Unit 7. Servlets and Java Server Pages

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.

A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.The **javax.servlet** and **javax.servlet.http** packages provide interfaces and classes for writing servlets. All servlets must implement the **Servlet** interface, which defines life-cycle methods.

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, when 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**. 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 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.

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.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 or Glass Fish server to obtain their functionality.

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.

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. The methods defined by Servlet are shown below:

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 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 below.

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.
String getRemoteAddr( )	Returns the string equivalent of the client IP address.
String getRemoteHost( )	Returns the string equivalent of the client host name.
String getScheme( )	Returns the transmission scheme of the URL used for the request (for example, "http", "ftp").
String getServerName( )	Returns the name of the server.
int getServerPort( )	Returns the port number.

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

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 below.

Method	Description
String getCharacterEncoding( )	Returns the character encoding for the response.
ServletOutputStream getOutputStream( ) throws IOException	Returns a **ServletOutputStream** that can be used to write binary data to the response. An **IllegalStateException** is thrown if **getWriter( )** has already been invoked for this request.
PrintWriter getWriter( ) throws IOException	Returns a **PrintWriter** that can be used to write character data to the response. An **IllegalStateException** is thrown if **getOutputStream( )** has already been invoked for this request.
void setContentLength(int *size*)	Sets the content length for the response to *size*.
void setContentType(String *type*)	Sets the content type for the response to *type*.

Table 27-4. *Various Methods Defined by* ServletResponse

Note: For detailed information about **javax.servlet** package refer to the following link
http://docs.oracle.com/javaee/1.4/api/javax/servlet/package-summary.html

Reading Servlet Parameters

The **ServletRequest class** includes methods that allow to read the **names and values of parameters** that are included in a client request. We will develop a servlet that illustrates their use. The example contains two files. A **Web page** is defined in **index.jsp** and a servlet is defined in **PostParametersServlet.java**. The HTML source code for index.jsp is shown in the following listing. It defines a table that contains two labels and two text fields. One of the labels is Employee and the other is Phone. There is also a submit button. Notice that the action parameter of the form tag specifies a URL. The URL identifies the servlet to process the HTTP POST request.

```
//index.jsp
<html>
<body>
<center>
```

```
<form name="Form1" method="post" action="PostParametersServlet">
<table>
<tr>
<td><B>Employee</td>
<td><input type=textbox name="e" size="25" value=""></td>
</tr>
<tr>
<td><B>Phone</td>
<td><input type=textbox name="p" size="25" value=""></td>
</tr>
</table>
<input type=submit value="Submit">
</body>
</html>

//PostParametersServlet.java
import java.io.*;
import java.util.*;
import javax.servlet.*;

public class PostParametersServlet extends GenericServlet {

   public void service(ServletRequest request,ServletResponse response)
  throws ServletException, IOException {
// Get print writer.
PrintWriter pw = response.getWriter();
// 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();
}
}
```

output

```
e = navin
p = 9841
```

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 below.

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*.

String getMethod( )	Returns the HTTP method for this request.
String getPathInfo( )	Returns any path information that is located after the servlet path and before a query string of the URL.
String getPathTranslated( )	Returns any path information that is located after the servlet path and before a query string of the URL after translating it to a real path.
String getQueryString( )	Returns any query string in the URL.
String getRemoteUser( )	Returns the name of the user who issued this request.
String getRequestedSessionId( )	Returns the ID of the session.
String getRequestURI( )	Returns the URI.
StringBuffer getRequestURL( )	Returns the URL.
String getServletPath( )	Returns that part of the URL that identifies the servlet.
HttpSession getSession( )	Returns the session for this request. If a session does not exist, one is created and then returned.
HttpSession getSession(boolean *new*)	If *new* is **true** and no session exists, creates and returns a session for this request. Otherwise, returns the existing session for this request.
boolean isRequestedSessionIdFromCookie( )	Returns **true** if a cookie contains the session ID. Otherwise, returns **false**.
boolean isRequestedSessionIdFromURL( )	Returns **true** if the URL contains the session ID. Otherwise, returns **false**.
boolean isRequestedSessionIdValid( )	Returns **true** if the requested session ID is valid in the current session context.

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

The HttpServletResponse Interface

The HttpServletResponse interface is implemented by the server. It enables a servlet to formulate an HTTP response to a client. Several constants are defined. These correspond to the different status codes that can be assigned to an HTTP response. For example, SC_OK indicates that the HTTP request succeeded and SC_NOT_FOUND indicates that the requested resource is not available. Several methods of this interface are summarized in Table below.

