**Servlet Listeners**

During the lifetime of a typical web application, a number of events take place, such as:

* requests are created or destroyed,
* request or session attributes are added, removed, or modified,

The Servlet API provides a number of listener interfaces we can implement in order to react to these events.

|  |  |
| --- | --- |
| **Listener Interface** | **Description** |
| ServletContextListener | Contains methods for handling context initialization and destruction events. |
| ServletContextAttributeListener | Contains methods for reacting to any attributes added, removed, or replaced in the servlet context (application scope). |
| ServletRequestListener | Contains methods for handling request initialization and destruction events. |
| ServletRequestAttributeListener | Contains methods for reacting to any attributes added, removed, or replaced in the request. |
| HttpSessionListener | Contains methods for handling HTTP session initialization and destruction events. |
| HttpSessionAttributeListener | Contains methods for reacting to any attributes added, removed, or replaced in the HTTP session. |

**How to use:**

What we need to do is to implement one or more of these interfaces and then:

* annotate the implemented class(es) with the @WebListener() annotation or
* declare the implemented class(es) in the web.xml (deployment descriptor) via the <listener> tag.

**@WebListener()** annotation has only one option parameter, value="", which is for description of the listener.

1. Using Annotation example:

@WebListener()

public class RequestListener implements ServletRequestListener {

public void requestDestroyed(ServletRequestEvent sre) {...}

public void requestInitialized(ServletRequestEvent sre) {...}

}

2. Using deployment descriptor (web.xml) example:

**<web-app .....>**

**<listener>**

**<listener-class>**RequestListener**</listener-class>** <!-- required -->

**<display-name></display-name>** <!-- optional -->

**<description></description>** <!-- optional -->

**<icon></icon>** <!-- optional -->

**</listener>**

**</web-app>**

**Description on Listener interfaces :**

1. ***ServletContextListener***

The ServletContextEvent is notified when web application is deployed on the server.

If we want to perform some action at the time of deploying the web application such as creating database connection, creating all the tables of the project etc, you need to implement ServletContextListener interface and provide the implementation of its methods.

* + **Constructor of ServletContextEvent class**

There is only one constructor defined in the ServletContextEvent class. The web container creates the instance of ServletContextEvent after the ServletContext instance.

* + 1. ServletContextEvent(ServletContext e)
  + **Method of ServletContextEvent class**

There is only one method defined in the ServletContextEvent class:

* + 1. **public ServletContext getServletContext():** returns the instance of ServletContext.
  + **Methods of ServletContextListener interface**

There are two methods declared in the ServletContextListener interface which must be implemented by the servlet programmer to perform some action such as creating database connection etc.

1. **public void contextInitialized(ServletContextEvent e):** is invoked when application is deployed on the server.
2. **public void contextDestroyed(ServletContextEvent e):** is invoked when application is undeployed from the server.

Example:

In this example, we are retrieving the data from the **Emp** table. we have created the connection object in the **listener class** and used the connection object in the **servlet**.

MyListener.java(Listener class)

1. **package** com.lti.demonstration;
2. **import** java.sql.\*;
3. **public** **class** MyListener **implements** ServletContextListener{

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

**try**{

Class.*forName*("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.*getConnection*( "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//storing connection object as an attribute in ServletContext

ServletContext ctx=event.getServletContext();

ctx.setAttribute("mycon", con);

}

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

}

**public** **void** contextDestroyed(ServletContextEvent arg0) {}

1. }

Comparing to conventional(Servlet connection)

1. **package** com.lti.demonstration;
2. **import** java.io.\*;
3. **import** javax.servlet.\*;
4. **import** javax.servlet.http.\*;
5. **import** java.sql.\*;
6. **public** **class** FetchData **extends** HttpServlet {
7. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)**throws** ServletException, IOException {
8. response.setContentType("text/html");
9. PrintWriter out = response.getWriter();
10. **try**{
11. //Retrieving connection object from ServletContext object
12. ServletContext ctx=getServletContext();
13. Connection con=(Connection)ctx.getAttribute("mycon");
14. //retieving data from emp32 table
15. PreparedStatement ps=con.prepareStatement("select \* from emp32",
16. ResultSet.***TYPE\_SCROLL\_SENSITIVE***,ResultSet.***CONCUR\_UPDATABLE***);
17. ResultSet rs=ps.executeQuery();
18. **while**(rs.next()){
19. out.print("<br>"+rs.getString(1)+" "+rs.getString(2));
20. }
21. con.close();
22. }**catch**(Exception e){e.printStackTrace();}
23. out.close();
24. }
25. }

xample 2:To create a table The same can be used to create a table in the following wayExample 2: The same can be used to create a table in the following way

1. **package** com.lti.demonstration;
2. **import** javax.servlet.\*;
3. **import** java.sql.\*;
4. **public** **class** MyListener **implements** ServletContextListener{
5. **public** **void** contextInitialized(ServletContextEvent arg0) {
6. **try**{
7. Class.forName("oracle.jdbc.driver.OracleDriver");
8. Connection con=DriverManager.getConnection("
9. jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
10. String query="create table emp32(id number(10),name varchar2(40))";
11. PreparedStatement ps=con.prepareStatement(query);
12. ps.executeUpdate();
13. System.out.println(query);
14. }**catch**(Exception e){e.printStackTrace();}
15. }
16. **public** **void** contextDestroyed(ServletContextEvent arg0) {
17. System.***out***.println("project undeployed");
18. }
19. }
20. ***ServletContextAttributeListener***

By implementing **ServletContextAttributeListener** interface, we can listen to these events –

* When an attribute is added to the ServletContext object in a web application.
* When an attribute of the ServletContext object is modified in a web application.
* When an attribute of the ServletContext object is deleted in a web application.

This can be by using the @WebListener annotation, adding the listener to the servlet descriptor or adding a listener with .addListener() to the servlet context. In this example we use the @WebListener annotation.

In order to listen to a attribute change in the servlet context we need to implement the javax.servlet.ServletContextAttributeListener interface. This interface lets us listen to the following events, the names speak for themselves.

* attributeAdded()
* attributeRemoved()
* attributeReplaced()

To register a listener we can add the @WebListener, define the listener in the servlet descriptor (web.xml) or add it to the servlet context. In this example we choose to add the listener through the @WebListener annotation.

**package** com.lti.demonstration;

**import** javax.servlet.ServletContextAttributeEvent;

**import** javax.servlet.ServletContextAttributeListener;

**import** javax.servlet.annotation.WebListener;

@WebListener

**public** **class** ApplicationContextAttributeListener **implements** ServletContextAttributeListener {

@Override

**public** **void** attributeAdded(ServletContextAttributeEvent event) {

System.***out***.println("attribute: " + event.getName() + " was added with value: event.getValue());

}

@Override

**public** **void** attributeRemoved(ServletContextAttributeEvent event) {

System.***out***.println("attribute: " + event.getName() + " was removed with value: " + event.getValue());

}

@Override

**public** **void** attributeReplaced(ServletContextAttributeEvent event) {

System.***out***.println("attribute: " + event.getName() + " was replaced with value: " + event.getValue());

}

}

The same can be done using web.xml servlet descriptor instead of @WebListener annotation where you can add the context listener as follows:

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

<web-app xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee*

*http://xmlns.jcp.org/xml/ns/javaee*

*/web-app\_3\_1.xsd"* version=*"3.1"*>

<listener>

<listener-class>

com.memorynotfound.ApplicationContextAttributeListener

</listener-class>

</listener>

</web-app>

**Working:**

Every time an attribute is added, replaced or removed the corresponding method is invoked. This allows us to track changes to certain attributes in our servlet context.

1. ***HttpSessionListener***

We can perform some operations at this event such as counting total and current logged-in users, maintaining a log of user details such as login time, logout time etc

**Methods of HttpSessionListener interface**

There are two methods declared in the HttpSessionListener interface which must be implemented by the servlet programmer to perform some action

1. **public void sessionCreated(HttpSessionEvent e)**: is invoked when session object is created.
2. **public void sessionDestroyed(ServletContextEvent e)**: is invoked when session is invalidated.

