What is Servlet?`

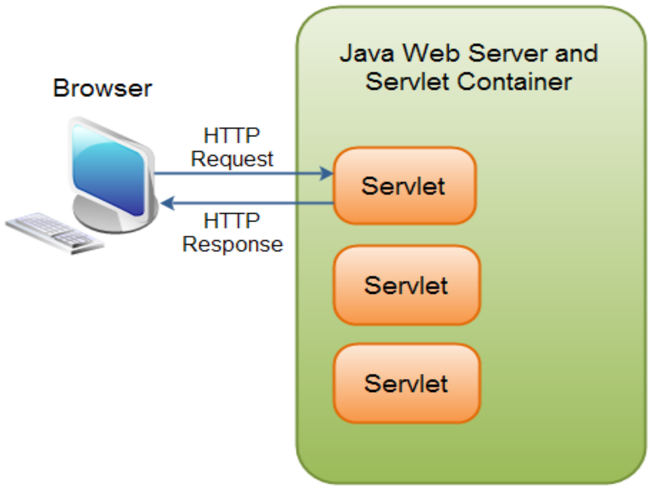
A Servlet is a Server Side Java program which extends the functionality of web application.

It is used to create dynamics web pages.

Key Features of Java Servlets:

- **Platform Independence:** Written in Java, servlets are platform-independent, ensuring compatibility across various operating systems.
- **Efficiency:** Unlike CGI scripts, servlets run within the server's process, eliminating the need to create a new process for each request, which enhances performance.
- **Robustness:** Leveraging Java's strong typing and exception handling, servlets provide a robust environment for web applications.

It runs inside a Servlet container



Servlets inside a Java Servlet Container

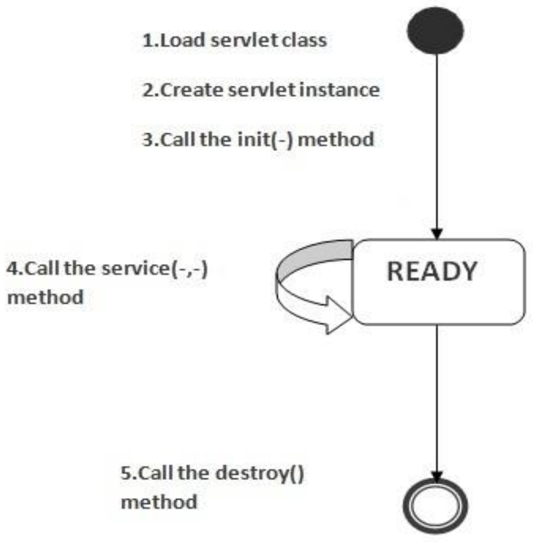
Servlet Containers: Java servlet containers are usually running inside a Java web server. A few common well known, free Java web servers are:

- Jetty
- Tomcat

Life Cycle of a Servlet (Servlet Life Cycle)

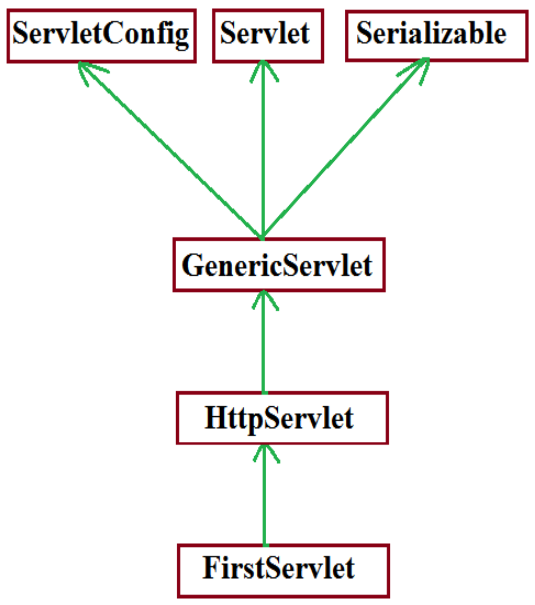
The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

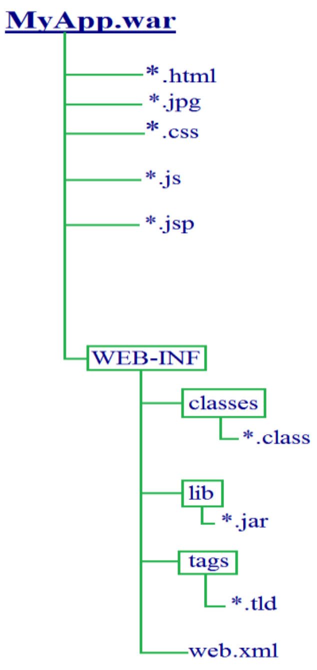
1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

The ways to create a Servlet

There are three ways to create the servlet.
1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class
The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.





