# Servlet

**By**

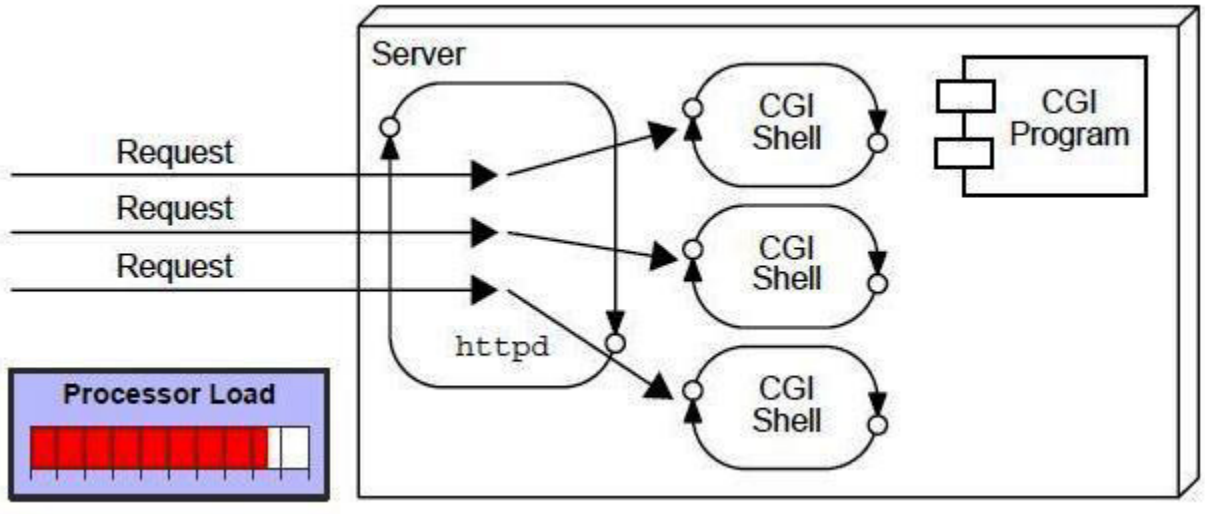
**Praveen Oruganti**

**Blog**: https://praveenoruganti.blogspot.com
**Facebook Group**: https://www.facebook.com/groups/2340886582907696/
**Github repo:** https://github.com/praveenoruganti

## Servlet concepts

1. **What is a Servlet?**
   Servlet technology is used to create web application (resides at server side and generates dynamic web page)
2. **What is web application?**
   A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.
3. **CGI(Commmon Gateway Interface)**

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

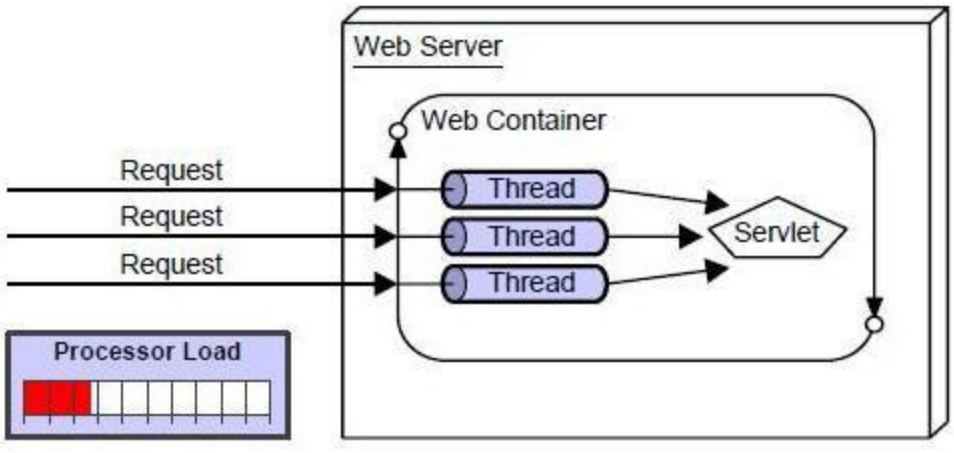



Disadvantages of CGI

There are many problems in CGI technology:

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

4. **Advantages of a Servlet**



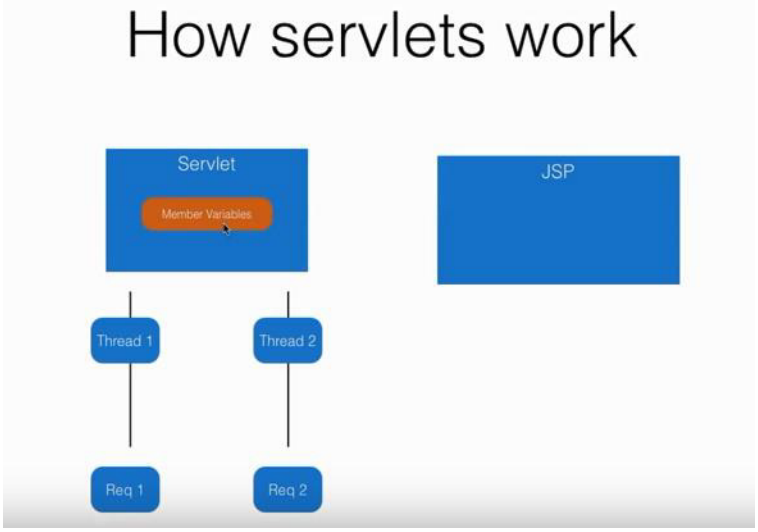

There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits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

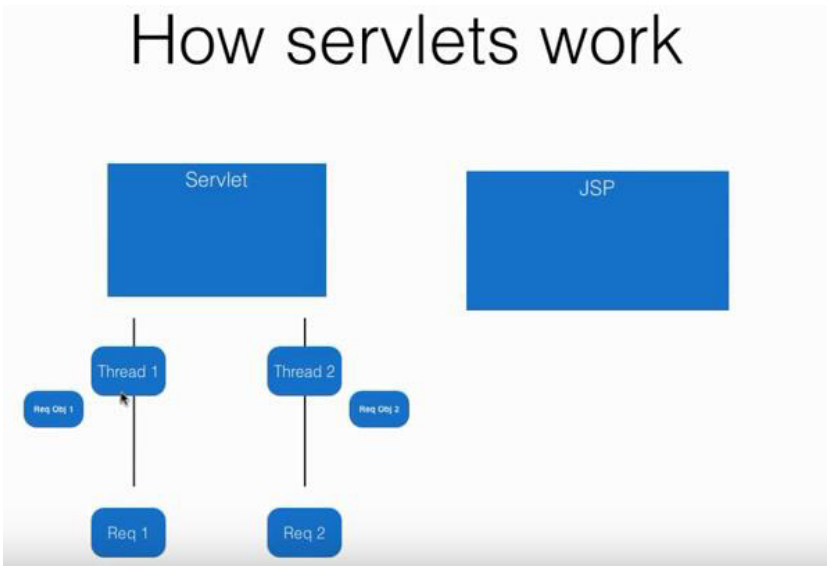
- ✓ **better performance:** because it creates a thread for each request not process.
- ✓ **Portability:** because it uses java language.
- ✓ **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
- ✓ **Secure:** because it uses java language..

