サーブレットに関するプログラマーズガイド

iPlanet Web Server, Enterprise Edition

Version 6.0

816-2142-01
2001 年 5 月

原典 : iPlanet Web Server, Enterprise Edition Programmer's Guide to Servlets

Part No: 816-1381-01

目 次

このマニュアルについて

このマニュアルでは、iPlanet™ Web Server, Enterprise Edition 6.0 での Java サーブレットおよび JSP (JavaServer Pages) の有効化とインストールの方法について説明します。

このマニュアルには、以下の章および付録があります。

- 第 1 章「サーブレットおよび JSP (JavaServer Pages)」

  Web アプリケーション、サーブレット、および JSP (JavaServer Pages) を紹介します。

- 第 2 章「Web アプリケーション」

  Web アプリケーションの構造とその iPlanet Web Server 6.0 への導入方法について説明します。また、Web アプリケーションの例の一覧を示し、web-apps.xml ファイルについて説明します。このファイルは、iPlanet Web Server 6.0 の仮想サーバ用の Web アプリケーションを構成するためのものです。

- 第 3 章「サーブレットの使用方法」

  iPlanet Web Server 6.0 でのサーブレットの有効化とインストールの方法について説明します。

- 第 4 章「JSP (JavaServer Pages) の使用方法」

  JDK のインストール方法、および iPlanet Web Server 6.0 での JSP (JavaServer Pages) の有効化とインストールの方法について説明します。

- 第 5 章「サーブレットおよび JSP のデバッグ」

  Forte for Java を使ったデバッグについて、およびサーブレットと JSP のデバッグの方法について説明します。

- 第 6 章「セッションマネージャ」

  iPlanet Web Server に付属のセッションマネージャについて説明し、さらに必要に応じてセッションの動作をカスタマイズする方法について説明します。

- 第 7 章「API に関するヒント」

  iPlanet Web Server で Sun Microsystems の Servlet API に関するマニュアルの記載とわずかに動作が異なる Servlet API のメソッド、または Sun Microsystems のマニュアルに記載の動作があいまいな Servlet API のメソッドについて説明します。

- 第 8 章「古いバージョンのサーブレットおよび JSP の構成」

  iPlanet Web Server 4.x の場合と同様のデフォルトの仮想サーバの構成方法と、`servlets.properties`、`rules.properties`、および `contexts.properties` ファイルの構成方法について説明します。

- 付録 A「magnus.conf および obj.conf のサーブレット設定」

  サーブレットおよび JSP の設定により変更される構成ファイル `obj.conf` について説明します。

- 付録 B「SSJS アプリケーションの変換」

  LiveWire アプリケーションの JSP への変換方法について説明します。

- 付録 C「JVM の構成」

  JVM 構成情報を手動で指定する方法について説明します。

- 付録 D「サーブレットのリモートプロファイリング」

  サーブレットのリモートプロファイリングを有効にする方法について説明します。

注

- このマニュアル全体を通して、特に Linux に言及している箇所を除いて、UNIX 固有の説明は Linux のオペレーティングシステムにも適用されます。
- このマニュアルでは、パスの区切り記号に「/」を使用していますが、Microsoft Windows 環境では、画面上に円記号 (¥) が表示されることがあります。また、パスの先頭にドライブ名の指定が必要な場合もあります。詳細は、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。
- このマニュアルには、「バックスラッシュ (\)」という表記が含まれていますが、これは円記号 (¥) に読み換えてください。

第 1 章

サーブレットおよび JSP (JavaServer Pages)

iPlanet Web Server 6.0 は、サーブレットおよび JSP (JavaServer Pages) をサポートしています。この章では、iPlanet Web Server 6.0 のサーブレットおよび JSP の概要を示します。

この章は、以下の節から構成されています。

- Web アプリケーション
- サーブレット
- JSP (JavaServer Pages)

iPlanet Web Server 6.0 は、オペレーティングシステムに依存する JRE または JDK の特定のバージョンとともに実行されます。詳しくは、38 ページの「JSP を実行するためにサーバで必要なもの」を参照してください。

Web アプリケーション

iPlanet Web Server 6.0 は、サーブレットおよび JSP を Web アプリケーションに組み込むことができる Servlet 2.2 API 仕様をサポートしています。

Web アプリケーションは、サーブレット、JSP (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その他のデータをインクルードする Web リソースのコレクションです。Web アプリケーションは、アーカイブ（WAR ファイル）にパッケージ化されているか、オープンディレクトリ構造になっています。

iPlanet Web Server の Web アプリケーションサポートの詳細は、第 2 章「Web アプリケーション」を参照してください。

Web アプリケーションの使用をお勧めしますが、サーブレットおよび JSP を iPlanet Web Server 4.*x* の場合と同様に構成することもできます。詳しくは、第 8 章「古いバージョンのサーブレットおよび JSP の構成」を参照してください。

サーブレット

Java サーブレットは、CGI プログラムと同様、Web サーバで実行してクライアントの要求に応答してコンテンツを生成することができる、サーバサイドの Java プログラムです。サーブレットは、サーバサイドで動作する、ユーザインタフェースのないアプレットと考えることができます。サーブレットは、URL で呼び出して起動します。

iPlanet Web Server 6.0 は、Java Servlet Specification バージョン 2.2 をサポートしています (Web Application および WAR ファイルもサポート)。

注	Servlet API バージョン 2.2 にはバージョン 2.1 との完全な下位互換性があるので、既存のサーブレットは変更またはコンパイルし直さなくても引き続き機能します。

サーブレットの開発には、Sun Microsystems の Java Servlet API を使用します。Java Servlet API の使用の詳細は、以下の Web サイトにある Sun Microsystems のドキュメントを参照してください。

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

JSP (JavaServer Pages)

iPlanet Web Server 6.0 は、JSP (JavaServer Pages) Specification バージョン 1.1 をサポートしています。

JSP は、HTML ページのように、Web ブラウザで表示できるページです。ただし、HTML タグのほかに、JSP タグや、Java コードと混合した指令を組み込むことができるので、Web ページの設計者はより多くの動的なコンテンツをページに取り込むことができます。これらの追加機能によって、属性値の表示や単純な条件の使用などが可能になります。

JSP の主な利点のひとつは、HTML ページと同様、コンパイルする必要がないことです。Web ページの設計者は、HTML と JSP のタグを使用したページを作成して Web サーバに置くだけです。Web ページの設計者は、Java クラスを定義する方法や Java コンパイラを使う方法を習得する必要はありません。

ただし、iPlanet Web Server は JSP の事前コンパイルをサポートしているため、本稼動用のサーバには JSP の事前コンパイルをお勧めします。

JSP ページは、以下の方法で Java の機能に完全にアクセスできます。

- Java コードをページのスクリプトレットに直接組み込む
- JavaBeans にアクセスする
- Java サーブレットがインクルードされるサーバサイドのタグを使用する

Bean もサーブレットもコンパイルの必要がある Java クラスですが、Java のプログラマが定義およびコンパイルを行い、その Bean またはサーブレットへのインタフェースを公開することができます。Web ページの設計者は、事前コンパイル済み Bean またはサーブレットに JSP ページからアクセスできます。

iPlanet Web Server 6.0 は、JSP タグのライブラリおよび標準のポータブルタグをサポートしています。

JSP の作成の詳細は、以下の Sun Microsystems の JSP に関する Web サイトを参照してください。

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

JavaBeans の詳細は、以下の Sun Microsystems の JavaBeans に関する Web サイトを参照してください。

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

第 2 章

Web アプリケーション

iPlanet Web Server 6.0 は、サーブレットおよび JSP の Web アプリケーションへの組み込みを可能にする Servlet 2.2 API 仕様をサポートしています。

Web アプリケーションは、サーブレット、JSP (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その他のデータをインクルードする Web リソースのコレクションです。Web アプリケーションは、アーカイブ（WAR ファイル）にパッケージ化されているか、オープンディレクトリ構成になっています。Web アプリケーションの詳細は、以下の Web サイトにある Servlet 2.2 API 仕様を参照してください。

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

この章では、Web アプリケーションが iPlanet Web Server でサポートされる方法について説明します。この章は、以下の節から構成されています。

- Web アプリケーションの構造
- 動的再構成 (DR)
- `web.xml` ファイルについて
- `web-apps.xml` ファイルおよび仮想サーバ
- `wdeploy` を使用した Web アプリケーションの導入
- Web アプリケーションの例

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になっていて、すべてマッピングからアプリケーションのドキュメントルート (/catalog など) にアクセス可能です。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あります。

ドキュメントルートの下にある特殊なディレクトリ WEB-INF には、アプリケーションの公開ドキュメントツリーにはない、アプリケーションに関連したファイルがすべて含まれています。WEB-INF に含まれているファイルは、直接クライアントに提供することはできません。WEB-INF の内容は次のとおりです。

- /WEB-INF/web.xml : XML ベースの導入記述子。マッピング、初期設定パラメータ、セキュリティ制約など、Web アプリケーションの構成を指定する
- /WEB-INF/classes/* : サーブレットおよびユーティリティのクラスのディレクトリ
- /WEB-INF/ lib/*.jar : サーブレット、Bean、その他のユーティリティのクラスが含まれる JAR ファイルのディレクトリ

WAR (Web アプリケーションのアーカイブ) ファイルには、完全な Web アプリケーションが圧縮されて入っています。iPlanet Web Server は、WAR ファイルのアプリケーションにはアクセスできません。 iPlanet Web Server が Web アプリケーションにアクセスするには、Web アプリケーション (wdeploy ユーティリティを使って導入) の圧縮を解除する必要があります。

Web アプリケーションの構成には次の 2 つが含まれています。

- web.xml ファイル : 標準の Servlet 2.2 導入記述子。各 Web アプリケーションには固有の web.xml ファイルがあります。web.xml の詳細は、以下の Web サイトにある Servlet 2.2 API 仕様を参照してください。

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

- web-apps.xml ファイル : iPlanet Web Server に固有のファイル。各仮想サーバには固有の web-apps.xml ファイルがあり、このファイルによってその仮想サーバで実行されるすべてのアプリケーションが構成されます。

動的再構成 (DR)

Web アプリケーションに対して変更を行うときに、サーバを再起動する必要はありません。変更は、web-apps.xml ファイルの class-loader 要素の reload-interval 属性で設定された頻度で、自動的に読み込み直されます。

web-xml または web-apps.xml ファイルに対して変更を行うときも、サーバを再起動する必要はありません。ただし、「Apply」リンクをクリックし、「Apply Changes」画面の「Load Configuration Files」ボタンをクリックして変更内容を適用する必要があります。

動的再構成の詳細は、iPlanet Web Server の『NSAPI プログラマーズガイド』の第 1 章「サーバの動作の基本」を参照してください。

Web アプリケーション、サーブレット、または JSP を追加または削除した場合は、サーバを再起動する必要があります。

web.xml ファイルについて

この節では、iPlanet Web Server 6.0 での web.xml ファイルの使用方法について説明します。

login-config

セキュリティ上の制約があって login-config 要素がない場合、または auth-method の指定がない場合は、auth-method はデフォルトで BASIC に設定されます。BASIC 認証では、realm-name の指定がない場合、realm-name はデフォルトで iWS Web Container に設定されます。auth-method が FORM でない場合、form-login-config 要素は指定した場合でも無視されます。

security-constraint

url-pattern の指定がない場合、制約は適用されません。
web-resource-collection 部分要素で http-method の指定がない場合、制約はすべての HTTP メソッドに適用されます。user-data-constraint 部分要素で transport-guarantee の指定がない場合、デフォルトで NONE に設定されます。auth-constraint 部分要素がない場合、または role-name の指定がない場合、アクセスは許可されません。

session-timeout

web.xml に session-timeout が指定されている場合、このタイムアウト値が web-apps.xml の session-manager 要素に指定されたすべての timeOut パラメータ値よりも優先して使用されます。仮想サーバのすべての Web アプリケーションで使用する単一セッションマネージャを構成する場合は、web.xml で指定された値は、適用されません。以下の場合にのみ適用されます。

- 仮想サーバのすべての Web アプリケーションに対してセッションマネージャを構成するわけではない (デフォルト動作)。各 Web アプリケーションには固有のセッションマネージャがある。
- 仮想サーバ全体のセッションマネージャに加えて、Web アプリケーション固有のセッションマネージャを設定する。この場合、Web アプリケーション固有のセッションマネージャの timeOut 値に対してだけ web.xml で指定された値が優先します。

session-timeout 値が分単位で指定されるのに対して、session-manager timeOut パラメータは秒単位で指定されることに注意してください。

セッションマネージャの詳細は、第 6 章「セッションマネージャ」を参照してください。

web-apps.xml ファイルおよび仮想サーバ

web-apps.xml ファイルでは、仮想サーバで実行される Web アプリケーションのコンテキストを定義します。コンテキスト情報には、Web アプリケーションのコンテキストパスや、Web アプリケーションでセッション管理や認証を処理する方法など他のプロパティが設定されています。

web-apps.xml ファイルは、標準の J2EE 導入記述子フォーマットである、.dtd ファイルで指定された適正な .XML ドキュメントに準拠しています。web-apps.xml ファイルでは、コンテキストパスおよび物理的位置によって Web アプリケーションが定義されています。

各 Web アプリケーションでは、Web アプリケーションの処理方法をカスタマイズする iPlanet Web Server 固有の構成要素を定義する場合もあります。たとえば、Web アプリケーションでは、プラグイン可能なセッションマネージメントの定義や Java コンパイラの指定を行うことができます。web-apps.xml ファイルを使うと、グローバル構成要素を定義して複数の Web アプリケーションで共有またはオーバーライドできます。

各 web-apps.xml ファイルは、仮想サーバを定義する server.xml ファイルから参照する必要があります。server.xml の詳細は、iPlanet Web Server の『NSAPI プログラマーズガイド』を参照してください。

仮想サーバのデフォルトコンテキスト

新しい仮想サーバを作成すると、空の web-apps.xml ファイルが作成されます。自動的に作成されるデフォルトコンテキストを使用するか、web-apps.xml ファイルを変更して独自のコンテキストを作成できます。

servlets.properties ファイルがデフォルトの仮想サーバのサーバインスタンスの config ディレクトリにある場合、Web アプリケーションのコンテキストではなく iPlanet Web Server 4.x サーブレットのコンテキストが作成されます。古いバージョンの構成の詳細は、第 8 章「古いバージョンのサーブレットおよび JSP の構成」を参照してください。

server.xml ファイルの例

server.xml ファイルには、Web アプリケーションに関連のある 2 つの変数、webapps_enable と webapps_file が指定されています。

webapps_enable	仮想サーバ用の Web アプリケーションを有効または無効にする true または false 変数。仮想サーバの定義にない場合、Web アプリケーションはデフォルトで有効になります。
webapps_file	仮想サーバ用の web-apps.xml ファイルへのパス

以下の server.xml ファイルでは上記の変数が使用されています。

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- declare any variables to be used in the obj.conf file in the
    ATTLIST below -->
<!DOCTYPE SERVER SYSTEM "server.dtd" [
<!ATTLIST VARS
    docroot CDATA #IMPLIED
     adminusers CDATA #IMPLIED
     webapps_file CDATA #IMPLIED
     webapps_enable CDATA #IMPLIED
    accesslog CDATA #IMPLIED
    user CDATA #IMPLIED
    group CDATA #IMPLIED
    chroot CDATA #IMPLIED
    dir CDATA #IMPLIED
    nice CDATA #IMPLIED
>
]>
```

```
<SERVER legacyls="ls1">
    <VARS accesslog="/iws60/https-server.iplanet.com/logs/access"/>
    <LS id="ls1" ip="0.0.0.0" port="80" security="off"
    acceptorthreads="1">
        <CONNECTIONGROUP id="group1" matchingip="default"
        servername="acme.com" defaultvs="acme.com"/>
    </LS>
    <MIME id="mime1" file="mime.types"/>
    <ACLFILE id="acl1"
    file="/iws60/httpacl/generated.https-server.iplanet.com.acl"/>
    <VSCLASS id="defaultclass" objectfile="obj.conf"
    rootobject="default">
        <VARS docroot="/iws60/docs"/>
        <VS id="acme.com" connections="group1" mime="mime1"
        aclids="acl1">
            <VARS webapps_file="web-apps.xml" webapps_enable="on"/>
            <USERDB id="default" database="default"/>
        </VS>
    </VSCLASS>
</SERVER>
```

`server.xml` ファイルの詳細は、iPlanet Web Server の『NSAPI プログラマーズガイド』の第 8 章「仮想サーバの構成ファイル」を参照してください。

web-apps.xml ファイルの例

`!DOCTYPE` 宣言が以下のフォーマットで存在する必要があることに注意してください。

<!DOCTYPE vs PUBLIC "-//Sun Microsystems, Inc.; iPlanet//DTD Virtual Server Web Applications 6.0//EN" "http://developer.iplanet.com/webserver/dtds/iws-webapps_6_0.dtd">

何らかの理由でこの URL にアクセスできない場合は、以下の DTD ファイルを参照してください。

`"file:/`*server_root*`/bin/https/dtds/iws-webapps_6_0.dtd"` (*server_root* はサーバルートを示す)

Windows NT では、以下のようにドライブ文字を指定します。

`"file:/`*drive*`:/`*server_root_path*`/bin/https/dtds/iws-webapps_6_0.dtd"` (*drive* はドライブ、*server_root_path* はサーバルートパスを示す)

以下の web-apps.xml ファイルでは、セッションマネージャと 1 つの Web アプリケーションを構成します。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- iWS 6.0 specific web application configuration. -->
<!DOCTYPE vs  PUBLIC "-//Sun Microsystems, Inc.; iPlanet//DTD
Virtual Server Web Applications 6.0//EN"
"http://developer.iplanet.com/webserver/dtds/iws-webapps_6_0.dtd">

<vs>

<!-- Define global configuration -->

<!- Configure a session manager and tracking configuration -->
<session-manager
   class='com.iplanet.server.http.session.IWSSessionManager'
   <init-param>
      <param-name>maxSessions</param-name>
      <param-value>1000</param-value>
   </init-param>
   <init-param>
      <param-name>timeOut</param-name>
      <param-value>1800</param-value>
   </init-param>
   <init-param>
      <param-name>reapInterval</param-name>
      <param-value>600</param-value>
   </init-param>
   <init-param>
      <param-name>session-data-dir</param-name>
      <param-value>/net/dotcom.com/sessions</param-value>
   </init-param>
</session-manager>
<session-tracking use-cookies="true" />

<!-- Define the web applications for this virtual server-->

<!-- catalog application -->
<web-app uri="/catalog" dir="/webapps/catalog">

<!-- Specify a tempory directory. A path returned in the
"javax.servlet.context.tempdir" property; -- defaults to
WEB-INF/tmp. -->
<tempdir dir='/var/catalog/tmp'/>

<!-- reload classes at every 5 minutes; also include mycatlog.jar
file in the classpath -->
```

```
<class-loader reload-interval='300'
classpath='/home/work/mycatalog.jar' />

</web-app>

</vs>
```

web-apps.xml 要素のリファレンス

この節では、web-apps.xml ファイルで使用できる要素を示します。以下の要素はマスター dtd ファイルで定義されています。

auth-native

認証およびロールマッピング用に、特定のユーザまたはグループのネイティブデータベースを構成します。この要素が指定されていない場合、認証はネイティブのデフォルト認証データベースを使って行われます。

部分要素：なし

属性：

authdb	ネイティブの認証データベース。 このデータベースは、server.xml ファイル中の USERDB 要素の database 属性にも定義する必要があります。

class-loader

仮想サーバまたは Web アプリケーションのクラスローダー。この要素を指定しない場合は、仮想サーバのデフォルトのクラスローダーが使用されます。仮想サーバレベルのクラスローダーで読み込まれるクラスは、再読み込みは動的に行われません。

ServletContext.getAttribute メソッドを使用すると、クラスローダーおよびクラスローダーのクラスパスまたはコンテキストの再読み込み間隔を検出できます。詳しくは、第 7 章「API に関するヒント」を参照してください。

部分要素：なし

属性：

classpath	クラスローダーが使用するクラスパス

delegate	仮想サーバまたはシステムのクラスローダーを最初に呼び出してクラスを読み込むよう指定します。使用できる値は true および false です。デフォルト値は false です。
reload-interval	Web アプリケーションの変更をサーバが確認する時間間隔 (秒単位)。デフォルト値は 30 です。

description

パラメータの説明。init-param 要素内で使用されます。iPlanet Web Server では、この要素を無視します。

部分要素 : なし

属性 : なし

filter、filter-mapping

Servlet 2.3 仕様から Filter API を実装します。web-app 要素内で使用されます。

iPlanet Web Server 6.0 がサポートするのは web.xml ファイルの Servlet 2.2 API だけですが、Servlet 2.3 仕様の Filter API は web-apps.xml ファイルで使用可能です。

filter 要素および filter-mapping 要素では、Filter API を実装します。どちらも、web-apps.xml ファイルの web-app 要素の部分要素です。filter および filter-mapping は、ファイルの位置以外は Servlet 2.3 仕様で説明されているとおりです。詳しくは、以下の Web サイトを参照してください。

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

form-login-session

シングルサインオンのフォームベースの認証を、仮想サーバのすべての Web アプリケーションに構成します。この要素が指定されない場合は、デフォルトの仮想サーバレベルのセッションマネージャが使用されます。