In this example we are counting the total and current logged-in users. For this purpose, we have created three files:

* + 1. **index.html:** to get input from the user.
    2. **CountUserListener.java:** A listener class that counts total and current logged-in users and stores this information in ServletContext object as an attribute.
    3. **First.java:** A Servlet class that creates session and prints the total and current logged-in users.
    4. **Logout.java:** A Servlet class that invalidates session.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<form action=*"servlet1"*>

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

Password:<input type=*"password"* name=*"userpass"*><br>

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

</form>

</body>

</html>

Index.html

**package** com.lti.demonstration;

**import** javax.servlet.ServletContext;

**import** javax.servlet.http.HttpSessionEvent;

**import** javax.servlet.http.HttpSessionListener;

**public** **class** CountUserListener **implements** HttpSessionListener{

ServletContext ctx=**null**;

**static** **int** *total*=0,*current*=0;

**public** **void** sessionCreated(HttpSessionEvent e) {

*total*++;

*current*++;

ctx=e.getSession().getServletContext();

ctx.setAttribute("totalusers", *total*);

ctx.setAttribute("currentusers", *current*);

}

**public** **void** sessionDestroyed(HttpSessionEvent e) {

*current*--;

ctx.setAttribute("currentusers",*current*);

}

}

**CountUserListener.java**

First.java

**package** com.lti.demonstration;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletContext;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.servlet.http.HttpSession;

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

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n=request.getParameter("username");

out.print("Welcome "+n);

HttpSession session=request.getSession();

session.setAttribute("uname",n);

//retrieving data from ServletContext object

ServletContext ctx=getServletContext();

**int** t=(Integer)ctx.getAttribute("totalusers");

**int** c=(Integer)ctx.getAttribute("currentusers");

out.print("<br>total users= "+t);

out.print("<br>current users= "+c);

out.print("<br><a href='logout'>logout</a>");

out.close();

}

}

Logout.java

**package** com.lti.demonstration;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.servlet.http.HttpSession;

**public** **class** LogoutServlet **extends** HttpServlet {

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

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

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

session.invalidate();//invalidating session

out.print("You are successfully logged out");

out.close();

}

}

1. ***HttpSessionAttributeListener***

HttpSessionAttributeListener is an interface which is extending from the base interface **java.util.EventListener**.

HttpSessionAttributeListener will be informed by the web container when there is a change to the attributes of a web application's session, like an attribute is added in a session or an attribute is removed or an attribute is replaced by another attribute etc.

**HttpSessionAttributeListener interface has following methods:**

1. attributeAdded(HttpSessionBindingEvent event)
2. attributeRemoved(HttpSessionBindingEvent event)
3. attributeReplaced(HttpSessionBindingEvent event)

web.xml

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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

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

xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*version=*"2.5"*>

<listener>

<listener-class>

com.listener.HttpSessionAttributeListenerExample

</listener-class>

</listener>

</web-app>

Example

**package** com.lti.demonstration

**import** javax.servlet.http.HttpSessionAttributeListener;

**import** javax.servlet.http.HttpSessionBindingEvent;

**public** **class** HttpSessionAttributeListenerExample **implements** HttpSessionAttributeListener {

@Override

**public** **void** attributeAdded(HttpSessionBindingEvent event) {

String attributeName = event.getName();

Object attributeValue = event.getValue();

System.***out***.println("Attribute added : " + attributeName + " : " + attributeValue);

}

@Override

**public** **void** attributeRemoved(HttpSessionBindingEvent event) {

String attributeName = event.getName();

Object attributeValue = event.getValue();

System.***out***.println("Attribute removed : " + attributeName + " : "+ attributeValue);

}

@Override

**public** **void** attributeReplaced(HttpSessionBindingEvent event) {

String attributeName = event.getName();

Object attributeValue = event.getValue();

System.***out***.println("Attribute replaced : " + attributeName + " : "+ attributeValue);

}

}

**Working:**

* If a new session’s attribute is added, the listener’s attributeAdded() will be executed.
* If a new session’s attribute is updated, the listener’s attributeReplaced() will be executed.
* If a new session’s attribute is removed, the listener’s attributeRemoved() will be executed.

1. ***HttpSessionBindingListener***

By implementing **HttpSessionBindingListener** interface, we can listen to these events –

* When an attribute of **HttpSessionBindingListener** type is added in an HttpSession within a web application.
* When an attribute of **HttpSessionBindingListener** type is replaced in an HttpSession within a web application.
* When an attribute of **HttpSessionBindingListener** type is deleted in an HttpSession within a web application.

The listener HttpSessionBindingListener is an interface extending from base interface **java.util.EventListener** interface. This interface will notify an object when it is bound to or unbound from a session.

**Method of javax.servlet.http.HttpSessionBindingListener interface**

* valueBound(HttpSessionBindingEvent event).
* valueUnBound(HttpSessionBindingEvent event).

The below example is demonstrating HttpSessionBindingListener

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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

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

xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

version=*"2.5"*>

<listener>

<listener-class>

com.listener.HttpSessionAttributeListenerExample

</listener-class>

</listener>

</web-app>

Example

**package** com.listener;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public** **class** HttpSessionBindingListenerExample **implements** HttpSessionBindingListener {

ServletContext context;

**public** HttpSessionBindingListenerExample(ServletContext context) {

**this**.context = context;

}

**public** **void** valueBound(HttpSessionBindingEvent event) {

context.log("The value bound is " + event.getName());

}

**public** **void** valueUnbound(HttpSessionBindingEvent event) {

context.log("The value unbound is " + event.getName());

}

}

**Invoke HttpSessionListener From Servlet**

// Get the current session object, create one if necessary

HttpSession session = req.getSession();

// Add a HttpSessionBindingListenerExample

session.setAttribute("name",**new** HttpSessionBindingListenerExample(getServletContext()));

session.removeAttribute("name");

1. ***ServletRequestListener***

By implementing **ServletRequestListener** interface, we can listen to these events

* + When a user request reaches the web application.
  + When a user request is served by the web application.

ServletRequestListener is used to listen to lifecycle events for a ServletRequest .Since, we may like to receive a notification whenever a request for a resource is made from the client so that we can log it.

**How?**

Java EE specification provides an interface called**ServletRequestListener** which receives notifications whenever a new request is about to come to the web application. Methods required :

**1.requestInitialized(ServletRequestEvent event) :**

Receives notification that a ServletRequest is about to come into scope of the web application.

**2. requestDestroyed(ServletRequestEvent event) :**

Receives notification that a ServletRequest is about to go out of scope of the web application.

**Example:**

1. We will create a servlet called HelloServlet which will simply receive some request from a web browser and do nothing. It is as follows.

**package** com.lti.demonstration.servlets;

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** HelloServlet **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**protected** **void** doGet(HttpServletRequest request,

HttpServletResponse response) **throws** ServletException, IOException {

}

}

1. Configure HelloServlet in web.xml

<servlet>

<description></description>

<display-name>HelloServlet</display-name>

<servlet-name>HelloServlet</servlet-name>

<servlet-class>com.thejavageek.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloServlet</servlet-name>

<url-pattern>/HelloServlet</url-pattern>

</servlet-mapping>

1. Now we will create a **ServletRequestListener** which receives the notifications whenever some request is being sent for any resource in this web application.

**package** com.lti.demonstration.example;

**import** javax.servlet.ServletRequestEvent;

**import** javax.servlet.ServletRequestListener;

**public** **class** RequestListener **implements** ServletRequestListener {

**public** **void** requestDestroyed(ServletRequestEvent event) {

System.***out***.println("request being sent to "

+ event.getServletRequest().getRemoteAddr());

}

**public** **void** requestInitialized(ServletRequestEvent event) {

System.***out***.println("now initializing request"

+ event.getServletRequest().getRemoteAddr());

}

}

1. Configure this listener in web.xml as follows.

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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

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

xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* version=*"2.5"*>

<listener>

<listener-class>com.thejavageek.RequestListener</listener-class>

</listener>

</web-app>

1. ***ServletRequestAttributeListener***

By implementing **ServletRequestAttributeListener** interface, we can listen to these events –

* + When an attribute is added to an ServletRequest object within a web application.
  + When an attribute of an ServletRequest object is replaced in within a web application.
  + When an attribute of an ServletRequest object is deleted in within a web application.

ServletRequestAttributeListener can be used to receive notifications whenever some attributes are added, removed or replaced in ServletRequest.

**How can we achieve it?**

This interface has three *methods to achieve this*.

1. **attributeAdded(ServletRequestAttributeEvent event)** : receives notification when an attribute is added in ServletRequest.
2. **attributeRemoved(ServletRequestAttributeEvent event)**  : receives notification when an attribute is removed from ServletRequest.
3. **attributeReplaced(ServletRequestAttributeEvent event)** : receives notification when an attribute is removed from ServletRequest.

**EXAMPLE:**

1. Create ServletRequestAttributeListener and configure in web.xmls

**package** com.lti.demonstration.example;

**import** javax.servlet.ServletRequestAttributeEvent;

**import** javax.servlet.ServletRequestAttributeListener;

**public** **class** AttributeListener **implements** ServletRequestAttributeListener {

**public** **void** attributeAdded(ServletRequestAttributeEvent event) {

System.***out***.println("Attribute added : " + event.getName() + " = "

+ event.getValue());

}

**public** **void** attributeRemoved(ServletRequestAttributeEvent event) {

System.***out***.println("Attribute removed : " + event.getName() + " = "

+ event.getValue());

}

**public** **void** attributeReplaced(ServletRequestAttributeEvent event) {

System.***out***.println("Attribute replaced : " + event.getName() + " = "

+ event.getValue());

}

}

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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

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

xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* version=*"2.5"*>

<listener>

<listener-class>com.thejavageek.AttributeListener</listener-class>

</listener>

<listener>

<listener-class>com.thejavageek.AttributeListener</listener-class>

</listener>

</web-app>

1. To listen to attribute change events, we must add, remove and replace them somewhere. Create a HelloServlet which will perform operations on attributes.

