Contents

[**Introduction to Web** 2](#_Toc473093150)

[**Introduction to Servlet** 7](#_Toc473093151)

[**Servlet API** 10](#_Toc473093152)

[**How a Servlet Application works** 14](#_Toc473093153)

[**Servlet Life Cycle** 18](#_Toc473093154)

[**Introduction to Request Dispatcher** 20](#_Toc473093155)

[Introduction to sendRedirect() Method 26](#_Toc473093156)

[**Introduction to ServletConfig interface** 27](#_Toc473093157)

[**Introduction to ServletContext Interface** 29](#_Toc473093158)

[**Introduction to Attribute** 33](#_Toc473093159)

[**Managing Session in Servlets** 35](#_Toc473093160)

[**What is HttpSession?** 36](#_Toc473093161)

[Using Cookies for Session Management 41](#_Toc473093162)

[Using URL Rewriting for Session Management 45](#_Toc473093163)

[**Using Hidden Form Field for Session Management** 48](#_Toc473093164)

[ServletContextEvent and ServletContextListener? 51](#_Toc473093165)

[What is ServletRequestAttributeEvent? 56](#_Toc473093166)

[HttpSessionEvent and HttpSessionListener 58](#_Toc473093167)

[ServletRequestEvent and ServletRequestListener 60](#_Toc473093168)

[**Introduction to Filter API** 62](#_Toc473093169)

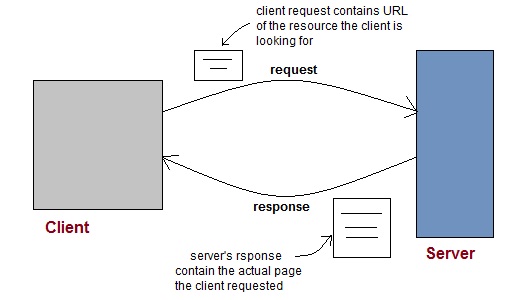
[Registration form in Servlet 68](#_Toc473093170)

[**Login System in Servlet** 71](#_Toc473093171)

[**Email Sending using Servlet** 75](#_Toc473093172)

**Introduction to Web**

Web consists of billions of clients and server connected through wires and wireless networks. The web clients make requests to web server. The web server receives the request, finds the resources and return the response to the client. When a server answers a request, it usually sends some type of content to the client. The client uses web browser to send request to the server. The server often sends response to the browser with a set of instructions written in HTML(HyperText Markup Language). All browsers know how to display HTML page to the client.

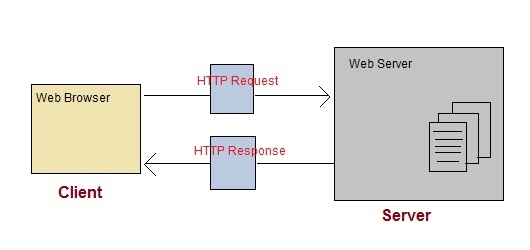


**Web Application**

A website is a collection of static files(webpages) such as HTML pages, images, graphics etc. A **Web application** is a web site with dynamic functionality on the server. **Google**, **Facebook**, **Twitter** are examples of web applications.

**HTTP (Hypertext Transfer Protocol)**

* HTTP is a protocol that clients and servers use on the web to communicate.
* It is similar to other internet protocols such as SMTP(Simple Mail Transfer Protocol) and FTP(File Transfer Protocol) but there is one fundamental difference.
* HTTP is a **stateless protocol** i.e HTTP supports only one request per connection. This means that with HTTP the clients connect to the server to send one request and then disconnects. This mechanism allows more users to connect to a given server over a period of time.
* The client sends an HTTP request and the server answers with an HTML page to the client, using HTTP.



**HTTP Methods**

HTTP request can be made using a variety of methods, but the ones you will use most often are **Get** and **Post**. The method name tells the server the kind of request that is being made, and how the rest of the message will be formated.

**HTTP Methods and Descriptions :**

| **Method Name** | **Description** |
| --- | --- |
| OPTIONS | Request for communication options that are available on the request/response chain. |
| GET | Request to retrieve information from server using a given URI. |
| HEAD | Identical to GET except that it does not return a message-body, only the headers and status line. |
| POST | Request for server to accept the entity enclosed in the body of HTTP method. |
| DELETE | Request for the Server to delete the resource. |
| CONNECT | Reserved for use with a proxy that can switch to being a tunnel. |
| PUT | This is same as POST, but POST is used to create, PUT can be used to create as well as update. It replaces all current representations of the target resource with the uploaded content. |

**Difference between GET and POST requests**

| **GET Request** | **POST Request** |
| --- | --- |
| Data is sent in header to the server | Data is sent in the request body |
| Get request can send only limited amount of data | Large amount of data can be sent. |
| Get request is not secured because data is exposed in URL | Post request is secured because data is not exposed in URL. |
| Get request can be bookmarked and is more efficient. | Post request cannot be bookmarked. |

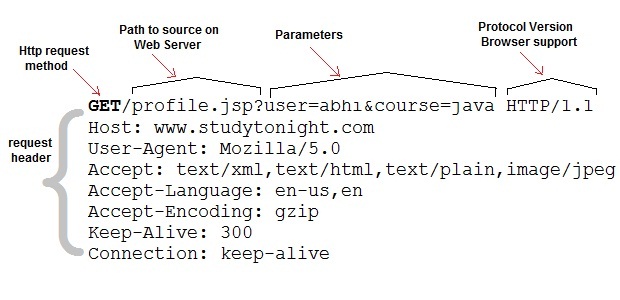
**General Difference between PUT and POST methods**

Following are some basic differences between the PUT and the POST methods :

* **POST** to a URL creates a child resource at a server defined URL while **PUT** to a URL creates/replaces the resource in its entirety at the client defined URL.
* POST creates a child resource, so POST to /books will create a resources that will live under the /booksresource. Eg. /books/1. Sending the same post request twice will create two resources.
* PUT is for creating or replacing a resource at a URL known by the client.
* PUT must be used for CREATE when the client already knows the url before the resource is created.
* PUT replaces the resource at the known url if it already exists, so sending the same request twice has no effect. In other words, calls to PUT are **idempotent.**

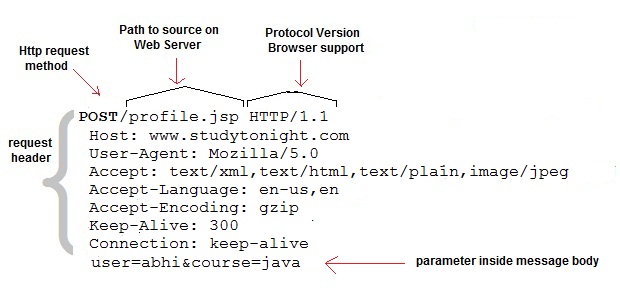
**Anatomy of an HTTP GET request**

Get request contains path to server and the parameters added to it.



