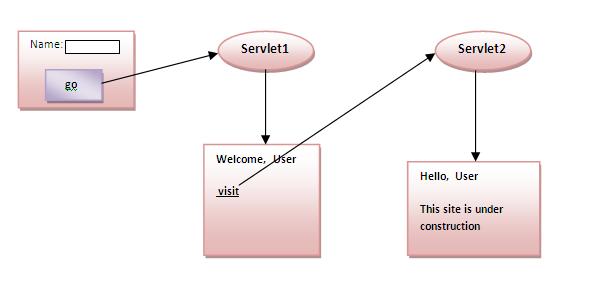
**URL REWRITING**

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&??

A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.



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

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

FirstServlet.java

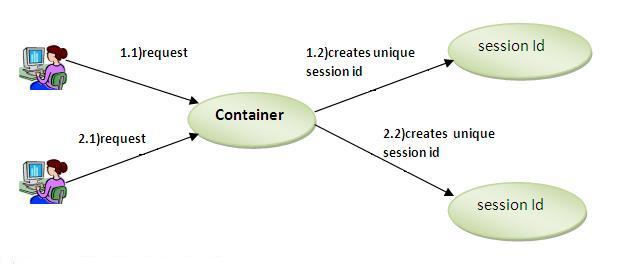
1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;

6. **public** **class** FirstServlet **extends** HttpServlet {
8. **public** **void** 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. }

**SESSION**

In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



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

1. <form action="servlet1">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>
5. <form action="servlet1">
6. Name:<input type="text" name="userName"/><br/>
7. <input type="submit" value="go"/>
8. </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** 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. }

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. }

23. }

**COOKIE**

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.

### How Cookie works

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.

**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.

**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. |

### 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. }

**HIDDEN FROM FIELD**

* + **Server**

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.

