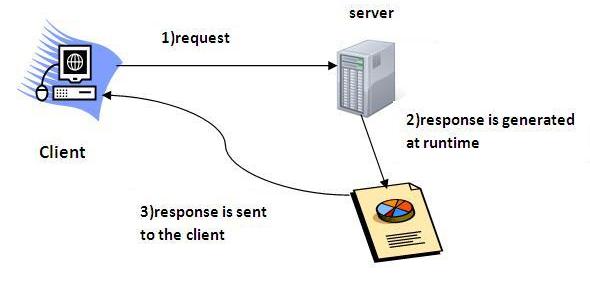
Servlet is a technology i.e used to create web application which provides many interfaces and classes. There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.



### CGI(Commmon Gateway Interface)

Disadvantages of CGI

There are many problems in CGI technology:

1. If number of client’s increases, it takes more time for sending response.
2. For each request, it starts a process and Web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

### problem in cgi and how servlet is better

There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

1. **better performance:** because it creates a thread for each request not process.
2. **Portability:** because it uses java language.
3. **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
4. **Secure:** because it uses java language..

|  |  |
| --- | --- |
| GET | POST |
| 1) In case of Get request, only limited amount of data can be sent because data is sent in header. | In case of post request, large amount of data can be sent because data is sent in body. |
| 2) Get request is not secured because data is exposed in URL bar. | Post request is secured because data is not exposed in URL bar. |
| 3) Get request can be bookmarked. | Post request cannot be bookmarked. |
| 4) Get request is idempotent. It means second request will be ignored until response of first request is delivered | Post request is non-idempotent. |
| 5) Get request is more efficient and used more than Post. | Post request is less efficient and used less than get. |

### Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **public void init(ServletConfig config)** | initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once. |
| **public void service(ServletRequest request,ServletResponse response)** | provides response for the incoming request. It is invoked at each request by the web container. |
| **public void destroy()** | is invoked only once and indicates that servlet is being destroyed. |
| **public ServletConfig getServletConfig()** | returns the object of ServletConfig. |
| **public String getServletInfo()** | returns information about servlet such as writer, copyright, version etc. |

# **Life Cycle of a Servlet**

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.

# **Steps to create a servlet example**

* By implementing Servlet interface,
* By inheriting GenericServlet class, (or)
* By inheriting HttpServlet class

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
4. **public** **class** First **implements** Servlet{
5. ServletConfig config=**null**;
7. **public** **void** init(ServletConfig config){
8. **this**.config=config;
9. System.out.println("servlet is initialized");
10. }
12. **public** **void** service(ServletRequest req,ServletResponse res)
13. **throws** IOException,ServletException{
15. res.setContentType("text/html");
17. PrintWriter out=res.getWriter();
18. out.print("<html><body>");
19. out.print("<b>hello simple servlet</b>");
20. out.print("</body></html>");
22. }
23. **public** **void** destroy(){System.out.println("servlet is destroyed");}
24. **public** ServletConfig getServletConfig(){**return** config;}
25. **public** String getServletInfo(){**return** "copyright 2007-1010";}
27. }
28. **import** java.io.\*;
29. **import** javax.servlet.\*;
31. **public** **class** First **extends** GenericServlet{
32. **public** **void** service(ServletRequest req,ServletResponse res)
33. **throws** IOException,ServletException{
35. res.setContentType("text/html");
37. PrintWriter out=res.getWriter();
38. out.print("<html><body>");
39. out.print("<b>hello generic servlet</b>");
40. out.print("</body></html>");
42. }
43. }
44. **import** javax.servlet.http.\*;
45. **import** javax.servlet.\*;
46. **import** java.io.\*;
47. **public** **class** DemoServlet **extends** HttpServlet{
48. **public** **void** doGet(HttpServletRequest req,HttpServletResponse res)
49. **throws** ServletException,IOException
50. {
51. res.setContentType("text/html");//setting the content type
52. PrintWriter pw=res.getWriter();//get the stream to write the data
54. //writing html in the stream
55. pw.println("<html><body>");
56. pw.println("Welcome to servlet");
57. pw.println("</body></html>");
59. pw.close();//closing the stream
60. }}

