Sun Web Server

The Essential Guide

William Nelson ▪ Arvind Srinivasan ▪ Murthy Chintalapati (CVR)
Foreword by Scott G. McNealy

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/ph

Library of Congress Cataloging-in-Publication Data:

Nelson, William C. (William Clayton)

Sun Web server : the essential guide / William C. Nelson, Arvind Srinivasan, Murthy Chintalapati.

p. cm.

ISBN 978-0-13-712892-1 (pbk. : alk. paper) 1. Web servers. 2. Web sites[md]Design. I. Srinivasan, Arvind, 1968- II. Chintalapati, Murthy. III. Sun Microsystems. IV. Title.

TK5105.888.N453 2009

006.7—dc22

2009025812

Copyright © 2010 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95054 U.S.A.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to

Pearson Education, Inc
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671-3447

ISBN-13: 978-0-13-712892-1
ISBN-10: 0-13-712892-4

Text printed in the United States on recycled paper at RR Donnelley, Crawfordsville, Indiana
First printing August 2009

Sun Microsystems, Inc. has intellectual property rights relating to implementations of the technology described in this publication. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents, foreign patents, or pending applications.

Sun, Sun Microsystems, the Sun logo, J2ME, J2EE, Java Card, and all Sun- and Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. THIS PUBLICATION IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

Foreword

NETWORK computing is the heart and soul of Sun Microsystems. Always has been; always will be. If the World Wide Web (WWW) were to be one big telephone switch, Web Server would be the technology that reliably delivers the dial tone or the *web tone,* as we call it here at Sun. From massively scalable chip-based multi-core multi-threaded servers to the Solaris operating system to Java and the Web Server infrastructure software, Sun has the technology.

As our systems are built to last, the software components of a Web Server are designed to scale from a single core to hundreds of cores, from a few hundred to hundreds of thousands of user connections. Sun Web Server is no exception and is one of the key technologies that drive innovation across our systems and software stack. With its rock solid quality, superior scalability, and its unique architecture combining Java and native web dynamic web serving technologies, Sun Web Server is widely deployed by enterprises like Sun, Major League Baseball (MLB.com), and *The New York Times*, to name a few.

Even after a decade and a half since use of the web became widespread in the '90s, the world of Web Server software continues to evolve with significant innovations to meet the demands of the Internet. Web Server 7.0 release delivers outstanding innovations in many areas, including the following:

- Dramatic scalability improvements demanded by modern multi-core systems and the Web 2.0 era of richer web applications
- Manageability enhancements, with a brand new server management interface that drastically simplifies the system administrator's job of deploying and managing clusters of web servers
- Security feature updates to protect online properties from growing threats of web vulnerabilities
- Ease of developing and deploying heterogeneous dynamic web scripting technologies.

So I am very pleased to present the most authoritative technical guide on the Sun Web Server 7.0 release. I am proud of the authors who have brought years of experience developing, teaching, training, and supporting customers on Sun Web Server. With concrete examples and lucid explanations of complex concepts, the authors of this book have delivered a marvelous handy reference guide. Whether you happen to be a developer developing your web site or a server administrator responsible for managing and monitoring a large-scale web server farm, I hope that you will find the information in this book useful.

Thank you for choosing Sun and Sun Web Server as part of your web infrastructure solution.

Scott G. McNealy

Chairman, Sun Microsystems Inc.

Preface

"In summary, Sun Web Server is generally two times as fast as Apache. The Sun Web Server is fast, flexible, scalable, and a downright joy to run…if performance and ease of management are paramount, Web Server is the way to go."

—ServerWatch Review, April 2007.

THE term *web server* broadly refers to the server computer and the software that hosts a web site and serves the site's content to the requesting user agents. At the very basic level, the web server software listens to the TCP/IP port for Hypertext Transport Protocol (HTTP) requests and serves the requested resource. Web servers are a critical part of the infrastructure that powers the public Internet, as well as the private intranets of institutions and corporations. Many web server software products are widely used in the market, namely Apache HTTP server, Microsoft Internet Information Server, Sun Java System Web Server, and newer servers such as lighttpd.

Sun Web Server (officially formerly known as Netscape Enterprise/SunONE/iPlanet Web Server) is the most secure web serving platform widely deployed by large-scale enterprises in the financial, telecommunications, government, travel, and sports worlds. The Sun Java System Web Server (also referred to as Sun Web Server) is a freely available, high-performance, feature-rich HTTP server that provides a simple, user-friendly administration interface. In addition to being a high-performance static-media delivery engine for items such as images and HTML documents, this versatile server supports a breadth of heterogeneous dynamic content technologies, including Java and popular non-Java native scripting technologies such as NSAPI (for Netscape Server Application Programming Interface), Java Servlets, JavaServer Pages (JSP), PHP, Perl, Python, Ruby On Rails, Active Server Pages (ASP), and Coldfusion. With support for hundreds of thousands of simultaneous connections, Sun Web Server demonstrates superior scalability and is the perfect choice for enterprises deploying Web 2.0 technologies, powered by multi-core systems and chip-based multi-threaded (CMT) architecture.

Sun Web Server's Scalability on CMT/Multi-core Systems Is the Industry Standard

A single 64-bit instance of Sun Web Server 7.0, on a Sun Fire T5220 CMT server running Solaris 10 Update 5 and JDK 1.6.0, set a world record SPECweb benchmark in April 2008. A few sample performance metrics showcase Web Server's superior scaling: over 250,000 simultaneous HTTP connections, 131,000 secure banking operations per second (which generated 1GB web server access log data per minute) and 1.4 terabytes of data over the HTTP interface.

With an in-depth discussion of its architecture, new features, administration, performance, and so forth, Sun's latest Web Server 7.0 release is the subject of this book. Gathering from their insights into the product's innovations and experience training the customers who use the product today in live productions, the authors have zeroed in on the most definitive technical information useful to administrators and architects, developers, and deployers alike.

Who This Book Is For

This book gives server administrators a handy guide for the day-to-day administrative tasks such as installation, configuration, cluster management, monitoring, and troubleshooting. You also get insights to developers and architects interested in understanding Sun Web Server internals and extending the server functionality with a variety of dynamic content technologies. Together with information to the deployers on sizing/tuning and reference configurations used in Web 2.0 style sites, this book is an essential guide for the users of the world's most scalable and versatile web infrastructure software.

This book is designed to help you—whether you are developing dynamic scripting technologies, deploying servers, or an administrator managing the server 24×7—get started quickly.

How This Book Is Organized

This book presents in-depth information that will help you build your product expertise gradually. First, Chapter 1 provides a bird's eye view of the product's new features. Then Chapter 2, "Web Server 7.0 Architecture," lays the foundation for your understanding of Sun Web Server's internals. Chapter 3 then explores various methods for performing initial installations or migrating from previous releases.

The following two chapters, Chapter 4, "Web Server 7.0 Administration," and Chapter 5, "Web Server 7.0 Configuration Files," provide the necessary foundation for the server administrator. Chapter 6 discusses how Web Server processes HTTP requests and is very handy when customizing Web Server, such as when you are adding a custom vanity URL rewrite rule. Chapter 7, "Monitoring Web Server 7.0," and Chapter 8, "Securing Web Server 7.0," both provide vital information to server administrators configuring and securing the web server and monitoring the server instances in live production for optimal performance. These chapters provide task-oriented hands-on examples of the administration interfaces and command-line examples—to the delight of server administrators, we hope. Chapters 9 and 10 turn to developers' needs, discussing how to build dynamic content through scripting languages and server-side Java based extensions.

Resolving the anomalies in the web server runtime behavior (such as those unexpected 500 Server Errors) rapidly could be very important for developers and deployers alike, so Chapter 11 is devoted to troubleshooting. The final chapter, Chapter 12, is devoted to building secure dynamic (Web 2.0/Enterprise 2.0 style) sites with Sun Web Server and explains why the product is ideal for building such sites with your dynamic content and database technologies of choice. This chapter also features a couple of Sun's own production deployments powered by Sun Web Server 7.0—namely, Sun Blogs and Sun Forums. Sun Blogs (http://blogs.sun.com) is an employee blogging site deployed with Sun Web Server 7.0, and the MySQL Sun Forums site (http://forums.sun.com) is a public forum for thousands of products and technologies deployed with Sun Web Server 7.0 and Oracle RDBMS.

Appendix A is a detailed reference guide to the main server configuration file, `server.xml`. Appendixes B and C provide sample reports of server runtime statistics, respectively in XML and plaintext format.

Getting the Most Out of This Book

There are many ways to get the most out of this book, whether you are starting to build a single server web site or managing a larger scale clustered web server deployment.

- Make sure you are using the latest version of Web Server because there may be security updates and bug fixes relevant to the job you like to accomplish. Links to official downloads and documentation can be found at this book's web site, http://www.sunwebserver.com/.

- Take advantage of the numerous command-line examples and the self-paced labs you will find throughout this book. Manageability is a major theme of the Sun Web Server 7.0 release, and its administration interfaces are designed to let you perform the common tasks with relative ease.
- This book is filled with numerous clear and concise how-to style examples, such as configuring a secure HTTP listener, a virtual server, a FastCGI interface, a reverse proxy front end, an LDAP authentication, or a data source such as MySQL database.

Conventions Used in This Book

It is our goal for you to get the most out of this book and get your job done efficiently. We have employed a few conventions to improve readability. Here are some conventions used in this book:

- The term "Sun Web Server" refers to all versions of the Web Server, whereas "Sun Web Server 7.0" refers to the specific version.
- The figures, tables, and examples are numbered consistently, using chapter number and a sequential number; for example, Figure 1.1, Table 6.1, and Example 6.2.
- A special font is used to distinguish configuration filenames (for example, `obj.conf`), variables, and snippets of command-line administration scripts from normal text.
- Special information is sometimes called out explicitly using notes like the following:

Note: Key themes behind the Sun Java System Web Server 7.0 release include manageability, security, interoperability, CMT scalability, ease of deployment via integrated support for dynamic scripting technologies, and proxy configurations.

- Most chapters end with a "Self-Paced Labs" section. These self-paced labs are designed to help validate your understanding of the concepts described in the respective chapters. The self-paced labs are modeled after Sun Web Server training and are designed by chief training instructor and one of the authors of this book, Bill Nelson. The answers to the self-paced exercises are available online at this book's ancillary web site, http://www.sunwebserver.com. At this site, you will also find FAQs, errata, discussions on the

book, and links to more information on the Sun Web Server product, including its download locations and product forums. Please visit this web site and share your insights and impressions of our work.

Note: We make every effort to ensure accuracy of the information—content or examples—presented in this book. However, we do anticipate that there might be inadvertent errors in the book. So if you do find errors (whether it is a defective example or a typo) that need correction, we would like to hear from you. Please visit the SunWebServer.com web site, click the Errata link, and follow the instructions to submit the errors online.

5

Web Server 7.0 Configuration Files

CHAPTER 4, "Web Server 7.0 Administration," introduced the concept of a Web Server *configuration*. That chapter described that a configuration consists of various components that are used to configure the runtime services of a Web Server 7.0 instance. Included in these components are various configuration files (such as `magnus.conf`, `server.xml`, and `obj.conf`) that are used to specify global variables and define how the server responds to specific events and client requests.

