Web アプリケーション開発者ガイド

Sun™ ONE Application Server

Version 7

817-0604-10
2002 年 9 月

目 次

本書について

このマニュアルでは、Sun™ Open Net Environment (Sun ONE) Application Server 7 上での新しい Java オープンスタンダードモデルであるサーブレットおよび JavaServer Pages (JSP) に準拠する Java 2 Platform, Enterprise Edition (J2EE) アプリケーションの作成方法および実行方法について説明します。また、プログラミングの概念およびタスクについても説明し、さらに、実装に関するヒントおよび関連資料も紹介します。

この章には次のトピックがあります。

- 対象読者
- マニュアルの使用法
- マニュアルの構成
- マニュアルの表記規則
- 関連情報
- 製品サポート

対象読者

このマニュアルは、企業内で Web アプリケーション (サーブレットおよび JSP) の開発、アセンブリ、および配備を担当する方々を対象としています。

このマニュアルでは、次のトピックに精通していることを前提としています。

- J2EE 仕様
- HTML
- Java プログラミング
- サーブレット、JSP、EJB、および JDBC の仕様に定義されている Java API
- SQL などの構造化データベースクエリ言語

- リレーショナルデータベースの概念
- デバッグ、ソースコード制御を含むソフトウェア開発プロセス

マニュアルの使用法

このマニュアルは、PDF 形式または HTML 形式でも入手できます。次のサイトを参照してください。

`http://docs.sun.com/`

次の表は、Sun ONE Application Server のマニュアルに記述されているタスクと概念を示しています。左の列にタスクと概念、右の列に参照するマニュアルを示します。

Sun ONE Application Server マニュアルの概要

情報の内容	参照するマニュアル
ソフトウェアおよびマニュアルの最新情報	リリースノート
サポート対象のプラットフォームと環境	プラットフォーム
アプリケーションサーバーの紹介。新機能、評価 (Evaluation) インストール情報、アーキテクチャの概要など	入門ガイド
Sun ONE Application Server とそのコンポーネント (サンプルアプリケーション、管理インタフェース、Sun ONE Message Queue など) のインストール	インストールガイド
Sun ONE Application Server 7 の Java オープンスタンダードモデルに準拠した J2EE アプリケーションの作成方法と実装方法。アプリケーションの設計、開発ツール、セキュリティ、アセンブリ、配備、デバッグ、ライフサイクルモジュールの作成に関する情報など	開発者ガイド
Sun ONE Application Server 7 の Web アプリケーション向け Java オープンスタンダードモデルに準拠した J2EE アプリケーションの作成方法と実装方法。Web アプリケーションプログラミングの概念とタスクの説明、サンプルコード、実装のヒント、関連資料の紹介など	Web アプリケーション開発者ガイド
Sun ONE Application Server 7 の エンタープライズ Beans 向け Java オープンスタンダードモデルに準拠した J2EE アプリケーションの作成方法と実装方法。EJB プログラミングの概念とタスクの説明、サンプルコード、実装のヒント、関連資料の紹介など	Enterprise JavaBeans 開発者ガイド

Sun ONE Application Server マニュアルの概要 (続き)

情報の内容	参照するマニュアル
Sun ONE Application Server 7 上の J2EE アプリケーションにアクセスするクライアントの作成方法	Developer's Guide to Clients
Web サービスの作成方法	Web アプリケーション開発者ガイド
JDBC、JNDI、JTS、JMS、JavaMial、リソース、コネクタなどの J2EE 機能	Developer's Guide to J2EE Features and Services
カスタム NSAPI プラグインの作成方法	NSAPI Developer's Guide
次の管理タスクの実行 • 管理インタフェースとコマンド行インタフェースの使用 • サーバーの作業環境の設定 • 管理ドメインの使用 • サーバーインスタンスの使用 • サーバーの稼動状況の監視およびログ記録 • Web サーバープラグインの設定 • Java Messaging Service の設定 • J2EE 機能の使用 • CORBA ベースのクライアント機能の設定 • データベース接続性の設定 • トランザクション管理の設定 • Web コンテナの設定 • アプリケーションの配備 • 仮想サーバーの管理	管理者ガイド
サーバー設定ファイルの編集	管理者用設定ファイルリファレンス
Sun ONE Application Server 7 環境のセキュリティの設定および管理。一般的なセキュリティ、証明書、SSL/TLS 暗号化に関する情報を含む。HTTP サーバーベースのセキュリティについても説明	セキュリティ管理者ガイド
Sun ONE Application Server 7 の J2EE CA コネクタのサービスプロバイダ実装の設定および管理。管理ツール、DTD に関する情報、サンプル XML ファイルなど	J2EE CA Service Provider Implementation 管理者ガイド

Sun ONE Application Server マニュアルの概要 (続き)

情報の内容	参照するマニュアル
Netscape Application Server バージョン 2.1 から新しい Sun ONE Application Server 7 プログラミングモデルへのアプリケーションの移行。Sun ONE Application Server に付属するオンラインバンクアプリケーションの移行サンプルなど	サーバーアプリケーションの移行および再配備
Sun ONE Message Queue の使用法	Sun ONE Message Queue については次の URL を参照 : http://docs.sun.com/

マニュアルの構成

このマニュアルでは、Web アプリケーション設計のための Sun ONE Application Server 環境の概要について説明します。内容は次のとおりです。

- 第 1 章「Web アプリケーション」

  Sun ONE Application Server における Web アプリケーションのサポートについて説明します。

- 第 2 章「サーブレットの使用法」

  サーブレットの作成および使用方法について説明します。

- 第 3 章「JavaServer Pages の使用法」

  JavaServer Pages (JSP) の作成および使用方法について説明します。

- 第 4 章「ユーザーセッションの作成と管理」

  対話間におけるのユーザーおよびトランザクションの情報を継続的に維持できるセッションを作成し、管理する方法について説明します。

- 第 5 章「Web アプリケーションのセキュリティ」

  Sun ONE Application Server 用の安全な Web アプリケーションの作成方法について説明します。

- 第 6 章「Web モジュールの構築と配備」

  Sun ONE Application Server で Web モジュールを構築して配備する方法について説明します。

- 第 7 章「サーバーでパースされる HTML の使用法」

  Sun ONE Application Server でのサーバーがパースする HTML の使用方法について説明します。

- 第 8 章「CGI の使用法」

  Sun ONE Application Server での CGI の使用方法について説明します。

最後に索引も用意されています。

マニュアルの表記規則

この節では、このマニュアルで使用する表記規則について説明します。

- 一般的な表記規則
- ディレクトリの表記規則

一般的な表記規則

このマニュアルは、次の表記規則に従っています。

- **ファイルとディレクトリのパス**は、UNIX の形式で表記します (ディレクトリ名をスラッシュで区切って表記)。Windows バージョンでは、ディレクトリパスについては UNIX と同じですが、ディレクトリの区切り記号にはスラッシュ記号ではなく円記号を使用します。
- URL は次の書式で記述します。

  http://*server*.*domain*/*path*/*file*.html

  これらの URL で、*server* はアプリケーションを実行するサーバー名、*domain* はユーザーのインターネットドメイン名、*path* はサーバー上のディレクトリの構造、*file* は個別のファイル名を示します。URL の斜体文字の部分は可変部分です。
- **フォント**は次のように使い分けます。
  - モノスペースフォントは、サンプルコード、コードのリスト、API および言語要素 (関数名、クラス名など)、ファイル名、パス名、ディレクトリ名、および HTML タグに使います。
  - 斜体文字はコード変数に使います。
  - 斜体文字は、マニュアルのタイトル、強調、変数および可変部分、およびリテラルに使われる文字にも使います。
  - **太字**は、段落の先頭またはリテラルに使われる文字の強調に使います。
- このマニュアルでは、ほとんどのプラットフォームの**インストールルートディレクトリ**を *install_dir* と記述します。例外については、14 ページの「ディレクトリの表記規則」を参照してください。

デフォルトでは、**ほとんどの**プラットフォームの *install_dir* は次の位置になります。

- Solaris 8 パッケージベースでない評価インストール :

  ユーザーのホームディレクトリ `/sun/appserver7`

- Solaris にアンバンドルの評価版以外のインストール :

  `/opt/SUNWappserver7`

- Windows のインストール :

  `C:¥Sun¥AppServer7`

上記の *default_config_dir* と *install_config_dir* は *install_dir* と同義です。例外と追加情報については、14 ページの「ディレクトリの表記規則」を参照してください。

- このマニュアルでは、**インスタンスルートディレクトリ**を *instance_dir* と記述します。これは以下のパスの省略書式です。

  default_config_dir`/domains/`*domain*`/`*instance*

- このマニュアルを通じて、特に明記のない限り、すべての **UNIX 固有の表記**は、Linux オペレーティングシステムにも適用されます。

注　　また、Forte for Java 4.0 は Sun ONE Studio 4 という名前に変更されています。

ディレクトリの表記規則

デフォルトでは、Solaris 8 および 9 パッケージベースのインストールおよび Solaris 9 バンドル版のインストールを使用すると、アプリケーションサーバーファイルはいくつかのルートディレクトリに分散してインストールされます。この節では、これらのディレクトリについて説明します。

- **Solaris 9 バンドル版のインストール**のデフォルトのインストールディレクトリは、次のように表記します。

  - *install_dir*: `/usr/appserver/` ディレクトリを表します。このディレクトリには、インストールイメージの静的な部分が格納されます。アプリケーションサーバーを構成するすべてのユーティリティ、実行可能ファイル、ライブラリが格納されます。

  - *default_config_dir*: 作成されたドメインのデフォルトの格納先となる `/var/appserver/domains` ディレクトリを表します。

- *install_config_dir*: `/etc/appserver/config` ディレクトリを表します。このディレクトリには、インストール全体の設定情報が格納されます。たとえば、このインストールのライセンス、管理ドメインのマスターリストなどが格納されます。

- **Solaris 8 および 9 パッケージーベースのアンバンドルの評価版以外のインストール**のデフォルトのインストールディレクトリは、次のように表記します。
  - *install_dir*: `/opt/SUNWappserver7` ディレクトリを表します。このディレクトリには、インストールイメージの静的な部分が格納されます。アプリケーションサーバーを構成するすべてのユーティリティ、実行可能ファイル、ライブラリが格納されます。
  - *default_config_dir*: 作成されたドメインのデフォルトの格納先となる `/var/opt/SUNWappserver7/domains` ディレクトリを表します。
  - *install_config_dir*: `/etc/opt/SUNWappserver7/config` ディレクトリを表します。このディレクトリには、インストール全体の設定情報が格納されます。たとえば、このインストールのライセンス、管理ドメインのマスターリストなどが格納されます。

関連情報

公式の仕様書の URL ディレクトリには、*install_dir*`/docs/index.htm` からアクセス可能です。また、次の書籍や Web サイトも参考にしてください。

サーブレットおよび JSP を使ったプログラミング

『Java Servlet Programming』、Jason Hunter 著、O'Reilly 発行

『Java Threads, 2nd Edition』、Scott Oaks、Henry Wong 共著、O'Reilly 発行

JDBC を使ったプログラミング

『Database Programming with JDBC and Java』、George Reese 著、O'Reilly 発行

『JDBC Database Access With Java: A Tutorial and Annotated Reference (Java Series)』、Graham Hamilton、Rick Cattell、Maydene Fisher 共著

製品サポート

ご使用のシステムに問題が発生した場合は、次のいずれかの方法でカスタマサポートにお問い合わせください。

- 次のオンラインサポート Web サイトをご利用ください。

  `http://www.sun.com/supportraining/`

- 保守契約を結んでいるお客様の場合は、専用ダイヤルをご利用ください。

事前に、次の情報を用意してください。サポート担当者がお客様の問題を解決するために必要な情報です。

- 問題が発生した箇所や動作への影響など、問題の具体的な説明
- マシン機種、OS バージョン、および、問題の原因と思われるパッチやそのほかのソフトウェアなどの製品バージョン
- 問題を再現するための具体的な手順の説明
- エラーログやコアダンプ

第 1 章

Web アプリケーション

この章では、Sun ONE Application Server で Web アプリケーションがサポートされる仕組みについて説明します。この章には次の節があります。

- Web アプリケーションの概要
- Web アプリケーションの作成
- Web アプリケーションの配備
- Web アプリケーションのデバッグ
- 国際化に関する問題
- 仮想サーバー
- デフォルトの Web モジュール
- サーブレットと JSP のキャッシュ
- データベース接続プール
- Web コンテナの設定
- Web アプリケーションの例

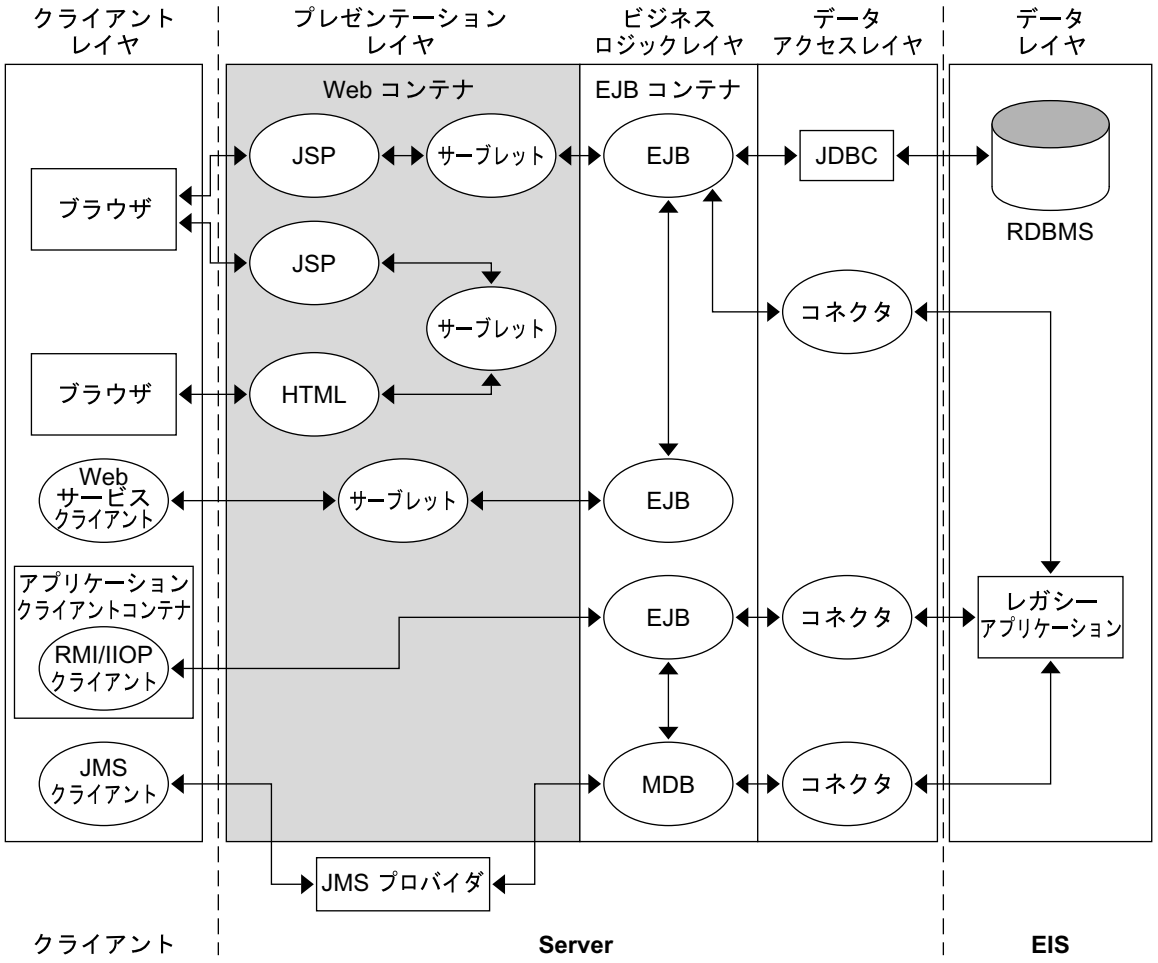
Web アプリケーションの概要

Sun ONE Application Server 7 は Servlet 2.3 API 仕様をサポートしています。この仕様では、サーブレットや JSP を Web アプリケーションに組み込むことができます。

Web アプリケーションは、サーブレット、JavaServer Pages、HTML ドキュメント、および、イメージファイルや圧縮アーカイブなどのデータを含むその他の Web リソースの集まりです。Web アプリケーションは、アーカイブ (WAR ファイル) にパッケージされている場合とオープンディレクトリ構造内に置かれている場合があります。

次の図に、J2EE 環境の詳細を示します。Web アプリケーションはプレゼンテーションレイヤにあります。Web コンテナ内で 2 つの Web アプリケーションが実行されています。1 つは EJB モジュールとコネクタを持つ完全なアプリケーションの一部として、もう 1 つは個別に配備された Web モジュールとして実行されています。

J2EE 環境における Web アプリケーション



また、Sun ONE Application Server 7 は、J2EE アプリケーションコンポーネントではない SHTML と CGI もサポートしています。

この節では、次のトピックについて説明します。

- サーブレット
- JavaServer Pages
- SHTML
- CGI

サーブレット

Java サーブレットはサーバーサイドの Java プログラムであり、クライアントの要求に応じてコンテンツを生成するためにアプリケーションサーバーで実行できます。サーブレットは、ユーザーインタフェースを使わずにサーバーサイドで実行されるアプレットと考えることもできます。サーブレットは URL を通して呼び出されるか、ほかのサーブレットから呼び出されます。

Sun ONE Application Server 7 は Java サーブレット仕様のバージョン 2.3 をサポートしています。

注	サーブレット API のバージョン 2.3 は、バージョン 2.1 および 2.2 との完全な下位互換性を実現しています。このため、既存のすべてのサーブレットは変更や再コンパイルなしで機能します。

サーブレットの開発には Sun Microsystems の Java サーブレット API を使用します。Java サーブレット API の使用法については、次の Web サイトで Sun Microsystems が提供しているマニュアルを参照してください。

`http://java.sun.com/products/servlet/index.html`

Sun ONE Application Server でのサーブレットの開発については、第 2 章「サーブレットの使用法」を参照してください。

JavaServer Pages

Sun ONE Application Server 7 は JavaServer Pages (JSP) 仕様のバージョン 1.2 をサポートしています。

JSP は HTML ページによく似たページであり、Web ブラウザで表示できます。ただし、HTML タグのほかに、一式の JSP タグやディレクティブに Java コードを組み合わせて使用できるため、動的コンテンツを組み込んだ Web ページを設計できます。これらの追加機能により、プロパティ値を表示したり、簡単な条件式を使用したりできます。

JSP の利点の 1 つは、HTML ページと類似していることです。HTML タグと JSP タグを使って簡単に Web ページを設計し、アプリケーションサーバーに置くことができます。ページは配備されるときに自動的にコンパイルされます。Web ページの設計に必要な Java クラスと Java コンパイラについての知識は最小限で済みます。

ただし、Sun ONE Application Server は JSP を事前にコンパイルすることをサポートしており、本稼動サーバー用には事前にコンパイルを行うことをお勧めします。

JSP ページでは、次の方法ですべての Java 機能を利用することができます。

- ページ内でスクリプトレットに直接 Java コードを埋め込む
- JavaBeans を利用する
- Java サーブレットが含まれているサーバーサイドタグを使用する

Beans もサーブレットもコンパイルを必要とする Java クラスですが、Java プログラマがその定義とコンパイルを行い、Bean やサーブレットへのインタフェースを公開できます。JSP ページからコンパイル済みの Bean やサーブレットを利用して Web ページを設計できます。

Sun ONE Application Server 7 は JSP タグライブラリおよび移植可能な標準タグをサポートしています。

JSP の作成方法については、Sun Microsystems の JavaServer Page に関する次の Web サイトを参照してください。

`http://java.sun.com/products/jsp/index.html`

Java Beans については、Sun Microsystems の JavaBeans に関する次の Web サイトを参照してください。

`http://java.sun.com/beans/index.html`

Sun ONE Application Server での JSP の開発については、第 3 章「JavaServer Pages の使用法」を参照してください。

SHTML

HTML ファイルには、サーバーで実行されるタグを含めることができます。SSI などの標準サーバーサイドタグをサポートするほかに、Sun ONE Application Server 7 は、サーブレットを埋め込んだり、自身のサーバーサイドタグを定義したりすることができます。詳細については、第 7 章「サーバーでパースされる HTML の使用法」を参照してください。

CGI

CGI (Common Gateway Interface) プログラムは、サーバー上で動作し、クライアントの要求に対する応答を生成します。CGI プログラムは、シェルスクリプトとして、C、C++、Java、Perl などさまざまな言語で記述されます。CGI プログラムは、URL を呼び出すことにより、起動します。Sun ONE Application Server は、CGI バージョン 1.1 仕様に準拠しています。詳細については、第 8 章「CGI の使用法」を参照してください。

Web アプリケーションの作成

Web アプリケーションを作成するには、次の手順を実行します。

1. Web アプリケーションファイル全体用のディレクトリを作成します。これは、Web アプリケーションのドキュメントルートになります。
2. 必要な HTML ファイル、イメージファイルおよびその他の静的コンテンツを作成します。これらのファイルをドキュメントルートディレクトリまたはサブディレクトリに置き、ほかのアプリケーション部分からアクセスできるようにします。
3. 必要な JSP ファイルを作成します。詳細については、第 3 章「JavaServer Pages の使用法」を参照してください。
4. 必要なサーブレットを作成します。詳細については、第 2 章「サーブレットの使用法」を参照してください。
5. サーブレットをコンパイルします。JSP のプリコンパイルについては、62 ページの「JSP のコンパイル : コマンド行コンパイラ」を参照してください。
6. Web アプリケーションを編成します。詳細については、87 ページの「Web アプリケーションの構造」を参照してください。
7. 配備記述子ファイルを作成します。詳細については、第 6 章「Web モジュールの構築と配備」を参照してください。

8. 必要であれば、WAR ファイルに Web アプリケーションをパッケージ化します。これはオプションです。次に例を示します。

   ```
   jar -cvf module_name.war *
   ```

9. Web アプリケーションを配備します。詳細については、第 6 章「Web モジュールの構築と配備」を参照してください。

Web アプリケーションは手動で、または Sun ONE Studio 4 を使って作成できます。Sun ONE Studio の詳細については、Sun ONE Studio 4, Enterprise Edition のチュートリアルを参照してください。

Web アプリケーションの配備

Web アプリケーションの配備記述子ファイルは、配備時に Sun ONE Application Server の管理インタフェースで作成されます。また、手作業で作成することもできます。これらの記述子ファイルは、Web アプリケーション のアーカイブ (`.war`) ファイル内にパッケージ化されています。アーカイブファイルには、メタデータとともに、サーブレットや JSP を識別しそのアプリケーションのロールを確立する情報が含まれています。これらの記述子ファイルの詳細については、第 6 章「Web モジュールの構築と配備」を参照してください。

Web アプリケーションのデバッグ

アプリケーションのデバッグについては、『Sun ONE Application Server 開発者ガイド』を参照してください。

国際化に関する問題

ここでは、次の対象に適用される国際化について説明します。

- サーバー
- サーブレット
- JSP

サーバー

Sun ONE Application Server 全体のデフォルトロケールを設定するには、次のどちらかの手順を実行します。デフォルトロケールにより、管理インタフェースやログなどのロケールが決まります。

- 管理インタフェースのサーバーインスタンスページに移動し、「詳細」タブをクリックします。「ロケール」フィールドに値を入力し、「保存」ボタンをクリックします。次に「一般」タブをクリックし、「変更の適用」ボタンを選択します。
- `server.xml` ファイルで `server` 要素の `locale` 属性を設定し、サーバーを再起動します。このファイルの詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

サーブレット

ここでは、サーブレット要求やサーブレット応答の文字エンコードを Sun ONE Application Server がどのように判断するかについて説明します。

使用できるエンコードについては、次の Web サイトを参照してください。

`http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html`

サーブレット要求

サーバーは、サーブレット要求を処理するとき、次の優先順位に従って要求の文字エンコードを判断します。

- `setCharacterEncoding()` メソッド
- `sun-web.xml` ファイルの `parameter-encoding` 要素で指定されている、フォーム内の隠しフィールド。詳細については、128 ページの「parameter-encoding」を参照
- `sun-web.xml` ファイルの `locale-charset-info` 要素で設定されている文字エンコード。この要素の詳細については、126 ページの「国際化に関する要素」を参照

- デフォルトの ISO-8859-1

サーブレット応答

サーバーは、サーブレット応答を処理するとき、次の優先順位に従って応答の文字エンコードを判断します。

- setContentType() メソッド
- setLocale() メソッド
- デフォルトの ISO-8859-1

JSP

使用できるエンコードについては、次の Web サイトを参照してください。

http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html

JSP の文字エンコードを設定するには、page 指令を使用します。次に例を示します。

```
<%@ page contentType="text/html; charset=SJIS" %>
```

contentType 属性は、次の項目を定義します。

- JSP ページの文字エンコード
- JSP ページ応答の文字エンコード
- JSP ページ応答の MIME タイプ

デフォルト値は text/html; charset=ISO-8859-1 です。

サーバーは、JSP ページを処理するとき、次の優先順位に従って文字エンコードを判断します。

- JSP ファイルの page 指令と contentType 属性
- デフォルトの ISO-8859-1

要求時の入力に応じて、異なるコンテンツ形式と文字エンコードでコンテンツを配信できるように設計されている JSP ページもあります。コンテンツ形式の動的設定は、response.setContentType() の呼び出しに依存しています。応答ストリームにコンテンツが送られていなければ、いつでもこのメソッドを呼び出すことができます。

仮想サーバー

仮想サーバー (仮想ホスト) は、特定の URL 用にコンテンツを提供する仮想 Web サーバーです。複数の仮想サーバーが、同一または異なったホスト名、ポート番号、IP アドレスなどを使ってコンテンツを提供できます。HTTP サービスは、URL に従って、受信する Web 要求を異なった仮想サーバーに送信できます。

Sun ONE Application Server の最初のインストール時に、デフォルトの仮想サーバーが作成されます。新しく作成した HTTP リスナーごとにデフォルトの仮想サーバーを割り当てることもできます。詳細については、『Sun ONE Application Server 管理者ガイド』を参照してください。

仮想サーバーには、Web アプリケーションや、Web コンポーネントを含む J2EE アプリケーションを割り当てることができます。仮想サーバーは、次のどちらかの方法で割り当てることができます。

- 管理インタフェースの使用
- `server.xml` ファイルの編集

管理インタフェースの使用

90 ページの「Web アプリケーションの配備」の手順に従って、配備時に Web モジュールに仮想サーバーを割り当てることができます。

管理インタフェースを使って仮想サーバーのデフォルトの Web モジュールを設定するには、次の手順を実行します。

1. Web アプリケーションまたは J2EE アプリケーションを配備します。詳細については、90 ページの「Web アプリケーションの配備」を参照してください。
2. サーバーインスタンスの下の HTTP Server コンポーネントを開きます。
3. HTTP Server コンポーネントの下にある Virtual Servers コンポーネントを開きます。
4. Web アプリケーションを割り当てる仮想サーバーを選択します。
5. 「デフォルト Web モジュール」ドロップダウンリストから Web モジュールを選択します。
6. 「Save」ボタンを選択します。
7. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