### What is written inside the public service method?

The public service method converts the ServletRequest object into the HttpServletRequest type and ServletResponse object into the HttpServletResponse type. Then, calls the service method passing these objects. Let's see the internal code:

1. **public** **void** service(ServletRequest req, ServletResponse res)
2. **throws** ServletException, IOException
3. {
4. HttpServletRequest request;
5. HttpServletResponse response;
6. **try**
7. {
8. request = (HttpServletRequest)req;
9. response = (HttpServletResponse)res;
10. }
11. **catch**(ClassCastException e)
12. {
13. **throw** **new** ServletException("non-HTTP request or response");
14. }
15. service(request, response);
16. }

# **load on startup in web.xml**

1. **<web-app>**
2. ....
4. **<servlet>**
5. **<servlet-name>**servlet1**</servlet-name>**
6. **<servlet-class>**com.javatpoint.FirstServlet**</servlet-class>**
7. **<load-on-startup>**0**</load-on-startup>**
8. **</servlet>**
10. **<servlet>**
11. **<servlet-name>**servlet2**</servlet-name>**
12. **<servlet-class>**com.javatpoint.SecondServlet**</servlet-class>**
13. **<load-on-startup>**1**</load-on-startup>**
14. **</servlet>**
16. ...
17. **</web-app>**

If you specify the load-on-startup in web.xml, servlet will be loaded at project deployment time or server start. So, it will take less time for responding to first request.

#### **Passing positive value**

If you pass the positive value, the lower integer value servlet will be loaded before the higher integer value servlet. In other words, container loads the servlets in ascending integer value. The 0 value will be loaded first then 1, 2, 3 and so on.

#### **Passing negative value**

If you pass the negative value, servlet will be loaded at request time, at first request.

**The RequestDispatcher**

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

**index.html**

<form method="**post**" action="**Validate**">

Name:<input type="text" name="user" /><br/>

Password:<input type="password" name="pass" ><br/>

<input type="submit" value="submit">

</form>

**Validate.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class *Validate* **extends** HttpServlet {

protected void **doPost**(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

try {

String **name** = request.getParameter("user");

String **password** = request.getParameter("pass");

if(**password**.equals("studytonight"))

{

RequestDispatcher *rd* = request.**getRequestDispatcher**("Welcome");

rd.**forward**(request, response);

}

else

{

out.println("<font color='red'><b>You have entered incorrect password</b></font>");

RequestDispatcher *rd* = request.**getRequestDispatcher**("index.html");

rd.**include**(request, response);

}

}finally {

out.close();

}

}

}

**Welcome.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class *Welcome* **extends** HttpServlet {

protected void **doPost**(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter **out** = response.getWriter();

try {

out.println("<h2>Welcome user</h2>");

} finally {

out.close();

}

}

}

**web.xml**

<web-app>

<servlet>

<servlet-name>Validate</servlet-name>

<servlet-class>Validate</servlet-class>

</servlet>

<servlet>

<servlet-name>Welcome</servlet-name>

<servlet-class>Welcome</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Validate</servlet-name>

<url-pattern>/Validate</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>Welcome</servlet-name>

<url-pattern>/Welcome</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

**sendRedirect()**

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
5. **public** **class** DemoServlet **extends** HttpServlet{
6. **public** **void** doGet(HttpServletRequest req,HttpServletResponse res)
7. **throws** ServletException,IOException
8. {
9. res.setContentType("text/html");
10. PrintWriter pw=res.getWriter();
12. response.sendRedirect("http://www.google.com");
14. pw.close();
15. }}

|  |  |
| --- | --- |
| **forward() method** | **sendRedirect() method** |
| The forward() method works at server side. | The sendRedirect() method works at client side. |
| It sends the same request and response objects to another servlet. | It always sends a new request. |
| It can work within the server only. | It can be used within and outside the server. |
| Example: request.getRequestDispacher("servlet2").forward(request,response); | Example: response.sendRedirect("servlet2"); |

# **ServletConfig Interface**

An object of ServletConfig is created by the web container for each servlet. This object can be used to get (initialization parameters) configuration information from web.xml file.