Deployment Structure :

The following are the methods defined in Servlet interface.

1. init( )
2. service( )
3. destroy( )
4. getServletConfig( )
5. getServletInfo( )

NOTE : init( ) , service( ) , destroy( ) are called life cycle methods of Servlet. we can call destroy() explicitly from the init( ) and service() , **in this case destroy( ) will be executed just like a normal method call and servlet object won't be destroyed.**

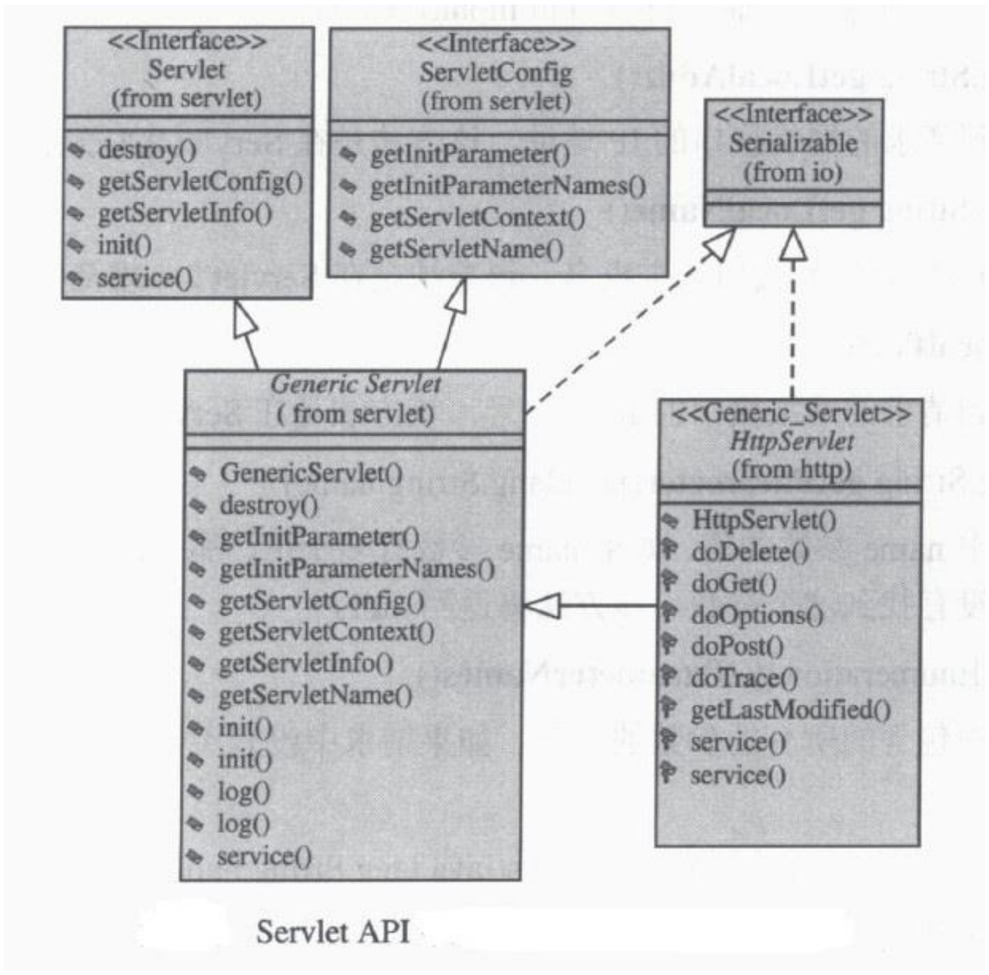
Servlet API

The **javax.servlet** and **javax.servlet.http** packages represent interfaces and classes for servlet api.

We can develop Servlets by using the following 2 packages.

The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.



What is the difference between Get and Post?

GET	POST
1) In case of Get request, only limited amount of data can be sent because data is sent in header.	In case of post request, large amount of data can be sent because data is sent in body.
2)We can send only text data.	We can send text & binary data.

3) It not secured because data is exposed in URL bar.	It is more secured because data is not exposed in URL bar.
4) Get request can be bookmarked	Post request cannot be bookmarked
5) Get request is idempotent. It means second request will be ignored until response of first request is delivered.	Post request is non-idempotent
6) Get request is more efficient and used more than Post	Post request is less efficient and used less than get.

Note:- All information stored in the body of the request rather than in the URL,which provides for more privacy.POST method is used for sending credit card no., updated database etc.

Idempotent request:

By repeating the request multiple times , if there is no change in response such type of requests are Idempotent requests.

Ex: GET requests are Idempotent and POST requests are not Idempotent.

Safe request :

By repeating the same request multiple times , if there is no side-effect at server side , such type of requests are called safe-requests .

Ex: GET requests are safe , where as POST requests are not safe to repeat.

