Servlets and Java Server Pages

Prepared By: Aakash Raj Shakya

Servlets

Servlets are small programs that execute on the server side.

Servlets provide a component-based, platform-independent method for building web based applications, without the performance limitations of Computer Gateway Interface(**CGI**) programs.

Servlet is a technology which is used to create a web application.

Servlet is a web component that is deployed on the server to create a dynamic web page.

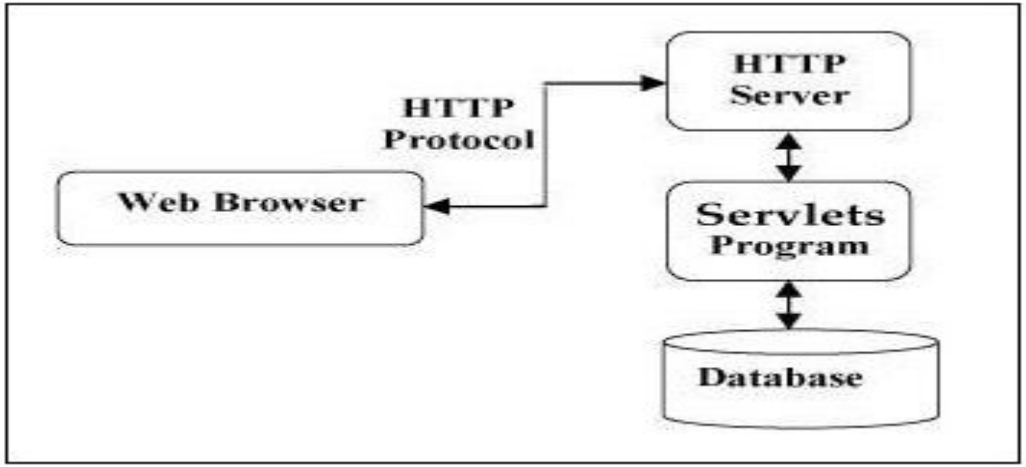
A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. (**Oracle**)

Why to Learn Servlet?

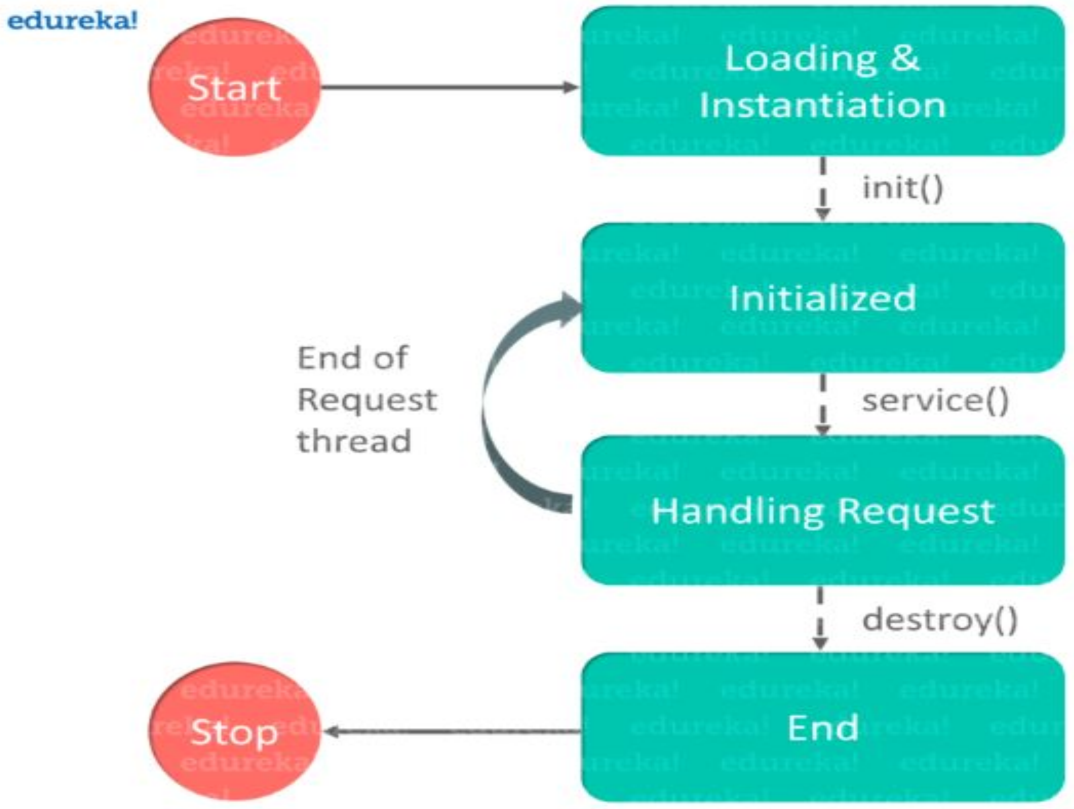
Using servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically. Some of the key important reasons are:

1. Performance is significantly better.
2. Servlets execute within the address space of a **Web server**. It is not necessary to create a separate process to handle each client request.
3. Servlets are platform-independent because they are written in Java.
4. Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
5. The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms.

Servlet Architecture



Life Cycle of Servlet



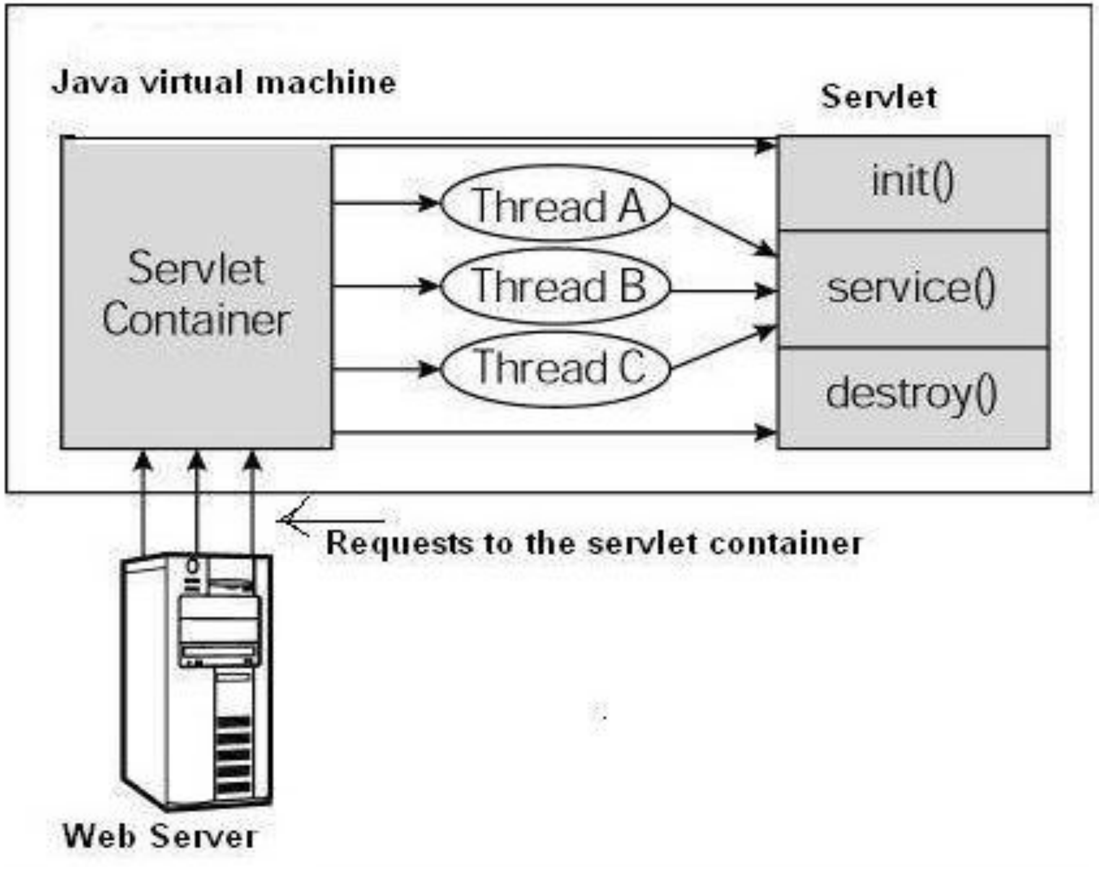
Life Cycle of Servlet contd..

