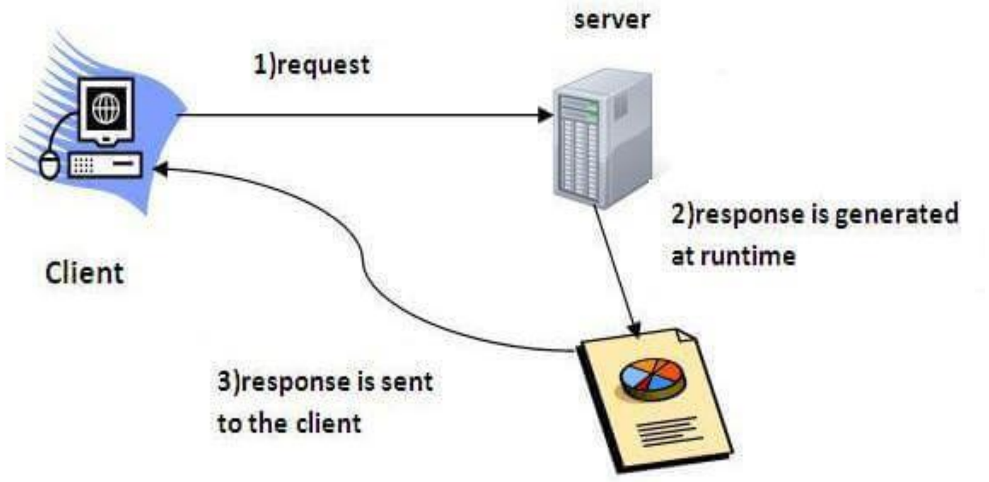
INTRODUCTION TO SERVLETS

3.1 Introduction to Servlets

What is a web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.


server
1)request
2)response is generated at runtime
Client
3)response is sent to the client

Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

What is a Servlet?

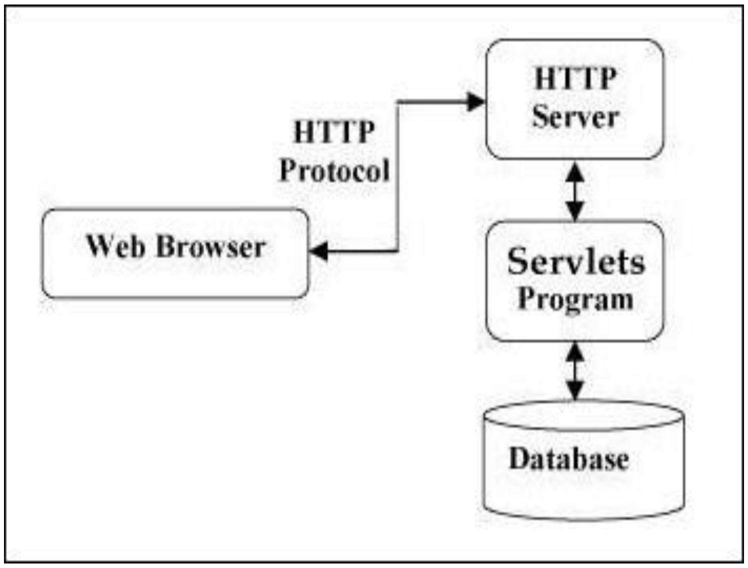
- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.
- Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.

Properties of Servlets are :

- Servlets work on the server-side.
- Servlets are capable of handling complex requests obtained from the webserver.

Execution of Servlets basically involves six basic steps:

1. The clients send the request to the webserver.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generates the response in the form of output.
5. The servlet sends the response back to the webserver.
6. The web server sends the response back to the client and the client browser displays it on the screen.



Why do we need For Server-Side extensions?

- The **server-side extensions** are nothing but the technologies that are used to create dynamic Web pages.
- Actually, to provide the facility of dynamic Web pages, Web pages need a container or Web server.
- To meet this requirement, independent Web server providers offer some proprietary solutions in the form of **APIs**(Application Programming Interface).
- These **APIs** allow us to build programs that can run with a Web server.
- **Java Servlet** is also one of the component APIs of Java Platform Enterprise Edition which sets standards for creating dynamic Web applications in Java.

