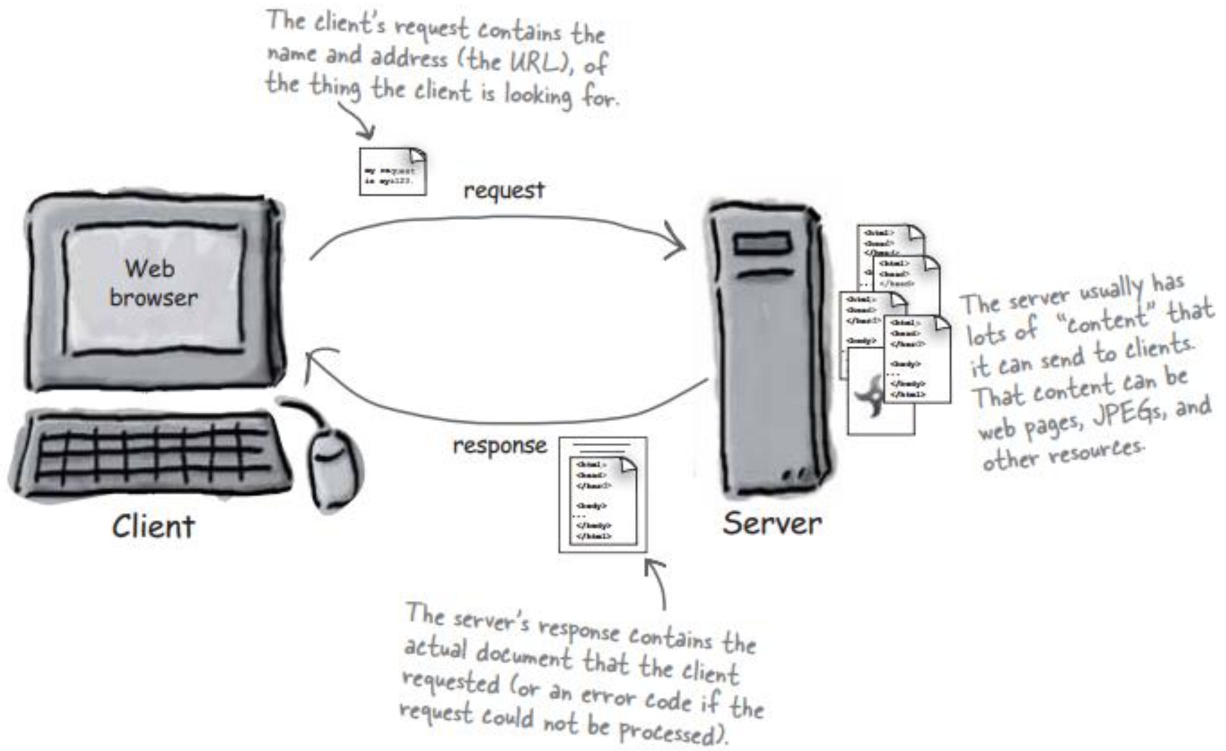
Unit-4

Servlet and JSP

Asst. Prof. Roshan Tandukar

Web Server

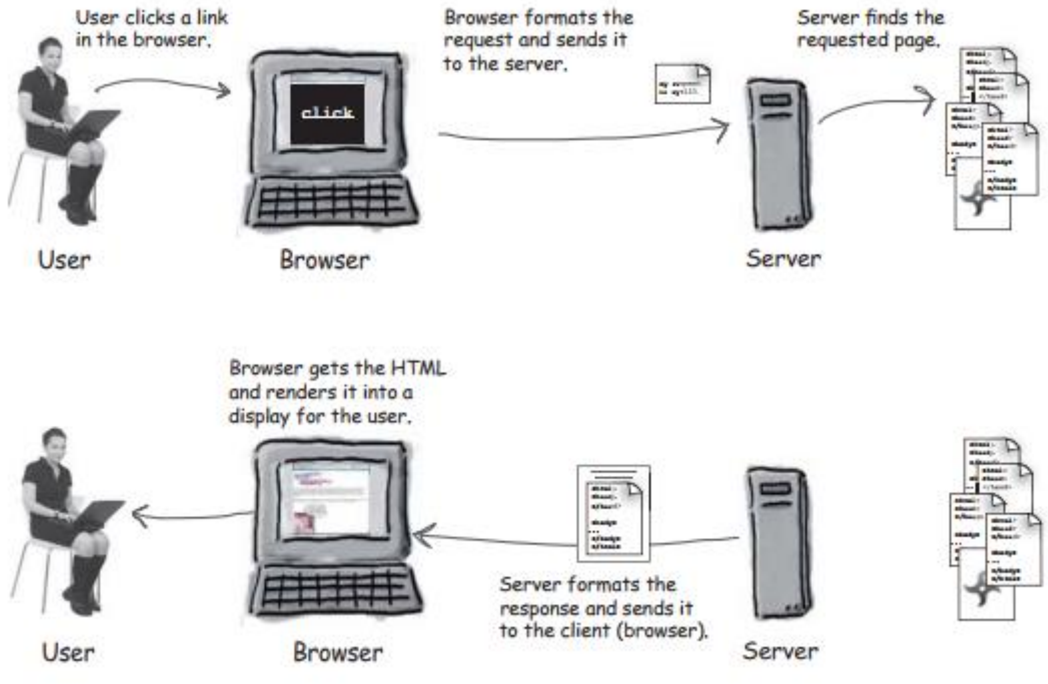
- A web server takes a client request and gives something back to the client.



The client's request contains the name and address (the URL), of the thing the client is looking for.
request
Web browser
Client
response
Server
The server usually has lots of "content" that it can send to clients. That content can be web pages, JPEGs, and other resources.
The server's response contains the actual document that the client requested (or an error code if the request could not be processed).

