APPENDIX A

The Server.xml File

In this appendix, we discuss the configuration of Tomcat containers and connectors in the server.xml configuration. This file is located in the `CATALINA_HOME/conf` directory and can be considered the heart of Tomcat. It allows you to completely configure Tomcat using XML configuration elements. Tomcat loads the configuration from server.xml file at startup, and any changes to this file require server restart. However, you can configure Tomcat is such way that it allows runtime changes to the deployed web applications.

Containers

Tomcat containers are objects that can contribute to the request-response communication between clients (e.g. browsers) and the targeted servlets. There are several types of Tomcat containers, each of which is configured within the server.xml based upon its type. We introduced Tomcat's containers when we discussed Tomcat's architecture in Chapter 1. In this section, we discuss the containers that are configured in the default server.xml file.

The Server Container

Let's first take a look at the top level server container, which is configured using the `<Server>` XML element. It is used as a top-level element for a single Tomcat instance; it is a simple singleton element that represents the entire Tomcat JVM. It may contain one or more Service containers. The server container is defined by the `org.apache.catalina.Server` interface. Table A-1 defines the possible attributes that can be set for the `<Server>` element.

Table A-1. *The Configurable Attributes for <Server> XML Element*

Attribute	Description
`className`	Names the fully qualified Java name of the class that implements the `org.apache.catalina.Server` interface. If no class name is specified, the default implementation is used, which is the `org.apache.catalina.core.StandardServer`.
`address`	This attribute specifies the TCP/IP address on which this server listens for the shutdown command. The default value is `localhost`, which means that the server can be shut down from the same machine where it is installed (i.e., remote shutdown is disabled).
`port`	Names the TCP/IP port number on which the server listens for a shutdown command. If you're running Tomcat as daemon (i.e.,as Windows service), you can disable shutdown port by setting this value to `-1`. This attribute is required.
`shutdown`	Defines the command string that must be received by the server on the configured address and port to shut down Tomcat. This attribute is required.

The default `server.xml` distributed with Tomcat uses following code snippet to configure the server container:

```
<Server port="8005" shutdown="SHUTDOWN">
…
</Server>
```

The `<Server>` element must be the root XML element in server.xml file, and cannot be configured as the child of any element. However, it can be configured as the parent of other XML elements. The allowed nested child XML elements for `<Server>` component are

- `<Service>` element, which we discuss in the next section, and
- `<GlobalNamingResources>` element, used for configuration of global JNDI resources, which are described in Chapter 13.

The Service Container

The next Tomcat container we're going to discuss is the service container. The service container is configured using the `<Service>` XML element in the `server.xml`. The service container holds a collection of one or more connectors, and a single engine container. The service container is configured as a nested XML element within the `<Server>` element. Multiple service containers can be configured within the same server container. The service container element is defined by the `org.apache.catalina.Service` Java interface. Table A-2 describes the `<Service>` element's attributes.

Table A-2. *The Configurable Attributes of the <Service> XML Element*

Attribute	Description
`className`	Names the fully qualified Java name of the class that implements the `org.apache.catalina.Service` interface. If no class name is specified, the implementation will be used, which is the `org.apache.catalina.core.StandardService`.
`name`	Defines the display name of the defined service. The service name must be unique within the enclosing server container. This value is used in all Tomcat log messages. This attribute is required.

The only `<Service>` definition that can be found in the default `server.xml` file is the Catalina service:

```
<Service name="Catalina">
…
</Service>
```

The `<Service>` XML element is configured as a child of the `<Server>` element. As we mentioned before, multiple connectors and a single engine container can be configured for each service container. Allowed nested XML elements within `<Service>` element are:

- the `<Connector>` element, which we describe in the "Connectors" section later in this Appendix, and
- the `<Engine>` element, which we will discuss in the following section.

The Engine Container

Engine container represents the heart of the request-processing mechanism in Tomcat. It is responsible for processing all incoming requests from configured connectors, and returns the processed response back to the connector for dispatching to the calling client. Engine container is configured in the `server.xml` file using the `<Engine>` XML element. Each defined service container can have one and only one engine container, and this single engine receives all requests received by all of the defined connectors. The `<Engine>` element must be nested after the `<Connector>` elements, inside its owning `<Service>` element.

The `<Engine>` element is defined by the `org.apache.catalina.Engine` interface. Table A-3 describes the possible `<Engine>` element attributes.

Table A-3. *The Attributes of the <Engine> Element*

Attribute	Description
`className`	Names the fully qualified Java name of the class that implements the `org.apache.catalina.Engine` interface. If no class name is specified, the implementation is used, which is the `org.apache.catalina.core.StandardEngine`.
`defaultHost`	Names the host name to which all requests are defaulted if not otherwise named. The name specified must reference a host defined by a child `<Host>` element. This attribute is required.
`Name`	Defines the logical name of this engine. The name defined is used in log messages, and must be unique within the server component that this engine belongs to. This attribute is required.
`backgroundProcessorDelay`	Defines the delay time in seconds before the child containers' background processes will be invoked, in case they are executing in the same thread. This background processing thread is responsible for live web application deployment tasks. The default value is `10`.
`jvmRoute`	String identifier appended to every session in load balancing scenarios. It's used to enable the front-end load balancer to send requests with the same session identifier to same Tomcat instances. If configured, the value of jvmRoute must be unique among all servers in a cluster.