**package** com.lti.demonstration.example;

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** HelloServlet **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**protected** **void** doGet(HttpServletRequest request,

HttpServletResponse response) **throws** ServletException, IOException {

request.setAttribute("firstName", "prasad");

request.setAttribute("firstName", "pranil");

request.removeAttribute("firstName");

}

}

1. Configure HelloServlet in web.xml
2. Now simply deploy the application under tomcat server and start tomcat

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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

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

xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* version=*"2.5"*>

<servlet>

<description></description>

<display-name>HelloServlet</display-name>

<servlet-name>HelloServlet</servlet-name>

<servlet-class>com.thejavageek.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloServlet</servlet-name>

<url-pattern>/HelloServlet</url-pattern>

</servlet-mapping>

</web-app>

**Why Servlet Listeners?**

Using ServletContext, can create an attribute with application scope that all other servlets can access but we can initialize **Se**

**rvletContext init parameters as String only** in deployment descriptor (web.xml). What if the application is database oriented and we want to set an attribute in ServletContext for Database Connection? If an application has a single-entry point (user login), then authentication can be done in the first servlet request but if it has multiple entry points then adding it everywhere will result in a lot of code redundancy. Also, if database is down or not configured properly, we won’t know until first client request comes to server. To handle these scenarios, servlet API provides Listener interfaces that we can implement and configure to listen to an event and do certain operations.

**Event** is occurrence of something, in web application world an event can be initialization of application, destroying an application, request from client, creating/destroying a session, attribute modification in session etc.

**Servlet API** provides different types of Listener interfaces that we can implement and configure in web.xml to process something when a particular event occurs. For example, in above scenario we can create a Listener for the application startup event to read context init parameters and create a database connection and set it to context attribute for use by other resources.

Listeners are the classes which listens to a particular type of events and when that event occurs, triggers the functionality. Each type of listener is bind to a type of event. In this chapter we will discuss the types of listeners supported by servlet framework.

**Types of Servlets**

There are total eight type of listeners available in servlet framework which listens to a particular event and they are –

* ServletContextListener
* ServletContextAttributeListener
* HttpSessionListener
* HttpSessionAttributeListener
* ServletRequestListener
* ServletRequestAttributeListener
* HttpSessionActivationListener
* HttpSessionBindingListener

As the configurations of servlets, filters goes inside web.xml , similarly listeners are also configured inside web.xml using <listener> </listener> tag.

Note – Listeners are neither a Servlets nor a JSP.

1. **ServletContextListener**

ServletContextListener listens to SessionContextEvent event which gives a notification when Servlet Context is initialized or destroyed on the events ServletContextListener executes the functionality.

ServletContextListener is the interface and it defines two methods –

* void contextDestroyed(ServletContextEvent e) – This method is executed when application is destroyed
* void contextInitialized(ServletContextEvent e)- This method is executed when application is initialized

ServletContext object can be obtained from ServletContextEvent and listener can set the attributes in Servlet context object which can be used in servlets later.

We can use the “ServletContextListener” listener for any activity that is required either at the application deployment time or any clean up activity required when application is destroyed. One of the practical examples that I can think of is initializing database connections and clean-up of database connections.

1. **ServletContextAttributeListener**

ServletContextAttributeListener listens to SessionContexAttributetEvent event which gives a notification when any object is added, removed or replaced from servlet context.

ServletContextAttributeListener is the interface and it defines three methods –

* attributeAdded(ServletContextAttributeEvent e): It notifies whenever a new attribute is added to the servlet context.
* attributeRemoved(ServletContextAttributeEvent e): It notifies whenever the attribute is removed from the servlet context.
* attributeReplaced(ServletContextAttributeEvent e): It notifies whenever the attribute gets replaced on the servlet context.

Attribute name and value that has been added, removed or replaced can be obtained from the getName() and getValue() method of ServletContextAttributeEvent.

1. **HttpSessionListener**

HttpSessionListener listens to HttpSessionEvent event which gives a notification when session is created or destroyed. HttpSessionListener is the interface and it defines two methods –

* void sessionDestroyed(HttpSessionEvent e) – This method is executed when session is destroyed
* void sessionCreated(HttpSessionEvent e)- This method is executed when session is created.

Session object can be obtained from HttpSessionEvent.

1. **HttpAttributeSessionListener**

HttpSessionAttributeListener listens to HttpSessionBindingEvent event which gives a notification when any object is added, removed or replaced from session.

HttpSessionAttributeListener is the interface and it defines three methods –

* attributeAdded(HttpSessionBindingEvent e): It notifies whenever a new attribute is added to the session.
* attributeRemoved(HttpSessionBindingEvent e): It notifies whenever the attribute is removed from the session.
* attributeReplaced(HttpSessionBindingEvent e): It notifies whenever the attribute gets replaced on the session.

Attribute name and value that has been added, removed or replaced can be obtained from the getName() and getValue() method of HttpSessionBindingEvent.

1. **ServletRequestListener**

ServletRequestListener listens to ServletRequestEvent event which gives a notification when request is created or destroyed.

ServletRequestListener is the interface and it defines two methods –

* void requestDestroyed(ServletRequestEvent e) – This method is executed when request is destroyed
* void requestInitialized(ServletRequestEvent e) – This method is executed when request is initialized.

Request object can be Obtained from HttpRequestEvent.

1. **ServletRequestAttributeListener**

ServletRequestAttributeListener listens to ServletRequestAttributeEvent event which gives a notification when any object is added, removed or replaced from request.

ServletRequestAttributeListener is the interface and it defines three methods –

* attributeAdded(ServletRequestAttributeEvent e) – It notifies whenever a new attribute is added to the request.
* attributeRemoved(ServletRequestAttributeEvent e) – It notifies whenever the attribute is removed from the request.
* attributeReplaced(ServletRequestAttributeEvent e) – It notifies whenever the attribute gets replaced on the request.

Attribute name and value that has been added, removed or replaced can be obtained from the getName() and getValue() method of ServletRequestAttributeEvent

**Servlet Listener Interfaces and Event Objects**

Servlet API provides different kind of listeners for different types of Events. Listener interfaces declare methods to work with a group of similar events, for example we have ServletContext Listener to listen to start-ups and shutdown event of context. Every method in listener interface takes Event object as input. Event object works as a wrapper to provide specific object to the listeners.

Servlet API provides following event objects.

* javax.servlet.AsyncEvent – Event that gets fired when the asynchronous operation initiated on a ServletRequest (via a call to ServletRequest#startAsync orServletRequest#startAsync(ServletRequest, ServletResponse)) has completed, timed out, or produced an error.
* javax.servlet.http.HttpSessionBindingEvent – Events of this type are either sent to an object that implements HttpSessionBindingListener when it is bound or unbound from a session, or to a HttpSessionAttributeListener that has been configured in the web.xml when any attribute is bound, unbound or replaced in a session.
* The session binds the object by a call to HttpSession.setAttribute and unbinds the object by a call to HttpSession.removeAttribute.
* We can use this event for clean-up activities when object is removed from session.
* javax.servlet.http.HttpSessionEvent – This is the class representing event notifications for changes to sessions within a web application.
* javax.servlet.ServletContextAttributeEvent – Event class for notifications about changes to the attributes of the ServletContext of a web application.
* javax.servlet.ServletContextEvent – This is the event class for notifications about changes to the servlet context of a web application.
* javax.servlet.ServletRequestEvent – Events of this kind indicate lifecycle events for a ServletRequest. The source of the event is the ServletContext of this web application.
* javax.servlet.ServletRequestAttributeEvent – This is the event class for notifications of changes to the attributes of the servlet request in an application.

Servlet API provides following Listener interfaces.

* javax.servlet.AsyncListener – Listener that will be notified in the event that an asynchronous operation initiated on a ServletRequest to which the listener had been added has completed, timed out, or resulted in an error.
* javax.servlet.ServletContextListener – Interface for receiving notification events about ServletContext lifecycle changes.
* javax.servlet.ServletContextAttributeListener – Interface for receiving notification events about ServletContext attribute changes.
* javax.servlet.ServletRequestListener – Interface for receiving notification events about requests coming into and going out of scope of a web application.
* javax.servlet.ServletRequestAttributeListener – Interface for receiving notification events about ServletRequest attribute changes.
* javax.servlet.http.HttpSessionListener – Interface for receiving notification events about HttpSession lifecycle changes.
* javax.servlet.http.HttpSessionBindingListener – Causes an object to be notified when it is bound to or unbound from a session.
* javax.servlet.http.HttpSessionAttributeListener – Interface for receiving notification events about HttpSession attribute changes.
* javax.servlet.http.HttpSessionActivationListener – Objects that are bound to a session may listen to container events notifying them that sessions will be passivated and that session will be activated. A container that migrates session between VMs or persists sessions is required to notify all attributes bound to sessions implementing HttpSessionActivationListener.