詳細については、26 ページの「デフォルトの Web モジュール」を参照してください。

server.xml ファイルの編集

Web モジュールをアプリケーションの一部として配備する場合は、配備時にそれを表す `j2ee-application` 要素が `server.xml` 内に作成されます。Web モジュールを個別モジュールとして配備する場合は、配備時にそれを表す `web-module` 要素が `server.xml` 内に作成されます。`j2ee-application` 要素と `web-module` 要素はどちらも `virtual-servers` 属性を持っています。この属性は仮想サーバー ID のリストを指定します。デフォルトでは、`virtual-servers` 属性は空です。つまり、Web アプリケーションはすべての仮想サーバーに割り当てられます。

`server.xml` 内の `virtual-server` 要素はそれぞれ `default-web-module` 属性を持っています。この属性を使って、各仮想サーバーにデフォルトの Web モジュールを設定できます。デフォルトの仮想サーバーのデフォルトの Web モジュールは、インストール時に指定されます。詳細については、26 ページの「デフォルトの Web モジュール」を参照してください。

`server.xml` と仮想サーバーの詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

デフォルトの Web モジュール

デフォルトの仮想サーバーと、新しく作成した個々の仮想サーバーに、デフォルト Web モジュールを割り当てることができます。詳細については、25 ページの「仮想サーバー」を参照してください。仮想サーバーのデフォルトの Web モジュールにアクセスするには、ブラウザで仮想サーバーの URL を指定します。コンテキストルートは指定しません。次に例を示します。

```
http://myvserver:3184
```

仮想サーバーにデフォルト Web モジュールを割り当てない場合、仮想サーバーは HTML または JSP コンテンツをドキュメントルート (通常 *instance_dir*`/docroot`) から提供します。この HTML または JSP コンテンツにアクセスするには、ブラウザで仮想サーバーの URL を指定します。コンテキストルートではなくターゲットファイルを指定してください。次に例を示します。

```
http://myvserver:3184/hellothere.jsp
```

サーブレットと JSP のキャッシュ

Sun ONE Application Server には、同じサーブレットや JSP への呼び出しをすばやく実行するために、サーブレットや JSP の結果をキャッシュする機能があります。Sun ONE Application Server は、要求の結果を一定の期間キャッシュします。そのため、ほかのデータ呼び出しがあったときに、Sun ONE Application Server は、再びオペレーションを実行する代わりに、キャッシュしておいたデータを返します。たとえば、5 分ごとに更新される株式相場をサーブレットが返す場合、キャッシュが 300 秒後に期限切れになるように設定します。

サーブレット関連の応答キャッシュの詳細については、46 ページの「サーブレットの結果のキャッシュ」を参照してください。JSP のキャッシュの詳細については、58 ページの「JSP キャッシュ」を参照してください。

データベース接続プール

データベース接続プールにより、サーブレットや JSP とデータベースとの対話のパフォーマンスが向上します。接続プールをサポートしている JDBC 2.0 互換のドライバとしては、PointBase (Sun ONE Application Server に付属。ただし Solaris 9 バンドル版のインストールを除く)、Oracle 8*i* アップデート、CloudScape 3.0 などいくつかあります。JDBC の詳細については、『Sun ONE Application Server Developer's Guide to J2EE Features and Services』を参照してください。

Web コンテナの設定

Web コンテナでサーバー全体のログを記録するには、次のどちらかの方法で設定できます。

- 管理インタフェースの使用。詳細については、『Sun ONE Application Server 管理者ガイド』を参照
- `server.xml` ファイルの編集。詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照

Web アプリケーションの例

Sun ONE Application Server には、参照および配備可能なサンプルアプリケーションが付属しています。これらは、*install_dir*/samples/webapps ディレクトリ内に置かれています。各サンプルには専用のマニュアルが用意されています。

第 2 章

サーブレットの使用法

この章では、Sun ONE Application Server 上で行われるアプリケーション対話の制御に有効なサーブレット (標準のサーブレットを含む) の作成方法について説明します。また、この章では標準を拡張するための Sun ONE Application Server の機能についても説明します。

この章には次の節があります。

- サーブレットについて
- サーブレットの作成
- サーブレットの呼び出し
- サーブレットの出力
- サーブレットの結果のキャッシュ
- サーブレットエンジンについて

サーブレットについて

サーブレットはアプレットと同様に再利用可能な Java アプリケーションです。ただし、サーブレットは Web ブラウザ上ではなく、アプリケーションサーバーまたは Web サーバー上で動作します。

Sun ONE Application Server がサポートしているサーブレットは、Java サーブレット仕様バージョン 2.3 に基づいています。関連するすべての仕様書へは *install_dir*`/docs/index.htm` からアクセス可能です。*install_dir* は、Sun ONE Application Server がインストールされているディレクトリです。

サーブレットはアプリケーションのプレゼンテーションロジックに使われます。サーブレットは、フォーム入力処理、EJB にカプセル化されているビジネスロジックのコンポーネントの起動、JSP を使った Web ページ出力のフォーマットなど、アプリケーションのセントラルディスパッチャとして機能します。サーブレットはユーザーからの要求に応じてコンテンツを生成し、ユーザー対話から次のユーザー対話に続くアプリケーションフローを制御します。

サーブレットの基本的な特徴は次のとおりです。

- サーブレットは、Sun ONE Application Server のサーブレットエンジンによって実行時に作成され、管理される
- サーブレットは、`request` オブジェクトにカプセル化されている入力データを処理する
- サーブレットは、`response` オブジェクトにカプセル化されているクエリデータに応答する
- サーブレットは EJB コンポーネントを呼び出してビジネスロジック機能を実行する
- サーブレットは JSP を呼び出してページレイアウト機能を実行する
- サーブレットは拡張可能であり、Sun ONE Application Server とともに提供される API を使って機能を追加する
- サーブレットは対話中のユーザーセッション情報の持続性を提供する
- サーブレットは、アプリケーションの一部にすることも、複数のアプリケーションで使えるように分離してアプリケーションサーバー上に置くこともできる
- サーブレットはサーバーの動作中に動的に再読み込みされる
- サーブレットは、URL でアドレスを指定できる。アプリケーションのページ上のボタンがサーブレットを指している場合がある
- サーブレットはほかのサーブレットを呼び出すこともできる

サーブレットのデータフロー

ユーザーが「送信」ボタンをクリックすると、表示ページに入力された情報がサーブレットに送信されます。サーブレットは受信データを処理し、EJB などのビジネスロジックコンポーネントに基づいてコンテンツを生成することにより、応答を編成します。コンテンツが作成されると、サーブレットは通常、そのコンテンツを JSP に転送して応答ページを作成します。応答はクライアントに返送され、次のユーザー対話を設定します。

次の図に、サーブレットとの間でやり取りされる情報のフローを示します。

1. サーブレットがクライアントの要求を処理する

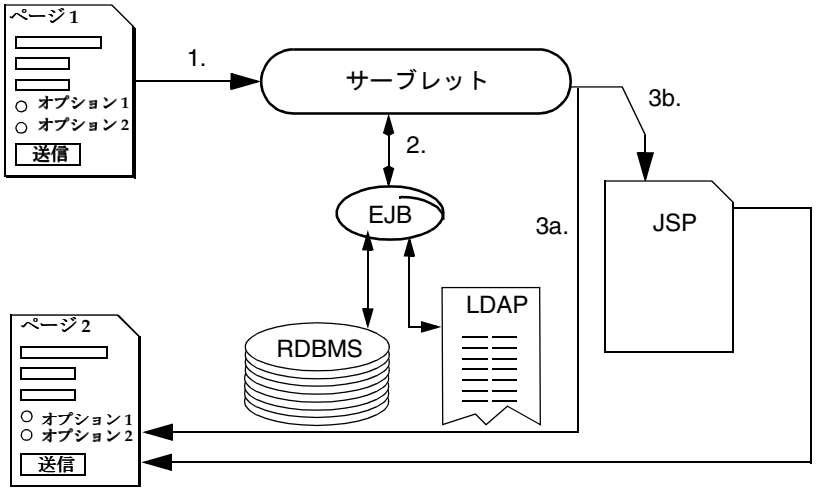
2. サーブレットがコンテンツを生成する
3. サーブレットが応答を作成し、
   a. それをクライアントに返送する

      または

   b. そのタスクを JSP にディスパッチする

サーブレットはほかの要求を処理できるようにメモリーに残ります。

サーブレットのデータフロー



サーブレットの種類

サーブレットには主に次の 2 種類があります。

- 汎用サーブレット
  - `javax.servlet.GenericServlet` を拡張する
  - プロトコルに依存しない。継承 HTTP のサポートやほかのトランスポートプロトコルは含まれていない
- HTTP サーブレット
  - `javax.servlet.HttpServlet` を拡張する

- Sun ONE Application Server 環境では、組み込み HTTP プロトコルがサポートされ、より便利になった

これらのサーブレットはそれぞれ、リソースの初期化や割り当て解除を行うコンストラクタメソッド `init()` とデストラクタメソッド `destroy()` を実装しています。

すべてのサーブレットが、サーブレットに対する要求を処理する `service()` メソッドを実装する必要があります。汎用サーブレットの場合は、サービスメソッドをオーバーライドして要求処理用のルーチンを用意します。HTTP サーブレットには、使用中の HTTP 転送メソッドに基づいてサーブレット内の別のメソッドに自動的に要求を転送するサービスメソッドがあります。したがって、HTTP サーブレットの場合は `doPost()` をオーバーライドして `POST` 要求を処理し、`doGet()` をオーバーライドして `GET` 要求を処理します。

サーブレットの作成

サーブレットを作成するには、次の手順を実行します。

- サーブレットをアプリケーションに組み込むように設計します。つまり、サーブレットに汎用的な方法でアクセスされた場合、サーブレットからアプリケーションデータにアクセスできないように設計します。
- `GenericServlet` または `HttpServlet` を拡張するクラスを作成し、要求を処理する適切なメソッドをオーバーライドします。
- Sun ONE Application Server 管理インタフェースを使って、Web アプリケーションの配備記述子を作成します。詳細については、第 6 章「Web モジュールの構築と配備」を参照してください。

この節の残りの部分では次のトピックについて説明します。

- クラス宣言の作成
- メソッドのオーバーライド
- パラメータのアクセスとデータの保存
- セッションとセキュリティの処理
- ビジネスロジックコンポーネントのアクセス
- スレッドの処理
- クライアントへの結果の配信

クラス宣言の作成

サーブレットを作成するには、基本的な入出力サポートと javax.servlet パッケージを持つ共有 Java クラスを記述します。このクラスは、GenericServlet または HttpServlet を拡張する必要があります。Sun ONE Application Server サーブレットは HTTP 環境に存在するので、HttpServlet の拡張をお勧めします。サーブレットがパッケージに含まれる場合は、クラスローダーがそのサーブレットを正しく配置できるようにパッケージ名も宣言する必要があります。

次のヘッダーの例は、myServlet という HTTP サーブレットの宣言を示しています。

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

public class myServlet extends HttpServlet {
    ...servlet methods...
}
```

メソッドのオーバーライド

次に、1 つまたは複数のメソッドをオーバーライドして、意図したタスクを実行するための指示をサーブレットに与えます。サーブレットによるすべての処理は、要求ごとに、サービスメソッド (汎用サーブレットの service() メソッドまたは HTTP サーブレットの do*Operation*() メソッドのどれか) で行われます。このメソッドは受信要求を受け取り、与えられた指示に従って要求を処理し、出力を適切に送信します。同じように、このサーブレットのほかのメソッドも作成できます。

ビジネスロジックには、トランザクションを実行するためのデータベースアクセスや EJB コンポーネントへの要求送信を含めることができます。

初期化メソッドのオーバーライド

カウンタなどのサーブレットインスタンスの生存期間にわたって使用するリソースを初期化したり割り当てたりするには、クラスを初期化する init() をオーバーライドします。init() メソッドは、サーブレットがインスタンス化されてから要求を受け入れるまでの間に実行されます。詳細については、サーブレットの API 仕様書を参照してください。

注 範囲を設定するために、すべての init() メソッドで super.init(ServletConfig) を呼び出す必要があります。これにより、このサーブレットの設定オブジェクトをほかのサーブレットメソッドでも使用できるようになります。この呼び出しを省略すると、サーブレットの呼び出し時に「500 SC_INTERNAL_SERVER_ERROR」がブラウザに表示されます。

注 初期化中に、フィルタなどの Web アプリケーションのコンポーネントが ServletException をスローすると、この Web アプリケーションは起動しません。つまり、Web アプリケーションの一部が実行されると、Web アプリケーション全体が実行されるということです。セキュリティコンポーネントに障害が発生した場合、Web アプリケーションは実行されないことが重要です。

次の init() メソッドの例では、thisMany という共有整数変数を作成することによってカウンタを初期化します。

```
public class myServlet extends HttpServlet {
   int thisMany;

   public void init (ServletConfig config) throws ServletException
   {
       super.init(config);
       thisMany = 0;
   }
}
```

これで、ほかのサーブレットメソッドがこの変数にアクセスできるようになります。

破棄メソッドのオーバーライド

ログメッセージを書き込んだり、サーブレットの生存期間内に使用されていたリソースを解放したりするには、クラスのデストラクタ destroy() をオーバーライドします。リソースの再利用やガベージコレクションを行うためには、リソースを適切に終了し、参照を解除する必要があります。destroy() メソッドは、サーブレット自体がメモリーの割り当てから解放される直前に実行されます。詳細については、サーブレットの API 仕様書を参照してください。

destroy() メソッドでは、たとえば上記の「初期化メソッドのオーバーライド」の例に基づいて、次のようなログメッセージを書き込むことができます。

```
out.println("myServlet was accessed " + thisMany " times.¥n");
```

Service、Get、および Post メソッドのオーバーライド

要求が発生すると、Sun ONE Application Server は受信データをサーブレットエンジンに渡して要求を処理します。要求にはフォームデータ、cookie、セッション情報、URL での名前 - 値のペアなどが含まれています。これらは、要求オブジェクトと呼ばれる `HttpServletRequest` 型のオブジェクトによって処理されます。クライアントのメタデータは、応答オブジェクトと呼ばれる `HttpServletResponse` 型のオブジェクトとしてカプセル化されます。サーブレットエンジンは、サーブレットの `service()` メソッドのパラメータとして両方のオブジェクトを渡します。

HTTP サーブレット内のデフォルトの `service()` メソッドは、HTTP 転送メソッド (POST、GET など) に基づいて要求をほかのメソッドに転送します。たとえば、HTTP POST 要求は `doPost()` メソッドに転送され、HTTP GET 要求は `doGet()` メソッドに転送されます。これにより、サーブレットは転送メソッドに応じてさまざまな要求データ処理を実行できます。要求の転送は `service()` で行われるため、一般的に HTTP サーブレットの `service()` をオーバーライドする必要はありません。その代わり、予想される要求型に従って `doGet()` や `doPost()` をオーバーライドします。

HTTP サーブレットでの自動転送は、HTTP 転送メソッドを備えている `request.getMethod()` の呼び出しに基づいています。Sun ONE Application Server では、サーブレットがデータを参照する前に、要求データが名前 - 値のリストに処理されています。このため、HTTP サーブレットの `service()` メソッドをオーバーライドしても機能は失われません。ただし、これによって前処理された要求データに依存することになるため、サーブレットの移植性は低下します。

要求に対応するために必要なタスクを実行するには、汎用サーブレットの場合は `service()` メソッドをオーバーライドし、HTTP サーブレットの場合は `doGet()` メソッドや `doPost()` メソッドをオーバーライドします。多くの場合、EJB コンポーネントにアクセスしてビジネストランザクションを実行し、要求オブジェクトまたは JDBC ResultSet のオブジェクト内で必要な情報を照合したあと、新たに作成されたコンテンツを JSP に渡してコンテンツのフォーマットとクライアントへの配信を行います。

フォームを伴う多くのオペレーションで GET または POST オペレーションが使われるため、ほとんどのサーブレットについて `doGet()` または `doPost()` をオーバーライドします。次の例のように、両方のメソッドを実装して両方の入力タイプに備えたり、要求オブジェクトを中央処理メソッドに渡したりできます。

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response}
          throws ServletException, IOException {
   doPost(request, response);
}
```

HTTP サーブレットの各要求のトラフィックはすべて、適切な do*Operation*() メソッドで処理されます。メソッドには、セッション管理、ユーザー認証、EJB コンポーネントや JSP のディスパッチ、および Sun ONE Application Server 機能へのアクセスなどがあります。

サーブレットが RequestDispatcher メソッドの include() または forward() を呼び出す場合は、要求情報が HTTP の POST や GET として転送されなくなるので注意してください。つまり、サーブレットが doPost() をオーバーライドする場合、呼び出し側のサーブレットが HTTP GET 経由でデータを受け取ると、ほかのサーブレットがこのメソッドを呼び出しても何も処理されません。このため、上記の例のように可能性のあるすべての入力タイプのルーチンを実装してください。RequestDispatcher メソッドは常に、service() を呼び出します。

詳細については、43 ページの「プログラムによるサーブレットの呼び出し」を参照してください。

注 デフォルトでは、Web コネクタは受信データを名前 - 値のペアに変換するため、アップロードされるファイルやイメージなど任意のバイナリデータが問題になることがあります。このような種類のデータを適切に処理し、要求オブジェクト内に正しくパッケージ化するように、Web コネクタをプログラムできます。

パラメータのアクセスとデータの保存

受信データは要求オブジェクトにカプセル化されます。HTTP サーブレットの場合、要求オブジェクト型は HttpServletRequest です。汎用サーブレットの場合、要求オブジェクト型は ServletRequest です。要求オブジェクトは、属性と呼ばれるユーザー独自の要求値を含む、すべての要求パラメータを持っています。

受信要求のすべてのパラメータにアクセスするには、getParameter() メソッドを使います。次に例を示します。

```
String username = request.getParameter("username");
```

要求オブジェクト内の値の設定および取得を行うには、それぞれ setAttribute() と getAttribute() を使います。次に例を示します。

```
request.setAttribute("favoriteDwarf", "Dwalin");
```

上記の例は、データを JSP に転送する方法を示しています。これは、JSP に暗黙的 Bean としての要求オブジェクトへのアクセス権があるためです。

セッションとセキュリティの処理

Web サーバーまたはアプリケーションサーバーから見ると、Web アプリケーションは関連のないサーバーヒットの連続です。数秒前に対話したばかりでも、ユーザーがそのサイトにアクセスしたことがあるかどうかの自動認識は行われません。セッションは、アプリケーションの状態を記憶することによってユーザーとの複数の対話を関連付けます。クライアントは各対話時に cookie を使って自己を特定します。また、cookie のないブラウザの場合は、URL にセッション識別子を入れることによって自己を特定します。

セッションオブジェクトは、表形式データ、アプリケーションの現在の状態についての情報、現在のユーザーについての情報などのオブジェクトを格納できます。セッションに関連付けられたオブジェクトは、同じセッションを使うほかのコンポーネントから使用できます。

詳細については、第 4 章「ユーザーセッションの作成と管理」を参照してください。

ログインに成功したら、標準オブジェクト内でユーザーの識別情報を確立するようにサーブレットに指示する必要があります。この標準オブジェクトはセッションオブジェクトと呼ばれ、ユーザーのログイン名や保存の必要がある補足情報など、現在のセッションに関する情報を保持しています。アプリケーションコンポーネントはセッションオブジェクトに対してクエリを実行し、ユーザー認証を取得できます。

アプリケーションでユーザーセッションを安全に行う方法については、第 5 章「Web アプリケーションのセキュリティ」を参照してください。

ビジネスロジックコンポーネントのアクセス

Sun ONE Application Server のプログラミングモデルでは、データベーストランザクション、ディレクトリトランザクション、複雑な計算などのビジネスロジックを EJB コンポーネントに実装します。`request` オブジェクトの参照は、指定されたタスクを実行するための EJB パラメータとして渡すことができます。

データベーストランザクションからの結果を JDBC `ResultSet` オブジェクトに格納し、フォーマットとクライアントへの配信を行うためにコンポーネントにオブジェクト参照を渡します。また、`request.setAttribute()` メソッドを使って要求オブジェクトの結果を格納したり、`session.setAttribute()` メソッドを使ってセッションに格納したりできます。要求オブジェクトに格納されるオブジェクトは、要求が有効であるかぎり有効です。つまり、個別のサーブレットスレッドだけに有効です。セッションに格納されるオブジェクトは、セッションが持続している間は存続します。

次の例は、`ShoppingCart` という EJB コンポーネントにアクセスするサーブレットを示しています。サーブレットは、カートのリモートインタフェースをインポートしてからユーザーのセッション ID をカートに割り当てることによって、カートのハンドルを作成します。カートはユーザーのセッション内に格納されます。

```
import cart.ShoppingCart;

   // ユーザーのセッションおよびショッピングカートを取得します。
   HttpSession session = request.getSession(true);
   ShoppingCart cart =
       (ShoppingCart)session.getAttribute(session.getId());

   // ユーザーがカートを持っていない場合は新規に作成します。
   if (cart == null) {
      String jndiNm = "java:comp/env/ejb/ShoppingCart";
      javax.naming.Context initCtx = null;
      Object home = null;
      try {
          initCtx = new javax.naming.InitialContext(env);
          java.util.Properties props = null;
          home = initCtx.lookup(jndiNm);
          cart = ((IShoppingCartHome) home).create();
          session.setValue(session.getId(),cart);
       }
       catch (Exception ex) {
       .....
       .....
       }
   }
```

Java Naming Directory Interface (JNDI) を使ってサーブレットから EJB コンポーネントにアクセスし、EJB コンポーネントへのハンドルまたはプロキシを確立します。次に、正規オブジェクトとして EJB を参照します。このとき、オーバーヘッドは Bean のコンテナによって管理されます。

次の例は、ショッピングカートのプロキシを検索する JNDI を示しています。

```
String jndiNm = "java:comp/env/ejb/ShoppingCart";
javax.naming.Context initCtx;
Object home;
   try
   {
       initCtx = new javax.naming.InitialContext(env);
   }
   catch (Exception ex)
   {
       return null;
   }
   try
   {
       java.util.Properties props = null;
       home = initCtx.lookup(jndiNm);
   }
```

```
    catch(javax.naming.NameNotFoundException e)
    {
        return null;
    }
    catch(javax.naming.NamingException e)
    {
        return null;
    }
    try
    {
        IShoppingCart cart = ((IShoppingCartHome) home).create();

}
catch (...) {...}
```

EJB コンポーネントの詳細については、『Sun ONE Application Server Enterprise JavaBeans 開発者ガイド』を参照してください。

注 JNDI 内のその他のエンタープライズリソース名との衝突や移植性の問題を回避するため、Sun ONE Application Server アプリケーション内のすべての名前を、文字列 `java:comp/env` で始める必要があります。

スレッドの処理

デフォルトでは、サーブレットはスレッドセーフになっていません。通常、1 つのサーブレットインスタンス内のメソッドは、使用可能なメモリーの範囲内で同時に何回も実行されます。メソッドの実行は、それぞれ別のスレッドで行われますが、サーブレットエンジンにはサーブレットのコピーが 1 つしか存在しません。

これによってシステムリソースの使用率が向上しますが、Java でのメモリー管理方法に起因する危険性があります。サーブレットクラスに属する変数は参照によって渡されるため、別のスレッドが同じメモリー空間を上書きしてしまうことがあります。サーブレット (またはサーブレット内のブロック) をスレッドセーフにするには、次のどちらかを行います。

- `public synchronized void` *method*`()` (メソッド全体) または `synchronized(this) {...}` (ブロックのみ) のように、すべてのインスタンス変数に対する書き込みアクセスの同期をとります。同期をとることによって応答時間が大幅に遅くなるため、ブロックだけの同期をとるか、またはサーブレット内のブロックで同期が不要かを確認してください。

  たとえば、次のサーブレットには `doGet()` 内にスレッドセーフのブロックがあり、さらに `mySafeMethod()` というスレッドセーフのメソッドがあります。

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

public class myServlet extends HttpServlet {

public void doGet (HttpServletRequest request,
                   HttpServletResponse response}
          throws ServletException, IOException {
    //前処理
    synchronized (this) {
       // このブロック内のコードはスレッドセーフです。
    }
    //前処理
    }

public synchronized int mySafeMethod (HttpServletRequest request)
    {
    // このメソッド内で行われる処理はすべてスレッドセーフです。
    }
}
```

- `SingleThreadModel` クラスを使ってシングルスレッドのサーブレットを作成します。シングルスレッドのサーブレットを Sun ONE Application Server に配備すると、サーブレットエンジンは受信要求に備えてサーブレットインスタンスをプールします。つまり、同じサーブレットの複数のコピーをメモリー内に用意します。このプール内のサーブレットインスタンスの数は、Sun ONE Application Server 固有の Web アプリケーション配備記述子の `singleThreadedServletPoolSize` プロパティを設定することによって変更できます。Sun ONE Application Server の Web アプリケーション配備記述子の詳細については、第 6 章「Web モジュールの構築と配備」を参照してください。シングルスレッドのサーブレットは、新規要求を処理するためにインスタンスの空きを待つ必要があるため、その負荷によって動作が遅くなります。ただし、ロードバランスが有効な分散アプリケーションでは、ビジーでないプロセスに負荷が自動的に移行するため、これが問題になることはありません。

  たとえば、次のサーブレットは完全なシングルスレッドです。

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

public class myServlet extends HttpServlet
   implements SingleThreadModel {
   servlet methods . . .
}
```

クライアントへの結果の配信

ユーザーとの対話の最終作業はクライアントに応答ページを配信することです。応答ページは次の 2 つの方法で配信できます。

- サーブレット応答ページの作成
- JSP 応答ページの作成

サーブレット応答ページの作成

出力ストリームに書き込むことによって、サーブレット内で出力ページを生成します。出力タイプによって推奨方法は異なります。

すべての出力を開始する前に、`setContentType()` を使って出力の MIME タイプを必ず指定します。次に例を示します。

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

単純な HTML などのテキスト出力の場合は、`PrintWriter` オブジェクトを作成してから `println` を使って書き込みます。次に例を示します。

```
PrintWriter output = response.getWriter();
output.println("Hello, World¥n");
```

バイナリ出力の場合は、`ServletOutputStream` オブジェクトを作成してから `print()` を使って書き込むことによって、出力ストリームに直接書き込みます。次に例を示します。

```
ServletOutputStream output = response.getOutputStream();
output.print(binary_data);
```

JSP 応答ページの作成

サーブレットは次の 2 つの方法で JSP を呼び出せます。

- `RequestDispatcher` インタフェースの `include()` メソッドは、JSP を呼び出し、JSP が返されてから対話処理を続行します。`include()` メソッドは、特定のサーブレット内で複数回呼び出すことができます。

  次の例は、`include()` を使った JSP を示しています。

  ```
  RequestDispatcher dispatcher =
     getServletContext().getRequestDispatcher("JSP_URI");
  dispatcher.include(request, response);
  ... // 処理の継続
  ```