The following code snippet contains the `<Engine>` element defined in the default `server.xml` file:

```
<Engine name="Catalina" defaultHost="localhost">
```

The `<Engine>` element is configured as a child of the `<Service>` element, and as a parent to the following elements:

- `<Host>`: Used to configure host container, which is discussed in the next section.
- `<Realm>`: Used to configure Tomcat's security realm, which is covered in Chapter 6.
- `<Valve>`: Used to configure Tomcat's valve, covered in Chapter 8.
- `<Listener>`: Used to configure Tomcat's listener, which is used to react on events occurring internally in the Tomcat engine.

The Host Container

The host container links the server machine where the Tomcat is running to the network name (for example, `www.apress.com` or `174.17.0.204`). The host container is configured using the `<Host>` XML element within the `<Engine>` element. Each `<Host>` can be a parent to one or more web applications, represented by a context container (which is described in the next section).

You must define at least one <Host> for each <Engine> element, and the name of one defined host must match the defaultHost attribute of the parent engine container. The default <Host> element is usually named localhost. The possible attributes for the <Host> element are described in Table A-4.

Table A-4. *The Attributes of the <Host> Element*

Attribute	Description
className	Names the fully qualified Java name of the class that implements the org.apache.catalina.Host interface. If no class name is specified, the implementation is used, which is the org.apache.catalina.core.StandardHost. This attribute is required.
Name	Defines the hostname of this virtual host. This attribute is required and must be unique among the virtual hosts running in this servlet container. This attribute is required.
appBase	Defines the directory for this virtual host. This directory is the pathname of the web applications to be executed in this virtual host. This value can be either an absolute path or a path that is relative to the CATALINA_HOME directory. If this value is not specified, the relative value webapps is used.
xmlBase	The directory location of the XML deployment descriptors that are deployed to this host. If not specified, the path [engine_name]/[host_name] is specified.
createDirs	Specifies whether the directories specified in appBase and xmlBase attributes should be created on Tomcat startup. The default value is true.
autoDeploy	Specifies whether Tomcat should check for new or updated web applications to be deployed to this host. The default is true.
backgroundProcessorDelay	The delay time in seconds between the background process method invocation on this container and any child containers. This background process is responsible for webapp deployment tasks. The default value is -1, which means that this host will rely on the delay setting of its parent engine.
deployIgnore	The regular expression pattern that specifies directories to skip when performing auto deployment, for example *.svn* will skip deployment of the SVN files in case your web applications are deployed directory from an SV version control system.
deployOnStartup	Specifies if the web applications found in appBase and xmlBase directories should be deployed on server startup. The default value is true.

If you're using the Tomcat's default StandardHost implementation, you can use the additional attributes listed in the Table A-5.

Table A-5. *The Additional Attributes for Configuration of StandardHost Implementation*

Attribute	Description
unpackWARs	Determines if WAR files should be unpacked or run directly from the WAR file. If not specified, the default value is true.
workDir	Specifies the work directory for all web applications deployed to this host. If not specified, the default value is CATALINA_HOME/work.
copyXML	If set to true, the context configuration file specified within the web application (/META-INF/context.xml) should be copied to the xmlBase directory. The default value is false.
deployXML	Specifies whether the context configuration supplied with web application (/META-INF/context.xml) should be parsed during web application deployment. Default value is true.
errorReportValveClass	Configures the valve responsible for rendering the Tomcat error pages. By default, the host will use org.apache.catalina.valves.ErrorReportValve.

The default server.xml configuration has one host configured for the Catalina engine:

```
<Host name="localhost"  appBase="webapps"
    unpackWARs="true" autoDeploy="true">
```

This host definition defines a Tomcat virtual host named localhost that can be accessed by opening the following URL (with default Tomcat port 8080):

```
http://localhost:8080/
```

The <Host> element is configured as a child of the <Engine> element, and can have the following nested XML elements:

- <Context>: Context container, described in the next section.
- <Realm>: Used to configure Tomcat's security realm, which is covered in Chapter 6.
- <Valve>: Used to configure Tomcat's valve, covered in Chapter 8.
- <Listener>: Used to configure Tomcat's listener, which is used to react on events occurring internally in the Tomcat engine.

The Context Container

The context container represents a single web application deployed to Tomcat. It is configured using the <Context> XML element, and is the most commonly used container in the server.xml file. Any number of contexts can be defined within a <Host>, but each <Context> definition must have a unique context path, which is defined using the path attribute. There are over 50 different attributes that you can configure for the <Context> element. Table A-6 lists the most important configuration attributes and attributes that are most commonly used for production Tomcat configuration.

