

JBoss Web Server User's Guide

A high performance deployment platform Java EE, PHP and
CGI applications

Table of Contents

1. About JBoss Web	1
2. Introducing JBoss Web	2
3. Getting Started	4
3.1. Download Software	4
3.2. Running JBoss Web	4
3.3. Deploying Web Applications	5
3.4. Run JBoss Web as a Windows Service	5
4. Architecture Overview	7
4.1. Core Server Components	7
4.1.1. Apache Tomcat	7
4.1.2. JBoss Cache	8
4.1.3. JBoss JTA	8
4.1.4. JBoss JCA	8
4.1.5. JBoss Microkernel	8
4.2. Modules Support	8
4.2.1. Tomcat Native	9
4.2.2. Native Proxy	10
4.2.3. Proxy Stream	11
4.2.4. Native Module API	11
4.2.5. URL Rewrite	11
4.3. Configuration files in JBoss Web Server	11
5. Configuring Connectors	13
5.1. The HTTP Connector	13
5.2. The HTTPS Connector	18
5.3. The AJP Connector	19
6. URL Rewriting	24
6.1. RewriteCond	25
6.2. RewriteRule	27
7. PHP	30
7.1. Installation	30
7.2. Use PHP scripts	31
7.3. PHP extension libraries	31

1

About JBoss Web

JBoss Web is a key component in the JBoss Enterprise Middleware Stack (JEMS). It runs Java EE, Microsoft ASP.Net, PHP, and CGI applications with native performance matching or even exceeding the Apache Web Server. JBoss Web will replace Tomcat and become the default servlet / JSP container in JBoss AS 5.x and later.

In this guide, we discuss how to install and use the JBoss Web Server. We cover JBoss Web configuration options, performance considerations, as well as how to run non-Java applications on the server.

This guide is work in progress. Please send your suggestions or comments to the JBoss Web user forum [<http://www.jboss.com/index.html?module=bb&op=viewforum&f=230>].

2

Introducing JBoss Web

JBoss Web Server provides enterprises with a single, high-performance deployment platform for Java EE, Microsoft ASP.NET, PHP and CGI script technologies. It is meant to be used as a replacement for the standard Web servers on all major platforms. JBoss Web is built on Apache Tomcat and Apache Portable Runtime (APR) technologies. It brings together the best of both worlds of Java EE servers and native HTTP servers.

Apache Tomcat is a full Java Servlet and Java Server Pages (JSP) server. It is a very popular Java application server and is the default servlet container inside the JBoss AS 4.x. However, the plain Apache Tomcat does not perform well in many areas, such as in supporting keep-alive connections, static contents, large files, HTTPS etc. JBoss Web solves those performance problems by leveraging a hybrid model with APR and Tomcat native technologies. The hybrid technology model offers the best from threading and event processing models from the latest OS technologies. As a result, JBoss Web server achieves scalability and performance characteristics that match or exceed the native Apache HTTP Server or IIS. For instance, SSL performance in JBoss Web is almost 4 times faster than that in plain Tomcat (Figure 2.1).

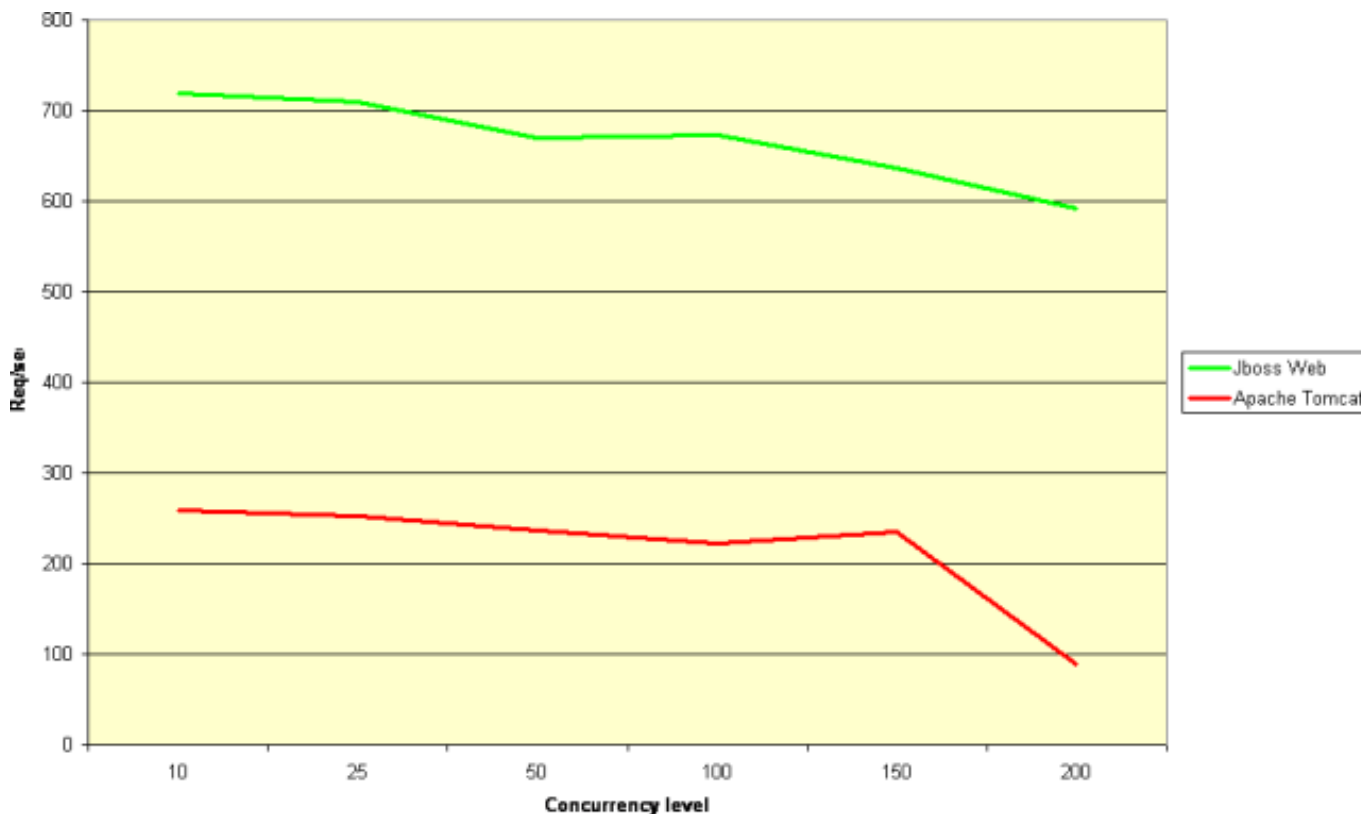


Figure 2.1. SSL benchmark (x86_64 GNU/Linux)

Aside from performance issues, another shortcoming of Apache Tomcat is that it is a limited integration platform.

Tomcat runs only Java applications. If you want to run PHP/CGI scripts side-by-side with your Java applications, you have to front Tomcat with the Apache Web Server or other native web servers. The APR-based hybrid model in JBoss Web supports both in- and out-of-process execution of CGI and PHP scripts. Furthermore, JBoss Web supports ASP.Net applications, and allows Java applications to access running .Net services and objects.

For Java EE application developers, JBoss Web is a complete replacement for the Apache Web Server + Tomcat + mod_jk stack.

Key features of the JBoss Web Server are as follows.

- **Full Java EE Support:** The power of JBoss Web comes with full Java EE support. Inheriting from Apache Tomcat, JBoss Web supports the JSP 2.0 and Servlet 2.4 specifications, providing a complete web development environment. With add-on JSF libraries (e.g., Apache MyFaces) and JBoss embeddable EJB3 libraries, you can deploy full Java EE 5.0 applications in JBoss Web. JBoss Web runs in the same VM as the JBoss application server, providing a complete environment for everything from simple web applications to complex high-end enterprise applications.
- **Highly Scalable:** JBoss Web scales to the levels required by the most demanding of applications. As we discussed, it out-performs Tomcat by several times and matches the performance of the native Apache Web Server and IIS. The hybrid connection model can handle client loads of 10,000+ concurrent connections.
- **Fast Static Content:** JBoss Web takes advantage of fast OS-specific features to achieve zero-copy transfer of static content. This decreases CPU load and improves application responsiveness.
- **Clustering Support:** JBoss Web can serve an amazing number of requests on a single machine, but when you need to scale beyond that, JBoss Web is there. JBoss web offers full clustering support for web applications. JBoss Web HTTP session replication is based on the powerful JBossCache technology, allowing your application state to be efficiently shared between all the nodes in your application cluster.
- **OpenSSL Support:** The security of your web application matters. SSL is the cornerstone of web security, but Java SSL engines are slow and can put a big burden on your CPU. JBossWeb integrates the native OpenSSL libraries, the fastest and most secure open source SSL implementation. The highly-optimized OpenSSL libraries also allow for hardware acceleration which yield a ten fold performance increase.
- **URL Rewriting:** JBoss Web provides a powerful URL rewriting module, similar to Apache mod_rewrite. URLs can be rewritten to support legacy URLs, handle errors or rapidly respond to web application issues that come up in the hectic day-to-day management of web applications.
- **Integration:** JBoss Web Server provides organizations with a single deployment platform for the most common types of web applications. Not only does JBoss Web support the latest Java web technologies, but also JBoss web can also support applications running on Microsoft .NET, PHP, and CGI. There is no longer any need to manage multiple servers for multiple web platforms. Legacy applications can maintained on JBoss Web, right along side your other applications. New development can take advantage of powerful features of any of these web platforms. JBoss Web gives you the power to choose the right platform for each task, and total flexibility to make changes along the way.