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

1. The servlet is initialized by calling the **init()** method.
2. The servlet calls **service()** method to process a client's request.
3. The servlet is terminated by calling the **destroy()** method.
4. Finally, servlet is garbage collected by the garbage collector of the JVM.

https://www.tutorialspoint.com/servlets/servlets-life-cycle.htm

Architecture Diagram



Servlet Packages

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the **javax.servlet** and **javax.servlet.http packages**, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

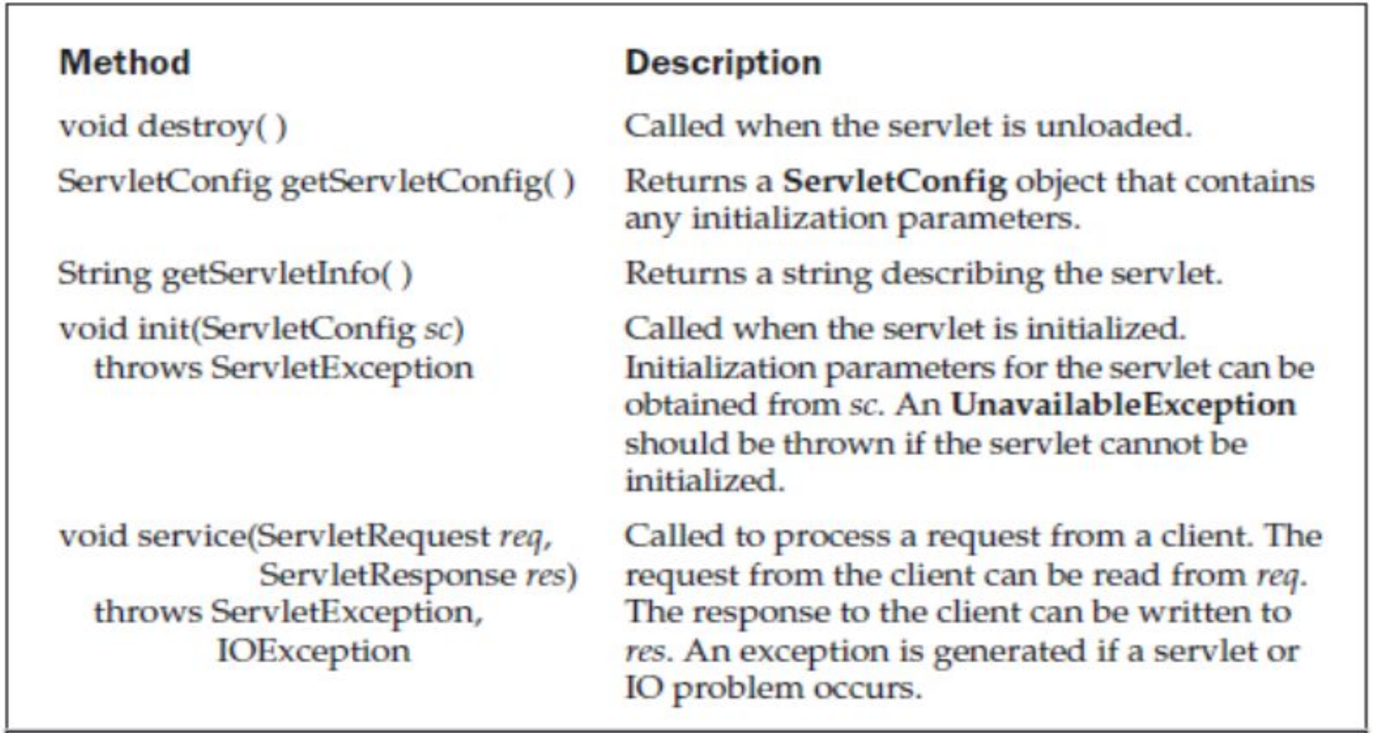
These classes implement the Java Servlet and JSP specifications

The following table summarizes the core interfaces that are provided in this package. The most significant of these is Servlet. All servlets must implement this interface or extend a class that implements the interface.

The **ServletRequest** and **ServletResponse** interfaces are also very important.

Interface	**Description**
Servlet	Declares life cycle methods for a servlet.
ServletConfig	Allows servlets to get initialization parameters.
ServletContext	Enables servlets to log events and access information about their environment.
ServletRequest	Used to read data from a client request.
ServletResponse	Used to write data to a client response.
SingleThreadModel	Indicates that the servlet is thread safe.

The Servlet Interface



Method	Description
void destroy( )	Called when the servlet is unloaded.
ServletConfig getServletConfig( )	Returns a **ServletConfig** object that contains any initialization parameters.
String getServletInfo( )	Returns a string describing the servlet.
void init(ServletConfig *sc*) throws ServletException	Called when the servlet is initialized. Initialization parameters for the servlet can be obtained from *sc*. An **UnavailableException** should be thrown if the servlet cannot be initialized.
void service(ServletRequest *req*, ServletResponse *res*) throws ServletException, IOException	Called to process a request from a client. The request from the client can be read from *req*. The response to the client can be written to *res*. An exception is generated if a servlet or IO problem occurs.