### Example of ServletConfig to get initialization parameter

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
5. **public** **class** DemoServlet **extends** HttpServlet {
6. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)
7. **throws** ServletException, IOException {
9. response.setContentType("text/html");
10. PrintWriter out = response.getWriter();
12. ServletConfig config=getServletConfig();
13. String driver=config.getInitParameter("driver");
14. out.print("Driver is: "+driver);
16. out.close();
17. }
19. }
20. <web-app>
22. <servlet>
23. <servlet-name>DemoServlet</servlet-name>
24. <servlet-**class**>DemoServlet</servlet-**class**>
26. <init-param>
27. <param-name>driver</param-name>
28. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
29. </init-param>
31. </servlet>
33. <servlet-mapping>
34. <servlet-name>DemoServlet</servlet-name>
35. <url-pattern>/servlet1</url-pattern>
36. </servlet-mapping>
38. </web-app>

### Example of ServletConfig to get all the initialization parameters

1. **import** java.io.IOException;
2. **import** java.io.PrintWriter;
3. **import** java.util.Enumeration;
5. **import** javax.servlet.ServletConfig;
6. **import** javax.servlet.ServletException;
7. **import** javax.servlet.http.HttpServlet;
8. **import** javax.servlet.http.HttpServletRequest;
9. **import** javax.servlet.http.HttpServletResponse;

12. **public** **class** DemoServlet **extends** HttpServlet {
13. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)
14. **throws** ServletException, IOException {
16. response.setContentType("text/html");
17. PrintWriter out = response.getWriter();
19. ServletConfig config=getServletConfig();
20. Enumeration<String> e=config.getInitParameterNames();
22. String str="";
23. **while**(e.hasMoreElements()){
24. str=e.nextElement();
25. out.print("<br>Name: "+str);
26. out.print(" value: "+config.getInitParameter(str));
27. }
29. out.close();
30. }
32. }
33. <web-app>
35. <servlet>
36. <servlet-name>DemoServlet</servlet-name>
37. <servlet-**class**>DemoServlet</servlet-**class**>
39. <init-param>
40. <param-name>username</param-name>
41. <param-value>system</param-value>
42. </init-param>
44. <init-param>
45. <param-name>password</param-name>
46. <param-value>oracle</param-value>
47. </init-param>
49. </servlet>
51. <servlet-mapping>
52. <servlet-name>DemoServlet</servlet-name>
53. <url-pattern>/servlet1</url-pattern>
54. </servlet-mapping>
56. </web-app>

# **ServletContext Interface**

### An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;

6. **public** **class** DemoServlet **extends** HttpServlet{
7. **public** **void** doGet(HttpServletRequest req,HttpServletResponse res)
8. **throws** ServletException,IOException
9. {
10. res.setContentType("text/html");
11. PrintWriter pw=res.getWriter();
12. //creating ServletContext object
13. ServletContext context=getServletContext();
15. //Getting the value of the initialization parameter and printing it
16. String driverName=context.getInitParameter("dname");
17. pw.println("driver name is="+driverName);
19. pw.close();
21. }}
22. <web-app>
24. <servlet>
25. <servlet-name>sonoojaiswal</servlet-name>
26. <servlet-**class**>DemoServlet</servlet-**class**>
27. </servlet>
29. <context-param>
30. <param-name>dname</param-name>
31. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
32. </context-param>
34. <servlet-mapping>
35. <servlet-name>sonoojaiswal</servlet-name>
36. <url-pattern>/context</url-pattern>
37. </servlet-mapping>
39. </web-app>

### Example of ServletContext to set and get attribute

### DemoServlet1.java

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;

6. **public** **class** DemoServlet1 **extends** HttpServlet{
7. **public** **void** doGet(HttpServletRequest req,HttpServletResponse res)
8. {
9. **try**{
11. res.setContentType("text/html");
12. PrintWriter out=res.getWriter();
14. ServletContext context=getServletContext();
15. context.setAttribute("company","IBM");
17. out.println("Welcome to first servlet");
18. out.println("<a href='servlet2'>visit</a>");
19. out.close();
21. }**catch**(Exception e){out.println(e);}
23. }}

### DemoServlet2.java

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;

6. **public** **class** DemoServlet2 **extends** HttpServlet{
7. **public** **void** doGet(HttpServletRequest req,HttpServletResponse res)
8. {
9. **try**{
11. res.setContentType("text/html");
12. PrintWriter out=res.getWriter();
14. ServletContext context=getServletContext();
15. String n=(String)context.getAttribute("company");
17. out.println("Welcome to "+n);
18. out.close();
20. }**catch**(Exception e){out.println(e);}
22. }}