3

Getting Started

In this chapter, we will cover the steps to install the JBoss Web Server and deploy Java EE applications on it. If you are interested in running PHP, CGI, and ASP.Net applications on JBoss Web, please refer to ???, ???, and ??? respectively.

3.1. Download Software

JBoss Web is distributed as a standalone webserver running on top of the JBoss 4 microkernel. The distribution can be downloaded directly from the JBoss Web downloads page [<http://labs.jboss.com/portal/jbossweb/downloads>]. JBoss Web is not pure Java. It contains native code, compiled and optimized for each operating system. Download the package appropriate for your platform, paying particular attention to whether or not you are running on the 64-bit system.

Uncompress the distribution wherever you would like JBoss Web installed. The resulting directory (jbossweb-1.0.1.GA-linux-i686, for example) contains the JBoss Web instance.

3.2. Running JBoss Web

Because JBoss Web is running on the JBoss Microkernel, it can be started like any JBoss AS instance. Before starting, make sure that your JAVA_HOME environment variable is set to your Java install directory. From the bin directory, run the run.bat or run.sh script, as is appropriate for your platform. If you run this from a shell, you will see the JBoss console log scroll by. If JBoss Web started correctly, the last few lines of output should look like the following:

```
16:33:10,684 INFO [Http11AprProtocol] Starting Coyote HTTP/1.1 on http-172.16.129.72-8080
16:33:10,727 INFO [AjpAprProtocol] Starting Coyote AJP/1.3 on ajp-172.16.129.72-8009
16:33:10,738 INFO [Server] JBoss (MX MicroKernel) [4.0.4RC1 (build: CVSTag=JBoss_4_0_4_RC1 date=200602071
```

The last line is the message JBoss sends when all services are up and running, letting you know that everything is good. However, the two lines before it are important for JBoss Web. They tell you that JBoss Web is listening on two ports, 8080 and 8009. More importantly, you can see from the output that JBoss Web is using the APR(Apache portable runtime) libraries. This means that you are using the optimized native libraries. Without the APR libraries, you would see output more like the following, with no reference to the APR code.

```
16:47:40,154 INFO [Http11BaseProtocol] Starting Coyote HTTP/1.1 on http-0.0.0.0-8080
16:47:40,914 INFO [ChannelSocket] JK: ajp13 listening on /0.0.0.0:8009
16:47:41,064 INFO [JkMain] Jk running ID=0 time=0/299 config=null
```

In worse cases, like using the wrong version of the libraries for your platform, JBoss Web may completely fail to start. If there is an error, check the console log (the output in your terminal window) or the server log in `server/default/log/server.log` for more information about the problem.

To access your JBoss Web instance, go to `http://localhost:8080/` in your web browser. If everything went well, you will see the JBoss Web start welcome screen. Congratulations, your JBoss Web instance is ready to go.

3.3. Deploying Web Applications

Developing web applications on JBoss Web is easy. To deploy a web application, copy the WAR file to the `server/default/deploy` directory. After a few seconds, JBoss will deploy the web application automatically. The following output shows the log messages after deploying an application named `myapp.war`.

```
18:03:01,830 INFO [TomcatDeployer] deploy, ctxPath=/myapp,
warUrl=.../tmp/deploy/tmp53493myapp-exp.war/
```

The `ctxPath` value is `/myapp`, which means the application would immediately be accessible at `http://localhost:8080/myapp`.

Changes to the application can be made by copying a newer version of the WAR file into the deploy directory. If you need to remove an application, remove the WAR file from the deploy directory. JBoss Web will undeploy the application, noting the fact in the log.

```
18:06:58,086 INFO [TomcatDeployer] undeploy, ctxPath=/myapp,
warUrl=.../tmp/deploy/tmp53493myapp-exp.war/
```

3.4. Run JBoss Web as a Windows Service

JBoss Web comes with Windows service executable that can run JBoss Web Server as service. The service executable `jbossSvc.exe` transforms the `run.bat` and `shutdown.bat` batch scripts to services. This means that any change made to those scripts will be used both in service and command line mode.

To install the JBoss Web server as Windows service use the provided `service.bat` batch file.

```
C:\> cd c:\jbossweb-1.0.1.GA-windows-i586\bin
C:\> service.bat install
```

To start the JBoss Web server as Windows service use Control panel or `net start` command. When running in service mode the console output is redirected to the file `run.log`. You can inspect the file for any errors during service startup.

```
C:\> net start JBossWebServer
The JBoss Web Server 4 service is starting.
The JBoss Web Server 4 service was started successfully.
```

To stop the JBoss Web server as Windows service use Control panel or net stop command. When running in service mode the console output is redirected to the file shutdown.log. You can inspect the file for any errors during service shutdown.

```
C:\> net stop JBossWebServer
The JBoss Web Server 4 service was stopped successfully.
```

To restart the JBoss Web server as Windows service use Control panel.

To remove the JBoss Web server as Windows service use the provided service.bat batch file.

```
C:\> cd c:\jbossweb-1.0.1.GA-windows-i586\bin
C:\> service.bat uninstall
```

Service customization is done by editing the service.bat script. Each command has a separate section that you can customize. The most common customization task would be changing service names if more than one service instances per box are required.

4

Architecture Overview

In the previous chapter, you learned how to download, install, and run the JBoss Web Server. In this chapter, we will discuss the overall architecture of the server. It provides the basis for the understanding the configuration options and modules later in this guide.

4.1. Core Server Components

The core functionality of JBoss Web Server is provided by the Apache Tomcat. Apache Tomcat is embedded inside JBoss Application Server using Embedded Engine. This allows the seamless integration with JBoss components by using the underlying Microkernel system.

Additional modules allows to use the JBoss Web Server as drop-in replacement for standard native web servers, while offering the Reference Implementation for the Java Servlet and JavaServer Pages technologies. The Java Servlet and JavaServer Pages specifications are developed by Sun under the Java Community Process.

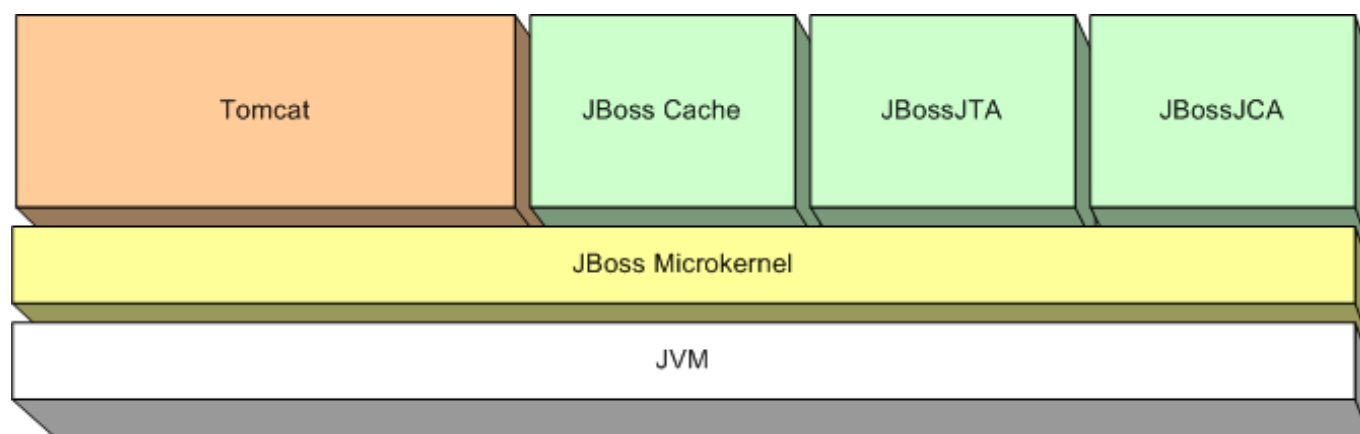


Figure 4.1. JBoss Web Architecture

Figure 4.1 shows the JBoss Web Server architecture. The brief description of architecture components is described in the following sections.

