Unit-4

Introducing 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.

4. 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.

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 ser vlet is unloaded.
Ser vletConfig getSer vletConfig( )	Returns a ServletConfig object that contains any initialization parameters.
String getSer vletInfo( )	Returns a string describing the ser vlet.
void init(Ser vletConfig sc) throws Ser vletException	Called when the ser vlet is initialized. Initialization parameters for the ser vlet can be obtained from sc. An UnavailableException should be thrown if the ser vlet cannot be initialized.
void ser vice(Ser vletRequest req, Ser vletResponse res) throws Ser vletException, 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 ser vlet or IO problem occurs.

TABLE 31-1 The Methods Defined by Servlet

The ServletConfig Interface

The **ServletConfig** interface 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<String> getInitParameterNames( )	Returns an enumeration of all initialization parameter names.
String getServletName( )	Returns the name of the invoking servlet.

The ServletContext Interface

The **ServletContext** interface enables servlets to obtain information about their environment. Several of its methods are summarized in Table 38-2.

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 (i.e., absolute) path that corresponds to the relative 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*)	Writes *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 38-2 Various Methods Defined by **ServletContext**

The ServletRequest Interface

The **ServletRequest** interface enables a servlet to obtain information about a client request. Several of its methods are summarized in Table 38-3.

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.
Ser vletInputStream getInputStream( ) throws IOException	Returns a ServletInputStream that can be used to read binar y 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.
Buf feredReader getReader( ) throws IOException	Returns a buf fered 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 getSer verName( )	Returns the name of the ser ver.
int getSer verPor t( )	Returns the por t number.

Table 38-3 Various Methods Defined by **ServletRequest**

The ServletResponse Interface

The **ServletResponse** interface enables a servlet to formulate a response for a client. Several of its methods are summarized in Table 31-4.

Method	Description
String getCharacterEncoding( )	Returns the character encoding for the response.
Ser vletOutputStream getOutputStream( ) throws IOException	Returns a ServletOutputStream that can be used to write binar y 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 31-4 Various Methods Defined by ServletResponse

The GenericServlet Class

The **GenericServlet** class provides implementations of the basic life cycle methods for a servlet. **GenericServlet** implements the **Servlet** and **ServletConfig** interfaces. In addition, a method to append a string to the server log file is available. The signatures of this method are shown here:

void log(String *s*)
void log(String *s*, Throwable *e*)

Here, *s* is the string to be appended to the log, and *e* is an exception that occurred.

The ServletInputStream Class

The **ServletInputStream** class extends **InputStream**. It is implemented by the servlet container and provides an input stream that a servlet developer can use to read the data from a client request. In addition to the input methods inherited from **InputStream**, a method is provided to read bytes from the stream. It is shown here:

int readLine(byte[ ] *buffer*, int *offset*, int *size*) throws IOException

Here, *buffer* is the array into which *size* bytes are placed starting at *offset*. The method returns the actual number of bytes read or –1 if an end-of-stream condition is encountered.

The ServletOutputStream Class

The **ServletOutputStream** class extends **OutputStream**. It is implemented by the servlet container and provides an output stream that a servlet developer can use to write data to a client response. In addition to the output methods provided

by **OutputStream**, it also defines the **print( )** and **println( )** methods, which output data to the stream.

The Servlet Exception Classes

javax.servlet defines two exceptions. The first is **ServletException**, which indicates that a servlet problem has occurred. The second is **UnavailableException**, which extends **ServletException**. It indicates that a servlet is unavailable.

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="http://localhost:8080/servlets-examples/
servlet/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
HttpSer vletRequest	Enables ser vlets to read data from an HTTP request.
HttpSer vletResponse	Enables ser vlets 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.
HttpSer vlet	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 ser vlet path and before a quer y string of the URL.
String getPathTranslated( )	Returns any path information that is located after the ser vlet path and before a quer y string of the URL after translating it to a real path.
String getQuer yString( )	Returns any quer y 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.
StringBuf fer getRequestURL( )	Returns the URL.
String getSer vletPath( )	Returns that par t of the URL that identifies the ser vlet.
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. Other wise, returns the existing session for this request.
boolean isRequestedSessionIdFromCookie( )	Returns true if a cookie contains the session ID. Other wise, returns false.
boolean isRequestedSessionIdFromURL( )	Returns true if the URL contains the session ID. Other wise, returns false.
boolean isRequestedSessionIdValid( )	Returns true if the requested session ID is valid in the current session context.

