# Unit 2
# Servlet and JSP

Web development using Servlets

# Syllabus…

- **Servlets:** Introduction
- Web application Architecture
- Http Protocol & Http Methods
- Web Server & Web Container
- Servlet Interface
- GenericServlet
- HttpServlet
- Servlet Life Cycle
- ServletConfig
- ServletContext
- Servlet Communication
- Session Tracking Mechanisms

# jsp

- **JSP:** Introduction
- JSP LifeCycle
- JSP Implicit Objects & Scopes
- JSP Directives
- JSP Scripting Elements
- JSP Actions: Standard actions and customized actions

# Servlet

- **Servlet** technology is used to create web application (resides at server side and generates dynamic web page).

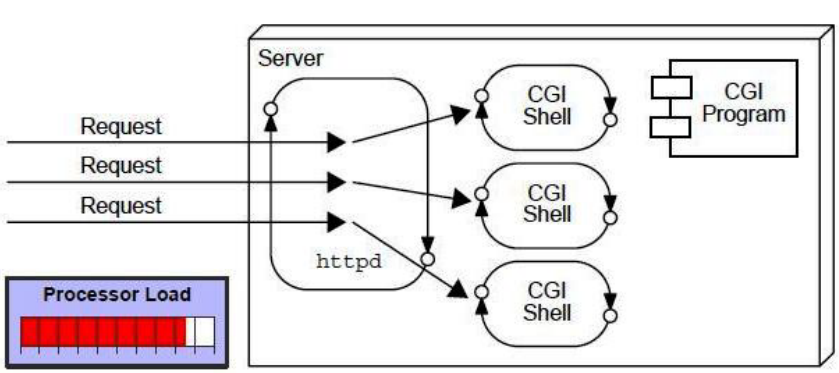
- **Servlet** technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there was many disadvantages of this technology.

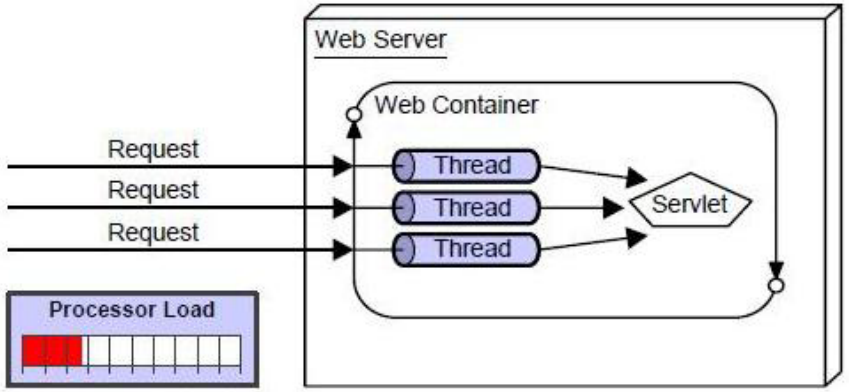
# What is a Servlet?

- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

# Servlet vs CGI

- 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

- 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.

# Benefits of servlets

- **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..

# Introduction - what is a request and response

## HTTP Request

Key elements of a "**request**" stream:

- HTTP method (action to be performed).
- The page to access (a URL).
- Form parameters.

## HTTP Response

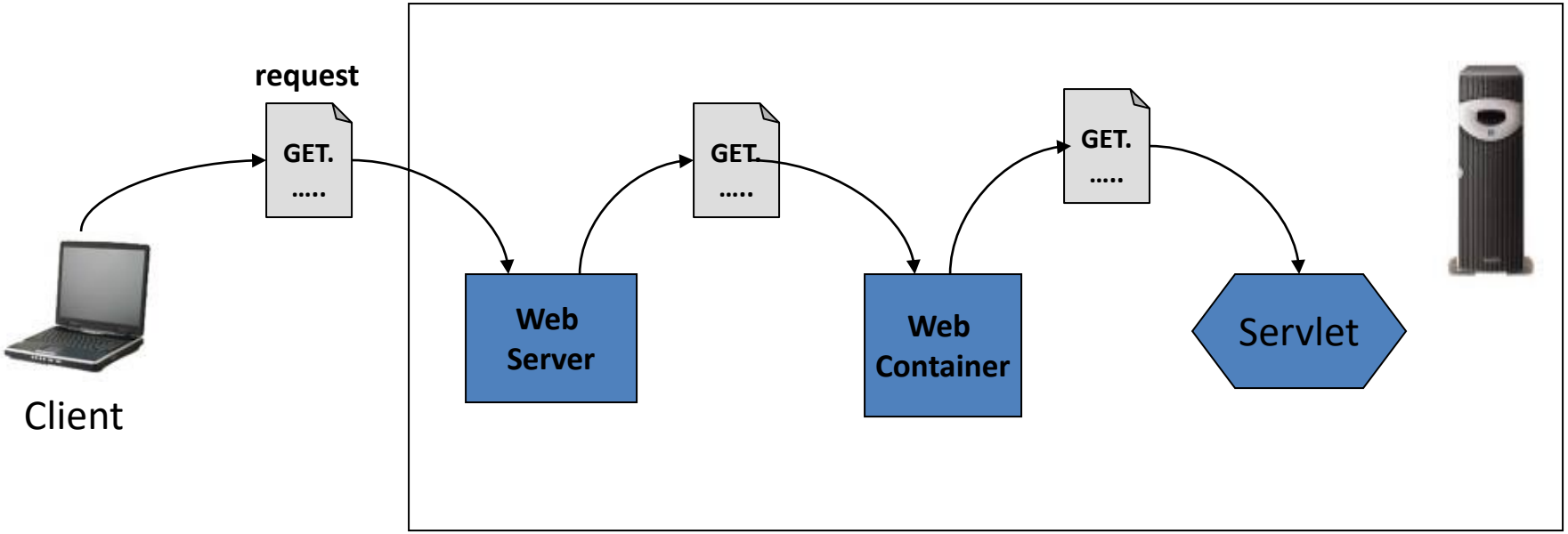
Key elements of a "**response**" stream:

- A status code (for whether the request was successful).
- Content-type (text, picture, html, etc…).
- The content ( the actual content).

# Servlet Architecture -Web Container

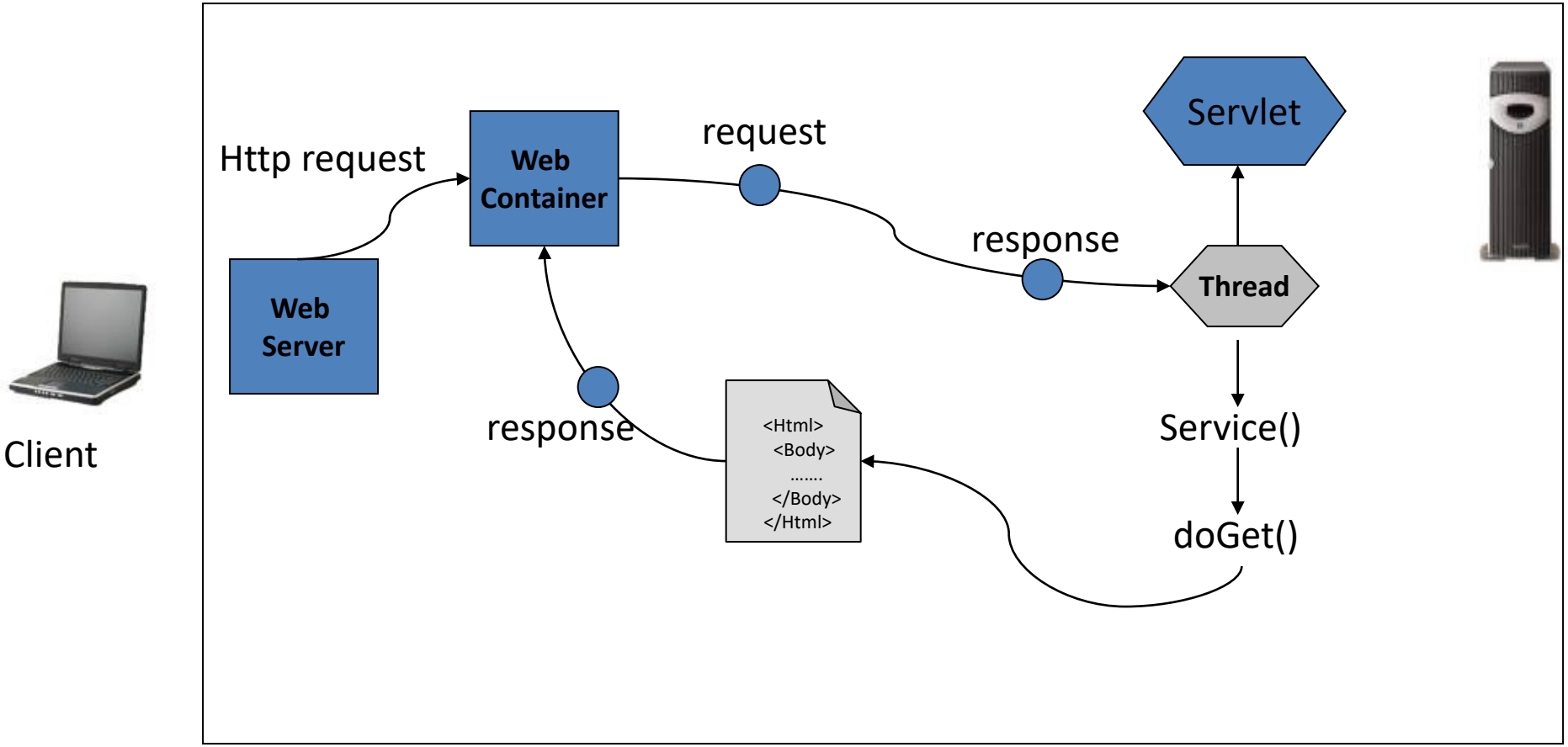
- **What is a Web Container?**

- Servlets don't have a main method.
  - They are under the control of another Java application called the "**Container**" **(e.g. Tomcat)**

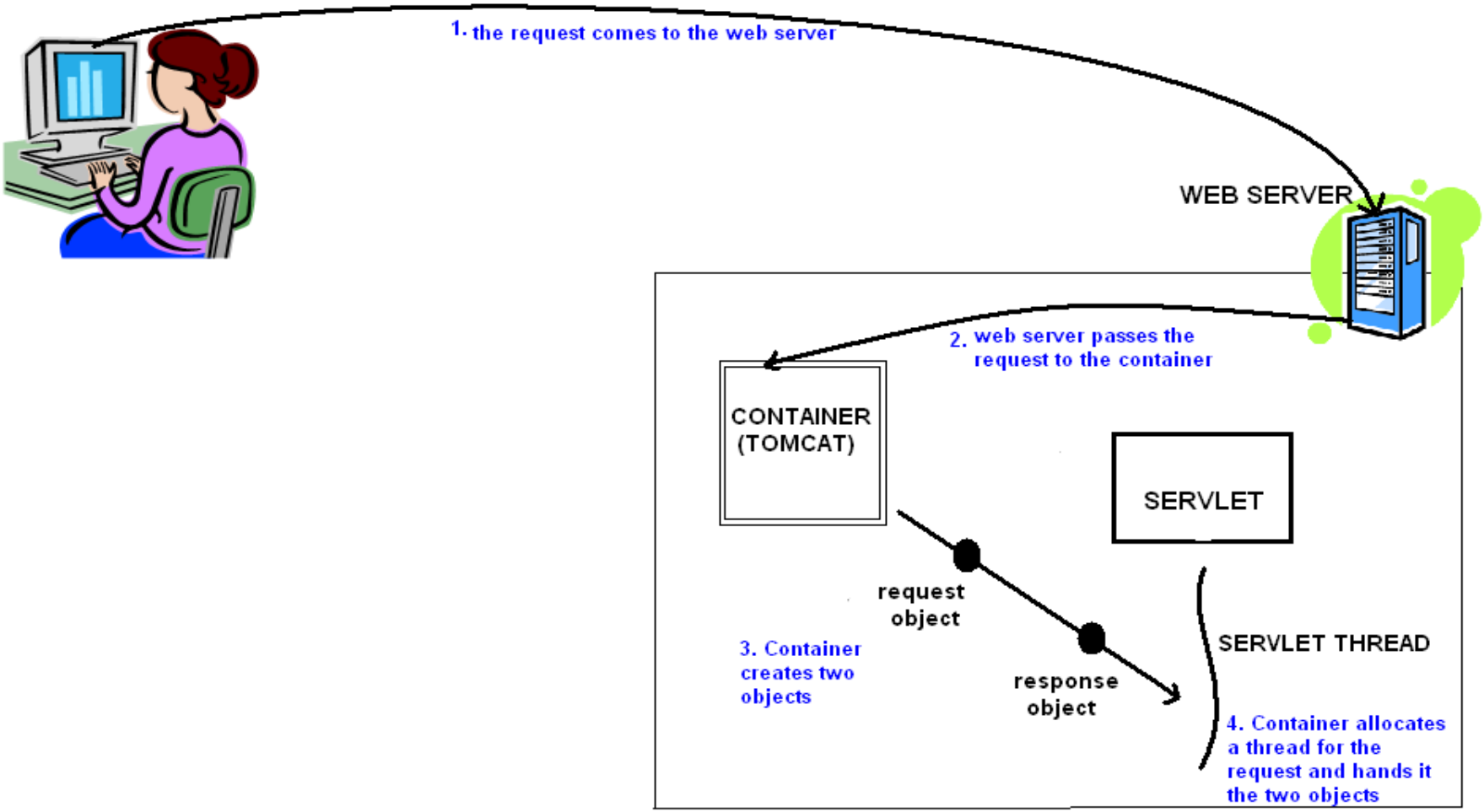


# Servlet Architecture - Web Container

- **How does the Container handle a request?**



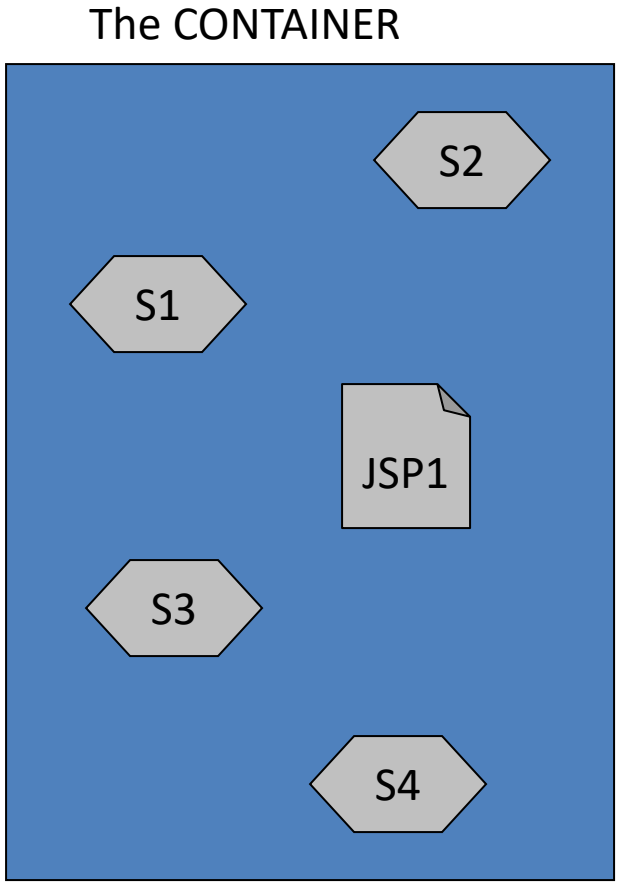
# How it works??


