Servlets - Overview

What are Servlets?

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

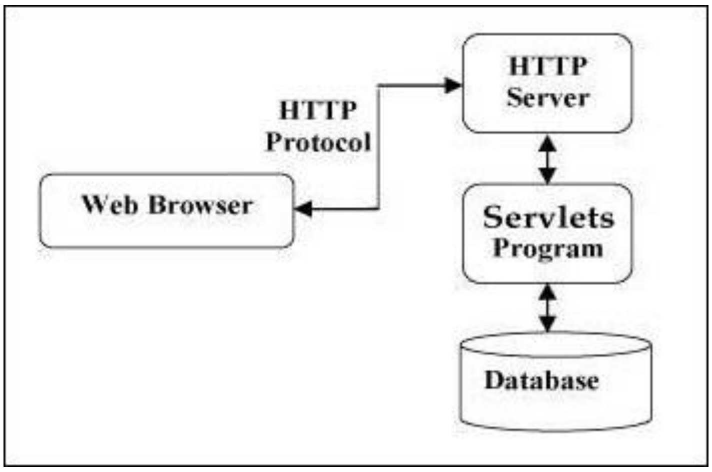
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.



Servlets Tasks

Servlets perform the following major tasks −

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Servlets Packages

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

Servlets - Life Cycle

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the **init()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in detail.

The init() Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this −

```
public void init() throws ServletException {
   // Initialization code...
}
```

The service() Method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method −

```
public void service(ServletRequest request, ServletResponse
response)
   throws ServletException, IOException {
}
```

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse
response)
   throws ServletException, IOException {
   // Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request,
HttpServletResponse response)
   throws ServletException, IOException {
   // Servlet code
}
```

The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

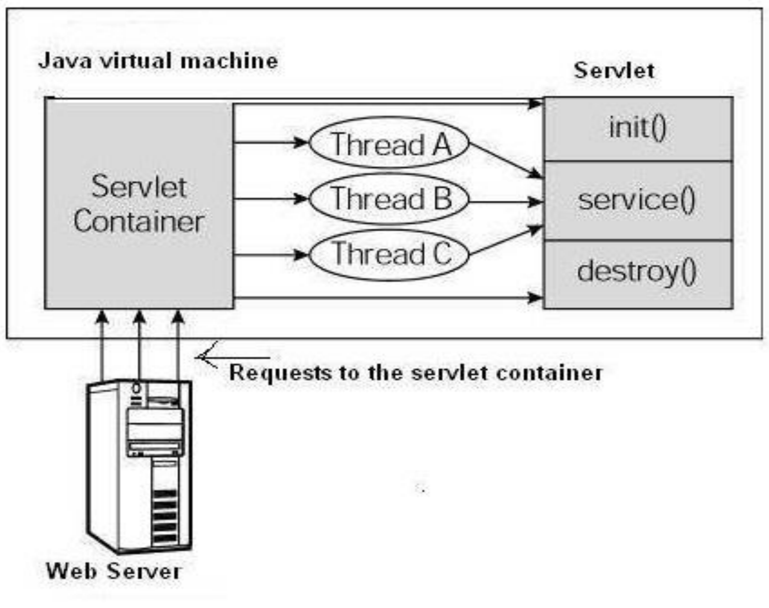
After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this −

```
public void destroy() {
   // Finalization code...
}
```

Architecture Diagram

The following figure depicts a typical servlet life-cycle scenario.

- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



Servlets - Examples

Servlets are Java classes which service HTTP requests and implement the **javax.servlet.Servlet** interface. Web application developers typically write servlets that extend javax.servlet.http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.

Sample Code

Following is the sample source code structure of a servlet example to show Hello World −

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

   private String message;

   public void init() throws ServletException {

      // Do required initialization
      message = "Hello World";
   }

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

      // Set response content type
      response.setContentType("text/html");

      // Actual logic goes here.
      PrintWriter out = response.getWriter();
      out.println("<h1>" + message + "</h1>");
   }

   public void destroy() {
      // do nothing.
   }
}
```

Compiling a Servlet

Let us create a file with name HelloWorld.java with the code shown above. Place this file at C:\ServletDevel (in Windows) or at /usr/ServletDevel (in Unix). This path location must be added to CLASSPATH before proceeding further.

Assuming your environment is setup properly, go in **ServletDevel** directory and compile HelloWorld.java as follows −

```
$ javac HelloWorld.java
```

If the servlet depends on any other libraries, you have to include those JAR files on your CLASSPATH as well. I have included only servlet-api.jar JAR file because I'm not using any other library in Hello World program.

