2000 年 4 月

Java Servlet 仕様書 v2.2 邦訳

㈱クレス

サーブレットはウェブサーバサイドで動く Servlet API 使用のモジュールであり、従来のブラウザ用サーバの CGI スクリプトとのインターフェイスである CGI(Common Gateway Interface)と比較して以下のような特徴を有する。

・クライアントの要求毎に新たなプロセスを必要としない
サーブレットは、サーバ上でひとつの常駐スレッドとして動作する。従って最初にインスタンス化するとき以外は CGI よりは高速で動作する
・マルチスレッドで動作する
複数のクライアントから同じ要求（例えば GET や POST）が到来した場合、その要求に対応したメソッドは最初のクライアントに対する処理が終了しなくても新たなクライアント用として、別のスレッドに要求を割り当てる
・サーバ上に常駐する
例えば 3 層モデル（例えばブラウザからなるクライアント、サーバ上のサービス プロセス群、そしてデータベースといった構成）では、データベースなどその先のコネクションを張ったまま常駐するので、処理が早くなる
・セキュリティ・モデルがある
すべての Java 環境は Security Manager を有する。またサーブレットを JAR ファイルからアップロードするときも Security Manager が関与し、システムに危害を及ぼす恐れのある処理を実行しない

サーブレットは、プラットフォームに依存しない洗練されたサーバサイドアプリケーションを作成することができ、安全なネットワーク配信、スマートカードからスーパーコンピューティングまでのスケーラビリティが実現される。従って現在ウェブサーバの主流となりつつある。更にサーブレットは JSP(JavaServer Pages)として発展を続け、非常に簡単にアプリケーションが記述できるようになってきている。

Sun Microsystems の Servlet API をサポートするサーブレット エンジンとして現在ポピュラーなものは、LiveSofware の Jrun、 New Atlanta の ServletExec、Gefion の WAICoolRunner、IBM の WebSphere、そして Apache の Apache JServ などである。

サーブレットに関する日本語の資料も最近多くなった。しかし、より詳細に勉強したい方には結局 Sun のチュートリアル (http://java.sun.com/docs/books/tutorial/servlets/overview/)と、公式仕様書 (http://java.sun.com/products/servlet/2.2/javadoc/index.html)をお勧めすることになる。特に公式仕様書はサーブレットの原点であるにも関わらず、まだ日本語化されていない。従ってこの資料では、この公式仕様書の翻訳を試みた。つたない翻訳なので、間違いがあれば連絡いただきたい。また、不明の個所は原文を参照いただきたい。

本文に入る前に、基本概念をまず説明しておきたい。

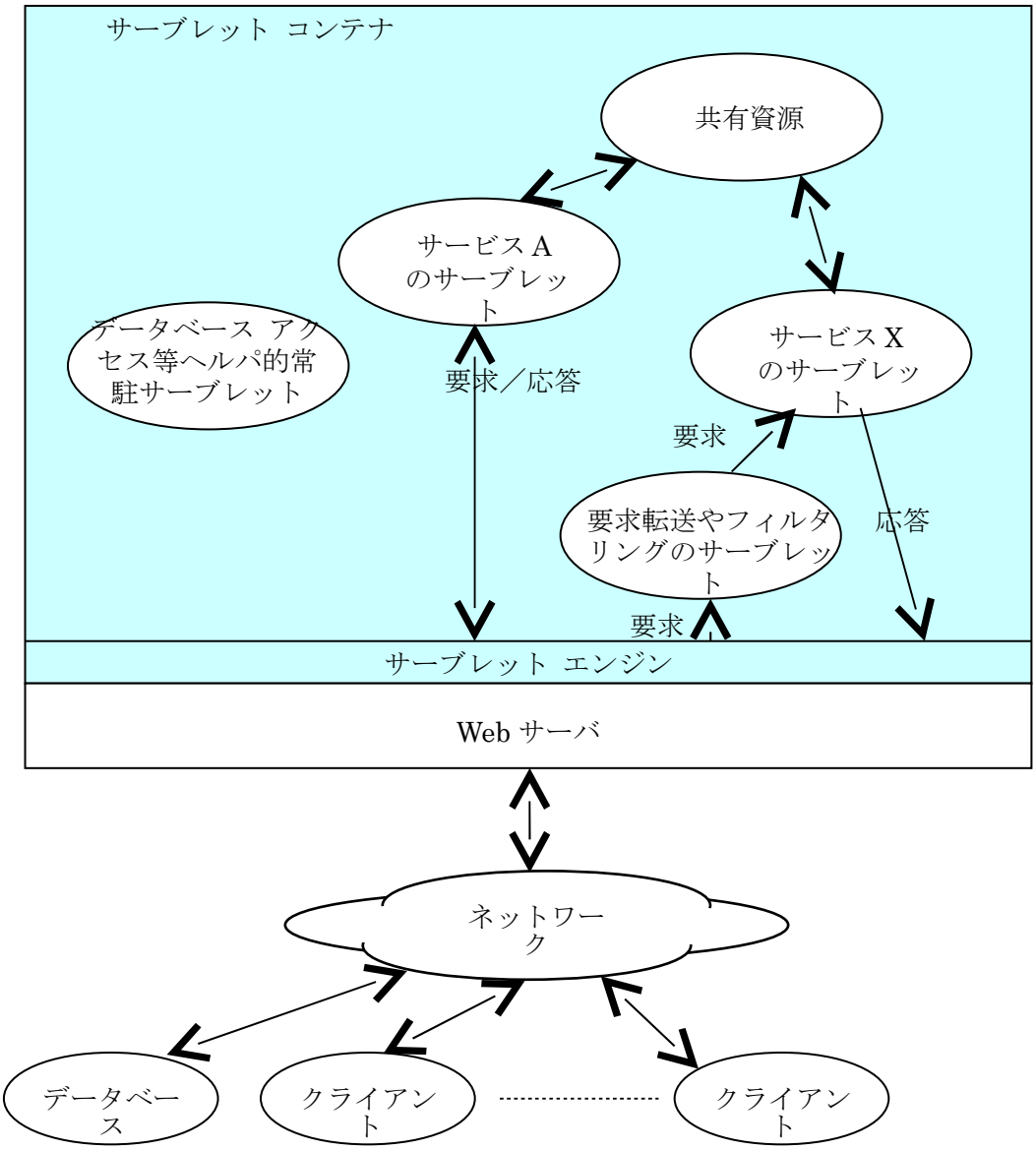
サーブレット(Servlet)の概念

サーブレット(Servlet)とは

サーブレットとは要求／応答(Request/Response)ベースのサーバ（例えば Java ベースのウェブサーバ）を拡張するモジュールである。上図のようなオーダエントリシステムを考えてみよう。このような系は、良く 3 層クライアントサーバ構造(Three-tier Architecture)と呼ばれるものである。サーブレットは、HTML のオーダエントリのフォームデータを受理し、これにこのシステムのビジネス ロジックを適用し、データベースを更新する。サーブレットはサーバ上で動作し、Applet はブラウザ上で機能する。Applet と異なり、サーブレットは GUI を有さない。これから我々が作成しようとするサーブレットに使用するサーブレット API は、サーバの環境やプロトコルを仮定してはいないので、サーブレットは各種のサーバ上に実装させることができる。特に現在は、HTTP サーバ上のサーブレットがもっとも一般的に使用されており、また多くのウェブ サーバがサーブレット API に対応している。クライアントが一般的なブラウザそのものである場合は、このような系はクライアントに変更をかけることなく、サーバサイドでビジネス ロ

ジックの変更ができ、また各種ブラウザがクライアントとして使えるので一般的である。

サーブレットの位置付けを、よりソフトウエア的に説明すれば次の構成図のように捉えることができよう。サーブレット API は、既存の成熟したウェブサーバに乗り、複雑性や特定のサーバに実装する為のプラットフォーム固有の振る舞いを遮蔽する。各サーブレットは、API が提供する標準化されたクラスを使用する。前述のとおりサーブレットは一回ロードされると意図的に終了させない限りメモリ上に駐在する。API からの service(), doGet(), doPost()などのメソッドが、複数のクライアントからの要求に対応して、並行処理のためのスレッドを提供する。各サーブレットが共通資源にアクセスするときは、スレッド フリーなプログラムを書かねばならない。この図のように、外部データベースに対しては、ヘルパとしてその為のサーブレットを用意するのが一般的であろう。



サーブレットはまた、コンテナ(Container)により管理されたウェブ コンポネントでもある。サーブレット コンテナとは何であろうか？サーブレット コンテナは、ウェブサーバ（あるいはアプリケーション サーバ）とともに、要求と応答の設定、MIME ベースのクライアントからの要求のデコード、及びクライアントへの MIME ベースの応答へのフォーマット化からなるネットワーク サービスを提供するものである。サーブレット コンテナは、複数のサーブレットを各々のライフサイクルにわたって包含し、管理するものである。

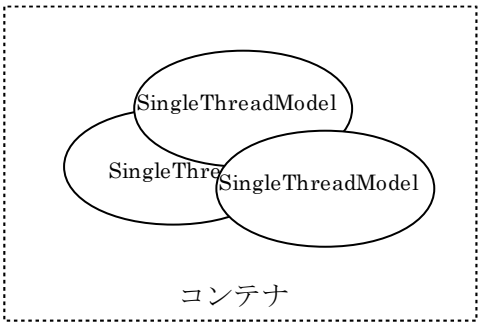
事例として、ブラウザのごときクライアント プログラムがウェブサーバを HTTP の Request でアクセスしたときを考えてみよう。この要求はウェブ サーバで処理され、サーブレット コンテナに引き渡される。サーブレット コンテナは、その内部設定をもとにどのサーブレットに該当するか決定し、要求と応答のオブジェクトとともに該サーブレットを呼び出す。サーブレット コンテナは、ホストのウェブ サーバと同じプロセスの中で実行できるし、同じホスト上の別のプロセスのなかで実行できるし、またはウェブ サーバとは別の、要求を処理するホスト上で走らせることも可能である。

サーブレットはこの要求オブジェクトを、リモート ユーザは誰か、この要求の部分として送られてきた HTML フォームパラメタはなにか、及びその他の関連するデータを取得するのに使用する。これにより、サーブレットはプログラミングされたロジックを実行し、該クライアントに送り返すデータを作成できる。このデータは応答オブジェクトを介してクライアントに返される。該要求に対応した処理をサーブレットが終了したら、サーブレットコンテナは応答が正しくクライアントに送られ応答オブジェクトが消去されたことを確認し、制御をホストのウェブ サーバにかえす。

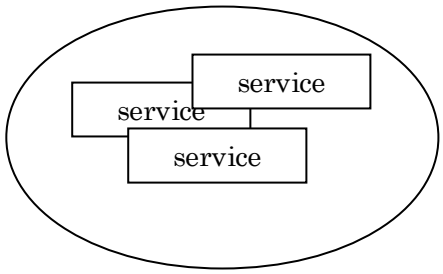
サーブレットのスレッド処理

ところでサーブレットにおいて並行処理はどのように扱われるのであろうか？

複数のクライアントからの要求の同時処理には、下図のように SingleThreadModel を実装してサーブレットのインスタンスのプールが使われるようにする方法と、あるサーブレットのインスタンスの service メソッドが複数のスレッドから呼ばれる方法とがあり得る。楕円で記したのがサーブレットのオブジェクトである。SingleThreadModel を実装したサーブレットでは、ひとつの service メソッドにはただひとつのスレッドしか存在しない。そうでない場合には複数のスレッドが service メソッドを走ることになる。この



コンテナ



複数のスレッドが service メソッドを呼ぶ

場合はこのメソッドをスレッドフリーとなるよう記述しなければならない。

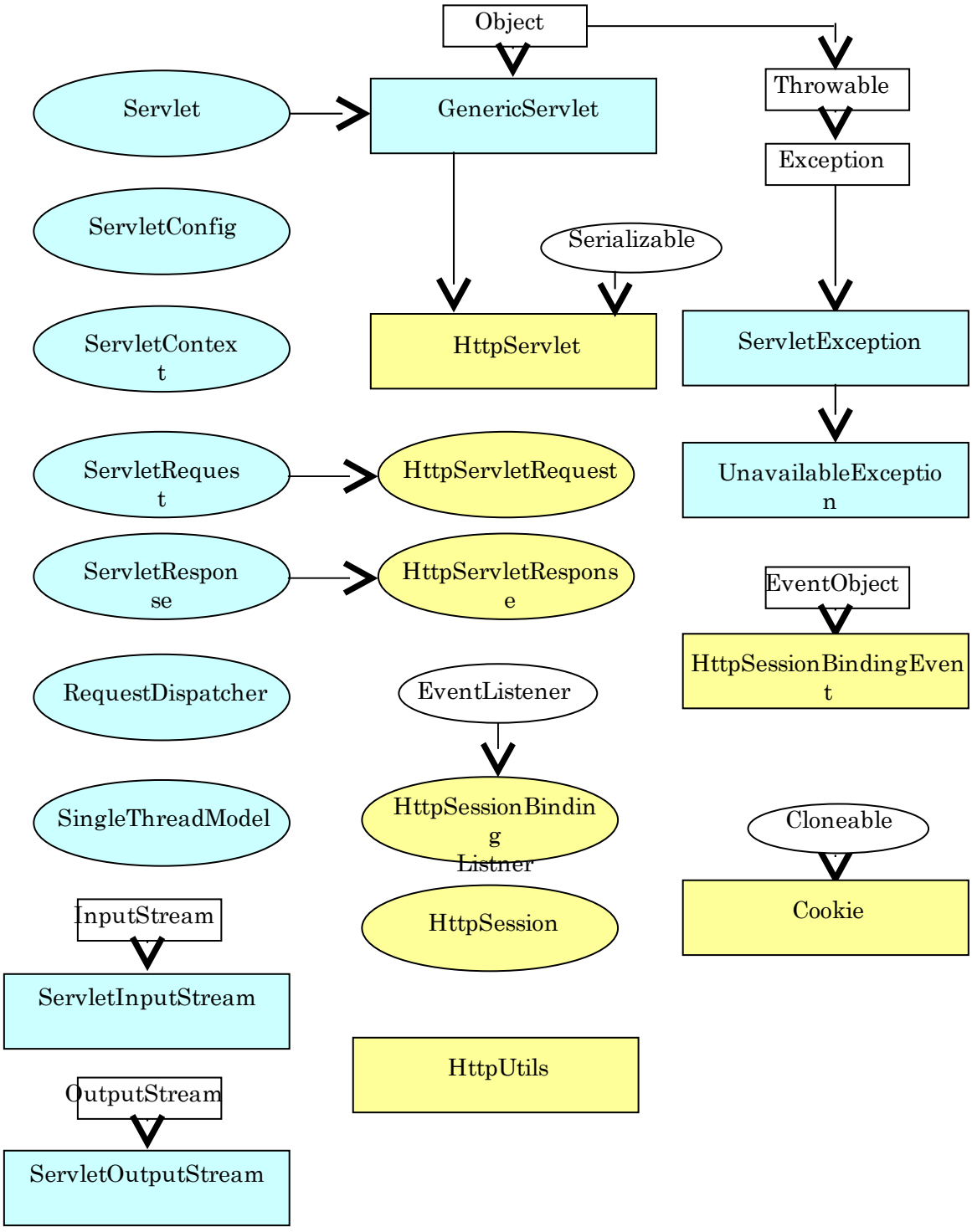
ところでスレッドは誰がどのように生成・管理しているのだろうか？そのメカニズムは、Web サーバの要求処理スレッドが行う一連の過程の一例を知れば理解できる。

1. スレッドは TCP ソケット(ポート 80)を調べ、クライアントが接続するまで待機状態となる。
2. クライアントが接続するとスレッドは要求処理のため待機状態から目覚める。
3. スレッドは要求の情報をストアするための構造体を作成する為の HTTP 受け取りを行う。
4. スレッドは誰が(file handler, cgi handler, webserver API plugin, servlet, etc..)該要求を処理するかを決定する。
5. スレッドは web サーバの API 層に入る。
6. スレッドはサーブレット エンジンに入る。
7. サーブレット エンジンはサーブレットのための要求と応答オブジェクトを組み立てる。
8. サーブレット エンジンはサーブレットを呼び出し、要求と応答のオブジェクトを渡す。
9. スレッドは該サーブレットの service メソッドに入る。
10. スレッドは該サーブレットの service メソッドを出る。
11. サーブレット エンジンは要求と応答のオブジェクトをクリーンアップする。（応答は web サーバの API 層に返す）
12. スレッドはサーブレット エンジンを出、web サーバの API 層に戻る。
13. web サーバの API 層は応答をクライアントに返す。
14. スレッドは web サーバの API 層を出る。
15. web サーバはスレッドをクリーンアップする。
16. ステップ 1 に戻る。

サーブレット API の構成

Servlet 仕様書の第 2.2 版の API を図示すると、下図のようになる。この図の見方は、楕円がインターフェイス、矩形がクラスである。矢印は拡張またはインターフェイスの実装を意味する。小さな楕円と矩形は外部のパッケージに属する。水色が Javax.servlet パッケージに、黄色が javax.servlet.http パッケージに属する。

サーブレットの API の中心となるのが抽象インターフェイスである Servlet である。アプリケーションは、これを実装した GenericServlet を拡張するか、HTTP ベースのサーブレットであれば HttpServlet を拡張させて各サービスに対応したクラスを作成する。


Object
Servlet
GenericServlet
Throwable
Exception
ServletConfig
Serializable
ServletContext
HttpServlet
ServletException
ServletRequest
HttpServletRequest
UnavailableException
ServletResponse
HttpServletResponse
EventObject
HttpSessionBindingEvent
RequestDispatcher
EventListener
SingleThreadModel
HttpSessionBinding Listner
Cloneable
Cookie
InputStream
HttpSession
ServletInputStream
HttpUtils
OutputStream
ServletOutputStream

Java™ サーブレット仕様書　v2.2
(Java™ Servlet Specification, v2.2)

第 0 章　　前書き (Preface)

本ドキュメントはJava™のサーブレット仕様書第 2.2 版は、Java サーブレット API の仕様書である。この仕様書に加えて、Java サーブレット API にはJavadoc 形式のドキュメンテーション（Java サーブレット API 参照第 2.2 版と呼ぶ）と参照となる実装ソフトウエア(Tomcat)が公開されており、以下の URL から取得できる。

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

この参照となる実装ソフトウエアは、動作のベンチマークとしても使える。もし両者の動作に相違が生じたときは、問題解決の順は、本仕様書、Java サーブレット API 参照第 2.2 版、そして最後にこの参照となる実装ソフトウエアである。

0.1 本仕様書の対象となる読者 (Who Should Read This Specification)