Triggers to send GET request :

1. Type url in the address bar and submit is always GET request.
2. Clicking hyperlink is always GET request.
3. Submitting the form , where method attribute specify with GET value is always GET request.
   ```
   <form action="/test"  method="GET">
   ----------- ----
   </form>
   ```
4. Submitting the form without method attribute is always GET request i.e., default method for form is GET.

How can you call a jsp from the servlet?

one of the way is RequestDispatcher interface for example:

```
RequestDispatcher rd=request.getRequestDispatcher("/login.jsp");
rd.forward(request,response);
```

SendRedirect

The sendRedirect() method of HttpServletResponseinterface can be used to redirect response to another resource, it may be servlet, jsp or html file.

It accepts relative as well as absolute URL.

It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

Difference between forward() and sendRedirect() method

There are many differences between the forward() method of RequestDispatcher and sendRedirect() method of HttpServletResponse interface. They are given below:

forward() method	sendRedirect() method
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
Example: request.getRequestDispacher("servlet2").forward (request,response);	Example: response.sendRedirect("servlet2");

Syntax of sendRedirect() method

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

Servlet with Annotation (feature of servlet3):

Annotation represents the metadata. If you use annotation, deployment descriptor (web.xml file) is not required. But you should have tomcat7 as it will not run in the previous versions of tomcat. @WebServlet annotation is used to map the servlet with the specified name.

1. In Servlet 3.0, web.xml is optional.
2. Configuration part managed by annotation called @webServlet.
3. URL-pattern attribute is mandatory.
4. It reduces the complexity of web.xml

Example of simple servlet by annotation

There is given the simple example of servlet with annotation.

Simple.java

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

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

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

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


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

        out.print("<html><body>");
        out.print("<h3>Hello Servlet</h3>");
        out.print("</body></html>");
    }
}
```

What are the annotations used in Servlet 3?

There are mainly 3 annotations used for the servlet.

1. @WebServlet : for servlet class.
2. @WebListener : for listener class.
3. @WebFilter : for filter class.

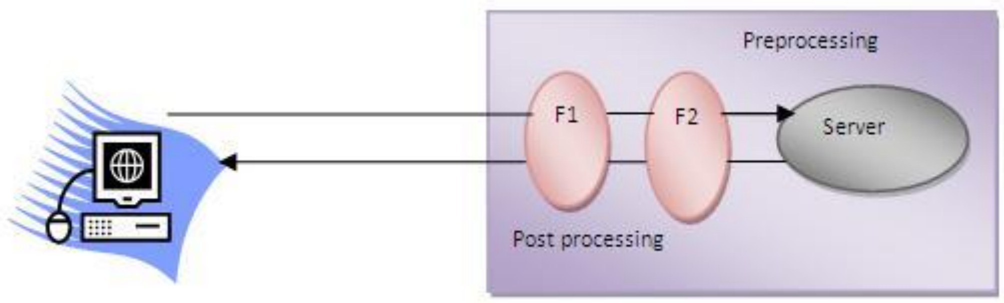
What is a Filter?

A filter is an object that is invoked at the preprocessing and post processing of a request.

It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.

The servlet filter is pluggable, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.

So maintenance cost will be less.



Note: Unlike Servlet, One filter doesn't have dependency on another filter.

Usage of Filter

- recording all incoming requests

- logs the IP addresses of the computers from which the requests originate
- conversion
- data compression
- encryption and decryption
- input validation etc.

Advantage of Filter

1. Filter is pluggable.
2. One filter don't have dependency onto another resource.
3. Less Maintenance

Filter API

Like servlet filter have its own API. The javax.servlet package contains the three interfaces of Filter API.

1. Filter
2. FilterChain
3. FilterConfig

How to define Filter

We can define filter same as servlet. Let's see the elements of filter and filter-mapping.

```
<web-app>

<filter>
<filter-name>...</filter-name>
<filter-class>...</filter-class>
</filter>

<filter-mapping>
<filter-name>...</filter-name>
<url-pattern>...</url-pattern>
</filter-mapping>

</web-app>
```

Session Tracking in Servlets

Session simply means a connection for a particular interval of time.

Why Session Tracking?

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.

HTTP is stateless that means each request is considered as the new request.

Dynamic applications are built on the assumption that the same client accesses the application multiple times.

Client data is lost with every request being considered as a new request.

Session Tracking Techniques

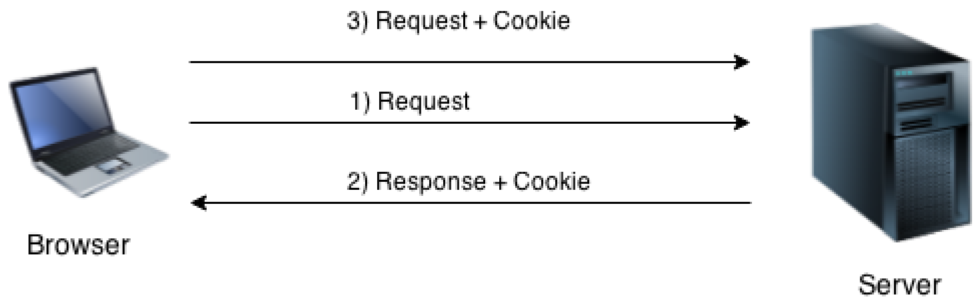
There are four techniques used in Session tracking:

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. HttpSession

1) Cookie

A cookie is a small piece of textual information sent by the server to the client, stored on the client, and returned by the client for all requests to the server.

- Stores data on the client-side (user's browser).
- Less secure than sessions for sensitive information (due to potential theft).



How to create Cookie?

Let's see the simple code to create cookie.