TABLE 31-5 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. Other wise, returns url. All URLs generated by a ser vlet 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. Other wise, 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, Januar y 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 31-6 Various Methods Defined by HttpServletResponse (continued)

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 getAttribute(String attr)	Returns the value associated with the name passed in attr. Returns null if attr is not found.
Enumeration getAttributeNames( )	Returns an enumeration of the attribute names associated with the session.
long getCreationTime( )	Returns the time (in milliseconds since midnight, Januar y 1, 1970, GMT) when this session was created.
String getId( )	Returns the session ID.
long getLastAccessedTime( )	Returns the time (in milliseconds since midnight, Januar y 1, 1970, GMT) when the client last made a request for this session.
void invalidate( )	Invalidates this session and removes it from the context.
boolean isNew( )	Returns true if the ser ver created the session and it has not yet been accessed by the client.
void removeAttribute(String attr)	Removes the attribute specified by attr from the session.
void setAttribute(String attr, Object val)	Associates the value passed in val with the attribute name passed in attr.

TABLE 31-7 The Methods Defined by HttpSession

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(HttpSer vletRequest req, HttpSer vletResponse res) throws IOException, Ser vletException	Handles an HTTP DELETE request.
void doGet(HttpSer vletRequest req, HttpSer vletResponse res) throws IOException, Ser vletException	Handles an HTTP GET request.
void doHead(HttpSer vletRequest req, HttpSer vletResponse res) throws IOException, Ser vletException	Handles an HTTP HEAD request.
void doOptions(HttpSer vletRequest req, HttpSer vletResponse res) throws IOException, Ser vletException	Handles an HTTP OPTIONS request.
void doPost(HttpSer vletRequest req, HttpSer vletResponse res) throws IOException, Ser vletException	Handles an HTTP POST request.
void doPut(HttpSer vletRequest req, HttpSer vletResponse res) throws IOException, Ser vletException	Handles an HTTP PUT request.
void doTrace(HttpSer vletRequest req, HttpSer vletResponse res) throws IOException, Ser vletException	Handles an HTTP TRACE request.
long getLastModified(HttpSer vletRequest req)	Returns the time (in milliseconds since midnight, Januar y 1, 1970, GMT) when the requested resource was last modified.
void ser vice(HttpSer vletRequest req, HttpSer vletResponse res) throws IOException, Ser vletException	Called by the ser ver when an HTTP request arrives for this servlet. The arguments provide access to the HTTP request and response, respectively.

TABLE 31-9 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

The servlet is invoked when a form on a web page is submitted. The example contains two files.

A web page is defined in **ColorGet.html**.

```
<html>
<body>
<center>
<form name="Form1"
  action="http://localhost:8080/examples/servlets/servlet/ColorGetServlet">
<B>Color:</B>
<select name="color" size="1">
<option value="Red">Red</option>
<option value="Green">Green</option>
<option value="Blue">Blue</option>
</select>
<br><br>
<input type=submit value="Submit">
</form>
</body>
</html>
```

A servlet is defined in **ColorGetServlet.java**.

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

public class ColorGetServlet extends HttpServlet {

  public void doGet(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException, IOException {

    String color = request.getParameter("color");
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<B>The selected color is: ");
    pw.println(color);
    pw.close();
  }
}
```

Compile the servlet. Next, copy it to the appropriate directory, and update the **web.xml** file. Then, 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. Select a color.
4. Submit the web page.

Assume that the user selects the red option and submits the form. The URL sent from the browser to the server is The characters to the right of the question mark are known as the *query string*.

```
http://localhost:8080/examples/servlets/servlet/ColorGetServlet?color=Red
```

Handling HTTP POST Requests

The servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined in **ColorPost.html**.

```
<html>
<body>
<center>
<form name="Form1"
  method="post"
  action="http://localhost:8080/examples/servlets/servlet/ColorPostServlet">
<B>Color:</B>
<select name="color" size="1">
<option value="Red">Red</option>
<option value="Green">Green</option>
<option value="Blue">Blue</option>
</select>
<br><br>
<input type=submit value="Submit">
</form>
</body>
</html>
```

A servlet is defined in **ColorPostServlet.java**.

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

public class ColorPostServlet extends HttpServlet {

  public void doPost(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException, IOException {

    String color = request.getParameter("color");
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<B>The selected color is: ");
    pw.println(color);
    pw.close();
  }
}
```

```
http://localhost:8080/examples/servlets/servlet/ColorPostServlet
```

Using Cookies

The example contains three files as summarized here:

File	Description
AddCookie.html	Allows a user to specify a value for the cookie named **MyCookie**.
AddCookieServlet.java	Processes the submission of **AddCookie.html.**
GetCookiesServlet.java	Displays cookie values.