部分要素 : session-manager

属性 :

cookie-name	セッション ID を追跡する cookie の名前。デフォルト値は iwsformloginid です。
timeOut	セッションのタイムアウト (秒単位)。デフォルト値は 600 (10 分) です。

init-param

上位の要素の初期設定パラメータを指定します。init-param の属性は、上位の要素が参照するオブジェクトによって決まります。

たとえば、上位の要素が session-manager でセッションマネージャが IWSSessionManager の場合、init-param の属性は IWSSessionManager の初期設定パラメータになります。

部分要素：param-name、param-value、description

属性：不定

jsp-servlet

JSP のコンパイル動作を構成します。初期設定パラメータの詳細は、44 ページの「JSP コマンドラインコンパイラ」を参照してください。

部分要素：init-param

属性：

enable	JSP を有効にします。使用できる値は true および false です。デフォルト値は true です。

param-name

パラメータの名前。init-param 要素内で使用されます。

部分要素：なし

属性：なし

param-value

パラメータの値。init-param 要素内で使用されます。

部分要素：なし

属性：なし

parameter-encoding

フォームからパラメータを復号化する方法について Web サーバに指示します。

部分要素：なし

属性:

enc	使用できる値は、auto (デフォルト値)、none、または utf8 や Shift_JIS などの特定のコードです。	
	サポートされている Java 文字のコード	utf8 や Shift_JIS など、特定の符号化方式。サーブレットのパラメータが使用する符号化方式がわかっている場合、このオプションを設定します。以下の Web サイトにリストがあります。 http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html
	none	システムのデフォルトの符号化方式を使用します。サーブレットのパラメータのデータの符号化方式がシステムのデフォルトの符号化方式と同じ場合、このオプションを設定します。
	auto	デフォルト。適切な符号化方式を以下の場所から順に探します。1. Content-Type ヘッダーに設定されている場合は charset。2. parameterEncoding 属性。75 ページの「ServletRequest.setAttribute」を参照してください。3. form-hint-field で定義されている隠しフォームフィールド。それ以外の場合、システムのデフォルトの符号化方式が使用されます。このオプションを設定すると、サーブレットパラメータの ASCII 以外の文字が誤って解釈されなくなります。 このプロパティを auto に設定した場合、サーバでは、ネイティブの文字を Java バイト配列に変換してから、要求された符号化方式に変換する必要があります。したがって、none または特定の符号化方式のほうがパフォーマンスがやや向上します。
form-hint-field	符号化方式を指定するフォームの隠しフィールドの名前。デフォルト値は j_encoding です。	

サーブレットで ServletRequest.getParameter メソッドを使用して、UTF-8 以外の文字があるフォームフィールドから値を取得する場合、enc 属性はデフォルトの auto に設定する必要があります。それ以外の場合、getParameter メソッドで取得される値は、ゼロまたは未定義になります。詳しくは、75ページの「ServletRequest.getParameter」を参照してください。

enc 属性の設定は、iPlanet Web Server が要求を受信したあとにサーブレットがパラメータを処理する方法に適用されます。特に、要求方法が GET の場合、サーバに送信される URI は標準の ASCII セットだけから構成されている必要があります。その他の文字はすべて符号化する必要があります。

たとえば、バックスラッシュを符号化する場合、% のあとに ASCII 文字セットでバックスラッシュに対応する 16 進数 5c を続けたものにバックスラッシュを置き換えます。したがって、vw\xyz は vw%5cxyz になります。

詳しくは、以下の場所にある Internet Engineering Task Force の URI ワーキンググループのドキュメントのセクション 2.2 を参照してください。

http://www.ietf.org/rfc/rfc1738.txt

response-buffer

HTTP サーブレットの応答バッファの、初期およびデフォルトのサイズを構成します。サーブレットでは、ServletResponse オブジェクトの setBufferSize メソッドを使用して応答バッファのサイズを再構成できます。

部分要素 : なし

属性 :

flush-timeout	最後のフラッシュ以降指定した秒数が経過した場合、ストリームがデータをフラッシュするようにします。デフォルトの 0 または負の値に設定されると、バッファがフルの場合だけ出力ストリームがフラッシュを行います。
size	バッファのサイズ (バイト単位)。デフォルト値は 8192 です。

response-cookie

指定した cookie のバージョンで応答するようサーバに要求します。

部分要素 : なし

属性 :

version	cookie のバージョン。デフォルト値は 0 です。

role-mapping

role-name 値を web.xml から LDAP ユーザ、グループ、またはロールに対応付けます。

部分要素 : なし

属性 :

map-to	role-name 値を web.xml から LDAP ユーザまたはグループに対応付けるかどうかを指定します。値は、group (デフォルト) または user です。

session-cookie

セッション cookie のパラメータを設定します。

部分要素 : なし

属性 :

domain	この属性がある場合、その値が cookie にタグ付けされます。デフォルト値はありません。
is-secure	true に設定されると、セキュリティ保護された接続で要求を受けた場合、サーバはセッション cookie で secure 属性を送信します。デフォルト値は false です。

session-manager

Web アプリケーションのセッションマネージャ。各セッションマネージャの初期設定パラメータの詳細は、第 6 章「セッションマネージャ」を参照してください。

web.xml で session-timeout が指定されている場合、セッションマネージャの timeOut 初期設定パラメータは無効になります。詳しくは、16 ページの「session-timeout」を参照してください。

部分要素 : init-param

属性 :

class	セッションマネージャのクラス

session-tracking

セッション追跡の方法を判定します。

部分要素：なし

属性：

use-cookies	デフォルトの true の場合、セッション追跡に cookie を使用します。使用できる値は true および false です。
use-url-rewriting	デフォルトの true の場合、セッション追跡に URL 書き換えを使用します。使用できる値は true および false です。

tempdir

Web アプリケーションによって使用される一時的なディレクトリ

部分要素：なし

属性：

dir	一時的なディレクトリ

vs

web-apps.xml ファイルのトップレベルの要素。web-app 以外の部分要素では、すべての Web アプリケーションに対してデフォルトを設定します。

部分要素：auth-native、class-loader、form-login-session、jsp-servlet、parameter-encoding、response-buffer、response-cookie、role-mapping、session-manager、session-tracking、session-cookie、tempdir、web-app

属性：なし

web-app

Web アプリケーション。Web アプリケーションは WAR ファイルにパッケージ化されており、サーブレット、JSP、HTML ページ、クラスファイル、その他のアプリケーションリソースを格納することができます。

web-app 要素の部分要素は、その Web アプリケーションの上位の vs 要素に同等の部分要素がある場合、それをオーバーライドします。

部分要素： auth-native、class-loader、filter、filter-mapping、jsp-servlet、parameter-encoding、response-buffer、response-cookie、role-mapping、session-manager、session-tracking、session-cookie、tempdir

属性：

dir	Web アプリケーションが置かれているディレクトリ
uri	クライアントが Web アプリケーションへのアクセスに使用する URI。この URI は正規表現にできます。

wdeploy を使用した Web アプリケーションの導入

手動で Web アプリケーションを導入する前に、*server_root*/bin/https/httpadmin/bin ディレクトリがパスにあり、IWS_SERVER_HOME 環境変数が *server_root* ディレクトリに設定されていることを確認してください。

コマンドラインで wdeploy ユーティリティを使用して、仮想サーバの Web アプリケーション環境に WAR ファイルを導入できます。

wdeploy deploy -u *uri_path* -i *instance* -v *vs_id* [-d *directory*] *war_file*

仮想サーバの Web アプリケーションを削除することもできます。

wdeploy delete -u *uri_path* -i *instance* -v *vs_id* hard|soft

仮想サーバの Web アプリケーションの URI およびディレクトリを一覧表示することもできます。

wdeploy list -i *instance* -v *vs_id*

コマンドのパラメータには以下のような意味があります。

uri_path	Web アプリケーションの URI 接頭辞
instance	サーバインスタンス名
vs_id	仮想サーバの ID
directory	省略可能。アプリケーションが導入される、または削除されるディレクトリ。導入用に指定しない場合、アプリケーションはドキュメントのルートディレクトリに導入されます。
hard|soft	ディレクトリと web-apps.xml エントリを削除 (hard) するか、web-apps.xml エントリだけを削除 (soft) するかを指定します。
war_file	WAR ファイルの名前

警告 Web アプリケーションを導入する際に *directory* を指定しない場合、アプリケーションはドキュメントのルートディレクトリに導入されます。その場合、hard パラメータを使用してアプリケーションを削除すると、ドキュメントのルートディレクトリも削除されます。

wdeploy deploy コマンドを実行すると、以下の 2 つの事柄が生じます。

- 指定された *uri_path* と *directory* とともに Web アプリケーションが web-apps.xml ファイルに追加されます。
- .WAR ファイルが指定された *directory* に抽出されます。

以下に例を示します。

```
wdeploy deploy -u /hello -i server.iplanet.com -v acme.com
-d /iws60/https-server.iplanet.com/acme.com/web-apps/hello
/iws60/plugins/servlets/examples/web-apps/HelloWorld/HelloWorld.war
```

このユーティリティを実行した結果として、以下の web-apps.xml エントリが生じます。

```
<vs>
    <web-app uri="/hello"
    dir="/iws60/https-server.iplanet.com/acme.com/webapps/hello"/>
</vs>
```

以下に、/iws60/https-server.iplanet.com/acme.com/web-apps/hello ディレクトリの内容を示します。

```
colors
index.jsp
META-INF
 WEB-INF/
    web.xml
    /classes/
        HelloWorldServlet.class
        HelloWorldServlet.java
        SnoopServlet.class
        SnoopServlet.java
```

サーバインスタンスの server.xml ファイルが仮想サーバの web-apps.xml ファイルを指していることを確認してから、導入した Web アプリケーションを実行します。

アプリケーションを導入すると、ブラウザの以下の URL からそのアプリケーションにアクセスできます。

http://*vs_urlhost*[:*vs_port*]/*uri_path*/[*index_page*]

URL の各部には以下のような意味があります。

vs_urlhost	仮想サーバの `urlhosts` 値のひとつ
vs_port	省略可能。仮想サーバがデフォルトのポートを使用しない場合に限り必要
uri_path	アプリケーションの導入に使用したものと同じ。コンテキストパス
index_page	省略可能。エンドユーザが最初にアクセスするアプリケーションのページ

以下に例を示します。

`http://acme.com:80/hello/index.jsp`

または

`http://acme.com/hello/`

Web アプリケーションの例

iPlanet Web Server 6.0 には Web アプリケーションの例が付属しています。Web アプリケーションの例は以下の場所にあります。

server_root`/plugins/servlets/examples/web-apps`

以下のディレクトリがあります。

- `HelloWorld`：`HelloWorld.war` ファイルにある単純な Web アプリケーション
- `filter-test`：`filter-test.war` ファイルにある Filter API の機能の例
- `utility-taglib`: `utility-taglib.war` ファイルにある JSP タグライブラリの例。このタグライブラリは Jakarta プロジェクトによって作成された。Web サイトは `jakarta.apache.org`

これらの例は、`wdeploy` ユーティリティを使用して導入できます。

第 3 章

サーブレットの使用方法

この章では、iPlanet Web Server 6.0 でサーブレットを有効にする方法および構成する方法について説明します。この章は、以下の節から構成されています。

- サーブレットを実行するためにサーバで必要なもの
- ユーザインタフェースの使用
- サーブレットの有効化
- クライアントでサーブレットを使用できるようにする方法
- <SERVLET> SHTML タグ
- サーブレットの出力
- JVM の構成
- サーブレットのパフォーマンスの最大化

サーブレットを実行するためにサーバで必要なもの

iPlanet Web Server 6.0 には、Java サーブレットの開発に必要なファイルがすべて含まれています。`servlet.jar` ファイルは、以下の iPlanet Web Server 6.0 インストールディレクトリにあります。

server_root`/bin/https/jar`

`servlet.jar` ファイルを Java コンパイラにアクセス可能にしてから、サーブレットのコンパイルを行います。CLASSPATH に `servlet.jar` ファイルを指定してください。

iPlanet Web Server 6.0 には JRE (Java Runtime Environment) が付属していますが、ライセンス制限のため JDK (Java Development Kit) は付属していません。サーバは、JRE だけまたは JDK を使用してサーブレットを実行できます。JDK のインストールの詳細は、38 ページの「JSP を実行するためにサーバで必要なもの」を参照してください。

ユーザインタフェースの使用

ユーザインタフェースを使用してサーブレットの設定を行う方法については、オンラインヘルプの以下の内容を参照してください。

次のページは Web Server Administration Server の「Global Settings」タブにあります。

- 「Configure JRE/JDK Paths」ページ

次のページはサーバマネージャの「Java」タブにあります。

- 「Enable/Disable Servlets/JSP」ページ
- 「Configure JVM Attributes」ページ
- 「Delete Version Files」ページ

次のページはクラスマネージャの「Virtual Servers」タブにあります。クラスマネージャを開くには、サーバマネージャの「Virtual Server Class」タブにある Manage Classes ページを選択し、リストからクラスを選択して「Manage」ボタンを押します。

- 「Java Web Apps Settings」ページ

サーブレットの有効化

サーブレットを有効にするには、サーバマネージャの「Java」タブを選択し、「Enable/Disable Servlets/JSP」タブを選択します。サーバ全体に対してサーブレットを有効にするには、「Enable Java Globally」ボックスをクリックします。単一の仮想サーバクラスに対してサーブレットを有効にするには、「Enable Java for Class」ボックスをクリックします。サーバ全体に対して Java を有効にしてから、単一クラスに対してサーブレットを有効にします。デフォルトでは、Java はサーバ全体および各仮想サーバクラスに対して有効です。

クライアントでサーブレットを使用できるようにする方法

以下のどちらかの方法で、クライアントでのサーブレットの使用が可能になります。

- サーブレットを Web アプリケーションに設定して、その Web アプリケーションを導入する。その方法の詳細は、第 2 章「Web アプリケーション」を参照してください。
- デフォルトの仮想サーバでサーブレットを設定する。これは、iPlanet Web Server 4.*x* との下位互換用に提供されています。その方法の詳細は、第 8 章「古いバージョンのサーブレットおよび JSP の構成」を参照してください。

<SERVLET> SHTML タグ

iPlanet Web Server 6.0 は、Java Web Server で紹介されたように、`<SERVLET>` タグをサポートしています。このタグを使用して SHTML ファイルにサーブレットの出力を組み込むことができます。構成を変更しないで、この動作を有効にできます。SSI とサーブレットがどちらも有効の場合、`<SERVLET>` タグも有効になります。

`<SERVLET>` タグの構文は、その他の SSI コマンドとはやや異なり、`<APPLET>` タグの構文に似ています。

```
<servlet name=name code=code codebase=path iParam1=v1  iParam2=v2>
<param name=param1 value=v3>
<param name=param2 value=v4>
.
.
</servlet>
```

サーブレットが Web アプリケーションの一部である場合、code パラメータが必要でその他のパラメータは無視されます。code パラメータには以下の値が設定されている必要があります。

- Web アプリケーションの web.xml ファイルで定義されている url-pattern 要素の値。web.xml の詳細は、以下の Web サイトにある Servlet 2.2 API 仕様を参照してください。

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

- Web アプリケーションの web-apps.xml ファイルで定義されている uri 属性の値。web-apps.xml の詳細は、第 2 章「Web アプリケーション」を参照してください。

たとえば、以下のタグ構文を SHTML ファイルに指定する場合、

```
<servlet name=pparams code="/PrintApp/PrintParams">
</servlet>
```

web-apps.xml ファイルに以下の値を指定する必要があります。

```
<web-app uri="/PrintApp"
dir="/iws60/https-server.iplanet.com/acme.com/webapps/PrintApp"/>
```

web.xml ファイルに以下のタグ構文も指定する必要があります。

```
<servlet>
    <servlet-name>pparams</servlet-name>
    <servlet-class> PrintPackage.PrintParams </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>pparams</servlet-name>
    <url-pattern> /PrintParams </url-pattern>
</servlet-mapping>
```

また、web.xml ファイルにサーブレットのすべての初期設定パラメータも指定する必要があります。

古いバージョン (iPlanet Web Server 4.*x*) のサーブレットの場合、サーブレットの .class ファイルを指定する code パラメータが必要です。servlets.properties ファイルにサーブレットの定義がなく、.class ファイルが <SERVLET> タグが指定されている HTML ファイルとは違うディレクトリにある場合、codebase パラメータが必要です。古いバージョンのサーブレットは、デフォルトの仮想サーバで構成されている必要があり、web.xml ファイルは必要ありません。

SSI コマンドの詳細は、iPlanet Web Server の『プログラマーズガイド』を参照してください。

サーブレットの出力

デフォルトで iPlanet Web Server がバックグラウンドで起動すると、サーブレットの `System.out` および `System.err` 出力は Web サーバのエラーログに送信されません。これは、サーブレットが iPlanet Web Server の外部にあるためです。

UNIX では、*server_root*`/https-`*server_id*`/start` ファイルを変更して、フォアグラウンドで iPlanet Web Server を起動するか、サーブレットの出力をリダイレクトできます。最初に、*server_root*`/https-`*server_id* ディレクトリで以下のコマンドを入力します。

```
./start -shell
```

このコマンドで *server_root*`/bin/https/bin` ディレクトリに入れます。次に、以下のコマンドを入力します。

```
./ns-httpd -d server_root/https-server_id/config
```

Windows NT では、以下の行を `magnus.conf` ファイルに指定すると、iPlanet Web Server を Windows NT コンソールで、フォアグラウンドで実行することができます。

```
Init fn="nt-console-init" stdout=console stderr=console
```

JVM の構成

必要に応じて、サーバマネージャのインタフェースで 「Java」>「Configure JVM Attributes」ページを使用するか、`jvm12.conf` を編集して JVM のパラメータを構成できます。JVM の設定の詳細は、付録 C「JVM の構成」を参照してください。

iPlanet Web Server の JVM 用のデフォルト設定は、サーブレットの実行に適しています。ただし、設定の変更が必要な場合もあります。たとえば、サーブレットまたは Bean ファイルで JAR ファイルを使用する場合、JVM クラスパスに JAR の場所を追加することができます。

サーブレットのパフォーマンスの最大化

サーブレットのパフォーマンスを向上させるため、以下のガイドラインを考慮してください。

- `jvm12.conf` ファイルには設定パラメータ、`jvm.stickyAttach` があります。このパラメータの値を 1 に設定すると、JVM に接続していることをスレッドが認識するため、`AttachCurrentThread` および `DetachCurrentThread` 呼び出しが排除されて要求の処理速度が向上します。ただし、そのために問題が起こる場合もあります。再利用されたスレッドのその他の処理が、ガベージコレクタによって任意に中断されることがあります。

  ほかのサブシステムでこの問題が起こらないようにするにはスレッドプールを使用します。スレッドプールの詳細は、iPlanet Web Server の『管理者ガイド』を参照してください。

- `StackSize` 指令を使用して `magnus.conf` のフロントエンドスレッドのスタックサイズを増やします。スレッドプールを使用している場合は、各プールのスタックサイズのパラメータを増やします。詳しくは、iPlanet Web Server の『NSAPI プログラマーズガイド』を参照してください。

- ヒープサイズを増やしてガベージコレクションを容易にします。`jvm.minHeapSize` または `maxHeapSize`、あるいは「Configure JVM Attributes」ページを使用します。

- 必要のない例がある場合は、`jvm.classpath` からはずします。`jvm.include.CLASSPATH=0` に設定すると、`CLASSPATH` 環境変数を継承しないようにできます。

- JIT コンパイラが有効になっているときにアプリケーションが深い再帰処理を使用すると、iPlanet Web Server 6.0 のスタック空間が不足することがあります。特に、デフォルトのスタックサイズが小さい UNIX プラットフォームで、または非常に複雑な JSP ページが使用されている場合には、注意が必要です。

  `magnus.conf` ファイルの `StackSize` 指令を使用してスタック空間を設定できます。詳しくは、iPlanet Web Server の『NSAPI プログラマーズガイド』を参照してください。

- サーブレットセッションに使用されるセッション ID ジェネレータでは、暗号化機能が強化された固有の乱数生成アルゴリズムが採用されています。これにより、処理速度の遅い古いマシンを使用している場合は、パフォーマンスに関する問題が発生することがあります。詳しくは、第 6 章「セッションマネージャ」を参照してください。

注　Java を使用しないで SSL サーバを稼動している場合、サーバの起動スクリプトで SmartHeap を有効にしてパフォーマンスを向上させることができます。ただし、SmartHeap には Java との互換性はありません。

第 4 章

JSP (JavaServer Pages) の使用方法

この章では、iPlanet Web Server 6.0 で JSP (Java Server Pages) を有効にする方法および構成する方法について説明します。この章は、以下の節から構成されています。

- JSP を実行するためにサーバで必要なもの
- JRE/JDK パスの構成
- ユーザインタフェースの使用
- JSP の有効化
- クライアントで JSP を使用できるようにする方法
- キャッシュのバージョンファイルの削除
- JSP コマンドラインコンパイラ
- JSP タグライブラリおよび標準のポータブルタグ