Table 27-1. *The Methods Defined by* Servlet

Compiling and running servlet

```java
import javax.servlet.*;
import java.io.IOException;

public class DemoServlet implements Servlet {
  private ServletConfig config = null;
  @Override
  public void init(ServletConfig servletConfig) throws ServletException {
     System.out.println("Inside init servlet.");
     this.config = servletConfig;
  }

  @Override
  public ServletConfig getServletConfig() {
     return null;
  }

  @Override
  public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws ServletException, IOException {
     System.out.println("Servlet service");
  }

  @Override
  public String getServletInfo() {
     return null;
  }

  @Override
  public void destroy() {
     System.out.println("Destroying the servlet.");
  }
}
```

Respective **web.xml** file for **DemoServlet** class

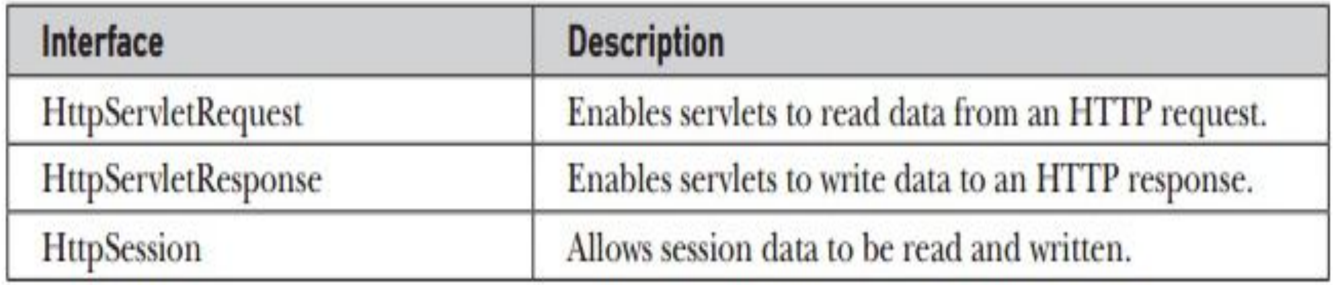
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
 <servlet>
   <servlet-name>DemoServlet</servlet-name>
   <servlet-class>
     DemoServlet
   </servlet-class>
 </servlet>
 <servlet-mapping>
   <servlet-name>DemoServlet</servlet-name>
   <url-pattern>/test</url-pattern>
 </servlet-mapping>
</web-app>
```

The javax.servlet.http Package

The preceding examples have used the classes and interfaces defined in **javax.servlet**, such as **ServletRequest**, **ServletResponse**, **Servlet,** and **GenericServlet** to illustrate the basic functionality of servlets.

However, when working with HTTP, you will normally use the interfaces and classes in **javax.servlet.http**. As you will see, its functionality makes it easy to build servlets that work with HTTP requests and responses.

The following table summarizes the interfaces used in this package:



Interface	Description
HttpServletRequest	Enables servlets to read data from an HTTP request.
HttpServletResponse	Enables servlets to write data to an HTTP response.
HttpSession	Allows session data to be read and written.

Reading the servlet Parameters

Servlets handles form data parsing automatically using the following methods depending on the situation

1. **getParameter()** − You call request.getParameter() method to get the value of a form parameter.
2. **getParameterValues()** − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
3. **getParameterNames()** − Call this method if you want a complete list of all parameters in the current request.

We can handle **GET** and **POST** request using these parameters.

GET Method Example

For an example we have a following URL
http://localhost:33974/School_Management/getRequest?<u>name</u>=Aakash&<u>age</u>=29&<u>hobbies</u>=Swimming,Basketball,Tennis,Dance

Here getRequest is the mapped url where the **GET HTTP** request is processed.

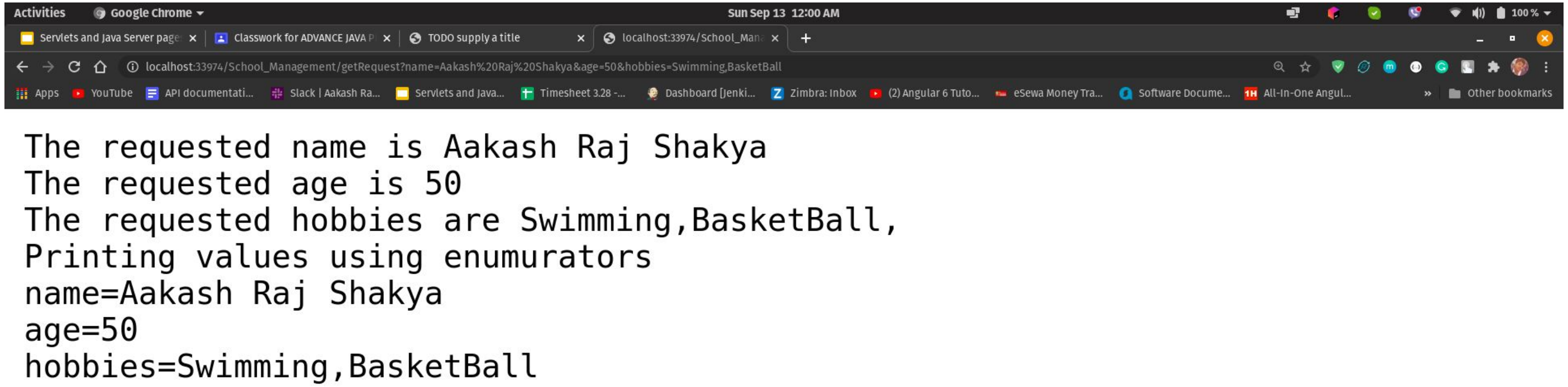
We have sent two parameters: **name** and **age** whose values are “Aakash” and “29” respectively.

Following is a basic example of servlet to process **GET** request:

```java
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

public class ServletGetParameterExample extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
     PrintWriter writer = response.getWriter();
     writer.println("The requested name is " + request.getParameter("name"));
     writer.println("The requested age is " + request.getParameter("age"));
     String[] hobbies = request.getParameterValues("hobbies");
     String hobbiesList = "";
     for(String hobby : hobbies) {
        hobbiesList += hobby + ",";
     }
     writer.println("The requested hobbies are " + hobbiesList);

     writer.println("Printing values using enumerators");

     Enumeration<String> parameters = request.getParameterNames();
     while(parameters.hasMoreElements()) {
        String nextElement = parameters.nextElement();
        writer.print(nextElement + "=");
        String value = request.getParameter(nextElement);
        writer.println(value);
     }
  }
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
   <servlet>
       <servlet-name>ServletGetParameterExample</servlet-name>
       <servlet-class>com.schoolmanagement.readingparameters.ServletGetParameterExample</servlet-class>
   </servlet>
   <servlet-mapping>
       <servlet-name>ServletGetParameterExample</servlet-name>
       <url-pattern>/getRequest</url-pattern>
   </servlet-mapping>
</web-app>
```

Output:



```
The requested name is Aakash Raj Shakya
The requested age is 50
The requested hobbies are Swimming,BasketBall,
Printing values using enumurators
name=Aakash Raj Shakya
age=50
hobbies=Swimming,BasketBall
```

POST Method Example:

```java
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

public class ServletPostParameterExample extends HttpServlet {
  @Override
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {
     PrintWriter writer = response.getWriter();
     writer.println("The posted value for name is " + request.getParameter("name"));
     writer.println("The posted value for age is " + request.getParameter("age"));
  }
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <servlet>
     <servlet-name>ServletPostParameterExample</servlet-name>
     <servlet-class>com.schoolmanagement.readingparameters.ServletPostParameterExample</servlet-class>
  </servlet>
  <servlet-mapping>
     <servlet-name>ServletPostParameterExample</servlet-name>
     <url-pattern>/postRequest</url-pattern>
  </servlet-mapping>
</web-app>
```