**5. How Servlet works**

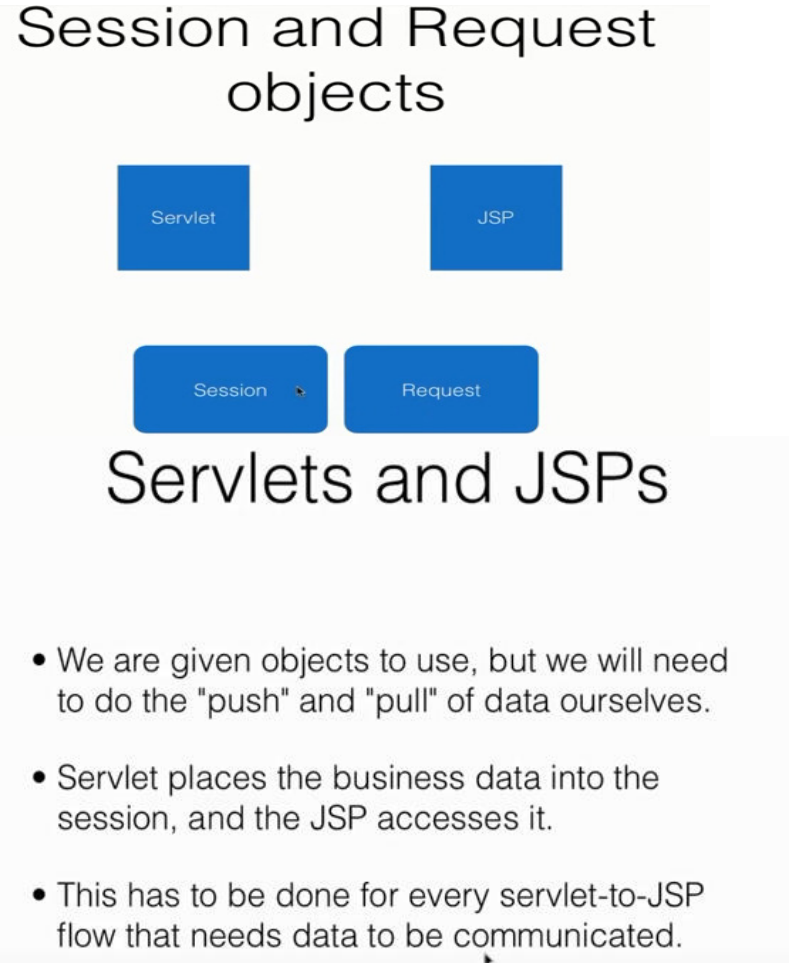
For every client request to Servlet, a thread will be created by webcontainer and process it accordingly. We need to make sure our application is thread safe as it is a thread base.

New request objects will be created to store the data.




Data transfer between Servlet and JSP is generally done based on request and session objects.




### 6. Servlet Container

It provides the runtime environment for JavaEE (j2ee) applications. The client/user can request only a static WebPages from the server. If the user wants to read the web pages as per input then the servlet container is used in java.

The servlet container is used in java for dynamically generate the web pages on the server side. Therefore the servlet container is the part of a web server that interacts with the servlet for handling the dynamic web pages from the client.

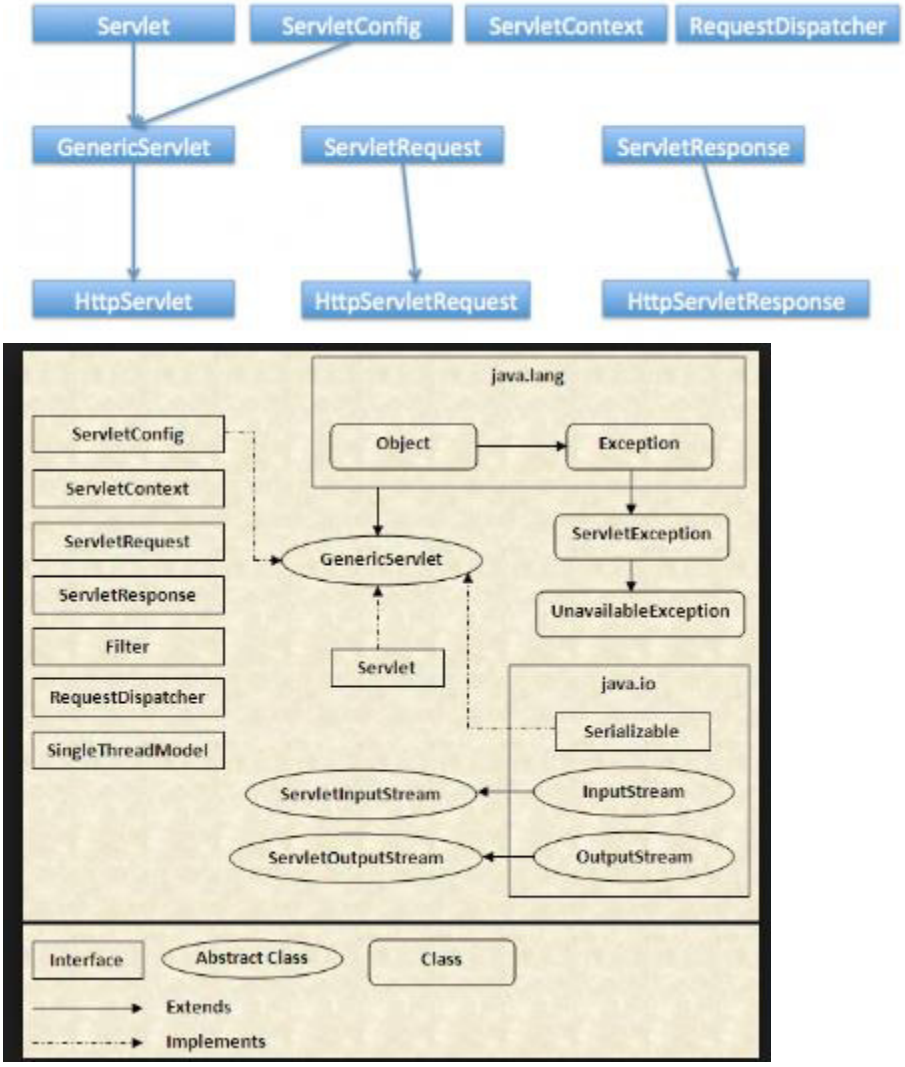
### Web Server

Web server contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB.