Table A-6. *The Main Attributes of the <Context> XML Element*

Attribute	Description
`className`	Names the fully qualified Java name of the class that implements the `org.apache.catalina.Context` interface. If no class name is specified, the implementation is used, which is the `org.apache.catalina.core.StandardContext`.
`cookies`	Determines if you want cookies to be used for a session identifier. The default value is `true`.
`crossContext`	If set to true, allows the `ServletContext.getContext()` method to successfully return the ServletContext for other web applications running in the same host. The default value is `false`, which prevents the access of cross context access.
`docBase`	Defines the directory for the web application associated with this `<Context>`. This is the pathname of a directory that contains the resources for the web application. This attribute is required.
`path`	Defines the context path for this web application. This value must be unique for each `<Context>` defined in a given `<Host>`.
`reloadable`	If set to true, causes Tomcat to check for class changes in the `/WEB-INF/classes/` and `/WEB-INF/lib` directories. If these classes have changed, the application owning these classes is automatically reloaded. This feature should be used only during development. Setting this attribute to true causes severe performance degradation and therefore should be set to false in a production environment.
`wrapperClass`	Defines the Java name of the `org.apache.catalina.Wrapper` implementation class that is used to wrap servlets managed by this context. If not specified, the standard value `org.apache.catalina.core.StandardWrapper` is used.
`sessionCookieName`	Overrides any session cookie name specified by individual web applications. If not specified, and no web application specific value is defined, `JSESSIONID` name will be used. You can configure other session cookie attributes using the `sessionCookiePath` and `sessionCookieDomain` attributes.
`override`	Should be set to true, if you wish to override the configuration settings inherited from parent `<Host>` or `<Engine>` elements. The default value is `true`.
`swallowOutput`	If set to true, all `System.out` and `System.err` output will be redirected to the web application logging engine, instead if being written to `catalina.out` file. The default value is `false`.

If you're using the Tomcat's default `StandardContext` implementation, you can use the additional attributes. Table A-7 describes some of the available attributes.

Table A-7. *The Additional Attributes for Configuration of StandardContext Implementation*

Attribute	Description
`unpackWar`	Determines if the WAR file for this web application should be unpacked or the web application should be executed directly from the WAR file. If not specified, the default value is `true`.
`workDir`	Defines the pathname to a work directory that this Context uses for temporary read and write access. The directory is made visible as a servlet context attribute of type `java.io.File`, with the standard key of `java.servlet.context.tempdir`. If this value is not specified, Tomcat uses the host's work directory `CATALINA_HOME/work` by default.
`useNaming`	Should be set to `true` (the default) if you wish to have Catalina enable JNDI.
`cachingAllowed`	If set to true, Tomcat will cache the static resources it serves (images, css and javascript file, for example).
`antiJARLocking`	Specifies whether Tomcat class loader should take extra care when reading resources from jar file, to avoid jar locking. Specifying this attribute to true will slow deployment of web applications to Tomcat. Default value is `false`. You can configure locking of any other resource file using `antiResourcesLocking` attribute.

■ **Note** You can find the complete reference of available context configuration attributes in the Apache Tomcat online documentation (`http://tomcat.apache.org/tomcat-7.0-doc/config/context.html`).

The `<Context>` element that defines the `/examples` application is included in the following code snippet:

```
<Context path="/examples" docBase="examples" reloadable="true">
```

The context definition defines a Web application under context path `/examples` that has all of its resources stored in the directory examples, relative to the `appBase` directory of the parent host (by default `CATALINA_HOME/webapps/examples`). This context also states that this application is reloaded when its files are updated.

The `<Context>` element is configured as a child of the `<Host>` element, and as a parent to the following elements:

- `<Loader>`: Used for configuration of web application class loader.
- `<Realm>`: Used to configure Tomcat's security realm, which is covered in Chapter 6.
- `<Valve>`: Used to configure Tomcat's valve, covered in Chapter 8.
- `<Listener>`: Used to configure Tomcat's listener, which is used to react on events occurring internally in the Tomcat engine.

- `<Manager>`: Used to configure session manager for the web application deployed in context container; we discuss sessions and session manager in Tomcat in Chapter 5.
- `<Parameter>`: Used to set parameters for `ServletContext` initialization.
- `<Environment>`: Used to configure values that will be available in the web application as environment entries.
- `<Resources>`: Used to configure JNDI resources for the web application deployed in this context; we cover JNDI resources configuration in Chapter 13.
- `<WatchedResource>`: Used to configure resources that will be monitored by auto deployer, and which will trigger web application redeployment if changed or updated.

Now that we covered the configuration of all containers available in Tomcat, let's take a look at the connector components, another key component of the Tomcat architecture.