Method	Description
void addCookie(Cookie *cookie*)	Adds *cookie* to the HTTP response.
boolean containsHeader(String *field*)	Returns **true** if the HTTP response header contains a field named *field*.
String encodeURL(String *url*)	Determines if the session ID must be encoded in the URL identified as *url*. If so, returns the modified version of *url*. Otherwise, returns *url*. All URLs generated by a servlet should be processed by this method.
String encodeRedirectURL(String *url*)	Determines if the session ID must be encoded in the URL identified as *url*. If so, returns the modified version of *url*. Otherwise, returns *url*. All URLs passed to **sendRedirect( )** should be processed by this method.
void sendError(int *c*) throws IOException	Sends the error code *c* to the client.
void sendError(int *c*, String *s*) throws IOException	Sends the error code *c* and message *s* to the client.
void sendRedirect(String *url*) throws IOException	Redirects the client to *url*.
void setDateHeader(String *field*, long *msec*)	Adds *field* to the header with date value equal to *msec* (milliseconds since midnight, January 1, 1970, GMT).
void setHeader(String *field*, String *value*)	Adds *field* to the header with value equal to *value*.
void setIntHeader(String *field*, int *value*)	Adds *field* to the header with value equal to *value*.
void setStatus(int *code*)	Sets the status code for this response to *code*.

Table 27-6. *Various Methods Defined by* HttpServletResponse

The Cookie Class

The Cookie class encapsulates a cookie. A cookie is stored on a client and contains state information. Cookies are valuable for tracking user activities. For example, assume that a user visits an online store. A cookie can save the user's name, address, and other information. The user does not need to enter this data each time he or she visits the store. A servlet can write a cookie to a user's machine via the addCookie( ) method of the HttpServletResponse interface. The data for that cookie is then included in the header of the HTTP response that is sent to the browser.

The names and values of cookies are stored on the user's machine. Some of the information that is

saved for each cookie includes the following:

- The name of the cookie
- The value of the cookie
- The expiration date of the cookie
- The domain and path of the cookie

The expiration date determines when this cookie is deleted from the user's machine. If an expiration date is not explicitly assigned to a cookie, it is deleted when the current browser session ends. Otherwise, the cookie is saved in a file on the user's machine.

The domain and path of the cookie determine when it is included in the header of an HTTP request. If the user enters a URL whose domain and path match these values, the cookie is then supplied to the Web server. Otherwise, it is not.

The methods of the Cookie class are summarized in Table below

Method	Description
Object clone( )	Returns a copy of this object.
String getComment( )	Returns the comment.
String getDomain( )	Returns the domain.
int getMaxAge( )	Returns the age (in seconds).
String getName( )	Returns the name.
String getPath( )	Returns the path.
boolean getSecure( )	Returns **true** if the cookie must be sent using only a secure protocol. Otherwise, returns **false**.
String getValue( )	Returns the value.
int getVersion( )	Returns the cookie protocol version. (Will be 0 or 1.)
void setComment(String *c*)	Sets the comment to *c*.
void setDomain(String *d*)	Sets the domain to *d*.
void setMaxAge(int *secs*)	Sets the maximum age of the cookie to *secs*. This is the number of seconds after which the cookie is deleted. Passing –1 causes the cookie to be removed when the browser is terminated.
void setPath(String *p*)	Sets the path to *p*.
void setSecure(boolean *secure*)	Sets the security flag to *secure*, which means that cookies will be sent only when a secure protocol is being used.
void setValue(String *v*)	Sets the value to *v*.
void setVersion(int v)	Sets the cookie protocol version to *v*, which will be 0 or 1.

Table 27-8. *The Methods Defined by* Cookie

The HttpServlet Class

The HttpServlet class extends GenericServlet. It is commonly used when developing servlets that receive

and process HTTP requests. The methods of the HttpServlet class are summarized in Table below.

Method	Description
void doDelete(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Performs an HTTP DELETE.
void doGet(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Performs an HTTP GET.
void doHead(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Performs an HTTP HEAD.
void doOptions(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Performs an HTTP OPTIONS.
void doPost(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Performs an HTTP POST.
void doPut(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Performs an HTTP PUT.
void doTrace(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Performs an HTTP TRACE.
long getLastModified(HttpServletRequest *req*)	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the requested resource was last modified.
void service(HttpServletRequest *req*, HttpServletResponse *res*) throws IOException, ServletException	Called by the server when an HTTP request arrives for this servlet. The arguments provide access to the HTTP request and response, respectively.

Table 27-9. *The Methods Defined by* HttpServlet

Handling HTTP Requests and Responses

The HttpServlet class provides specialized methods that handle the various types of HTTP requests. A servlet developer typically overrides one of these methods. These methods are **doDelete( ), doGet( ), doHead( ), doOptions( ), doPost( ), doPut( ), and doTrace( ).**

Handling HTTP GET Requests

Here we will develop a servlet that handles an HTTP GET request. The servlet is invoked when a form on a Web page is submitted.