JSP を実行するためにサーバで必要なもの

iPlanet Web Server 6.0 には JRE (Java Runtime Environment) が付属していますが、ライセンス制限のため JDK (Java Development Kit) は付属していません。サーバでは JRE を使用してサーブレットやコンパイル済みの JSP を実行できますが、新しい JSP の開発やコンパイルされていない JSP の導入には JDK が必要です。JSP の事前コンパイルの詳細は、44 ページの「JSP コマンドラインコンパイラ」を参照してください。

iPlanet Web Server 6.0 では、表 4-1 に示したバージョン以降の JRE または JDK を、プラットフォームに合わせて使用する必要があります。

表 4-1 プラットフォームでサポートされる JRE または JDK のバージョン

プラットフォーム	JRE/JDK/JVM/JIT のバージョン	コメント
Sun Solaris 2.6、2.8	Solaris VM (ビルド Solaris_JDK_1.2.2_07、ネイティブスレッド、sunwjit)	jvm12.conf にある -Xrs フラグをコメントにしてスタックトレースを生成します。詳しくは、51 ページの「デバッグ用のスタックトレースの生成」を参照してください。 hprof など JVMPI ベースのプロファイル、または Solaris dbx を使用するデバッグには、以下の Web サイトからダウンロードできるリファレンス実装を使用してください。 http://java.sun.com/products/jdk/1.2/jre/
Windows NT 4.0	Java バージョン 1.2.2 Classic VM (ビルド JDK-1.2.2_007、ネイティブスレッド、symcjit)	
HPUX	Java バージョン 1.2.2.07 Classic VM (ビルド JDK1.2.2.07-00/12/08-PA_RISC1.1、ネイティブスレッド、HP)	iPlanet には、別の HotSpot VM (1.0.1fcs、混合モード、PA2.0 build 1.2.2.07-00/12/08-PA_RISC2.0) もバンドルされています。この VM は有効になっていません。このバージョンの使用法の詳細は、以下の Web サイトを参照してください。 http://www.unix.hp.com/java/infolibrary/prog_guide/java2/hotspot.html
AIX	Java バージョン 1.2.2 Classic VM (J2RE 1.2.2 IBM ビルド ca122-20001206 (JIT 有効 : jitc))	
Compaq Tru64	Java バージョン 1.2.2-8 Classic VM (ビルド J2SDK.v.1.2.2:10/31/2000-18:00、ネイティブスレッド、jit_122)	iPlanet Web Server の Compaq バージョンは Compaq から入手できます。

表 4-1　プラットフォームでサポートされる JRE または JDK のバージョン (続き)

プラットフォーム	JRE/JDK/JVM/JIT のバージョン	コメント
RedHat Linux 6.2	Java バージョン 1.2.2 Classic VM (ビルド Linux_JDK-1.2.2_FCS、ネイティブスレッド、sunwjit)	この JVM のバージョンは blackdown.org から入手できます。

必要な JDK バージョンの更新については、iPlanet Web Server の『インストールと移行』および最新のリリースノートを参照してください。

JDK 1.2 およびその他の JDK バージョンは、次の Sun Microsystems の Web サイトで入手できます。

```
http://java.sun.com/products/jdk/1.2/
```

JDK へのパスは以下のいずれかの方法で指定することができます。

- サーバのインストール中にパスの指定ができます。

  iPlanet Web Server 6.0 のインストール時に、カスタム JDK (Java Development Kit) を使用するかどうか尋ねるダイアログボックスが表示されるので、そこでパスを指定できます。

- サーバのインストール後に指定することもできます。

  JDK へのパスを指定するには、Web Server Administration Server に切り替えて、「Global Settings」タブを選択し、「Configure JRE/JDK Paths」ページを使用します。詳しくは、40 ページの「JRE/JDK パスの構成」を参照してください。

インストール中またはインストール後に JDK へのパスを指定する場合、そのパスには JDK をインストールしたディレクトリを指定します。

JRE/JDK パスの構成

iPlanet Web Server 6.0 のインストール時に、サーバに付属の JRE (Java Runtime Environment) をインストールするか、使用している JRE または JDK (Java Development Kit) へのパスを指定するかを選択できます。

サーバでは、JRE を使用してサーブレットを実行できますが、コンパイルされていない JSP を実行するには JDK が必要です。JDK は、iPlanet Web Server にバンドルされていませんが、以下の Sun Microsystems の Web サイトからダウンロードできます。

`http://java.sun.com/products/jdk/1.2/`

iPlanet Web Server 6.0 では、38 ページの「JSP を実行するためにサーバで必要なもの」に一覧表示されたバージョンの JDK を使用してください。

インストール中に JRE のインストールまたは JDK へのパスの指定のどちらを選択するかにかかわらず、JRE または JDK の使用の切り替えをいつでも iPlanet Web Server に命令することができます。Web Server Administration Server に切り替えて、「Global Settings」タブを選択し、「Configure JRE/JDK Paths」ページを使用します。このページでも JDK へのパスを変更することができます。

「Configure JRE/JDK Paths」ページで 「JDK」ラジオボタンを選択した場合、以下のフィールドに値を入力します。

- JDK Path

  JDK のパスを入力します。これは JDK をインストールしたディレクトリです。

- JDK Runtime Libpath

  JDK の実行時ライブラリパスを入力します。

- JDK Runtime Classpath

  クラスパスには、サーブレットエンジンの実行に必要なディレクトリと `jar` ファイルへのパス、サーブレットの例、および追加したサーブレットに必要なその他のパスが設定されます。新しいクラスパスの値を追加することはできますが、既存の値にはサーブレットの操作に必須のパスが設定されているので削除しないでください。

「JRE」ラジオボタンを選択した場合、以下のフィールドに値を入力します。

- JRE Path

  JRE のパスを入力します。これは JRE をインストールしたディレクトリです。

- JRE Runtime Libpath

  JRE の実行時ライブラリパスを入力します。

注 JDK 実行時ライブラリパス、JDK 実行時クラスパス、または JRE 実行時ライブラリパスがわからない場合、フィールドには何も入力せずデフォルトのパスが使用されるようにしてください。

JRE と JDK の切り替えには「Configure JRE/JDK Paths」ページを使用するのがいちばん簡単ですが、以下のようにプログラミング用ファイルを編集しても行えます。

- UNIX では次の操作を行います。

  server_root/https-admserv/start-jvm ファイルを編集します。

  現在サーバで JRE を使用している場合、このファイルには変数 NSES_JRE がインクルードされています。サーバで JDK を使用するには、変数 NSES_JDK を追加し、その値として JDK のディレクトリを指定します。NSES_JRE 変数の値を変更する必要もあります。

  NSES_JDK は JDK のインストールディレクトリを、NSES_JRE は JDK のインストールディレクトリにある JRE ディレクトリ、つまり *jdk_dir*/jre を指している必要があります。

- Windows NT では次の操作を行います。

  Java ライブラリへのパスを magnus.conf の中の extrapath 設定に追加します。

  レジストリ HKEY_LOCAL_MACHINE/SOFTWARE/Netscape/Enterprise/6.0/ で NSES_JDK および NSES_JRE 変数を編集します。サーバで JDK の使用が有効になっている場合、両方の変数が必要です。サーバで使用するのが JRE の場合は、NSES_JRE 変数だけを設定します。

  NSES_JDK は JDK のインストールディレクトリを、NSES_JRE は JDK のインストールディレクトリにある JRE ディレクトリ、つまり *jdk_dir*/jre を指している必要があります。

警告 これらの Windows NT レジストリエントリを編集するときは、十分注意してください。値を間違えると、iPlanet Web Server の再インストールが必要になる場合もあります。

注 JRE または JDK パスへの変更を有効にするには、サーバマネージャの「Preferences」タブにある「On/Off」オプションからサーバを再起動する必要があります。

ユーザインタフェースの使用

ユーザインタフェースを使用した JSP の設定の詳細は、オンラインヘルプで以下の内容を参照してください。

次のページは Web Server Administration Server の「Global Settings」タブにあります。

- 「Configure JRE/JDK Paths」ページ

次のページはサーバマネージャの「Java」タブにあります。

- 「Enable/Disable Servlets/JSP」ページ
- 「Configure JVM Attributes」ページ
- 「Delete Version Files」ページ

次のページはクラスマネージャの「Virtual Servers」タブにあります。クラスマネージャを開くには、サーバマネージャの「Virtual Server Class」タブにある「Manage Classes」ページを選択し、リストからクラスを選択して「Manage」ボタンを選択します。

- 「Java Web Apps Settings」ページ

JSP の有効化

JSP を有効にするには、以下の手順に従います。

1. サーブレットを有効にします。サーバマネージャの「Java」タブを選択し、「Enable/Disable Servlets/JSP」タブを選択します。「Enable Java Globally」ボックスをチェックし、サーバ全体に対してサーブレットを有効にします。単一仮想サーバクラスに対してサーブレットを有効にするには、「Enable Java for Class」ボックスをチェックします。サーバ全体に対して Java を有効にしてから、単一クラスに対してサーブレットを有効にします。デフォルトでは、Java はサーバ全体および各仮想サーバクラスに対して有効になっています。
2. `enable="true"` に設定されている `jsp-servlet` 要素を `web-apps.xml` ファイルに追加します。`web-apps.xml` ファイルの詳細は、第 2 章「Web アプリケーション」を参照してください。
3. JVM クラスパスに `tools.jar` を追加します。詳しくは、付録 C「JVM の構成」を参照してください。

クライアントで JSP を使用できるようにする方法

以下のどちらかの方法で、JSP がクライアントで使用できるようになります。

- JSP を Web アプリケーションに追加して、その Web アプリケーションを導入する。その方法の詳細は、第 2 章「Web アプリケーション」を参照してください。
- デフォルトの仮想サーバで JSP を設定する。これは、iPlanet Web Server 4.*x* との下位互換用に提供されています。その方法の詳細は、第 8 章「古いバージョンのサーブレットおよび JSP の構成」を参照してください。

キャッシュのバージョンファイルの削除

JSP (JavaServer Pages) の情報をキャッシュするために、サーバでは以下のディレクトリが使用されます。

server_root/https-*server_id*/ClassCache/*virtual_server_id*/*webapp_uri*/
(*virtual_server_id* は仮想サーバ ID、*webapp_uri* は Web アプリケーションの URI を示す)

web-apps.xml ファイルにある jsp-servlet 要素の scratchdir 初期設定パラメータを使用して、JSP クラスキャッシュディレクトリの場所を変更できます。詳しくは、47 ページの「その他の JSP 初期設定パラメータ」を参照してください。

サーバは JSP ページを処理すると、JSP に関連付けられた .java および .class ファイルを作成し、ClassCache ディレクトリの下位の JSP クラスキャッシュに格納します。

このキャッシュには、バージョン番号が記述されている Version ファイルが格納されており、サーバはこのファイルを使用してディレクトリの構造とキャッシュ内のファイルを判別します。バージョンファイルを削除するだけでキャッシュをクリーンアップすることができます。

サーバは、起動時にバージョンファイルを検出できない場合、対応するキャッシュのディレクトリ構造を削除してバージョンファイルを作成し直します。次回サーバが JSP ページを処理するときに、JSP クラスキャッシュが作成し直します。

バージョンファイルの削除は、通常のファイルを削除するときと同じように ClassCache ディレクトリから削除するだけで行えます。または、サーバマネージャの「Java」>「Delete Version Files」ページで削除することもできます。バージョンファイルを削除したあとは、必ず iPlanet Web Server を再起動してください。そうすることにより、該当するキャッシュがクリーンアップされ、サーバが JSP を処理する前にバージョンファイルが作成し直されます。

JSP コマンドラインコンパイラ

JSP を事前にコンパイルしてパフォーマンスを向上させることができます。これは本稼動用のサーバにお勧めします。コマンドラインの JSP コンパイラが iPlanet Web Server にインクルードされています。JSP コンパイラは *server_root*/bin/https/bin にあります。このディレクトリがパスにあることを確認してください。IWS_SERVER_HOME 環境変数は、*server_root* ディレクトリに設定する必要があります。

jspc コマンドの形式を以下に示します。

jspc [*options*] *jsp_files* (*options* はオプションを示す)

jsp_files は、以下のどちらかにできます。

files	コンパイルされる一つ以上の JSP ファイル
-webapp *dir*	Web アプリケーションが含まれるディレクトリ。このディレクトリとサブディレクトリに存在する JSP がすべてコンパイルされます。WAR、JAR、または ZIP ファイルを指定することはできません。

jspc コマンドのオプションを以下に示します。

-q	-v0 と同様、Quiet モードを有効にします。重大なエラーメッセージだけが表示されます。
-v[*level*]	冗長モードを有効にします。*level* は省略可能で、デフォルト値は 2 です。*level* の値を以下に示します。 • 0 - 重大なエラーメッセージだけ • 1 - エラーメッセージだけ • 2 - エラーおよび警告メッセージだけ • 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
-d *dir*	コンパイル済みの JSP の出力ディレクトリを指定します。パッケージディレクトリは、コンパイルされていない JSP のあるディレクトリに基づいて自動的に生成されます。デフォルトのトップレベルディレクトリは、jspc が起動されるディレクトリです。

`-dd` *dir*	コンパイル済みの JSP のリテラル出力ディレクトリを指定します。パッケージディレクトリは作成されません。デフォルトは `jspc` が起動されるディレクトリです。
`-p` *name*	指定したすべての JSP のターゲットパッケージの名前を指定して、`-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取り消します。
`-c` *name*	最初にコンパイルされる JSP のターゲットクラスの名前を指定します。以降にコンパオイルされる JSP は影響されません。
`-mapped`	HTML 行ごとの `write` 呼び出しと、JSP ファイル内の各行の位置を記述したコメントを生成します。デフォルトでは、隣接した `write` 呼び出しはすべて結合されており、位置に関するコメントは生成されません。
`-die[`*code*`]`	重大なエラーが発生した場合、JVM を終了してエラーリターン *code* を生成します。*code* がないか、解析できない場合は、デフォルト値の `1` に設定されます。
`-uribase` *dir*	コンパイルされたファイルと関係のある URI ディレクトリを指定します。明示的に宣言された JSP ファイルだけに適用されます。 これは、`uriroot` と関係のある各 JSP ファイルの場所です。判定できない場合、デフォルトは「`/`」になります。
`-uriroot` *dir*	URI ファイルを解決する対象となるルートディレクトリを指定します。明示的に宣言された JSP ファイルだけに適用されます。 このオプションが指定されていない場合、最初の JSP ページのすべての親ディレクトリの `WEB-INF` サブディレクトリが検索されます。サブディレクトリがあって JSP ページにもっとも近いディレクトリが使用されます。 JSP の親ディレクトリに `WEB-INF` サブディレクトリがない場合、`jspc` が起動されるディレクトリが使用されます。
`-webinc` *file*	`-webapp` オプションの部分サーブレットマッピングを作成します。このマッピングは、`web.xml` ファイルにペーストできます。
`-webxml` *file*	`-webapp` オプションの完全な `web.xml` ファイルを作成します。
`-ieplugin` *class_id*	Internet Explorer の Java プラグイン COM クラス ID を指定します。`<jsp:plugin>` タグによって使用されます。
`-genclass`	Java ファイルとクラスファイルを生成します。JDK の `tools.jar` ファイルが JVM クラスパスにある必要があります。

-webinc および -webxml オプションは、iPlanet Web Server の JSP には役立たない場合があります。

たとえば、以下のコマンド (1 行に入力) では、HelloWorld Web アプリケーションの JSP をコンパイルします。

```
jspc -d dir -genclass -webapp
server_root/plugins/servlets/examples/web-apps/HelloWorld
```

コンパイルされた JSP は、*dir*/_jsps/ の下に書き込まれます。次に、これらのクラスファイルを .JAR ファイルに置くことができます。

JSP コンパイラの詳細は、以下の Jakarta の Web サイトを参照してください。

```
http://jakarta.apache.org/
```

Jasper と iPlanet Web Server 6.0 は密接に統合されてはいないので、タグライブラリ、Bean などを使用した JSP の導入には、JVM クラスパスの編集が必要な場合があります。JVM クラスパスは、サーバマネージャの「Configure JVM Attributes」ページ、または jvm12.conf ファイルにあります。JVM の設定の詳細は、付録 C「JVM の構成」を参照してください。

JSP コンパイラで生成されたパッケージ名

コンパイルされた JSP には、パッケージが作成されます。パッケージ名は _jsps で始まり、先頭に下線が付いた JSP のパス名の構成要素がきます。たとえば、/myjsps/hello.jsp に対して生成されるパッケージ名は _jsps._myjsps になります。

生成されたサーブレットに関連付けられるパッケージ名は暗黙のものであるため、JSP で Bean を使用する場合は、明示的な import 指令が必要です。Bean にパッケージ名がない場合は特に必要です。以下に例を示します。

```
<%@page import="MyBean" %>
<jsp:useBean id="myBean" class="MyBean" />
```

JSP のプリコンパイルの指定

web-apps.xml ファイルの jsp-servlet 要素を使って、仮想サーバの JSP が事前にコンパイルされるよう iPlanet Web Server に命令することができます。vs 要素に以下のタグを指定します。

```
<jsp-servlet enable="true">
    <init-param>
        <param-name>use-precompiled</param-name>
        <param-value>true</param-value>
    </init-param>
</jsp-servlet>
```

web-apps.xml ファイルの詳細は、第 2 章「Web アプリケーション」を参照してください。

事前にコンパイルされた JSP を実行するために JDK をインストールする必要はありません。ただし、新しい JSP を開発するには JDK が必要です。JDK のインストールの詳細は、38 ページの「JSP を実行するためにサーバで必要なもの」を参照してください。

その他の JSP 初期設定パラメータ

web-apps.xml ファイルの jsp-servlet 要素には、以下の初期設定パラメータを指定することができます。JSP コンパイラでは、ファイルに追加されていないパラメータにはデフォルト値が使用されます。

keepgenerated	デフォルトの true に設定すると、生成された Java ファイルを保持します。false に設定すると、Java ファイルを削除します。
largeFile	true に設定すると、静的 HTML は別のデータファイルに保存されます。この機能は、JSP が非常に大きい場合に役立ちます。デフォルト値は false です。
scratchdir	生成されたすべてのコードを格納するために作成される作業ディレクトリ。このパラメータが指定されていない場合、デフォルトの場所は *server_root*/https-*server_id*/ClassCache/*virtual_server_id*/*webapp_uri*/ になります。
mappedfile	true に設定すると、HTML 行ごとの write 呼び出しと、JSP ファイル内の各行の位置を記述したコメントを生成します。デフォルトでは、隣接した write 呼び出しはすべて結合されており、位置に関するコメントは生成されません。
ieClassId	Internet Explorer の Java プラグイン COM クラス ID。<jsp:plugin> タグによって使用されます。

use-precompiled	true に設定すると、仮想サーバの JSP が事前にコンパイルされ、実行時にコンパイルする必要がないよう指定されます。デフォルト値は false です。 true に設定すると、JSP への変更は自動的には読み込み直されません。 コマンドライン JSP コンパイラを使用して JSP をコンパイルし、クラスを JAR ファイルに保存して、JAR ファイルを Web アプリケーションの WEB_INF/lib ディレクトリに置くことができます。

JSP タグライブラリおよび標準のポータブルタグ

iPlanet Web Server は、JSP タグライブラリおよび標準のポータブルタグをサポートしています。タグライブラリの詳細は、以下の Web サイトにある JSP 1.1 仕様を参照してください。

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

第 5 章

サーブレットおよび JSP のデバッグ

この章では、iPlanet Web Server 6.0 でサーブレットおよび JSP のデバッグを行うガイドラインを紹介します。以下の節で構成されています。

- サーブレットのデバッグ
- JSP のデバッグ
- デバッグ用のスタックトレースの生成
- Forte for Java を使用したサーブレットおよび JSP のデバッグ
- デバッグ用の JPDA オプション

サーブレットおよび JSP のデバッグには、この章で説明されているように、`jvm12.conf` ファイルの編集が必要です。このファイルに関する一般的な情報については、付録 C「JVM の構成」を参照してください。

サーブレットのデバッグ

サーバで JDK を使用するよう指示している場合は、サーブレットのリモートデバッグを行うことができます。サーバが JRE を使用している場合は、JDK を使用するように切り替えてからリモートデバッグを行います。JDK または JRE の使用をサーバに命令する方法の詳細は、40 ページの「JRE/JDK パスの構成」を参照してください。

サーバが JDK を使用していると仮定すると、以下の手順でリモートデバッグを行うことができます。

1. サーバが単一処理モードで稼動していることを確認します。単一処理モードはデフォルト設定ですが、`magnus.conf` ファイルで `MaxProcs` パラメータに 1 より大きい値が設定されていないことを確認します。`magnus.conf` に `MaxProcs` が設定されていない場合、デフォルト値の 1 が使用されます。単一処理モードと多重処理モードの詳細は、iPlanet Web Server の『管理者ガイド』を参照してください。
2. `jvm12.conf` で以下のパラメータを適宜設定します。

   ```
   jvm.enableDebug=1
   java.compiler=NONE
   ```

3. 例外をログファイルとクライアントに送信するには、以下のパラメータを設定します。クライアントがブラウザの場合、例外はブラウザに表示されます。

   ```
   jvm.trace=7
   ```

4. プラットフォームによっては、起動クラスパスを指定する必要があります。たとえば、Solaris プラットフォームでは、Java 1.2 が `/java` にある場合、`jvm12.conf` で以下のように設定します。

   ```
   jvm.option=-Xbootclasspath:/java/lib/tools.jar:/java/jre/lib/rt.
   jar
   ```