本ドキュメントは以下の読者を対象としている：

- この仕様書に適合したサーブレット エンジンを提供しようとするウェブ サーバとアプリケーション サーバの企業
- この仕様書に適合したウェブ アプリケーションを生成しようとするウェブ オーサリング ツールの開発者
- サーブレットのテクノロジの根底にあるメカニズムを理解しようとする高度なサーブレットの作成者

この仕様書はユーザーズ ガイドではなく、そのような目的に使われることを意図していないことに注意頂きたい。

0.2 API 参照ドキュメント (API Reference)

Java サーブレット API 参照ドキュメント第 2.2 版は、サーブレット API を構成する全てのインターフェイス、クラス、例外、およびメソッドの完全な記述である。この仕様書をとおして簡略化したメソッドの使用法が示されている。完全なメソッドの使用法に関してはこの API 参照を参照されたい。

0.3 他の Java™ プラットフォーム仕様書 (Other Java™ Platform Specifications)

本仕様書をとおして以下の Java の API 仕様書が頻繁に参照されている。

- Java2 プラットフォーム エンタープライズ エディション第 1.2 版(J2EE)
- Java サーバ・ページ第 1.1 版(JSP)
- Java ネーミングおよびディレクトリ インターフェイス(JNDI)

これらの仕様書は Java2 エンタープライズ エディションの Web サイトから見出すことが出来る：

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

0.4 その他の重要な参照仕様書 (Other Important References)

本サーブレット API をサポートするサーブレット API とエンジンの開発と実装のために、以下のインターネット(IETF)の仕様書が関連する情報源となる。：

- RFC 1630 Uniform Resource Identifiers (URI)
- RFC 1738 Uniform Resource Locators (URL)
- RFC 1808 Relative Uniform Resource Locators
- RFC 1945 Hypertext Transfer Protocol (HTTP/1.0)
- RFC 2045 MIME Part One: Format of Internet Message Bodies
- RFC 2046 MIME Part Two: Media Types
- RFC 2047 MIME Part Three: Message Header Extensions for non-ASCII text
- RFC 2048 MIME Part Four: Registration Procedures
- RFC 2049 MIME Part Five: Conformance Criteria and Examples

- RFC 2109　HTTP State Management Mechanism
- RFC 2145　Use and Implementation of HTTP Version Numbers
- RFC 2324　Hypertext Coffee Pot Control Protocol (HTCPCP/1.0)[1]
- RFC 2616　Hypertext Transfer Protocol (HTTP/1.1)
- RFC 2617　HTTP Authentication: Basic and Digest Authentication

これらの RFC のオンライン版は次の URL から見出すことが出来る：
http://www.rfc-editor.org/
WWW コンソーシアム(http://www.w3c.org/)は、この仕様書とその実装に影響を与える HTTP 関連情報の最終的な原点である。

本仕様書のなかで記載されているデプロイメント ディスクリプタ(Deployment Descriptors)に拡張可能マークアップ言語(XML)が使われている。XML に関するより詳細は以下の Web サイトから見つけることができる。
http://java.sun.com
http://www.xml.org/

注 1　この HTCPCP の RFC に記されているほとんどのコンセプトは、よく設計されたウェブ サーバの全てに関連しているものの、本参照 RFC の大部分は皮肉が記されている。

0.5 フィードバック (Providing Feedback)

Java コミュニティのプロセスが成功するか否かは、読者がこのコミュニティへ参加してくれるかに依存している。本仕様書に関する如何なるそして全てのフィードバックを歓迎する。コメントは次のアドレスにメールされたい。
Servletapi-feedback@eng.sun.com
われわれが受信するフィードバックの量が多いため、通常直接担当技術者から解答されないことに注意いただきたい。しかしながら、頂いた各そして全てのコメントはこの仕様書のチームによって読まれ、検討されかつ保管される。

0.6 謝辞 (Acknowledgements)

以下の諸氏は本仕様書の進化に貴重なインプットを頂いた。諸氏の名前はアルファベット順以外のなにものでもない。
Anselm Baird-Smith, Elias Bayeh, Vince Bonfanti, Larry Cable, Robert Clark, Daniel Coward, Satish Dharmaraj, Jim Driscoll, Shel Finkelstein, Mark Hapner, Jason Hunter, Rod McChesney, Stefano Mazzocchi, Craig McClanahan, Adam Messinger, Ron Monzillo, Vivek Nagar, Kevin Osborn, Bob Pasker, Eduardo Pelegri-Lopart, Harish Prabandham, Bill Shannon, Jon S. Stevens, James Todd, Spike Washburn, Alan Williamson
Connie Weiss, Jeff Jackson, Mala Chandra の皆さんは、Sun にあってサーブレットの支援と推進のための管理を驚異的になしとげた。

この仕様書は進行中のものであり、Sun および他の参加企業の多くのグループからの多岐にわたる貢献によるものである。とりわけ以下の企業やグループがこのサーブレット仕様書の開発過程で多大の協力を頂いた。
The Apache Developer Community, Art Technology Group, BEA Weblogic, Clear Ink, IBM, Gefion Software, Live Software, Netscape Communications, New Atlanta Communications, Oracle

そして継続した仕様書の再吟味の過程は極めて貴重である。われわれのパートナーや一般双方から頂いた多くのコメントは、この仕様書の定義づけと革新の助けとなった。貢献したフィードバックを頂いた全てのかたがたに感謝する。

第 1 章　　概説 (Overview)

1.1 サーブレットとは何か？ (What is a Servlet?)

サーブレットはウェブのコンポネントであり、コンテナにより管理され、ダイナミックなコンテンツを生成するものである。サーブレットはコンパクトで、プラットフォームに依存しないクラスの集まりで、アーキテクチャに非依存のバイトコードにコンパイルされ、ウェブサーバにダイナミックにロードされ実行される。サーブレットはサーブレット コンテナに実装された要求(Request)／応答(Response)のパラダイムを介してウェブ・クライアントとかかわりあう。この要求／応答モデルは、ハイパーテキスト転送プロトコル(HTTP: Hyper Text Transfer Protocol)の動作をベースとしている。

1.2 サーブレット コンテナとは何か？ (What is a Servlet Container?)

サーブレットコンテナは、ウェブサーバまたはアプリケーション サーバとともになって、要求と応答がセットされるネットワーク サービスを行い、MIME ベースの要求をデコードし、MIME ベースの応答のフォーマットを作成する。サーブレット コンテナはまた、サーブレットをそのライフサイクルにわたって包含しかつ管理する。

サーブレット コンテナはホストのウェブ サーバに組み込まれるか、そのサーバのネーティブの拡張を介してウェブ サーバにアドオンのコンポネントとしてインストールされる。サーブレット コンテナはまた、ウェブ対応アプリケーション サーバのなかに組み込み、あるいはインストールされ得る。

全てのサーブレットコンテナは要求と応答のプロトコルとして HTTP をサポートしなければならないが、たとえば HTTPS(HTTP over SSL)のような付加的な要求／応答ベースのプロトコルにも対応しても良い。コンテナが実装しなければならない HTTP の最低要求バージョンは HTTP/1.0 である。HTTP/1.1 仕様も実装したコンテナを強く推奨する。

サーブレット コンテナは、サーブレットが実行する環境上でのセキュリティの制約を課してもよい。Java 2 プラットフォーム スタンダード エディション 1.2(J2SE)または 、

Java 2 プラットフォーム エンタープライズ エディション 1.2(J2EE)環境においては 、Java 2 プラットフォームで定義された許可のアーキテクチャを使ってこれらの制約が課されなければならない。例えば、ハイエンドのアプリケーション サーバでは、このコンテナの他のコンポネントに絶対マイナスのインパクトを与えないように、Thread オブジェクトの生成のようなある種のアクションには制限を課しても良い。

1.3 事例 (An Example)

ウェブのブラウザの如きクライアント プログラムがウェブ サーバをアクセスして HTTP 要求を行う。この要求はウェブ サーバによって処理され、サーブレット コンテナに引き渡される。サーブレット コンテナは、内部の構成に基づき呼び出すべきサーブレットを確定し、要求と応答を表現するオブジェクトを付けてこれを呼び出す。サーブレット コンテナはホストのウェブ サーバと同じプロセスの中で、あるいは同じホストの別のプロセスの中で、あるいはウェブ サーバと別のホスト上で実行させ要求を処理することができる。

サーブレットはこの要求のオブジェクトを使ってリモート ユーザは誰か、この要求の部分としてどのような HTML のフォーム パラメタが送られてきたか、あるいは他の関連データを見出すことが出来る。サーブレットは次にプログラムされた何らかのビジネスロジックを実行し、クライアントに送り返すべきデータを作り出す。サーブレットは次にこのデータを応答のオブジェクトを使ってクライアントに送り返す。

このサーブレットが概要求の処理を終了したら、サーブレット コンテナは応答が正しく吐き出されたことを確認して、制御をホストのウェブ サーバに戻す。

1.4 サーブレットと他のテクノロジとの比較 (Comparing Servlets with Other Technologies)

機能的にはサーブレットは共通ゲートウェイ インターフェイス(CGI: Common Gateway Interface)とネットスケープ サーバ API(NSAPI: Netscape Server API)や Apache モジュールの如き独自のサーバ拡張機能との間に属するといえる。

サーブレットはサーバ拡張のメカニズムと比較して次のような特徴がある：

- 異なったプロセス モデルを使っているので、CGI スクリプトよりは一般にはるかに高速である
- 多くのウェブサーバが対応している標準の API を使っている
- 開発の容易性とプラットフォームからの独立性を含む Java プログラミング言語の全部の優位性を保持する
- Java プラットフォーム用に使える大量の API セットにアクセスできる

1.5 Java 2 プラットフォーム エンタープライズ版との関係 (Relationship to Java 2 Platform Enterprise Edition)

サーブレット API は、Java 2 エンタープライズ エディション v1.2[1]が要求している API である。J2EE 仕様には、J2EE 環境で実行するサーブレット コンテナと、それに使われるサーブレットに対する追加的な要求が記されている。

注 1： 下記の URL から取得できる Java 2 エンタープライズ エディションの仕様書を参照のこと。
http://java.sun.com/j2ee/

1.6 分散可能サーブレット コンテナ (Distributable Servlet Containers)

本仕様書の今回の版で新規のものとして、ウェブ アプリケーションに *distributable*（分散可能）としてマークできるようになったことがある。この表示により、同一ホストあるいは複数のホスト上で実行している複数の Java バーチャル マシンをまたがるウェブ アプリケーションに、サーブレット コンテナの供給者がこのサーブレットを使えるようになる。分散可能なアプリケーションをサポートするコンテナがクラスタリングとかフェイルオーバとかの特徴を実装できるように、分散可能としてマークされたアプリケーションにはいくつかの制約に従わねばならない。

スケーラビリティ、クラスタリング、およびフェイルオーバが許容される（例えば J2EE 実装対応）高性能の環境下で走らねばならないような全てのウェブ アプリケーション

は、分散可能なウェブ アプリケーションとして書かれねばならない。これによりアプリケーションはこれらの機能を持つサーバの利点を最大に活かすことが出来る。このようなサーバに非分散可能なアプリケーションを採用してしまうと、そのようなサーバに与えられている特長を十分に活かすことが出来なくなる。

1.7 2.1 版からの変更 (Changes Since Version 2.1)

第 2.1 版から本仕様になされた主要な変更点は以下のとおりである：

- ウェブ アプリケーションのコンセプトの導入
- ウェブ アプリケーションのアーカイブ ファイルの導入
- 応答のバッファリングの導入
- 分散可能サーブレットの導入
- RequestDispatcher の名前による取得の可能化
- RequestDispatcher の相対パスを使った取得の可能性
- 国際化の改善
- サーブレット エンジンの言葉の意味の明確化

以下の変更が API になされている：

- getServletName メソッドを ServletConfig インターフェイスに追加し、サーブレットに名前を持たせ、必要ならシステムがそれを知ることを可能とした
- ServletContext インターフェイスに getInitParameter と getParameterNames メソッドを追加し、初期化パラメタがアプリケーション レベルで設定でき、このアプリケーションに含まれる全てのサーブレットで共有できるようにした
- ServletRequest インターフェイスに getLocale メソッドを追加し、クライアントがどの地域にいるかの決定に役立てる
- ServletRequest インターフェイスに isSecure メソッドを追加、概要求が HTTPS のようなセキュアなトランスポート経由で送られてきたかどうかを示すようにした
- 既存のコンストラクタのシグネチュアではデベロッパでの混乱が生じたため 、UnavailableException のコンストラクト法を置換えた。これらのコンストラクタはよりシンプルはシグネチュアに置換えられた
- HttpServletRequest インターフェイスに getHeaders メソッドを追加、特定の名前のを持つ全てのヘッダをその要求から取り出せるようにした

- HttpServletRequest インターフェイスに getContextPath メソッドを追加、ウェブアプリケーションに関連した要求パスの一部が取得できるようにした
- IsUserInRole と GetUserPrinciple メソッドを HttpServletRequest インターフェイスに追加し、サーブレットが抽象ロール ベースの認証を使えるようにした
- HttpServletResponse インターフェイスに addHeader、addIntHeader および addDateHeader メソッドを追加、同じヘッダ名で複数のヘッダが生成できるようにした
- HttpSession インターフェイスに getAttribute、getAttributeNames、setAttribute および removeAttribute メソッドを追加、API のネーミング規約を改良した。本変更の一環として getValue、betValueNames、setValue および removeValue メソッドは使用不可(deprecated)とした

加えて、本仕様書では数多くの不明確な個所の明確化がなされている。

第 2 章　　使用されている用語 (Terms Used)

以下の用語は、本仕様書のなかで広く使われている。

2.1　基本用語 (Basic Terms)

2.1.1　ユニフォーム リソース ロケータ（URL: Uniform Resource Locators）

ユニフォーム リソース ロケータ（URL)は、ネットワークを介して取得可能な資源を表現するコンパクトな文字列である。URL で表現されたある資源がアクセスされたとき、そのリソース上では種種の操作が実行され得る。[1]　URL は**ユニフォーム リソース識別子**(URI: Uniform resource Identifier)のひとつの形式である。URL は一般に次の形式をとる：
<プロトコル>://<サーバ名>[:ポート]/<URL パス>[？<クエリ文字列>]

例えば：
http://java.sun.com/products/servlet/index.html
https://javashop.sun.com/purchase
HTTP ベースの URL においては、’/’ 文字は URL の URL パスの中のパスの階層的構造を区切るために予約されている。サーバは階層構造の意味を知る責任を持つ。URL パスと付与済みファイル システムのパスとは対応は無い。

注 1:　RFC 1738 参照

2.1.2　サーブレット定義（Servlet Definition）

サーブレット定義とは Servlet インターフェイスを実装した完全な形式のクラス名(FQCN: Fully Qualified Class Name)に連結した固有の名前である。初期化パラメタのセットはサーブレット定義に連結し得る。

2.1.3　サーブレット マッピング (Servlet Mapping)

サーブレット マッピングとは、URL パス パタンでサーブレット コンテナにより連結された**サーブレット定義**をいう。そのパス パタンへの全ての要求はその**サーブレット定義**で連結されたサーブレットにより処理される。

2.1.4　ウェブ アプリケーション (Web Application)

ウェブ アプリケーションは、サーブレット、JavaServer ページ[2]、HTML ドキュメント、およびその他の資源の収集である。その他の資源にはイメージ ファイル、圧縮したアーカイブ、および他のデータなどが含まれ得る。ウェブ アプリケーションはひとつのアーカイブにパックしたり、あるいはオープンなディレクトリ構造のなかで存在したりしうる。

全ての互換性を持つサーブレット コンテナは、ウェブ アプリケーションを受理し、その内容をコンテナのランタイムに組み入れなければならない。このことは、そのアプリケーションを直接ウェブアプリケーションアーカイブから実行させることを意味したり、あるいは、ウェブアプリケーションの内容をそのコンテナの為のしかるべき場所に移動させることを意味する。

注 2:　http://java.sun.com/products/jsp の JavaServer ページの仕様書を参照のこと

2.1.5　ウェブ アプリケーション アーカイブ (Web Application Archive)

ウェブ アプリケーション アーカイブとは、ウェブ アプリケーションの全てのコンポネントを含む単一のファイルである。このアーカイブ ファイルは、如何なるあるいは全てのウェブ コンポネントが登録できる標準の JAR ツールを使って作成される。

ウェブ アプリケーション アーカイブ ファイルは.war 拡張子で識別される。.jar の替りに新しい拡張子が使われているのは、この.jar 拡張子がクラス ファイルのセットを含み、そのクラス パスに配置されるか、アプリケーションを開始させるのに GUI を使ってダブルクリックされるために予約されているからである。ウェブ アプリケーション アーカイブはそのような使い方に適していないので新しい拡張子がふさわしかったのである。

2.2 ロール (Roles)

サーブレットベースのアプリケーションの開発、実装、実行の過程で種々の組織が持つ行為と責任を規定する助けとする意図で以下の役割を定義したものである。

2.2.1 アプリケーション 開発者 (Application developer)

アプリケーション開発者とは、ウェブベースのアプリケーションの製作者である。**アプリケーション開発者**の成果物はそのウェブアプリケーションのサーブレットクラス、JSP ページ、HTML ページ、および支援ライブラリとファイル（イメージや圧縮アーカイブファイルなど）のセットである。**アプリケーション開発者**は、一般にアプリケーション領域でのエキスパートである。同時性（コンカレンシー）を含むプログラミングにあたってはこの開発者は、サーブレットの環境とその結果を承知しており、それに基づいてウェブ アプリケーションを作ることが要求される。

2.2.2 アプリケーション アセンブラ (Application Assembler)

アプリケーション アセンブラは開発者の仕事を引継ぎ、それが実装可能なものとする。**アプリケーション アセンブラ**への入力は、その**ウェブ アプリケーション**のためのサーブレットクラス、JSP ページ、HTML ページおよび支援ライブラリとファイルである。**アプリケーション アセンブラ**の成果物は**ウェブ アプリケーション アーカイブ**またはオープンなディレクトリ構造の中の**ウェブ アプリケーション**である。

2.2.3 デプロイヤ (Deployer)

デプロイヤ（導入者）は**アプリケーション開発者**が提供する一つあるいは複数のアーカイブファイルあるいはディレクトリ構造をあるひとつの実行環境に導入する。実行環境にはそのサーブレットコンテナとウェブサーバが含まれる。**デプロイヤ**は開発者が宣言したすべての外部との依存性を解決しなければならない。その役割を実行するには、**デプロイヤ**はサーブレットコンテナが提供するツールを使用する。

2.2.4　システム管理者（System Administrator）

システム管理者はサーブレットコンテナとウェブサーバの設定と管理の責任を持つ。この管理者はまた、ランタイムにおける導入済みウェブ アプリケーションの健全性を監視する責も持つ。本仕様書ではシステムの管理（マネージメント）と監督（アドミニストレーション）の係わり合いについては定めていない。**管理者**は自分のタスクを達成するのに、コンテナのベンダやサーバのベンダが提供するランタイムの監視と管理ツールを使用する。