Each Web Server configuration has its own set of files that are located beneath the `config` directory for the server instance. Table 5.1 provides an overview of the configuration files associated with Web Server 7.0, as well as a brief description of their purpose.

Table 5.1 Web Server 7.0 Configuration Files

Filename	Overview
`cert8.db`	Network Security Services (NSS) certificate database. This database stores publicly accessible objects (such as certificates, certificate revocation lists, and S/MIME records) and works with the `key3.db` database for certificate management. This file is part of the instance's trust database.
`certmap.conf`	Certificate to LDAP DN mapping configuration.
`default-web.xml`	Default values for all web applications.
`default.acl`	Default access control list (ACL) file for the server instance. This file contains the default rules governing access within the server instance.
`key3.db`	NSS private key database. This database stores the private keys generated by the server and works with the `cert8.db` database for certificate management. This file is part of the instance's trust database.

Table 5.1 Web Server 7.0 Configuration Files *continued*

Filename	Overview
`keyfile`	File containing usernames and hashed passwords for flat file authentication.
`login.conf`	Information for file authentication used by the Java Authentication and Authorization Service (JAAS).
`magnus.conf`	Contains the Netscape server application programming interface (NSAPI) plug-in initialization directives and settings that control the way NSAPI plug-ins are run.
`mime.types`	Multipurpose Internet mail extension (MIME) type mapping file. This file contains a mapping of file extensions to MIME types and enables the server to determine the content type of a requested resource.
`obj.conf`	NSAPI request processing configuration file. This file contains the instructions that tell the Web Server how to handle various HTTP requests from clients.
`secmod.db`	NSS PKCS #11 module database. This database stores PKCS #11 module configuration information that is used for various cryptographic modules (for example, hardware accelerator cards).
`server.policy`	Policy controlling the access that applications have to resources.
`server.xml`	The main instance configuration file, which contains the majority of settings necessary to run the server.

Configuration files are updated when you perform administrative tasks through the Administration Console or command line interface. Additionally, you can directly edit each configuration file and modify it as appropriate.

Warning: You are permitted to edit configuration files for an instance on an Administration Node, but do not edit any files beneath the Administration Server's `config-store` directory. The files in this directory are modified by the Administration Server and command line interface and are for internal use only.

This chapter takes a closer look at each configuration file as well as the syntax, usage, and considerations for each file. Although the configuration files in Table 5.1 appear in alphabetical order, the following sections discuss Web Server 7.0's configuration files in order of importance.

5.1 The `magnus.conf` File

The `magnus.conf` file contains the directives necessary to initialize Netscape Server Application Programming Interface (NSAPI) plug-ins. Directives may also contain additional settings that control the way the NSAPI plug-ins are run.

Note: A *directive* is a statement that defines a setting within the Web Server.

The Web Server consists of functionality that enables you to perform basic request processing immediately after installation. This functionality is provided by server application functions (SAFs) that are part of the Web Server platform.

You can change the way the server responds to client requests by adding or modifying directives in the `magnus.conf` and `obj.conf` files. Each directive references a SAF that is used to perform the work during a particular stage of request processing. You can elect to add directives that call SAFs provided by Web Server 7.0, or extend the functionally of the Web Server by creating your own shared libraries (or plug-ins) through the NSAPI. Additionally, you might find it necessary to include functionality that is provided by third-party vendors or by other Sun Microsystems products.

Plug-ins must be registered with the Web Server in the `magnus.conf` file; this is accomplished with the `Init` directive. Once registered, you can use the SAFs contained within the plug-ins to customize request processing within the `obj.conf`.

Example 5.1 demonstrates the use of the `Init` directive in a customized `magnus.conf` file.

Example 5.1 Sample `magnus.conf` File

```
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
Init fn="load-modules"now)"
Init fn="load-modules" shlib="libfastcgi.so"
Init fn="load-modules"
  shlib="/sun/webserver7/plugins/myplugin/myplugin.so"
  funcs="myfunc1,myfunc2"
```

The `load-modules` SAF is used to load and register a plug-in with the Web Server. This function is most often used to load third-party plug-ins into the Web Server to provide functionality that is not included in the base platform. An example of this might include a plug-in that is used to provide an interface to an application server.

This example specifies three plug-ins that are registered during the initialization process. The `libj2eeplugin.so` and `libfastcgi.so` plug-ins are included with Web Server 7.0. The `myplugin.so` plug-in is a user-created shared library that can

be found beneath the /sun/webserver7/plugins/myplugin directory. The shlib parameter specifies the local file system path to the shared library.

Note: If you do not specify the fully qualified filename in the shlib parameter, the Web Server searches for the plug-in in the path specified by the SERVER_LIB_DIR variable defined in the startup script (startserv). If you installed your Web Server in the /sun/webserver7 directory, the default value for SERVER_LIB_DIR would be /sun/webserver7/lib.

The shlib_flags parameter (as seen in the first directive) specifies how a shared library will be loaded into the Web Server process. After it is started, the Web Server process uses the dlopen function to load additional library files into memory. The options of global and now shown in the first directive refer to the dlopen options of RTLD_GLOBAL and RTLD_NOW, respectively. You can read more about these and other options by using the UNIX man command as follows:

```
# man dlopen
```

The third Init directive also demonstrates the capability to register specific functions within a particular plug-in (i.e., myfunc1 and myfunc2). The funcs parameter is not specified in the first two directives, so all functions defined within those plug-ins are available for use.

Note: Refer to the Sun Java System Web Server 7.0 NSAPI Developer's Guide (document number 819-2632) for additional information on Web Server directives.

5.1.1 Syntax

Table 5.2 provides the general syntax for the magnus.conf file.

Table 5.2 magnus.conf Syntax

Syntax	Rules
Case Sensitivity	Items in the magnus.conf file are case-sensitive, including function names, parameter names, parameter values, and pathnames.
Comments	Hash symbols (#) are used to designate comments in the magnus.conf file. Any text that appears to the right of a hash symbol is ignored by the Web Server. The first four lines of Example 5.1 include comments.

Syntax	Rules
Directives	Directives in the `magnus.conf` file either set a value or invoke a SAF. You can add directives or edit existing directives in the `magnus.conf` file, but be very careful when doing so. Simple mistakes can make the server fail to operate correctly. There are three `Init` directives in Example 5.1. Each of these invoke the `load-modules` SAF to load the appropriate plug-in. The `Init` directives load and initialize server modules and NSAPI plug-ins. Refer to the Sun Java System Web Server 7.0 NSAPI Developer's Guide for a complete list of directives available in the `magnus.conf` file.
Directive Parameters	For predefined SAFs, the number and names of parameters depend on the function. The order of parameters on the line is not important.
Path Names	Always use forward slashes (/) rather than backslashes (\) in path names, even on the Windows platform. A backslash character is used to escape the character that follows it.
Quotation Marks	Quotation marks (") are required around the value strings only when there is a space in the string; otherwise, they are optional. Each open quotation mark must be matched by a closed quotation mark.
Spacing	Directives in the `magnus.conf` file appear on their own line. You may continue long directives on the next line by beginning the next line with a space or tab. Spaces are not allowed before or after the equal (=) sign that separates a name and value. Spaces are not allowed at the end of a line or on a blank line. Line continuation is demonstrated for the first and third directives in Example 5.1. The second line for both of these directives begins with a space.

5.1.2 Context

An Administration Node may contain multiple configurations that have been deployed to the node by the Administration Server. These deployed configurations are called instances. Each instance can contain only one `magnus.conf` file; therefore, there is a one-to-one correspondence between an instance and the `magnus.conf` file. The directives defined within the `magnus.conf` file apply to the entire server instance and therefore any virtual servers defined within that instance.

5.1.3 Modifications

Some changes made through the Administration Console or the command line interface update the `magnus.conf` file. If this file is updated as a result of changes made through either of these two interfaces, you must deploy the updated config-

uration before the changes are reflected on the appropriate Administration Node(s).

The `magnus.conf` file is read when the instance is started; therefore, any changes made to the `magnus.conf` file require a server restart for the changes to take effect. The Administration Console and command line interface detect changes to the `magnus.conf` file during the deployment process. If you use these tools to deploy the new configuration, you are prompted to restart the instance. If you directly edit the `magnus.conf` file, you need to restart the sever on your own.

The `magnus.conf` file is validated at start-up time. Errors found within the file might prevent the server from starting or processing requests properly.

5.2 The `server.xml` File

The `server.xml` file is the main configuration file for a Web Server instance. It contains initial values for listen sockets, virtual servers, and other components that were configured during the installation process. It also contains several default settings that allow the Web Server to work immediately after installation. You can modify the values for these settings through the Administration Console or the command line interface, or by editing the `server.xml` file on a particular Administration Node.

Note: The `server.xml` file is one of the most important configuration files. It is important that you understand the syntax of this file and how it is structured if you decide to manually edit it.

5.2.1 Syntax

As the name indicates, the `server.xml` file contains information in an extensible markup language (XML) format. As such, an understanding of basic XML structure and terminology is essential to understanding the `server.xml` file.

An XML document contains a hierarchy of *elements* and *values*. An element is a unit of XML data and is the basic building block of an XML document. Elements can have associated with them either values or other subelements; thus forming a parent/child relationship.

Elements are delimited by opening and closing tags that have a structure similar to what you might see if you viewed the source of an HTML document (they use the < and > brackets). Elements begin with an opening tag, for example <virtual-server>, and end with the closing tag, for example </virtual-server>. The tags are case-sensitive.

An example of the platform element for a Web Server running in 64-bit mode is as follows:

```
<platform>64</platform>
```

An XML document must contain a single *root* element, which is the topmost element of the hierarchy. Like any other XML element, the root element can contain subelements. The <server> element is the root element of the server.xml file. The <server> element has many subelements, many of which have subelements of their own.

Example 5.2 demonstrates the server.xml file for the default Administration Node.