4.1.1. Apache Tomcat

JBoss Web Server currently uses the Apache Tomcat 5.5.x branch. While it supports the same Servlet and JSP Specification versions (i.e., servlet 2.4 and JSP 2.0) as Apache Tomcat 5.0.x, there are significant changes in many areas under the hood, resulting in improved performance, stability, and total cost of ownership.

The Java servlet 2.5 and JSP 2.1 specifications are to be supported by Tomcat 6.0.x, planned for the second half of

2006.

4.1.2. JBoss Cache

JBoss Cache is a replicated and transactional cache. It is replicated since multiple JBossCache instances can be distributed (within the same JVM or across several JVMs whether they reside on the same machine or on different machines on a network) and data is replicated across the whole group. It is transactional because a user can configure a JTA transaction manager and make the cache operation transactional. Note that the cache can also be run without any replication; this is the local mode.

Currently, it consists of two components: TreeCache and TreeCacheAop. TreeCache is a tree-structured cache that provides replication and transaction context, while TreeCacheAop extends the functionality of TreeCache but behaves as a true object cache providing transparent and finer-grained object mapping into internal cache.

4.1.3. JBoss JTA

A clean room implementation of the Java Transaction API part of the J2EE specification, JBoss JTA is made up of a number of a components and interfaces. The interfaces can be used when you change the transaction manager implementation (i.e., plugin a thirdparty implementation).

4.1.4. JBoss JCA

A clean room implementation of the J2EE Connector API part of the J2EE specification, JBoss JCA handles the deployment of resource adapters. These include DataSourcees and ConnectionFactorys and MessageListeners to and from databases, legacy systems or JMS Servers

4.1.5. JBoss Microkernel

The JBoss Microkernel is a set of MBeans that runs on top of JBossMX. It has been designed to be totally modular from the ground up. Support classes exists to extend the notion of an MBean into a Service or Deployer.

JBoss MX is JBoss implementation of the JMX technology. JBoss includes a clean room implementation of the JMX API's published by Sun Microsystems. While implementing all the required management functions it is also geared towards acting as a core library of the JBoss Microkernel. JBoss MX supports version 1.2 of the JMX specification (JSR 003) since JBoss versions 3.2.5 and 4.0.0. Support for the JMX Remote API (JSR 160) is under development.

4.2. Modules Support

JBoss Web Server comes with various extension modules that extends the core functionality of both Apache Tomcat standalone, and standard JBoss Application Server. Some modules are platform specific like Microsoft .NET module, and the others are cross platform, and requires a designated platform binaries. Figure 4.2 shows the JBoss Web Server extensions modules.

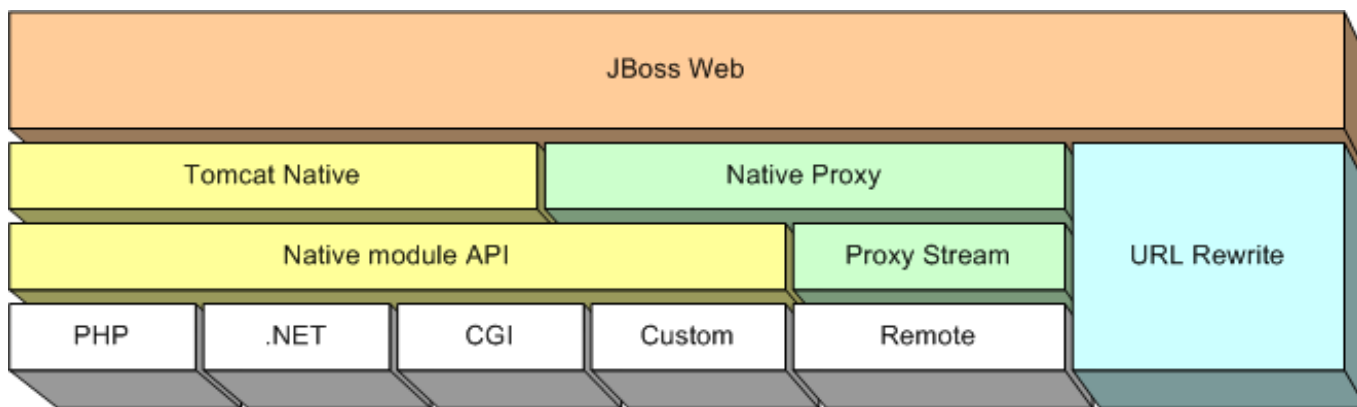


Figure 4.2. JBoss Web Extensions

4.2.1. Tomcat Native

Tomcat Native is JNI based library for Operating System Abstraction. The core functionality of Tomcat Native is provided by using Apache Portable Runtime (APR). The primary goal of APR is to provide an API to which software developers may code and be assured of predictable if not identical behavior regardless of the platform on which their software is built, relieving them of the need to code special-case conditions to work around or take advantage of platform-specific deficiencies or features.

SSL support is enabled by using the OpenSSL library. This gives much higher performance as well as hardware encryption software support than standard Java offers. It also gives existing users of Apache HTTPD a seamless transition from mod_ssl.

Tomcat Native uses APR and OpenSSL to provide superior scalability, performance, and better integration with native server technologies (see Figure 4.3). The Apache Portable Runtime is a highly portable library that is at the heart of Apache HTTP Server 2.x. APR has many uses, including access to advanced IO functionality (such as sendfile, epoll and TCP corking), OS level functionality (random number generation, system status, etc), and native process handling (shared memory, NT pipes and Unix domain sockets).

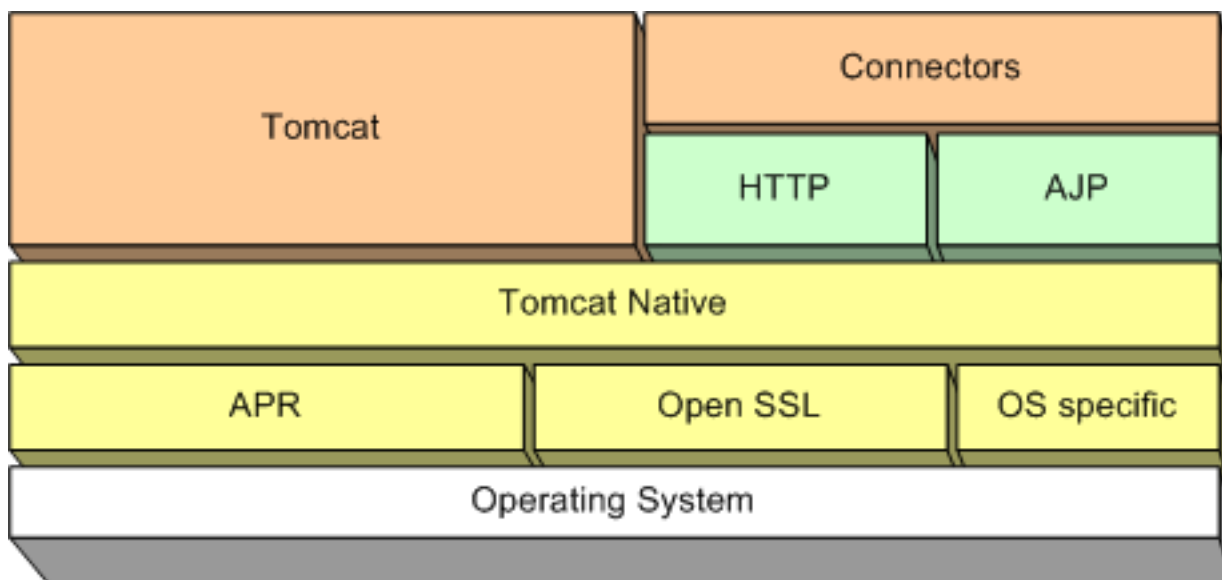


Figure 4.3. Tomcat Native

Tomcat Native comes as a set of libraries compiled for the designated Operating System. JBoss Web Server download comes with prebuild native libraries for the following platforms:

- WIN32
- WIN64/AMD64
- WIN64/IA64
- Linux i386
- Linux amd64
- Sun Solaris Sparc

4.2.2. Native Proxy

NativeProxy is Servlet abstraction that enables both in-process and out-of-process access to the legacy subsystems regardless of the method that is used for the integrated within the JBoss Web Server (see Figure 4.4). NativeProxy has integrated advanced application load balancer that offers both high availability and application segmentation for remotely based subsystems.