2.2.5　サーブレット コンテナ提供者（Servlet Container Provider）

サーブレット コンテナ提供者はランタイムの環境、即ちサーブレットコンテナと多分ウェブサーバとをも提供する責任を持つ。この環境の中でウェブアプリケーションと、ウェブアプリケーションを導入するのに必要なツールが走る。

コンテナ提供者が必要な専門性は、HTTP レベルのプログラミングである。本仕様書ではウェブサーバとサーブレット コンテナのインターフェイスに関しては規定はしていないので、コンテナとサーバの必要とされている機能の実装の分割は、**コンテナ提供者**に任されている。

2.3 セキュリティ事項 (Security Terms)

2.3.1　プリンシパル（Principal）

プリンシパルは、認証プロトコルで認証できるエンティティである。**プリンシパル**はプリンシパル名(principal name)で識別され、認証データ(authentication data)を使って認証される。プリンシパル名と認証データの内容とフォーマットは認証プロトコルに依存する。

2.3.2　セキュリティ ポリシー ドメイン（Security Policy Domain）

セキュリティ ポリシー ドメインはセキュリティ サービスの管理者によってセキュリティ ポリシーが定義され施行される範囲をいう。**セキュリティ ポリシー ドメイン**はまたレルムとも呼ばれる。

2.3.3 セキュリティ テクノロジ ドメイン (Security Technology Domain)

セキュリ ティテクノロジ ドメインは、カーベロス(Kerberos)のような同じセキュリティのメカニズムがセキュリティのポリシーを実行するのに使われている範囲をいう。単一の**テクノロジ ドメイン**の中に複数の**ポリシー ドメイン**が存在し得る。

2.3.4 ロール (Role)

ロールとはアプリケーションにおいて**開発者**が使う抽象化された用語である。**セキュリティポリシー ドメイン内**にて**デプロイヤ**がユーザあるいはユーザのグループにマッピングできるものである。

第 3 章　　Servlet インターフェイス (The Servlet Interface)

この Servlet インターフェイスは、サーブレットの API の中心となる抽象メソッドである。総てのサーブレットはこのインターフェイスを直接、あるいは間接的に実装する 。Servlet を実装したクラスは、GenericServlet と HttpServlet の二つである。もっとも一般的なのは、プログラマは自分のサーブレットを作成するのに HttpServlet を拡張する。

3.1 要求処理メソッド (Request Handling Methods)

Servlet インターフェイスには、要求処理のためのメソッドとして service メソッドが定義されている。サーブレットコンテナがサーブレットのインスタンスに引き渡す各要求毎にこのメソッドが呼び出される。いつの時点においても、この service メソッドのなかでは複数の要求スレッドが実行していても良い。

3.1.1　HTTP 固有の要求処理メソッド（HTTP Specific Request Handling Methods）

抽象サブクラスである HttpServlet では、HTTP ベースの要求処理の便をはかって service メソッドが自動的に呼び出す幾つかのメソッドが追加されている。即ち：

- HTTP GET を処理する doGet
- HTTP POST を処理する doPost
- HTTP PUT を処理する doPut
- HTTP DELETE を処理する doDelete
- HTTP HEAD を処理する doHead
- HTTP OPTIONS を処理する doOptions
- HTTP TRACE を処理する doTrace

通常 HTTP ベースのサーブレットを開発する場合は、doGet と doPost の二つのメソッドにのみ集中すれば良いはずである。のこりのメソッドは、HTTP プログラミングに熟知したプログラマのための上級メソッドといえる。

doPut と doDelete は、サーブレット開発者がこれらの機能を持つ HTTP/1.1 クライアン

トに対応させる為のものである。HttpServlet の doHead メソッドは doGet メソッドが実行するものの特殊化したメソッドであるが、クライアントには doGet メソッドで作られたヘッダ部分のみを送り返す。DoOptions メソッドは、どの HTTP メソッドが直接該サーブレットでサポートされているかを知り、この情報をクライアントに返す。doTrace メソッドは、TRACE 要求に送られている総てのヘッダを附した応答を発生させる。

HTTP/1.0 をサポートするコンテナにおいては、HTTP/1.0 では
PUT、DELETE、OPTIONS、TRACE などが定義されていないので 、
doGet、doHead、doPost の 3 つのメソッドのみが使用される。

3.1.2 条件付 GET のサポート（Conditional GET Support）

条件付 GET に対応するために、HttpServlet インターフェイスでは getLastModified メソッドが定義されている。条件付 GET 動作とは、コンテント ボディ部分が、指定された時刻移行に変更された場合のみに送信するように指定したヘッダが付加された HTTP GET メソッドで、クライアントが資源を要求してきた場合のことをいう。

DoGet メソッドを実装、要求ごとに必ずしも変化しないようなコンテンツを出すようなサーブレットでは、ネットワーク資源の有効活用のためこのメソッドを実装すべきである。

3.2 インスタンスの数 (Number of Instances)

デフォルトとしては、ひとつのコンテナ内の、サーブレット定義（Servlet インターフェイスを実装したクラス名と、それに関連する初期化パラメタ）あたりのサーブレット クラスのインスタンスは唯一つでなければならない。

SingleThreadModel インターフェイスを実装したサーブレットの場合は、サーブレットコンテナは該サーブレットのインスタンスを複数インスタンス化し、一方では要求を単一のインスタンスにシリアル化しつつ、大量の要求の重負荷を処理する。

組込み記述子(deployment descriptor)に *distributable* としてマークし、アプリケーションの一部としてサーブレットが導入されたときは、コンテナには、Java 仮想マシン(VM)

あたりのサーブレット定義あたりのサーブレットクラスのインスタンスはひとつである。もしサーブレットが分散可能な Web アプリケーションであるとともに 、SingleThreadModel インターフェイスを実装しているときは、該コンテナの各仮想マシン内では、該コンテナは該サーブレットの複数のインスタンスをインスタンス化させても良い。

3.2.1 SingleThreadMode に関する注記 (Note about SingleThreadMode)

SingleThreadModel インターフェイスを実装すれば、同一時刻においては単一のスレッドが与えられたサーブレットのインスタンスの service メソッドを通して実行中であることが保証される。この保証はサーブレットのインスタンスに対してのみ適用されることに注意のこと。HttpSession のインスタンスの如く、同一時刻において複数のサーブレットのインスタンスにアクセス可能なオブジェクトに対しては、SingleThreadModel を実装したものも含めてどの時刻においても複数のサーブレットがアクセス可能である。

3.3 サーブレットのライフ サイクル (Servlet Life Cycle)

メモリへのロード、インスタンス化と初期化、クライアントからの要求処理、サービスからの除去といった、きちんと定義されたライフ サイクルを通してサーブレットは管理されている。このライフ サイクルは API の javax.servlet.Servlet インターフェイスにある init、service、そして destroy メソッドで記述されており、総てのサーブレットが直接的に、あるいは GenericServlet または HttpServlet 抽象クラス経由で間接的に、実装しなければならない。

3.3.1 ロードとインスタンス化 (Loading and Instantiation)

サーブレットのロードとインスタンス化はサーブレット コンテナの責任である。ロードとインスタンス化はエンジンが開始したとき生じさせ得るし、コンテナが要求をサービスするためにそのサーブレットが必要になったと判断するまでこれを保留させることも可能である。

最初に、そのサーブレット型のクラスの所在をサーブレット コンテナが把握せねばなら

ない。必要とあらば、サーブレット コンテナはローカルのファイル システム、リモートのファイル システム、あるいは他のネットワーク サービスから通常の Java クラスロード機能を使ってサーブレットをロードする。

コンテナがその Servlet クラスをロードしたら、該クラスを使うためにそのクラスのオブジェクト インスタンスにインスタンス化する。

あるサーブレット コンテナ内には、与えられた Servlet クラスの複数のインスタンスが存在し得ることに注意。このことは、例えばある特定のクラスを異なった初期化パラメタで使う、複数のサーブレット定義の場合が該当する。また、サーブレットが SingleThreadModel インターフェイスを実装し、かつコンテナがその複数のインスタンスをプールする場合にも生じ得る。

3.3.2 初期化（Initialization）

サーブレットのオブジェクトがロードされインスタンス化されたら、クライアントからの要求受付可能となる前に、コンテナはこのサーブレットの初期化をしなければならない。初期化は、サーブレットが継続的に蓄積されているコンフィギュレーション データの読出し、手間のかかるリソースの初期化（たとえば JDBC ベースの接続）、あるいはその他の一時的な処理を実行する為に用意されているものである。コンテナは、Servlet インターフェイスの init メソッドを、ServletConfig インターフェイスを実装した特定の（サーブレット定義あたり）オブジェクトを使って呼びだすことでそのサーブレットの初期化をおこなう。このコンフィギュレーションオブジェクトにより、サーブレットはコンテナのコンフィギュレーション情報から名前-値の初期化パラメタをアクセスすることができる。コンフィギュレーションオブジェクトはまた、そのなかでサーブレットが走っているランタイムの環境を記述した ServletContext インターフェイスを実装したオブジェクトに、サーブレットがアクセス可能ならしめるものでもある。

3.3.2.1 初期化におけるエラーの条件（Error Conditions on Initialization）

初期化中、サーブレット インスタンスは UnavailebleException または ServletException の例外を発生させることでアクティブなサービスの状態に移行できないことを通知できる。この種の例外をサーブレット インスタンスが発生したら、これをサービス可能な状態に持っていってはいけないし、このインスタンスは直ちにコンテナ

によって開放さればならない。この場合初期化が成功しているとは考えられないので 、destroy メソッドは呼ばれない。

不成功のサーブレットを開放したら、コンテナはどの時点でも新たなインスタンスをインスタンス化し初期化して良い。この規則の唯一つの例外は、失敗したサーブレットからアベーラブルでない最小時間を示す UnavailableException 例外が発生されたときで、このときは、新しいサーブレットのインスタンスの作成と初期化をこの最小時間が経過するまで待たねばならない。

3.3.2.2　ツールの考察（Tool Considerations）

ツールがウェブ アプリケーションをロードし診断するとき、このアプリケーションのメンバークラスをロードし診断して良い。これがスタティックな初期化メソッドの実行を開始させる。この機能のゆえ、ソフトウエア開発者は Servlet インターフェイスの init メソッドが呼ばれていない限り、サーブレットがアクティブなコンテナのランタイム下にあると仮定してはいけない。例えば、そのスタティックな初期化メソッドが実行呼出されたときは、サーブレットはデータベースあるいはエンタープライズ JavaBeans コンポネント アーキテクチャ コンテナとの接続を開始したりしてはならないことを意味する。

3.3.3　要求の処理（Request Handling）

サーブレットの初期化が無事終了したら、サーブレット コンテナはこれをクライアントからの要求処理のために使用して良い。各要求は ServletRequest 型の要求オブジェクトとして渡され、これによりサーブレットは用意された ServletResponse 型のオブジェクトで応答を作成することができる。これらのオブジェクトは Servlet インターフェイスの service メソッドのパラメタとして渡される。HTTP 要求の場合は、コンテナは HttpServiceRequest と HttpServiceResponse を実装した要求と応答のオブジェクトをわたさねばならない。

サーブレットのインスタンスが作成されサーブレット コンテナにサービス可能な状態におかれても、そのライフタイムにわたって要求を処理しないサーブレットもあることに注意されたい。

3.3.3.1　マルチスレッドの問題（Multithreading Issues）

クライアントからの要求をサービスしている最中は何時でも、サーブレット コンテナは複数のクライアントからの複数の要求を該サーブレットの service メソッドを介して送りこんで良い。このことは、開発者はサーブレットがきちんと並行処理に対応するように注意しなければならないことを意味する。

開発者がこのデフォルトの機能を望まないなら、SingleThreadModel インターフェイスを実装したサーブレットをプログラムして良い。このインターフェイスを実装することにより、任意の時間においては唯一つの要求スレッドのみが service メソッドを使っていることが保証される。サーブレット コンテナは、あるサーブレットへの要求を直列化するか、あるいはサーブレットのインスタンスをプールすることでこれを保証させる。サーブレットが分散可能（distributable）とマークしたアプリケーションの一部である場合は、コンテナはこのアプリケーションが分散される各 VM にサーブレットのインスタンスのプールを保持して良い。

開発者が service メソッド（あるいは HttpServlet 抽象クラスの service メソッドから要求及び応答が回送される doGet や doPost などのメソッド）を synchronized キーワードで定義した場合には、サーブレットコンテナは実行中の Java ランタイムの必要に応じて要求を直列化する。しかしながら、コンテナは SingleThreadModel を実装したサーブレットに対して行うと同じようにインスタンスのプールを作ってはならない。プログラマが service メソッドあるいは doGet や doPost などの HttpServlet のメソッドに synchronized キーワードを使わないことを強く推奨する。

3.3.3.2　要求処理中の例外（Exceptions During Request Handling）

サーブレットは要求のサービス中に ServletException あるいは UnavailableException のいずれかをスローさせて良い。ServletException は、要求の処理中になんらかのエラーが発生し、コンテナはこの要求のクリーンアップの為に適切な処置をとるべきであることを通知するものである。UnavailableException はこの要求をサーブレットは一時的あるいは永久に処理できないことを通知する。

UnavailableException により永久的な非アベーラビリティが通知された場合には、サーブレット コンテナはこのサーブレットをサービスから除去し、その destroy メソッドを呼び、このサーブレット インスタンスを除去しなければならない。

UnavailableException により一次的な非アベーラビリティが通知された場合には、コンテナは一時的にアベーラブルでない期間中は要求を回さないように選択できる。この期間は、コンテナは拒絶された要求に対して SERVICE_UNAVAILBLE(503)応答を何時アベーラブルになるかを示す Retry-After ヘッダを附して返さなければならない。コンテナは一時的と恒久的の区別を無視して、すべての UnavailableException を恒久的とみなしてこのサーブレットをサービスから外す選択をしてもよい。

3.3.3.3　スレッド安全性（Thread Safety）

開発者は、要求と応答のオブジェクトの実装はスレッド セーフを保証していないことに注意しなければならない。このことは、これらのオブジェクトは要求処理スレッドのスコープ内でのみしか使ってはならないことを意味する。この要求及び応答オブジェクトへの参照は、結果が不定となるので、他のスレッドで実行中のオブジェクトには許されない。

3.3.4　サービスの終了（End of Service）

サーブレット コンテナは、何時もサーブレットをロードしたままでいるように要求されているわけではない。サーブレットのインスタンスはサーブレットコンテナにはたったの 1 ミリ秒、あるいはサーブレットコンテナのライフタイム中（数日、数ヶ月、あるいは数年間）、あるいはその間のどんな期間でもアクティブな状態に維持され得る。

サーブレット コンテナがサーブレットをサービスから外さねばならなくなったとき（例えばコンテナがメモリ資源の節約が必要になったり、自分自身がシャットダウン中のとき）、サーブレットに対して各サーブレットが使用中のリソースを開放し、継続的な状態の保管の為の猶予をあたえねばならない。その為に、サーブレット コンテナは Servlet インターフェイスの destroy メソッドを呼び出す。

サーブレット コンテナが destroy メソッドを呼べるようになる前に、該サーブレットの service メソッドで現在走っているどのスレッドに対してもこれを終了させるか、あるいはサーバが定めたタイムリミットを超過させるかのための余裕をコンテナが destroy メソッドを呼ぶ前に与えねばならない。

ひとたびあるサーブレット インスタンスの destroy メソッドが呼ばれたら、コンテナは

該サーブレットインスタンスあての要求をもはや送りこんではならない。該サーブレットが再び必要になったときは、コンテナはこのサーブレット クラスの新しいインスタンスでこれに対応しなければならない。

destroy メソッドが終了したら、サーブレット コンテナはこのサーブレット インスタンスを開放し、ガーベージ コレクションの対象としなければならない。

第 4 章　　サーブレットコンテキスト (Servlet Context)

ServletContext インターフェイスは、このサーブレットが走っている Web アプリケーション環境へのサーブレット側からの観点を定義する。ServletContext インターフェイスはまたサーブレットが、それが使えるリソースへのアクセスを可能ならしめる。かかるオブジェクトを使って、サーブレットはイベントのログをとったり、リソースへの URL の参照を取得したり、またこのコンテキストの他のサーブレットが使えるよう属性の読み書きをしたりすることができる。サーブレット コンテナに ServletContext インターフェイスを実装するのはコンテナのプロバイダである。

ServletContext は Web サーバの特定のパスのルートに置かれる。例えば、コンテキストは http://www.mycorp.com/catalog に置かれる。/catalog 要求パス（コンテキスト パス：context path として知られる）で始まるすべての要求はこのサーブレット コンテキストに回される。

ServletContext の唯一つのインスタンスが web アプリケーション内のサーブレット（複数）に参照可能である。Web アプリケーションが分散可能としている場合は、Java 仮想マシンあたり、アプリケーションあたりの ServletContext オブジェクトの唯一つのインスタンスが使用中でなければならない。

4.1 ServletContext のスコープ (Scope of a ServletContext)

ひとつのコンテナにコンテナに導入（デプロイ）された各ウェブ アプリケーションに関連づけられた ServletContext インターフェイスのインスタンスはひとつである。コンテナが複数の仮想マシンにわたって分散しているときは、VM あたりのウェブ アプリケーションあたりのインスタンスはひとつである。

ウェブ アプリケーションの一部として導入されていないサーブレットがコンテナに存在するときは、これは暗示的にデフォルトのウェブ アプリケーションの一部であるとし、デフォルトの ServletContext に含まれているものとする。分散コンテナの場合は、デフォルトの ServletContext は分散不可であり、ひとつの仮想マシン上にのみ存在しなければならない。

4.2 初期化パラメタ (Initialization Parameters)

コンテキスト初期化パラメタのセットはウェブ アプリケーションと結び付けられ 、ServletContext インターフェイスの以下のメソッドにより取得可能である。

- getInitParameter
- getInitParamterNames

初期化パラメタはアプリケーション開発者がセットアップのための情報、例えば webmaster の E-mail アドレスまたはクリチカルなデータを管理するシステムの名前などを伝達するのに使用される。

4.3 Context の属性 (Context Attributes)

サーブレットはオブジェクトの属性を名前でコンテキストにバインドすることができる。コンテキストにバインドされたいかなるオブジェクトも同じウェブ アプリケーションに属する他の総てのサーブレットからアクセスできる。ServletContext インターフェイスの以下のメソッドによりこの機能にアクセスすることができる。