Example 5.2 Sample server.xml File

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

<!--
  Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  Use is subject to license terms.
-->

<server>
  <cluster>
    <local-host>boulder.example.com</local-host>
    <instance>
      <host>boulder.example.com</host>
    </instance>
  </cluster>

  <log>
    <log-file>../logs/errors</log-file>
    <log-level>info</log-level>
  </log>

  <platform>64</platform>

  <temp-path>/tmp/https-boulder.example.com-6accbd0a</temp-path>
```

```
  <user>webservd</user>

  <jvm>
    <java-home>/sun/webserver7/jdk</java-home>
    <server-class-path>
/sun/webserver7/lib/webserv-rt.jar:/sun/webserver7/lib/pwc.jar:
/sun/webserver7/lib/ant.jar:${java.home}/lib/tools.jar:
/sun/webserver7/lib/ktsearch.jar:/sun/webserver7/lib/webserv-jstl.jar:
/sun/webserver7/lib/jsf-impl.jar:/sun/webserver7/lib/jsf-api.jar:
/sun/webserver7/lib/webserv-jwsdp.jar:/sun/webserver7/lib/
   container-auth.jar:
/sun/webserver7/lib/mail.jar:/sun/webserver7/lib/activation.jar
</server-class-path>
    <debug>false</debug>
    <debug-jvm-options>
-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=7896
</debug-jvm-options>
    <jvm-options>-Djava.security.auth.login.config=login.conf
      </jvm-options>
    <jvm-options>-Xms128m -Xmx256m</jvm-options>
  </jvm>

  <thread-pool>
    <max-threads>128</max-threads>
    <stack-size>131072</stack-size>
  </thread-pool>

  <default-auth-db-name>keyfile</default-auth-db-name>

  <auth-db>
    <name>keyfile</name>
    <url>file</url>
    <property>
      <name>syntax</name>
      <value>keyfile</value>
    </property>
    <property>
      <name>keyfile</name>
      <value>keyfile</value>
    </property>
  </auth-db>

  <acl-file>default.acl</acl-file>

  <mime-file>mime.types</mime-file>

  <access-log>
    <file>../logs/access</file>
  </access-log>
```

```
  <http-listener>
    <name>http-listener-1</name>
    <port>80</port>
    <server-name>boulder.example.com</server-name>
    <default-virtual-server-name>boulder.example.com</default-
      virtual-server-name>
  </http-listener>

  <virtual-server>
    <name>boulder.example.com</name>
    <host>boulder.example.com</host>
    <http-listener-name>http-listener-1</http-listener-name>
    <document-root>/sun/webserver7/https-boulder.example.com/docs
      </document-root>
  </virtual-server>
</server>
```

If a given variable name is defined at both the `<virtual-server>` and `<server>` levels, the `<virtual-server>` value takes precedence.

Note: A detailed explanation for each of the data elements found in this example can be found in Appendix A, "A Detailed Look at the `server.xml` File."

5.2.2 XML Schema

An XML schema provides a means for defining the structure, content, and semantics of XML documents. It can specify the constraints (that is, valid element names, data types, and values) on XML documents, using an XML-based language. An XML schema is hierarchical, so it is easier to create an unambiguous specification, and it's possible to determine the scope over which a comment is meant to apply.

Web Server 7.0 validates the format and content of the `server.xml` against the `sun-web-server_7_0.xsd` schema. This file can be found in the `lib/dtds` subdirectory; directly beneath the Web Server 7.0 installation directory. An understanding of this file will help you understand valid element names and the structure of those elements as they are found in the `server.xml` file.

For example, the following definition provides the structure for the <user> element:

```
<xs:element name="user" type="userType" minOccurs="0"
    maxOccurs="1"/>
```

The userType attribute describes the format of the data (such as the minimum length) and can also be found in the schema file. The <user> element is optional in the file because the minOccurs value is set to 0. If this element appears in the server.xml file, however, the maximum number of times it can occur is once (as specified by the maxOccurs value).

Note: Refer to the XML Schema Primer at http://www.w3.org/TR/xmlschema-0/ for more information.

5.2.3 Context

Each instance can contain only one server.xml file; therefore, there is a one-to-one correspondence between an instance and the server.xml file. The server.xml file contains definitions for each virtual server contained within the instance, so there is a one-to-many relationship between the server.xml file and virtual servers.

5.2.4 Modifications

Some changes made through the Administration Console or the command line interface update the server.xml file. If this file is updated as a result of changes made through either of these two interfaces, you must deploy the updated configuration before the changes are reflected on the appropriate Administration Node(s).

The server.xml file is validated against the sun-web-server_7_0.xsd schema when you start or dynamically reconfigure the instance. Errors found at that time prevent the server from starting properly. You can use the -configtest option to the startserv script to validate the server.xml file before you stop the server. This enables you to detect errors to the file without impacting a running server instance.

5.3 The obj.conf File

The obj.conf (or object configuration) file contains instructions on how to process HTTP client requests. This file consists of various directives that map directly to request processing stages and enable your Web Server to process client requests immediately after installation.

An initial object configuration file is created for each Web Server configuration; the name of the file is simply obj.conf. Additional object configuration files may also exist for any virtual servers created using the Administration Console or command line interface. The default name for each virtual server object configuration file is *vsname*-obj.conf (where *vsname* is the name of the virtual server). The <object-file> element in the server.xml file specifies the name of the object configuration file to use to process requests for that virtual server. Example 5.3 demonstrates the use of the <object-file> element in the server.xml file.

Example 5.3 Virtual Server Definition for the Object Configuration File

```
<virtual-server>
  <name>www.example.com</name>
  <http-listener-name>http-listener-1</http-listener-name>
  <host>www.example.com</host>
  <object-file>www.example.com-obj.conf</object-file>
  <document-root>/export/home/example.com/public_html</document-root>
  <access-log>
  <file>/export/home/example.com/logs/access</file>
  </access-log>
</virtual-server>
```

When the Web Server receives a request, it uses information contained within the server.xml file to select an appropriate virtual server. It then uses the file specified by the <object-file> element to determine how to process the request. Multiple object configuration files allow the flexibility to process requests differently for each virtual server.

5.3.1 File Structure

The obj.conf file contains a series of instructions (or directives) that tell the server what to do at each stage of the request-handling process. These directives are grouped together by <Object> tags. Each directive invokes a SAF with one or more arguments.

Example 5.4 demonstrates the `obj.conf` file for the default Administration Node.

Example 5.4 Default `obj.conf` File

```
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

# You can edit this file, but comments and formatting changes
# might be lost when you use the administration GUI or CLI.

<Object name="default">
AuthTrans fn="match-browser" browser="*MSIE*"
  ssl-unclean-shutdown="true"
NameTrans fn="ntrans-j2ee" name="j2ee"
NameTrans fn="pfx2dir" from="/mc-icons" dir="/opt/webserver7/lib/icons"
  name="es-internal"
PathCheck fn="unix-uri-clean"
PathCheck fn="find-pathinfo"
PathCheck fn="find-index-j2ee"
PathCheck fn="find-index" index-names="index.html,home.html,index.jsp"
ObjectType fn="type-j2ee"
ObjectType fn="type-by-extension"
ObjectType fn="force-type" type="text/plain"
Service method="(GET|HEAD)" type="magnus-internal/directory"
  fn="index-common"
Service method="(GET|HEAD|POST)" type="*~magnus-internal/*"
  fn="send-file"
Service method="TRACE" fn="service-trace"
Error fn="error-j2ee"
AddLog fn="flex-log"
</Object>

<Object name="j2ee">
Service fn="service-j2ee" method="*"
</Object>

<Object name="es-internal">
</Object>

<Object name="es-internal">
</Object>

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

```
<Object name="send-precompressed">
PathCheck fn="find-compressed"
</Object>

<Object name="compress-on-demand">
Output fn="insert-filter" filter="http-compression"
</Object>
```

5.3.2 Syntax

Directives in the object configuration file follow a syntax similar to those contained in the `magnus.conf` (refer to Table 5.2 for details). The exception is that the object configuration file also contains objects (called *templates*) for grouping directives together. Templates enable you to process directives on a conditional basis.

Note: For information on how requests are processed on a conditional basis, see Chapter 6, "Web Server 7.0 Request Processing."

The overall structure of the object configuration file is as follows:

```
<Object name="default">
directives
</Object>

<Object name="objectname">
directives
</Object>
...
<Object name="objectname">
directives
</Object>
```

The order of these templates is not important, but there must exist one template with the name `default`. The directives contained in the `default` template are used to process every request. The object configuration file for Web Server 7 contains a standard `default` template with directives for standard request processing. This enables you to process requests immediately after installation.

Warning: Do not change the name of the default template or you cannot process requests. The instance will start without a default template, but it will flag an error similar to the following during request processing:

```
[05/Aug/2007:07:28:02] config ( 3208): for host 127.0.0.1 trying to GET
/, process-uri-objects reports: HTTP2020: cannot find template default
```

Some syntax errors in the object configuration file might cause situations where the server instance cannot start, whereas others might not be noticed until a request is processed.

Examples of errors that prevent an instance from starting include an incorrect spelling of the directive name or the name of a directive parameter and can be found in the following example:

```
NameTrans func="pfx2dir" from="/mc-icons" dir="/opt/webserver7/lib/
    icons" name="es-internal"
```

In this example, the directive parameter, `fn`, has been incorrectly specified as `func`. If this were to occur, you would see an error similar to the following when you attempted to start the instance:

```
config: CONF2265: Error parsing file obj.conf, line 12, column 1:
    Missing parameter (need fn)
config: CORE3235: File server.xml line 71: Error processing
  <object-file> element: Error processing file obj.conf
failure: server initialization failed
```

Other syntax errors are not so easily noticed and might not be recognized until a request is processed. For example, if you specify an incorrect SAF in the function (`fn`) specification as follows:

```
NameTrans fn="badfunc" from="/mc-icons" dir="/opt/webserver7/lib/
    icons" name="es-internal"
```

you would see a warning message at start-up as follows:

```
config: trying to GET /, func_exec reports: HTTP2122: cannot find
    function named badfunc
```

and the following message in the errors log:

```
[22/Jul/2007:08:23:55] config ( 3546): for host 24.26.101.117 trying to
   GET /mc-icons/image.gif, func_exec reports: HTTP2122: cannot find
   function named badfunc
info: HTTP3072: http-listener-1: http://www.example.com:80 ready to
accept requests
info: CORE3274: successful server startup
```

These types of errors might not be noticed if the instance is configured to start when the server reboots. Instead, the server might start properly and flag an error message, but it isn't until a request comes in for that resource that the server error is noticed.

The following subsections contain an overview of directives and objects found in the object configuration file. Refer to Chapter 6 for a detailed explanation of request processing.

5.3.2.1 Directives

Directives in the object configuration file invoke SAFs at various request processing stages. The stage is specified as the first parameter of the directive.

Note: The `magnus.conf` file contains directives that are processed only at instance startup time. The `obj.conf` file contains directives that are processed for every request.

Each directive calls a function and specifies zero or more parameters that are necessary for the SAF to process the request at that stage. The function name and/or parameters are specified with reserved words, so the order in which they appear in the directive is not important. In general, however, the syntax for each directive in the object configuration file is as follows:

Directive `fn=`"*function*" *name1*="*value1*" ... *nameN*="*valueN*"

Where *Directive* is the stage at which the directive is processed. The value of the function (`fn`) parameter is the name of the SAF to execute. All directives must supply a value for the `fn` parameter; if there is no function, the instruction will do nothing. Function names can be composed of letters, digits, underscores (_), or

hyphens (-). The remaining parameters are the arguments needed by the function, and they vary from function to function.

An example of a directive that applies to the NameTrans (Name Translation) stage of request processing would be

```
NameTrans fn="document-root" root="/opt/Sun/webserver7/https-www.
  example.com/docs"
```

In this example, the directive is executed during the NameTrans stage of request processing and invokes the document-root SAF to specify the document root directory for the server. The document-root SAF uses one parameter, root, to specify the path to the document root directory.

Parameters can contain references to variables and expressions. The variables can be predefined variables, variables defined at request time using the set-variable SAF, or variables defined in server.xml.

It is not required, but it is a best practice to group directives according to the stage in which they are processed (for example, all NameTrans directives should be grouped together). This enables you to easily recognize and debug problems within the object configuration file.

The order in which directives appear within a particular group becomes important if the directives are conditionally executed. A common error is to place directives that are processed unconditionally before those that are conditional in nature. In such a case, the conditional directives might never be processed.

Another best practice is to place directive groups in the order of request processing stages (for example, AuthTrans, NameTrans, PathCheck, ObjectType, Service, Error, and AddLog) because this also aids in debugging.