1. the request comes to the web server
WEB SERVER
2. web server passes the request to the container
CONTAINER (TOMCAT)
SERVLET
request object
response object
3. Container creates two objects
SERVLET THREAD
4. Container allocates a thread for the request and hands it the two objects

# Servlet Architecture - Web Container

## What is the role of Web Container ?

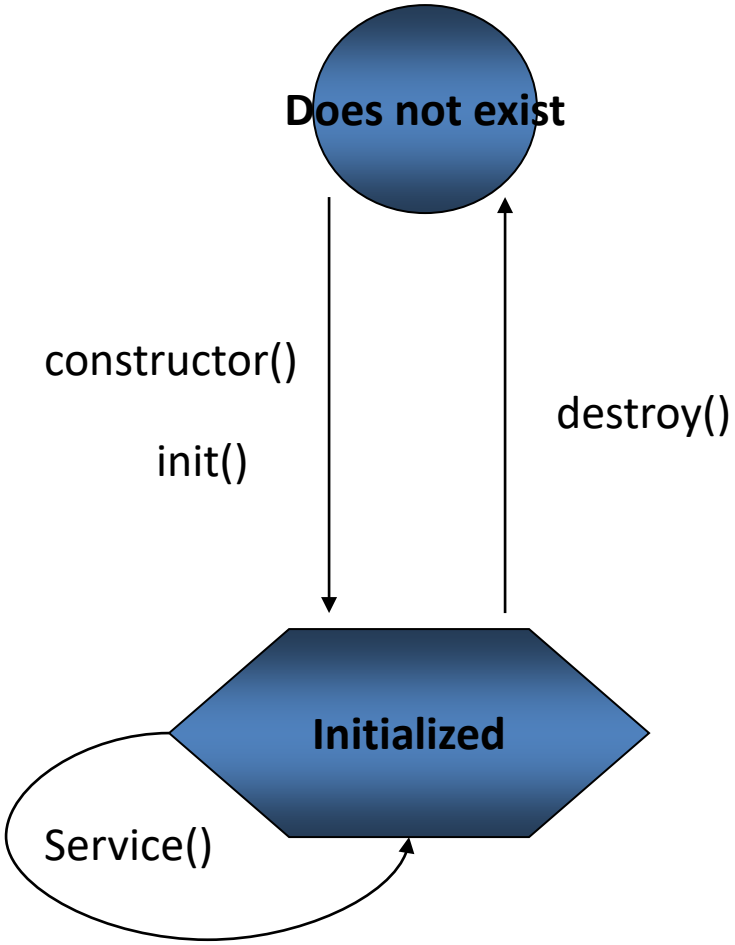
- Communication Support
- Lifecycle Management
- Multi-threading support
- Security
- JSP Support



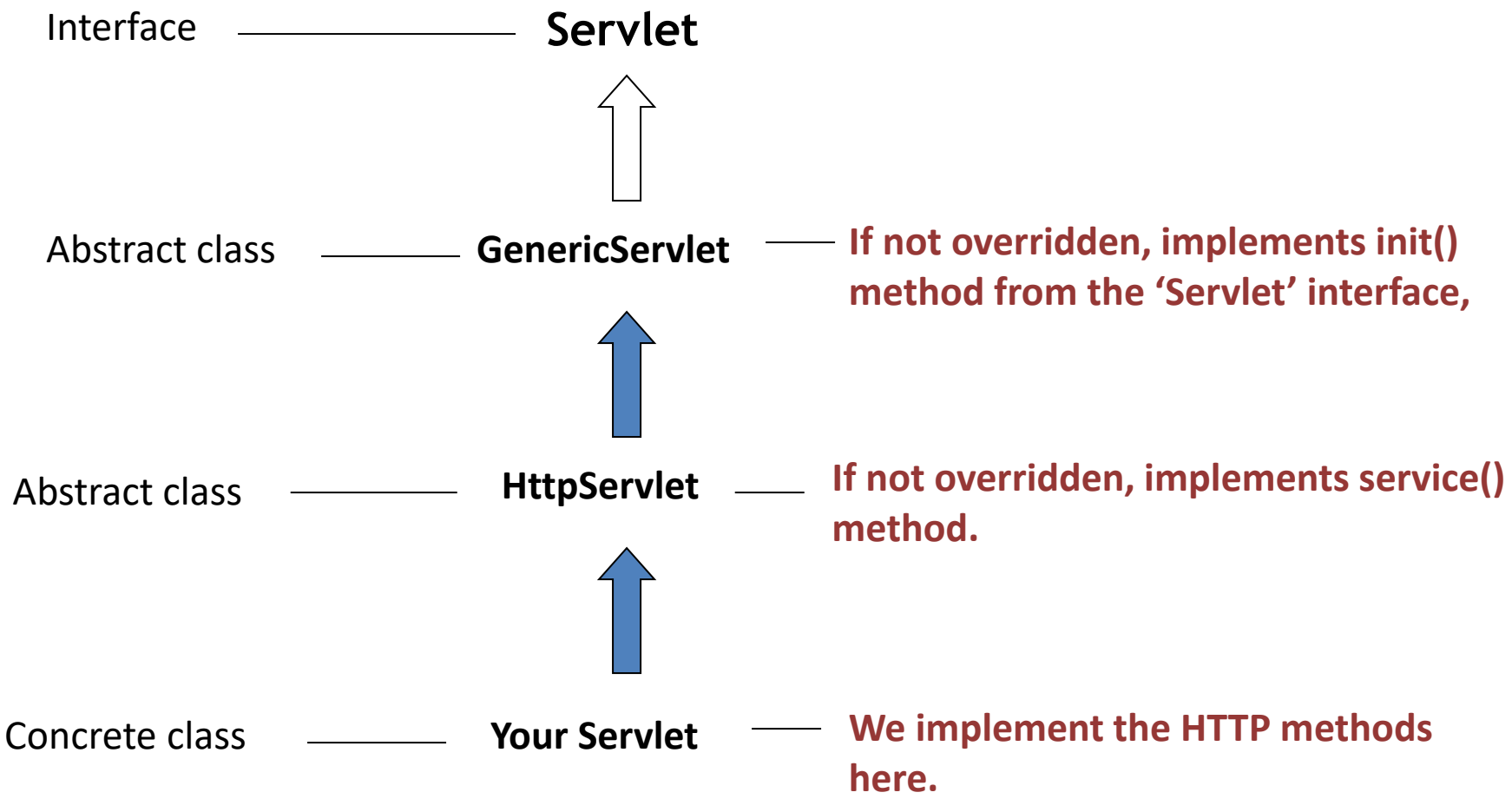

The container can contain multiple Servlets & JSPs within it

# Servlet Lifecycle

- **The Servlet lifecycle is simple, there is only one main state – "Initialized".**



# Servlet Lifecycle - Hierarchy

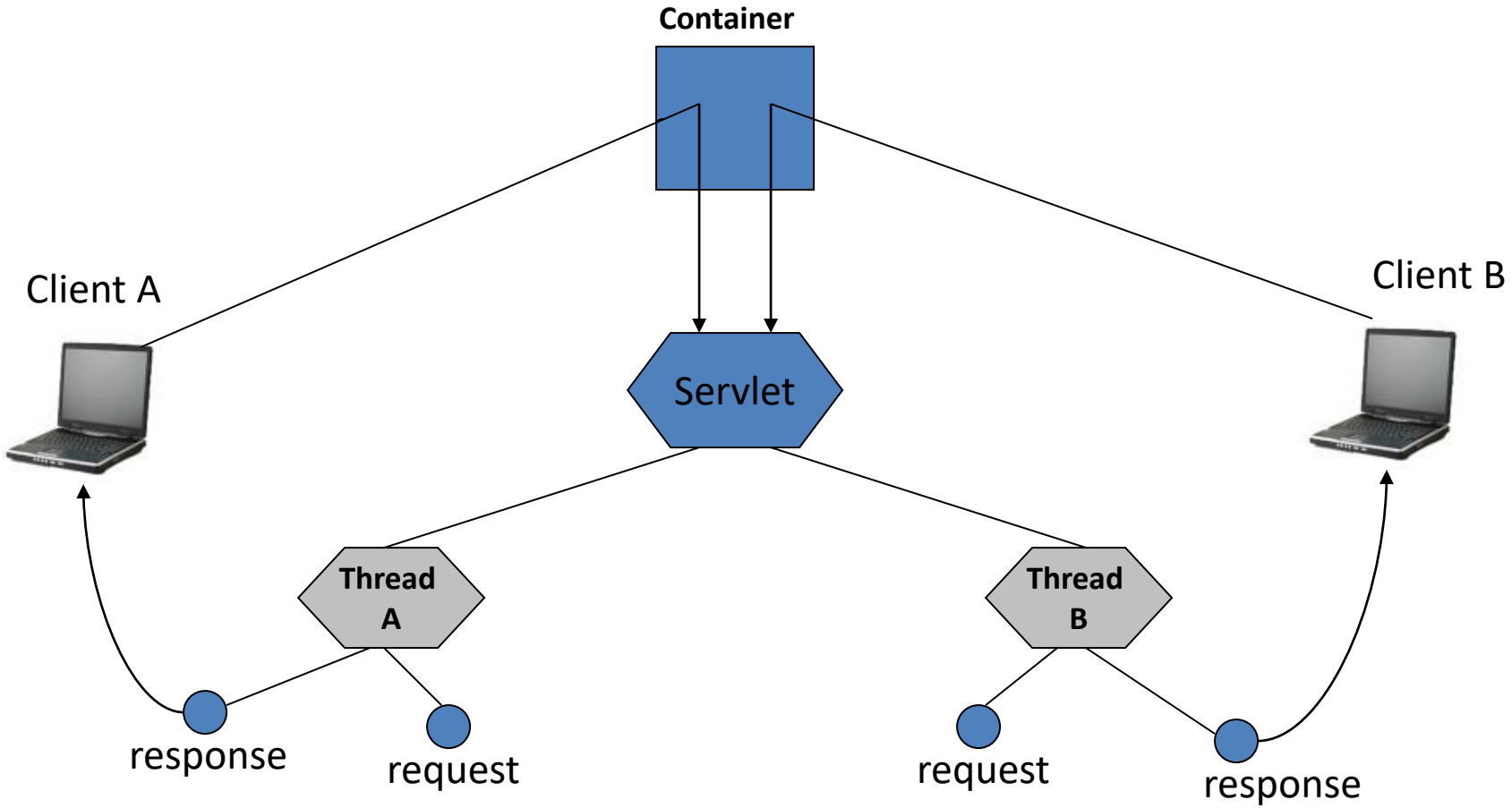


# Servlet Lifecycle – 3 big moments

| | When is it called | What it's for | Do you override it |
|---|---|---|---|
| **init()** | The container calls the init() before the servlet can service any client requests. | To initialize your servlet before handling any client requests. | Possibly |
| **service()** | When a new request for that servlet comes in. | To determine which HTTP method should be called. | No. Very unlikely |
| **doGet() or doPost()** | The service() method invokes it based on the HTTP method from the request. | To handle the business logic. | Always |

# Servlet Lifecycle - Thread handling

- **The Container runs multiple threads to process multiple requests to a single servlet.**

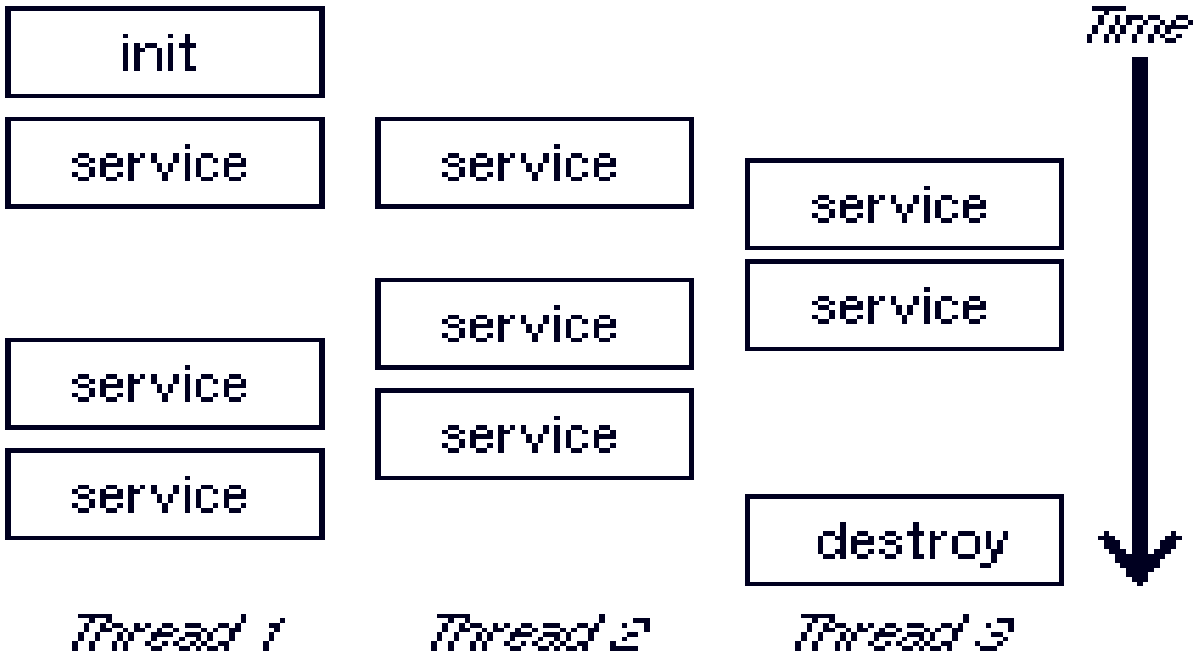


# Request and Response – GET v/s POST

- **The HTTP request method determines whether doGet() or doPost() runs.**

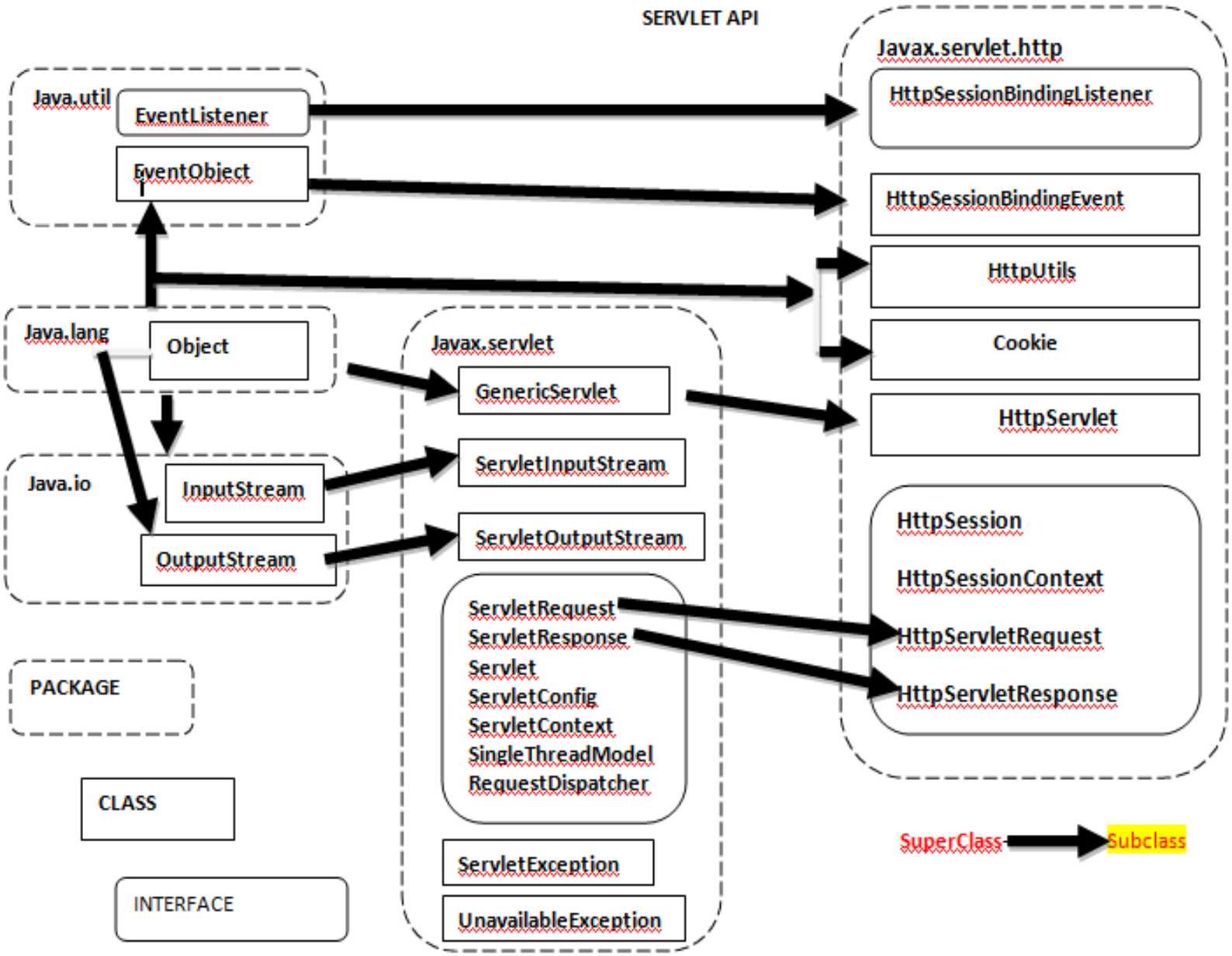
| | GET (doGet()) | POST (doPost()) |
|---|---|---|
| **HTTP Request** | The request contains only the request line and HTTP header. | Along with request line and header it also contains HTTP body. |
| **Parameter passing** | The form elements are passed to the server by appending at the end of the URL. | The form elements are passed in the body of the HTTP request. |
| **Size** | The parameter data is limited (the limit depends on the container) | Can send huge amount of data to the server. |
| **Idempotency** | GET is Idempotent | POST is not idempotent |
| **Usage** | Generally used to fetch some information from the host. | Generally used to process the sent data. |

# A Typical Servlet Lifecycle



# Servlet Life Cycle Steps

1. Server loads the servlet
2. Creates 1 or more instances of class
3. Calls the init()
4. Servlet request is received
5. Calls the service method
6. Service()  processes the request
7. Waits for next request or unloads it self
8. Use destroy()  to unload.