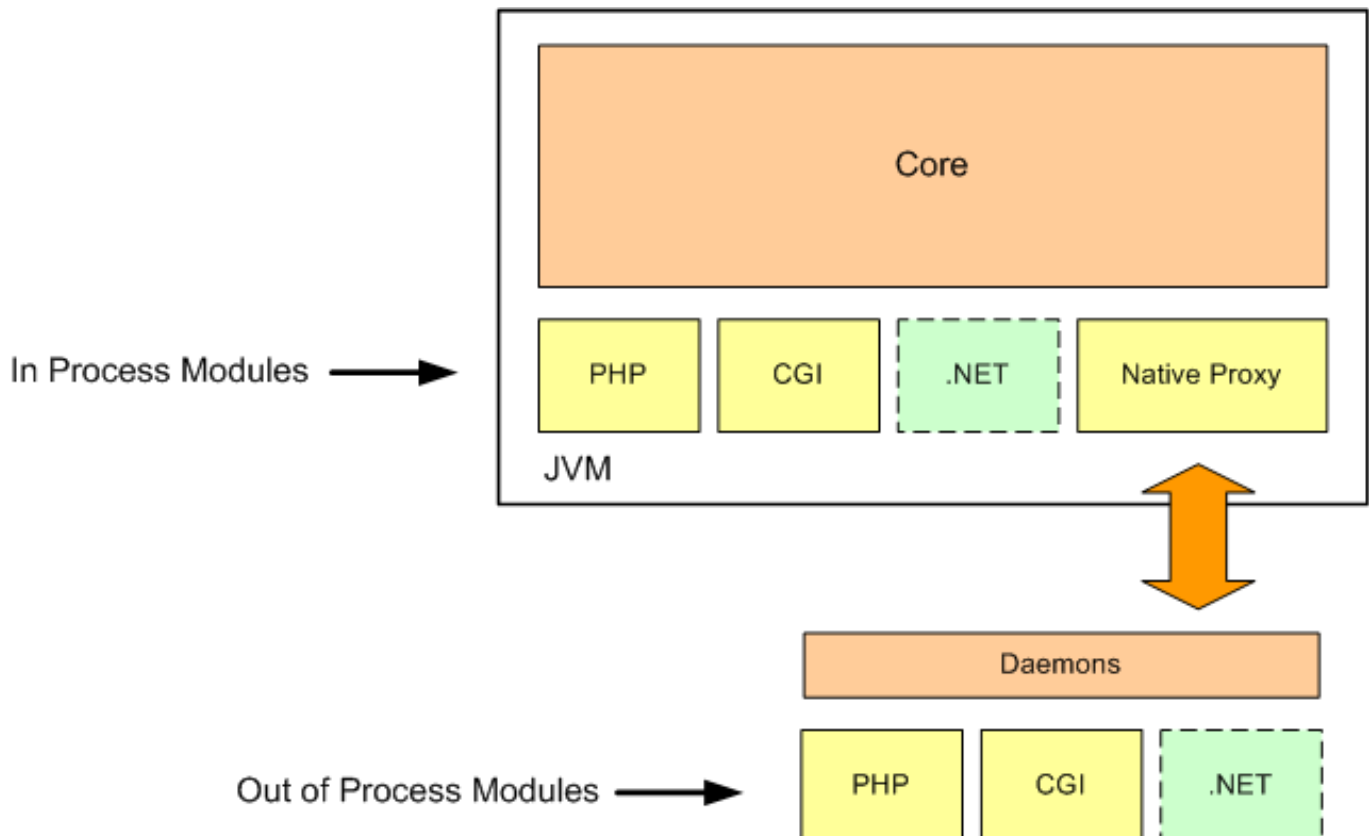


Figure 4.4. Native Proxy

Out of process modules are accessed using Native proxy. Each of the module is run as a daemon either on the local or remote machine. This gives much higher security and stability in case of faulty modules.

4.2.3. Proxy Stream

Proxy stream is a protocol very much similar to the Tomcat AJP/1.3 protocol, with additions for using Operating system advanced connection mechanisms like Unix Domain Sockets or Microsoft Windows Named Pipes. It offers both connection reuse and connection multiplexing. Unlike AJP/1.3 the data transferred can be encrypted, thus rising the security without the need for special network hardware and topology.

The main purpose for Proxy Stream Protocol is to give transparent access to the out-of-process legacy subsystems that can be hosted on the remote machine.

4.2.4. Native Module API

This component is JNI based native abstraction layer for various http centric legacy subsystem. It is responsible for loading the legacy applications inside the process address space of the running JVM.

4.2.5. URL Rewrite

This module uses a rule-based rewriting engine (based on a regular-expression parser) to rewrite requested URLs on the fly. It supports an unlimited number of rules and an unlimited number of attached rule conditions for each rule to provide a really flexible and powerful URL manipulation mechanism. The URL manipulations can depend on various tests, for instance server variables, environment variables, HTTP headers, time stamps and even external database lookups in various formats can be used to achieve a really granular URL matching.

4.3. Configuration files in JBoss Web Server

JBoss Web Server has the same configuration model as the JBoss Application Server. That gives you an easy transition from JBoss Web Server to full-blown Application Server. Figure 4.5 shows how the configuration files are organized in JBoss Web Server. Among them, the most important one is the server.xml file.

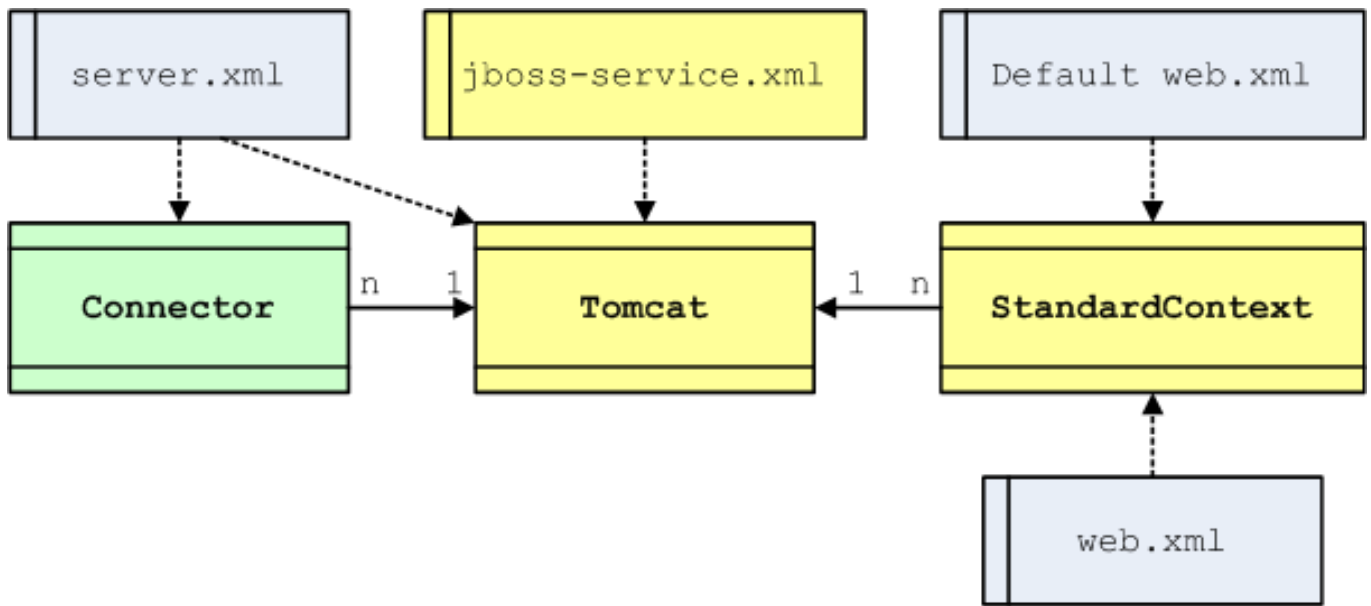


Figure 4.5. JBoss Web Architecture

5

Configuring Connectors

Connectors are crucial components in the JBoss Web Server. They are outward interface of the server. For instance, when a web browser client needs to connect to the server, it has to connect to one of the connectors. The connectors are configured as XML elements in the server.xml file. In this chapter, we will cover HTTP, HTTPS and AJP connectors.