```html
<html>
        <head>
          <title>Servlet Form Post Example</title>
        </head>
        <body>
                <h1>Servlet POST example</h1>
                <div>
                  <form method="POST" action="http://localhost:33974/School_Management/postRequest">
                     Enter Name: <input type="text" name="name" /> <br />
                     Enter Age: <input type="text" name="age" /> <br />
                     <input type="submit" />
                  </form>
                </div>
        </body>
</html>
```

Reading Initialization Parameter

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

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

DemoServlet.java

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DemoServlet extends HttpServlet {
   @Overrride
   public void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {

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

      ServletConfig config=getServletConfig();
      String driver=config.getInitParameter("driver");
      out.print("Driver is: "+driver);

      out.close();
   }

}
```

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

 <servlet>
   <servlet-name>DemoServlet</servlet-name>
   <servlet-class>DemoServlet</servlet-class>

   <init-param>
     <param-name>driver</param-name>
     <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
   </init-param>

 </servlet>

 <servlet-mapping>
   <servlet-name>DemoServlet</servlet-name>
   <url-pattern>/demo</url-pattern>
 </servlet-mapping>

</web-app>
```

Cookies in Servlet

Cookies are text files stored on the client computer and they are kept for various information tracking purpose.

A cookie is a small piece of information that is persisted between the multiple client requests.

Java Servlets transparently supports **HTTP cookies**.

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

For more info:
https://www.javatpoint.com/cookies-in-servlet

Constructor of Cookie class:

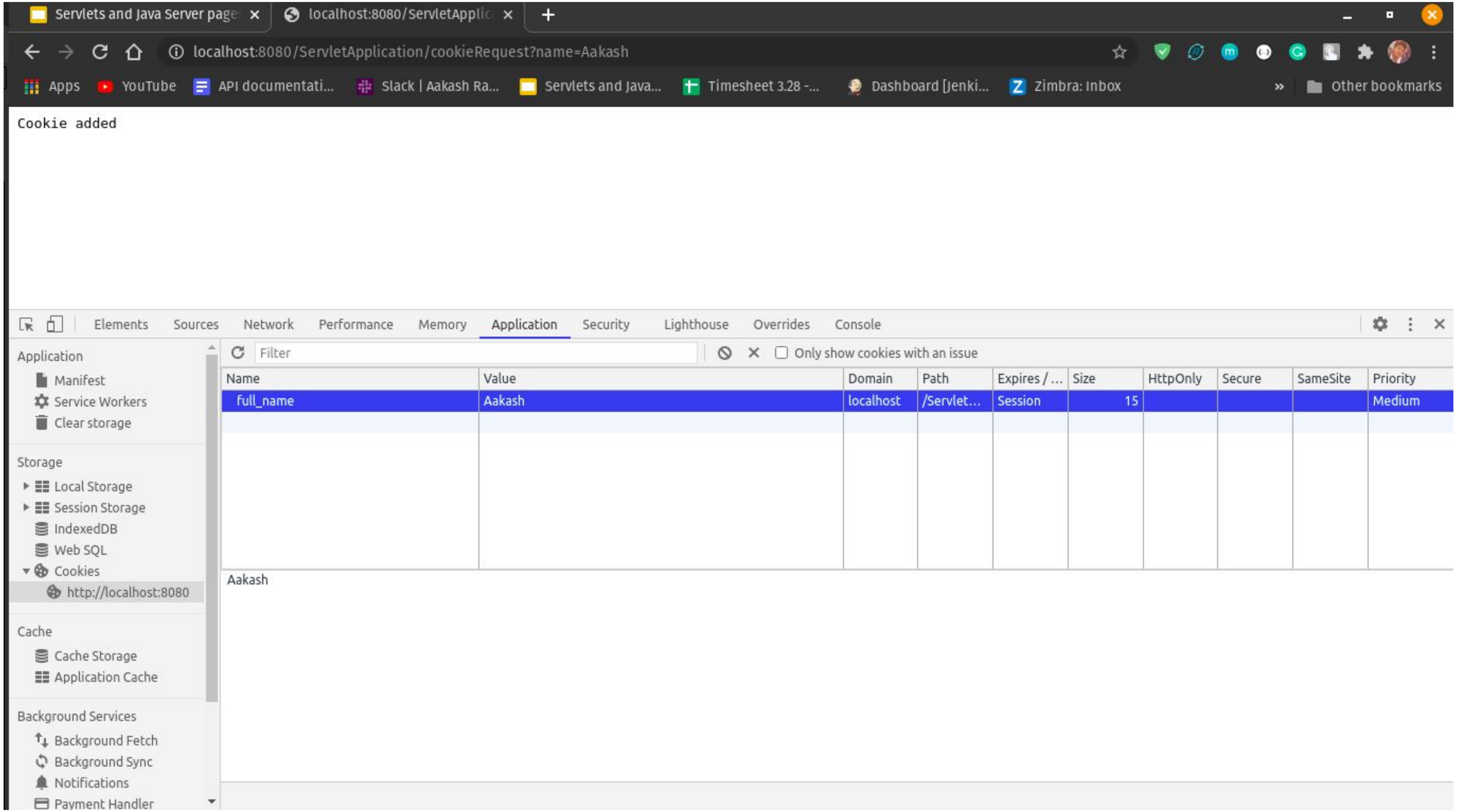
Constructor	**Description**
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Servlet Cookies Methods:
Following is the list of useful methods which you can use while manipulating cookies in servlet.

Method	**Description**
public void **setMaxAge**(int **expiry**)	Sets the maximum age of the cookie in seconds.
public String **getName**()	Returns the name of the cookie. The name cannot be changed after creation.
public String **getValue**()	Returns the value of the cookie.
public void **setName**(String name)	changes the name of the cookie.
public void **setValue**(String value)	changes the value of the cookie.

Cookie Example:

```
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;