SERVLET API
Javax.servlet.http
Java.util
EventListener
EventObject
HttpSessionBindingListener
HttpSessionBindingEvent
HttpUtils
Cookie
Java.lang
Object
Javax.servlet
GenericServlet
HttpServlet
Java.io
InputStream
OutputStream
ServletInputStream
ServletOutputStream
HttpSession
HttpSessionContext
HttpServletRequest
HttpServletResponse
ServletRequest
ServletResponse
Servlet
ServletConfig
ServletContext
SingleThreadModel
RequestDispatcher
PACKAGE
CLASS
INTERFACE
ServletException
UnavailableException
SuperClass
Subclass

# RequestDispatcher Interface

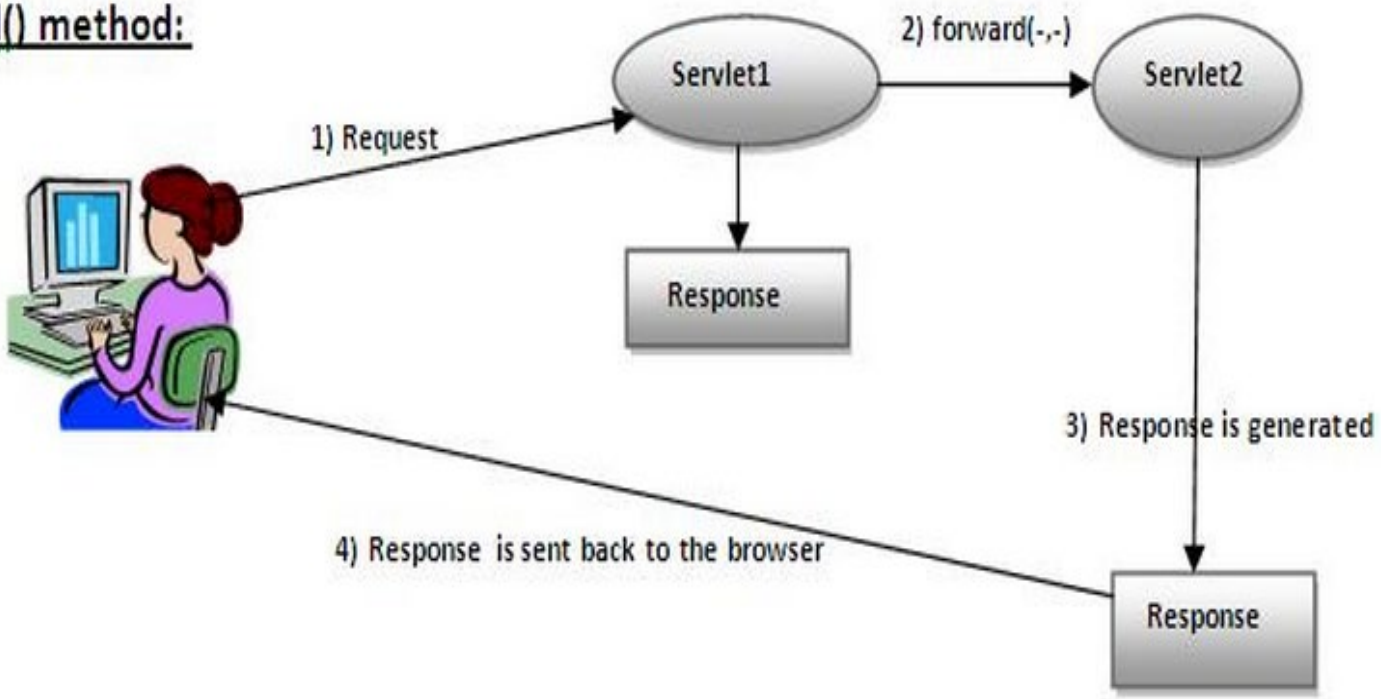
The RequestDispacher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp.This interface can also be used to include the content of antoher resource also. It is one of the way of servlet collaboration.

## Methods of RequestDispatcher interface

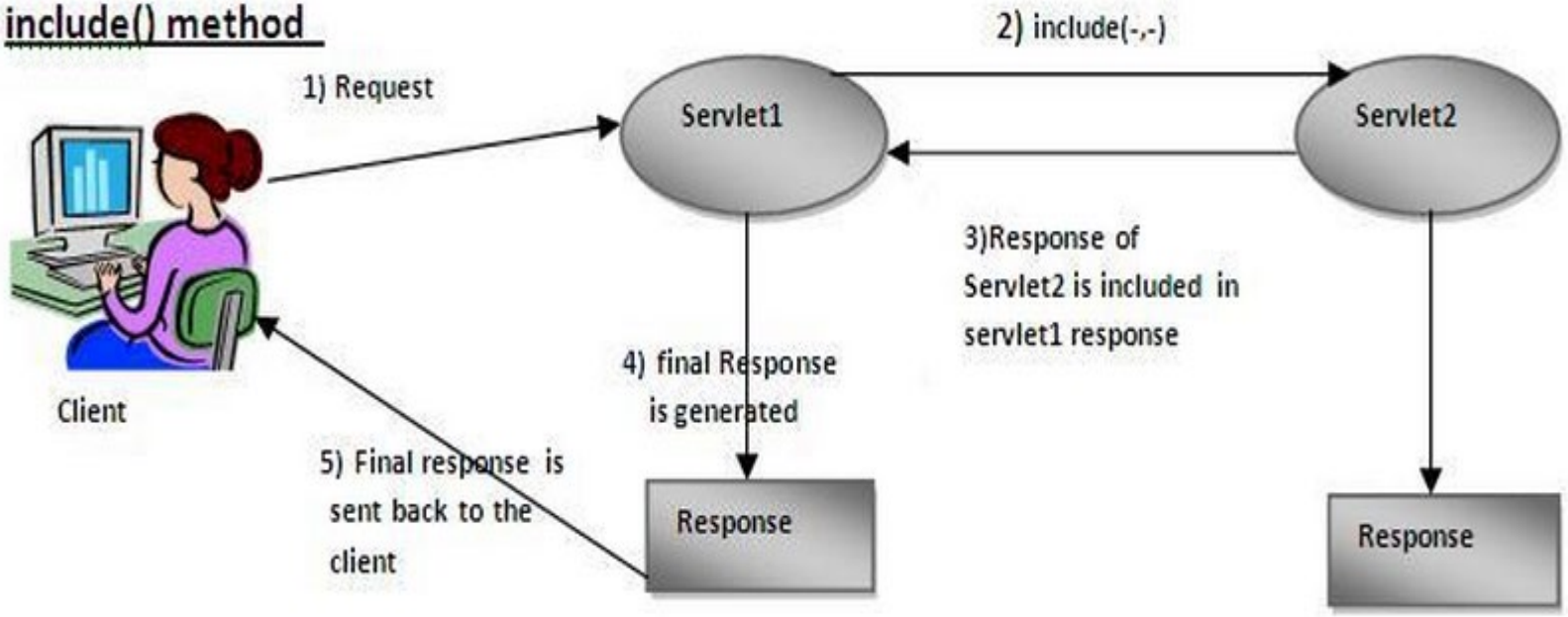
The RequestDispatcher interface provides two methods. They are:

1. **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
2. **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

## forward() method:



## include() method



## How to get the object of RequestDispatcher

The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher.
Syntax:

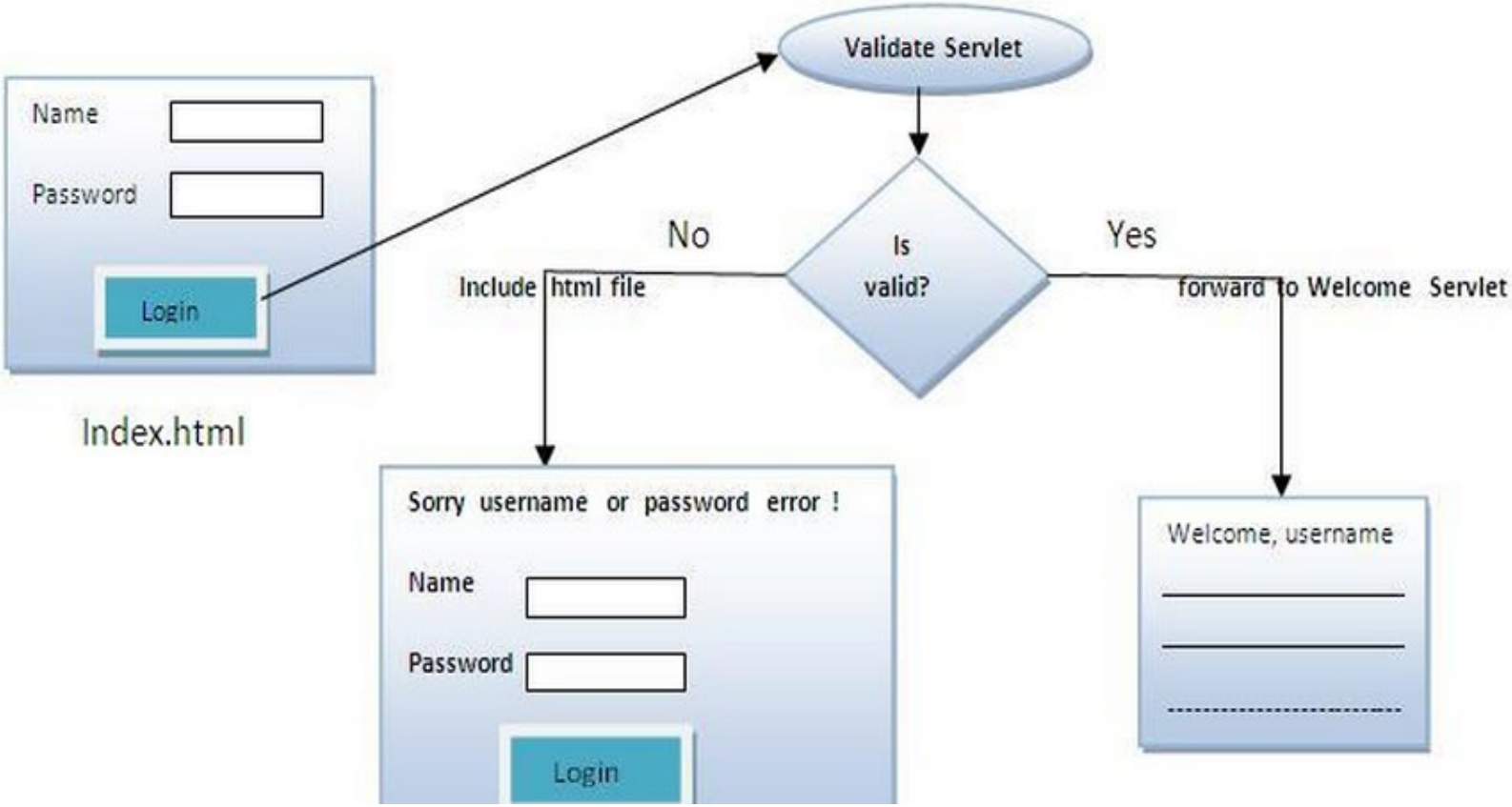
### Syntax of getRequestDispatcher method

```
public RequestDispatcher getRequestDispatcher(String resource);
```

## Example of using getRequestDispatcher method

```
RequestDispatcher rd=request.getRequestDispatcher("servlet2");
//servlet2 is the url-pattern of the second servlet

rd.forward(request, response);//method may be include or forward
```

- **index.html file:** for getting input from the user.
- **Login.java file:** a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.
- **WelcomeServlet.java file:** a servlet class for displaying the welcome message.
- **web.xml file:** a deployment descriptor file that contains the information about the servlet.


Validate Servlet
Name
Password
Login
Index.html
No
Is
valid?
Yes
Include html file
forward to Welcome Servlet
Sorry username or password error !
Name
Password
Login
Welcome, username

# Index.html

**index.html**

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
Password:<input type="password" name="userPass"/><br/>
<input type="submit" value="login"/>
</form>
```

```
public class Login extends HttpServlet {

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

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

        String n=request.getParameter("userName");
        String p=request.getParameter("userPass");

        if(p.equals("servlet")){
                RequestDispatcher rd=request.getRequestDispatcher("servlet2");
                rd.forward(request, response);
        }
        else{
                out.print("Sorry UserName or Password Error!");
                RequestDispatcher rd=request.getRequestDispatcher("/index.html");
                rd.include(request, response);

                }
        }
```

**WelcomeServlet.java**

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

public class WelcomeServlet extends HttpServlet {

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

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

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

}
```

# Web.xml

```
<web-app>
 <servlet>
    <servlet-name>Login</servlet-name>
    <servlet-class>Login</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>WelcomeServlet</servlet-name>
    <servlet-class>WelcomeServlet</servlet-class>
  </servlet>


  <servlet-mapping>
    <servlet-name>Login</servlet-name>
    <url-pattern>/servlet1</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/servlet2</url-pattern>
  </servlet-mapping>

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

# ServletConfig Interface


1. ServletConfig Interface
2. Methods of ServletConfig interface
3. How to get the object of ServletConfig
4. Syntax to provide the initialization parameter for a servlet
5. Example of ServletConfig to get initialization parameter
6. Example of ServletConfig to get all the 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.

# Advantage of ServletConfig

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

## Methods of ServletConfig interface

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

---

## How to get the object of ServletConfig

1. **getServletConfig() method** of Servlet interface returns the object of ServletConfig.

### Syntax of getServletConfig() method

```
1. public ServletConfig getServletConfig();
```

### Example of getServletConfig() method

```
1. ServletConfig config=getServletConfig();
2. //Now we can call the methods of ServletConfig interface
```

---

## Syntax to provide the initialization parameter for a servlet

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

```
1. <web-app>
2.   <servlet>
3.     ......
4.
5.     <init-param>
6.       <param-name>parametername</param-name>
7.       <param-value>parametervalue</param-value>
8.     </init-param>
9.     ......
10.         </servlet>
11.       </web-app>
```

---

## Example of ServletConfig to get initialization parameter

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5. public class DemoServlet extends HttpServlet {
6. public void doGet(HttpServletRequest request, HttpServletResponse respon
   se)
7.     throws ServletException, IOException {
8.
9.     response.setContentType("text/html");
10.          PrintWriter out = response.getWriter();
11.
12.          ServletConfig config=getServletConfig();
13.          String driver=config.getInitParameter("driver");
14.          out.print("Driver is: "+driver);
15.
16.          out.close();
17.          }
18.
19.      }
```

**web.xml**

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>DemoServlet</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6.
7. <init-param>
8. <param-name>driver</param-name>
```

```
9.  <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
10.         </init-param>
11.
12.         </servlet>
13.
14.         <servlet-mapping>
15.         <servlet-name>DemoServlet</servlet-name>
16.         <url-pattern>/servlet1</url-pattern>
17.         </servlet-mapping>
18.
19.         </web-app>
```

download this example (developed in Myeclipse IDE)
download this example(developed in Eclipse IDE)
download this example(developed in Netbeans IDE)

## Example of ServletConfig to get all the initialization parameters

In this example, we are getting all the initialization parameter from the web.xml file and printing this information in the servlet.

**DemoServlet.java**

```
1.  import java.io.IOException;
2.  import java.io.PrintWriter;
3.  import java.util.Enumeration;
4.
5.  import javax.servlet.ServletConfig;
6.  import javax.servlet.ServletException;
7.  import javax.servlet.http.HttpServlet;
8.  import javax.servlet.http.HttpServletRequest;
9.  import javax.servlet.http.HttpServletResponse;
10.
```

```
11.
12.       public class DemoServlet extends HttpServlet {
13.       public void doGet(HttpServletRequest request, HttpServletResponse r
    esponse)
14.               throws ServletException, IOException {
15.
16.           response.setContentType("text/html");
17.           PrintWriter out = response.getWriter();
18.
19.           ServletConfig config=getServletConfig();
20.           Enumeration<String> e=config.getInitParameterNames();
21.
22.           String str="";
23.           while(e.hasMoreElements()){
24.           str=e.nextElement();
25.           out.print("<br>Name: "+str);
26.           out.print(" value: "+config.getInitParameter(str));
27.           }
28.
29.           out.close();
30.       }
31.
32.       }
```

**web.xml**

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>DemoServlet</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6.
7. <init-param>
8. <param-name>username</param-name>
```

```
9.  <param-value>system</param-value>
10.         </init-param>
11.
12.         <init-param>
13.         <param-name>password</param-name>
14.         <param-value>oracle</param-value>
15.         </init-param>
16.
17.         </servlet>
18.
19.         <servlet-mapping>
20.         <servlet-name>DemoServlet</servlet-name>
21.         <url-pattern>/servlet1</url-pattern>
22.         </servlet-mapping>
23.
24.         </web-app>
```

# ServletContext Interface


1. ServletContext Interface
2. Usage of ServletContext Interface
3. Methods of ServletContext interface
4. How to get the object of ServletContext
5. Syntax to provide the initialization parameter in Context scope
6. Example of ServletContext to get initialization parameter
7. Example of ServletContext to get all the initialization parameter


An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

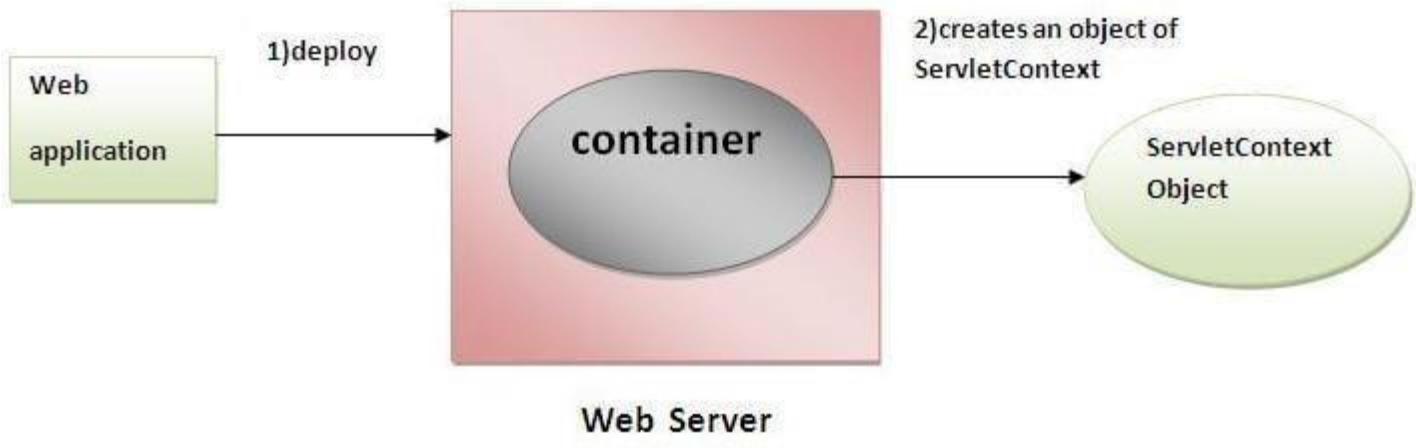
# Advantage of ServletContext

**Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

## Usage of ServletContext Interface

There can be a lot of usage of ServletContext object. Some of them are as follows:

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.




