Java Server Page

JSP Basics

- **JSP** technology is used to create dynamic web applications.
- **JSP** pages are easier to maintain then a **Servlet**.
- JSP pages are opposite of Servlets.
- Servlet adds HTML code inside Java code.
- JSP adds Java code inside HTML using JSP tags.
- Everything a Servlet can do, a JSP page can also do it.

JSP Basics

- JSP enables us to write HTML pages containing tags, inside which we can include powerful Java programs.
- **Using JSP, can easily separate Presentation and Business logic.**
- Web designer can design and update JSP pages creating the presentation layer and java developer can write server side complex computational code without concerning the web design.
- And both the layers can easily interact over HTTP requests.

Advantage of JSP over Servlet

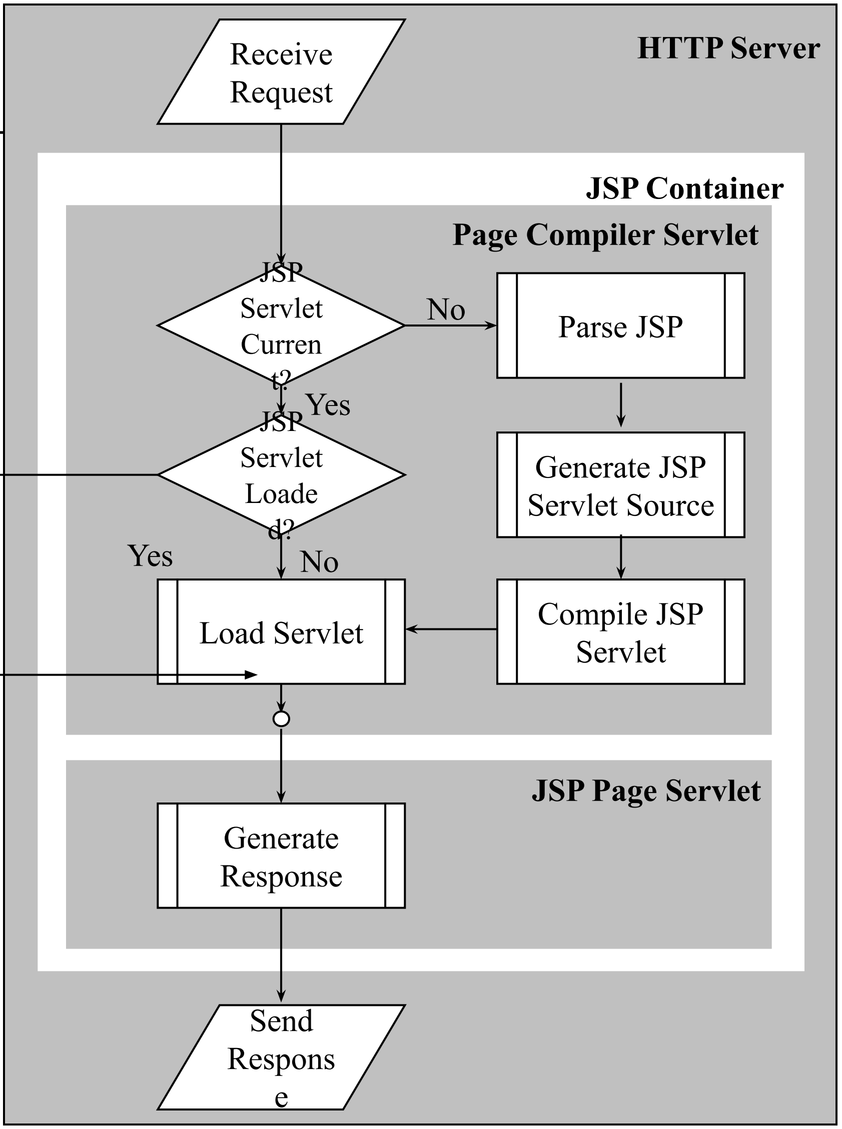
- **1) Extension to Servlet**
- JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
- **2) Easy to maintain**
- JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

Advantage of JSP over Servlet

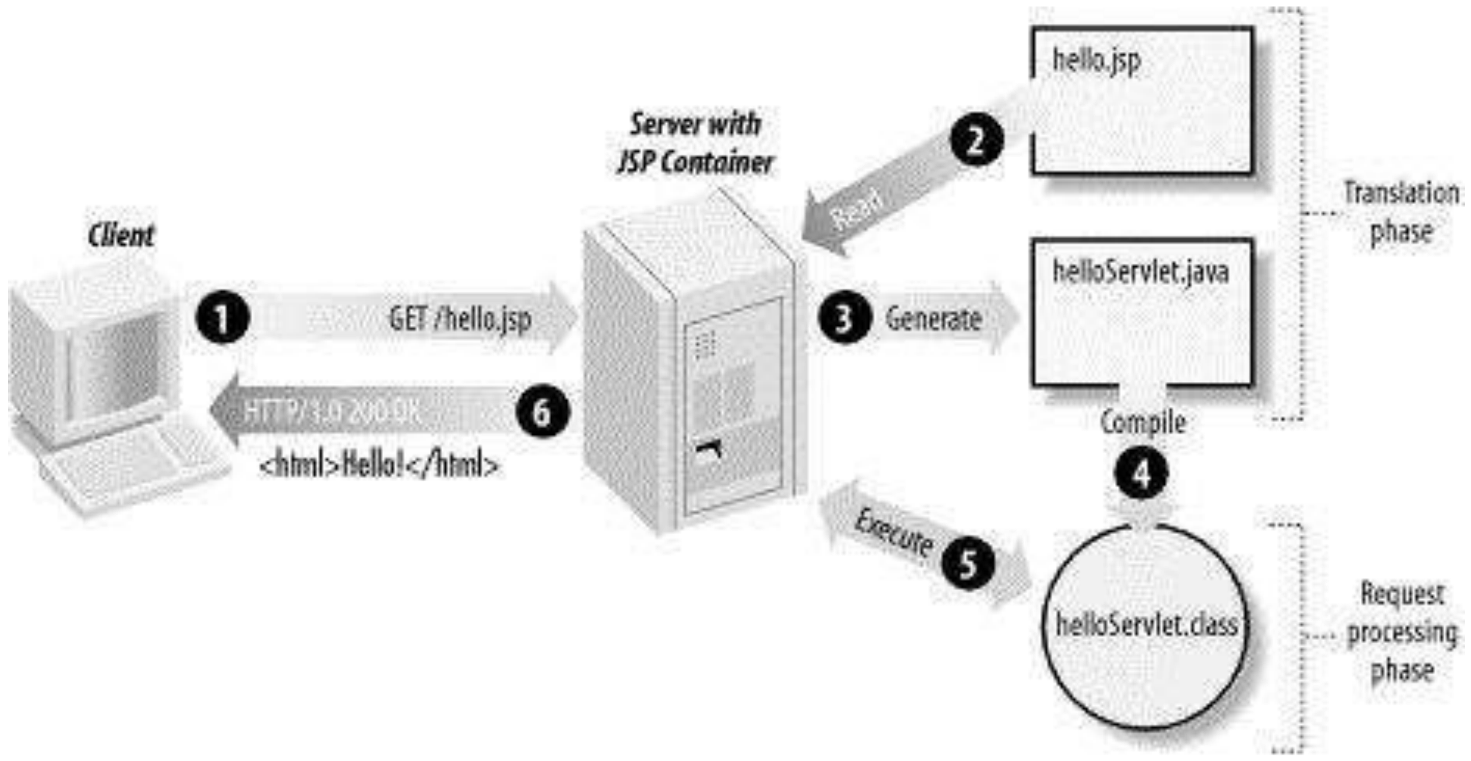
- **3) Fast Development: No need to recompile and redeploy**
- If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
- **4) Less code than Servlet**
- In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use implicit objects etc.