### Application Server

Application server contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc. It is a component based product that lies in the middle-tier of a server centric architecture.

## 7. Servlet API




### Servlet Interface

javax.servlet.Servlet is the base interface of Java Servlet API. Servlet interface declares the life cycle methods of servlet. All the servlet classes are required to implement this interface. The methods declared in this interface are:

- ✓ public abstract void init(ServletConfig paramServletConfig) throws ServletException
- ✓ public abstract ServletConfig getServletConfig()

- ✓ public abstract void service(ServletRequest req, ServletResponse res) throws ServletException, IOException
- ✓ public abstract String getServletInfo()
- ✓ public abstract void destroy()

| Method | Description |
|---|---|
| public void init (ServletConfig config) | Called by the servlet container to indicate to a servlet that the servlet is being placed into service.<br>It initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once. |
| public void service (ServletRequest request, ServletResponse response) | Called by the servlet container to allow the servlet to respond to a request.<br>It provides response for the incoming request. It is invoked at each request by the web container. |
| public void destroy() | Called by the servlet container to indicate to a servlet that the servlet is being taken out of service.<br>It is invoked only once and indicates that servlet is being destroyed. |
| public ServletConfig getServletConfig() | returns the object of ServletConfig, which the servlet can use to get any startup information |
| public String getServletInfo() | which allows the servlet to return basic information about itself, such as author, version, and copyright |

ServletConfig Interface

javax.servlet.ServletConfig is used to pass configuration information to Servlet. Every servlet has it's own ServletConfig object and servlet container is responsible for instantiating this object. We can provide servlet init parameters in **web.xml** file or through use of @WebInitParam annotation. We can use **getServletConfig()** method to get the ServletConfig object of the servlet.

```
@WebServlet(
		description = "Login Servlet",
		urlPatterns = { "/LoginServlet" },
		initParams = {
				@WebInitParam(name = "user", value = "praveen"),
				@WebInitParam(name = "password", value = "praveen")
		})
```

```
<servlet>
  <servlet-name>loginServlet</servlet-name>
  <servlet-class>com.praveen.LoginServlet</servlet-class>

    <init-param>
    <param-name>user</param-name>
    <param-value>Praveen</param-value>
    </init-param>
</servlet>
```

The important methods of ServletConfig interface are:

- ✓ public abstract ServletContext getServletContext()
- ✓ public abstract Enumeration<String> getInitParameterNames()
- ✓ public abstract String getInitParameter(String paramString)

ServletContext interface

javax.servlet.ServletContext interface provides access to web application variables to the servlet. The ServletContext is unique object and available to all the servlets in the web application. When we want some init parameters to be available to multiple or all of the servlets in the web application, we can use ServletContext object and define parameters in web.xml using **<context-param>** element. We can get the ServletContext object via the getServletContext() method of ServletConfig. Servlet engines may also provide context objects that are unique to a group of servlets and which is tied to a specific portion of the URL path namespace of the host.

```
<context-param>
		<param-name>contextConfigLocation</param-name>
		<param-value>/WEB-INF/applicationContext.xml</param-value>
		<description>WebFlow context configuration</description>
	</context-param>
```

Some of the important methods of ServletContext are:

- ✓ public abstract ServletContext getContext(String uripath)
- ✓ public abstract URL getResource(String path) throws MalformedURLException
- ✓ public abstract InputStream getResourceAsStream(String path)
- ✓ public abstract RequestDispatcher getRequestDispatcher(String urlpath)
- ✓ public abstract void log(String msg)
- ✓ public abstract Object getAttribute(String name)
- ✓ public abstract void setAttribute(String paramString, Object paramObject)
- ✓ String getInitParameter(String name)
- ✓ boolean setInitParameter(String paramString1, String paramString2)

ServletRequest interface

ServletRequest interface is used to provide client request information to the servlet. Servlet container creates ServletRequest object from client request and pass it to the servlet service() method for processing.

Some of the important methods of ServletRequest interface are:

- ✓ Object getAttribute(String name)
- ✓ String getParameter(String name)
- ✓ String getServerName()
- ✓ int getServerPort()

ServletResponse interface

ServletResponse interface is used by servlet in sending response to the client. Servlet container creates the ServletResponse object and pass it to servlet service() method and later use the response object to generate the HTML response for client.

Some of the important methods in HttpServletResponse are:

- ✓ void addCookie(Cookie cookie)
- ✓ void addHeader(String name, String value)
- ✓ String encodeURL(java.lang.String url)
- ✓ String getHeader(String name)
- ✓ void sendRedirect(String location) - used to send a temporary redirect response to the client using the specified redirect location URL.
- ✓ void setStatus(int sc)

RequestDispatcher interface

RequestDispatcher interface is used to forward the request to another resource that can be HTML, JSP or another servlet in the same context. We can also use this to include the content of another resource to the response. This interface is used for servlet communication within the same context.

There are two methods defined in this interface:

- ✓ void forward(ServletRequest request, ServletResponse response) – forwards the 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.

GenericServlet class

GenericServlet is an abstract class that implements Servlet, ServletConfig and Serializable interface. GenericServlet provide default implementation of all the Servlet life cycle methods and ServletConfig methods and makes our life easier when we extend this class, we need to override only the methods we want and rest of them we can work with the default implementation. Most of the methods defined in this class are only for easy access to common methods defined in Servlet and ServletConfig interfaces.

One of the important method in GenericServlet class is no-argument init() method and we should override this method in our servlet program if we have to initialize some resources before processing any request from servlet.

HTTPServlet class

HTTPServlet is an abstract class that extends GenericServlet and provides base for creating HTTP based web applications. There are methods defined to be overridden by subclasses for different HTTP methods.

- ✓ doGet(), for HTTP GET requests
- ✓ doPost(), for HTTP POST requests
- ✓ doPut(), for HTTP PUT requests
- ✓ doDelete(), for HTTP DELETE requests

Servlet Attributes

Servlet attributes are used for inter-servlet communication, we can set, get and remove attributes in web application. There are three scopes for servlet attributes – **request scope**, **session scope** and **application scope**.

**ServletRequest**, **HttpSession** and **ServletContext** interfaces provide methods to get/set/remove attributes from request, session and application scope respectively.

Servlet attributes are different from init parameters defined in **web.xml** for ServletConfig or ServletContext.

Annotations in Servlet 3

Prior to Servlet 3, all the servlet mapping and it's init parameters were used to defined in web.xml, this was not convenient and more error prone when number of servlets are huge in an application.

Servlet 3 introduced use of java annotations to define a servlet, filter and listener servlets and init parameters.