Connectors

The connector components are responsible for accepting incoming requests in Tomcat, passing the request to the engine container defined for the given connector, accepting the resulting response from the engine container and passing the response to the calling client. The connector elements are configured in Tomcat's `server.xml` file using the `<Connector>` XML element. The `<Connector>` XML element is defined as a nested element within the `<Service>` element, at the same level as the engine container it communicates to.

The `<Connector>` element is defined by the `org.apache.catalina.Connector` interface. There are two main connector types available in Tomcat, based on the protocol they support:

- HTTP connector component, that supports HTTP protocol, which enables Catalina engine to run as a web server and servlet container, handling HTTP request from users via the browser.
- AJP connector component that supports the communication using AJP protocol, used for integrating Tomcat with Apache Web server, as described in Chapter 10.

Table A-8 describes the common attributes used to configure both types of connector component.

Table A-8. *Common Configuration Attributes for the <Connector> XML Element That Can Be Used for Any Type of Connector Component*

Attribute	Description
`Port`	Names the TCP/IP port number on which the connector listens for requests. The default value is `8080`. This is the only required attribute for `<Connector>` element.
`enableLookups`	Determines whether DNS lookups are enabled. The default value for this attribute is `true`. When DNS lookups are enabled, an application calling `request.getRemoteHost()` is returned the domain name of the calling client. Enabling DNS lookups can adversely affect performance. Therefore, this value should most often be set to `false`.
`redirectPort`	Names the TCP/IP port number to which a request should be redirected, if it comes in on a non-SSL port and is subject to a security constraint with a transport guarantee that requires SSL.
`proxyName`	Specifies the server name to use if this instance of Tomcat is behind a firewall. This attribute is optional.
`proxyPort`	Specifies the HTTP port to use if this instance of Tomcat is behind a firewall. Also an optional attribute.
`scheme`	The name of the protocol scheme for the incoming requests. The default value is `http`. For SSL configuration this should be set to value `https`.
`secure`	Specifies the value returned by `request.isSecure()` method call. The default value is `false`, but should be set to true for secure SSL communication.
`allowTrace`	Specifies whether or not the TRACE HTTP method is allowed for incoming requests. Default value is `false`.
`maxPostSize`	The maximum size of the content submitted using POST HTTP method. If set to 0 or negative value, no limit will be set. By default maximum POST content size is set to 2MB.
`parseBodyMethods`	Specifies the list of HTTP methods that will be parsed to fetch request parameters, similarly to POST form submission handling. The advantage of this parameter is that it can be used for REST web applications, where PUT HTTP method is required. The default value is `POST`.
`asyncTimeout`	Timeout of asynchronous requests in milliseconds. The default value is `10,000`.
`uriEncoding`	This attribute is used to configure the character encoding used to encode/decode URI values. By default it's set to `ISO-8859-1.`

Attribute	Description
useIPVHosts	If set to true, Tomcat will use the IP address of the network interface that received the request to determine the host container to redirect the request to. It's set to false by default.

The <Connector> element is configured as a child of the <Service> element, and cannot have any nested child elements configured.

The default server.xml configuration file defines two connectors, one HTTP connector for HTTP traffic and one AJP connector for communication using AJP protocol. Based on the connector type, you can set additional connector configuration attributes. In the next two sections, we will cover the configuration of HTTP and AJP connectors respectively.

The HTTP Connector

The HTTP connector handles all direct HTTP request received by Tomcat. In addition to standard connector attributes (described in Table A-8), you can configure additional attributes when configuring the HTTP connector. Table A-9 describes some of the possible attributes for the HTTP connector configuration. The full list of available attributes can be found in the online Tomcat configuration reference.

Table A-9. *The Additional Attributes for the HTTP Connector Configuration*

Attribute	Description
protocol	Sets the protocol to be used to transport incoming requests. Allowed protocol implementations are: org.apache.coyote.http11.Http11Protocol - blocking Java connector org.apache.coyote.http11.Http11NioProtocol - non-blocking Java connector org.apache.coyote.http11.Http11AprProtocol - the APR/native connector. The default value is HTTP/1.1, which uses Tomcat internal mechanism to pick either blocking Java connector or the APR connector.
address	Used for servers with more than one IP address. It specifies which address is used for listening on the specified port. If this attribute is not specified, this named port number is used on all IP addresses associated with this server.
compression	Specifies whether Tomcat should use compression when sending text-based content, saving bandwidth. Allowed values are off, which disables the compression; on, which enables compression for text-based content only; and force, which forces compression for all content. The default value is off.
compressableMimeType	Comma separated list of mime types that can be compressed if the compression attribute is set. The default value is text/html,text/xml,text/plain.
SSLEnabled	If set to true, the SSL traffic will be enabled on the connector. The default value is false.

