() をオーバーライドする場合は、まず super.init() を呼び出して、デフォルトの初期化アクションが先に実行されるようにします。

HTTP 応答の提供

この節では、HTTP サーブレットでクライアントへの応答を提供する方法について説明します。応答はすべて、サーブレットの service() メソッドにパラメータとして渡される HttpServletResponse オブジェクトを使用して渡さなければなりません。

1. HttpServletResponse をコンフィグレーションします。

   HttpServletResponse オブジェクトを使用して、HTTP ヘッダ情報に変換される複数のサーブレット プロパティを設定できます。

   - 少なくとも、ページのコンテンツを書きこむ出力ストリームを取得する前に、setContentType() メソッドを使用してコンテンツ タイプを設定します。HTML ページの場合、コンテンツ タイプは text/html に設定します。次に例を示します。

     ```
     res.setContentType("text/html");
     ```

   - （省略可能）setContentType() メソッドを使用して文字エンコードを設定することもできます。次に例を示します。

     ```
     res.setContentType("text/html;ISO-88859-4");
     ```

   - setHeader() メソッドを使用して、ヘッダ属性を設定します。動的な応答の場合は、「pragma」属性を no-cache に設定するとよいでしょう。こうするとブラウザは常にページを再ロードし、確実に最新データを表示します。次に例を示します。

     ```
     res.setHeader("Pragma", "no-cache");
     ```

2. HTML ページを作成します。

   サーブレットがクライアントに送り返す応答は、通常の HTTP コンテンツ (基本的には HTML 形式) のように見える必要があります。サーブレットは、service() メソッドの応答パラメータを使用して取得した出力ストリームを通じて、HTTP 応答を送り返します。HTTP 応答を送信するには、次の手順に従います。

   a. HttpServletResponse オブジェクトおよび次の例のいずれかのメソッドを使用して、出力ストリームを取得します。
      - PrintWriter out = res.getWriter();
      - ServletOutputStream out = res.getOutputStream();

同じサーブレット内 (または、サーブレットの中にインクルードされた別のサーブレット内) で、PrintWriter と ServletOutputStream を両方とも使用できます。どちらの出力も、同じバッファに書き込まれます。

b. print() メソッドを使用して、応答の内容を出力ストリームに書き出します。これらの文では、HTML タグを使うことができます。次に例を示します。

```
out.print(“<html><head><title>My Servlet</title>”);
out.print(“</head><body><h1>”);
out.print("Welcome");
out.print(“</h1></body></html>”);
```

ユーザが入力したデータを出力するときは常に、入力されている HTML の特殊文字をすべて削除することをお勧めします。特殊文字を削除しないと、Web サイトがクロスサイト スクリプティングに利用される危険があります。8 ページの「サーブレットでのクライアント入力のセキュリティ対策」を参照してください。

出力ストリームは close() メソッドでクローズしてはいけません。また、ストリームのコンテンツをフラッシュすることも避けてください。出力ストリームをクローズしたりフラッシュしたりしないことによって、WebLogic Server は次の手順で説明する永続的 HTTP 接続の利点を活かすことができます。

3. 応答を最適化します。

デフォルトでは、WebLogic Server は、可能な限り HTTP の永続的接続を使用しようとします。永続的接続は、クライアントとサーバ間の一連の通信のために、同一の HTTP TCP/IP 接続を再利用しようとします。各リクエストごとに新しい接続をオープンする必要がないので、アプリケーションの性能を高めることができます。永続的接続は、HTML ページにインライン画像が多く含まれる場合に便利です。接続を再利用しないと、画像が要求されるごとに新しい TCP/IP 接続が必要になるためです。

WebLogic Server Administration Console を使用すると、WebLogic Server が HTTP 接続をオープンに保つ時間をコンフィグレーションできます。詳細については、[Keep Alive 時間] を参照してください。

永続的接続を確立するために、WebLogic Server は HTTP 応答の長さを知る必要があるので、HTTP 応答ヘッダに Content-Length プロパティを自動的に追加します。コンテンツ長を確認するために、WebLogic Server では応答をバッファリングする必要があります。ただし、サーブレットが ServletOutputStream を明示的にフラッシュすると、WebLogic Server は応

答の長さを判別できないため、永続的接続を使用できません。このため、サーブレットで HTTP 応答を明示的にフラッシュすることは避けます。

場合によっては、ページ完成前にクライアントに情報を表示するために応答を早くフラッシュした方がよいこともあります。たとえば、時間のかかるページのコンテンツを計算している途中で、バナー広告を表示する場合などです。逆に、サーブレット エンジンが使用するバッファ サイズを増やして、フラッシュする前に、もっと長い応答を入れておきたいという場合もあります。javax.servlet.ServletResponse インタフェースの関連メソッドを用いて、応答バッファのサイズを操作することができます。

WebLogic Server のサーブレット エンジンは、チャンクに分割されたデータを自動的にマージするようになりました。コンテンツの長さは引き続き設定する必要があり、Transfer-encoding パラメータは chunked のままですが、サーブレットまたは JSP のコードでチャンクに分かれたデータをデコードする必要はありません。この操作は、サーブレット エンジンが代わりに行うようになっています。

クライアント入力の取得

HTTP サーブレット API は、Web ページからユーザ入力を取得するためのインタフェースを提供しています。

Web ブラウザからの HTTP リクエストには、クライアント、ブラウザ、クッキー、およびユーザのクエリ パラメータに関する情報といった、URL 以外の情報も含めることができます。ブラウザからのユーザ入力を渡すには、クエリ パラメータを使用します。`GET` メソッドは URL アドレスにパラメータを付加し、`POST` メソッドはそれらを HTTP リクエスト本文の中に含めます。

HTTP サーブレットは、これらの細部を扱う必要はありません。リクエストの中の情報は、送った方法に関係なく、`HttpServletRequest` オブジェクトを通じて取得され、`request.getParameters()` メソッドを使用してアクセスできるようになります。

クエリ パラメータをクライアントから送る方法については、以下を参照してください。

- ページのリンクの URL に、パラメータを直接エンコードします。この方法では、`GET` メソッドを使用してパラメータを送ります。パラメータは URL

の後ろに ? を付けてその後に追加します。パラメータが複数ある場合は & で区切ります。パラメータは、常に名前 = 値 の組で指定されます。指定される順序は重要ではありません。たとえば、次のようなリンクを Web ページに入れる場合、ここでは ColorServlet と呼ばれる HTTP サーブレットに、パラメータ color の値を purple にして送ります。

```
<a href=
   "http://localhost:7001/myWebApp/ColorServlet?color=purple">
   Click Here For Purple!</a>
```

- クエリ パラメータを付けた URL をブラウザの場所フィールドに手動で入力します。これは前の例で示したリンクをクリックするのと同じことです。
- HTML フォームで、ユーザに入力を要求します。このフォームの送信ボタンがクリックされると、フォーム上の各ユーザ入力フィールドの内容が、クエリ パラメータとして送られます。このフォームがクエリ パラメータを送るために使うメソッド（POST または GET）を、METHOD="GET|POST" 属性を使用して <FORM> タグに指定します。

クエリ パラメータは常に名前 = 値の組で送られ、HttpServletRequest オブジェクトを通じてアクセスされます。クエリのすべてのパラメータ名の Enumeration を取得し、そのパラメータ名を使用して各パラメータの値を取得できます。1 つのパラメータの値は通常 1 つだけですが、値の配列を持つこともできます。パラメータの値は常に String として読み取られるので、より適切なタイプにキャストすることが必要な場合もあります。

service() メソッドからの次のサンプルでは、フォームから得たクエリ パラメータ名とその値を調べます。request は HttpServletRequest オブジェクトであることに注意してください。

```
Enumeration params = request.getParameterNames();
String paramName = null;
String[] paramValues = null;

while (params.hasMoreElements()) {
  paramName = (String) params.nextElement();
  paramValues = request.getParameterValues(paramName);
  System.out.println("\nParameter name is " + paramName);
  for (int i = 0; i < paramValues.length; i++) {
    System.out.println(", value " + i + " is " +
                       paramValues[i].toString());
  }
}
```