Some of the important Servlet annotations are:

- ✓ **WebServlet** – We can use this annotation with Servlet classes to define init parameters, loadOnStartup value, description and url patterns etc. At least one URL pattern MUST be declared in either the value or urlPattern attribute of the annotation, but not both. The class on which this annotation is declared MUST extend HttpServlet.
- ✓ **WebInitParam** – This annotation is used to define init parameters for servlet or filter, it contains name, value pair and we can provide description also. This annotation can be used within a WebFilter or WebServlet annotation.
- ✓ **WebFilter** – This annotation is used to declare a servlet filter. This annotation is processed by the container during deployment, the Filter class in which it is found will be created as per the configuration and applied to the URL patterns, Servlets and DispatcherTypes. The annotated class MUST implement javax.servlet.Filter interface.
- ✓ **WebListener** – The annotation used to declare a listener for various types of event, in a given web application context.

**LoginServlet Example**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
 <display-name>LoginExample</display-name>
  <welcome-file-list>
    <welcome-file>login.html</welcome-file>
  </welcome-file-list>

  <context-param>
  <param-name>dbURL</param-name>
  <param-value>jdbc:mysql://localhost/mysql_db</param-value>
  </context-param>
  <context-param>
  <param-name>dbUser</param-name>
  <param-value>mysql_user</param-value>
  </context-param>
  <context-param>
  <param-name>dbUserPwd</param-name>
  <param-value>mysql_pwd</param-value>
  </context-param>
</web-app>
```

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>
```

LoginSuccess.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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>
<h3>Hi Praveen, Login successful.</h3>
<a href="login.html">Login Page</a>
</body>
</html>
```

LoginServlet.java

```
package com.praveen.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(
                description = "Login Servlet",
                urlPatterns = { "/LoginServlet" },
                initParams = {
                                @WebInitParam(name = "user", value = "praveen"),
                                @WebInitParam(name = "password", value = "praveen")
                })
public class LoginServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;


        public void init() throws ServletException {
                //we can create DB connection resource here and set it to Servlet context
```

```
        if(getServletContext().getInitParameter("dbURL").equals("jdbc:mysql://localhost/
mysql_db") &&

        getServletContext().getInitParameter("dbUser").equals("mysql_user") &&

        getServletContext().getInitParameter("dbUserPwd").equals("mysql_pwd"))
                getServletContext().setAttribute("DB_Success", "True");
                else throw new ServletException("DB Connection error");
        }


        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");

                //get servlet config init params
                String userID = getServletConfig().getInitParameter("user");
                String password = getServletConfig().getInitParameter("password");
                //logging example
                log("User="+user+"::password="+pwd);

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

                }

        }
}
```

## 8. Session Management in Java – HttpServlet, Cookies, URL Rewriting

**Session Management in Java** Servlet Web Applications is a very interesting topic. **Session in Java** Servlet are managed through different ways, such as Cookies, **HttpSession** API, URL rewriting etc.

What is a Session?

HTTP protocol and Web Servers are stateless, what it means is that for web server every request is a new request to process and they can't identify if it's coming from client that has been sending request previously.

But sometimes in web applications, we should know who the client is and process the request accordingly. For example, a shopping cart application should know who is sending the request to add an item and in which cart the item has to be added or who is sending checkout request so that it can charge the amount to correct client.

**Session** is a conversional state between client and server and it can consists of multiple request and response between client and server. Since HTTP and Web Server both are stateless, the only way to maintain a session is when some unique information about the session (session id) is passed between server and client in every request and response.

There are several ways through which we can provide unique identifier in request and response.

- ✓ **User Authentication** – This is the very common way where we user can provide authentication credentials from the login page and then we can pass the authentication information between server and client to maintain the session. This is not very effective method because it wont work if the same user is logged in from different browsers.
- ✓ **HTML Hidden Field** – We can create a unique hidden field in the HTML and when user starts navigating, we can set its value unique to the user and keep track of the session. This method can't be used with links because it needs the form to be submitted every time request is made from client to server with the hidden field. Also it's not secure because we can get the hidden field value from the HTML source and use it to hack the session.
- ✓ **URL Rewriting** – We can append a session identifier parameter with every request and response to keep track of the session. This is very tedious because we need to keep track of this parameter in every response and make sure it's not clashing with other parameters.
- ✓ **Cookies** – Cookies are small piece of information that is sent by web server in response header and gets stored in the browser cookies. When client make further request, it adds the cookie to the request header and we can utilize it to keep track of the session. We can maintain a session with cookies but if the client disables the cookies, then it won't work.

- ✓ **Session Management API** – Session Management API is built on top of above methods for session tracking. Some of the major disadvantages of all the above methods are:
  - Most of the time we don't want to only track the session, we have to store some data into the session that we can use in future requests. This will require a lot of effort if we try to implement this.
  - All the above methods are not complete in themselves, all of them won't work in a particular scenario. So we need a solution that can utilize these methods of session tracking to provide session management in all cases.

  That's why we need Session Management API and J2EE Servlet technology comes with session management API that we can use.

**Session Management in Java – Cookies**

Cookies are used a lot in web applications to personalize response based on your choice or to keep track of session. Before moving forward to the Servlet Session Management API, I would like to show how can we keep track of session with cookies through a small web application.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
 <display-name>LoginExample</display-name>
  <welcome-file-list>
    <welcome-file>loginWithCookie.html</welcome-file>
  </welcome-file-list>
</web-app>
```

loginWithCookie.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="US-ASCII">
<title>Login Page</title>
</head>
<body>

<form action="LoginServletWithCookie" method="post">
```

```
Username: <input type="text" name="user">
<br>
Password: <input type="password" name="pwd">
<br>
<input type="submit" value="Login">
</form>
</body>
</html>
```

LoginServletWithCookie.java

```
package com.praveen.servlet;

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;


@WebServlet("/LoginServletWithCookie")
public class LoginServletWithCookie extends HttpServlet {
	private static final long serialVersionUID = 1L;
	private final String userID = "praveen";
	private final String password = "praveen";

	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)){
			Cookie loginCookie = new Cookie("user",user);
			//setting cookie to expiry in 30 mins
			loginCookie.setMaxAge(30*60);
			response.addCookie(loginCookie);
			response.sendRedirect("LoginSuccessWithCookie.jsp");
```

```
		}else{
			RequestDispatcher rd =
getServletContext().getRequestDispatcher("/loginWithCookie.html");
			PrintWriter out= response.getWriter();
			out.println("<font color=red>Either user name or password is
wrong.</font>");
			rd.include(request, response);
		}