Attribute	Description
connectionTimeout	Defines the time, in milliseconds, before a request terminates. The default value is 60,000 milliseconds. To disable connection timeouts, the connectionTimeout value should be set to -1.
maxThreads	Specifies the maximum number of threads that will be created to process requests on this connector. The default value is 200.
acceptCount	Specifies the number of requests that can be queued on the listening port. The default value is 10.

The following code snippet is an example <Connector> defining an HTTP connector:

```
<Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
```

Based on the protocol class specified in the protocol attribute, you can set additional configuration options specific to the protocol used. The list of all configuration options for all protocols is too big to be listed here, but you can find all available protocol configuration options on the Tomcat's online resources for the HTTP connector: http://tomcat.apache.org/tomcat-7.0-doc/config/http.html.

The AJP Connector

The AJP connector handles requests that have been forwarded by a web server that Tomcat integrates with, like the Apache Web server that sits in front of Tomcat. The AJP connector can be configured using the set of attributes described in Table A-10.

Table A-10. *The Additional Attributes for the AJP Connector Configuration*

Attribute	Description
Protocol	Sets the protocol to be used to transport incoming requests. Allowed protocol implementations are: org.apache.coyote.http11.AjpProtocol - blocking Java connector org.apache.coyote.http11.AjpNioProtocol - non-blocking Java connector org.apache.coyote.http11.AjpAprProtocol - the APR/native connector The default value is AJP/1.3, which uses Tomcat internal mechanism to pick either blocking Java connector or the APR connector.
Address	Used for servers with more than one IP address. It specifies which address is used for listening on the specified port. If this attribute is not specified, this named port number is used on all IP addresses associated with this server.
packetSize	Specifies the AJP packet size in bytes. The maximum value you can set this attribute to is 65536, and the minimum value is 8192. The default value is 8192.

Attribute	Description
requiredSecret	If specified, the AJP requests must contain the value of this attribute in order to be processed.
connectionTimeout	Defines the time, in milliseconds, before a request terminates. The default value is 60,000 milliseconds. To disable connection timeouts, the connectionTimeout value should be set to -1.
maxThreads	Specifies the maximum number of threads that will be created to process requests on this connector. The default value is 200.
acceptCount	Specifies the number of requests that can be queued on the listening port. The default value is 10.

The following code snippet illustrates the default AJP connector defined in the Tomcat's server.xml file:

```
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

Based on the protocol class specified in protocol attribute, you can set additional configuration options specific to the protocol used. The list of all configuration options for all protocols is too big to include here, but you can find all available protocol configuration options on the Tomcat's online resources for the AJP connector: http://tomcat.apache.org/tomcat-7.0-doc/config/ajp.html.

Summary

In this appendix, we explained the configuration of the Tomcat containers and connectors in the CATALINA_HOME/conf/server.xml file. We also demonstrated the default configuration of all components in the server.xml file distributed out of the box with Tomcat. We included the most common options that should get you up and running quickly when configuring Tomcat. However, due to the richness of Tomcat's configuration options, we couldn't list all available options in this appendix. For the full details about all configuration options for Tomcat components, take a look at Tomcat's online documentation (http://tomcat.apache.org/tomcat-7.0-doc/index.html).

APPENDIX B

The Web.xml File

In this appendix, we discuss the web application deployment descriptor, or `web.xml` file. The `web.xml` file is an XML file, defined by the servlet specification, with the purpose of acting as a configuration file for a web application. This file and its elements are completely independent of the Tomcat container. In this appendix, we will explain the Servlet 3.0 specific annotation based configuration that can be used instead of some `web.xml` configuration elements.

The Basic web.xml Configuration

The minimum requirements for web deployment descriptor is to have opening and closing `<webapp>` elements including Servlet API namespaces and schema definitions. Listing B-1 illustrates the minimal `web.xml` configuration file.

Listing B-1. *Contents of the Minimal web.xml File*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">

</web-app>
```

Every web application's web.xml configuration file will have at least few lines from listing B-1. The first line elements define the XML version and the document type definition (DTD) for the web.xml file, and you can see this element in most XML files, regardless of their purpose. The first element that is important to us is the `<web-app>` element, because this element is the container for all web application components. We will be examining the components that are the children of this element, but we won't examine every element of the deployment descriptor, which would be beyond the scope of this text. We'll describe only those elements that are most commonly used.

■ **Note** As from Java Servlet specification 2.4, the order of the children components of `<webapp>` element in `web.xml` file does not matter. Because Apache Tomcat 7 implements latest version of Servlet API (3.0), it applies to any web.xml configuration in latest Tomcat version. However, if you're using an earlier Servlet API version (2.4 or before), all of the definitions that we add to the `web.xml` file must be added in the specific order. For the correct order of elements in earlier Servlet API version, please consult the relevant Java Servlet specification, which you can find on Oracle Java website: `http://jcp.org/aboutJava/communityprocess/pfd/jsr315/index.html`.