## Commonly used methods of ServletContext interface

There is given some commonly used methods of ServletContext interface.

1. **public String getInitParameter(String name):**Returns the parameter value
2. **public Enumeration getInitParameterNames():**Returns the names of the co
3. **public void setAttribute(String name,Object object):**sets the given object
4. **public Object getAttribute(String name):**Returns the attribute for the specif
5. **public Enumeration getInitParameterNames():**Returns the names of the co Enumeration of String objects.
6. **public void removeAttribute(String name):**Removes the attribute with the g

---

## How to get the object of ServletContext interface

1. **getServletContext() method** of ServletConfig interface returns the object of ServletContext.
2. **getServletContext() method** of GenericServlet class returns the object of ServletContext.

### Syntax of getServletContext() method

1. **public** ServletContext getServletContext()

### Example of getServletContext() method

```
1. //We can get the ServletContext object from ServletConfig object
2. ServletContext application=getServletConfig().getServletContext();
3.
4. //Another convenient way to get the ServletContext object
5. ServletContext application=getServletContext();
```

---

## Syntax to provide the initialization parameter in Context scope

The **context-param** element, subelement of web-app, is used to define the initializat
The param-name and param-value are the sub-elements of the context-param. The p
name and and param-value defines its value.

```
<web-app>
 ......

  <context-param>
    <param-name>parametername</param-name>
    <param-value>parametervalue</param-value>
  </context-param>
 ......
</web-app>
```

---

## Example of ServletContext to get the initialization parameter

In this example, we are getting the initialization parameter from the web.xml file and
parameter. Notice that the object of ServletContext represents the application scope.
parameter from the web.xml file, all the servlet classes will get the changed value. So
it is better to have the common information for most of the servlets in the web.xml fil
the simple example:

**DemoServlet.java**

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


public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
```

```
10.       res.setContentType("text/html");
11.       PrintWriter pw=res.getWriter();
12.
13.       //creating ServletContext object
14.       ServletContext context=getServletContext();
15.
16.       //Getting the value of the initialization parameter and printing it
17.       String driverName=context.getInitParameter("dname");
18.       pw.println("driver name is="+driverName);
19.
20.       pw.close();
21.
22.       }}
```

**web.xml**

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6. </servlet>
7.
8. <context-param>
9. <param-name>dname</param-name>
10.       <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
11.       </context-param>
12.
13.       <servlet-mapping>
14.       <servlet-name>sonoojaiswal</servlet-name>
15.       <url-pattern>/context</url-pattern>
16.       </servlet-mapping>
17.
18.       </web-app>
```

# Example of ServletContext to get all the initialization parameters

In this example, we are getting all the initialization parameter from the web.xml file. I used the getInitParameterNames() method in the servlet class.

**DemoServlet.java**

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5.
6. public class DemoServlet extends HttpServlet{
7. public void doGet(HttpServletRequest req,HttpServletResponse res)
8. throws ServletException,IOException
9. {
10.       res.setContentType("text/html");
11.       PrintWriter out=res.getWriter();
12.
13.       ServletContext context=getServletContext();
14.       Enumeration<String> e=context.getInitParameterNames();
15.
16.       String str="";
17.       while(e.hasMoreElements()){
18.          str=e.nextElement();
19.          out.print("<br> "+context.getInitParameter(str));
20.       }
21.       }}
```

**web.xml**

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
```

```
5. <servlet-class>DemoServlet</servlet-class>
6. </servlet>
7.
8. <context-param>
9. <param-name>dname</param-name>
10.     <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
11.     </context-param>
12.
13.     <context-param>
14.     <param-name>username</param-name>
15.     <param-value>system</param-value>
16.     </context-param>
17.
18.     <context-param>
19.     <param-name>password</param-name>
20.     <param-value>oracle</param-value>
21.     </context-param>
22.
23.     <servlet-mapping>
24.     <servlet-name>sonoojaiswal</servlet-name>
25.     <url-pattern>/context</url-pattern>
26.     </servlet-mapping>
27.
28.     </web-app>
```

# Session tracking

- HTTP is a stateless protocol that provides no way for a server to recognize that a sequence of requests comm. From the same client.
- But many web applications are not stateless.
- A server can not identify the client by the ip address,

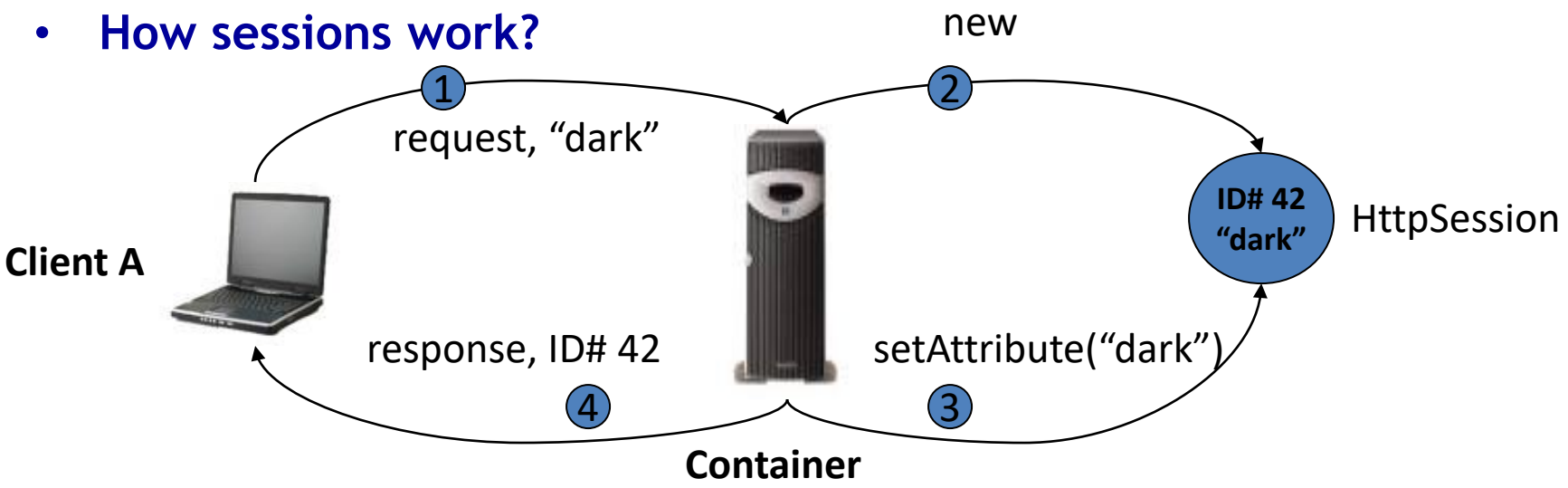
  because the reported ip address may be the address of a proxy server or the address of a server machine that hosts several clients.
- Session tracking gives servlets and other server-side applications the ability to keep track of the user as the user moves through the site.
- Web server maintains user state by creating a session object for each user with a unique session id.
- The session object is passed as part of the request.
- Servlets can add information to session object and read information from them.

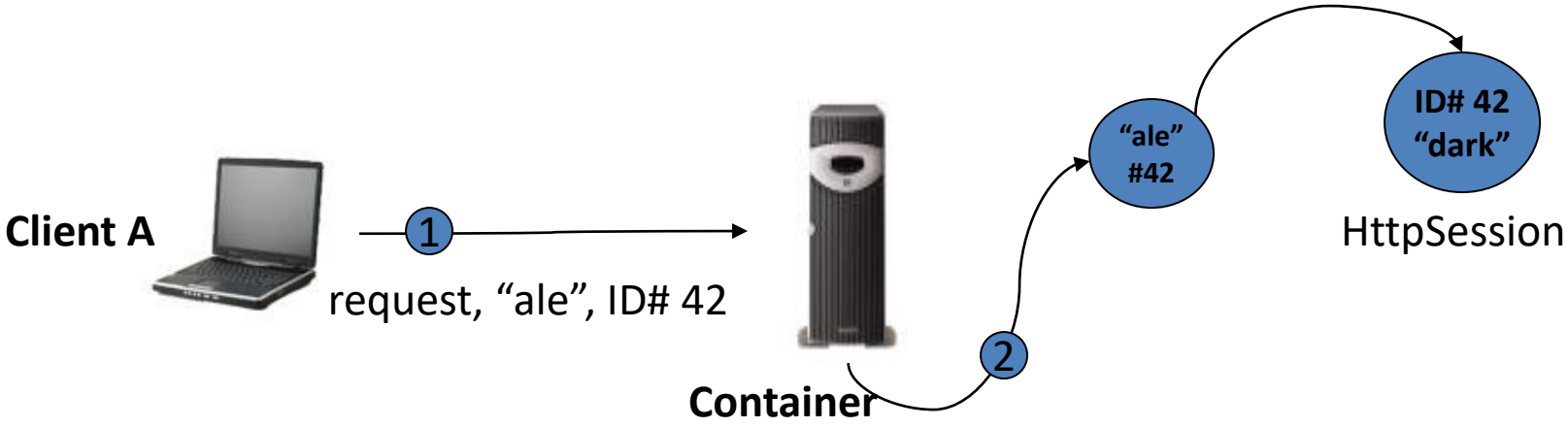
# What is Session Tracking?

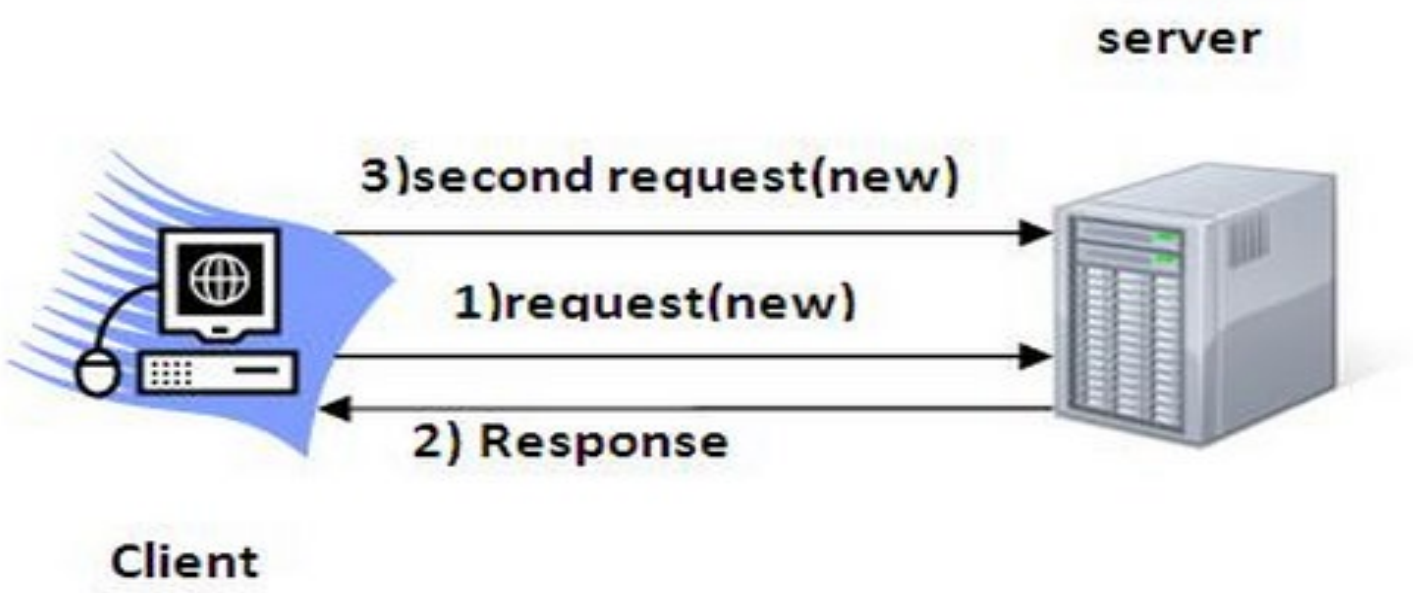
- HTTP is a "stateless" protocol.
- when you are doing on-line shopping,
- the Web server can't easily remember previous transactions.
- This makes applications like shopping carts very problematic:
- when you add an entry to your cart,
- how does the server know what's already in your cart?
- When you move from the page where you specify what you want to buy
- (hosted on the regular Web server)
- to the page that takes your credit card number and
- shipping address (hosted on the secure server that uses SSL), how does the server remember what you were buying?

# Session Management - Session Tracking

- How sessions work?





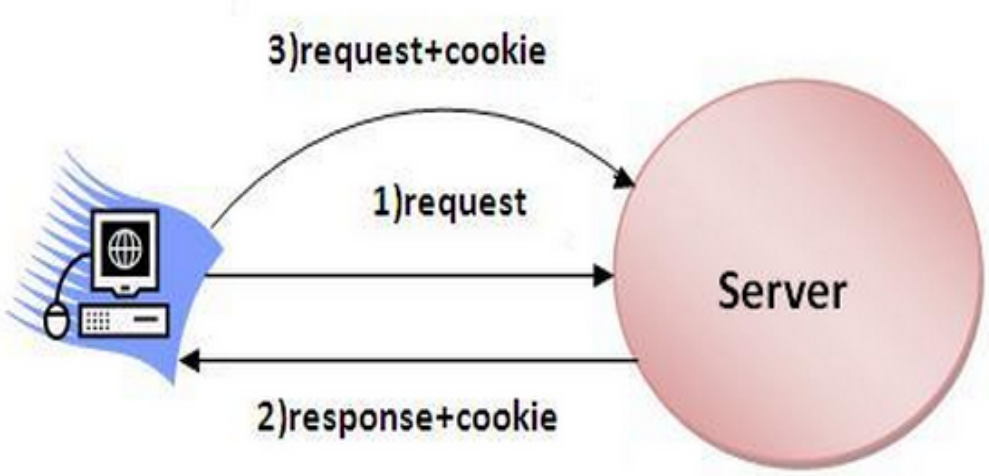

## Why use Session Tracking?

To recognize the user.

# There are four typical solutions to this problem.

- 1. **Cookies.**
- 2. **URL Rewriting.**
- 3. **Hidden form fields.**
- 4. HTTP Session

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.



## javax.servlet.http.Cookie class

**javax.servlet.http.Cookie** class provides the functionality of using cookies.

## Constructor of Cookie class

**Cookie(String name, String value):** Constructs a cookie with a specified name and value.

## Commonly used methods of Cookie class

There are given some commonly used methods of the Cookie class.

1. **public void setMaxAge(int expiry):**Sets the maximum age of the cookie in seconds.
2. **public String getName():**Returns the name of the cookie. The name cannot be changed after creation.
3. **public String getValue():**Returns the value of the cookie.

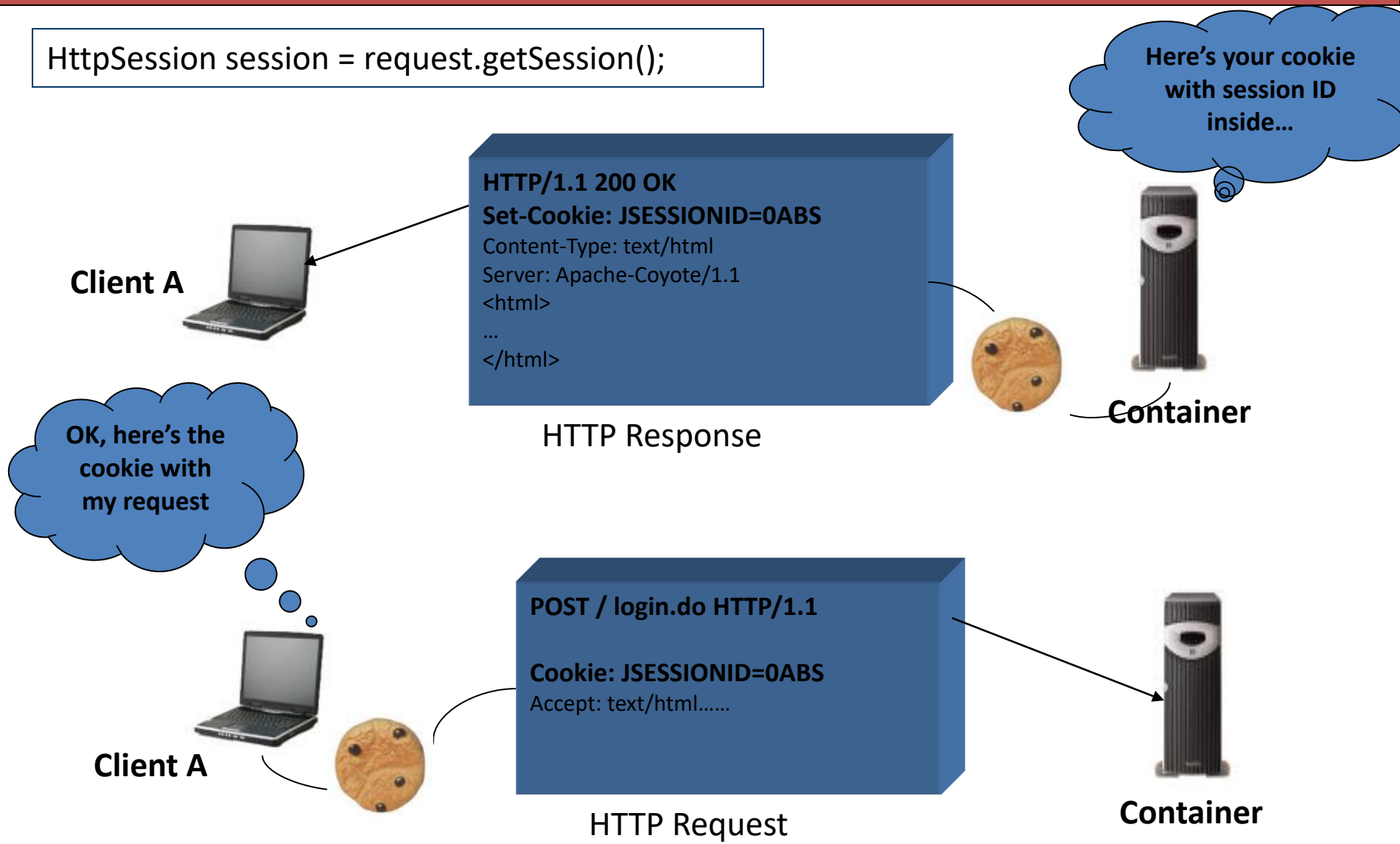
## Other methods required for using Cookies

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.