<input type="hidden" name="uname" value="SSSDIIT">

**Real application of hidden form field**

It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

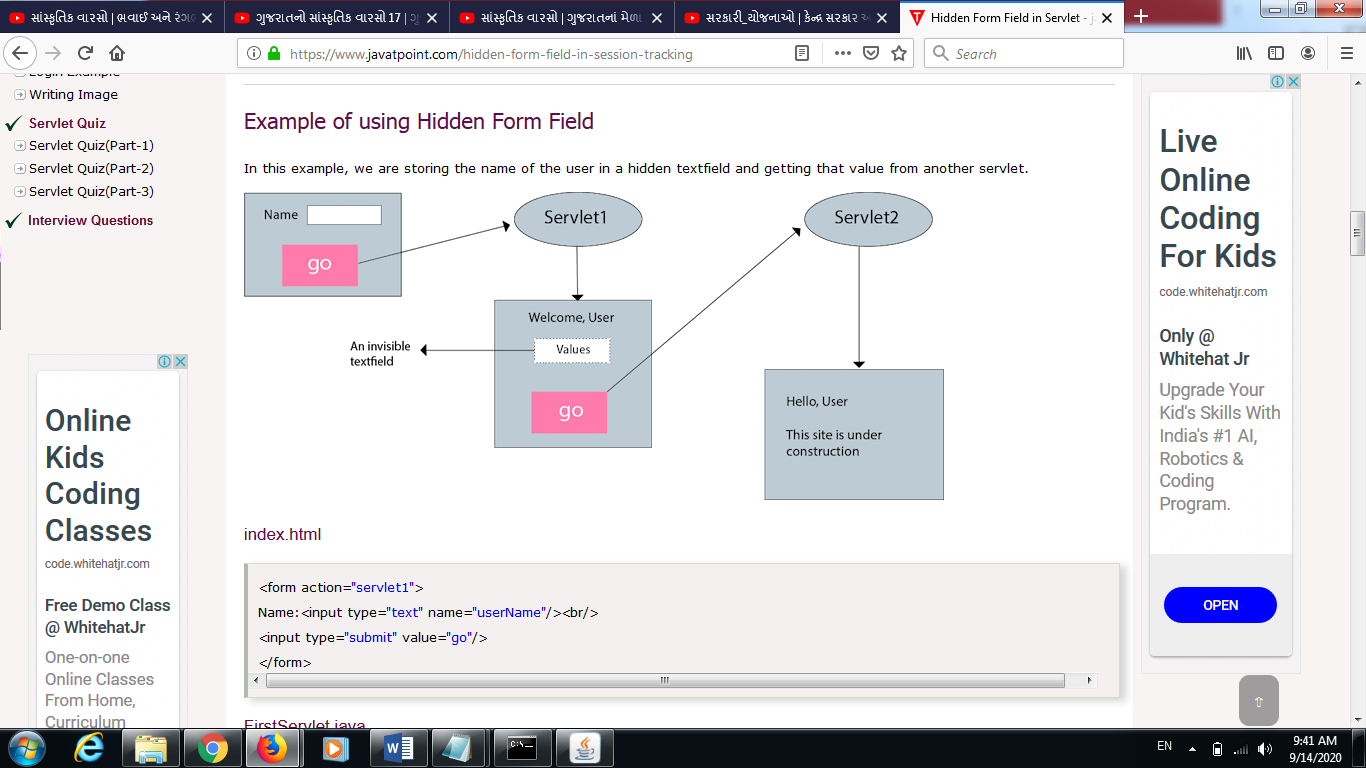
**Advantage of Hidden Form Field**

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.

 In this example, we are storing the name of the user in a hidden textfield and getting that value from another servlet.



### index.html

1. <form action="servlet1">
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.\*;
5. public class FirstServlet extends HttpServlet {
6. public void doGet(HttpServletRequest request, HttpServletResponse response){
7. try{
9. response.setContentType("text/html");
10. PrintWriter out = response.getWriter();
12. String n=request.getParameter("userName");
13. out.print("Welcome "+n);
15. //creating form that have invisible textfield
16. out.print("<form action='servlet2'>");
17. out.print("<input type='hidden' name='uname' value='"+n+"'>");
18. out.print("<input type='submit' value='go'>");
19. out.print("</form>");
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.\*;
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. }

**SERVLET COLLABORATION**

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.

There are two methods defined in the RequestDispatcher interface.

### Methods of RequestDispatcher interface

The RequestDispatcher interface provides two methods. They are:

1. **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
2. **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

As you see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user.

|  |
| --- |
| As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client. |

### How to get the object of RequestDispatcher

The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher. Syntax:

#### Syntax of getRequestDispatcher method

1. public RequestDispatcher getRequestDispatcher(String resource);

#### Example of using getRequestDispatcher method

1. RequestDispatcher rd=request.getRequestDispatcher("servlet2");
2. //servlet2 is the url-pattern of the second servlet
4. rd.forward(request, response);//method may be include or forward

### Example of RequestDispatcher interface

In this example, we are validating the password entered by the user. If password is servlet, it will forward the request to the WelcomeServlet, otherwise will show an error message: sorry username or password error!. In this program, we are cheking for hardcoded information. But you can check it to the database also that we will see in the development chapter. In this example, we have created following files:

* **index.html file:** for getting input from the user.
* **Login.java file:** a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.
* **WelcomeServlet.java file:** a servlet class for displaying the welcome message.
* **web.xml file:** a deployment descriptor file that contains the information about the servlet.

