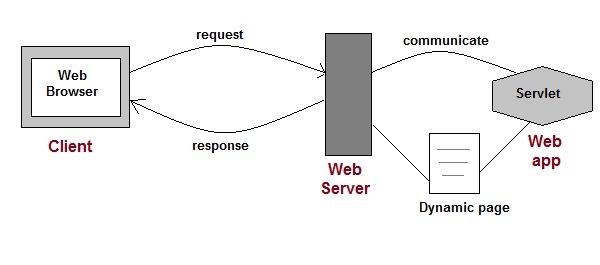
**Introduction to Servlet**

**Servlet** Technology is used to create web applications. **Servlet** technology uses Java language to create web applications.

Web applications are helper applications that resides at web server and build dynamic web pages. A dynamic page could be anything like a page that randomly chooses picture to display or even a page that displays the current time.



As Servlet Technology uses Java, web applications made using Servlet are **Secured**, **Scalable** and **Robust**.

### Servlet API (Application programing interface (library or data) which can be access over the internet)

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

* **javax.servlet**
* **javax.servlet.http**

#### Some Important Classes and Interfaces of javax.servlet

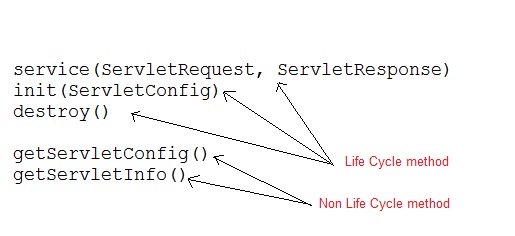
|  |  |
| --- | --- |
| **INTERFACES** | **CLASSES** |
| Servlet | ServletInputStream |
| ServletContext | ServletOutputStream |
| ServletConfig | ServletRequestWrapper |
| ServletRequest | ServletResponseWrapper |
| ServletResponse | ServletRequestEvent |
| ServletContextListener | ServletContextEvent |
| RequestDispatcher | ServletRequestAttributeEvent |
| SingleThreadModel | ServletContextAttributeEvent |
| Filter | ServletException |
| FilterConfig | UnavailableException |
| FilterChain | GenericServlet |
| ServletRequestListener |  |

#### Some Important Classes and Interface of javax.servlet.http

|  |  |
| --- | --- |
| **CLASSES and INTERFACES** | |
| HttpServlet | HttpServletRequest |
| HttpServletResponse | HttpSessionAttributeListener |
| HttpSession | HttpSessionListener |
| Cookie | HttpSessionEvent |

#### Servlet Interface

Servlet Interface provides five methods. Out of these five methods, three methods are **Servlet life cycle** methods and rest two are non life cycle methods.



#### GenericServlet Class

GenericServlet is an abstract class that provides implementation of most of the basic servlet methods. This is a very important class.

**Methods of GenericServlet class**

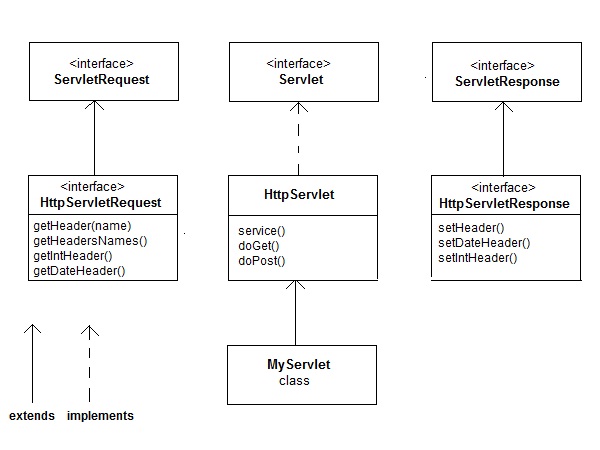
* public void init(ServletConfig)
* public abstract void service(ServletRequest request,ServletResposne response)
* public void destroy()
* public ServletConfig getServletConfig()
* public String getServletInfo()
* public ServletContext getServletContext()
* public String getInitParameter(String name)
* public Enumeration getInitParameterNames()
* public String getServletName()
* public void log(String msg)
* public void log(String msg, Throwable t)

#### HttpServlet class

HttpServlet is also an abstract class. This class gives implementation of various service() methods of **Servlet** interface.