サーブレットでのクライアント入力のセキュリティ対策

このようにユーザが入力したデータを取得して返す機能があると、**クロスサイト スクリプティング**と呼ばれるセキュリティ上の弱点が発生し、ユーザのセキュリティ認可を盗むために利用される危険があります。クロスサイト スクリプティングの詳細については、http://www.cert.org/tech_tips/malicious_code_mitigation.html の「Understanding Malicious Content Mitigation for Web Developers」(CERT によるセキュリティ勧告) を参照してください。

このようなセキュリティ上の弱点を排除するには、ユーザが入力したデータを返す前に、表 3-1 に示したような HTML の特殊文字がデータに含まれるかどうかを調べます。特殊文字がある場合は、それを HTML の実体参照または文字参照に置き換えます。文字を置き換えることで、ユーザが入力したデータをブラウザが HTML として実行することを防止できます。

表 3-1 置き換える必要がある HTML の特殊文字

置き換える必要のある特殊文字	置き換える実体 / 文字参照
<	<
>	>
(	&40;
)	&41;
#	&35;
&	&38;

WebLogic Server ユーティリティ メソッドの使用

WebLogic Server には、ユーザ入力データに含まれる特殊文字を置換するための `weblogic.servlet.security.Utils.encodeXSS()` メソッドが用意されています。このメソッドを使用するには、入力としてユーザ入力データを渡します。次に示すのは、コード リスト 3-1 のユーザ入力データを安全なものにする例です。

```
out.print(defaultGreeting + " " + name + "!");
```

上のような行は、次のように変更します。

```
out.print(defaultGreeting + " " +
weblogic.security.servlet.encodeXSS(name) + "!");
```

アプリケーション全体を保護するには、ユーザ入力データを返す**すべての箇所で** `encodeXSS()` メソッドを使用する必要があります。先に示したのは `encodeXSS()` メソッドを使用しなければならないことが明白な場所の例ですが、他にも 表 3-2 に挙げたような場所において `encodeXSS()` メソッドの使用を検討する必要があります。

表 3-2 ユーザ入力データを返すコード

ページのタイプ	ユーザ入力データ	例
エラー ページ	誤った入力文字列、無効な URL やユーザ名	「*ユーザ名*がアクセスを許可されていない」ことを示すエラー ページ
ステータス ページ	ユーザ名、それ以前のページにおける入力の要約	それ以前のページでの入力の確認を求める要約ページ
データベース表示	データベースからの表示データ	それ以前にユーザが入力しているデータベース エントリのリストを表示するページ

HTTP リクエストを使用するメソッド

この節では、リクエスト オブジェクトからデータを取得できる `javax.servlet.HttpServletRequest` インタフェースのメソッドを定義します。次の制限事項に注意してください。

- この節の `getParameter` メソッドを使用してリクエスト パラメータを読み込んだ後に、`getInputStream()` メソッドをでリクエストを読もうとすることはできません。
- `getInputStream()` でリクエストを読み込んだ後に、`getParameter()` メソッドの 1 つを使ってリクエスト パラメータを読み込もうとすることもできません。

上のいずれかの手順を実行しようとすると、`illegalStateException` が送出されます。

javax.servlet.HttpServeletRequest の次のメソッドを使用して、リクエスト オブジェクトからデータを取得できます。

HttpServletRequest.getMethod()

GET や POST などのリクエスト メソッドを決定できます。

HttpServletRequest.getQueryString()
クエリ文字列にアクセスできます（クエリ文字列は、要求された URL で ? の後に続く部分です）。

HttpServletRequest.getParameter()
パラメータの値を返します。

HttpServletRequest.getParameterNames()
パラメータ名の配列を返します。

HttpServletRequest.getParameterValues()
パラメータの値の配列を返します。

HttpServletRequest.getInputStream()
リクエスト本体をバイナリ データとして読み込みます。リクエスト パラメータを getParameter()、getParameterNames()、または getParameterValues() で読み込んだ後にこのメソッドを呼び出すと、illegalStateException が送出されます。

例 : クエリ パラメータによる入力の取得

この例では、より個人的な挨拶文を表示するために、クエリ パラメータとしてユーザ名を受け付けるように HelloWorld2.java サーブレットのサンプルを修正しています。（コードの全文については、WebLogic Server 配布キットの samples\examples\servlets ディレクトリにある HelloWorld3.java サンプルを参照してください）。service() メソッドを次に示します。

コード リスト 3-1 service() メソッドによる入力の取得

```
public void service(HttpServletRequest req,
         HttpServletResponse res)
     throws IOException
{
  String name, paramName[];
```

```
  if ((paramName = req.getParameterValues("name"))
      != null) {
    name = paramName[0];
  }
    else {
    name = defaultName;
  }

    // まず、コンテンツ タイプを設定する
  res.setContentType("text/html");
    // PrintWriter を出力ストリームとして取得する
  PrintWriter out = res.getWriter();

  out.print("<html><head><title>" +
              "Hello World!" + </title></head>");
  out.print("<body><h1>");
  out.print(defaultGreeting + " " + name + "!");
  out.print("</h1></body></html>");
}
```

getParameterValues() メソッドは、HTTP クエリ パラメータから name パラメータの値を取得します。これらの値は、String タイプの配列で取得します。このパラメータに 1 つの値が返され、name 配列の第 1 要素に割り当てられます。クエリ データにこのパラメータがなければ、戻り値は null になります。この場合は、init() メソッドによって <init-param> プロパティから読み込んだデフォルトの名前に name を割り当てます。

パラメータが HTTP リクエストに含まれていると仮定してサーブレット コードを記述しないでください。getParameter() メソッドは非推奨となったため、配列の添え字にその末尾までタグ付けすることによって、getParameterValues() メソッドを略記しようとする場合があります。しかし、このメソッドは指定したパラメータが有効でないと null を返すことがあり、NullPointerException が発生します。

たとえば、次のコードでは NullPointerException が発生します。

```
String myStr = req.getParameterValues("paramName")[0];
```

代わりに、次のコードを使用します。

```
if ((String myStr[] =
          req.getParameterValues("paramName"))!=null) {
  // これで myStr[0] を使うことができる
}
    else {
  // paramName がクエリ パラメータの中になかった！
}
```

サーブレットからのセッション トラッキング

セッション トラッキングを使用すると、複数のサーブレットか HTML ページにわたって、本来はステートレスであるユーザの状況を追跡できます。「セッション」の定義は、ある一定期間中に同じクライアントから出される一連の関連性のあるブラウザ リクエストです。セッション トラッキングは、ショッピング カート アプリケーションのように、全体として何らかの意味を持つ一連のブラウザ リクエスト（これをページとみなす）を結合します。

以下の節では、HTTP サーブレットからのセッション トラッキングについて、さまざまな観点から説明します。

- セッション トラッキングの履歴
- HttpSession オブジェクトを用いたセッションのトラッキング
- セッションの有効期間
- セッション トラッキングの仕組み
- セッションの開始の検出
- セッション名 / 値の属性の設定と取得
- セッションのログアウトと終了
- セッション トラッキングのコンフィグレーション
- クッキーに代わる URL 書き換えの使用
- URL 書き換えと Wireless Access Protocol (WAP)
- セッションの永続化

セッション トラッキングの履歴

セッション トラッキングという概念が発展する前は、ページの非表示フィールドに情報を詰め込むか、長い文字列をリンクで使われる URL に追加してユーザの選択内容を埋め込むことで、ページに状態を組み込んでいました。このよい例