	}

}
```

LoginSuccessWithCookie.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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;
Cookie[] cookies = request.getCookies();
if(cookies !=null){
for(Cookie cookie : cookies){
	if(cookie.getName().equals("user")) userName = cookie.getValue();
}
}
if(userName == null) response.sendRedirect("loginWithCookie.html");
%>
<h3>Hi <%=userName %>, Login successful.</h3>
<br>
<form action="LogoutServletWithCookie" method="post">
<input type="submit" value="Logout" >
</form>
</body>
</html>
```

LogoutServletWithCookie.java

```
package com.praveen.servlet;

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;

@WebServlet("/LogoutServletWithCookie")
public class LogoutServletWithCookie extends HttpServlet {
	private static final long serialVersionUID = 1L;

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
	response.setContentType("text/html");
	Cookie loginCookie = null;
	Cookie[] cookies = request.getCookies();
	if(cookies != null){
	for(Cookie cookie : cookies){
		if(cookie.getName().equals("user")){
			loginCookie = cookie;
			break;
		}
	}
	}
	if(loginCookie != null){
		loginCookie.setMaxAge(0);
	response.addCookie(loginCookie);
	}
	response.sendRedirect("loginWithCookie.html");
    }

}
```

**Session in Java Servlet – HttpSession**

Servlet API provides Session management through HttpSession interface. We can get session from HttpServletRequest object using following methods. HttpSession allows us to set objects as attributes that can be retrieved in future requests.

- ✓ HttpSession getSession() – This method always returns a HttpSession object. It returns the session object attached with the request, if the request has no session attached, then it creates a new session and return it.

- ✓ HttpSession getSession(boolean flag) – This method returns HttpSession object if request has session else it returns null.

Some of the important methods of HttpSession are:

- ✓ String getId() – Returns a string containing the unique identifier assigned to this session.
- ✓ Object getAttribute(String name) – Returns the object bound with the specified name in this session, or null if no object is bound under the name. Some other methods to work with Session attributes are getAttributeNames(), removeAttribute(String name) and setAttribute(String name, Object value).
- ✓ long getCreationTime() – Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT. We can get last accessed time with getLastAccessedTime() method.
- ✓ setMaxInactiveInterval(int interval) – Specifies the time, in seconds, between client requests before the servlet container will invalidate this session. We can get session timeout value from getMaxInactiveInterval() method.
- ✓ ServletContext getServletContext() – Returns ServletContext object for the application.
- ✓ boolean isNew() – Returns true if the client does not yet know about the session or if the client chooses not to join the session.
- ✓ void invalidate() – Invalidates this session then unbinds any objects bound to it.

Understanding JSESSIONID Cookie

When we use HttpServletRequest getSession() method and it creates a new request, it creates the new HttpSession object and also add a Cookie to the response object with name JSESSIONID and value as session id. This cookie is used to identify the HttpSession object in further requests from client. If the cookies are disabled at client side and we are using URL rewriting then this method uses the jsessionid value from the request URL to find the corresponding session. JSESSIONID cookie is used for session tracking, so we should not use it for our application purposes to avoid any session related issues.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
 <display-name>LoginExample</display-name>
```

```
  <welcome-file-list>
    <welcome-file>loginWithSession.html</welcome-file>
  </welcome-file-list>
</web-app>
```

loginWithSession.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="US-ASCII">
<title>Login Page</title>
</head>
<body>

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

Username: <input type="text" name="user">
<br>
Password: <input type="password" name="pwd">
<br>
<input type="submit" value="Login">
</form>
</body>
</html>
```

LoginServletWithSession.java

```
package com.praveen.servlet;



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;

@WebServlet("/LoginServletWithSession")
public class LoginServletWithSession 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", "Praveen");
                //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("CheckoutPage.jsp");
            }else{
                RequestDispatcher rd =
getServletContext().getRequestDispatcher("/loginWithSession.html");
                PrintWriter out= response.getWriter();
                out.println("<font color=red>Either user name or password is
wrong.</font>");
                rd.include(request, response);
            }

        }

}
```

CheckoutPage.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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
if(session.getAttribute("user") == null){
        response.sendRedirect("loginWithSession.html");
}
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="LogoutServletWithSession" method="post">
<input type="submit" value="Logout" >
</form>
</body>
</html>
```

LogoutServletWithSession.java

```
package com.praveen.servlet;



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;


@WebServlet("/LogoutServletWithSession")

public class LogoutServletWithSession 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("loginWithSession.html");
    }

}
```

**Session Management in Java Servlet – URL Rewriting**

As we saw in last section that we can manage a session with HttpSession but if we disable the cookies in browser, it won't work because server will not receive the JSESSIONID cookie from client. Servlet API provides support for URL rewriting that we can use to manage session in this case.

The best part is that from coding point of view, it's very easy to use and involves one step – encoding the URL. Another good thing with Servlet URL Encoding is that it's a fallback approach and it kicks in only if browser cookies are disabled.

We can encode URL with HttpServletResponse encodeURL() method and if we have to redirect the request to another resource and we want to provide session information, we can use encodeRedirectURL() method.

We will create a similar project like above except that we will use URL rewriting methods to make sure session management works fine even if cookies are disabled in browser.

web.xml

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

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
 <display-name>LoginExample</display-name>
  <welcome-file-list>
    <welcome-file>loginWithURLRewriting.html</welcome-file>
  </welcome-file-list>
</web-app>
```

LoginWithURLRewriting.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="US-ASCII">
<title>Login Page</title>
</head>
<body>

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

Username: <input type="text" name="user">
<br>
Password: <input type="password" name="pwd">
<br>
<input type="submit" value="Login">
</form>
</body>
</html>
```

LoginWithURLRewriting.java

```
package com.praveen.servlet;

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;

@WebServlet("/LoginServletWithURLRewriting")
public class LoginServletWithURLRewriting 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", "Praveen");
			//setting session to expiry in 30 mins
			session.setMaxInactiveInterval(30*60);
			Cookie userName = new Cookie("user", user);
			response.addCookie(userName);
			//Get the encoded URL string
			String encodedURL =
response.encodeRedirectURL("LoginSuccessWithURLRewriting.jsp");
			response.sendRedirect(encodedURL);
		}else{
			RequestDispatcher rd =
getServletContext().getRequestDispatcher("/loginWithURLRewriting.html");
			PrintWriter out= response.getWriter();
			out.println("<font color=red>Either user name or password is
wrong.</font>");
			rd.include(request, response);
		}

	}

}
```

LoginSuccessWithURLRewriting.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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 = null;
if(session.getAttribute("user") == null){
        response.sendRedirect("loginWithURLRewriting.html");
}else 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();
}
}else{
        sessionID = session.getId();
}
%>
<h3>Hi <%=userName %>, Login successful. Your Session ID=<%=sessionID %></h3>
<br>
User=<%=user %>
<br>
<!-- need to encode all the URLs where we want session information to be passed -->
<a href="<%=response.encodeURL("CheckoutPageWithURLRewriting.jsp")
%>">Checkout Page</a>
<form action="<%=response.encodeURL("LogoutServletWithURLRewriting") %>"
method="post">
<input type="submit" value="Logout" >
</form>
</body>
</html>
```