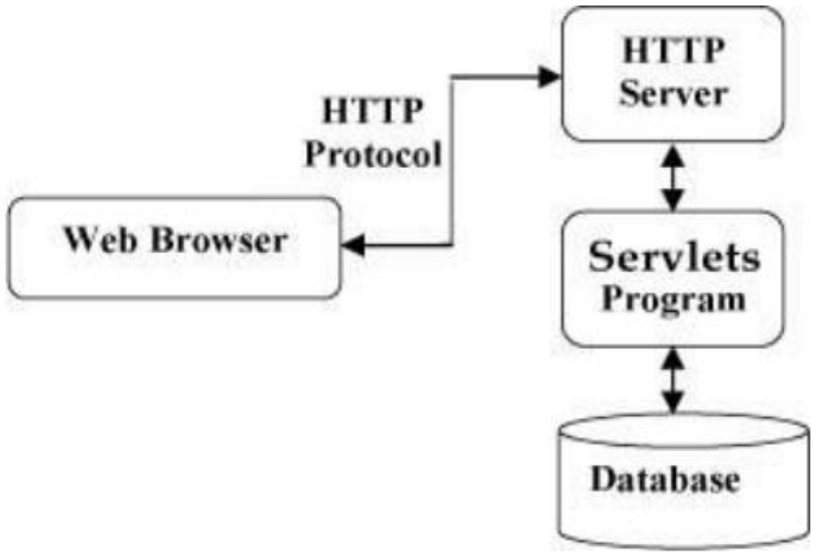
Web client

- A web client lets the user request on the server and shows the user the result of the request.


User clicks a link in the browser.
click
Browser formats the request and sends it to the server.
Server finds the requested page.
User
Browser
Server
Browser gets the HTML and renders it into a display for the user.
Server formats the response and sends it to the client (browser).
User
Browser
Server

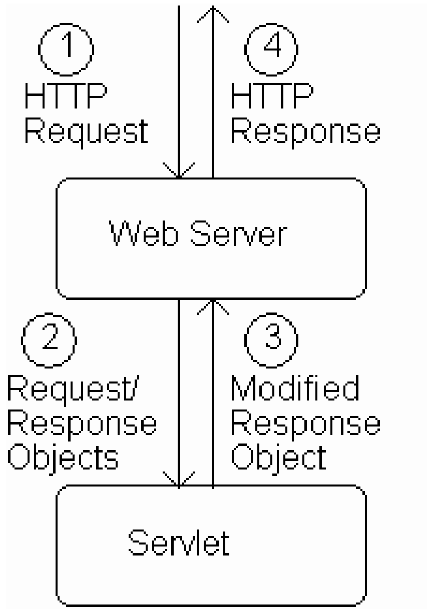
Servlets

- Java Servlets are programs that run on a Web or Application server
- Servlets are also called server side programs i.e the code of the program executes at server side, acting as a middle layer between request coming from Web browsers or other HTTP clients and databases or applications on the HTTP server.



Servlets

- Java Servlets are programs that run on a Web or Application server
- Servlets are also called server side programs i.e the code of the program executes at server side, acting as a middle layer between request coming from Web browsers or other HTTP clients and databases or applications on the HTTP server.



Servlets

- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- A servlet container uses a Java Virtual Machine to run servlet code as requested by a web server.
- A servlet dynamically loaded module that services requests from Web server. It runs entirely inside the Java Virtual Machine.

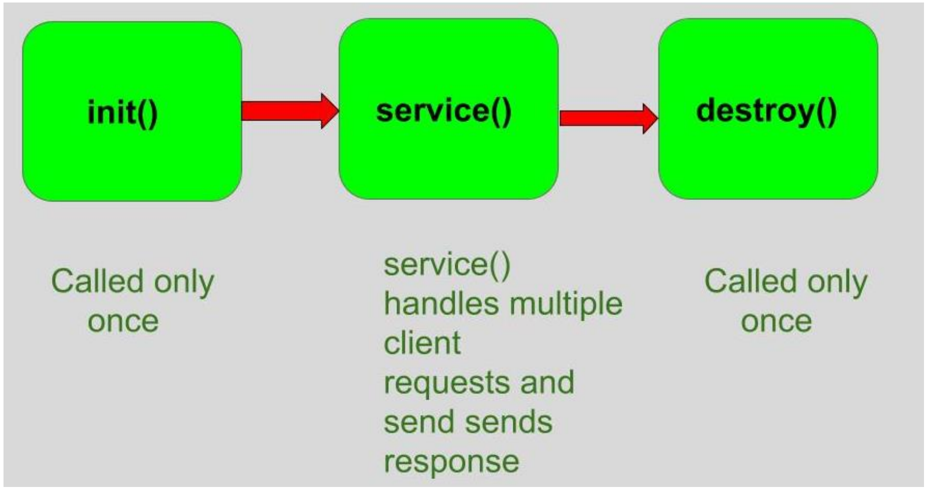
Servlets Packages

- Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.
- These classes implement the Java Servlet and JSP specifications.
- Java servlets have been created and compiled just like any other Java class.
- After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