3.2 The Servlet API

The following packages represent interfaces and classes for servlet API.

1. javax.servlet
2. javax.servlet.http

✓ **javax.servlet package:** There are many interfaces and classes in javax.servlet package. They are as follows:

Interfaces	Classes
1. Servlet	1. GenericServlet
2. ServletRequest	2. ServletInputStream
3. ServletResponse	3. ServletOutputStream
4. RequestDispatcher	4. ServletRequestWrapper
5. ServletConfig	5. ServletResponseWrapper
6. ServletContext	6. ServletRequestEvent
7. SingleThreadModel	7. ServletContextEvent
8. Filter	8. ServletRequestAttributeEvent
9. FilterConfig	9. ServletContextAttributeEvent
10. FilterChain	10. ServletException
11. ServletRequestListener	11. UnavailableException
12. ServletRequestAttributeListener	
13. ServletContextListener	
14. ServletContextAttributeListener	

✓ **javax.servlet.http package**

There are many interfaces and classes in javax.servlet.http package. They are as follows:

Interfaces	Classes
1. HttpServletRequest	1. HttpServlet
2. HttpServletResponse	2. Cookie
3. HttpSession	3. HttpServletRequestWrapper
4. HttpSessionListener	4. HttpServletResponseWrapper
5. HttpSessionAttributeListener	5. HttpSessionEvent
6. HttpSessionBindingListener	6. HttpSessionBindingEvent
7. HttpSessionActivationListener	
8. HttpSessionContext (deprecated now)	

The Servlet Container

- **Servlet container**, also known as **Servlet engine** is an integrated set of objects that provide a run time environment for Java Servlet components.

Servlet Types:

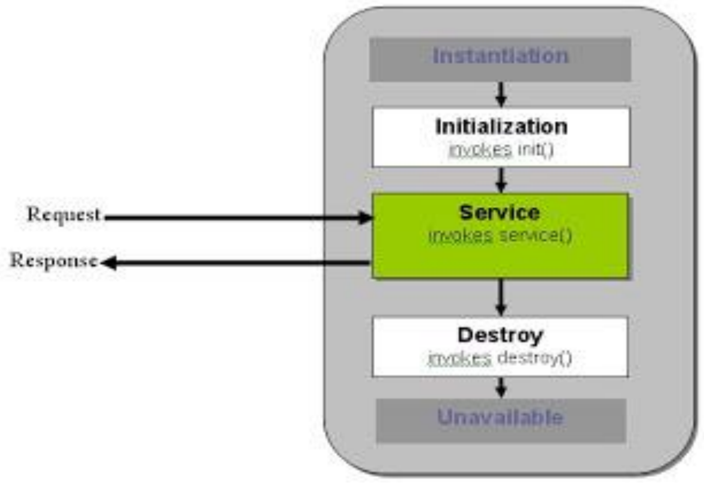
- Generic servlets
- HTTP servlets

GenericServlet	HttpServlet
It is defined by javax.servlet package.	It is defined by javax.servlethttp package.
It describes protocol-independent servlet	It describes protocol-dependent servlet.
GenericServiet is not dependent on any particular protocol. It can be used with any protocol such as HTTP, SMTP, FTP, and so on.	HttpServlet is a dependent protocol and is only used with HTTP protocol.
service( ) method is an abstract method.	service( ) can be replaced by doGet( ) or doPost( ) methods.
GenericServlet doesn't allow session management with cookies and HTTP sessions.	HTTPServlet allows session management with cookies and HTTP sessions.
GenericServlet is a superclass of HttpServlet class.	HttpServlet is a subclass of GenericServlet class.
It forwards and includes a request and is also possible to redirect a request.	It forwards and includes a request but it is not possible to redirect the request.

3.2 Lifecycle of a Servlet:

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.



1. 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.

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

2. 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 doGe, doPost, doPut, doDelete, etc. methods as appropriate.

The doGet( ) and doPost( ) are most frequently used methods within each service request.

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
}
```

3. 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...}
```

3.3 Deploying a servlet:

There are given 6 steps to create a servlet example. These steps are required for all the servers.

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

1. Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and responds to the client.