Note: This is the order in which the default directives in the object configuration file appear. You can reorder these if you want, but if you use the Administration Console or command line interface to manage your configuration, directives are reordered to follow this format.

5.3.2.2 Objects

Directives in the object configuration file are grouped together by *objects* (which are also referred to as *containers* or *templates*). Objects are specified by the <Object> tag and enable you to define directives that are executed only on certain

conditions. Example 5.4 demonstrates various objects that are defined as part of the default object configuration file.

One of the most common attributes for the <Object> tag is the name attribute, which uniquely identifies the object within the configuration file. The syntax for objects that use the name attribute is as follows:

```
<Object name="objectname">
directives
</Object>
```

The object configuration file contains a default object that tells the instance how to process requests by default.

```
<Object name="default">
directives
</Object>
```

The object configuration file can contain objects that are executed only when certain conditions are true for a particular NameTrans directive. Two such conditions involve the use of additional named objects or ppath objects.

Named Object Processing

One such example of conditional processing is the use of the optional name attribute in the NameTrans directive. Assume that a client is requesting the following URL:

```
http://www.example.com/mc-icons/back.gif
```

During request processing, the server evaluates each NameTrans directive in the default object in an attempt to locate a match. In this case, the second directive is matched because the URI begins with /mc-icons.

```
NameTrans fn="ntrans-j2ee" name="j2ee"
NameTrans fn="pfx2dir" from="/mc-icons" dir="/opt/webserver7/lib/
  icons" name="es-internal"
```

The server then determines whether a name attribute has been specified for the directive.

```
NameTrans fn="pfx2dir" from="/mc-icons" dir="/opt/webserver7/lib/
   icons" name="es-internal"
```

If the directive specifies a `name` attribute, the value of the attribute *points to* another object that contains additional directives that should be used for processing the request.

```
<Object name="es-internal">
directives
</Object>
```

Any directives found in the additional object are processed prior to those found in the default object for the particular stage being processed.

Note: Request processing is discussed in more detail in Chapter 6.

`ppath` Object Processing

Another example of conditional processing is the use of the `ppath` (partial path) object.

```
<Object ppath="path">
directives
</Object>
```

The `ppath` object enables you to specify a path to a document or resource where directives contained in the object are executed only if the path to the resource can be found beneath the location specified in the `ppath` value. For example, suppose you specify a `ppath` object as follows:

```
<Object ppath="/opt/webserver7/https-www.example.com/docs/
   private/*">
directives
</Object>
```

During the processing of the `NameTrans` directive, if it is determined that the resource can be found beneath the `/opt/webserver7/https-www.example.com/docs/private/` directory, then any directives found within this `ppath` object are processed prior to those found in the `default` object.

In general, the server always starts processing requests with the `default` object and may process directives in other objects based on conditions within the

NameTrans directive. Each new object added to the object configuration file has the potential to modify the default object's behavior.

5.3.2.3 Variables

SAF parameters can contain references to variables and expressions. The variables can be predefined variables, variables defined at request time using the set-variable SAF, or variables defined in the server.xml file.

Note: The set-variable SAF allows the dynamic creation of variables during request processing. This can be a very powerful feature, but it also can be somewhat confusing because undefined variables are not reported until they are used (during request time).

Within the server.xml file itself, a variable can be defined at various levels (for example, <server> and <virtual-server>). As such, the server must have a method for resolving duplicate variable definitions. The server consults the following namespaces (in the following order) when attempting to resolve a variable:

1. Predefined variables
2. Variables defined at request time through the use of the set-variable SAF
3. Virtual Server–specific variables defined at the <virtual-server> level
4. Server variables defined at the <server> level

Web Server variables begin with a dollar sign character ($), followed by either upper- or lowercase letters as the next character. Subsequent characters can include any combination of upper- or lowercase alphanumeric characters or underscores (_).

A regular expression notation for variable syntax would be as follows:

```
\$[A-Za-z][A-Za-z0-9_]*
```

Examples of valid variable names include $variable, $Variable, $var_iable, or $var9. Examples of invalid variable names include $_variable and $9variable.

5.3.3 Context

The `server.xml` file contains definitions for each virtual server contained within the instance, so there is a one-to-many relationship between the `server.xml` file and virtual servers.

Each virtual server can use a different object configuration file for processing requests. Therefore, there is a one-to-many relationship between an instance and the object configuration file(s), but there is a one-to-one relationship between the virtual server and its object configuration file.

5.3.4 Modifications

Some changes made through the Administration Console or the command line interface update the appropriate object configuration file. If this file is updated as a result of changes made through either of these two interfaces, you must deploy the updated configuration before the changes are reflected on the appropriate Administration Node(s).

The object configuration file is read when the instance is started or when a dynamic reconfiguration is performed. The syntax for the directives and parameters contained in the object configuration file is validated within the start-up or reconfiguration code base. They are not validated against a schema such as the `server.xml` file.

Errors found within the file may prevent the instance from starting or processing requests properly.

5.4 The `mime.types` File

The Multipurpose Internet Mail Extensions (MIME) types file contains mappings between file extensions and MIME types. This file is utilized during request processing to tell the server what type of resource is being requested. It bases this information on the extension of the resource (such as `.txt`, `.html`, or `.cgi`) and associates a type, language, or encoding method based on the extension.

An initial MIME types file is created for each Web Server configuration; the name of the file is simply `mime.types`. You may create additional MIME types files and associate them to different virtual servers by using the `<mime-file>` element in the

server.xml. Example 5.5 demonstrates the use of the <mime-file> element in the server.xml file.

Example 5.5 Virtual Server Definition for the MIME File

```
<virtual-server>
  <name>www.example.com</name>
  <http-listener-name>http-listener-1</http-listener-name>
  <host>www.example.com</host>
  <object-file>www.example.com-obj.conf</object-file>
  <mime-file>mymime.types</mime-file>
  <document-root>/export/home/example.com/public_html</document-root>
  <access-log>
     <file>/export/home/example.com/logs/access</file>
  </access-log>
</virtual-server>
```

Associating different MIME files to each virtual server gives you the flexibility to support different resources at the virtual server level.

5.4.1 File Structure

The mime.types file contains a series of associations that helps the instance identity the request type and subsequently understand how to process the request. Example 5.6 demonstrates the MIME types file for the default Administration Node.

Example 5.6 Sample mime.types File

```
#--Sun Microsystems Inc. MIME Information
# Do not delete the above line. It is used to identify the file type.
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

type=application/octet-stream                    exts=bin
type=application/astound                         exts=asd,asn
type=application/fastman                         exts=lcc
type=application/java-archive                    exts=jar
type=application/java-serialized-object          exts=ser
type=application/java-vm                         exts=class
type=application/mac-binhex40                    exts=hqx
type=application/x-stuffit                       exts=sit
type=application/mbedlet                         exts=mbd
```

```
type=application/msword                          exts=doc,dot,wiz,rtf
type=application/oda                             exts=oda
type=application/pdf                             exts=pdf
type=application/postscript                      exts=ai,eps,ps
type=application/studiom                         exts=smp
type=application/timbuktu                        exts=tbt
type=application/vnd.ms-excel                    exts=xls,xlw,xla,
                                                 xlc,xlm,xlt
type=application/vnd.ms-powerpoint               exts=ppt,pps,pot

[entries deleted]

enc=x-gzip                                       exts=gz
enc=x-compress                                   exts=z
enc=x-uuencode                                   exts=uu,uue

type=magnus-internal/parsed-html                 exts=shtml
type=magnus-internal/cgi                         exts=cgi,exe,bat

type=application/x-x509-ca-cert                  exts=cacert
type=application/x-x509-server-cert              exts=scert
type=application/x-x509-user-cert                exts=ucert
type=application/x-x509-email-cert               exts=ecert

type=application/vnd.sun.xml.writer              exts=sxw
type=application/vnd.sun.xml.writer.template     exts=stw
type=application/vnd.sun.xml.calc                exts=sxc
type=application/vnd.sun.xml.calc.template       exts=stc
type=application/vnd.sun.xml.draw                exts=sxd
type=application/vnd.sun.xml.draw.template       exts=std
type=application/vnd.sun.xml.impress             exts=sxi
type=application/vnd.sun.xml.impress.template    exts=sti
type=application/vnd.sun.xml.writer.global       exts=sxg
type=application/vnd.sun.xml.math                exts=sxm

type=application/vnd.stardivision.writer         exts=sdw
type=application/vnd.stardivision.writer-global  exts=sgl
type=application/vnd.stardivision.calc           exts=sdc
type=application/vnd.stardivision.draw           exts=sda
type=application/vnd.stardivision.impress        exts=sdd
type=application/vnd.stardivision.impress-packed exts=sdp
type=application/vnd.stardivision.math           exts=smf,sdf
type=application/vnd.stardivision.chart          exts=sds
type=application/vnd.stardivision.mail           exts=sdm
```

5.4.2 File Structure

The format is similar to the format of request/response information that is exchanged between a web browser and the Web Server to which it connects. This related format is specified as part of the Hypertext Transfer Protocol (HTTP) and has the following rules:

The first line in the MIME types file identifies the file format:

```
#--Sun Microsystems MIME Information
```

Other uncommented lines have the following format:

```
type=type/subtype exts=[file extensions]
```

where `type/subtype` refers to the MIME type and `subtype` and `exts` refers to the file extensions associated with this type. For example, the MIME types file uses the following mapping to associate the extensions `.html` and `.htm` to the type `text/html`:

```
type=text/html exts=htm,html
```

Note: For the definitive information on HTTP, you should consult RFC 2616, "Hypertext Transfer Protocol—HTTP/1.1."

5.4.3 Processing

The MIME types configuration file determines how your Web Server's virtual server maps filename extensions to MIME types that are returned to the browser. Your browser then maps these MIME types to "helper" applications or in-line plug-ins.

The following subsections describe how MIME mappings are used on both the Web Server and the user agent during request/response processing.

5.4.3.1 Server Processing

When the Web Server receives a request for a resource from a client, it uses the MIME type mappings to determine what kind of resource is being requested.

During the ObjectType stage in the request handling process, the server determines the MIME type attribute of the resource requested by the client. Several different SAFs can be used to determine the MIME type, but the most commonly used one is type-by-extension. This function tells the server to look up the MIME type according to the requested resource's file extension in the MIME types table (stored in the MIME types file).

MIME types are defined by three attributes: language (lang), encoding (enc), and content-type (type). At least one of these attributes must be present for each type. The most commonly used attribute is type. When the server sends the response to the client, the type, language, and encoding values are transmitted in the Content-Type headers of the response.

Note: The server frequently considers the type attribute when deciding how to generate the response to the client in the Service directive stage. (The enc and lang attributes are rarely used.)

If there is more than one ObjectType directive, the server applies all the directives in the order in which they appear. However, after a directive sets an attribute of the MIME type, encoding, or language, further attempts to set the same attribute are ignored.

Figure 5.1 demonstrates a client making a request for a particular file and how the server uses the mime.types file to determine how to process that request.

The following steps indicate a high-level overview of how the server processes this request:

1. A client makes a URL request of http://www.example.com/index.html.
2. The server accepts the request and selects the appropriate virtual server (not shown).
3. The server determines the name of the object configuration file, based on the virtual server's object-file settings (not shown). This instructs the server on how to process the request (not shown).
4. The server executes the request-response process, as dictated by the object configuration file.