- `RequestDispatcher` インタフェースの `forward()` メソッドは対話の制御を JSP に渡します。`forward()` を呼び出すと、サーブレットは現在の対話の出力と無関係になります。つまり、特定のサーブレットで `forward()` メソッドを呼び出すことができるのは 1 回だけです。

注	PrintWriter オブジェクトまたは ServletOutputStream オブジェクトをすでに定義している場合は、forward() メソッドを使うことができないので注意してください。

次の例は、forward() を使った JSP を示しています。

```
RequestDispatcher dispatcher =
   getServletContext().getRequestDispatcher("JSP_URI");
dispatcher.forward(request, response);
```

注	URI (Universal Resource Identifier) を指定して、呼び出す JSP を指定してください。このパスは、ServletContext の範囲内にあるパスを記述する String です。また、絶対パスを示す String 引数を取る要求オブジェクトには、getRequestDispatcher() メソッドもあります。このメソッドの詳細については、Java サーブレット仕様バージョン 2.3 の第 8 章を参照してください。

JSP の詳細については、第 3 章「JavaServer Pages の使用法」を参照してください。

サーブレットの呼び出し

サーブレットを呼び出すには、URL を使ってアプリケーションページから直接アドレスを指定するか、すでに実行しているサーブレットからプログラムで呼び出します。次の節を参照してください。

- URL によるサーブレットの呼び出し
- プログラムによるサーブレットの呼び出し

URL によるサーブレットの呼び出し

アプリケーションの HTML または JSP ページ内にリンクとして埋め込まれた URL を使って、サーブレットを呼び出すことができます。URL の形式は次のとおりです。

http://*server*:*port*/*context_root*/servlet/*servlet_name*?*name*=*value*

次の表は、URL の各セクションについて説明しています。左の列は URL 要素、右の列は各 URL 要素の説明です。

アプリケーション内のサーブレットを表す URL の各フィールド

URL 要素	説明
server:*port*	IP アドレス (またはホスト名) およびオプションのポート番号 仮想サーバーのデフォルト Web モジュールにアクセスするには、この URL セクションだけを指定する。名前 - 値のパラメータを指定する場合を除き、*context_root* や *servlet_name* の指定する必要はない
context_root	アプリケーションの場合、`application.xml` ファイルか `sun-application.xml` ファイル内の `context-root` 要素にコンテキストルートが定義される。個別に配備される Web モジュールの場合、配備時にコンテキストルートを指定する
`servlet`	`web.xml` ファイルに `servlet-mapping` が定義されていない場合にのみ必要
servlet_name	`web.xml` ファイルに設定されている `servlet` の名前 (定義されている場合は `servlet-mapping`)
?*name*=*value*...	サーブレットのオプションの名前 - 値のパラメータ

この例では、sun はホスト名、`MortPages` はコンテキストルート、`calcMortgage` はサーブレット名です。

```
http://www.sun.com/MortPages/servlet/calcMortgage?rate=8.0&per=360&bal=180000
```

プログラムによるサーブレットの呼び出し

まず、URI を指定して、呼び出すサーブレットを指定します。通常、これは現在のアプリケーションに対応する相対パスになります。たとえば、サーブレットが `OfficeFrontEnd` という名前のコンテキストルートを持つアプリケーションの一部である場合、ブラウザから `ShowSupplies` という名前のサーブレットへの URL は次のようになります。

```
http://server:port/OfficeApp/OfficeFrontEnd/servlet/ShowSupplies?name=value
```

プログラムによってほかのサーブレットからこのサーブレットを呼び出すには次の 2 つの方法があります。

- ほかのサーブレットの出力を含めるには、`RequestDispatcher` インタフェースから `include()` メソッドを使います。このメソッドは、URI で サーブレットを呼び出し、サーブレットが返されてから対話処理を続行します。`include()` メソッドは、特定のサーブレット内で複数回呼び出すことができます。

  次に例を示します。

```
RequestDispatcher dispatcher =
   getServletContext().getRequestDispatcher("/ShowSupplies");
dispatcher.include(request, response);
```

- ほかのサーブレットに対話制御を渡すには、RequestDispatcher インタフェースの forward() メソッドを使用します。このとき、サーブレットの URI をパラメータとして指定します。

注	要求を転送した場合、forward() を呼び出すと、元のサーブレットは現在の対話の出力とは無関係になるので注意してください。つまり、特定のサーブレットで forward() を呼び出せるのは 1 回だけです。

次の例は、include() を使った JSP を示しています。

```
RequestDispatcher dispatcher =
   getServletContext().getRequestDispatcher("/ShowSupplies");
dispatcher.forward(request, response);
```

サーブレットの出力

ServletContext.log メッセージはサーバーログに送信されます。

デフォルトでは、サーブレットの出力 System.out および System.err はサーバーログに送られます。また、起動時にサーバーログメッセージが System.err 出力にエコーされます。またデフォルトでは、System.err 出力用の Windows コンソールはありません。これらのデフォルト設定は次の方法で変更できます。

- 管理インタフェースの使用
- server.xml ファイルの編集

管理インタフェースの使用

次の手順に従って管理インタフェースを使用します。

1. 管理インタフェースのサーバーインスタンスページで「ログ」タブをクリックします。
2. 次のチェックボックスをオンまたはオフにします。
   - 「標準出力をイベントログとして記録」- true にすると、System.out 出力がサーバーログに送られます。

- 「標準エラー出力をイベントログとして記録」- true にすると、System.err 出力がサーバーログに送られます。
- 「標準エラー出力にエコー」- true にすると、サーバーログメッセージが System.err 出力にエコーされます。
- 「コンソールを作成」- System.err 出力用の Windows コンソールを作成します。

3. 「保存」ボタンをクリックします。
4. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

詳細については、『Sun ONE Application Server 管理者ガイド』を参照してください。

server.xml ファイルの編集

server.xml ファイルを次のように編集し、サーバーを再起動します。

```
<log-service log-stdout=false
             log-stderr=false
             echo-log-messages-to-stderr=false
             create-console=true />
```

create-console は Windows 専用の属性です。server.xml の詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

サーブレットの結果のキャッシュ

Sun ONE Application Server は、サーブレット、JSP、または任意の URL パターン を呼び出した結果をキャッシュしておくことができ、その後に同じサーブレット、JSP、または URL パターンを呼び出す際、すばやく呼び出すことができます。Sun ONE Application Server は、要求の結果を一定の期間キャッシュします。そのため、ほかのデータ呼び出しがあったときに、Sun ONE Application Server は、再びオペレーションを実行する代わりに、キャッシュしておいたデータを返します。たとえば、5 分ごとに更新される株式相場をサーブレットが返す場合、キャッシュが 300 秒後に期限切れになるように設定します。

結果をキャッシュするかどうかや、キャッシュする方法は、結果に含まれているデータによって異なります。たとえば、クイズの投稿の結果は、サーブレットへの入力が毎回異なるため、キャッシュしても意味がありません。ただし、1 時間ごとに更新される、クイズの結果から収集した人口統計データを示す、高度なレポートをキャッシュすることは考えられます。

`sun-web.xml` ファイル内で特定のフィールドを編集することによって、Sun ONE Application Server の Web アプリケーションで応答キャッシュを処理する方法を定義できます。このように、有効な Sun ONE Application Server 機能を利用した標準サーブレットをプログラムで作成できます。

JSP のキャッシュの詳細については、58 ページの「JSP キャッシュ」を参照してください。

注 静的ファイルコンテンツのキャッシュについては、『Sun ONE Application Server 管理者用設定ファイルリファレンス』で説明されている `nsfc.conf` ファイルを参照してください。

この節の残りの部分では次のトピックについて説明します。

- キャッシュ機能
- デフォルトのキャッシュ設定
- キャッシュの例
- CacheHelper インタフェース
- CacheKeyGenerator インタフェース

キャッシュ機能

Sun ONE Application Server 7 には、次の Web アプリケーション応答キャッシュ機能があります。

- キャッシュはサーブレットの名前または URI に基づいて設定可能
- URI に基づくキャッシュの場合は、クエリ文字列にユーザーが指定したパラメータは URI に含まれる。たとえば、`/garden/catalog?category=roses` からの応答と `/garden/catalog?category=lilies` からの応答とは異なる。これらの応答は異なるキーでキャッシュに格納される
- キャッシュサイズ、エントリのタイムアウトなどのキャッシュ動作は設定可能
- エントリのタイムアウトは、エントリの作成または更新が行われた時点から測定される。個々のキャッシュ割り当てのタイムアウト値は、`cache-mapping` サブ要素を `timeout` に指定することによって上書き可能
- キャッシュヘルパークラスを作成することで、キャッシュ条件をプログラムで判断できる。たとえば、バックエンドのデータソースの最終更新時刻に関する情報だけをサーブレットが持っている場合は、最終更新のタイムスタンプをデータソースから取得し、応答をキャッシュするかどうかをそのタイムスタンプに基づいて判断するヘルパークラスを作成できる。49 ページの「CacheHelper インタフェース」を参照
- キャッシュキージェネレータクラスを作成することで、キャッシュキーの生成条件をプログラムで判断できる。51 ページの「CacheKeyGenerator インタフェース」を参照
- キャッシュキー要素で指定された、ASCII 以外の要求パラメータ値はすべて、URL でエンコードされる必要がある。キャッシュのサブシステムは、要求のクエリ文字列にある生パラメータ値に一致するよう試みる
- ただし、クラスが新たに更新されるとキャッシュされる内容にも影響を与えるため、クラスの動的配備や動的再読み込みを行う際、Web コンテナはキャッシュをクリアする
- `HttpServletRequest` 要求の次の属性は公開される
  - `com.sun.appserv.web.cachedServletName` (キャッシュ対象となるサーブレット)
  - `com.sun.appserv.web.cachedURLPattern` (キャッシュ対象となる URL パターン)

デフォルトのキャッシュ設定

サーブレットや JSP のキャッシュ機能を有効にし、特別な設定を行わない場合、デフォルトのキャッシュの設定は次のようになります。

- デフォルトのキャッシュタイムアウトは 30 秒
- HTTP GET メソッドだけをキャッシュに使用できる
- cookie やセッションを使った HTTP 要求があると、キャッシュ機能は自動的に無効になる
- `Pragma:`、`Cache-control:`、および `Vary:` の各ヘッダーは特に考慮されない
- デフォルトキーはサーブレットパス (`pathInfo` とクエリ文字列を除いたもの) で構成される
- 最近使用されたキャッシュエントリのリストが保持されているため、最大キャッシュサイズを超えた場合に古いキャッシュエントリを削除できる
- サーブレットのパスとキーのフィールド値 (指定されている場合) を連結してキーが生成される

キャッシュの例

`sun-web.xml` ファイルの cache 要素の例を次に示します。

```
<cache max-capacity="8192" timeout="60">
   <cache-helper name="myHelper" class-name="MyCacheHelper"/>
   <cache-mapping>
      <servlet-name>myservlet</servlet name
      <timeout name="timefield">120</timeout>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
   </cache-mapping>
   <cache-mapping>
      <url-pattern> /catalog/* </url-pattern>
      <!-- 最もよく売れているカテゴリをキャッシュします。
        -- 特定のパラメータが存在するときだけ、このリソースへの応答をキャッシュします。
        -- catalog パラメータが 'lilies' または 'roses' の場合だけキャッシュし、
        -- ほかの値のときはキャッシュしません。つまり、
        -- /orchard/catalog?best&category='lilies'
        -- /orchard/catalog?best&category='roses'
        -- の結果はキャッシュしますが、次の結果はキャッシュしません。
        -- /orchard/catalog?best&category='wild'
      -->
      <constraint-field name='best' scope='request.parameter'/>
      <constraint-field name='category' scope='request.parameter'>
```

```
         <value> roses </value>
         <value> lilies </value>
      </constraint-field>
      <!-- 特定のフィールドの範囲を指定します。ただし、このフィールドはすべての要求に
        -- 含まれていなくてもかまいません。>
      <constraint-field name='SKUnum' scope='request.parameter'>
         <value match-expr='in-range'> 1000 - 2000 </value>
      </constraint-field>
      <!-- 特定の値以外のいずれかの値と category が一致する場合だけキャッシュします。>
      <constraint-field name="category" scope="request.parameter>
         <value match-expr="equals" cache-on-match-failure="true">bogus</value>
      </constraint-field>
   </cache-mapping>
   <cache-mapping>
      <servlet-name> InfoServlet </servlet name>
      <cache-helper-ref>myHelper</cache-helper-ref>
   </cache-mapping>
</cache>
```

sun-web.xml のキャッシュ設定の詳細については、113 ページの「キャッシュに関する要素」を参照してください。

CacheHelper インタフェース

CacheHelper インタフェースは次のとおりです。

```
package com.sun.appserv.web.cache;

import java.util.Map

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;

/** CacheHelper インタフェースはユーザーによる拡張が可能なインタフェースであり、
 *  応答をキャッシュするかどうかと、キー生成をカスタマイズできます。
 */
public interface CacheHelper {

   // 要求の属性の名前
   public static final String ATTR_CACHE_MAPPED_SERVLET_NAME =
                              "com.sun.appserv.web.cachedServletName";
   public static final String ATTR_CACHE_MAPPED_URL_PATTERN =
                              "com.sun.appserv.web.cachedURLPattern";

   public static final int TIMEOUT_VALUE_NOT_SET = -2;
```

```
    /** ヘルパーを初期化します。
     *  @引数 context は、このヘルパーが属している Web アプリケーションです。
     *  @例外 Exception は、起動エラーが発生した場合にスローされます。
     */
    public void init(ServletContext context, Map props) throws Exception;

    /** getCacheKey: この要求をキャッシュするためのキーを生成します。
     *  @引数 request は、受信した <code>HttpServletRequest</code> オブジェクトです。
     *  @この要求されたキャッシュ可能なリソースについて生成したキーを返します。
     */
    public String getCacheKey(HttpServletRequest request);

    /** isCacheable: 特定の要求に対する応答がキャッシュ可能かどうかを調べます。
     *  @引数 request は、受信した <code>HttpServletRequest</code> オブジェクトです。
     *  @応答がキャッシュ可能な場合は <code>true</code> を返し、
     *  キャッシュ不可能な場合は <code>false</code> を返します。
     */
    public boolean isCacheable(HttpServletRequest request);

    /** isRefreshNeeded: 特定の要求に対する応答について、更新が必要かどうかを調べます。
     *  @引数 request は、受信した <code>HttpServletRequest</code> オブジェクトです。
     *  @応答の更新が必要な場合は <code>true</code> を返し、
     *  この要求の結果を更新する必要がない場合は <code>false</code> を返します。

     */
    public boolean isRefreshNeeded(HttpServletRequest request);

    /** キャッシュされた応答のタイムアウトを取得します。
     *  @引数 request は、受信した <code>HttpServletRequest</code> オブジェクトです。
     *  @キャッシュされた応答のタイムアウトを秒単位で返します。
     *  値 -1 は、この応答が無期限に有効であることを示し、
     *  値 -2 は、ヘルパーでタイムアウトを調べられないことを示します (コンテナがデフォルトの
     *  タイムアウトを割り当てる)。
     */
    public int getTimeout(HttpServletRequest request);

    /**
     * ヘルパーの使用を中止します。
     * @例外 Exception は、エラーが発生した場合にスローされます。
     */
    public void destroy() throws Exception;
}
```

CacheKeyGenerator インタフェース

Web アプリケーションは、デフォルトで組み込まれている CacheHelper 実装を使ってキー生成をカスタマイズできます。サーブレットまたは JSP 内のアプリケーションコンポーネントは、ServletContext 内の属性としてカスタム CacheKeyGenerator 実装を設定できます。

コンテキスト属性の名前は、sun-web.xml 配備記述子の default-helper 要素内の cacheKeyGeneratorAttrName プロパティ値 value として設定できます。詳細については、116 ページの「default-helper」を参照してください。

CacheKeyGenerator インタフェースは次のとおりです。

```
package com.sun.appserv.web.cache;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;

/** CacheKeyGenerator: この要求をキャッシュするためのキーを生成するヘルパー
 *  インタフェース
 *
 * CacheKeyGenerator を実装する ServletContext 属性の名前は、
 * sun-web.xml 内の default-helper プロパティを使って設定できます。

 * <default-helper>
 *   <property
 *     name="cacheKeyGeneratorAttrName"
 *     value="com.acme.web.MyCacheKeyGenerator" />
 * </default-helper>
 *
 * キャッシュエンジンは、指定された属性をサーブレットコンテキスト内で検索します。
 * 検索結果は CacheKeyGenerator インタフェースの実装であることが必要です。

 */

public interface CacheKeyGenerator {

   /** getCacheKey: この応答をキャッシュするためのキーを生成します。
    *  @引数 context は、この Web アプリケーションのコンテキストです。
    *  @引数 request は、受信した <code>HttpServletRequest</code>
    *  オブジェクトです。
    *  @キャッシュエントリにアクセスするためのキー文字列を返します。
    *  null が返された場合は、デフォルトのキーが使用されます。
    */
   public String getCacheKey(ServletContext context,
                             HttpServletRequest request);
}
```

サーブレットエンジンについて

サーブレットは Sun ONE Application Server のサーブレットエンジン内に存在し、このエンジンによって管理されます。このサーブレットエンジンは、サーブレットのすべてのメタファンクションを処理する内部オブジェクトです。このメタファンクションには、インスタンス化、初期化、破棄、ほかのコンポーネントからのアクセス、設定管理などがあります。

サーブレットのインスタンス化と削除

サーブレットエンジンは、サーブレットをインスタンス化したあと、その `init()` メソッドを実行して必要な初期化を行います。カウンタの初期化など、サーブレットの生存期間にわたって使用される関数の初期化を実行するには、このメソッドをオーバーライドします。

サーブレットがサービスから削除されると、サーバーエンジンは、最後のタスクを実行し、リソースの割り当てを解除できるように、サーブレット内の `destroy()` メソッドを呼び出します。ログメッセージを書き込んだり、ガベージコレクション時に検出されない残留コネクションを削除したりするには、このメソッドをオーバーライドします。

要求処理

要求が発生すると、Sun ONE Application Server は受信データをサーブレットエンジンに渡します。サーブレットエンジンは、要求の入力データ (フォームデータ、cookie、セッション情報、URL の名前 - 値のペアなど) を処理して `HttpServletRequest` 要求オブジェクトタイプにします。

サーブレットエンジンは `HttpServletResponse` 応答オブジェクトタイプも作成します。その後、両方のオブジェクトをパラメータとしてサーブレットの `service()` メソッドに渡します。

HTTP サーブレット内のデフォルトの `service()` メソッドは、HTTP 転送メソッド (`POST` や `GET` など) に基づいて要求をほかのメソッドに転送します。たとえば、HTTP `POST` 要求は `doPost()` メソッドに転送され、HTTP `GET` 要求は `doGet()` メソッドに転送されます。これにより、サーブレットは使用中の転送メソッドに応じて要求データをさまざまな方法で処理できます。要求の転送はサービスメソッドによって行われるため、一般的に HTTP サーブレットの `service()` をオーバーライドする必要はありません。その代わり、予想される要求タイプに従って `doGet()` や `doPost()` などをオーバーライドします。

ヒント HTTP サーブレットでの自動転送を有効にするには、HTTP 転送メソッドを備えている `request.getMethod()` を呼び出します。Sun ONE Application Server で、すでに要求データは名前 - 値のリストに処理されているため、機能を失うことなく簡単に HTTP サーブレットの `service()` メソッドをオーバーライドできます。ただし、これによって前処理された要求データに依存することになるため、サーブレットの移植性は低下します。

要求に応答するタスクを実行するには、`service()` メソッド (汎用サーブレット) または `doGet()` メソッドや `doPost()` メソッド (HTTP サーブレット) をオーバーライドします。多くの場合、EJB コンポーネントにアクセスしてビジネストランザクションを実行し、要求オブジェクトまたは JDBC `ResultSet` オブジェクト内で情報を照合したあと、新たに作成されたコンテンツを JSP に渡してコンテンツのフォーマットとユーザーへの配信を行います。

サーブレットエンジンのリソースの割り当て

デフォルトでは、サーブレットエンジンは新規要求ごとにスレッドを作成します。この場合、要求ごとにメモリー内でサーブレットの新規コピーをインスタンス化するよりもリソースに与える影響は小さくなります。各スレッドが同じメモリー空間で動作し、サーブレットオブジェクトの変数が互いを上書きする場合があるので、スレッド問題の発生を避ける必要があります。

サーブレットが特にシングルスレッドとして作成されている場合、サーブレットエンジンは受信要求に使用するサーブレットインスタンスをプールします。すべてのインスタンスが使用中であるときに要求を受信すると、インスタンスが使用可能になるまで要求はキューに入れられます。プールするインスタンスの数は、`sun-web.xml` ファイル内の `sun-web-app` 要素の `singleThreadedServletPoolSize` プロパティで設定できます。

`sun-web.xml` ファイルの詳細については、第 6 章「Web モジュールの構築と配備」を参照してください。スレッド問題の詳細については、39 ページの「スレッドの処理」を参照してください。

第 3 章

JavaServer Pages の使用法

この章では、JSP (JavaServer Pages) を Sun ONE Appllication Server Web アプリケーションのページテンプレートとして使う方法について説明します。

この 章には次の節があります。

- JSP の紹介
- JSP の作成
- JSP タグライブラリおよび移植可能な標準タグ
- JSP キャッシュ
- JSP のコンパイル : コマンド行コンパイラ
- JSP のデバッグ

JSP の紹介

JSP は HTML または XML で書かれたブラウザページです。JSP には Java コードを含めることもできるため、複雑な処理を実行したり、出力に条件を付けたり、アプリケーション内のほかのオブジェクトと通信したりできます。Sun ONE Application Server の JSP は JSP 1.2 仕様に準拠しています。この仕様は *install_dir*`/docs/index.htm` からアクセス可能です。*install_dir* は、Sun ONE Application Server がインストールされているディレクトリです。

Sun ONE Application Server アプリケーションでは、JSP はアプリケーションを構成する個々のページです。サーブレットから JSP を呼び出してユーザー対話からの出力を処理できます。また、JSP はほかのアプリケーションコンポーネントと同じ方法でアプリケーション環境にアクセスするので、JSP を対話の相手として利用できます。

JSP は、JSP 要素とテンプレートデータから構成されています。テンプレートデータとは、JSP 仕様に定義されていないテキストや HTML タグなどのデータのことです。たとえば、最小の JSP は JSP エンジンによる処理が不要な静的 HTML ページです。

Sun ONE Application Server は、JSP が最初に呼び出されたときにその JSP を HTTP サーブレットにコンパイルします。また、パフォーマンスを向上させるために、JSP を事前にコンパイルしておくこともできます。これにより、JSP を標準オブジェクトとしてアプリケーション環境で使えるようになり、URL を使ってクライアントから JSP を呼び出すことができるようになります。

JSP エンジン内で動作する JSP は、JSP 固有のタグを解釈し、そのタグが指定するアクションを実行することによって動的コンテンツを生成します。このコンテンツは、それを囲んでいるテンプレートデータとともに出力ページにまとめられ、呼び出したユーザーに返されます。

JSP の作成

JSP の作成方法は、HTML ファイルの作成方法と基本的に同じです。HTML エディタを使ってページを作成したりレイアウトを編集したりできます。ページを JSP にするには、生のソースコードの適切な位置に JSP 固有のタグを挿入し、ファイル拡張子を `.jsp` にします。

JSP 1.2 仕様に準拠する JSP は、ほとんどの部分で HTML との整合性がある XML 構文に従っています。使用可能な JSP タグについては、58 ページの「JSP タグライブラリおよび移植可能な標準タグ」を参照してください。

JSP はサーブレットにコンパイルされるので、サーブレットの設計上の注意事項は JSP にも当てはまります。JSP とサーブレットは同じタスクを実行できますが、それぞれ得意とするタスクは異なります。サーブレットは処理能力と適応性に優れています。ただし、サーブレットから HTML の出力を実行すると、手動でコーディングする必要があるため処理が面倒な `println` 文が多量に発生します。それに対し、JSP は HTML ファイルなので HTML エディタで編集でき、レイアウト作業に優れています。ただし、複雑な計算タスクや処理タスクの実行には不向きです。サーブレットについては第 2 章「サーブレットの使用法」を参照してください。

JSP は、次の特徴もあります。

- メンテナンスの容易さを考慮した設計
- 移植性を考慮した設計
- 例外の処理

メンテナンスの容易さを考慮した設計

各 JSP はほかの任意の JSP を呼び出したり、取り込んだりできます。たとえば、汎用のコーポレートヘッダー、標準のナビゲーションバー、左側の目次カラムなどを作成できます。このカラムの各要素は、個別の JSP に入っており、作成されたページごとに取り込まれています。このページは、各サブフレームを読み込むページを動的に決めるフレームセットとして機能する JSP で構成できます。JSP は、サーブレットへのコンパイル時または要求の到着時に取り込むこともできます。

移植性を考慮した設計

JSP は、異なるアプリケーションおよび異なるサーバー間で完全に移植できます。特定のアプリケーションデータの知識を保持しないという欠点がありますが、そのようなデータが不要な場合は問題ありません。

汎用 JSP を使う例としては、ナビゲーションバーやコーポレートヘッダーおよびフッターのような移植性のあるページ要素があります。これはほかの JSP に取り込まれることを想定しています。再利用可能な汎用ページ要素のライブラリを作成してアプリケーション全体で使ったり、ほかの複数のアプリケーションで使ったりすることができます。

たとえば、もっとも簡単な汎用 JSP は JSP 固有のタグを持たない静的 HTML ページです。これよりやや複雑な JSP は、日時の表示などの一般データを操作したり、要求オブジェクトの標準値セットに基づいたページ構造を変更したりする Java コードを含みます。

例外の処理

検出できない例外が JSP ファイルで発生すると、通常、Sun ONE Application Server は `404` または `500` エラーの例外を生成します。この問題を避けるには、`<%@ page%>` タグの `errorPage` 属性を設定します。

JSP タグライブラリおよび移植可能な標準タグ

Sun ONE Application Server はタグライブラリおよび移植可能な標準タグをサポートしています。タグライブラリの詳細については、次の Web サイトで JSP 1.2 仕様書を参照してください。

`http://java.sun.com/products/jsp/download.html`

JSP 1.2 のタグ構文の要約については、次の PDF ファイルを参照してください。

`http://java.sun.com/products/jsp/pdf/card12.pdf`

JSP キャッシュ

JSP キャッシュを使うと、JSP ページのフラグメントを Java エンジン内にキャッシュできます。フラグメントごとに異なるキャッシュ基準を使ってキャッシュできます。たとえば、株式相場を表示するフラグメントや気象情報を表示するフラグメントなどがあるとします。株式相場のフラグメントは 10 分間キャッシュし、気象情報のフラグメントは 30 分間キャッシュするというように設定できます。