This command line uses the built-in javac compiler that comes with the Sun Microsystems Java Software Development Kit (JDK). For this command to work properly, you have to include the location of the Java SDK that you are using in the PATH environment variable.

If everything goes fine, above compilation would produce **HelloWorld.class** file in the same directory. Next section would explain how a compiled servlet would be deployed in production.

Servlet Deployment

By default, a servlet application is located at the path <Tomcat-installationdirectory>/webapps/ROOT and the class file would reside in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in WEB-INF/classes/com/myorg/MyServlet.class.

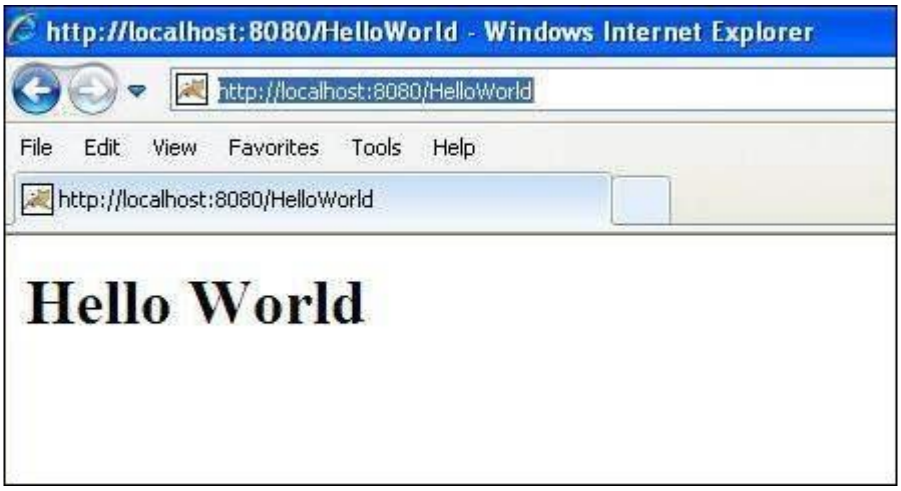
For now, let us copy HelloWorld.class into <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
   <servlet-name>HelloWorld</servlet-name>
   <servlet-class>HelloWorld</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>HelloWorld</servlet-name>
   <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be various entries in this table already available, but never mind.

You are almost done, now let us start tomcat server using <Tomcat-installationdirectory>\bin\startup.bat (on Windows) or <Tomcat-installationdirectory>/bin/startup.sh (on Linux/Solaris etc.) and finally type **http://localhost:8080/HelloWorld** in the browser's address box. If everything goes fine, you would get the following result


http://localhost:8080/HelloWorld - Windows Internet Explorer
http://localhost:8080/HelloWorld
File Edit View Favorites Tools Help
http://localhost:8080/HelloWorld

Hello World

Servlets - Form Data

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your backend program. The browser uses two methods to pass

this information to web server. These methods are GET Method and POST Method.

GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** (question mark) symbol as follows −

```
http://www.test.com/hello?key1 = value1&key2 = value2
```

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string.

This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using **doGet()** method.

POST Method

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using **doPost()** method.

Reading Form Data using Servlet

Servlets handles form data parsing automatically using the following methods depending on the situation −

- **getParameter()** − You call request.getParameter() method to get the value of a form parameter.
- **getParameterValues()** − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames()** − Call this method if you want a complete list of all parameters in the current request.

GET Method Example using URL

Here is a simple URL which will pass two values to HelloForm program using GET method.

http://localhost:8080/HelloForm?first_name = ZARA&last_name = ALI

Given below is the **HelloForm.java** servlet program to handle input given by web browser. We are going to use **getParameter()** method which makes it very easy to access passed information –

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

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

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Using GET Method to Read Form Data";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<ul>\n" +
                  "  <li><b>First Name</b>: "
                  + request.getParameter("first_name") + "\n" +
                  "  <li><b>Last Name</b>: "
                  + request.getParameter("last_name") + "\n" +
               "</ul>\n" +
            "</body>" +
         "</html>"
      );
   }
}
```

Assuming your environment is set up properly, compile HelloForm.java as follows −