5. サーバを手動で起動し、コンソールに表示されるリモートデバッグのパスワードを記録します。
6. Java デバッガを起動します。

   ```
   jdb -host your_host -password the_password
   ```

これで、`jdb` コマンドを使用して Java クラスのデバッグを行えます。

JSP のデバッグ

以下の手順に従って JSP のデバッグを行います。

1. サーバが単一処理モードで稼動していることを確認します。単一処理モードはデフォルト設定ですが、magnus.conf ファイルで MaxProcs パラメータが 1 より大きい値に設定されていないことを確認します。magnus.conf に MaxProcs が設定されていない場合、デフォルト値の 1 が使用されます。単一処理モードと多重処理モードの詳細は、iPlanet Web Server の『管理者ガイド』を参照してください。
2. jvm12.conf で以下のパラメータを適宜設定します。

   ```
   java.compiler=NONE
   jvm.trace=6
   nes.jsp.enabledebug=1
   ```

3. 例外をログファイルとクライアントに送信するには、以下のパラメータを設定します。クライアントがブラウザの場合、例外はブラウザに表示されます。

   ```
   jvm.trace=7
   ```

java.compiler=NONE に設定すると、ログファイルの詳細出力に Java ソースコードの行番号が追加されます。jvm.trace=6 または jvm.trace=7 に設定すると、JSP コンパイラおよびサーブレットエンジンから詳細出力が行えます。
nes.jsp.enabledebug=1 に設定すると、iPlanet Web Server 6.0 では、デバッグ可能な Java サーブレットを JSP から生成します。

デバッグ用のスタックトレースの生成

以下の手順に従ってデバッグ用の Java スタックトレースを生成できます。

http://developer.java.sun.com/developer/technicalArticles/Programming/Stacktrace/

シグナル使用率の削減のために jvm12.conf ファイルの jvm.option=-Xrs フラグをコメントにしてから、スタックトレースを生成します。-Xrs フラグが使用されていると、トレースの生成のためにシグナルを送信したときに、サーバがコアダンプして再起動する可能性があります。

jvm12.conf ファイル、および jvm.option の使用の詳細は、付録 C「JVM の構成」を参照してください。

Forte for Java を使用したサーブレットおよび JSP のデバッグ

Forte for Java デバッガを使用できるように iPlanet Web Server を設定するには、以下の手順に従います。

1. iPlanet Web Server に必要な JDK バージョン 1.2.*x* がまだインストールされていない場合は、38 ページの「JSP を実行するためにサーバで必要なもの」を参照してインストールします。必要なバージョンはプラットフォームによって異なります。
2. 以下の Web サイトから入手可能な JDK バージョン 1.3 をインストールします。

   `http://java.sun.com/j2se/1.3/`

   iPlanet Web Server では JDK バージョン 1.2.*x* を使用しますが、Forte for Java にはバージョン 1.3 が必要です。
3. 以下の Web サイトから入手可能な Forte for Java の Community Edition 1.0 をインストールします。

   `http://www.sun.com/forte/ffj/ce/`
4. 以下の Web サイトから入手可能な JPDA もインストールすることを強くお勧めします。

   `http://java.sun.com/products/jpda/`
5. JPDA をインストールしたら、*jpda_install*`/bin` ディレクトリのすべてのファイルを *jdk1.2_install*`/jre/bin` ディレクトリにコピーします (*jpda_install* は JPDA のインストール先、*jdk1.2_install* は JDK バージョン 1.2 のインストール先を示す)。また、`jpda.jar` ファイルを *jpda_install*`/lib` ディレクトリから *jdk1.2_install*`/jre/lib/ext` ディレクトリにコピーします。

   JPDA は JDK 1.3 ではなく 1.2 で稼動することに注意してください。
6. Windows NT では、`magnus.conf` ファイルに以下の行を追加して Windows NT コンソールを有効にします。

   ```
   Init fn="nt-console-init" stdout=console stderr=console
   ```

7. `jvm12.conf` ファイルを編集してリモートデバッグを有効にします。JPDA をインストールしなかった場合は、以下の行を追加します。

   ```
   jvm.enableDebug=1
   jvm.compiler=NONE
   ```

   JPDA をインストールした場合は、以下の行を追加します。

   ```
   jvm.enableDebug=1
   jvm.compiler=NONE
   jvm.option=-classic
   jvm.option=-Xnoagent
   jvm.option=-Xrunjdwp:transport=dt_socket,server=y,suspend=n
   ```

8. Forte for Java を起動し、デバッグするサーブレットまたは JSP が設定されているディレクトリをマウントします。

9. iPlanet Web Server を起動します。以下のような行がコンソールに表示されます。

   `Listening for transport dt_socket at address:`*port_number*
   (*port_number* はポート番号を示す)

   この *port_number* を書き留めます。

10. Forte for Java で「Debug」メニューを選択し、「Attach to VM...」オプションを選択します。*port_number* を「Port」テキストボックスに入力し、「OK」を選択します。

サーブレットまたは JSP のデバッグの準備ができました。

デバッグ用の JPDA オプション

JPDA がインストール済みの場合に `jvm12.conf` ファイルに設定することができるデバッグオプションのリストが、以下の Web サイトにあります。

`http://java.sun.com/products/jpda/doc/conninv.html#Invocation`

第 6 章

セッションマネージャ

セッションオブジェクトは、通常ステートレスの HTTP プロトコルを介した複数のページ要求にわたり、状態とユーザの識別情報を管理します。セッションは、ユーザからの複数の接続またはページ要求にわたって、指定された期間、持続します。各セッションは、通常、一人のユーザに対応付けられており、そのユーザはサイトを頻繁に訪れています。サーバは、cookie を使用するか、URL を書き換えることによって、セッションを継続します。サーブレットは、セッションオブジェクトにアクセスして、セッションに関する状態情報を取得することができます。

この章は、以下の節から構成されています。

- セッションの概要
- セッションマネージャの指定
- IWSSessionManager
- MMapSessionManager (UNIX のみ)
- 非推奨のセッションマネージャ
- ロードバランシング、セッションフェイルオーバー、およびセッション ID

セッションの概要

HTTP セッションは、サーバ側のセッションを表しています。サーバは、以下の条件に一致した場合、セッションを新規のものであると判断します。

- クライアントがそのセッションを認識できない
- そのセッションがまだ開始されていない

セッションマネージャは、新規のセッションが開始されると、必ず新規のセッションオブジェクトを自動的に作成します。場合によっては、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が cookie を使用しているのに対して、クライアントが cookie を受け入れていない場合などです。

iPlanet Web Server 6.0 には、セッションの作成および管理用に、以下のセッションマネージャが付属しています。

- `IWSSessionManager` - デフォルトのセッションマネージャ。持続セッション用にデータベースストアまたはファイルストアを使用したり、シングルプロセスまたはマルチプロセスモードで実行することができます。
- `MMapSessionManager`（`UNIX` のみ）- サーバをマルチプロセスモードで実行するためのセッションマネージャ。分散セッションはサポートしていません。
- `SimpleSessionManager` - 非推奨の簡易セッションマネージャ。分散セッションはサポートしていません。
- `JdbcSessionManager` - 非推奨のセッションマネージャ。JDBC API を使用してセッション情報をデータベースに保管し、分散セッションをサポートしています。

マルチプロセスモードは、UNIX プラットフォーム上でのみサポートされています。セッションマネージャのすべてのマルチプロセスモード機能は、Windows NT 上では無視されます。

iPlanet Web Server 6.0 では、ユーザ独自のセッションマネージャを開発し、それをサーバに読み込むこともできます。セッションマネージャクラスのソースコードファイルは、ユーザが独自のセッションマネージャを定義するときに、取り掛かりとして利用できます。これらの Java ファイルは、ディレクトリ *server_root*`/plugins/servlets/iws-apis/sessions` にあります。

セッションマネージャの指定

デフォルトでは、iPlanet Web Server はサーブレット用のセッションマネージャとして IWSSessionManager を使用します。セッションマネージャは以下のいずれかの方法で変更することができます。

- ディレクトリ *server_id*/config に格納されているファイル web-apps.xml を編集する

  以下の例のように、サーブレットまたは JSP の web-app 要素内に session-manager 要素を追加します。

```
<session-manager
       class='com.iplanet.server.http.session.YourSesMgr'>
       <init-param>
              <param-name>maxSessions</param-name>
              <param-value>1000</param-value>
       </init-param>
       <init-param>
              <param-name>timeOut</param-name>
              <param-value>1800</param-value>
       </init-param>
       <init-param>
              <param-name>reapInterval</param-name>
              <param-value>600</param-value>
       </init-param>
</session-manager>
```

  web-apps.xml ファイルの詳細は、第 2 章「Web アプリケーション」を参照してください。

- サーバマネージャインタフェースの「Legacy Servlets」>「Configure Global Servlet Attributes」ページを使用する

  「Session Manager」フィールドでセッションマネージャを指定し、必要に応じて「Session Manager Args」フィールドにパラメータを指定します。

- ディレクトリ *server_id*/config に格納されているファイル servlets.properties を編集する。この方法は、デフォルトの仮想サーバにのみ適用できます。

  servlets.sessionmgr の値を指定する行を追加します。また、必要に応じて、セッションマネージャのパラメータを指定する行も追加します。以下に例を示します。

```
servlets.sessionmgr=com.iplanet.server.http.session.YourSesMgr
servlets.sessionmgr.initArgs=maxSessions=20,timeOut=300,reapInterval=
150
```

- ディレクトリ *server_id*/config に格納されているファイル contexts.properties を編集する。この方法は、デフォルトの仮想サーバにのみ適用できます。

  context.*context_name*.sessionmgr の値を指定する行を追加します。また、必要に応じて、セッションマネージャのパラメータを指定する行も追加します。以下に例を示します。

```
context.global.sessionmgr=com.iplanet.server.http.session.YourSesMg
r
context.global.sessionmgr.initArgs=maxSessions=20,timeOut=300
```

  グローバルコンテキストを変更したり、新しいコンテキストを定義してそれにサーブレットを割り当てたりすることもできます。詳しくは、第 8 章「古いバージョンのサーブレットおよび JSP の構成」を参照してください。

IWSSessionManager

IWSSessionManager は、デフォルトのセッションマネージャです。

IWSSessionManager は、シングルプロセスモードとマルチプロセスモードの両方で動作可能です。異なるマシンで実行されている複数のプロセスのセッション情報を共有するために使用できます。magnus.conf ファイルの MaxProcs 指令によって、サーバがシングルプロセスモードとマルチプロセスモードのどちらで稼動しているかを判別できます。詳しくは、iPlanet Web Server の『NSAPI プログラマーズガイド』を参照してください。

セッションを持続するために、IWSSessionManager は、サーバファームのすべてのサーバからアクセス可能な、データベースまたは分散ファイルシステム (DFS) のパスを使用します。各セッションは、データベースまたは分散ファイルシステムに直列化されます。また、ユーザ独自の持続メカニズムを作成することもできます。

iPlanet Web Server がシングルプロセスモードで稼動している場合、デフォルトではセッションの持続モードは定義されていないため、セッションは持続的ではありません。

iPlanet Web Server がマルチプロセスモードで稼動している場合、デフォルトではセッションは持続的です。持続モードが定義されていない場合は、IWSSessionManager では DFS が使用されます。

マルチプロセスモードは、UNIX プラットフォーム上でのみサポートされています。IWSSessionManager のすべてのマルチプロセスモード機能は、Windows NT 上では無視されます。

IWSSessionManager のパラメータ

IWSSessionManager では、以下のパラメータが使用されます。

- maxSessions - セッションマネージャが一度に管理できるセッションの最大数。セッションマネージャは、すでにセッション数が maxSessions になっている場合、それ以上新規セッションは作成しません。デフォルト値は 1000 です。
- timeOut - クライアントがセッションにアクセスしてからセッションマネージャがそのセッションを無効にするまでの経過時間 (秒単位)。timeOut 秒の間アクセスされなかったセッションは、reaper メソッドによって無効になります。デフォルト値は 1800 (30 分) です。

  web.xml で session-timeout が指定されている場合は、この timeOut パラメータ値は無効になります。詳しくは、16 ページの「session-timeout」を参照してください。
- reapInterval - reaper メソッドが再び呼び出されるまでの SessionReaper スレッドのスリープ時間 (秒単位)。デフォルト値は 600 (10 分) です。
- maxLocks - 複数のプロセスにわたる個別のセッションへのアクセスを同期させるために使用する、クロスプロセスロックの数。デフォルト値は 10 です。このデフォルト値は、値に 0 が指定された場合に使用されます。このパラメータは、シングルプロセスモードでは無視されます。
- session-data-store - セッションを持続する方法を決定するクラスの名前。iPlanet Web Server で提供されているクラスは、以下のとおりです。
  - com.iplanet.server.http.session.JdbcStore
  - com.iplanet.server.http.session.FileStore

  session-data-store パラメータを指定しない場合は、セッションはシングルプロセスモードでは持続的でなくなります。マルチプロセスモードでは、FileStore がデフォルトです。

  JdbcStore および FileStore クラスは、SessionDataStore のサブクラスです。SessionDataStore を拡張することにより、セッションの持続性を実装するための独自のクラスを作成することもできます。

注 JdbcStore を使用する前に、セッション情報を格納するためのテーブルを作成する必要があります。テーブル名は table パラメータを使用して指定し、テーブルの 4 つの列は accessTimeColumn、timeOutColumn、sessionIdColumn、および valueColumn パラメータを使用して指定します。

session-data-store パラメータを JdbcStore または FileStore クラスに設定した場合、IWSSessionManager では以下のパラメータがさらに使用されます。

- session-failover-enabled - 要求のたびにセッションを持続ストアから読み込み直すかどうかと、マルチプロセスモードで常に true に設定するかどうかを指定します。

session-data-store パラメータを FileStore クラスに設定した場合、IWSSessionManager では以下のパラメータがさらに使用されます。

- session-data-dir - すべてのサーバおよび Web アプリケーションのセッションデータを格納するディレクトリ

  session-data-dir パラメータを指定しない場合は、以下のディレクトリがデフォルトで使用されます。

  server_root/*server_id*/SessionData/*virtual_server_id*/*web_app_URI*

session-data-store パラメータを JdbcStore クラスに設定した場合、IWSSessionManager では以下のパラメータがさらに使用されます。

- provider - JDBC ドライバ (デフォルトは sun.jdbc.odbc.JdbcOdbcDriver)。JDBC API の詳細は、以下の Web サイトを参照してください。

  http://java.sun.com/products/jdbc/index.html

注 JdbcStore クラスは、web-apps.xml の class-loader 要素の classpath 属性に割り当てられた JDBC ドライバクラスを認識しません。JDBC ドライバクラスを jvm12.conf ファイルの jvm.classpath 変数に割り当ててください。詳しくは、付録 C「JVM の構成」を参照してください。

- url - データソース (デフォルトは jdbc:odbc:LocalServer)
- table - セッションを格納する SQL テーブルの名前 (デフォルトは sessions)
- username - データベースのログインユーザ名
- password - データベースのログインパスワード
- reaperActive - true に設定された場合 (デフォルト値)、セッション reaper を実行して期限切れのセッションをデータベースから削除するよう、セッションマネージャに指示します。reaper を実行しているサーバがクラスタ内で 1 つのみになるようにしてください。
- accessTimeColumn - 最後にアクセスされた時間 (分単位) を格納する列の名前 (デフォルトは AccessTime)。SQL 型は NUMERIC(9) です。

- `timeOutColumn` - セッションタイムアウト (分単位) を格納する列の名前 (デフォルトは `TimeOut`)。SQL 型は NUMERIC(9) です。
- `sessionIdColumn` - セッション ID を格納する列の名前 (デフォルトは `SessionID`)。SQL 型は VARCHAR(100) です。
- `valueColumn` - セッションオブジェクトを格納する列の名前 (デフォルトは `Value`)。SQL 型は VARBINARY(4096) です。この列は、すべてのセッションデータを格納できるくらいの大きさである必要があります。

セッション情報を処理するためのデータベース上での各操作 (検索、挿入、更新、および削除) は、対応する専用の接続によって実行されます。これらの接続はそれぞれ、パフォーマンスを向上させるために、コンパイル済みの SQL 文を持っています。以下のパラメータを使用して、各操作専用の接続の数をカスタマイズできます。

- `lookupPool` - 検索操作を行う接続の数 (デフォルトは 4)
- `insertPool` - 挿入操作を行う接続の数 (デフォルトは 4)
- `updatePool` - 更新操作を行う接続の数 (デフォルトは 4)
- `deletePool` - 削除操作を行う接続の数 (デフォルトは 2)

IWSSessionManager を有効にする

`IWSSessionManager` を有効にして、デフォルトパラメータを変更することができます。また、サーバがシングルプロセスモードで稼動している場合は、特定のコンテキストで `IWSSessionManager` を有効にすることができます。iPlanet Web Server で `IWSSessionManager` を使用できるようにするには、以下のいずれかの方法を実行してください。

- ディレクトリ *server_id*`/config` に格納されているファイル `web-apps.xml` を編集する

  以下の例のように、サーブレットまたは JSP の `web-app` 要素内に `session-manager` 要素を追加します。

```
<session-manager
       class='com.iplanet.server.http.session.IWSSessionManager'>
       <init-param>
              <param-name>maxSessions</param-name>
              <param-value>1000</param-value>
       </init-param>
       <init-param>
              <param-name>timeOut</param-name>
              <param-value>1800</param-value>
       </init-param>
       <init-param>
```

```
                <param-name>reapInterval</param-name>
                <param-value>600</param-value>
        </init-param>
        <init-param>
                <param-name>session-data-dir</param-name>
                <param-value>/net/dotcom.com/sessions</param-value>
        </init-param>
</session-manager>
```

web-apps.xml ファイルの詳細は、第 2 章「Web アプリケーション」を参照してください。

- サーバマネージャインタフェースの「Legacy Servlets」>「Configure Global Servlet Attributes」ページを使用する

  「Session Manager」フィールドで、以下のように指定します。

  ```
  com.iplanet.server.http.session.IWSSessionManager
  ```

  「Session Manager Args」フィールドにセッションマネージャのパラメータを指定することもできます。以下に例を示します。

  ```
  maxSessions=20,session-data-dir=/net/dotcom.com/sessions
  ```

IWSSessionManager のソースコード

IWSSessionManager は、各セッションに対して IWSHttpSession オブジェクトを作成します。IWSSessionManager.java および IWSHttpSession.java のソースファイルは、*server_root*/plugins/servlets/iws-apis/sessions ディレクトリに格納されています。IWSSessionManager.java および IWSHttpSession.java のソースコードファイルは、ユーザが独自のセッションマネージャおよびセッションオブジェクトを定義するときに、取り掛かりとして利用できます。

IWSSessionManager は、IWSHttpSessionManager を拡張します。IWSHttpSessionManager のクラスファイルは、ディレクトリ *server_root*/bin/https/jar にある JAR ファイル NSServletLayer.jar に格納されています。IWSSessionManager は、IWSHttpSessionManager 内の実装する必要のあるすべてのメソッドを実装しています。そのため、IWSHttpSessionManager の拡張方法の例として IWSSessionManager を使用することができます。IWSSessionManager または IWSHttpSessionManager のサブクラスをコンパイルするときは、必ず JAR ファイル NSServletLayer.jar がコンパイラのクラスパスにあることを確認してください。

JdbcStore.java および FileStore.java ソースファイルと、その親クラスのソースファイル SessionDataStore.java は、ユーザが IWSSessionManager のセッション持続メカニズムを変更できるようにするために用意されています。これらのファイルも、ディレクトリ *server_root*/plugins/servlets/iws-apis/sessions に格納されています。

MMapSessionManager (UNIX のみ)

これは、シングルモードとマルチモードの両方で稼動する、メモリマップ (mmap) ファイルベースの持続的なセッションマネージャです。

magnus.conf ファイルの MaxProcs 指令によって、サーバがシングルプロセスモードとマルチプロセスモードのどちらで稼動しているかを判別できます。詳しくは、iPlanet Web Server の『NSAPI プログラマーズガイド』を参照してください。

パラメータ

MMapSessionManager では、以下のパラメータが使用されます。

- maxSessions - セッションマネージャが一度に管理できるセッションの最大数。セッションマネージャは、すでにセッション数が maxSessions になっている場合、それ以上新規セッションは作成しません。デフォルト値は 1000 です。
- maxValuesPerSession - セッションが保持できる値またはオブジェクトの最大数。デフォルト値は 10 です。
- maxValueSize - セッションが保持できる各値またはオブジェクトの最大サイズ。デフォルト値は 4096 です。
- timeOut - クライアントが最後にセッションにアクセスしてからセッションマネージャがそのセッションを無効にするまでの経過時間 (秒単位)。timeOut 秒の間アクセスされなかったセッションは、reaper メソッドによって無効にされます。デフォルト値は 1800 (30 分) です。

  web.xml で session-timeout が指定されている場合は、timeOut パラメータ値は無効になります。詳しくは、16 ページの「session-timeout」を参照してください。
- reapInterval - reaper メソッドが再び呼び出されるまでの SessionReaper スレッドのスリープ時間 (秒単位)。デフォルト値は 600 (10 分) です。
- maxLocks - 複数のプロセスにわたる個別のセッションへのアクセスを同期させるために使用する、クロスプロセスロックの数。デフォルト値は 1 です。このデフォルト値は、値に 0 が指定された場合に使用されます。このパラメータは、シングルプロセスモードでは無視されます。