To create a servlet, we should create a class that extends **HttpServlet** abstract class. The Servlet class that we will create, must not override service() method. Our servlet class will override only the doGet() and/or doPost() methods.

The service() method of **HttpServlet** class listens to the Http methods (GET, POST etc) from request stream and invokes doGet() or doPost() methods based on Http Method type.



### How a Servlet Application works

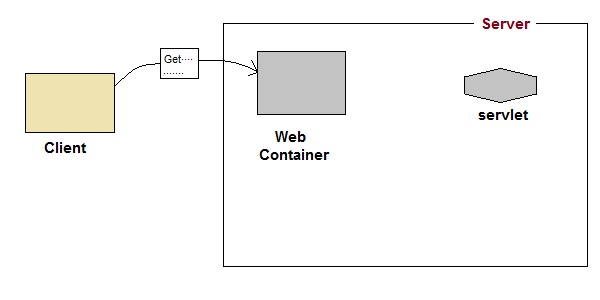
**Web container** is responsible for managing execution of servlets and JSP pages for Java EE application.

When a request comes in for a servlet, the server hands the request to the Web Container. **Web Container** is responsible for instantiating the servlet or creating a new thread to handle the request. Its the job of Web Container to get the request and response to the servlet. The container creates multiple threads to process multiple requests to a single servlet.

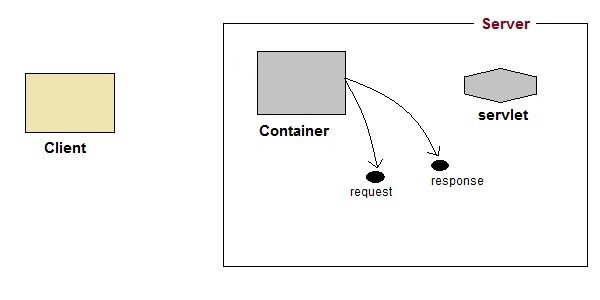
**Servlets don't have a main() method**. Web Container manages the life cycle of a Servlet instance.

#### Quick Revision on How a Servlet works

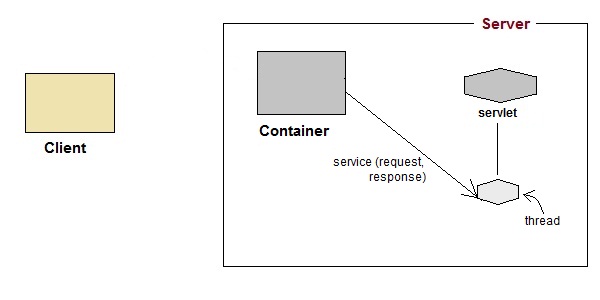
1. User sends request for a servlet by clicking a link that has URL to a servlet.



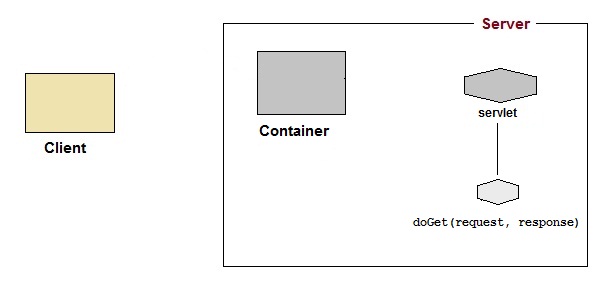
1. The container finds the servlet using **deployment descriptor** and creates two objects :
   1. **HttpServletRequest**
   2. **HttpServletResponse**



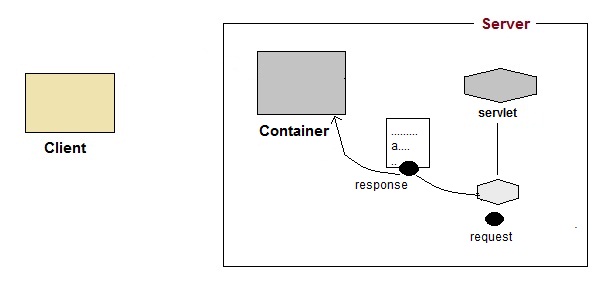
1. Then the container creates or allocates a thread for that request and calls the Servlet's service()method and passes the **request, response** objects as arguments.