**Anatomy of an HTTP POST request**

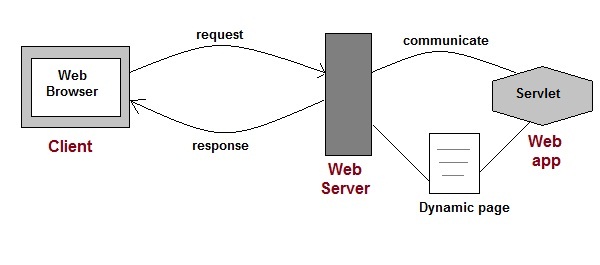
Post requests are used to make more complex requests on the server. For instance, if a user has filled a form with multiple fields and the application wants to save all the form data to the database. Then the form data will be sent to the server in POST request body, which is also known as Message body.



**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.

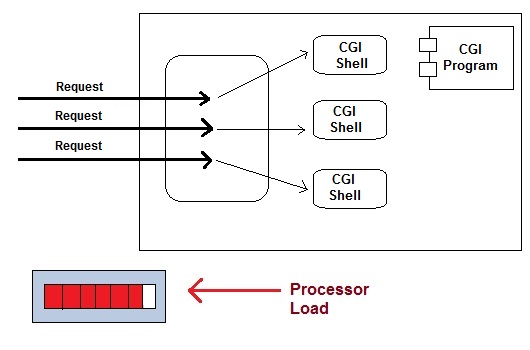


As Servlet Technology uses Java, web applications made using Servlet are **Secured**, **Scalable** and **Robust**.

**CGI (Common Gateway Interface)**

Before Servlets, CGI(Common Gateway Interface) programming was used to create web applications. Here's how a CGI program works :

* User clicks a link that has URL to a dynamic page instead of a static page.
* The URL decides which CGI program to execute.
* Web Servers run the CGI program in seperate OS shell. The shell includes OS enviroment and the process to execute code of the CGI program.
* The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.



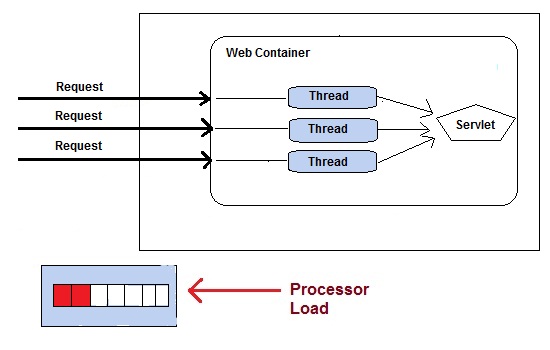
**Drawbacks of CGI programs**

* High resposne time because CGI programs execute in their own OS shell.
* CGI is not scalable.
* CGI programs are not always secure or object-oriented.
* It is Platform dependent.

Because of these disadvantages, developers started looking for better CGI solutions. And then Sun Microsystems developed **Servlet** as a solution over traditional CGI technology.

**Advantages of using Servlets**

* Less response time because each request runs in a separate thread.
* Servlets are scalable.
* Servlets are robust and object oriented.
* Servlets are platform independent.



**Servlet API**

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**

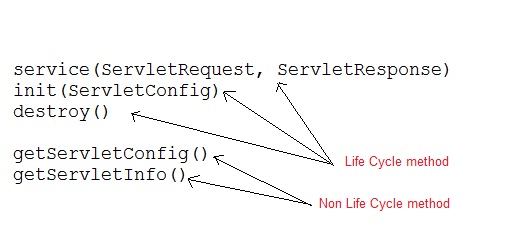
|  |  |
| --- | --- |
| **INTERFACES** | **CLASSES** |
| Servlet | ServletInputStream |
| ServletContext | ServletOutputStream |
| ServletConfig | ServletRequestWrapper |
| ServletRequest | ServletResponseWrapper |
| ServletResponse | ServletRequestEvent |
| ServletContextListener | ServletContextEvent |
| 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**

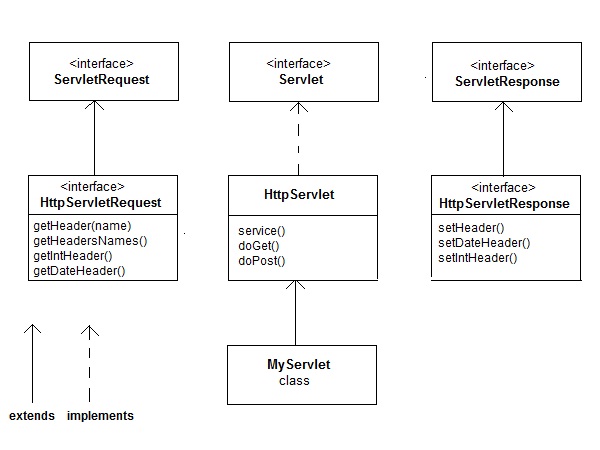
* public void init(ServletConfig)
* public abstract void service(ServletRequest request,ServletResposne response)
* public void destroy()
* public ServletConfig getServletConfig()
* public String getServletInfo()
* public ServletContext getServletContext()
* public String getInitParameter(String name)
* 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**

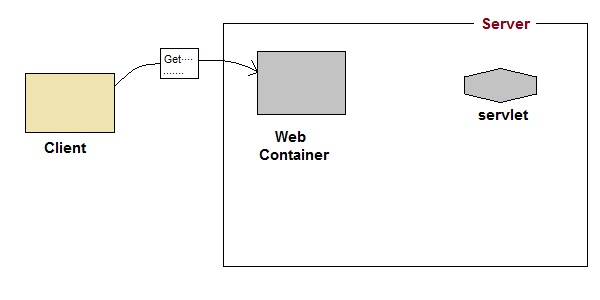
**Web container** is responsible for managing execution of servlets and JSP pages for Java EE application.