サーブレット関連の応答キャッシュの詳細については、46 ページの「サーブレットの結果のキャッシュ」を参照してください。

JSP キャッシュは JSP 1.2 で提供されているカスタムタグライブラリのサポートを使います。JSP キャッシュは、*install_dir*`/lib/appserv-tags.jar` ファイルにパッケージされているタグライブラリによって実装されます。このファイルは、ユーザーの Web アプリケーションの `WEB-INF/lib` ディレクトリにコピーして使用できます。`appserv-tags.tld` というタグ記述ファイルは、この JAR ファイルと *install_dir*`/lib/tlds` ディレクトリにあります。

これらのタグを JSP ファイル内で次のように参照します。

```
<%@ taglib prefix="prefix" uri="Sun ONE Application Server Tags" %>
```

これ以降は、<*prefix*`:cache`> や <*prefix*`:flush`> のように cache タグを利用できます。たとえば、*prefix* が `mypfx` の場合は、<`mypfx:cache`> や <`mypfx:flush`> のように cache タグを利用できます。

このタグライブラリに別の URI を使用するには、`web.xml` ファイルで <`taglib`> 要素を明示的に使用します。

使用できるタグは次のとおりです。

- cache
- flush

cache

cache タグは、指定された属性に従って、開始タグと終了タグの間にある本体をキャッシュします。このタグが初めて見つかると、本体のコンテンツが実行され、キャッシュされます。次回以降の実行時には、キャッシュされているコンテンツが毎回チェックされます。コンテンツの更新が必要な場合は、コンテンツがもう一度実行され、キャッシュされているデータが更新されます。コンテンツの更新が不要な場合は、キャッシュされているデータが提供されます。

属性

次の表は、cache タグの属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

cache の属性

属性	デフォルト値	説明
key	*ServletPath_Suffix*	(省略可能) この名前は、キャッシュエントリにアクセスするためにコンテナで使用される。キャッシュキーの末尾にサーブレットのパスが追加されて、キャッシュエントリにアクセスするためのキーが生成される。キーが指定されていない場合は、ページ内でのタグの位置に従って番号が生成される
timeout	60s	(省略可能) この時間が経過すると、タグの本体が実行され、キャッシュが更新される。デフォルトでは、この値の単位は秒と見なされる。別の単位を指定するには、タイムアウト値の末尾に次の単位を追加する。s は秒、m は分、h は時間、d は日を表す。たとえば、2 時間を指定するには 2h とする
nocache	false	(省略可能) true を設定すると、cache タグがないかのように本体のコンテンツが実行され、提供される。この属性を使うと、キャッシュされている応答を送信するか、本体を実行する (ただし、応答はキャッシュされない) のかをプログラムで決定できる
refresh	false	(省略可能) true を設定すると、本体のコンテンツがもう一度実行され、応答がキャッシュされる。この場合、timeout 設定には関係なく、プログラムを使って即座にキャッシュを更新できる

例

次の例はキャッシュされた JSP ページを示しています。

```
<%@ taglib prefix="mypfx" uri="Sun ONE Application Server Tags" %>

<%
   String cacheKey = null;
   if (session != null)
      cacheKey = (String)session.getAttribute("loginId");

   // キャッシュの有無のチェック
   boolean noCache = false;
   String nc = request.getParameter("nocache");
   if (nc != null)
      noCache = "true";

   // 再読み込みの強制
   boolean reload=false;
   String refresh = request.getParameter("refresh");
   if (refresh != null)
      reload = true;
%>

<mypfx:cache key="<%= cacheKey %>" nocache="<%= noCache %>"
refresh="<%= reload %>" timeout="10m">
<%
   String page = request.getParameter("page");
   if (page.equals("frontPage") {
      // データベースからの見出し取得
   } else {
      .....
%>
</mypfx:cache>

<mypfx:cache timeout="1h">
<h2> Local News </h2>
<%
   // ヘッドラインニュースの取得とキャッシュ
%>
</mypfx:cache>
```

flush

キャッシュを強制的にフラッシュします。key が指定されている場合は、そのキーを持つエントリだけがフラッシュされます。キーが指定されていない場合は、キャッシュ全体がフラッシュされます。

属性

次の表は、flush タグの属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

flush の属性

属性	デフォルト値	説明
key	*ServletPath_Suffix*	(省略可能) この名前は、キャッシュエントリにアクセスするためにコンテナで使用される。キャッシュキーの末尾にサーブレットのパスが追加されて、キャッシュエントリにアクセスするためのキーが生成される。キーが指定されていない場合は、ページ内でのタグの位置に従って番号が生成される

例

key="foobar" を持つエントリだけをフラッシュするには

```
<mypfx:flush key="foobar"/>
```

キャッシュ全体をフラッシュするには

```
<% if (session != null && session.getAttribute("clearCache") != null) { %>
    <mypfx:flush />
<% } %>
```

JSP のコンパイル : コマンド行コンパイラ

Sun ONE Application Server は、次の方法で JSP 1.2 準拠のソースファイルをサーブレットにコンパイルできます。

- JSP は実行時に自動的にコンパイルされる
- `asadmin deploy` コマンドには `precompilejsp` オプションがある。『Sun ONE Application Server 開発者ガイド』を参照
- Ant タスク `sun-appserv-jspc` で JSP を事前にコンパイルできる。『Sun ONE Application Server 開発者ガイド』を参照
- この節で説明する `jspc` コマンド行ツールを使って、コマンド行で JSP をプリコンパイルする

JSP コンテナが JAR ファイルから事前にコンパイルした JSP を取り出せるようにするには、JSP の動的再読み込みを無効にする必要があります。そのためには、`sun-web.xml` ファイルの `jsp-config` 要素の `reload-interval` プロパティの値を `-1` に設定します。124 ページの「JSP に関する要素」を参照してください。

`jspc` コマンド行ツールは、*install_dir*`/bin` の下にあります。このディレクトリがパスに含まれていることを確認してください。`jspc` コマンドの形式は次のとおりです。

`jspc` [*options*] *file_specifier*

次の表は、`jspc` コマンドで指定できる *jsp_specifier* を示しています。左の列はファイル指示子、右の列は各ファイル指示子の説明です。

`jspc` コマンドのファイル指示子

ファイル指示子	説明
files	コンパイルする 1 つまたは複数の JSP ファイル
`-webapp` *dir*	Web アプリケーションがあるディレクトリ。指定したディレクトリとそのサブディレクトリ内のすべての JSP がコンパイルされる。WAR、JAR、または ZIP ファイルは指定できない。これらのファイルは最初にオープンディレクトリ構造に抽出する必要がある

次の表は、jspc コマンドの基本的なオプション (*options*) を示しています。左の列はオプション、右の列は各オプションの説明です。

jspc の基本的なオプション

オプション	説明
-q	非出力モードを有効にする (-v0 と同じ)。重大なエラーメッセージだけ表示する
-d *dir*	コンパイル済み JSP の出力ディレクトリを指定する。コンパイルされていない JSP が含まれているディレクトリに基づいてパッケージディレクトリが自動的に生成される。デフォルトの最上位ディレクトリは jspc が起動されるディレクトリである
-p *name*	すべての指定済み JSP にターゲットパッケージの名前を指定し、-d オプションによって実行されるデフォルトのパッケージ生成をオーバーライドする
-c *name*	最初にコンパイルされる JSP のターゲットクラス名を指定する。後続の JSP は影響を受けない
-uribase *dir*	コンパイルに相対的な URI ディレクトリを指定する。コマンドに一覧表示された JSP ファイルだけに適用され、-webapp で指定された JSP ファイルには適用されない
	uriroot に相対的な各 JSP ファイルの場所を指定する。指定されない場合、デフォルトの場所は / （ルート）
-uriroot *dir*	URI ファイルを解決するルートディレクトリを指定する。コマンドに一覧表示された JSP ファイルだけに適用され、-webapp で指定された JSP ファイルには適用されない
	このオプションを指定しない場合は、最初の JSP ページのすべての親ディレクトリ内で WEB-INF サブディレクトリを検索する。WEB-INF サブディレクトリを持つ JSP ページにもっとも近いディレクトリが使われる
	JSP のどの親ディレクトリにも WEB-INF サブディレクトリがない場合は、jspc が起動されるディレクトリが使われる
-genclass	生成されたサーブレットをクラスファイルにコンパイルする

次の表は、jspc コマンドの高度なオプション (*options*) を示しています。左の列はオプション、右の列は各オプションの説明です。

jspc の高度なオプション

オプション	説明
-v [*level*]	詳細モードを有効にする。*level* はオプション。デフォルトは 2。可能な *level* 値は次のとおり • 0 - 重大なエラーメッセージのみ • 1 - エラーメッセージのみ • 2 - エラーおよび警告メッセージのみ • 3 - エラー、警告、および情報メッセージ • 4 - エラー、警告、情報、およびデバッグメッセージ
-mapped	各 HTML 行の write 呼び出しと、JSP ファイルの各行の場所を記述するコメントを生成する。デフォルトでは、すべての隣接した write 呼び出しが結合され、場所のコメントは生成されない
-die [*code*]	エラーが発生した場合に、*code* によって指定されたエラー番号を返す。*code* がない場合やパースできない場合は、デフォルトで 1 に設定される
-webinc *file*	-webapp オプションの部分的なサーブレットマッピングを作成する。これを web.xml ファイルに貼り付け可能
-webxml *file*	-webapp オプションの web.xml ファイル全体を作成する
-ieplugin *class_id*	Internet Explorer の Java プラグイン COM クラス ID を指定する。<jsp:plugin> タグによって使われる

たとえば、このコマンドは、hello という JSP ファイルをコンパイルし、コンパイル済みの JSP を hellodir の下に書き込みます。

```
jspc -d hellodir -genclass hello.jsp
```

このコマンドは、webappdir の下の Web アプリケーション内のすべての JSP ファイルをコンパイルし、jspclassdir の下のクラスファイルに書き込みます。

```
jspc -d jspclassdir -genclass -webapp webappdir
```

これらの事前コンパイル済み JSP のいずれかを Web アプリケーションで使用するには、`hellodir` または `jspclassdir` の下のクラスを JAR ファイルに書き込み、この JAR ファイルを `WEB-INF/lib` に置きます。さらに、`sun-web.xml` ファイルの `reload-interval` プロパティの値を `-1` に設定します。

JSP のデバッグ

Sun ONE Studio 4 を使って JSP をデバッグする場合、JSP コードと生成されたサーブレットコードの両方にブレークポイントを設定し、これらの間で表示を切り替えて両方の同一ブレークポイントを見ることができます。

Sun ONE Studio でのデバッグの設定方法については、『Sun ONE Application Server 開発者ガイド』を参照してください。詳細については、Sun ONE Studio 4, Enterprise Edition のチュートリアルを参照してください。

第 4 章

ユーザーセッションの作成と管理

この章では、対話間におけるユーザーおよびトランザクションの情報を継続的に維持できるセッションを作成し、管理する方法について説明します。

この章には次の節があります。

- セッションについて
- セッションの使用法
- セッションマネージャ

セッションについて

ユーザーセッションという用語は、サーバーが記録するユーザーとアプリケーション間の一連の対話を意味します。セッションは、持続性のあるオブジェクト (EJB コンポーネントやデータベース結果セットへのハンドルなど) や認証されたユーザー ID などのユーザー固有の状態を多数の対話間で維持するために使われます。たとえば、検証されたユーザーログインと、そのあとにそのユーザーに対して指示された一連のアクティビティを記録するためにセッションを使用できます。

セッション自体はサーバー内に存在します。クライアントは、要求ごとに、cookie 内のセッション ID を送信します。または、ブラウザが cookie を許可しない場合、サーバーは自動的にセッション ID を URL に書き込みます。

Sun ONE Application Server はすべてのセッションアクティビティに対して、`HttpSession` と呼ばれるサーブレットの標準セッションインタフェースをサポートしています。このインタフェースによって移植可能で安全なサーブレットを作成できます。

セッションと cookie

cookie は、サーバーが同じクライアントからの呼び出しかを認識できるように、呼び出し側のブラウザに送信され、後続の呼び出しのたびに取り出される小さな情報の集まりです。cookie は、期限切れになるまで、それを作成したサイトを呼び出すたびに返されます。

セッションは、その最初の作成時にクライアントに送信されるセッション cookie によって自動的に維持されます。セッション cookie には、継続的な各対話でブラウザに接続するクライアントを識別するセッション ID が含まれています。クライアントが cookie をサポートしない場合や許可しない場合、サーバーはセッション ID がそのクライアントからの URL 内に現れている部分の URL を書き換えます。

セッションで cookie を使用するかどうかや、どのように使用するかを設定できます。`sun-web.xml` ファイル内の `session-properties` 要素と `cookie-properties` 要素については、第 6 章「Web モジュールの構築と配備」を参照してください。

セッションと URL の書き換え

Sun ONE Application Server プラグインが URL を暗黙的に書き換える状況には、次の 2 つがあります。

- Sun ONE Application Server から応答があったとき。URL を暗黙的に書き換えるよう指定されている場合、プラグインは応答をクライアントに渡す前に URL を書き換える
- クライアントからの要求が Sun ONE Application Server に送信される必要がなく、Web サーバーサイドで処理できるとき。このような要求はセッションの途中で発生する可能性があり、応答が必要ない場合がある

セッションで URL の書き換えを使用するかどうかを設定できます。`sun-web.xml` ファイル内の `session-properties` 要素については、第 6 章「Web モジュールの構築と配備」を参照してください。

セッションとセキュリティ

Sun ONE Application Server のセキュリティモデルは、認証されたユーザーセッションをベースにしています。セッションが作成されると、アプリケーションユーザーは認証を受け (ユーザー認証が使用される場合)、そのセッションにログインします。EJB 要求を受け取るサーブレットの対話の各ステップでは、JSP で出力をフォーマット化するためのコンテンツを生成するほか、ユーザーが正しく認証されていることを確認します。

さらに、セッション cookie が保護された接続 (HTTPS) だけに渡されるように指定できます。したがって、安全なチャンネル上に限りセッションをアクティブな状態で維持できます。

セキュリティの詳細については、第 5 章「Web アプリケーションのセキュリティ」を参照してください。

セッションの使用法

セッションを使うには、まず HttpServletRequest メソッドの getSession() メソッドを使ってセッションを作成します。セッションが確立したら、所定のメソッドを使ってそのプロパティを調べたり、設定したりします。必要に応じて、非アクティブな状態が一定時間続いたあとでタイムアウトになるようにセッションを設定したり、セッションを手動で無効にしたりします。ほかのコンポーネントも使用できるように、オブジェクトを保存するセッションにバインドすることもできます。

セッションの作成またはセッションへのアクセス

新しいセッションを作成したり、既存のセッションにアクセスしたりするには、次の例に示すように HttpServletRequest の getSession() メソッドを使います。

```
HttpSession mySession = request.getSession();
```

getSession() は、要求に関連付けられた正当なセッションオブジェクトを返します。このセッションオブジェクトは、要求オブジェクト内にカプセル化されているセッション cookie 内で識別されます。引数を指定せずにこのメソッドを呼び出すと、要求に関連付けられているセッションがまだ存在していない場合にはセッションが作成されます。さらに、ブール値の引数でメソッドを呼び出すと、その引数が true の場合だけ、セッションが作成されます。

次の例は、サーブレットの doPost() メソッドを示しています。このメソッドは、セッションが存在する場合だけ サーブレットの主な関数を実行します。getSession() に false パラメータを指定すると、セッションがまだ存在しない場合でもサーブレットは新しいセッションを作成しないので注意してください。

```
public void doPost (HttpServletRequest req,
                    HttpServletResponse res)
             throws ServletException, IOException
{
   if ( HttpSession session = req.getSession(false) )
   {
      // セッションが取得されたので、サーブレットのオペレーションを続行します。
   }
```

```
    else
        // セッションがないので、エラーページが返されます。

    }
}
```

注 getSession() メソッドは、応答ストリームに書き込みが行われる前に呼び出す必要があります。書き込みの前に呼び出さないと、SetCookie 文字列は、HTTP ヘッダーではなく HTTP 応答の本体に配置されます。

getSession() の詳細については、Java サーブレット仕様バージョン 2.3 を参照してください。

セッションプロパティの調査

セッション ID を確立したら、HttpSession インタフェース内のメソッドを使って、セッションのプロパティを調べ、HttpServletRequest インタフェース内のメソッドを使ってそのセッションに関連する要求プロパティを調べます。

次の表は、セッションのプロパティを検証するためのメソッドを示しています。左の列は HttpSession のメソッド、右の列は各メソッドの説明です。

HttpSession のメソッド

HttpSession のメソッド	説明
getCreationTime()	セッション時刻を返す (1970 年 1 月 1 日 00:00:00 GMT 以降の時刻でミリ秒単位)
getId()	割り当てられたセッション識別子を返す。HTTP セッションの識別子は、サーバーが作成し管理する一意の文字列
getLastAccessedTime()	割り当てられたセッション識別子を持つ要求をクライアントが送信した最後の時刻を返す (1970 年 1 月 1 日 00:00:00 GMT 以降の時刻でミリ秒単位)。新しいセッションの場合は -1 を返す
isNew()	このセッションが新規と見なされるかどうかを示すブール値を返す。サーバーがセッションを作成し、クライアントがそのセッションに要求を送信していない場合は、新規のセッションになる。つまり、クライアントはセッションを「認識」または「結合」しておらず、次の要求を出すときに正しいセッション識別情報を返さない可能性がある

次に例を示します。

```
String mySessionID = mySession.getId();
if ( mySession.isNew() ) {
   log.println(currentDate);
   log.println("client has not yet joined session " + mySessionID);
}
```

次の表は、サーブレット要求のプロパティを検証するためのメソッドを示しています。左の列は HttpServletRequest のメソッド、右の列は各メソッドの説明です。

HttpServletRequest のメソッド

HttpServletRequest のメソッド	説明
getRemoteUser()	要求を出したユーザーの名前を取得する (HTTP 認証によって情報を取得)。要求にユーザー名の情報がない場合には NULL を返す
getRequestedSessionId()	この要求とともに指定されたセッション ID を返す。クライアントが指定したセッション ID が無効で新しいセッションが作成された場合は、現在のセッション内のセッション ID と異なる場合がある。要求に関連付けられたセッションがない場合は NULL を返す
isRequestedSessionIdValid()	この要求が現在有効なセッションに関連付けられているかどうかを確認する。要求されたセッションが有効でない場合、getSession() メソッドからは返されない
isRequestedSessionIdFromCookie()	クライアントから指定された要求のセッション ID が cookie である場合は true を返し、それ以外の場合は false を返す
isRequestedSessionIdFromURL()	クライアントから指定された要求のセッション ID が URL の一部である場合は true を返し、それ以外の場合は false を返す

次に例を示します。

```
if ( request.isRequestedSessionIdValid() ) {
   if ( request.isRequestedSessionIdFromCookie() ) {
      // このセッションはセッション cookie 内で維持されます。
   }
   // 有効なセッションを必要とするほかのタスク
} else {
   // アプリケーションエラーを記録します。
}
```

セッションへのデータのバインド

複数のユーザー対話間で利用できるように、オブジェクトをセッションにバインドできます。

次の表は、オブジェクトをセッションオブジェクトにバインドするのをサポートする HttpSession のメソッドを示しています。左の列は HttpSession のメソッド、右の列は各メソッドの説明です。

HttpSession のメソッド

HttpSession のメソッド	説明
getAttribute()	セッション内の所定の名前にバインドされたオブジェクトを返す。バインドされたものがなければ NULL を返す
getAttributeNames()	セッションにバインドされたすべての属性の名前の配列を返す
setAttribute()	指定された名前を使って、指定されたオブジェクトをセッションにバインドする。同じ名前でバインドされている既存のオブジェクトは上書きされる。セッションにバインドされたオブジェクトを分散するには、serializable インタフェースを実装する必要がある
removeAttribute()	指定した名前を持つセッション内のオブジェクトのバインドを解除する。指定した名前のオブジェクトがバインドされていなければ、このメソッドの影響はない

HttpSessionBindingListener によるバインドの通知

オブジェクトによっては、セッションに入れられたとき、またはセッションから削除されたときにユーザーがそれを認識する必要があるものもあります。この情報を取得するには、これらのオブジェクト内に HttpSessionBindingListener インタフェースを実装します。アプリケーションがセッションにデータを格納したり、セッションからデータを削除するとき、サーブレットエンジンは、バインドまたはバインド解除されるオブジェクトが HttpSessionBindingListener を実装しているかどうかを確認します。このオブジェクトが HttpSessionBindingListener インタフェースを実装している場合、Sun ONE Application Server はこのインタフェースを介して、セッションにバインドされることやセッションからバインド解除されることをこのオブジェクトに通知します。

セッションの無効化

非アクティブな状態が一定時間続いたあとで、セッションが自動的に無効になるように指定できます。または、HttpSession の invalidate() メソッドを使って手動でセッションを無効にすることもできます。

ヒント セッションの API には明示的なセッションログアウト API はありません。したがって、ログアウトを実行するには session.invalidate() API を呼び出す必要があります。

手動によるセッションの無効化

手動でセッションを無効にするには、次のメソッドを呼び出します。

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

セッションにバインドされたオブジェクトはすべて削除されます。

セッションタイムアウトの設定

セッションタイムアウトは、sun-web.xml 配備記述子ファイルを使って設定します。詳細については、第 6 章「Web モジュールの構築と配備」の session-properties 要素を参照してください。

セッションマネージャ

セッションマネージャは、新しいセッションが開始されるたびに、新しいセッションオブジェクトを自動的に作成します。状況によっては、クライアントがセッションに参加しないこともあります。たとえば、セッションマネージャは cookie を使用するがクライアントは cookie を受け入れない場合などです。

Sun ONE Application Server 7 には、次のようなセッション管理オプションが用意されています。

- StandardManager (デフォルトのセッションマネージャ)
- PersistentManager (持続性のあるデータストアを使用する、付属のセッションマネージャ)

注 セッションマネージャインタフェースは不確定なインタフェースです。不確定なインタフェースは試験的または一時的なインタフェースであるため、次のリリースで互換性がなくなったり、削除されたり、または安定したインタフェースに置き換えられたりする場合があります。

StandardManager

StandardManager はデフォルトのセッションマネージャです。

StandardManager の有効化

StandardManager を明示的に指定して、そのデフォルトパラメータを変更することもできます。そのためには、Web アプリケーションの sun-web.xml ファイルを、次の例のように編集します。

```
<sun-web-app>
   ...
   <session-config>
      <session-manager>
         <manager-properties>
            <property name="reapIntervalSeconds" value="20" />
         </manager-properties>
      </session-manager>
      ...
   </session-config>
   ...
</sun-web-app>
```

sun-web.xml ファイルの詳細については、第 6 章「Web モジュールの構築と配備」を参照してください。

StandardManager のマネージャプロパティ

次の表は、StandardManager セッションマネージャの manager-properties のプロパティを示します。左の列にプロパティの名前、中央の列にデフォルト値、右の列に各プロパティの説明を示します。

manager-properties のプロパティ

プロパティ名	デフォルト値	説明
reapIntervalSeconds	60	期限切れのセッションをチェックする周期を秒単位で指定する セッションデータが変更される頻度より小さい値を設定することを推奨する。たとえば、頻繁にアクセスされる Web サイト上のヒットカウンタサーブレットの場合、この値をできるだけ小さくする必要がある (1 秒)。そうしないと、サーバーを再起動するたびに最後の数回分のヒットが失われる可能性がある
maxSessions	-1	アクティブセッションの最大数を指定する。制限しない場合は -1 (デフォルト値)

manager-properties のプロパティ (続き)

プロパティ名	デフォルト値	説明
sessionFilename	なし。再起動すると、その前の状態は破棄される	アプリケーションの再起動の間にセッション状態を保持するファイルの絶対パスまたは相対パスを指定する (状態の保持が可能な場合)。相対パスは、この Web アプリケーションの一時ディレクトリを基準とした場所を示す

PersistentManager

PersistentManager は、Sun ONE Application Server に付属しているもう 1 つのセッションマネージャです。セッションの持続性を実現するために、PersistentManager はファイルを使用し、そこに各セッションを直列化します。ユーザー独自の持続性メカニズムを作成することもできます。

PersistentManager の有効化

PersistentManager を明示的に指定して、そのデフォルトパラメータを変更することもできます。そのためには、Web アプリケーションの sun-web.xml ファイルを、次の例のように編集します。persistence-type には file を設定する必要があります。

```
<sun-web-app>
   ...
   <session-config>
      <session-manager persistence-type=file>
         <manager-properties>
             <property name=reapIntervalSeconds value=20 />
         </manager-properties>
         <store-properties>
             <property name=directory value=sessions />
         </store-properties>
      </session-manager>
      ...
   </session-config>
   ...
</sun-web-app>
```

sun-web.xml ファイルの詳細については、第 6 章「Web モジュールの構築と配備」を参照してください。

PersistentManager のマネージャプロパティ

次の表は、PersistentManager セッションマネージャの manager-properties のプロパティを示しています。左の列にプロパティの名前、中央の列にデフォルト値、右の列にそのプロパティの説明を示します。

manager-properties のプロパティ

プロパティ名	デフォルト値	説明
reapIntervalSeconds	60	期限切れのセッションをチェックする周期を秒単位で指定する セッションデータが変更される頻度より小さい値を設定することを推奨する。たとえば、頻繁にアクセスされる Web サイト上のヒットカウンタサーブレットの場合、この値をできるだけ小さくする必要がある (1 秒)。そうしないと、サーバーを再起動するたびに最後の数回分のヒットが失われる可能性がある
maxSessions	-1	アクティブセッションの最大数を指定する。制限しない場合は -1 (デフォルト値)

PersistentManager のストアプロパティ

次の表は、PersistentManager セッションマネージャの store-properties のプロパティを示しています。左の列にプロパティの名前、中央の列にデフォルト値、右の列にそのプロパティの説明を示します。

store-properties のプロパティ

プロパティ名	デフォルト値	説明
reapIntervalSeconds	60	現在スワップアウトされているセッションについて期限切れをチェックする周期を秒単位で指定する セッションデータが変更される頻度より小さい値を設定することを推奨する。たとえば、頻繁にアクセスされる Web サイト上のヒットカウンタサーブレットの場合、この値をできるだけ小さくする必要がある (1 秒)。そうしないと、サーバーを再起動するたびに最後の数回分のヒットが失われる可能性がある

store-properties のプロパティ (続き)

プロパティ名	デフォルト値	説明
directory	javax.servlet.context.tempdir コンテキスト属性によって指定されたディレクトリ	個々のセッションファイルが書き込まれるディレクトリの絶対パスまたは相対パスを指定する。相対パスは、この Web アプリケーションの一時作業ディレクトリを基準とした場所を示す

第 5 章

Web アプリケーションのセキュリティ

この章では、Sun ONE Application Server でユーザー認証とアクセス承認を実行するコンポーネントを持つ、安全な Web アプリケーションを作成する方法について説明します。

この 章には次の節があります。

- サーブレットによるユーザー認証
- シングルサインオンでのユーザー認証
- サーブレットによるユーザー承認
- クライアント証明書のフェッチ
- SHTML と CGI のセキュリティ

サーブレットによるユーザー認証

J2EE 仕様バージョン 1.3 で要求される Web ベースのログインメカニズムは Sun ONE Application Server でサポートされています。これらのメカニズムは次のとおりです。