Java Server Pages (JSP) Architecture

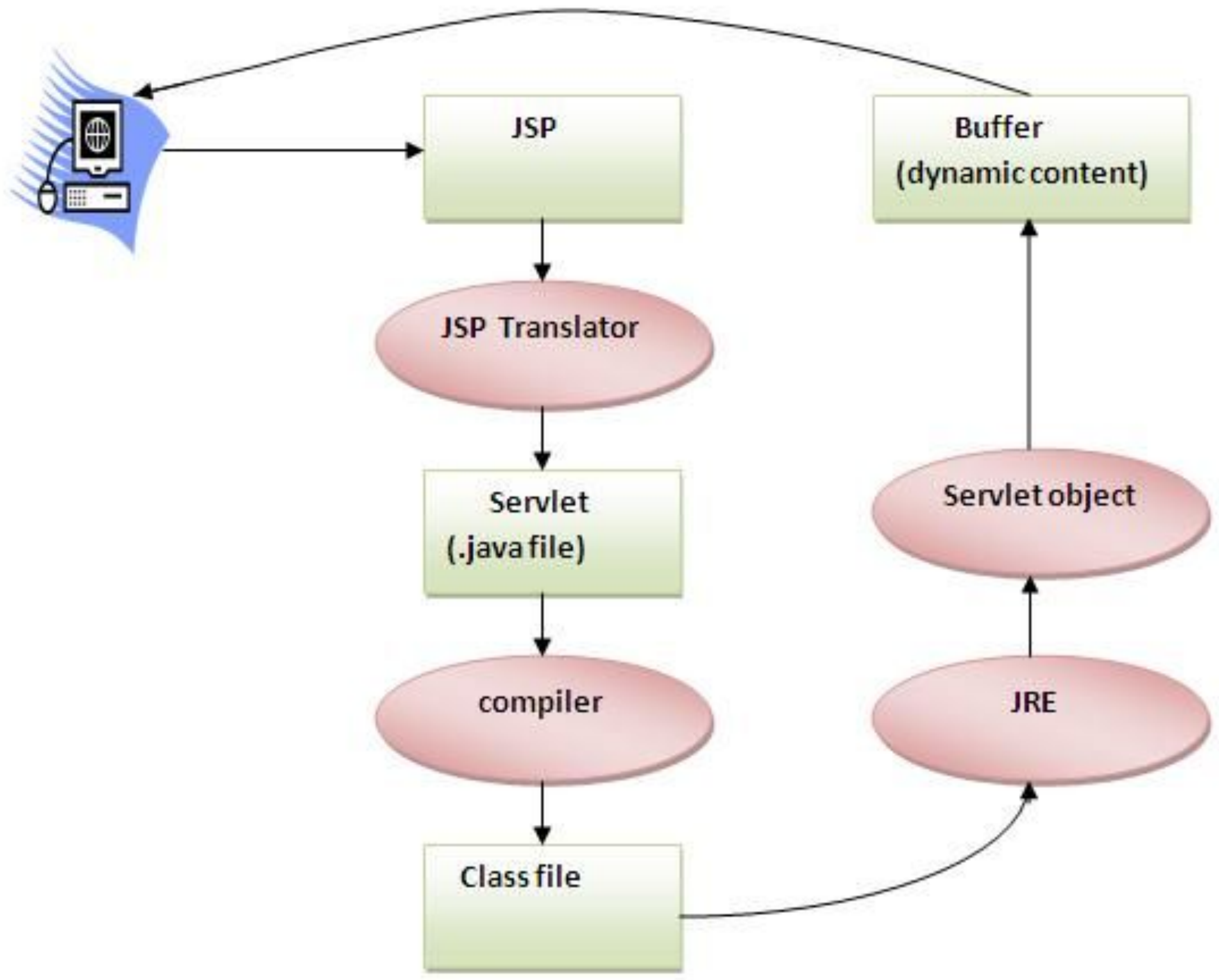
- JSPs run in two phases
  - Translation Phase
  - Execution Phase
- In translation phase JSP page is compiled into a servlet
  - called JSP Page Implementation class
- In execution phase the compliled JSP is processed



Java Server Pages (JSP) Architecture



Life cycle of a JSP Page



Life cycle of a JSP Page

1. Translation of JSP to Servlet code.
2. Compilation of Servlet to bytecode.
3. Loading Servlet class.
4. Creating servlet instance.
5. Initialization by calling jspInit() method
6. Request Processing by calling _jspService() method
7. Destroying by calling jspDestroy() method

What happens to a JSP when it is translated into Servlet?