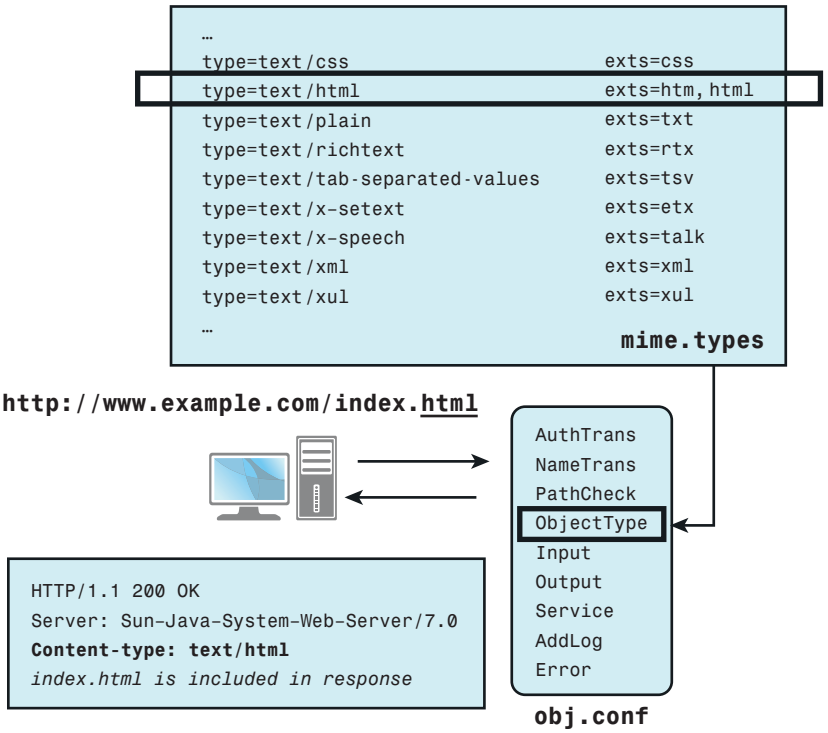

Figure 5.1 MIME File Usage During Request Processing

5. At the point when the server reaches the `ObjectType` stage, it reads the MIME types file and searches for an extension that matches the file requested (in this case it is an HTML file).
6. The server finds a match and sets the type to `text/html`.
7. The server selects the appropriate service to process the request (not shown).
8. The server sends the response to the client with the appropriate `Content-type` set in the response header.

5.4.3.2 Client Processing

The SAF defined for the appropriate service stage generates the data and sends it back to the client that made the request. When the server sends the data to the client, it also sends headers. These headers include the MIME type attributes that are known (which are usually defined by type attributes).

The client maintains its own MIME types table, as shown in Figure 5.2.

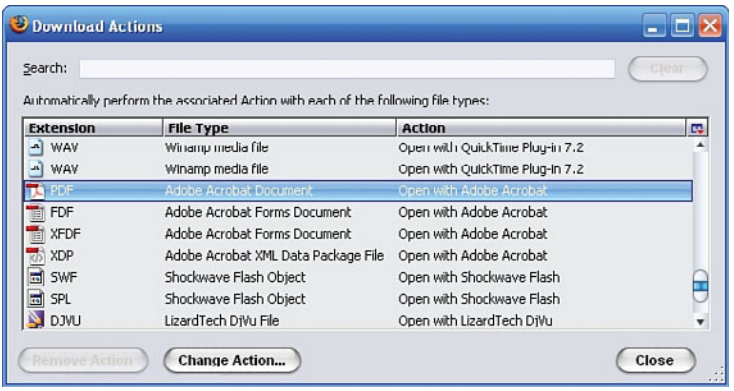

Figure 5.2 Client MIME Definitions

When the client receives the header, it uses the information contained in its own MIME types table to determine what to do with the data. For browser clients, the usual action is to display the data in the browser window.

Sometimes the requested resource cannot be displayed in the browser window and needs to be handled by another application. In such cases, the `Content-type` sent back by the server typically starts with `application/`; for example, `application/x-pdf` (for `.pdf` file extensions) or `application/x-maker` (for `.fm` file extensions).

The client has its own set of user-editable mappings that tells the browser which application to use for certain data types. For example, if the type is `application/x-pdf`, the client usually handles it by opening the Adobe Acrobat Reader application to display the file.

5.4.4 Context

The `server.xml` file contains definitions for each virtual server contained within the instance, so there is a one-to-many relationship between the `server.xml` file and virtual servers.

Each virtual server can use a different MIME types file for mapping associations. Therefore, there is a one-to-many relationship between an instance and the MIME types file(s), but there is a one-to-one relationship between the virtual server and its MIME types file.

5.4.5 Modifications

Although the default MIME types file includes a definition of the most commonly known MIME types, you are free to modify the file to add support for any additional MIME types. To add a new MIME type definition, simply append the definition to the existing MIME types in the file in the following format (where `type/subtype` is the MIME type of the document whose filename ends with one of the extensions listed):

```
type=type/subtype exts=extension1,extension2,...,extensionN
```

The extension list includes any number of comma-separated filename extensions. Examples of MIME type entries can be found in the default MIME types file included with your virtual server.

You can also use the Administration Console or the command line interface to update the MIME types file for the virtual server. After the file is updated, you must deploy the updated configuration before the changes are reflected on the appropriate Administration Node(s).

The MIME types file is read when the instance is started or when a dynamic reconfiguration is performed. The syntax contained in the MIME types file is validated within the start-up or reconfiguration code base. Errors found within the MIME types file or within the `server.xml` file (as it pertains to the `<mime-file>` attribute) may prevent the instance from starting.

5.5 Trust Database Files (* . db Files)

The Web Server stores security-based information in three Network Security Services (NSS) `libdbm` database files as follows:

- `cert8.db`—Stores publicly accessible objects (such as certificates, certificate revocation lists, and S/MIME records)
- `key3.db`—Stores the private keys generated by the server
- `secmod.db`—Stores PKCS #11 module configuration information

The combination of these files is commonly called the *trust database*, and each file plays a different role in securing your Web Server.

For example, the `cert8.db` and `key3.db` files are used to store public and private keys and certificates used for enabling secure socket layer (SSL). The `secmod.db`

file stores information for enabling and configuring additional security modules that can be used with the Web Server (such as hardware accelerator cards). See Chapter 8, "Securing Web Server 7.0," for more information about how these files are used to provide security for your Web Server.

Note: NSS is a set of libraries designed to support cross-platform development of security-enabled client and server applications. Applications built with NSS can support SSL v2 and v3, TLS, PKCS #5, PKCS #7, PKCS #11, PKCS #12, S/MIME, X.509 v3 certificates, and other security standards. For more information on NSS, go to https://www.mozilla.org/projects/security/pki/nss/.

5.5.1 File Structure

Trust database files are formatted in a Berkeley DB 1.85 hash format so they are not viewable or editable with a standard text editor. You can, however, use a hex editor or the UNIX `strings` command to obtain an insight into the contents of these files. Example 5.7 provides a sample of the `cert8.db` file.

Example 5.7 Snippet of Hex Output from the `cert8.db` File

```
.....
0000bc10  00 00 08 03 00 00 01 00 15 00 00 63 65 72 74 2d  ...........cert-
0000bc20  66 6f 6f 2e 65 78 61 6d 70 6c 65 2e 63 6f 6d 00  foo.example.com.
0000bc30  00 6f 00 00 00 87 1b db 31 30 68 31 0b 30 09 06  .o...‡.Û1Oh1.O..
0000bc40  03 55 04 06 13 02 55 53 31 0b 30 09 06 03 55 04  .U....US1.O...U.
0000bc50  08 13 02 46 4c 31 0e 30 0c 06 03 55 04 07 13 05  ...FL1.O...U....
0000bc60  54 61 6d 70 61 31 0d 30 0b 06 03 55 04 0a 13 04  Tampa1.O...U....
0000bc70  54 65 73 74 31 13 30 11 06 03 55 04 0b 13 0a 57  Test1.O...U....W
0000bc80  65 62 20 53 65 72 76 65 72 31 18 30 16 06 03 55  eb Server1.O...U
0000bc90  04 03 13 0f 66 6f 6f 2e 65 78 61 6d 70 6c 65 2e  ....foo.example.
0000bca0  63 6f 6d 03 30 68 31 0b 30 09 06 03 55 04 06 13  com.Oh1.O...U...
0000bcb0  02 55 53 31 0b 30 09 06 03 55 04 08 13 02 46 4c  .US1.O...U....FL
0000bcc0  31 0e 30 0c 06 03 55 04 07 13 05 54 61 6d 70 61  1.O...U....Tampa
0000bcd0  31 0d 30 0b 06 03 55 04 0a 13 04 54 65 73 74 31  1.O...U....Test1
0000bce0  13 30 11 06 03 55 04 0b 13 0a 57 65 62 20 53 65  .O...U....Web Se
0000bcf0  72 76 65 72 31 18 30 16 06 03 55 04 03 13 0f 66  rver1.O...U....f
0000bd00  6f 6f 2e 65 78 61 6d 70 6c 65 2e 63 6f 6d 08 01
oo.example.com..
.....
```

If you look closely at the text portion of the file (the far right column), you can see certain data elements that can be found within the details of the certificate shown in Table 5.3.

Table 5.3 Sample Certificate

Attribute	Value
Nickname	`cert-foo.example.com`
Subject	`CN=foo.example.com,OU=Web Server,O=Test,L=Tampa, ST=FL,C=US`
Issuer	`Self Signed`
Key Type	`RSA`
Key Size (bits)	`1024`
Valid From	`August 29, 2007 7:21:30 PM EDT`
Valid Till	`August 29, 2008 7:21:30 PM EDT`
Finger Print	`0B:DE:C8:80:17:38:EC:C6:6F:98:5A:5C:8F:3A:54:76`
Serial Number	`00:87:1B:DB:31`

5.5.2 Context

Each server instance has its own trust database; therefore, there is a one-to-one correspondence between the server instance and the trust database.

5.5.3 Modifications

You cannot edit files within the trust database directly. These files must be managed with the Administration Console or the command line interface. If any of these files are updated through either of these two interfaces, you must deploy the updated configuration before the changes are reflected on the appropriate Administration Node(s).

5.6 The server.policy File

Web Server 7 is a Java EE 1.4–compliant web server. As such, it follows the recommendations and requirements of the Java EE specification, including the optional presence of the Security Manager, which is the Java component that enforces policy, and a limited permission set for Java EE application code.

Each Web Server instance has its own standard Java Platform, Standard Edition (Java SE platform) server policy file named `server.policy`. The server policy file controls the access that applications have to the resources such as files on the file system.

5.6.1 Syntax

Directives in the server policy file grant explicit permission to access a particular resource. Without this permission, they are implicitly denied access. Server policy directives adhere to the following syntax:

```
grant [codeBase "path"] {
permission permission_class "package", "permission_type";
...
};
```

For example, the following directive grants web applications explicit permission to access shared system library files:

```
grant codeBase "file:/usr/share/lib/-" {
   permission java.security.AllPermission;
};
```

Note: Refer to the *Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications* for more information on how to modify the server policy file for your web application.