**EXAMPLE**

**MyServlet.java**

package com.journaldev.servlet;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletContext;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

@WebServlet("/MyServlet")

public class MyServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

ServletContext ctx = request.getServletContext();

ctx.setAttribute("User", "Saurav");

String user = (String) ctx.getAttribute("User");

ctx.removeAttribute("User");

HttpSession session = request.getSession();

session.invalidate();

PrintWriter out = response.getWriter();

out.write("Hi "+user);

}

}

**AppContextAttributeListener.java**

package com.journaldev.listener;

import javax.servlet.ServletContextAttributeEvent;

import javax.servlet.ServletContextAttributeListener;

import javax.servlet.annotation.WebListener;

@WebListener

public class AppContextAttributeListener implements ServletContextAttributeListener {

public void attributeAdded(ServletContextAttributeEvent servletContextAttributeEvent) {

System.out.println("ServletContext attribute added::{"+servletContextAttributeEvent.getName()+","+servletContextAttributeEvent.getValue()+"}");

}

public void attributeReplaced(ServletContextAttributeEvent servletContextAttributeEvent) {

System.out.println("ServletContext attribute replaced::{"+servletContextAttributeEvent.getName()+","+servletContextAttributeEvent.getValue()+"}");

}

public void attributeRemoved(ServletContextAttributeEvent servletContextAttributeEvent) {

System.out.println("ServletContext attribute removed::{"+servletContextAttributeEvent.getName()+","+servletContextAttributeEvent.getValue()+"}");

}

}

**AppContextListener.java**

package com.journaldev.listener;

import javax.servlet.ServletContext;

import javax.servlet.ServletContextEvent;

import javax.servlet.ServletContextListener;

import javax.servlet.annotation.WebListener;

import com.journaldev.db.DBConnectionManager;

@WebListener

public class AppContextListener implements ServletContextListener {

public void contextInitialized(ServletContextEvent servletContextEvent) {

ServletContext ctx = servletContextEvent.getServletContext();

String url = ctx.getInitParameter("DBURL");

String u = ctx.getInitParameter("DBUSER");

String p = ctx.getInitParameter("DBPWD");

//create database connection from init parameters and set it to context

DBConnectionManager dbManager = new DBConnectionManager(url, u, p);

ctx.setAttribute("DBManager", dbManager);

System.out.println("Database connection initialized for Application.");

}

public void contextDestroyed(ServletContextEvent servletContextEvent) {

ServletContext ctx = servletContextEvent.getServletContext();

DBConnectionManager dbManager = (DBConnectionManager) ctx.getAttribute("DBManager");

dbManager.closeConnection();

System.out.println("Database connection closed for Application.");

}

}

**MyServletRequestListener.java**

package com.journaldev.listener;

import javax.servlet.ServletRequest;

import javax.servlet.ServletRequestEvent;

import javax.servlet.ServletRequestListener;

import javax.servlet.annotation.WebListener;

@WebListener

public class MyServletRequestListener implements ServletRequestListener {

public void requestDestroyed(ServletRequestEvent servletRequestEvent) {

ServletRequest servletRequest = servletRequestEvent.getServletRequest();

System.out.println("ServletRequest destroyed. Remote IP="+servletRequest.getRemoteAddr());

}

public void requestInitialized(ServletRequestEvent servletRequestEvent) {

ServletRequest servletRequest = servletRequestEvent.getServletRequest();

System.out.println("ServletRequest initialized. Remote IP="+servletRequest.getRemoteAddr());

}

}

**MySessionListener.java**

package com.journaldev.listener;

import javax.servlet.annotation.WebListener;

import javax.servlet.http.HttpSessionEvent;

import javax.servlet.http.HttpSessionListener;

@WebListener

public class MySessionListener implements HttpSessionListener {

public void sessionCreated(HttpSessionEvent sessionEvent) {

System.out.println("Session Created:: ID="+sessionEvent.getSession().getId());

}

public void sessionDestroyed(HttpSessionEvent sessionEvent) {

System.out.println("Session Destroyed:: ID="+sessionEvent.getSession().getId());

}

}

**Web.xml**

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

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

<**display-name**>ServletListenerExample</**display-name**>

<**context-param**>

<**param-name**>DBUSER</**param-name**>

<**param-value**>Saurav</**param-value**>

</**context-param**>

<**context-param**>

<**param-name**>DBPWD</**param-name**>

<**param-value**>password</**param-value**>

</**context-param**>

<**context-param**>

<**param-name**>DBURL</**param-name**>

<**param-value**>jdbc:mysql://localhost/mysql\_db</**param-value**>

</**context-param**>

<**listener**>

<**listener-class**>com.journaldev.listener.AppContextListener</**listener-class**>

</**listener**>

<**listener**>

<**listener-class**>com.journaldev.listener.AppContextAttributeListener</**listener-class**>

</**listener**>

<**listener**>

<**listener-class**>com.journaldev.listener.MySessionListener</**listener-class**>