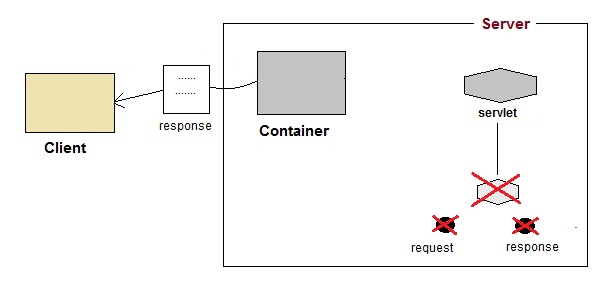
1. The service() method, then decides which servlet method, doGet() or doPost() to call, based on **HTTP Request Method**(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the service() will call Servlet's doGet() method.



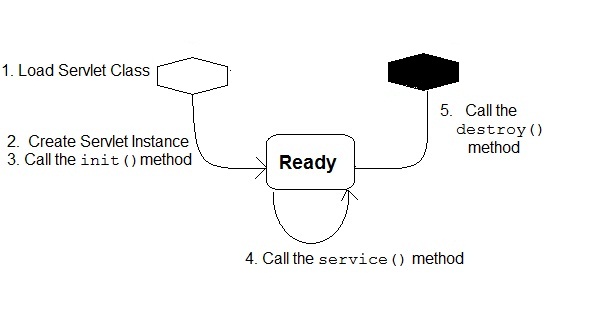
1. Then the Servlet uses response object to write the response back to the client.



1. After the service() method is completed the **thread** dies. And the request and response objects are ready for **garbage collection**.



# Servlet Life Cycle



1. **Loading Servlet Class:** A Servlet class is loaded when first request for the servlet is received by the Web Container.
2. **Servlet instance creation :**After the Servlet class is loaded, Web Container creates the instance of it. Servlet instance is created only once in the life cycle.
3. **Call to the init() method :** init() method is called by the Web Container on servlet instance to initialize the servlet.

**Signature of init() method :**

public void **init**(ServletConfig config) throws ServletException

1. **Call to the service() method :** The containers call the service() method each time the request for servlet is received. The service() method will then call the doGet() or doPost() methods based on the type of the HTTP request, as explained in previous lessons.

**Signature of service() method :**

public void **service**(ServletRequest request, ServletResponse response) throws ServletException, IOException

1. **Call to destroy() method:** The Web Container call the destroy() method before removing servlet instance, giving it a chance for cleanup activity.

### Introduction to Servlet Request

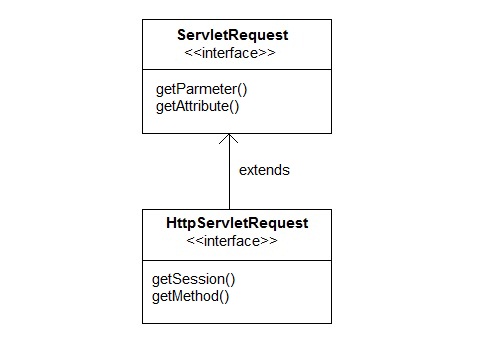
True job of a Servlet is to handle client request. Servlet API provides two important interfaces **javax.servlet.ServletRequest** and **javax.servlet.http.HttpServletRequest** to encapsulate client request. Implementation of these interfaces provide important information about client request to a servlet.

#### Some Important Methods of ServletRequest

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Object getAttribute(String name) | return attribute set on request object by name |
| Enumeration getAttributeName() | return an Enumeration containing the names of the attributes available inthis request |
| int getContentLength() | return size of request body |
| int getContentType() | return media type of request content |
| ServletInputStream getInputStream() | returns a input stream for reading binary data |
| String getParameter(String name) | returns value of parameter by name |
| String getLocalAddr() | returns the Internet Protocol(IP) address of the interface on which the request was received |
| Enumeration getParameterNames() | returns an enumeration of all parameter names |
| String[] getParameterValues(String name) | returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist |
| ServletContext getServletContext() | return the servlet context of current request. |
| String getServerName() | returns the host name of the server to which the request was sent |
| int getServerPort() | returns the port number to which the request was sent |
| boolean isSecure() | returns a boolean indicating whether this request was made using a secure channel, such as HTTPS. |
| void removeAttribute(String name) | removes an attribute from this request |
| void setAttribute(String name, Object o) | stores an attribute in this request. |