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.

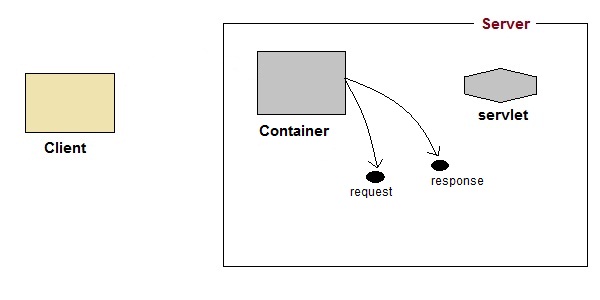
**Servlets don't have a main() method**. Web Container manages the life cycle of a Servlet instance.

**Quick Revision on How a Servlet works**

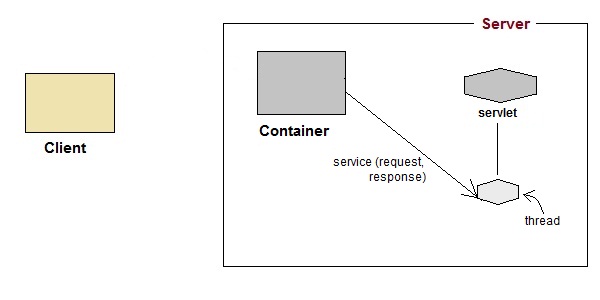
1. User sends request for a servlet by clicking a link that has URL to a servlet.



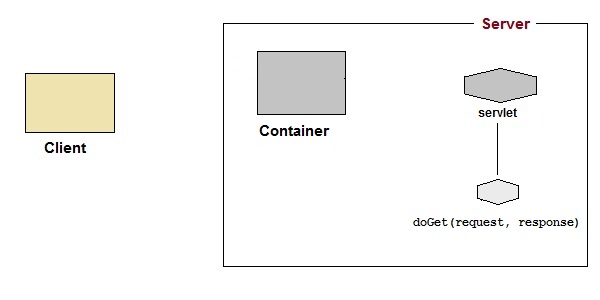
1. The container finds the servlet using **deployment descriptor** and creates two objects :
   1. **HttpServletRequest**
   2. **HttpServletResponse**



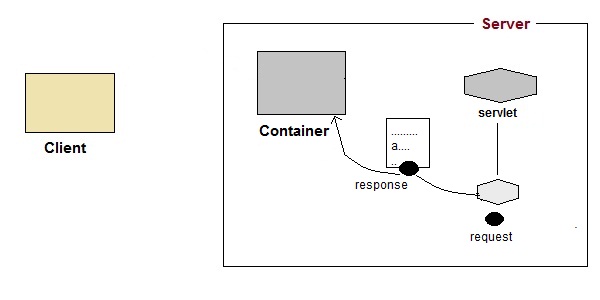
1. Then the container creates or allocates a thread for that request and calls the Servlet's service()method and passes the **request, response** objects as arguments.



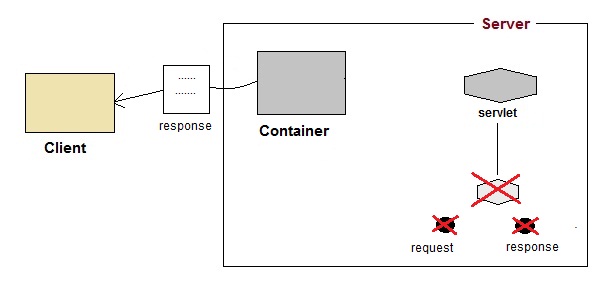
1. The service() method, then decides which servlet method, doGet() or doPost() to call, based on **HTTP Request Method**(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the service() will call Servlet's doGet() method.



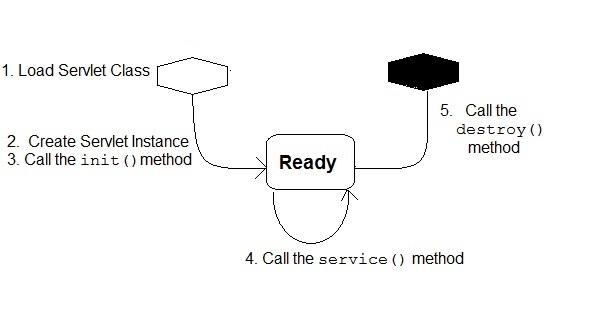
1. Then the Servlet uses response object to write the response back to the client.



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() methos based ont eh 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 Request Dispatcher**

**RequestDispatcher** is an interface, implementation of which defines an object which can dispatch request to any resources(such as HTML, Image, JSP, Servlet) on the server.

**Methods of RequestDispatcher**

**RequestDispatcher** interface provides two important methods

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void forward(ServletRequest request, ServletResponse response) | forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server |
| void include(ServletRequest request, ServletResponse response) | includes the content of a resource (servlet, JSP page, HTML file) in the response |

**How to get an Object of RequestDispatcher**

getRequestDispatcher() method of **ServletRequest** returns the object of **RequestDispatcher**.

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

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



**OR**

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

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



**Example demonstrating usage of RequestDispatcher**

In this example, we will show you how RequestDispatcher is used to **forward** or **include** response of a resource in a Servlet. Here we are using **index.html** to get username and password from the user, **Validate** Servlet will validate the password entered by the user, if the user has entered "studytonight" as password, then he will be forwarded to **Welcome** Servlet else the user will stay on the index.html page and an error message will be displayed.