が、サーチ エンジン サイトです。サーチ エンジン サイトの多くはいまだに CGI に依存しています。これらのサイトは、URL の後に HTTP で予約されている ? 記号を付け、その後に続く URL パラメータの名前 = 値の組を使用して、ユーザの選択を追跡します。このようにすると、URL は非常に長いものになる場合があり、CGI スクリプトはこれを慎重に解析し、管理する必要があります。この手法の問題点は、情報をセッションからセッションへと渡せないことです。URL の制御を失うと、つまりユーザがページから離れると、ユーザ情報は永久に失われます。

その後、Netscape はブラウザのクッキーを発表しました。これを使用してサーバごとにクライアント上のユーザ関連情報を格納することができます。しかし、ブラウザによってはまだクッキーを完全にサポートしていないものもあり、またブラウザのクッキー オプションをオフにしておくことを選ぶユーザもいます。もう 1 つの考慮すべき要因は、ほとんどのブラウザではクッキーで格納できるデータ量を制限しているということです。

CGI の手法と異なり、HTTP サーブレットの仕様では、サーバが単一のセッションを超えるサーバ上のユーザの詳細情報を格納することを可能にし、コードがセッションのトラッキングにより複雑化することを防ぐ方法が定義されています。サーブレットは、`HttpSession` オブジェクトを使用して、単一のセッション期間中のユーザ入力を追跡し、セッションの詳細を複数サーブレットで共有することができます。セッション データは、WebLogic サービスで使用できるさまざまなメソッドにより保持できます。

HttpSession オブジェクトを用いたセッションのトラッキング

WebLogic が実装してサポートしている Java Servlet API では、各サーブレットは `HttpSession` オブジェクトを使用して、サーバサイド セッションにアクセスすることができます。`HttpServletRequest` オブジェクトを使用して、サーブレットの `service()` メソッド内の `HttpSession` オブジェクトにアクセスできます。次の例のように変数 `request` を使用します。

```
HttpSession session = request.getSession(true);
```

引数 `true` で `request.getSession(true)` メソッドが呼び出されると、そのクライアントに対して既存の `HttpSession` オブジェクトがない場合には、HttpSession オブジェクトが生成されます。セッション オブジェクトは、その

セッションの有効期間の間 WebLogic Server に存在し、そのクライアントに関連した情報を蓄積します。サーブレットは、必要に応じて、セッション オブジェクトで情報の追加や削除を行います。セッションは特定のクライアントに関連付けられます。クライアントがサーブレットを訪れるごとに、`getSession()` メソッドが呼び出されたときと同一の、関連付けられた `HttpSession` オブジェクトが取得されます。

`HttpSession` でサポートされるメソッドの詳細については、「HttpServlet API」を参照してください。

次の例では、`service()` メソッドは、セッション中にユーザがサーブレットを要求する回数を数えます。

```
public void service(HttpServletRequest request,
                    HttpServletResponse, response)
            throws IOException
{
  // セッションとカウンタ パラメータ属性を取得する
  HttpSession session = request.getSession (true);
  Integer ival = (Integer)
                 session.getAttribute("simplesession.counter");
  if (ival == null) // カウンタを初期化する
    ival = new Integer (1);
  else // カウンタをインクリメントする
    ival = new Integer (ival.intValue () + 1);
  // セッションに新しい属性値を設定する
  session.setAttribute("simplesession.counter", ival);
  // HTML ページを出力する
  out.print("<HTML><body>");
  out.print("<center> You have hit this page ");
  out.print(ival + " times!");
  out.print("</body></html>");
}
```

セッションの有効期間

セッションは、1 つのトランザクションの一連のページにわたるユーザの選択を追跡します。1 つのトランザクションは、ある品物を閲覧し、それをショッピング カートに追加した後、支払手続きをするといった複数のタスクで構成されます。セッションは永続的なものではなく、以下のいずれかの時点で、その有効期間は終了します。

- ユーザがサイトを離れ、ブラウザがクッキーを受け入れない場合。
- ユーザがブラウザを終了した場合。

- 動作がないため、セッションがタイム アウトになった場合。
- セッションが完了し、サーブレットによって無効にされた場合。
- ユーザがログアウトし、サーブレットによって無効にされた場合。

より永続的な長い時間データを保存するには、サーブレットは、JDBC または EJB を使用してデータベースに詳細を書き込み、存続期間の長いクッキーやユーザ名とパスワードによって、クライアントとこのデータを関連付ける必要があります。このマニュアルでは、セッションが内部的にクッキーや永続性を使用すると説明していますが、ユーザに関するデータの格納のための一般的なメカニズムとしてセッションを使用してはなりません。

セッション トラッキングの仕組み

WebLogic Server は、各クライアントにどのセッションが関連付けられているのかを、どのようにして認識するのでしょうか。`HttpSession` がサーブレットで生成されると、ユニークな ID と関連付けられます。ブラウザは、このセッション ID をそのリクエストに付与し、サーバが再びセッション データを見つけられるようにしなければなりません。サーバは、クライアントにクッキーを設定することによって、この ID を保存しようとします。一度クッキーが設定されると、ブラウザがサーバにリクエストを送るたびに、リクエストにはその ID を内包したクッキーが含まれます。サーバは自動的にクッキーを解析し、サーブレットが `getSession()` メソッドを呼び出すと、セッション データを提供します。

クライアントがクッキーを受け入れない場合、代わりの方法としては、クライアントへ送り返されるページのなかで、URL リンクにその ID をエンコードするしかありません。このため、サーブレットの応答に URL を入れる場合は、必ず `encodeURL()` メソッドを使用します。WebLogic Server はブラウザがクッキーを受け入れるかどうかを検出して、不必要な URL のエンコードは行いません。自動的に、エンコードされた URL からセッション ID を解析し、`getSession()` メソッドを呼び出すと、正しいセッション データを取得します。`encodeURL()` メソッドを使用すると、セッション トラッキングにどの手順を使用しても、サーブレットのコードが破壊されることはありません。詳細については、3-19 ページの「クッキーに代わる URL 書き換えの使用」を参照してください。

セッションの開始の検出

getSession(true) メソッドを使用してセッションを取得した後、HttpSession.isNew() メソッドを呼び出すことにより、そのセッションが生成されたばかりかどうかがわかります。このメソッドが true を返した場合、クライアントはすでに有効なセッションを持っておらず、またこの時点では新規のセッションを認識しません。クライアントが新規のセッションを認識するのは、サーバから応答がポストされて戻ってからです。

ビジネス ロジックに合った方法で、新規または既存のセッションに適応するようにアプリケーションを設計してください。たとえば、セッションがまだ開始していないと判断すれば、次のサンプル コードのように、アプリケーションでクライアントの URL をログイン / パスワードのページにリダイレクトしてもよいでしょう。

```
HttpSession session = request.getSession(true);
if (session.isNew()) {
  response.sendRedirect(welcomeURL);
}
```

ログインのページでは、システムにログインするか、新規アカウントを作成するかを選択できるようにします。また、Web アプリケーションでログイン ページを指定することもできます。詳細については、「login-config 要素」を参照してください。

セッション名 / 値の属性の設定と取得

名前 = 値 の組み合わせを使用して、HttpSession オブジェクトにデータを格納できます。セッションに格納されたデータは、そのセッションを通じて利用することができます。セッションにデータを格納するには、次のメソッドを HttpSession インタフェースから使用します。

```
getAttribute()
getAttributeNames()
setAttribute()
removeAttribute()
```

次のコードでは、既存の名前 = 値の組み合わせをすべて取得する方法を示します。

```
Enumeration sessionNames = session.getAttributeNames();
String sessionName = null;
```

```
Object sessionValue = null;

while (sessionNames.hasMoreElements()) {
  sessionName = (String)sessionNames.nextElement();
  sessionValue = session.getAttribute(sessionName);
  System.out.println("Session name is " + sessionName +
                     ", value is " + sessionValue);
}
```

名前の付いた属性を追加または上書きするには、`setAttribute()` メソッドを使用します。名前の付いた属性を完全に削除するには、`removeAttribute()` メソッドを使用します。