### web.xml

1. <web-app>
3. <servlet>
4. <servlet-name>s1</servlet-name>
5. <servlet-**class**>DemoServlet1</servlet-**class**>
6. </servlet>
8. <servlet-mapping>
9. <servlet-name>s1</servlet-name>
10. <url-pattern>/servlet1</url-pattern>
11. </servlet-mapping>
13. <servlet>
14. <servlet-name>s2</servlet-name>
15. <servlet-**class**>DemoServlet2</servlet-**class**>
16. </servlet>
18. <servlet-mapping>
19. <servlet-name>s2</servlet-name>
20. <url-pattern>/servlet2</url-pattern>
21. </servlet-mapping>
23. </web-app>

# **Session Tracking in Servlets**

### **Session** simply means a particular interval of time.

### Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:

Session Tracking Techniques

There are four techniques used in Session tracking:

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

A **cookie** is a small piece of information that is persisted between the multiple client requests.

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

### index.html

1. <form action="servlet1" method="post">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>

### FirstServlet.java

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;

6. **public** **class** FirstServlet **extends** HttpServlet {
8. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response){
9. **try**{
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. String n=request.getParameter("userName");
15. out.print("Welcome "+n);
17. Cookie ck=**new** Cookie("uname",n);//creating cookie object
18. response.addCookie(ck);//adding cookie in the response
20. //creating submit button
21. out.print("<form action='servlet2'>");
22. out.print("<input type='submit' value='go'>");
23. out.print("</form>");
25. out.close();
27. }**catch**(Exception e){System.out.println(e);}
28. }
29. }

### SecondServlet.java

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
5. **public** **class** SecondServlet **extends** HttpServlet {
7. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response){
8. **try**{
10. response.setContentType("text/html");
11. PrintWriter out = response.getWriter();
13. Cookie ck[]=request.getCookies();
14. out.print("Hello "+ck[0].getValue());
16. out.close();
18. }**catch**(Exception e){System.out.println(e);}
19. }

22. }

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| How Cookie works**Cookies in Servlet** A **cookie** is a small piece of information that is persisted between the multiple client requests.  A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.  By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.  cookies in servlet  **Types of Cookie**  There are 2 types of cookies in servlets.   1. Non-persistent cookie 2. Persistent cookie  Non-persistent cookie It is **valid for single session** only. It is removed each time when user closes the browser. Persistent cookie It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout. Advantage of Cookies  1. Simplest technique of maintaining the state. 2. Cookies are maintained at client side.  Disadvantage of Cookies  1. It will not work if cookie is disabled from the browser. 2. Only textual information can be set in Cookie object.  Note: Gmail uses cookie technique for login. If you disable the cookie, gmail won't work.Cookie class **javax.servlet.http.Cookie** class provides the functionality of using cookies. It provides a lot of useful methods for cookies. Constructor of Cookie class  |  |  | | --- | --- | | **Constructor** | **Description** | | Cookie() | constructs a cookie. | | Cookie(String name, String value) | constructs a cookie with a specified name and value. |  Useful Methods of Cookie class There are given some commonly used methods of the Cookie class.   |  |  | | --- | --- | | **Method** | **Description** | | public void setMaxAge(int expiry) | Sets the maximum age of the cookie in seconds. | | public String getName() | Returns the name of the cookie. The name cannot be changed after creation. | | public String getValue() | Returns the value of the cookie. | | public void setName(String name) | changes the name of the cookie. | | public void setValue(String value) | changes the value of the cookie. |  Other methods required for using Cookies  |  | | --- | | For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:   1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object. 2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser. |  How to create Cookie?  1. Cookie ck=**new** Cookie("user","sonoo jaiswal");//creating cookie object 2. response.addCookie(ck);//adding cookie in the response  How to delete Cookie? Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.   1. Cookie ck=**new** Cookie("user","");//deleting value of cookie 2. ck.setMaxAge(0);//changing the maximum age to 0 seconds 3. response.addCookie(ck);//adding cookie in the response  How to get Cookies? Let's see the simple code to get all the cookies.   1. Cookie ck[]=request.getCookies(); 2. **for**(**int** i=0;i<ck.length;i++){ 3. out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie 4. }  Simple example of Servlet CookiesIn this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value. cookies in session tracking index.html  1. <form action="servlet1" method="post"> 2. Name:<input type="text" name="userName"/><br/> 3. <input type="submit" value="go"/> 4. </form>  FirstServlet.java  1. **import** java.io.\*; 2. **import** javax.servlet.\*; 3. **import** javax.servlet.http.\*;  6. **public** **class** FirstServlet **extends** HttpServlet { 8. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response){ 9. **try**{ 11. response.setContentType("text/html"); 12. PrintWriter out = response.getWriter(); 14. String n=request.getParameter("userName"); 15. out.print("Welcome "+n); 17. Cookie ck=**new** Cookie("uname",n);//creating cookie object 18. response.addCookie(ck);//adding cookie in the response 20. //creating submit button 21. out.print("<form action='servlet2'>"); 22. out.print("<input type='submit' value='go'>"); 23. out.print("</form>"); 25. out.close(); 27. }**catch**(Exception e){System.out.println(e);} 28. } 29. }  SecondServlet.java  1. **import** java.io.\*; 2. **import** javax.servlet.\*; 3. **import** javax.servlet.http.\*; 5. **public** **class** SecondServlet **extends** HttpServlet { 7. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response){ 8. **try**{ 10. response.setContentType("text/html"); 11. PrintWriter out = response.getWriter(); 13. Cookie ck[]=request.getCookies(); 14. out.print("Hello "+ck[0].getValue()); 16. out.close(); 18. }**catch**(Exception e){System.out.println(e);} 19. }  22. }  web.xml |