**Files to be created :**

* **index.html** will have form fields to get user information.
* **Validate.java** will validate the data entered by the user.
* **Welcome.java** will be the welcome page.
* **web.xml** , the deployment descriptor.

**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>

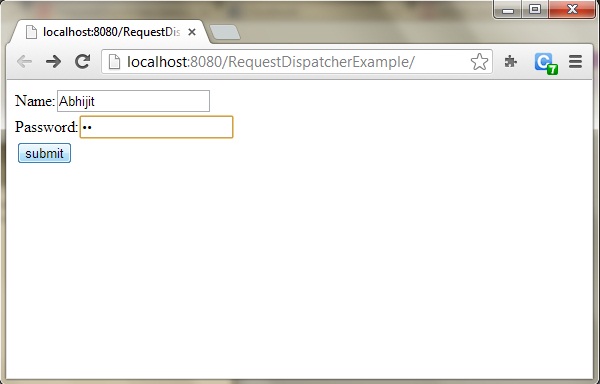
<welcome-file-list>

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

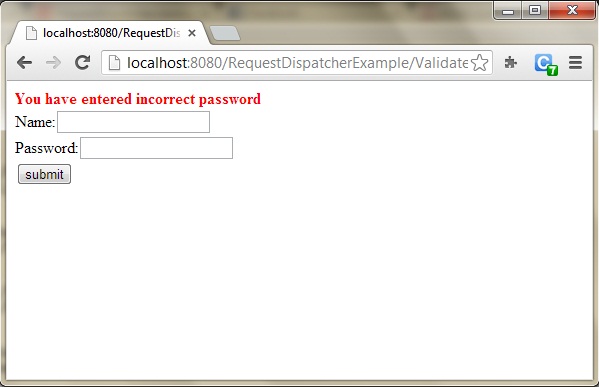
</welcome-file-list>

</web-app>

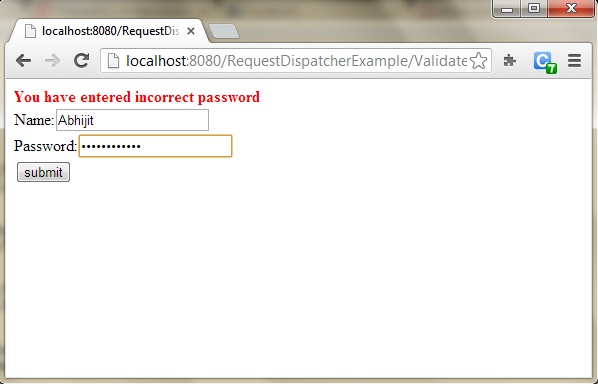
This will be the first screen. You can enter your Username and Password here.



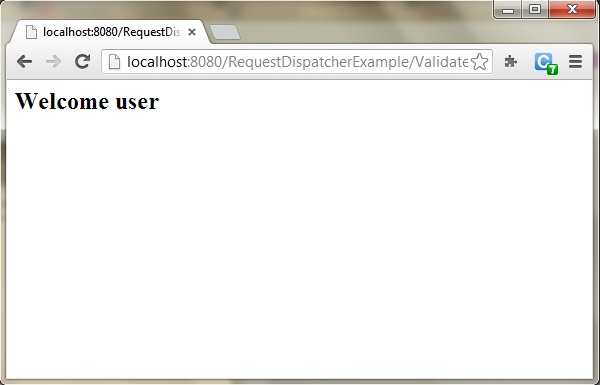
When you click on Submit, Password will be validated, if it is not 'studytonight' , error message will be displayed.



Enter any Username, but enter 'studytonight' as password.



Password will be successfully validated and you will be directed to the Welcome Servlet.



### Introduction to sendRedirect() Method

sendRedirect() method redirects the response to another resource. This method actually makes the client(browser) to create a new request to get to the resource. The client can see the new url in the browser.

**sendRedirect()** accepts relative **URL**, so it can go for resources inside or outside the server.

#### sendRedirect() and Request Dispatcher

The main difference between a **redirection** and a **request dispatching** is that, redirection makes the client(browser) create a new request to get to the resource, the user can see the new URL while request dispatch get the resource in same request and URL does not changes.

Also, another very important difference is that, sendRedirect() works on **response** object while request dispatch work on **request** object.

#### Example demonstrating usage of sendRedirect()

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

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

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

throws ServletException, IOException {

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

PrintWriter *out* = response.getWriter();

try {

response.**sendRedirect**("http:/ /www.studytonight.com");

}finally {

out.close();

}

}

}

**Introduction to ServletConfig interface**

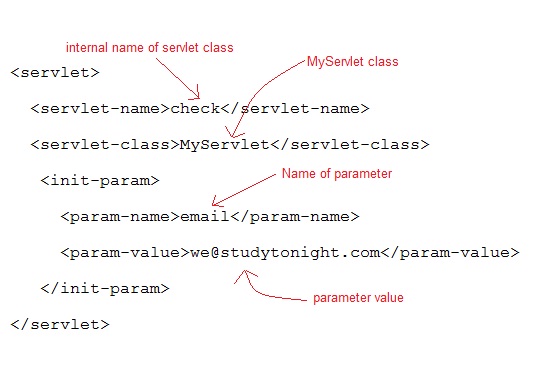
When the **Web Container** initializes a servlet, it creates a **ServletConfig** object for the servlet. ServletConfig object is used to pass information to a servlet during initialization by getting configuration information from **web.xml**(Deployment Descriptor).