//index.jsp

<html>

```
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Testing GET</title>
  </head>
  <body>
    <form action="testingget" method="get">
      <label style="color: green;"> <b>Testing Get:</b></label><br/></br>
      First Name: <input type="text" name="firstName" size="20"><br /><br/>
      Last Name:  <input type="text" name="surname" size="20">
    <br /><br />
    <input type="submit" value="Submit"></br></br>
  </form>
</body>
</html>

//TestingGet
import java.io.PrintWriter;
import java.io.IOException;
import java.sql.*;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.UnavailableException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestingGet extends HttpServlet {

   private Connection connection;
   private Statement statement;

  // set up database connection and create SQL statement
  public void init( ServletConfig config ) throws ServletException
  {
    // attempt database connection and create Statement
    try
    {

      connection=DriverManager.getConnection( "jdbc:mysql://localhost:3306/testingget","root","");

      // create Statement to query database
      statement = connection.createStatement();
    } // end try
    // for any exception throw an UnavailableException to
    // indicate that the servlet is not currently available
    catch ( Exception exception )
    {
      exception.printStackTrace();
```

```
      throw new UnavailableException( exception.getMessage() );
    } // end catch
  }  // end method init

  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
     response.setContentType("text/html;charset=UTF-8");
     PrintWriter out = response.getWriter();
     try {
     String firstName = request.getParameter("firstName").toString();
     String surname = request.getParameter("surname").toString();
     try {
     statement = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_UPDATABLE);

     ResultSet uprs = statement.executeQuery(
        "SELECT * FROM names");

     uprs.moveToInsertRow();
     uprs.updateString("firstname",firstName);
     uprs.updateString("lastname",surname);
     uprs.insertRow();
     uprs.beforeFirst();
  }
     catch ( SQLException sqlException )
   {
     sqlException.printStackTrace();
   }
     try
   {

     // create Statement for querying database
     statement = connection.createStatement();

     // query database
     ResultSet resultSet = statement.executeQuery(
        "SELECT * from names" );
     out.println("<html>");
     out.println("<head>");
     out.println("</head>");
     out.println("<body>");
     out.println("<p>Welcome " + firstName + " " + surname + "</p>");
     out.println( "<p>People currently in the database:</p>" );
     // process query results
     ResultSetMetaData metaData = resultSet.getMetaData();
     int numberOfColumns = metaData.getColumnCount();
     for ( int i = 1; i <= numberOfColumns; i++ )
```

```
          out.println("<label style='color:red'>"+ metaData.getColumnName( i )+"</label>" );
       out.println("</br>");
        while ( resultSet.next() )
        {
          for ( int i = 1; i <= numberOfColumns; i++ )
            out.println("<label style='color:blue'>"+ resultSet.getObject( i )+"</label>" );
          out.println("</br>");
        } // end while
       out.println("</body>");
       out.println("</html>");
     }  // end try
     catch ( SQLException sqlException )
     {
       sqlException.printStackTrace();
     } // end catch

    }//end try

     finally {
          out.close();
       }
   }
 // close SQL statements and database when servlet terminates
  public void destroy()
  {
     // attempt to close statements and database connection
     try
     {
       statement.close();
       connection.close();
     } // end try
     // handle database exceptions by returning error to client
     catch( SQLException sqlException )
     {
       sqlException.printStackTrace();
     } // end catch
  } // end method destroy

}
```

Handling HTTP POST Requests

Here we will develop a servlet that handles an HTTP POST request. The servlet is invoked when a form on a Web page is submitted.

```
//index.jsp
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Testing POST</title>
```

```
  </head>
  <body>
    <form action="testingpost" method="post">
      <label style="color: red;"> <b>Testing Post:</b></label><br/></br>
      First Name: <input type="text" name="firstName" size="20"><br /><br/>
    <input type="submit" value="Submit"><br/><br/>
  </form>
</body>
</html>

//TestingPost
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestingPost extends HttpServlet {
  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      String firstName = request.getParameter("firstName").toString();
    out.println("<html>");
    out.println("<head>");
    out.println("</head>");
    out.println("<body>");
    out.println("<label style='color:red'>Welcome </label>");
    out.print("<label style='color:green'>"+firstName+"</label>");
    out.println("</body>");
    out.println("</html>");}
    finally {
      out.close();
    }
  }
}
```

Using Cookies

Now, let's develop a servlet that illustrates how to use cookies. The servlet is invoked when a form on a Web page is submitted. The example contains three files as summarized here:

File	Description
index.jsp	Allows a user to specify a value for the cookie named MyCookie.
AddCookie.java	Processes the submission of AddCookie.htm.
GetCookie.java	Displays cookie values.

```
//index.jsp
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Testing Cookies</title>
  </head>
  <body>
    </form>
    <form action="addCookie" method="post">
      <label style="color: red;"> <b>Testing Cookies<b></label><br/></br>
            <b>Enter the value for cookie</b></br>
      First Name: <input type="text" name="firstName" size="20"><br /><br/>
       Last Name:  <input type="text" name="surname" size="20"><br/><br/>
    <input type="submit" value="Submit"><br/><br/>
  </form>
    <label><b>Click below to get Cookies Value</b></label></br>
    <a href="getCookie">click here</a> <br/><br/>
</body>
</html>

//AddCookie.java
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      // Get parameter from HTTP request.
      String data = request.getParameter("firstName");
      String data1 = request.getParameter("surname");

      // Create cookie.
      Cookie cookie = new Cookie("FirstCookie", data);
      Cookie cookie1 = new Cookie("SecondCookie", data1);

      // Add cookie to HTTP response.
      response.addCookie(cookie);
      response.addCookie(cookie1);
      // Write output to browser.
      out.println("<html>");
```

```
            out.println("<head>");
            out.println("<title>Servlet AddCookie</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<B>MyCookie has been set to");
            out.println(data);
             out.println("<br/>");
             out.println(data1);
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }
}

//GetCookie.java
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            Cookie[] cookies = request.getCookies();
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet GetCookie</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<B>");
            for(int i = 0; i < cookies.length; i++) {
            String name = cookies[i].getName();
            String value = cookies[i].getValue();
            out.println("name = " + name +
            "; value = " + value);
            out.println("</br>");
            out.println("</body>");
            out.println("</html>");
        }
        }
        finally {
```

```
        out.close();
      }
    }
}
```

Session Tracking

HTTP is a stateless protocol. Each request is independent of the previous one. However, in some applications, it is necessary to save state information so that information can be collected from several interactions between a browser and a server. Sessions provide such a mechanism.

A session can be created via the **getSession( )** method of HttpServletRequest. An HttpSession object is returned. This object can store a set of bindings that associate names with objects. The **setAttribute( ), getAttribute( ), getAttributeNames( ), and removeAttribute( ) methods** of HttpSession manage these bindings. It is important to note that session state is shared among all the servlets that are associated with a particular client.

```
//index.jsp
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Testing Cookies</title>
  </head>
  <body>
    <label style="color: blue"><b>Testing Session</b></label></br>
    <label><b>Click below to get Session Value</b></label></br>
    <a href="getSession">click here</a>
</body>
</html>
```

```
//GetSession.java
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      // Get the HttpSession object.
      HttpSession hs = request.getSession(true);
      // Get writer.
      // response.setContentType("text/html");
```

```
        //PrintWriter pw = response.getWriter();
        out.print("<B>");
        // Display date/time of last access.
        Date date = (Date)hs.getAttribute("date");

        // Display current date/time.

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet GetSession</title>");
        out.println("</head>");
        out.println("<body>");
       if(date != null) {
        out.print("Last access: " + date + "<br>");
        }
       date = new Date();
        hs.setAttribute("date", date);
        out.println("Current date: " + date);
        out.println("</body>");
        out.println("</html>");
     } finally {
        out.close();
     }
  }
}
```

JavaServer Pages (JSP)

In the previous chapter, you learned how to generate dynamic Web pages with servlets. You probably have already noticed in our examples that most of the code in our servlets generated output that consisted of the HTML elements that composed the response to the client. Only a small portion of the code dealt with the business logic. Generating responses from servlets requires that Web application developers be familiar with Java. However, many people involved in Web application development, such as Web site designers, do not know Java. It is difficult for people who are not Java programmers to implement, maintain and extend a Web application that consists of primarily of servlets. The solution to this problem is JavaServer Pages (JSP)an extension of servlet technology that separates the presentation from the business logic. This lets Java programmers and Web-site designers focus on their strengthswriting Java code and designing Web pages, respectively.

JavaServer Pages simplify the delivery of dynamic Web content. They enable Web application programmers to create dynamic content by reusing predefined components and by interacting with components using server-side scripting. Custom-tag libraries are a powerful feature of JSP that allows Java developers to hide complex code for database access and other useful services for dynamic Web pages in custom tags. Web sites use these custom tags like any other Web page element to take advantage of the more complex functionality hidden by the tag. Thus, Web-page designers who are not familiar with Java can enhance Web pages with powerful dynamic content and processing capabilities.