```
$ javac HelloForm.java
```

If everything goes fine, above compilation would produce **HelloForm.class** file. Next you would have to copy this class file in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
   <servlet-name>HelloForm</servlet-name>
   <servlet-class>HelloForm</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>HelloForm</servlet-name>
   <url-pattern>/HelloForm</url-pattern>
```

```
</servlet-mapping>
```

Now type *http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI* in your browser's Location:box and make sure you already started tomcat server, before firing above command in the browser. This would generate following result −

Using GET Method to Read Form Data

- **First Name:** ZARA

- **Last Name:** ALI

GET Method Example Using Form

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this input.

```
<html>
   <body>
      <form action = "HelloForm" method = "GET">
         First Name: <input type = "text" name = "first_name">
         <br />
         Last Name: <input type = "text" name = "last_name" />
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

Keep this HTML in a file Hello.htm and put it in <Tomcat-installationdirectory>/webapps/ROOT directory. When you would access *http://localhost:8080/Hello.htm*, here is the actual output of the above form.

First Name: Last Name:

Try to enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

POST Method Example Using Form

Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is **HelloForm.java** servlet program to handle input given by web browser using GET or POST methods.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

   // Method to handle GET method request.
   public void doGet(HttpServletRequest request, 
HttpServletResponse response)
      throws ServletException, IOException {
      
      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Using GET Method to Read Form Data";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
         "transitional//en\">\n";
         
      out.println(docType +
         "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<ul>\n" +
                  "  <li><b>First Name</b>: "
                  + request.getParameter("first_name") + "\n" +
                  "  <li><b>Last Name</b>: "
                  + request.getParameter("last_name") + "\n" +
               "</ul>\n" +
            "</body>"
         "</html>"
      );
   }

   // Method to handle POST method request.
   public void doPost(HttpServletRequest request, 
HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
}
```

Now compile and deploy the above Servlet and test it using Hello.htm with the POST method as follows −

```
<html>
   <body>
      <form action = "HelloForm" method = "POST">
         First Name: <input type = "text" name = "first_name">
         <br />
         Last Name: <input type = "text" name = "last_name" />
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

Here is the actual output of the above form, Try to enter First and Last Name and then click submit button to see the result on your local machine where tomcat is running.

First Name: ☐ Last Name: ☐

Based on the input provided, it would generate similar result as mentioned in the above examples.

Passing Checkbox Data to Servlet Program

Checkboxes are used when more than one option is required to be selected.

Here is example HTML code, CheckBox.htm, for a form with two checkboxes

```
<html>
   <body>
      <form action = "CheckBox" method = "POST" target = 
"_blank">
         <input type = "checkbox" name = "maths" checked = 
"checked" /> Maths
         <input type = "checkbox" name = "physics"  /> Physics
         <input type = "checkbox" name = "chemistry" checked = 
"checked" />
                                          Chemistry
         <input type = "submit" value = "Select Subject" />
      </form>
   </body>
</html>
```

The result of this code is the following form

☑ Maths ☐ Physics ☑ Chemistry

Given below is the CheckBox.java servlet program to handle input given by web browser for checkbox button.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class CheckBox extends HttpServlet {

   // Method to handle GET method request.
   public void doGet(HttpServletRequest request, 
HttpServletResponse response)
      throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Reading Checkbox Data";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<ul>\n" +
                  "  <li><b>Maths Flag : </b>: "
                  + request.getParameter("maths") + "\n" +
                  "  <li><b>Physics Flag: </b>: "
                  + request.getParameter("physics") + "\n" +
                  "  <li><b>Chemistry Flag: </b>: "
                  + request.getParameter("chemistry") + "\n" +
               "</ul>\n" +
            "</body>"
         "</html>"
      );
   }

   // Method to handle POST method request.
   public void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
}
```

For the above example, it would display following result −

```
           Reading Checkbox Data

  • Maths Flag : : on

  • Physics Flag: : null

  • Chemistry Flag: : on
```

Reading All Form Parameters

Following is the generic example which uses **getParameterNames()** method of HttpServletRequest to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order

Once we have an Enumeration, we can loop down the Enumeration in standard way by, using *hasMoreElements()* method to determine when to stop and using *nextElement()* method to get each parameter name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class ReadParams extends HttpServlet {

   // Method to handle GET method request.
   public void doGet(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Reading All Form Parameters";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n" +
         "<body bgcolor = \"#f0f0f0\">\n" +
         "<h1 align = \"center\">" + title + "</h1>\n" +
         "<table width = \"100%\" border = \"1\" align =
\"center\">\n" +
         "<tr bgcolor = \"#949494\">\n" +
            "<th>Param Name</th>"
            "<th>Param Value(s)</th>\n"+
         "</tr>\n"
      );