5.1. The HTTP Connector

The HTTP connector uses sendfile for handling large static files (all such files will be sent asynchronously using high performance kernel level calls), and uses a socket poller for keepalive, increasing scalability of the server. Below is an example configuration for the HTTP connector in server.xml.

```
<Connector port="8080"
  maxThreads="250" strategy="ms" maxHttpHeaderSize="8192"
  emptySessionPath="true"
  enableLookups="false" redirectPort="8443" acceptCount="100"
  connectionTimeout="20000" disableUploadTimeout="true"/>
```

The following attributes are supported in the HTTP connector:

Table 5.1. Attributes in the HTTP Connector

Attribute	Description
acceptCount	The maximum queue length for incoming connection requests when all possible request processing threads are in use. Any requests received when the queue is full will be refused. The default value is 10.
address	For servers with more than one IP address, this attribute specifies which address will be used for listening on the specified port. By default, this port will be used on all IP addresses associated with the server.
allowTrace	A boolean value which can be used to enable or disable the TRACE HTTP method. If not specified, this attribute is set to false.
bufferSize	The size (in bytes) of the buffer to be provided for input streams created by this connector. By default, buffers of 2048 bytes will be provided.

Attribute	Description
compressableMimeType	The value is a comma separated list of MIME types for which HTTP compression may be used. The default value is text/html,text/xml,text/plain.
compression	The Connector may use HTTP/1.1 GZIP compression in an attempt to save server bandwidth. The acceptable values for the parameter is "off" (disable compression), "on" (allow compression, which causes text data to be compressed), "force" (forces compression in all cases), or a numerical integer value (which is equivalent to "on", but specifies the minimum amount of data before the output is compressed). If the content-length is not known and compression is set to "on" or more aggressive, the output will also be compressed. If not specified, this attribute is set to "off".
connectionLinger	The number of milliseconds during which the sockets used by this Connector will linger when they are closed. The default value is -1 (socket linger is disabled).
connectionTimeout	The number of milliseconds this Connector will wait, after accepting a connection, for the request URI line to be presented. The default value is 60000 (i.e. 60 seconds).
disableUploadTimeout	This flag allows the servlet container to use a different, longer connection timeout while a servlet is being executed, which in the end allows either the servlet a longer amount of time to complete its execution, or a longer timeout during data upload. If not specified, this attribute is set to "false".
emptySessionPath	If set to true, all paths for session cookies will be set to /. This can be useful for portlet specification implementations, but will greatly affect performance if many applications are accessed on a given server by the client. If not specified, this attribute is set to false.
enableLookups	Set to true if you want calls to request.getRemoteHost() to perform DNS lookups in order to return the actual host name of the remote client. Set to false to skip the DNS lookup and return the IP address in String form instead (thereby improving performance). By default, DNS lookups are enabled.
firstReadTimeout	The first read of a request will be made using the specified timeout. If no data is available after the specified time, the socket will be placed in the poller. Setting this value to 0 will increase scalability, but

Attribute	Description
	will have a minor impact on latency (see the related pollTime attribute). The default value is 100 (100ms). Note: on Windows, the actual value of firstRead-Timeout will be 500 + the specified value.
maxHttpHeaderSize	The maximum size of the request and response HTTP header, specified in bytes. If not specified, this attribute is set to 4096 (4 KB).
maxKeepAliveRequests	The maximum number of HTTP requests which can be pipelined until the connection is closed by the server. Setting this attribute to 1 will disable HTTP/1.0 keep-alive, as well as HTTP/1.1 keep-alive and pipelining. Setting this to -1 will allow an unlimited amount of pipelined or keep-alive HTTP requests. If not specified, this attribute is set to 100.
maxPostSize	The maximum size in bytes of the POST which will be handled by the container FORM URL parameter parsing. The limit can be disabled by setting this attribute to a value less than or equal to 0. If not specified, this attribute is set to 2097152 (2 megabytes).
maxSavePostSize	The maximum size in bytes of the POST which will be saved/buffered by the container during FORM or CLIENT-CERT authentication. For both types of authentication, the POST will be saved/buffered before the user is authenticated. For CLIENT-CERT authentication, the POST is buffered for the duration of the SSL handshake and the buffer emptied when the request is processed. For FORM authentication the POST is saved whilst the user is re-directed to the login form and is retained until the user successfully authenticates or the session associated with the authentication request expires. The limit can be disabled by setting this attribute to -1. Setting the attribute to zero will disable the saving of POST data during authentication. If not specified, this attribute is set to 4096 (4 kilobytes).
maxThreads	The maximum number of request processing threads to be created by this Connector, which therefore determines the maximum number of simultaneous requests that can be handled. If not specified, this attribute is set to 200.
noCompressionUserAgents	The value is a comma separated list of regular expressions matching user-agents of HTTP clients for which compression should not be used, because these clients, although they do advertise support for the fea-

Attribute	Description
	ture, have a broken implementation. The default value is an empty String (regex matching disabled).
pollTime	Duration of a poll call. Lowering this value will slightly decrease latency of connections being kept alive in some cases, but will use more CPU as more poll calls are being made. The default value is 5000 (5ms).
pollerSize	Amount of sockets that the poller responsible for polling kept alive connections can hold at a given time. Extra connections will be closed right away. The default value is 768, corresponding to 768 keepalive connections.
port	The TCP port number on which this Connector will create a server socket and await incoming connections. Your operating system will allow only one server application to listen to a particular port number on a particular IP address.
protocol	This attribute value must be HTTP/1.1 to use the HTTP handler, which is the default.
proxyName	If this Connector is being used in a proxy configuration, configure this attribute to specify the server name to be returned for calls to <code>request.getServerName()</code> .
proxyPort	If this Connector is being used in a proxy configuration, configure this attribute to specify the server port to be returned for calls to <code>request.getServerPort()</code> .
restrictedUserAgents	The value is a comma separated list of regular expressions matching user-agents of HTTP clients for which HTTP/1.1 or HTTP/1.0 keep alive should not be used, even if the clients advertise support for these features. The default value is an empty String (regex matching disabled).
scheme	Set this attribute to the name of the protocol you wish to have returned by calls to <code>request.getScheme()</code> . For example, you would set this attribute to "https" for an SSL Connector. The default value is "http".
secure	Set this attribute to true if you wish to have calls to <code>request.isSecure()</code> to return true for requests received by this Connector (you would want this on an SSL Connector). The default value is false.
sendfileSize	Amount of sockets that the poller responsible for sending static files asynchronously can hold at a giv-

Attribute	Description
	<p>en time. Extra connections will be closed right away without any data being sent (resulting in a zero length file on the client side). Note that in most cases, sendfile is a call that will return right away (being taken care of "synchronously" by the kernel), and the sendfile poller will not be used, so the amount of static files which can be sent concurrently is much larger than the specified amount. The default value is 256.</p>
server	<p>The Server header for the http response. Unless your paranoid, you won't need this feature.</p>
tcpNoDelay	<p>If set to true, the TCP_NO_DELAY option will be set on the server socket, which improves performance under most circumstances. This is set to true by default.</p>
threadPriority	<p>The priority of the request processing threads within the JVM. The default value is <code>java.lang.Thread#NORM_PRIORITY</code>. See the JavaDoc for the <code>java.lang.Thread</code> class for more details on what this priority means.</p>
URIEncoding	<p>This specifies the character encoding used to decode the URI bytes, after %xx decoding the URL. If not specified, ISO-8859-1 will be used.</p>
useBodyEncodingForURI	<p>This specifies if the encoding specified in contentType should be used for URI query parameters, instead of using the URIEncoding. This setting is present for compatibility with Tomcat 4.1.x, where the encoding specified in the contentType, or explicitly set using <code>Request.setCharacterEncoding</code> method was also used for the parameters from the URL. The default value is false.</p>
useIPVHosts	<p>Set this attribute to true to cause JBoss Web to use the IP address that the request was recieved on to determine the Host to send the request to. The default value is false.</p>
useSendfile	<p>Use kernel level sendfile for certain static files. The default value is true.</p>
xpoweredBy	<p>Set this attribute to true to cause JBoss Web to advertise support for the Servlet specification using the header recommended in the specification. The default value is false.</p>

5.2. The HTTPS Connector

When APR is enabled, the HTTPS connector will use a socket poller for keepalive, increasing scalability of the server. It also uses OpenSSL, which may be more optimized than JSSE depending on the processor being used, and can be complemented with many commercial accelerator components. Unlike the HTTP connector, the HTTPS connector cannot use sendfile to optimize static file processing.