Adding a Servlet Definition

The first Web component definition that we are going to add is a servlet. To do this, we use the `<servlet>` element and its sub-elements. The following code snippet contains a sample servlet definition:

```
<!-- Define a servlet -->
<servlet>
        <servlet-name>myServlet</servlet-name>
        <servlet-class>com.apress.MyServlet</servlet-class>
        <init-param>
                <param-name>paramName</param-name>
                <param-value>paramValue</param-value>
        </init-param>
        <load-on-startup>0</load-on-startup>
</servlet>
```

Descriptions of the `<servlet>` sub-elements can be found in Table B-1.

Table B-1. *The <servlet> Sub-Elements*

Sub-Element	Description
`<servlet-name>`	The string that is used to uniquely identify the servlet. It is used in the `<servlet-mapping>` sub-element to identify the servlet to be executed, when a defined URL pattern is requested, if there is a `<servlet-mapping>` sub-element.
`<servlet-class>`	Names the fully qualified servlet class to be executed.
`<init-param>`	Defines a name/value pair as an initialization parameter of the servlet. There can be any number of this optional sub-element. It also has two sub-elements of its own that define the name and value of the initialization parameter: • `<param-name>` element: Used to define the parameter name. • `<param-value>` element: Used to configure the value of the parameter passed to the servlet.

Sub-Element	Description
<load-on-startup>	Indicates that this servlet should be loaded when the web application starts. If the value of this element is a negative integer, or if the element is not present, the container is open to load the servlet whenever it chooses. If the value is a positive integer or 0, the container guarantees that servlets with lower integer values are loaded before servlets with higher integer values.
<async-supported>	Specifies whether the servlet supports asynchronous processing, new feature of Servlet API 3.0. Allowed values are true and false.

After examining the sub-element definitions, you can see that this servlet element defines a servlet named `myServlet` that is implemented in a class named `com.apress.MyServlet`. It has single initialization parameter named `paramName`, with a value `paramValue`. It also is one of the first preloaded servlets when the Web application starts.

Adding a Servlet Mapping

The next web component that we are going to add is a servlet mapping. A servlet mapping defines a mapping between a servlet and a URL pattern. To do this, we use the <servlet-mapping> element and its sub-elements. The following code snippet contains a sample servlet mapping definition:

```
<!-- The mapping for the Controller servlet -->
<servlet-mapping>
        <servlet-name>myServlet</servlet-name>
        <url-pattern>*.ap</url-pattern>
</servlet-mapping>
```

Descriptions of the <servlet-mapping> sub-elements can be found in Table B-2.

Table B-2. *The <servlet-mapping> Sub-Elements*

Sub-Element	Description
<servlet-name>	The string that is used to uniquely identify the servlet that is executed when the following defined <url-pattern> is requested.
<url-pattern>	Defines the URL pattern that must be matched to execute the servlet named in the <servlet-name> element.

This previous servlet mapping states that the servlet named `myServlet` is executed whenever a resource in this Web application, ending with .ap extension, is requested.

Configuring a Servlet Using Annotations

Servlet 3.0 introduced additional way to configure web application components, using annotations on the web application classes. In order to configure your servlet using annotations, you have to add the following XML element to `web.xml` file:

```
<metadata-complete>false</metadata-complete>
```

This element will tell servlet container that the configuration in web deployment descriptor isn't complete, and that it should scan all web application classes, looking for Servlet 3.0 annotations. In addition, you have to remove all `<servlet>` and `<servlet-mapping>` XML elements from your web.xml file, which are going to be replaced by annotation configured servlet.

To configure servlet and servlet mappings you will need only one annotation: `@WebServlet`. The following code snippet illustrates the configuration matching the XML configuration we used in earlier examples:

```
package com.apress;

import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;

@WebServlet(
        name = "myServlet",
        urlPatterns = "*.ap",
        loadOnStartup = 1,
        initParams = {
            @WebInitParam(name = "paramName", value = "paramValue")
        },
        asyncSupported = false
)
public class MyServlet extends HttpServlet{
    //standard servlet implementation goes here
}
```

The servlet implementation class extends `HttpServlet` class, just like any typical servlet implementation. The implantation details include overriding `doGet(..)` or `doPost()` methods (or both), for handling GET and POST HTTP requests. The only addition, are the annotations on the class level, replacing the servlet configuration in `web.xml` file.

Adding a Servlet Filter

Servlet filters provide the necessary functionality to preprocess `ServletRequest` and `ServletResponse` objects as part of web application's lifecycle. To add a new servlet filter to a web application, you must add a `<filter>` element and a `<filter-mapping>` element to the `web.xml` file. The following code snippet contains a sample filter entry:

```
<!-- Define a Filter -->
<filter>
        <filter-name>SampleFilter</filter-name>
        <filter-class>com.apress.SampleFilter</filter-class>
        <init-param>
                <param-name>email</param-name>
                <param-value>admin@apress.com</param-value>
        </init-param>
</filter>
```

This filter definition defines a filter named `SampleFilter` that is implemented in a class named `com.apress.SampleFilter`. Descriptions of the `<filter>` element's sub-elements can be found in Table B-3.