The classes and interfaces that are specific to JavaServer Pages programming are located in packages

javax.servlet.jsp and **javax.servlet.jsp.tagext**.

JavaServer Pages Overview

There are **four key components** to JSPs-**directives, actions, scripting elements and tag libraries**. **Directives** are messages to the JSP container-the server component that executes JSPs-that enable the programmer to specify page settings, to include content from other resources and to specify custom tag libraries for use in a JSP. **Actions** encapsulate functionality in predefined tags that programmers can embed in a JSP. Actions often are performed based on the information sent to the server as part of a particular client request. They also can create Java objects for use in JSP scriptlets. **Scripting elements** enable programmers to insert Java code that interacts with components in a JSP (and possibly other Web application components) to perform request processing. **Scriptlets**, one kind of scripting element, contain code fragments that describe the action to be performed in response to a user request**. Tag libraries** are part of the tag extension mechanism that enables programmers to create custom tags. Such tags enable Web page designers to manipulate JSP content without prior Java knowledge.

In some ways, JavaServer Pages look like standard XHTML or XML documents. In fact, JSPs normally include XHTML or XML markup. Such markup is known as **fixed-template data or fixed-template text.** Fixed-template data often helps a programmer decide whether to use a servlet or a JSP. Programmers tend to use JSPs when most of the content sent to the client is fixed-template data and little or none of the content is generated dynamically with Java code. Programmers typically use servlets when only a small portion of the content sent to the client is fixed-template data. In fact, some servlets do not produce content. Rather, they perform a task on behalf of the client, then invoke other servlets or JSPs to provide a response. Note that in most cases servlet and JSP technologies are interchangeable. As with servlets, JSPs normally execute as part of a Web server.

When a JSP-enabled server receives the first request for a JSP, the JSP container translates the JSP into a Java servlet that handles the current request and future requests to the JSP. Literal text in a JSP becomes string literals in the servlet that represents the translated JSP. Any errors that occur in compiling the new servlet result in translation-time errors. The JSP container places the Java statements that implement the JSP's response in method _jspService at translation time. If the new servlet compiles properly, the JSP container invokes method _jspService to process the request. The JSP may respond directly or may invoke other Web application components to assist in processing the request. Any errors that occur during request processing are known as request-time errors.

Overall, the request-response mechanism and the JSP life cycle are the same as those of a servlet. JSPs can override methods **jspInit** and **jspDestroy** (similar to servlet methods init and destroy), which the JSP container invokes when initializing and terminating a JSP, respectively. JSP programmers can define these methods using JSP declarations--part of the JSP scripting mechanism.

A Simple JSP Example

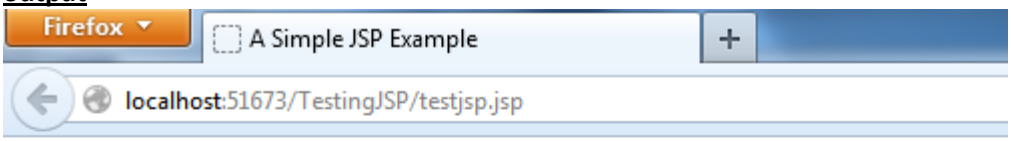
JSP expression inserting the date and time into a Web page.

//test.jsp

```
<html>
  <head>
    <meta http-equiv = "refresh" content = "60" />
     <title>A Simple JSP Example</title>
     <style type = "text/css">
       .big { font-family: helvetica, arial, sans-serif;
```

```
            font-weight: bold;
            font-size: 2em; }
    </style>
  </head>
  <body>
    <p class = "big">Simple JSP Example</p>
    <table style = "border: 6px outset;">
      <tr>
        <td style = "background-color: black;">
          <p class = "big" style = "color: cyan;">
            <!-- JSP expression to insert date/time -->
            <%= new java.util.Date() %>
          </p>
        </td>
      </tr>
    </table>
  </body
</html>
```

output



Simple JSP Example

As you can see, most of test.jsp consists of XHTML markup.In cases like this, JSPs are easier to implement than servlets. In a servlet that performs the same task as this JSP, each line of XHTML markup typically is a separate Java statement that outputs the string representing the markup as part of the response to the client. Writing code to output markup can often lead to errors.That's whhy in such scenarios JSP is preferred than Servlets.The key line in the above program is the expression

<%= new java.util.Date() %>

JSP expressions are delimited by <%= and %>. The preceding expression creates a new instance of class Date (package java.util). By default, a Date object is initialized with the current date and time. When the client requests this JSP, the preceding expression inserts the String representation of the date and time

in the response to the client. [Note: **Because the client of a JSP could be anywhere in the world, the JSP should return the date in the client locale's format**. However, the JSP executes on the server, so the server's locale determines the String representation of the Date.