#### HttpServletRequest interface

**HttpServletRequest** interface adds the methods that relates to the **HTTP** protocol.



#### Some important methods of HttpServletRequest

|  |  |
| --- | --- |
| **Methods** | **Description** |
| String getContextPath() | returns the portion of the request URI that indicates the context of the request |
| Cookies getCookies() | returns an array containing all of the Cookie objects the client sent with this request |
| String getQueryString() | returns the query string that is contained in the request URL after the path |
| HttpSession getSession() | returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session |
| String getMethod() | Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT. |
| Part getPart(String name) | gets the Part with the given name |
| String getPathInfo() | returns any extra path information associated with the URL the client sent when it made this request. |
| String getServletPath() | returns the part of this request's URL that calls the servlet |

### Managing Session in Servlets

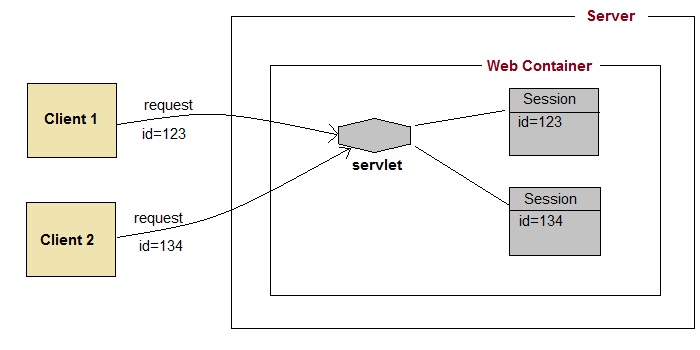
We all know that **HTTP** is a stateless protocol. All requests and responses are independent. But sometimes you need to keep track of client's activity across multiple requests. For eg. When a User logs into your website, not matter on which web page he visits after logging in, his credentials will be with the server, until he logs out. So this is managed by creating a session.

**Session Management** is a mechanism used by the **Web container** to store session information for a particular user. There are four different techniques used by Servlet application for session management. They are as follows:

1. **Cookies**
2. **Hidden form field**
3. **URL Rewriting**
4. **HttpSession**

Session is used to store everything that we can get from the client from all the requests the client makes.

#### How Session Works

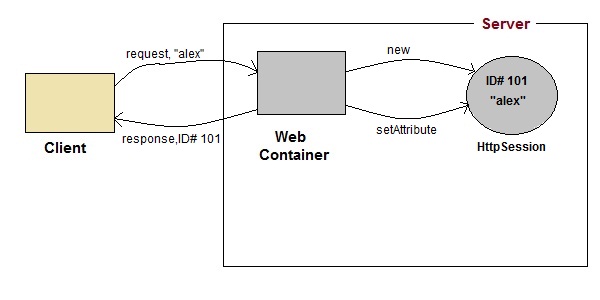


The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until its destroyed. So wherever the user goes, we will always have his information and we can always manage which user is doing what. Whenever a user wants to exit from your application, destroy the object with his information.

### What is HttpSession?

**HttpSession** object is used to store entire session with a specific client. We can store, retrieve and remove attribute from **HttpSession** object. Any servlet can have access to **HttpSession** object throughout the getSession() method of the **HttpServletRequest** object.

#### How HttpSession works



1. On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.
2. The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.
3. The **Web Container** uses this ID, finds the matching session with the ID and associates the session with the request.

#### HttpSession Interface



#### Some Important Methods of HttpSession

|  |  |
| --- | --- |
| **Methods** | **Description** |
| long getCreationTime() | returns the time when the session was created, measured in milliseconds since midnight January 1, 1970 GMT. |
| String getId() | returns a string containing the unique identifier assigned to the session. |
| long getLastAccessedTime() | returns the last time the client sent a request associated with the session |
| int getMaxInactiveInterval() | returns the maximum time interval, in seconds. |
| void invalidate() | destroy the session |
| boolean isNew() | returns true if the session is new else false |
| void setMaxInactiveInterval(int interval) | Specifies the time, in seconds,after servlet container will invalidate the session. |

#### Complete Example demonstrating usage of HttpSession

All the files mentioned below are required for the example.