AddCookie.html

```
<html>
<body>
<center>
<form name="Form1"
  method="post"
  action="http://localhost:8080/examples/servlets/servlet/AddCookieServlet">
<B>Enter a value for MyCookie:</B>
<input type=textbox name="data" size=25 value="">
<input type=submit value="Submit">
</form>
</body>
</html>
```

AddCookieServlet.java

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

public class AddCookieServlet extends HttpServlet {

  public void doPost(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException, IOException {

    // Get parameter from HTTP request.
    String data = request.getParameter("data");

    // Create cookie.
    Cookie cookie = new Cookie("MyCookie", data);

    // Add cookie to HTTP response.
    response.addCookie(cookie);

    // Write output to browser.
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<B>MyCookie has been set to");
    pw.println(data);
    pw.close();
  }
}
```

GetCookiesServlet.java

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

public class GetCookiesServlet extends HttpServlet {

  public void doGet(HttpServletRequest request,
    HttpServletResponse response)
  throws ServletException, IOException {

    // Get cookies from header of HTTP request.
    Cookie[] cookies = request.getCookies();

    // Display these cookies.
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.println("<B>");
    for(int i = 0; i < cookies.length; i++) {
      String name = cookies[i].getName();
      String value = cookies[i].getValue();
      pw.println("name = " + name +
        "; value = " + value);
    }
    pw.close();
  }
}
```

Compile the servlets. Next, copy them to the appropriate directory, and update the **web.xml** file. Then, perform these steps to test this example:

1. Start Tomcat, if it is not already running.
2. Display **AddCookie.html** in a browser.
3. Enter a value for **MyCookie**.
4. Submit the web page.

Next, request the following URL via the browser:

```
http://localhost:8080/examples/servlets/servlet/GetCookiesServlet
```

Session Tracking

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. Session state is shared by all servlets that are associated with a client.

The following servlet illustrates how to use session state:

That object is a **Date** object that encapsulates the date and time when this page was last accessed. A **Date** object encapsulating the current date and time is then created. The **setAttribute( )** method is called to bind the name "date" to this object.

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

public class DateServlet extends HttpServlet {

  public void doGet(HttpServletRequest request,
     HttpServletResponse response)
  throws ServletException, IOException {

     // Get the HttpSession object.
     HttpSession hs = request.getSession(true);

     // Get writer.
     response.setContentType("text/html");
     PrintWriter pw = response.getWriter();
     pw.print("<B>");

     // Display date/time of last access.
     Date date = (Date)hs.getAttribute("date");
     if(date != null) {
       pw.print("Last access: " + date + "<br>");
     }