</**listener**>

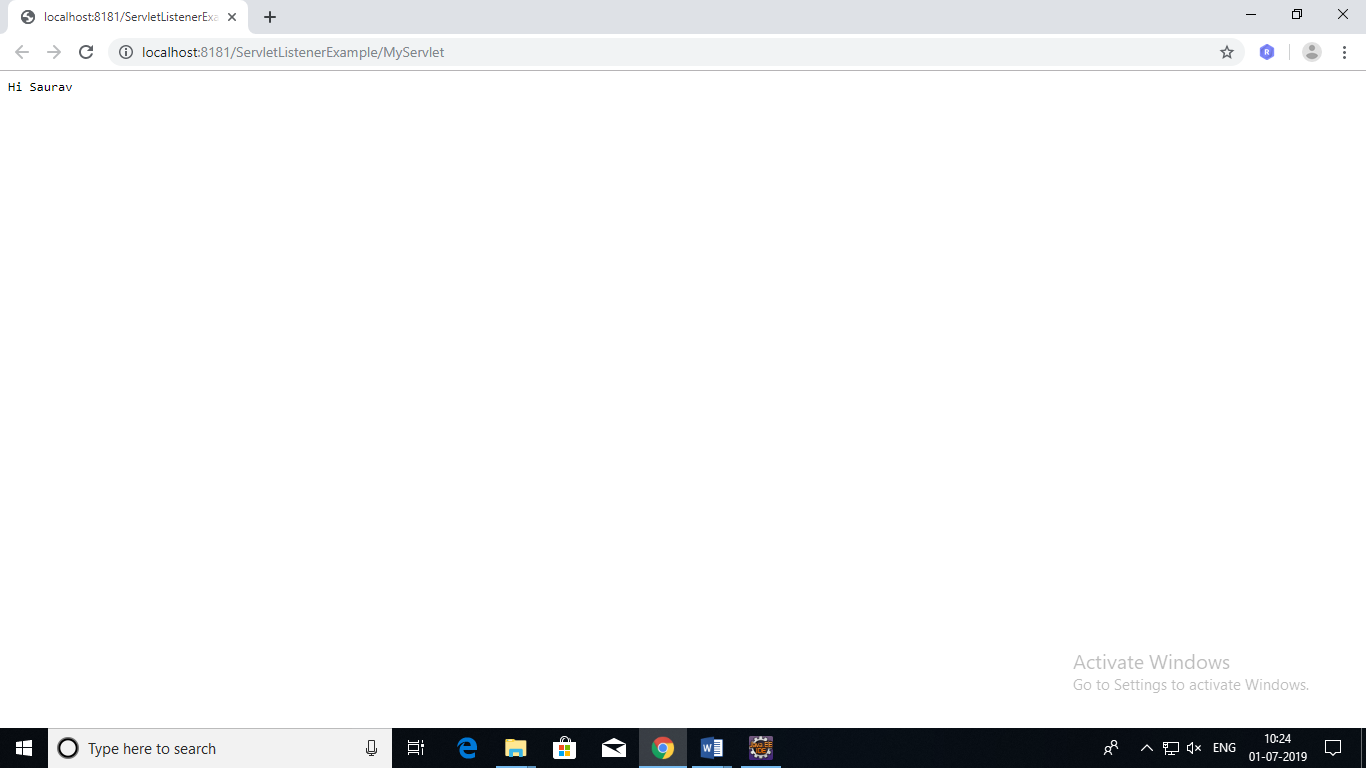
<**listener**>

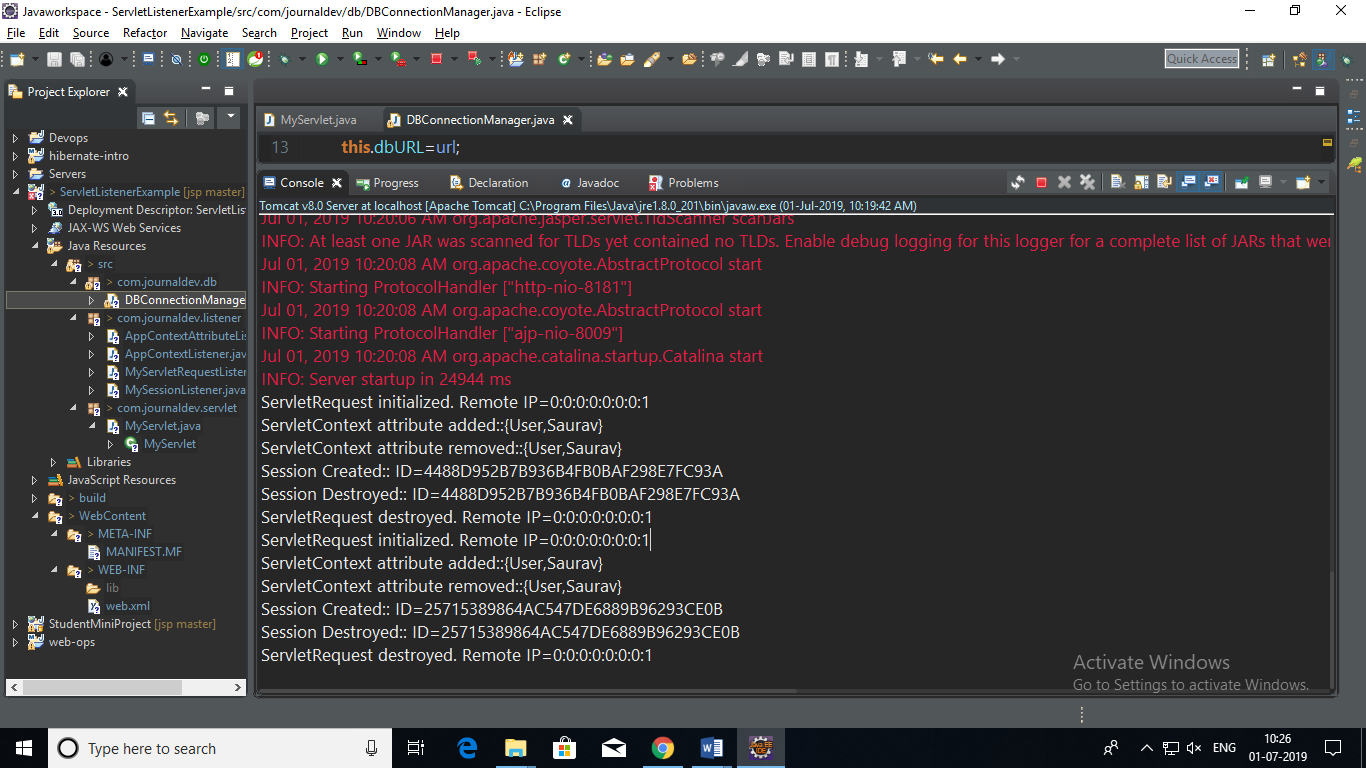
<**listener-class**>com.journaldev.listener.MyServletRequestListener</**listener-class**>

</**listener**>

</**web-app**>

**OUTPUT**





**FILTERS**

A **filter** is an object that is invoked at the pre-processing and post processing of a request. It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc. JSP Specification provides a concept of filters to intercept the requests before reaching the actual JSP or servlets and similarly can update the response before sending to client. Servlet and JSP Filters are Java classes that can be used in Servlet and JSP Programming for the following purposes −

* To intercept requests from a client before they access a resource at back end.
* To manipulate responses from server before they are sent back to the client.

There are various types of filters suggested by the specifications −

* Authentication Filters
* Data compression Filters
* Encryption Filters
* Filters that trigger resource access events
* Image Conversion Filters
* Logging and Auditing Filters
* MIME-TYPE Chain Filters
* Tokenizing Filters
* XSL/T Filters That Transform XML Content

Filters are deployed in the deployment descriptor file **web.xml** and then map to either servlet or JSP names or URL patterns in your application's deployment descriptor. The **servlet filter is pluggable**, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet. So maintenance cost will be less.

**Overview of the servlet filters**

When the servlet container calls a method in a servlet on behalf of the client, the HTTP request that the client sent is, by default, passed directly to the servlet. The response that the servlet generates is, by default, passed directly back to the client, with its content unmodified by the container. In this scenario, the servlet must process the request and generate as much of the response as the application requires.

But there are many cases in which some pre-processing of the request for servlets would be useful. In addition, it is sometimes useful to modify the response from a class of servlets. One example is encryption. A servlet, or a group of servlets in an application, may generate response data that is sensitive and should not go out over the network in clear-text form, especially when the connection has been made using a nonsecure protocol such as HTTP. A filter can encrypt the responses. Of course, in this case the client must be able to decrypt the responses.

A common scenario for a filter is one in which you want to apply pre-processing or post processing to requests or responses for a group of servlets, not just a single servlet. If you need to modify the request or response for just one servlet, there is no need to create a filter—just do what is required directly in the servlet itself.

Note that filters are not servlets. They do not implement and override HttpServlet methods such as doGet() or doPost(). Rather, a filter implements the methods of the javax.servlet.Filter interface. The methods are:

* **init()**
* **destroy()**
* **doFilter()**

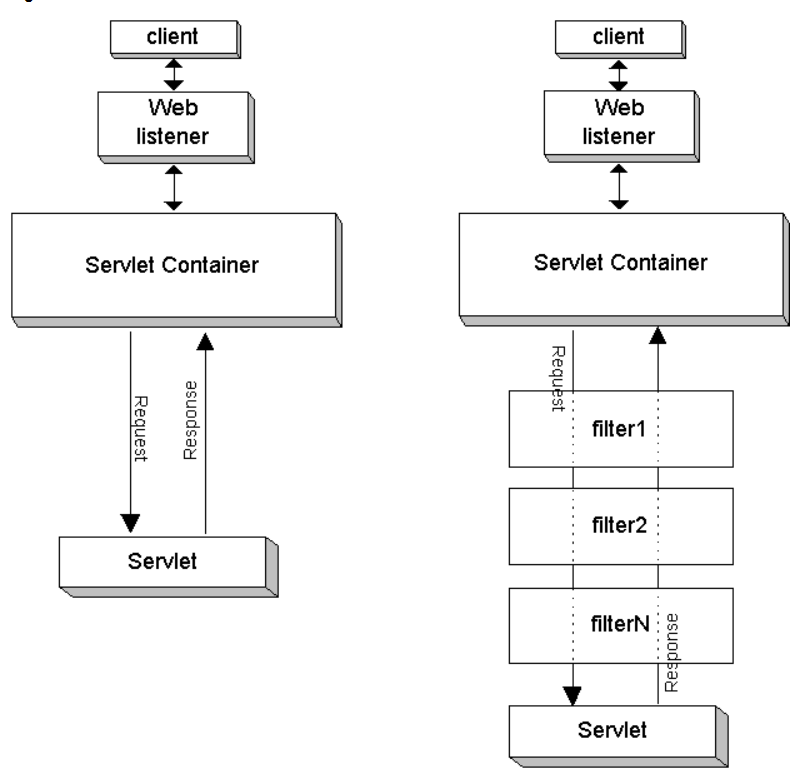
**How the Servlet Container Invokes Filters?**

[Figure BELOW](https://docs.oracle.com/cd/B14099_19/web.1012/b14017/filters.htm#i1000029) shows how the servlet container invokes filters. On the left is a scenario in which no filters are configured for the servlet being called. On the right, several filters (1, 2, ..., N) have been configured in a chain to be invoked by the container before the servlet is called and after it has responded. The web.xml file specifies which servlets cause the container to invoke the filters.

The order in which filters are invoked depends on the order in which they are configured in the web.xml file. The first filter in web.xml is the first one invoked during the request, and the last filter in web.xml is the first one invoked during the response. Note the reverse order during the response.

**Note:**

Be careful in coordinating any use of multiple filters, in case of possible overlap in functionality or in what the filters are overwriting.



**JSP Filter Interface**

JSP Filter class has to implement javax.servlet.Filter interface. Filter interface defines three methods which means classes implements filter interface has to implement these methods.

· **void init(FilterConfig)**

· **doFilter(ServletRequest, ServletResponse, FilterChain)**

· **public void destroy()**

**JSP Filter Life Cycle**

* The life cycle of a JSP filter is managed by a container and consists of implementing the following methods:  
  **init()** This method is called only once after instantiation to perform any initialization task. We can define a initialization parameters in web.xml for filters similar to init-params of servlets.
* **doFilter()** This method is called after the init() method and is called each time a  
  filter needs to perform any function. This method performs the actual work of a filter, either modifying the request or the response.
* **destroy()** This method is used to perform any clean-up operation before the container removes a filter instance.