5.6.2 Context

Each server instance has its own server policy file; therefore, there is a one-to-one correspondence between the server instance and the server policy file.

5.6.3 Modifications

In Web Server 7, the Java SE SecurityManager (the Java component that enforces the policy) is not active by default. The policies granted in the server policy file do not have any effect unless the SecurityManager is enabled in the `server.xml`. You can enable the Java SE SecurityManager by adding the following Java Virtual Machine (JVM) options to the `server.xml` file:

```
<jvm>
   <jvm-options>-Djava.security.manager</jvm-options>
   <jvm-options>-Djava.security.policy=instance_dir/config/
     server.policy
   </jvm-options>
</jvm>
```

You can also add JVM options by using the Administration Console or the command line interface. After this has been performed, you must deploy the updated configuration before the changes are reflected on the appropriate Administration Node(s).

The Administration Console and command line interface do not provide a method for managing the server policy file. As such, directives must be added to the `server.policy` file directly on a particular Administration Node. After this has been performed, the modifications must be pulled back into the configuration and then pushed out to additional Administration Nodes as appropriate.

5.7 The `certmap.conf` File

Web Server 7 can be configured to allow client authentication through the use of an X.509 digital certificate. This can be performed under the SSL settings for a particular HTTP listener.

When a server receives a request from a client, it can ask for the client's certificate before proceeding. A client is programmed to respond by sending a client certificate to the server.

After checking that a client certificate chain ends with a trusted Certificate Authority (CA), the Web Server can optionally determine which user is identified by the client certificate and then look up that user's entry in a directory server. The Web Server authenticates the user by comparing the information in the certificate with the data in the user's directory entry.

To locate user entries in the directory, a server must know how to interpret certificates from different CAs. You provide the server with interpretation information by means of the certificate mapping configuration file (`certmap.conf`). This file provides three kinds of information for each listed CA:

- It maps the distinguished name (DN) in the certificate to a branch point in the LDAP directory.
- It specifies which DN values from the certificate (username, e-mail address, and so on) the server should use for the purpose of searching the directory.
- It specifies whether the server should go through an additional verification process. This process involves comparing the certificate presented by the client for authentication with the certificate stored in the user's directory entry. By comparing the certificate, the server determines whether to allow access or whether to revoke a certificate by removing it from the user's directory entry.

If more than one directory entry contains the information in the user's certificate, the server can examine all matching entries to determine which user is trying to authenticate. When examining a directory entry, the server compares the presented certificate with the certificate stored in the entry. If the presented certificate doesn't match any directory entries or if matching entries don't contain matching certificates, client authentication fails.

After the server finds a matching entry and certificate in the directory, it can determine the appropriate kind of authorization for the client. For example, some servers use information from a user's entry to determine group membership, which in turn can be used during evaluation of access control instruction ACIs to determine what resources the user is authorized to access.

Note: See Chapter 8 for more information about access control instructions.

5.7.1 File Structure

The `certmap.conf` file contains information on how a client certificate is mapped to a directory server entry. Example 5.8 demonstrates the certificate mapping configuration file for the default Administration Node.

Example 5.8 Default `certmap.conf` File

```
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#
# This file configures how a certificate is mapped to an LDAP entry.
# See the documentation for more information on this file.
#
# The format of this file is as follows:
#   certmap <name> <issuerDN>
#   <name>:<prop1> [<val1>]
#   <name>:<prop2> [<val2>]
#
# Notes:
#
# 1.  Mapping can be defined per issuer of a certificate.  If mapping
#    doesn't exists for a particular 'issuerDN' then the server uses
#    the default mapping.
#
# 2.  There must be an entry for <name>=default and issuerDN "default".
#    This mapping is the default mapping.
#
# 3.  '#' can be used to comment out a line.
#
# 4.  DNComps & FilterComps are used to form the base DN and filter
#    resp. for performing an LDAP search while mapping the cert to a
#    user entry.
#
# 5.  DNComps can be one of the following:
#   commented out - take the user's DN from the cert as is
#   empty        - search the entire LDAP tree (DN == suffix)
#   attr names    - a comma separated list of attributes to form DN
#
```

```
# 6.  FilterComps can be one of the following:
#    commented out - set the filter to "objectclass=*"
#    empty        - set the filter to "objectclass=*"
#    attr names    - a comma separated list of attributes to form the
#                    filter
#

certmap default       default
#default:DNComps
#default:FilterComps    e, uid
#default:verifycert    on
#default:CmapLdapAttr   certSubjectDN
#default:library       <path_to_shared_lib_or_dll>
#default:InitFn      <Init function's name>
```

5.7.2 Syntax

The file contains one or more named mappings, each applying to a different certificate authority (CA).

Hash symbols (#) at the beginning of a line indicate that the line is a comment. These are ignored by Web Server. Additional lines have the following syntax:

```
certmap <name> <issuerDN>
<name>:<property> [<value>]
```

The first line uses the `name` parameter to specify a name for the certificate mapping. It uses the `issuerDN` parameter to specify the attributes that form the distinguished name found in the CA certificate.

The `name` attribute is arbitrary; you can call it whatever you want it to be. The `issuerDN` attribute, however, must exactly match the distinguished name of the CA who issued the client certificate. For example, the following two `issuerDN` lines differ only in the spaces separating the attributes, but the server treats these two entries as different:

```
certmap name ou=Example CA,o=Example,c=US
certmap name ou=Example CA, o=Example, c=US
```

The second and subsequent lines of a mapping identify the rules that the server should use when searching the directory for information extracted from a certificate:

```
certmap name issuerDN
name:property1 [value1]
name:property2 [value2]
...
```

These rules are specified through the use of one or more of the following properties: `DNComps`, `FilterComps`, `VerifyCert`, `CmapLdapAttr`, `Library`, and `InitFn`. Additionally, you could use the certificate API to customize your own properties.

Table 5.4 describes each of the properties contained in the certificate mapping configuration file.

Table 5.4 `certmap.conf` Properties

Property	Description
`DNComps`	A list of comma-separated attributes used to determine where in the LDAP directory the server should start searching for entries that match the user's information (that is, the owner of the client certificate). The server gathers values for these attributes from the client certificate and uses the values to form an LDAP DN, which then determines where the server starts its search in the LDAP directory. For example, if you set `DNComps` to use the organization (o) and country attributes of the DN, the server starts the search from the `o=`*`<organization>`*`, c=`*`<country>` entry in the LDAP directory, where *`<organization>`* and *`<country>`* are replaced with values from the DN in the certificate. Note the following situations: • If there isn't a `DNComps` entry in the mapping, the server uses either the `CmapLdapAttr` setting or the entire subject DN in the client certificate. • If the `DNComps` entry is present but has no value, the server searches the entire LDAP tree for entries matching the filter.

Table 5.4 `certmap.conf` Properties *continued*

Property	Description
`FilterComps`	A list of comma-separated attributes used to create a filter by gathering information from the user's DN in the client certificate. The server uses the values for these attributes to form the search criteria used to match entries in the LDAP directory. If the server finds one or more entries in the LDAP directory that match the user's information gathered from the certificate, the search is successful and the server optionally performs a verification. For example, if `FilterComps` is set to use the e-mail and user ID attributes (`FilterComps=e,uid`), the server searches the directory for an entry whose values for e-mail and user ID match the end user's information gathered from the client certificate. E-mail addresses and user IDs are good filters because they are usually unique entries in the directory. The filter needs to be specific enough to match only one entry in the LDAP database. The following provides a list of valid X.509v3 certificate attributes: • `c`—Country • `o`—Organization • `cn`—Common name • `l`—Location • `st`—State • `ou`—Organizational unit • `uid`—User ID • `e` or `mail`—E-mail address Note that the attribute names for the filters must be attribute names from the certificate, not from the LDAP directory. For example, some certificates have an `e` attribute for the user's e-mail address; whereas LDAP calls that attribute `mail`.
`verifycert`	Tells the server whether it should compare the client's certificate with the certificate found in the LDAP directory. It takes two values, `on` and `off`. You should use this property only if your LDAP directory contains certificates. This feature is useful to ensure your end users have a valid, unrevoked certificate.
`CmapLdapAttr`	A name for the attribute in the LDAP directory that contains subject DNs from all certificates belonging to the user. The default for this property is `certSubjectDN`. This attribute isn't a standard LDAP attribute, so to use this property, you must extend the LDAP schema. Additionally, the search takes place more quickly if the attribute specified by `CmapLdapAttr` is indexed. If this property exists in the `certmap.conf` file, the server searches the entire LDAP directory for an entry whose attribute (named with this property) matches the subject's full DN (taken from the certificate). If the search doesn't find any entries, the server retries the search, using the `DNComps` and `FilterComps` mappings. This approach to matching a certificate to an LDAP entry is useful when it's difficult to match entries using `DNComps` and `FilterComps`.
`Library`	A property whose value is a pathname to a shared library or DLL. You need to use this property only if you create your own properties using the certificate API. For more information, see the *Sun Java System Web Server 7.0 Developer's Guide*.
`InitFn`	A property whose value is the name of an initialization (`init`) function from a custom library. You need to use this property only if you create your own properties by using the certificate API.

If you are using the `Library` and `InitFn` properties, a complete mapping might look like this:

```
certmap Example CA        ou=Example CA, o=Example, c=US
Example CA:CmapLdapAttr   certSubjectDN
Example CA:DNComps        o, c
Example CA:FilterComps    e, uid
Example CA:VerifyCert     on
```

In this example, the name of the certificate mapping is Example CA, and it is mapped to the issuer's distinguished name of `ou=Example CA, o=Example, c=US`.

Note: Note how the remaining lines of the mapping's properties begin with the name of the certificate mapping (Example CA). This is how the Web Server knows that the properties apply to this mapping.

The `CmapLdapAttr` property tells the server to search the entire directory for an entry whose `certSubjectDN` matches the subject's full DN as listed in the client certificate. If the search doesn't yield any entries, the server retries the search, using the `DNComps` and `FilterComps` mappings.

The `DNComps` property indicates that the base distinguished name (`baseDN`) of the search should be created from the organization and country attributes that are extracted from the client certificate.

The `FilterComps` property indicates that the Web Server should search for entries that match the e-mail and user ID attributes extracted from the client certificate.

The `VerifyCert` property tells the server whether it should compare the client's certificate with the certificate found in the user's directory entry. A value of `on` ensures that the server does not authenticate the client unless the certificate presented exactly matches the certificate stored in the directory.

5.7.3 Context

Each server instance has its own certificate mapping configuration file; therefore, there is a one-to-one correspondence between the server instance and this file.

5.7.4 Modifications

The Administration Console and command line interface do not provide a method for managing the certificate mapping configuration file. As such, entries must be added to the `certmap.conf` file directly on a particular Administration Node. After this has been performed, the modifications must be pulled back into the configuration and then pushed out to additional Administration Nodes as appropriate.

5.8 The `default.acl` File

You can control access to the entire Web Server or to parts of the server (for example, directories, files, and file types). When the Web Server evaluates an incoming request, it determines access based on a hierarchy of rules called access control entries (ACEs). The Web Server uses the matching entries to determine whether the client request is allowed or denied.