CheckoutPageWithURLRewriting.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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;
//allow access only if session exists
if(session.getAttribute("user") == null){
        response.sendRedirect("loginWithURLRewriting.html");
}else userName = (String) session.getAttribute("user");
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="<%=response.encodeURL("LogoutServletWithURLRewriting") %>"
method="post">
<input type="submit" value="Logout" >
</form>
</body>
</html>
```

LogoutServletWithURLRewriting.java

```
package com.praveen.servlet;



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;


@WebServlet("/LogoutServletWithURLRewriting")

public class LogoutServletWithURLRewriting 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());

                }

                cookie.setMaxAge(0);

                response.addCookie(cookie);
```

```
        }
        }
        //invalidate the session if exists
        HttpSession session = request.getSession(false);
        System.out.println("User="+session.getAttribute("user"));
        if(session != null){
                session.invalidate();
        }
        //no encoding because we have invalidated the session
        response.sendRedirect("loginWithURLRewriting.html");
    }

}
```

## Recap of Session Management.

**See the following Example:**

**Eg : 5**

**test1.html:**

```
<Form action="Servlet3" name = "Form1">

        Param 1      : <inputtype="text"name="param1"/><br>
```

```
Param 2        :
                <inputtype="text"name="para
                m2"/><br>
                <inputtype="submit"value="su
                bmit"/>

</Form>
```

**Servlet1.java:**

```
Public class Servlet1 extends HttpServlet {

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

                PrintWriter out = response.getWriter();

                out.println("Param 1 : " + request.getParameter("param1") + "<br>");
                out.println("Param 2 : " + request.getParameter("param2") + "<br>");

                out.println("<Form action = 'Servlet2' name = 'Form2'>");
                out.println("Param 3 : <input type = 'text' name = 'param3' /><br>");
                out.println("Param 4 : <input type = 'text' name = 'param4' /><br>");
                out.println("<input type = 'submit' value = 'submit' />");
                out.println("</Form>");

        }
}
```

**Servlet2.java:**

```
public class Servlet2 extends HttpServlet {
        protected void doGet(HttpServletRequest request,

                                HttpServletResponse response)
                        throws ServletException, IOException {

                PrintWriter out = response.getWriter();

                out.println("Param 1 : " + request.getParameter("param1") + "<br>");
                out.println("Param 2 : " + request.getParameter("param2") + "<br>");

                out.println("Param 3 : " + request.getParameter("param3") + "<br>");

                out.println("Param 4 : " + request.getParameter("param4") + "<br>");
        }
```

**What is a session?**

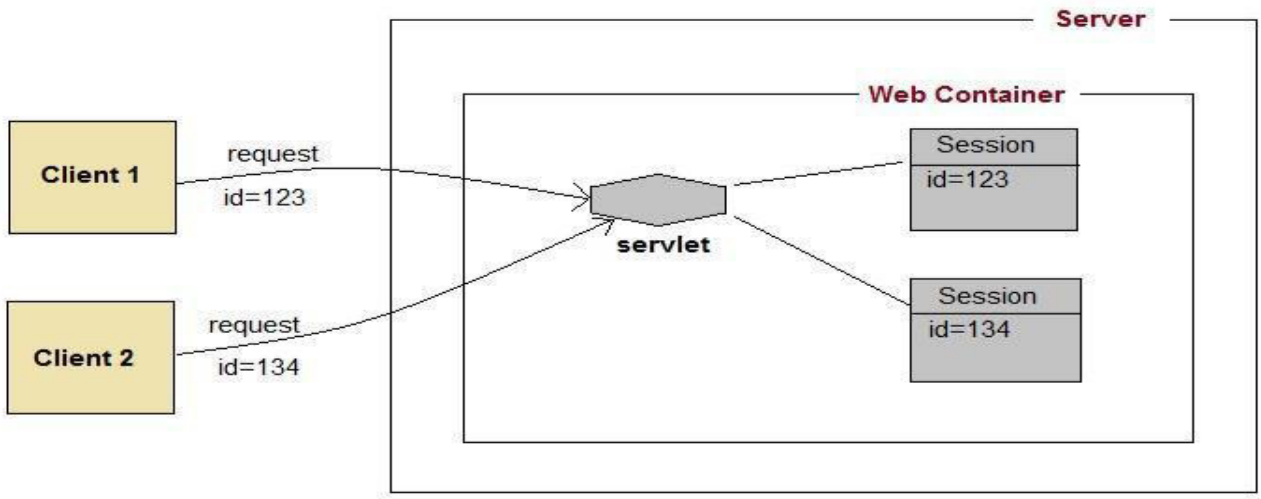
HTTP protocol and Web Servers are stateless, what it means is that for web server every request is a new request to process and they can't identify if it's coming from client that has been sending request previously.

We all know that **Http** is a stateless protocol. Each request and response is independent. But somehow you need to keep track of client activity across multiple request i.e storing session information for a particular client.

When a client sends a request, the server sends back a response but does not keep any information about that requestand the client state.

In most web applications a client has to access various pagesbefore completing a specific task and the client state shouldbe kept along all those pages

### *How sessions work*




**The techniques for managing the state of an end user are:**

1. Hidden form field
2. URL rewriting
3. Cookies

4. Servlet session API [ HttpSession]

**1. <u>Hidden form field:</u>**

- ✓ Simplest technique to maintain the state of an end user.
- ✓ Embedded in an HTML form.

- ✓ The client state is passed from the server to the client and back to the server in a hidden

  field of a form

- ✓ 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 retrieve from another servlet.

- ✓ Not visible when you view an HTML file in a browser window.

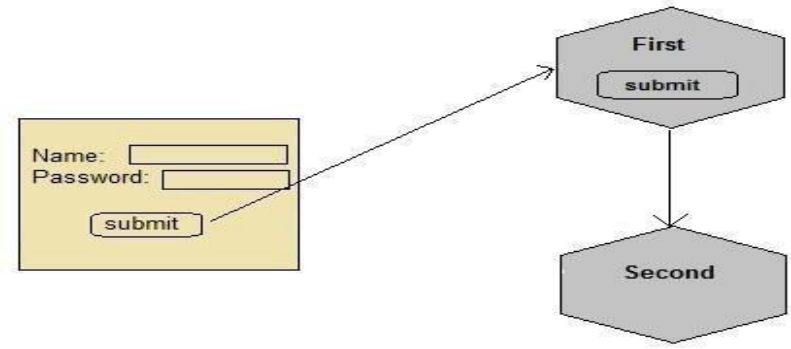
- ✓ Not able to maintain the state of an end user when it encounters a static document.

### *Advantage*

- ✓ Does not have to depend on browser whether the cookie is disabled or not.

### *Disadvantage*

- ✓ Extra form submission is required on every page.




**Eg :6**

**test1.html:**