1. <web-app>
3. <servlet>
4. <servlet-name>s1</servlet-name>
5. <servlet-**class**>FirstServlet</servlet-**class**>
6. </servlet>
8. <servlet-mapping>
9. <servlet-name>s1</servlet-name>
10. <url-pattern>/servlet1</url-pattern>
11. </servlet-mapping>
13. <servlet>
14. <servlet-name>s2</servlet-name>
15. <servlet-**class**>SecondServlet</servlet-**class**>
16. </servlet>
18. <servlet-mapping>
19. <servlet-name>s2</servlet-name>
20. <url-pattern>/servlet2</url-pattern>
21. </servlet-mapping>
23. </web-app>

# **2) Hidden Form Field**

### In case of Hidden Form Field a hidden (invisible) textfield is used for maintaining the state of an user. In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

### Advantage of Hidden Form Field

1. It will always work whether cookie is disabled or not.

### Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

### index.html- same as cookie

### FirstServlet.java

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
4. **public** **class** FirstServlet **extends** HttpServlet {
5. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response){
6. **try**{
8. response.setContentType("text/html");
9. PrintWriter out = response.getWriter();
11. String n=request.getParameter("userName");
12. out.print("Welcome "+n);
14. //creating form that have invisible textfield
15. out.print("<form action='servlet2'>");
16. out.print("<input type='hidden' name='uname' value='"+n+"'>");
17. out.print("<input type='submit' value='go'>");
18. out.print("</form>");
19. out.close();
21. }**catch**(Exception e){System.out.println(e);}
22. }
24. }

### SecondServlet.java

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
4. **public** **class** SecondServlet **extends** HttpServlet {
5. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)
6. **try**{
7. response.setContentType("text/html");
8. PrintWriter out = response.getWriter();
10. //Getting the value from the hidden field
11. String n=request.getParameter("uname");
12. out.print("Hello "+n);
14. out.close();
15. }**catch**(Exception e){System.out.println(e);}
16. }
17. }

### web.xml- same as cookie

# **3)URL Rewriting**

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

### Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

### Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

### index.html- same as cookie

### FirstServlet.java

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;

6. **public** **class** FirstServlet **extends** HttpServlet {
8. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response){
9. **try**{
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. String n=request.getParameter("userName");
15. out.print("Welcome "+n);
17. //appending the username in the query string
18. out.print("<a href='servlet2?uname="+n+"'>visit</a>");
20. out.close();
22. }**catch**(Exception e){System.out.println(e);}
23. }
25. }

### SecondServlet.java

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
5. **public** **class** SecondServlet **extends** HttpServlet {
7. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)
8. **try**{
10. response.setContentType("text/html");
11. PrintWriter out = response.getWriter();
13. //getting value from the query string
14. String n=request.getParameter("uname");
15. out.print("Hello "+n);
17. out.close();
19. }**catch**(Exception e){System.out.println(e);}
20. }
21. }

# **4) HttpSession interface**

In such case, container creates a session id for each user. The container uses this id to identify the particular user.