The collection of ACEs is called an access control list (ACL). The Web Server processes the ACL list from top to bottom. If at any time an ACE evaluates to a value of `false`, then processing of the ACL stops and the user is denied access to the resource.

ACL files contain rules that define who can access resources stored on the Web Server. By default, the Web Server uses one ACL file (`default.acl`) that contains the default access list. You can change access control rules by editing this file or by creating additional ACL files for one or more instances. If you create additional ACL files, you can associate them to different virtual servers by using the `<acl-file>` element in the `server.xml`.

Example 5.9 demonstrates the use of the `<acl-file>` element to specify an additional ACL file for a particular virtual server:

Example 5.9 Virtual Server Definition for an ACL File

```
<virtual-server>
  <name>www.example.com</name>
  <http-listener-name>http-listener-1</http-listener-name>
  <host>www.example.com</host>
  <object-file>www.example.com-obj.conf</object-file>
  <mime-file>mime.types</mime-file>
  <acl-file>example.acl</acl-file>
```

```
  <document-root>/export/home/example.com/public_html</document-root>
  <access-log>
  <file>/export/home/example.com/logs/access</file>
  </access-log>
</virtual-server>
```

Associating different ACL files to each virtual server provides you with the flexibility to provide different access control at the virtual server level.

5.8.1 File Structure

The ACL file is a text file containing one or more ACLs. Example 5.10 demonstrates the access control list file for the default Administration Node.

Example 5.10 Default `default.acl` File

```
#--Sun Microsystems Inc. MIME Information
# Do not delete the above line. It is used to identify the file type.
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

version 3.0;
acl "default";
authenticate (user, group) {
  prompt = "Sun Java System Web Server";
};
allow (read, execute, info) user = "anyone";
allow (list, write, delete) user = "all";

acl "es-internal";
allow (read, execute, info) user = "anyone";
deny (list, write, delete) user = "anyone";
```

These access control rules allow anyone to read resources on the server but restrict listing, writing, and deleting resources to authenticated users.

5.8.2 Syntax

All ACL files must follow a specific format and syntax. All ACL files must begin with the version number they use. There can be only one version line and it can appear after any comment line. Web Server 7 uses version 3.0. For example:

```
version 3.0;
```

You can include comments in the file by beginning the comment line with the hash (#) character.

Each ACL in the file begins with a statement that defines its type. ACLs can follow one of the following three types:

- Path ACLs specify an absolute path to the resource they affect.
- URI ACLs specify a directory or file relative to the server's document root.
- Named ACLs specify a name that is referenced in the `obj.conf` file. Web Server 7 comes with a `default` named resource that allows read access to all users and write access to users in the LDAP directory. Even though you can create a named ACL from the Web Server user interface, you must manually reference the named ACLs with resources in the `obj.conf` file.

Path and URI ACLs can include a wildcard at the end of the entry, for example, `/a/b/*`. Wildcards placed anywhere except at the end of the entry do not work.

The type line begins with the letters `acl` and includes the type information in double quotation marks followed by a semicolon. Each type information for all ACLs must be a unique name, even among different ACL files. The following lines are examples of several different types of ACLs:

```
acl "default";
...
acl "path=C:/docs/mydocs/";
...
acl "uri=/mydocs/";
...
```

After you define the type of ACL, you can have one or more statements that define the method used with the ACL (authentication statements) and the users and computers who are allowed or denied access (authorization statements). The following sections describe the syntax for these statements.

5.8.2.1 General Syntax

Input strings can contain the following characters:

- Alphanumeric characters a through z or 0 through 9
- A period (.) or underscore (_) character

If you use any other characters, add double quotation marks around them. A single statement can be placed on its own line and terminated with a semicolon. Multiple statements are placed within braces. A list of items must be separated by commas and enclosed in double quotation marks.

5.8.2.2 Authentication Methods

ACLs can optionally specify the authentication method that the server must use when processing the ACL. There are three methods:

- `basic`
- `digest`
- `ssl`

The `basic` and `digest` methods both require users to enter a username and password before accessing a resource. The two differ, however, in the manner in which data is transmitted between the client and the Web Server and how the password is verified within the authentication database.

The `basic` authentication method enables users to enter a username and password and transmits the information to the Web Server in known Base 64-Encoded text. For security reasons, `basic` authentication is almost always used with SSL connections so that the username and password are encrypted before being sent from the client.

The `digest` authentication method prevents the client's username and password from being sent in known text. The browser uses the MD5 algorithm to create a message digest using the password and information sent by Web Server. The digest value is computed and compared on the server side using the Digest Authentication plug-in.

The `ssl` method requires the user to have a client certificate. The Web Server must have the encryption turned on, and the user's certificate issuer must be in the list of trusted certificate authorities (CAs) to be authenticated. See Section 5.7, "The `certmap.conf` File," for more information on using client certificates for authentication.

By default, the server uses the `basic` method for any ACL that does not explicitly specify a method. If you use the `digest` method, the server's authentication database must be able to handle `digest` authentication. Authentication databases are configured in `server.xml` with the `<auth-db>` element. See Chapter 8 for more information on configuring authentication databases.

Each authenticated line must specify the attribute (users, groups, or both users and groups) that the server authenticates. The following authentication statement, which appears after the ACL type line, specifies basic authentication with users matched to individual users in the database or directory:

```
authenticate (user) { method = "basic"; };
```

The following example uses `ssl` as the authentication method for users and groups:

```
authenticate (user, group) { method = "ssl"; };
```

The following example allows any user whose username begins with `sales`:

```
authenticate (user)
allow (all)
user = sales*
```

If the last line is changed to `group = sales`, the ACL will fail because the group attribute is not authenticated.

5.8.2.3 Authorization Statements

Each ACL entry can include one or more authorization statements. Authorization statements specify who is allowed or denied access to a server resource. Use the following syntax to write authorization statements:

```
allow|deny [absolute] (right[,right...]) attribute expression;
```

Start each line with either `allow` or `deny`. Because of the hierarchy rules, it is usually a good practice to deny access to everyone in the first rule and then specifically allow access for users, groups, or computers in subsequent rules. That is, if you allow anyone access to a directory called `/my_stuff`, and you have a subdirectory `/my_stuff/personal` that allows access to a few users, the access control on the subdirectory does not work because anyone allowed access to the `/my_stuff` directory is also allowed access to the `/my_stuff/personal` directory. To prevent

this, create a rule for the subdirectory that first denies access to anyone and then allows it for the few users who need access.

In some cases, if you set the default ACL to deny access to everyone, your other ACL rules do not need a `deny all` rule.

The following line denies access to everyone:

```
deny (all) user = "anyone";
```

5.8.2.4 Hierarchy of Authorization Statements

ACLs have a hierarchy that depends on the resource. For example, if the server receives a request for the document (URI) `/my_stuff/web/presentation.html`, the server builds a list of ACLs that apply for this URI. The server first adds ACLs listed in the `check-acl` statement of its `obj.conf` file. Then the server appends matching `URI` and `PATH` ACLs.

The server processes this list in the same order. Unless absolute ACL statements are present, all statements are evaluated in order. If an `absolute allow` or `absolute deny` statement evaluates to `true`, the server stops processing and accepts this result.

If more than one ACL matches, the server uses the last statement that matches. However, if you use an absolute statement, the server stops looking for other matches and uses the ACL containing the absolute statement. If you have two absolute statements for the same resource, the server uses the first one in the file and stops looking for other resources that match.

Example 5.11 demonstrates how an ACL might be written to allow `user1` access to the content found at `/my_stuff/web/presentation.html`.

Example 5.11 Sample ACL Entries to Protect Web Server Content

```
version 3.0;
acl "default";
authenticate (user, group) {
prompt = "Sun Java System Web Server";
};
allow (read, execute, info) user = "anyone";
allow (list, write, delete) user = "all";
acl "uri=/my_stuff/web/presentation.html";
deny (all) user = "anyone";
allow (all) user = "user1";
```

5.8.2.5 Expression Attribute

Attribute expressions define *who* is allowed or denied access based on username, group name, hostname, or IP address. The following are examples of allowing access to different users or computers:

- `user = "anyone"`
- `user = "smith*"`
- `group = "sales"`
- `dns = "*.sun.com"`
- `dns = "*.sun.com,*.mozilla.com"`
- `ip = "198.*"`
- `ciphers = "rc4"`
- `ssl = "on"`

You can also restrict access to your server by time of day (based on the local time on the server) by using the `timeofday` attribute. For example, you can use the `timeofday` attribute to restrict access to certain users during specific hours.

Note: Use 24-hour time to specify times. For example, use 0400 to specify 4:00 a.m. or 2230 for 10:30 p.m.

The following example restricts access to a group of users called guests between 8:00 a.m. and 5:00 p.m.:

```
allow (read) (group="guests") and (timeofday<0800 or timeofday=1700);
```

You can also restrict access by day of the week. Use the following three-letter abbreviations to specify days: `Sun`, `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, and `Sat`.

The following statement allows access for users in the premium group any day and any time. Users in the discount group get access all day on weekends and on weekdays, any time except 8:00 a.m. to 5:00 p.m.

```
allow (read) (group="discount" and dayofweek="Sat,Sun") or
   (group="discount" and (dayofweek="mon,tue,wed,thu,fri"
   and(timeofday<0800 or timeofday=1700)))or (group="premium");
```

5.8.2.6 Expression Operators

You can use various operators in an expression. Parentheses delineate the operator order of precedence. With `user`, `group`, `dns`, and `ip`, you can use the following operators:

- `and`
- `or`
- `not`
- `=` (equals)
- `!=` (not equal to)

With `timeofday` and `dayofweek`, you can use

- `>` (greater than)
- `<` (less than)
- `>=` (greater than or equal to)
- `<=` (less than or equal to)

5.8.3 Context

The `server.xml` file contains definitions for each virtual server contained within the instance, so there is a one-to-many relationship between the `server.xml` file and virtual servers.

Each virtual server can use one or more access control list files for protecting resources specific to that virtual server; therefore, there is a one-to-many relationship between an instance and the access control list file(s). Additionally, there is a one-to-many relationship between the virtual server and its access control list file(s).

5.8.4 Modifications

Administrators can update any of the access control list files through the Administration Console or the command line interface. If this file is updated as a result of changes made through either of these two interfaces, you must deploy the updated configuration before the changes are reflected on the appropriate Administration Node(s).

Access control list files are read when the instance is started or when a dynamic reconfiguration is performed. The syntax for the entries contained in these files is validated within the start-up or reconfiguration code base. They are not validated against a schema like the `server.xml` file is.

Errors found within the file may prevent the instance from starting or processing requests properly.

5.9 The `default-web.xml` File

The `default-web.xml` is a virtual server–specific web deployment descriptor file whose configuration settings are inherited by all web applications deployed on the virtual server. This file configures various servlets (such as the `DefaultServlet` and the `JspServlet`) and specifies the MIME types associated with specific extensions. (For more information about MIME mapping elements, see the Java Servlet specification.)

Individual web applications can override the configuration settings inherited from this file with settings contained in their own deployment descriptor file (`web.xml`).

Note: Deployment descriptor files for individual web applications are processed *after* the `default-web.xml` file is processed. This allows configuration settings in the web application's `WEB-INF/web.xml` file to override those contained in the `default-web.xml` file.