- setAttribute
- getAttribute
- getAttributeNames
- removeAttribute

4.3.1 分散コンテナでのコンテキスト属性（Context Attributes in a Distributed Container）

Context 属性はそれが作成され配置された仮想マシンにローカルに存在する。これにより ServletContext が分散共用メモリとして使用されるのを防止する。分散環境で走っているサーブレット間で共用する情報が必要になったときは、その情報はセッション（第 7 章参照）、データ ベース、あるいはエンタープライズ JavaBean の形で蓄積されねば

ならない。

4.4 資源 (Resources)

ServletContext インターフェイスにより、次のメソッドを使ってたとえば HTML、GIF、JPEG ファイルといったウェブ アプリケーションの一部であるコンテンツのドキュメントの静的なドキュメント階層を直接アクセスすることが出来るようになる：

- GetResource
- getResourceAsStream

GetResource と getResourceAsStream の双方のメソッドともに、変数としてこのコンテキストのルートに相対的な資源のパスを与える String 変数を持つ。

これらのメソッドによりサーバが使用しているリポジトリに関わらず、静的なリソースのアクセスが可能となることに注意することが重要である。このドキュメントの階層は、ファイルシステム、Web アプリケーションのアーカイブファイル、リモートサーバ、あるいは他の場所に存在し得る。上記のメソッドはダイナミックなコンテンツの取得には使用されない。例えば、JavaServer Pages 仕様[1]をサポートするコンテナにおいては 、getResource(“/index.jsp”)の形式のメソッド呼び出しは JSP ソースコードを返し、処理した出力を返さない。詳細は第 8 章の「要求のディスパッチ」を参照のこと。

注 1:　JavaServer ページの仕様は http://java.sun.com/products/jsp にある。

4.5 複数のホストとサーブレット コンテキスト (Multiple Hosts and Servlet Context)

多くの Web サーバは、複数の論理ホストがサーバ上でひとつの IP アドレスを共有する機能を有している。この機能は「バーチャル ホスティング」として呼ばれるものである。サーブレット コンテナのホストがこの機能をサポートする場合は、各論理ホストはそれ自身のサーブレット コンテキストを保有するか,サーブレット コンテキスト（複数）

のセットを保有しなければならない。サーブレット コンテキストはバーチャル ホスト間で共用できない。

4.6 再ロードする場合の考察点 (Reloading Considerations)

開発が容易であるように多くのサーブレット コンテナがサーブレットの再ロード（リロード）をサポートしている。サーブレット クラス再ロードはすでに前のサーブレットコンテナの作成時にこのサーブレットをロードするために新しいクラスのローダを作成することで達成されてしまっている。このクラスローダは、他のサーブレット コンテキストで使われる他のサーブレットあるいはクラスをロードするときにつかわれるローダとは区別される。このことは好ましくない副作用として、あるサーブレット コンテキスト内のオブジェクトの参照を期待したものと異なるクラスあるいはオブジェクトにしてしまい、予期せぬ結果をもたらす可能性がある。

従って,コンテナのプロバイダはクラス再ロードの機能を実装するときは、使用されるであろう総てのサーブレット及びクラスが単一のクラスローダのスコープ内にロードされ、アプリケーションが所定のとおり振舞うことを保証しなければならない。

4.7 一時的な作業用ディレクトリ (Temporary Working Directories)

アプリケーション開発者にとっては、ローカルなファイル システムに一時的な作業領域を持つことがしばしば有用となる。すべてのサーブレット コンテナはサーブレット コンテキスト毎にプライベートな一次ディレクトリを用意して 、javax.servlet.context.tempdir なるコンテキスト属性でアクセス可能としなければならない。この属性で関連付けされたオブジェクトは java.io.file 型でなければならない。

第 5 章　　要求 (The Request)

要求オブジェクトは、クライアントからの要求の総ての情報をカプセル化する。HTTP プロトコルにおいては、この情報はクライアントからサーバへ要求のヘッダとメッセージ本体により送られる。

5.1 パラメタ (Parameters)

要求パラメタは、クライアントからサーブレット コンテナに要求の一部として送られた文字ストリングである。要求が HttpServletRequest の場合は、属性は URI クエリストリングと、加えて一般的にはクライアントがポストしたフォーム データであろうが、これらを集めたものである。これらのパラメタはサーブレット コンテナに名前－値の対(name-value pair)のセットとして蓄積される。与えられたパラメタ名には複数のパラメタ値が存在し得る。ServletRequest インターフェイスの以下のメソッドは、パラメタへのアクセスの為のものである。

- getParameter
- getParameterNames
- getParameterValues

getParameterValues メソッドはパラメタ名に対する総てのパラメタ値を含んだ String オブジェクトの配列を返す。getParameter メソッドで返された値は、何時も getParameterValues で返された String オブジェクトの配列の最初の値と等しくなければならない。

クエリ ストリングとポスト本体からの総てのフォーム データは要求パラメタのセットとして集積される。集積の順番は、クエリ ストリングがポスト本体のパラメタ データに先行する。例えば、クエリ ストリングが a=hello、ポスト本体が a=goodbye&a=world で要求が構成されていたときは、結果としてのパラメタ セットは a=(hello, goodbye, world)の順番となろう。

ポストされたフォーム データは要求の入力ストリームからのみ読み出され、以下の総ての条件に合致したときはパラメタ セットとして集積するのに使用される。

1. 該要求が HTTP または HTTPS 要求である
2. 該 HTTP メソッドが POST である
3. コンテント タイプが application/x-www-form-urlencoded である
4. サーブレットが該要求オブジェクトの getParameter ファミリのいずれかのメソッドを呼んでいる

getParameter ファミリのいずれかのメソッドをも読んでいないとき、あるいは上記の全条件が満足されていないときであっても、サーブレットが要求の入力ストリームを通してポスト データを取得できなければならない。

つまりサーブレットは、要求データを上記のメソッドを使って取得することも、要求の入力ストリームを読出しながら取得することも可能である。

5.2 属性 (Attributes)

属性は要求に関するオブジェクトである。でなければ API を介して表現できない情報をコンテナがセットするものであるが、あるいは、サーブレットが他のサーブレットと交信するために（RequestDispatcher 経由）セットすることもある。属性は ServletRequest インターフェイスの次のメソッドによってアクセス可能である。

- getAttribute
- getAttributeNames
- setAttribute

ひとつの属性名に対してはただひとつの属性値が附される。

”java.”と“javax.”のプレフィックスで始まる属性名は本仕様書による定義の為にリザーブされている。同じように”sun.”と”com.sun.”のプレフィックスで始まる属性名は Sun Microsystems 社が定義するのに予約されている。属性のセットとして纏められる総ての属性の名前は、Java 言語仕様書(http://java.sun.com/docs/books/jls)でパッケージのネーミングで規定された逆パッケージ名規約(reverse package name convention)に準拠することを勧める。

5.3 ヘッダ (Headers)

サーブレットはHTTP要求のヘッダをHttpServletRequestインターフェイスの以下のメソッドを使ってアクセスできる：

- getHeader
- getHeaders
- getHeaderNames

getHeaderメソッドはヘッダの名前を与えてその値をアクセスする。Cache-Controlヘッダのように多重ヘッダもHTTP要求には存在し得る。ひとつの要求の中に同じ名前のヘッダが複数存在した場合には、getHeaderメソッドは該要求に含まれている最初のヘッダを返す。getHeadersメソッドのほうは、Stringオブジェクトの Enumerationで指定したヘッダ名の総てのヘッダ値を返す。

ヘッダにはint型あるいはDate型オブジェクトで返したほうが便利なものもある 。HttpServletRequestインターフェイスには次のような便利なメソッドが用意されている。

- getIntHeader
- getDateHeader

getIntHeaderメソッドがヘッダ値をint型に変換できないときは 、NumbetFormatExceptionがスローされる。getDateHeaderメソッドがDate型に変換できないときにはIlligalArgumentExceptionがスローされる。

5.4 要求パス要素 (Request Path Elements)

サーブレットが要求に対してのサービスに至らしめる要求パスは、多くの重要な区間で構成されている。以下の要素は要求URIパスから取得され、要求オブジェクトを介して知ることが出来る。

- **Context Path:** このサーブレットが属する ServletContext で関連付けられたパスプレフィックス。このコンテキストが Web サーバの URL 名空間のベースのルートに配置されているデフォルト(“default”)のコンテキストのときは、このパスは空のストリングである。そうでない時は、このパスは”/”文字で始まるが”/”文字では終端されない。
- **Servlet Path:** このパス区間は、この要求を能動化したマッピングに直接対応する。このパスは”/”文字で始まる。
- **PathInfo:** Context Path でも Servlet Path でもない要求パスの部分。

HttpServletRequest インターフェイスには、以下のメソッドがこの情報のアクセス用に用意されている。

- getContextPath
- getServletPath
- getPathInfo

URL エンコーディングの要求 URI とパス部分間の相違を除いて、以下の式が常に成立することに注意することが重要である：

RequestURI = contextPath + sevletPath + pathInfo

上記を明確にするための例として、次のものを考えてみよう。/catalog というルート ディレクトリに置かれた lawn、garden、JSP のサービス サーブレットに対するコンテキストが下表なようなものであるとする：

表 1：Context セットアップ例

ContextPath	/catalog
Servlet Mapping	Pattern: /lawn Servlet: LawnServlet
Servlet Mapping	Pattern: /garden Servlet: GardenServlet
Servlet Mapping	Pattern: *.jsp Servlet: JSPServlet

この場合次のような動作が観察されることになる。

表 2：パス要素の結果

Request Path	Path Elements

/catalog/lawn/index.html	ContextPath: /catalog ServletPath: /lawn PathInfo: /index.html
/catalog/garden/implements/	ContextPath: /catalog ServletPath: /garden PathInfo: /implements/
/catalog/help/feedback.jsp	ContextPath: /catalog ServletPath: /help/feedback.jsp PathInfo: /null

5.5 パス変換法 (Path Translation Methods)

HttpServletRequest インターフェイスには、プログラム開発者が特定のパスに等価なファイル システム パスを取得するのに便利な以下の二つのメソッドがある。

- getRealPath
- getPathTransleted

getRealPath メソッドは、String 型の変数をとり、そのパスに対応するローカル ファイル システム上のファイルを String 型で返す。GetPathTranslated メソッドは、この要求の pathInfo の実際のパスを計算する。

サーブレット コンテナがこれらのメソッドに対して有効なファイル パスを求められなかった状態に陥ったとき、例えばアーカイブから Web アプリケーションが実行されたとき、リモートのファイル システムがローカルとしてアクセスできないとき、あるいはデータ ベースのときなどは、これらのメソッドはヌル(null)を返さねばならない。

5.6 クッキー (Cookies)

HttpServletRequest インターフェイスには、その要求に出てきた cookies の配列を取得する為に getCookies メソッドが用意されている。これらの cookies は、クライアントが作成する各要求にのせ、クライアント側からサーバに送られるデータである。一般的には、

クライアントが cookie の部分として送り返す情報は cookie 名と cookie 値のみである。例えばコメントのように、cookie がブラウザに送られたときにセットされ得るような、その他の属性は、通常は返されない。

5.7 SSL 属性 (SSL Attributes)

要求が HTTPS などの秘匿がかかったプロトコルを介して送られてきたときは、この情報は ServletRequest インターフェイスの isSequre メソッドにより開示されねばならない。

Java 2 Standard Edition, v.1.2 あるいは Java 2 Enterprise Edition, v.1.2 環境におけるサーブレット コンテナにおいては、その要求が SSL 認証されているときは 、java.security.cert.X509Certificate 型のオブジェクトの配列としてサーブレットのプログラマに開示されねばならないし、javax.servlet.request.X509Certificate の ServletRequest 属性を介してアクセスできねばならない。

Java 2 Standard Edition, v.1.2 環境下で動作していないサーブレット コンテナでは 、SSL 認証情報にアクセスするために固有の要求の属性をベンダが用意してもよい。

5.8 国際化 (Intenationalization)

クライアント側から応答に際してどの言語が好ましいかをサーバに指定することもあろう。この情報は、HTTP/1.1 仕様に記述されたほかのメカニズムに併せて Accept-Language ヘッダを附してクライアントから通信できる。ServlerRequest インターフェイスの以下のメソッドが、送信元の指定ロケール（言語地域）を求めるのに用意されている：

- getLocale
- getLocales

getLocale メソッドは、クライアントがコンテンツを受けるときに好ましいロケールを返す。如何に Accept-Language ヘッダがクライアントの指定言語として解釈すべきかの詳

細は、RFC 2626(HTTP/1.1)の第 14.4 節を参照のこと。

getLocales メソッドは、クライアントが受理可能なロケールであり、指定ロケールから開始する Locale オブジェクトの降順の列挙型(Enumeration)を返えす。

クライアント側から好みのロケールを指定してこないときは、getLocale メソッドはこのサーブレットコンテナのデフォルトのロケールを返し、getLocales メソッドはデフォルトロケールのひとつの Locale 要素を返さねばならない。

第 6 章　　応答 (Response)

responseオブジェクトは、サーバからクライアントへ返すべき総ての情報をカプセル化する。HTTPプロトコルにおいては、この情報は応答メッセージのHTTPヘッダまたはメッセージ本体のどちらかでサーバからクライアントに送り出される。

6.1 バッファリング (Buffering)

効率改善の為、デフォルトとしては要求されてはいないがサーブレットコンテナはクライアントへの出力をバッファリングすることが許されている。以下のメソッドが 、ServletResponseインターフェイスを介してバッファリング情報のアクセスと設定の為に用意されている：

- getBufferSize
- setBufferSize
- isCommitted
- reset
- flushBuffer

これらのServletResponseインターフェイスで用意されているメソッドにより、サーブレットがServletOutputStreamとWriterのいずれを使っていようともバッファリング動作が出来る。

getBufferSizeメソッドは、使用中のバッファのサイズを返す。この応答の為にバッファが使われていないときには、このメソッドはint型のゼロ(0)の値を返さねばならない。

サーブレットはsetBufferSizeメソッドを使って応答の為に欲しいバッファのサイズを要求できる。実際に割り当てられるバッファサイズはサーブレットが要求したとおなじサイズにすることは要求されてはいないが、少なくとも要求されたサイズ以上でなければならない。これによりコンテナが固定バッファ長のバッファを再利用して、よければ要求より大きなバッファを用意できる。本メソッドは, ServletOutputStreamあるいはWriterを使ってコンテンツを書き出す前に呼ばねばならない。何らかのコンテンツが既に書きこまれていれば、本メソッドはIllegalStateExceptionをスローする。

メソッド isCommitted は、応答のどのバイトもまだクライアントに送出されていないか否かをブール値で返す。メソッド flushBuffer はバッファの中のどの内容をもクライアントあてに送り出す。

メソッド reset は、該応答が committed されていない状態である場合にかぎりバッファに存在するデータをクリアする。この reset 以前にサーブレットがセットした総てのヘッダとステータスコードも同様にクリアされる。

すでに isCommitted の状態のときに reset メソッドが呼ばれたときは 、IllegalStateException がスローされる。このときは、応答とそれに関連する情報に変更は生じない。

使用中のバッファが一杯になったときは、コンテナは直ちにバッファの内容をフラッシュしなければならない。クライアントにデータが送信中であるときの初めての状態のときあは、この応答はこの時点で committed であると考えられる。

6.2 ヘッダ (Headers)

サーブレットは HTTP 応答のヘッダを以下の HttpSevletResponse インターフェイスのメソッドにより設定できる：

- setHeader
- addHeader

メソッド setHeader は､名前と値を与えてヘッダを設定する｡すでにヘッダが存在する場合は新しいヘッダに置きかえられる。指定の名前に複数のヘッダ値のセットが存在する場合には、総ての値がクリアされ、新しい値に置きかえられる。

メソッド addHeader は､ヘッダの指定した名前と､値のセットにヘッダ値を追加する｡与えた名前のヘッダがまだ存在していないときには、本メソッドは新しいセットを作成する。

ヘッダには int や object 型で表現したほうが良いものもある。HttpServletResponse イ

ンターフェイスの以下の便利なメソッドにより、サーブレットは適切なデータ型に対した正しいフォーマットでヘッダをセットすることができる。

- setIntHeader
- setDateHeader
- addIntHeader
- addDateHeader

無事クライアントに送り返される為に、応答がコミットされる前にヘッダが設定されていなければならない。応答がコミットされてしまった後からの如何なるヘッダもサーブレット コンテナは無視する。

6.3 便利なメソッド (Convenience Methods)

HttpServletResponse インターフェイスには、以下のような使い勝手の便を考えたメソッドがある：

- sendRedirect
- sendError

メソッド sendRedirect は、クライアントを異なった URL にリダイレクトするために、しかるべきヘッダとコンテンツ本体をセットする。このメソッドを相対 URL で呼ぶことは文法的には誤りではないが、コンテナはこの相対パスを完全に認定されている URL に変換して、クライアントに送り返さねばならない。部分的な URL しか与えられないときで、如何なる理由にしろ、有効な URL に変換できないときは、本メソッドは IllegalArgumentException をスローする。

メソッド sendError は、クライアントに返すべきしかるべきヘッダとコンテンツ本体を設定する。オプションとしての String 型のパラメタは、該エラーのコンテンツ本体のものとして sendError メソッドに用意されている。

これらのメソッドは、まだコミットされていないときはこの応答をコミット化の状態にしてしまうとともにこれを終了してしまうという副次的効果がある。これらのメソッドが呼ばれた後は、サーブレットはクライアントにいかなる出力をもしてはならない。これ

らのメソッドが呼ばれた後の応答への書きこみデータは無視される。

応答バッファにデータが既に書きこまれてはいるがクライアントへはまだ返されたいないとき（言い換えれば応答がコミットされていないときは）、応答バッファのデータはクリアされ、これらのメソッドによるデータで置換えられる。応答がコミットの状態であればこれらのメソッドは IllegalStateException をスローしなければならない。

6.4 国際化 (Internationalization)

特定の言語によるドキュメントの取得をクライアントが要求したり、クライアントが特定の言語を設定したときに対応して、サーブレットは応答の言語属性をクライアントに送り返すことが出来る。この情報は HTTP/1.1 仕様書に記された他のメカニズムとともに、Content-Language ヘッダを介して送られる。応答の言語は ServletResponse インターフェイスの setLocate メソッドで設定される。このメソッドで正しく HTTP ヘッダを設定されないと、クライアントと正しい言語で通信できない。

一番効果的なのは、ServletResponse インターフェイスの getWriter メソッドを呼ぶ前にこのメソッドを呼ぶようにプログラミングすることである。これにより、返された PrintWriter が正しく目的の言語地域用に設定されることが保証される。

メソッド setLocale のあとに setContentType メソッドが呼ばれ、与えられたコンテント型にある charset 要素が存在していれば、コンテント型に指定された charset が setLocale メソッドで設定した値をオーバライドする。