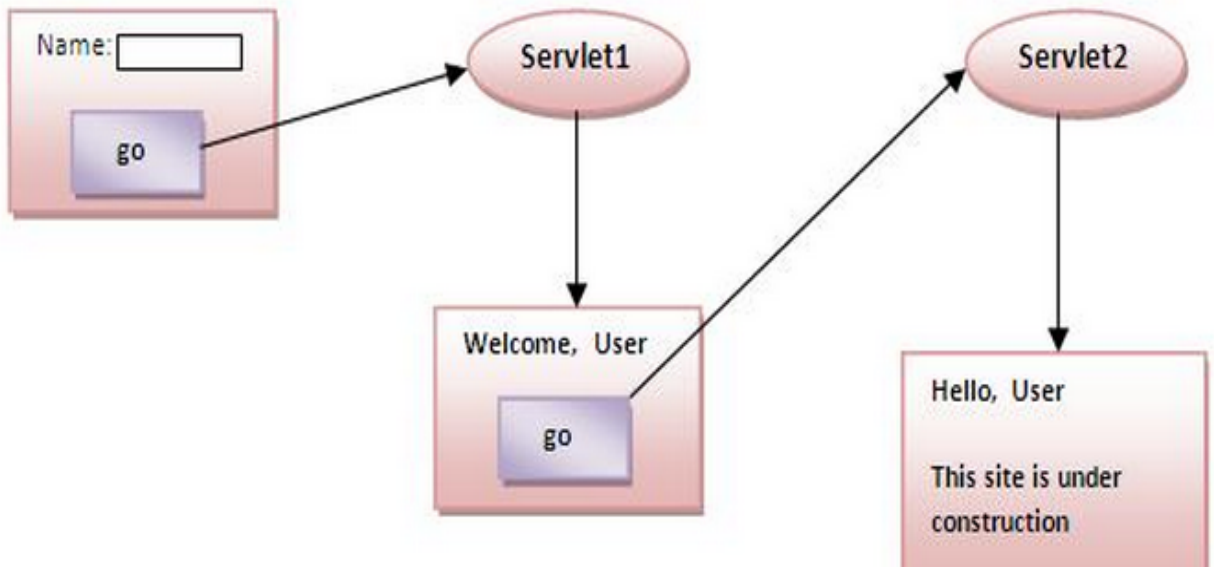
# Session Tracking - Cookies

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



## ✓*Example of using Cookies*

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



## index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

# First servlet

```
public void doPost(HttpServletRequest request, HttpServletResponse response){
      try{

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

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

      Cookie ck=new Cookie("uname",n);//creating cookie object
      response.addCookie(ck);//adding cookie in the response

      //creating submit button
      out.print("<form action='servlet2'>");
      out.print("<input type='submit' value='go'>");
      out.print("</form>");

      out.close();

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

# Second servlet

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

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
        try{

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

        Cookie ck[]=request.getCookies();
        out.print("Hello "+ck[0].getName());

        out.close();

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

# Web.xml

```
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
```

## Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

## Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

# URL Rewriting.

1. You can append some extra data on the end of each URL that identifies the session,
2. and the server can associate that session identifier with data it has stored about that session.
3. This is also an excellent solution, and
4. even has the advantage that it works with browsers that don't support cookies or where the user has disabled cookies.
5. However, it has most of the same problems as cookies,
6. namely that the server-side program has a lot of straightforward but tedious processing to do.
7. In addition, you have to be very careful that every URL returned to the user
8. (even via indirect means like Location fields in server redirects) has the extra information appended.
9. And, if the user leaves the session and comes back via a bookmark or link, the session information can be lost.

# Session tracking-url rewriting

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.

# Session Tracking - URL Rewriting

**URL + ;jsessionid=1234567**

**Client A**

**HTTP/1.1 200 OK**
Content-Type: text/html
Server: Apache-Coyote/1.1
<html>
<body>
**< a href =" http:// www.sharmanj.com/Metavante;jsessionid=0AAB">**
**click me </a>**
</html>

HTTP Response

**Container**

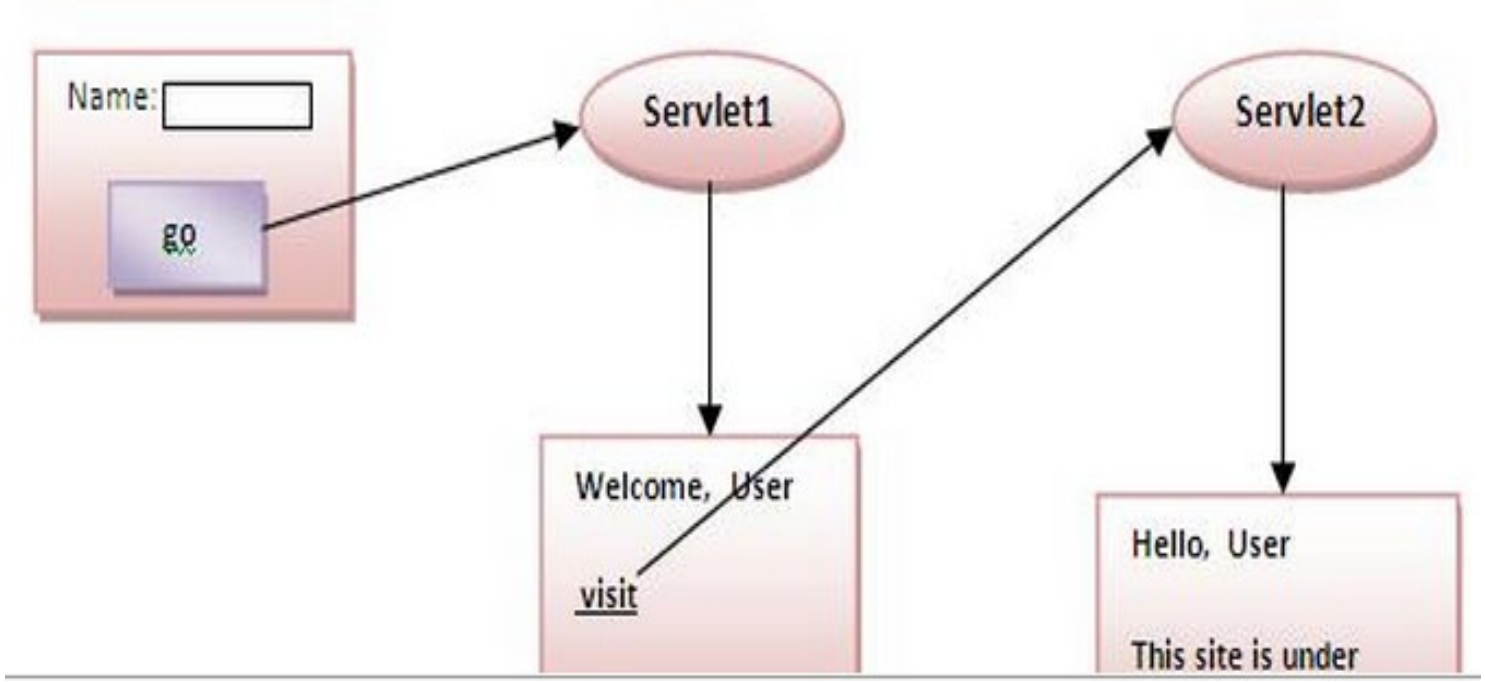
**GET /Metavante;jsessionid=0AAB**

HTTP / 1.1
Host: www.sharmanj.com
Accept: text/html

HTTP Request

**Client A**

**Container**


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

## ✓*Example of using URL Rewriting*

In this example, we are maintaning the state of the user using link. For this purpose, we are appending the name of the user in the query string and getting the value from the query string in another page.

### index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

# First servlet

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


public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
                try{

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

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

                //appending the username in the query string
                out.print("<a href='servlet2?uname="+n+"'>visit</a>");

                out.close();

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

# second servlet

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

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
                try{

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

                //getting value from the query string
                String n=request.getParameter("uname");
                out.print("Hello "+n);

                out.close();

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

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

## 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.

# Hidden form fields.

HTML forms have an entry that looks like the following:

<INPUT TYPE="HIDDEN" NAME="session" VALUE="...">.

when the form is submitted,

the specified name and value are included in the GET or POST data.

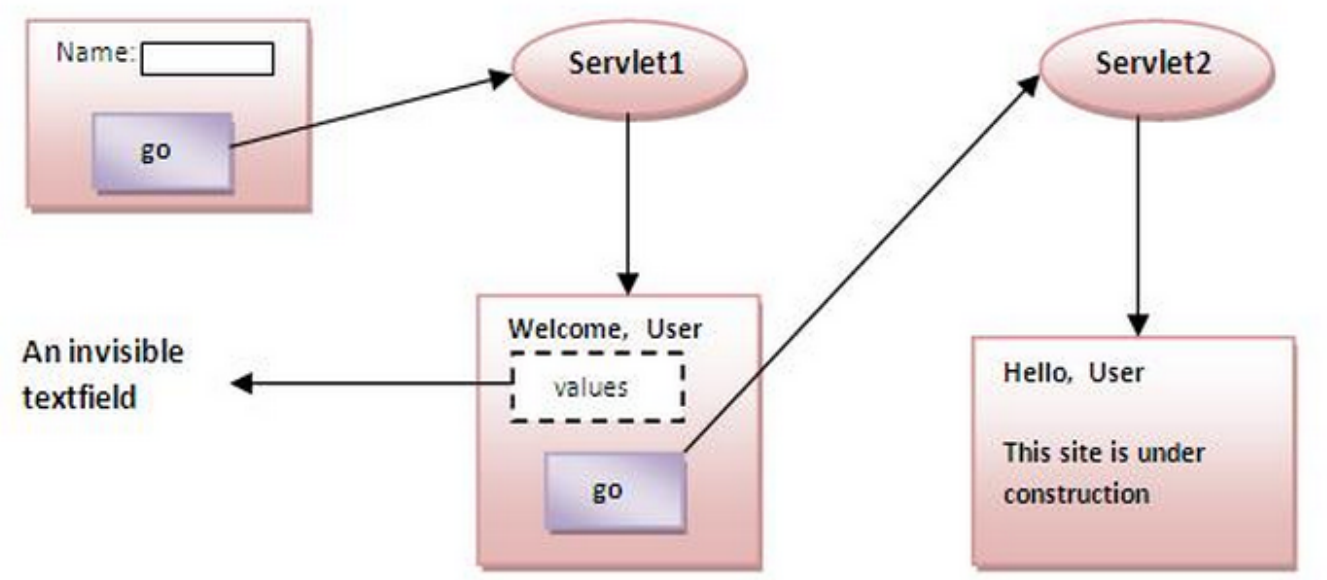
This can be used to store information about the session. Disadvantage

that it only works if every page is dynamically generated,

since the whole point is that each session has a unique identifier.

# 3.Hidden form field

In case of Hidden Form Field an invisible textfield is used for maintaing the state of an user. In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.



## Advantage of Hidden Form Field

1. It will always work whether cookie is disabled or not.

## Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

## ✓Example of using Hidden Form Field

In this example, we are storing the name of the user in a hidden textfield and getting that value from another servlet.

### index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

```
public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
                try{

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

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

                //creating form that have invisible textfield
                out.print("<form action='servlet2'>");
                out.print("<input type='hidden' name="uname" value='"+n+"'>");
                out.print("<input type='submit' value='go'>");
                out.print("</form>");

                out.close();

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

```
public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
                try{

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

                //Getting the value from the hidden field
                String n=request.getParameter("uname");
                out.print("Hello "+n);

                out.close();

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


}
```

# Web.xml

```
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
```

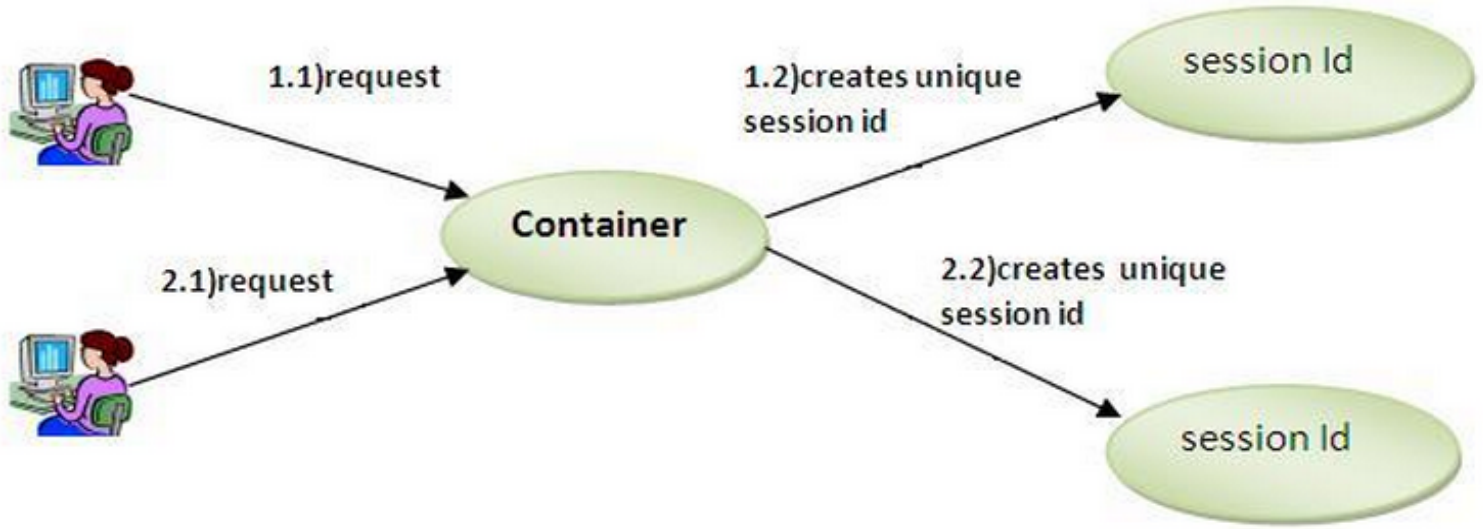
# HttpSession API.

- This is a high-level interface built on top of cookies or URL-rewriting.
- In fact, on many servers, they use cookies if the browser supports them,
- but automatically revert to URL-rewriting when cookies are unsupported or explicitly disabled. But the servlet author doesn't need to bother with many of the details,
- doesn't have to explicitly manipulate cookies or information appended to the URL, and
- is automatically given a convenient place to store data that is associated with each session.

# 4.HttpSession

In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



## How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

## Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

## ✓Example of using HttpSession

In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the setAttribute() method of HttpSession interface and to get the attribute, we have used the getAttribute method.