public class ServletCookieExample extends HttpServlet {
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException {
       String name = request.getParameter("name");
       Cookie nameCookie = new Cookie("full_name", name);
       nameCookie.setMaxAge(0);
       response.addCookie(nameCookie);
       PrintWriter writer = response.getWriter();
       writer.print("Cookie added");
   }
}
```



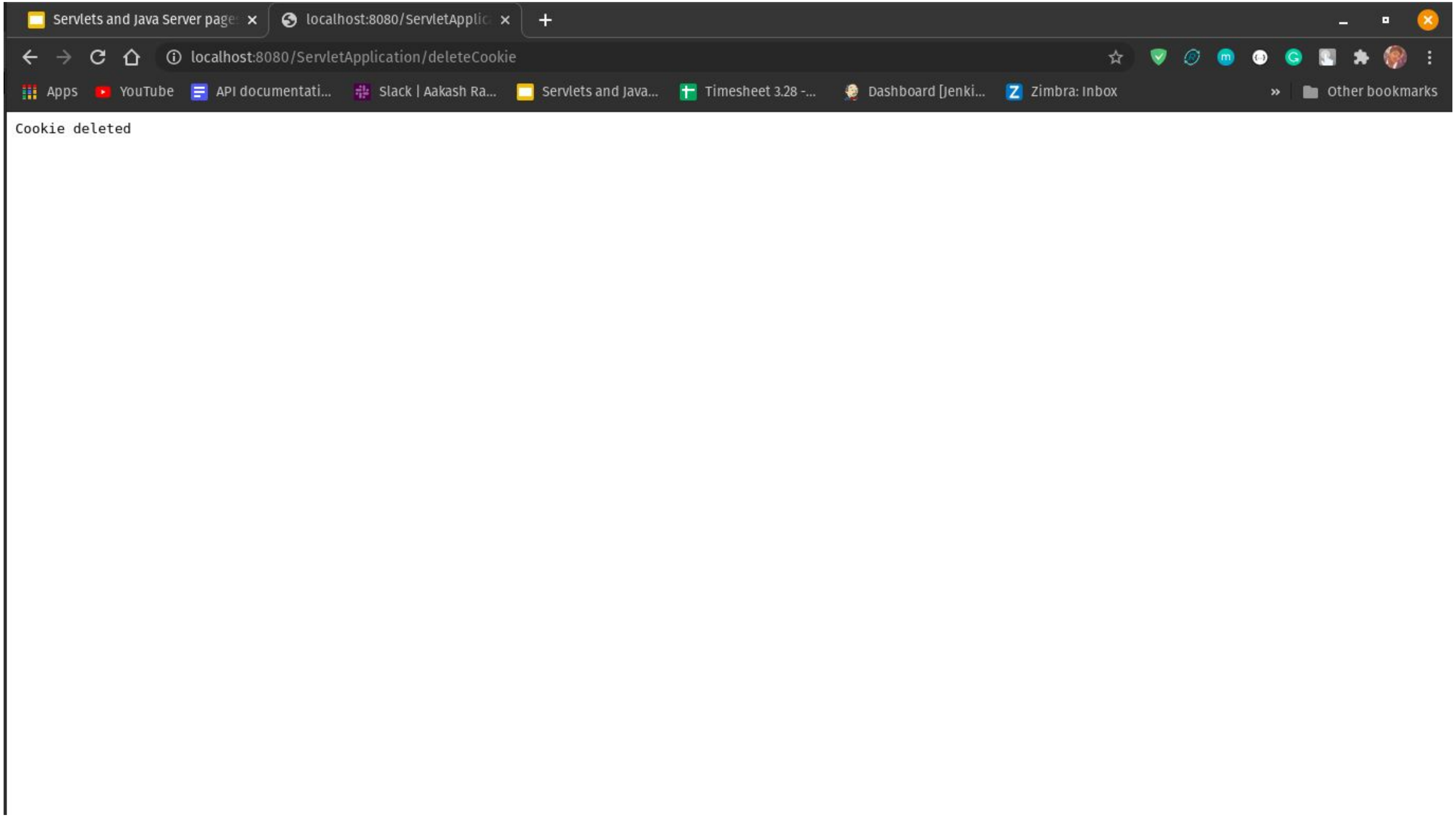
```
Cookie added
```

Name	Value	Domain	Path	Expires / ...	Size	HttpOnly	Secure	SameSite	Priority
full_name	Aakash	localhost	/Servlet...	Session	15				Medium

Aakash

Deleting the cookie

```
import java.io.IOException;
import javax.servlet.http.*;

public class DeleteCookieExample extends HttpServlet {

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException {
       Cookie[] cookies = request.getCookies();
       if(cookies != null) {
           for(Cookie cookie : cookies) {
               if(cookie.getName().equals("full name")) {
                   response.getWriter().print("Cookie deleted");
                   cookie.setMaxAge(0);
                   response.addCookie(cookie);
               }
           }
       }
   }
}
```



```
Cookie deleted
```

Session Tracking

HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

Servlet provides **HttpSession Interface** which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

You would get **HttpSession** object by calling the public method **getSession**() of **HttpServletRequest**, as below:

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

Method	**Description**
public Object **getAttribute**(String name)	This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
public Enumeration **getAttributeNames**()	This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
public long **getCreationTime**()	This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
public String **getId**()	This method returns a string containing the unique identifier assigned to this session.
public long **getLastAccessedTime**()	This method returns the last accessed time of the session, in the format of milliseconds since midnight January 1, 1970 GMT
public int **getMaxInactiveInterval**()	This method returns the maximum time interval (seconds), that the servlet container will keep the session open between client accesses.
public void **invalidate**()	This method invalidates this session and unbinds any objects bound to it.
public boolean **isNew**()	This method returns true if the client does not yet know about the session or if the client chooses not to join the session.

public void **removeAttribute**(String name)	This method removes the object bound with the specified name from this session.
public void **setAttribute**(String name, Object value)	This method binds an object to this session, using the name specified.
public void **setMaxInactiveInterval**(int interval)	This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

Creating Session Example:

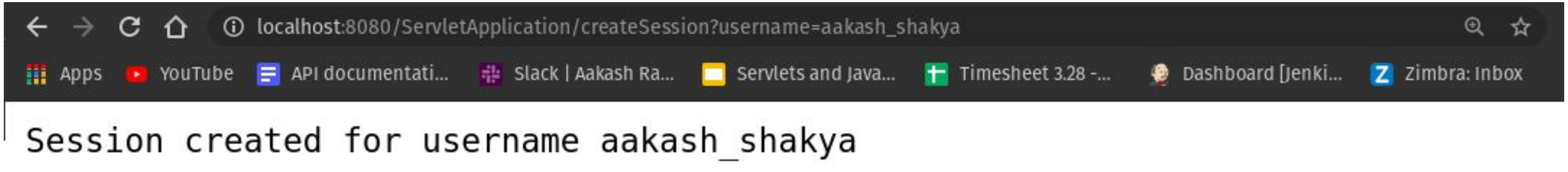
CreateSessionExample.java

```java
import java.io.*;
import javax.servlet.http. *;

public class CreateSessionExample extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
       HttpSession session = request.getSession();
       session.setAttribute("username", request.getParameter("username"));
       PrintWriter writer = response.getWriter();
       writer.println("Session created for username " + session.getAttribute("username"));
   }
}
```

web.xml

```xml
<web-app>
   <servlet>
       <servlet-name>CreateSessionExample</servlet-name>
       <servlet-class>CreateSessionExample</servlet-class>
   </servlet>
   <servlet-mapping>
       <servlet-name>CreateSessionExample</servlet-name>
       <url-pattern>/createSession</url-pattern>
   </servlet-mapping>
</web-app>
```

Output