6.5 応答オブジェクトのクローズ (Closure of Response Object)

幾つかのイベントが、このサーブレットが要求を満足するコンテンツの総てを用意し、この応答オブジェクトをクローズして良いことを通知する。それらのイベントとは以下のものである：

- サーブレットの service メソッドの終了
- setContentLength で指定された量に応答として書き込まれてしまった

- メソッド sendError が呼ばれた
- メソッド sendRedirect が呼ばれた

応答が閉じられると、総てのバッファの内容は、もし残っていれば、直ちにクライアントに向けてフラッシュ（流しだし）されねばならない。

第 7 章　　セッション (Sessions)

ハイパーテキスト転送プロトコル(HTTP)はステートレスとして設計されたものである。効果的な Web アプリケーションとするために、あるクライアントからの一連の異なった要求が相互に関連させることが肝要である。多くのセッション トラッキングの方法が進化してきてはいるが、プログラマがそれを直接使うにはそれらの総てが難しいか、トラブルの種となるものであった。

本仕様では、単純な HttpSession インターフェイスが用意されており、これにより個々のクライアントからのアプローチに開発者を巻き込まなくても、サーブレット コンテナはどんな数のアプローチでもユーザのセッションを追いかけるために使うことができる。

7.1 セッション トラッキングのメカニズム (Session Tracking Mechanisms)

7.1.1 URL 再書き込み (URL Rewriting)

URL 再書き込みはセッショント ラッキングの最小公分母である。クライアントが cookie を受け付けないときは、サーバはセッション トラッキングを確立するのに URL 再書き込みを使用して良い。URL 再書き込みは、URL パスにデータを追加して、次の要求に際してコンテナがあるセッションとの関連付けを解釈できるようにする。

結果としての URL ストリングのパス パラメタとしてセッション ID(session id)がエンコードされねばならない。パラメタの名前は jsessionid でなければならない。パス情報がエンコードされた URL に一例を示すと：

http://www.myserver.com/catalog/index.html;jsessionid=1234

7.1.2 クッキー (Cookies)

HTTP cookies を介してのセッション トラッキングは一番多用されるセッション トラ

ッキングのメカニズムで、総てのサーブレット コンテナがサポートすることが要求されている。コンテナは cookie をクライアントに送出する。クライアントは次にその後に引き続くサーバへの要求上にその cookie を返す。この要求はクライアントでは一義的に該要求とセッションを対応させる。セッション トラッキング cookie の名前は JSESSIONID でなければならない。

7.1.3 SSL セッション (SSL Sessions)

セキュアなソケット層(Secure Socket Layer)、即ち HTTPS で使われている秘匿技術は、クライアントからの複数の要求が、受理されたセッションの一部であるかの明確な識別のためのメカニズムが組み込まれている。サーブレット コンテナは簡単にこのデータをセッション定義のメカニズム用に使うことが出来る。

7.2 セッションの生成

HTTP は要求／応答ベースのプロトコルである為、クライアントがそれに参加(joins)するまではセッションは新しいと考えられる。セッションが確立されたことを意味するセッショントラッキング情報がサーバに無事返されたとき、クライアントはこのセッションに参加したことになる。クライアントがセッションに参加するまでは、次に到来する該クライアントからの要求をそのセッションの部分であると仮定してはいけない。

以下のいずれかが成立すれば、セッションは new であると考えられる：

- クライアントがまだ本セッションを知っていない
- クライアントがセッションに参加しないことを選択している。このことは、サーブレットコンテナはこれによって前回の要求と今回の要求を関連付けするメカニズムを有さないということを意味する。

サーブレット開発者は彼らの開発したアプリケーションが、クライアント セッションにがまだ参加していないか、あるいは参加しようとしていない事態にも対応できるようにしておかねばならない。

7.3 セッションのスコープ (Session Scope)

HttpSession オブジェクトは、application / servlet コンテキストレベルでスコープされていなければならない。その下のメカニズム、例えばセッション確立のために使われる cookie などは、コンテキスト間で共用できが、このオブジェクトは開示されており、もっと重要なことはそのオブジェクトの属性は二つのコンテキスト間で共用されてはいけない。

7.4 属性のセッションへのバインド (Binding Attributes into a Session)

サーブレットはオブジェクトの属性を HttpSession インプリメンテーションに名前でバインドすることができる。あるセッションへバインドされたどのオブジェクトも同じ ServletContext に属する他のサーブレットからもアクセス可能であるし、他のサーブレットが該要求を同じセッションの部分であると識別することが出来る。

オブジェクトによっては、それがあるセッションに配置されたとか外されたとかしたときに通知する必要がある場合もある。この情報はオブジェクトが HttpSessionBindingListener インターフェイスを実装することで取得できる。このインターフェイスでは、オブジェクトがバインド中であるか、あるいはバインドから外されつつあるかを通知する為の以下のメソッドが定義されている。

- valueBound
- valueUnBound

valueBound メソッドはオブジェクトが HttpSession インターフェイスの getAttribute メソッドによりオブジェクトがアクセス可能になる前に呼ばれなければならない 。valueUnBound は HttpSession インターフェイスの getAttribute メソッドを介してもはやアクセス不可能となったあとで呼ばれねばならない。

7.5 セッションのタイムアウト (Session Timeouts)

HTTP プロトコルにおいては、クライアントがもはやアクティブで無くなったときの明示的な終了信号といったものは存在しない。このことは、クライアントがもはやアクティブで無くなったことを示す唯一つのメカニズムはタイムアウトであることを意味する。

セッションのデフォルトのタイムアウト期間はサーブレットコンテナで定義され 、HttpSession インターフェイスの getMaxInactiveInterval メソッドにより取得できる。このタイムアウトは、開発者が HttpSession インターフェイスの setMaxInactiveInterval メソッドを使って変更が可能である。これらのメソッドで使われているタイムアウト期間は秒で定義されている。そのセッションのタイムアウト期間が-1 にセットされているときは、該セッションはタイムアウトを発生させない。

7.6 最終アクセス時刻 (Last Accessed Times)

HttpSession インターフェイスの getLastAccessedTime メソッドにより、サーブレットが現在の要求の前にこのセッションがアクセスされた最後の時刻を知ることが可能となる。このセッションは、このセッションの部分であるひとつの要求がサーブレット コンテキストにより処理されたとき、このセッションがアクセスされたと考えられる。

7.7 重要なセッションの意味 (Important Session Semantics)

7.7.1 スレッドの問題（Threading Issues）

要求スレッドを処理中の複数のサーブレットが単一のセッション オブジェクトを同時にアクティブにアクセスする事があり得る。該セッションにストアされている資源をアクセスするときに必要とあらば同期化するのは開発者の責任である。

7.7.2 分散環境（Distributed Environments）

分散可能とマークされたアプリケーション内では、あるセッションに属する総ての要求はある時刻においては単一の VM により処理されている。加えて、setAttribute または

putValue メソッドを使って HttpSession クラスのインスタンスに配置された総てのオブジェクトは、Serializable インターフェイスを実装しなければならない。非 serializable オブジェクトを該セッションに配置されたときはサーブレット コンテナは IllegalArgumentException 例外をスローする。

上記の制約は非分散コンテナで遭遇する以上に更なる並行処理の問題に遭遇はしないことが保証されることを意味する。加えて、コンテナの提供者は、セッション オブジェクトとそのコンテンツを分散システムのどのアクティブなノードから他のノードへ移動させる機能をもたせてスケーラビリティを確保することができる。

7.7.3 クライアントのセマンティクス（Client Semantics）

cookies あるいは SSL 認証は通常 web ブラウザのプロセスで処理され、そのブラウザの特定のウインドウとは関連図けられてはいないので、クライアントアプリケーションからサーブレットコンテナへの総てのウインドウからの要求は、同じセッションの部分である可能性がある。ポータビリティを最大とする為に、開発者はクライアントの総てのウインドウが同じセッションに参加しているものと仮定すべきである。

第 8 章　　要求のディスパッチ (Dispatching Requests)

ウェブ アプリケーションを構築するとき、要求の処理を別のサーブレットに転送する、あるいは別のサーブレットの出力を応答に含めることができると有用である 。RequestDispatcher インターフェイスはこれを実現するメカニズムを与えるものである。

8.1 RequestDispatcher の取得 (Obtaining a RequestDispatcher)

RequestDispatcher インターフェイスを実装したオブジェクトは、ServletContext から次のメソッドを介して取得できる：

- getRequestDispatcher
- getNamedDispatcher

getDispatcher メソッドは、ServletContext のスコープ内のパスを記述する String の変数をとる。このパスは ServletContext のルートに対して相対でなければならない。このパスはサーブレットをルックアップし、それを RequestDispatcher オブジェクトにラップし、それを返す。与えられたパスではサーブレットはひとつも得られないときは、返される RequestDispatcher は単にそのパスのコンテンツのみで返される。

getNamedDispatcher は、ServletContext で知られたサーブレット名を示す String 型の変数を取る。あるサーブレットが与えられた名前で ServletContext に知られているときは、それは RequestDispatcher オブジェクトでラップされ、返される。その与えられた名前では関連付けられるサーブレットが存在しないときは、このメソッドはヌル(null)を返さなければならない。

RequestDispatcher オブジェクトが、ServletContext のルートに相対ではではないが、その代り現在の要求のパスに相対であるような相対パスを使って取得できるようにするには、ServletRequest インターフェイスには次のようなメソッドが用意されている：

- getRequestDispatcher

このメソッドの振舞いは ServletContext にある同じ名前の同じであるが、動作のための

変数の部分として与えられるために、コンテキスト内に完全なパスを要求しない。サーブレットコンテナはその要求オブジェクト内の情報を与えられた相対パスから完全なパスに変換するのに使用できる。例えば、’/’にあるコンテキストルート、/garden/tools.html への要求、ServletRequest.getRequestDispatcher(“header.html”)を介して得られた要求ディスパッチャは、ServletContext.getRequestDispatcher(“/garden/header.html”)を呼んだと全く同じ振舞いを示す。

8.1.1 要求ディスパッチャ パスのクエリ ストリング（Query Strings in Request Dispatcher Path）

パス情報を使って RequestDispatcher を生成できる ServletContext と ServlerRequest では、そのパスにオプショナルとしてクエリの文字ストリング情報を付けることが出来る。例えば、開発者は以下のコードを使って RequestDispatcher を取得してもよい：

```
String path = “/raisons.jsp?orderno=5”;
RequestDispatcher rd = context.getRequestDispatcher(path);
rd.include(request, response);
```

クエリ文字列の内容はパラメタのセットに追加され、それに含まれるサーブレットがアクセスできる。パラメタは順番付けられており、RequestDispatcher を生成するのに使われたクエリ文字列で指定されたどのパラメタも優先性を持つ。RequestDispatcher に関連付けされたパラメタは、include または forward 呼び出しの期間にのみのスコープとなる。

8.2 要求ディスパッチャの使用 (Using a Request Dispatcher)

要求ディスパッチャを使用するには、開発者は RequestDispatcher インターフェイスの include または forward メソッドのいずれかを、Servlet インターフェイスの service メソッドを介して渡された request と response の変数を使って呼び出す必要がある。

コンテナのプロバイダは、必ず元の要求と同じ同一 VM のスレッド内に、ターゲットのサーブレットへの配送が生ずるようにしなければならない。

8.3 インクルード (Include)

RequestDispatcher インターフェイスの include メソッドは何時でも呼ばれてもよい。ターゲットのサーブレットはその要求オブジェクトの全てのアスペクトにアクセスできるが、応答バッファの終わり以上にコンテンツを書き込むかあるいは明示的に ServletResponse インターフェイスの flush メソッドを呼ぶかで応答をコミットする権限とともに、応答オブジェクトの ServletOutputStream または Writer にのみに情報を書き込むことが出来る。インクルードされたサーブレットはヘッダをセットできないし、応答のヘッダに影響を及ぼすどのメソッドも呼ぶことは出来ない。

8.3.1 インクルードされた要求パラメタ（Included Request Parameters）

あるサーブレットがある include 内で使用されているときは、時にはそのサーブレットが、それが呼ばれかつ元の要求パスではないパスを知る必要がある。以下の要求の属性がセットされる：

javax.servlet.include.request_uri
javax.servlet.include.context_path
javax.servlet.include.servlet_path
javax.servlet.include.path_info
javax.servlet.include.query_string

これらの属性はインクルードされたサーブレットからは request オブジェクトの getAttribute メソッドを介してアクセスできる。

インクルードされたサーブレットは、NamedDispatcher を使って取得されたが、これらの属性はセットされない。

8.4 転送 (Forward)

RequestDispatcher インターフェイスの forward メソッドは、そのクライアントへの出

力がコミットされていないときに呼び出しサーブレットから呼ばれる。応答のバッファにコミットされていない出力が存在するときは、ターゲットのサーブレットの service メソッドが呼ばれる前にリセット（バッファのクリア）されていなければならない。応答がコミットされてしまっているときは、IllegalStateExceotion 例外がスローされねばならない。

ターゲットのサーブレットに開示された要求オブジェクトのパス要素は 、RequestDispatcher を取得するときに使ったパスを反映していなければならない。これの唯一の例外は、RequestDispatcher が getNamedDispatcher メソッドを介して取得されたときである。この場合、要求オブジェクトのパス要素は元の要求のそれを反映する。

RequestDispatcher インターフェイスの forward メソッドが戻る前に、その応答はサーブレット コンテナによりコミットされ、閉じられねばならない。

8.5 誤り処理 (Error Handling)

要求ディスパッチャのターゲットからスローされたとき、ランタイム例外と 、ServletException または IOException 型のチェック済み例外だけが呼び出しサーブレットに伝播されねばならない。その他の全ての例外は、ServletException でラップされ、その例外の根因は元の例外にセットされる。

第 9 章　　ウェブ アプリケーション (Web Applications)

ウェブ アプリケーションとは、サーブレット、HTML ページ、クラス、および他のバンドル可能な資源の集まりであり、複数のベンダの複数のコンテナ上で走る。ウェブ アプリケーションはウェブ サーバのなかの特定のパスのルート化される。例えば、カタログのアプリケーションは、http://www.mycorp.com/catalog なるパスに配置され得る。このプレフィックスで始まる全ての要求はカタログのアプリケーションを代表する ServletContext へ転送される。

サーブレットコンテナはまた、ウェブ アプリケーションの自動生成のルールを確立することも可能である。例えば、~user/ マッピングは/home/user/public_html/にベースがあるウェブ アプリケーションにマッピングするのに使える。

デフォルトとしては、ウェブ アプリケーションのインスタンスはどの時刻においてもひとつの VM 上でしか走ってはいけない。この振舞いは、そのアプリケーションがそのデプロイメントディスクリプタで「分散可能(distributable)」とマークされていればそれにオーバライドされる。あるアプリケーションが分散可能とマークされているとき、開発者は通常のウェブアプリケーションで課せられているよりももっと厳しい制約の規則のセットに従わねばならない。これらの特定の規則は本仕様書のなかの各所に記されている。

9.1 ServletContext との関係 (Relationship to ServletContext)

サーブレットコンテナはウェブアプリケーションと ServletContext との 1 対 1 の対応をとらねばならない。ServletContext オブジェクトはそのアプリケーションへのサーブレットから見た投影図と見ることが出来る。

9.2 ウェブ アプリケーションの要素 (Elements of a Web Application)

ウェブ アプリケーションは以下のアイテムからなる：

- サーブレット

- JavaServer ページ[1]
- ユーティリティのクラス
- スタティックなドキュメント（HTML、イメージ、サウンド、等）
- クライアント側へのアプレット、ビーンズ、およびクラス
- 上記の全ての要素をともにつなぐ記述型メタ情報

注 1：http://java.sun.com/products/jsp から得られる JavaServer ページの仕様を参照のこと

9.3 リプレゼンテーション間の区別 (Distinction Between Representations)

本仕様書では、オープンなファイル システム、アーカイブ ファイル、あるいは導入の目的のための幾つかの他の形式に存在可能なような階層構造を定義している。サーブレット コンテナは、ランタイムのリプレゼンテーションとしてこの構造をサポートすることを勧告するが、これは要求事項ではない。

9.4 ディレクトリ構造 (Directory Structure)

ウェブ アプリケーションは、構造化階層のディレクトリとして存在する。この階層のルータは、このコンテキストの部分である支援ファイルとしてのドキュメントのルートとなる。例えば、ウェブ サーバの/catalog に配置されたウェブ アプリケーションはでは、ウェブ アプリケーション階層のベースに配置された index.html ファイルは 、/catalog/index.html への要求を満足させる。

特別なディレクトリとして、”WEB-INF”と名づけられたディレクトリがそのアプリケーションの階層に存在する。このディレクトリは、そのアプリケーションのドキュメントルートに存在しない、そのアプリケーションに関連する全てが含まれる。WEB-INF ノードはそのアプリケーションのパブリックなドキュメントのトリーの部分ではないことに注意することが大切である。WEB-INF ディレクトリに含まれるどのファイルも直接クライアントに使われることは無い。

WEB-INF ディレクトリの中身は：

- /WEB-INF/web.xml　デプロイメントディスクリプタ
- /WEB-INF/classes/*　サーブレットとユーティリティのクラス。このディレクトリ内のこれらのクラスは、アプリケーション クラス ローダがそこからクラス ファイルをロードするのに使われる。
- /WEB-INF/lib/*.jar　Java のアーカイブ ファイルのエリアで、サーブレット、ビーンズ、および他のそのアプリケーションに有用なユーティリティ クラスを含める。全てのこれらのアーカイブ ファイルはウェブ アプリケーション クラス ローダがそこからクラスをロードするのに使われる。

9.4.1 ウェブア プリケーション ディレクトリ構造の例（Sample Web Application Directory Structure）

ここに示したのは、同じウェブ アプリケーション内の全てのファイルのリストである：

/index.html
/howto.jsp
/feedback.jsp
/images/banner.gif
/images/jumping.gif
/WEB-INF/web.xml
/WEB-INF/lib/jspbean.jar
/WEB-INF/classes/com/mycorp/servlets/Myservlet.class
/WEB-INF/classes/com/mycorp/util/MyUtils.class

9.5 ウェブ アプリケーション アーカイブ ファイル (Web Application Archive File)

ウェブ アプリケーションは標準の Java のアーカイブ ツールを使ってウェブ アーカイブ　フォーマット(war)のファイルにパッケージングされサインすることが出来る。例えば、問題追跡(issuue tracking)なるアプリケーションは、issuetrack.war なるファイルネームのアーカイブで分散できる。

そのような形式でパッケージングされたとき、Java アーカイブ ツールに有用な情報を含む META-INF なるディレクトリが現れる。このがディレクトリ生じたときは、サーブレット コンテナはそれをウェブ クライアントの要求へのコンテンツに使ってはならない。