The Sun Microsystems defines a unique standard to be followed by all the server vendors.

Let's see the directory structure that must be followed to create the servlet

```
P1
|
|------ WEB-INF   ┌── classes
|                 └── Xml files
|
└------ .html files
```

As you can see that the servlet class file must be in the classes' folder. The web.xml file must be under the WEB-INF folder.

2. Create Servlet:

There are three ways to create the servlet.

- ✓ By implementing the Servlet interface
- ✓ By inheriting the GenericServlet class
- ✓ By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

DemoServlet.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)  throws ServletException,IOE
xception
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data
pw.println("<html><body>");    //writing html in the stream
pw.println("Welcome to servlet");
pw.println("</body></html>");
pw.close( );//closing the stream
} }
```

3. Compile the Servlet:

Compile java program: javac Demoservlet.java

and place DemoServlet.class file in P1/WEB-INF/classes

4. Create the Deployment Descriptor

A *deployment descriptor* is an optional component in a servlet application, taking the form of an XML document called web.xml. The descriptor must be located in the WEB-INF directory of the servlet application.

```
<web-app>
 <servlet>
  <servlet-name>s1</servlet-name>
  <servlet-class>FirstServlet</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>s1</servlet-name>
  <url-pattern>/hello</url-pattern>
 </servlet-mapping>
 </web-app>
```

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

5. How to deploy the servlet project and Run tomcat server

Copy the project and paste it in the webapps folder under apache tomcat. Run Tomcat

If it is not already running, you need to start Tomcat.

6. Call Your Servlet from a Web Browser

You are ready to call your servlet from a web browser. By default, Tomcat runs on port 8080 in P1 virtual directory under the servlet subdirectory. http://localhost:8080/P1/hello

3.3 Generic Servlet Example

First.java

```
import java.io.*;
import javax.servlet.*;
public class First extends GenericServlet{
public void service(ServletRequest req,ServletResponse res)
throws IOException,ServletException{
res.setContentType("text/html");
PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello generic servlet</b>");
out.print("</body></html>");
}
}
```

web.xml

```
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>First</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/hello</url-pattern>
</servlet-mapping>
</web-app>
```

3.4 HTTP Servlet Example

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res) throws
ServletException,IOException
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data
pw.println("<html><body>");    //writing html in the stream
pw.println("Welcome to servlet");
pw.println("</body></html>");
pw.close();//closing the stream
} }
```

web.xml

```
<web-app>
 <servlet>
  <servlet-name>s1</servlet-name>
  <servlet-class>DemoServlet</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>s1</servlet-name>
  <url-pattern>/hello</url-pattern>
 </servlet-mapping>
 </web-app>
```

3.5 Reading Form Parameters

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.

Example:

“sum.html”

```
<html>
<body> <form action="./First">
Enter n1 Value:<input type="text" name="t1"/><br>
Enter n2 Value:<input type="text" name="t2"/><br>
<input type="submit" value="click me"/>
</form>
```

```
</body>
</html>
```

“web.xml”

```
<web-app>
 <welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>First</servlet-name>
<servlet-class>First</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>First</servlet-name>
<url-pattern>/First</url-pattern>
</servlet-mapping>
</web-app>
```

“First.java”

```
import java.io.IOException;
import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class First extends GenericServlet {
public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
int a=Integer.parseInt(request.getParameter("t1"));
int b=Integer.parseInt(request.getParameter("t2"));
int c=a+b;
response.getWriter().print("Sum="+c);
}
}
```

3.5 Reading Servlet parameters & Reading initialization parameters

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

Syntax :

```
<web-app>
  <servlet>
   ......
    <init-param>
    <param-name>parametername</param-name>
    <param-value>parametervalue</param-value>
   </init-param>
   ......
  </servlet>
</web-app>
```

ServletConfig to get initialization parameter.In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

DemoServletInit.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DemoServletInit extends HttpServlet
{
  public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException
  {
  response.setContentType("text/html");
  PrintWriter PW = response.getWriter();
  ServletConfig config=getServletConfig();
  String name=config.getInitParameter("name");
  pw.print("Name is: "+name);
  pw.close();
  }
}
```

Here web.xml is updated as following