### index.html- same as cookie

### FirstServlet.java

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;

6. **public** **class** FirstServlet **extends** HttpServlet {
8. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response){
9. **try**{
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. String n=request.getParameter("userName");
15. out.print("Welcome "+n);
17. HttpSession session=request.getSession();
18. session.setAttribute("uname",n);
20. out.print("<a href='servlet2'>visit</a>");
22. out.close();
24. }**catch**(Exception e){System.out.println(e);}
25. }
27. }

### SecondServlet.java

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
5. **public** **class** SecondServlet **extends** HttpServlet {
7. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)
8. **try**{
10. response.setContentType("text/html");
11. PrintWriter out = response.getWriter();
13. HttpSession session=request.getSession(**false**);
14. String n=(String)session.getAttribute("uname");
15. out.print("Hello "+n);
17. out.close();
19. }**catch**(Exception e){System.out.println(e);}
20. }

23. }

# **Servlet Filter**

A **filter** is an object that is invoked at the preprocessing and postprocessing of a request.

It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.

The **servlet filter is pluggable**, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.

### Simple Example of Filter

In this example, we are simply displaying information that filter is invoked automatically after the post processing of the request.

### index.html

1. <a href="servlet1">click here</a>

### MyFilter.java

1. **import** java.io.IOException;
2. **import** java.io.PrintWriter;
4. **import** javax.servlet.\*;
6. **public** **class** MyFilter **implements** Filter{
8. **public** **void** init(FilterConfig arg0) **throws** ServletException {}
10. **public** **void** doFilter(ServletRequest req, ServletResponse resp,
11. FilterChain chain) **throws** IOException, ServletException {
13. PrintWriter out=resp.getWriter();
14. out.print("filter is invoked before");
16. chain.doFilter(req, resp);//sends request to next resource
18. out.print("filter is invoked after");
19. }
20. **public** **void** destroy() {}
21. }

### HelloServlet.java

1. **import** java.io.IOException;
2. **import** java.io.PrintWriter;
4. **import** javax.servlet.ServletException;
5. **import** javax.servlet.http.\*;
7. **public** **class** HelloServlet **extends** HttpServlet {
8. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)
9. **throws** ServletException, IOException {
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. out.print("<br>welcome to servlet<br>");
16. }
18. }

**web.xml**

|  |
| --- |
| For defining the filter, filter element of web-app must be defined just like servlet. |

1. <web-app>
3. <servlet>
4. <servlet-name>s1</servlet-name>
5. <servlet-**class**>HelloServlet</servlet-**class**>
6. </servlet>
8. <servlet-mapping>
9. <servlet-name>s1</servlet-name>
10. <url-pattern>/servlet1</url-pattern>
11. </servlet-mapping>
13. <filter>
14. <filter-name>f1</filter-name>
15. <filter-**class**>MyFilter</filter-**class**>
16. </filter>
18. <filter-mapping>
19. <filter-name>f1</filter-name>
20. <url-pattern>/servlet1</url-pattern>
21. </filter-mapping>
22. </web-app>

# **FilterConfig**

An object of FilterConfig is created by the web container. This object can be used to get the configuration information from the web.xml file.