9.6 ウェブ アプリケーション設定ディスクリプタ (Web Application Configuration Descriptor)

以下の種類の設定と導入の情報が、ウェブ アプリケーションデプロイメントディスクリプタのなかに存在する：

- ServletContext の Init パラメタ
- セッションの設定
- サーブレット／JSP 定義
- サーブレット／JSP マッピング
- MIME タイプ マッピング
- ウエルカム ファイル リスト
- エラー ページ
- セキュリティ

全てのこれらの種類の情報はデプロイメント ディスクリプタで伝達される。（第 13 章「デプロイメント ディスクリプタ」を参照）

9.7 ウェブ アプリケーションの置換 (Replacing a Web Application)

アプリケーションは改善がなされるので、随時置換されねばならない。継続して走っているサーバにおいては、コンテナをリスタートすることなく新しいウェブ アプリケーションをロードし、古いアプリケーションをシャット ダウン出来ることが理想的である。アプリケーションが置換されたときは、コンテナはそのアプリケーション内でのセッションのデータを保存する堅固なアプローチを提供しなければならない。

9.8 誤り処理 (Error Handling)

ウェブ アプリケーションはエラーが発生したらそのアプリケーションの他のリソースを使うよう指定してもよい。これらのリソースはデプロイメントディスクリプタに規定されている（第13章「デプロイメントディスクリプタ」を参照）。もしも誤りハンドラの場所がサーブレットまたはJSPのときは、以下の要求の属性がセットされうる：

- javax.servlet.error.status_code
- javax.servlet.error.exception_type
- javax.servlet.error.message

これらの属性により、サーブレットはステータ スコード、例外タイプ、およびエラー メッセージに基づいて特別のコンテンツを発生させることが出来る。

9.9 ウェブ アプリケーションの環境 (Web Application Environment)

Java 2プラットホーム エンタープライズ版v1.2では、外部情報がどのように名前付けされ組織化されているかの明示的な知識を必要としないで、アプリケーションが容易にリソースや外部情報をアクセスできるよう、名前付けの環境を定めている。

サーブレットはJ2EEの統合コンポネントのタイプで、サーブレットがリソースやエンタープライズ ビーンへの参照を取得することが出来るようにする情報を指定するウェブ アプリケーションデプロイメントディスクリプタの中で設定はなされている。この情報を含むデプロイメント エレメントは：

- env-entry
- enj-ref
- resource-ref

env-entryエレメントは、java:comp/envコンテキストに相対な基本的な環境エントリ名、期待されるJavaタイプの環境エントリ値（JNDIルックアップ法で返されるオブジェクトの型）、及び付加的な環境エントリ値、をセットアップする情報を含む。enj-refエレメントは、サーブレットがエンタープライズ ビーンのホーム インターフェイスを見つ

けるのに必要な情報を含む。resource-refエレメントはリソース ファクトリをセットアップするのに必要な情報を含む。

環境のセットアップに関するJ2EE環境の要求事項はJava 2プラットホーム エンタープライズ版v1.2の仕様書の第5章に記されている[1]。J2EE対応の実装の一部ではないサーブレット コンテナにおいてはJ2EE仕様書に記されたアプリケーション環境機能を実装することは必要とはしないが推奨はされる。

注1：J2EEの仕様はhttp://java.sun.com/j2eeから入手できる。

第 10 章　　要求とサーブレットとのマッピング (Mapping Requests to Servlets)

本仕様書の以前の版では、クライアントからの要求とサーブレットのマッピングは、推奨するマッピング技術によるマッピングのセットを指定することで、フレキシブルなものとするものであった。この仕様書では、ウェブ アプリケーション デプロイメント メカニズムを介してデプロイメントされるウェブ アプリケーションが使うために、マッピング技術のセットを必要とする。ランタイムの表現としてサーブレット コンテナがデプロイメント表現を使うことを強く推奨されるのとまったく同じく、単にウェブ アプリケーションのデプロイメントの一部としてではなく、全ての目的の為にサーバにおいてこれらのパス マッピングのルールを使用することが強く推奨される。

10.1　URL パスの使用 (Use of URL Paths)

サーブレット コンテナは、要求とサーブレットとのマッピングに URL パスを使用しなければならない。要求からの RequestURI を使用するコンテナは、サーブレットへのマッピング パスとして、それからコンテキスト パスを引かなければならない。URL パス マッピングの規則は以下のとおりである（最初に合致した項目があれば、以降の項目への適合は調べなくてよい）：

1. サーブレット コンテナは、要求のパスがそのままサーブレットのパスに合致するかをまず試みる。
2. コンテナは次に最長プレフィックス パス・マッピングに合致するか繰り返し試みる。このプロセスは、’/’文字をパス セパレータとして、ディレクトリのパス トリーをひとつずつ下げていき、それと合致するサーブレットがあるかを調べるときに生ずる。
3. 最後のノードが拡張子（例えば.jsp）を含むときは、サーブレット コンテナはその拡張子の要求を取り扱う合致サーブレットがあるかを調べる。拡張子は、そのパスの最後の’.’文字の後の部分である。
4. 上記の二つのルールのどれにもサーブレットの一致が取れないときは、コンテナは要求されているリソースに適切なコンテンツのサービスを行う。そのアプリケーションで「デフォルト」のサーブレットが定義されているときは、この場合そのサーブレットが適用される。

10.2　マッピングの仕様 (Specification of Mapping)

ウェブ アプリケーションデプロイメント ディスクリプタにおいて、マッピングを定義する為に、以下のシンタックスが使われる：

- ‘/’文字で始まり’/*’で終わる文字列は、パス マッピングに使われる。
- ‘*.’のプレフィックスで始まる文字列は拡張マッピングとして使われる。
- その他の全ての文字列は、そのままで正しく合致する用途にのみ使われる。
- ‘/’文字のみを含む文字列は、そのマッピングで指定されたサーブレットがそのアプリケーションのデフォルトのサーブレットとなる。

10.2.1　暗黙的マッピング（Implicit Mappings）

コンテナが JSP コンテナを包含しているときは、*.jsp 拡張子は暗黙的にそれにマッピングされ、JSP ページがオンデマンドで実行される。ウェブ アプリケーションが*.jsp マッピングを定義しているときは、そのマッピングのほうがこの内示的マッピングに優先する。

サーブレット コンテナは明示的なマッピングが優先される限り他の暗黙的マッピングを作ってもよい。例えば、*.shtml の暗黙的マッピングは、サーバ サイド インクルード機能へコンテナによってマッピングさせられよう。

10.2.2　マッピング・セットの例（Example Mapping Set）

以下のマッピングを考えてみよう：

表 3：　マップのセットの例

パス・パタン	サーブレット
/foo/bar/*	servlet1
/baz/*	servlet2
/catalog	servlet3
*.bop	servlet4

表 4： マップのセットの例に適用される到来パス

到来パス	要求を処理するサーブレット
/foo/bar/index.html	servlet1
/foo/bar/index.pop	servlet1
/baz	servlet2
/baz/index.html	servlet2
/catalog	servlet3
/catalog/index.html	"default"のサーブレット
/catalog/racecar.bop	servlet4
/index.bop	servlet4

結果は以下の動作となる：

/catalog/index.html 及び/catalog/racecar.bop の場合、/catalog にマッピングされたサーブレットは使われない。なぜなら、正確なマッチングがとれなく、ルールには’*’文字が含まれていないからである。

第 11 章　　セキュリティ (Security)

ウェブ アプリケーションは**開発者**(Developer)によって作られる。**開発者**はそれを次に与えるとか売るとか、さもなければそのアプリケーションを**デプロイヤ**(導入者：Deployer)に渡しランタイムの環境内にインストールされる。**開発者**にとって、デプロイメント（導入）されるアプリケーションに如何にセキュリティがセットアップされるべきかの属性をやりとりできると有用である。

ウェブ アプリケーションのディレクトリ レイアウトとデプロイメント ディスクリプタにより、本章の各要素はデプロイメントのリプレゼンテーションにのみ必要であり、ランタイムのリプレゼンテーションでは必要としない。しかしながら、ランタイムのリプレゼンテーションの一部としてこれらの要素をコンテナが実装することを推奨する。

11.1　イントロダクション (Introduction)

ウェブ アプリケーションは、多くのユーザからアクセスされる多くのリソースを含む。重要な情報がしばしばインターネットなどの保護されていないオープンなネットワークを通過する。そのような環境にあっては、あるレベルのセキュリティのが要求されるウェブ アプリケーションが相当数存在する。ほとんどのサーブレット コンテナがこれらの要求を満たすためのそれぞれのメカニズムとインフラストラクチャを備えている。品質保証と実装の詳細はおのおの異なるが、それらのメカニズムの全てが以下の機能の幾つかを共有している：

- **認証：**　通信エンティティ同士が、互いに特定のエンティティに代わって動作していることを証明するためのメカニズム。
- **リソースへのアクセス制御：**　取得性、保全性、または秘匿性の適用実施のために、ユーザやプログラムの集まりに対して、リソースへの関与を制限するメカニズム。
- **データ保全性：**　その情報が通過する間に、第 3 者によって変更されていないことを証明するメカニズム
- **機密性またはデータのプライバシー：**　その情報がそれをアクセスする為にオーソライズされたユーザに対してのみ取得可能であり、伝送の際不信をまねくことになっていないものであることを確たるものとするメカニズム。

11.2　宣言型セキュリティ (Declarative Security)

宣言型セキュリティは、ロール、アクセス制御、認証の要求事項などを含むアプリケーションのセキュリティの構造をフォームとしてそのアプリケーションの外部に表記する手法をいう。ウェブ アプリケーションにおいて、デプロイメント ディスクリプタが宣言型セキュリティの主たる手段となる。

デプロイヤは、そのアプリケーションの論理的なセキュリティの要求をランタイムの環境ごとに固有のセキュリティ ポリシーのレプレゼンテーションにマッピングする。ランタイム時は、サーブレット コンテナはそのセキュリティ ポリシー、即ちデプロイメント ディスクリプタから導き出され、**デプロイヤ**によって認証を施行するように設定されたセキュリティ ポリシーを使用する。

11.3　プログラマティック セキュリティ (Programmatic Security)

プログラマティック セキュリティは、セキュリティが必要とされるアプリケーションで、宣言型セキュリティのみではそのアプリケーションのセキュリティ モデルを十分に記述できないようなときに用いられる。プログラマティック セキュリティは 、HttpServletRequest インターフェイスの以下のメソッドからなる：

- getRemoteUser
- isUserInRole
- getUserPrincipal

getRemoteUser メソッドは、クライアント認証を受けたユーザ名をかえす 。
isUserInRole は、そのユーザがある与えられたセキュリティ ロール内にあるかどうかを、そのコンテナで使われているセキュリティのメカニズムに問い合わせる 。
getUserPrincipal メソッドは、java.security.Principal オブジェクトを返す。

これらの API により、サーブレットはリモートのユーザの論理的なロールに基づくビジネス ロジックの判断ができる。またこれらにより、サーブレットは現在のユーザのプリンシパル名を求めることができる。

getRemoteUser が null を返すときは（即ち認証されたユーザが存在しない）、isUserInRole メソッドは常に false を返し、getUserPrincipal メソッドは常に null を返す。

11.4 ロール (Rolls)

ロール（職務とか役割）というのは、**アプリケーション開発者**または**アセンブラ**が定めた抽象的なユーザの論理グルーピングのことである。そのアプリケーションがデプロイメントされたら、これらのロールはデプロイヤにより、ランタイムの環境下に、プリンシパルとかグループなどのセキュリティのエンティティにマッピングされる。

到来した要求に結び付けられたプリンシパルへの宣言型またはプログラマティックのセキュリティを、呼び出しプリンシパルのセキュリティの属性にもとづき、サーブレットは実施する。例えば、

1. デプロイヤがセキュリティのロールをその運用環境においてあるユーザ グループにマッピングするとき。呼び出しプリンシパルが属しているユーザ グループが、そのセキュリティの属性から検索される。そのプリンシパルのユーザ グループが運用環境においてそのセキュリティ ロールがマッピングされているユーザグループと一致するときは、そのプリンシパルはそのセキュリティ ロールの中にいることになる。
2. デプロイヤがあるセキュリティ ロールをあるセキュリティ ポリシー ドメインのプリンシパル名にマッピングしたときは、その呼び出しプリンシパルのプリンシパル名がそのセキュリティ属性から検索される。そのセキュリティ ロールにマッピングされているプリンシパルと同じプリンシパルのときは、呼び出しプリンシパルはそのセキュリティ ロールの中にいる。

11.5 認証 (Authentication)

ウェブ クライアントは、以下のメカニズムを使ったウェブ サーバへのユーザ認証ができる：

- HTTP 基本認証（HTTP Basic Aythentication）
- HTTP ダイジェスト認証（HTTP Digest Authentication）
- HTTS クライアント認証（HTTPS Client Authentication）
- フォーム ベース認証（Form Based Authentication）

11.5.1 HTTP 基本認証（HTTP Basic Aythentication）

HTTP 基本認証は HTTP/1.1 仕様書で規定されている認証のメカニズムである。このメカニズムはユーザ名とパスワードに基づいている。ウェブ サーバはウェブ クライアントにユーザ認証を要求する。要求の一部として、ウェブ サーバはその中でユーザが認証されるその要求のレルム(*realm*)と呼ばれる文字列を渡す。基本認証メカニズムのレルムの文字列は、特定のセキュリティ ポリシー ドメインを反映（混乱を生じやすいが、これもレルムと呼ばれる）しなければならなくは無いことに注意のこと。ウェブ クライアントはユーザ名とパスワードをユーザから取得し、これをサーバに送信する。ウェブ サーバは次に指定レルム内でそのユーザを認証する。

基本認証は、ユーザのパスワードが単純な base64 エンコードで送信され、ターゲットのサーバは認証されないので、セキュアな認証プロトコルではない。しかしながら、セキュアなトランスポートメカニズム(HTTPS)、あるいはネットワーク レベル（IPSEC プロトコルや VPN 戦術など）でのセキュリティなどの付加の保護により、これらの心配を緩和することができる。

11.5.2 HTTP ダイジェスト認証（HTTP Digest Authentication）

HTTP 基本認証と似て、HTTP ダイジェスト認証はユーザ名とパスワードに基づいて認証を行う。しかしながらこの認証では、基本認証で使われていた単純な base64 エンコーディングよりはもっと安全な暗号化によりパスワードを送信して認証を行う。この認証は HTTPS クライアント認証のようなプライベート鍵のスキームほどセキュアではない。ダイジェスト認証は、現在広く使われてはいないので、サーブレット コンテナはこれをサポートすることは要求はされていないが、推奨はされている。

11.5.3 フォーム ベース認証（Form Based Authentication）

ブラウザの組込み認証メカニズムでは、「ログイン画面」の見た目と感じを制御することは出来ない。従ってこの仕様では**開発者**がログイン画面の見た目と感じを制御できるフォーム ベースの認証を定めている。

ウェブ アプリケーションのデプロイメント デスプリプタには、このメカニズムで使われるログイン フォームとエラー ページのエントリを含む。ログイン フォームはユーザがユーザ名とパスワードを指定するフィールドを含んでいなければならない。これらのフィールドは、各々’j_username’と’j_password’と名前が付されていなければならない。

ユーザがプロテクトされたウェブのリソースにアクセスしようとした場合は、コンテナはそのユーザが認証されているかをチェックする。認証されており、かつそのリソースへのアクセスがそのユーザに許可されていれば、要求されたリソースが起動され、返される。ユーザが認証されていなければ、以下のステップ全部が生じる：

1. そのセキュリティ制約に対応したログイン フォームがクライアントに返される。認証のトリガーとなったURLパスはコンテナに蓄積される。
2. クライアントは、ユーザ名とパスワードのフィールドを含んだそのフォームを埋める。
3. そのフォームはサーバにポストで戻される。
4. コンテナはそのユーザの認証するため、このフォームを処理する。もし認証に失敗すれば、エラー ページが返される。
5. 認証されたプリンシパルは、元のウェブ要求をアクセスするためにオーソライズされたロール内にあるかどうかがチェックされる。
6. クライアントは蓄積されている元のURLパスを使ってオリジナルのリソースにリダイレクトされる。

ユーザ認証が成功しなかった場合には、エラー ページがクライアントに返される。ユーザが失敗したオーソライゼーションが判るような情報をそのエラー ページに含めることを推奨する。

基本認証と似て、ユーザのパスワードがプレインなテキストとして送信され、またターゲットのサーバが認証されないので、これはセキュアな認証プロトコルとはいえない。しかしながら、セキュアなトランスポート メカニズム(HTTPS)を適用したり、またはネットワーク レベルでのセキュリティ(IPSECまたはVPN)を使ったりして保護を追加することで、これらの懸念がある程度軽減される。

11.5.3.1 ログイン フォームに関する注 (Login Form Notes)

フォーム ベースのログインと URL ベースのセッション トラッキングは実装に際して問題を生じやすい。フォーム ベースのログインは、そのセッションが cookies か SSL セッション情報によって維持されているときにのみ使われるようにすることを強く推奨する。

認証が適正になされるよう、ログイン フォームのアクションは常に”j_security_check”でなければならない。この制約は、それが要求するリソースがなんであれこのログインのフォームがいつも機能し、またサーバが下りのフォームをアクション フィールドを訂正するようサーバに要求してしまうことを防止するために設定されている。

ここに、このフォームがどのように HTML ページにエンコーディングされるかのサンプルを示す：

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

11.5.4 HTTPS クライアント認証 (HTTPS Client Authentication)

HTTPS（SSL 上の HTTP）を使ったエンド ユーザ認証は強力な認証メカニズムである。このメカニズムではユーザは公開鍵認定(PKC: Public Key Certificate)を持つことが要求される。PKC は e-コマースのアプリケーションでは有用であり、また企業内でのブラウザの単一のサイン オンとしても有用である。J2EE 対応でないサーブレット コンテナでは、HTTPS プロトコルのサポートは要求されていない。

11.6 サーバによる認証情報のトラッキング (Server Tracking of Authentication Information)

ランタイム環境においてロールがマッピングされるセキュリティの識別（ユーザーズとかグループなど）は、アプリケーション依存というよりは環境依存であるので、以下のことが好ましい：

1. ログインのメカニズムやポリシーは、そのウェブ アプリケーションが導入される環境のプロパティとすること。
2. 同じ認証情報が、同じコンテナに導入される全てのアプリケーションに対して、プリンシパルをリプレゼントするのに使えるようにする。
3. ユーザがセキュリティ ポリシー ドメインをまたぐときにのみ、そのユーザに最認証を要求すること。

従って、サーブレット コンテナには、ひとつのウェブ アプリケーションで認証を受けたユーザが、同じセキュリティのアイデンティティで制約されたコンテナによって管理された他のどんなリソースにもアクセス出来るように、ウェブ アプリケーション レベルではなくコンテナ レベルでの認証情報のトラッキングすることが要求される。