**Methods of ServletConfig**

* String getInitParameter(String name): returns a String value initialized parameter, or NULL if the parameter does not exist.
* Enumeration getInitParameterNames(): returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.
* ServletContext getServletContext(): returns a reference to the ServletContext
* String getServletName(): returns the name of the servlet instance

**How to Initialize a Servlet inside web.xml**

**In the Deployment Descriptor(web.xml) file,**



**Or, Inside the Servlet class, using following code,**

ServletConfig **sc** = getServletConfig();

out.println(sc.**getInitParameter**("email"));

**Example demonstrating usage of ServletConfig**

**web.xml**

<web-app...>

<servlet>

<servlet-name>check</servlet-name>

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

<init-param>

<param-name>email</param-name>

<param-value>we@studytonight.com</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>check</servlet-name>

<url-pattern>/check</url-pattern>

</servlet-mapping>

<welcome-file-list>

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

</welcome-file-list>

</web-app>

**MyServlet class :**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

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

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

throws ServletException, IOException {

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

PrintWriter *out* = response.getWriter();

ServletConfig *sc*=**getServletConfig()**;

out.println(**sc.getInitParameter("email")**);

}

}

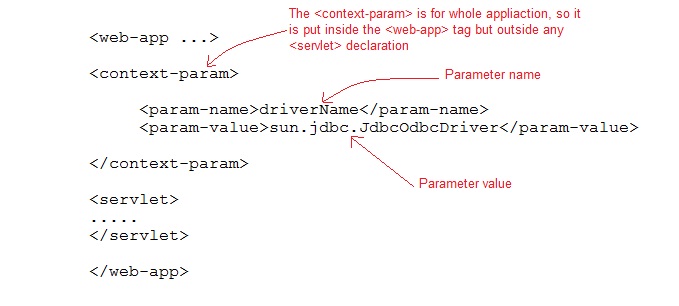
**Introduction to ServletContext Interface**

For every **Web application** a **ServletContext** object is created by the web container. ServletContext object is used to get configuration information from **Deployment Descriptor**(web.xml) which will be available to any servlet or JSPs that are part of the web app.

**Some Important method of ServletContext**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Object getAttribute(String name) | returns the container attribute with the given name, or NULL if there is no attribute by that name. |
| String getInitParameter(String name) | returns parameter value for the specified parameter name, or NULL if the parameter does not exist |
| Enumeration getInitParameterNames() | returns the names of the context's initialization parameters as an Enumeration of String objects |
| void setAttribute(String name,Object obj) | set an object with the given attribute name in the application scope |
| void removeAttribute(String name) | removes the attribute with the specified name from the application context |

**How Context Parameter is Initialized inside web.xml**



**How to get the Object of ServletContext**

ServletContext **app** = getServletContext();

*OR*

ServletContext **app** = getServletConfig().getServletContext();

**Advantages of ServletContext**

* Provides communication between servlets
* Available to all servlets and JSPs that are part of the web app
* Used to get configuration information from web.xml

**Difference between Context Init Parameters and Servlet Init Parameter**

|  |  |
| --- | --- |
| **Context Init parameters** | **Servlet Init parameter** |
| Available to all servlets and JSPs that are part of web | Available to only servlet for which the <init-param> was configured |
| Context Init parameters are initialized within the <web-app> not within a specific <servlet> elements | Initialized within the <servlet> for each specific servlet. |
| ServletContext object is used to get Context Init parameters | ServletConfig object is used to get Servlet Init parameters |
| Only one ServletContext object for entire web app | Each servlet has its own ServletConfig object |

**Example demonstrating usage of ServletContext**

**web.xml**

<web-app ...>

**<context-param>**

**<param-name>driverName</param-name>**

**<param-value>sun.jdbc.JdbcOdbcDriver</param-value>**

**</context-param>**

<servlet>

<servlet-name>hello</servlet-name>

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

</servlet>

<servlet-mapping>

<servlet-name>hello</servlet-name>

<url-pattern>/hello</url-pattern>

</servlet-mapping>

</web-app>

**MyServlet class :**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

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

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

throws ServletException, IOException {

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

PrintWriter *out* = response.getWriter();

ServletContext sc = **getServletContext()**;

out.println(sc.**getInitParameter**("driverName"));

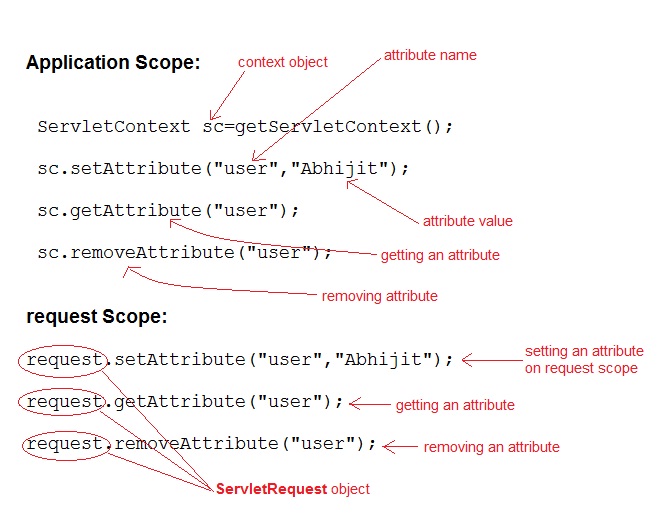
}

}

**Introduction to Attribute**

An **attribute** is an object that is used to share information in a web app. Attribute allows Servlets to share information among themselves. Attributes can be SET and GET from one of the following scopes :

1. request
2. session
3. application



**How to SET an Attribute**

public void setAttribute(String name, Object obj) method is used to SET an Attribute.

**Example demonstrating Setting Attribute**

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();

ServletContext sc = getServletContext();

sc.**setAttribute**("*user*","*Abhijit*"); //setting attribute on context scope

}

}

**How to GET an Attribute**

Object getAttribute(String name) method is used to GET an attribute.