```
Session created for username aakash_shakya
```

Retrieving and Removing Session Example:

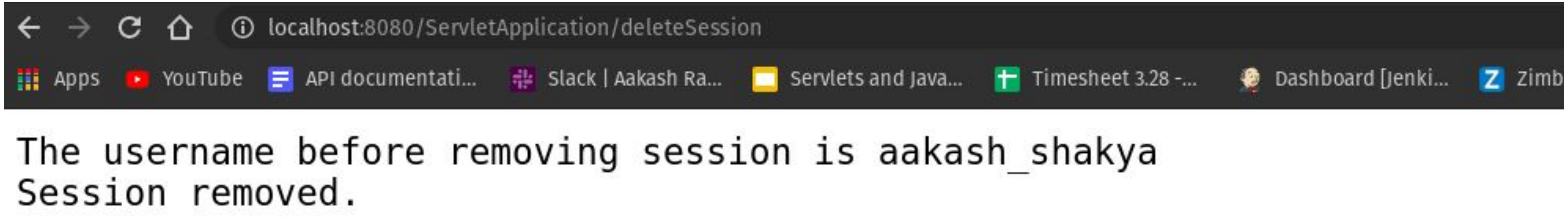
DeleteSessionExample.java

```java
import java.io.*;
import javax.servlet.http.*;

public class DeleteSessionExample extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
       HttpSession session = request.getSession();
       PrintWriter writer = response.getWriter();
       if (session.getAttribute("username") != null) {
           //Retrieving information
           writer.println("The username before removing session is" + session.getAttribute("username"));
       }
       session.invalidate();    //Removes all the session information
       writer.println("Session removed.");
   }
}
```

web.xml

```xml
<web-app>
   <servlet>
       <servlet-name>DeleteSessionExample</servlet-name>
       <servlet-class>DeleteSessionExample</servlet-class>
   </servlet>
   <servlet-mapping>
       <servlet-name>DeleteSessionExample</servlet-name>
       <url-pattern>/createSession</url-pattern>
   </servlet-mapping>
</web-app>
```

Output



localhost:8080/ServletApplication/deleteSession

```
The username before removing session is aakash_shakya
Session removed.
```

Java Server Pages

Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications.

It is a web-based powerful technology in which an html page has Java code embedded to it.

It is one of the powerful technology which is implemented on server side to provide dynamic as well as static content to a client browser.

The dynamic content is been generated by Java code embedded into the web-page.

The extension of JSP files is **.jsp**

Advantages of JSP over Active Server Pages(ASP)

Following are the advantages of JSP over ASP:

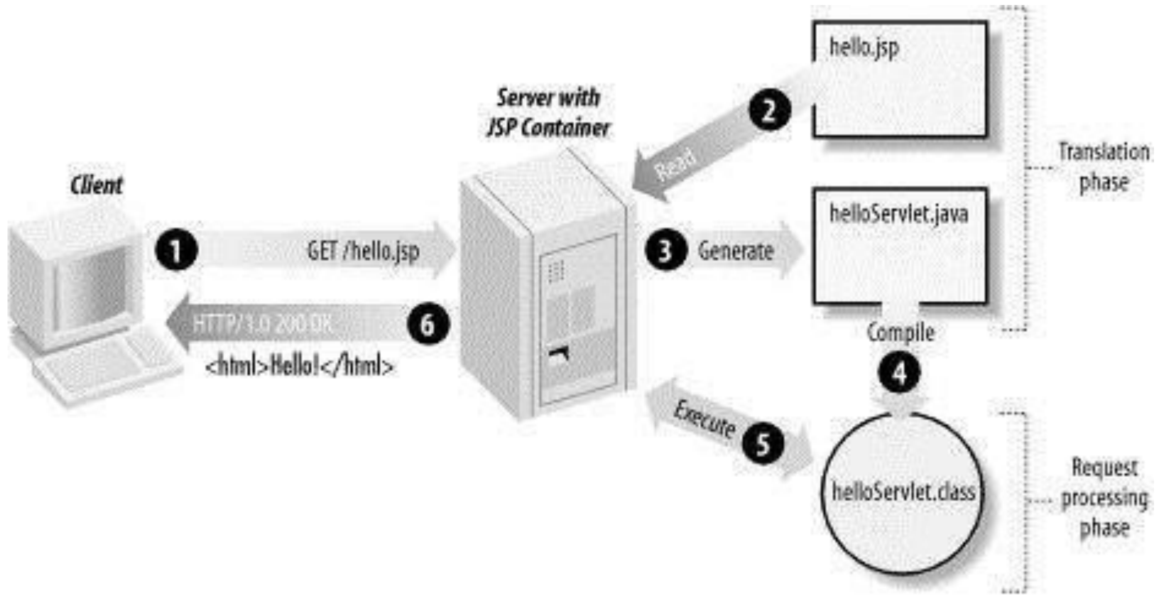
1. **Cost:** JSP is free to use whereas ASP is paid.
2. **Platform Independence:** JSP being Java based is platform independent whereas ASP is platform dependent.
3. **Memory leak Protection:** JSP has inbuilt memory leak protection whereas ASP lacks inbuilt memory leak protection.
4. **Security:** JSP provides better inbuilt security mechanism whereas ASP lacks an inbuilt security mechanism.

Advantages of JSP over Servlet

1. Servlets use println statements for printing an HTML document which is usually very difficult to use. JSP has no such tedious task to maintain.
2. JSP needs no compilation, CLASSPATH setting and packaging.
3. In a JSP page visual content and logic are seperated, which is not possible in a servlet.
4. There is automatic deployment of a JSP, recompilation is done automatically when changes are made to JSP pages.
5. Usually with JSP, Java Beans and custom tags web application is simplified.

Note: Servlets are much faster(performance) that JSP.

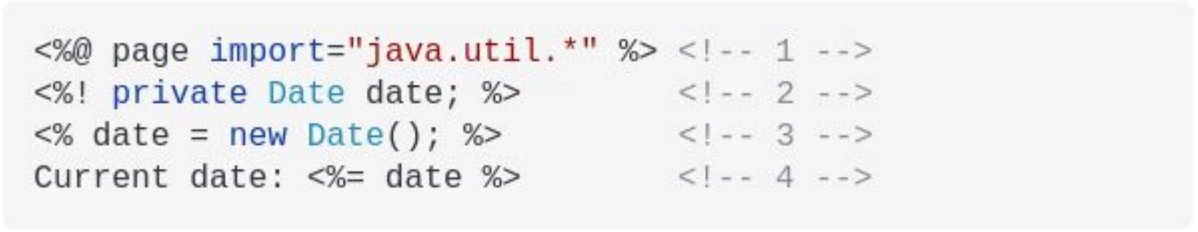
JSP Architecture



The following steps explain how the web server creates the Webpage using JSP:

1. As with a normal page, your browser sends an HTTP request to the web server.
2. The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.
3. The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.
4. The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
5. A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is further passed on to the web server by the servlet engine inside an HTTP response.
6. The web server forwards the HTTP response to your browser in terms of static HTML content.
7. Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

The JSP code



```
<%@ page import="java.util.*" %> <!-- 1 -->
<%! private Date date; %>        <!-- 2 -->
<% date = new Date(); %>         <!-- 3 -->
Current date: <%= date %>        <!-- 4 -->
```