```
1. Cookie ck=new Cookie("user","kumar");//creating cookie object
2. response.addCookie(ck);//adding cookie in the response
```

How to delete Cookie?

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
Cookie ck=new Cookie("user","");//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

Adding Cookies to the response :

HttpServletResponse contains the following method to addCookie to the Response object.

Ex:

```
Cookie c=new Cookie(String name , String value) ;
response.addCookie(c);
```

How to get Cookies?

Let's see the simple code to get all the cookies.

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and va
lue of cookie
}
```

URL Rewriting

```
// (Pseudocode, not recommended for real applications)

String sessionId = request.getSession().getId();

String targetUrl = "welcome.jsp?sessionid=" + sessionId;

response.sendRedirect(targetUrl);
```

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

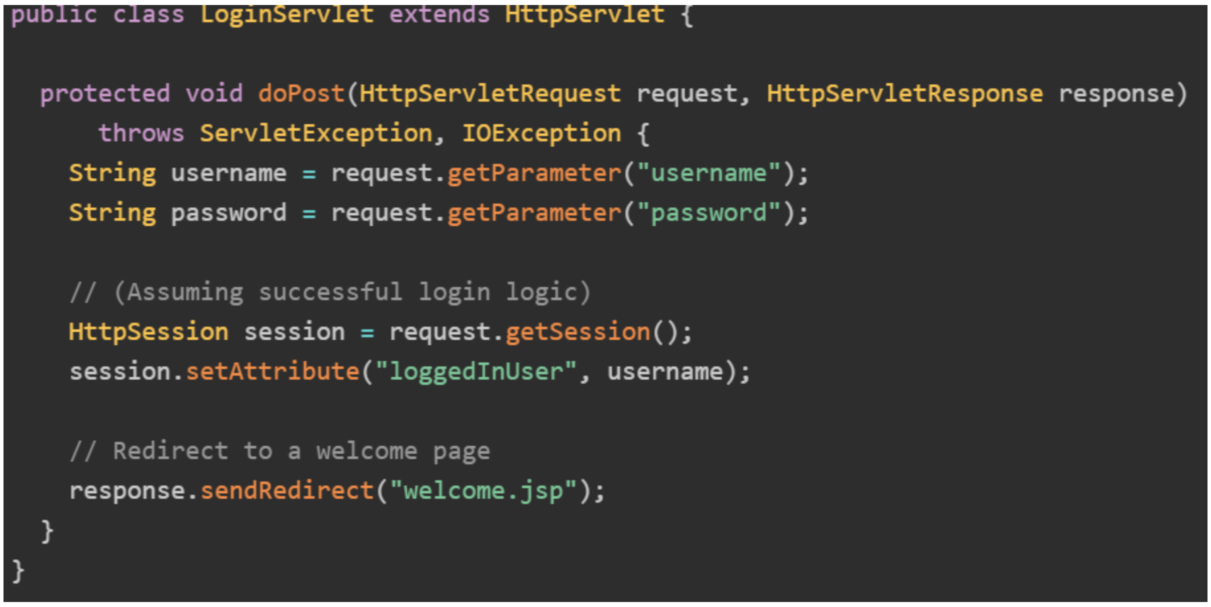
1. It will work only with links.
2. It can send Only textual information.
3. 1 Static page in the cycle cannot be the part of url rewriing
4. 2 User Book marks the Url

HttpSession interface

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.



```java
public class LoginServlet extends HttpServlet {

  protected void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    // (Assuming successful login logic)
    HttpSession session = request.getSession();
    session.setAttribute("loggedInUser", username);

    // Redirect to a welcome page
    response.sendRedirect("welcome.jsp");
  }
}
```

Hidden Form Fields in HTML

In this approach the unique token is embedded within each HTML form. For example

<input type=”Hidden” name=”uid” value=”dac”>

When the request is submitted, the server receives the token as part of the request, which in turn can be used to identify the client by matching the unique token. The servlet specification does not recommend this approach

Introduction to JSP

What is JSP?