**Example demonstrating getting a value of set Attribute**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

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

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

throws ServletException, IOException {

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

PrintWriter *out* = response.getWriter();

ServletContext sc = getServletContext();

String str = sc.**getAttribute**("*user*"); //getting attribute from context scope

out.println("Welcome"+str); // Prints : Welcome Abhijit

}

}

**Managing Session in Servlets**

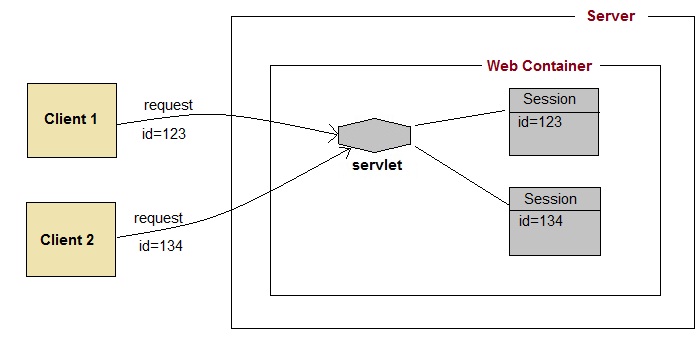
We all know that **HTTP** is a stateless protocol. All requests and responses are independent. But sometimes you need to keep track of client's activity across multiple requests. For eg. When a User logs into your website, no matter on which web page he visits after logging in, his credentials will be 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**

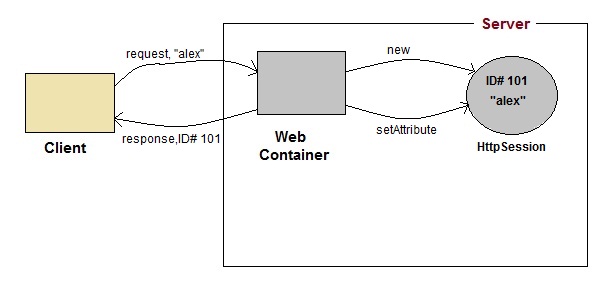


The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until its destroyed. So wherever the user goes, we will always have his information and we can always 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.

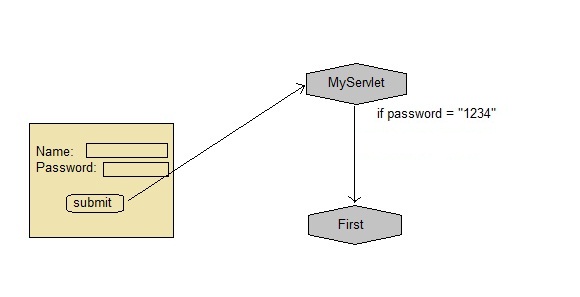
Using Cookies for storing client state has one shortcoming though, if the client has turned off COokie saving settings in his browser then, client state can never be saved because the browser will not allow the application to store cookies.

#### 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()**);

}

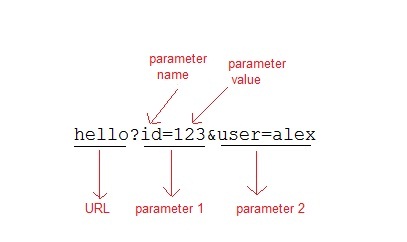
}

### 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.

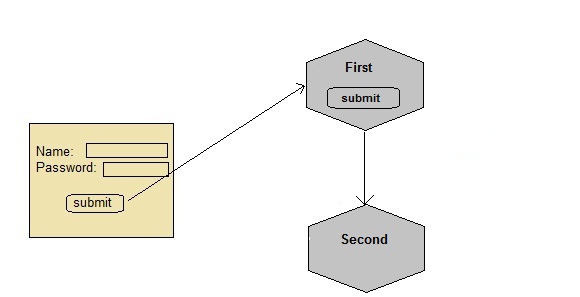
**Advantages :**

* 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.

### ServletContextEvent and ServletContextListener?

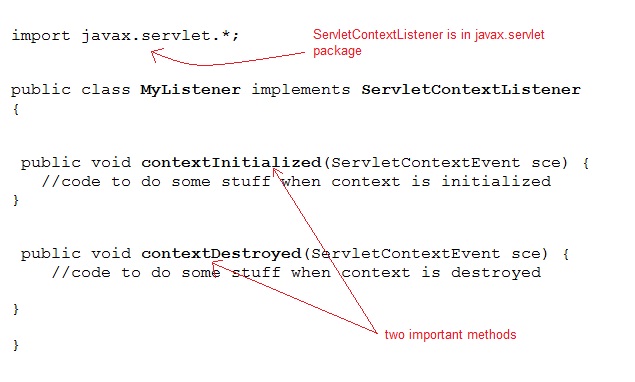
**ServletContextEvent** class gives notifications about changes to the servlet context of a web application. **ServletContextListener** receives the notifications about changes to the servlet context and perform some action. ServletContextListener is used to perform important task at the time when context is initialized and destroyed. In short, ServletContextEvent and ServletContextListener works in pair, whenever Servlet COntext changes, ServletContextEvent publishes a notification which is received by ServletContextListener and then, based on that certain tasks are performed by it.

#### Methods of ServletContextListener Interface

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void contextDestroyed(ServletContextEvent e) | is invoked when the application is destroyed. |
| void contextInitialized(ServletContextEvent e) | is invoked when the application is initialized. |

#### Making and Using a context listener

Context listener is not a servlet or JSP, it's a class that implements **ServletContextListener** interface and provides definition of contextDestroyed() and contextInitialized().

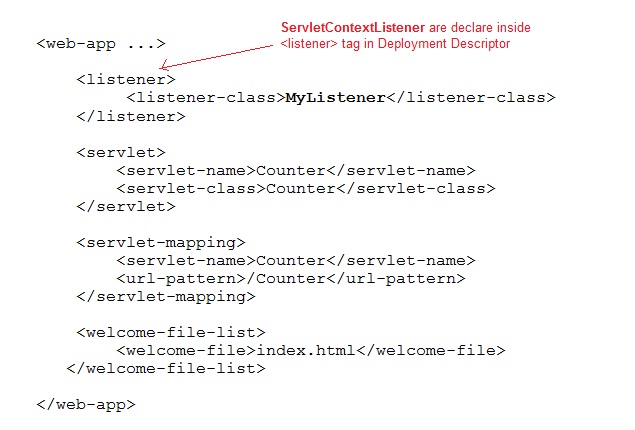


#### Example demontrating usage of ServletContextListener

**index.html**

<a href="**Counter**">Total Page views</a>

**web.xml**



For this example we will have to create a table named **counter** with a column named **pageview** to save the number of pageviews.