The HTTPS APR connector has the same basic attributes than the HTTP APR connector, but adds OpenSSL specific ones. For instance, here is an example of the HTTPS connector:

```
<Connector port="443" maxHttpHeaderSize="8192"
  maxThreads="50" enableLookups="false"
  disableUploadTimeout="true"
  acceptCount="100" scheme="https" secure="true"
  SSLEngine="on"
  SSLCertificateFile="\${catalina.base}/conf/localhost.crt"
  SSLCertificateKeyFile="\${catalina.base}/conf/localhost.key" />
```

For the full details on using OpenSSL, please refer to OpenSSL documentations and the many books available for it (see the Official OpenSSL website). The SSL specific attributes for the connector are listed in the following table.

Table 5.2. Attributes in the HTTPS Connector

Attribute	Description
SSLEngine	Name of the SSLEngine to use. off: Do not use SSL, on: Use SSL but no specific ENGINE. The default value is off.
SSLProtocol	Protocol which may be used for communicating with clients. The default is "all", with other acceptable values being "SSLv2", "SSLv3", "TLSv1", and "SSLv2+SSLv3".
SSLCipherSuite	Ciphers which may be used for communicating with clients. The default is "ALL", with other acceptable values being a list of ciphers, with ":" used as the delimiter (see OpenSSL documentation for the list of ciphers supported).
SSLCertificateFile	Name of the file that contains the server certificate. The format is PEM-encoded.
SSLCertificateKeyFile	Name of the file that contains the server private key. The format is PEM-encoded. The default value is the value of "SSLCertificateFile" and in this case both certificate and private key have to be in this file (NOT RECOMMENDED).
SSLPassword	Pass phrase for the encrypted private key. If "SSLPass-

Attribute	Description
	word" is not provided, the callback function should prompt for the pass phrase.
SSLVerifyClient	Ask client for certificate. The default is "none", meaning the client will not have the opportunity to submit a certificate. Other acceptable values include "optional", "require" and "optionalNoCA".
SSLVerifyDepth	Maximum verification depth for client certificates. The default is "10".
SSLCACertificateFile	See the mod_ssl documentation [http://httpd.apache.org/docs/2.2/mod/mod_ssl.html]
SSLCACertificatePath	See the mod_ssl documentation [http://httpd.apache.org/docs/2.2/mod/mod_ssl.html]
SSLCertificateChainFile	See the mod_ssl documentation [http://httpd.apache.org/docs/2.2/mod/mod_ssl.html]
SSLCARevocationFile	See the mod_ssl documentation [http://httpd.apache.org/docs/2.2/mod/mod_ssl.html]
SSLCARevocationPath	See the mod_ssl documentation [http://httpd.apache.org/docs/2.2/mod/mod_ssl.html]

5.3. The AJP Connector

The AJP connector is used for machine-to-machine communication. If another server (e.g., an Apache web server or a load balancer) needs to access services on JBoss Web Server, it can use the AJP connector. It is more efficient than the HTTP connector, which targets web browsers.

The AJP connector uses a socket poller for keepalive, increasing scalability of the server. As AJP is designed around a pool of persistent (or almost persistent) connections, this will reduce significantly the amount of processing threads needed by JBoss Web. Unlike the HTTP connector, the AJP connector cannot use sendfile to optimize static file processing. Below is an example configuration for an AJP connector.

```
<Connector port="8009"
  emptySessionPath="true" enableLookups="false"
  redirectPort="8443" protocol="AJP/1.3"/>
```

The following attributes are supported in the AJP connector. Many of the attributes are very similar to the HTTP connector we have seen above.

Table 5.3. Attributes in the AJP Connector

Attribute	Description
acceptCount	The maximum queue length for incoming connection requests when all possible request processing threads are in use. Any requests received when the queue is full will be refused. The default value is 10.
address	For servers with more than one IP address, this attribute specifies which address will be used for listening on the specified port. By default, this port will be used on all IP addresses associated with the server. A value of 127.0.0.1 indicates that the Connector will only listen on the loopback interface.
allowTrace	A boolean value which can be used to enable or disable the TRACE HTTP method. If not specified, this attribute is set to false.
connectionTimeout	The number of milliseconds this Connector will wait, after accepting a connection, for the request URI line to be presented. The default value is infinite (i.e. no timeout).
emptySessionPath	If set to true, all paths for session cookies will be set to /. This can be useful for portlet specification implementations, but will greatly affect performance if many applications are accessed on a given server by the client. If not specified, this attribute is set to false.
enableLookups	Set to true if you want calls to request.getRemoteHost() to perform DNS lookups in order to return the actual host name of the remote client. Set to false to skip the DNS lookup and return the IP address in String form instead (thereby improving performance). By default, DNS lookups are enabled.
firstReadTimeout	The first read of a request will be made using the specified timeout. If no data is available after the specified time, the socket will be placed in the poller. Setting this value to 0 will increase scalability, but will have a minor impact on latency (see the related pollTime attribute). The default value is 100 (100ms). Note: on Windows, the actual value of firstReadTimeout will be 500 + the specified value.
maxPostSize	The maximum size in bytes of the POST which will be handled by the container FORM URL parameter parsing. The feature can be disabled by setting this attribute to a value less than or equal to 0. If not specified, this attribute is set to 2097152 (2 megabytes).
maxSavePostSize	The maximum size in bytes of the POST which will be saved/buffered by the container during FORM or

Attribute	Description
	CLIENT-CERT authentication. For both types of authentication, the POST will be saved/buffered before the user is authenticated. For CLIENT-CERT authentication, the POST is buffered for the duration of the SSL handshake and the buffer emptied when the request is processed. For FORM authentication the POST is saved whilst the user is re-directed to the login form and is retained until the user successfully authenticates or the session associated with the authentication request expires. The limit can be disabled by setting this attribute to -1. Setting the attribute to zero will disable the saving of POST data during authentication. If not specified, this attribute is set to 4096 (4 kilobytes).
maxThreads	The maximum number of request processing threads to be created by this Connector, which therefore determines the maximum number of simultaneous requests that can be handled. If not specified, this attribute is set to 200.
pollTime	Duration of a poll call. Lowering this value will slightly decrease latency of connections being kept alive in some cases, but will use more CPU as more poll calls are being made. The default value is 5000 (5ms).
pollerSize	Amount of sockets that the poller responsible for polling kept alive connections can hold at a given time. Extra connections will be closed right away. The default value is 768, corresponding to 768 keepalive connections.
port	The TCP port number on which this Connector will create a server socket and await incoming connections. Your operating system will allow only one server application to listen to a particular port number on a particular IP address.
protocol	This attribute value must be AJP/1.3 to use the AJP handler.
proxyName	If this Connector is being used in a proxy configuration, configure this attribute to specify the server name to be returned for calls to request.getServerName().
proxyPort	If this Connector is being used in a proxy configuration, configure this attribute to specify the server port to be returned for calls to request.getServerPort().

Attribute	Description
redirectPort	If this Connector is supporting non-SSL requests, and a request is received for which a matching <security-constraint> requires SSL transport, Catalina will automatically redirect the request to the port number specified here.
request.registerRequests	This attribute controls request registration for JMX monitoring of the Connector. It is enabled by default, but may be turned it off to save a bit of memory.
scheme	Set this attribute to the name of the protocol you wish to have returned by calls to request.getScheme(). For example, you would set this attribute to "https" for an SSL Connector. The default value is "http".
secure	Set this attribute to true if you wish to have calls to request.isSecure() to return true for requests received by this Connector (you would want this on an SSL Connector). The default value is false.
tcpNoDelay	If set to true, the TCP_NO_DELAY option will be set on the server socket, which improves performance under most circumstances. This is set to true by default.
tomcatAuthentication	If set to true, the authentication will be done in JBoss Web. Otherwise, the authenticated principal will be propagated from the native webserver and used for authorization in JBoss Web. The default value is true.
URIEncoding	This specifies the character encoding used to decode the URI bytes, after %xx decoding the URL. If not specified, ISO-8859-1 will be used.
useBodyEncodingForURI	This specifies if the encoding specified in contentType should be used for URI query parameters, instead of using the URIEncoding. This setting is present for compatibility with Tomcat 4.1.x, where the encoding specified in the contentType, or explicitly set using Request.setCharacterEncoding method was also used for the parameters from the URL. The default value is false.
useIPVHosts	Set this attribute to true to cause JBoss Web to use the ServerName passed by the native web server to determine the Host to send the request to. The default value is false.
xpoweredBy	Set this attribute to true to cause JBoss Web to advertise support for the Servlet specification using the header recommended in the specification. The default

Attribute	Description
	value is false.