Advantages of Servlets

- Efficient
- Portable
- Robust
- Extensible
- Secure
- Performance

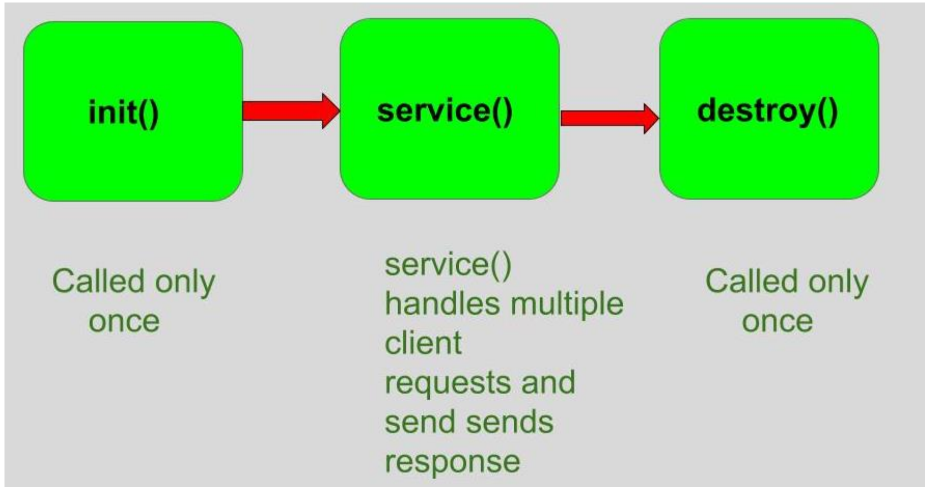
Life Cycle of Java Servlets


init()
service()
destroy()
Called only once
service() handles multiple client requests and send sends response
Called only once

- The init() method is where the servlets life begins and is called by the server immediately after the servlet is instantiated
- The database connection, opening a file, or a network connection may be established through it
- The init method is not called again for each request
- The servlet is normally created when a user first invokes a URL corresponding to servlet

```
public void init (ServletConfig config) throws ServletException
```

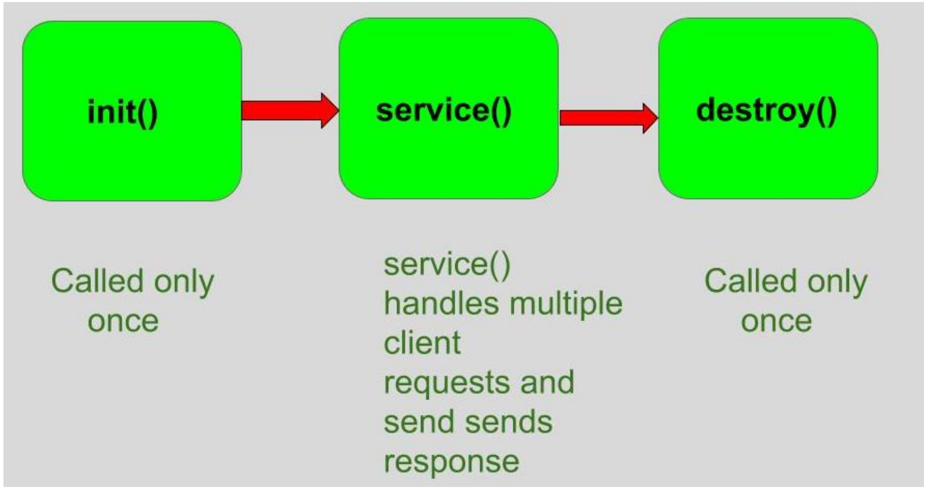
Life Cycle of Java Servlets


init()
service()
destroy()
Called only once
service() handles multiple client requests and send sends response
Called only once

The following are the task performed while implementing init() method

- Reading initializing parameters using ServletConfig object
- Initializing a database driver
- Opening a database connection
- Initialization of objects

Life Cycle of Java Servlets

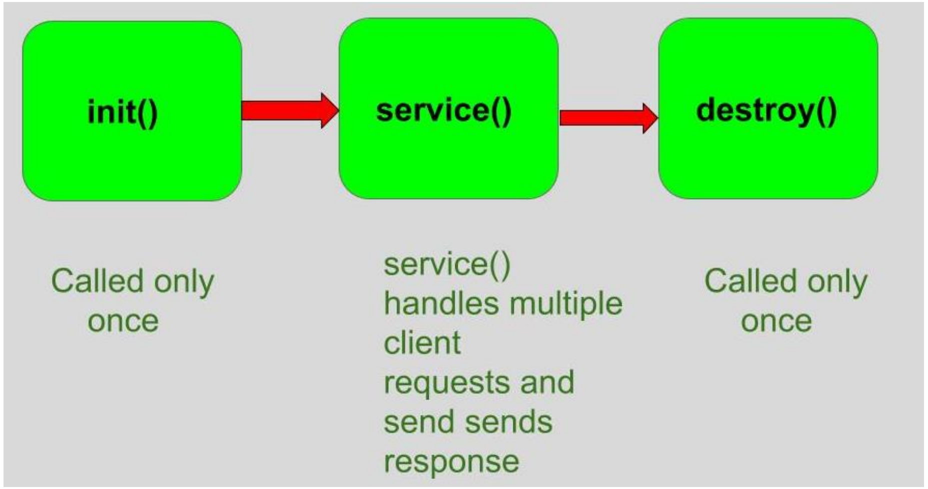

init()
service()
destroy()
Called only once
service() handles multiple client requests and send sends response
Called only once

service()

- The service method handles all requests sent by a client
- Each time the server receive a request for a servlet, the server spawns a new thread and calls service method
- The service method checks the request type GET, POST, PUT, DELETE etc
- For, every request to the servlet, its service method is called with two arguments ServletRequest and ServletResponse