- HTTP 基本認証
- SSL 相互認証
- フォームによるログイン

`web.xml` 配備記述子ファイルの `login-config` 要素は、使用される認証方法、HTTP 基本認証で表示されるアプリケーションのレルム名、およびフォームログインメカニズムの属性を記述します。

`login-config` 要素の構文は次のとおりです。

```
<!ELEMENT login-config
(auth-method?,realm-name?,form-login-config?)>
```

注	login-config の auth-method サブ要素は公式には省略可能ですが、このサブ要素がないと、サーバーはデフォルトで HTTP 基本認証を使用します。この認証はあまり安全ではありません。

web.xml の要素の詳細については、Java Servlet 仕様バージョン 2.3 の第 13 章「Deployment Descriptor」を参照してください。

sun-web.xml の要素の詳細については、第 6 章「Web モジュールの構築と配備」を参照してください。

プログラムによるログインの詳細については、『Sun ONE Application Server 開発者ガイド』を参照してください。

HTTP 基本認証

HTTP 基本認証 (RFC2068) は Sun ONE Application Server でサポートされています。パスワードは base64 エンコード方式で送信されるので、この種の認証はそれほど安全ではありません。転送時にパスワードを保護するために、SSL または同等のトランスポートの暗号化を使用することをお勧めします。

SSL 相互認証

Secure Socket Layer (SSL) 3.0、およびクライアントとサーバーの相互の証明書ベースの認証を実行する方法は、J2EE 仕様バージョン 1.3 の要件です。このセキュリティメカニズムによって、HTTPS (SSL 上の HTTP) を使ったユーザー認証が提供されます。

Sun ONE Application Server の SSL 相互認証メカニズム (HTTPS 認証とも呼ばれる) は、次の一連の暗号をサポートしています。

```
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
```

フォームによるログイン

ログイン画面の見た目と使いやすさは、HTTP ブラウザの組み込みメカニズムでは制御できません。J2EE では、ログイン用の標準 HTML フォームやサーブレットまたは JSP ベースのフォームをパッケージ化する機能を導入しています。ログインフォームは Web 保護ドメイン (HTTP レルム) に関連付けられ、まだ認証されていないユーザーを認証するために使われます。

基礎となっているトランスポートで保護されていない限り、パスワードがセキュリティで保護されずに送信されるため、この種の認証はあまり安全ではありません。転送時にパスワードを保護するために、SSL または同等のトランスポートの暗号化を使用することをお勧めします。

認証を適切に処理するには、ログインフォームのアクションが常に `j_security_check` である必要があります。

HTML ページ内でフォームをプログラムする方法を示す HTML サンプルは次のとおりです。

```
<form method="POST" action="j_security_check">
   <input type="text" name="j_username">
   <input type="password" name="j_password">
</form>
```

フォームのパラメータエンコードを指定できます。詳細については、128 ページの「parameter-encoding」を参照してください。

シングルサインオンでのユーザー認証

Sun ONE Application Server 上の複数のアプリケーションにわたるシングルサインオンは、Sun ONE Application Server のサーブレットおよび JSP によってサポートされます。この機能によって、同一のサインオン情報を複数のアプリケーションで共有できるので、ユーザーはアプリケーションごとにサインオンする必要がありません。これらのアプリケーションはユーザー認証を 1 度だけ行うように作成されており、この認証情報は必要に応じてほかの関連するアプリケーションに伝えられます。

シングルサインオンのシナリオを使うアプリケーションの例としては、すべての航空会社を検索し、各航空会社の Web サイトへのリンクを提供する、統合航空券予約サービスがあります。ユーザーが統合予約サービスにサインオンすると、そのユーザー情報を各航空会社のサイトで使用できるので、別のサインオンを要求する必要はありません。

シングルサインオンは次の規則に従って動作します。

- シングルサインオンは、同じレルムと仮想サーバー向けに設定された Web アプリケーションに適用される。レルムは `web.xml` ファイル内の `realm-name` 要素で定義される。仮想サーバーについては、『Sun ONE Application Server 管理者ガイド』または『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照
- 仮想サーバー上にある任意の Web アプリケーション内の保護されていないリソースだけにユーザーがアクセスしている場合、ユーザー認証は行われない
- 仮想サーバーに関連付けられている任意の Web アプリケーション内の保護されているリソースにユーザーがアクセスすると、ユーザーに対して認証が要求される。このとき、現在アクセス中の Web アプリケーションに定義されているログイン方法が使用される
- ユーザーが認証されると、関連付けられているすべての Web アプリケーションで、このユーザーに関連付けられているロールがアクセス許可の判断に使用される。各アプリケーションで個別にユーザー認証を行う必要はない
- ユーザーが 1 つの Web アプリケーションからログアウトすると (たとえば、フォームベースのログインを使用している場合は、対応するセッションを無効またはタイムアウトにして)、ユーザーのセッションはすべての Web アプリケーションで無効になる。その後、いずれかの Web アプリケーションで保護されているリソースにアクセスしようとすると、もう一度認証が要求される

シングルサインオン機能は、保存されているユーザー識別情報を各要求に関連付けるトークンを送信するために HTTP cookie を使用します。したがって、cookie がサポートされているクライアント環境でのみ使用できます。

シングルサインオン機能を設定するには、`server.xml` ファイルの `virtual-server` 要素内に次のプロパティを設定します。

- sso-enabled : false の場合、この仮想サーバーに対してシングルサインオンが無効になり、ユーザーは仮想サーバー上のアプリケーションごとに個別に認証を行う必要がある。デフォルトは true
- sso-max-inactive-seconds : クライアントが活動停止後、ユーザーのシングルサインオンの記録がパージされるまでの時間を指定する。シングルサインオンは同一仮想サーバー上の複数のアプリケーションに適用されるので、これらのアプリケーションのいずれかにアクセスすることでシングルサインオンの記録は有効なまま確保される。デフォルト値は 5 分 (300 秒)。値を大きくすればユーザーのシングルサインオンの持続時間が長くなるが、サーバー上のメモリー消費量も増える
- sso-reap-interval-seconds : 期限切れのサインオンの記録がパージされる間隔を指定する。デフォルト値は 60

次に、すべてデフォルト値を設定した例を示します。

```
<virtual-server id="server1" ... >
   ...
   <property name="sso-enabled" value="true"/>
   <property name="sso-max-inactive-seconds" value="300"/>
   <property name="sso-reap-interval-seconds" value="60"/>
</virtual-server>
```

サーブレットによるユーザー承認

適切な承認レベルを持つユーザーのアクセスだけを許可するようにサーブレットを設定できます。この節では次のトピックについて説明します。

- ロールの定義
- サーブレットによる承認の制約の定義

ロールの定義

ロールは、J2EE 配備記述子ファイル web.xml に定義し、対応するロール割り当ては、Sun ONE Application Server 配備記述子ファイル sun-application.xml (個別に配備された Web モジュールの場合は sun-web.xml) 内に定義します。sun-web.xml の詳細については、第 6 章「Web モジュールの構築と配備」を参照してください。

sun-application.xml ファイルまたは sun-web.xml ファイル内の各 security-role-mapping 要素は、アプリケーションで許可されるロール名を宣言し、それを主体とグループに割り当てます。たとえば、個別に配備された Web モジュールの sun-web.xml ファイルには次の内容が含まれています。

```
<sun-web-app>
   <security-role-mapping>
      <role-name>manager</role-name>
      <principal-name>jgarcia</principal-name>
      <principal-name>mwebster</principal-name>
      <group-name>team-leads</group-name>
   </security-role-mapping>
   <security-role-mapping>
      <role-name>administrator</role-name>
      <principal-name>dsmith</principal-name>
   </security-role-mapping>
</sun-web-app>
```

この例での role-name は、対応する web.xml ファイル内の security-role 要素の role-name と一致する必要があります。

J2EE アプリケーション (EAR ファイル) の場合、アプリケーションモジュールのすべてのセキュリティロールの割り当ては sun-application.xml ファイルに指定する必要があります。個別に配備された Web モジュールについては、常に sun-web.xml ファイルでロールを指定します。ロールは特定の主体またはグループ、あるいはその両方に割り当てることができます。割り当てに使用する主体名やグループ名は、現在のデフォルトレルムで有効な主体またはグループであることが必要です。

サーブレットによる承認の制約の定義

サーブレットレベルでは、web.xml ファイルの auth-constraint 要素を使ってアクセス権を定義します。

リソースコレクションの auth-constraint 要素を使って、リソースコレクションが許可されるユーザーロールを指定する必要があります。サーブレットによる承認の制約を設定する方法については、サーブレットの仕様を参照してください。

クライアント証明書のフェッチ

SSL を有効にしてクライアント証明書の承認を行う場合、サーブレットは次の例のようにクライアント証明書へアクセスします。

```
if (request.isSecure()) {
   java.security.cert.X509Certificate[] certs;
   certs = request.getAttribute("javax.servlet.request.X509Certificate");
   if (certs != null) {
      clientCert = certs[0];
      if (clientCert != null) {
         // ユーザーの識別名を取得します。
         java.security.Principal userDN = clientCert.getSubjectDN();
         ...
      }
   }
}
```

userDn はユーザーの完全修飾識別名です。

SHTML と CGI のセキュリティ

セキュリティ確保のために、サーバーでパースされる HTML タグや CGI スクリプトは、サーバーのセキュリティの設定に依存します。次の J2EE 固有のセキュリティ機能は、サーバーでパースされる HTML タグや CGI スクリプトには使用できません。

- J2EE レルム
- J2EE ロール
- フォームベースログイン
- シングルサインオン
- プログラムによるログイン
- J2EE 承認の制約

サーバーセキュリティの設定の詳細については、『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。

第 6 章

Web モジュールの構築と配備

この章では、Sun ONE Application Server で Web モジュールを構築して配備する方法について説明します。構築と配備の概要については、『Sun ONE Application Server 開発者ガイド』を参照してください。

この章には次の節があります。

- Web アプリケーションの構造
- Web 配備記述子の作成
- Web アプリケーションの配備
- Web アプリケーションの動的再読み込み
- `sun-web-app_2_3-0.dtd` ファイル
- `sun-web.xml` ファイルの要素
- Web モジュール XML ファイルのサンプル

Web アプリケーションの構造

Web アプリケーションはディレクトリ構造を持っていて、すべてアプリケーションのドキュメントルート (`/hello` など) に対するマッピングからアクセスできます。ドキュメントルートには、JSP ファイル、HTML ファイル、およびイメージファイルなどの静的ファイルが含まれます。

WAR (Web アプリケーションアーカイブ) ファイルには、完全な Web アプリケーションが圧縮形式で格納されます。

アプリケーションに関連するファイルで、アプリケーションの一般のドキュメントツリーに含まれないものはすべて、ドキュメントルートの下の `WEB-INF` という特殊なディレクトリに置かれます。`WEB-INF` 内のファイルはどれも、直接クライアントに提供することはできません。`WEB-INF` に置かれる内容は次のとおりです。

- /WEB-INF/classes/* (サーブレットやほかのクラスのディレクトリ)
- /WEB-INF/lib/*.jar (Beans やほかのユーティリティクラスを格納した JAR ファイルのディレクトリ)
- /WEB-INF/web.xml および /WEB-INF/sun-web.xml (XML ベースの配備記述子。マッピング、初期化パラメータ、およびセキュリティ制約を含む Web アプリケーション設定を指定する)

Web アプリケーションのディレクトリ構造は、J2EE 仕様に示された構造に従います。簡単な Web アプリケーションのディレクトリ構造の例を次に示します。

```
+ hello/
|--- index.jsp
|--+ META-INF/
|  |--- MANIFEST.MF
'--+ WEB-INF/
   |--- web.xml
   '--- sun-web.xml
```

Web モジュールを含む簡単な J2EE アプリケーションのディレクトリ構造の例を次に示します。

```
+ converter_1/
|--- converterClient.jar
|--+ META-INF/
|  |--- MANIFEST.MF
|  |--- application.xml
|  '--- sun-application.xml
|--+ war-ic_war/
|  |--- index.jsp
|  |--+ META-INF/
|  |  |--- MANIFEST.MF
|  '--+ WEB-INF/
|     |--- web.xml
|     '--- sun-web.xml
|--+ ejb-jar-ic_jar/
|  |--- Converter.class
|  |--- ConverterBean.class
|  |--- ConverterHome.class
|  '--+ META-INF/
|     |--- MANIFEST.MF
|     |--- ejb-jar.xml
|     '--- sun-ejb-jar.xml
'--+ app-client-ic_jar/
   |--- ConverterClient.class
   '--+ META-INF/
      |--- MANIFEST.MF
      |--- application-client.xml
      '--- sun-application-client.xml
```

Web 配備記述子の作成

Sun ONE Application Server の Web モジュールには、次の 2 つの配備記述子ファイルが含まれています。

- J2EE 標準ファイル (`web.xml`)。詳細については、Java サーブレット仕様バージョン 2.3 の第 13 章「Deployment Descriptor」を参照。この仕様は次の Web サイトにある

  `http://java.sun.com/products/servlet/index.html`

- Sun ONE Application Server 固有のオプションファイル (`sun-web.xml`)。詳細については、この章を参照

`web.xml` ファイルと `sun-web.xml` ファイルを作成し、Web モジュールを配備するには、管理インタフェースまたは Sun ONE Studio 4 を使用するのが最も簡単な方法です。詳細については、次の節または『Sun ONE Application Server 開発者ガイド』を参照してください。`web.xml` ファイルと `sun-web.xml` ファイルの例については、129 ページの「Web モジュール XML ファイルのサンプル」を参照してください。

これらのファイルを作成した後に、管理インタフェースまたは、エディタとコマンド行ユーティリティ (Ant など) の組み合わせを使用して、再構築および再配備を行い配備記述子の情報を更新できます。Apache Ant 1.4.1 は、Sun ONE Application Server に付属しています。詳細については、『Sun ONE Application Server 開発者ガイド』を参照してください。

Web アプリケーションの配備

Web アプリケーションの配備、配備取り消し、および再配備を行うときに、サーバーを再起動する必要はありません。つまり、配備は動的に行われます。

Web アプリケーションを配備するには次の方法があります。ここではそれぞれの方法について、簡単に説明します。

- コマンド行の使用
- 管理インタフェースの使用
- Sun ONE Studio の使用

配備の詳細については、『Sun ONE Application Server 開発者ガイド』を参照してください。

JSP 用に生成されたソースを保存するには、sun-web.xml ファイル内の jsp-config 要素に -keepgenerated プロパティを追加します。Web アプリケーションを配備するときにこのプロパティを追加すると、生成されたソースが保存されます。保存先は、アプリケーションの場合は *instance_dir*/generated/jsp/j2ee-apps/*app_name*/*module_name*、個別に配備された Web モジュールの場合は *instance_dir*/generated/jsp/j2ee-modules/*module_name* です。

コマンド行の使用

コマンド行を使って Web アプリケーションを配備するには、次の手順を実行します。

1. 配備記述子ファイル (web.xml および sun-web.xml) を手動で編集します。
2. Ant ビルドコマンド (build war など) を実行して適切な WAR モジュールを再構築します。
3. 必要に応じて、Web アプリケーションを WAR ファイルに書き込みます。これはオプションです。次に例を示します。

   ```
   jar -cvf module_name.war *
   ```

4. asadmin deploy コマンドを使って WAR モジュールを配備します。構文は次のとおりです。

   ```
   asadmin deploy --user admin_user [--password admin_password]
   [--passwordfile password_file] --host hostname --port adminport [--secure
   | -s] [--virtualservers virtual_servers] [--type
   application|ejb|web|connector] [--contextroot contextroot]
   [--force=true] [--precompilejsp=false] [--verify=false] [--name
   component_name] [--upload=true] [--retrieve local_dirpath] [--instance
   instance_name] filepath
   ```

   たとえば、次のコマンドは Web アプリケーションを個別モジュールとして配備します。

   ```
   asadmin deploy --user jadams --password secret --host localhost
   --port 4848 --type web --instance server1 myWebApp.war
   ```

   upload が false に設定されている場合、*filepath* はサーバーマシン上の絶対パスで指定します。

管理インタフェースの使用

管理インタフェースを使って Web アプリケーションを配備するには、次の手順を実行します。

1. サーバーインスタンスの下にあるアプリケーションコンポーネントを開きます。
2. 「Web アプリケーション」ページに移動します。
3. 「配備」ボタンをクリックします。
4. WAR モジュールへのフルパスを入力するか、「ブラウズ」をクリックしてこのモジュールを探し、「了解」ボタンをクリックします。
5. Web アプリケーション名とコンテキストルートを入力します。

   Web アプリケーションが存在する場合は、ボックスにチェックマークを付けて、このアプリケーションを再配備できます。これはオプションです。
6. 仮想サーバー名の横のボックスにチェックマークを付けて、Web アプリケーションを 1 個以上の仮想サーバーに割り当てます。
7. 「了解」ボタンをクリックします。

Sun ONE Studio の使用

Sun ONE Studio 4 を使って、Web アプリケーションを構築および配備できます。Sun ONE Studio の詳細については、Sun ONE Studio 4, Enterprise Edition のチュートリアルを参照してください。

注	Sun ONE Studio では、Web アプリケーションを配備することを「実行」と呼んでいます。

Web アプリケーションの動的再読み込み

Web アプリケーションのコードを変更した場合、動的再読み込みが有効になっていれば、Web アプリケーションの再配備やサーバーの再起動を行う必要はありません。動的再読み込みを有効にするには、次のいずれかを行います。

- 管理インタフェースを使用する
  a. サーバーインスタンスの下にあるアプリケーションコンポーネントを開きます。
  b. 「アプリケーション」ページに移動します。
  c. 「再読み込みを有効」ボックスをオンにして動的再読み込みを有効にします。
  d. 「再読込のポーリング間隔」フィールドに秒数を入力して、アプリケーションとモジュールにコードの変更がないか確認して動的に再読み込みする間隔を設定します。
  e. 「Save」ボタンをクリックします。
  f. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

  詳細については、『Sun ONE Application Server 管理者ガイド』を参照してください。
- `server.xml` ファイルの `applications` 要素の次の属性を編集後、サーバーを再起動する
  - `dynamic-reload-enabled="true"` に設定して、動的再読み込みを有効にします。
  - `dynamic-reload-poll-interval-in-seconds` で、アプリケーションとモジュールにコードの変更がないか確認して動的に再読み込みする間隔を設定します。

  `server.xml` の詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

さらに、新しいサーブレットファイルの読み込み、変更に関連する EJB の再読み込み、または配備記述子の変更の再読み込みを行うには、次の操作を行う必要があります。

1. 配備されたアプリケーションのルートに `.reload` という名前の空のファイルを作成します。

   instance_dir`/applications/j2ee-apps/`*app_name*`/.reload`

   または個別に配備されたモジュールに作成します。

   instance_dir`/applications/j2ee-modules/`*module_name*`/.reload`

2. 上記の変更を行うたびに、`.reload` ファイルのタイムスタンプ (UNIX では `touch.reload`) を明示的に更新します。

JSP の場合、変更は自動的に再読み込みされます。その周期は、sun-web.xml ファイル内の jsp-config 要素の reload-interval プロパティに設定します。JSP の動的再読み込みを無効にするには、reload-interval プロパティの値を -1 に設定します。

sun-web-app_2_3-0.dtd ファイル

sun-application_2_3-0.dtd ファイルは、sun-web.xml ファイルで使用できる要素とサブ要素、およびこれらの要素に指定できる属性などの、ファイル構造を定義します。sun-web-app_2_3-0.dtd ファイルは *install_dir*/lib/dtds ディレクトリにあります。

注	sun-web-app_2_3-0.dtd ファイルは編集しないでください。このファイルの内容は、Sun ONE Application Server のバージョンが変更された場合にのみ変更されます。

DTD ファイルおよび XML の全般的な情報については、次のサイトにある XML 仕様を参照してください。

http://www.w3.org/TR/REC-xml

DTD ファイルに定義された各要素 (対応する XML ファイル内に置かれている場合もある) には、次の要素が含まれています。

- サブ要素
- データ
- 属性

サブ要素

要素にはサブ要素を含めることができます。たとえば、次のファイルコードは cache 要素を定義します。

```
<!ELEMENT cache (cache-helper*, default-helper?, property*, cache-mapping*)>
```

この ELEMENT タグは、cache 要素には cache-helper、default-helper、property、および cache-mapping という各サブ要素を含めることができると指定しています。

次の表に、サブ要素のサフィックス文字 (省略可能) によって決まる必要指定数、つまり指定可能なサブ要素の数を示します。左の列にサブ要素の末尾文字、右の列に対応する必要指定数を示します。

必要指定数とサブ要素のサフィックス

サブ要素のサフィックス	必要指定数
element`*`	このサブ要素を含まないか、1 個以上含めることができる
element`?`	このサブ要素を含まないか、1 個含めることができる
element`+`	このサブ要素を 1 個以上含まなければならない
element (サフィックスなし)	このサブ要素を 1 個だけ含まなければならない

要素にほかの要素を含めることができない場合は、カッコで囲まれた要素名のリストの代わりに、`EMPTY` または `(#PCDATA)` が表示されます。

データ

要素の中には、サブ要素の代わりに文字データを含むものもあります。これらの要素は、次の形式で定義されます。

```
<!ELEMENT element-name (#PCDATA)>
```

次に例を示します。

```
<!ELEMENT description (#PCDATA)>
```

`sun-web.xml` ファイルでは、データ要素の中の空白はデータの一部として扱われます。そのため、データ要素で区切られたデータの前後には余分な空白がないようにする必要があります。次に例を示します。

```
<description>class name of session manager</description>
```

属性

ATTLIST タグを持つ要素には属性 (名前 - 値のペア) が含まれています。次に例を示します。

```
<!ATTLIST cache    max-capacity    CDATA     "4096"
                   timeout         CDATA     "30"
                   enabled         %boolean; "false">
```

cache 要素には、max-capacity、timeout、または enabled という属性を指定することができます。

#REQUIRED ラベルは、値を指定する必要があることを示します。#IMPLIED ラベルは、その属性の指定は省略可能であり、Sun ONE Application Server がデフォルト値を生成することを示します。可能な場合は、"true" などの明示的なデフォルト値が示されます。

属性宣言は、属性のタイプを指定します。たとえば、CDATA は文字データ、%boolean は事前定義された列挙型データを表します。

sun-web.xml ファイルの要素

この節では、sun-web.xml ファイル内の XML 要素について説明します。要素は次のグループに分類されます。

- 全般的な要素
- セキュリティに関する要素
- セッションに関する要素
- 参照に関する要素
- キャッシュに関する要素
- クラスローダーに関する要素
- JSP に関する要素
- 国際化に関する要素

注　特に指定しない限り、サブ要素は各**サブ要素**欄に示されている順に定義する必要があります。

全般的な要素

全般的な要素は次のとおりです。

- sun-web-app
- property
- description

sun-web-app

Web モジュールに対する Sun ONE Application Server 固有の設定を定義します。sun-web-app 要素はルート要素であり、1 つの sun-web.xml ファイルに 1 つだけ存在します。

サブ要素

次の表は、sun-web-app 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

sun-web-app のサブ要素

要素	必要指定数	説明
security-role-mapping	0 または 1 個以上	現在のアクティブレルムに存在するユーザーやグループにロールを割り当てる
servlet	0 または 1 個以上	サーブレットの主体名を指定する。この名前は、web.xml で定義された run-as ロールに使用される
session-config	0 または 1 個	セッションマネージャやセッション cookie など、セッション関連の情報を指定する
resource-env-ref	0 または 1 個以上	対応する J2EE XML ファイルの resource-env-ref に JNDI 絶対名を割り当てる
resource-ref	0 または 1 個以上	対応する J2EE XML ファイルの resource-ref に JNDI 絶対名を割り当てる
ejb-ref	0 または 1 個以上	対応する J2EE XML ファイルの ejb-ref に JNDI 絶対名を割り当てる
cache	0 または 1 個	Web アプリケーションコンポーネントのキャッシュ機能を設定する
class-loader	0 または 1 個	クラスローダーの設定情報を指定する

sun-web-app のサブ要素 (続き)

要素	必要指定数	説明
jsp-config	0 または 1 個	JSP の設定情報を指定する
locale-charset-info	0 または 1 個	国際化設定を指定する
property	0 または 1 個以上	名前と値を持つプロパティを指定する

属性

なし

プロパティ

次の表は、sun-web-app 要素のプロパティを示します。左の列にプロパティの名前、中央の列にデフォルト値、右の列にそのプロパティの説明を示します。

sun-web-app のプロパティ

プロパティ名	デフォルト値	説明
crossContextAllowed	true	true の場合は、Web アプリケーションが ServletContext.getContext() メソッドを使ってほかの Web アプリケーションのコンテキストにアクセスすることを許可する
tempdir	*instance_dir*/generated/j2ee-apps/*app_name* または *instance_dir*/generated/j2ee-modules/*module_name*	この Web モジュールで使用する一時ディレクトリを指定する。この値を使って javax.servlet.context.tempdir コンテキスト属性の値が作成される。コンパイルされた JSP もこのディレクトリに置かれる
singleThreadedServletPoolSize	5	Web アプリケーション内の個々の SingleThreadModel サーブレットに割り当てられるサーブレットインスタンスの最大数を指定する

property

名前と値を持つプロパティを指定します。property は、次のいずれかまたは両方の親要素に設定情報を追加します。

- Sun ONE Application Server の省略可能な要素
- Sun ONE Application Server が認識していない、LDAP サーバーや Java クラスなどのシステムまたはオブジェクトに必要な要素

たとえば、manager-properties 要素には property サブ要素を指定することができます。

```
<manager-properties>
   <property name="reapIntervalSeconds" value="20" />
</manager-properties>
```

manager-properties 要素が使用するプロパティは、親要素である session-manager 要素の persistence-type 属性の値によって決定します。詳細については、session-manager 要素の説明を参照してください。

サブ要素

次の表は、property 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

property のサブ要素

要素	必要指定数	説明
description	0 または 1 個	プロパティの説明文を指定する

属性

次の表は、property 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

property 属性

属性	デフォルト値	説明
name	なし	プロパティの名前を指定する
value	なし	プロパティの値を指定する

description

この要素を含んでいる要素に関する説明文を指定するデータです。

サブ要素

なし

属性

なし

セキュリティに関する要素

セキュリティに関する要素は次のとおりです。

- `security-role-mapping`
- `servlet`
- `servlet-name`
- `role-name`
- `principal-name`
- `group-name`

security-role-mapping

現在のアクティブレルムに存在するユーザーやグループにロールを割り当てます。現在のアクティブレルムを定義する方法については、『Sun ONE Application Server 開発者ガイド』を参照してください。

サブ要素

次の表は、`security-role-mapping` 要素のサブ要素について示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

`security-role-mapping` のサブ要素

要素	必要指定数	説明
`role-name`	1 個のみ	ロール名
`principal-name`	1 個以上の `principal-name` または `group-name` が必要	現在のレルムに存在する主体 (ユーザー) 名
`group-name`	1 個以上の `principal-name` または `group-name` が必要	現在のレルムに存在するグループ名