1. JSP stands for Java Server Pages technology (JSP)
2. JSP is a combination of java and html
3. It is used to create dynamic web page.
4. It is an extension to the servlet technology.
5. A JSP page is internally converted into servlet.

Advantage of JSP over Servlet

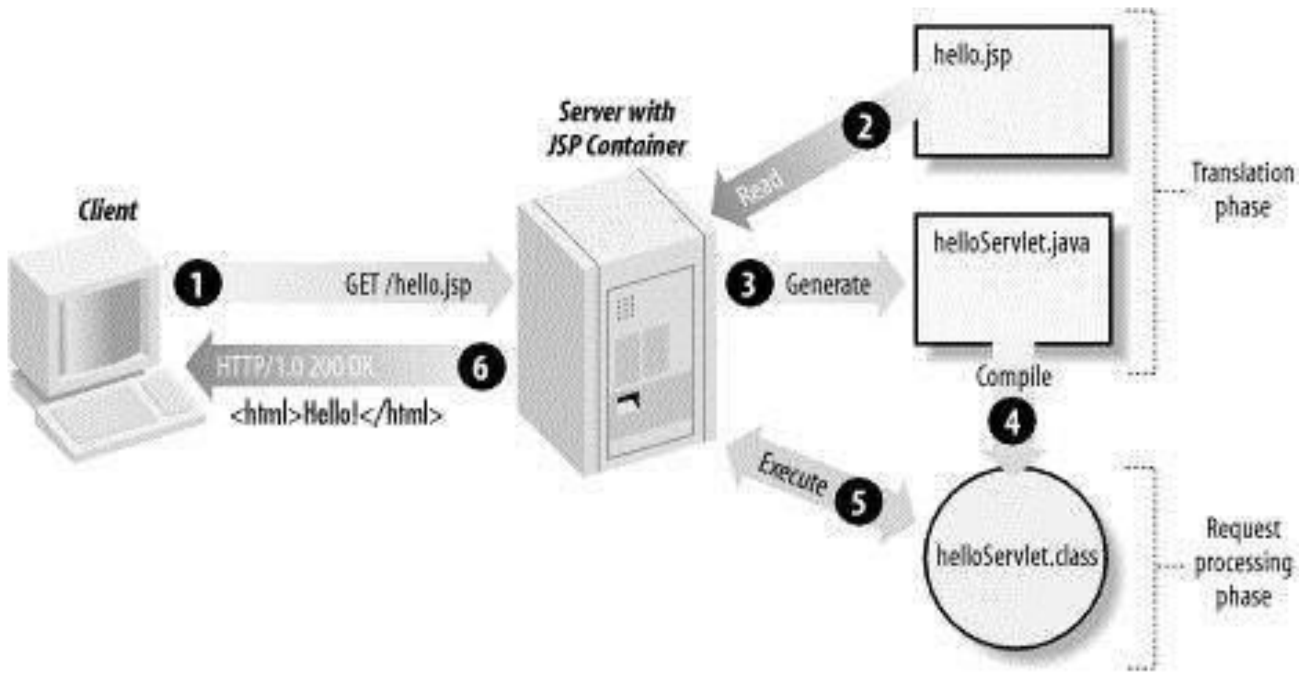
There are many advantages of JSP over servlet. They are as follows:

1) **Extension to Servlet:** We can use all the features of servlet in JSP

2) **Easy to maintain**: we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

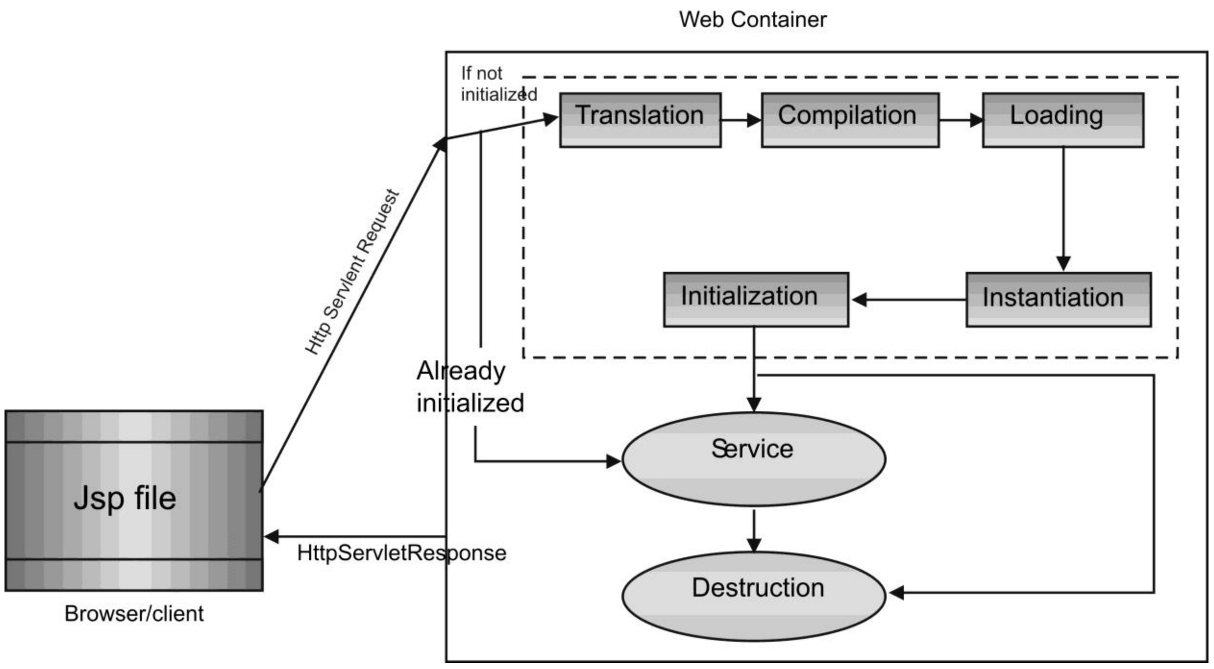
3) **Fast Development**: No need to recompile and redeploy

