The Complete Reference

Java™ 2

Chapter 27

Servlets

This chapter presents an overview of *servlets*. Servlets are small programs that execute on the server side of a Web connection. Just as applets dynamically extend the functionality of a Web browser, servlets dynamically extend the functionality of a Web server. The topic of servlets is quite large, and it is beyond the scope of this chapter to cover it all. Instead, we will focus on the core concepts, interfaces, and classes, and develop several examples.

Background

In order to understand the advantages of servlets, you must have a basic understanding of how Web browsers and servers cooperate to provide content to a user. Consider a request for a static Web page. A user enters a Uniform Resource Locator (URL) into a browser. The browser generates an HTTP request to the appropriate Web server. The Web server maps this request to a specific file. That file is returned in an HTTP response to the browser. The HTTP header in the response indicates the type of the content. The Multipurpose Internet Mail Extensions (MIME) are used for this purpose. For example, ordinary ASCII text has a MIME type of text/plain. The Hypertext Markup Language (HTML) source code of a Web page has a MIME type of text/html.

Now consider dynamic content. Assume that an online store uses a database to store information about its business. This would include items for sale, prices, availability, orders, and so forth. It wishes to make this information accessible to customers via Web pages. The contents of those Web pages must be dynamically generated in order to reflect the latest information in the database.

In the early days of the Web, a server could dynamically construct a page by creating a separate process to handle each client request. The process would open connections to one or more databases in order to obtain the necessary information. It communicated with the Web server via an interface known as the Common Gateway Interface (CGI). CGI allowed the separate process to read data from the HTTP request and write data to the HTTP response. A variety of different languages were used to build CGI programs. These included C, C++, and Perl.

However, CGI suffered serious performance problems. It was expensive in terms of processor and memory resources to create a separate process for each client request. It was also expensive to open and close database connections for each client request. In addition, the CGI programs were not platform-independent. Therefore, other techniques were introduced. Among these are servlets.

Servlets offer several advantages in comparison with CGI. First, performance is significantly better. Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request. Second, servlets are platform-independent because they are written in Java. A number of Web servers from different vendors offer the Servlet API. Programs developed for this API can be moved to any of these environments without recompilation. Third, the Java security manager on the server enforces a set of restrictions to protect the resources on a server

machine. You will see that some servlets are trusted and others are untrusted. Finally, the full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

The Life Cycle of a Servlet

Three methods are central to the life cycle of a servlet. These are **init( )**, **service( )**, and **destroy( )**. They are implemented by every servlet and are invoked at specific times by the server. Let us consider a typical user scenario to understand when these methods are called.

First, assume that a user enters a Uniform Resource Locator (URL) to a Web browser. The browser then generates an HTTP request for this URL. This request is then sent to the appropriate server.

Second, this HTTP request is received by the Web server. The server maps this request to a particular servlet. The servlet is dynamically retrieved and loaded into the address space of the server.

Third, the server invokes the **init( )** method of the servlet. This method is invoked only when the servlet is first loaded into memory. It is possible to pass initialization parameters to the servlet so it may configure itself.

Fourth, the server invokes the **service( )** method of the servlet. This method is called to process the HTTP request. You will see that it is possible for the servlet to read data that has been provided in the HTTP request. It may also formulate an HTTP response for the client.

The servlet remains in the server's address space and is available to process any other HTTP requests received from clients. The **service( )** method is called for each HTTP request.

Finally, the server may decide to unload the servlet from its memory. The algorithms by which this determination is made are specific to each server. The server calls the **destroy( )** method to relinquish any resources such as file handles that are allocated for the servlet. Important data may be saved to a persistent store. The memory allocated for the servlet and its objects can then be garbage collected.

Using Tomcat For Servlet Development

To create servlets, you will need to download a servlet development environment. The one currently recommended by Sun is Tomcat 4.0, which supports the latest servlet specification, which is 2.3. (The complete servlet specification is available for download through **java.sun.com**.) Tomcat replaces the old JSDK (Java Servlet Development Kit) that was previously provided by Sun. Tomcat is an open-source product maintained by the Jakarta Project of the Apache Software Foundation. It contains the class libraries, documentation, and run-time support that you will need to create and test servlets.