- hello.jsp

```
<html>
    <head>
    <title>My First JSP Page</title>
    </head>
   <%   int count = 0;   %>

<body>
            Page Count is:
    <% out.println(++count); %>
 </body>
</html>
```

JSP page(hello.jsp) becomes this Servlet

```
public class hello_jsp extends HttpServlet
 {
 public void _jspService(HttpServletRequest request,
HttpServletResponse response) throws
IOException,ServletException
 {
PrintWriter out = response.getWriter();
response.setContenType("text/html");
out.write("<html><body>");
int count=0;
out.write("Page count is:");
out.print(++count);
out.write("</body></html>");
} }
```

Components of JSP

Four different elements are used in constructing JSPs

1. Scripting Elements
2. Implicit Objects
3. Directives
4. Actions

1. Scripting Elements

- There are three kinds of scripting elements

1. scriptlet tag
2. expression tag
3. declaration tag

JSP scriptlet tag

- A scriptlet tag is used to execute java source code in JSP.

<% java source code %>

Example of JSP scriptlet tag

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Example of JSP scriptlet tag

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

Example of JSP scriptlet tag

Welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

JSP expression tag

- The code placed within **JSP expression tag** is *written to the output stream of the response*.
- So you need not write out.print() to write data.
- It is mainly used to print the values of variable or method.

```
<%=  statement %>
```

Example of JSP expression tag

```
<html>

<body>

<%= "welcome to jsp" %>

</body>

</html>
```

Example of JSP expression tag

To display current time:

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

JSP Declaration Tag

- The **JSP declaration tag** is used *to declare fields and methods*.
- The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.
- So it doesn't get memory at each request.

<%! field or method declaration %>

JSP Declaration Tag

```
<html>
<body>

<%! int data=50; %>
<%= "Value of the variable is:"+data
%>

</body>
</html>
```

```
<html>
<body>

<%!
int cube(int n)
{
return n*n*n*;
}
%>

<%= "Cube of 3 is:"+cube(3) %>

</body>
</html>
```

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

JSP Comments: Two Types

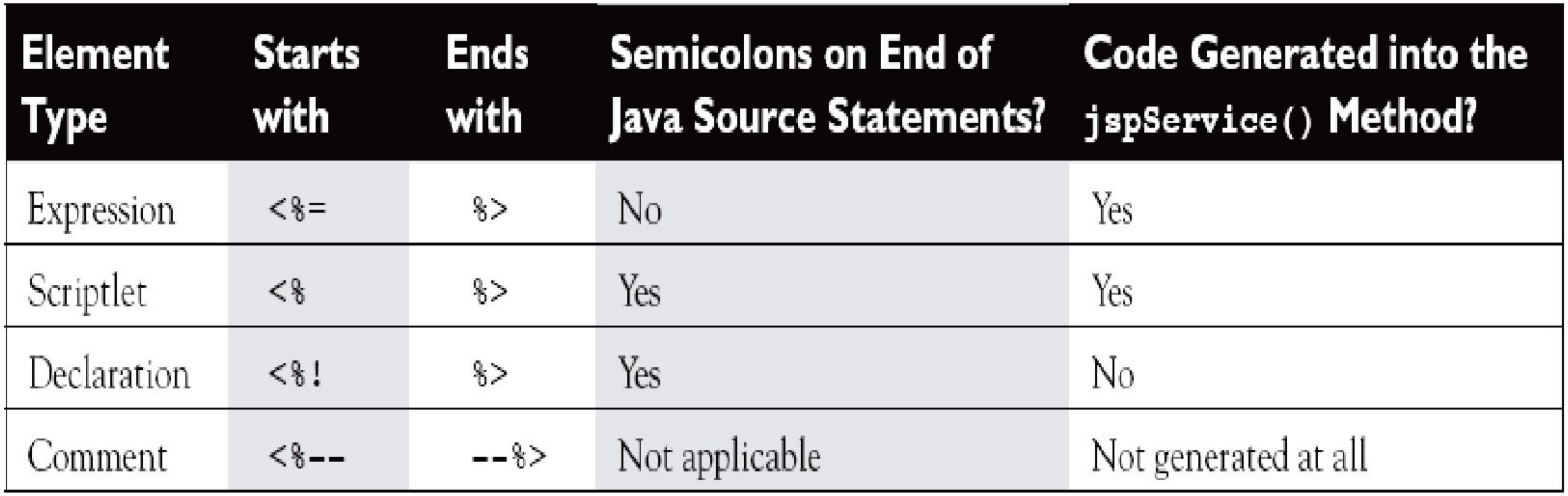
- JSP comments
  - A JSP comment.
  - Ignored by the JSP engine.
  - Not visible in client machine (Browser source code).

```
<%-- This is a JSP comment --%>
```

- HTML comments
  - An HTML comment.
  - Ignored by the browser.
  - It is visible in client machine (Browser source code) as a comment.

```
<!-- This comment is HTML -->
```

More Details



Element Type	Starts with	Ends with	Semicolons on End of Java Source Statements?	Code Generated into the `jspService()` Method?
Expression	`<%=`	`%>`	No	Yes
Scriptlet	`<%`	`%>`	Yes	Yes
Declaration	`<%!`	`%>`	Yes	No
Comment	`<%--`	`--%>`	Not applicable	Not generated at all

JSP Implicit Objects

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

JSP Implicit Objects

Implicit Object	**Description**
request	The HttpServletRequest object associated with the request.
response	The HttpServletResponse object associated with the response that is sent back to the browser.
out	The JspWriter object associated with the output stream of the response.

JSP Implicit Objects

Implicit Object	**Description**
session	The HttpSession object associated with the session for the given user of request.
application	The ServletContext object for the web application.
config	The ServletConfig object associated with the servlet for current JSP page.

JSP Implicit Objects

Implicit Object	Description
pageContext	The PageContext object that encapsulates the enviroment of a single request for this current JSP page
page	The page variable is equivalent to this variable of Java programming language.
exception	The exception object represents the Throwable object that was thrown by some other JSP page.

The request Object

- The request object provides methods to get the HTTP header information including form data, cookies, HTTP methods etc.