注意： Java の `Object` の子孫をセッション属性として追加し、それに名前を関連付けることができます。しかしながら、セッション永続性を利用している場合、属性 *`value`* のオブジェクトは `java.io.Serializable` を実装しなければなりません。

セッションのログアウトと終了

アプリケーションがデリケートな情報を扱う場合、セッションからログアウトする機能の提供を検討することがあります。これは、ショッピング カートやインターネットの電子メール アカウントを使用する際には一般的な機能です。同一のブラウザがサービスに戻るとき、ユーザはシステムにログインし直さなければなりません。

単一の Web アプリケーションに対する session.invalidate() の使用

ユーザ認証の情報は、ユーザのセッション データ、およびサーバのコンテキストまたは Web アプリケーションにより割り当てられた仮想ホストのコンテキストの両方に格納されます。ユーザのログアウトに多く使われる `session.invalidate()` メソッドを使用すると、ユーザの現在のセッションのみが無効になり、ユーザの認証情報は有効なまま、サーバまたは仮想ホストのコンテキストに格納されます。サーバまたは仮想ホストが 1 つの Web アプリケーションのみのホストである場合、実際には `session.invalidate()` メソッドでユーザをログアウトします。

session.invalidate() を呼び出した後、無効にされたセッションを参照しないでください。参照した場合、IllegalStateException が送出されます。ユーザが次に同じブラウザからサーブレットにアクセスしたときには、セッション データは失われています。getSession(true) メソッドを呼び出すと、新しいセッションが作成されます。この時点で、ユーザにサイドログイン ページを送信できます。

複数のアプリケーションに対する単一のサインオンの実装

サーバまたは仮想ホストが複数の Web アプリケーションに割り当てられている場合、すべての Web アプリケーションからユーザをログアウトするには、別の方法が必要になります。サーブレット仕様では、すべての Web アプリケーションからユーザをログアウトするための API が用意されていないため、以下のメソッドを使用します。

`weblogic.servlet.security.ServletAuthentication.logout()`
: 認証データをユーザのセッション データから削除します。これにより、ユーザはログアウトされますが、セッションは引き続き有効です。

`weblogic.servlet.security.ServletAuthentication.invalidateAll()`
: ユーザのセッション データ、およびサーバまたは仮想ホストのコンテキストから認証データを削除します。クッキーも無効になります。

`weblogic.servlet.security.ServletAuthentication.killCookie()`
: 応答がブラウザに送信された直後に有効期限が切れるようにクッキーを設定して、現在のクッキーを無効にします。このメソッドでは、応答がユーザのブラウザに正常に届く必要があります。セッションはタイムアウトするまで維持されます。

単一のサインオンからの Web アプリケーションの除外

単一のサインオンへの参加から Web アプリケーションを除外するには、除外する Web アプリケーションに異なるクッキー名を定義します。詳細については、「セッション クッキーのコンフィグレーション」を参照してください。

セッション トラッキングのコンフィグレーション

WebLogic Server では、セッション トラッキングの処理方法を決定するさまざまな属性をコンフィグレーションできます。これらのセッション トラッキング属性のコンフィグレーションの詳細については、「session-descriptor 要素」を参照してください。

クッキーに代わる URL 書き換えの使用

状況によっては、ブラウザがクッキーを受け入れないこともあります。この場合、クッキーによるセッション トラッキングができません。代わりに URL 書き換えを使用すると、ブラウザがクッキーを受け入れないことを WebLogic Server が検出したときに、こうした状況を自動的に置き換えることができます。URL 書き換えでは、セッション ID を Web ページのハイパーリンクにエンコードし、サーブレットはそれらをブラウザに送り返します。ユーザが以後これらのリンクをクリックすると、WebLogic Server は URL からその ID を抽出し、適切な `HttpSession` を見つけ出します。その後、`getSession()` メソッドを使用して、セッション データにアクセスします。

WebLogic Server で URL 書き換えを有効にするには、WebLogic 固有のデプロイメント記述子の「session-descriptor」要素で `UrlRewritingEnabled` 属性を true に設定します。

URL 書き換えをサポートするために、コードで URL を適切に処理するには、以下のガイドラインに従います。

- 次に示すように、URL を出力ストリームに直接書き出すことは避けます。

  ```
  out.println("<a href=\"/myshop/catalog.jsp\">catalog</a>");
  ```

  代わりに、`HttpServletResponse.encodeURL()` メソッドを使用します。次に例を示します。

  ```
  out.println("<a href=\""
       + response.encodeURL("myshop/catalog.jsp")
       + "\">catalog</a>");
  ```

- `encodeURL()` メソッドを呼び出すと、URL を書き換える必要があるかどうかが調べられます。必要である場合、URL にセッション ID を組み込むことによって書き換えを行います。

- WebLogic Server への応答として返される URL に加えて、リダイレクトを送信する URL をエンコードします。次に例を示します。

```
if (session.isNew())
 response.sendRedirect(response.encodeRedirectUrl(welcomeURL));
```

WebLogic Server はセッションが新しいときには、ブラウザがクッキーを受け入れる場合でも URL 書き換えを使用します。これは、セッションの最初ではサーバはブラウザがクッキーを受け入れるかどうかを判断できないからです。

サーブレットは、`HttpServletRequest.isRequestedSessionIdFromCookie()` メソッドから返されるブール値をチェックすることによって、所定のセッションがクッキーから返されたかを確認できます。WebLogic Server アプリケーションは適切に応答するか、WebLogic Server による URL 書き換えに依存します。

注意： CISCO Local Director ロード バランサでは、URL 書き換えの区切り記号として疑問符 (?) が想定されています。WLS の URL 書き換えメカニズムは区切り記号としてセミコロン (;) を使用するので、WLS の URL 書き換えとこのロード バランサの間には互換性がありません。

URL 書き換えと Wireless Access Protocol (WAP)

WAP アプリケーションを作成する場合、WAP プロトコルはクッキーをサポートしていないため、URL を書き換える必要があります。また、一部の WAP デバイスでは、URL の長さが 128 文字（パラメータも含む）に制限されます。これにより、URL 書き換えによって転送できるデータ サイズが制限されます。パラメータ用の領域を増やすために、WebLogic Server によってランダムに生成されるセッション ID のサイズを制限できます。そのためには、WebLogic 固有のデプロイメント記述子である `weblogic.xml` の `<session-descriptor>` 要素の `IDLength` 属性を使用してバイト数を指定します。

最小値は 8 バイト、デフォルト値は 52 バイトです。最大値は `Integer.MAX_VALUE` です。

セッションの永続化

WebLogic Server は、永続ストレージにセッション データを記録するように設定できます。セッション永続性を使用する場合、以下の特徴があります。

- フェイルオーバのよさ。サーバに障害が発生してもセッションが保存されるためです。
- ロード バランスのよさ。どのサーバでも、セッションがいくつあってもそのリクエストを処理し、キャッシュを使用して、パフォーマンスを最適化できるためです。詳細については、「セッションの永続性のコンフィグレーション」の `cacheEntries` プロパティを参照してください。
- セッションを、クラスタ化された WebLogic Server 間で共有できます。WebLogic Cluster ではセッション永続性は必要条件ではなくなりました。その代わりに、ステートのインメモリ レプリケーションを使用します。詳細については、『WebLogic Server Clusters ユーザーズ ガイド』を参照してください。
- 顧客が最高品質のサーブレット セッション永続性を求めている場合は、JDBC ベースの永続性が最適です。セッション永続性はある程度犠牲にしても、パフォーマンスを大幅に高めたい顧客の場合には、インメモリ レプリケーションが適切な選択です。JDBC ベースの永続性は、インメモリ レプリケーションよりも著しく低速です。サーブレット セッションに対して、インメモリ レプリケーションが JDBC ベースの永続性よりも 8 倍高いパフォーマンスを提供した場合もあります。
- どの種類の Java オブジェクトでもセッションに入れることはできますが、ファイル、JDBC、およびインメモリ レプリケーションについては、`java.io.Serializable` であるオブジェクトだけがセッションに格納されます。詳細については、「セッションの永続性のコンフィグレーション」を参照してください。