     // Display current date/time.
     date = new Date();
     hs.setAttribute("date", date);
     pw.println("Current date: " + date);
  }
}
```

When you first request this servlet, the browser displays one line with the current date and time information. On subsequent invocations, two lines are displayed. The first line shows the date and time when the servlet was last accessed. The second line shows the current date and time.

11 Introduction to JSP

- It stands for **Java Server Pages**.
- It is a server side technology.
- It is used for creating web application.
- It is used to create dynamic web content.
- In this JSP tags are used to insert JAVA code into HTML pages.
- It is an advanced version of Servlet Technology.
- It is a Web based technology helps us to create dynamic and platform independent web pages.
- In this, Java code can be inserted in HTML/ XML pages or both.
- JSP is first converted into servlet by JSP container before processing the client's request.

Comparing JSP with Servlet

Sr. No.	Key	Servlet	JSP
1	Implementation	Servlet is developed on Java language.	JSP is primarily written in HTML language although Java code could also be written.
2	MVC	In contrast to MVC we can state servlet as a controller which receives the request process and send back the response.	On the other hand, JSP plays the role of view to render the response returned by the servlet.
3	Request type	Servlets can accept and process all type of protocol requests.	JSP on the other hand is compatible with HTTP request only.
4	Session Management	In Servlet by default session management is not enabled, the user has to enable it explicitly.	On the other hand in JSP session management is automatically enabled.
5	Performance	Servlet is faster than JSP.	JSP is slower than Servlet because first the translation of JSP to java code is taking place and then compiles.
6	Modification reflected	Modification in Servlet is a time-consuming task because it includes reloading, recompiling and restarting the server as we made any change in our code to get reflected.	On the other hands JSP modification is fast as just need to click the refresh button and code change would get reflected.

JSP pages are more advantageous than Servlet:

- They are easy to maintain.
- No recompilation or redeployment is required.
- JSP has access to entire API of JAVA .
- JSP are extended version of Servlet.

Features of JSP

- **Coding in JSP is easy** :- As it is just adding JAVA code to HTML/XML.
- **Reduction in the length of Code** :- In JSP we use action tags, custom tags etc.
- **Connection to Database is easier** :-It is easier to connect website to database and allows to read or write data easily to the database.
- **Make Interactive websites** :- In this we can create dynamic web pages which helps user to interact in real time environment.
- **Portable, Powerful, flexible and easy to maintain** :- as these are browser and server independent.
- **No Redeployment and No Re-Compilation** :- It is dynamic, secure and platform independent so no need to re-compilation.
- **Extension to Servlet** :- as it has all features of servlets, implicit objects and custom tags

Why Use JSP?

JavaServer Pages often serve the same purpose as programs implemented using the **Common Gateway Interface (CGI)**. But JSP offers several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.
- JSP are always compiled before they are processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including **JDBC, JNDI, EJB, JAXP,** etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

Using JSP

JSP syntax

Syntax available in JSP are following:

1. **Declaration Tag** :-It is used to declare variables.

Syntax:-

```
<%!  Dec var  %>
```

Example:-

```
<%! int var=10; %>
```

2. **Java Scriplets** :- It allows us to add any number of JAVA code, variables and expressions.

Syntax:-

```
<% java code %>
```

3. **JSP Expression** :- It evaluates and convert the expression to a string.

Syntax:-

```
<%= expression %>
```

Example:-

```
<% num1 = num1+num2 %>
```

4. **JAVA Comments** :- It contains the text that is added for information which has to be ignored.

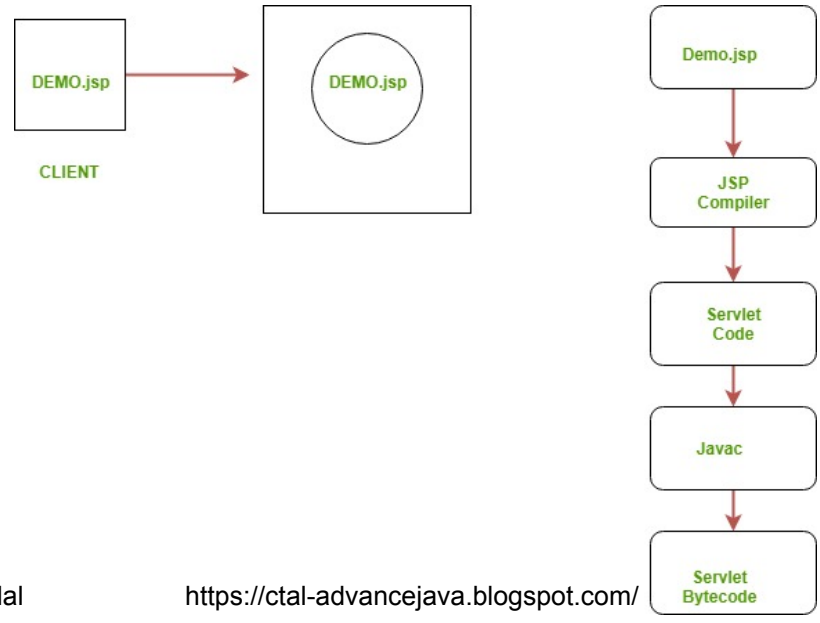
Syntax:-

```
<% -- JSP Comments %>
```

Process of Execution

Steps for Execution of JSP are following:-

- Create html page from where request will be sent to server eg try.html.
- To handle to request of user next is to create .jsp file Eg. new.jsp
- Create project folder structure.
- Create XML file eg my.xml.
- Create WAR file.
- Start Tomcat
- Run Application



Your First JSP

Let's start learning JSP with a **simple JSP**.