6

URL Rewriting

One of the most popular module in the Apache Web Server is the `mod_rewrite` module, which allows the web master to map any internal URL to arbitrary public URLs. The JBoss Web Server provides a servlet valve that implement the same functionalities as `mod_rewrite`.

In order to use the URL rewrite valve, you have to add the following XML element to the JBoss Web's `server.xml`, or the web application's `context.xml` configuration file.

```
<Valve className="org.jboss.web.rewrite.RewriteValve" />
```

The valve will then use a `rewrite.properties` file for the rewrite conditions and rules. The `rewrite.properties` file location depends on where you put the Valve element in the configuration file:

- If the Valve element is inside an Engine element in `server.xml`, the `rewrite.properties` file should be placed in a folder with the same name as the Engine. This folder should be placed either in the classloader, or in the `$JBoss/server/default/conf` directory (replace `default` with the configuration you use, if needed).
- If the Valve element is inside a Host element in `server.xml`, the `rewrite.properties` file should be placed in a folder named `engine_name/host_name`, which is placed either in the classloader, or in the `conf` folder of the current JBoss configuration.
- If the Valve element is inside the `context.xml` file for a web application, the `rewrite.properties` file should be placed in the `WEB-INF` folder of the web application.

The `rewrite.properties` file contains a list of directives which closely resemble the directives used by `mod_rewrite`, in particular the central `RewriteRule` and `RewriteCond` directives. For instance, the following rules in the `rewrite.properties` file make the server serve different files based on the client browser type in the HTTP request's "User-Agent:" header.

```
RewriteCond %{HTTP_USER_AGENT} ^Mozilla.*
RewriteRule ^/$ /homepage.max.html [L]

RewriteCond %{HTTP_USER_AGENT} ^Lynx.*
RewriteRule ^/$ /homepage.min.html [L]

RewriteRule ^/$ /homepage.std.html [L]
```

If you use a browser which identifies itself as 'Mozilla' (including Netscape Navigator, Mozilla etc), then you get the max homepage (which could include frames, or other special features). If you use the Lynx browser (which is terminal-based), then you get the min homepage (which could be a version designed for easy, text-only browsing).

If neither of these conditions apply (you use any other browser, or your browser identifies itself as something non-standard), you get the std (standard) homepage.

In the next two sections, we will discuss the syntax of the RewriteRule and RewriteCond directives.

6.1. RewriteCond

The RewriteCond directive defines a rule condition. One or more RewriteCond can precede a RewriteRule directive. The following rule is then only used if both the current state of the URI matches its pattern, and if these conditions are met. Its syntax is as follows.

```
RewriteCond TestString CondPattern
```

The TestString typically contains variables in the form of %{NAME_OF_VARIABLE}, where the NAME_OF_VARIABLE refers to a server property or a property related to the current HTTP request. The following

Table 6.1. Server variables for the RewriteCond text string

Category	Variable Name
HTTP headers	HTTP_USER_AGENT, HTTP_REFERER, HTTP_COOKIE, HTTP_FORWARDED, HTTP_HOST, HTTP_PROXY_CONNECTION, HTTP_ACCEPT
Connection and request	REMOTE_ADDR, REMOTE_HOST, REMOTE_PORT, REMOTE_USER, REMOTE_IDENT, REQUEST_METHOD, SCRIPT_FILENAME, REQUEST_PATH, CONTEXT_PATH, SERVLET_PATH, PATH_INFO, QUERY_STRING, AUTH_TYPE
Server internals	DOCUMENT_ROOT, SERVER_NAME, SERVER_ADDR, SERVER_PORT, SERVER_PROTOCOL, SERVER_SOFTWARE
Date and time	TIME_YEAR, TIME_MON, TIME_DAY, TIME_HOUR, TIME_MIN, TIME_SEC, TIME_WDAY, TIME
Specials	THE_REQUEST, REQUEST_URI, REQUEST_FILENAME, HTTPS
Dynamic variables	ENV:variable, SSL:variable, HTTP:header

These variables all correspond to the similarly named HTTP MIME-headers and Servlet API methods. Most are documented elsewhere in the Manual or in the CGI specification. Those that are special to the rewrite valve include

those below.

- `REQUEST_PATH`: Corresponds to the full path that is used for mapping.
- `CONTEXT_PATH`: Corresponds to the path of the mapped context.
- `SERVLET_PATH`: Corresponds to the servlet path.
- `THE_REQUEST`: The full HTTP request line sent by the browser to the server (e.g., "GET /index.html HTTP/1.1"). This does not include any additional headers sent by the browser.
- `REQUEST_URI`: The resource requested in the HTTP request line. (In the example above, this would be "/index.html".)
- `REQUEST_FILENAME`: The full local filesystem path to the file or script matching the request.
- `HTTPS`: Will contain the text "on" if the connection is using SSL/TLS, or "off" otherwise.
- The variables `SCRIPT_FILENAME` and `REQUEST_FILENAME` contain the same value - the value of the filename field of the internal request_rec structure of the Apache server. The first name is the commonly known CGI variable name while the second is the appropriate counterpart of `REQUEST_URI` (which contains the value of the uri field of request_rec).
- `ENV:variable`: the variable can be the name of any Java system property, is also available.
- `SSL:variable`: the variable is the name of an SSL environment variable. This feature has not been implemented yet.
- `HTTP:header`: the header can be any HTTP MIME-header name. It can always be used to obtain the value of a header sent in the HTTP request. For instance, the `{HTTP:Proxy-Connection}` symbol represents the value of the HTTP header "Proxy-Connection:".

You can also insert back reference to the Nth matched group (in parentheses) in the current RewriteRule and last matched RewriteCond regular expressions using the `$N` and `%N` symbols in the text string.

In addition, you can put arbitrary dynamic content into the text string via the RewriteMap directive. A RewriteMap directive looks like the following:

```
RewriteMap name rewriteMapClassName optionalParameters
```

The maps are implemented using an interface that users must implement. Its class name is `org.jboss.web.rewrite.RewriteMap`, and its code is:

```
package org.jboss.web.rewrite;

public interface RewriteMap {
    public String setParameters(String params);
    public String lookup(String key);
}
```

Then, in the RewriteCond text string, we can use the form `${mapname:key|default}`.

CondPattern is the condition pattern, a regular expression which is applied to the current instance of the TestString. TestString is first evaluated, before being matched against CondPattern. CondPattern is a perl compatible regular expression with the following additions.

- You can prefix the pattern string with a '!' character (exclamation mark) to specify a non-matching pattern.
- There are some special variants of CondPatterns. Instead of real regular expression strings you can also use one of the following:
 - '<CondPattern' (lexicographically precedes) treats the CondPattern as a plain string and compares it lexicographically to TestString. True if TestString lexicographically precedes CondPattern.
 - '>CondPattern' (lexicographically follows) treats the CondPattern as a plain string and compares it lexicographically to TestString. True if TestString lexicographically follows CondPattern.
 - '=CondPattern' (lexicographically equal) treats the CondPattern as a plain string and compares it lexicographically to TestString. True if TestString is lexicographically equal to CondPattern (the two strings are exactly equal, character for character). If CondPattern is "" (two quotation marks) this compares TestString to the empty string.
 - '-d' (is directory) treats the TestString as a pathname and tests whether or not it exists, and is a directory.
 - '-f' (is regular file) treats the TestString as a pathname and tests whether or not it exists, and is a regular file.
 - '-s' (is regular file, with size) treats the TestString as a pathname and tests whether or not it exists, and is a regular file with size greater than zero.
- You can also set special flags for CondPattern by appending [flags] as the third argument to the RewriteCond directive, where flags is a comma-separated list of any of the following flags:
 - 'nocase|NC' (no case) makes the test case-insensitive - differences between 'A-Z' and 'a-z' are ignored, both in the expanded TestString and the CondPattern. This flag is effective only for comparisons between TestString and CondPattern. It has no effect on filesystem and subrequest checks.
 - 'ornext|OR' (or next condition) combines rule conditions with a local OR instead of the implicit AND.

An example of the OR flag is as follows. Without this flag you would have to write the condition/rule pair three times.

```
RewriteCond %{REMOTE_HOST} ^host1.* [OR]
RewriteCond %{REMOTE_HOST} ^host2.* [OR]
RewriteCond %{REMOTE_HOST} ^host3.*
RewriteRule ...some special stuff for any of these hosts...
```

6.2. RewriteRule

The RewriteRule directive is the real rewriting workhorse. The directive can occur more than once, with each instance defining a single rewrite rule. The order in which these rules are defined is important - this is the order in which they will be applied at run-time. The RewriteRule directive has the following syntax.

```
RewriteRule Pattern Substitution
```

Pattern is a perl compatible regular expression, which is applied to the current URL, which is the value of the URL when this rule is applied. This may not be the originally requested URL, which may already have matched a previous rule, and have been altered.