4) **Less code** than Servlet



Life cycle of a JSP Page

Note: jspInit(), _jspService() and jspDestroy() are the life cycle methods of JSP.



<u>Syntax and Semantics of JSP</u>

1.	Directives	<%@ %>
2.	Declaration	<%! %>
3.	Scriplet	<% %>
4.	Expressions	<%= %>
5.	JSP Comment	<%- - - -%>

A simple JSP

```
<html>
<body>
<% out.print("Hello God"); %>
</body>
</html>
```

It will print Hello God on the browser.

<html><body>

```
<%@ page language="java" %>        ← Directive
<%! int count = 0; %>              ← Declaration
<% count++; %>                      ← Scriptlet
Welcome! You are visitor number
<%= count %>                        ← Expression

</body></html>
```

JSP Implicit Objects

There are 9 jsp implicit objects. These objects are created by the web container that are available to all the jsp pages.

A list of the 9 implicit objects is given below:

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

Example of JSP request implicit object

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);
%>
```

Example of response implicit object

index.html

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

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```

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=(String)session.getAttribute("user");
out.print("Hello "+name);
  %>
</body>
</html>
```

JSP directives

Directives gives direction to JSP container about the JSP page.The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

1. page directive
2. include directive
3. taglib directive

3 types of directives:

1. **page**
   Controls properties of the JSP page
2. **include**
   Include the contents of a file into the JSP page at translation time.

3. **taglib**
   Makes a custom tag library available within the including page.

JSP page directive

The page directive defines attributes that apply to an entire JSP page.

Syntax of JSP page directive

<%@ page attribute="value" %>
<%@ page language="java" %>
<%@ page import="java.util.*,java.sql.* %>
<%@ page session="true"%>

JSP include directive

Includes the contents of a file into the JSP page during the translation phase

<%@ include file="fileName.jsp %>

Attributes of JSP page directive

1. language
2. extends
3. import
4. session

5. autoFlush
6. buffer

7. errorPage
8. isErrorPage

9. isThreadSafe
10. info

11. contentType
12. pageEncoding

13. isScriptingEnable
14. isELIgnored

1)import

The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface.

Example of import attribute

```
<html>
<body>

<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>

</body>
</html>
```

Example of errorPage attribute

index.jsp

```
<html>
<body>

<%@ page errorPage="myerrorpage.jsp" %>

 <%= 100/0 %>

</body>
</html>
```

---

isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page.

Note: The exception object can only be used in the error page.

Example of isErrorPage attribute

myerrorpage.jsp

```
<html>
<body>
```

```
<%@ page isErrorPage="true" %>

 Sorry an exception occured!<br/>
The exception is: <%= exception %>

</body>
</html>
```

Scriplet

- This elements actually deal with Java code.
- Scripting elements are divided into 3 parts:
    1. **Declarations -** Declare and define variables and methods

       <%! int count=0 ; %>

    2. **Scriplets -** Java code fragments that are embedded in the JSP page

```
<%
count++;
out.println(“welcome!! You are visitor number: ”+ count);
%>
```