You can download Tomcat through the Sun Microsystems Web site at **java.sun.com**. The current version is 4.0. Follow the instructions to install this toolkit on your machine. The examples in this chapter assume a Windows environment. The default location for Tomcat 4.0 is

```
C:\Program Files\Apache Tomcat 4.0\
```

This is the location assumed by the examples in this book. If you load Tomcat in a different location, you will need to make appropriate changes to the examples. You may need to set the environmental variable **JAVA_HOME** to the top-level directory in which the Java Software Development Kit is installed. For Java 2, version 1.4, the default directory is **C:\j2sdk1.4.0**, but you will need to confirm this for your environment.

To start Tomcat, select Start Tomcat in the Start | Programs menu, or run **startup.bat** from the

```
C:\Program Files\Apache Tomcat 4.0\bin\
```

directory. When you are done testing servlets, you can stop Tomcat by selecting Stop Tomcat in the Start | Programs menu, or run **shutdown.bat**.

The directory

```
C:\Program Files\Apache Tomcat 4.0\common\lib\
```

contains **servlet.jar**. This JAR file contains the classes and interfaces that are needed to build servlets. To make this file accessible, update your **CLASSPATH** environment variable so that it includes

```
C:\Program Files\Apache Tomcat 4.0\common\lib\servlet.jar.
```

Alternatively, you can specify this class file when you compile the servlets. For example, the following command compiles the first servlet example:

```
javac HelloServlet.java -classpath "C:\Program Files\Apache Tomcat
4.0\common\lib\servlet.jar"
```

Once you have compiled a servlet, you must copy the class file into the directory that Tomcat uses for example servlet class files. For the purposes of this chapter, you must put the servlet files into the following directory:

```
C:\Program Files\Apache Tomcat 4.0\webapps\examples\WEB-INF\classes
```

A Simple Servlet

To become familiar with the key servlet concepts, we will begin by building and testing a simple servlet. The basic steps are the following:

1. Create and compile the servlet source code.
2. Start Tomcat.
3. Start a Web browser and request the servlet.

Let us examine each of these steps in detail.

Create and Compile the Servlet Source Code

To begin, create a file named **HelloServlet.java** that contains the following program:

```
import java.io.*;
import javax.servlet.*;

public class HelloServlet extends GenericServlet {

  public void service(ServletRequest request,
    ServletResponse response)
  throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<B>Hello!");
    pw.close();
  }
}
```

Let's look closely at this program. First, note that it imports the **javax.servlet** package. This package contains the classes and interfaces required to build servlets. You will learn more about these later in this chapter. Next, the program defines **HelloServlet** as a subclass of **GenericServlet**. The **GenericServlet** class provides functionality that makes it easy to handle requests and responses.

Inside **HelloServet**, the **service( )** method (which is inherited from **GenericServlet**) is overridden. This method handles requests from a client. Notice that the first argument is a **ServletRequest** object. This enables the servlet to read data that is provided via the client request. The second argument is a **ServletResponse** object. This enables the servlet to formulate a response for the client.

The call to **setContentType( )** establishes the MIME type of the HTTP response. In this program, the MIME type is text/html. This indicates that the browser should interpret the content as HTML source code.

Next, the **getWriter( )** method obtains a **PrintWriter**. Anything written to this stream is sent to the client as part of the HTTP response. Then **println( )** is used to write some simple HTML source code as the HTTP response.

Compile this source code and place the **HelloServlet.class** file in the Tomcat class files directory as described in the previous section.

Start Tomcat

As explained, to start Tomcat, select Start Tomcat in the Start | Programs menu, or run **startup.bat** from the

```
C:\Program Files\Apache Tomcat 4.0\bin\
```

directory.

Start a Web Browser and Request the Servlet

Start a Web browser and enter the URL shown here:

```
http://localhost:8080/examples/servlet/HelloServlet
```

Alternatively, you may enter the URL shown here:

```
http://127.0.0.1:8080/examples/servlet/HelloServlet
```

This can be done because 127.0.0.1 is defined as the IP address of the local machine.

You will observe the output of the servlet in the browser display area. It will contain the string **Hello!** in bold type.