Table B-3. *The Filter Configurable XML Sub-Elements*

Sub-Element	Description
`<filter-name>`	The string that is used to uniquely identify the servlet filter. It is used in the `<filter-mapping>` sub-element to identify the filter to be executed, when a defined URL pattern is requested.
`<filter-class>`	Names the fully qualified filter class to be executed when the string defined in the `<filter-name>` sub-element is referenced in the `<filter-mapping>` element.
`<init-param>`	Specifies the initialization parameters passed to filter implementation at creation time.

Configuring Filter Mapping

To deploy a filter, you must add a `<filter-mapping>` element. The `<filter-mapping>` describes the servlet filter to execute and the URL pattern that must be requested to execute the filter. The following code snippet contains a `<filter-mapping>` for the previous filter:

```
<!-- Define a  Mapping for the previous Filter -->
<filter-mapping>
    <filter-name>SampleFilter</filter-name>
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

Descriptions of the sub-elements of the `<filter-mapping>` are described in Table B-4.

Table B-4. *The <filter-mapping> Sub-Elements*

Sub-Elements	Description
`<filter-name>`	The string that names the servlet filter to execute when the defined URL pattern is requested.
`<url-pattern>`	Defines the URL pattern that must be requested to execute the named servlet filter.
`<servlet-name>`	If defined, this filter will preprocess all requests mapped to the specified servlet. The value of this element must reference the servlet name as defined in `<servlet-name>` element.
`<dispatcher>`	Specifies whether the filter should preprocess requests originating from within the web application itself (such as `RequestDispacter.forward()` and `RequestDispatecher.include()` requests). The allowed values are: • REQUEST: Filter will preprocess requests originating from the client only. • INCLUDE: Filter will preprocess internal include requests. • FORWARD: Filter will preprocess internal forward requests. • ERROR: Filter will preprocess requests to error handling components. Multiple values are allowed, and the default value is REQUEST.

■ **Note** Make sure that the <filter-name> sub-element in both the <filter> and <filter-mapping> elements match. This is the link between these two elements.

The result of these combined elements is a filter named `SampleFilter` that is executed whenever a JSP resource is requested in the application that owns this deployment descriptor.

Configuring Servlet Filter Using Annotations

Similarly to servlet configuration, you can configure your filters using annotations on the class that implements `Filter` interface. All filter settings are configured using `@WebFilter` annotations available as part of Servlet 3.0 API. Following snippet illustrates the sample annotation-based filter configuration.

```
package com.apress;

import javax.servlet.DispatcherType;
import javax.servlet.annotation.WebFilter;
import java.util.logging.Filter;
```

```
@WebFilter(
        filterName="SampleFilter",
        urlPatterns={"*.jsp", "*.do"},
        dispatcherTypes = {DispatcherType.REQUEST},
        servletNames = {"myServlet"}

)
public class SampleFilter implements Filter{
    //standard filter implementation goes here
}
```

For more details is servlet filters implementation, configuration and deployment in Tomcat, please refer to Chapter 8.

Configuring ServletContext Parameters

You can specify one or more parameters of the `ServletContext` in the `web.xml` file. Each of the parameters specified will be available to all servlet components defined in the same context (which includes every servlet, filter and any other component defined in the same `web.xml` file). You specify `ServletContext` parameters using `<context-param>` XML element, like in the following code snippet:

```
<context-param>
         <param-name>adminEmailAddress</param-name>
         <param-value>admin@apress.com</param-value>
</context-param>
```

The value of every specified parameter can be loaded in the servlet code, referenced by the parameter name:

```
String value = getServletContext().getInitParameter("adminEmailAddress ");
```

Configuring the Session

The next web component that we are going to add determines the life of each `HttpSession` in the current web application. The following code snippet contains a sample session configuration:

```
<!-- Set the default session timeout (in minutes) -->
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```

The `<session-config>` element contains only one sub-element, `<session-timeout>`, which defines the length of time that an `HttpSession` object can remain inactive before the container marks it as invalid. The value must be an integer measured in minutes.

To learn more about Tomcat's session configuration and handling, please read Chapter 5.