11.7 セキュリティ制約の指定 (Specifying Security Constrains)

セキュリティ制約は、ウェブ コンテンツに意図するプロテクションを注釈する宣言型の手法である。制約には以下の要素が含まれる：

- ウェブ リソースのコレクション
- オーソライズの制約
- ユーザ データの制約

ウェブ リソースのコレクションとはプロテクトされるべきリソースのセットを記述するための、URL パタンと HTTP メソッドのセットである。ウェブ リソースのコレクションに記述された URL パタンに一致する要求が含まれる全ての要求がこの制約の対象となる。

オーソライズの制約とは、そのウェブ リソースのコレクションで記述されたリソースにアクセスするためには、その一部でなければならないユーザのロールのセットを言う。もしそのユーザが許可されたロールの一部でない場合は、そのユーザはそのリソースへのアクセスが拒否される。

ユーザ データの制約とはクライアントとサーバの通信プロセスのトランスポート層が、コンテンツの完全性（送信中の改変の防止）を保証しているか、または信頼性（送信中の盗聴の防止）を保証しているかのいずれかが満足しているとかをいう。

11.7.1 デフォルトのポリシー (Default Policies)

デフォルトでは、リソースのアクセスには認証は必要とされない。認証は、ディプロイメント デスクリプタで指定されている場合にのみ、その特定のリソースのコレクションの要求にのみ必要とされる。

第 12 章　アプリケーション・プログラミング・インターフェイス (Application Programming Interface)

ここに示すのは、サーブレット API を構成するインターフェイス、クラス、及び例外のリストである。これらのメンバとそのメソッドの詳細な記述は、Java Servlet API Reference, v2.2 を参照されたい。

太字で示されているのは、本仕様書の今回のバージョンで新しくなった部分である。

表 5:　サーブレット API パッケージのサマリ

Package javax.servlet	**Packege javax.servlet.http**
RequestDispatcher	HttpServletRequest
Servlet	HttpServletResponse
ServletConfig	HttpSession
ServletConftext	HttpSessionBindingListener
ServletRequest	HttpSessionContext
ServletResponse	Cookie
SingleThreadModel	HttpServlet
GenericServlet	HttpSessionBindingEvent
ServletInputStream	HttpUtils
ServletOutputStream	
ServletException	
UnavailableException	

12.1　Package javax.servlet

12.1.1　RequestDispatcher

```
public interface RequestDispatcher

public void forward(ServletRequest req, ServletResponse
      res);
public void include(ServletRequest req, ServletResponse
     res);
```

12.1.2 Servlet

```
public interface Servlet

public void init(ServletConfig config) throws
    ServletException;
    public ServletConfig getServletConfig();
public void service(ServletRequest req, ServletResponse
    res) throws IOException, ServletException;
public String getServletInfo();
public void destroy();
```

12.1.3 ServletConfig

```
public interface ServletConfig

public ServletContext getServletContext();
public String getInitParameter(String name);
public Enumeration getInitParameterNames();
public String getServletName();
```

12.1.4 ServletContext

```
public interface ServletContext

public String getMimeType(String filename);
public URL getResource(String path) throws
    MalformedURLException;
public InputStream getResourceAsStream(String path);
public RequestDispatcher getRequestDispatcher(String
    name);
public RequestDispatcher getNamedDispatcher(String name);
public String getRealPath(String path);
public ServletContext getContext(String uripath);
```

```
public String getServerInfo();
public String getInitParameter(String name);
public Enumeration getInitParameterNames();
public Object getAttribute(String name);
public Enumeration getAttributeNames();
public void setAttribute(String name, Object attribute);
public void removeAttribute(String name);
public int getMajorVersion();
public int getMinorVersion();
public void log(String message);
public void log(String message, Throwable cause);

//deprecated methods
public Servlet getServlet(String name) throws
     ServletException;
public Enumeration getServlets();
public Enumeration getServletNames();
public void log(Exception exception, String message);
```

12.1.5 ServletRequest

```
public interface ServletRequest

public Object getAttribute(String name);
public Object getAttribute(String name, Object attribute);
public Enumeration getAttributeNames();
public void removeAttribute(String name);
public Locale getLocale();
public Enumeration getLocales();
public String getCharacterEncoding();
public int getContentLength();
public String getContentType();
public ServletInputStream getInputStream() throws
     IOException;
public String getParameter(String name);
```

```
public String getParameterNames();
public String getParameterValues();
public String getProtocol();
public String getScheme();
public String getServerName();
public String gerServerPort();
public BufferReader getReader() throws IOException;
public String getRemoteAddr();
public String getRemoteHost();
public Boolean isSecure();
public RequestDispatcher getRequestDispatcher(String
    path);

// deprecated methods
public String getRaelPath();
```

12.1.6 ServletResponse

```
public interface ServletResponse

public String getCharacterEncoding();
public ServletOutputStream getOutputStream() throws
    IOException;
public PrintWriter getPrintWriter() throws IOException;
public void setContentLength(int length);
public void setContentType(String type);
public void setBufferSize(int size);
public int getBufferSize();
public void reset();
public Boolean isComitted();
public void flushBuffer() throws IOException;
public void setLocale(Locale locale);
public Locale getLocale();
```

12.1.7 SingleThreadModel

```
public interface SingleThreadModel

// no methods
```

12.1.8 GenericServlet

```
public abstract class GenericServlet implements Servlet

public GenericServlet();

public String getInitParameter();
public Enumeration getInitParameterNames();
public ServerConfig getServerConfig();
public ServletContext gerServletContext();
public String getServletInfo();
public void init();
public void init(ServletConfig config) throws
     ServletException;
public void log(String message);
public void log(String message, Throwable cause);
public abstract void service(ServletRequest req,
     ServletResponse res) throws ServletException,
     IOException;
public void destroy();
```

12.1.9 ServletInputStream

```
public abstract class ServletInputStream extends
     InputStream

public ServletInputStream();
```

```
public int readLine(byte[] buffer, int offset, int length)
    throws IOException;
```

12.1.10 ServletOutputStream

```
public abstract class ServletOutputStream extends
    OutputStream

public ServletOutputStream();

public void print(String s) throws IOException;
public void print(Boolean b) throws IOException;
public void print(char c) throws IOException;
public void print(int i) throws IOException;
public void print(long l) throws IOException;
public void print(float f) throws IOException;
public void print(double d) throws IOException;
public void println() throws IOException;
public void println(String s) throws IOException;
public void println(boolean b) throws IOException;
public void println(char c) throws IOException;
public void println(int i) throws IOException;
public void println(long l) throws IOException;
public void println(float f) throws IOException;
public void println(double d) throws IOException;
```

12.1.11 ServletException

```
public class ServletException extends Exception;

public ServletException();
public ServletException(String message);
public ServletException(String message, Throwable cause);
public ServletException(Throwable cause);
```

```
public Throwable getRootCause();
```

12.1.12 UnavailableException

```
public class UnavailableException extends ServletException

public UnavailableException(String message);
public UnavailableException(String message, int sec);

public int getUnavailableException();
public Boolean isPermanent();

// newly deprecated methods
public UnavailableException(servlet servlet, String
     message);
public UnavailableException(int sec, Servlet servlet,
     String msg);

public Servlet getDervlet();
```

12.2 Package javax.servlet.http

```
interface HttpServletRequest
interface HttpServletResponse
interface HttpSession
interface HttpSessionBindingListener
interface HttpSessionContext

class cookie
class HttpServlet
class HttpSessionBindingEvent
class HttpUtils
```

12.2.1 HttpServletRequest

```
public interface HttpServletRequest extends
     ServletRequest;

public String getAuthType();
public Cookie[] getCookies();
public long getDateHeader(String name);
public String getHeader(String name);
public Enumeration getHeaders(String name);
public Enumeration getHeaderNames();
public int getIntHeader(String name);
public String getMethod();
public String getContextPath();
public String getPathInfo();
public String getPathTranslated();
public String getQueryString();
public String getRemoteUser();
public Boolean isUserInRole(String role);
public java.sequrity.principal getUserPrincipal();
public String getRequestedSessionId();
public boolean isRequestedSessionIdValid();
public Boolean isRequestedSessionIdFromCookie();
public Boolean isRequestedSessionIdFromURL();
public String getRequestURI();
public String getServletPath();
public HttpSession getSession();
public HttpSession getSession(Boolean create);

// deprecated methods
public Boolean isRequestSessionIdFromUrl();
```

12.2.2 HttpServletResponse

```
public interface HttpServletResponse extends
     ServletResponse
<<< STATUS CODE 416 AND 417 REPORTED MISSING>>>

public static final int SC_CONTINUE;
public static final int SC_SWITCHING_PROTOCOLS;
public static final int SC_OK;
public static final int SC_CREATED;
public static final int SC_ACCEPTED;
public static final int SC_NON_AUTHORITATIVE_INFORMATION;
public static final int SC_NO_CONTENT;
public static final int SC_RESET_CONTENT;
public static final int SC_PARTIAL_CONTENT;
public static final int SC_MULTIPLE_CHOICES;
public static final int SC_MOVED_PERMANENTLY;
public static final int SC_MOVED_TEMPORARILY;
public static final int SC_SEE_OTHER;
public static final int SC_NOT_MODIFIED;
public static final int SC_USE_PROXY;
public static final int SC_BAD_REQUEST;
public static final int SC_UNAUTHORIZED;
public static final int SC_ PAYMENT_REQUIRED;
public static final int SC_FORBIDDEN;
public static final int SC_NOT_FOUND;
public static final int SC_METHOD_NOT_ALLOWED;
public static final int SC_NOT_ACCEPTABLE;
public static final int SC_PROXY_AUTHENTICATION_REQUIRED;
public static final int SC_REQUEST_TIMEOUT;
public static final int SC_CONFLICT;
public static final int SC_GONE;
public static final int SC_ LENGTH_REQUIRED;
public static final int SC_PRECONDITION_FAILED;
public static final int SC_REQUEST_ENTITY_TOO_LARGE;
public static final int SC_REQUEST_URI_TOO_LONG;
public static final int SC_UNSUPPORTED_MEDIA_TYPE;
public static final int
```

```
    SC_REQUESTED_RANGE_NOT_SATISFIABLE;
public static final int SC_EXPECTATION_FAILED;
public static final int SC_INTERNAL_SERVER_ERROR;
public static final int SC_NOT_IMPLEMENTED;
public static final int SC_BAD_GATEWAY;
public static final int SC_SERVICE_UNAVAILABLE;
public static final int SC_GATEWAY_TIMEOUT;
public static final int SC_HTTP_VERSION_NOT_SUPPORTED;

public void addCookie(Cookie cookie);
public Boolean containsHeader(String name);
public String encodeURL(String url);
public String encodeRedirectURL(String url);
public void sendError(int status) throws IOException;
public void sendError(int status, String message) throws
    IOException;
public void sendRedirect(String location) throws
    IOException;
public void setDateHeader(String headername, long date);
public void setHeader(String headername, String value);
public void addHeader(String headername, String value);
public void addDateHeader(String headername, long date);
public void addIntHeader(String headername, int value);
public void setIntHeader(String headername, int value);
public void setStatus(int statuscode);

// deprecated methods
public void setStatus(int statuscode, String message);
public String encodeUrl(String url);
public String encodeRedirectUrl(String url);
```

12.2.3 HttpSession

```
public interface HttpSession
```

```
public long getCreationTime();
public String getId();
public long getLastAccessedTime();
public boolean isNew();
public int getMaxInactiveInterval();
public void setMaxInactiveInterval(int interval);
public Object getAttribute(String name);
public Enumeration getAttributeNames();
public void setAttribute(String name, Object attribute);
public void removeAttribute(String name);
public  void invalidate();

// deprecated methods
public Object getValue(String name);
public String[] getValueNames();
public void putValue(String name, Object value);
public void removeValue(String name);
public HttpSessionContext getSessionContext();
```

12.2.4 HttpSessionBindingListener

```
public interface HttpSessionBindingListener extends
     EventListener

public void valueBound(HttpSessionBindingEvent event);
public void valueUnbound(HttpSessionBindingEvent event);
```

12.2.5 HttpSessionContext

```
// deprecated
public abstract interface HttpSessionContext

// deprecated methods
public void Enumeration getIds();
```

```
public HttpSession getSession(String id);
```

12.2.6 Cookie

```
public class Cookie implements Colnable

public Cookie(String name, String value);
public void setComment(String comment);
public String getComment();
public void setDomain(String domain);
public String getDomain();
public void setMaxAge(int expiry);
public int getMaxAge();
public void setPath(String uriPath);
public String getPath();
public void getSecure();
public boolean getSecure();
public String getName();
public void setValue(String value);
public String getValue();
public int getVersion();
public void setVersion(int version);
public Object clone();
```

12.2.7 HttpServlet

```
public abstract class HttpServlet extends GenericServlet
     implements Serializable

public HttpServlet();

protected void doGet(HttpServletRequest req,
     HttpServletResponse res) throws ServletException,
     IOException;
```

```
protected void doPost(HttpServletRequest req,
     HttpServletResponse res) throws ServletException,
     IOException;
protected void doPut(HttpServletRequest req,
     HttpServletResponse res) throws ServletException,
     IOException;
protected void doDelete(HttpServletRequest req,
     HttpServletResponse res) throws ServletException,
     IOException;
protected void doOptions(HttpServletRequest req,
     HttpServletResponse res) throws ServletException,
     IOException;
protected void doTrace(HttpServletRequest req,
     HttpServletResponse res) throws ServletException,
     IOException;
protected void service(HttpServletRequest req,
     HttpServletResponse res) throws ServletException,
     IOException;
public void service(HttpServletRequest req,
     HttpServletResponse res) throws ServletException,
     IOException;
protected lond getLastModified(HttpServletRequest req);
```

12.2.8 HttpSessionBindingEvent

```
public class HttpSessionBindingEvent extends EventObject

public HttpSessionBindingEvent(HttpSession session, String
     name);

public String getName();
public HttpSession getSession();
```

12.2.9 HttpUtils

```
public class HttpUtils

public HttpUtils();

public static Hashtable parseQueryString(String
     queryString);
public static Hashtable parsePostData(int length,
     ServletInputStream in);
public static StringBuffer
     getRequestURL(HttpServletRequest req);
```

第 13 章　ディプロイメント デスクリプタ (Deployment Descriptor)

ディプロイメント デスクリプタは、**開発者**、**アセンブラ**、**ディプロイヤ**間でのあるアプリケーションの要素と設定に関する情報を交わすものである。

13.1　ディプロイメント デスクリプタのエレメント (Deployment Descriptor Elements)

本ウェブ アプリケーション ディプロイメント デスクリプタには、以下のタイプの設定と導入の情報が存在する：

- ServletContext 初期化パラメタ（ServletContext Init Parameters）
- セッションの設定(Session Configuration)
- Servlet / JSP 定義（Servlet / JSP Definitions）
- Servlet / JSP マッピング（Servlet / JSP Mappings）
- MIME タイプ マッピング(Mime Type Mapping)
- ウェルカム ファイルのリスト(Welcome File list)
- エラー ページ(Erroe Pages)
- セキュリティ(Security)

これらのエレメントの詳細は、DTD コメントを参照のこと。

13.1.1　ディプロイメント デスクリプタの DOCTYPE (Deployment Descriptor DOCTYPE)

全ての有効なウェブ アプリケーション ディプロイメント デスクリプタは、以下の DOCTYPE 宣言を含まねばならない：

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
```

13.2 DTD (DTD)

以下の DTD で、ウェブ アプリケーション ディプロイメント デスクリプタの XML 文法を定義する。

<!--
web-app エレメントはウェブ アプリケーションのディプロイメント デスクリプタのルートである
-->

```
<!ELEMENT  web-app (icon?, display-name?, description?,
distributable?, context-param*, servlet*, servlet-mapping*,
session-config?, mime-mapping*, welcome-file-list?, error-page*,
taglib*, resource-ref*, security-constraint*, login-config?,
security-role*, env-entry*, ejb-ref*)>
```

<!--
icon エレメントは small-icon と large-icon エレメントからなり、GUI ツール内でそのウェブアプリケーションを代表させるために用いる小型及び大型のアイコンの場所を指定する。ツールは少なくとも GIF と JPEG イメージを受理できること。
-->

```
<!ELEMENT icon (small-icon?, large-icon?)>
```

<!--
small-icon エレメントには小さな（16×16 ピクセル）アイコンのイメージを含むファイルのそのウェブ アプリケーション内の位置をいれる。
-->

```
<!ELEMENT small-icon (#PCDATA)>
```

<!--
large-icon エレメントには大きな（32×32 ピクセル）アイコンのイメージを含むファイルのそのウェブ アプリケーション内の位置を入れる。

```
-->

<!ELEMENT large-icon (#PCDATA)>

<!--
display-name エレメントには GUI ツールで表示されることを意図した短い名前を含める。
-->

<!ELEMENT display-name (#PCDATA)>

<!--
description エレメントは、その親エレメントに関して記述したテキストを提供するために用いられる
-->

<!ELEMENT description (#PCDATA)>

<!--
distributable エレメントは、これがそのそのウェブ アプリケーションのディプロイメント デスクリプタに存在することにより、このウェブ アプリケーションが分散したサーブレット コンテナに導入されるべく適切にプログラムされていることを示す。
-->

<!ELEMENT distributable EMPTY)>

<!--
context-param エレメントには、そのウェブ アプリケーションのサーブレット コンテキストの初期化パラメタの宣言を含める。
-->

<!ELEMENT context-param (param-name, param-value, description?)>

<!--
param-name エレメントには、パラメタの名前を入れる。
-->
```

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

<!--
param-value エレメントには、パラメタの値を入れる。
-->

<!ELEMENT param-value (#PCDATA)>

<!--
servlet エレメントには、サーブレットの宣言データを含める。jsp-file が指定され、load-on-startup エレメントが存在するときは、この JSP はあらかじめコンパイルされ、ロードされなければならない。
-->

<ELEMENT servlet (icon?, servlet-name, display-name?,
description?, (servlet-class|jsp-file), init-param*, load-on-
startup?, security-role-ref*)>

<!--
servlet-name エレメントには、そのサーブレットのカノニカル名（公式名）を入れる。
-->

<!ELEMENT servlet-name (#PCDATA)>

<!--
servlet-class エレメントには、そのサーブレットの完全な公式クラス名(FQCN)を含める
-->

<!ELEMENT servlet-class (#PCDATA)>

<!--
jsp-file エレメントには、そのウェブ アプリケーション内の JSP ファイルの完全なパスを含める。
-->
```

```
<!ELEMENT jsp-file (#PCDATA)>

<!--
init-param エレメントには、そのサーブレットの初期化パラメタとしての名前／値のペアを含める。
-->

<!ELEMENT init-param (param-name, param-value, description?)>

<!--
load-on-startup エレメントは、このサーブレットが本ウェブ アプリケーションのスタートアップ時にロードされねばならないことを示す。これらのエレメントのオプション部分のコンテンツは正の整数で、このサーブレットがロードされる順番を示す。小さな番号ののほうが高い番号より先にロードされる。値が指定されていないとき、あるいは指定された値が正の整数でないときは、コンテナはスタートアップ シーケンスの際に何時でもロードできる自由を持つ。
-->

<!ELEMENT load-on-startup (#PCDATA)>

<!--
servlet-mapping エレメントは、サーブレットと URL パタンとのマッピングを指定する。
-->

<!ELEMENT servlet-mapping (servlet-name, url-pattern)>

<!--
url-pattern エレメントには、そのマッピングの URL パタンを含める。
-->

<!ELEMENT url-pattern (#PCDATA)>

<!--
session-config エレメントには、のセッションのパラメタを定める。
-->
```