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

The Response object

- It used to add or manipulate response such as redirect response to another resource, send error etc.
- **Methods of response Implicit Object**
- void setContentType(String type)
  void sendRedirect(String address)
  void addHeader(String name, String value)
  void setHeader(String name, String value)
  boolean containsHeader(String name)
  void addCookie(Cookie value)
  void sendError(int status_code, String message)
  void setStatus(int statuscode)

The Response object

- response.setContentType("text/html");
- response.sendRedirect("http://google.com");
- response.addCookie(Cookie Author);
- response.sendError(404, "Page not found error");

JSP out object

- For writing any data to the buffer, JSP provides an implicit object named out.
- It is the object of JspWriter. In case of servlet you need to write:

  PrintWriter out=response.getWriter();
- But in JSP, you don't need to write this code.

JSP session object

- In JSP, session is an implicit object of type HttpSession.
- The Java developer can use this object to set, get or remove attribute or to get session information.

  e.g. session.setAttribute("user",name);

JSP application object

- In JSP, application is an implicit object of type *ServletContext*.
- The instance of ServletContext is created only once by the web container when application or project is deployed on the server.
- This object can be used to get initialization parameter from configuaration file (web.xml).
- It can also be used to get, set or remove attribute from the application scope.
- This initialization parameter can be used by all jsp pages.

JSP application object

Methods:

- Object getAttribute(String attributeName)
- void setAttribute(String attributeName, Object object)
- void removeAttribute(String objectName)
- Enumeration getAttributeNames()
- String getInitParameter(String paramname)
- Enumeration getInitParameterNames()
- void log(String message)
- URL getResource(String value)
- String getServerInfo()

JSP application object

- **index.html**

```
<form action="welcome">

<input type="text" name="uname">

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

</form>
```

JSP application object

web.xml file

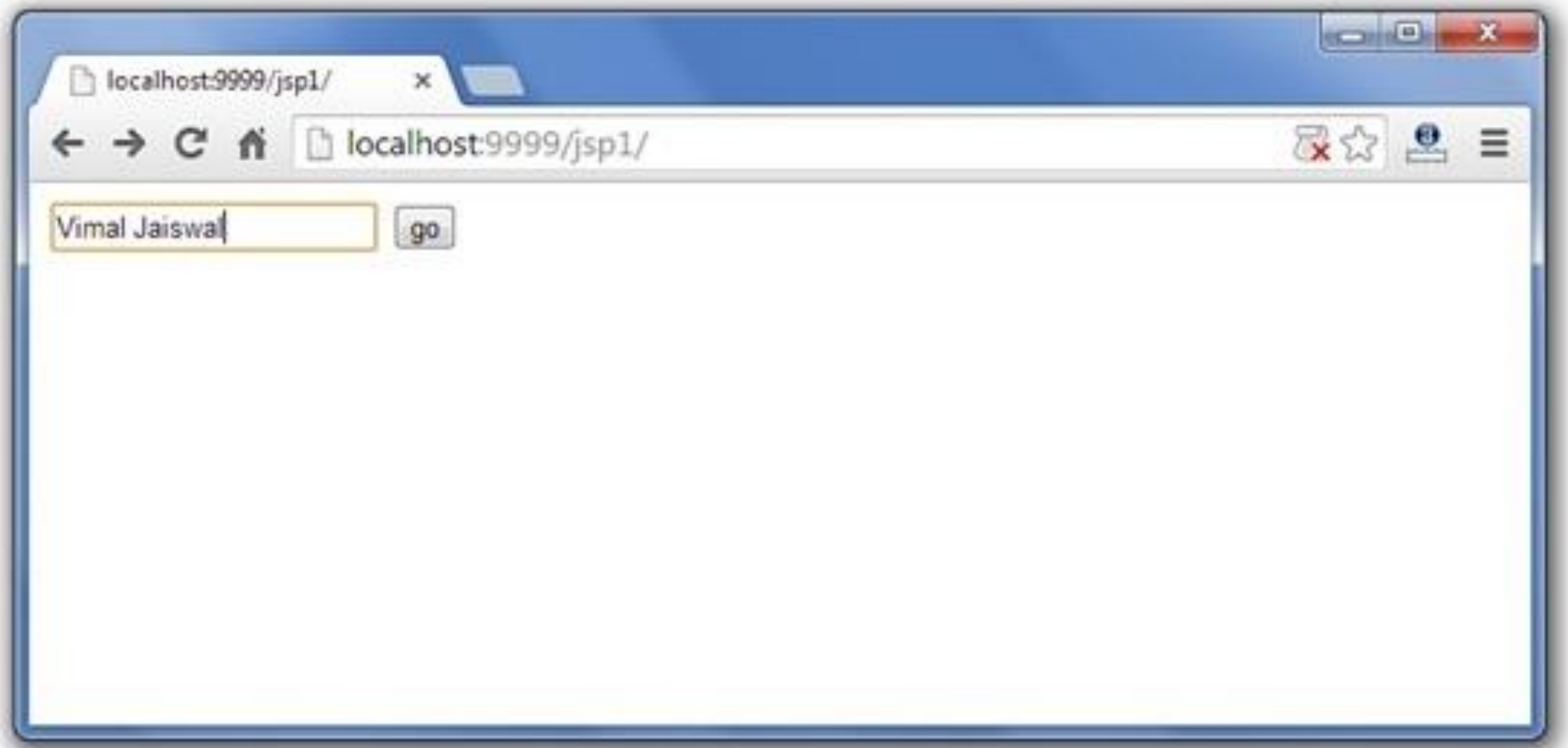
```
<web-app>

<servlet>
<servlet-name>Servlet1</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>
</servlet>

<servlet-mapping>
<servlet-name> Servlet1 </servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

<context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>

</web-app>
```