      Enumeration paramNames = request.getParameterNames();

      while(paramNames.hasMoreElements()) {
         String paramName = (String)paramNames.nextElement();
         out.print("<tr><td>" + paramName + "</td>\n<td>");
         String[] paramValues =
request.getParameterValues(paramName);
```

```
         // Read single valued data
         if (paramValues.length == 1) {
            String paramValue = paramValues[0];
            if (paramValue.length() == 0)
               out.println("<i>No Value</i>");
               else
               out.println(paramValue);
         } else {
            // Read multiple valued data
            out.println("<ul>");

            for(int i = 0; i < paramValues.length; i++) {
               out.println("<li>" + paramValues[i]);
            }
            out.println("</ul>");
         }
      }
      out.println("</tr>\n</table>\n</body></html>");
   }

   // Method to handle POST method request.
   public void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
}
```

Now, try the above servlet with the following form −

```
<html>
   <body>
      <form action = "ReadParams" method = "POST" target =
"_blank">
         <input type = "checkbox" name = "maths" checked =
"checked" /> Maths
         <input type = "checkbox" name = "physics"  /> Physics
         <input type = "checkbox" name = "chemistry" checked =
"checked" /> Chem
         <input type = "submit" value = "Select Subject" />
      </form>
   </body>
</html>
```

Now calling servlet using the above form would generate the following result −

Reading All Form Parameters

Param Name	Param Value(s)
maths	on

chemistry	on

You can try the above servlet to read any other form's data having other objects like text box, radio button or drop down box etc.

Servlets - Session Tracking

HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

Still there are following three ways to maintain session between web client and web server −

Cookies

A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the recieved cookie.

This may not be an effective way because many time browser does not support a cookie, so I would not recommend to use this procedure to maintain the sessions.

Hidden Form Fields

A web server can send a hidden HTML form field along with a unique session ID as follows −

```
<input type = "hidden" name = "sessionid" value = "12345">
```

This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or POST data. Each time when web browser sends request back, then session_id value can be used to keep the track of different web browsers.

This could be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

URL Rewriting

You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session.

For example, with http://tutorialspoint.com/file.htm;sessionid = 12345, the session identifier is attached as sessionid = 12345 which can be accessed at the web server to identify the client.

URL rewriting is a better way to maintain sessions and it works even when browsers don't support cookies. The drawback of URL re-writing is that you would have to generate every URL dynamically to assign a session ID, even in case of a simple static HTML page.

The HttpSession Object

Apart from the above mentioned three ways, servlet provides HttpSession Interface which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user.

You would get HttpSession object by calling the public method **getSession()** of HttpServletRequest, as below −