MMapSessionManager を有効にする

MMapSessionManager を有効にして、デフォルトパラメータを変更することができます。また、サーバがシングルプロセスモードで稼動している場合は、特定のコンテキストで MMapSessionManager を有効にすることができます。iPlanet Web Server で MMapSessionManager を使用するには、以下のいずれかの方法を実行してください。

- サーバマネージャインタフェースの「Legacy Servlets」>「Configure Global Servlet Attributes」ページを使用する

  「Session Manager」フィールドで、以下のように指定します。

  ```
  com.iplanet.server.http.session.MMapSessionManager
  ```

  「Session Manager Args」フィールドにセッションマネージャのパラメータを指定することもできます。以下に例を示します。

  ```
  maxSessions=20,maxValueSize=1024,timeOut=300
  ```

- ディレクトリ *server_id*/config に格納されているファイル servlets.properties を編集する。この方法は、デフォルトの仮想サーバにのみ適用できます。

  servlets.sessionmgr の値を指定する行を追加します。また、セッションマネージャのパラメータを指定する行も追加します。

```
servlets.sessionmgr=com.iplanet.server.http.session.MMapSessionMana
ger
servlets.sessionmgr.initArgs=maxSessions=20,maxValueSize=1024,timeO
ut=300
```

- ディレクトリ *server_id*/config に格納されているファイル contexts.properties を編集する。この方法は、デフォルトの仮想サーバにのみ適用できます。

  context.*context_name*.sessionmgr の値を指定する行を追加します。また、セッションマネージャのパラメータを指定する行も追加します。

```
context.global.sessionmgr=com.iplanet.server.http.session.MMapSessi
onManager
context.global.sessionmgr.initArgs=maxSessions=20,maxValueSize=1024
,timeOut=300
```

  グローバルコンテキストを変更したり、新しいコンテキストを定義してそれにサーブレットを割り当てたりすることもできます。詳しくは、第 8 章「古いバージョンのサーブレットおよび JSP の構成」を参照してください。

このセッションマネージャは、java.io.Serializable を実装するオブジェクトのみ格納できます。

SessionData のバージョンファイルの削除

サーバが MMapSessionManager セッションマネージャを使用している場合は、SessionData ディレクトリに持続セッション情報が格納されています。このキャッシュには、バージョン番号が記述されている Version ファイルが格納されており、サーバはこのファイルを使用して、ディレクトリの構造とキャッシュ内のファイルを判別します。バージョンファイルを削除するだけで、キャッシュをクリーンアップすることができます。

サーバは、起動時にバージョンファイルを検出できない場合、該当するキャッシュのディレクトリ構造を削除し、そのバージョンファイルを再作成します。次回サーバが MMapSessionManager セッションマネージャを使用してサーブレットを実行するときに、セッションデータキャッシュが再作成されます。

バージョンファイルは、通常ファイルを削除するときと同じように SessionData ディレクトリから削除するだけで、削除できます。または、サーバマネージャの「Java」>「Delete Version Files」ページで削除することもできます。バージョンファイルを削除した後は、必ず iPlanet Web Server を再起動してください。そうすることにより、該当するキャッシュがクリーンアップされ、サーバがサーブレットを実行する前にバージョンファイルが再作成されます。

非推奨のセッションマネージャ

SimpleSessionManager および JdbcSessionManager は、iPlanet Web Server 4.*x* との下位互換性を提供するために用意されています。

注 次の非推奨のセッションマネージャの親クラスも非推奨であり、下位互換性を提供するためにのみ用意されています。

```
com.netscape.server.http.session.NSHttpSessionManager
```

代わりに、次のクラスを拡張してください。

```
com.iplanet.server.http.session.IWSHttpSessionManager
```

SimpleSessionManager

SimpleSessionManager は、シングルプロセスモードでのみ動作します。セッションは持続的ではなく、すべてのセッションはサーバが停止したときに失われます。

注	SimpleSessionManager は非推奨であり、下位互換性を提供するためにのみ用意されています。代わりに、セッションの持続性を持たない IWSSessionManager を使用してください。

パラメータ

SimpleSessionManager クラスでは、以下のパラメータが使用されます。

- maxSessions - セッションマネージャが一度に管理できるセッションの最大数。セッションマネージャは、すでにセッション数が maxSessions になっている場合、それ以上新規セッションは作成しません。デフォルト値は 1000 です。
- timeOut - クライアントがセッションにアクセスしてからセッションマネージャがそのセッションを無効にするまでの経過時間 (秒単位)。timeOut 秒の間アクセスされなかったセッションは、reaper メソッドによって無効にされます。デフォルト値は 1800 (30 分) です。

  web.xml で session-timeout が指定されている場合は、この timeOut パラメータ値は無効になります。詳しくは、16 ページの「session-timeout」を参照してください。
- reapInterval - reaper メソッドが再び呼び出されるまでの SessionReaper スレッドのスリープ時間 (秒単位)。デフォルト値は 600 (10 分) です。

SimpleSessionManager を有効にする

SimpleSessionManager を有効にして、デフォルトパラメータを変更することができます。また、サーバがマルチプロセスモードで稼動している場合は、特定のコンテキストで SimpleSessionManager を有効にすることができます。iPlanet Web Server で SimpleSessionManager を使用するには、以下のいずれかの方法を実行してください。

- サーバマネージャインタフェースの「Legacy Servlets」>「Configure Global Servlet Attributes」ページを使用する

  「Session Manager」フィールドで、以下のように指定します。

  ```
  com.netscape.server.http.session.SimpleSessionManager
  ```

  「Session Manager Args」フィールドにセッションマネージャのパラメータを指定することもできます。以下に例を示します。

  ```
  maxSessions=20,timeOut=300,reapInterval=150
  ```
- ディレクトリ *server_id*/config に格納されているファイル servlets.properties を編集する。この方法は、デフォルトの仮想サーバにのみ適用できます。

servlets.sessionmgr の値を指定する行を追加します。また、セッションマネージャのパラメータを指定する行も追加します。

```
servlets.sessionmgr=com.netscape.server.http.session.SimpleSessionM
anager
servlets.sessionmgr.initArgs=maxSessions=20,timeOut=300,reapInterva
l=150
```

- ディレクトリ *server_id*/config に格納されているファイル contexts.properties を編集する。この方法は、デフォルトの仮想サーバにのみ適用できます。

  context.*context_name*.sessionmgr の値を指定する行を追加します。また、セッションマネージャのパラメータを指定する行も追加します。

```
context.global.sessionmgr=com.netscape.server.http.session.SimpleSe
ssionManager
context.global.sessionmgr.initArgs=maxSessions=20,timeOut=300,reapI
nterval=150
```

  グローバルコンテキストを変更したり、新しいコンテキストを定義してそれにサーブレットを割り当てたりすることもできます。詳しくは、第 8 章「古いバージョンのサーブレットおよび JSP の構成」を参照してください。

JdbcSessionManager

これは、シングルモードとマルチモードの両方で動作する、JDBC ベースの持続的なセッションマネージャです。このセッションマネージャを使用して、カスタムデータベースにセッションを格納できます。このカスタムベースは、異なるマシンで実行されている複数のプロセスで共有されます。

注 JDBCSessionManager は非推奨であり、下位互換性を提供するためにのみ用意されています。代わりに、JdbcStore セッションの持続性を持つ IWSSessionManager を使用してください。

このサンプル JDBC セッションマネージャは、本稼動用に作成またはテストされているのではありません。目的に合わせて動作をカスタマイズするために提供されています。

JdbcSessionManager は、標準 JDBC-ODBC ドライバで Microsoft SQL Server 7.0SP1 に関してテスト済みです。セッションマネージャが使用する ODBC ソース、データベース、およびテーブルを設定する必要があります。検索のパフォーマンスを向上させるため、「Session ID」列にインデックスを生成することをお勧めします。

パラメータ

JdbcSessionManager では、以下のパラメータが使用されます。

- timeOut - クライアントがセッションにアクセスしてからセッションマネージャがそのセッションを無効にするまでの経過時間 (秒単位)。timeOut 秒の間アクセスされなかったセッションは、reaper メソッドによって無効になります。デフォルト値は 1800 (30 分) です。

  web.xml で session-timeout が指定されている場合は、この timeOut パラメータ値は無効になります。詳しくは、16 ページの「session-timeout」を参照してください。

- provider - JDBC ドライバ (デフォルトは sun.jdbc.odbc.JdbcOdbcDriver)。JDBC API の詳細は、以下の Web サイトを参照してください。

  http://java.sun.com/products/jdbc/index.html

注 JdbcStore クラスは、web-apps.xml の class-loader 要素の classpath 属性に割り当てられた JDBC ドライバクラスを認識しません。JDBC ドライバクラスを jvm12.conf ファイルの jvm.classpath 変数に割り当ててください。詳しくは、付録 C「JVM の構成」を参照してください。

- url - データソース (デフォルトは jdbc:odbc:LocalServer)
- table - セッションを格納する SQL テーブルの名前 (デフォルトは sessions)
- username - データベースのログインユーザ名
- password - データベースのログインパスワード
- reaperActive - true に設定された場合 (デフォルト値)、セッション reaper を実行して期限切れのセッションをデータベースから削除するよう、セッションマネージャに指示します。reaper を実行しているサーバがクラスタ内で 1 つのみになるようにしてください。
- accessTimeColumn - 最後にアクセスされた時間 (分単位) を格納する列の名前 (デフォルトは AccessTime)。SQL 型は NUMERIC(9) です。
- sessionIdColumn - セッション ID を格納する列の名前 (デフォルトは SessionID)。SQL 型は VARCHAR(100) です。
- valueColumn - セッションオブジェクトを格納する列の名前 (デフォルトは Value)。SQL 型は VARBINARY(4096) です。この列は、すべてのセッションデータを格納できるくらいの大きさである必要があります。

セッション情報を処理するためのデータベース上での各操作 (検索、挿入、更新、および削除) は、対応する専用の接続によって実行されます。これらの接続はそれぞれ、パフォーマンスを向上させるために、コンパイル済みの SQL 文を持っています。以下のパラメータを使用して、各操作専用の接続の数をカスタマイズできます。

- `lookupPool` - 検索操作を行う接続の数 (デフォルトは 4)
- `insertPool` - 挿入操作を行う接続の数 (デフォルトは 4)
- `updatePool` - 更新操作を行う接続の数 (デフォルトは 4)
- `deletePool` - 削除操作を行う接続の数 (デフォルトは 2)

JdbcSessionManager を有効にする

`JdbcSessionManager` を有効にして、デフォルトパラメータを変更することができます。また、サーバがシングルプロセスモードで稼動している場合は、特定のコンテキストで `JdbcSessionManager` を有効にすることができます。iPlanet Web Server で `JdbcSessionManager` を使用するには、以下のいずれかの方法を実行してください。

- サーバマネージャインタフェースの「Legacy Servlets」>「Configure Global Servlet Attributes」ページを使用する

  「Session Manager」フィールドで、以下のように指定します。

  ```
  com.netscape.server.http.session.JdbcSessionManager
  ```

  「Session Manager Args」フィールドにセッションマネージャのパラメータを指定することもできます。以下に例を示します。

  ```
  timeOut=1200,username=mysession,password=mypassword
  ```

- ディレクトリ *server_id*`/config` に格納されているファイル `servlets.properties` を編集する。この方法は、デフォルトの仮想サーバにのみ適用できます。

  `servlets.sessionmgr` の値を指定する行を追加します。また、セッションマネージャのパラメータを指定する行も追加します。

```
servlets.sessionmgr=com.netscape.server.http.session.JdbcSessionMan
ager
servlets.sessionmgr.initArgs=timeOut=1200,username=mysession,passwo
rd=mypassword
```

- ディレクトリ *server_id*`/config` に格納されているファイル `contexts.properties` を編集する。この方法は、デフォルトの仮想サーバにのみ適用できます。

  `context.`*context_name*`.sessionmgr` の値を指定する行を追加します。また、セッションマネージャのパラメータを指定する行も追加します。

```
context.global.sessionmgr=com.netscape.server.http.session.JdbcSess
ionManager
context.global.sessionmgr.initArgs=timeOut=1200,username=mysession,
password=mypassword
```

グローバルコンテキストを変更したり、新しいコンテキストを定義してそれにサーブレットを割り当てたりすることもできます。詳しくは、第 8 章「古いバージョンのサーブレットおよび JSP の構成」を参照してください。

このセッションマネージャは、`java.io.Serializable` を実装するオブジェクトのみ格納できます。

ロードバランシング、セッションフェイルオーバー、およびセッション ID

iPlanet Web Server 6.0 では、セッション対応の既存のフロントエンドロードバランサ (Resonate など) を使用したサーバファーム (またはクラスタ) 構成がサポートされています。iPlanet Web Server 6.0 は、セッションを生成したサーバホストの *node_id* を JSESSIONID に接頭辞として付けることにより、スティッキーセッションを実装します。JSESSIONID は、Servlets 2.2 仕様で指定されたセッション cookie 名で、以下のように指定されます。

```
Set-Cookie: JSESSIONID=node_id-3ad%253A39c02099%253Ad19e53e2a2;
path=/app;expires=Thu, 14-Sep-2000 01:19:30 GMT
```

これにより、フロントエンドロードバランサは、そのセッションを生成した同じホストに今後の要求を転送できるようになります。

クラスタによって使用されているセッションマネージャが分散セッションをサポートしている場合、障害が発生したサーバが作成したセッションを別のサーバが処理することができます。`IWSSessionManager` では、`session-data-store` パラメータ値でセッションの持続性が定義されている場合、分散セッションがサポートされます。`JdbcSessionManager` も分散セッションをサポートしています。

注 サーブレットセッションに使用されるセッション ID ジェネレータでは、暗号化機能が強化された、固有の乱数生成アルゴリズムが採用されています。これにより、処理速度の遅い古いマシンを使用している場合は、パフォーマンスに関する問題が発生することがあります。セッションマネージャ API では、ランダム ID の生成方法を再定義し、目的に応じてカスタマイズすることができます
(62 ページの「IWSSessionManager のソースコード」で説明されているサンプルファイル `IWSSessionManager.java` を参照)。

注 iPlanet Web Server のセッションマネージャでは、最大 108 文字 (ASCII) のセッション ID を生成できます。

第 7 章

API に関するヒント

この章では、Servlet 2.2 API 仕様が iPlanet Web Server 6.0 で実装される方法を説明します。また、以下の節から構成されています。

- HttpSession スコープについて
- メソッドの使用方法について
- その他の関連情報

HttpSession スコープについて

Servlet 2.2 API 仕様は、HttpSession オブジェクトのスコープに関して流動的です。

デフォルトでは、iPlanet Web Server は、コンテキストパスまたはアプリケーションの contextPath への cookie のパスを追跡して、セッションにマークを付けます。これにより、ブラウザで、関係のないアプリケーションに対してセッション cookie が再実行されることがなくなります。したがって、HttpSession オブジェクトが適切にスコープ指定されます。

しかし、あるコンテキスト内の サーブレット A が異なるコンテキスト内の サーブレット B に要求をディスパッチすると、仕様は流動的になります。デフォルトでは、iPlanet Web Server ではこのディスパッチが許可されているため、HttpSession オブジェクトの属性は、関連アプリケーションに共通するクラスローダーによって読み込まれるようにしてください。通常はお勧めできませんが、仮想サーバレベルのクラスローダーやシステムクラスローダーを使用することもできます。

異なるアプリケーションでセッションおよびセッション属性を共有するには次の項目に従います。

- 仮想サーバレベルで共通のセッションマネージャを使用する

```
<vs>
       <session-manager class="..." />
</vs>
```

- セッション属性が仮想サーバレベルのクラスローダーによって読み込まれるようにし、クラスパスにセッション属性として使用する共通 Bean などを追加する

```
<vs>
       <class-loader classpath="/myapps/sessionattrs.jar" />
</vs>
```

この設定により、仮想サーバ内のすべてのアプリケーションでセッションを共有できるようになります。この設定を行わない場合は、すべてのアプリケーションでセッションは共有されません。iPlanet Web Server 6.0 では、一部のアプリケーションでのみ同じセッションデータを共有するような設定はサポートされていません。

メソッドの使用方法について

この節では、iPlanet Web Server 6.0 で以下の Servlet 2.2 API メソッドを使用する方法について説明します。

- HttpServlet.service
- ServletContext.getAttribute
- ServletRequest.setAttribute
- ServletRequest.getParameter
- ServletResponse.getOutputStream および getWriter
- RequestDispatcher.forward および include

ここで説明されているメソッドおよびすべての Servlet API メソッドに関する公式マニュアルは、以下の Web サイトにある Sun Microsystems の Servlets API Javadoc を参照してください。

`http://java.sun.com/products/servlet/2.2/javadoc/index.html`

HttpServlet.service

```
public void service(ServletRequest req, ServletResponse res) throws
ServletException, java.io.IOException
```

このメソッドは、限定公開の `service` メソッドにクライアント要求をディスパッチします。

ヒント

サーブレットは、サービスロジックを処理するための、追加スレッドを作成することがあります。ただし、これらのオブジェクトによって呼び出された API 関数は、要求を処理するスレッドか、サーブレットが作成したスレッドのどちらか 1 つによってアクセスされている必要があります。これら両方のスレッドが同時にその関数にアクセスすることはできません。

ServletContext.getAttribute

```
public java.lang.Object getAttribute(java.lang.String name)
```

指定した名前のサーブレット コンテナ属性を返します。その名前の属性がない場合は、`null` を返します。

ヒント

コンテキストクラスローダー (`java.lang.ClassLoader` オブジェクト) を取得するには、`com.iplanet.server.http.servlet.classloader` 属性を使用します。コンテキストクラスローダーのクラスパス (`java.lang.String` オブジェクト) を取得するには、`com.iplanet.server.http.servlet.classpath` 属性を使用します。コンテキストクラスローダーの再読み込み間隔 (`java.lang.Integer` オブジェクト) を取得するには、`com.iplanet.server.http.servlet.reload-interval` 属性を使用します。

ServletRequest.setAttribute

```
public void setAttribute(java.lang.String name, java.lang.Object o)
```

現在の要求に属性を格納します。属性は、要求間で設定し直されます。このメソッドは、多くの場合、`RequestDispatcher` と共に使用されます。

ヒント

要求オブジェクトに `com.iplanet.server.http.servlet.parameterEncoding` 属性を設定すると、`getParameter` メソッドに、取り出すパラメータの符号化を認識させることができます。

ServletRequest.getParameter

```
public java.lang.String getParameter(java.lang.String name)
```

パラメータ名に関連付けられている値を取得します。

ヒント

フォームフィールドに UTF-8 以外の文字がある場合は、以下のいずれかの操作を行う必要があります。そうしない場合、getParameter メソッドで取得される値はゼロまたは未定義になります。

- web-apps.xml ファイルで、parameter-encoding 要素の enc 属性を auto (デフォルト) に設定する。詳しくは、22 ページの「parameter-encoding」を参照してください。
- iPlanet Web Server 4.*x* のときと同様、contexts.properties ファイルで、parameterEncoding プロパティを auto (デフォルト) または responseCT に設定する。詳しくは、94 ページの「parameterEncoding」を参照してください。

フォームフィールドにデータを入力するときに使用される元の符号化は、そのデータが URL に符号化されると失われるため、以下の操作を行う必要があります。

- クライアントにフォームを送信するときには、必ず応答コンテンツの種類を設定する。これにより、フォーム全体がクライントに正常に送信されます。
- フォームフィールドと異なるロケールを使用するサーバにフォームデータを送信するときには、getParameter メソッドを呼び出す前に、以下のように charset を指定する
  - フォームを生成するサーブレットまたは JSP と、フォームを処理するサーブレットまたは JSP が異なる場合は、フォームの隠しフィールド (デフォルトでは j_encoding と呼ばれる) を使用します。以下に例を示します。

    ```
    <input type="hidden" name="j_encoding" value="US_ASCII">
    ```

  - 要求オブジェクトに com.iplanet.server.http.servlet.parameterEncoding 属性を設定します (75 ページの「ServletRequest.setAttribute」を参照)。この属性は、パラメータを復号化するために getParameter メソッドによって使用されます。
  - contexts.properties で parameterEncoding=responseCT に設定されていて、同じサーブレットまたは JSP がフォームの生成および処理を両方行なっている場合は、応答コンテンツの種類を設定することができます。サーブレットの場合は、以下の例のように、応答コンテンツの種類を明示的に設定します。

    ```
    res.setContentType("text/plain; charset=Shift_JIS");
    ```

    JSP の場合は、以下の例のように、page 指令を使用して応答コンテンツの種類を設定します。

    ```
    <%@ page contentType="text/html; charset=gb2312"%>
    ```

ServletResponse.getOutputStream および getWriter

```
public ServletOutputStream getOutputStream() throws
java.io.IOException
```

バイナリデータを応答に書き込むために適している ServletOutputStream を返します。サーブレット コンテナは、バイナリデータを符号化しません。本文を書く際には、このメソッドまたは getWriter のどちらか一方を呼び出します。両方を呼び出す必要はありません。

```
public java.io.PrintWriter getWriter() throws java.io.IOException
```