JSP application object

welcome.jsp

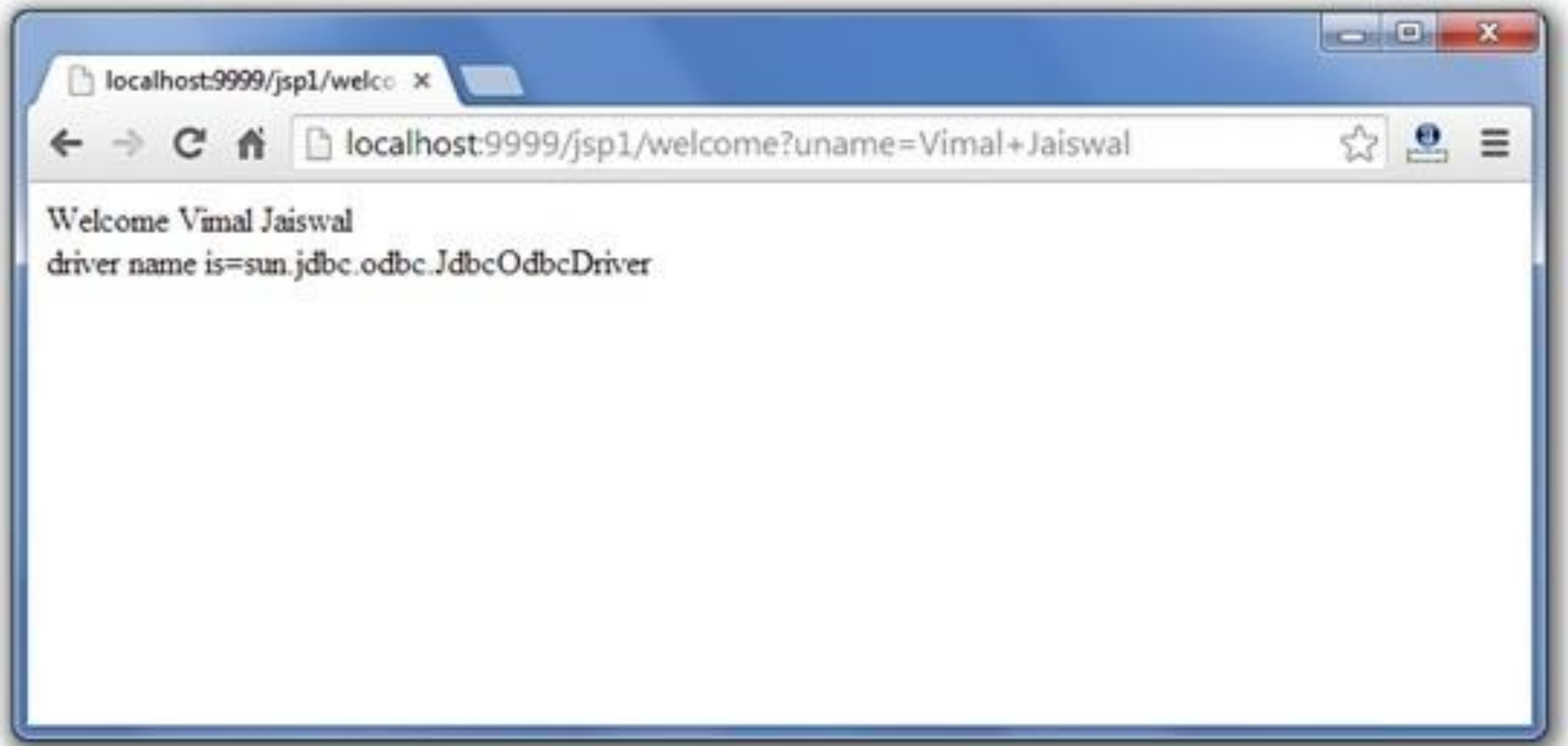
```
<%

out.print("Welcome "+request.getParameter("uname"));

String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);

%>
```

JSP application object



Vimal Jaiswal

JSP application object



localhost:9999/jsp1/welcome?uname=Vimal+Jaiswal

```
Welcome Vimal Jaiswal
driver name is=sun.jdbc.odbc.JdbcOdbcDriver
```

JSP config object

- In JSP, config is an implicit object of type ServletConfig.
- This object can be used to get initialization parameter for a particular JSP page.
- The config object is created by the web container for each jsp page.
- Generally, it is used to get initialization parameter from the web.xml file.

JSP config object

- **index.html**

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

JSP config object

web.xml file

```
<web-app>

<servlet>
<servlet-name>servlet1</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>

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

</servlet>

<servlet-mapping>
<servlet-name>servlet1</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

JSP config object

welcome.jsp

```
<%

out.print("Welcome "+request.getParameter("uname"));


String driver=config.getInitParameter("dname");

out.print("driver name is="+driver);

%>
```

Servlet Config	**Servlet Context**
Servlet config object represent single servlet	It represent whole web application running on particular JVM and common for all the servlet
Its like local parameter associated with particular servlet	Its like global parameter associated with whole application
It's a name value pair defined inside the servlet section of web.xml file so it has servlet wide scope	`ServletContext` has application wide scope so define outside of servlet tag in web.xml file.
`getServletConfig()` method is used to get the config object	`getServletContext()` method is used to get the context object.
for example shopping cart of a user is a specific to particular user so here we can use servlet config	To get the MIME type of a file or application session related information is stored using servlet context object.

JSP pageContext object

- In JSP, pageContext is an implicit object of type PageContext class.
- The pageContext object can be used to set,get or remove attribute from one of the following scopes:

1. page
2. request
3. session
4. application

In JSP, page scope is the default scope.

JSP pageContext object

- JSP Page – Scope: PAGE_CONTEXT
- HTTP Request – Scope: REQUEST_CONTEXT
- HTTP Session – Scope: SESSION_CONTEXT
- Application Level – Scope: APPLICATION_CONTEXT

- JSP pageContext object Example

JSP page object

- This object is an actual reference to the instance of the page.
- It can be thought of as an object that represents the entire JSP page.
- The page object is really a direct synonym for the this object.
- `<%=this.getClass().getName() %>`

JSP exception object

- It’s an instance of java.lang.Throwable.
- It is used for exception handling in JSP.
- This object is only available for error pages, which means a JSP page should have isErrorPage to true in order to use exception implicit object.
- JSP exception object Example

JAVA Beans

A Java Bean is a java class that should follow following conventions:

1. It should have a no-arg constructor.
2. It should be Serializable.
3. It should provide methods to set and get the values of the properties, known as getter and setter methods.

JAVA Beans

Why use Java Bean?

- It is a reusable software component.
- A bean encapsulates many objects into one object.
- We can access this object from multiple places.

JavaBeans Example