**index.html**

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

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

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

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

</form>

**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>

**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");

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

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

if(pass.*equals*("1234"))

{

//creating a session

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

session.**setAttribute**("*user*", *name*);

response.**sendRedirect**("Welcome");

}

}

}

**Welcome.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

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

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

throws ServletException, IOException {

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

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

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

String **user** = (String)session.getAttribute("*user*");

out.println("Hello "+**user**);

}

}

### Using Cookies for Session Management

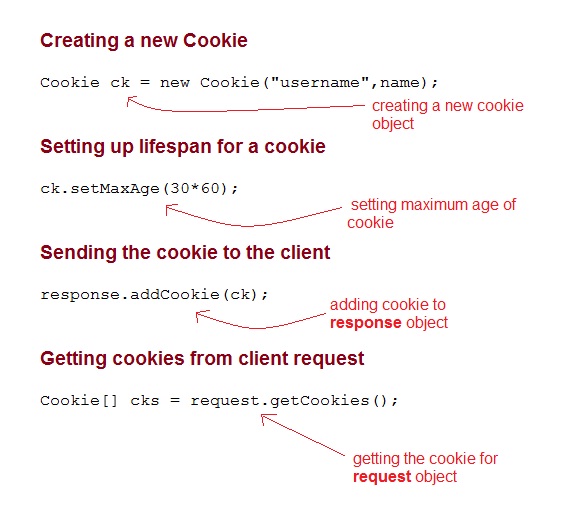
**Cookies** are small pieces of information that are sent in response from the web server to the client. **Cookies** are the simplest technique used for storing client state.

**Cookies** are stored on client's computer. They have a lifespan and are destroyed by the client browser at the end of that lifespan.

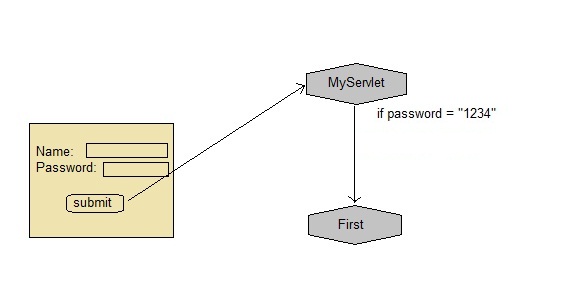
Using Cookies for storing client state has one shortcoming though, if the client has turned of COokie saving settings in his browser then, client state can never be saved because the browser will not allow the application to store cookies.

#### Cookies API

Cookies are created using **Cookie** class present in Servlet API. Cookies are added to **response**object using the addCookie() method. This method sends cookie information over the HTTP response stream. getCookies() method is used to access the cookies that are added to response object.



#### Example demonstrating usage of Cookies



Below mentioned files are required for the example:

**index.html**

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

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

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

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

</form>

**web.xml**

<web-app...>

<servlet>

<servlet-name>**validate**</servlet-name>

<servlet-class>**MyServlet**</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>**validate**</servlet-name>

<url-pattern>/validate</url-pattern>

</servlet-mapping>

<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>

<welcome-file-list>

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

</welcome-file-list>

</web-app>

**MyServlet.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class *MyServlet* **extends** HttpServlet {

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

throws ServletException, IOException {

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

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

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

if(pass.*equals*("1234"))

{

Cookie **ck** = new Cookie("*username*",*name*);

response.**addCookie**(*ck*);

response.sendRedirect("*First*");

}

}

}

**First.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class *First* **extends** HttpServlet {

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

throws ServletException, IOException {

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

PrintWriter out = response.getWriter();

Cookie[] **cks** = request.getCookies();

out.println("Welcome "+**cks[0].getValue()**);

}

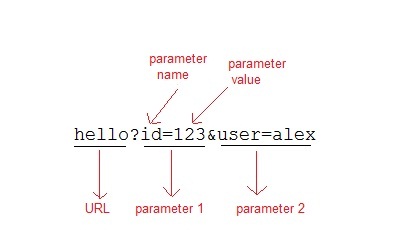
}

### Using URL Rewriting for Session Management

If the client has disabled cookies in the browser then session management using cookie wont work. In that case **URL Rewriting** can be used as a backup. **URL rewriting** will always work.