**index.html**

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

**Login.java**

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

6. public class Login extends HttpServlet {
8. public void doPost(HttpServletRequest request, HttpServletResponse response)
9. throws ServletException, IOException {
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. String n=request.getParameter("userName");
15. String p=request.getParameter("userPass");
17. if(p.equals("servlet"){
18. RequestDispatcher rd=request.getRequestDispatcher("servlet2");
19. rd.forward(request, response);
20. }
21. else{
22. out.print("Sorry UserName or Password Error!");
23. RequestDispatcher rd=request.getRequestDispatcher("/index.html");
24. rd.include(request, response);
26. }
27. }
29. }

**WelcomeServlet.java**

1. import java.io.\*;
2. import javax.servlet.\*;
3. import javax.servlet.http.\*;
5. public class WelcomeServlet extends HttpServlet {
7. public void doPost(HttpServletRequest request, HttpServletResponse response)
8. throws ServletException, IOException {
10. response.setContentType("text/html");
11. PrintWriter out = response.getWriter();
13. String n=request.getParameter("userName");
14. out.print("Welcome "+n);
15. }
17. }

**SEND REDIRECT**

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

It accepts relative as well as absolute URL.

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.

## Difference between forward() and sendRedirect() method

There are many differences between the forward() method of RequestDispatcher and sendRedirect() method of HttpServletResponse interface. They are given below:

|  |  |
| --- | --- |
| **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"); |

### Syntax of sendRedirect() method

1. public void sendRedirect(String URL)throws IOException;

**Servlet config interface**

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

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

### Advantage of ServletConfig

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

### Methods of ServletConfig interface

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

### How to get the object of ServletConfig

1. **getServletConfig() method** of Servlet interface returns the object of ServletConfig.

#### Syntax of getServletConfig() method

1. public ServletConfig getServletConfig();

### Example of getServletConfig() method

1. ServletConfig config=getServletConfig();
2. //Now we can call the methods of ServletConfig interface

### Syntax to provide the initialization parameter for a servlet

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

1. <web-app>
2. <servlet>
3. ......
5. <init-param>
6. <param-name>parametername</param-name>
7. <param-value>parametervalue</param-value>
8. </init-param>
9. ......
10. </servlet>
11. </web-app>

### Example of 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.

**DemoServlet.java**

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. }

**web.xml**

1. <web-app>
3. <servlet>
4. <servlet-name>DemoServlet</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
7. <init-param>
8. <param-name>driver</param-name>
9. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
10. </init-param>
12. </servlet>
14. <servlet-mapping>
15. <servlet-name>DemoServlet</servlet-name>
16. <url-pattern>/servlet1</url-pattern>
17. </servlet-mapping>
19. </web-app>

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.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

### Advantage of ServletContext

**Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

### Usage of ServletContext Interface

There can be a lot of usage of ServletContext object. Some of them are as follows:

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.

### Commonly used methods of ServletContext interface

|  |
| --- |
| There is given some commonly used methods of ServletContext interface.   1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name. 2. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters. 3. **public void setAttribute(String name,Object object):**sets the given object in the application scope. 4. **public Object getAttribute(String name):**Returns the attribute for the specified name. 5. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects. 6. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context. |

### How to get the object of ServletContext interface

1. **getServletContext() method** of ServletConfig interface returns the object of ServletContext.
2. **getServletContext() method** of GenericServlet class returns the object of ServletContext.

#### Syntax of getServletContext() method

1. public ServletContext getServletContext()

#### Example of getServletContext() method

1. //We can get the ServletContext object from ServletConfig object
2. ServletContext application=getServletConfig().getServletContext();
4. //Another convenient way to get the ServletContext object
5. ServletContext application=getServletContext();

### Syntax to provide the initialization parameter in Context scope

|  |
| --- |
| The **context-param** element, subelement of web-app, is used to define the initialization parameter in the application scope. The param-name and param-value are the sub-elements of the context-param. The param-name element defines parameter name and and param-value defines its value. |

1. <web-app>
2. ......
4. <context-param>
5. <param-name>parametername</param-name>
6. <param-value>parametervalue</param-value>
7. </context-param>
8. ......
9. </web-app>

### Example of ServletContext to get the initialization parameter

|  |
| --- |
| In this example, we are getting the initialization parameter from the web.xml file and printing the value of the initialization parameter. Notice that the object of ServletContext represents the application scope. So if we change the value of the parameter from the web.xml file, all the servlet classes will get the changed value. So we don't need to modify the servlet. So it is better to have the common information for most of the servlets in the web.xml file by context-param element. Let's see the simple example: |