### Example of counting number of visitors for a single page

**MyFilter.java**

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
4. **public** **class** MyFilter **implements** Filter{
5. **static** **int** count=0;
6. **public** **void** init(FilterConfig arg0) **throws** ServletException {}
8. **public** **void** doFilter(ServletRequest req, ServletResponse res,
9. FilterChain chain) **throws** IOException, ServletException {
11. PrintWriter out=res.getWriter();
12. chain.doFilter(request,response);
14. out.print("<br/>Total visitors "+(++count));
15. out.close();
17. }
18. **public** **void** destroy() {}
19. }

### Example of checking total response time in filter

**MyFilter.java**

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
4. **public** **class** MyFilter **implements** Filter{
5. **static** **int** count=0;
6. **public** **void** init(FilterConfig arg0) **throws** ServletException {}
8. **public** **void** doFilter(ServletRequest req, ServletResponse res,
9. FilterChain chain) **throws** IOException, ServletException {
11. PrintWriter out=res.getWriter();
12. **long** before=System.currentTimeMillis();
14. chain.doFilter(request,response);
16. **long** after=System.currentTimeMillis();
17. out.print("<br/>Total response time "+(after-before)+" miliseconds");
18. out.close();
20. }
21. **public** **void** destroy() {}
22. }

# **ServletInputStream class**

**ServletInputStream** class provides stream to read binary data such as image etc. from the request object. It is an abstract class.

The **getInputStream()** method of **ServletRequest** interface returns the instance of ServletInputStream class.

### Method of ServletInputStream class

There are only one method defined in the ServletInputStream class.

1. **int readLine(byte[] b, int off, int len)**it reads the input stream.

# **Example to display image using Servlet**

**index.html**

This file creates a link that invokes the servlet. The url-pattern of the servlet is servlet1.

1. <a href="servlet1">click **for** photo</a>

**DisplayImage.java**

This servlet class reads the image from the mentioned directory and writes the content in the response object using ServletOutputStream and BufferedOutputStream classes.

1. **package** com.javatpoint;
2. **import** java.io.\*;
3. **import** javax.servlet.\*;
4. **import** javax.servlet.http.\*;
5. **public** **class** DisplayImage **extends** HttpServlet {
7. **public** **void** doGet(HttpServletRequest request,HttpServletResponse response)
8. **throws** IOException
9. {
10. response.setContentType("image/jpeg");
11. ServletOutputStream out;
12. out = response.getOutputStream();
13. FileInputStream fin = **new** FileInputStream("c:\\test\\java.jpg");
15. BufferedInputStream bin = **new** BufferedInputStream(fin);
16. BufferedOutputStream bout = **new** BufferedOutputStream(out);
17. **int** ch =0; ;
18. **while**((ch=bin.read())!=-1)
19. {
20. bout.write(ch);
21. }
23. bin.close();
24. fin.close();
25. bout.close();
26. out.close();
27. }
28. }
29. **What are the differences between sling:resourceType and sling:resourceSuperType ?**

**sling:resourceSuperType:** It is used to achieve inheritance in cq. When set, it inherits the specified component to this component.

**sling:resourceType:** It is a path, which locates the script to be used for rendering the content. Path used can be absolute or relative.

1. **List Implicit objects of CQ?**  
   slingRequest ,slingResponse ,resource , currentNode, log , properties , pageManager ,component ,designer ,currentDesign ,currentStyle
2. **What are the differences between CRX DE and Ecllipse ? What are the advantages of using CRXDE over Ecllipse for AEM Development?**

**The major difference between Crxde and eclipse**

Crxde comes with AEM and it runs on server side where as eclipse runs on client side. CRXDE contains build in API’s where as in eclipse we need to add extra plugin for supporting CRX API.

**Built in API’s of crxde**

**Java Content Repository API**—a complete JCR 2.0 implementation Content Management Interoperability Services—CMIS 1.0

**WebDAV**—with versioning, access control, and search Windows Network File Share—CIFS/SMB

**RESTful Web API** for JavaScript and Flash/Flex  

1. **How many wcm modes are there?**

* Edit mode
* Design mode
* Preview mode

**Preview mode**: This mode removes all the authoring interface and shows you how the web page would look on the website. It's best to use this mode for checking the end results.

**Edit mode:** Whenever author makes changes or configures components (Example: Adding title to a page by editing the Title component) in this mode the data is stored under the components' node. The path of the configured data will look something like this: /content/your website/path\_to\_the\_page/jcr/component

**Design mode:** This mode is used to configure components that have same data on different pages like the logo or footer. Any changes you make on the component in any one of the pages' will be reflected on all the pages that have the component. The catch is all these pages must have the same design.

1. **What is service?**

A service is *any object* that is registered in the OSGi Service Registry and can be looked up using its interface name(s). The only prerequisite is that a service should implement some interface...