```
HttpSession session = request.getSession();
```

You need to call *request.getSession()* before you send any document content to the client. Here is a summary of the important methods available through HttpSession object −

Sr.No.	Method & Description
1	**public Object getAttribute(String name)** This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
2	**public Enumeration getAttributeNames()** This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
3	**public long getCreationTime()** This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
4	**public String getId()** This method returns a string containing the unique identifier assigned to this session.
5	**public long getLastAccessedTime()**

	This method returns the last accessed time of the session, in the format of milliseconds since midnight January 1, 1970 GMT
6	**public int getMaxInactiveInterval()** This method returns the maximum time interval (seconds), that the servlet container wil keep the session open between client accesses.
7	**public void invalidate()** This method invalidates this session and unbinds any objects bound to it.
8	**public boolean isNew(** This method returns true if the client does not yet know about the session or if the clien chooses not to join the session.
9	**public void removeAttribute(String name)** This method removes the object bound with the specified name from this session.
10	**public void setAttribute(String name, Object value)** This method binds an object to this session, using the name specified.
11	**public void setMaxInactiveInterval(int interval)** This method specifies the time, in seconds, between client requests before the servle container will invalidate this session.

Session Tracking Example

This example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session. We would associate a new session with the request if one does not already exist.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class SessionTrack extends HttpServlet {

   public void doGet(HttpServletRequest request,
HttpServletResponse response)
```

```
      throws ServletException, IOException {

      // Create a session object if it is already not  created.
      HttpSession session = request.getSession(true);

      // Get session creation time.
      Date createTime = new Date(session.getCreationTime());

      // Get last access time of this web page.
      Date lastAccessTime = new
Date(session.getLastAccessedTime());

      String title = "Welcome Back to my website";
      Integer visitCount = new Integer(0);
      String visitCountKey = new String("visitCount");
      String userIDKey = new String("userID");
      String userID = new String("ABCD");

      // Check if this is new comer on your web page.
      if (session.isNew()) {
         title = "Welcome to my website";
         session.setAttribute(userIDKey, userID);
      } else {
         visitCount =
(Integer)session.getAttribute(visitCountKey);
         visitCount = visitCount + 1;
         userID = (String)session.getAttribute(userIDKey);
      }
      session.setAttribute(visitCountKey,  visitCount);

      // Set response content type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
         "transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +

            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<h2 align = \"center\">Session Infomation</h2>\n"
+
               "<table border = \"1\" align = \"center\">\n" +

                  "<tr bgcolor = \"#949494\">\n" +
                     "  <th>Session info</th><th>value</th>
                  </tr>\n" +

                  "<tr>\n" +
                     "  <td>id</td>\n" +
                     "  <td>" + session.getId() + "</td>
                  </tr>\n" +

                  "<tr>\n" +
                     "  <td>Creation Time</td>\n" +
                     "  <td>" + createTime + "  </td>
                  </tr>\n" +

                  "<tr>\n" +
                     "  <td>Time of Last Access</td>\n" +
                     "  <td>" + lastAccessTime + "  </td>
                  </tr>\n" +

                  "<tr>\n" +
                     "  <td>User ID</td>\n" +
                     "  <td>" + userID + "  </td>
                  </tr>\n" +

                  "<tr>\n" +
                     "  <td>Number of visits</td>\n" +
                     "  <td>" + visitCount + "</td>
                  </tr>\n" +
               "</table>\n" +
            "</body>
         </html>"
      );
   }
}
```

Compile the above servlet **SessionTrack** and create appropriate entry in web.xml file. Now running *http://localhost:8080/SessionTrack* would display the following result when you would run for the first time −

Welcome to my website

Session Infomation

Session info	**value**
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010

Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	0

Now try to run the same servlet for second time, it would display following result.

Welcome Back to my website

Session Infomation

info type	value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	1

Deleting Session Data

When you are done with a user's session data, you have several options −

- **Remove a particular attribute** − You can call *public void removeAttribute(String name)* method to delete the value associated with a particular key.
- **Delete the whole session** − You can call *public void invalidate()* method to discard an entire session.

- **Setting Session timeout −** You can call *public void setMaxInactiveInterval(int interval)* method to set the timeout for a session individually.
- **Log the user out −** The servers that support servlets 2.4, you can call **logout** to log the client out of the Web server and invalidate all sessions belonging to all the users.
- **web.xml Configuration −** If you are using Tomcat, apart from the above mentioned methods, you can configure session time out in web.xml file as follows.

```
<session-config>
   <session-timeout>15</session-timeout>
</session-config>
```

The timeout is expressed as minutes, and overrides the default timeout which is 30 minutes in Tomcat.

The getMaxInactiveInterval( ) method in a servlet returns the timeout period for that session in seconds. So if your session is configured in web.xml for 15 minutes, getMaxInactiveInterval( ) returns 900.

Servlets - Database Access

This tutorial assumes you have understanding on how JDBC application works. Before starting with database access through a servlet, make sure you have proper JDBC environment setup along with a database.

For more detail on how to access database using JDBC and its environment setup you can go through our JDBC Tutorial.

To start with basic concept, let us create a simple table and create few records in that table as follows −

Create Table

To create the **Employees** table in TEST database, use the following steps −

Step 1

Open a **Command Prompt** and change to the installation directory as follows −

```
C:\>
C:\>cd Program Files\MySQL\bin
C:\Program Files\MySQL\bin>
```

Step 2

Login to database as follows

```
C:\Program Files\MySQL\bin>mysql -u root -p
Enter password: ********
mysql>
```

Step 3

Create the table **Employee** in **TEST** database as follows −

```
mysql> use TEST;
mysql> create table Employees (
   id int not null,
   age int not null,
   first varchar (255),
   last varchar (255)
);
Query OK, 0 rows affected (0.08 sec)
mysql>
```

Create Data Records

Finally you create few records in Employee table as follows −

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
Query OK, 1 row affected (0.00 sec)

mysql>
```

Accessing a Database

Here is an example which shows how to access TEST database using Servlet.