文字テキストをクライアントに送信するための PrintWriter オブジェクトを返します。文字の符号化方式は、setContentType(java.lang.String) メソッドの charset= プロパティで指定したものが使用されます。上記のメソッドを呼び出す前にこのメソッドを呼び出さないと、Charset が有効になりません。

本文を書く際には、このメソッドまたは getOutputStream のどちらか一方を呼び出します。両方を呼び出す必要はありません。

ヒント

仕様では、getOutputStream メソッドがすでに呼び出されている場合にサーブレットが応答で getWriter メソッドを呼び出すとき (またはその逆) は、サーブレット コンテナが IllegalStateException をスローすることが推奨されています。

iPlanet Web Server 6.0 では、例外はスローされません。ただし、writer および出力ストリームへの書き込みの順序付けが、必ず行われます。このような実装条件は厳しくありませんが、正確さは保持されるため、以下のようなシナリオが可能です。

- サーブレットが getOutputStream を呼び出して例外をスローし、JSP が例外ページを表示するために使用される。仕様に厳密に準拠すれば、このシナリオは失敗します。なぜなら、JSP が応答オブジェクトで getWriter を呼び出そうとするからです。
- getWriter を呼び出す JSP に、getOutputStream を呼び出すサーブレットが指定されている

RequestDispatcher.forward および include

```
public void forward(ServletRequest request, ServletResponse
response) throws ServletException, IOException;
```

このメソッドはサーブレットから、Web サーバ上の別のリソースに要求を転送するときに使用します。このメソッドは、サーブレットが要求の事前処理を行い、別のオブジェクトに応答を生成させる場合に便利です。

ターゲットオブジェクトに受け渡された要求オブジェクトは、その要求 URL パスとほかのパスパラメータを、ターゲットオブジェクトのターゲット URL パスを反映するように調整します。

このメソッドは、応答で `ServletOutputStream` オブジェクトまたは `PrintWriter` オブジェクトが取得されている場合は使用できません。その場合、メソッドは `IllegalStateException` をスローします。

```
public void include(ServletRequest request, ServletResponse
response) throws ServletException, IOException;
```

別のサーバリソースによって生成されたコンテンツを応答の本体にインクルードするときに使用します。本質的に、このメソッドは、プログラムをサーバ側でインクルードすることを可能にするものです。ターゲットオブジェクトに受け渡された要求オブジェクトは、その要求 URL パスと要求の呼び出し元のパス情報を反映します。応答オブジェクトは、呼び出し元のサーブレットの `ServletOutputStream` オブジェクトまたは `PrintWriter` オブジェクトにのみアクセスできます。

インクルードされているサーブレットは、ヘッダーを設定できません。このサーブレットがヘッダー (Cookies など) を設定する必要があるメソッドを呼び出しても、うまく機能しません。サーブレット開発者は、ヘッダーに直接アクセスする必要のあるメソッドが適切に処理されるようにする必要があります。セッションを正常に動作させるには、セッショントラッキングを行う場合であっても、インクルードされているサーブレットの外でセッションを開始するようにしてください。

ヒント

iPlanet Web Server 6.0 では、`Writer` または `OutputStream` が取得されている場合は、`dispatcher.forward` メソッドが `IllegalStateException` をスローするときとスローしないときがあります。この動作は、ドラフト 2.2 に従っており、JSP エラーページの処理をするために必要です。実際のデータがフラッシュされ、クライアントに送信された場合にのみ、例外がスローされます。それ以外の場合は、バッファにある保留データがただ破棄されるだけです。

ターゲット URI が安全でない URI であると識別された場合 (つまり、URI の最後が //、/./、/../、/.、/.. (Windows NT の場合 ./) など、セキュリティ保護されていないパス文字の場合)、forward および include メソッドは、ServletException をスローすることがあります。

magnus.conf の requestDispatcherNestDepth パラメータを使用して、RequestDispatcher.forward および include メソッドの入れ子の深さを制御できます。詳しくは、付録 A を参照してください。

その他の関連情報

この節では、以下の項目に関する情報を説明します。

- データベース接続プール
- クライアント証明書の取得

データベース接続プール

データベース接続プールによって、サーブレットまたは JSP のデータベース操作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 互換ドライバには、Oracle 8i アップデート版や CloudScape 3.0 など、さまざまなものがあります。

クライアント証明書の取得

SSL を有効にしたためにクライアント証明書が必要になった場合、サーブレットは以下のようにクライアント証明書にアクセスします。

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

userDn は、ユーザの完全修飾識別名です。

第 8 章

古いバージョンのサーブレットおよび JSP の構成

この章では、以下の節に分けて、iPlanet Web Server 4.*x* との下位互換用として提供されている古いバージョンの構成手順および構成ファイルについて説明します。

- デフォルトの仮想サーバ
- サーブレットおよび JSP の有効化
- クライアントで JSP を使用可能にする
- デフォルトの仮想サーバでのサーブレットの構成
- ユーザインタフェースの使用
- 古いバージョンの構成ファイル
- 古いバージョンの例
- 古いバージョンのサーブレットのパフォーマンスを最大限にする

注 この章で説明されているように、古いバージョンのサーブレットや JSP は非推奨です。第 2 章「Web アプリケーション」で説明しているように、Web アプリケーションを作成および導入することをお勧めします。

デフォルトの仮想サーバ

デフォルトの仮想サーバは、古いバージョンのアプリケーションを実行できる唯一の仮想サーバです。デフォルトの仮想サーバは、iPlanet Web Server が初めてインストールされるときに存在している唯一の仮想サーバです。デフォルトの仮想サーバの設定方法については、iPlanet Web Server の『NSAPI プログラマーズガイド』の第 8 章「仮想サーバの構成ファイル」を参照してください。

サーブレットおよび JSP の有効化

サーブレットや JSP を有効または無効にするには、サーバマネージャインタフェースの「Java>Enable/Disable Servlets/JSP」ページを使用してください。

サーブレットが有効か無効かにかかわらず、デフォルトでは、ファイル `obj.conf` には `servlet`、`jsp`、`ServletByExt` のような名前を持つオブジェクトが指定されます。これらのオブジェクトを削除しないでください。これらのオブジェクトを削除すると、サーバマネージャでサーブレットを起動できなくなります。

クライアントで JSP を使用可能にする

サーブレットおよび JSP が iPlanet Web Server で有効になっていれば、クライアントで JSP ページを使用可能にするために特別なことをする必要はありません。JSP が有効になっている限り、iPlanet Web Server は、`.jsp` 拡張子を持つすべてのファイルを JSP として扱います。(JSP ファイルを登録されているサーブレットディレクトリに置かないでください。iPlanet Web Server は、登録されているサーブレットディレクトリ内のすべてのファイルをサーブレットとみなします。)

注	iPlanet Web Server の別名ディレクトリで JSP を実行することはできません。たとえば、ドキュメントルートが *server_root*`/docs` の場合、`http://foo.com/myjsp` を *server_root*`/docs/myjsp` ではなく `/some/other/dir` にマッピングすると、動作しなくなります。

デフォルトの仮想サーバでのサーブレットの構成

デフォルトの仮想サーバのクライアントからサーブレットにアクセスできるようにするための方法には 3 つあります。2 番目と 3 番目の方法は、iPlanet Web Server 4.*x* との下位互換用として提供されています。

- ほかのすべての仮想サーバと同じように、Web アプリケーションにサーブレットを組み込み、その Web アプリケーションを導入します。この方法は、第 2 章「Web アプリケーション」で説明しています。これは、iPlanet Web Server 6.0 の新しい機能です。
- サーブレットディレクトリとして iPlanet Web Server に登録済みのディレクトリの 1 つに、サーブレットクラスファイルを置きます。詳細は、83 ページの「サーブレットディレクトリの登録」を参照してください。

- サーブレット用にサーブレット仮想パスを定義します。この場合、サーブレットクラスは、ファイルシステムのどこにでも置くことができます。リモートマシンに置くこともできます。詳細は、86 ページの「サーブレット仮想パスの指定」を参照してください。

デフォルトの仮想サーバのサーブレットを iPlanet Web Server 4.*x* のときと同じように使うには、以下の手順に従います。

1. グローバルサーブレットの属性設定
2. サーブレットディレクトリの登録
3. 必要に応じて、サーブレットの個別登録
4. 必要に応じて、サーブレット仮想パスの指定
5. 必要に応じて、サーブレットのコンテキストの指定

グローバルサーブレットの属性設定

以下のサーブレット属性をオプションで指定できます。

- 起動サーブレット - iPlanet Web Server の起動時に読み込まれるサーブレット
- セッションマネージャ - サーブレット用のセッションマネージャ。セッションマネージャの詳細は、第 6 章「セッションマネージャ」を参照してください。
- セッションマネージャ引数 - サーブレットエンジン用のセッションマネージャの引数。セッションマネージャの詳細は、第 6 章「セッションマネージャ」を参照してください。
- 再読み込み間隔 - サーブレットや JSP をサーバ上で変更したときに、読み込みし直すまでのサーバの待ち時間。デフォルト値は 5 秒です。

これらの属性は、サーバマネージャインタフェースの「Legacy Servlets>Configure Global Servlet Attributes」ページで対話式に設定できます。または、サーバの `config` ディレクトリ内の構成ファイル `servlets.properties` および `contexts.properties` を編集することもできます。

サーブレットディレクトリの登録

クライアントがサーブレットにアクセスできるようにする 1 つの方法は、サーブレットディレクトリとして iPlanet Web Server に登録されているディレクトリにサーブレットを置くことです。登録済みのサーブレットディレクトリ内のサーブレットは、必要なときに動的に読み込まれます。サーバは、サーブレットファイルを監視し、ファイルに変更が加えられたら自動的にそれらを再読み込みします。

iPlanet Web Server のサーブレットディレクトリの登録数に上限はありません。iPlanet Web Server は、最初 1 つのサーブレットディレクトリ (*server_root*/docs/servlet/) を持っています。

たとえば、SimpleServlet.class のサーブレットが、サーバのドキュメントルートディレクトリ (デフォルトのサーブレットディレクトリ) の servlet サブディレクトリ内にある場合、Web ブラウザに次のように指定してサーブレットを起動できます。

http://*your_server*/servlet/SimpleServlet

iPlanet Web Server は、登録済みサーブレットディレクトリ内のすべてのファイルがサーブレットであるとみなします。サーバは、.class 拡張子を持つディレクトリ内のすべてのファイルをサーブレットとして扱います。iPlanet Web Server は、サーブレットディレクトリに置かれている HTML ファイルや JSP などのその他のファイルは正しく処理できません。

サーバは、複数のサーブレットディレクトリを持つことができます。必要に応じて、サーブレットディレクトリを仮想ディレクトリにマッピングすることができます。たとえば、http://poppy.my_domain.com/products/ が、ディレクトリ *server_root*/docs/january/products/servlets/ 内のサーブレットを起動するように指定できます。

サーブレットディレクトリを登録し、URL 接頭辞を指定するには、サーバマネージャインタフェースの「Legacy Servlets>Servlet Directory」ページを使用してください。

または、ファイル obj.conf でデフォルトのオブジェクトに次のような適切な NameTrans 指令を追加することによって、サーブレットディレクトリを登録することもできます。

```
NameTrans fn="pfx2dir" from="/products"
dir="d:/netscape/server4/docs/january/products/servlets/"
name="ServletByExt"
```

登録済みサーブレットディレクトリからのパスを指定して package 指令をサーブレットコード内に記述した場合は、登録済みサーブレットディレクトリのサブディレクトリ内にあるサーブレットを起動できます。たとえば、サーブレットが次の場所にあり、*server_root*/docs/servlet/ が登録済みサーブレットディレクトリであると仮定します。

server_root/docs/servlet/HelloWorld/HelloWorldServlet.class

Java ソースファイルの最初の行に、次の package 指令を指定します。

```
package HelloWorld;
```

この後、Web ブラウザで次のように指定すると、サーブレットを起動できるようになります。

`http://`*your_server*`/servlet/SimpleServlet`

パッケージ化されているサーブレットの再読み込みの詳細は、94 ページの「isModifiedCheckAggressive」を参照してください。

サーブレットの個別登録

iPlanet Web Server は、登録済みのサーブレットディレクトリ内のすべてのファイルをサーブレットとして扱います。以下の条件にあてはまらない限り、ディレクトリ内に置かれているサーブレットを個別に登録する必要はありません。

- サーブレットが、要求 URL を介して渡されていない入力パラメータを使用している
- サーブレット用の仮想 URL を追加して設定する場合
- サーブレットが、パッケージ化されている、または `.jar` ファイル内にある。サーバは、パッケージ化されたサーブレット用の `.class` ファイルまたは `.jar` ファイルは検索しません。

これらの条件のどれかにあてはまる場合には、サーバマネージャインタフェースの「Legacy Servlets>Configure Servlet Attributes」ページを使って、個別にサーブレットを登録してください。または、ファイル `servlets.properties` を編集してサーブレット用のエントリを追加することもできます。

個別にサーブレットを登録する場合、次の属性を指定します。

- サーブレット名 - iPlanet Web Server はこの値をサーブレット識別子として使って、サーブレットを内部的に識別します。(識別子がクラスコード名と偶然に一致した場合を除き、この識別子はサーブレットの起動に使用されている URL の一部ではありません。)
- サーブレットコード (クラス名) - クラスファイル名。`.class` 拡張子を指定する必要はありません。
- サーブレットクラスパス - サーブレットのあるディレクトリ、または `zip` ファイルや `jar` ファイルへの絶対パス名または URL。クラスパスは、ファイルシステム内の任意の場所を指定できます。サーブレットクラスパスには、ディレクトリ、`.jar` ファイル、`.zip` ファイルや、ディレクトリへの URL が指定できます。(`zip` ファイルや `jar` ファイルのクラスパスとして URL を指定することはできません。)

  サーブレットクラスパスが登録済みサーブレットディレクトリでない場合は、サーブレット仮想パスを追加して、クライアントがサーブレットにアクセスできるようにする必要があります (86 ページの「サーブレット仮想パスの指定」参照)。

iPlanet Web Server では、サーブレットクラスパスにディレクトリ、jar、zip、および URL を複数指定することができます。

- サーブレット引数 - 必要に応じて追加するサーブレット用の引数で、コンマで区切られたリスト

次のコードは、servlets.properties 内の同じサーブレット用の構成情報の例を示します。

```
servlet.BuyNowServlet.classpath=D:/Netscape/server4/docs/servlet
/buy;D:/Netscape/server4/docs/myclasses
servlet.BuyNowServlet.code=BuyNow1A
servlet.BuyNowServlet.initArgs=arg1=45,arg2=online,arg3="quick
shopping
```

必要に応じて、サーブレットクラスパスとして値を複数指定することができます。

サーブレット仮想パスの指定

サーブレットディレクトリにサーブレットを置かずに個別に登録する場合、サーブレット仮想パスを定義する必要があります。たとえば、次の URL

`http://poppy.my_domain.com/plans/plan1`

が、次のディレクトリに定義されているサーブレットを起動するように指定できます。

server_root`/docs/plans/releaseA/planP2Version1A.class`

登録されているサーブレットディレクトリの内外にかかわらず、ファイルシステムのどこに置かれているサーブレットに対しても、サーブレット仮想パスを設定できます。

サーブレット仮想パスを指定するには、サーバマネージャインタフェースの「Legacy Servlets>Configure Servlet Virtual Path Translation」ページを使用します。このページで、仮想パス名およびサーブレット名を指定します。または、rules.properties 構成ファイルを編集して、サーブレット仮想パスを追加することもできます。仮想パスが設定されているサーブレットだけが、初期引数を使用できます。

サーブレット仮想パスを使う前に、インタフェースの「Legacy Servlets>Configure Servlet Attributes」ページ (または servlets.properties 構成ファイル) で、サーブレット用のサーブレット識別子 (サーブレット名) を追加する必要があります。

サーブレット仮想パスの例

論理 URL `http://poppy.my_domain.com/plans/plan1` が、次のように定義されたサーブレットを起動するように指定します。

server_root`/docs/servlet/plans/releaseA/planP2Version1A.class.`

1. サーブレット識別子、クラスファイル、およびクラスパスを指定します。

   インタフェースの「Legacy Servlets>Configure Servlet Attributes」ページで、次のように指定します。

   - 「Servlet Name」フィールドで、`plan1A` のように、サーブレットの識別子を入力します。(クラスファイル名と同じである必要はありません)。
   - 「Servlet Code」フィールドで、`planP2Version1A` のように、クラスファイル名を入力します。ディレクトリは指定しないでください。`.class` 拡張子は必要ありません。
   - 「Servlet Classpath」フィールドで、サーブレットクラスファイルが置かれるディレクトリ、または `jar` ファイルや `zip` ファイルの絶対パス名、またはディレクトリの URL を入力します。この例では、*server_root*`/docs/servlet/plans/releaseA` と入力します。(例 : `D:/netscape/server4/docs/servlet/plans/releaseA`)
   - 「Servlet Args」フィールドで、サーブレットに必要な引数を追加入力します。(この例では追加の引数は使いません。)

   変更を保存します。

   手動でこの変更を行うには、構成ファイル `servlets.properties` に次の行を追加します。

```
/servlet.planA.classpath=D:/netscape/server4/docs/servlet/pla
ns/releaseA/
servlet.plan1A.code=planP2Version1A
```

2. サーブレットの仮想パスを指定します。

   「Legacy Servlets>Configure Servlet Virtual Path Translations」ページで、次のように指定します。

   - 「Virtual Path」フィールドで、仮想パス名を入力します。サーバ名は接頭辞として暗黙指定されるので、この場合は、仮想パス `http://poppy.mcom.com/plans/plan1` を指定するために `/plans/plan1` と入力するだけで済みます。
   - 「Servlet Name」フィールドで、この仮想パスによって起動されるサーブレット用の識別子を入力します。これは、「Configure Servlet Attributes」ページで指定したサーブレット識別子です。この場合は、`plan1A` です。

変更を保存します。

ファイルを編集してこの変更を行うには、rules.properties に次の行を追加します。

```
/plans/plan1=plan1A
```

このサーブレット仮想パスが設定された後は、クライアントが URL http://poppy.my_domain.com/plans/plan1 に対する要求をサーバに送ると、サーバは *server_root*/docs/servlet/plans/releaseA/plan2PVersion1A.class 内にあるサーブレットの呼び出し結果を送り返します。

サーブレットのコンテキストの指定

コンテキストによって、複数のサーブレットがデータを交換したり互いのフィールドにアクセスしたりすることが可能になります。コンテキストは、仮想サーバの定義やコードの識別に役立ちます。コンテキストは、servlets.properties および contexts.properties ファイルで定義します。

ユーザインタフェースの使用

古いバージョンのサーブレットおよび JSP 用の設定を指定するためにユーザインタフェースを使用する方法の詳細は、オンラインヘルプの次の項目を参照してください。

次のページは Web Server Administration Server の「Global Settings」タブにあります。

- 「Configure JRE/JDK Paths」ページ

次のページはサーバマネージャの「Java」タブにあります。

- 「Enable/Disable Servlets/JSP」ページ
- 「Configure JVM Attributes」ページ
- 「Delete Version Files」ページ

次のページはサーバマネージャの「Legacy Servlets」タブにあります。

- 「Configure Global Servlet Attributes」ページ
- 「Configure Servlet Attributes」ページ
- 「Configure Servlet Virtual Path Translation」ページ
- 「Configure Servlet Directory」ページ

古いバージョンの構成ファイル

この節では、次のファイルの用途と使用方法について説明します。

- servlets.properties
- rules.properties
- contexts.properties

これらのファイルのすべては、ディレクトリ *server_id*/config に置かれています。

注 iPlanet Web Server 6.0 では、web-apps.xml ファイルが各仮想サーバ用に用意されています。これによって、各仮想サーバ用に Web アプリケーションを別々に構成できます。servlets.properties、rules.properties、および contexts.properties ファイルは、サーバインスタンス用にデフォルトの仮想サーバにのみ適用します。これらのファイルは下位互換用です。すべての Web アプリケーションの構成を web-apps.xml ファイルを使用して行うことをお勧めします。web-apps.xml ファイルの詳細は、第 2 章「Web アプリケーション」を参照してください。

servlets.properties

servlets.properties ファイルは、デフォルトの仮想サーバに対してのみ、グローバルサーブレット設定およびシステムのサーブレットリストを定義します。

グローバルサーブレット設定の例には、iPlanet Web Server を起動したときに実行するサーブレットの指定があります。servlets.properties ファイルは、個々のサーブレットの構成情報も指定します。構成情報には、クラス名、クラスパス、およびサーブレットに必要な入力引数が設定されます。

サーブレットの仮想パス変換を指定する場合、サーブレットは servlets.properties ファイルで設定されている必要があります。

サーバマネージャインタフェースの「Legacy Servlets>Configure Servlet Attributes」ページを使って、または servlets.properties を直接編集して、サーブレットの構成情報を指定できます。サーバマネージャインタフェースの「Legacy Servlets>Configure Servlet Attributes」ページで変更するときには、システムは常に自動的に servlets.properties を更新します。

サーブレットの属性を指定する場合、サーブレットの name パラメータを指定します。この名前は、サーブレットのクラスファイル名である必要はありません。これは、サーブレットの内部識別子です。code パラメータの値としてクラスファイル名を指定します。