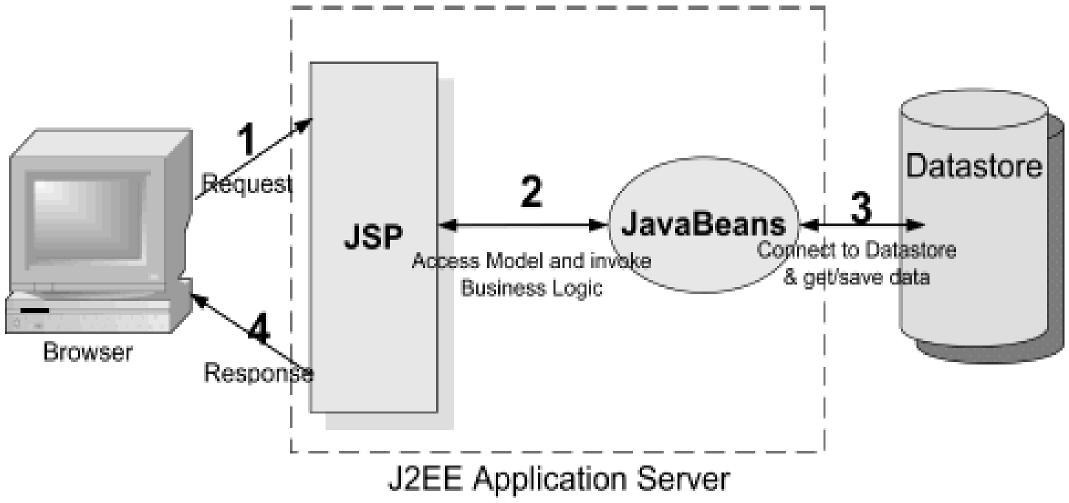
3. **Expressions** – It is shoutcut for out.print() .it act as placeholders for Java language expressions

   <%= count %>

Java Bean

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

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


1
Request
JSP
2
Access Model and invoke Business Logic
JavaBeans
3
Connect to Datastore & get/save data
Datastore
Browser
4
Response
J2EE Application Server

JSTL (JSP Standard Tag Library)

1. JSTL
2. JSTL Core Tags

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

Advantage of JSTL

1. Fast Developement JSTL provides many tags that simplifies the JSP.
2. Code Reusability We can use the JSTL tags in various pages.
3. No need to use scriptlet tag It avoids the use of scriptlet tag.

Syntax for defining core tags
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

---

c:catch

It is an alternative apporach of global exception handling of JSP. It handles the exception and doesn't propagate the exception to error page. The exception object thrown at runtime is stored in a variable named **var**.

Example of c:catch

Let's see the simple example of c:catch.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:catch>
int a=10/0;
</c:catch>
```

---

c:out

It is just like JSP expression tag but it is used for exression. It renders data to the page.

Example of c:out

index.jsp

```
<form action="process.jsp" method="post">
FirstName:<input type="text" name="fname"/><br/>
LastName:<input type="text" name="lname"/><br/>
<input type="submit" value="submit"/>
</form>
```

process.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
First Name:<c:out value="${param.fname}"></c:out><br/>
Last Name:<c:out value="${param.lname}"></c:out>
```

---

c:import

It is just like jsp include but it can include the content of any resource either within server or outside the server.

Example of c:import

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<h1>ABC.com</h1>
<hr/>
<c:import url="http://www.dacinfotech.com"></c:import>
```

Example of c:import to display the source code

Let's see the simple example of c:import to display the source code of other site.

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<h1>ABC.com</h1>
<hr/>
<c:import var="data" url="http://www.dacinfotech.com"></c:import>

<h2>Data is:</h2>
<c:out value="${data}"></c:out>
```

---

c:forEach

It repeats the nested body content for fixed number of times or over collection.

Example of c:forEach

Let's see the simple example of c:forEach.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:forEach var="number" begin="5" end="10">
<c:out value="${number}"></c:out>
</c:forEach>
```

---

c:if

It tests the condition.

Example of c:if

Let's see the simple example of c:if.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="number" value="${200}">
<c:if test="${number<500}">
<c:out value="number is less than 500"></c:out>
</c:if>
```