### index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

# First servlet

```
public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
                try{

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

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

                HttpSession session=request.getSession();
                session.setAttribute("uname",n);

                out.print("<a href='servlet2'>visit</a>");

                out.close();

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

}
```

# Second servlet

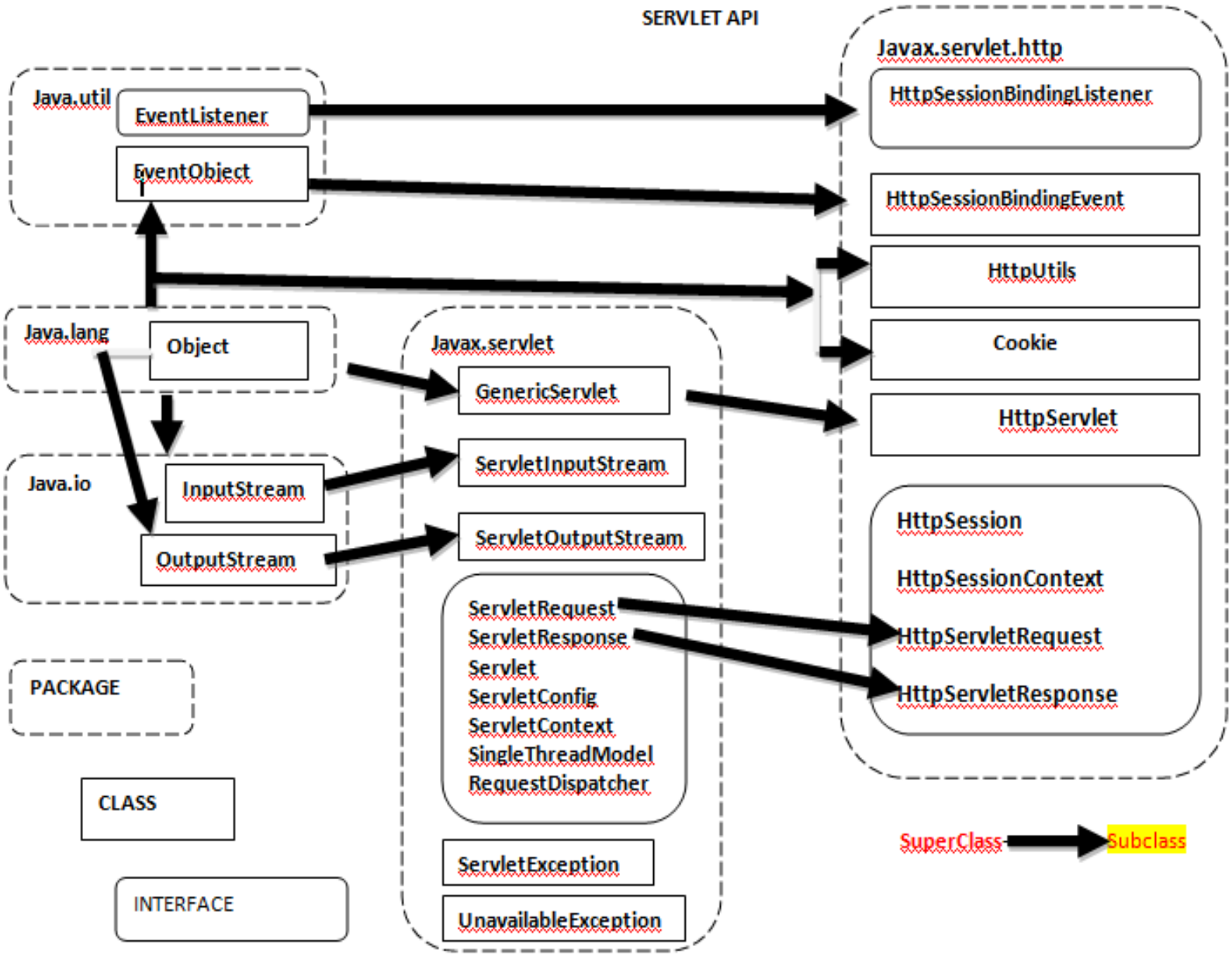
```
public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
                try{

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

                HttpSession session=request.getSession(false);
                String n=(String)session.getAttribute("uname");
                out.print("Hello "+n);

                out.close();

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


}
```

```
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
```

# JSP-JAVA SERVER PAGES

# Syllabus…

- **Servlets:** Introduction
- Web application Architecture
- Http Protocol & Http Methods
- Web Server & Web Container
- Servlet Interface
- GenericServlet
- HttpServlet
- Servlet Life Cycle
- ServletConfig
- ServletContext
- Servlet Communication
- Session Tracking Mechanisms


SERVLET API
Javax.servlet.http
Java.util
EventListener
EventObject
HttpSessionBindingListener
HttpSessionBindingEvent
HttpUtils
Cookie
Java.lang
Object
Javax.servlet
GenericServlet
HttpServlet
Java.io
InputStream
OutputStream
ServletInputStream
ServletOutputStream
HttpSession
HttpSessionContext
HttpServletRequest
HttpServletResponse
ServletRequest
ServletResponse
Servlet
ServletConfig
ServletContext
SingleThreadModel
RequestDispatcher
PACKAGE
CLASS
INTERFACE
ServletException
UnavailableException
SuperClass
Subclass

# jsp

- **JSP:** Introduction
- JSP LifeCycle
- JSP Implicit Objects & Scopes
- JSP Directives
- JSP Scripting Elements
- JSP Actions: Standard actions and customized actions

# JSP Overview - History

- Earlier dynamic web pages were developed with CGI
  - Web Applications eventually outgrew CGI because of the following:
    - Datasets became large
    - Increased traffic on sites
    - Trouble maintaining state and session information
    - Performance bottlenecks
    - Can involve proprietary APIs

# JSP Overview - Software

- With a foundation in Java, JSP
  - Threads incoming requests
  - Supports the definition of tags that abstract functionality within a page in a library
  - Interacts with a wide array of databases
  - Serves dynamic content in a persistent and efficient manner
    - Initially the server automatically separates HTML and JSP code, compiles the JSP code, and starts the servlet, from then on the servlets are stored in the web server's memory in Java byte code
  - Extends the same portability, large class library, object-oriented code methodology, and robust security measures that have made Java a top choice
  - Does not require a Java programmer

# JSP

- JSP is another technology for developing web applications.
- It is a template for web page that uses java code to generate an HTML document dynamically
- It is not meant to replace servlet.
- JSP is an extension of servlet technology, & it is common practice to use both in same web applications.
- JSP  run in server-side component known as a JSP container which translates them into equivalent Servlet
- JSP uses the same techniques as those found in servlet programming.

# Java Server Pages(JSP)

- JSP pages typically comprise of:
- Static HTML/XML components.
- Special JSP tags
- Optionally, snippets of code written in the Java programming language called "scriptlets."

# JSP Advantages

- Separation of static from dynamic content.
- Write Once Run Anywhere
- Dynamic content can be served in a variety of formats.
- Recommended Web access layer for n-tier architecture.

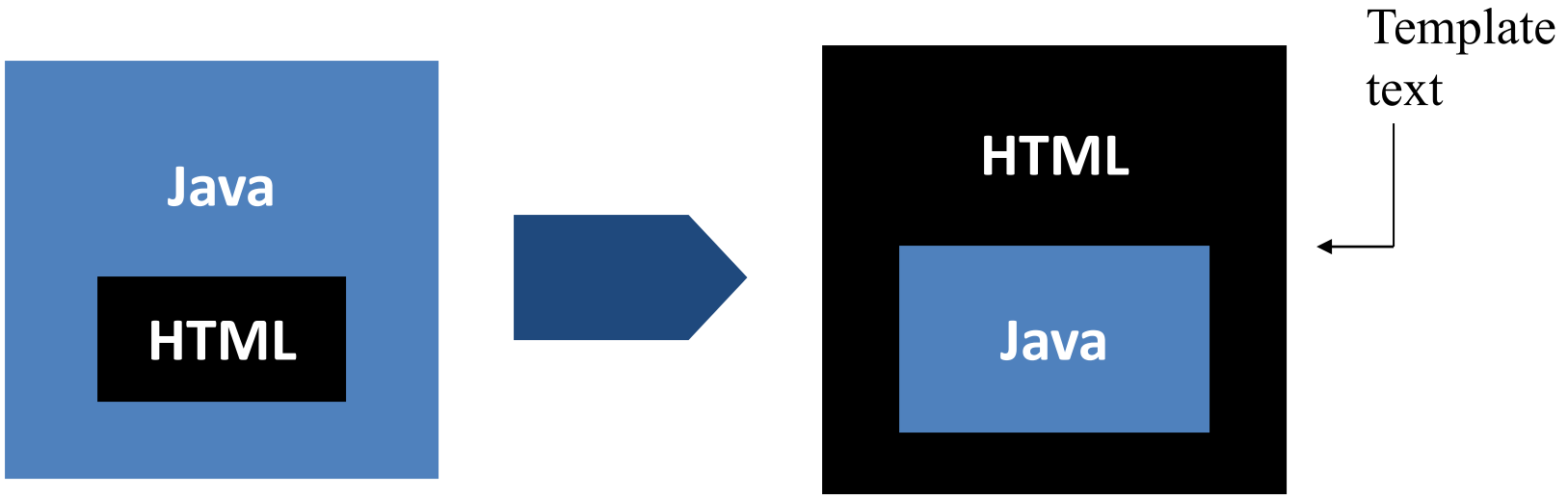
  Completely leverages the Servlet API.
- JSP uses simplified scripting language based syntax for embedding HTML into JSP.
- JSP containers provide easy way for accessing standard objects and actions.
- JSP reaps all the benefits provided by JAVA servlets and web container environment, but they have an added advantage of being simpler and more natural program for web enabling enterprise developer
- JSP use HTTP as default request /response communication paradigm and thus make JSP ideal as Web Enabling Technology.

# Why Use JSP

- JSP is built on top of servlets, so it has all the advantages of servlets
- JSP is compiled into its corresponding servlet when it is requested at the first time
- The servlet stays in memory, speeding response times
- Extensive use of Java API (Applications Programming Interface) for networking, database access, distributed objects, and others like it
- Powerful, reusable and portable to other operating systems and Web servers
- servlets is a programmatic technology requiring significant developer expertise,
- JSP can be used not only by developers, but also by page designers
- JSP is the inherent separation of presentation from content facilitated

# Relationships

- In servlets,
  - HTML code is printed from java code
- In JSP pages
  - Java code is embadded in HTML code

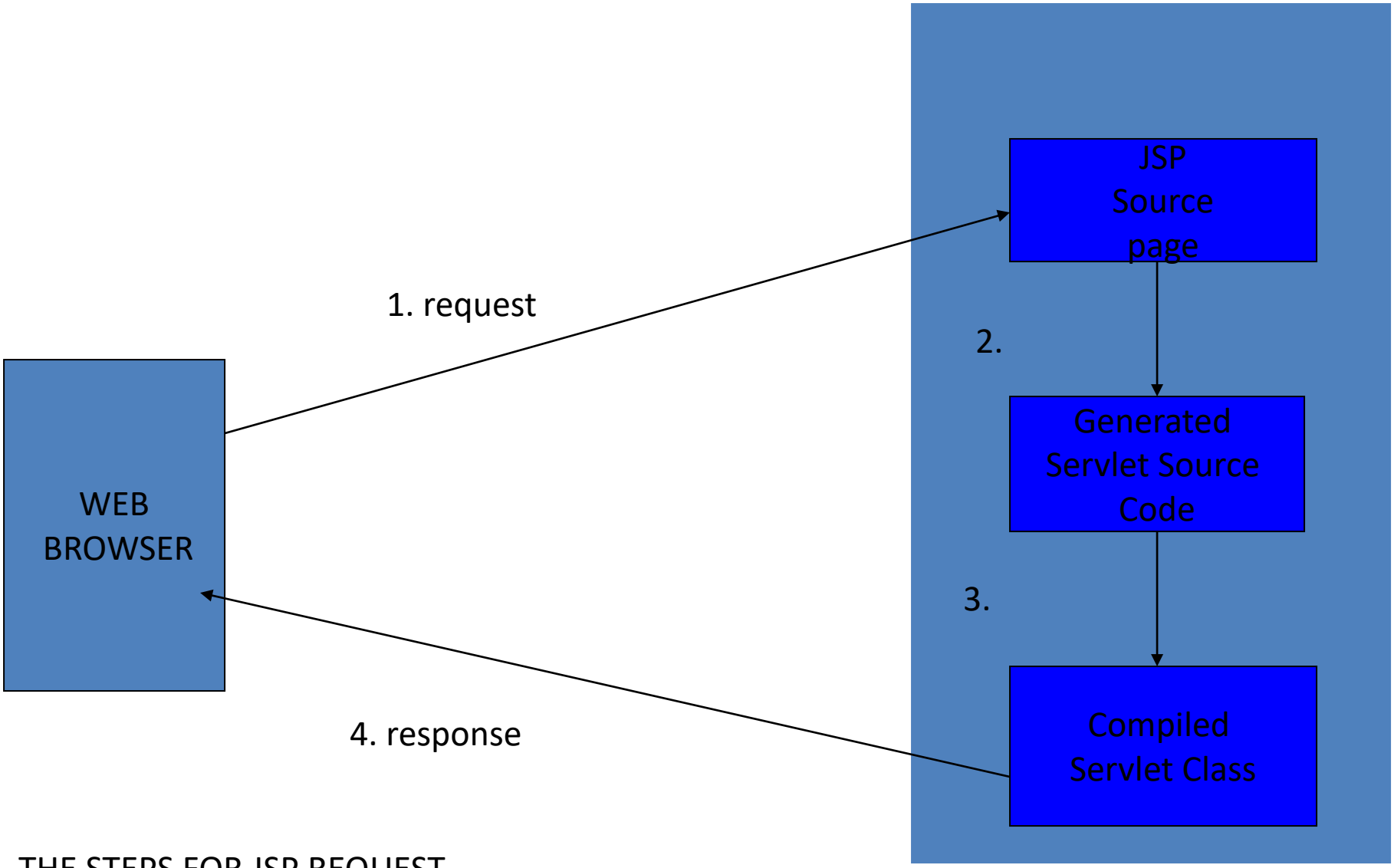


# JSP

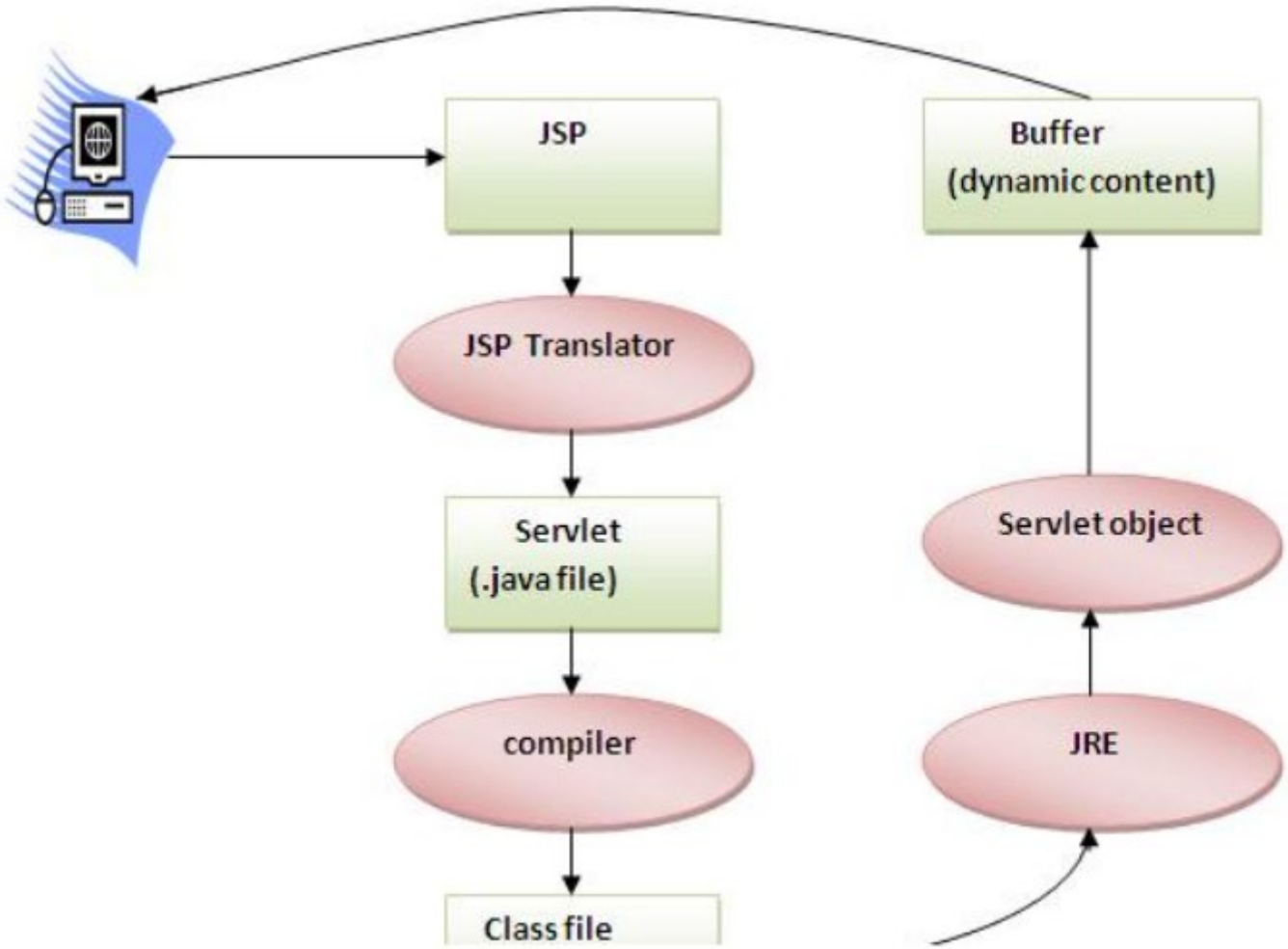
- They are automatically recompiled when necessary
- They exist in web server document space, no special URL mapping is required to address them
- They are like HTML pages ,they have compatibility with web development tools