```
<Formaction="Servlet3" name = “Form1”>

        Param 1       : <inputtype="text"name="param1"/><br>

        Param 2       :
                        <inputtype="text"name="param2
                        "/><br>
                        <inputtype="submit"value="sub
                        mit"/>

</Form>
```

**Servlet3.java:**

```
public class Servlet3 extends HttpServlet {

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

                PrintWriter out = response.getWriter();

                String s1 =
                request.getParameter("param1"); String
                s2 = request.getParameter("param2");
                out.println("Param 1 : " + s1 + "<br>");
                out.println("Param 2 : " + s2 + "<br>");
```

```
            out.println("<Form Action = 'Servlet4' name = 'Form 2' method = 'post'>");

            // Adding 2 Hidden Fields.

            out.println("<input type = 'hidden' name = 'param1' value = '"+s1+"'/> ");
            out.println("<input type = 'hidden' name = 'param2' value = '"+s2+"'/> ");

            out.println("Param 3 : <input type = 'text' name = 'param3' /><br>");
            out.println("Param 4 : <input type = 'text' name = 'param4' /><br>");
            out.println("<input type = 'submit' value = 'submit' />");
            out.println("</Form>");
        }

}
```

**Servlet4.java:**

```
public class Servlet4 extends HttpServlet {
        protected void doGet(HttpServletRequest request,

                                HttpServletResponse response)
                        throws ServletException, IOException {

            PrintWriter out = response.getWriter();

            out.println("Param 1 : " + request.getParameter("param1") + "<br>");

            out.println("Param 2 : " + request.getParameter("param2") + "<br>");

            out.println("Param 3 : " + request.getParameter("param3") + "<br>");
```

```
            out.println("Param 4 : " + request.getParameter("param4") + "<br>");

        }

    }
```

**2. URL Rewriting**

- ✓ Maintains the state of end user by modifying the URL.
- ✓ Is used when the information to be transferred is not critical.
- ✓ Cannot be used for maintaining the state of an end user when a static document is encountered.
- ✓ User can't modify the URL.

Explanation:

The client state is passed from the server to the client andback to the server in the query string, accompanying theURL

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:url?name1=value1&name2=value2&??

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

**Advantage of URL Rewriting**

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

### Disadvantage of URL Rewriting

1. It will work only with links.

2. It can send Only textual information.

**Eg :7**

**test1.html:**

```
<Formaction="Servlet5" name = “Form1”>

        Param 1       : <inputtype="text"name="param1"/><br>

      Param 2         :
                        <inputtype="text"name="param2
                        "/><br>
                        <inputtype="submit"value="sub
                        mit"/>
</Form>
```

**Servlet5.java:**

```
public class Servlet5 extends
        HttpServlet { @Override

        Protected void doGet(HttpServletRequest
                                req,
                                HttpServletRespons
                                e res)

                        throws ServletException, IOException {
```

```
        String s1 = req.getParameter("param1");

        String s2 = req.getParameter("param2");

        PrintWriter out = res.getWriter();

        out.println("Param 1 : " + s1 + "<br>");
        out.println("Param 2 : " + s2 + "<br>");

        String newUrl = "Servlet6?param1= "+s1+"&param2="+s2;

        out.println("<Form action = '" +newUrl+"' name = 'Form1' method =
        'post'>");

        out.println("Param 3 : <input type = 'text' name = 'param3' /><BR>");
        out.println("Param 4 : <input type = 'text' name = 'param4' /><BR>");
        out.println("<input type = 'submit' value = 'submit' /> ");
        out.println("</Form> ");

    }
}
```

**Servlet6.java:**

```
public class Servlet6 extends
    HttpServlet { @Override

    Protected void doPost(HttpServletRequest
                          req,
                          HttpServletRespons
                          e res)
```

```
                throws ServletException, IOException {

            PrintWriter out = res.getWriter();

            out.println("Param 1 : " + req.getParameter("param1") + "<BR>");
            out.println("Param 2 : " + req.getParameter("param2") + "<BR>");

            out.println("Param 3 : " + req.getParameter("param3") + "<BR>");
            out.println("Param 4 : " + req.getParameter("param4") + "<BR>");
        }

}
```

**test1.html:**

```
<form action="Servlet7">
        Name
        :           <inputtype="text"name="userName"/><br/>
                    <inputtype="submit"value="Go"/>
</form>
```

**Servlet5.java:**

```
public class Servlet7 extends HttpServlet {

        public void doGet(HttpServletRequest
                          request,
                          HttpServletResponse
                          response) {

            try{
```

```
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                String user =
                request.getParameter("userName");
                out.print("Welcome "+ user);

                //appending the username in the query
                string out.print("<a
                href='Servlet8?uname="+user+"'>"

                        + "Click here to Visit</a>");

                out.close();

            }catch(Exception e){
                System.out.println(e);
            }

        }
    }
```

**Servlet6.java:**

```
public class Servlet8 extends HttpServlet {

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

```
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        //getting value from the query string

        String user =
        request.getParameter("uname");
        out.print("Hello "+ user);

        out.close();

    }
}
```

9. **Java Servlet Filter Example Tutorial**
   Java Servlet Filter is used to intercept request and do some pre-processing and can be used to intercept response and do post-processing before sending to client in web application.

**Why do we have Servlet Filter?**

We can manage session in web application and if we want to make sure that a resource is accessible only when user session is valid, we can achieve this using servlet session attributes. The approach is simple but if we have a lot of servlets and jsps, then it will become hard to maintain because of redundant code. If we want to change the attribute name in future, we will have to change all the places where we have session authentication.

That's why we have servlet filter. Servlet Filters are **pluggable** java components that we can use to intercept and process requests *before* they are sent to servlets and response *after* servlet code is finished and before container sends the response back to the client.

In the last article, we learned how we can manage session in web application and if we want to make sure that a resource is accessible only when user session is valid, we can achieve this using servlet session attributes. The approach is simple but if we have a lot of servlets and jsps, then it will become hard to maintain because of redundant code. If

we want to change the attribute name in future, we will have to change all the places where we have session authentication.

That's why we have servlet filter. Servlet Filters are **pluggable** java components that we can use to intercept and process requests *before* they are sent to servlets and response *after* servlet code is finished and before container sends the response back to the client.

Servlet Filter interface

Servlet Filter interface is similar to Servlet interface and we need to implement it to create our own servlet filter. Servlet Filter interface contains lifecycle methods of a Filter and it's managed by servlet container.