**DemoServlet.java**

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();
13. //creating ServletContext object
14. ServletContext context=getServletContext();
16. //Getting the value of the initialization parameter and printing it
17. String driverName=context.getInitParameter("dname");
18. pw.println("driver name is="+driverName);
20. pw.close();
22. }}

**web.xml**

1. <web-app>
3. <servlet>
4. <servlet-name>Servlet</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6. </servlet>
8. <context-param>
9. <param-name>dname</param-name>
10. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
11. </context-param>
13. <servlet-mapping>
14. <servlet-name>Servlet</servlet-name>
15. <url-pattern>/context</url-pattern>
16. </servlet-mapping>
18. </web-app>

**Login & Logout using Cookie**

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset="ISO-8859-1">
5. <title>Servlet Login Example</title>
6. </head>
7. <body>
9. <h1>Welcome to Login App by Cookie</h1>
10. <a href="login.html">Login</a>|
11. <a href="LogoutServlet">Logout</a>|
12. <a href="ProfileServlet">Profile</a>
14. </body>
15. </html>

File: link.html

1. <a href="login.html">Login</a> |
2. <a href="LogoutServlet">Logout</a> |
3. <a href="ProfileServlet">Profile</a>
4. <hr>

File: login.html

1. <form action="LoginServlet" method="post">
2. Name:<input type="text" name="name"><br>
3. Password:<input type="password" name="password"><br>
4. <input type="submit" value="login">
5. </form>

File: LoginServlet.java

1. package com.javatpoint;
3. import java.io.IOException;
4. import java.io.PrintWriter;
5. import javax.servlet.ServletException;
6. import javax.servlet.http.Cookie;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10. public class LoginServlet extends HttpServlet {
11. protected void doPost(HttpServletRequest request, HttpServletResponse response)
12. throws ServletException, IOException {
13. response.setContentType("text/html");
14. PrintWriter out=response.getWriter();
16. request.getRequestDispatcher("link.html").include(request, response);
18. String name=request.getParameter("name");
19. String password=request.getParameter("password");
21. if(password.equals("admin123")){
22. out.print("You are successfully logged in!");
23. out.print("<br>Welcome, "+name);
25. Cookie ck=new Cookie("name",name);
26. response.addCookie(ck);
27. }else{
28. out.print("sorry, username or password error!");
29. request.getRequestDispatcher("login.html").include(request, response);
30. }
32. out.close();
33. }
35. }

File: LogoutServlet.java

1. package com.javatpoint;
3. import java.io.IOException;
4. import java.io.PrintWriter;
5. import javax.servlet.ServletException;
6. import javax.servlet.http.Cookie;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10. public class LogoutServlet extends HttpServlet {
11. protected void doGet(HttpServletRequest request, HttpServletResponse response)
12. throws ServletException, IOException {
13. response.setContentType("text/html");
14. PrintWriter out=response.getWriter();

17. request.getRequestDispatcher("link.html").include(request, response);
19. Cookie ck=new Cookie("name","");
20. ck.setMaxAge(0);
21. response.addCookie(ck);
23. out.print("you are successfully logged out!");
24. }
25. }

File: ProfileServlet.java

1. package com.javatpoint;
3. import java.io.IOException;
4. import java.io.PrintWriter;
5. import javax.servlet.ServletException;
6. import javax.servlet.http.Cookie;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10. public class ProfileServlet extends HttpServlet {
11. protected void doGet(HttpServletRequest request, HttpServletResponse response)
12. throws ServletException, IOException {
13. response.setContentType("text/html");
14. PrintWriter out=response.getWriter();
16. request.getRequestDispatcher("link.html").include(request, response);
18. Cookie ck[]=request.getCookies();
19. if(ck!=null){
20. String name=ck[0].getValue();
21. if(!name.equals("")||name!=null){
22. out.print("<b>Welcome to Profile</b>");
23. out.print("<br>Welcome, "+name);
24. }
25. }else{
26. out.print("Please login first");
27. request.getRequestDispatcher("login.html").include(request, response);
28. }
29. out.close();
30. }
31. }