The Servlet API

Two packages contain the classes and interfaces that are required to build servlets. These are **javax.servlet** and **javax.servlet.http**. They constitute the Servlet API. Keep in mind that these packages are not part of the Java core packages. Instead, they are standard extensions. Therefore, they are not included in the Java Software Development Kit. You must download Tomcat to obtain their functionality.

The Servlet API has been in a process of ongoing development and enhancement. The current servlet specification is version is 2.3 and that is the one used in this book. However, because changes happen fast in the world of Java, you will want to check for any additions or alterations. This chapter discusses the core of the Servlet API, which will be available to most readers.

The Servlet API is supported by most Web servers, such as those from Sun, Microsoft, and others. Check at **http://java.sun.com** for the latest information.

The javax.servlet Package

The **javax.servlet** package contains a number of interfaces and classes that establish the framework in which servlets operate. The following table summarizes the core interfaces that are provided in this package. The most significant of these is **Servlet**. All servlets must implement this interface or extend a class that implements the interface. The **ServletRequest** and **ServletResponse** interfaces are also very important.

Interface	Description
Servlet	Declares life cycle methods for a servlet.
ServletConfig	Allows servlets to get initialization parameters.
ServletContext	Enables servlets to log events and access information about their environment.
ServletRequest	Used to read data from a client request.
ServletResponse	Used to write data to a client response.
SingleThreadModel	Indicates that the servlet is thread safe.

The following table summarizes the core classes that are provided in the **javax.servlet** package.

Class	Description
GenericServlet	Implements the **Servlet** and **ServletConfig** interfaces.
ServletInputStream	Provides an input stream for reading requests from a client.
ServletOutputStream	Provides an output stream for writing responses to a client.
ServletException	Indicates a servlet error occurred.
UnavailableException	Indicates a servlet is unavailable.

Let us examine these interfaces and classes in more detail.

The Servlet Interface

All servlets must implement the **Servlet** interface. It declares the **init( )**, **service( )**, and **destroy( )** methods that are called by the server during the life cycle of a servlet. A method is also provided that allows a servlet to obtain any initialization parameters. The methods defined by **Servlet** are shown in Table 27-1.

Method	Description
void destroy( )	Called when the servlet is unloaded.
ServletConfig getServletConfig( )	Returns a **ServletConfig** object that contains any initialization parameters.
String getServletInfo( )	Returns a string describing the servlet.
void init(ServletConfig *sc*) throws ServletException	Called when the servlet is initialized. Initialization parameters for the servlet can be obtained from *sc*. An **UnavailableException** should be thrown if the servlet cannot be initialized.
void service(ServletRequest *req*, ServletResponse *res*) throws ServletException, IOException	Called to process a request from a client. The request from the client can be read from *req*. The response to the client can be written to *res*. An exception is generated if a servlet or IO problem occurs.

Table 27-1. *The Methods Defined by* Servlet

The **init( )**, **service( )**, and **destroy( )** methods are the life cycle methods of the servlet. These are invoked by the server. The **getServletConfig( )** method is called by the servlet to obtain initialization parameters. A servlet developer overrides the **getServletInfo( )** method to provide a string with useful information (for example, author, version, date, copyright). This method is also invoked by the server.

The ServletConfig Interface

The **ServletConfig** interface is implemented by the server. It allows a servlet to obtain configuration data when it is loaded. The methods declared by this interface are summarized here:

Method	Description
ServletContext getServletContext( )	Returns the context for this servlet.
String getInitParameter(String *param*)	Returns the value of the initialization parameter named *param*.
Enumeration getInitParameterNames( )	Returns an enumeration of all initialization parameter names.
String getServletName( )	Returns the name of the invoking servlet.

The ServletContext Interface

The **ServletContext** interface is implemented by the server. It enables servlets to obtain information about their environment. Several of its methods are summarized in Table 27-2.

The ServletRequest Interface

The **ServletRequest** interface is implemented by the server. It enables a servlet to obtain information about a client request. Several of its methods are summarized in Table 27-3.

The ServletResponse Interface

The **ServletResponse** interface is implemented by the server. It enables a servlet to formulate a response for a client. Several of its methods are summarized in Table 27-4.