**EXAMPLE**

1. **Implementation of Servlet Filter Class**

**Login.java**

package com.jcg.filter;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

public class Login implements Filter {

    public void init(FilterConfig filterConfig) throws ServletException {   }

    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chainObj) throws IOException, ServletException {

        RequestDispatcher rdObj = null;

        PrintWriter out = resp.getWriter();

        out.write("<html><body><div id='servletResponse' style='text-align: center;'>");

        String password = req.getParameter("password");

        System.out.println("Password Is?= " + password);

        if(password != null && password.equals("admin")) {

            /\*\*\*\*\* Send Request To Next Resource \*\*\*\*\*/

            chainObj.doFilter(req, resp);

        } else {

            out.print("<p id='errMsg' style='color: red; font-size: larger;'>Username Or Password Is Invalid. Please Try Again ....!</p>");

            rdObj = req.getRequestDispatcher("/index.jsp");

            rdObj.include(req, resp);

        }

        out.write("</div></body></html>");

        out.close();

    }

    public void destroy() { }

}

1. **Implementation of Servlet Controller Class**

**Admin.java**

package com.jcg.filter;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class Admin extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        handleRequest(req, resp);

    }

    private void handleRequest(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        resp.setContentType("text/html");

        /\*\*\*\*\* Building & Printing The HTML Response Code \*\*\*\*\*/

        PrintWriter out = resp.getWriter();

        out.write("<html><body><div id='servletResponse' style='text-align: center;'>");

        out.write("<h2>Java Sevlet Filter Example</h2>");

        out.write("<p style='color: green; font-size: large;'>Welcome, Administrator!</p>");

        out.write("</div></body></html>");

        out.close();

    }

}

1. **Creating JSP Views**

**Index.jsp**

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "<http://www.w3.org/TR/html4/loose.dtd>">

<html>

    <head>

        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

        <title>Java Sevlet Filter</title>

        <style type="text/css">

            .paddingBtm {

                padding-bottom: 12px;

            }

        </style>

    </head>

    <body>

        <center>

            <h2>Java Sevlet Filter Example</h2>

            <form id="loginForm" action="servlet1">

                <div id="uDiv" class="paddingBtm">

                    Username: <input type="text" name="username" />

                </div>

                <div id="pDiv" class="paddingBtm">

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

                </div>

                <div id="sDiv">

                    <input id="btn" type="submit" value="Login" />

                </div>

            </form>

        </center>

    </body>

</html>

1. **Web Deployment Descriptor**

**Web.xml**

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

<web-app 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%C2%A0%C2%A0%C2%A0%C2%A0%C2%A0%C2%A0%C2%A0) <http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd>" version="3.0">

   <display-name>Servlet Application Login Example</display-name>

   <servlet>

      <servlet-name>Admin</servlet-name>

      <servlet-class>com.jcg.filter.Admin</servlet-class>

   </servlet>

   <servlet-mapping>

      <servlet-name>Admin</servlet-name>

      <url-pattern>/servlet1</url-pattern>

   </servlet-mapping>

   <filter>

      <filter-name>Login</filter-name>

      <filter-class>com.jcg.filter.Login</filter-class>

   </filter>

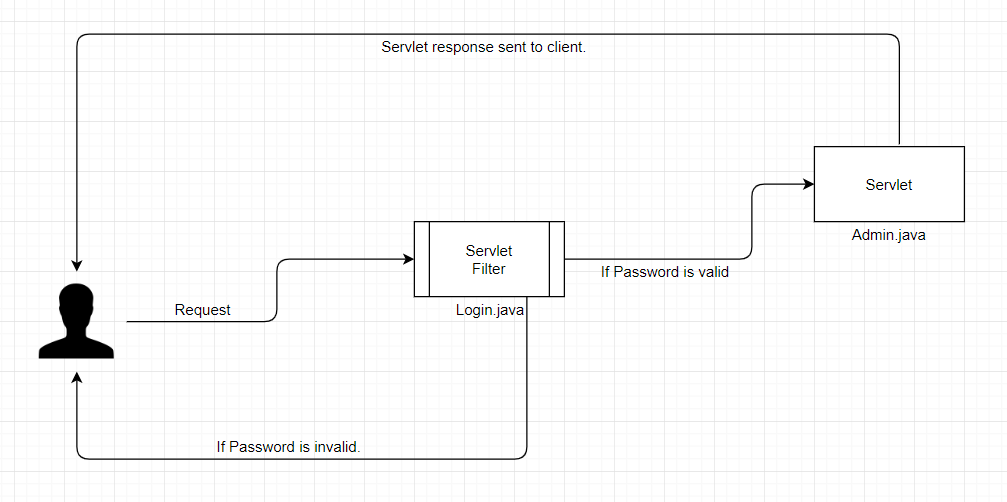
   <filter-mapping>

      <filter-name>Login</filter-name>

      <url-pattern>/servlet1</url-pattern>

   </filter-mapping>

</web-app>



**EXAMPLE**

**AuthenticationFilter.java**

package com.journaldev.servlet.filters;

import java.io.IOException;

import java.util.Enumeration;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletContext;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.annotation.WebFilter;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServletRequest;

/\*\*

\* Servlet Filter implementation class RequestLoggingFilter

\*/

@WebFilter("/RequestLoggingFilter")

public class RequestLoggingFilter implements Filter {

private ServletContext context;

public void init(FilterConfig fConfig) throws ServletException {

this.context = fConfig.getServletContext();

this.context.log("RequestLoggingFilter initialized");

}

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {

HttpServletRequest req = (HttpServletRequest) request;

Enumeration<String> params = req.getParameterNames();

while(params.hasMoreElements()){

String name = params.nextElement();

String value = request.getParameter(name);

this.context.log(req.getRemoteAddr() + "::Request Params::{"+name+"="+value+"}");

}

Cookie[] cookies = req.getCookies();

if(cookies != null){

for(Cookie cookie : cookies){

this.context.log(req.getRemoteAddr() + "::Cookie::{"+cookie.getName()+","+cookie.getValue()+"}");

}

}

// pass the request along the filter chain

chain.doFilter(request, response);

}

public void destroy() {

//we can close resources here

}

}

**RequestLoggingFilter.java**

package com.journaldev.servlet.filters;

import java.io.IOException;

import java.util.Enumeration;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletContext;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.annotation.WebFilter;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServletRequest;

/\*\*

\* Servlet Filter implementation class RequestLoggingFilter

\*/

@WebFilter("/RequestLoggingFilter")

public class RequestLoggingFilter implements Filter {

private ServletContext context;

public void init(FilterConfig fConfig) throws ServletException {

this.context = fConfig.getServletContext();

this.context.log("RequestLoggingFilter initialized");

}

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {

HttpServletRequest req = (HttpServletRequest) request;

Enumeration<String> params = req.getParameterNames();

while(params.hasMoreElements()){

String name = params.nextElement();

String value = request.getParameter(name);

this.context.log(req.getRemoteAddr() + "::Request Params::{"+name+"="+value+"}");

}

Cookie[] cookies = req.getCookies();

if(cookies != null){

for(Cookie cookie : cookies){

this.context.log(req.getRemoteAddr() + "::Cookie::{"+cookie.getName()+","+cookie.getValue()+"}");

}

}

// pass the request along the filter chain

chain.doFilter(request, response);

}

public void destroy() {

//we can close resources here

}

}

**LoginServlet.java**

package com.journaldev.servlet.session;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

/\*\*

\* Servlet implementation class LoginServlet

\*/

@WebServlet("/LoginServlet")

public class LoginServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

private final String userID = "admin";

private final String password = "password";

protected void doPost(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException {

// get request parameters for userID and password

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

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

if(userID.equals(user) && password.equals(pwd)){

HttpSession session = request.getSession();

session.setAttribute("user", "Saurav");

//setting session to expiry in 30 mins

session.setMaxInactiveInterval(30\*60);

Cookie userName = new Cookie("user", user);

userName.setMaxAge(30\*60);

response.addCookie(userName);

response.sendRedirect("LoginSuccess.jsp");

}else{

RequestDispatcher rd = getServletContext().getRequestDispatcher("/login.html");

PrintWriter out= response.getWriter();

out.println("<font color=red>Either user name or password is wrong.</font>");

rd.include(request, response);

}

}

}

**LogoutServlet.java**

package com.journaldev.servlet.session;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

/\*\*

\* Servlet implementation class LogoutServlet

\*/

@WebServlet("/LogoutServlet")

public class LogoutServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html");

Cookie[] cookies = request.getCookies();

if(cookies != null){

for(Cookie cookie : cookies){

if(cookie.getName().equals("JSESSIONID")){

System.out.println("JSESSIONID="+cookie.getValue());

break;

}

}

}

//invalidate the session if exists

HttpSession session = request.getSession(false);

System.out.println("User="+session.getAttribute("user"));

if(session != null){

session.invalidate();

}

response.sendRedirect("login.html");

}

}

**Login.html**

<!DOCTYPE html>

<html>

<head>

<meta charset="US-ASCII">

<title>Login Page</title>

</head>

<body>

<form action="LoginServlet" method="post">

Username: <input type="text" name="user">

<br>

Password: <input type="password" name="pwd">

<br>

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

</form>

</body>

</html>

**CheckoutPage.jsp**

<%@ **page** language="java" contentType="text/html; charset=US-ASCII"

pageEncoding="US-ASCII"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">

<title>Login Success Page</title>

</head>

<body>

<%

String userName = null;