5.9.1 Syntax

The `default-web.xml` file is an XML-formatted file. All entries in this file are validated against the XML data type definition (DTD) for the Servlet 2.3 deployment descriptor as follows:

```
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
   "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
```

5.9.2 Context

Each server instance has its own `default-web.xml` file; therefore, there is a one-to-one correspondence between the server instance and this file.

5.9.3 Modifications

The Administration Console and command line interface do not provide a method for managing the `default-web.xml`. As such, all modifications to this file must be made on a particular Administration Node. After this has been performed, the modifications must be pulled back into the configuration and then pushed out to additional Administration Nodes as appropriate.

5.10 The `login.conf` File

The Java Authentication and Authorization Service (JAAS) is a set of APIs that enable services to authenticate and enforce access controls upon users. It implements a Java technology version of the standard Pluggable Authentication Module (PAM) framework, and supports user-based authorization.

The JAAS can be used for two purposes:

- For authentication of users, to reliably and securely determine who is currently executing Java code, regardless of whether the code is running as an application, an applet, a bean, or a servlet.
- For authorization of users to ensure they have the access control rights (permissions) required to do the actions performed.

Note: See the Java Authentication and Authorization Service page on Sun's web site at http://java.sun.com/products/jaas/ for more information.

The login configuration file (`login.conf`) contains the login module definitions used by the JAAS for client authentication. It is referenced in the JVM settings for a particular server instance, as shown in Example 5.12.

Example 5.12 Reference to Login Configuration File in the server.xml

```
<jvm>
  <java-home>/opt/webserver7/jdk</java-home>
  <server-class-path>...</server-class-path>
  <debug>false</debug>
  <debug-jvm-options>-Xdebug -Xrunjdwp:transport=dt_socket,
    server=y,suspend=n,address=7896</debug-jvm-options>
  <jvm-options>-Djava.security.auth.login.config=login.conf
  </jvm-options>
  <jvm-options>-Djava.util.logging.manager=com.sun.webserver.logging
    .ServerLogManager</jvm-options>
  <jvm-options>-Xms128m -Xmx256m</jvm-options>
</jvm>
```

5.10.1 File Structure

The login configuration file specifies the Java class used for each authentication realm. Example 5.13 demonstrates the `login.conf` file for the default Administration Node.

Example 5.13 Default `login.conf` File

```
/*  Copyright 2006 Sun Microsystems, Inc.  All rights reserved.  */
/*  Use is subject to license terms.                             */

fileRealm {
   com.iplanet.ias.security.auth.login.FileLoginModule required;
};

ldapRealm {
   com.iplanet.ias.security.auth.login.LDAPLoginModule required;
};

solarisRealm {
   com.iplanet.ias.security.auth.login.SolarisLoginModule required;
};

nativeRealm {
   com.iplanet.ias.security.auth.login.NativeLoginModule required;
};
```

5.10.2 Syntax

The basic format for the login module definitions contained in the login configuration file is as follows:

```
Application {
  ModuleClass  Flag    ModuleOptions;
  ModuleClass  Flag    ModuleOptions;
  ModuleClass  Flag    ModuleOptions;
};
```

Each entry in the login configuration file is indexed by an application name (*Application*). Each application contains a list of login modules configured for that application. Each login module is specified by its fully qualified class name (*ModuleClass*). Authentication proceeds down the module list in the exact order specified. The *Flag* value controls the overall behavior as authentication proceeds down the stack. Flags can be one of the following: `Required`, `Requisite`, `Sufficient`, or `Optional`.

ModuleOptions is a space-separated list of login module-specific values that are passed directly to the underlying login module. Options are defined by the login module itself and control the behavior within it.

Note: See the `javax.security.auth.login` class configuration page at http://java.sun.com/j2se/1.4.2/docs/api/javax/security/auth/login/Configuration.html for more information.

5.10.3 Context

Each server instance has its own login configuration file; therefore, there is a one-to-one correspondence between the server instance and this file.

5.10.4 Modifications

The Administration Console and command line interface do not provide a method for managing the login configuration file. As such, all modifications to this file must be made on a particular Administration Node. After this has been performed, the modifications must be pulled back into the configuration and then pushed out to additional Administration Nodes as appropriate.

5.11 The keyfile File

Authentication databases are repositories for maintaining user credentials. These credentials can be used to validate a user before granting access to resources on the Web Server. Common repositories include directory servers, databases, and flat files. Web Server 7 ships with a default file-based repository called `keyfile` that contains usernames and hashed passwords that can be used for flat file authentication.

The `keyfile` is empty by default, but entries can easily be added through the Administration Console. Before doing so, however, the server must be configured to use the `keyfile` as an authentication database.

Authentication databases are configured in `server.xml` at either the instance level, for a particular virtual server, or both. Example 5.14 demonstrates the settings for defining a `keyfile` as an authentication database.

Example 5.14 `server.xml` Authentication Database Definition for `keyfile`

```
<auth-db>
  <name>keyFile</name>
  <url>file</url>
  <property>
    <name>keyfile</name>
    <value>/opt/webserver7/https-www.example.com/config/keyFile</value>
  </property>
  <property>
    <name>syntax</name>
    <value>keyfile</value>
  </property>
</auth-db>
```

Table 5.5 provides an overview of the elements found in the authentication database definition.

Table 5.5 Authentication Database Properties

Element	Description
`<auth-db>`	Specifies the beginning and end of an authentication database definition.
`<name>`	The name of the authentication database. This is used to reference the database within the Web Server and must be unique. This value is arbitrary.
`<url>`	A value of `file` indicates that this is a file-based authentication database. Other values for this element might include an appropriate LDAP URL or `pam`.

Element	Description
<property>	Specifies a set of name/value property pairs for this authentication database. The first property listed is called keyfile. This indicates that the database is of type keyfile. The location of the keyfile database can be found at the following: /opt/webserver7/https-www.example.com/config/keyFile. The next property specifies the file's syntax.

5.11.1 File Structure

The keyfile for an Administration Node is empty by default. Each new user creates an entry in the keyfile, as demonstrated in Example 5.15.

Example 5.15 Sample keyfile File

```
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

# List of users for simple file realm. Empty by default.
rodale;{SSHA}0Fata3ioPWgQhd8wXOUWNMKkL7J2FydGVyAA==;hr
relise;{SSHA}5QkLGJmZJ7Z2YaEobLcw5LEk1qdmVkZGVyAA==;it
wclay;{SSHA}h3y4+I6f75k7+5XH2EClfv6ZIixhZG1pbgAAAA==;qa
```

The Administration Server instance uses a keyfile to store its own authentication credentials. Example 5.16 demonstrates the default keyfile for the Administration Node:

Example 5.16 Default keyfile File for the Administration Server Instance

```
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

# List of users for simple file realm. Empty by default.
admin;{SSHA}h3y4+I6f75k7+5XH2EClfv6ZIixhZG1pbgAAAA==;wsadmin
```

5.11.2 Syntax

The basic format for the keyfile is

username;*hashedpassword*;*group*

The maximum length of a line in a file-based authentication database file is 255 characters. If any line exceeds this limit, the server fails to start and an error is logged in the errors log file.

5.11.3 Context

Each server instance has its own `keyfile` file; therefore, there is a one-to-one correspondence between the server instance and this file.

5.11.4 Modifications

You can configure authentication databases with either the Administration Console or the command line interface. This causes changes to the `server.xml` file. If this file is updated as a result of changes made through either of these two interfaces, you must deploy the updated configuration before the changes are reflected on the appropriate Administration Node(s).

The addition of users to the `keyfile` does not require a redeployment of the configuration.

Errors found within the file may prevent the instance from starting.

5.12 Summary

Knowing how to use the Administration Console or CLI tools is not enough to effectively manage your Web Server instances. You should have knowledge of the server's configuration files and how changes made with the Administration Console or CLI tools affect these files. You might never have to directly edit configuration files, but having the knowledge of how to do so will greatly increase your chances of successfully managing your server and debugging problems that arise from misconfigurations.

5.13 Self-Paced Labs

Use the information contained in this chapter to perform the following exercises. These will help validate your understanding of the concepts described in this chapter.

1. View the `server.xml` file for your default server installation. Do this for the files on the node, not beneath the `admin-server` instance.
2. Locate the `cluster` element. What is the name of the host of this instance? (Look for the `local-host` element.)
3. The `instance` element defines a member of the server cluster. What is the host name of the single instance that is part of this server cluster?
4. Locate the `log` element. What is the logging level for this server instance, and where is the log file located on your file system?
5. The `platform` element describes whether the server runs as a 32-bit or 64-bit process. As what platform does this server run?
6. Locate the `user` element (UNIX only). This is the system account the server runs as. As what account will the Web Server process run?

Tip: You can execute a `ps -ef | grep http` command string to verify that the process is running as this user.

7. Locate the `auth-db` element. This element replaces the functionality of `dbswitch.conf` from Web Server 6.1. What is the URL of the ACL authorization database?
8. The `auth-db` element contains `property` elements. All `property` elements must contain the `name` and `value` subelements (for a name/value pair). They can also contain a `description` subelement and for `auth-db` they might contain the encoded `subelement`, indicating that the value is Base 64–encoded. List the name/value pairs for the `property` elements defined in the `auth-db` element.
9. Locate the `acl-file` element. This element contains the name of the file that contains the access control list for this server instance. The value of this element can either be the absolute path to the file on the file system or the name of the file relative to the instance's `config` folder. What is the name of the file that controls access to this instance and where is it located?
10. Locate the `mime-file` element. This element contains the name of the file that contains the MIME types file used for request processing. What is the name of the file that contains the MIME type mappings, and where is it located?
11. Web Server 6.1 listen sockets are referred to as Web Server 7.0 HTTP listeners. You define HTTP listeners by using the `http-listener` element. What subelements are defined for the HTTP listener for this server?

12. Virtual servers are defined with the `virtual-server` element. Many of the elements that have been discussed thus far can be redefined within the body of a `virtual-server` element. This includes the `auth-db`, `acl-file`, and `mime-file` elements. Allowing the virtual server to redefine these elements provides the flexibility of having global settings for the server instance as well as localized settings for particular virtual servers. What subelements are defined for any virtual servers defined for this server?
13. Have these virtual servers redefined any of the global elements (`auth-db`, `acl-file`, or `mime-file`)? If so, which one(s)?

Note: Go to http://www.sunwebserver.com for detailed instructions on how to answer each of these questions.

14. Additional Exercises:

    Make a backup of the `server.xml` file for your default server instance. Download a copy of a corrupted `server.xml` file from the http://www.sunwebserver.com web site and place this file in the `config` directory for your default server instance. Update the downloaded `server.xml` file with information specific to your host and domain. Change all occurrences of the string, `yourhost`, with your actual hostname and all occurrences of the string, `yourdomain`, with your actual domain name. Restart the default Web Server instance and resolve all errors found when attempting to start the server. This step is completed when you have addressed all errors and the server successfully starts and serves up data. After you have completed this exercise, restore the backup copy of your `server.xml` file or use the Administration Console to overwrite the configuration on the node (do not pull the configuration from the node or you will overwrite the data in the `config-store`).

Index

Symbols & Numerics

A

B-C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q-R

S

T

U

V

W

X