The SingleThreadModel Interface

This interface is used to indicate that only a single thread will execute the **service( )** method of a servlet at a given time. It defines no constants and declares no methods. If a servlet implements this interface, the server has two options. First, it can create several instances of the servlet. When a client request arrives, it is sent to an available instance of the servlet. Second, it can synchronize access to the servlet.

Method	Description
Object getAttribute(String *attr*)	Returns the value of the server attribute named *attr*.
String getMimeType(String *file*)	Returns the MIME type of *file*.
String getRealPath(String *vpath*)	Returns the real path that corresponds to the virtual path *vpath*.
String getServerInfo( )	Returns information about the server.
void log(String *s*)	Writes *s* to the servlet log.
void log(String *s*, Throwable *e*)	Write *s* and the stack trace for *e* to the servlet log.
void setAttribute(String *attr*, Object *val*)	Sets the attribute specified by *attr* to the value passed in *val*.

Table 27-2. *Various Methods Defined by* ServletContext

Method	Description
Object getAttribute(String *attr*)	Returns the value of the attribute named *attr*.
String getCharacterEncoding( )	Returns the character encoding of the request.
int getContentLength( )	Returns the size of the request. The value –1 is returned if the size is unavailable.
String getContentType( )	Returns the type of the request. A **null** value is returned if the type cannot be determined.
ServletInputStream getInputStream( ) throws IOException	Returns a **ServletInputStream** that can be used to read binary data from the request. An **IllegalStateException** is thrown if **getReader( )** has already been invoked for this request.
String getParameter(String *pname*)	Returns the value of the parameter named *pname*.
Enumeration getParameterNames( )	Returns an enumeration of the parameter names for this request.
String[ ] getParameterValues(String *name*)	Returns an array containing values associated with the parameter specified by *name*.
String getProtocol( )	Returns a description of the protocol.
BufferedReader getReader( ) throws IOException	Returns a buffered reader that can be used to read text from the request. An **IllegalStateException** is thrown if **getInputStream( )** has already been invoked for this request.

Table 27-3. *Various Methods Defined by* ServletRequest

```
        // Get enumeration of parameter names.
        Enumeration e = request.getParameterNames();

        // Display parameter names and values.
        while(e.hasMoreElements()) {
          String pname = (String)e.nextElement();
          pw.print(pname + " = ");
          String pvalue = request.getParameter(pname);
          pw.println(pvalue);
        }
        pw.close();
    }
}
```

Compile the servlet and perform these steps to test this example:

1. Start Tomcat (if it is not already running).
2. Display the Web page in a browser.
3. Enter an employee name and phone number in the text fields.
4. Submit the Web page.

After following these steps, the browser will display a response that is dynamically generated by the servlet.

The javax.servlet.http Package

The **javax.servlet.http** package contains a number of interfaces and classes that are commonly used by servlet developers. You will see that its functionality makes it easy to build servlets that work with HTTP requests and responses.

The following table summarizes the core interfaces that are provided in this package:

Interface	Description
HttpServletRequest	Enables servlets to read data from an HTTP request.
HttpServletResponse	Enables servlets to write data to an HTTP response.
HttpSession	Allows session data to be read and written.
HttpSessionBindingListener	Informs an object that it is bound to or unbound from a session.

The following table summarizes the core classes that are provided in this package. The most important of these is **HttpServlet**. Servlet developers typically extend this class in order to process HTTP requests.

Class	Description
Cookie	Allows state information to be stored on a client machine.
HttpServlet	Provides methods to handle HTTP requests and responses.
HttpSessionEvent	Encapsulates a session-changed event.
HttpSessionBindingEvent	Indicates when a listener is bound to or unbound from a session value, or that a session attribute changed.

The HttpServletRequest Interface

The **HttpServletRequest** interface is implemented by the server. It enables a servlet to obtain information about a client request. Several of its methods are shown in Table 27-5.

Method	Description
String getAuthType( )	Returns authentication scheme.
Cookie[ ] getCookies( )	Returns an array of the cookies in this request.
long getDateHeader(String *field*)	Returns the value of the date header field named *field*.
String getHeader(String *field*)	Returns the value of the header field named *field*.
Enumeration getHeaderNames( )	Returns an enumeration of the header names.
int getIntHeader(String *field*)	Returns the **int** equivalent of the header field named *field*.

Table 27-5. *Various Methods Defined by* HttpServletRequest