セッション使用時に避けるべき状況

セッション永続性を、セッション間の長期データを格納する目的には使わないでください。つまり、後日クライアントがサイトに戻ったときに、アクティブなままのセッションがあっても、それに依存してはいけません。むしろアプリケーションでは、長期にわたる情報や重要な情報は、データベースの中に記録すべきです。

セッションはクッキーのコンビニエンス ラッパーではありません。セッションには長期間であろうと一定期間であろうと、クライアント データを格納しようとしてはいけません。それよりも、アプリケーションのほうでクッキーをブラウザ上に作成し設定すべきです。例として、クッキーが長期間存続できる自動ログ

イン機能、クッキーが短期間で失効する自動ログアウト機能などがあります。この場合、HTTP セッションを使用しないでください。代わりに、アプリケーション固有のロジックを記述します。

シリアライズ可能な属性値の使い方

永続セッションを使用する場合、セッションに追加するすべての属性 value オブジェクトは java.io.Serializable を実装する必要があります。シリアライズ可能なクラスの作成の詳細については、シリアライズ可能なオブジェクトに関するオンライン Java チュートリアルを参照してください。独自のシリアライズ可能なクラスを永続セッションに加えるときは、クラスの各インスタンス変数もシリアライズ可能になっているか注意してください。シリアライズ可能でない場合は、それを transient として宣言できます。WebLogic Server は、その変数を永続ストレージには保存しません。transient とするべきインスタンス変数の一般的な例としては、HttpSession オブジェクトが挙げられます（3-20 ページの「セッションの永続化」の、セッションにおけるシリアライズされたオブジェクトの使用に関する注意を参照してください）。

セッションの永続性のコンフィグレーション

永続セッションの設定の詳細については、「セッションの永続性のコンフィグレーション」を参照してください。

サーブレットでのクッキーの使い方

クッキーは情報の一片です。サーバはこの情報をユーザのディスク上へローカルに保存するよう、クライアント ブラウザに要求します。ブラウザは、同じサーバにアクセスするたび、HTTP リクエストと共にそのサーバに関連したクッキーをすべて送ります。クッキーは、クライアントからサーバに戻されるので、そのクライアントを識別するために便利なものです。

各クッキーは名前と値を持っています。通常、クッキーをサポートしているブラウザでは、各サーバのドメインに、1 つあたり最大 4K のクッキーを 20 まで格納することができます。

HTTP サーブレットでのクッキーの設定

ブラウザ上にクッキーを設定するには、クッキーを作成し、値を与え、サーブレットのサービス メソッドの 2 番目のパラメータである HttpServletResponse オブジェクトに追加します。次に例を示します。

```
Cookie myCookie = new Cookie("ChocolateChip", "100");
myCookie.setMaxAge(Integer.MAX_VALUE);
response.addCookie(myCookie);
```

上記のサンプルでは、値が 100 の ChocolateChip と呼ばれるクッキーを、応答の送信時にブラウザ クライアントに追加します。クッキーの有効期限は指定できる最大値に設定されているため、クッキーは永久に有効です。クッキーは文字列型の値のみを受け入れるので、クッキーに格納するための必要な型との間でキャストします。EJB の場合、一般的にはクッキーの値に対する EJB インスタンスのホーム ハンドルを使用し、EJB にユーザの詳細情報を格納して、後で参照できるようにします。

HTTP サーブレットでのクッキーの取得

service() メソッドへの引数としてサーブレットに渡される HttpServletRequest から、クッキー オブジェクトを取得することができます。クッキーそのものは、javax.servlet.http.Cookie オブジェクトとして示されます。

サーブレット コードでは、getCookies() メソッドを呼び出すことにより、ブラウザから送られたすべてのクッキーを取得することができます。次に例を示します。

```
Cookie[] cookies = request.getCookies();
```

このメソッドは、ブラウザから送られたすべてのクッキーの配列を返すか、またはブラウザから送られたクッキーがない場合、null を返します。サーブレットは、その配列を処理して正しい名前のクッキーを探す必要があります。クッキーの名前は、Cookie.getName() メソッドを使って取得することができます。同一の名前で、パス属性の異なるクッキーが複数ある可能性があります。サーブレットが、同一の名前でパス属性の異なる複数のクッキーを設定した場合、Cookie.getPath() メソッドを使ってこれらを比較することも必要です。以下のコードは、ブラウザから送られたクッキーの詳細にアクセスする方法を示してい

ます。このサーバに送られたクッキーはすべてユニークな名前を持ち、ブラウザクライアントで以前に設定したと考えられる ChocolateChip というクッキーを探しているという前提です。

```
Cookie[] cookies = request.getCookies();
boolean cookieFound = false;

for(int i=0; i < cookies.length; i++) {
  thisCookie = cookies[i];
  if (thisCookie.getName().equals("ChocolateChip")) {
    cookieFound = true;
    break;
  }
}

if (cookieFound) {
  // クッキーが見つかったので、その値を取得する
  int cookieOrder = String.parseInt(thisCookie.getValue());
}
```

クッキーの詳細については、以下を参照してください。

- 「Cookie API」
- 「The Java Tutorial: Using Cookies」

HTTP と HTTPS の両方で送信されるクッキーの使い方

HTTP リクエストと HTTPS リクエストは異なるポートに送られるので、ブラウザによっては、HTTP リクエストに入れて送られてきたクッキーを、その後続の HTTPS リクエストに包含しない（あるいはその逆）ことがあります。このような場合には、サーブレット リクエストが HTTP と HTTPS の間で切り替わると、新しいセッションが作成されることになります。セッション内でリクエストが行われるたびに、特定のドメインによって設定されるクッキーがサーバに送られるようにするには、CookieDomain 属性をドメイン名に設定します。CookieDomain 属性は、サーブレットが含まれる Web アプリケーションのための WebLogic 固有のデプロイメント記述子（weblogic.xml）の <session-descriptor> 要素で設定します。次に例を示します。

```
<session-descriptor>
  <session-param>
    <param-name>CookieDomain</param-name>
    <param-value>mydomain.com</param-value>
```

```
  </session-param>
</session-descriptor>
```

CookieDomain 属性は、mydomain.com によって指定されたドメイン内のホストに、すべてのリクエストについて適切なクッキーを入れるようブラウザに指示します。このプロパティやセッション クッキーのコンフィグレーションの詳細については、「セッション管理の設定」を参照してください。

安全なクッキー

サービス パック 2 では、sessionCookie を保護するように指定するための新しいパラメータが導入されました。このパラメータを設定すると、クライアントのブラウザは、HTTPS 接続を通してだけクッキーを返送します。この機能は、クッキーの ID が安全であることを保証するするもので、HTTPS だけを使用する Web サイトでのみ使用する必要があります。この機能を有効にすると、HTTP 経由でのセッション クッキーは動作しなくなり、HTTPS 以外を使用する場所にクライアントを渡そうとしても、セッションは送信されません。この機能を使用する場合は、URLRewriting をオフにすることを強くお勧めします。アプリケーションが URL をコード化しようとした場合、セッション ID は HTTP を介して共有されます。この機能を使用するには、weblogic.xml に以下のコードを追加してください。

```
<session-param>
<param-name>CookieSecure</param-name>
<param-value>true</param-value>
</session-param>
<session-param>
```

アプリケーションのセキュリティとクッキー

クッキーの使用は、マシン上での自動的なアカウント アクセスを可能にして便利ですが、セキュリティの観点からは望ましくない場合があります。クッキーを使用するアプリケーション設計時には、以下のガイドラインに従ってください。

- クッキーが常にユーザに対して正確であると考えないようにします。マシンが共有されている場合や、同一のユーザが異なるアカウントにアクセスしようとする場合もあります。