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

Life Cycle of Java Servlets

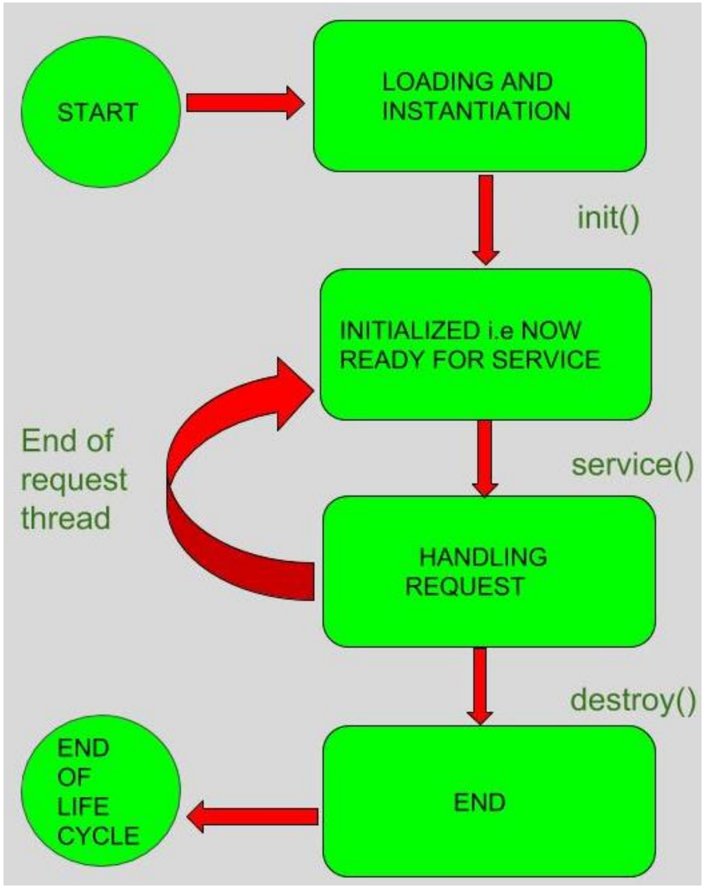

init()
service()
destroy()
Called only once
service() handles multiple client requests and send sends response
Called only once

destroy()

- This method signifies the end of a servlet life
- The resources that are previously allocated are destroyed
- The method is invoked by server administrator or programmatically by servlet itself

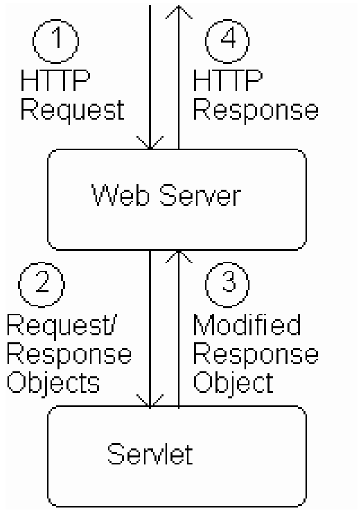
  public void destroy()

Advantages of Servlets



Servlet API

- Servlet API makes manipulating an HTTP request and response pair simply through the use of HTTPServletRequest and HTTPServletResponse objects
- HTTPServletRequest object
    - Information about an HTTP request
        - Headers
        - Query String
        - Session
        - Cookies
- HTTPServletResponse object
    - Used for formatting an HTTP response
        - Headers
        - Status codes
        - Cookies



HTTPServletResponse

- To send the information to a client
- Use either getWriter( ) or getOutputStream( ): returning suitable objects for sending either text or binary content to the client, respectively
- Only one fo the methods may be used with a HTTPServletResponse object and attempting to call both methods causes an exception to be thrown

```
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<head>");
out.println("<title>Hello World!</title>");
```

HTTPServletResponse

Generated Markup	HelloWorld.java
`<html>`	`out.println("<html>");`
`<head>`	`out.println("<head>");`
`<title>Hello World!</title>`	`out.println("<title>Hello World!</title>");`
`</head>`	`out.println("</head>");`
`<body>`	`out.println("</head>");`
`<h1>Hello World!</h1>`	`out.println("<h1>Hello World!</h1>");`
`</body>`	`out.println("</body>");`
`</html>`	`out.println("</html>");`

HTTPServletRequest

- A client’s HTTP request is represented by an HTTPServletRequest object
- It is primarily used for getting request headers, parameters and files or data sent by a client
- However, the servlet specification enhances this object to also interact with a web applications
- Some of the helpful features include session management and forwarding of request between servlets

Reading Data from a Client

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

1. getParameter() − You call request.getParameter() method to get the value of a form parameter by taking a parameter as a parameter name

2. getParameterValues() − Call this method if the parameter appears more than once and returns multiple values, for example checkbox or combobox (sends multiple values for the same parameter name)

3. getParameterNames() − Call this method if you want a complete list of all parameters in the current request.

Reading Data from a Client

- When server starts it instantiates servlets
- Server receives HTTP request, determines need for dynamic response
- Server selects the appropriate servlet to generate the response, creates request/response objects, and passes them to a method on the servlet instance
- Servlet adds information to response object via method calls
- Server generates HTTP response based on information stored in response object

Reading Data from a Client

- The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

GET Method (doGet())

- The GET method sends the encoded user information appended to the page request.
- The page and the encoded information are separated by the ?(question mark) symbol as follows −

http://www.test.com/hello?key1 = value1&key2 = value2

Reading Data from a Client

<u>POST Method</u>

- A generally more reliable method of passing information to a backend program is the POST method.
- This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL it sends it as a separate message.
- This message comes to the backend program in the form of the standard input which you can parse and use for your processing.
- Servlet handles this type of requests using doPost() method.