```
 public class StudentsBean implements
java.io.Serializable {
  private String firstName = null;
  private String lastName = null;
  private int age = 0;

  public StudentsBean() {
  }
  public String getFirstName(){
    return firstName;
  }
  public String getLastName(){
    return lastName;
  }
```

```
public int getAge(){
    return age;
  }

public void setFirstName(String firstName){
    this.firstName = firstName;
  }
  public void setLastName(String lastName){
    this.lastName = lastName;
  }
  public void setAge(Integer age){
    this.age = age;
  }
}
```

JSP Actions :useBean

Syntax of jsp:useBean action tag

```
<jsp:useBean id= "instanceName"

            scope= "page | request | session | application"

            class= "packageName.className"

            type= "packageName.className"

            beanName="packageName.className >

</jsp:useBean>
```

JSP Actions :useBean

Attributes of jsp:useBean action tag

- **id:** is used to identify the bean(beans's name).
- **scope:** The default scope is page.
  - **page:** bean is used within the JSP page.
  - **request:** bean is used from any JSP page that processes the same request. It has wider scope than page.
  - **session:** bean is used from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
  - **application:** bean is used from any JSP page in the same application. It has wider scope than session.

JSP Actions :useBean

Attributes of jsp:useBean action tag

- **class:** instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
- **type:** provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
- **beanName:** instantiates the bean using the java.beans.Beans.instantiate() method.

JSP Actions:setProperty

- setProperty is used to set bean properties
- Example (setProperty)

<jsp:setProperty name="myBean" property="firstName" value="Sanjay"/>

  - Sets the name property of myBean to SanjayExample (setProperty)

<jsp:setProperty name="myBean" property="*">

  - Sets property to the corresponding value in request

JSP Actions:getProperty

- getProperty is used in conjunction with useBean.
- used to get property values of the bean defined by the useBean action
- Example (getProperty)

  <jsp:getProperty name=“myBean” property=“firstName” />

  - Name corresponds to the id value in the useBean
  - Property refers to the name of the bean property

JSP Actions :useBean
Example 1

MyBean.java

```
package my;
public class MyBean {
    private String name=new String();
        public String getName() {
        return name;
      }
      public void setName(String name) {
        this.name = name;
      }
  }
```

JSP Actions :useBean Example 1

useBeanExample.jsp

```
<html>
<head>
<h1>Java bean example in jsp</h1>
<hr></hr>
</head>
<body>
<jsp:useBean id="mybean" class="my.MyBean" scope="session" >
            <jsp:setProperty name="mybean" property="name" value=" Hello world " />
 </jsp:useBean>

<h1> <jsp:getProperty name="mybean" property="name" /></h1>
</body>
</html>
```

JSP Actions :useBean   Example 2

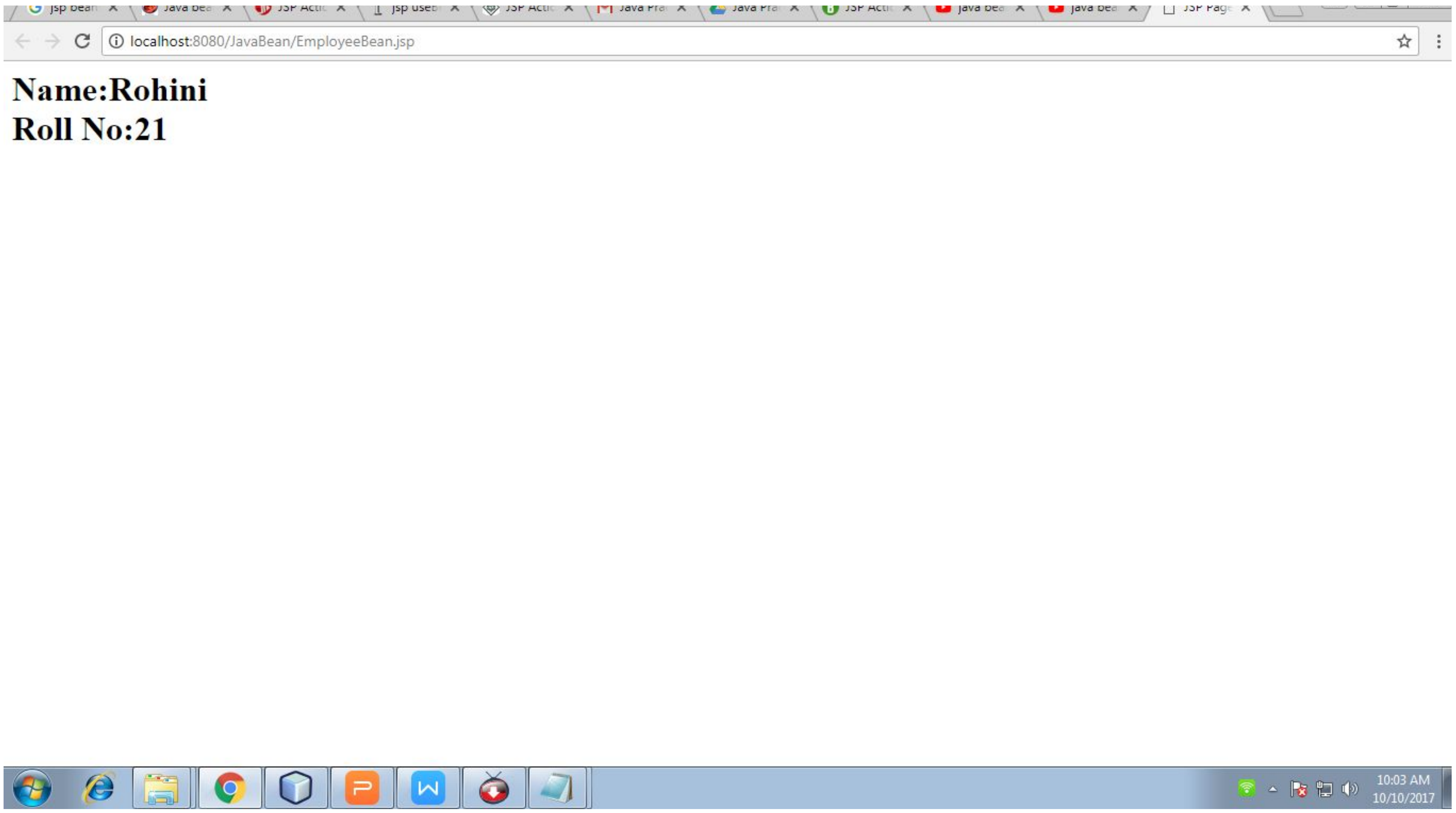
StudentBean.java