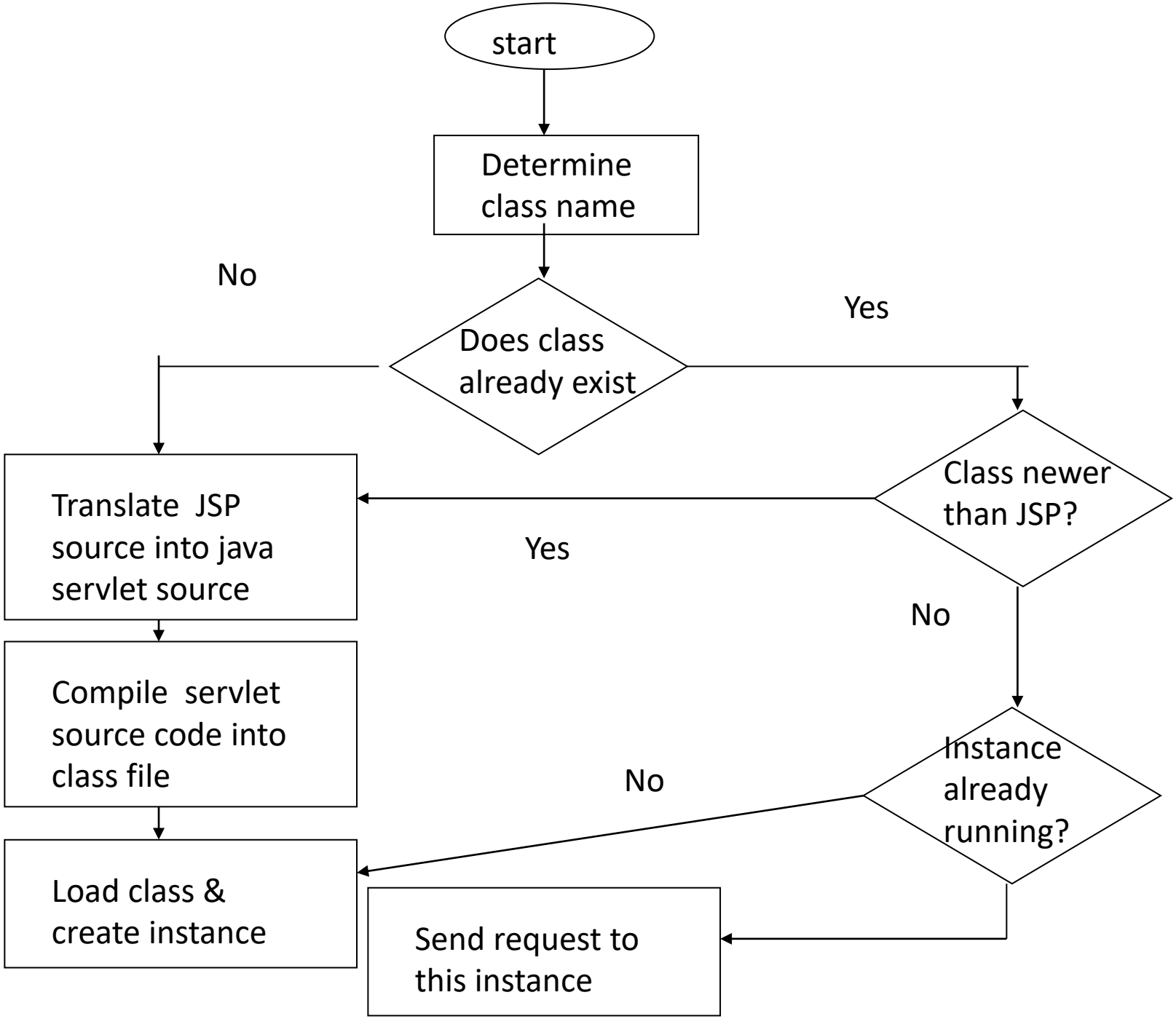
# How it works

- Request for JSP , container determines name of class
- If it does not exist or older one then it creates Java Source Code
- Compiles it , loads its instance  if not  loaded
- If JSP file is modified then container retranslates and recompiles it



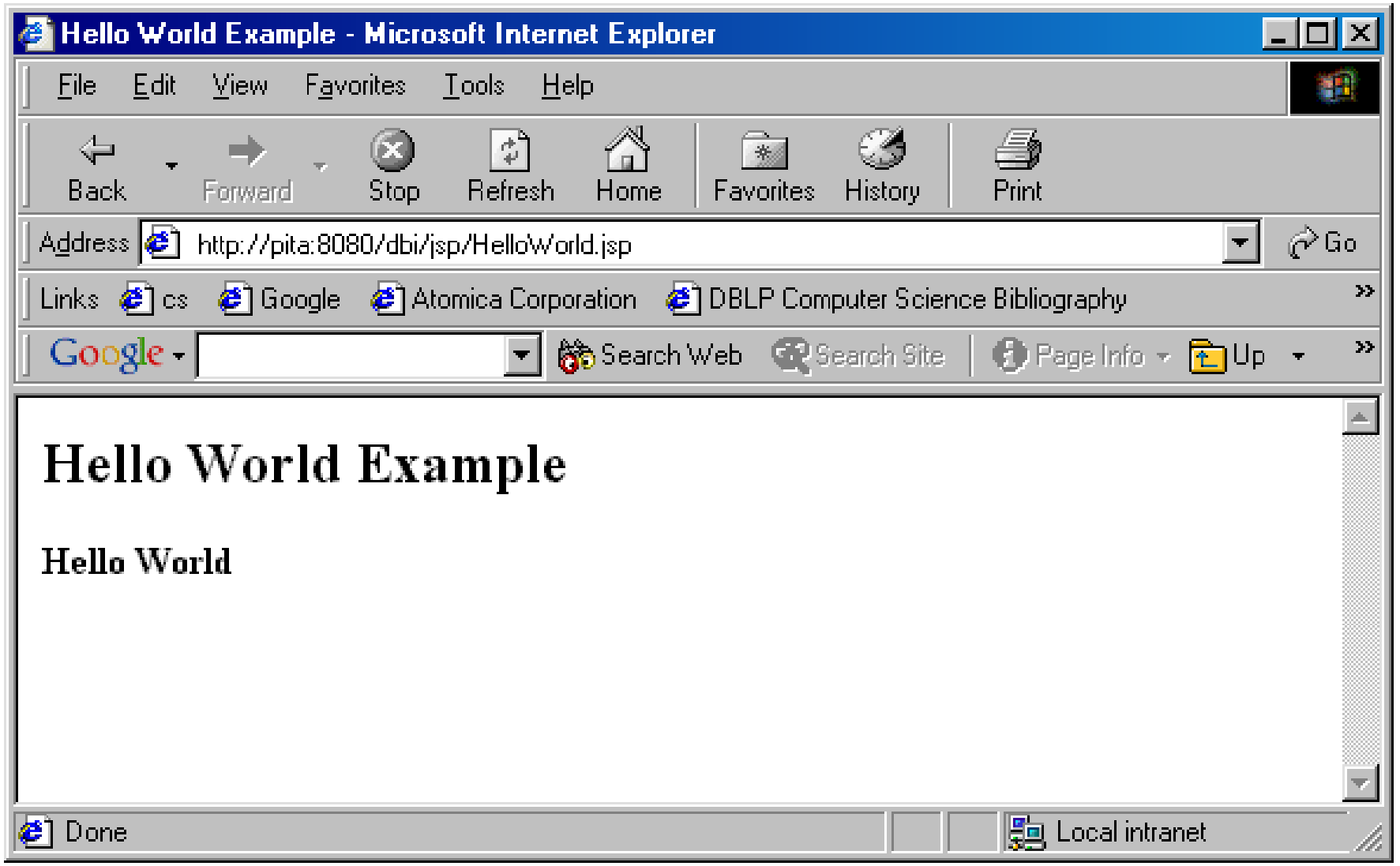

THE STEPS FOR JSP REQUEST


JSP
Buffer
(dynamic content)
JSP Translator
Servlet
(.java file)
Servlet object
compiler
JRE
Class file


start
Determine class name
No
Does class already exist
Yes
Translate JSP source into java servlet source
Class newer than JSP?
Yes
No
Compile servlet source code into class file
Instance already running?
No
Load class & create instance
Send request to this instance

# An Example

```
<HTML>
<HEAD>
<TITLE>Hello World Example</TITLE>
</HEAD>

<BODY>
<H2>Hello World Example</H2>
<B><% out.println("Hello World"); %></B>
</BODY>
</HTML>
```


Hello World Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Favorites History Print
Address http://pita:8080/dbi/jsp/HelloWorld.jsp
Go
Links cs Google Atomica Corporation DBLP Computer Science Bibliography
Google Search Web Search Site Page Info Up
Hello World Example
Hello World
Done
Local intranet

# JSP Architecture

- The JSP page implementation class file extends HttpJspBase,
- which in turn implements the Servlet interface.
- the service method of this class, _jspService(), essentially inlines the contents of the JSP page.
- Although _jspService() cannot be overridden,
- the developer can describe initialization and
- destroy events by providing implementations for the jspInit() and jspDestroy() methods within their JSP pages.

- Once this class file is loaded within the servlet container,
- the _jspService() method is responsible for replying to a client's request.
- By default, the _jspService() method is dispatched on a separate thread
- by the servlet container in processing concurrent client requests,

# JSP -API

The JSP API consists of two packages:

1. javax.servlet.jsp
2. javax.servlet.jsp.tagext


- The JSP API
- javax.servlet.jsp package
  - The JspPage interface
  - The HttpJspPage interface


## javax.servlet.jsp package

The javax.servlet.jsp package has two interfaces and classes.The two interfaces are as follows:

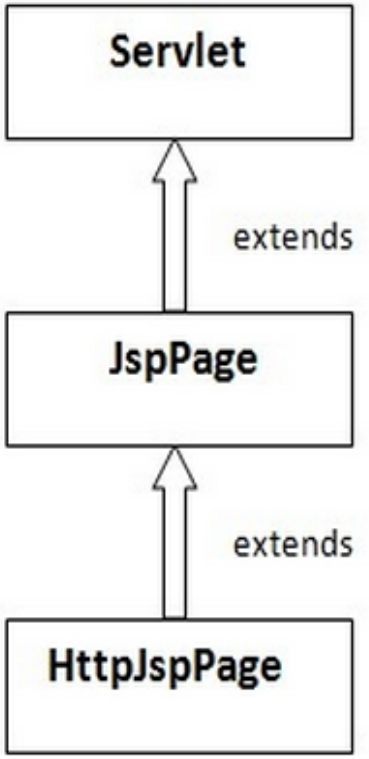
1. JspPage
2. HttpJspPage

The classes are as follows:

- JspWriter
- PageContext
- JspFactory
- JspEngineInfo
- JspException
- JspError

# The JspPage interface

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.



## Methods of JspPage interface

1. **public void jspInit():** It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.
2. **public void jspDestroy():** It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

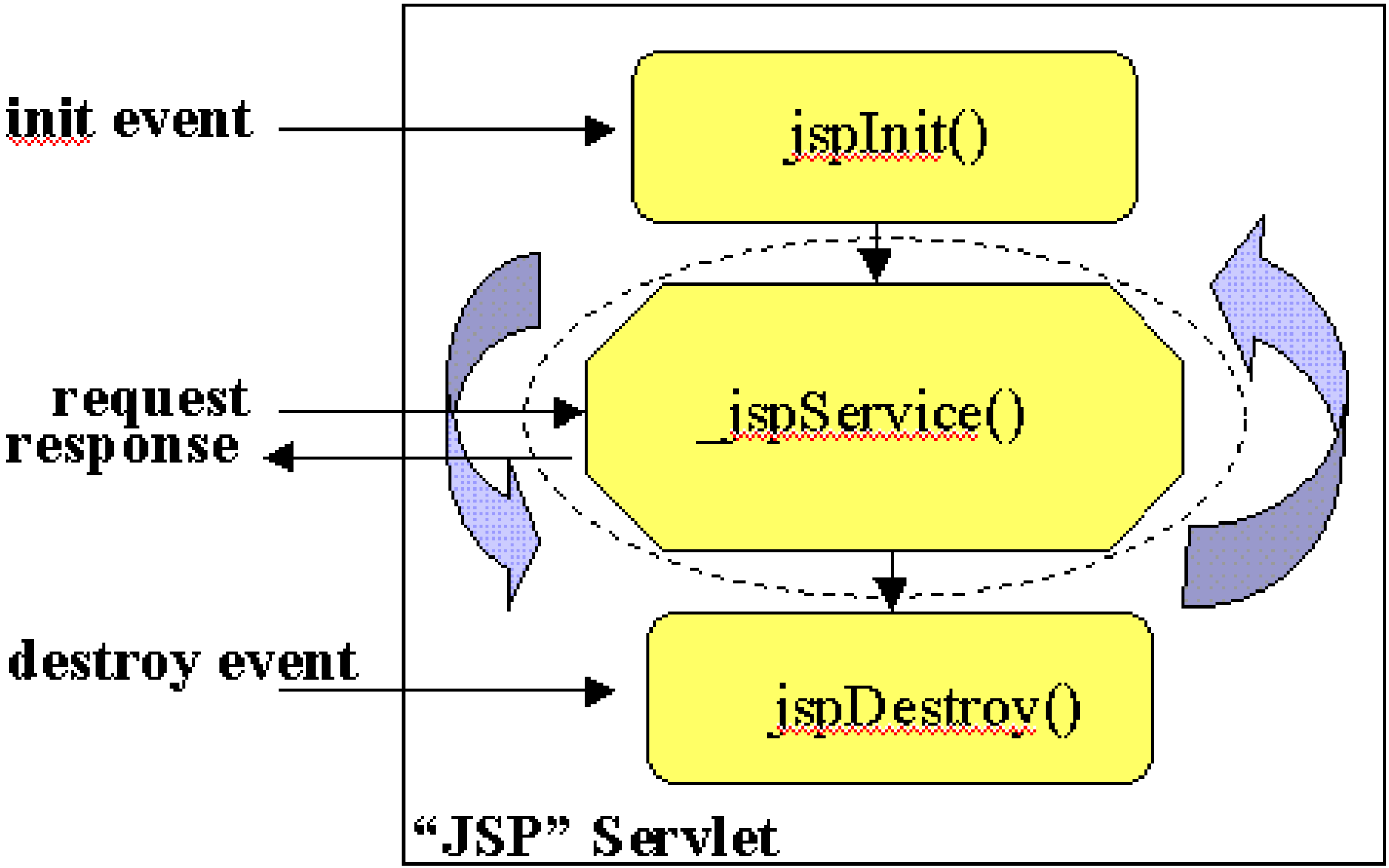
---

# The HttpJspPage interface

The HttpJspPage interface provides the one life cycle method of JSP. It extends the JspPage interface.

## Method of HttpJspPage interface:

1. **public void _jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore _ signifies that you cannot override this method.


init event
jspInit()
request
response
_jspService()
destroy event
jspDestroy()
"JSP" Servlet

# Component of JSP

- JSP file can contain JSP elements, fixed template data or both .
- The elements which are called jsp tags make up the syntax & semantics of jsp.
- Template data includes parts that the jsp container does not understand such as html tags.

# Introduction - JSP v/s Servlet

## How is JSP different / similar to Servlet?

<table>
<tr><th>Servlets</th><th>JSP</th></tr>
<tr><td colspan="2">Handles dynamic data</td></tr>
<tr><td>Handles business logic</td><td>Handles presentation logic</td></tr>
<tr><td>Lifecylce methods<br>init() : can be overridden<br>service() : can be overridden<br>destroy() : can be overridden</td><td>Lifecylce methods<br>jspInit() : can be overridden<br>_jspService() : cannot be overridden<br>jspDestroy() : can be overridden</td></tr>
<tr><td>Html within java<br>out.println(“<htrml><body>”);<br>out.println(“Time is” + new Date());<br>out.println(“</body></html>”);</td><td>Java within html<br><html><body><br>Time is <%=new Date()%><br></body></html></td></tr>
<tr><td colspan="2">Runs within a Web Container</td></tr>
</table>

# Introduction - Comments

- **JSP Comments**
  - <%-- {CODE HERE} --%>
  - Does not show the comments on the page
  - Does not show the comments in page source
  - Can only be used outside JSP Elements
- **HTML Comments**
  - <!- {CODE HERE} -->
  - Does not show the comments on the page
  - Shows the comments in page source
  - Can only be used outside JSP Elements
- **Single line Comments**
  - // {CODE HERE}
  - When put outside JSP Elements, shows comments on the page & source
  - When put inside Scriptlets/Declarations, does not show on page & source
  - Can be used inside scriptlets/declarations and outside JSP Elements

# JSP Elements - Intro

- Need to write some Java in your HTML?
- Want to make your HTML more dynamic?

- **JSP Declarations** :: Code that goes outside the service method
- **JSP Scriptlets** :: Code that goes within the service method
- **JSP Expressions** :: Code inside expressions is evaluated
- **JSP Directives** :: Commands given to the JSP engine

# JSP Scriptlet tag (Scripting elements)

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

## Scripting elements

The scripting elements provides the ability to insert java code inside the jsp. scripting elements:

- scriptlet tag
- expression tag
- declaration tag

---

# JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

```
<%  java source code %>
```

## ✓*Simple Example of JSP scriptlet tag*

In this example, we are displaying a welcome message.

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

# JSP expression tag

The code placed within 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.