In URL rewriting, a token(parameter) is added at the end of the URL. The token consist of name/value pair seperated by an equal(=) sign.

**For Example:**



When the User clicks on the URL having parameters, the request goes to the **Web Container** with extra bit of information at the end of URL. The **Web Container** will fetch the extra part of the requested URL and use it for session management.

The getParameter() method is used to get the parameter value at the server side.

#### Example demonstrating usage of URL rewriting

Below mentioned files are required for the example:

**index.html**

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

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

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

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

</form>

**web.xml**

<web-app...>

<servlet>

<servlet-name>**validate**</servlet-name>

<servlet-class>**MyServlet**</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>**validate**</servlet-name>

<url-pattern>/validate</url-pattern>

</servlet-mapping>

<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>

<welcome-file-list>

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

</welcome-file-list>

</web-app>

**MyServlet.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class *MyServlet* **extends** HttpServlet {

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

throws ServletException, IOException {

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

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

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

if(pass.*equals*("1234"))

{

response.sendRedirect(*"First?user\_name="+****name****+""*);

}

}

}

**First.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class *First* **extends** HttpServlet {

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

throws ServletException, IOException {

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

PrintWriter out = response.getWriter();

String **user** = request.getParameter("*user\_name*");

out.println("Welcome "+user);

}

}

### Using Hidden Form Field for Session Management

Hidden form field can also be used to store session information for a particular client. In case of hidden form field a hidden field is used to store client state. In this case user information is stored in hidden field value and retrieved from another servlet.

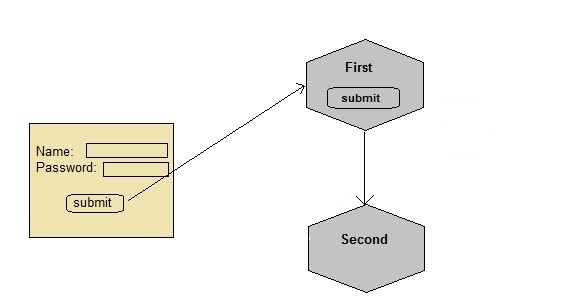
#### Advantages :

* Does not have to depend on browser whether the cookie is disabled or not.
* Inserting a simple HTML Input field of type hidden is required. Hence, its easier to implement.

#### Disadvantage :

* Extra form submission is required on every page. This is a big overhead.

#### Example demonstrating usage of Hidden Form Field for Session



Below mentioned files are required for the example:

**index.html**

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

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

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

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

</form>

**web.xml**

<web-app...>

<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>

<servlet>

<servlet-name>**Second**</servlet-name>

<servlet-class>**Second**</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>**Second**</servlet-name>

<url-pattern>/Second</url-pattern>

</servlet-mapping>

<welcome-file-list>

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

</welcome-file-list>

</web-app>

**First.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class *First* **extends** HttpServlet {

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

throws ServletException, IOException {

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

PrintWriter out = response.getWriter();

//getting value submitted in form from HTML file

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

//creating a new hidden form field

out.println("<form action='Second'>");

out.println("<input type='hidden' name='user' value='"+**user**+"'>");

out.println("<input type='submit' value='submit' >");

out.println("</form>");

}

}

**Second.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class *Second* **extends** HttpServlet {

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

throws ServletException, IOException {

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

PrintWriter out = response.getWriter();

//getting parameter from the hidden field

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

out.println("Welcome "+**user**);

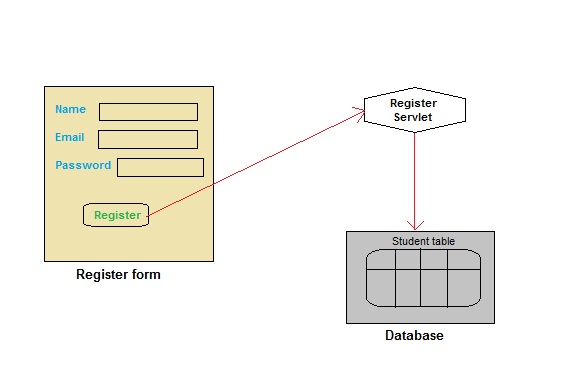
}

}

Like we created a hidden field in **First** Servlet, populated the value of user, and sent it to the **Second**Servlet, now Second servlet also has the user information. Similarly we will have to keep sending this information, wherever we need this, using hidden fields.