**MyListener.java**

import java.sql.\*;

import javax.servlet.\*;

public class **MyListener** *implements* **ServletContextListener**

{

ServletContext **ctx**;

Connection con;

Statement s;

PreparedStatement ps;

ResultSet rs;

int count;

public void **contextInitialized**(ServletContextEvent sce) {

try{

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

con= DriverManager.

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

s=con.createStatement();

//fetching pageviews value from table counter

rs=s.executeQuery("*select pageview from counter*");

while(rs.next())

{

count=rs.getInt(1);

}

ctx=sce.getServletContext();

ctx.setAttribute("*pcount*", **count**);

}

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

}

public void **contextDestroyed**(ServletContextEvent sce) {

try

{

ctx=sce.getServletContext();

count=(Integer)ctx.getAttribute("*pcount*");

ps=con.prepareStatement("*update counter set pcount='*"+**count**+"*'*");

ps.executeUpdate();

}

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

}

}

**Counter.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class *Counter* **extends** *HttpServlet* {

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

throws ServletException, IOException {

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

PrintWriter out = response.getWriter();

ServletContext ctx = getServletContext();

Integer **count** = (Integer)ctx.getAttribute("*pcount*");

out.println(**count**+"*: pageview*");

ctx.setAttribute("*pcount*", ++**count**);

}

}

### What is ServletRequestAttributeEvent?

**ServletRequestAttributeEvent** class gives notifications about changes to the attributes of **ServletRequest** in an application.

This class listens to the notifications and performs some important tasks whenever there is any change to the request attribute.

#### Some Important Methods of ServletRequestAttributeListener

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void attributeAdded(ServletRequestAttributeEvent e) | notification that a new attribute was added to the servlet request. |
| void attributeRemoved(ServletRequestAttributeEvent e) | notification that an existing attribute was removed from the servlet request. |
| void attributeReplaced(ServletRequestAttributeEvent e) | notification that an attribute was replaced on the servlet request |

#### What is ServletContextAttributeEvent?

**ServletContextAttributeEvent** class let's you know if an attribute in a web application *context* has been added, removed or replaced.

Implementation of ServletContextAttributeListener interface, receive notifications of changes to the attribute list on the servlet context of a web application.

#### Some Important Methods of ServletContextAttributeListener

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void attributeAdded(ServletContextAttributeEvent e) | notification that a new attribute was added to the context. |
| void attributeRemoved(ServletContextAttributeEvent e) | notification that an existing attribute was removed from the context. |
| void attributeReplaced(ServletContextAttributeEvent e) | notification that an attribute was replaced on the context |

### HttpSessionEvent and HttpSessionListener

**HttpSessionEvent** class gives notifications for changes to sessions within a web application. **HttpSessionListener** receives notifications of changes to the list of active sessions in a web application and perform some action. HttpSessionListener is used to perform some important tasks when a session is created or destroyed. For example: counting the number of active session.

#### Some other Session related Listeners

|  |  |
| --- | --- |
| **Listener** | **Description** |
| **HttpSessionActivationListener** | Let's you know when a session moves from one Virtual machine to another. |
| **HttpSessionBindingListener** | Le's your attribute class object get notified when they are added or removed from session. |
| **HttpSessionAttributeListener** | Let's you know when any attribute is added, removed or replaced in a session. |

#### Methods of HttpSessionListener

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void sessionCreated(HttpSessionEvent e) | notification that a session was created. |
| void sessionDestroyed(HttpSessioEvent e) | notification that a session was destroyed. |

#### Example of HttpSessionListener

In this example we will create a session listener that will count the number of active sessions in a web application.

**MySessionCounter.java**

import javax.servlet.http.\*;

public class **MySessionCounter** implements **HttpSessionListener** {

private static int **sessionCount**;

public int **getActiveSession**()

{

return **sessionCount**;

}

public void **sessionCreated**(HttpSessionEvent e)

{

**sessionCount**++;

}

public void **sessionDestroyed**(HttpSessionEvent e)

{

**sessionCount**--;

}

}

**web.xml**

<web-app ...>

<listener>

**<listener-class>***MySessionCounter***</listener-class>**

</listener>

</web-app>

### ServletRequestEvent and ServletRequestListener

**ServletRequestEvent** class gives notification about lifecycle events for a **ServletRequest**. The source of the event is the ServletContext of the web application. **ServletRequestListener** receives the notifications generated by ServletRequestEvent and performs some specified tasks based on them.

#### Some Importants Methods of ServletRequestListener

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void requestDestroyed(ServletRequestEvent e) | is invoked when request is about to go out of scope of the web application.. |
| void requestInitialized(ServletRequestEvent e) | is invoked when request is about to come into scope of the web application. |

#### Example

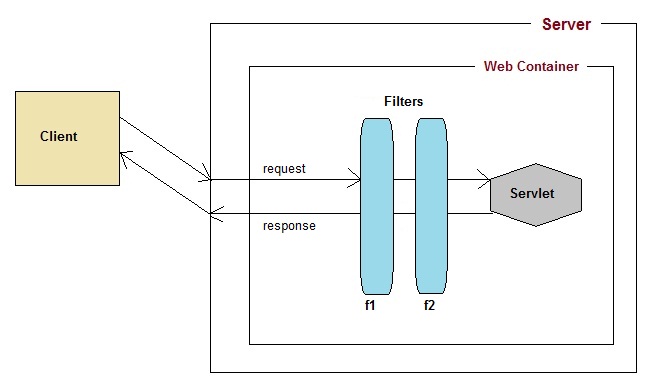


Hence one can easily implement the ServletRequestListener Interface and provide definitions to the two methods provided by it, requestDestroyed() and requestInitialized().

As the name suggests these methods will be called when request is initialized and when it is destroyed, hence we can use them for various purposes, like **variable cleanup** when request is destroyed, or performing some **initial checks** when request is received etc.

**Introduction to Filter API**

**Filters** are components that you can use and configure to perform some filtering tasks. Filter is used for pre-processing of requests and post-processing of responses. You can have any number of filters for pre-processing of a request and post-processing of a response. Filters are configured in the deployment descriptor of a web application.