```
<%-- JSP comment --%>
<HTML>
<HEAD>
<TITLE>MESSAGE</TITLE>
</HEAD>
<BODY>
<%out.print("Hello, Sample JSP code");%>
</BODY>
</HTML>
```

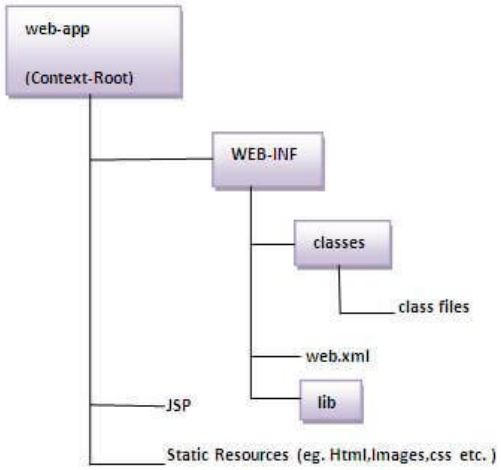
The above JSP generates the following output:
Hello, Sample JSP code.

Your Second JSP

```
HTML>
<BODY>
Hello BeginnersBook Readers!
Current time is: <%= new java.util.Date() %>
</BODY>
</HTML>
```

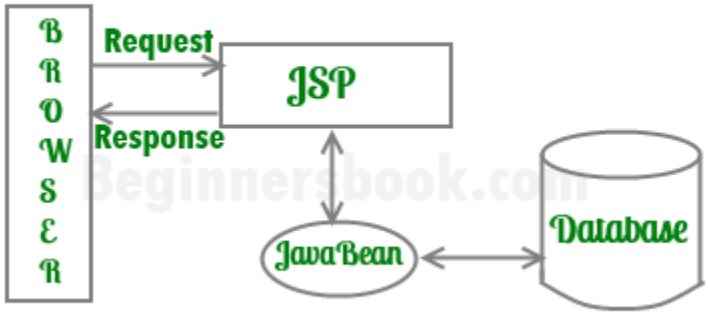
The Directory structure of JSP

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.

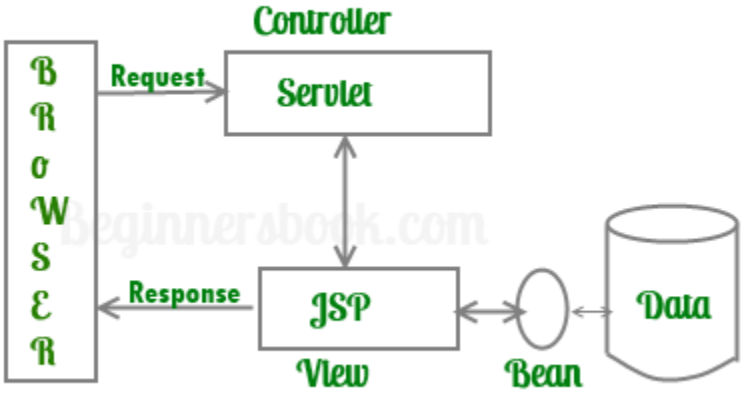


Architecture of a JSP Application

1) Model1 Architecture: In this Model, JSP plays a key role and it is responsible for of processing the request made by client.



2) Model2 Architecture: In this Model, Servlet plays a major role and it is responsible for processing the client's(web browser) request.



Java Web Frameworks

What is Framework in Java

Java Framework is the body or platform of pre-written codes used by Java developers to develop Java applications or web applications. In other words, **Java Framework** is a collection of predefined classes and functions that is used to process input, manage hardware devices interacts with system software. It acts like a skeleton that helps the developer to develop an application by writing their own code.

What is a framework?

Framework are the bodies that contains the pre-written codes (classes and functions) in which we can add our code to overcome the problem. We can also say that frameworks use programmer's code because the framework is in control of the programmer. We can use the framework by calling its methods, inheritance, and supplying "callbacks", listeners, or other implementations of the Observer pattern.

Popular Java Frameworks

Some of the most popular Java frameworks are:

- Spring
- Hibernate
- Grails
- Play
- JavaServer Faces (JSF)
- Google Web Toolkit (GWT)
- Quarkus

Spring

It is a light-weighted, powerful Java application development framework. It is used for JEE. Its other modules are **Spring Security, Spring MVC, Spring Batch, Spring ORM, Spring Boot** and **Spring Cloud** etc.

Advantages

- Loose coupling
- Lightweight
- Fast Development
- Powerful abstraction
- Easy to test

Hibernate

Hibernate is ORM (Object-Relation Mapping) framework that allows us establish communication between Java programming language and the RDBMS.

Advantages

- Portability, productivity, maintainability.
- Open source framework.
- It avoids repetitive code from the JDBC API.

Grails

It is a dynamic framework created by using the Groovy programming language. It is an OOPs language. Its purpose is to enhance the productivity. The syntax of Grails is

matched with Java and the code is compiled to JVM. It also works with Java, JEE, Spring and Hibernate.

Advantages

- It uses Groovy programming standard instead of Java programming standard because Groovy is similar to Java.
- Its object mapping feature is easy to use.
- It provides the facility of reusing the code (in the form of plugin) between different Grail applications.
- Provides flexible profiles.

Play

It is a unique Java framework because it does not follow JEE standards. It follows MVC architecture pattern. It is used when we want to develop highly scalable Java application. Using Play framework, we can develop lightweight and web-friendly Java application for both mobile and desktop.

Advantages

- No configuration is required.
- It enhances the developer productivity and target the RESTful architectures.
- It offers hot code reload and error message in the browser.
- It also supports popular IDEs.

JavaServer Faces

It stands for JavaServer Faces. It is a component-based UI framework developed by Oracle that is used to build user interfaces for Java-based applications. It follows MVC design pattern. The application developed using JSF has an architecture that defines a distinction

Advantages

- It is an important part of JEE.