Javax.servlet Package

- Provides the contract between the servlet/web application and the web container
- Used for creating protocol independent server applications
- Servlet interface defines the core of the entire package
    - Other interfaces provide additional services to the developer
- Contains 12 interfaces
    - 7 interfaces implemented by the package
    - 5 interfaces implemented by the user

Javax.servlet Package

- 7 interfaces implemented by the package
- 5 interfaces implemented by the user
- Server implemented interfaces
  - ServletConfig
  - ServletContext
  - ServletRequest
  - ServletResponse
  - RequestDispatcher
  - FilterChain
  - FilterConfig
- User implemented interfaces
  - Servlet
  - ServletContextListener
  - ServletContextAttributeListener
  - SingleThreadModel
  - Filter

javax.servlet.http

- Javax.servlet package provides interfaces and classes to service client requests in protocol independent manner.
    - Javax.servlet.http package supports http-specific functions.
- Several of the classes are derived from the javax.servlet packaage
- Some methods from the javax.servlet package are also used
- Contains
    - 8 interfaces
    - 7 classes

Servlets: **Writing a Servlet**

- Create a servletclass
    - extend HttpServlet
- Implement the doGet() or doPost() method
    - Both methods accept two parameters
        - HttpServletRequest
        - HttpServletResponse
    - Obtain parameters from HttpServletRequest Interface using
        - getParameter(String name)
    - Obtain the writer from the response object
    - Process input data and generate output (in html form) and write to the writer
    - Close the writer

Example

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException
        {
          // Set the HTTP content type in response header
          response.setContentType("text/html; charset=\"UTF-8\"");

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();
```

Example

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException
        {
          // Set the HTTP content type in response header
          response.setContentType("text/html; charset=\"UTF-8\"");

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();
```

All servlets we will write are subclasses of `HttpServlet`

Example

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException
        {
          // Set the HTTP content type in response header
          response.setContentType("text/html; charset=\"UTF-8\"");

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();
```

Server calls `doGet()` in response to GET request

Example

Interfaces implemented by request/response objects

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException
        {
          // Set the HTTP content type in response header
          response.setContentType("text/html; charset=\"UTF-8\"");

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();
```

Example

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException
        {
          // Set the HTTP content type in response header
          response.setContentType("text/html; charset=\"UTF-8\"");

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();
```

Production servlet should catch these exceptions

Example

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
      throws ServletException, IOException
        {
          // Set the HTTP content type in response header
          response.setContentType("text/html; charset=\"UTF-8\"");

          // Obtain a PrintWriter object for creating the body
          // of the response
          PrintWriter servletOut = response.getWriter();
```

First two things done by typical servlet; must be in this order

Servlets vs. Java Applications

- Servlets do not have a main()
    - The main() is in the server
    - Entry point to servlet code is via call to a method (doGet() in the example)
- Servlet interaction with end user is indirect via request/response object APIs
    - Actual HTTP request/response processing is handled by the server
- Primary servlet output is typically HTML

Maintaining State

There are 3 ways to maintain state

1. Cookies
2. Hidden fields in `form`s
3. Session level authentication

Cookies

- A Cookie is data (String) that the server passes to the browser and the browser stores on the server
  - Set of name value pairs
- Web servers place cookies on user machines with id to track the users
- Cookies can be one of the following:
  - Persistent cookies: Stored on hard drive in text format
  - Non-persistent cookies: Stored in memory and goes away after you reboot or turn off the machine

Cookies

- Cookies are small text files that are used to maintain the different session
- Cookies are send by the web server to the client browser and the browser later returns unchanged when visit the same web page again.
- Once the web server receives cookie information the servlet program processes the cookie information and recognizes it as a new user or already visited user
- Cookies are therefore, a accepted method used by the web server for session tracking.
- Cookies let a user to log in automatically. Cookies can be used to remember user preferences.

Types of Cookies

1. **Permanent Cookies**

- Permanent cookies are stored in client machine, they are not deleted when browsing session is closed. Permanent cookies can be used to identify individual users.
- It persists after the user restart the computer.

2. **SessionCookies**

- Session cookies are also called transient cookies.
- These cookies exists till the browsing session continues and automatically gets deleted as soon as the user closes the web browser.
- These cookies usually store a session ID that is not permanently bound to the user, allowing the user to move from page to page without login each time.

Tracking State: Cookie Attributes

- Attributes of a cookie
  - Name: Name of a cookie
  - Value: Value of the cookie
  - Comment: Text explaining purpose of cookie
  - Max-Age: Time in seconds after which the client should not send cookie back to server
  - Domain: Domain to which the cookie should be sent
  - Path: The path to which the cookie should be sent
  - Secure: Specifies if cookie should be sent via https
  - Version: Cookie version

    (0 – original Netscape version of Cookie

    1 – cookies standardized via RFC 2109)

Cookie Class

- Cookies are created by the web server and stored in client computer.
- A Cookie can be created with the help of Cookie class which is then placed in HTTP response header with the help of method addCookie(cookie_name)
- Constructor
    - Cookie (String name, String value)
- Can be added to the response by using
    - void addCookie(Cookie cookie) of HttpServletResponse
- Can be obtained from the request by using
    - Cookie[] getCookies() method of the HttpServletRequest

Methods in Cookie

Methods	Operation
public void setPath (String uri)	Specifies the path for the cookie. A cookie path must be such that it includes servlet that creates the cookie. e.g. /servlet/cookies
public String getPath()	Returns path of the cookie
public void setMaxAge(int limit)	Used to set life time of a cookie and the life time is specified in seconds and a negative value indicate that cookie will expire when session closes. • c.setMaxAge(60 * 60 * 24 * 365); This cookie will expire after one year • c.setMaxAge(60 * 60 * 24 * 7); The cookie expires after one week
public int getMaxAge()	Returns a integer value which indicates the life of the existing cookie. A negative value indicates that cookie will be deleted as soon as the browsing session is closed. int x = c.getMaxAge(); x contains the maximum age limit of the cookie c.
public void setComment(String str)	Used to set the comment field in the cookie
public String getComment()	Returns the comment field from the cookie. The getComment return null value if cookie has no comment