---

c:redirect

It redirects the request to the given url.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:redirect url="http://www.sdcinfotech.com"></c:redirect>
```

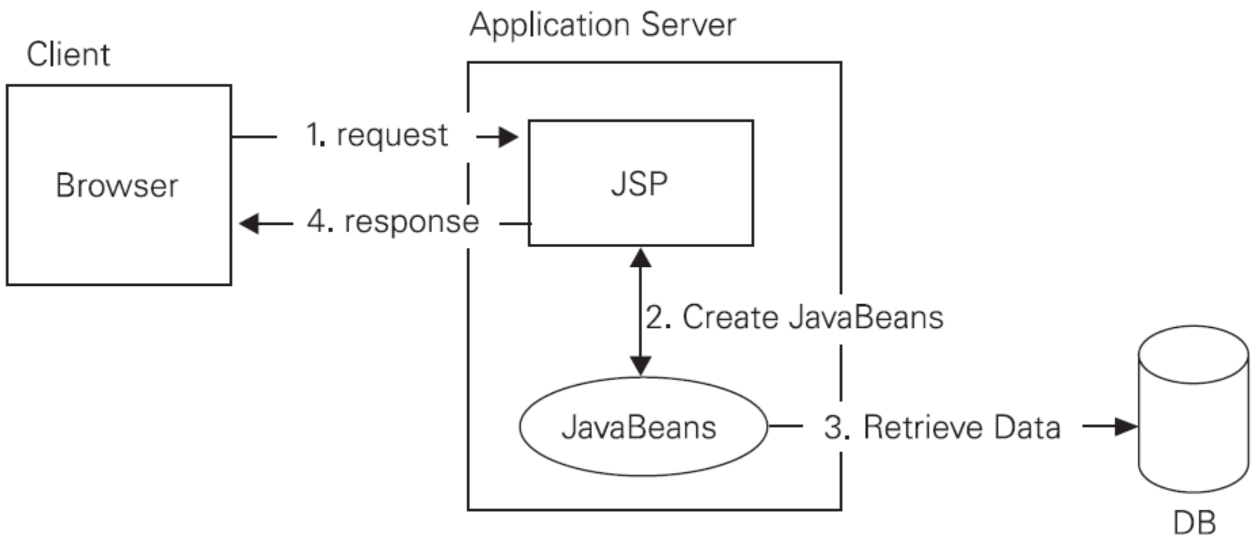
JSP ARCHITECTURE MODELS

There are 2 types of architecture in JSP
JSP Model 1 and JSP Model 2 architectures.

Model 1 and Model 2 (MVC) Architecture

1. Model 1 Architecture (page centric)
2. Model 2 (MVC) Architecture (Servlet centric)

Model 1 Architecture (page centric)



Advantage of Model 1 Architecture

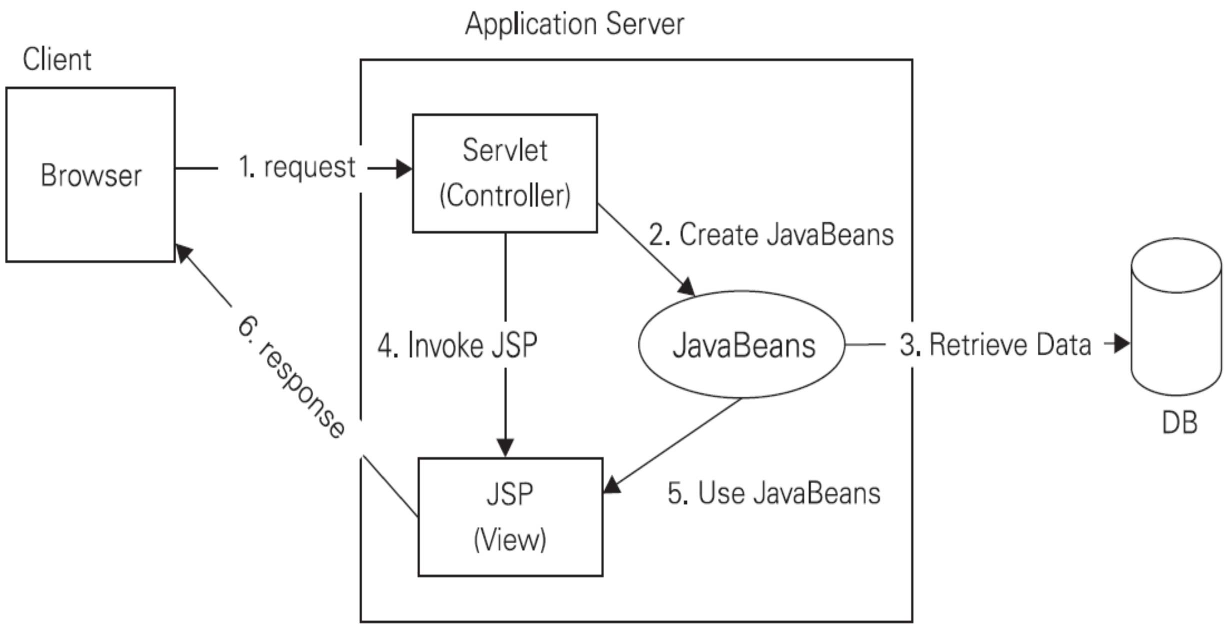
This architecture is suitable for simple applications. So it becomes easy and Quick to develop web application

Disadvantages

1. It becomes difficult to manage and maintain when application grows
2. Navigation path is embedded in the JSP Page ,

1. It cannot be used for complex applications.
2. It creates a problem for the web page designers who are usually not comfortable
with server-side programming.
3. It does not promote reusability of application components. For example, the code written in a JSP page for authenticating a user cannot be reused in other JSP pages.

Model 2 (MVC) Architecture (Servlet centric)



The JSP Model 2 architecture

MVC pattern consists of three modules: model, view, and controller.
The view takes care of the display of the application.
The model encapsulates the application data and business logic.
The controller receives user input and commands the model and/or the view to change accordingly