**Login & Logout using session**

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset="ISO-8859-1">
5. <title>Servlet Login Example</title>
6. </head>
7. <body>
9. <h1>Login App using HttpSession</h1>
10. <a href="login.html">Login</a>|
11. <a href="LogoutServlet">Logout</a>|
12. <a href="ProfileServlet">Profile</a>
14. </body>
15. </html>

File: link.html

1. <a href="login.html">Login</a> |
2. <a href="LogoutServlet">Logout</a> |
3. <a href="ProfileServlet">Profile</a>
4. <hr>

File: login.html

1. <form action="LoginServlet" method="post">
2. Name:<input type="text" name="name"><br>
3. Password:<input type="password" name="password"><br>
4. <input type="submit" value="login">
5. </form>

File: LoginServlet.java

1. import java.io.IOException;
2. import java.io.PrintWriter;
4. import javax.servlet.ServletException;
5. import javax.servlet.http.HttpServlet;
6. import javax.servlet.http.HttpServletRequest;
7. import javax.servlet.http.HttpServletResponse;
8. import javax.servlet.http.HttpSession;
9. public class LoginServlet extends HttpServlet {
10. protected void doPost(HttpServletRequest request, HttpServletResponse response)
11. throws ServletException, IOException {
12. response.setContentType("text/html");
13. PrintWriter out=response.getWriter();
14. request.getRequestDispatcher("link.html").include(request, response);
16. String name=request.getParameter("name");
17. String password=request.getParameter("password");
19. if(password.equals("admin123")){
20. out.print("Welcome, "+name);
21. HttpSession session=request.getSession();
22. session.setAttribute("name",name);
23. }
24. else{
25. out.print("Sorry, username or password error!");
26. request.getRequestDispatcher("login.html").include(request, response);
27. }
28. out.close();
29. }
30. }

File: LogoutServlet.java

1. import java.io.IOException;
2. import java.io.PrintWriter;
4. import javax.servlet.ServletException;
5. import javax.servlet.http.HttpServlet;
6. import javax.servlet.http.HttpServletRequest;
7. import javax.servlet.http.HttpServletResponse;
8. import javax.servlet.http.HttpSession;
9. public class LogoutServlet extends HttpServlet {
10. protected void doGet(HttpServletRequest request, HttpServletResponse response)
11. throws ServletException, IOException {
12. response.setContentType("text/html");
13. PrintWriter out=response.getWriter();
15. request.getRequestDispatcher("link.html").include(request, response);
17. HttpSession session=request.getSession();
18. session.invalidate();
20. out.print("You are successfully logged out!");
22. out.close();
23. }
24. }

File: ProfileServlet.java

1. import java.io.IOException;
2. import java.io.PrintWriter;
3. import javax.servlet.ServletException;
4. import javax.servlet.http.HttpServlet;
5. import javax.servlet.http.HttpServletRequest;
6. import javax.servlet.http.HttpServletResponse;
7. import javax.servlet.http.HttpSession;
8. public class ProfileServlet extends HttpServlet {
9. protected void doGet(HttpServletRequest request, HttpServletResponse response)
10. throws ServletException, IOException {
11. response.setContentType("text/html");
12. PrintWriter out=response.getWriter();
13. request.getRequestDispatcher("link.html").include(request, response);
15. HttpSession session=request.getSession(false);
16. if(session!=null){
17. String name=(String)session.getAttribute("name");
19. out.print("Hello, "+name+" Welcome to Profile");
20. }
21. else{
22. out.print("Please login first");
23. request.getRequestDispatcher("login.html").include(request, response);
24. }
25. out.close();
26. }
27. }