<!ELEMENT session-config (session-timeout?)>

<!--
session-timeout エレメントは、本ウェブ アプリケーションで作られる全てのセッションのデフォルトのセッション タイムアウト インターバルを定める。指定するタイムアウト値は分を示す整数で記述のこと。
-->

<!ELEMENT session-timeout (#PCDATA)>

<!--
mime-mapping エレメントは、拡張子(extension)と MIME タイプ(mime type)とのマッピングを指定する。
-->

<!ELEMENT mime-mapping (extension, mime-type)>

<!--
extension エレメントは、拡張子を示す文字列を入れる。例えば：”txt”
-->

<!ELEMENT extension (#PCDATA)>

<!--
mime-type エレメントには、指定する MIME タイプを含める。例えば：”text/plain”
-->

<!ELEMENT mime-type (#PCDATA)>

<!--
welcome-file-list エレメントには、ウェルカムファイルのエレメントの順序付けされたリストを含める。
-->

<!ELEMENT welcome-file-list (welcome-file+)>

<!--
welcome-file エレメントには、例えば index.html のようなデフォルトのウェルカムファイルのファイル名を含める。
-->

<!ELEMENT welcome-file (#PCDATA)>

<!--
taglib エレメントは、JSP のタグ ライブラリを記述するのに用いる。
-->

<!ELEMENT taglib (taglib-uri, taglib-location)>

<!--
taglib-uri エレメントでは、web.xml ドキュメントの場所に相対的で、そのウェブ アプリケーションで使われる TAG ライブラリを指定する URI を記述する。
-->

<!ELEMENT taglib-uri (#PCDATA)>

<!--
taglib-location エレメントには、そのタグ ライブラリのタグ ライブラリ記述ファイル (Tag Library Description file)を求めるための場所（そのウェブ アプリケーションのルートに相対的な）を入れる。
-->

<!ELEMENT taglib-location (#PCDATA)>

<!--
error-page エレメントには、エラーコードまたは例外のタイプとそのウェブ アプリケーションのリソースのパスとのマッピングを入れる。
-->

<!ELEMENT error-page ((error-code | exception-type), location)>

<!--

error-code エレメントには、HTTP のエラーコード、例えば 404、を含める。
-->

<!ELEMENT error-code (#PCDATA)>

<!--
exception-type エレメントには、Java の例外タイプの完全な公式クラス名(FQCN)を含める。
-->

<!ELEMENT exception-type (#PCDATA)>

<!--
location エレメントには、そのウェブ アプリケーションのリソースの場所を含める。
-->

<!ELEMENT location (#PCDATA)>

<!--
resource-ref エレメントには、そのウェブ アプリケーションの外部リソースへの参照の宣言を含める。
-->

<!ELEMENT resource-ref (description?, res-ref-name, res-type, res-auth)>

<!--
resource-ref-name エレメントには、リソース ファクトリの参照名指定する。
-->

<!ELEMENT resource-ref-name (#PCDATA)>

<!--
res-type エレメントには、データ リソースの（Java クラス）タイプを規定する。
-->

<!ELEMENT res-type (#PCDATA)>

<!--
res-auth エレメントは、そのアプリケーション コンポネント コードがリソースのサインオンをプログラムで実施するか、または、デプロイヤで供給されるプリンシプルのマッピング情報に基づきそのリソースへのサインオンをコンテナが実施するかを示す 。
CONTAINER か SERVLET のどちらかでなければならない。
-->

<!ELEMENT res-auth (#PCDATA)>

<!--
security-constraint エレメントは、ひとつまたはそれ以上のリソースのコレクションに対し、セキュリティ制約を結びつけるのに用いる。
-->

<!ELEMENT security-constraint (web-resource-collection+, auth-constraint?, user-data-constraint?)>

<!--
security-resource-collection エレメントは、リソースとそれらのリソースに対する HTTP メソッドのサブセットを作り、それにひとつのセキュリティ制約を課すのに用いる。HTTP メソッドが指定されないときは、そのセキュリティ制約は全ての HTTP メソッドに適用される。
-->

<!ELEMENT web-resource-collection (web-resource-name, description?, url-pattern*, http-method*)>

<!--
web-resource-name エレメントには、このウェブ リソース コレクションの名前を入れる。
-->

<!ELEMENT web-resource-name (#PCDATA)>

<!--
http-method エレメントには、HTTP のメソッド(GET | POST | ...)を入れる。
-->

<!ELEMENT http-method (#PCDATA)>

<!--
user-data-constraint エレメントは、クライアントとコンテナ間のデータ通信がどのように保護さるべきかを示すのに用いる。
-->

<!ELEMENT user-data-constraint (description?, transport-guarantee)>

<!--
transport-guarantee エレメントは、クライアントとサーバ間の通信が
NONE、INTEGRAL あるいは CONFIDENTIAL のいづれであるかを指定する。NONE はそのアプリケーションがトランスポートの保証を必要としないことを意味する 。
INTERGAL の値は、そのアプリケーションがクライアントとサーバ間のデータ通信がそれが伝達される間に変更を加えられないような方法で行うことを要求していることを意味する。CONFIDENTIAL は、伝送中に他者からそのコンテンツを覗かれることを防止する方法でデータが通信されることを要求していることを意味する。ほとんどの場合 、INTEGRAL または CONFIDENTIAL フラグが存在すると、それは SSL を要求していることを示すことになろう。
-->

<!ELEMENT transport-guarantee (#PCDATA)>

<!--
auth-constraint エレメントは、このリソースのコレクションへのアクセスが許可されるべきユーザのロールを示す。ここで用いられるロールは security-role-ref エレメントに現れていなければならない。
-->

<!ELEMENT auth-constraint (description?, role-name*)>

<!--
role-name エレメントには、セキュリティ ロールの名前を入れる。
-->

<!ELEMENT role-name (#PCDATA)>

<!--
login-config エレメントには、使用すべき認証法、本アプリケーションで使われるべきレルム、及びフォームのログインのメカニズムで必要とする属性を設定するのに用いられる。
-->

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

<!--
realm-name エレメントでは、HTTP 基本認証で用いるレルム名を指定する。
-->

<!ELEMENT realm-name (#PCDATA)>

<!--
form-login-config エレメントは、フォーム ベースのログインで使うべきログインとエラーのページを指定する。フォーム ベースの認証が使われないときは、これらの要素は無視される。
-->

<!ELEMENT form-login-config (form-login-page, form-error-page)>

<!--
form-login-page エレメントは、ウェブ アプリケーションにおいて、ログインに使えるページの場所を指定する。
-->

<!ELEMENT form-login-page (#PCDATA)>

<!--
form-error-page エレメントは、ウェブ アプリケーションにおいて、ログインに失敗したときに表示するエラー ページの場所を指定する。
-->

<!ELEMENT form-error-page (#PCDATA)>

<!--
auth-method エレメントは、このウェブ アプリケーションの認証メカニズムを設定するのに用いる。認証の制約で保護されたウェブ リソースへのアクセスを取得する前提条件として、ユーザはこれで設定された認証メカニズムで認証されねばならない。このエレメントとして有効な値は”BASIC”、”DIGEST”、”FORM”または”CLIENT-CERT”である。
-->

<!ELEMENT auth-method (#PCDATA)>

<!--
security-role エレメントには、このウェブ アプリケーションに置かれた security-constraints のなかで使われるセキュリティ ロールの宣言を入れる。
-->

<!ELEMENT sequrity-role (description?, role-name)>

<!--
role-name エレメントにはロールの名前を入れる。このエレメントは空でない文字列を含んでいなければならない。
-->

<!ELEMENT sequrity-role-ref (description?, role-name, role-link)>

<!--
role-link エレメントは、セキュリティ ロールの参照を指定したセキュリティ ロールにリンクさせるのに使う。role-link エレメントは、security-role エレメントで指定したセキュリティ ロールのひとつの名前が入らねばならない。
-->

```
<!ELEMENT role-link (#PCDATA)>

<!--
env-entry エレメントには、アプリケーションの環境エントリの宣言を入れる。このエレメントは J2EE 対応のサーブレット コンテナで必要とされる。
-->

<!ELEMENT env-entry (description?, env-entry-nane?, env-entry-value?, env-entry-type)>

<!--
env-entry-nane エレメントは、アプリケーションの環境エントリの名前を入れる。
-->

<!ELEMENT env-entry-nane (#PCDATA)>

<!--
env-entry-value エレメントは、アプリケーションの環境エントリの値を入れる。
-->

<!ELEMENT env-entry-value (#PCDATA)>

<!--
env-entry-type エレメントは、環境エントリ値の完全な形式の Java のタイプで、そのアプリケーションのコードが予期しているものである。以下のものが env-entry-type として有効である： java.lang.Boolean, java.lang.String, java.lang.Integer, java.lang.Double, java.lang.Float.
-->

<!ELEMENT env-entry-type (#PCDATA)>

<!--
ejb-ref エレメントは、エンタープライズ ビーンへの参照を宣言するのに用いる。
-->
```

<!ELEMENT ejb-ref (description?, ejb-ref-nane, ejb-ref-type, home, remote, jb-link?)>

<!--
ejb-ref-nane エレメントは、EJB 参照の名前を入れる。これはエンタープライズ ビーンへの参照をサーブレット コードが取得するときに用いる JNDI 名である。
-->

<!ELEMENT ejb-ref-nane (#PCDATA)>

<!--
ejb-ref-type エレメントは、参照される EJB の予期される Java のクラスのタイプを入れる。
-->

<!ELEMENT ejb-ref-type (#PCDATA)>

<!--
ejb-home エレメントは、EJB の home インターフェイスの正規の形式の名前を入れる。
-->

<!ELEMENT home (#PCDATA)>

<!--
ejb-remote エレメントは、EJB の remote インターフェイスの正規の形式の名前を入れる。
-->

<!ELEMENT remote (#PCDATA)>

<!--
ejb-link エレメントは、ejb-ref エレメントで、ある EJB 参照がそれを包含している Java2 エンタープライズ エディション(J2EE: Java2 Enterprize Edition)アプリケーション パッケージにリンクしていることを規定するのに用いられる。ejb-link エレメントの値は、J2EE アプリケーション パッケージにある EJB の ejb-name で泣ければならない。

```
-->

<!ELEMENT ejb-link (#PCDATA)>

<!--
ID のメカニズムにより、デプロイメント デスクリプタのエレメントへのツール固有の
参照を容易にするものである。これにより、更なる追加のデプロイメントの情報（即ち標
準のデプロイメント デスクリプタ情報を超えた情報）ををを生成するツールが、非標準の
情報を分離されたファイルに蓄積し、そしてこれらツール固有のファイルから標準のウ
ェブ アプリケーション デプロイメント デスクリプタにある情報への参照を容易にす
る。
-->

<!ATTLIST web-app id ID #IMPLIED>
<!ATTLIST icon id ID #IMPLIED>
<!ATTLIST small-icon id ID #IMPLIED>
<!ATTLIST large-icon id ID #IMPLIED>
<!ATTLIST display-name id ID #IMPLIED>
<!ATTLIST description id ID #IMPLIED>
<!ATTLIST distributable id ID #IMPLIED>
<!ATTLIST context-param id ID #IMPLIED>
<!ATTLIST param-name id ID #IMPLIED>
<!ATTLIST param-value id ID #IMPLIED>
<!ATTLIST servlet id ID #IMPLIED>
<!ATTLIST servlet-name id ID #IMPLIED>
<!ATTLIST servlet-class id ID #IMPLIED>
<!ATTLIST jsp-file id ID #IMPLIED>
<!ATTLIST init-param id ID #IMPLIED>
<!ATTLIST load-on-startup id ID #IMPLIED>
<!ATTLIST servlet-mapping id ID #IMPLIED>
<!ATTLIST url-pattern id ID #IMPLIED>
<!ATTLIST session-config id ID #IMPLIED>
<!ATTLIST session-timeout id ID #IMPLIED>
<!ATTLIST mime-mapping id ID #IMPLIED>
<!ATTLIST extension id ID #IMPLIED>
<!ATTLIST mime-type id ID #IMPLIED>
```

```
<!ATTLIST welcome-file-list id ID #IMPLIED>
<!ATTLIST welcome-file id ID #IMPLIED>
<!ATTLIST taglib id ID #IMPLIED>
<!ATTLIST taglib-uri id ID #IMPLIED>
<!ATTLIST taglib-location id ID #IMPLIED>
<!ATTLIST error-page id ID #IMPLIED>
<!ATTLIST error-code id ID #IMPLIED>
<!ATTLIST exception-type id ID #IMPLIED>
<!ATTLIST location id ID #IMPLIED>
<!ATTLIST resource-ref id ID #IMPLIED>
<!ATTLIST res-ref-name id ID #IMPLIED>
<!ATTLIST res-type id ID #IMPLIED>
<!ATTLIST res-auth id ID #IMPLIED>
<!ATTLIST security-constraint id ID #IMPLIED>
<!ATTLIST web-resource-collection id ID #IMPLIED>
<!ATTLIST web-resource-name id ID #IMPLIED>
<!ATTLIST http-method id ID #IMPLIED>
<!ATTLIST user-data-constraint id ID #IMPLIED>
<!ATTLIST transport-guarantee id ID #IMPLIED>
<!ATTLIST auth-constraint id ID #IMPLIED>
<!ATTLIST role-name id ID #IMPLIED>
<!ATTLIST login-config id ID #IMPLIED>
<!ATTLIST realm-name id ID #IMPLIED>
<!ATTLIST form-login-config id ID #IMPLIED>
<!ATTLIST form-login-page id ID #IMPLIED>
<!ATTLIST form-error-page id ID #IMPLIED>
<!ATTLIST auth-method id ID #IMPLIED>
<!ATTLIST security-role id ID #IMPLIED>
<!ATTLIST security-role-ref id ID #IMPLIED>
<!ATTLIST role-link id ID #IMPLIED>
<!ATTLIST env-entry id ID #IMPLIED>
<!ATTLIST env-entry-name id ID #IMPLIED>
<!ATTLIST env-entry-value id ID #IMPLIED>
<!ATTLIST env-entry-type id ID #IMPLIED>
<!ATTLIST ejb-ref id ID #IMPLIED>
<!ATTLIST ejb-ref-name id ID #IMPLIED>
```

```
<!ATTLIST ejb-ref-type id ID #IMPLIED>
<!ATTLIST home id ID #IMPLIED>
<!ATTLIST remote id ID #IMPLIED>
<!ATTLIST ejb-link id ID #IMPLIED>
```

13.3 事例 (Examples)

以下の事例で上記 DTD に記載された定義の使用法を示す。

13.3.1 ベーシックな事例 (Basic Example)

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.2//EN" http://java.sun.com/j2ee/dtds/web-
app_2_2.dtd>
<web-app>
  <display-name>A Simple Application</display-name>
  <context-param>
    <param-name>Webmaster</param-name>
    <param-value>webmaster@mycorp.com</param-value>
  </context-param>
  <servlet>
    <servlet-name>catalog</servlet-name>
    <servlet-class>com.mycorp.CatalogServlet</servlet-class>
    <init-param>
      <param-name>catalog</param-name>
      <param-value>Sptiong</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>catalog</servlet-name>
    <url-pattern>/catalog/*</url-pattern>
  </servlet-mapping>
  <session-config>
```

```
    <session-timeout>30</session-timeout>
  </session-config>
  <mime-mapping>
    <extension>pdf</extension>
    <mime-type>application/pdf</mime-type>
  </mime-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-fille>
  </welcome-file-list>
  <error-page>
    <error-code>404</error_code>
    <location>404.html</location>
  </error-page>
</web-app>
```

13.3.2 セキュリティ事例 (An Example of Security)

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.2//EN" http://java.sun.com/j2ee/dtds/web-
appweb_2_2.dtd>
<web-app>
  <display-name>A Secure Application</display-name>
  <security-role>
    <role-name>manager</role-name>
  </security-role>
  <servlet>
    <servlet-name>catalog</servlet-name>
    <servlet-class>com.mycorp.CatalogServlet</servlet-class>
    <init-param>
      <param-name>catalog</param-name>
      <param-value>Spring</param-value>
    </init-param>
    <security-role-ref>
```

```
      <role-name>MGR</role-name>← role name used in code →
      <role-link>manager</role-link>
    </security-role-ref>
  </servlet>
  <servlet-mapping>
    <servlet-name>catalog</servlet-name>
    <url-pattern>/catalog/*</url-pattern>
  </servlet-mapping>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>SalesInfo</web-resource-name>
      <url-pattern>/salesinfo/*</url-pattern>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
      <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
      </user-data-constraint>
      <auth-constraint>
        <role-name>manager</role-name>
      </auth-constraint>
    </web-resource-collection>
  </security-constraint>
</web-app>
```

第 14 章　　将来 (Futures)

我々のパートナーや一般からの貢献者たちにより、本仕様への追加に対して多くの優れた提言がなされた。市場に出す時間の配慮から、本仕様への全ての改版事項にもかけうる作業量に制約があり、本仕様のこの版にはこれらの提言の幾つかは組み入れることが出来なかった。しかしながら、これらの事項を将来の指示書として記す事で、本仕様書の将来の改版にはこれらを含めることを検討することを示したい。

以下に検討中の項目を示す：

応答コンテンツのフィルタリング
ServletContext インターフェイスの getResource メソッドで国際化されたコンテンツを容易にアクセス出来るようにする
ウェブ アプリケーションのコンテンツの国際化
WebDAV のインテグレーション
アプリケーション イベント ハンドラ
HTTP 拡張フレームワークのインテグレーション

このリストの項目の取り込みはこの仕様書の将来の版への取入れを約束するものではないことに注意されてい。これらの事項は真剣に検討中で、多分将来の版に含まれるであろうことを示しているだけである。