String sessionID = null;

Cookie[] cookies = request.getCookies();

if(cookies !=null){

for(Cookie cookie : cookies){

if(cookie.getName().equals("user")) userName = cookie.getValue();

}

}

%>

<h3>Hi <%=userName %>, do the checkout.</h3>

<br>

<form action="LogoutServlet" method="post">

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

</form>

</body>

</html>

**LoginSuccess.jsp**

<%@ **page** language="java" contentType="text/html; charset=US-ASCII"

pageEncoding="US-ASCII"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">

<title>Login Success Page</title>

</head>

<body>

<%

//allow access only if session exists

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

String userName = null;

String sessionID = null;

Cookie[] cookies = request.getCookies();

if(cookies !=null){

for(Cookie cookie : cookies){

if(cookie.getName().equals("user")) userName = cookie.getValue();

if(cookie.getName().equals("JSESSIONID")) sessionID = cookie.getValue();

}

}

%>

<h3>Hi <%=userName %>, Login successful. Your Session ID=<%=sessionID %></h3>

<br>

User=<%=user %>

<br>

<a href="CheckoutPage.jsp">Checkout Page</a>

<form action="LogoutServlet" method="post">

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

</form>

</body>

</html>

**Web.xml**

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

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

<display-name>ServletFilterExample</display-name>

<welcome-file-list>

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

</welcome-file-list>

<filter>

<filter-name>RequestLoggingFilter</filter-name>

<filter-class>com.journaldev.servlet.filters.RequestLoggingFilter</filter-class>

</filter>

<filter>

<filter-name>AuthenticationFilter</filter-name>

<filter-class>com.journaldev.servlet.filters.AuthenticationFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>RequestLoggingFilter</filter-name>

<url-pattern>/\*</url-pattern>

<dispatcher>REQUEST</dispatcher>

</filter-mapping>

<filter-mapping>

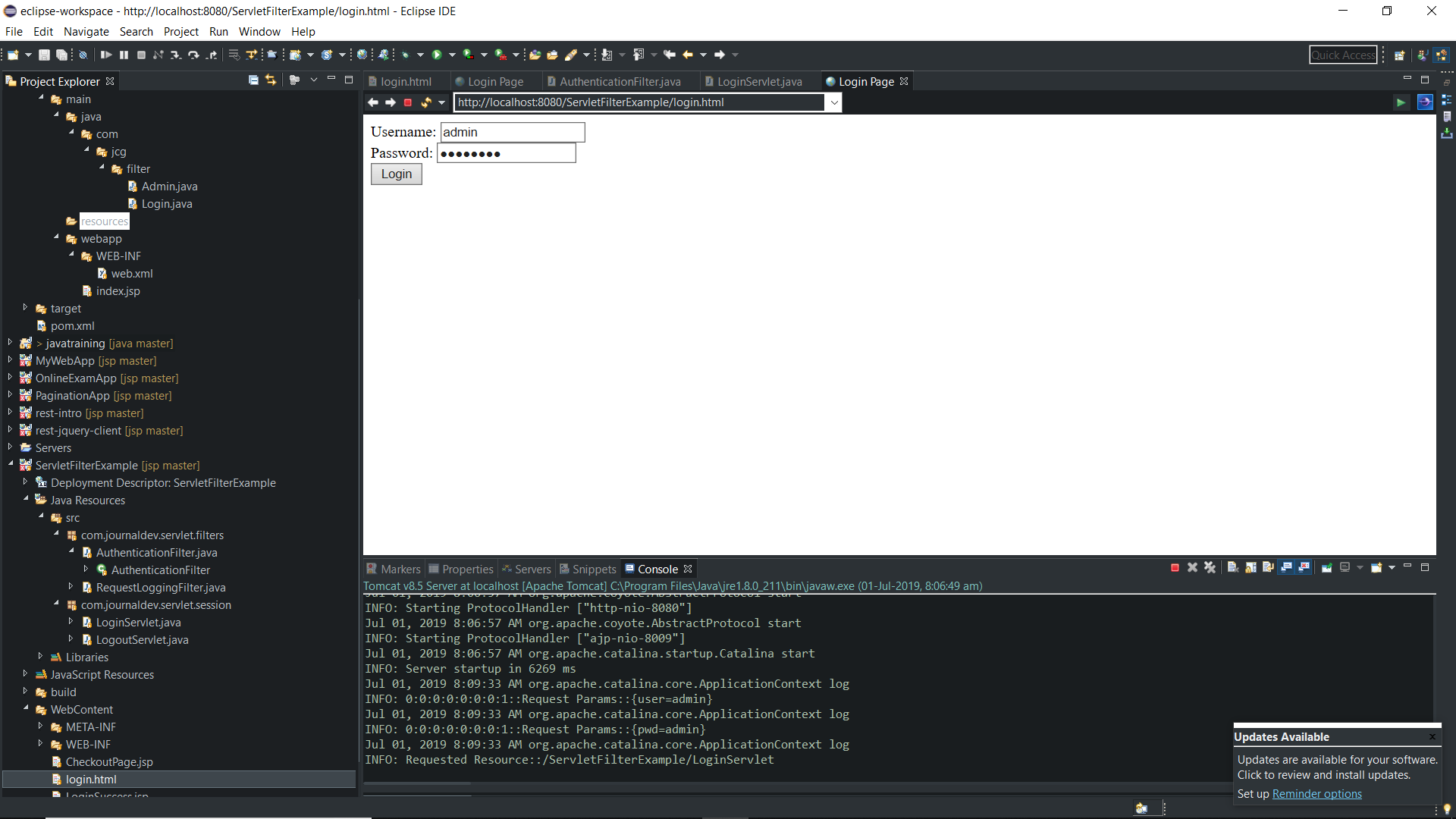
<filter-name>AuthenticationFilter</filter-name>