We use the XHTML meta element in line 9 to set a refresh interval of 60 seconds for the document. This causes the browser to request test.jsp every 60 seconds. For each request to test.jsp, the JSP container reevaluates the expression in line 24, creating a new Date object with the server's current date and time.

When you first invoke the JSP, you may notice a brief delay as GlassFish Server translates the JSP into a servlet and invokes the servlet to respond to your request

Implicit Objects

Implicit objects provide access to many servlet capabilities in the context of a JavaServer Page. Implicit objects have **four scopes: application, page, request and session**. The JSP container owns objects with application scope. Any JSP can manipulate such objects. Objects with page scope exist only in the page that defines them. Each page has its own instances of the page-scope implicit objects. Objects with request scope exist for the duration of the request. For example, a JSP can partially process a request, then forward it to a servlet or another JSP for further processing. Request-scope objects go out of scope when request processing completes with a response to the client. Objects with session scope exist for the client's entire browsing session. Figure below describes the JSP implicit objects and their scopes.

Implicit object	Description
Application Scope	
`application`	A `javax.servlet.ServletContext` object that represents the container in which the JSP executes.
Page Scope	
`config`	A `javax.servlet.ServletConfig` object that represents the JSP configuration options. As with servlets, configuration options can be specified in a Web application descriptor.
`exception`	A `java.lang.Throwable` object that represents an exception that is passed to a JSP error page. This object is available only in a JSP error page.
`out`	A `javax.servlet.jsp.JspWriter` object that writes text as part of the response to a request. This object is used implicitly with JSP expressions and actions that insert string content in a response.
`page`	An `Object` that represents the `this` reference for the current JSP instance.
`pageContext`	A `javax.servlet.jsp.PageContext` object that provides JSP programmers with access to the implicit objects discussed in this table.
`response`	An object that represents the response to the client and is normally an instance of a class that implements `HttpServletResponse` (package `javax.servlet.http`). If a protocol other than HTTP is used, this object is an instance of a class that implements `javax.servlet.ServletResponse`.
Request Scope	
`request`	An object that represents the client request and is normally an instance of a class that implements `HttpServletRequest` (package `javax.servlet.http`). If a protocol other than HTTP is used, this object is an instance of a subclass of `javax.servlet.Servlet-Request`.
Session Scope	
`session`	A `javax.servlet.http.HttpSession` object that represents the client session information if such a session has been created. This object is available only in pages that participate in a session.

fig. JSP implicit objects.

Scripting

JavaServer Pages often present dynamically generated content as part of an XHTML document that is

sent to the client in response to a request. In some cases, the content is static but is output only if certain conditions are met during a request (e.g., providing values in a form that submits a request). **JSP programmers can insert Java code and logic in a JSP using scripting.**

Scripting Components
The JSP scripting components include **scriptlets, comments, expressions, declarations and escape sequences.**

Scriptlets are blocks of code delimited by **<% and %>.** They contain Java statements that the container places in method _jspService at translation time.

JSPs support three **comment** styles: **JSP comments, XHTML comments and scripting-language comments. JSP comments** are delimited by **<%-- and --%>.** These can be placed throughout a JSP, but not inside scriptlets. **XHTML comments** are delimited with **<!-- and -->.** These, too, can be placed throughout a JSP, but not inside scriptlets. **Scripting language comments** are currently **Java comments**, because Java currently is the only JSP scripting language. Scriptlets can use Java's **end-of-line //** comments and traditional comments (delimited by **/* and */).** JSP comments and scripting-language comments are ignored and do not appear in the response to a client. When clients view the source code of a JSP response, they will see only the XHTML comments in the source code. The different comment styles are useful for separating comments that the user should be able to see from those that document logic processed on the server.

JSP expressions are delimited by **<%= and %>** and contain a Java expression that is evaluated when a client requests the JSP containing the expression. The container converts the result of a JSP expression to a String object, then outputs the String as part of the response to the client.

Declarations, delimited by **<%! and %>,** enable a JSP programmer to define variables and methods for use in a JSP. Variables become instance variables of the servlet class that represents the translated JSP. Similarly, methods become members of the class that represents the translated JSP. Declarations of variables and methods in a JSP use Java syntax. Thus, a variable declaration must end with a semicolon, as in

```
<%! int counter = 0; %>
```

Special characters or character sequences that the JSP container normally uses to delimit JSP code can be included in a JSP as literal characters in scripting elements, fixed template data and attribute values using **escape sequences**. Figure below shows the literal character or characters and the corresponding escape sequences and discusses where to use the escape sequences.