Creating a Cookie

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DemoCookie extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();

        Cookie c = new Cookie("DemoCookie","123456");
        c.setMaxAge(7*24*60*60);    /* set life of cookie to one week */
        response.addCookie(c);
        pw.println("<HTML><HEAD><TITLE>");
        pw.println("Demo Cookie");
        pw.println("</TITLE></HEAD><BODY>");
        pw.println("<H3>");
        pw.println("The Cookie name is : "+ c.getName());
        pw.println("<BR>");
        pw.println("The Value of Cookie : " + c.getValue
        pw.println("<BR>");
        pw.println("Life of Cookie is : " + c.getMaxAge()
        seconds");
        pw.println("<BR>");
        pw.println("</H3>");
        pw.println("</BODY></HTML>");
        pw.close();
    }
}
```

Searching cookie

- Cookies can be used to check whether the user is the first time visitor or the users has already visited
- To search a specific cookie, all the available cookies need to be checked to find the particular cookie using the getCookies() method of HttpServletRequest

Searching cookie

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FindCookie extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException{
              boolean find = true;
              Cookie [] cookies = request.getCookies();
              if(cookies != null ){
                   for(int i =0; i < cookies.length; i++ ){
                             Cookie c = cookies[i];
                             if (( c.getName(). equals("DemoCookie")) && (c.getValue().equals("123456"))){
                                  find = false;
                                  break;
                             }
              }
}
```

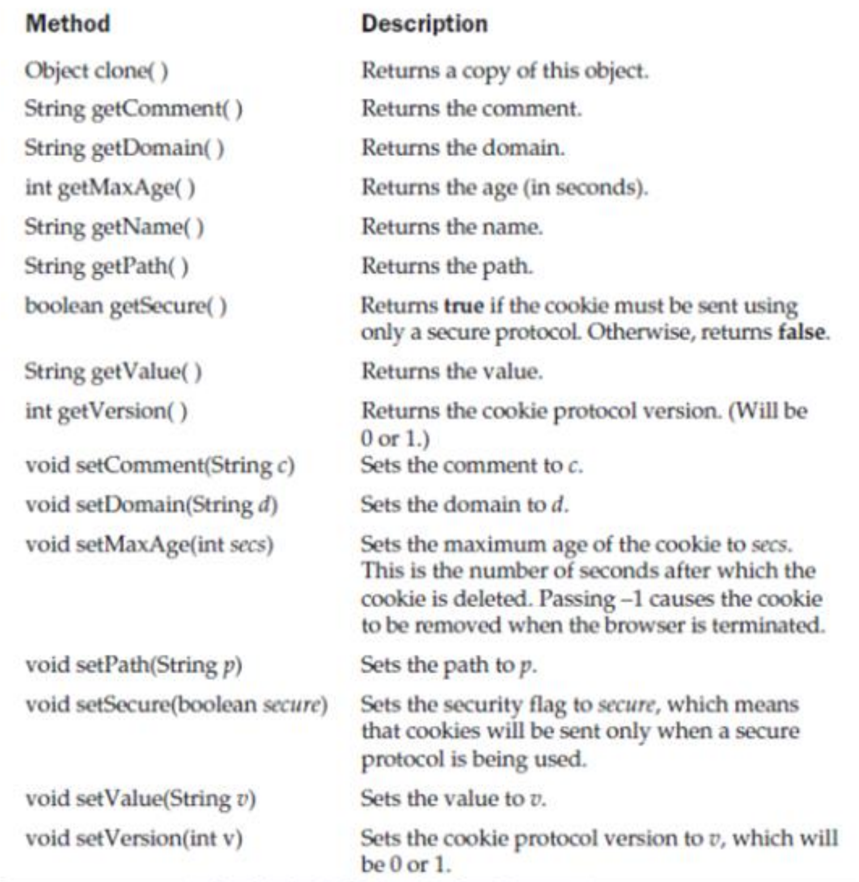
Deleting the cookies

- To delete cookies from local disk start Microsoft Internet Explorer select internet option from Tools menu and press Delete Cookies button of General Tab.
- A cookie can also be deleted with following step.
  1. Set the maximum age of cookie to zero.
  2. Assign null value to cookie

```
Cookie c = new Cookie("JavaCookie");
c.setMaxAge(0);
response.addCookie(c);
```

- First create the cookie of same as to be deleted and set its age to zero and add it to the response header of the HTTP response.

Methods in Cookie



Method	Description
Object clone( )	Returns a copy of this object.
String getComment( )	Returns the comment.
String getDomain( )	Returns the domain.
int getMaxAge( )	Returns the age (in seconds).
String getName( )	Returns the name.
String getPath( )	Returns the path.
boolean getSecure( )	Returns **true** if the cookie must be sent using only a secure protocol. Otherwise, returns **false**.
String getValue( )	Returns the value.
int getVersion( )	Returns the cookie protocol version. (Will be 0 or 1.)
void setComment(String *c*)	Sets the comment to *c*.
void setDomain(String *d*)	Sets the domain to *d*.
void setMaxAge(int *secs*)	Sets the maximum age of the cookie to *secs*. This is the number of seconds after which the cookie is deleted. Passing –1 causes the cookie to be removed when the browser is terminated.
void setPath(String *p*)	Sets the path to *p*.
void setSecure(boolean *secure*)	Sets the security flag to *secure*, which means that cookies will be sent only when a secure protocol is being used.
void setValue(String *v*)	Sets the value to *v*.
void setVersion(int v)	Sets the cookie protocol version to *v*, which will be 0 or 1.