Servlet Filter interface lifecycle methods are:

- ✓ **void init(FilterConfig paramFilterConfig)** – When container initializes the Filter, this is the method that gets invoked. This method is called only once in the lifecycle of filter and we should initialize any resources in this method. **FilterConfig** is used by container to provide init parameters and servlet context object to the Filter. We can throw ServletException in this method.
- ✓ **doFilter(ServletRequest paramServletRequest, ServletResponse paramServletResponse, FilterChain paramFilterChain)** – This is the method invoked every time by container when it has to apply filter to a resource. Container provides request and response object references to filter as argument. **FilterChain** is used to invoke the next filter in the chain. This is a great example of **Chain of Responsibility Pattern**.
- ✓ **void destroy()** – When container offloads the Filter instance, it invokes the destroy() method. This is the method where we can close any resources opened by filter. This method is called only once in the lifetime of filter.

Servlet WebFilter annotation

javax.servlet.annotation.WebFilter was introduced in Servlet 3.0 and we can use this annotation to declare a servlet filter. We can use this annotation to define init parameters, filter name and description, servlets, url patterns and dispatcher types to apply the filter. If you make frequent changes to the filter configurations, its better to use web.xml because that will not require you to recompile the filter class.

Servlet Filter configuration in web.xml

We can declare a servlet filter in web.xml like below.

```
<filter>
  <filter-name>RequestLoggingFilter</filter-name> <!-- mandatory -->
  <filter-class>com.praveen.servlet.filters.RequestLoggingFilter</filter-class> <!--
mandatory -->
  <init-param> <!-- optional -->
  <param-name>test</param-name>
  <param-value>testValue</param-value>
  </init-param>
      </filter>
```

We can map a Filter to servlet classes or url-patterns like below.

```
 <filter-mapping>
  <filter-name>RequestLoggingFilter</filter-name> <!-- mandatory -->
  <url-pattern>/*</url-pattern> <!-- either url-pattern or servlet-name is mandatory -->
  <servlet-name>LoginServlet</servlet-name>
  <dispatcher>REQUEST</dispatcher>
      </filter-mapping>
```

Note: While creating the filter chain for a servlet, container first processes the url-patterns and then servlet-names, so if you have to make sure that filters are getting executed in a particular order, give extra attention while defining the filter mapping.

Servlet Filters are generally used for client requests but sometimes we want to apply filters with RequestDispatcher also, we can use dispatcher element in this case, the possible values are REQUEST, FORWARD, INCLUDE, ERROR and ASYNC. If no dispatcher is defined then it's applied only to client requests.

Servlet Filter Example for Logging and session validation

In our **servlet filter example**, we will create filters to log request cookies and parameters and validate session to all the resources except static HTMLs and LoginServlet because it will not have a session.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
 <display-name>LoginExample</display-name>
  <welcome-file-list>
    <welcome-file>loginWithURLRewriting.html</welcome-file>
  </welcome-file-list>

   <filter>
    <filter-name>RequestLoggingFilter</filter-name>
    <filter-class>com.praveen.servlet.filters.RequestLoggingFilter</filter-class>
  </filter>
  <filter>
    <filter-name>AuthenticationFilter</filter-name>
    <filter-class>com.praveen.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>
```

RequestLoggingFilter.java

```
package com.praveen.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;

@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
	}

}
```

AuthenticationFilter.java

```
package com.praveen.servlet.filters;

import java.io.IOException;

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.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebFilter("/AuthenticationFilter")
public class AuthenticationFilter implements Filter {

	private ServletContext context;

	public void init(FilterConfig fConfig) throws ServletException {
		this.context = fConfig.getServletContext();
		this.context.log("AuthenticationFilter initialized");
	}

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

		HttpServletRequest req = (HttpServletRequest) request;
		HttpServletResponse res = (HttpServletResponse) response;

		String uri = req.getRequestURI();
		this.context.log("Requested Resource::"+uri);

		HttpSession session = req.getSession(false);

		if(session == null && !(uri.endsWith("html") ||
uri.endsWith("LoginServletWithURLRewriting"))){
			this.context.log("Unauthorized access request");
			res.sendRedirect("login.html");
		}else{
```

```
                // pass the request along the filter chain
                chain.doFilter(request, response);
        }


    }



    public void destroy() {
            //close any resources here
    }

}
```

## 10. Servlet Listeners

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.

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, and so on and so forth.

The Servlet API provides a number of listener interfaces we can implement in order to react to these events. There are total eight type of listeners available in servlet framework which listens to a particular event and they are –

1. ServletRequestListener
2. ServletRequestAttributeListener
3. ServletContextListener
4. ServletContextAttributeListener
5. HttpSessionListener
6. HttpSessionAttributeListener
7. HttpSessionActivationListener
8. HttpSessionBindingListener

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

Syntax: <listener>
<listener-class></listerner-class>
<
/listener> Note –
Listeners are
neither a Servlets
nor a JSP.

ServletRequestListener

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

What is the need?

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 can we achieve it?

Java EE specification provides an interface called ServletRequestListener which receives notifications whenever a new request is about to come to the web application. This is what javadoc says about it.

A ServletRequest is defined as coming into scope of a web application when it is about to enter the first servlet or filter of the web application, and as going out of scope as it exits the last servlet or the first filter in the chain.

requestInitialized (ServletRequestEvent event): Receives notification that a ServletRequest is about to come into scope of the web application.

requestDestroyed (ServletRequestEvent event) : Receives notification that a ServletRequest is about to go out of scope of the web application.

Example:

----------MyServletRequestListener.java----------

```
package com.ashoksoft.apps;

import javax.servlet.ServletRequest;

import javax.servlet.ServletRequestEvent; import
javax.servlet.ServletRequestListener;

public class MyServletRequestListener implements
ServletRequestListener {

  /**

   * Default constructor.

   */

  public MyServletRequestListener() {

  }

  public void requestDestroyed(ServletRequestEvent
  servletRequestEvent)

  {

    ServletRequest request =
    servletRequestEvent.getServletRe
    quest();
    System.out.println("Request
    Destroyed");

  }

  public void requestInitialized(ServletRequestEvent
  servletRequestEvent)

  {

    ServletRequest request =
    servletRequestEvent.getServletRe
    quest();
    System.out.println("Request
    initialized");

  }

}
```

----------web.xml----------

```
<listener>
<description> Servlet Request Listener
Example</description>

<listener-class>
com.ashoksoft.apps.MyServletRequestListener</listener-
class>

</listener>
```

---

**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

**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.

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 example that I can think of is initializing database connections and clean up of database connections.

**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.

**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.

**HttpSessionAttributeListener**
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.