**How Filters Works?**

* When a request reaches the **Web Container**, it checks if any filter has URL patterns that matches the requested URL.
* The **Web Container** locates the first filter with a matching URL pattern and filter's code is executed.
* If another filter has a matching URL pattern, its code is then executed. This continues until there are no filters with matching URL patterns left.
* If no error occurs, the request passes to the target servlet. Hence we know, that the request will be passed to the target servlet only when all the related Filters are successfully executed.
* The servlet returns the response back to its caller. The last filter that was applied to the request is the first filter applied to the response.
* At last the response will be passed to the **Web Container** which passes it to the client.

**More about Filter API**

**Filter API** is part of **Servlet API**. Filter interface is found in the **javax.servlet** package.

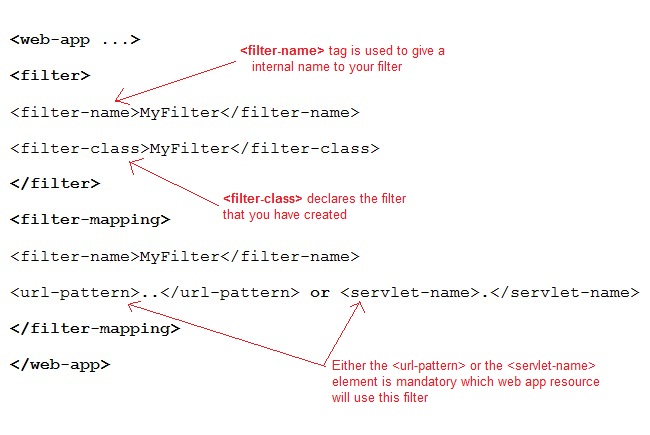
For creating a filter, we must implement Filter interface. Filter interface gives the following life cycle methods for a filter:

1. void init(FilterConfig filterConfig): invoked by the web container to indicate to a filter that it is being placed into service.
2. void doFilter(ServletRequest request, ServletResponse response, FilterChain chain): invoked by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.
3. void destroy(): invoked by the web container to indicate to a filter that it is being taken out of service.

**What is FilterChain Interface?**

**FilterChain** object is used to invoke the next filter in the chain, or if the calling filter is the last filter in the chain then the rosource at the end of the chain invoked. The resources at the end of Filter chain can either be a target Servlet(in case of request flow) or the Client(in case of response flow) as described in the diagram above.

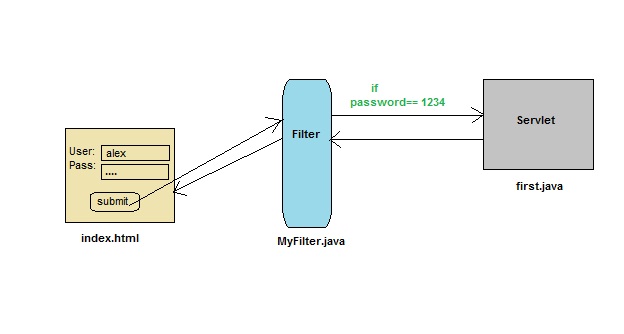
**Declaring a Filter inside Deployment Descriptor**



**Example demonstrating Filter usage**

In this example we are using Filter to authenticate(check correct username and password). Here **index.html** will ask username and password from the user, **MyFilter** will validate the password entered by the user, if the user has entered "1234" as password, then he will be forwarded to **first** servlet else the index.html will be shown again to the user.

This is exactly what we used to do earlier using two servlet classes earlier, one for validation and the other to Welcome the user. Now we will insert a Filter for validating the user.



**index.html**

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

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

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

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

</form>

**web.xml**

<web-app..>

<filter>

<filter-name>MyFilter</filter-name>

**<filter-class>MyFilter</filter-class>**

</filter>

<filter-mapping>

<filter-name>MyFilter</filter-name>

**<servlet-name>first</servlet-name>**

</filter-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>

**MyFilter.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class **MyFilter** implements **Filter** {

public void **init**(FilterConfig fc) throws ServletException {}

public void **doFilter**(ServletRequest request, ServletResponse response,

FilterChain chain) throws IOException, ServletException {

PrintWriter out = response.getWriter();

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

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

{

**chain.doFilter(request, response);**

}

else

{

out.println("You have enter a wrong password");

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

rs.include(request, response);

}

}

public void destroy() { }

}

**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();

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

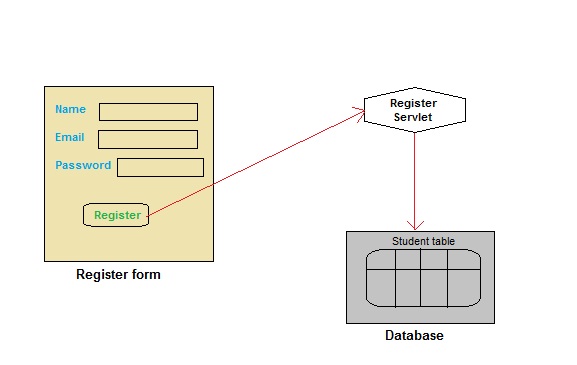
out.println("Wellcome "+**user**);

}

}

### 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>

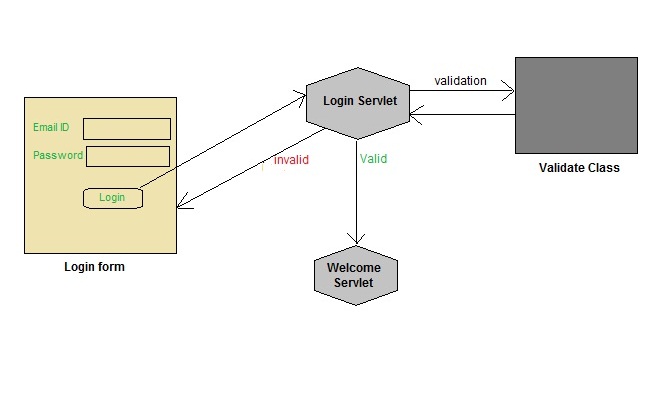
</welcome-file-list>

</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);

}

}

}

**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.