The Methods Defined by Cookie

Session Tracking: HTTPSession

- HTTP is a stateless protocol where each request is independent of the previous one.
- However, in some applications, it is necessary to save state information so that information can be collected from several interactions between a browser and a server. Sessions provide such a mechanism.
- A session can be created via the **getSession( )** method of HttpServletRequest.
- An HttpSession object is returned.
- This object can store a set of bindings that associate names with objects.
- The **setAttribute( ), getAttribute( ), getAttributeNames( ), and removeAttribute( ) methods** of HttpSession manage these bindings.
- It is important to note that session state is shared among all the servlets that are associated with a particular client.

How Do Servlets track The Session?

- Each session has a unique ID
- The first time a session is begun an ID is assigned
- Every subsequent connection must send the ID (through a cookie or the URL)

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

- If null then this is a new session
- Force a new session like this…

```
request.getSession(true);
```

Storing Information in Sessions

- **`setAttribute`**(String name, Object value)
- **`getAttribute`**(String name)
- **`removeAttribute`**(String name)
- **`getAttributeNames`**()

Information About Sessions

- `getId()`
- `isNew()`
- `getCreationTime()`
- `getLastAccessedTime()`
- `getMaxInactiveInterval()`

Java Server Pages (JSP)

- JavaServer Pages (JSP)an extension of servlet technology that separates the presentation from the business logic
- JavaServer Pages simplify the delivery of dynamic Web content
- They enable Web application programmers to create dynamic content by reusing predefined components and by interacting with components using server-side scripting
- Custom-tag libraries are a powerful feature of JSP that allows Java developers to hide complex code for database access and other useful services for dynamic Web pages in custom tags.
- Web sites use these custom tags like any other Web page element to take advantage of the more complex functionality hidden by the tag
- Thus, Web-page designers who are not familiar with Java can enhance Web pages with powerful dynamic content and processing capabilities

Java Server Pages (JSP)

- The classes and interfaces that are specific to JavaServer Pages programming are located in packages **javax.servlet.jsp** and **javax.servlet.jsp.tagext**.
- There are **four key components** to JSPs:

1. **Directives**
2. **Actions**
3. **Scripting elements**
4. **Tag libraries.**

Java Server Pages (JSP)

- **Directives** are messages to the JSP container-the server component that executes JSPs-that enable the programmer to specify page settings, to include content from other resources and to specify custom tag libraries for use in a JSP.
- **Actions** encapsulate functionality in predefined tags that programmers can embed in a JSP.
- Actions often are performed based on the information sent to the server as part of a particular client request. They also can create Java objects for use in JSP scriptlets.
- **Scripting elements** enable programmers to insert Java code that interacts with components in a JSP to perform request processing.
- **Scriptlets**, one kind of scripting element, contain code fragments that describe the action to be performed in response to a user request.
- **Tag libraries** are part of the tag extension mechanism that enables programmers to create custom tags.
- Such tags enable Web page designers to manipulate JSP content without prior Java knowledge.

Scripting

- JavaServer Pages often present dynamically generated content as part of an XHTML document that is sent to the client in response to a request
- In some cases, the content is static but is output only if certain conditions are met during a request (e.g., providing values in a form that submits a request).
- **JSP programmers can insert Java code and logic in a JSP using scripting.**

Scripting Components

- The JSP scripting components include **scriptlets, comments, expressions, declarations and escape sequences**
- **Scriptlets** are blocks of code delimited by **<% and %>.**
- They contain Java statements that the container places in method _jspService at translation time.

Scripting Components

- JSPs support three **comment** styles:
  - **JSP comments,**
  - **XHTML comments and**
  - **scripting-language comments**
- **JSP comments** are delimited by **<%-- and --%>.** These can be placed throughout a JSP, but not inside scriptlets.
- **XHTML comments** are delimited with **<!-- and -->.** These, too, can be placed throughout a JSP, but not inside scriptlets.
- **Scripting language comments** are currently **Java comments**, because Java currently is the only JSP scripting language.
- Scriptlets can use Java's **end-of-line //** comments and traditional comments (delimited by **/* and */).**

Scripting Components

- **JSP expressions** are delimited by **<%= and %>**
- It contain a Java expression that is evaluated when a client requests the JSP containing the expression.
- The container converts the result of a JSP expression to a String object, then outputs the String as part of the response to the client.

Scripting Components

- **Declarations** are delimited by **<%! and %>**
- It enable a JSP programmer to define variables and methods for use in a JSP.
- Variables become instance variables of the servlet class that represents the translated JSP.
- Similarly, methods become members of the class that represents the translated JSP.
- Declarations of variables and methods in a JSP use Java syntax. Thus, a variable declaration must end with a semicolon, as in

```
<%! int counter = 0; %>
```

- Special characters or character sequences that the JSP container normally uses to delimit JSP code can be included in a JSP as literal characters in scripting elements, fixed template data and attribute values using escape sequences.

JSP scriptlet tag

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

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

Examples:

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

JSP scriptlet tag

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

<% java source code %>

<u>**Examples:**</u>

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

index.html

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

welcome.jsp

JSP expression tag

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

<%= statement %>

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

JSP expression tag

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

<%= statement %>

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

index.html

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().get
Time() %>
<%= "Welcome "+request.getParameter("uname") %>
</body>
</html>
```

welcome.jsp

JSP Declaration Tag

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

<%! field or method declaration %**>**

```
<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 Scriptlet tag vs Declaration tag

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

JSP response

- In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.
- It can be used to add or manipulate response such as redirect response to another resource, send error etc.