属性
なし

servlet

サーブレットの主体名を指定します。この名前は、web-xml で定義された run-as ロールに使用されます。

サブ要素
次の表は、servlet 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

servlet のサブ要素

要素	必要指定数	説明
servlet-name	1 個のみ	サーブレットの名前。この名前は web.xml 内の servlet-name と一致する
principal-name	1 個のみ	現在のレルムに存在する主体 (ユーザー) 名

属性
なし

servlet-name

サーブレットの名前を指定するデータです。この名前は web.xml 内の servlet-name と一致します。この名前は web.xml 内に存在している必要があります。

サブ要素
なし

属性
なし

role-name

web.xml ファイル内の security-role 要素の role-name を指定するデータです。

サブ要素
なし

属性
なし

principal-name

現在のレルムに存在する主体 (ユーザー) 名を指定するデータです。

サブ要素
なし

属性
なし

group-name

現在のレルムに存在するグループ名を指定するデータです。

サブ要素
なし

属性
なし

セッションに関する要素

セッションに関する要素は次のとおりです。

- `session-config`
- `session-manager`
- `manager-properties`
- `store-properties`
- `session-properties`
- `cookie-properties`

注 セッションマネージャインタフェースは不確定なインタフェースです。不確定なインタフェースは試験的または一時的なインタフェースであるため、次のリリースで互換性がなくなったり、削除されたり、または安定したインタフェースに置き換えられたりする場合があります。

session-config

セッションの設定情報を指定します。

サブ要素

次の表は、session-config 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

session-config のサブ要素

要素	必要指定数	説明
session-manager	0 または 1 個	セッションマネージャの設定情報を指定する
session-properties	0 または 1 個	セッションのプロパティを指定する
cookie-properties	0 または 1 個	セッション cookie のプロパティを指定する

属性

なし

session-manager

セッションマネージャの情報を指定します。

サブ要素

次の表は、session-manager 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

session-manager のサブ要素

要素	必要指定数	説明
manager-properties	0 または 1 個	セッションマネージャのプロパティを指定する
store-properties	0 または 1 個	セッションの持続性 (格納) のプロパティを指定する

属性

次の表は、session-manager 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

session-manager の属性

属性	デフォルト値	説明
persistence-type	memory	(省略可能) セッションの持続性メカニズムを指定する。有効な値は memory および file custom 値は実装されないので使用できない

manager-properties

セッションマネージャのプロパティを指定します。

サブ要素

次の表は、manager-properties 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

manager-properties のサブ要素

要素	必要指定数	説明
property	0 または 1 個以上	名前と値を持つプロパティを指定する

属性

なし

プロパティ

次の表は、manager-properties 要素のプロパティを示します。左の列にプロパティの名前、中央の列にデフォルト値、右の列にそのプロパティの説明を示します。

manager-properties のプロパティ

プロパティ名	デフォルト値	説明
reapIntervalSeconds	60	期限切れのセッションをチェックする周期を秒単位で指定する セッションデータが変更される頻度より小さい値を設定することを推奨する。たとえば、頻繁にアクセスされる Web サイト上のヒットカウンタサーブレットの場合、この値をできるだけ小さくする必要がある (1 秒)。そうしないと、サーバーを再起動するたびに最後の数回分のヒットが失われる可能性がある

manager-properties のプロパティ (続き)

プロパティ名	デフォルト値	説明
maxSessions	-1	アクティブセッションの最大数を指定する。制限しない場合は -1 (デフォルト値)
sessionFilename	なし。再起動すると、その前の状態は破棄される	アプリケーションの再起動の間にセッション状態を保持するファイルの絶対パスまたは相対パスを指定する (状態の保持が可能な場合)。相対パスは、この Web モジュールの一時ディレクトリを基準とした場所を示す session-manager 要素の persistence-type 属性が memory の場合にのみ設定可能

store-properties

セッションの持続性 (格納) のプロパティを指定します。

サブ要素

次の表は、store-properties 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

store-properties のサブ要素

要素	必要指定数	説明
property	0 または 1 個以上	名前と値を持つプロパティを指定する

属性

なし

プロパティ

次の表は、store-properties 要素のプロパティを示します。左の列にプロパティの名前、中央の列にデフォルト値、右の列にそのプロパティの説明を示します。

store-properties のプロパティ

プロパティ名	デフォルト値	説明
reapIntervalSeconds	60	現在スワップアウトされているセッションについて期限切れをチェックする周期を秒単位で指定する セッションデータが変更される頻度より小さい値を設定することを推奨する。たとえば、頻繁にアクセスされる Web サイト上のヒットカウンタサーブレットの場合、この値をできるだけ小さくする必要がある (1 秒)。そうしないと、サーバーを再起動するたびに最後の数回分のヒットが失われる可能性がある
directory	javax.servlet.context.tempdir コンテキスト属性によって指定されたディレクトリ	個々のセッションファイルが書き込まれるディレクトリの絶対パスまたは相対パスを指定する。相対パスは、この Web モジュールの一時作業ディレクトリを基準とする

session-properties

セッションのプロパティを指定します。

サブ要素

次の表は、session-properties 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

session-properties のサブ要素

要素	必要指定数	説明
property	0 または 1 個以上	名前と値を持つプロパティを指定する

属性

なし

プロパティ

次の表は、session-properties 要素のプロパティを示します。左の列にプロパティの名前、中央の列にデフォルト値、右の列にそのプロパティの説明を示します。

session-properties のプロパティ

プロパティ名	デフォルト値	説明
timeoutSeconds	600	この Web モジュールで作成されたすべてのセッションについて、非アクティブな状態が継続する最長時間のデフォルトの値を秒単位で指定する。0 またはそれ以下の値を指定した場合、この Web モジュールのセッションは期限切れになることがない web.xml ファイルに session-timeout 要素を指定した場合、session-timeout 値は timeoutSeconds 値をオーバーライドする。session-timeout も timeoutSeconds も指定しなかった場合、timeoutSeconds のデフォルトの値が使用される web.xml には session-timeout 要素を秒単位ではなく分単位で指定する
enableCookies	true	true を指定すると、セッションの追跡に cookie が使用される
enableURLRewriting	true	URL の書き換えを有効にする。ブラウザが cookie を受け入れない場合に、URL の書き換えを通してセッションを追跡できる。ただし、サーブレットや JSP で encodeURL または encodeRedirectURL の呼び出しを使用することも必要
idLengthBytes	128	Web モジュールのセッション ID の長さをバイト単位で指定する

cookie-properties

セッション cookie のプロパティを指定します。

サブ要素

次の表は、cookie-properties 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

cookie-properties のサブ要素

要素	必要指定数	説明
property	0 または 1 個以上	名前と値を持つプロパティを指定する

属性

なし

プロパティ

次の表は、cookie-properties 要素のプロパティを示します。左の列にプロパティの名前、中央の列にデフォルト値、右の列にそのプロパティの説明を示します。

cookie-properties のプロパティ

プロパティ名	デフォルト値	説明
cookieName	JSESSIONID	セッションの追跡に使用する cookie の名前を指定する
cookiePath	Web モジュールがインストールされているコンテキストパス	cookie の作成時に設定されるパス名を指定する。要求のパス名にこのパス名が含まれている場合、ブラウザは cookie を送信する。/ (スラッシュ) を設定すると、Sun ONE Application Server が提供するすべての URL に対してブラウザは cookie を送信する。より狭いパスを設定すると、ブラウザから cookie を送信する要求 URL を制限できる
cookieMaxAgeSeconds	-1	cookie の有効期限を秒単位で指定する。ブラウザでは、この時間が経過した cookie は期限切れとなる
cookieDomain	(設定なし)	cookie が有効となるドメインを指定する
cookieComment	Sun ONE Application Server Session Tracking Cookie	cookie ファイル内でセッション追跡 cookie を識別するコメントを指定する。アプリケーションでは、この cookie についてより詳細なコメントを指定できる

参照に関する要素

参照に関する要素は次のとおりです。

- resource-env-ref
- resource-env-ref-name
- resource-ref
- res-ref-name
- default-resource-principal
- name
- password
- ejb-ref
- ejb-ref-name
- jndi-name

resource-env-ref

対応する J2EE web.xml ファイルの resource-env-ref エントリ内の res-ref-name に、リソースの JNDI 絶対名 jndi-name を割り当てます。

サブ要素

次の表は、resource-env-ref 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

resource-env-ref のサブ要素

要素	必要指定数	説明
resource-env-ref-name	1 個のみ	対応する J2EE web.xml ファイルの resource-env-ref エントリ内の res-ref-name を指定する
jndi-name	1 個のみ	リソースの JNDI 絶対名 jndi-name を指定する

属性

なし

resource-env-ref-name

対応する J2EE web.xml ファイルの resource-env-ref エントリ内の res-ref-name を指定するデータです。

サブ要素
なし

属性
なし

resource-ref

対応する J2EE web.xml ファイルの resource-ref エントリ内の res-ref-name に、リソースの JNDI 絶対名 jndi-name を割り当てます。

サブ要素
次の表は、resource-ref 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

resource-ref のサブ要素

要素	必要指定数	説明
res-ref-name	1 個のみ	対応する J2EE web.xml ファイルの resource-ref エントリ内の res-ref-name を指定する
jndi-name	1 個のみ	リソースの JNDI 絶対名 jndi-name を指定する
default-resource-principal	0 または 1 個	リソースのデフォルトの主体 (ユーザー) を指定する

属性
なし

res-ref-name

対応する J2EE web.xml ファイルの resource-ref エントリ内の res-ref-name を指定するデータです。

サブ要素
なし

属性
なし

default-resource-principal

リソースのデフォルトの主体 (ユーザー) を指定します。

この要素を JMS Connection Factory リソースとともに使用する場合、サブ要素 `name` および `password` は Sun ONE Message Queue のブローカユーザーリポジトリ内の有効なエントリでなければなりません。詳細については、『Sun ONE Message Queue 管理者ガイド』の「セキュリティ管理」の章を参照してください。

サブ要素
次の表は、`default-resource-principal` 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

`default-resource-principal` のサブ要素

要素	必要指定数	説明
`name`	1 個のみ	主体の名前
`password`	1 個のみ	主体のパスワード

属性
なし

name

主体の名前を指定するデータです。

サブ要素
なし

属性
なし

password

主体のパスワードを指定するデータです。

サブ要素
なし

属性
なし

ejb-ref

対応する J2EE `ejb-jar.xml` ファイルの `ejb-ref` エントリ内の `ejb-ref-name` に、リソースの JNDI 絶対名 `jndi-name` を割り当てます。

サブ要素
次の表は、`ejb-ref` 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

`ejb-ref` のサブ要素

要素	必要指定数	説明
`ejb-ref-name`	1 個のみ	対応する J2EE `ejb-jar.xml` ファイルの `ejb-ref` エントリ内の `ejb-ref-name` を指定する
`jndi-name`	1 個のみ	リソースの JNDI 絶対名 `jndi-name` を指定する

属性
なし

ejb-ref-name

対応する J2EE `ejb-jar.xml` ファイルの `ejb-ref` エントリ内の `ejb-ref-name` を指定するデータです。

サブ要素
なし

属性
なし

jndi-name

URL リソースまたは `server.xml` ファイル内のリソースの、JNDI 絶対名 `jndi-name` を指定するデータです。

注 JNDI 内のその他のエンタープライズリソース名との衝突や移植性の問題を回避するため、Sun ONE Application Server アプリケーション内のすべての名前をすべて、文字列 `java:comp/env` で始める必要があります。

サブ要素
なし

属性
なし

キャッシュに関する要素

サーブレット関連の応答キャッシュの詳細については、46 ページの「サーブレットの結果のキャッシュ」を参照してください。JSP のキャッシュの詳細については、58 ページの「JSP キャッシュ」を参照してください。

キャッシュに関する要素は次のとおりです。

- `cache`
- `cache-helper`
- `default-helper`
- `cache-mapping`
- `url-pattern`
- `timeout`
- `http-method`
- `key-field`
- `constraint-field`
- `value`

cache

Web アプリケーションコンポーネントのキャッシュ機能を設定します。

サブ要素
次の表は、`cache` 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

`cache` のサブ要素

要素	必要指定数	説明
`cache-helper`	0 または 1 個以上	`CacheHelper` インタフェースを実装するカスタムクラスを指定する

cache のサブ要素 (続き)

要素	必要指定数	説明
default-helper	0 または 1 個	この要素を使って、組み込まれている cache-helper クラスのデフォルトのプロパティを変更できる
property	0 または 1 個以上	名前と値を持つ cache プロパティを指定する
cache-mapping	0 または 1 個以上	URL パターンまたはサーブレット名に、そのキャッシュ制約条件を割り当てる

属性

次の表は、cache 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

cache の属性

属性	デフォルト値	説明
max-entries	4096	(省略可能) キャッシュに格納できるエントリの最大数を指定する。正の整数であること
timeout-in-seconds	30	(省略可能) エントリが作成または更新された時点からキャッシュ内に保持される最大期間を秒単位で指定する。timeout 要素を使ってオーバーライドできる
enabled	false	(省略可能) サーブレットと JSP のキャッシュを有効にするかどうかを指定する。有効な値は、on、off、yes、no、1、0、true、false

プロパティ

次の表は、cache 要素のプロパティを示します。左の列にプロパティの名前、中央の列にデフォルト値、右の列にそのプロパティの説明を示します。

cache のプロパティ

プロパティ	デフォルト値	説明
cacheClassName	com.sun.appserv.web.cache.LruCache	キャッシュ機能を実装するクラスの完全修飾名を指定する。指定できる値については、後述の表「cacheClassName の値」を参照

cache のプロパティ (続き)

プロパティ	デフォルト値	説明
MultiLRUSegmentSize	4096	独自の LRU (過去に使用されたエントリの古い順) リストを持つ、キャッシュテーブルの 1 セグメントのエントリ数を指定する。cacheClassName に com.sun.appserv.web.cache.MultiLruCache を設定した場合のみ、このプロパティを設定する必要がある
MaxSize	無制限 (Long.MAX_VALUE)	キャッシュのメモリーサイズの上限値を K バイト単位または M バイト単位で指定する。たとえば、32 KB や 2 MB などの値を指定する。cacheClassName に com.sun.appserv.web.cache.BoundedMultiLruCache を設定した場合のみ、このプロパティを設定する必要がある

キャッシュクラス名

次の表は、cacheClassName プロパティに指定できる値を示します。左の列に値、右の列にその値が指定するキャッシュの種類を示します。

cacheClassName の値

値	説明
com.sun.appserv.web.cache.LruCache	LRU (過去に使用されたエントリの古い順) キャッシュから置換されるポリシーを持つ有限キャッシュ
com.sun.appserv.web.cache.BaseCache	エントリの最大数がわかっている場合に適する無制限キャッシュ
com.sun.appserv.web.cache.MultiLruCache	多数のエントリ (>4096) に適するキャッシュ。MultiLRUSegmentSize プロパティを使用する
com.sun.appserv.web.cache.BoundedMultiLruCache	エントリ数ではなくメモリーでキャッシュサイズを制限する場合に適するキャッシュ。MaxSize プロパティを使用する

cache-helper

CacheHelper インタフェースを実装するクラスを指定します。詳細については、49 ページの「CacheHelper インタフェース」を参照してください。

サブ要素

次の表は、cache-helper 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

cache-helper のサブ要素

要素	必要指定数	説明
property	0 または 1 個以上	名前と値を持つプロパティを指定する

属性

次の表は、cache-helper 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

cache-helper の属性

属性	デフォルト値	説明
name	デフォルト	ヘルパークラスの一意の名前を指定する。この名前は cache-mapping 要素で参照される
class-name	なし	キャッシュヘルパーの完全修飾クラス名を指定する。このキャッシュヘルパーは com.sun.appserv.web.CacheHelper インタフェースを実装する必要がある

default-helper

この要素を使って、組み込まれている default cache-helper クラスのプロパティを変更できます。

サブ要素

次の表は、default-helper 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

default-helper のサブ要素

要素	必要指定数	説明
property	0 または 1 個以上	名前と値を持つプロパティを指定する

属性
なし

プロパティ
次の表は、default-helper 要素のプロパティを示します。左の列にプロパティの名前、中央の列にデフォルト値、右の列にそのプロパティの説明を示します。

default-helper のプロパティ

プロパティ	デフォルト値	説明
cacheKeyGeneratorAttrName	組み込まれている default cache-helper のキー生成を使用する。この場合、サーブレットのパスに key-field の値が連結されてキーが生成される	キャッシュエンジンは ServletContext 内でこのプロパティに指定された値と同等の名前の属性を検索し、カスタマイズされた CacheKeyGenerator が使用されているかどうかを判断する。アプリケーションでカスタムのキージェネレータを用意し、デフォルトのヘルパーの代わりに使用できる 51 ページの「CacheKeyGenerator インタフェース」を参照

cache-mapping
URL パターンまたはサーブレット名に、そのキャッシュ制約条件を割り当てます。

サブ要素
次の表は、cache-mapping 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

cache-mapping のサブ要素

要素	必要指定数	説明
servlet-name	servlet-name または url-pattern が 1 個必要	サーブレットの名前

cache-mapping のサブ要素 (続き)

要素	必要指定数	説明
url-pattern	servlet-name または url-pattern が 1 個必要	キャッシュを有効にする、サーブレットの URL パターン
cache-helper-ref	timeout、refresh-field、http-method、key-field、および constraint-field を使用しない場合は必須	親要素 cache-mapping で使用されている cache-helper の name
timeout	cache-helper-ref を使用しない場合、0 個または 1 個	エントリが作成または更新された時点からキャッシュ内に保持される最大期間を秒単位で指定する。cache-mapping ごとに固有の値
refresh-field	cache-helper-ref を使用しない場合、0 個または 1 個	アプリケーションコンポーネントでプログラムによるキャッシュエントリの更新ができるようにするフィールドを指定する
http-method	cache-helper-ref を使用しない場合、0 個以上	キャッシュに使用できる HTTP メソッド
key-field	cache-helper-ref を使用しない場合、0 個以上	キャッシュエントリの検索や抽出に使用されるキーのコンポーネントを指定する
constraint-field	cache-helper-ref を使用しない場合、0 個以上	特定の url-pattern または servlet-name に対するキャッシュ制約条件を指定する

属性

なし

url-pattern

キャッシュを有効にする、サーブレットの URL パターンを指定するデータです。指定できるパターンについては、サーブレット仕様バージョン 2.3 の SRV. 11.2 節を参照してください。

サブ要素
なし

属性
なし

cache-helper-ref

親要素 cache-mapping で使用されている cache-helper の name を指定するデータです。

サブ要素
なし

属性
なし

timeout

エントリが作成または更新された時点からキャッシュ内に保持される最大期間を秒単位で指定するデータです。cache-mapping ごとに固有の値です。この要素を指定しない場合、デフォルトとして cache 要素の timeout 属性の値が使用されます。

サブ要素
なし

属性
次の表は、timeout 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

timeout の属性

属性	デフォルト値	説明
name	なし	タイムアウトのパラメータを指定する。指定した値は秒単位として解釈される。フィールドの型は java.lang.Long または java.lang.Integer であることが必要

timeout の属性 (続き)

属性	デフォルト値	説明
scope	request.attribute	(省略可能) 指定したパラメータが有効な範囲を指定する。指定できる値は、context.attribute、request.header、request.parameter、request.cookie、request.attribute、および session.attribute

refresh-field

アプリケーションコンポーネントでプログラムによるキャッシュエントリの更新ができるようにするフィールドを指定します。

サブ要素
なし

属性
次の表は、refresh-field 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

refresh-field の属性

属性	デフォルト値	説明
name	なし	パラメータを指定する
scope	request.parameter	(省略可能) 指定したパラメータが有効な範囲を指定する。指定できる値は、context.attribute、request.header、request.parameter、request.cookie、session.id、および session.attribute

http-method

キャッシュに使用できる HTTP メソッドを指定するデータです。デフォルトは、GET です。

サブ要素
なし

属性
なし

key-field

キャッシュエントリの検索や抽出に使用されるキーのコンポーネントを指定します。Web コンテナは、この名前のパラメータ (フィールド) を指定された範囲内で検索します。

この要素が存在しない場合、Web コンテナはサーブレットパスを使用します。サーブレットパスは、現在の要求をアクティブにしたサーブレットマッピングに対応するパスセクションです。サーブレットパスの詳細については、サーブレット仕様バージョン 2.3 の SRV 4.4 節を参照してください。

サブ要素

なし

属性

次の表は、key-field 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

key-field の属性

属性	デフォルト値	説明
name	なし	名前を指定する
scope	request.parameter	(省略可能) 指定したパラメータの有効範囲を指定する。指定できる値は、context.attribute、request.header、request.parameter、request.cookie、session.id、および session.attribute

constraint-field

特定の url-pattern または servlet-name に対するキャッシュ制約条件を指定します。

応答をキャッシュするにはすべての constraint-field 制約を渡す必要があります。value 制約がある場合、少なくともその 1 つを渡す必要があります。

サブ要素

次の表は、constraint-field 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

constraint-field のサブ要素

要素	必要指定数	説明
value	0 または 1 個以上	パラメータの値と比較する条件値

属性

次の表は、constraint-field 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

constraint-field の属性

属性	デフォルト値	説明
name	なし	名前を指定する
scope	request.parameter	(省略可能) 指定したパラメータの有効な範囲を指定する。指定できる値は、context.attribute、request.header、request.parameter、request.cookie、request.attribute、および session.attribute
cache-on-match	true	(省略可能) true を指定すると、条件に一致する応答がキャッシュされる。value サブ要素にある同じ属性よりも優先される
cache-on-match-failure	false	(省略可能) true を指定すると、条件に一致しない応答がキャッシュされる。value サブ要素にある同じ属性よりも優先される

value

入力パラメータの値と比較する条件値を指定するデータです。比較では、大文字と小文字が区別されます。次に例を示します。

```
<value match-expr="in-range">1-60</value>
```

サブ要素

なし

属性

次の表は、value 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

value の属性

属性	デフォルト値	説明
match-expr	equals	(省略可能) 値に対して実行する比較の種類を指定する。指定できる値は、equals、not-equals、greater、lesser、および in-range match-expr に greater または lesser を指定した場合、値は数値であることが必要。match-expr に in-range を指定した場合、値は *n1*-*n2* という形式であることが必要。ここで、*n1* と *n2* は数値
cache-on-match	true	(省略可能) true を指定すると、条件に一致する応答がキャッシュされる
cache-on-match-failure	false	(省略可能) true を指定すると、条件に一致しない応答がキャッシュされる

クラスローダーに関する要素

クラスローダーに関する要素は次のとおりです。

- class-loader

class-loader

Web モジュールのクラスローダーを設定します。

サブ要素

なし

属性

次の表は、class-loader 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

class-loader の属性

属性	デフォルト値	説明
extra-class-path	null	(省略可能) この Web モジュールに対して追加のクラスパス設定を指定する

class-loader の属性 (続き)

属性	デフォルト値	説明
delegate	false	(省略可能) true を指定すると、Web モジュールは標準のクラスローダー委任モデルに従い、ローカルのクラスローダーを調べる前にその親クラスローダーに委任する。false を指定すると、Web モジュールはサーブレット仕様で指定されている委譲モデルに従い、親クラスローダーを調べる前にそのクラスローダーを調べる Web サービスのコンポーネントの場合、この値を true に設定する必要があります。 有効な値は、on、off、yes、no、1、0、true、false

JSP に関する要素

JSP に関する要素は次のとおりです。

- jsp-config

jsp-config

JSP の設定情報を指定します。

サブ要素

次の表は、jsp-config 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

jsp-config のサブ要素

要素	必要指定数	説明
property	0 または 1 個以上	プロパティを指定する

属性

なし

プロパティ

次の表は、jsp-config 要素のプロパティを示します。左の列にプロパティの名前、中央の列にデフォルト値、右の列にそのプロパティの説明を示します。

jsp-config のプロパティ

プロパティ名	デフォルト値	説明
ieClassId	clsid:8AD9C840-044E-11D1-B3E9-00805F499D93	Internet Explorer の Java プラグイン COM クラス ID。<jsp:plugin> タグによって使われる
javaCompilerPlugin	組み込みの JDK コンパイラ (javac)	使用する Java コンパイラプラグインの完全修飾クラス名。デフォルトのコンパイラを使用する場合は不要 たとえば、JSP ページの jikes コンパイラを使用するには、javaCompilerPlugin プロパティの値を org.apache.jasper.compiler.JikesJavaCompiler に設定し、javaCompilerPath プロパティの値が jikes 実行可能ファイルを指すように設定する
javaCompilerPath	なし	jikes など、アウトプロセス Java コンパイラの実行可能ファイルへのパスを指定する。デフォルトコンパイラの場合は無視される。javaCompilerPlugin を指定した場合のみ、このプロパティを設定する必要がある
javaEncoding	UTF8	生成された サーブレットをエンコードする方法を指定する。サーブレットをコンパイルする Java コンパイラにもこのエンコードする方法が渡される。デフォルトでは、コンテナは UTF8 の使用を試みる。これに失敗すると、javaEncoding の値の使用を試みる 使用できるエンコードについては、次の Web サイトを参照 http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html
classdebuginfo	false	生成された Java サーブレットをコンパイルするときにデバッグオプション (javac の場合は -g) を設定するかどうかを指定する
keepgenerated	true	true を設定すると、生成された Java ファイルが保存される。false を設定すると、Java ファイルは削除される
largefile	false	true を設定すると、JSP ファイルのコンパイル時に、静的 HTML が別のデータファイルに格納される。生成される サーブレットのサイズを最小限に抑えることができるため、JSP が非常に大きい場合に役立つ
mappedfile	false	true を設定すると、各 HTML 行の write 呼び出しと、JSP ファイル内での各行の位置を記述するコメントが生成される。デフォルトでは、隣接した write 呼び出しは、すべて結合され、位置のコメントは生成されない

jsp-config のプロパティ (続き)

プロパティ名	デフォルト値	説明
scratchdir	Web アプリケーションのデフォルトの作業ディレクトリ	生成されたコードを格納するために作成された作業ディレクトリ
reload-interval	0	JSP ファイルの変更をチェックする周期を秒単位で指定する。この値に 0 を設定すると、要求のたびに JSP の変更がチェックされる。この値を -1 に設定すると、JSP の変更および再コンパイルのチェックが無効になる

国際化に関する要素

国際化に関する要素は次のとおりです。

- locale-charset-info
- locale-charset-map
- parameter-encoding

locale-charset-info

アプリケーションの国際化の設定に関する情報を指定します。

サブ要素

次の表は、locale-charset-info 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

locale-charset-info のサブ要素

要素	必要指定数	説明
locale-charset-map	1 個以上	文字セットにロケールとエージェントを割り当てる
parameter-encoding	0 または 1 個	Web コンテナでこの Web アプリケーションのフォームからパラメータをデコードする方法を、隠しフィールドの値に従って決定する

属性

次の表は、locale-charset-info 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

locale-charset-info の属性

属性	デフォルト値	説明
default-locale	なし	デフォルトのロケールを指定する

locale-charset-map

文字セットにロケールとエージェントを割り当てます。

使用できるエンコードについては、次の Web サイトを参照してください。

http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html

サブ要素

次の表は、locale-charset-map 要素のサブ要素を示します。左の列にサブ要素名、中央の列に必要指定数、右の列に要素の説明を示します。

locale-charset-map のサブ要素

要素	必要指定数	説明
description	0 または 1 個	マッピングの説明文を指定する

属性

次の表は、locale-charset-map 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

locale-charset-map の属性

属性	デフォルト値	説明
locale	なし	ロケール名を指定する
agent	なし	(省略可能) アプリケーションサーバーと対話するクライアントの種類を指定する。特定のロケールについて、複数のエージェントで異なる優先文字セットを使用することもできる。この属性の値は、クライアントから送信された user-agent HTTP 要求ヘッダーの値と正確に一致する必要がある。詳細については、表「agent 属性値の例」を参照
charset	なし	文字セットを指定する

エージェントの例

次の表は、agent 属性値の例を示します。左の列にエージェント、右の列に対応する属性値を示します。

agent 属性値の例

エージェント	user-agent ヘッダーと agent 属性値
Internet Explorer 5.00 (Windows 2000 用)	`Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)`
Netscape 4.7.7 (Windows 2000 用)	`Mozilla/4.77 [en] (Windows NT 5.0; U)`
Netscape 4.7 (Solaris 用)	`Mozilla/4.7 [en] (X11; u; Sun OS 5.6 sun4u)`

parameter-encoding

要求の `content-type` に `charset` が設定されていない場合に、Web コンテナが使用する文字エンコードを決定する隠しフィールドを指定します。このエンコードは、`request.getParameter` を呼び出すためのパラメータをデコードするのに使用されます。

使用できるエンコードについては、次の Web サイトを参照してください。

`http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html`

サブ要素

なし

属性

次の表は、`parameter-encoding` 要素の属性を示します。左の列に属性の名前、中央の列にデフォルト値、右の列にその属性の説明を示します。

`parameter-encoding` の属性

属性	デフォルト値	説明
`form-hint-field`	なし	パラメータをエンコードする方法を指定するフォーム内の隠しフィールドの名前

Web モジュール XML ファイルのサンプル

この節には次の項目があります。

- web.xml ファイルのサンプル
- sun-web.xml ファイルのサンプル

web.xml ファイルのサンプル

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN'
'http://java.sun.com/j2ee/dtds/web-app_2_2.dtd'>

<web-app>

   <display-name>webapps-simple</display-name>

   <description>

      The jakarta-tomcat-4.0.3 sample apps ports over to S1AS.