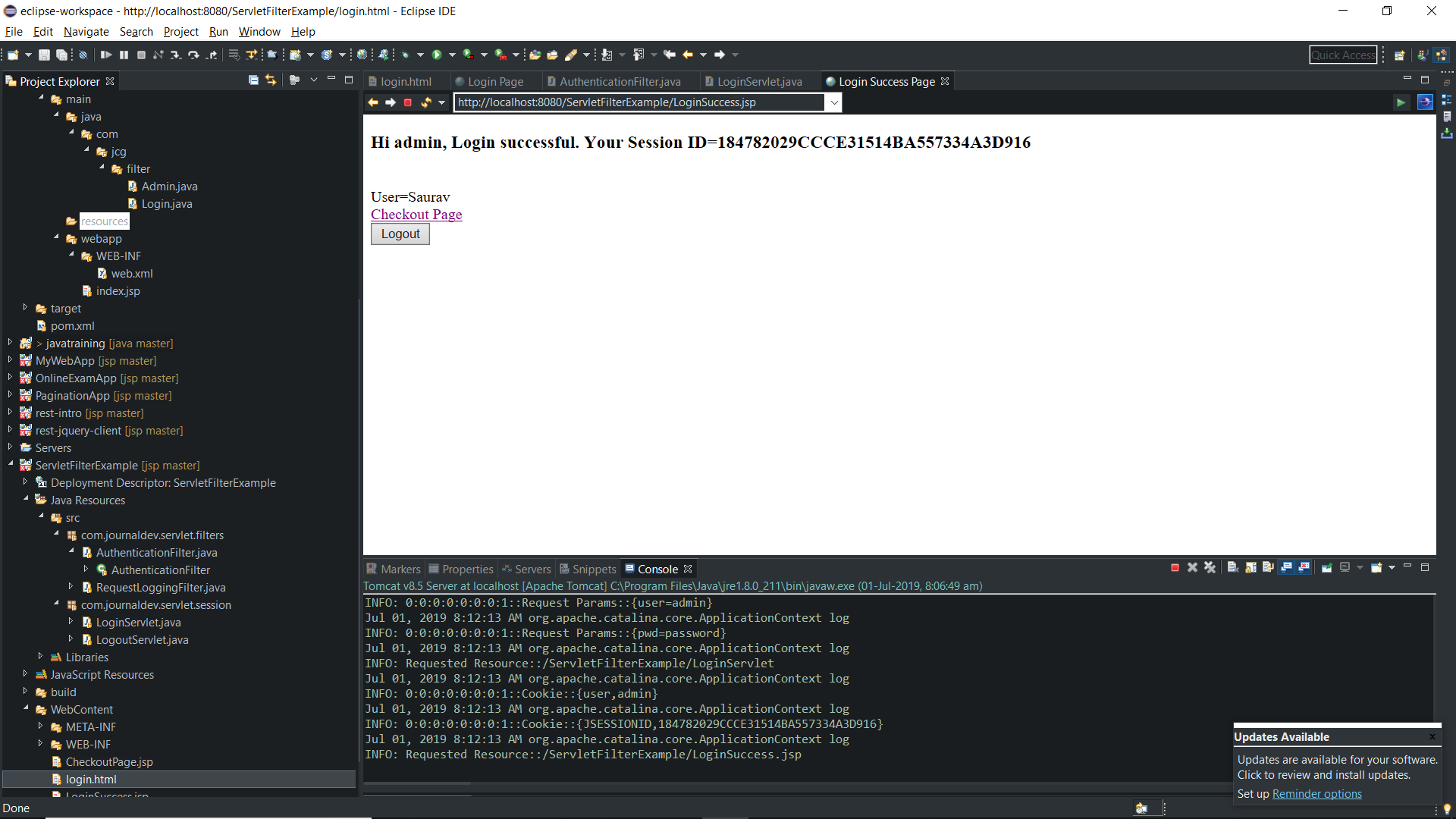
<url-pattern>/\*</url-pattern>

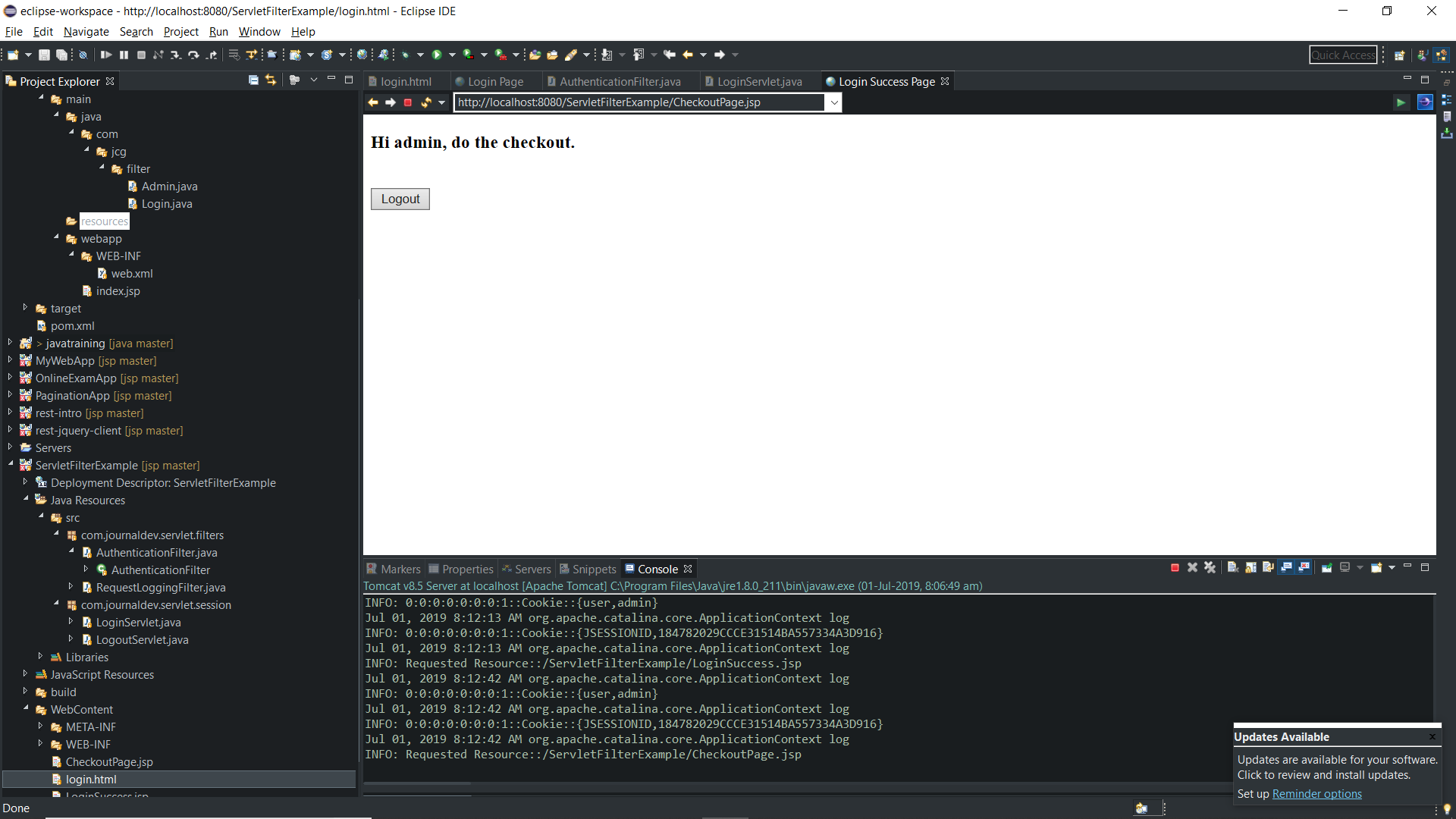
</filter-mapping>

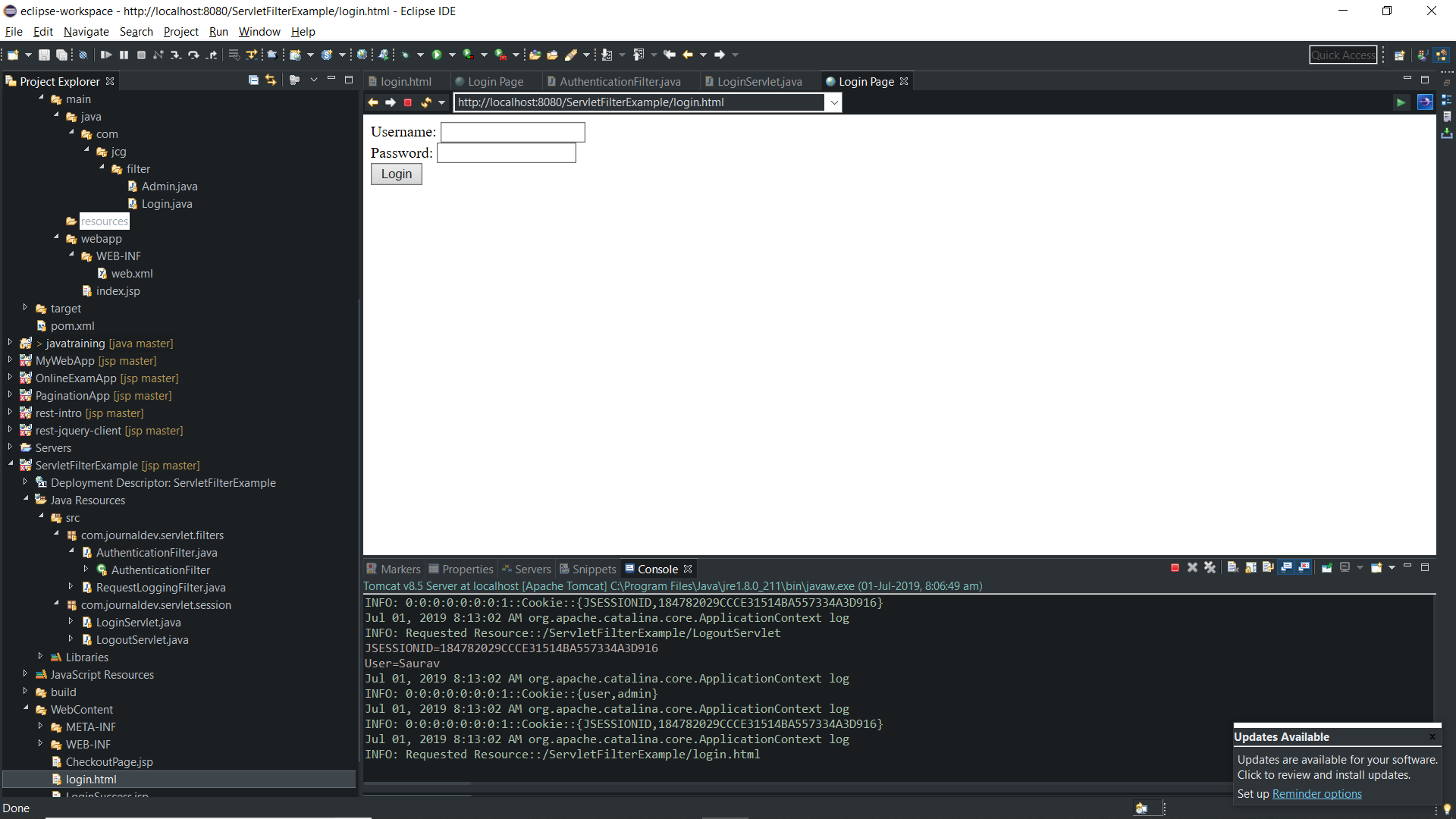
</web-app>

**OUTPUT**









**Code Download Links:-**

<https://github.com/white-wolf9/jsp/tree/master/JavaSevletFilter>

<https://github.com/white-wolf9/jsp/tree/master/ServletFilterExample>

MadeBy

1. Shreyansh Sinha
2. Saurav Sanyal