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

index.html

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

welcome.jsp

JSP request

- The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container.
- It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.
- It can also be used to set, get and remove attributes from the jsp request scope.

```
<form action="welcome.jsp">

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

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

</form>
```

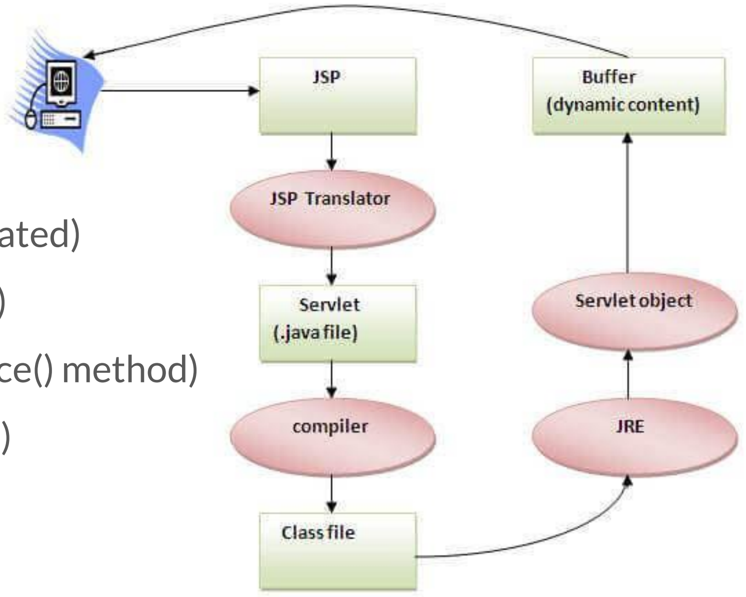
index.html

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

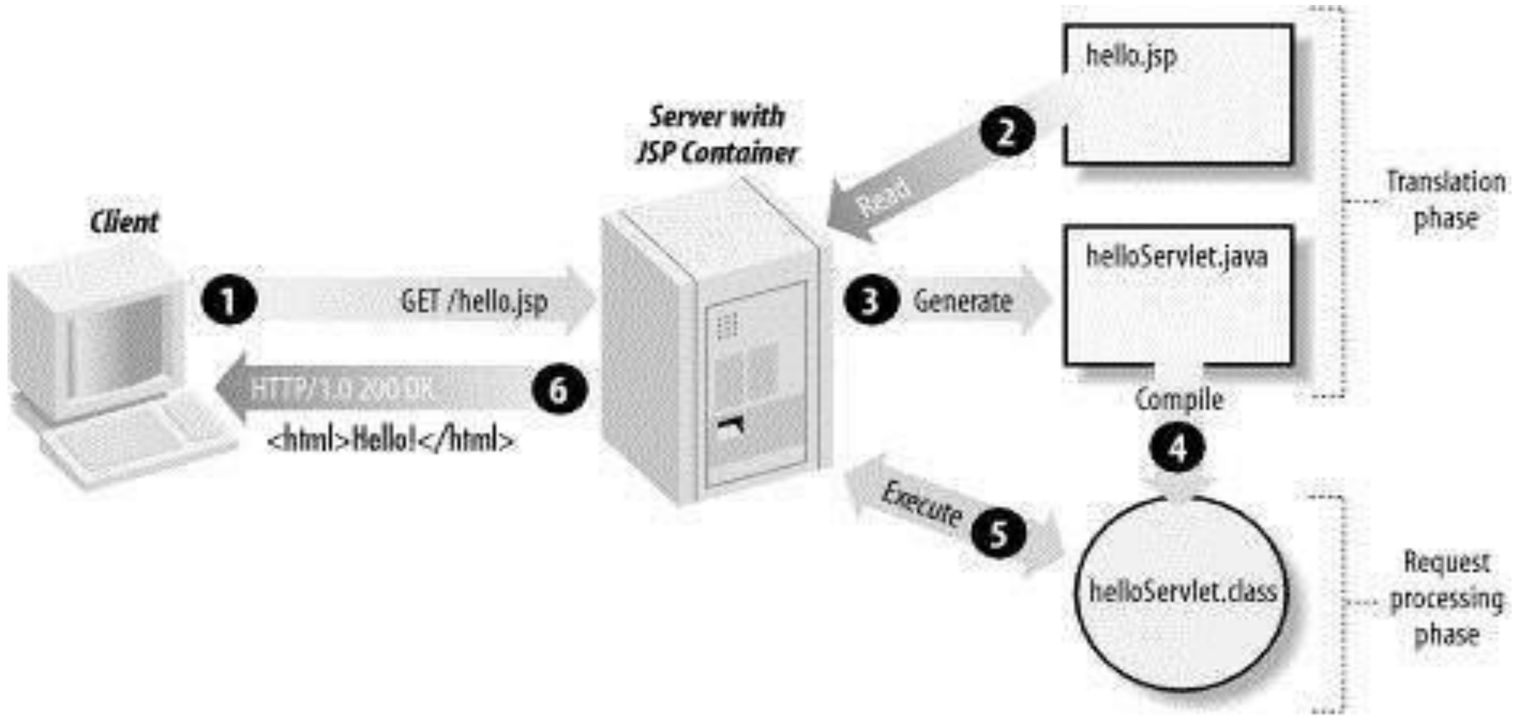
welcome.jsp

Lifecycle of a JSP Page

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created)
- Initialization (the container invokes jspInit() method)
- Request processing (the container invokes _jspService() method)
- Destroy (the container invokes jspDestroy() method)



Lifecycle of a JSP Page



Advantages of JSP

- Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

- Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

- Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

- Less code than Servlet

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.