```
<servlet>
   <servlet-name>D</servlet-name>
   <servlet-class>DemoServletInit</servlet-class>
<init-param>
<param-name>name</param-name>
<param-value>Ricky</param-value>
</init-param>
</servlet>
<servlet-mapping>
   <servlet-name>D</servlet-name>
   <url-pattern>/Demo</url-pattern>
</servlet-mapping>
```

3.6 Handling Http request & 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( ). The GET and POST requests are commonly used when handling form input.

Handling HTTP GET Requests:

The following programs are for servlets that handles HTTP GET request. The servlet is invoked when a form on a Web page is submitted.

Get.html:

```
<html><body><center>
<form name="Form1"   action="http://localhost:8080/servlet/GS">
<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>
```

GetServlet.java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GetServlet 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();
        }
}
```

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 Post.html and a servlet is defined in PostServlet.java. The HTML source code for Post.html is shown in the following listing. It is identical toPost.html except that the method parameter for the form tag explicitly specifies that the POST method should be used, and the action parameter for the form tag specifies a different servlet.

Post.html:

```
<html><body><center>
<form name="Form1" method="post" action="http://localhost:8080/servlet/PS">
<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>
```

PostServlet.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();
	}
}
```

3.8 Using Cookies and Sessions

Cookies are usually small text files, given ID tags that are stored on your computer's browser directory or program data subfolders. Cookies are created when you use your browser to visit a website that uses cookies to keep track of your movements within the site, help you resume where you left off, remember your registered login, theme selection, preferences, and other customization functions.The website stores a corresponding file(with same ID tag)to the one they set in your browser and in this file they can track and keep information on your movements within the site and any information you may have voluntarily given while visiting the website, such as email address.

There are two types of cookies: session cookies and persistent cookies. Session cookies are created temporarily in your browser's subfolder while you are visiting a website. Once you leave the site, the session cookie is deleted. On the other hand, persistent cookie files remain in your browser's subfolder and are activated again once you visit the website that created that particular cookie. A persistent cookie remains in the browser's subfolder for the duration period set within the cookie's file.

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
Home.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.

The HTML source code for AddCookie.html is shown in the following listing. This page contains a text field in which a value can be entered. There is also a submit button on the page. When this button is pressed, the value in the text field is sent to AddCookieServlet via an HTTP POST request.

<u>“Home.html”</u>

```
<html><body><center>
<form action="addCookie"><pre>
Name:<input type="text" name="cname"/><br><br>
Value:<input type="text" name="cvalue"/><br>
<input type="submit" value="AddCookie"/>
</pre></form></center></body></html>
```

<u>“AddCookieServlet.java”</u>

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class AddCookieServlet extends HttpServlet
{	public void service(HttpServletRequest req,HttpServletResponse res)throws
	ServletException,IOException
	{	String name=req.getParameter("cname");
		String value=req.getParameter("cvalue");
		Cookie c=new Cookie(name,value);
		res.addCookie(c);
		res.setContentType("text/html");
		PrintWriter out=res.getWriter();
		String output="<html><body>";
		output+="Cookie Added Successfully <br/>";
		output+="<a href='viewcookies'>View Cookies</a>";
		output+="</body></html>";
		out.println(output);
	}
}
```

“GetCookieServlet.java”

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class GetCookieServlet extends HttpServlet
{	public void service(HttpServletRequest req,HttpServletResponse res)throws
	ServletException,IOException
	{	res.setContentType("text/html");
		PrintWriter out=res.getWriter();
		out.println("<html><body><table border=1>");
		out.println("<tr><th>Name</th><th>Value</th></tr>");
		Cookie []c=req.getCookies();
		if(c!=null)
		{	for(int i=0;i<c.length;i++)
			{	out.println("<tr><td>"+c[i].getName()+"</td>");
				out.println("<td>"+c[i].getValue()+"</td></tr>");
			}
		}
		out.println("</table><br><br>");
		out.println("</body></html>");
	}
}
```

“web.xml”