- ユーザが、サーバ上にクッキーを残すかどうか選択できるようにします。共有マシンでは、ユーザがそのアカウントに対する自動的なログインをそのままにしておくことを望まない場合があります。ユーザがクッキーについて理解していると仮定せずに、以下のような質問をします。

  「このコンピュータから自動ログインしますか？」

- 注意の必要なデータを取得するためにログインするユーザに対しては、常にパスワードを要求します。ユーザがそれ以外の方法を要求しない限り、この選択結果とパスワードはユーザのセッション データに格納できます。セッションのクッキーは、ユーザがブラウザを終了したときに有効期限が切れるようにコンフィグレーションします。

HTTP サーブレットからの WebLogic サービスの使い方

HTTP サーブレットを記述する際には、JNDI、RMI、EJB、JDBC 接続など、WebLogic Server の豊富な機能を利用できます。

次のマニュアルには、これらの機能の詳細が記載されています。

- 『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』
- 『WebLogic JDBC プログラミング ガイド』
- 『WebLogic JNDI プログラマーズ ガイド』
- 『WebLogic JMS プログラマーズ ガイド』

データベースへのアクセス

WebLogic Server は、サーバサイド Java クラス（サーブレットなど）からの Java Database Connectivity（JDBC）の使用をサポートしています。JDBC を使うと、Java クラスから SQL クエリを実行し、クエリの結果を処理できます。JDBC と WebLogic Server の詳細については、『WebLogic JDBC プログラミング ガイド』を参照してください。

以下の節で説明するように、JDBC はサーブレットで使用できます。

JDBC 接続プールを用いたデータベースへの接続

接続プールとは、接続プールが登録されるとき（通常は WebLogic Server の起動時）に作成される、データベースへの同一 JDBC 接続のグループに名前を付けたものです。サーブレットはプールから接続を「借り」、使用後に接続を閉じることでプールに接続を返します。このプロセスは、接続プールの使用は、データベースへのアクセスが必要になるたびにクライアントごとに新しい接続を確立するよりもはるかに効率的です。もう 1 つの利点は、データベースについての詳細をサーブレットのコードに組み込む必要がないということです。

JDBC 接続プールに接続するには、次の多層 JDBC ドライバのうち 1 つを使用します。

- プール ドライバ。ほとんどのサーバサイド処理に使用します。
  - ドライバ URL : `jdbc:weblogic:pool`
  - ドライバ パッケージ名 : `weblogic.jdbc.pool.Driver`
- JTS プール ドライバ。データベース操作にトランザクションのサポートが必要な場合に使用します。
  - ドライバ URL : `jdbc:weblogic:jts`
  - ドライバ パッケージ名 : `weblogic.jdbc.jts.Driver`

サーブレットでの接続プールの使い方

次の例では、サーブレットからのデータベース接続プールの使い方を示します。

1. プール ドライバをロードし、`java.sql.Driver` にキャストします。ドライバの絶対パス名は、`weblogic.jdbc.pool.Driver` です。次に例を示します。

```
Driver myDriver = (Driver)
   Class.forName("weblogic.jdbc.pool.Driver").newInstance();
```

2. ドライバの URL を使用した接続に加えて、登録された接続プールの名前（省略可能）を作成します。プール ドライバの URL は、`jdbc:weblogic:pool` です。

   プールは、以下の 2 通りの方法で識別できます。

   - `connectionPoolID` キーを使用して、`java.util.Properties` オブジェクトで接続プール名を指定します。次に例を示します。

     ```
     Properties props = new Properties();
     props.put("connectionPoolID", "myConnectionPool");
     Connection conn =
       myDriver.connect("jdbc:weblogic:pool", props);
     ```

   - プール名を URL の末尾に追加します。この場合、プールからの接続を使用するためにユーザ名とパスワードを設定していない限り、`Properties` オブジェクトは不要です。次に例を示します。

     ```
     Connection conn =
        myDriver.connect("jdbc:weblogic:pool:myConnectionPool",
     null);
     ```

   上のサンプルでは、`DriverManger.getConnection()` メソッドの代わりに、`Driver.connect()` メソッドが使われています。データベース接続を取得するために `DriverManger.getConnection()` を使用することもできますが、`Driver.connect()` を使用することをお勧めします。このメソッドは同期を取られることがなく、よりよいパフォーマンスを提供するためです。

   `connect()` が返す Connection は、`weblogic.jdbc.pool.Connection` のインスタンスであることに注意してください。

3. JDBC の呼び出しが終わったら、`Connection` オブジェクトに対して `close()` メソッドを呼び出して、接続を正しくプールに戻します。コーディング方法としては、`try` ブロックに接続を作成してから、`finally` ブロックで接続を閉じ、いかなる場合でも確実に接続がクローズされるようしてください。

   ```
   conn.close();
   ```

DataSource オブジェクトを用いたデータベースへの接続

DataSource は、接続プールを参照するサーバサイド オブジェクトです。接続プールの登録により、JDBC ドライバ、データベース、ログインなど、データベース接続と関連するパラメータが定義されます。DataSource オブジェクトおよび接続プールは、Administration Console で作成します。J2EE 準拠のアプリケーションを作成する場合は、DataSource オブジェクトの使用をお勧めします。

サーブレットでの DataSource の使用

1. Administration Console を使って接続プールを登録します。詳細については、「JDBC 接続プールの作成」を参照してください。
2. 接続プールを指す DataSource オブジェクトを登録します。詳細については、「JDBC データ ソース」を参照してください。
3. JNDI ツリーで、DataSource オブジェクトをルックアップします。次に例を示します。

```
Context ctx = null;

// JNDI ルックアップのためのコンテキストを取得する
    ctx = new InitialContext(ht);

// DataSource オブジェクトをルックアップする
    javax.sql.DataSource ds
      = (javax.sql.DataSource) ctx.lookup ("myDataSource");
```

4. DataSource を使用して、JDBC 接続を作成します。次に例を示します。

```
java.sql.Connection conn = ds.getConnection();
```

5. 接続を使用して、SQL 文を実行します。次に例を示します。

```
Statement stmt = conn.createStatement();
stmt.execute("select * from emp");

. . .
```

JDBC ドライバを用いたデータベースへの直接接続

データベースへの直接接続は、データベース接続を確立する方法としては、最も効率の悪いものです。リクエストごとに新しいデータベース接続を確立しなければならないからです。データベースへの接続には、どの JDBC ドライバも使用できます。BEA では、Oracle、Microsoft SQL Server、および Informix 用の JDBC ドライバを提供しています。詳細については、『BEA WebLogic JDBC プログラミング ガイド』を参照してください。

HTTP サーブレットにおけるスレッドの問題

サーブレットの設計時に、高い負荷のもとで、WebLogic Server がサーブレットをどのように呼び出すか検討する必要があります。複数のクライアントが同時にサーブレットをヒットすることは避けられません。したがって、サーブレットのコードは、共有リソースやインスタンス変数の共有違反を防ぐように記述します。この問題を避けて設計するためのヒントを以下に示します。

SingleThreadModel

`SingleThreadModel` を実装したクラスのインスタンスは、同時に複数のスレッドで呼び出されないことが保証されています。`SingleThreadModel` サーブレットの複数のインスタンスを使用して、それぞれをシングル スレッドで実行しながら、同時に発生するリクエストを処理します。

`SingleThreadModel` を効率的に使用するため、WebLogic Server は `SingleThreadModel` を実装する各サーブレットについて、サーブレット インスタンスのプールを作成します。サーブレットに対する最初のリクエストが行われると、WebLogic Server はサーブレット インスタンスのプールを作成し、必要に応じてプール内のサーブレット インスタンスの数を増やしていきます。

WebLogic Server Administration Console で設定された属性 [シングル スレッド サーブレット プール サイズ] は、サーブレットに初めてリクエストが送られたときに作成されるサーブレット インスタンスの最初の数を指定します。この属性は、`SingleThreadModel` サーブレットが処理する予定の同時リクエストの平均数に設定します。