Adding a Welcome File List

We are now going to add a default list of files that will be loaded automatically when a web application is referenced without a filename. An example `<welcome-file-list>` is contained in the following code snippet:

```
<!-- Establish the default list of welcome files -->
<welcome-file-list>
    <welcome-file>login.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

The <welcome-file-list> contains an ordered list of <welcome-files> sub-elements that contain the filenames to present to the user. The files are served in order of appearance and existence. In this example, the Web application first tries to serve up the login.jsp file. If this file does not exist in the web application, the application tries to serve up the file index.html. If none of the files in the welcome list exists, an HTTP 404 Not Found error is returned.

Configuring Error Handlers

Tomcat, like any other servlet container, comes with the default error pages presented to the user when something goes wrong in the web application. You can customize these error pages in web.xml file using <error-page> configuration element. The following code snippet shows an example of error handler configuration:

```
<error-page>
    <error-code>404</error-code>
    <location>/not-found.jsp</location>
</error-page>
<error-page>
    <error-code>500</error-code>
    <location>/unexpected-error.html</location>
</error-page>
```

Each error handler is mapped to the specific HTTP status code, so that whenever servlet container returns the specified status code, the custom error page will be displayed to the user.

Configuring Mime Types

You can configure additional mime types handled by your web application using <mime-mapping> element. If your application uses non-standard URL extensions that do not match default mime mappings (for example serving pdf using .portable URL extension), the response mime type will be set to application-octet/stream by default. You can set the mime type for any custom URL extension in web.xml file, so that the content displays correctly in user browser (in the pdf example, the browser's pdf plug-in will open the content in the pdf-reader application). The following code snippet illustrates mime mapping configuration:

```
<mime-mapping>
        <extension>portable</extension>
        <mime-type>application/pdf</mime-type>
<mime-mapping>
```

Configuring Web Application Security

In this section we are going to take a look at the web.xml configuration elements used for web application security settings.

Adding a Security Constraint

We are going to add a security constraint to protect a resource in our web application. The following code snippet contains a sample <security-constraint> element:

```
<!-- Define a  Security Constraint on this Application -->
<security-constraint>
    <web-resource-collection>
    <web-resource-name>Apress Application</web-resource-name>
    <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
    <role-name>apressuser</role-name>
    </auth-constraint>
</security-constraint>
```

Descriptions of the <security-constraint> sub-elements can be found in Table B-5.

Table B-5. *The <security-constraint> Sub-Elements*

Sub-Element	Description
<web-resource-collection>	Used to identify a subset of the resources and HTTP methods on those resources within a web application to which a security constraint applies. The <web-resource-collection> sub-element contains two sub-elements of its own that are defined in Table B-6.
<auth-constraint>	Defines the user roles that should be permitted access to this resource collection. It contains a single sub-element, <role-name>, which defines the actual role name that has access to the defined constraint. If this value is set to an *, all roles have access to the constraint.

The <web-resource-collection> element specifies the configuration of secured web resources. Table B-6 lists the XML sub-elements used to configure <web-resource-collection>.

Table B-6. *The <web-resource-collection> Sub-Elements*

Sub-Element	Description
`<web-resource-name>`	Defines the name of this Web resource collection.
`<url-pattern>`	Defines the URL pattern that will be protected by the resource.

This security constraint protects the entire Apress Application web application, allowing only users with a defined `<role-name>` of `apressuser`.

Adding a Login Config

To make a security constraint effective, you must define a method in which a user can log in, so that his role can be checked. To do this, you must add a login configuration component to the Web application. An example of this is contained in the following code snippet:

```
<!-- Define the Login Configuration for this Application -->
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Apress Application</realm-name>
</login-config>
```

Descriptions of the <login-config> sub-elements can be found in Table B-7.

Table B-7. *The <login-config> Sub-Elements*

Sub-Element	Description
`<auth-method>`	Used to configure the method by which the user is authenticated for this Web application. The possible values are BASIC, DIGEST, FORM, and CLIENT-CERT. If this value is set to FORM, the <form-login-config> sub-element must be defined.
`<form-login-config>`	Specifies the login and error page that should be used in FORM-based authentication. The sub-elements of the <form-login-config> are defined in Table B-8.
`<realm-name>`	Defines the name of the resource that this login configuration applies. This value must match a <web-resource-name> that was defined in a security constraint.

If you specify FORM based authentication in the <login-config> section, you have to specify the form login page and login error page that will be displayed to the user. Table B-8 describes the XML elements used to configure these details.

Table B-8. *The <form-login-config> Sub-Elements*

Sub-Element	Description
`<form-login-page>`	Defines the location and name of the page that will serve as the login page when using FORM-based authentication.
`<form-error-page>`	Defines the location and name of the page that will serve as the error page when a FORM-based login fails.

The results of this `<login-config>` sub-element definition states that the `<web-resource-collection>`, with a Web resource named Apress Application, uses a login method of BASIC authentication.

Tomcat security and login configuration are covered in more detail in Chapters 6 and 7.

Summary

In this appendix we presented the most commonly used settings used to configure servlets and its components in the web.xml file. We also illustrated the annotation-based servlet configuration introduced as part of Servlet API 3.0. The described configuration options will enable you to configure servlets for a typical web application. However, not all Servlet API configuration options could fit in this Appendix. For complete reference of configurable options take a look at Java Servlet specification (`http://jcp.org/aboutJava/communityprocess/pfd/jsr315/index.html`).

Index

E

F

G

H

I

J, K

L

M, N, O

P, Q

R

S

T

U

V

W

X, Y, Z