- Provides rich libraries.

Google Web Toolkit (GWT)

It is an open-source framework that allows developers to write client-side Java code. With the help of GWT, we can rapidly develop complex browse application. The advantages to use GWT is that we can easily develop and debug Ajax application. the product of Google such as Google AdSense, Blogger are developed using GWT.

Advantages

- It employs reusability for web application development
- We can use Google API to develop application.
- It provides functionality such as, internationalization, UI abstraction, and history management.

Quarkus

It is a modern, **full-stack**, and **Kubernetes-native Java framework**. It offers small memory footprint and reduced boot time. It works well if the infrastructure is cloud-native. It optimizes Java specifically for Kubernetes and enables it to become an effective platform for serverless, cloud, and Kubernetes environments.

Advantages

- It is used in cloud, containers, and serverless environments.
- It supports microservices architecture and development.
- The developers are free to choose own development model.
- It is compatible with popular frameworks like, Eclipse, Spring Dependency Injection, and Hibernate.

Advantages of Java Frameworks

The advantages of the Java Frameworks are as follows:

- **Security:** It is the most important advantage of the Java framework. If we found any security loop hole or vulnerability in an application, we can directly move to the official website of the framework to fix the security related issues.
- **Support:** The widely used framework provides large forums or groups where we can put our problem for the solution. It also provides the documentation of the framework that helps us to understand the working of the framework.
- **Efficiency:** If we are doing a task without using the framework, it may take time to complete. But if we are doing the same work by using the framework, we can complete that task in easy and fast way. Therefore, using the Java framework development become faster, easier and effective. It also saves time and efforts.
- **Expenses:** Another advantage of using the framework is to reduce the cost of the application. Because the maintenance cost of the framework is low.

Examples of Frameworks in Java

In Java, **Collection** is an example of the framework. It reduces the programming efforts because it provides useful data structure and algorithms. It is referred to as library that do not provides inversion of control.

Another example of framework is, **Swing and AWT classes**. Swing is a GUI based framework used to develop windows-based application. It contains a large number of interfaces. There is inversion of control because of listeners.

Framework vs. Library

Library	Framework
Library is the collection of frequently used, pre-compiled classes.	Framework is the collection of libraries.
It is a set of reusable functions used by computer programs.	It is a piece of code that dictates the architecture of your project and aids in programs.
You are in full control when you call a method from a library and the control is then returned.	The code never calls into a framework, instead the framework calls you.
It is in corporate seamlessly into existing projects to add functionality that you can access using an API.	It cannot be seamlessly incorporated into an existing project. Instead it can be used when a new project is started.
They are important in program for linking and binding process.	They provide a standard way to build and deploy applications.
Libraries do no employ an inverted flow of control between itself and its clients.	Framework employs an inverted flow of control between itself and its clients.
Example: jQuery is a JavaScript library that simplifies DOM manipulation.	**Example:** Angular JS is a JavaScript-based framework for dynamic web applications.

Reference:

https://www3.ntu.edu.sg/home/ehchua/programming/java/JSPByExample.html