Literal	Escape sequence	Description
<%	<\%	The character sequence <% normally indicates the beginning of a scriptlet. The <\% escape sequence places the literal characters <% in the response to the client.
%>	%\>	The character sequence %> normally indicates the end of a scriptlet. The %\> escape sequence places the literal characters %> in the response to the client.
' " \	\' \" \\	As with string literals in a Java program, the escape sequences for characters ',", and \ allow these characters to appear in attribute values. Remember that the literal text in a JSP becomes string literals in the servlet that represents the translated JSP.

fig. JSP escape sequences

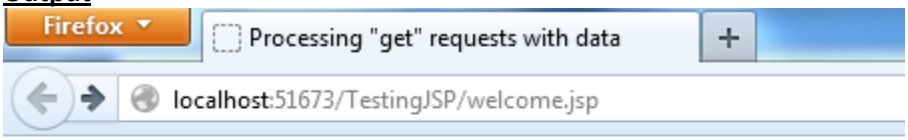
Scripting Example

//welcome.jsp

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Processing "get" requests with data</title>
  </head>
  <!-- body section of document -->
   <body>
    <% // begin scriptlet
     String name = request.getParameter( "firstName" );

      if ( name != null )
      {
    %> <%-- end scriptlet to insert fixed template data --%>

        <h1>
          Hello <%= name %>, <br />
          Welcome to JavaServer Pages!
        </h1>

    <% // continue scriptlet

      } // end if
      else {

    %> <%-- end scriptlet to insert fixed template data --%>
```

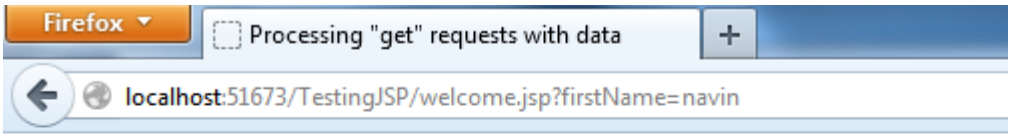
```
        <form action = "welcome.jsp" method = "get">
          <p>Type your first name and press Submit</p>

          <p><input type = "text" name = "firstName" />
            <input type = "submit" value = "Submit" />
          </p>
        </form>

    <% // continue scriptlet

      } // end else

    %> <%-- end scriptlet --%>
  </body>
</html>
```

Output



Type your first name and press Submit

navin Submit



Hello navin,
Welcome to JavaServer Pages!

Standard Actions

Standard actions provide JSP implementors with access to several of the most common tasks performed in a JSP, **such as including content from other resources, forwarding requests to other resources and interacting with JavaBean software components**. JSP containers process actions at request time. Actions are delimited by **<jsp:action> and </jsp:action>,** where **action is the standard action name**. In cases where nothing appears between the starting and ending tags, the XML empty element syntax <jsp: action /> can be used. Figure below summarizes the JSP standard actions.

Action	Description
`<jsp:include>`	Dynamically includes another resource in a JSP. As the JSP executes, the referenced resource is included and processed.
`<jsp:forward>`	Forwards request processing to another JSP, servlet or static page. This action terminates the current JSP's execution.
`<jsp:plugin>`	Allows a plug-in component to be added to a page in the form of a browser-specific `object` or `embed` HTML element. In the case of a Java applet, this action enables the browser to download and install the Java Plug-in, if it is not already installed on the client computer.
`<jsp:param>`	Used with the `include`, `forward` and `plugin` actions to specify additional name-value pairs of information for use by these actions.
JavaBean Manipulation	
`<jsp:useBean>`	Specifies that the JSP uses a JavaBean instance (i.e., an object of the class that declares the JavaBean). This action specifies the scope of the object and assigns it an ID (i.e., a variable name) that scripting components can use to manipulate the bean.
`<jsp:setProperty>`	Sets a property in the specified JavaBean instance. A special feature of this action is automatic matching of request parameters to bean properties of the same name.
`<jsp:getProperty>`	Gets a property in the specified JavaBean instance and converts the result to a string for output in the response.

fig. JSP standard actions

<jsp:include> Action

JavaServer Pages support two include mechanisms-the **<jsp:include> action** and the **include directive**. Action <jsp:include> enables dynamic content to be included in a JavaServer Page at request time. If the included resource changes between requests, the next request to the JSP containing the <jsp:include> action includes the resource's new content. On the other hand, the include directive copies the content into the JSP once, at JSP translation time. If the included resource changes, the new content will not be reflected in the JSP that used the include directive, unless that JSP is recompiled, which normally would occur only if a new version of the JSP is installed. Figure below describes the attributes of action <jsp:include>.

Attribute	Description
page	Specifies the relative URI path of the resource to include. The resource must be part of the same Web application.
flush	Specifies whether the implicit object out should be flushed before the include is performed. If true, the JspWriter out is flushed prior to the inclusion, hence you could no longer forward to another page later on. The default value is false.

fig. Action <jsp:include> attributes.

//index.jsp

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>LN TECH PVT. LTD</title>

   <style type = "text/css">
     body
     {
       font-family: tahoma, helvetica, arial, sans-serif;
     }

     table, tr, td
     {
       font-size: .9em;
       border: 3px groove;
       padding: 5px;
       background-color: yellowgreen;
     }
   </style>

  </head>
  <body>
    <table style="width: 1280px; height: 675px">
     <tr>
       <td style = "width: 215px; text-align: center">
         <img src = "LN_Tech_logo.jpg"
           width = "140" height = "93"
           alt = "LN Tech Logo" />
       </td>
       <td>
         <%-- include banner.html in this JSP --%>
         <jsp:include page = "banner.html"
```

```
            flush = "true" />
        </td>
      </tr>
      <tr>
        <td style = "width: 215px">
          <%-- include toc.html in this JSP --%>
          <jsp:include page = "toc.html" flush = "true" />
        </td>
        <td style = "vertical-align: top">
          <%-- include clock.jsp in this JSP --%>
          <jsp:include page = "clock.jsp"
            flush = "true" />
        </td>
      </tr>
    </table>
  </body>
</html>
```