次に、servlets.properties ファイルの例を示します。

```
# サーブレットのプロパティ
# 起動時に読み込むサーブレット
servlets.startup= hello
# ServletContext.getRealPath が動作するために必要な
# デフォルトのドキュメントルート
servlets.config.docRoot=d:/Netscape/Server4/docs
# トラッカサーブレット
servlet.tracker.code=MyTrackerServlet
servlet.tracker.classpath=d:/Netscape/Server4/docs/servlet
# demo1 サーブレット
servlet.demo1.code=Demo1Servlet
servlet.demo1.classpath=d:/Netscape/Server4/docs/demos
servlet.demo1.initArgs=a1=0,b1=3456
servlet.demo1.context=context1
```

rules.properties

rules.properties ファイルは、デフォルトの仮想サーバに対してのみ、サーブレットの仮想パス変換を定義します。たとえば、/mytest2 を指す URL が servlets.properties ファイル中に記述された demo1 という名前のサーブレットを起動するようにマッピングを設定できます。サーバマネージャインタフェースの「Legacy Servlets>Configure Servlet Virtual Path Translation」ページでパラメータを設定するか、rules.properties ファイル内にパスを指定することによって、仮想パスを指定できます。

servlets.properties のサーブレットと関連付けられた name は、ファイル rules.properties で使用されます。サーブレットのクラス名は、rules.properties では表示されません。たとえば、servlets.properties の次の行は、サーブレット名 demo1 をディレクトリ d:/Netscape/Server4/docs/demos 内のサーブレットクラスファイル Demo1Servlet.class と関連付けます。

```
# servlets.properties 内
# demo1 サーブレット
servlet.demo1.code=Demo1Servlet
servlet.demo1.classpath=d:/Netscape/Server4/docs/demos
```

rules.properties の次の行は、URL http://*server_id*/mytest2 が d:/Netscape/Server4/docs/demos/Demo1Servlet.class のサーブレットを起動するように、サーブレット仮想パス変換を定義しています。

```
/mytest2=demo1
```

次に、rules.properties の例を示します。

```
# Servlet rules properties
#
# This file specifies the translation rules for invoking servlets.
# The syntax is:
#   <virtual-path>=<servlet-name>
#   or
#   @regular_expression=<servlet-name> (use double back-slashes)
#
# where <virtual-path> is the virtual path used to invoke the
# servlet, and <servlet-name> is the name of the servlet.
# Surrounding white space is ignored. The ordering of the rules is
# not important, as the longest match will always be used first.
# Use of regular expression can lead to a heavy peformance penalty
#
################################## rules
##################################
/mytest1=tracker
/mytest2=demo1
```

rules.properties での正規表現の使用

iPlanet Web Server は、rules.properties ファイルで正規表現をサポートしており、入力された URL が、ある正規表現と一致すると指定されたサーブレットが実行されます。ただし、次に挙げたファイルヘッダーの例は誤りです。

```
# 例
#
# /simple=SimpleServlet\n
# @.*\\.foo=wasp
```

「\\」は、(.) 拡張子を認識しません。この例の意図は、/my/*xxx*.foo などの URL の要求があった場合は常に、wasp サーブレットを実行することです。しかし、iPlanet Web Server は、「/」または「\」を「/」に置き換えてしまうため、全体の意味を変更してしまいます。この事態を避けるには、「/」または「\」は使わずに、正規表現を次のように指定してください。

```
@.*[.]foo$=wasp
```

特定のタイプのファイルを特定のサーブレットに割り当てる別の方法は、次の URL にあるサーブレット 2.2 API 仕様で記述されている `web.xml` ファイルの `servlet-mapping` の箇所の説明を参照してください。

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

特定のタイプのファイルを `obj.conf` ファイル内で特定のサーブレットに割り当てる方法は、100 ページの「obj.conf のオブジェクト」を参照してください。

contexts.properties

`contexts.properties` ファイルは、デフォルトの仮想サーバに対してのみ、コンテキストを定義します。これによって、複数のサーブレットがデータを交換したり互いのフィールドにアクセスしたりすることが可能になります。コンテキストは、仮想サーバの定義やコードの識別に役立ちます。コンテキストが定義されていないと、デフォルトのグローバルコンテキストがすべてのサーブレット用に使用されます。

注 すべての JSP はデフォルトのグローバルコンテキストに属しています。JSP にカスタムコンテキストは定義できません。ただし、グローバルコンテキストのプロパティを変更して、JSP に変更を反映させることはできます。

サーブレット用のコンテキストが定義されていない場合、サーブレットはグローバルコンテキストに属しています。複数のコンテキスト内で同じサーブレットを使用できます。

コンテキストの `name` のみが必須です。その他のすべての未指定のプロパティは、グローバルコンテキストから引き継がれます。グローバルコンテキストのプロパティを変更することもできます。`contexts.properties` ファイル内のコメントは、グローバルコンテキストのデフォルトプロパティの値を一覧表示しています。

次に、`contexts.properties` の例を示します。

```
# @(#)contexts.properties (autogenerated)
#
# Contexts Properties:
#
# context.<context_name>.sessionmgr=session manager (some session managers
#               (like MMapSessionManager) can only be instatiated once within the
#               server
# context.<context_name>.sessionmgr.initArgs=list of (name, value) pairs which
```

```
#               will represent parameters specific to the session manager
# context.<context_name>.initArgs=list of (name, value) pairs which will be added
#               to this context's attributes
# context.<context_name>.respondCookieVersion=(cookie version) tells the server
#               whether to respond with specific cookie version
# context.<context_name>.sessionExpireOnClose(true|false) tells the server to
#               mark session cookies as directed to expire when the user quits
#               the browser
# context.<context_name>.tempDir=path (forward slashes only) - sets up Servlet API
#               2.2 property for the temporary directory
# context.<context_name>.reloadInterval=seconds - time interval within which the
#               server checks for jsp and servlet files being modified (global
#               context only)
# context.<context_name>.bufferSize=bytes - initial http output stream buffer size
# context.<context_name>.docRoot=path (forward slashes only) - this context
#               document root when not specified - web server's document root
#               will be used (default)
# context.<context_name>.inputStreamLengthCheck=(true|false) - makes
#               ServletInputStream stop reading data when Content-Length bytes
#               are read
# context.<context_name>.outputStreamFlushTimer=(seconds|0) - forces the stream
#               to flush the data if certain time elapsed since the last flush;
#               0 - ignore it
# context.<context_name>.uri=contex_uri_base - additional URI prefix which
#               servces as a context base
# context.<context_name>.authdb=name - authentication database
# context.<context_name>.classpath=name - global classpath for this context
# context.<context_name>.signleClassLoader=(true|false) - tells the servlet
#               engine whether to use a single class loader for all servlets in
#               the context
# context.<context_name>.serverName=name - server instance name
# context.<context_name>.contentTypeIgnoreFromSSI=(true|false) - ignore
#               setContentType when invoked from SSI
# context.<context_name>.parameterEncoding=(utf8,none,auto) - advises the web
#               server on how to decode parameters from forms
# context.<context_name>.isModifiedCheckAggressive=(true|false) - determines
#               whether to be aggressively checking dependencies for the servlet
#               loader to reload modified servlets
#
# <context_name>="global" is reserved for the global context. Every new context
#               will inherit initial settings of the global context
#
# Context properties:
# context.global.sessionmgr=com.netscape.server.http.session.SimpleSessionManager
# context.global.sessionmgr.initArgs=
# context.global.initArgs=initial=0
# context.global.respondCookieVersion=0
```

```
# context.global.tempDir=/tmp
# context.global.reloadInterval=5
# context.global.bufferSize=4096
# context.global.docRoot=/foo/bar
# context.global.inputStreamLengthCheck=true
# context.global.outputStreamFlushTimer=0
# context.global.uri=/
# context.global.authdb=default
# context.global.classpath=
# context.global.singleClassLoader=false
# context.global.contentTypeIgnoreFromSSI=true
# context.global.parameterEncoding=utf8
# context.global.isModifiedCheckAggressive=false


#
################################## Contexts ##############################
context.global.initArgs=docRoot=C:/iPlanet/Servers
context.context1.name=context1
```

以下の節では、いくつかのコンテキストプロパティを詳しく説明します。

isModifiedCheckAggressive

新しいバージョンでは、次のどちらかの操作を実行していないと、パッケージ化されたサーブレットを変更したときに自動的に再読み込みされません。

- `isModifiedCheckAggressive` プロパティを `true` に設定する。

例：`context.global.isModifiedCheckAggressive=true`

- サーブレットを起動するときに、`.class` 拡張子を使用する。

例：`http://`*`your_server`*`/servlet/HelloWorld.HelloWorldServlet.class`

parameterEncoding

`context.global.parameterEncoding` プロパティを使って、サーブレットのパラメータの文字の符号化方式を決めることができます。次のようなオプションがあります。

`none`	システムのデフォルトの符号化方式を使用します。サーブレットのパラメータデータの符号化方式をシステムのデフォルトの符号化方式と同じにする場合に、このオプションを設定します。

auto	(デフォルト) 適切な符号化方式を次の順番で判別します : 1) Content-Type ヘッダーに設定されている場合は charset、2) parameterEncoding 属性 (75 ページの「ServletRequest.setAttribute」を参照)、3) j_encoding などの隠しフォームフィールド。判別できなかった場合は、システムのデフォルトの符号化方式が使用されます。このオプションを設定すると、サーブレットパラメータ内で ASCII 以外の文字の変換ミスを防ぐことができます。 このプロパティが auto に設定されている場合、サーバは要求されたコードに変換する前に、ネイティブな文字コードを java バイト配列に変換する必要があります。したがって、none または特定のコード方式の方が、パフォーマンスが若干向上します。
サポートされている java 文字の符号化方式	utf8 または Shift_JIS などの特定の符号化方式。サーブレットパラメータが使うコード方式を知っている場合は、このオプションを設定します。文字コードの全リストは、以下の URL で入手できます。 http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html
responseCT	可能であれば、応答コンテンツの種類から適切な符号化方式を判別します。判別できなかった場合は、システムのデフォルトの符号化方式が使用されます。

サーブレットが ServletRequest.getParameter メソッドを使用して、フォームフィールドから UTF-8 以外の文字がある値を取得する場合、contexts.properties ファイルでparameterEncoding プロパティがデフォルトの auto または responseCT に設定されている必要があります。そうでない場合、getParameter メソッドによって抽出された値は、ゼロか定義されていない値となります。 詳細は、75 ページの「ServletRequest.getParameter」を参照してください。

parameterEncoding プロパティ設定は、iPlanet Web Server が要求を受け取った後に、サーブレットがパラメータを処理する際に適用されます。サーバに送られた URI は、要求方法が GET の場合は特に、標準 ASCII セットだけで構成されている必要があります。その他のすべての文字は符号化される必要があります。

たとえば、バックスラッシュを符号化するには、% とその後ろに ASCII 文字セットに対応する 16 進数 (5c) を置いてバックスラッシュを置き換えます。その結果、vw\xyz は、vw%5cxyz となります。符号化する必要のある文字の詳細は、22 ページの「parameter-encoding」を参照してください。

singleClassLoader

デフォルトでは、singleClassLoader プロパティは false です。これは、サーブレットがコンテキストを共有していても、各サーブレットは異なるクラスローダで読み込まれることを意味します。これによって、2 つのサーブレットが互いの静的クラスメンバーにアクセスすることが困難になっています。同じクラスローダでコンテキスト内にすべてのサーブレットを読み込むには、singleClassLoader プロパティを true に設定します。

サーブレットの再読み込み (.class または .jsp ファイルが変更されたときに行われる) は、singleClassLoader=true のときには行われません。

singleClassLoader=true の場合、ClassCache ディレクトリを contexts.properties の context.global.classpath 行に追加しない限り、JSP への RequestDispatcher.forward 呼び出しは行われません。以下に例を示します。

```
context.global.singleClassLoader=true
context.global.classpath=server_root/https-server_id/ClassCache
```

古いバージョンの例

iPlanet Web Server 6.0 では、古いバージョンのサーブレットと JSP ファイルの例のセットが提供されています。これらは、次の場所に格納されています。

server_root/plugins/servlets/examples/legacy

legacy サブディレクトリには次のディレクトリが含まれています。

- beans.10 - JSP 1.*x* の Java Bean ファイルの例が記述されています。
- jsp.10 - JSP 1.*x* の例をそれぞれ記述するサブディレクトリが含まれています。
- jsp.10/hangman - iPlanet Web Server 4.*x* で SSJS アプリケーションであった、JSP アプリケーションが含まれています。SSJS アプリケーションを JSP に変換する方法の詳細は、付録 B「SSJS アプリケーションの変換」を参照してください。
- make - サーブレットの Makefile の例が記述されています。これらは、ほかのすべての Makefile での共通の規則が記述されている Makefile です。
- servlets - サーブレットの例用の Java ソースファイルおよび Makefile をそれぞれ持つサブディレクトリが含まれています。
- tools - SDKTools.jar ファイルおよびその他のユーティリティファイルが含まれています。

古いバージョンのサーブレットのパフォーマンスを最大限にする

サーブレットのパフォーマンスを改善するため、次のガイドラインについて考慮してください。

- `obj.conf` ファイルを手動で編集する場合、サーブレットの `NameTrans` (`NameTrans fn="NSServletNameTrans" name="servlet"`) が常に最初の `NameTrans` 指令になるようにします。

  この指令は、読み込まれたサーブレットのために高度に最適化された URI キャッシュを使用しており、一致すると `REQ_PROCEED` を返します。これにより、ほかの `NameTrans` 指令を実行する必要がなくなります。

- デフォルトの仮想サーバでは、(「Configure Servlet Attributes」ページまたは `rules.properties` および `servlets.properties` を使用して) 個別に定義されたサーブレットは、サーブレットディレクトリに動的に読み込まれたサーブレットより少し速く動作します。

magnus.conf および obj.conf のサーブレット設定

iPlanet Web Server 6.0 Administration Server は、サーブレットが有効な場合、サーブレットエンジンを読み込むために、magnus.conf ファイルと obj.conf ファイルを自動的に変更します。サーバマネージャインタフェースを使ってサーブレットの設定に変更を加えるたびに、システムが自動的にこれらのファイルを適切に更新します。

ただし、サーブレットに影響を与える設定に関しては、この付録ではそれを次の節で説明しています。

- magnus.conf の Init 指令
- obj.conf のオブジェクト
- 登録済みサーブレットディレクトリ用の指令
- JSP 指令

magnus.conf の Init 指令

magnus.conf の Init セクションの以下の指令は、サーブレットエンジンを読み込み、初期化して、サーブレットを有効にします (Windows NT 用)。

```
Init fn="load-modules"
shlib="server_root/bin/https/bin/NSServletPlugin.dll"
funcs="NSServletEarlyInit,NSServletLateInit,NSServletNameTrans,
NSServletService" shlib_flags="(global|now)"
Init fn="NSServletEarlyInit" EarlyInit=yes
Init fn="NSServletLateInit" LateInit=yes
```

UNIX の場合の指令は、次のパラメータ以外は同じです。

```
shlib="server_root/bin/https/lib/libNSServletPlugin.so"
```

NSServletEarlyInit には、オプションのパラメータ cache_dir (JSP クラスの一時キャッシュディレクトリの位置を指定する) を指定できます。デフォルトでは、ディレクトリは ClassCache という名前で、サーバルートディレクトリの下に置かれます。

NSServletLateInit には、次のオプションのパラメータを指定できます。

CatchSignals	Java スレッドダンプを記録するかどうかを指定します。値は、yes または no です。
requestDispatcherNestDepth	RequestDispatcher.forward メソッドおよび include メソッドの入れ子の深さを制御します。デフォルトの深さは、10 です。このパラメータは、JSP の forward 呼び出しおよび include 呼び出しの入れ子のレベルにも影響します。magnus.conf 中の StackSize 指令に適切な値が設定されないと、この深さに任意の値を割り当てることはできません。このパラメータに任意に高い値を設定すると、スタックオーバーフローやサーバクラッシュの原因になる可能性があります。

obj.conf のオブジェクト

注 この節の情報は、古いバージョンのサーブレット (iPlanet Web Server 4.x) にあてはまりますが、推奨されていません。

NSServletService には、2 つのオプションのパラメータ servlet="*servlet_name*" と context="*context_name*" を指定できます。これらのパラメータによって、特定のサーブレットまたはコンテキストに対応して応答を生成するオブジェクトを、obj.conf に定義できます。1 つの指令に、1 つまたは両方のパラメータを使用できます。サーブレットまたはコンテキストは、servlets.properties ファイルまたは

contexts.properties ファイルに定義する必要があります。特定のサーブレットまたは特定のサーブレットコンテキスト、またはその両方に属するオブジェクトを定義できます。たとえば、次のように、特定のタイプ (ここでは、*.foo) のファイルに特定のサーブレット (ここでは、wasp サーブレット) を割り当てることができます。

```
<Object name="default">
NameTrans fn=assign-name name=foo from=*.foo
... 他の指令 ...
</Object>

<Object name="foo">
ObjectType fn=force-type type=magnus-internal/servlet
Service fn="NSServletService" servlet="wasp"
</Object>
```

特定のタイプのファイルを特定のサーブレットに割り当てる別の方法は、次の URL にあるサーブレット 2.2 API 仕様で記述されている web.xml ファイルの servlet-mapping の箇所の説明を参照してください。

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

iPlanet Web Server 4.*x* で、特定のタイプのファイルを特定のサーブレットに割り当てる方法は、91 ページの「rules.properties での正規表現の使用」を参照してください。

NSServletService の基本的な使用例は、iPlanet Web Server の『NSAPI プログラマーズガイド』の第 2 章「obj.conf の構文と使用法」にある Service の例の説明を参照してください。

サーブレットが有効な場合、次の指令がデフォルトのオブジェクトに追加されます。

```
NameTrans fn="NSServletNameTrans" name="servlet"
```

この指令は、サーブレット仮想パス変換および URI キャッシュのために使用されます。この行は消さないでください。

また、obj.conf には通常、次のオブジェクトがあります。このオブジェクトは削除しないでください。

```
<Object name="servlet">
ObjectType fn=force-type type=text/html
Service fn="NSServletService"
</Object>
```

このオブジェクトを削除すると、サーバマネージャインタフェースを使って、サーブレットを有効にしたりサーブレットの設定を変更したりすることができなくなります。

詳細は、iPlanet Web Server の『NSAPI プログラマーズガイド』を参照してください。

登録済みサーブレットディレクトリ用の指令

注	この節の情報は、古いバージョンのサーブレット (iPlanet Web Server 4.x) にあてはまりますが、推奨されていません。

obj.conf 内のデフォルトのオブジェクトは、すべての要求が、登録済みサーブレットディレクトリにアクセスできるように ServletByExt という名前を割り当てる NameTrans 指令を持っています。次に例を示します。

```
NameTrans fn="pfx2dir" from="/servlet"
dir="D:/Netscape/Server4/docs/servlet" name="ServletByExt"
```

ServletByExt という名前の別のオブジェクトは、次のように、サーブレットの要求を処理する命令を持っています。

```
<Object name="ServletByExt">
ObjectType fn="force-type" type="magnus-internal/servlet"
Service type="magnus-internal/servlet" fn="NSServletService"
</Object>
```

サーブレットディレクトリが現在登録されていなくても、このオブジェクトは削除しないでください。このオブジェクトを削除すると、サーバマネージャインタフェースを使ってサーブレットディレクトリを登録できなくなります。

JSP 指令

mime.types の次の行は、拡張子が .jsp のファイルのタイプを設定します。

```
type=magnus-internal/jsp exts=jsp
```

obj.conf の次の指令は、magnus-internal/jsp タイプのファイル (つまり、JSP ファイル) の要求の処理を扱います。これは、古いバージョンの JSP のためにだけ必要です。

```
Service fn="NSServletService" type="magnus-internal/jsp"
```

SSJS アプリケーションの変換

この付録では、サーバサイド JavaScript のアプリケーションを JSP に変換する方法についての情報を説明します。次の節で構成されています。

- JavaScript と Java の違い
- JavaScript から Java クラスへの変換
- 変換手順
- 変換例

JavaScript と Java の違い

SSJS アプリケーションを JSP に変換する前に、JavaScript と Java の違いを理解する必要があります。JavaScript と Java はある点では似ていますが、それ以外は根本的に異なります。

JavaScript には、Java のような静的型定義および強力な型チェック機能がありません。JavaScript は、実行時システムをサポートしており、数値、ブール値、および文字列値を表す少数のデータタイプがベースになっています。Java は、宣言文で構築されたクラスのコンパイル時システムをサポートしています。

JavaScript はまた、特別な宣言が必要のない関数をサポートしています。関数は、あいまいに型定義されたメソッドとして実行することも、オブジェクトのプロパティとして使用することもできます。Java では、メソッドはクラスで定義され、クラスに属し、明確に型定義されます。

JavaScript は、Java と比較すると、非常に自由な形式の言語です。Java では、すべての変数、クラス、およびメソッドを宣言する必要があります。メソッドは、Public、Private、または Protected として、宣言する必要があります。変数、パラメータ、およびメソッドの戻り値の型は、明示的に定義されます。