Like the regular expression in RewriteCond, the Pattern string here can also take the '!' prefix to negate a pattern. When using the '!' character to negate a pattern, you cannot include grouped wildcard parts in that pattern. This is because, when the pattern does NOT match (ie, the negation matches), there are no contents for the groups. Thus, if negated patterns are used, you cannot use \$N in the substitution string!

The Substitution of a rewrite rule is the string which is substituted for (or replaces) the original URL which Pattern matched. In addition to plain text, it can include any server variable, back reference to matched parts, and RewriteMap extensions discussed in the previous section.

As already mentioned, all rewrite rules are applied to the Substitution (in the order in which they are defined in the config file). The URL is completely replaced by the Substitution and the rewriting process continues until all rules have been applied, or it is explicitly terminated by a flag.

There is a special substitution string named '-' which means: NO substitution! This is useful in providing rewriting rules which only match URLs but do not substitute anything for them. It is commonly used in conjunction with the C (chain) flag, in order to apply more than one pattern before substitution occurs.

Additionally you can set special flags for Substitution by appending [flags] as the third argument to the RewriteRule directive. Flags is a comma-separated list of any of the following flags:

- 'chain|C' (chained with next rule): This flag chains the current rule with the next rule (which itself can be chained with the following rule, and so on). This has the following effect: if a rule matches, then processing continues as usual - the flag has no effect. If the rule does not match, then all following chained rules are skipped. For instance, it can be used to remove the ".www" part, inside a per-directory rule set, when you let an external redirect happen (where the ".www" part should not occur!).
- 'cookie|CO=NAME:VAL:domain[:lifetime[:path]]' (set cookie): This sets a cookie in the client's browser. The cookie's name is specified by NAME and the value is VAL. The domain field is the domain of the cookie, such as '.apache.org', the optional lifetime is the lifetime of the cookie in minutes, and the optional path is the path of the cookie
- 'env|E=VAR:VAL' (set environment variable): This forces an environment variable named VAR to be set to the value VAL, where VAL can contain regex backreferences (\$N and %N) which will be expanded. You can use this flag more than once, to set more than one variable. The variables can later be dereferenced in many situations, most commonly from within XSSI (via <!--#echo var="VAR"-->) or CGI (\$ENV{'VAR'}). You can also dereference the variable in a later RewriteCond pattern, using %{ENV:VAR}. Use this to strip information from URLs, while maintaining a record of that information.

- 'forbidden|F' (force URL to be forbidden): This forces the current URL to be forbidden - it immediately sends back a HTTP response of 403 (FORBIDDEN). Use this flag in conjunction with appropriate RewriteConds to conditionally block some URLs.
- 'gone|G' (force URL to be gone): This forces the current URL to be gone - it immediately sends back a HTTP response of 410 (GONE). Use this flag to mark pages which no longer exist as gone.
- 'host|H=Host' (apply rewriting to host): Rather than rewrite the URL, the virtual host will be rewritten.
- 'last|L' (last rule): Stop the rewriting process here and don't apply any more rewrite rules. This corresponds to the Perl last command or the break command in C. Use this flag to prevent the currently rewritten URL from being rewritten further by following rules. For example, use it to rewrite the root-path URL ('/') to a real one, e.g., '/e/www/'.
- 'next|N' (next round): Re-run the rewriting process (starting again with the first rewriting rule). This time, the URL to match is no longer the original URL, but rather the URL returned by the last rewriting rule. This corresponds to the Perl next command or the continue command in C. Use this flag to restart the rewriting process - to immediately go to the top of the loop. Be careful not to create an infinite loop!
- 'nocase|NC' (no case): This makes the Pattern case-insensitive, ignoring difference between 'A-Z' and 'a-z' when Pattern is matched against the current URL.
- 'noescape|NE' (no URI escaping of output): This flag prevents the rewrite valve from applying the usual URI escaping rules to the result of a rewrite. Ordinarily, special characters (such as '%', '\$', ';', and so on) will be escaped into their hexcode equivalents ('%25', '%24', and '%3B', respectively); this flag prevents this from happening. This allows percent symbols to appear in the output, as in RewriteRule /foo/(.*) /bar?arg=P1%3d\$1 [R,NE] which would turn '/foo/zed' into a safe request for '/bar?arg=P1=zed'.
- 'qsappend|QSA' (query string append: This flag forces the rewrite engine to append a query string part of the substitution string to the existing string, instead of replacing it. Use this when you want to add more data to the query string via a rewrite rule.
- 'redirect|R [=code]' (force redirect): Prefix Substitution with http://thishost[:thisport]/ (which makes the new URL a URI) to force an external redirection. If no code is given, a HTTP response of 302 (MOVED TEMPORARILY) will be returned. If you want to use other response codes in the range 300-400, simply specify the appropriate number or use one of the following symbolic names: temp (default), permanent, seeother. Use this for rules to canonicalize the URL and return it to the client - to translate ``/~" into ``/u'', or to always append a slash to /u/user, etc. Note: When you use this flag, make sure that the substitution field is a valid URL! Otherwise, you will be redirecting to an invalid location. Remember that this flag on its own will only prepend http://thishost[:thisport]/ to the URL, and rewriting will continue. Usually, you will want to stop rewriting at this point, and redirect immediately. To stop rewriting, you should add the 'L' flag.
- 'skip|S=num' (skip next rule(s)): This flag forces the rewriting engine to skip the next num rules in sequence, if the current rule matches. Use this to make pseudo if-then-else constructs: The last rule of the then-clause becomes skip=N, where N is the number of rules in the else-clause. (This is not the same as the 'chain|C' flag!)
- 'type|T=MIME-type' (force MIME type): Force the MIME-type of the target file to be MIME-type. This can be used to set up the content-type based on some conditions.

7

PHP

The PHP Module allows JBoss Web Server to run PHP scripts side-by-side with Java applications. It is a servlet that calls a native embedded PHP engine with libraries extentions.

7.1. Installation

To install the PHP module, you have to download the binary library files for your specific platform from the JBoss Web Server download page. If you do not see a pre-build binary for your platform, you can build it from the source code package by running the buildphp.sh script.

Extract all the files from the downloaded tarball (or the build result), and put the PHP/lib/* files in your JBoss Web Server installation directory. The add the following Listener element under the Server element of your conf/server.xml file.

```
<Listener className="org.apache.catalina.servlets.php.LifecycleListener" />
```

Then, in the conf/web.xml file, add the PHP servlet configuration.

```
<servlet>
  <servlet-name>php</servlet-name>
  <servlet-class>org.apache.catalina.servlets.php.Handler</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>0</param-value>
  </init-param>
  <load-on-startup>6</load-on-startup>
</servlet>
<servlet>
  <servlet-name>phps</servlet-name>
  <servlet-class>org.apache.catalina.servlets.php.Highlight</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>php</servlet-name>
  <url-pattern>*.php</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>phps</servlet-name>
  <url-pattern>*.phps</url-pattern>
</servlet-mapping>
```

Finally, edit the bin/setenv.sh file to add the LD_LIBRARY_PATH variable and modify/add the java.library.path parameter of the JVM:

```
CATALINA_OPTS="$CATALINA_OPTS -Djava.library.path=$CATALINA_HOME/PHP/lib"  
LD_LIBRARY_PATH=$CATALINA_HOME/PHP/lib  
export LD_LIBRARY_PATH
```

7.2. Use PHP scripts

You can now place any PHP scripts in your .war applications side-by-side with your JSP pages. You can access PHP scripts via URLs mapped from your .war application just like you access JSP pages.

The webapps/php-examples.war war file of the PHP module tarball contains some php demo scripts. They are deployed under /php-examples to use them you only need to modify the bin/setenv.sh as described above. You don't have to modify the conf/web.xml nor the conf/server.xml files since the .war application itself configures the PHP servlet and listener. You have to start and restart the servlet container because of the environment modifications. To use the examples, try <http://localhost:8080/php-examples/index.php>.

7.3. PHP extension libraries

For the moment this feature is only supported on Solaris. Edit the php.ini file and add your library extensions as in the following example:

```
extension_dir=/home/jfclere/SunOS_i386_tools/PHP/lib/php/extensions/  
extension=openssl.so  
extension=pdo_pgsql.so  
extension=pgsql.so
```