//banner.html

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
<div style = "width: 580px">
  <p><b>
    LN Tech....a dedicated team of Engineers <br /> Working
    in the field of Web<br />
    welcomes you to explore our site</b>
  </p>
  <p>
    <a href = "mailto:admin@lntech.com">admin@lntech.com</a>
    <br />Baneshwor<br />Kathmandu, Nepal
  </p>
</div>
  </body>
</html>
```

//toc.html

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
```

```
<p><a href = "signup.jsp">
      <b>Sign up</b>
</a></p>
<p><a href = "http://theastutetech.com/index.php?page=about-us">
      <b>About us</b>
</a></p>

<p><a href = "http://theastutetech.com/index.php?page=services">
      <b>Services</b>
</a></p>

<p><a href = "http://theastutetech.com/index.php?page=our-works">
      <b>Our works/Porfolios</b>
</a></p>

<p><a href = "http://theastutetech.com/index.php?page=jobs">
      <b>Jobs</b>
</a></p>
<p><a href = "http://theastutetech.com/">
      <b>Home Page</b>
</a></p>

<p>Send questions or comments about this site to
  <a href = "mailto:lntech.com">
    admin@lntech.com
  </a><br />
  Copyright 2009-2012 by LN Tech Pvt Ltd.
  All Rights Reserved.
</p>
   </body>
</html>
```

//clock.jsp

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Clock Page</title>
   </head>
   <body>
     <table>
  <tr>
    <td style = "background-color: blanchedalmond;">
      <p class = "big" style = "color: black; font-size: 3em;
        font-weight: bold;">

        <%-- script to determine client local and --%>
        <%-- format date accordingly           --%>
```

```
        <%
          // get client locale
          java.util.Locale locale = request.getLocale();

          // get DateFormat for client's Locale
          java.text.DateFormat dateFormat =
            java.text.DateFormat.getDateTimeInstance(
              java.text.DateFormat.LONG,
              java.text.DateFormat.LONG, locale );

        %>  <%-- end script --%>

        <%-- output date --%>
        <%= dateFormat.format( new java.util.Date() ) %>
      </p>
    </td>
  </tr>
</table>
  </body>
</html>
```

//signup.jsp

```
<!DOCTYPE html>
<html>
  <!-- head section of document -->
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Sign up Page</title>
  </head>
  <!-- body section of document -->
    <body>
   <% // begin scriptlet

      String name = request.getParameter( "firstName" );

      if ( name != null )
      {
    %> <%-- end scriptlet to insert fixed template data --%>

        <h1>
          Hello <%= name %>, <br />
          Welcome to LN Tech!
        </h1>

    <% // continue scriptlet

      } // end if
      else {
```

```
    %> <%-- end scriptlet to insert fixed template data --%>

        <form action = "signup.jsp" method = "get">
          <p>Type your first name and press Submit</p>

          <p><input type = "text" name = "firstName" />
            <input type = "submit" value = "Submit" />
          </p>
        </form>

    <% // continue scriptlet

      } // end else

    %> <%-- end scriptlet --%>
  </body>
   </body>
</html> <!-- end XHTML document -->
```

output