```
package javabeansample;
public class StuBean
{
public StuBean() { }
   private String name;
   private int rollno;
   public void setName(String name)
      { this.name=name; }
   public String getName()
       { return name; }
   public void setRollno(int rollno)
      { this.rollno=rollno; }
   public int getRollno()
      { return rollno; }
}
```

JSP Actions :useBean Example2

EmployeeBeanTest.jsp

```
<html>
<head>
 <title>JSP Page to show use of useBean action
</title>
</head> <body>
<jsp:useBean id="student" class="javabeansample.StuBean"/>

<jsp:setProperty name="student" property="name" value="Rohini"/>
<jsp:setProperty name="student" property="rollno" value="21"/>

<h1> name:<jsp:getProperty name="student" property="name"/>
<br> empno:<jsp:getProperty name="student" property="rollno"/>
<br>
</h1> </body>
 </html>
```



localhost:8080/JavaBean/EmployeeBean.jsp

```
Name:Rohini
Roll No:21
```

JSP Actions :useBean Example3

- index.jsp
- process.jsp
- User.java

JSP Actions :useBean Example3

- index.jsp

```
<form action="process.jsp" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
Email:<input type="text" name="email"><br>
<input type="submit" value="register">
</form>
```

JSP Actions :useBean Example3

- process.jsp

```
<jsp:useBean id="userbean"
class="Mybean.User"></jsp:useBean>
<jsp:setProperty name="userbean" property="*" />

Record:<br>
<jsp:getProperty name="userbean" property="name" /><br>
<jsp:getProperty name="userbean" property="password" /><br>
<jsp:getProperty name="userbean" property="email" /><br>
```

JSP Actions :useBean Example3

- **User.java**

```
package Mybean;
public class User {
    private String name, password, email;
    public String getEmail() {
        return email;
    }
   public void setEmail(String email) {
        this.email = email;
    }
    public String getPassword() {
        return password;
    }
public void setPassword(String password)
{
        this.password = password;
    }

    public String getName()
{
        return name;
    }

        public void setName(String name)
{
        this.name = name;
    }

}
```

JSP Actions: plugIn

- ~~The jsp:plugin action tag is used to embed java components~~ like applet or beans in the jsp file.
- The jsp:plugin action tag downloads plugin at client side to execute an applet or bean.
- **Syntax of jsp:plugin action tag**