```
<?xml version="1.0"?>
<web-app>
	<servlet>
		<servlet-name>AddCookieServlet</servlet-name>
		<servlet-class>AddCookieServlet</servlet-class>
	</servlet>
	<servlet-mapping>
```

```
		<servlet-name>AddCookieServlet</servlet-name>
		<url-pattern>/addCookie</url-pattern>
	</servlet-mapping>
	<servlet>
		<servlet-name>GetCookieServlet</servlet-name>
		<servlet-class>GetCookieServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>GetCookieServlet</servlet-name>
		<url-pattern>/viewcookies</url-pattern>
	</servlet-mapping>
	<welcome-file-list>
		<welcome-file>htmlfiles/Home.html</welcome-file>
	</welcome-file-list>
</web-app>
```

Session Tracking Techniques

There are four techniques used in Session tracking:

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. HttpSession

Example for Hidden Form Field:

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
 public class FirstServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response){
    try{
```

```
                    response.setContentType("text/html");
                   PrintWriter out = response.getWriter();
                    String n=request.getParameter("userName");
                     out.print("Welcome "+n);
                    //creating form that have invisible textfield
                     out.print("<form action='servlet2'>");
                    out.print("<input type='hidden' name='uname' value='"+n+"'>");
                    out.print("<input type='submit' value='go'>");
                   out.print("</form>");
                   out.close();
                    }
                  catch(Exception e){
                  System.out.println(e);}
                }
        }
```

SecondServlet.java

```
    import java.io.*;
    import javax.servlet.*;
    import javax.servlet.http.*;
    public class SecondServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        try{
                  response.setContentType("text/html");
                   PrintWriter out = response.getWriter();
                     //Getting the value from the hidden field
                   String n=request.getParameter("uname");
                   out.print("Hello "+n);
                   out.close();
                  }
                 catch(Exception e){
                 System.out.println(e);
                 }
        }
}
```

web.xml

```
<web-app>
 <servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
 <servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
 <servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
 <servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
 </web-app>
```

Example of using URL Rewriting: In this example, we are maintaining the state of the user using link. For this purpose, we are appending the name of the user in the query string and getting the value from the query string in another page.

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response){
     try{
```

```
                    response.setContentType("text/html");
                  PrintWriter out = response.getWriter();
                   String n=request.getParameter("userName");
                  out.print("Welcome "+n);
                  out.print("<a href='servlet2?uname="+n+"'>visit</a>");
                  out.close();
              }
          catch(Exception e){
                  System.out.println(e);
          }     } }
```

SecondServlet.java

```
    import java.io.*;
    import javax.servlet.*;
    import javax.servlet.http.*;
     public class SecondServlet extends HttpServlet {
     public void doGet(HttpServletRequest request, HttpServletResponse response)
         try{
                    response.setContentType("text/html");
                     PrintWriter out = response.getWriter();
                    //getting value from the query string
                    String n=request.getParameter("uname");
                   out.print("Hello "+n);
                    out.close();
               }
              catch(Exception e)
              {
                  System.out.println(e);
              }     }      }
```

web.xml

```
    <web-app>
     <servlet>
    <servlet-name>s1</servlet-name>
    <servlet-class>FirstServlet</servlet-class>
    </servlet>
```

```
 <servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
 <servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
 <servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
 </web-app>
```

How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

Example of using HttpSession: In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the setAttribute() method of HttpSession interface and to get the attribute, we have used the getAttribute method.

index.html

```
<html><body>
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
</body></html>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
{
    try{
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                String n=request.getParameter("userName");
                out.print("Welcome "+n);
                HttpSession session=request.getSession();
                session.setAttribute("uname",n);
                out.print("<a href='servlet2'>visit</a>");
               out.close();
        }
        catch(Exception e){
            System.out.println(e);
        }
     }
}
```

SecondServlet.java

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

```
  public class SecondServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
{
                response.setContentType("text/html");
                 PrintWriter out = response.getWriter();
                 HttpSession session=request.getSession(false);
                String n=(String)session.getAttribute("uname");
                  out.print("Hello "+n);
                   out.close();
            }
}
```

web.xml

```
<web-app>
  <servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
  <servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
  <servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
  <servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
  </web-app>
```