Java は、高速実行および型の安全性のために設計された、クラスベースのプログラミング言語です。たとえば、型の安全性とは、Java の整数をオブジェクト参照にキャストしたり、Java バイトコードを破棄して専用メモリにアクセスしたりできないことを意味します。Java のクラスベースのモデルとは、プログラムがクラスとそのクラスのメソッドのみで構成されていることを意味します。Java のクラス継承や強力な型定義には通常、密接に結合されたオブジェクト階層を必要とします。Java のプログラミングにはこれらの条件が必要となるため、JavaScript のオーサリングよりも複雑です。

表 B-1 JavaScript と Java の比較

JavaScript	Java
クライアントによって解釈される (コンパイルではない)	サーバからダウンロードされたコンパイル済みのバイトコードがクライアント上で実行される
オブジェクトをサポートするスクリプト言語	オブジェクト指向言語
オブジェクトのタイプの区別がない。継承はプロトタイプメカニズムによるものであり、プロパティや関数はすべてのオブジェクトに動的に追加される	オブジェクトは、クラス階層のすべての継承を持つクラスとインスタンスに分割される。クラスとインスタンスは、動的に追加されるプロパティまたはメソッドを持つことはできない
関数は、クラスの内部にある場合があるが、必ずしもそうである必要はない	すべてのメソッドが、クラスの内部にある必要がある
コードは、HTML と統合され、HTML に埋め込まれる	コードは、HTML とは区別される。ただし、JSP に HTML のタグを埋め込むことはできる
柔軟な型変換 : 変数および関数のタイプは宣言されない （動的型定義)	厳密な型変換 : 変数およびメソッドタイプは、宣言する必要がある (静的型定義)
ハードディスクに自動的に書き込まれない	ハードディスクに自動的に書き込まれる
ステートメントの終わりのセミコロンは任意である	すべてのステートメントはセミコロン (;) で終わらなければならない

JavaScript から Java クラスへの変換

SSJS アプリケーションで使用されるオブジェクトは、Java では、クラスに変換する必要があります。表 B-2 に、これらのオブジェクトの変換結果を示します。

表 B-2 JavaScript と Java の基本的なクラス

SSJS クラス	Java クラス	コメント
`Client`	`Session`	Java セッションは、SSJS クライアントのように自動ではない
`Project`	`Context`	
`Request`	`Request`	
`Server`	`System` または `Context`	メソッドは、2 つの Java クラスに分割される

SSJS には、データベース接続を扱う特別なライブラリが含まれています。JSP は Java なので、すべてのデータベース接続は JDBC 経由で処理されます。JDBC データベースドライバは、データベースベンダーから直接入手できます。入手可能ならば、常に Pure Java (タイプ 4) ドライバを使うことをお勧めします。表 B-3 に、データベース接続クラスの一覧を示します。

表 B-3 JavaScript と Java のデータベースクラス

SSJS クラス	Java クラス	コメント
`Connection`	`Connection`	
`Cursor`	`ResultSet`	`Cursor` オブジェクトはない。`ResultSet` クラスのメソッドを使ってカーソルを動かす
`DbPool`	`PooledConnection`	このクラスは、JDBC のオプションのパッケージにある
`Stproc`	`CallableStatement`	
`ResultSet`	`ResultSet`	

変換手順

アプリケーションを変換する場合は、以下の手順に従ってください。

1. アプリケーションの構造をよく調べて、ページ間で使用されるヘルパークラスがあるかどうかを確認します。ページで使用できるようにするために、最初にこれらのヘルパークラスを変換します。(hangman の例には、元の JavaScript アプリケーションでは `hangman.js` と呼ばれ、iPlanet Web Server 6.0 で提供された JSP アプリケーションでは `JavaHangManUtil.java` と呼ばれる、ヘルパークラスが含まれています。) ヘルパーを `.java` ファイルとして記述し、そのファイルを使用する前に、`.class` ファイルにコンパイルします。

2. クラス内にメソッドを定義します。メソッド (JavaScript の関数) は、メソッドが返すもの、およびメソッドにアクセスできるものを宣言する必要があります。たとえば、次のように JavaScript に記述できます。

   ```
   function InitAnswer(str) { function_code }
   ```

   Java では、同じメソッドを次のように宣言する必要があります。

   ```
   public static String InitAnswer(String str) { method_code }
   ```

   このメソッドは、`String` を渡され、`String` を返します。`public` キーワードは、ほかのすべてのクラスがこのメソッドを呼び出すことができることを意味します。`static` キーワードは、仮想マシンが、このメソッドを含むクラスのオブジェクトを作成しなくても、このメソッドを実行できることを意味します。

3. アプリケーションのクライアントオブジェクトをセッション Bean に変換します。サーバサイド JavaScript では、データは、クライアントオブジェクト内でより使いやすくなっています。Java セッションオブジェクトでは、プログラマは、セッションからデータを受け取り、もし変更があればデータを戻す必要があります。この作業は、ラッパー Bean で行うことができます。JSP の機能を扱う Bean は、非常に強力なものです。スコープを `session` になるように指定できます。スコープは、セッションに自動的に保存されます。

4. HTML ファイルを JSP ファイルに変換します。これは、`<SERVER>` タグおよび `</SERVER>` タグを見つけ出し、それらを `<%` タグおよび `%>` JSP タグに置き換える作業でほとんど占められます。次に、構文を調べて、正しい JSP または Java 構文に変更します。

5. 必要に応じて、特別な行をファイルに追加します。たとえば、次の行はヘルパークラスをインポートします。

   ```
   <%@ page import="HangManUtil,HangBean" %>
   ```

   そして、

   ```
   <jsp:useBean id="client" scope="session" class="HangBean"/>
   ```

   上記の行は、セッション Bean の使用を指定します。

変換例

iPlanet Web Server のバージョン 4.1 のサーバサイド JavaScript の例であった、hangman は、バージョン 6.0 では JavaServer Page の例に変換されました。変換された例は、次のディレクトリにあります。

server_root/plugins/servlets/examples/legacy/jsp.10/hangman

付録 C

JVM の構成

Java 仮想マシン (JVM) は、適切に設定されていれば、構成を追加しなくてもデフォルトで動作します。

しかし、クラスパス情報などの JVM の設定を追加指定する必要がある場合は、Administrator インタフェースを介して iPlanet Web Server 用に JVM プロパティを追加できます。最大 64 個のプロパティを追加できます。

注 「Java」タブの「Configure JVM Attributes」ページ上にあるいくつかの属性は、「Default」と表示されています。JVM によってデフォルト値は異なります。実際のデフォルト値を知るために JVM に照会することはできません。JVM のマニュアルを参照してください。たとえば、Sun の JVM では、デフォルトで JIT は JVM で有効なので、「Enable JIT」オプションで「Yes」を選択すると、「Default」と表示されます。しかし、「No」を選択すると、明示的なエントリ `java.compiler=NONE` が `jvm12.conf` ファイルに追加されます。

JVM の詳細は、次のサイトにある Sun の『The Java Virtual Machine Specification』を参照してください。

`http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html`

jvm12.conf ファイル

サーバの config ディレクトリに置かれている jvm12.conf 構成ファイルを編集して、JVM パラメータを構成することもできます。

たとえば、JIT を無効にするには、jvm12.conf に次の行を追加します。

```
java.compiler=NONE
```

jvm12.conf ファイルの例を次に示します。jvm.classpath の値は、実際のファイルでは 1 行にする必要があります。

```
[JVMConfig]
#jvm.minHeapSize=1048576
#jvm.maxHeapSize=16777216
#jvm.enableClassGC=0
#jvm.verboseMode=1
#jvm.enableDebug=1
#jvm.printErrors=0
#jvm.option=-Xrunoii
#jvm.profiler=optimizeit
#jvm.disableThreadRecycling=0
#jvm.serializeAttach=0
#jvm.stickyAttach=0
#jvm.trace=5
#java.compiler=NONE
#OPTITDIR=D:/App/IntuitiveSystems/OptimizeIt30D
#jvm.serializeFirstRequest=0
#jvm.include.CLASSPATH=1
#nes.jsp.forkjavac=0
#nes.jsp.enabledebug=1
jvm.classpath=/usr/java/tools.jar
```

通常、JDK1.2 の構成には普通のプロパティオプション (*name=value* など) を、JVM ベンダー依存の構成には jvm.option=*options* を使用してください。jvm.option パラメータは複数指定できます。

JPDA がインストールされている場合は、jvm12.conf ファイルに設定できるデバッグオプションのリストを次のサイトで入手できます。

```
http://java.sun.com/products/jpda/doc/conninv.html#Invocation
```

サーブレットと JSP のデバッグ、およびデバッグに必要な jvm12.conf パラメータの詳細は、第 5 章「サーブレットおよび JSP のデバッグ」を参照してください。

JVM 環境変数の使用

jvm12.conf ファイルの jvm.option 設定、または Administration Server を使って、JVM 環境変数を定義した場合、サーブレットはそれらを認識しません。環境変数は、普通の *name=value* で指定してください。

スタンドアロンのコマンドライン java プログラムを実行している場合、次のオプションを使ってシステムプロパティを java プログラムに渡します。

```
java -Dorg.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB myprogram
```

myprogram.java ファイル中の次の行は、上記で設定したステムプロパティセットを取得します。

```
System.out.println("org.omg.CORBA.ORBClass="+System.getProperty("org.omg.CORBA.ORBClass"));
```

iPlanet Web Server のサーブレットにも同じ処理をさせる場合は、次の行を jvm12.conf ファイルに置く必要があります。

```
org.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB
```

次のような jvm.option 設定は使わないでください。

```
jvm.option=-Dorg.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB
```

または

```
jvm.option=org.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB
```

サーブレットまたは JSP では、次の行を使って、上記のシステムプロパティを取得できます。

```
System.out.println("org.omg.CORBA.ORBClass="+System.getProperty("org.omg.CORBA.ORBClass"));
```

jvm12.conf パラメータのリファレンス

jvm12.conf ファイルに設定できる JVM パラメータを表 C-1 に示します。

表 C-1　jvm12.conf の設定

設定	指定できる値	デフォルト値	説明
jvm.minHeapSize		1048576 (1M バイト)	Java に割り当てられた最小ヒープサイズ Solaris の場合は、この値を 3145278 (3M バイト) に変更する。HPUX の場合は、この値を 4194304 (4M バイト) に変更する。その他のすべてのシステムでは、1M バイトが最適である。
jvm.maxHeapSize		16777216 (16 MB)	Java に割り当てられた最大ヒープサイズ
jvm.enableClassGC	0 (オフ)、1 (オン)	0	クラスガベージコレクションを有効または無効にする -Xnoclassgc の代わりに使用する
jvm.verboseMode	0 (オフ)、1 (オン)	0	JVM の冗長モードを有効または無効にする。オンの場合、JVM は、クラスの読み込みなど、実行中の内容を注釈として記録する。注釈は、エラーログに追加される
jvm.enableClassGC	0 (オフ)、1 (オン)	0	JVM のリモートデバッギングを有効または無効にする -Xdebug の代わりに使用する。 リモートデバッギングの詳細は、第 5 章「サーブレットおよび JSP のデバッグ」を参照
jvm.printErrors	0 (オフ)、1 (ログファイルに記録)、2 (stderr に記録)	0	vfprintf によるエラーレポートの出力を有効または無効にする

表 C-1　jvm12.conf の設定 (続き)

設定	指定できる値	デフォルト値	説明
jvm.option			ベンダー JVM オプションを設定可能にする 同等のパラメータが存在するので、次のオプションは無視される。-D、-Xnoclassgc、-Xdebug、-Xms、-Xmx、-verbose
jvm.profiler			プロフィラを指定する。Intuitive Systems 社製の optimizeit プロファイラを使用する場合は、OPTIDIR 設定も指定する必要がある このオプティマイザの詳細は、付録 D「サーブレットのリモートプロファイリング」を参照
jvm.disableThreadRecycling	0 (オフ)、1 (オン)	0	スレッドの再利用を有効または無効にする。オンの場合、サーバは常にグローバルスコープスレッドを作成してサーブレットを実行する。オフの場合、グローバルスコープスレッドは、スレッドを処理する要求がグローバルスコープ内にない場合にのみ作成される
jvm.serializeAttach	0 (オフ)、1 (オン)	0	オンの場合、JVM に接続するスレッドは直列に並べられる。デフォルトでは (オフの場合)、同時に複数スレッドを JVM に接続できる

表 C-1　jvm12.conf の設定 (続き)

設定	指定できる値	デフォルト値	説明
jvm.stickyAttach	0 (オフ)、1 (オン)	0	このパラメータの値を 1 に設定すると、スレッドが JVM に接続されることをスレッドに記憶させることができる これにより、AttachCurrentThread 呼び出しおよび DetachCurrentThread 呼び出しがなくなり、要求処理の速度が上がる。ただし、次のような不利な点を持つ可能性がある。別の処理を行っている再利用スレッドは、ガベージコレクタによって任意に処理が中断される可能性がある スレッドプールを使うと、ほかのサブシステムでの、この不利な点を解消することができる。スレッドプールの詳細は、iPlanet Web Server の『管理者ガイド』を参照
jvm.trace		5	追跡レベルを決める。 サーブレットおよび JSP のデバッグの推奨レベルは 7。レベル 5 は、サーブレットエンジンメッセージを表示する。レベル 6 は、サーブレットおよび JSP エンジンメッセージを表示する。レベル 7 は、ブラウザにこれらのメッセージおよびその他の例外エラーメッセージを表示する
jvm.allowExit	0 (オフ)、1 (オン)	0	処理の終了を有効または無効にする
java.compiler			Java コンパイラを指定する。JIT (Just In Time) コンパイラをオンおよびオフにするオプションについては、JVM のマニュアルを参照。jvm.enableDebug がオンの場合、NONE に設定する必要がある

表 C-1　jvm12.conf の設定 (続き)

設定	指定できる値	デフォルト値	説明
OPTITDIR	パス	*	プロファイラが optimizeit の場合、プロファイラにパスを指定する
nes.jsp.enabledebug	0 (オフ)、1 (オン)	1	冗長な JSP コンパイル追跡を有効または無効にする リモートデバッギングの詳細は、第 5 章「サーブレットおよび JSP のデバッグ」を参照。
jvm.include.CLASSPATH	0 (オフ)、1 (オン)	1	jvm.classpath 設定に CLASSPATH 環境変数の値を追加するかどうかを指定する
nes.jsp.forkjavac	0 (オフ)、1 (オン)	0	オンの場合、JSP の Java コンパイルは別のプロセスで実行する
jvm.serializeFirstRequest	0 (オフ)、1 (オン)	1 Linux、AIX、および Compaq (DEC) 用。0 その他のプラットフォーム用。	オンの場合、1 つの要求スレッドだけが、サーブレットオブジェクトを読み込み、構築する。サーブレットが読み込まれ、初期化されると、同じサーブレットへの新しい要求も同時に発生する。この設定は、Linux、AIX、および Compaq (DEC) ではオンにする必要がある
jvm.classpath	スラッシュだけ持つパス		JVM に従属する JAR ファイルへのパスを指定する。必要に応じて、クラスパスの値を追加入力する たとえば、JSP が JAR にパッケージ化されている Bean を使う場合、JAR パスをクラスパスに追加する JDBC ドライバを使うセッションマネージャを使用している場合は、このクラスパスにドライバの JAR ファイルを必ず設定する

* *N*:/App/IntuitiveSystems/OptimizeIt30D の *N* は、OptimizeIt がインストールされているドライブ

サーブレットのリモートプロファイリング

サーバ側のパフォーマンスのボトルネックを見つけるために、プロファイラを使って、iPlanet Web Server 上でリモートプロファイリングを行うことができます。この付録では、次の 2 つのプロファイラについて説明します。

- Optimizeit! プロファイラ
- HPROF プロファイラ

Optimizeit! プロファイラ

Optimizeit! は、次の Intuitive Systems の Web サイトで購入できます。

`http://www.optimizeit.com/index.html`

次の指示に従って Optimizeit! をインストールすることで、Optimizeit! が iPlanet Web Server 6.0 に統合されます。

リモートプロファイリングを有効にするには、必要に応じて `jvm12.conf` ファイルに次のような変更を行います。

```
jvm.enableClassGC=0
jvm.option=-Xrunoii
jvm.profiler=optimizeit
java.compiler=NONE
OPTITDIR=optimizeit_root_dir/OptimizeIt30D
```

サーバがこの構成で起動したら、プロファイラを接続できます (詳細は、Optimizeit! のマニュアルを参照)。

また、プロファイラの所有する jar ファイルおよび dll ファイルをインクルードするために、PATH システム変数および NSES_CLASSPATH システム変数を更新します。

注	構成オプションのいずれかが指定されていないか正しくない場合、プロファイラに iPlanet Web Server のパフォーマンスに影響する問題が発生することがあります。

HPROF プロファイラ

HPROF は、Java 2 SDK で提供される簡単なプロファイラエージェントです。HPROF は動的にリンクされるライブラリで、JVMPI と相互に作用し、ASCII 形式またはバイナリ形式でファイルまたはソケットにプロファイリング情報を書き出します。この情報は、HAT などのプロファイラフロントエンドツールによってさらに処理されます。

HPROF は CPU の利用状況、ヒープ割り当て統計、およびモニター競合プロファイルを表示できます。また、すべてのヒープダンプや、Java 仮想マシンのすべてのモニターやスレッドの状態をレポートすることもできます。HPROF プロファイラの詳細は、次の Web サイトの JDK マニュアルを参照してください。

http://java.sun.com/products/jdk/1.2/docs/guide/jvmpi/jvmpi.html#hprof

UNIX 上で HPROF プロファイルを使用するには、次の手順に従ってください。

1. HPROF プロファイルを有効にするために、次のように、jvm12.conf ファイルを編集します。

```
jvm.printErrors=2
jvm.profiler=hprof
jvm.option=-Xrunhprof:options
#jvm.option=-Xrs must be commented out
java.compiler=NONE
```

HPROF を iPlanet Web Server で使うための推奨オプションは、次のとおりです。

```
jvm.option=-Xrunhprof:file=/tmp/hprof.txt,heap=all,format=a
```

または

```
jvm.option=-Xrunhprof:file=/tmp/hprof.txt,cpu=samples,format=a
```

HPROF の構文は、次のとおりです。

`-Xrunhprof[:help]|[:`*option*`=`*value*`,`*option2*`=`*value2*`, ...]`

`help` を使って、HPROF に渡すことができるオプションを列挙します。出力結果は、次のとおりです。

```
Hprof usage: -Xrunhprof[:help]|[:<option>=<value>, ...]

Option Name and Value   Description             Default
---------------------   -----------             -------
heap=dump|sites|all     heap profiling          all
cpu=samples|old         CPU usage               off
format=a|b              ascii or binary output  a
file=<file>             write data to file      java.hprof
                                                (.txt for ascii)
net=<host>:<port>       send data over a socket write to file
depth=<size>            stack trace depth       4
cutoff=<value>          output cutoff point     0.0001
lineno=y|n              line number in traces?  y
thread=y|n              thread in traces?       n
doe=y|n                 dump on exit?           y
```

2. iPlanet Web Server の起動スクリプトの行も変更する必要があります。起動スクリプトファイルは、*server_root*`/https-`*server_id*`/start` です。

   `PRODUCT_BIN=uxwdog`

   の行を、次のように変更します。

   `PRODUCT_BIN=ns-httpd`

3. 起動スクリプトを実行して、サーバを起動します。サーバはフォアグラウンドで稼動するので (手順 2 で変更)、コマンドプロンプトは、サーバが停止した後にのみ返されます。

4. 別のウィンドウまたは端末で、サーバプロセスのプロセス ID を探します。

   `% ps -ef | grep ns-httpd`

   このコマンドは、2 つの `ns-httpd` プロセスを列挙します。PPID (親プロセス ID) のカラムを見て、2 つのプロセスのどちらが親プロセスでどちらが子プロセスかを判断します。子プロセス ID の PID (プロセス ID) に注意してください。

5. 次の SIGQUIT 信号 (信号 3) を子プロセスに送ります。

   `% kill -QUIT` *child_PID*

   これにより、起動スクリプトが起動されたウィンドウに、次の ASCII メニューが表示されます。

```
% start
iPlanet-WebServer-Enterprise/6.0
[LS ls1] http://test, port 9000 ready to accept requests
Default selection: alloc and heap dump
```

```
startup: server started successfully
SIGQUIT
A SIGQUIT has been received. Do you want to:
[ 0 ]  continue program
[ 1 ]  check & print one deadlock
[ 2 ]  check & print all deadlocks
[ 3 ]  dump thread stacks
[ 4 ]  dump lock registry
[ 5 ]  heap inspection
[ 6 ]  hprof dump
[ 7 ]  continue program
Type number corresponding to selected action:
```

6. 数字の 6 を入力し、Enter キーを押して、メニューのオプション 6 を選択します。

   これにより、HPROF データが `jvm12.conf` の `jvm.option` 行で指定されたファイル (たとえば、`/tmp/hprof.txt`) に保存されます。

7. `0` を入力し、Enter キーを押して、プログラム (Web サーバ) を続行します。
8. さらに HPROF プロファイルを取り込むためには、手順 6 と 7 を繰り返します。
9. Web サーバを終了するには、別のウィンドウから終了スクリプトを実行します。

   ```
   % ./stop
   ```

10. 手順 1 と 2 の変更を取り消して、Web サーバを元の構成に戻します。

索引

K

L

M

N

O

P

R

S

T

U

V

し

す

せ

そ

た

つ

て

と

は

ふ

も

ゆ

よ

り

れ