```
import java.util.*; // 1

public class ServletAbc extends GenericServlet {

   private Date date; // 2

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

        PrintWriter out=response.getWriter();

        date = new Date(); // 3

        out.println("Current date: "); // 4
        out.println(date);
   }
}
```

JSP Access Model

In the design of Java Web applications, there are two commonly used design models, referred to as **Model 1** and **Model 2.**

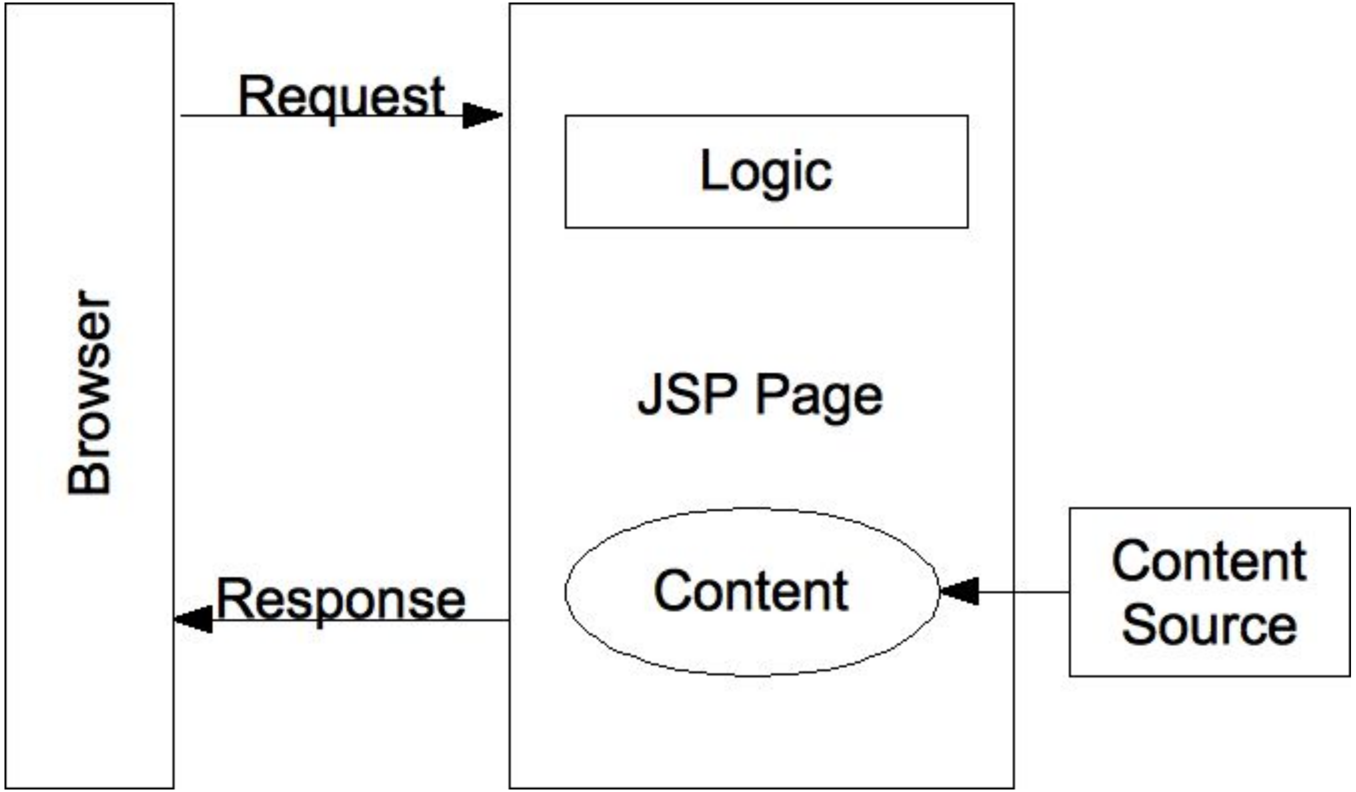
Model 1 is also known as JSP Centric architecture

Model 2 is also known as Servlet-Centric Architecture

https://www.geeksforgeeks.org/jsp-access-model/

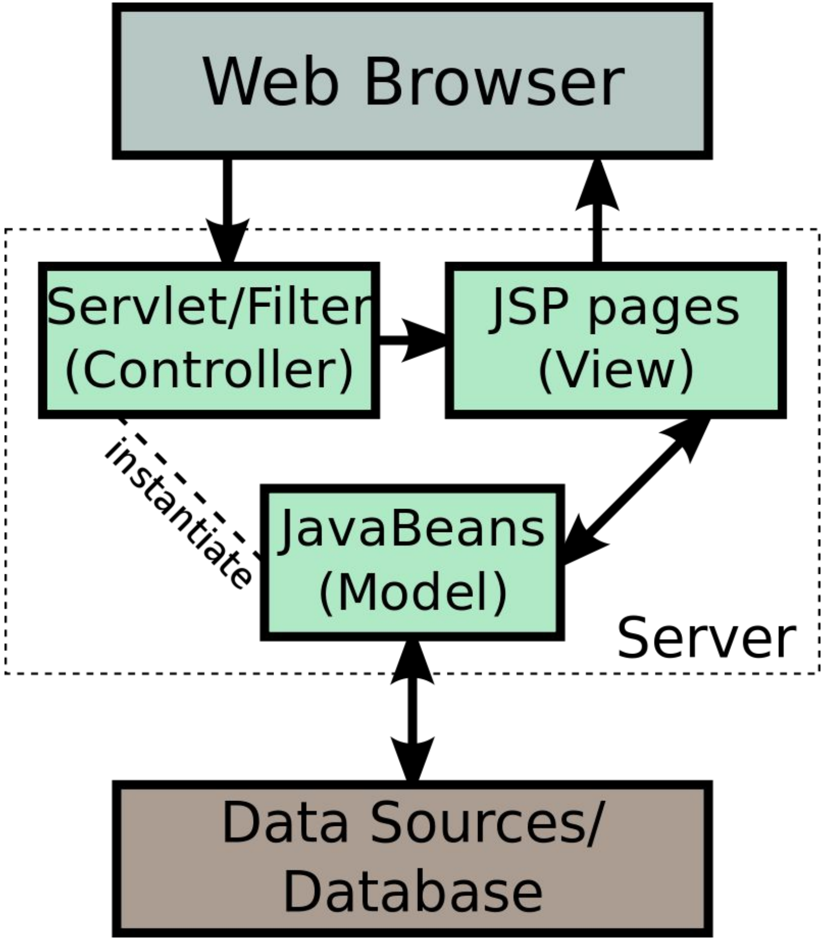
JSP Access Model: Model 1 Architecture

The incoming request is directly sent to the JSP page from a web browser and JSP page is responsible for processing it and sending back to the client. All the data access is performed using beans so there is still a separation of presentation from content.


Browser
Request
Logic
JSP Page
Response
Content
Content Source

JSP Access Model: Model 2 Architecture

It is basically a Model View Controller approach involved notification/event models, direct manipulation of model objects. MVC basically interposes controller component between View and Model Components where the controller is responsible for navigation, presentation-tier logic, validation and emphasizes the separation of presentation logic and model objects.


Web Browser
Servlet/Filter (Controller)
JSP pages (View)
instantiate
JavaBeans (Model)
Server
Data Sources/ Database

JSP Implicit Object

JSP Implicit Objects are the Java objects that the **JSP Container** makes available to the developers in each page and the developer can call them directly without being explicitly declared.

JSP Implicit Objects are also called **pre-defined variables.**

https://www.tutorialspoint.com/jsp/jsp_implicit_objects.htm

Following are the nine Implicit Objects that JSP supports:

Objects	**Description**
request	This is the **HttpServletRequest** object associated with the request.
response	This is the **HttpServletResponse** object associated with the response to the client.
out	This is the **PrintWriter** object used to send output to the client.
session	This is the **HttpSession** object associated with the request.
application	This is the **ServletContext** object associated with the application context.
config	This is the **ServletConfig** object associated with the page.
pageContext	This encapsulates use of server-specific features like higher performance **JspWriters**.
page	This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.
exception	The **Exception** object allows the exception data to be accessed by designated JSP.

Scope of JSP Objects

The availability of a JSP object for use from a particular place of the application is defined as the scope of that JSP object. Every object created in a JSP page will have a scope. Object scope in JSP is segregated into four parts and they are:

1. Page Scope -> out, exception, response, pageContext, config and page
2. Request Scope -> request
3. Session Scope -> session
4. Application Scope -> application

https://javapapers.com/jsp/explain-the-scope-of-jsp-objects/

A basic jsp file.