```
// Loading required libraries
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class DatabaseAccess extends HttpServlet{

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

      // JDBC driver name and database URL
      static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
      static final String DB_URL="jdbc:mysql://localhost/TEST";

      //  Database credentials
      static final String USER = "root";
      static final String PASS = "password";
      // Set response content type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Database Result";

      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n" +
         "<body bgcolor = \"#f0f0f0\">\n" +
         "<h1 align = \"center\">" + title + "</h1>\n");
      try {
         // Register JDBC driver
         Class.forName("com.mysql.jdbc.Driver");

         // Open a connection
         Connection conn = DriverManager.getConnection(DB_URL,
USER, PASS);

         // Execute SQL query
         Statement stmt = conn.createStatement();
         String sql;
         sql = "SELECT id, first, last, age FROM Employees";
         ResultSet rs = stmt.executeQuery(sql);

         // Extract data from result set
         while(rs.next()){
            //Retrieve by column name
            int id  = rs.getInt("id");
            int age = rs.getInt("age");
            String first = rs.getString("first");
            String last = rs.getString("last");

            //Display values
            out.println("ID: " + id + "<br>");
            out.println(", Age: " + age + "<br>");
            out.println(", First: " + first + "<br>");
            out.println(", Last: " + last + "<br>");
         }
         out.println("</body></html>");

         // Clean-up environment
         rs.close();
         stmt.close();
         conn.close();
      } catch(SQLException se) {
         //Handle errors for JDBC
         se.printStackTrace();
      } catch(Exception e) {
         //Handle errors for Class.forName
         e.printStackTrace();
      } finally {
         //finally block used to close resources
         try {
            if(stmt!=null)
               stmt.close();
         } catch(SQLException se2) {
         } // nothing we can do
         try {
            if(conn!=null)
            conn.close();
         } catch(SQLException se) {
            se.printStackTrace();
         } //end finally try
      } //end try
   }
}
```

Now let us compile above servlet and create following entries in web.xml

```
....
<servlet>
   <servlet-name>DatabaseAccess</servlet-name>
   <servlet-class>DatabaseAccess</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>DatabaseAccess</servlet-name>
   <url-pattern>/DatabaseAccess</url-pattern>
</servlet-mapping>
....
```

Now call this servlet using URL http://localhost:8080/DatabaseAccess which would display following response −

```
                    Database Result

ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
```

Example2

Example of Fetching Result for the given rollno

In this example, we have create three files

- index.html

- Search.java
- web.xml

---

index.html

This page gets rollno from the user and forwards this data to servlet which is responsible to show the records based on the given rollno.

```
<html>
<body>
<form action="servlet/Search">
Enter your Rollno:<input type="text" name="roll"/><br/>

<input type="submit" value="search"/>
</form>
</body>
</html>
```

Search.java

This is the servlet file which gets the input from the user and maps this data with the database and prints the record for the matched data. In this page, we are displaying the column name of the database along with data, so we are using ResultSetMetaData interface.

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

public class Search extends HttpServlet {

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

response.setContentType("text/html");
PrintWriter out = response.getWriter();

String rollno=request.getParameter("roll");
int roll=Integer.valueOf(rollno);

try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("select * from result where rollno=?");
ps.setInt(1,roll);

out.print("<table width=50% border=1>");
out.print("<caption>Result:</caption>");

ResultSet rs=ps.executeQuery();

/* Printing column names */
ResultSetMetaData rsmd=rs.getMetaData();
int total=rsmd.getColumnCount();
out.print("<tr>");
for(int i=1;i<=total;i++)
{
out.print("<th>"+rsmd.getColumnName(i)+"</th>");
}

out.print("</tr>");

/* Printing result */

while(rs.next())
{
out.print("<tr><td>"+rs.getInt(1)+"</td><td>"+rs.getString(2)+"
</td><td>"+rs.getString(3)+"</td><td>"+rs.getString(4)+"</td></tr>");

}

out.print("</table>");

}catch (Exception e2) {e2.printStackTrace();}

finally{out.close();}

}
}
```

web.xml file

This is the configuration file which provides information of the servlet to the container.

```
<web-app>
```

```

<servlet>
<servlet-name>Search</servlet-name>
<servlet-class>Search</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>Search</servlet-name>
<url-pattern>/servlet/Search</url-pattern>
</servlet-mapping>

</web-app>
```