## Syntax of JSP expression tag

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

## ✓Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

```
<html>
<body>
<%= "welcome to jsp" %>
</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.

## Syntax of JSP declaration tag

The syntax of the declaration tag is as follows:

```
<%!  statement %>
```

```
<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>
```

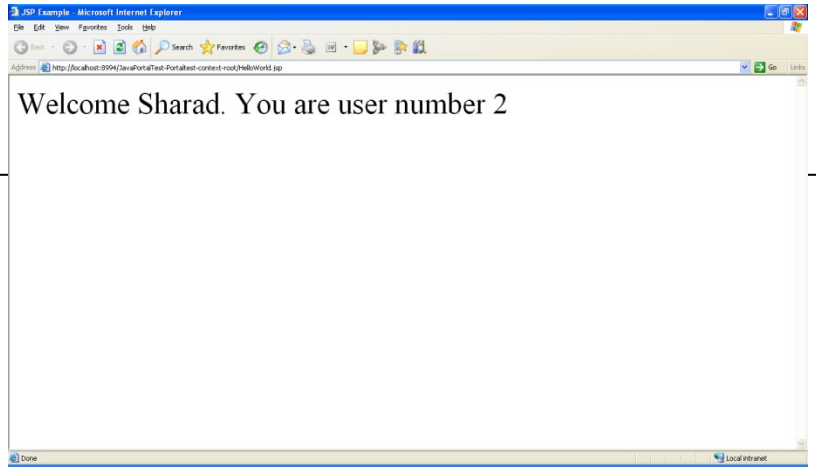
# JSP Elements - Example

```
<html><head><title>JSP Elements Example</title>
 </head>
 <body>
  <%! int userCnt = 0; %>

  <%
    String name = "Sharad";
    userCnt++;
  %>
   <table>
    <tr><td>
      Welcome <%=name%>. You are user number <%=userCnt%>
    </td></tr>
   </table>
 </body>
</html>
```

Declarations

Scriptlets

Expressions



# JSP Implicit Objects

There are 9 implicit objects available for the JSP page. The Auto Generated Servlet contains many objects like out, request, config, session, application etc. The 9 implicit objects are as follows:

| JSP Implicit Object | Super Class | Annotation |
|---|---|---|
| request | HttpServletRequest | Provides HTTP request information. |
| response | HttpServletResponse | Send data back to the client |
| out | JspWriter | Write the data to the response stream |
| session | HttpSession | Track information about a user from one request to another |
| application | ServletContext | Data shared by all JSPs and servlets in the application. |
| pageContext | PageContext | Contains data associated with the whole page |
| config | ServletConfig | Provides servlet configuration data. |
| page | Object | Similar with “this” in Java |
| exception | Throwable | Exceptions not caught by application code. |

## 1) out implicit object

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

```
PrintWriter out=response.getWriter();
```

But in JSP, you don't need to write this code.

### ✓*Example of out implicit object*

In this example we are simply displaying date and time.

### index.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
```

# 2) request implicit object:

In JSP, request is an implicit object of type HttpServletRequest.

## Example of request implicit object:

### index.html

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

## welcome.jsp

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

# 3) response implicit object:

In JSP, response is an implicit object of type HttpServletResponse.

## Example of response implicit object:

### index.html

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

## welcome.jsp

```
<html>
<body>
<%

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

%>
</body>
</html>
```

# Remaining implicit objects

## 4) config implicit object:

In JSP, config is an implicit object of type ServletConfig. This object can be used to get configuration information for a particular JSP page.This variable information can be used for one jsp page only.

## welcome.jsp

```
<html>
<body>
<%

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

String driver=config.getParameter("dname");
out.print("driver name is="+driver);


%>
</body>
</html>
```

# 5) application implicit object:

In JSP, application is an implicit object of type ServletContext.This object can be used to get configuration information from configuaration file(web.xml). This variable information can be used for all jsp pages.

## Example of application implicit object:

## index.html

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

## welcome.jsp

```
<html>
<body>
<%

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

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

%>
</body>
</html>
```

# 6) session implicit 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.

## Example of session implicit object:

## index.html

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

## welcome.jsp

```
<html>
<body>
<%

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

session.setAttribute("user",name);

<a href="second.jsp">second jsp page</a>

%>
</body>
</html>
```

## second.jsp

```
<html>
<body>
<%

String name=session.getAttribute("user");
out.print("Hello "+name);

%>
</body>
</html>
```

# 7) pageContext implicit 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:

- page
- request
- session
- application

In JSP, page scope is the default scope.

## index.html

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

## welcome.jsp

```
<html>
<body>
<%

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

pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);

<a href="second.jsp">second jsp page</a>

%>
</body>
</html>
```

## second.jsp

```
<html>
<body>
<%

String name=pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
out.print("Hello "+name);

%>
</body>
</html>
```

# 8) page implicit object:

In JSP, page is an implicit object of type Object class.This object is assigned to the reference of auto generated servlet class. It is written as:

Object page=this;

For using this object it must be cast to Servlet type.For example:

<% (HttpServlet)page.log("message"); %>

Since, it is of type Object it is less used because you can use this object directly in jsp.For example:

<% this.log("message"); %>

# Server Side Development

## Java Server Pages

# Topics

- JSP Fundamentals
- JSP Scripting Elements
- JSP Implicit Objects
- JSP Directives
- JSP Actions
- JSP Example (Loan Calculator)
- Servlets & JSPs together
- Tag Libraries
- Deploying and Running a JSP Application

# Java Server Pages (JSP)

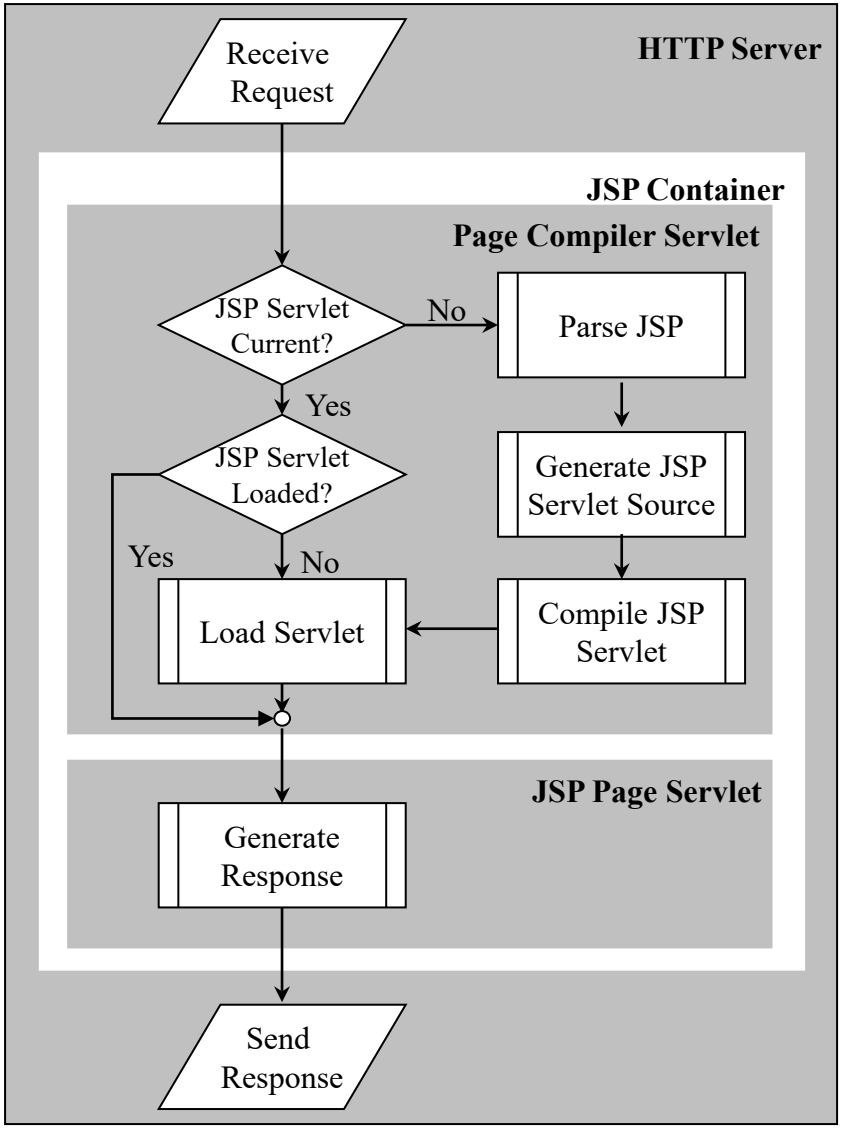
## Fundamentals

- Java Server Pages are HTML pages embedded with snippets of Java code.
  - It is an inverse of a Java Servlet
- Four different elements are used in constructing JSPs
  - Scripting Elements
  - Implicit Objects
  - Directives
  - Actions

# 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



# Scripting Elements

## Types

- There are three kinds of scripting elements
  - Declarations
  - Scriptlets
  - Expressions

# Declarations

## Basics

- Declarations are used to define methods & instance variables
  - Do not produce any output that is sent to client
  - Embedded in <%! and %> delimiters

Example:

```
<%!
    Public void jspDestroy() {
        System.out.println("JSP Destroyed");
    }
    Public void jspInit() {
        System.out.println("JSP Loaded");
    }
    int myVar = 123;
%>
```

  - The functions and variables defined are available to the JSP Page as well as to the servlet in which it is compiled

# Scriptlets

## Basics

- Used to embed java code in JSP pages.
  - Contents of JSP go into _JSPpageservice() method
  - Code should comply with syntactical and semantic constuct of java
  - Embedded in <% and %> delimiters

Example:

```
<%
    int x = 5;
    int y = 7;
    int z = x + y;
%>
```

# Expressions

## Basics

- Used to write dynamic content back to the browser.
  - If the output of expression is Java primitive the value is printed back to the browser
  - If the output is an object then the result of calling toString on the object is output to the browser
  - Embedded in <%= and %> delimiters

  Example:
  - <%="Fred"+ " " + "Flintstone %>

    prints "Fred Flintstone" to the browser
  - <%=Math.sqrt(100)%>

    prints 10 to the browser

# Java Implicit Objects

## Scope

- Implicit objects provide access to server side objects
  - e.g. request, response, session etc.
- There are four scopes of the objects
  - Page: Objects can only be accessed in the page where they are referenced
  - Request: Objects can be accessed within all pages that serve the current request.

    (Including the pages that are forwarded to and included in the original jsp page)
  - Session: Objects can be accessed within the JSP pages for which the objects are defined
  - Application: Objects can be accessed by all JSP pages in a given context

# Java Implicit Objects

## List

- request: Reference to the current request
- response: Response to the request
- session: session associated woth current request
- application: Servlet context to which a page belongs
- pageContext: Object to access request, response, session and application associated with a page
- config: Servlet configuration for the page
- out: Object that writes to the response output stream
- page: instance of the page implementation class (this)
- exception: Available with JSP pages which are error pages

# Java Implicit Objects

## Example

```
<html>
 <head>
  <title>Implicit Objects</title>
 </head>
 <body style="font-family:verdana;font-size:10pt">
  <p>
   Using Request parameters...<br>
   <b>Name:</b> <%= request.getParameter("name") %>
  </p>
  <p>
   <% out.println("This is printed using the out implicit
        variable"); %>
  </p>
  <p>
   Storing a string to the session...<br>
   <% session.setAttribute("name", "Meeraj"); %>
   Retrieving the string from session...<br>
   <b>Name:</b> <%= session.getAttribute("name") %>
  </p>
  <p>
   Storing a string to the application...<br>
   <% application.setAttribute("name", "Meeraj"); %>
   Retrieving the string from application...<br>
   <b>Name:</b>
   <%= application.getAttribute("name") %>
  </p>
  <p>
   Storing a string to the page context...<br>
   <% pageContext.setAttribute("name", "Meeraj"); %>
   Retrieving the string from page context...</br>
   <b>Name:</b>
   <%= pageContext.getAttribute("name") %>
  </p>
 </body>
</html>
```

# Example Implicit Objects

## Deploy & Run

- Save file:
  - $TOMCAT_HOME/webapps/jsp/Implicit.jsp
- Access file
  - http://localhost:8080/jsp/Implicit.jsp?name=Sanjay
- Results of the execution

Using Request parameters...
**Name:** sanjay
This is printed using the out implicit variable
Storing a string to the session...
Retrieving the string from session...
**Name:** Meeraj
Storing a string to the application...
Retrieving the string from application...
**Name:** Meeraj
Storing a string to the page context...
Retrieving the string from page context...
**Name:** Meeraj

# Directives

## Basics & Types

- Messages sent to the JSP container
  - Aids the container in page translation
- Used for
  - Importing tag libraries
  - Import required classes
  - Set output buffering options
  - Include content from external files
- The jsp specification defines three directives
  - Page: provder information about page, such as scripting language that is used, content type, or buffer size
  - Include – used to include the content of external files
  - Taglib – used to import custom actions defined in tag libraries

# Page Directives

## Basics & Types

- Page directive sets page properties used during translation
  - JSP Page can have any number of directives
  - Import directive can only occur once
  - Embedded in <%@ and %> delimiters
- Different directives are
  - Language: (Default Java) Defines server side scripting language (e.g. java)
  - Extends: Declares the class which the servlet compiled from JSP needs to extend
  - Import: Declares the packages and classes that need to be imported for using in the java code (comma separated list)
  - Session: (Default true) Boolean which says if the session implicit variable is allowed or not
  - Buffer: defines buffer size of the jsp in kilobytes (if set to none no buffering is done)

# Page Directives

## Types con't.

- Different directives are (cont'd.)
  - autoFlush:When true the buffer is flushed when max buffer size is reached (if set to false an exception is thrown when buffer exceeds the limit)
  - isThreadSafe: (default true) If false the compiled servlet implements SingleThreadModel interface
  - Info: String returned by the getServletInfo() of the compiled servlet
  - errorPage: Defines the relative URI of web resource to which the response should be forwarded in case of an exception
  - contentType: (Default text/html) Defines MIME type for the output response
  - isErrorPage: True for JSP pages that are defined as error pages
  - pageEncoding: Defines the character encoding for the jsp page

# Page Directives

## Example

```
<%@
    page language="java"
    buffer="10kb"
    autoflush="true"
    errorPage="/error.jsp"
    import="java.util.*, javax.sql.RowSet"
%>
```

# Include Directive

## Basics

- Used to insert template text and JSP code during the translation phase.
  - The content of the included file specified by the directive is included in the including JSP page
- Example
  - <%@ include file="included.jsp" %>

# JSP Actions

## Basics & Types

- Processed during the request processing phase.
  - As opposed to JSP directives which are processed during translation
- Standard actions should be supported by J2EE compliant web servers
- Custom actions can be created using tag libraries
- The different actions are
  - Include action
  - Forward action
  - Param action
  - useBean action
  - getProperty action
  - setProperty action
  - plugIn action

# JSP Actions

## Include

- Include action used for including resources in a JSP page
  - Include directive includes resources in a JSP page at translation time
  - Include action includes response of a resource into the response of the JSP page
  - Same as including resources using RequestDispatcher interface
  - Changes in the included resource reflected while accessing the page.
  - Normally used for including dynamic resources
- Example
  - <jsp:include page="inlcudedPage.jsp">
  - Includes the the output of includedPage.jsp into the page where this is included.

# JSP Actions

## Forward

- Forwards the response to other web specification resources
  - Same as forwarding to resources using RequestDispatcher interface
- Forwarded only when content is not committed to other web application resources
  - Otherwise an IllegalStateException is thrown
  - Can be avoided by setting a high buffer size for the forwarding jsp page
- Example
  - <jsp:forward page="Forwarded.html">
  - Forwards the request to Forwarded.html

# JSP Actions - Include Action v/s Include Directive

| Include Directive | Include Action |
| --- | --- |
| Translation time | Run time |
| Copies the included file | References to the included file |
| For static content | For dynamic content |
| Cannot pass parameters | Can pass parameters |

# JSP Actions

## Param

- Used in conjunction with Include & Forward actions to include additional request parameters to the included or forwarded resource
- Example

```
<jsp:forward page="Param2.jsp">
    <jsp:param name="FirstName" value="Sanjay">
</jsp:forward>
```

  - This will result in the forwarded resource having an additional parameter FirstName with a value of Sanjay

# JSP Actions

## useBean

- Creates or finds a Java object with the defined scope.
  - Object is also available in the current JSP as a scripting variable
- Syntax:

```
<jsp:useBean id="name"
scope="page | request | session | application"
class="className" type="typeName" |
bean="beanName" type="typeName" |
type="typeName" />
```

  - At least one of the type and class attributes must be present
  - We can't specify values for bith the class and bean name.
- Example

```
<jsp:useBean id="myName" scope="request" class="java.lang.String">
        <% firstName="Sanjay"; %>
</jsp:useBean>
```

# JSP Actions

## get/setProperty

- getProperty is used in conjunction with useBean 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
- 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="firstName" param="fname"/>
  - Sets the name property of myBean to the request parameter fname
  - <jsp:setProperty name="myBean" property="*">
  - Sets property to the corresponding value in request

# JSP Actions

## plugIn

- Enables the JSP container to render appropriate HTML (based on the browser type) to:
  - Initiate the download of the Java plugin
  - Execution of the specified applet or bean
- plugIn standard action allows the applet to be embedded in a browser neutral fashion
- Example

```
<jsp: plugin type="applet" code="MyApplet.class" codebase="/">
        <jsp:params>
                <jsp:param name="myParam" value="122"/>
        </jsp:params>
        <jsp:fallback><b>Unable to load applet</b></jsp:fallback>
</jsp:plugin>
```