### Registration form in Servlet

In this example you will see how to develop a registration form in Servlet. To develop a registration form you will need to connect your servlet application with database. Here we are using **MySQL**database.



#### Create a Table in your Database

create table **Student**

(

**name** varchar(60),

**email** varchar(60),

**pass** varchar(100)

)

#### index.html

<html>

<head>

<title>Register form</title>

</head>

<body>

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

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

**Email ID**:<input type="text" name="**email**" /><br/>

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

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

</form>

</body>

</html>

#### Register.java

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.sql.\*;

public class *Register* **extends** HttpServlet {

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

throws ServletException, IOException {

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

PrintWriter out = response.getWriter();

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

String email = request.getParameter("email");

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

try{

//loading drivers for mysql

Class.forName("*com.mysql.jdbc.Driver*");

//creating connection with the database

Connection con=DriverManager.getConnection

("*jdbc:mysql:/ /localhost:3306/test*","*username*","*password*");

PreparedStatement **ps**=con.prepareStatement

("*insert into Student values(?,?,?)*");

ps.setString(1, name);

ps.setString(2, email);

ps.setString(3, pass);

int i=ps.**executeUpdate**();

if(i>0)

{

out.println("*You are sucessfully registered*");

}

}

catch(Exception se)

{

se.printStackTrace();

}

}

}

#### web.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="3.0"

xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" >

<servlet>

<servlet-name>**register**</servlet-name>

<servlet-class>**Register**</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>**register**</servlet-name>

<url-pattern>**/register**</url-pattern>

</servlet-mapping>

<welcome-file-list>

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

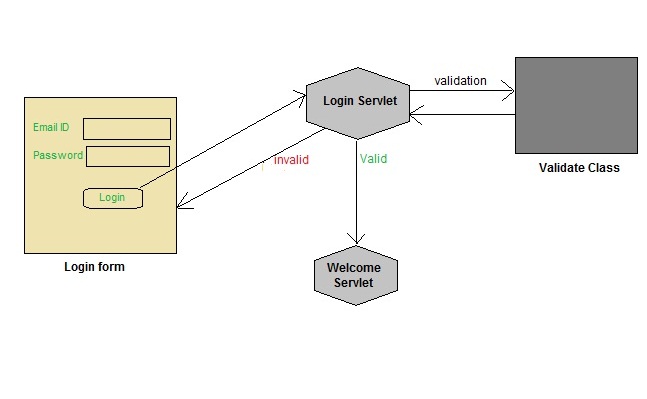
</welcome-file-list>

</web-app>

### Login System in Servlet

In this example we will show you how to develop a login form using servlet. Here we are using **MySql**database. List of file to be created are:

* **index.html**
* **Login.java**
* **Validate.java**
* **Welcome.java**
* **web.xml**



To try this application you will need to create a table in your database and enter some record into it. Refer the previos Lesson for creating table.

#### index.html

<html>

<head>

<title>login form</title>

</head>

<body>

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

**Email ID**:<input type="text" name="**email**" /><br/>

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

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

</form>

</body>

</html>

#### Login.java

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.sql.\*;

public class **Login** extends **HttpServlet** {

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

throws ServletException, IOException {

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

PrintWriter out = response.getWriter();

String **email** = request.getParameter("*email*");

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

if(**Validate**.checkUser(**email**, **pass**))

{

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

rs.forward(request, response);

}

else

{

out.println("*Username or Password incorrect*");

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

rs.include(request, response);

response.append(“<div> <a> … ></div>”)

}

}

}

#### Validate.java

import java.sql.\*;

public class **Validate**

{

public static boolean **checkUser**(String email,String pass)

{

boolean st =false;

try{

//loading drivers for mysql

Class.forName("*com.mysql.jdbc.Driver*");

**//creating connection with the database**

Connection con=DriverManager.getConnection

("*jdbc:mysql:/ /localhost:3306/test*","*root*","*studytonight*");