サーブレットの設計時に、ファイルやデータベースへのアクセスのようなサーブレット クラスの外部の共有リソースの使い方に注意を払う必要があります。同一のサーブレット インスタンスが複数存在し、まったく同じリソースを使用する可能性があるため、`SingleThreadModel` を実装した場合でも、解決の必要がある同期と共有の問題が発生します。

共有リソース

共有リソースの問題は、個々のサーブレットごとに処理することをお勧めします。次のガイドラインを念頭に置いてください。

- 可能な限り、同期を避けます。引き続き発生するサーブレットのリクエストが、現行のスレッドが完了するまで、ボトルネックになるためです。
- 各サーブレット リクエストに固有の変数は、サービス メソッドのスコープ内で定義します。ローカル スコープ変数は、スタックに保存されるため、同一のメソッド内で実行されている複数のスレッドで共有されることはありません。したがって、同期の必要性はなくなります。
- 外部リソースへのアクセスは、Class レベルで同期を取るか、トランザクションにカプセル化します。

別のリソースへのリクエストのディスパッチ

この節では、リクエストをサーブレットから別のリソースへディスパッチするのによく使用されるメソッドの概要を説明します。

サーブレットでは、リクエストを別のリソース（サーブレット、JSP、または HTML ページなど）に渡すことができます。このプロセスは、リクエストのディスパッチと呼ばれます。リクエストをディスパッチする場合は、`RequestDispatcher` インタフェースの `include()` または `forward()` を使用します。`forward()` または `include()` メソッドを使用する場合、出力を応答オブジェクトに書き込める時期には制限があります。この制限についても、この節で説明します。

リクエストのディスパッチに関する完全な説明については、Sun Microsystems が提供するサーブレット 2.2 仕様を参照してください。

`RequestDispatcher` を使用すると、HTTP リダイレクト応答をクライアントに送り返す必要がなくなります。`RequestDispatcher` は、HTTP リクエストを要求されたリソースに渡します。

リソースを特定のリソースにディスパッチするには、以下の手順に従います。

1. 次のように、`ServletContext` への参照を取得します。

   ```
   ServletContext sc = getServletConfig().getServletContext();
   ```

2. 以下のメソッドの 1 つを用いて、`RequestDispatcher` オブジェクトをルックアップします。

   - `RequestDispatcher rd = sc.getRequestDispatcher(String` *`path`*`);`

     `path` は、Web アプリケーションのルートに対する相対パスでなければなりません。

   - `RequestDispatcher rd = sc.getNamedDispatcher(String` *`name`*`);`

     `name` を、Web アプリケーションのデプロイメント記述子の中で `<servlet-name>` 要素によってそのサーブレットに割り当てられた名前で置き換えます。詳細については、「servlet 要素」を参照してください。

   - `RequestDispatcher rd = ServletRequest.getRequestDispatcher(String` *`path`*`);`

     このメソッドは `RequestDispatcher` オブジェクトを返すものであって、`ServletContext.getRequestDispatcher(String` *`path`*`)` メソッドに似ています。ただし、ここでは、*`path`* を現在のサーブレットに対して相対的になるように指定することができます。「/」記号で始まるパスは、Web アプリケーションに対して相対的になるように解釈されます。

   HTTP サーブレット、JSP ページ、通常の HTML ページなど、Web アプリケーション内のどのリソースについても、`getRequestDispatcher()` メ

ソッドでリソースの適切な URL を要求することによって、RequestDispatcher を取得できます。返された RequestDispatcher を使用して、リクエストを別のサーブレットに転送します。

3. 適切なメソッドを使用して、リクエストを転送またはインクルードします。
   - `rd.forward(request,response);`
   - `rd.include(request,response);`

   これらのメソッドは、次の 2 つの節で説明しています。

リクエストの転送

一度、正しい RequestDispatcher が得られると、サーブレットは、引数として、HTTPServletRequest と HTTPServletResponse を渡し、RequestDispatcher.forward() メソッドを使用して、リクエストを転送します。出力がすでにクライアントに送られた状態でこのメソッドを呼び出すと、IllegalStateException が送出されます。応答バッファの中に、コミットされていない保留中の出力がある場合には、バッファはリセットされます。

サーブレットは、応答に対する以前の出力を書き込もうとしてはいけません。リクエストを転送する前に、応答のためにサーブレットが ServletOutputStream または、PrintWriter を取得すると、IllegalStateException が送出されます。

元のサーブレットからのそれ以外の出力はすべて、リクエストが転送された後は無視されます。

どのタイプの認証を使用する場合でも、転送されたリクエストは、デフォルトではユーザに再認証を要求しません。この動作を変更して、転送されたリクエストの認証を実行するには、<check-auth-on-forward/> 要素を WebLogic 固有のデプロイメント記述子（weblogic.xml）の <container-descriptor> 要素に追加します。次に例を示します。

```
<container-descriptor>
    <check-auth-on-forward/>
</container-descriptor>
```

デフォルトの動作は、サーブレット仕様 2.3 のリリースで変更されたことに注意してください。サーブレット 2.3 仕様では、転送されたリクエストの認証は要求されないことが規定されています。

WebLogic 固有のデプロイメント記述子の編集方法については、「WebLogic 固有のデプロイメント記述子（weblogic.xml）の記述」を参照してください。

リクエストのインクルード

サーブレットには、`RequestDispatcher.include()` メソッドを使用し、引数として `HTTPServletRequest` と `HTTPServletResponse` を渡すことにより、他のリソースからの出力をインクルードすることができます。その場合、インクルードされたリソースは、リクエスト オブジェクトにアクセスできます。

インクルードされたリソースは、応答オブジェクトの `ServletOutputStream` または `Writer` オブジェクトにデータを書き戻すことができ、その後、応答バッファにデータを追加するか、応答オブジェクトに対し `flush()` メソッドを呼び出すかのいずれかを行うことができます。応答のステータス コード、またはインクルードされたサーブレットの応答からの HTTP ヘッダ情報を設定しようとすると、すべて無視されます。

実際には、`include()` メソッドを使用して、サーブレット コードから他の HTTP リソースの「サーバサイドインクルード」を実現できます。

4　管理とコンフィグレーション

以下の節では、WebLogic HTTP サーブレットの管理およびコンフィグレーション タスクについて概要します。サーブレットの管理とコンフィグレーションの詳細については、「サーブレットのコンフィグレーション」を参照してください。

この節では以下の内容について説明します。

- WebLogic HTTP サーブレットの管理の概要
- Web アプリケーションでのサーブレットの参照
- Web アプリケーションのディレクトリ構造
- サーブレットのセキュリティ
- サーブレット開発のヒント
- サーブレットのクラスタ化

WebLogic HTTP サーブレットの管理の概要

Java 2 Enterprise Edition の規格に準拠するため、HTTP サーブレットは Web アプリケーションの一部としてデプロイされます。Web アプリケーションとは、サーブレット クラス、JavaServer Pages（JSP）、静的な HTML ページ、画像、ユーティリティ クラスなどのアプリケーション コンポーネントをグループ化したものです。

Web アプリケーションでは、コンポーネントは標準的なディレクトリ構造を用いてデプロイされます。このディレクトリ構造は、`.war` ファイルと呼ばれるファイルにアーカイブされて、WebLogic Server 上にデプロイされます。Web ア