```
<jsp:plugin type= "applet | bean" code= "nameOfClassFile"
codebase= "directoryNameOfClassFile"
</jsp:plugin>
```

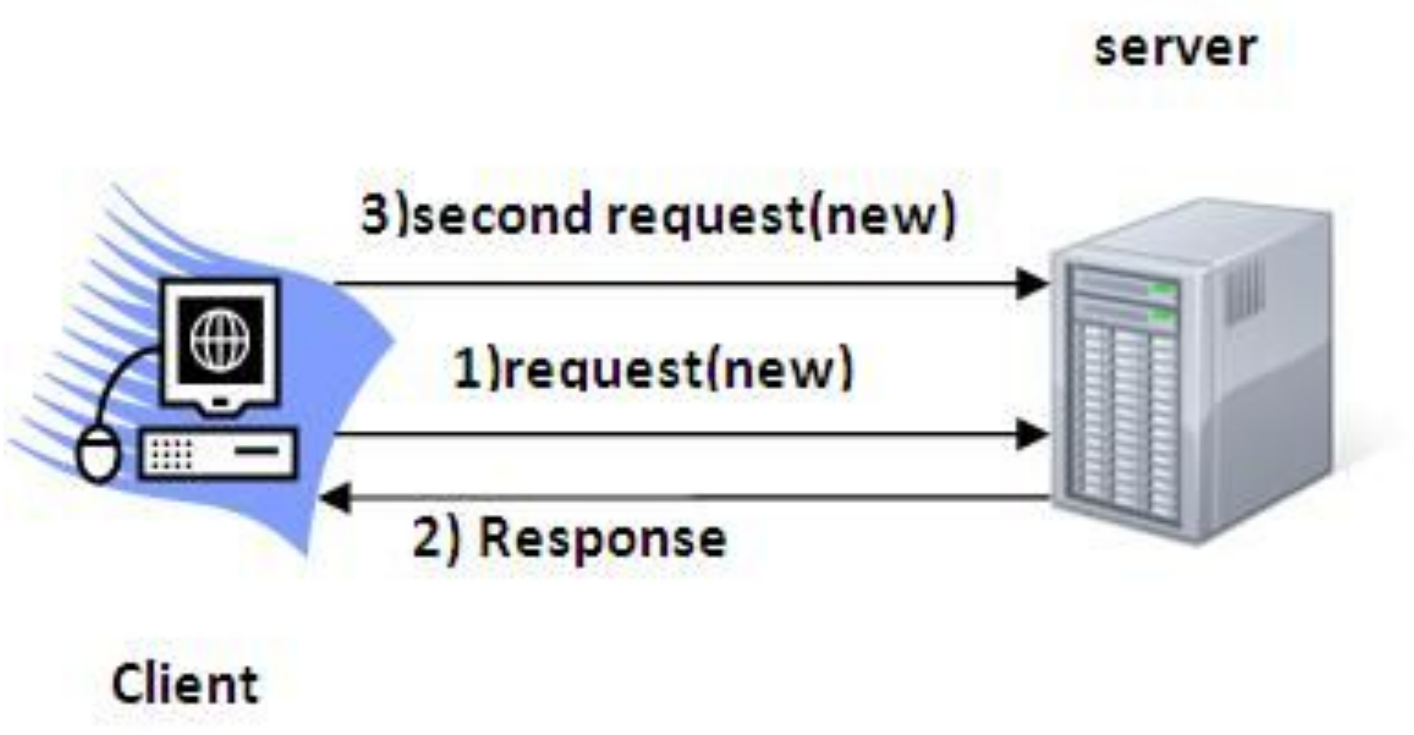
JSP Actions: plugIn

- Example of JSP Actions: plugIn
- MouseDrag.java
- index.jsp

Session Managment

- Session simply means a particular interval of time.
- Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.
- Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

Session Managment



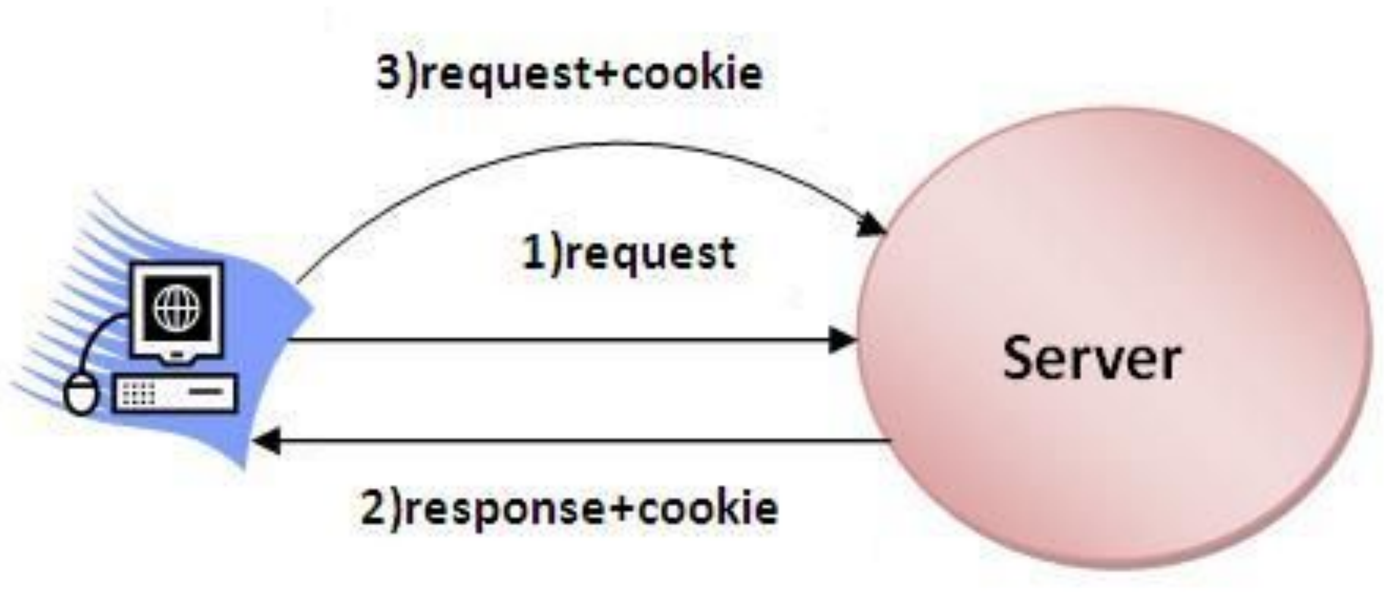
Session Tracking/ Management Techniques

- There are four techniques used in Session tracking:
- **Cookies**
- **Hidden Form Field**
- **URL Rewriting**
- **HttpSession**

Cookie

- A cookie is a small piece of information that is persisted between the multiple client requests.
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

- It is **valid for single session** only. It is removed each time when user closes the browser.

Persistent cookie

- It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Cookie class

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

Methods of Cookie class

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.

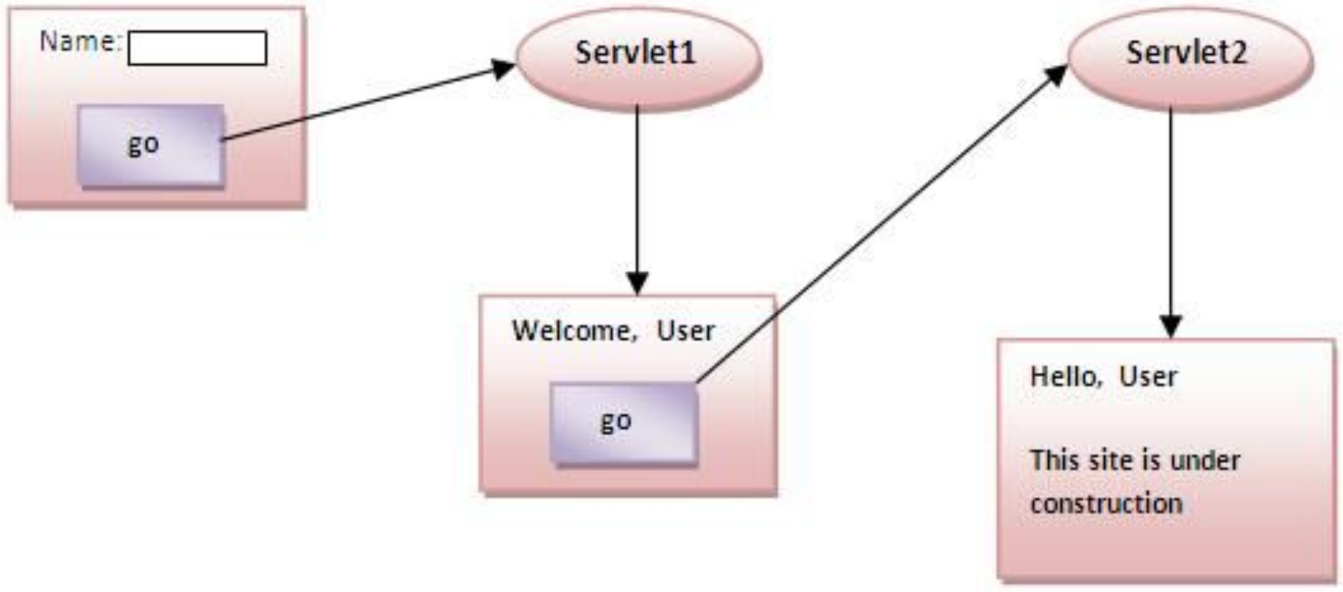
Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck):**
   method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():**
   method of HttpServletRequest interface is used to return all the cookies from the browser.

Cookie Example

- Cookie Example


Name:
go
Servlet1
Welcome, User
go
Servlet2
Hello, User
This site is under construction