```
<!DOCTYPE html>
<html>
   <head>
       <title>JSP Page</title>
   </head>
   <body>
       <h1>Hello World!</h1>
       <p>
           <%
               out.println("Welcome this is first file of JSP");
           %>
       </p>
   </body>
</html>
```

JSP - Syntax

Elements of JSP:

1. The Scriptlet **<% code fragment %>**
2. JSP Declarations **<%! declaration; [declaration;]+ ... %>**
3. JSP Expression **<%= expression %>**
4. JSP Comments **<%-- This is JSP comment --%>**
5. JSP Directives **<%@ directive attribute="value" %>**
6. JSP Actions **<jsp:action_name attribute="value" />**

Exception Handling in JSP

The implicit object **exception** is used to handle the exception in JSP.

It is an object of **java.lang.Throwable** class, and is used to print exceptions. However, it can only be used in error pages.

Handling Exception using page directive attributes

The page directive in JSP provides two attributes to be used in exception handling. They’re:

errorPage: Used to site which page to be displayed when exception occurred.

Syntax : **<%@page errorPage="url of the error page"%>**

isErrorPage : Used to mark a page as an error page where exceptions are displayed.

Syntax : **<%@page isErrorPage="true"%>**

Example:
exception_index.jsp

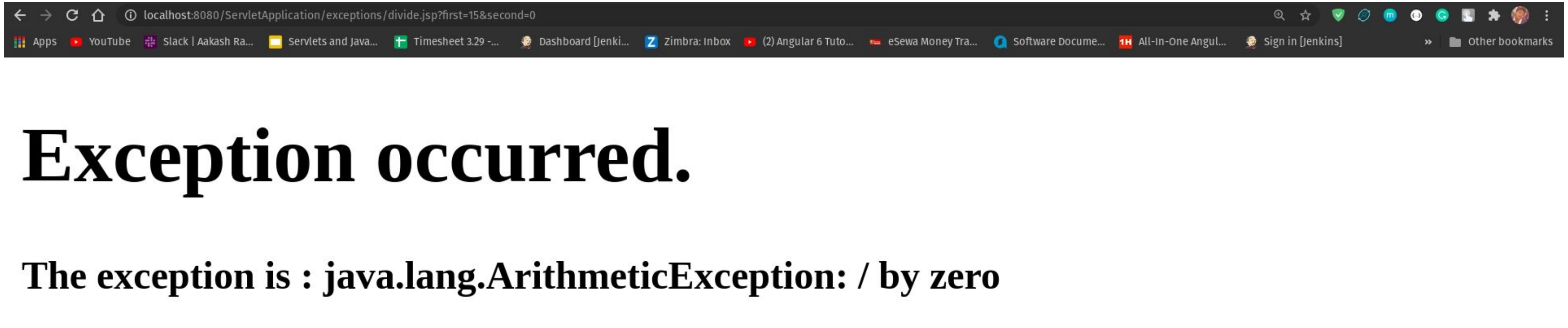
```
<html>
   <head>
   <body>
       <form action="divide.jsp">
           Number1:<input type="text" name="first" >
           Number2:<input type="text" name="second" >
           <input type="submit" value="divide">
       </form>
   </body>
</html>
```

divide.jsp

```
<%@page errorPage="error.jsp" %>
<%
   int firstNumber = Integer.parseInt(request.getParameter("first"));
   int secondNumber = Integer.parseInt(request.getParameter("second"));
   out.println("Division of two numbers is " + (firstNumber / secondNumber));
%>
```

error.jsp

```
<%@page isErrorPage = "true" %>
<!DOCTYPE html>
<html>
   <head>
       <title>JSP Page</title>
   </head>
   <body>
       <h1>Exception occurred.</h1>
       <h4>The exception is : <%= exception %></h4>
   </body>
</html>
```



localhost:8080/ServletApplication/exceptions/divide.jsp?first=15&second=0

Apps YouTube Slack | Aakash Ra... Servlets and Java... Timesheet 3.29 -... Dashboard [Jenki... Zimbra: Inbox (2) Angular 6 Tuto... eSewa Money Tra... Software Docume... All-In-One Angul... Sign in [Jenkins] Other bookmarks

Exception occurred.

The exception is : java.lang.ArithmeticException: / by zero

Session Management

Session in JSP is similar to session used in servlet.

We use the implicit object of **session** which is an object of **javax.servlet.http.HttpSession** interface.

Example:
session_example.jsp

```
<!DOCTYPE html>
<html>
   <head>
       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" >
       <title>Session example page</ title>
   </head>
   <body>
       <form method="POST" action="<%= application.getContextPath() %>/session/check_session.jsp" >
           Enter username: < input type="text" name="username" /> <br>
           Enter password: < input type="password" name="password" /> <br>
           <input type="submit" /> <br>
       </ form>
   </body>
</html>
```

check_session.jsp

```
<!DOCTYPE html>
<html>
   <head>
       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
       <title>Check Session Page</title>
   </head>
   <body>
       <%!
           String stackUsername = "admin";
           String stackPassword = "password";
       %>
       <%
           String username = request.getParameter("username");
           String password = request.getParameter("password");
           String sessionUsername = (String) session.getAttribute("username");
           if (sessionUsername == null) {
               if (username != null && username.equals(stackUsername) &&
                       password != null && password.equals(stackPassword)) {
                   session.setAttribute("username", username);
                   sessionUsername =username;
               } else {
                   response.sendRedirect(application.getContextPath() + "/session/session_example.jsp");
               }
           }
       %>
       <h1>Welcome, You are logged in as<%=sessionUsername%></h1>
   </body>
</html>
```