PreparedStatement ps =con.prepareStatement

("*select \* from register where email=? and pass=?*");

ps.setString(1, email);

ps.setString(2, pass);

ResultSet rs =ps.executeQuery();

st = rs.next();

}catch(Exception e)

{

e.printStackTrace();

}

return st;

}

}

#### Welcome.java

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.sql.\*;

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();

out.println("*Welcome user*");

}

}

#### web.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" >

<servlet>

<servlet-name>login</servlet-name>

<servlet-class>Login</servlet-class>

</servlet>

<servlet>

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

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

</servlet>

<servlet-mapping>

<servlet-name>login</servlet-name>

<url-pattern>/login</url-pattern>

</servlet-mapping>

<servlet-mapping>

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

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

</servlet-mapping>

</web-app>

### Email Sending using Servlet

In this example we will see how to send an email in Servlet application. We will be using the **JavaMail**API that provides all the classes required for sending an email. JavaMail API encapsulates two important packages **javax.mail** and **javax.mail.internet**. These packages provide classes that can be used to send and recieve simple emails. You simply need an Internet connection to send email using this simple Application.

Following are the names of Files to be created :

* **index.html** will get the input from user
* **MailApp.java** servlet file will control the request and response. It will invoke **send()** of **SendMail**class that we have created to send the mail.
* **SendMail.java**, a java class that contains method to send mail.

#### index.html

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

**To**:<input type="text" name="to" /><br/>

**Subject**:<input type="text" name="subject" /><br/>

**Message**:<input type="text" name="message" /><br/>

**Your Email id**:<input type="text" name="user" ><br/>

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

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

</form>

#### MailApp.java

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class **MailApp** extends **HttpServlet** {

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

throws ServletException, IOException {

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

PrintWriter out = response.getWriter();

String **to** = request.getParameter("to");

String **subject** = request.getParameter("subject");

String **message** = request.getParameter("message");

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

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

*SendMail.send*(**to**,**subject**, **message**, **user**, **pass**);

out.println("Mail send successfully");

}

}

#### SendMail.java

import java.io.\*;

import java.util.\*;

import javax.mail.\*;

import javax.mail.internet.\*;

public class **SendMail**

{

public static void **send**(String to, String sub,

String msg, final String user,final String pass)

{

//create an instance of Properties Class

Properties **props** = new Properties();

*/\* Specifies the IP address of your default mail server*

*for e.g if you are using gmail server as an email sever*

*you will pass smtp.gmail.com as value of mail.smtp host.*

*As shown here in the code.*

*Change accordingly, if your email id is not a gmail id*

*\*/*

props.put("mail.smtp.host", "smtp.gmail.com");

*//below mentioned mail.smtp.port is optional*

props.put("mail.smtp.port", "587");

props.put("mail.smtp.auth", "true");

props.put("mail.smtp.starttls.enable", "true");

*/\* Pass Properties object(props) and Authenticator object*

*for authentication to Session instance*

*\*/*

Session **session** = Session.**getInstance**(props,new javax.mail.Authenticator()

{

protected PasswordAuthentication **getPasswordAuthentication**()

{

return new PasswordAuthentication(user,pass);

}

});

try

{

*/\* Create an instance of MimeMessage,*

*it accept MIME types and headers*

*\*/*

**MimeMessage** message = new MimeMessage(session);

message.setFrom(new InternetAddress(user));

message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));

message.setSubject(sub);

message.setText(msg);

*/\* Transport class is used to deliver the message to the recipients \*/*

**Transport**.send(message);

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

#### web.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd">

<servlet>

<servlet-name>**mail**</servlet-name>

<servlet-class>**MailApp**</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>**mail**</servlet-name>

<url-pattern>**/mail**</url-pattern>

</servlet-mapping>

<welcome-file-list>

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

</welcome-file-list>

</web-app>

In this example the **SendMail** class is the main class, as it will setup a connection with the Email Server, it will authenticate the username and password and will then use the information entered in the form like, receiver's email address, the email text etc and will send the email.

**MailApp** is the servlet ti which the HTML page form will submit the inputs, this Servlet will fetch all the parameters and call the **send()** method of **SendMail** Java class.