   </description>

   <distributable></distributable>

   <servlet>

      <servlet-name>HelloWorldExample</servlet-name>

      <servlet-class>

         samples.webapps.simple.servlet.HelloWorldExample

      </servlet-class>

   </servlet>

   <servlet>

      <servlet-name>RequestHeaderExample</servlet-name>

      <servlet-class>

         samples.webapps.simple.servlet.RequestHeaderExample

      </servlet-class>

   </servlet>

   <servlet>

      <servlet-name>SnoopServlet</servlet-name>
```

```
    <servlet-class>
      samples.webapps.simple.servlet.SnoopServlet
    </servlet-class>
  </servlet>
  <servlet>
    <servlet-name>servletToJsp</servlet-name>
    <servlet-class>
      samples.webapps.simple.servlet.servletToJsp
    </servlet-class>
  </servlet>
  <servlet>
    <servlet-name>RequestInfoExample</servlet-name>
    <servlet-class>
      samples.webapps.simple.servlet.RequestInfoExample
    </servlet-class>
  </servlet>
  <servlet>
    <servlet-name>SessionExample</servlet-name>
    <servlet-class>
      samples.webapps.simple.servlet.SessionExample
    </servlet-class>
  </servlet>
  <servlet>
    <servlet-name>CookieExample</servlet-name>
    <servlet-class>
      samples.webapps.simple.servlet.CookieExample
    </servlet-class>
  </servlet>
  <servlet>
    <servlet-name>RequestParamExample</servlet-name>
    <servlet-class>
```

```
      samples.webapps.simple.servlet.RequestParamExample
   </servlet-class>
</servlet>
<servlet>
   <servlet-name>SendMailServlet</servlet-name>
   <servlet-class>
      samples.webapps.simple.servlet.SendMailServlet
   </servlet-class>
</servlet>
<servlet>
   <servlet-name>JndiServlet</servlet-name>
   <servlet-class>
      samples.webapps.simple.servlet.JndiServlet
   </servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>HelloWorldExample</servlet-name>
   <url-pattern>/helloworld</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>RequestHeaderExample</servlet-name>
   <url-pattern>/helloworld</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>SnoopServlet</servlet-name>
   <url-pattern>/snoop</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>servletToJsp</servlet-name>
   <url-pattern>/servletToJsp</url-pattern>
</servlet-mapping>
```

```
<servlet-mapping>
   <servlet-name>RequestInfoExample</servlet-name>
   <url-pattern>/requestinfo</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>SessionExample</servlet-name>
   <url-pattern>/session</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>CookieExample</servlet-name>
   <url-pattern>/cookie</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>RequestParamExample</servlet-name>
   <url-pattern>/requestparam</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>SendMailServlet</servlet-name>
   <url-pattern>/SendMailServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>JndiServlet</servlet-name>
   <url-pattern>/JndiServlet</url-pattern>
</servlet-mapping>
<welcome-file-list>
   <welcome-file>index.html</welcome-file>
</welcome-file-list>
<taglib>
   <taglib-uri>
      http://java.apache.org/tomcat/examples-taglib
   </taglib-uri>
```

```
      <taglib-location>
         /WEB-INF/tlds/example-taglib.tld
      </taglib-location>
   </taglib>
   <resource-ref>
      <res-ref-name>mail/Session</res-ref-name>
      <res-type>javax.mail.Session</res-type>
      <res-auth>Container</res-auth>
   </resource-ref>
   <security-constraint>
      <web-resource-collection>
         <web-resource-name>Protected Area</web-resource-name>
<! -- 保護されるコンテキスト関連の URL -->
         <url-pattern>/jsp/security/protected/*</url-pattern>
<! -- http メソッドを一覧表示する場合、これらのメソッドだけが保護される -->
         <http-method>DELETE</http-method>
         <http-method>GET</http-method>
         <http-method>POST</http-method>
         <http-method>PUT</http-method>
      </web-resource-collection>
      <auth-constraint>
   <! -- リスト上のいずれかのロールを持つユーザーはこの領域にアクセス
         できる -->
         <role-name>tomcat</role-name>
         <role-name>role1</role-name>
      </auth-constraint>
   </security-constraint>
<! -- 環境エントリの例 -->
   <env-entry>
      <description>
         The maximum number of tax exemptions allowed to be set.
```

```
        </description>
        <env-entry-name>maxExemptions</env-entry-name>
        <env-entry-value>15</env-entry-value>
        <env-entry-type>java.lang.Integer</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>minExemptions</env-entry-name>
        <env-entry-value>1</env-entry-value>
        <env-entry-type>java.lang.Integer</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>foo/name1</env-entry-name>
        <env-entry-value>value1</env-entry-value>
        <env-entry-type>java.lang.String</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>foo/bar/name2</env-entry-name>
        <env-entry-value>true</env-entry-value>
        <env-entry-type>java.lang.Boolean</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>name3</env-entry-name>
        <env-entry-value>1</env-entry-value>
        <env-entry-type>java.lang.Integer</env-entry-type>
    </env-entry>
    <env-entry>
        <env-entry-name>foo/name4</env-entry-name>
        <env-entry-value>10</env-entry-value>
        <env-entry-type>java.lang.Integer</env-entry-type>
    </env-entry>
</web-app>
```

sun-web.xml ファイルのサンプル

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sun-web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Sun ONE
Application Server 7.0 Servlet 2.3//EN'
'http://www.sun.com/software/sunone/appserver/dtds/sun-web-app_2_3-
0.dtd'>

<sun-web-app>

   <session-config>

      <session-manager/>

   </session-config>

   <resource-ref>

      <res-ref-name>mail/Session</res-ref-name>

      <jndi-name>mail/Session</jndi-name>

   </resource-ref>

   <jsp-config/>

</sun-web-app>
```

第 7 章

サーバーでパースされる HTML の使用法

HTML ファイルには、サーバーで実行されるタグを指定することができます。Sun ONE Application Server 7 では、標準のサーバーサイドタグのほかに、サーブレットを埋め込んだ独自のサーバーサイドタグを定義できます。

サーバーでパースされるカスタムの HTML タグを作成できます。詳細については、『Sun ONE Application Server NSAPI Developer's Guide』を参照してください。

セキュリティ確保のために、サーバーでパースされる HTML タグは、サーバーのセキュリティ設定に依存します。詳細については、85 ページの「SHTML と CGI のセキュリティ」および『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。

この章には次の節があります。

- サーバーサイド HTML と J2EE Web アプリケーション
- サーバーサイド HTML の有効化
- サーバーサイド HTML コマンドの使用法
- サーブレットの埋め込み
- 日時の形式

サーバーサイド HTML と J2EE Web アプリケーション

Sun ONE Application Server では、サーバーでパースされる HTML と J2EE Web アプリケーションの相互運用はできません。特に、次の点に注意してください。

- サーバーでパースされる HTML を Web アプリケーションのコンテキストルートに置かない
- サーバーでパースされる HTML の出力をサーブレットや JSP に取り込まない
- サーバーでパースされる HTML へサーブレットや JSP から要求を転送しない
- J2EE の `security-constraint` 機能や `filter-mapping` 機能は、サーバーでパースされる HTML には適用できない

サーバーサイド HTML の有効化

サーバーサイド HTML を有効にするには、次の手順を実行します。

1. 管理インタフェースで、サーバーインスタンスの下の HTTP サーバーコンポーネントを開きます。
2. 「仮想サーバー」ページに移動します。
3. サーバーサイド HTML を有効にする仮想サーバーの名前をクリックします。
4. 「HTTP/HTML」タブをクリックします。
5. 「HTML をパース」オプションをクリックします。
6. サーバーで HTML をパースされる対象のリソースを選択します。

   仮想サーバーか、仮想サーバー内の特定のディレクトリを選択します。

   ディレクトリを選択した場合、サーバーはそのディレクトリかそのディレクトリ内のファイルを表す URL を受信したときだけ HTML をパースします。
7. サーバーでパースされる HTML を有効にするかどうかを選択します。

   HTML ファイルに対してだけ有効にし、`exec` タグに対しては無効にすることができます。または、HTML ファイルと `exec` タグの両方に対して有効にすることもでき、この場合は HTML ファイルでサーバー上のほかのプログラムを実行できます。

8. どのファイルをパースするかを選択します。

   .shtml という拡張子を持つファイルだけをパースするか、すべての HTML ファイルをパースするかを選択できます。後者の場合は処理速度が低下します。UNIX を使用している場合は、実行権限がある UNIX ファイルを選択することもできますが、信頼性は保証されません。

9. 「了解」ボタンをクリックします。

10. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

パースを有効にする場合は、次の指令が init.conf ファイルに追加されていることを確認してください (ネイティブスレッドをオフにします)。

```
Init funcs="shtml_init,shtml_send" shlib="install _dir/bin/Shtml.dll"
NativeThread="no" fn="load-modules"
```

Sun ONE Application Server 7 では NativeThread="no" を設定する必要があります。また、Windows の場合は *install_dir*/bin にある Shtml.dll で、UNIX の場合は *install_dir*/lib にある libShtml.so で、これらの関数が提供されるようになりました。

さらに、次の指令が obj.conf ファイルに追加されていることを確認してください。

```
<Object name="default">
...
...
Service fn="shtml_send" type="magnus-internal/parsed-html" method="(GET|HEAD)"
...
</Object>
```

サーバーサイド HTML コマンドの使用法

この節では、サーバーでパースされるタグを HTML ファイルに組み込むための HTML コマンドについて説明します。これらのコマンドは HTML ファイルに埋め込まれ、`obj.conf` ファイルの `parse-html` 関数によって処理されます。

サーバーは、コマンドとその属性で決定されたデータで、各コマンドを置換します。コマンドの形式は次のとおりです。

`<!--#`*command attribute1 attribute2* `<Body>... -->`

各 `attribute` の形式は、次のような名前 - 値のペアです。

name`="`*value*`"`

コマンドと属性名には小文字を使用してください。

コマンドは HTML コメントの中に隠されているため、サーバーでパースされない場合は無視されます。標準のサーバーサイドコマンドは次のとおりです。

- `config`
- `include`
- `echo`
- `fsize`
- `flastmod`
- `exec`

config

`config` コマンドは、ほかのコマンドの形式を初期化します。

- `errmsg` 属性は、ファイルのパース中にエラーが発生した場合にクライアントへ送信するメッセージを定義する。このエラーはサーバーのログファイルにも記録される
- `timefmt` 属性は、`flastmod` コマンドの日時の形式を指定する。この属性では、`util_strftime` 関数と同じ書式設定文字が使用される。日時のデフォルト形式は `"%A, %d-%b-%y %T"`

  日時の形式の詳細については、144 ページの「日時の形式」を参照してください。
- `sizefmt` 属性は、`fsize` コマンドのファイルサイズの形式を指定する。この属性には、次のいずれかの値を指定できる
  - `bytes` を指定すると、12,345,678 のような自然数としてファイルサイズが報告される

- abbrev (デフォルト) を指定すると、K バイトまたは M バイト単位でファイルサイズが報告される

例

```
<!--#config timefmt="%r %a %b %e, %Y" sizefmt="abbrev"-->
```

このコマンドは、08:23:15 AM Wed Apr 15, 1996 のような日時の形式を指定し、ファイルで使用されている文字数に K バイトまたは M バイト単位で表すファイルサイズ形式を指定します。

include

include コマンドは、パースされるファイルの中にファイルを挿入します。挿入したファイルの中にさらに別のファイルを挿入することによって、パースされるファイルを入れ子にできます。パースされるドキュメントが置かれているディレクトリに対してサーバーがアクセス制御を使用している場合、このドキュメントを要求するクライアントは、組み込まれるファイルに対してもアクセス権を持っている必要があります。

Sun ONE Application Server 7 では、include コマンドに virtual 属性を指定することで、CGI プログラムファイルを組み込むことができます。CGI プログラムを実行するために exec コマンドも使用する必要があります。

- virtual 属性には、サーバー上のファイルの URI を指定する
- file 属性には、現在のディレクトリからの相対パスを指定する。この属性に ../ などの要素を含めることはできない。また、絶対パスは指定できない

例

```
<!--#include file="bottle.gif"-->
```

echo

echo コマンドは、環境変数の値を挿入します。var 属性は、挿入する環境変数を指定します。この変数が見つからない場合は "(none)" が挿入されます。環境変数の一覧については、142 ページの「サーバーサイド HTML コマンド内の環境変数」を参照してください。

例

```
<!--#echo var="DATE_GMT"-->
```

fsize

fsize コマンドは、ファイルのサイズを挿入します。このコマンドの属性は、include コマンドの属性と同じです (virtual および file)。ファイルサイズの形式は、config コマンドの sizefmt 属性で指定されます。

例

```
<!--#fsize file="bottle.gif"-->
```

flastmod

flastmod コマンドは、ファイルが最後に変更された日時を挿入します。このコマンドの属性は、include コマンドの属性と同じです (virtual および file)。日時の形式は、config コマンドの timefmt 属性で指定されます。

例

```
<!--#flastmod file="bottle.gif"-->
```

exec

exec コマンドは、シェルコマンドまたは CGI プログラムを実行します。

- cmd 属性 (UNIX のみ) は、/bin/sh を使ってコマンドを実行する。このコマンドには、任意の特殊環境変数を含めることができる
- cgi 属性は、CGI プログラムを実行し、その出力をパースされるファイルに取り込む

例

```
<!--#exec cgi="workit.pl"-->
```

サーバーサイド HTML コマンド内の環境変数

パースされるコマンドには、CGI で通常使用される一連の環境変数のほかに、次の変数を含めることができます。

- DOCUMENT_NAME

  パースされたファイルの名前

- DOCUMENT_URI

  パースされたファイルへの仮想パス (/shtml/test.shtml など)

- QUERY_STRING_UNESCAPED

  クライアントから送信された検索クエリ。ただし、シェルの特殊文字に付加されていたエスケープ文字 ¥ を外したもの

- DATE_LOCAL

  現地時間による現在の日付と時刻

- DATE_GMT

  グリニッジ標準時による現在の日付と時刻

- LAST_MODIFIED

  ファイルが最後に変更された日時

サーブレットの埋め込み

Sun ONE Application Server 7 は、Java Web Server で導入された <SERVLET> タグをサポートしています。このタグを使うと、サーブレットの出力を SHTML ファイルに埋め込むことができます。この動作を有効にするために設定を変更する必要はありません。SSI とサーブレットがどちらも有効になっていれば、<SERVLET> タグは有効になります。

<SERVLET> タグの構文は、ほかの SSI コマンドの構文とは多少異なり、<APPLET> タグの構文に似ています。

```
<servlet code=code><param name=param1 value=v3><param name=param2
value=v4>.
.
</servlet>
```

code パラメータは、サーブレットの URI を指定します。この URI には、Web アプリケーションのコンテキストルートも含まれます。この URI は、J2EE 配備記述子 (web.xml) 内の servlet-mapping 要素の url-pattern サブ要素と一致する必要があります。

日時の形式

次の表は、サーバーでパースされる HTML に使用する日付と時刻の書式文字列について説明しています。左の列は日時の形式を表す記号、右の列は各記号の説明です。

日時の形式

記号	意味
%a	曜日の名前 (3 文字の短縮形)
%d	月内の日付 (01 から 31 までの 10 進数)
%S	秒 (00 から 59 までの 10 進数)
%M	分 (00 から 59 までの 10 進数)
%H	時 (00 から 23 までの 24 時間形式)
%Y	年 (2099 までの 10 進数)
%b	月の名前 (3 文字の短縮形)
%h	月の名前 (3 文字の短縮形)
%T	`"HH:MM:SS"` 形式の時刻
%X	`"HH:MM:SS"` 形式の時刻
%A	曜日の名前 (完全形)
%B	月の名前 (完全形)
%C	`"%a %b %e %H:%M:%S %Y"`
%c	`"%m/%d/%y %H:%M:%S"` 形式の日付と時刻
%D	`"%m/%d/%y"` 形式の日付
%e	月内の日付 (先頭にゼロを付加しない、1 から 31 までの 10 進数)
%I	時 (01 から 12 までの 12 時間形式)
%j	年内の日付 (001 から 366 までの 10 進数)
%k	時 (先頭にゼロを付加しない、0 から 23 までの 24 時間形式)
%l	時 (先頭にゼロを付加しない、1 から 12 までの 12 時間形式)
%m	月 (01 から 12 までの 10 進数)
%n	改行文字
%p	A.M./P.M. 表示 (12 時間形式の場合)
%R	`"%H:%M"` 形式の時刻
%r	`"%I:%M:%S %p"` 形式の時刻

日時の形式 (続き)

記号	意味
%t	タブ文字
%U	年内の週 (00 から 51 までの 10 進数。日曜日を週の開始日とする)
%w	曜日 (0 から 6 までの 10 進数。日曜日を 0 とする)
%W	年内の週 (00 から 51 までの 10 進数。月曜日を週の開始日とする)
%x	`"%m/%d/%y"` 形式の日付
%y	年 (下 2 桁だけの、00 から 99 までの 10 進数)
%%	パーセント記号

第 8 章

CGI の使用法

CGI (Common Gateway Interface) プログラムは、サーバー上で動作し、要求を送信したクライアントへ返す応答を生成します。CGI プログラムは、C、C++、Perl、シェルスクリプトなどいくつかの言語で作成できます。CGI プログラムは、URL を呼び出すことにより起動します。

CGI プログラムの作成については数多くの情報があります。最初に次の Web サイトの「The Common Gateway Interface」を参照することをお勧めします。

`http://hoohoo.ncsa.uiuc.edu/cgi/overview.html`

Sun ONE Application Server は、CGI バージョン 1.1 仕様に準拠しています。

サーバーは CGI スクリプトやプログラムを実行するたびにプロセスを起動するため、サーバーのプログラミング手法としてはコストが高くなります。

CGI スクリプトのセキュリティは、サーバーのセキュリティ設定次第です。詳細については、85 ページの「SHTML と CGI のセキュリティ」および『Sun ONE Application Server セキュリティ管理者ガイド』を参照してください。

この章では、次のトピックについて説明します。

- CGI と J2EE Web アプリケーション
- CGI の有効化
- CGI プログラムのカスタム実行環境の作成 (UNIX のみ)
- CGI プログラムをサーバーに追加
- CGI プログラムの優先度の設定
- Windows CGI プログラム
- Windows のシェル CGI プログラム
- クエリハンドラ
- Perl CGI プログラム

- CGI のグローバル設定
- CGI 変数

CGI と J2EE Web アプリケーション

Sun ONE Application Server では、CGI プログラムと J2EE Web アプリケーションの相互運用はできません。特に、次の点に注意してください。

- CGI プログラムを Web アプリケーションのコンテキストルートに置かない
- CGI プログラムの出力をサーブレットや JSP に取り込まない
- CGI プログラムへサーブレットや JSP から要求を転送しない
- J2EE の `security-constraint` 機能や `filter-mapping` 機能は、CGI プログラムには適用できない

CGI の有効化

Sun ONE Application Server では、次の方法で CGI プログラムを識別できます。

- CGI 用ディレクトリを指定する : サーバーは、CGI 用ディレクトリにあるすべてのファイルを CGI プログラムとして扱う
- CGI ファイル拡張子を指定する : サーバーは、指定された拡張子を持つすべてのファイルを CGI プログラムとして扱う

CGI 用ディレクトリを指定する

CGI プログラムだけを格納する、CGI 専用のディレクトリを指定するには、次の手順を実行します。

1. 使用中のコンピュータ上に CGI ディレクトリを作成します。このディレクトリは、ドキュメントルートディレクトリのサブディレクトリ以外でもかまいません。手順 7 で URL プレフィックスを指定するのはこのためです。
2. 管理インタフェースで、サーバーインスタンスの下の HTTP サーバーコンポーネントを開きます。
3. 「仮想サーバー」ページに移動します。
4. CGI ディレクトリを指定する仮想サーバーの名前をクリックします。
5. 「CGI」タブをクリックします。

6. 「CGI ディレクトリ」オプションをクリックします。
7. このディレクトリに対して使用する URL プレフィックスを「URL プレフィックス」フィールドに入力します。つまり、ここに入力したテキストは、CGI プログラム用のディレクトリとして URL に表示されます。

   たとえば、URL プレフィックスとして「cgi-bin」と入力すると、これらの CGI プログラムを指す URL はすべて次のような形式になります。

   ```
   http://yourserver.domain.com/cgi-bin/program-name
   ```

注 ここで指定する URL プレフィックスは、前の手順で指定した実際の CGI 用ディレクトリとは同じでなくてもかまいません。

8. ディレクトリの場所を示す絶対パスを「CGI ディレクトリ」テキストフィールドに入力します。
9. 「了解」ボタンをクリックします。
10. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

サーバーは、これらのディレクトリにあるすべてのファイルを CGI プログラムとして扱います。

既存の CGI 用ディレクトリを削除するには、「CGI ディレクトリ」ページでそのディレクトリの「削除」ボタンをクリックします。既存のディレクトリの URL プレフィックスや CGI 用ディレクトリを変更するには、そのディレクトリの「編集」ボタンをクリックします。

指定したディレクトリに CGI プログラムをコピーします。CGI 用ディレクトリにあるファイルはすべて CGI プログラムとして処理されるため、CGI 用ディレクトリに HTML ファイルを置かないでください。

obj.conf ファイルには、各 CGI 用ディレクトリに対応する NameTrans 指令があります。この指令は、そのディレクトリ内のリソースに対する各要求に cgi という名前を関連付けます。これらの指令は、管理インタフェースで CGI 用ディレクトリを指定すると自動的に obj.conf に追加されます。必要に応じて、指令を obj.conf に手動で追加することもできます。

たとえば、次の命令は、http://*server-name*/cgi-local にあるリソースに対する要求はすべて、C:/SunServer/docs/mycgi ディレクトリにある CGI プログラムを呼び出す要求として解釈されます。

```
NameTrans fn="pfx2dir" from="/cgi-local"
dir="C:/SunServer/docs/mycgi" name="cgi"
```

obj.conf ファイルには、次の名前付きのオブジェクトが必要です。

```
<Object name="cgi">
ObjectType fn="force-type" type="magnus-internal/cgi"
Service fn="send-cgi"
</Object>
```

このオブジェクトを obj.conf から削除しないでください。このオブジェクトを削除すると、管理インタフェースで CGI 用ディレクトリを指定したり、NameTrans 指令を obj.conf に手動で追加したりしても、サーバーは CGI 用ディレクトリを認識しなくなります。

CGI ファイル拡張子を指定する

ファイルがどのディレクトリに置かれていても、特定の拡張子を持つすべてのファイルをサーバーが CGI プログラムとして扱うようにするには、次の手順を実行します。

1. 管理インタフェースで、サーバーインスタンスの下の HTTP Server コンポーネントを開きます。
2. 「Virtual Servers」ページに移動します。
3. CGI ファイルタイプを指定する仮想サーバーの名前をクリックします。
4. 「CGI」タブをクリックします。
5. 「CGI File Type」オプションをクリックします。
6. この変更を適用するリソースを「Editing picker」から選択します。
7. 「ファイルの種類として CGI を有効」の下の「はい」ラジオボタンをクリックします。
8. 「了解」ボタンをクリックします。
9. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

デフォルトの CGI 拡張子は、.cgi、.bat、および .exe です。

CGI プログラムを表す拡張子を変更するには、mime.types 内の次の行を編集して、目的の拡張子を指定します。mime.types を編集した後は、必ずサーバーを再起動してください。

```
type=magnus-internal/cgi exts=cgi,exe,bat
```

適切な拡張子を持ったファイルをすべて CGI プログラムとして扱うようにサーバーに指示した場合、obj.conf ファイルには次の Service 指令が追加されます。

```
Service fn="send-cgi" type="magnus-internal/cgi"
```

CGI プログラムのカスタム実行環境の作成 (UNIX のみ)

カスタム実行環境を作成するには、ルートユーザーとして suid Cgistub をインストールして実行する必要があります。

1. スーパーユーザーとしてログインします。

   ```
   su
   ```

2. Cgistub 用に private ディレクトリを作成します。

   ```
   cd instance_dir
   mkdir private
   ```

3. Cgistub を private ディレクトリにコピーします。

   ```
   cd private
   cp install_dir/lib/Cgistub
   ```

4. サーバーユーザーを private の所有者に設定します。

   ```
   chown username
   ```

5. private にアクセス権を設定します。

   ```
   chmod 500 .
   ```

6. ルートを Cgistub の所有者に設定します。

   ```
   chown root Cgistub
   ```

7. Cgistub にアクセス権を設定します。

   ```
   chmod 4711 Cgistub
   ```

8. obj.conf 内の send-cgi 関数 への各参照に user パラメータを指定します。次に例を示します。

   ```
   Service fn="send-cgi" user="username"
   ```

   変数置換も使用できます。たとえば、server.xml で、virtual-server 要素に次の property サブ要素を指定します。

   ```
   <property name="user" value="username"/>
   ```

   これにより、obj.conf の send-cgi 関数行を次のように記述できます。

   ```
   Service fn="send-cgi" user="$user"
   ```

   send-cgi と obj.conf の詳細については、『Sun ONE Application Server Developer's Guide to NSAPI』を参照してください。server.xml の詳細については、『Sun ONE Application Server Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