プリケーションのリソースと操作パラメータに関する情報は、Web アプリケーションと共にパッケージ化されている 2 つのデプロイメント記述子で定義されます。

サーブレットをコンフィグレーション、デプロイするためのデプロイメント記述子の使い方

第 1 のデプロイメント記述子、`web.xml` は、Sun Microsystems のサーブレット 2.2 仕様に従って定義され、Web アプリケーションを記述する標準フォーマットを指定します。第 2 のデプロイメント記述子、`weblogic.xml` は、`web.xml` ファイルで定義されているリソースを WebLogic Server 内で使用可能なリソースにマップして、JSP の動作と HTTP セッション パラメータを定義する WebLogic 固有のデプロイメント記述子です。

web.xml（Web アプリケーション デプロイメント記述子）

Web アプリケーションのデプロイメント記述子では、HTTP サーブレットの以下の属性を定義します。

- サーブレット名
- サーブレットの Java クラス
- サーブレット初期化パラメータ
- サーブレットの `init()` メソッドを WebLogic Server の起動時に実行するかどうか
- 一致した場合には、サーブレットを呼び出す URL パターン
- セキュリティ
- MIME タイプ
- エラー ページ
- EJB への参照
- 他のリソースへの参照

web.xml ファイルの作成に関する詳細については、「Web アプリケーションのデプロイメント記述子の記述」を参照してください。

weblogic.xml （WebLogic 固有のデプロイメント記述子）

WebLogic 固有のデプロイメント記述子では、HTTP サーブレットの以下の属性を定義します。

- HTTP セッションのコンフィグレーション
- クッキーのコンフィグレーション
- EJB リソースのマッピング
- JSP のコンフィグレーション

weblogic.xml ファイルの作成に関する詳細については、「Web アプリケーションのデプロイメント記述子の記述」を参照してください。

WebLogic Server Administration Console

WebLogic Server Administration Console を使用して、以下のパラメータを設定します。

- HTTP パラメータ
- ログ ファイル
- URL 書き換え
- キープアライブ
- デフォルト MIME タイプ
- クラスタ化パラメータ
- 仮想ホスティングのための URL マッピング

詳細については、以下のリソースを参照してください。

- Administration Console:「Web アプリケーション」
- Administration Console: 「仮想ホスト」

Web アプリケーションのディレクトリ構造

すべての Web アプリケーションについて、以下のディレクトリ構造を使用します。

```
WebApplicationRoot\（.jsp、.html、.jpg、.gif などの
                    |  公開されるファイル）
                    |
                    +WEB-INF\-+
                              |
                              + classes\（Web アプリケーションで使用
                              |          されるサーブレットなどの
                              |          Java クラスを格納する
                              |          ディレクトリ）
                              |
                              + lib\（Web アプリケーションで使用
                              |          される jar ファイルを格納する
                              |          ディレクトリ）
                              |
                              + web.xml
                              |
                              + weblogic.xml
```

Web アプリケーションでのサーブレットの参照

Web アプリケーションでサーブレットを参照するための URL は、次のように構成されます。

```
http://myHostName:port/myContextPath/myRequest/?myRequestParameters
```

URL の各要素は次のように定義します。

myHostName

WebLogic Server Administration Console で定義される、WebLogic Server にマップされる DNS 名です。

URL のこの部分は、*host*:*port* に置き換えることができます。*host* は、WebLogic Server が実行されているマシン名、*port* は WebLogic Server がリクエストをリスンしているポートです。

port
WebLogic Server がリクエストをリスンしているポートです。

myContextPath
WebLogic Server Administration Console で定義される Web アプリケーション名です。

myRequest
web.xml で定義されるサーブレット名です。

?myRequestParameters
URL にエンコードされる HTTP リクエスト パラメータです（省略可能）。HTTP サーブレットで読み取り可能です。

サーブレットのセキュリティ

サーブレットのセキュリティは、そのサーブレットが含まれる Web アプリケーションのコンテキストで定義されます。セキュリティは WebLogic Server で処理することも、プログラムによってサーブレット クラスに組み込むこともできます。

詳細については、「Web アプリケーションでのセキュリティのコンフィグレーション」を参照してください。

認証

次の 3 つの手法のうちいずれかを使用して、サーブレットにユーザ認証を組み込むことができます。

- BASIC ― ブラウザを使って、ユーザ名とパスワードを収集します。
- FORM ― HTML のフォームを使って、ユーザ名とパスワードを収集します。
- クライアント証明書 ― デジタル証明書を使って、ユーザを認証します。詳細については、「デジタル証明書」を参照してください。

BASIC および FORM の手法は、ユーザとパスワードの情報が格納されたセキュリティ レルム内に呼び出しを行うものです。WebLogic Server に付属しているデフォルトのレルムを使うことも、Windows NT、UNIX、RDBMS の各レルム、ユーザ定義のレルムなど、既存のさまざまなレルムを使うこともできます。セキュリティ レルムの詳細については、「セキュリティ レルム」を参照してください。

認可（セキュリティ制約）

セキュリティ制約を使用すると、Web アプリケーションにおけるサーブレットなどのリソースへのアクセスを制限することができます。セキュリティ制約は、Web アプリケーション デプロイメント記述子（`web.xml`）で定義されています。セキュリティ制約には、3 つの基本的なタイプがあります。

- ロールやリソースによるリソースの制約
- セキュア ソケット レイヤ（SSL）の暗号化
- プログラムによる認可

ロールは、レルム内のプリンシパルにマップできます。特定リソースの制約は、URL パターンと Web アプリケーション内のリソースを一致させることにより実現します。また、セキュリティ制約としてセキュア ソケット レイヤ（SSL）を使用できます。

`HttpServletRequest` インタフェースの次のいずれかのメソッドを使用してプログラミングし、認可を実行することも可能です。

- `getRemoteUser()`
- `isUserInRole()`
- `getUserPrincipal()`

詳細については、「javax.servlet API」を参照してください。

サーブレット開発のヒント

HTTP サーブレットを作成する際には、次のヒントを考慮してください。

- サーブレットは、Web アプリケーションの WEB-INF\classes ディレクトリにコンパイルします。
- サーブレットは、必ず Web アプリケーションのデプロイメント記述子（web.xml）に登録します。
- サーブレットのリクエストに応答する際、WebLogic Server はサーブレット クラス ファイルのタイム スタンプをチェックし、メモリ内にある既存のサーブレット インスタンスと比較します。バージョンの新しいサーブレット クラスがあれば、WebLogic Server はそのサーブレット クラスを再ロードします。サーブレットが再ロードされると、そのサーブレットの init() メソッドが呼び出されます。

  WebLogic Server がタイム スタンプをチェックする間隔（秒単位）を [再ロード間隔（秒）] 属性で設定できます。この属性は、Administration Console で、Web アプリケーションの [ファイル] タブで設定します。この属性をゼロにすると、WebLogic Server はリクエストごとにタイム スタンプをチェックします。これはサーブレットの開発とテスト中には便利ですが、プロダクション環境では必要以上に時間を消費します。この属性を -1 に設定すると、WebLogic Server は変更されたサーブレットについてはチェックを行いません。

サーブレットのクラスタ化

サーブレットをクラスタ化することにより、サーブレットのフェイルオーバとロードバランシングが可能になります。サーブレットを WebLogic Server クラスタにデプロイするには、サーブレットを含んでいる Web アプリケーションをクラスタ内の全サーバにデプロイします。手順については、『WebLogic Server クラスタ ユーザーズ ガイド』の「Web アプリケーションと EJB をデプロイする」を参照してください。

クラスタ化されるサーブレットのフェイルオーバについて、詳しくは『WebLogic Server クラスタ ユーザーズ ガイド』の「クラスタ化されたサービスのフェイルオーバ サポート」および「HTTP セッション ステートのレプリケーションについて」を参照してください。

注意： サーブレットの自動フェイルオーバを行うためには、サーブレットのセッション ステートをメモリ内にレプリケートすることが必要になります。手順については、『WebLogic Server クラスタ ユーザーズ ガイド』の「クラスタにおけるインメモリ HTTP レプリケーションのコンフィグレーション」を参照してください。

『WebLogic Server クラスタ ユーザーズ ガイド』の「HTTP セッション ステートのロード バランシング」では、アーキテクトと管理者向けに、WebLogic Server クラスタで可能なサーブレットのロードバランシングについての情報と、それに関連する計画およびコンフィグレーション上の考慮事項を示しています。

索引