9. これらの変更を有効にするためにサーバーを再起動します。

注	Cgistub は *instance_dir*/private ディレクトリにインストールすることをお勧めします。ほかの場所にインストールする場合は、Cgistub へのパスを init.conf 内の init-cgi 関数に指定する必要があります。詳細については、『Sun ONE Application Server Developer's Guide to NSAPI』を参照してください。

注	NFS マウントには suid Cgistub プログラムをインストールできないことがあります。suid Cgistub を使用するには、サーバーインスタンスをローカルファイルシステムにインストールする必要があります。

Cgistub は、次のセキュリティ制限を適用します。

- CGI プログラムを実行するユーザーの uid は 100 以上であること。この制限は、第三者が Cgistub を使って root アクセス権を取得することを防止する
- CGI プログラムを実行するユーザーはその所有者であること。また、所有者以外が CGI プログラムを書き換えることはできない。この制限は、第三者が密かにプログラムを挿入し、リモートで実行することを困難にする
- Cgistub は、UNIX 待機ソケットを作成してアクセス権 0700 を設定する

注	SunOS/Solaris の現在のバージョンも含む UNIX 系環境の多くで、ソケットのアクセス権は考慮されません。悪意のあるユーザーが Cgistub を利用できないようにするには、init.conf TempDir ディレクティブを使って、サーバーの一時ディレクトリをサーバーユーザーだけがアクセスできるディレクトリに変更します。詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

Cgistub のインストールを完了したら、次の方法でカスタム実行環境を作成できます。

- 仮想サーバーに一意の CGI 用ディレクトリと UNIX ユーザーおよびグループを指定する
- 仮想サーバーディレクトリを chroot で指定する

仮想サーバーに一意の CGI 用ディレクトリと UNIX ユーザーおよびグループを指定する

仮想サーバーの CGI プログラムがほかのユーザーと干渉しないようにするには、プログラムを一意のディレクトリに保存し、一意の UNIX ユーザーおよびグループの権限を使って実行されるようにする必要があります。

最初に、UNIX ユーザーおよびグループを作成します。ユーザーおよびグループの詳細な作成手順は、オペレーティングシステムによって異なります。詳細については、オペレーティングシステムのマニュアルを参照してください。

次に、仮想サーバーの `cgi-bin` ディレクトリを次の手順で作成します。

1. スーパーユーザーとしてログインします。

   ```
   su
   ```

2. 仮想サーバーのディレクトリに移動します。

   ```
   cd vs_dir
   ```

3. `cgi-bin` ディレクトリを作成します。

   ```
   mkdir cgi-bin
   chown user:group cgi-bin
   chmod 755 cgi-bin
   ```

仮想サーバーの CGI 用ディレクトリ、ユーザーおよびグループを次のいずれかの方法で設定します。

- `obj.conf` ファイル内の `send-cgi` 関数のパラメータ `dir`、`user`、および `group` を使用する。詳細については、『Sun ONE Application Server Developer's Guide to NSAPI』を参照
- 管理インタフェースを使って、次の手順でこの情報を入力する
  a. サーバーインスタンスの下の HTTP サーバーコンポーネントを開きます。
  b. 「仮想サーバー」ページに移動します。
  c. CGI ディレクトリを指定する仮想サーバーの名前をクリックします。
  d. 「一般」タブをクリックします。
  e. 「ディレクトリ」、「ユーザー」、および「グループ」の各フィールドに値を入力します。
  f. 「保存」ボタンをクリックします。
  g. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

詳細については、『Sun ONE Application Server 管理者ガイド』を参照してください。

仮想サーバーディレクトリを chroot で指定する

セキュリティ向上のために、仮想サーバーディレクトリの上位および外部にあるデータに CGI スクリプトがアクセスできないようにする必要があります。

最初に、chroot 環境を設定します。chroot 環境の詳細な設定手順は、オペレーティングシステムによって異なります。詳細については、オペレーティングシステムのマニュアルを参照してください。最初に、ftpd と chroot のマニュアルページを参照することをお勧めします。

Solaris バージョン 2.6 から 8 での手順は次のとおりです。

1. スーパーユーザーとしてログインします。

   ```
   su
   ```

2. chroot ディレクトリに移動します。通常、このディレクトリは前節で説明した *vs_dir* ディレクトリです。

   ```
   cd chroot
   ```

3. chroot ディレクトリに tmp を作成します。

   ```
   mkdir tmp
   chmod 1777 tmp
   ```

4. chroot ディレクトリに dev を作成します。

   ```
   mkdir dev
   chmod 755 dev
   ```

5. /dev/tcp の内容を一覧表示し、出力結果のメジャー番号とマイナー番号を書き留めます。この例では、メジャー番号は 11、マイナー番号は 42 です。

   ```
   ls -lL /dev/tcp
   crw-rw-rw-   1 root     sys       11, 42 Apr  9  1998 /dev/tcp
   ```

6. メジャー番号とマイナー番号を使って tcp デバイスを作成します。

   ```
   mknod dev/tcp c 11 42
   chmod 666 dev/tcp
   ```

7. 次の各デバイスについて手順 5 と 6 を繰り返します (デバイスごとにメジャー番号とマイナー番号は異なります)。

```
/dev/udp
/dev/ip
/dev/kmem
/dev/kstat
/dev/ksyms
/dev/mem
/dev/null
/dev/stderr
/dev/stdin
/dev/stdout
/dev/ticotsord
/dev/zero
```

8. chroot ディレクトリの dev にあるデバイスにアクセス権を設定します。

```
chmod 666 dev/*
```

9. chroot ディレクトリに lib と usr/lib を作成し、内容を設定します。

```
mkdir usr

mkdir usr/lib

ln -s /usr/lib

ln /usr/lib/* usr/lib
```

このコマンドで生成されるメッセージは無視してかまいません。

/usr/lib ディレクトリが別のファイルシステムにある場合は、最後のコマンドを次のように置き換えます。

```
cp -rf /usr/lib/* usr/lib
```

10. chroot ディレクトリに bin と usr/bin を作成し、内容を設定します。

```
mkdir usr/bin

ln -s /usr/bin

ln /usr/bin/* usr/bin
```

このコマンドで生成されるメッセージは無視してかまいません。

/usr/bin ディレクトリが別のファイルシステムにある場合は、最後のコマンドを次のように置き換えます。

```
cp -rf /usr/bin/* usr/bin
```

11. chroot ディレクトリに etc 作成し、内容を設定します。

```
mkdir etc

ln /etc/passwd /etc/group /etc/netconfig etc
```

12. chroot 環境をテストします。

    ```
    chroot chroot bin/ls -l
    ```

    出力結果の例を次に示します。

    ```
    total 14
    lrwxrwxrwx    1 root     other      8 Jan 13 03:32 bin -> /usr/bin
    drwxr-xr-x    2 user     group    512 Jan 13 03:42 cgi-bin
    drwxr-xr-x    2 root     other    512 Jan 13 03:28 dev
    drwxr-xr-x    2 user     group    512 Jan 13 03:26 docs
    drwxr-xr-x    2 root     other    512 Jan 13 03:33 etc
    lrwxrwxrwx    1 root     other      8 Jan 13 03:30 lib -> /usr/lib
    drwxr-xr-x    4 root     other    512 Jan 13 03:32 usr
    ```

仮想サーバーの chroot ディレクトリを次のいずれかの方法で設定します。

- `obj.conf` ファイル内の `send-cgi` 関数の `chroot` パラメータを使用する。詳細については、『Sun ONE Application Server Developer's Guide to NSAPI』を参照
- 管理インタフェースを使って、次の手順でこの情報を入力する
  a. サーバーインスタンスの下の HTTP サーバーコンポーネントを開きます。
  b. 「仮想サーバー」ページに移動します。
  c. CGI ディレクトリを指定する仮想サーバーの名前をクリックします。
  d. 「一般」タブをクリックします。
  e. 「ディレクトリ変更」フィールドに値を入力します。
  f. 「保存」ボタンをクリックします。
  g. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

  詳細については、『Sun ONE Application Server 管理者ガイド』を参照してください。

CGI プログラムをサーバーに追加

CGI プログラムを Sun ONE Application Server に追加するには、次のいずれかの手順を実行します。

- CGI 用ディレクトリを作成した場合は、その中にプログラムファイルをドラッグ＆ドロップします。
- CGI プログラムをファイルタイプで認識するように設定した場合は、サーバーが CGI プログラムとして認識できるようなファイル名で、ドキュメントルートまたはその下のディレクトリに格納します。

UNIX の場合は、このプログラムファイルのアクセス権に実行権が必要です。

CGI プログラムの優先度の設定

CGI プログラムの優先度を設定するには、次の手順を実行します。

1. 管理インタフェースで、サーバーインスタンスの下の HTTP サーバーコンポーネントを開きます。
2. 「仮想サーバー」ページに移動します。
3. CGI ディレクトリを指定する仮想サーバーの名前をクリックします。
4. 「一般」タブをクリックします。
5. 「優先順位」フィールドに値を入力します。この増分値によって、サーバーに対する CGI プログラムの優先度が決まります。通常、サーバーは nice 値 `0` で動作しており、CGI の nice 増分値は `0` から `19` の範囲で指定します。0 を指定すると CGI プログラムはサーバーと同じ優先度で動作し、19 を指定すると CGI プログラムはサーバーよりもかなり低い優先度で動作します。nice 増分値として `-1` を指定することで CGI プログラムの優先度をサーバーより高くすることも可能ですが、お勧めできません。
6. 「保存」ボタンをクリックします。
7. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

詳細については、『Sun ONE Application Server 管理者ガイド』を参照してください。

Windows CGI プログラム

この節では、Windows CGI プログラムのインストール方法について説明します。この節では次のトピックについて説明します。

- Windows CGI プログラムの概要
- Windows CGI ディレクトリの指定
- Windows CGI をファイルタイプで指定

Windows CGI プログラムの概要

Windows CGI プログラムはその他の CGI プログラムと同様に処理されます。Windows CGI プログラムだけが格納されているディレクトリを指定して、すべての Windows CGI プログラムに同じファイル拡張子を指定するか、またはそのいずれかの処理を行います。

Windows CGI プログラムは通常の CGI プログラムと同様に動作しますが、サーバーによる実際の処理方法は若干異なります。このため、Windows CGI プログラムには異なったディレクトリを指定する必要があります。Windows CGI ファイルタイプを有効にした場合、ファイル拡張子 `.wcg` が使用されます。

Sun ONE Application Server は、非公式の Windows CGI 1.3a 仕様をサポートします。この仕様には次の特徴があります。

- セキュリティメソッドをサポートするため、`[CGI]` セクションに次のキーワードが追加される
    - HTTPS: トランザクションが SSL によって処理されるかどうかによって on または off の値をとる
    - HTTPS キーサイズ : HTTPS が有効になっている場合、この値によって暗号化に使用されるセッションキー内のビット数が報告される
    - HTTPS 秘密キーサイズ : HTTPS が有効になっている場合、この値によってサーバーの秘密鍵の生成に使用されるビット数が報告される
- `[CGI]` セクション内のキーワード `Document Root` は、サーバーが単一のドキュメントルートを持たない場合、ドキュメントルートを参照しない可能性があります。この変数には、Windows CGI プログラムのルートディレクトリが返されます。
- `[CGI]` セクション内のキーワード `Server Admin` はサポートされていません。
- `[CGI]` セクション内のキーワード `Authentication Realm` はサポートされません。
- マルチパートフォームデータエンコーディングでのフォーム送信はサポートされません。

Windows CGI ディレクトリの指定

WinCGI プログラムのみを格納するディレクトリを指定するには、次の手順を実行します。

1. 使用中のコンピュータ上に Windows CGI ディレクトリを作成します。このディレクトリは、ドキュメントルートディレクトリのサブディレクトリ以外でもかまいません。手順 7 で URL プレフィックスを指定するのはこのためです。
2. 管理インタフェースで、サーバーインスタンスの下の HTTP サーバーコンポーネントを開きます。
3. 「仮想サーバー」ページに移動します。
4. Windows CGI ディレクトリを指定する仮想サーバーの名前をクリックします。
5. 「CGI」タブをクリックします。
6. 「WINCGI ディレクトリ」オプションをクリックします。
7. このディレクトリに対して使用する URL プレフィックスを「URL プレフィックス」フィールドに入力します。つまり、ここに入力したテキストは、CGI プログラム用のディレクトリとして URL に表示されます。

   たとえば、URL プレフィックスとして「`cgi-bin`」と入力すると、これらの CGI プログラムを指す URL はすべて次のような形式になります。

   ```
   http://yourserver.domain.com/cgi-bin/program-name
   ```

注 ここで指定する URL プレフィックスは、前の手順で指定した実際の CGI 用ディレクトリとは同じでなくてもかまいません。

8. ディレクトリの場所を示す絶対パスを「WINCGI ディレクトリ」テキストフィールドに入力します。
9. スクリプトの追跡を有効にするには、「はい」ラジオボタンを選択します。
10. 「了解」ボタンをクリックします。
11. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

既存の Windows CGI 用ディレクトリを削除するには、「WINCGI ディレクトリ」ページでそのディレクトリの「削除」ボタンをクリックします。既存のディレクトリの URL プレフィックスや Windows CGI 用ディレクトリを変更するには、そのディレクトリの「編集」ボタンをクリックします。

指定したディレクトリに Windows CGI プログラムをコピーします。このディレクトリ内のすべてのファイルは Windows CGI ファイルとして処理される点に注意してください。

Windows CGI をファイルタイプで指定

Windows CGI ファイルのファイル拡張子を指定するには、次の手順を実行します。

1. 管理インタフェースで、サーバーインスタンスの下の HTTP サーバーコンポーネントを開きます。
2. 「仮想サーバー」ページに移動します。
3. Windows CGI ファイルタイプを指定する仮想サーバーの名前をクリックします。
4. 仮想サーバーの MIME タイプファイルの名前を書き留めます。
5. 「MIME タイプファイル」ページに移動します。
6. 手順 4 で書き留めた名前に一致する名前をクリックします。
7. 「MIME タイプ ...」ボタンをクリックします。
8. 次の設定の新しい MIME タイプを追加します。
   - Category: `type`
   - Content-type: `magnus-internal/wincgi`
   - File Suffix: サーバーを Windows CGI に関連付けるファイルサフィックスを入力します。CGI、WinCGI、シェル CGI の各ファイルタイプを有効にした場合、それぞれの CGI に異なったサフィックスを指定する必要があります。たとえば、CGI プログラムとシェル CGI プログラムの両方に `.exe` というサフィックスを指定することはできません。一意のサフィックスになるように、必要に応じてページ上のその他の MIME タイプフィールドを編集することもできます。
9. 「新規タイプ」ボタンをクリックします。
10. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

Windows のシェル CGI プログラム

この節では、Windows のシェル CGI プログラムのインストール方法について説明します。この節では次のトピックについて説明します。

- Windows のシェル CGI プログラムの概要
- シェル CGI ディレクトリの指定 (Windows の場合)
- シェル CGI をファイルタイプで指定 (Windows の場合)

Windows のシェル CGI プログラムの概要

シェル CGI は、Windows 内にファイルの関連付けを設定することによって CGI アプリケーションを実行するサーバー設定です。

たとえば、`hello.pl` というシェル CGI ファイルの要求を取得した場合、サーバーは Windows のファイルの関連付けにより、`.pl` 拡張子に関連付けられたプログラムを使ってファイルを実行します。`.pl` 拡張子が `C:¥bin¥perl.exe` というプログラムに関連付けられている場合、サーバーは次の手順で `hello.pl` ファイルの実行を試みます。

```
c:¥bin¥perl.exe hello.pl
```

シェル CGI の設定方法としては、サーバーのドキュメントルートに、シェル CGI ファイルだけが格納されたディレクトリを作成する方法が最も簡単です。ただし、Sun ONE Application Server の MIME タイプを編集して、シェル CGI と特定のファイル拡張子を関連付けるようにサーバーを設定することもできます。

注 Windows ファイル拡張子の設定方法については、Windows のマニュアルを参照してください。

シェル CGI ディレクトリの指定 (Windows の場合)

シェル CGI プログラムだけを格納する専用ディレクトリを指定するには、次の手順を実行します。

1. 使用中のコンピュータ上にシェル CGI ディレクトリを作成します。このディレクトリは、ドキュメントルートディレクトリのサブディレクトリ以外でもかまいません。手順 7 で URL プレフィックスを指定するのはこのためです。
2. 管理インタフェースで、サーバーインスタンスの下の HTTP サーバーコンポーネントを開きます。
3. 「仮想サーバー」ページに移動します。
4. シェル CGI ディレクトリを指定する仮想サーバーの名前をクリックします。
5. 「CGI」タブをクリックします。
6. 「シェル CGI ディレクトリ」オプションをクリックします。
7. このディレクトリに対して使用する URL プレフィックスを「URL プレフィックス」フィールドに入力します。つまり、ここに入力したテキストは、CGI プログラム用のディレクトリとして URL に表示されます。

   たとえば、URL プレフィックスとして「`cgi-bin`」と入力すると、これらの CGI プログラムを指す URL はすべて次のような形式になります。

   ```
   http://yourserver.domain.com/cgi-bin/program-name
   ```

注 ここで指定する URL プレフィックスは、前の手順で指定した実際の CGI 用ディレクトリとは同じでなくてもかまいません。

8. ディレクトリの場所を示す絶対パスを「シェル CGI ディレクトリ」テキストフィールドに入力します。
9. 「了解」ボタンをクリックします。
10. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。
11. シェル CGI ディレクトリ内のすべてのファイルが Windows に設定されているファイルの関連付けを持っていることを確認します。関連付けのないファイル拡張子を持ったファイルを実行しようとすると、サーバーはエラーを返します。

警告 サーバーには、シェル CGI ディレクトリに対する読み込み権限と実行権限が必要です。Windows の場合、サーバーを実行するユーザーアカウント (`LocalSystem` など) に、シェル CGI ディレクトリ内のプログラムを読み込み、実行する権限を付与する必要があります。

既存のシェル CGI ディレクトリを削除するには、「シェル CGI ディレクトリ」ページでそのディレクトリの「削除」ボタンをクリックします。既存のディレクトリの URL プレフィックスやシェル CGI ディレクトリを変更するには、そのディレクトリの「編集」ボタンをクリックします。

指定したディレクトリにシェル CGI プログラムをコピーします。このディレクトリ内のすべてのファイルはシェル CGI ファイルとして処理される点に注意してください。

シェル CGI をファイルタイプで指定 (Windows の場合)

Sun ONE Application Server の `mime.types` ファイルを使って、ファイル拡張子とシェル CGI 機能を関連付けることができます。この関連付けは、Windows で関連付けを作成する方法とは別の方法で行います。

ファイル拡張子とサーバーのシェル CGI 機能を関連付けるには、たとえば、`.pl` 拡張子を持つファイルの関連付けを作成します。サーバーは、この拡張子を持つファイルの要求を受け取ると、Windows 内のこのファイル拡張子に関連付けられた実行可能ファイルを呼び出して、要求されたファイルをシェル CGI ファイルとして処理します。

シェル CGI ファイルとしてファイル拡張子を関連付けるには、次の手順を実行します。

1. 管理インタフェースで、サーバーインスタンスの下の HTTP サーバーコンポーネントを開きます。
2. 「仮想サーバー」ページに移動します。
3. Windows CGI ファイルタイプを指定する仮想サーバーの名前をクリックします。
4. 仮想サーバーの MIME タイプファイルの名前を書き留めます。
5. 「MIME タイプファイル」ページに移動します。
6. 手順 4 で書き留めた名前に一致する名前をクリックします。
7. 「MIME タイプ ...」ボタンをクリックします。
8. 次の設定の新しい MIME タイプを追加します。
   - Category: `type`
   - Content-type: `magnus-internal/shellcgi`

- File Suffix: サーバーを Windows CGI に関連付けるファイルサフィックスを入力します。CGI、WinCGI、シェル CGI の各ファイルタイプを有効にした場合、それぞれの CGI に異なったサフィックスを指定する必要があります。たとえば、CGI プログラムとシェル CGI プログラムの両方に `.exe` というサフィックスを指定することはできません。一意のサフィックスになるように、必要に応じてページ上のその他の MIME タイプフィールドを編集することもできます。

9. 「新規タイプ」ボタンをクリックします。
10. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

クエリハンドラ

注 クエリハンドラを使用する方法は、現在ではあまり一般的ではありません。Sun ONE Application Server や Netscape Navigator のクライアントはクエリハンドラをサポートしていますが、実際に使用することはほとんどありません。現在では、多くのユーザーが、HTML ページ内でフォームを使ってクエリを送信しています。

デフォルトのクエリハンドラ CGI プログラムを指定できます。クエリハンドラは、HTML ファイル内の ISINDEX タグを使って、送信したテキストを処理します。

ISINDEX は、HTML ページ内に型付きの入力を受け付けるテキストフィールドを作成するという点で、フォームテキストフィールドとよく似ています。ただし、ISINDEX ボックス内の情報は、ユーザーが Return キーを押すとすぐに送信されるという点でフォームテキストフィールドとは異なります。デフォルトのクエリハンドラを指定する際、サーバーが入力の送信に使用するプログラムを指定します。ISINDEX タグの詳細については、HTML リファレンスマニュアルを参照してください。

クエリハンドラを設定するには、次の手順を実行します。

1. 管理インタフェースで、サーバーインスタンスの下の HTTP サーバーコンポーネントを開きます。
2. 「仮想サーバー」ページに移動します。
3. クエリハンドラを指定する仮想サーバーの名前をクリックします。
4. 「CGI」タブをクリックします。
5. 「クエリハンドラ」オプションをクリックします。
6. 「編集」を使って、デフォルトのクエリハンドラで設定するリソースを選択します。

ディレクトリを選択した場合、サーバーがそのディレクトリの URL を受け取ったとき、またはそのディレクトリ内のファイルを受け取ったときにだけ指定したクエリハンドラが実行されます。

7. 「デフォルトクエリハンドラ」フィールドに、選択したリソースのデフォルト CGI プログラムのフルパスを入力します。
8. 「了解」ボタンをクリックします。
9. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

Perl CGI プログラム

Perl 5.6.*x* では、`-w` フラグを指定して CGI を実行することはできません。代わりに、次のコードをファイルに記述してください。

```
use warnings;
```

CGI のグローバル設定

CGI のグローバル設定を変更するには、次の手順を実行します。

1. 管理インタフェースで、サーバーインスタンスの下の HTTP サーバーコンポーネントを開きます。
2. 「HTTP サーバー」ページに移動します。
3. 「詳細」タブをクリックします。
4. 「CGI」オプションをクリックします。
5. 次の設定を変更できます。
   - `MinCGIStubs` - デフォルトで起動する CGIStub プロセス数を制御します。`MaxCGIStubs` より小さい値を指定します。デフォルト値は `2` です。
   - `CGIExpirationTimeout` - CGI プロセスが終了前に実行できる最大秒数を指定します。デフォルト値は `0` です。この場合、プロセスの実行時間に制限はありません。
   - `CGIStubIdleTimeout` - アイドル時間がこの秒数に達した場合、その CGIStub プロセスを強制終了します。デフォルト値は `30` です。
   - `MaxCGIStubs` - サーバーが同時に実行できる最大 CGIStub プロセス数を設定します。デフォルト値は `10` です。
6. 「了解」ボタンをクリックします。

7. サーバーインスタンスページに移動し、「変更の適用」ボタンを選択します。

CGI のグローバル設定の詳細については、『Sun ONE Application Server 管理者用設定ファイルリファレンス』の `init.conf` ファイルの説明を参照してください。

CGI 変数

CGI プログラムでは、標準の CGI 変数のほかに、Sun ONE Application Server の CGI 変数も使用できます。この変数を使用すると、サーバーが保護モードで動作している場合、クライアント証明書情報にアクセスできます。CLIENT_CERT 変数と REVOCATION 変数は、クライアント証明書ベースの認証が有効になっている場合のみ使用できます。

次の表は、Sun ONE Application Server の CGI 変数を示しています。左の列は変数、右の列は各変数の説明です。

CGI 変数

変数	説明
SERVER_URL	クライアントが要求したサーバーの URL
HTTP_*xxx*	受信した HTTP 要求のヘッダー。*xxx* はヘッダー名
HTTPS	サーバーが保護モードの場合は ON、それ以外の場合は OFF
HTTPS_KEYSIZE	SSL ハンドシェークのキーサイズ (サーバーが保護モードの場合に使用可能)
HTTPS_SECRETKEYSIZE	SSL ハンドシェークの秘密部分のキーサイズ (サーバーが保護モードの場合に使用可能)
HTTPS_SESSIONID	接続のセッション ID (サーバーが保護モードの場合に使用可能)
CLIENT_CERT	クライアントが用意した証明書 (バイナリ DER 形式)
CLIENT_CERT_SUBJECT_DN	クライアント証明書対象者の識別名
CLIENT_CERT_SUBJECT_OU	クライアント証明書対象者の組織単位
CLIENT_CERT_SUBJECT_O	クライアント証明書対象者の組織
CLIENT_CERT_SUBJECT_C	クライアント証明書対象者の国
CLIENT_CERT_SUBJECT_L	クライアント証明書対象者の場所
CLIENT_CERT_SUBJECT_ST	クライアント証明書対象者の州

CGI 変数 (続き)

変数	説明
CLIENT_CERT_SUBJECT_E	クライアント証明書対象者の電子メール
CLIENT_CERT_SUBJECT_UID	クライアント証明書対象者の CN の UID 部分
CLIENT_CERT_ISSUER_DN	クライアント証明書発行者の識別名
CLIENT_CERT_ISSUER_OU	クライアント証明書発行者の組織単位
CLIENT_CERT_ISSUER_O	クライアント証明書発行者の組織
CLIENT_CERT_ISSUER_C	クライアント証明書発行者の国
CLIENT_CERT_ISSUER_L	クライアント証明書発行者の場所
CLIENT_CERT_ISSUER_ST	クライアント証明書発行者の州
CLIENT_CERT_ISSUER_E	クライアント証明書発行者の電子メール
CLIENT_CERT_ISSUER_UID	クライアント証明書発行者の CN の UID 部分
CLIENT_CERT_VALIDITY_START	証明書の開始日
CLIENT_CERT_VALIDITY_EXIRES	証明書の有効期限
CLIENT_CERT_EXTENSION_*xxx*	証明書の拡張子。*xxx* は拡張子名
REVOCATION_METHOD	証明書の取り消し方法がある場合は、その方法の名前
REVOCATION_STATUS	証明書の取り消しがある場合は、取り消し状況

索引

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

て

と