Servlet Structure, Servlet Packaging, HTML building utilities, Life cycle, Single Threaded Model interface, Handling Client Request: Form Data, Handling Client Request: HTTP Request Headers. Generating server Response: HTTP Status codes, Generating server Response: HTTP Response Headers, Handling Cookies, Session Tracking.

Overview of JSP: JSP Technology, Need of JSP, Benefits of JSP, Advantages of JSP, Basic syntax

----

SERVLETS

- A **servlet** is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.
- The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods
- The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

1.1Servlet Structure

- Class should extend HttpServlet and override doGet or doPost, depending on whether the data is being sent by GET or by POST. Both of these methods take two arguments an **HttpServletRequest** and an **HttpServletResponse**.
- Syntax:

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

public class ServletTemplate extends HttpServlet {

  public void doGet(HttpServletRequest request,
  HttpServletResponse response)throws ServletException, IOException
      {
      // Use "request" to read incoming HTTP headers(eg.cookies)
            and HTML form data(eg. data the user entered and
      submitted).

      // Use "response" to specify the HTTP response status code and
      headers(eg. the content type, cookies).

         PrintWriter out = response.getWriter();

      // Use "out" to send content to browser
  }
}
```

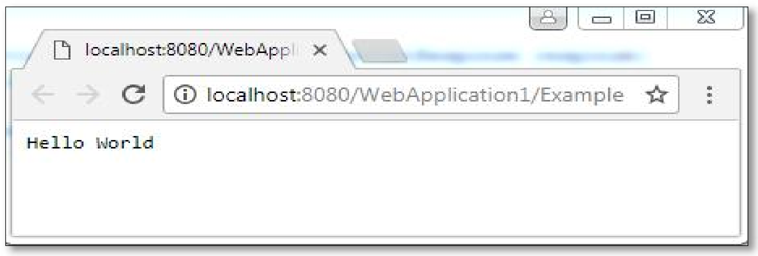
- The **HttpServletRequest** has methods by which you can find out about incoming information:
    - such as form data,
    - HTTP request headers,
    - client's hostname.

- The **HttpServletResponse** lets you specify outgoing information:
    - such as HTTP status codes (200, 404, etc.),
    - response headers (Content-Type, Set-Cookie, etc.),
    - **PrintWriter** used to send the document content back to the client.

- ***Example: Simple Servlet Generating (HelloWorld.java)***

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

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

  PrintWriter out = response.getWriter();
  out.println("Hello World");
  }
}
```

Output:



```
Hello World
```

1.2Servlet Packaging

- Placing all the servlets in the same directory results in massive, hard-to-manage collection of classes and risks name conflicts when two developers choose same name for a servlet or a utility class
- When you put your servlets in packages, perform the following two additional steps.
    - **Place the files in a subdirectory that makes the intended package name:**
        - Example: all class files should go under directory called as `coreservlets`
    - **Insert a package statement in the class files:**
    - The class files should in some package.
    - Example: package coreservlets;

1.3 HTML building utilities

- An HTML document is structured as follows

```
<!DOCTYPE ...>
<HTML>
     <HEAD><TITLE>...</TITLE>...</HEAD>
          <BODY ...>
                    ...
          </BODY>
</HTML>
```

- **Advantage of have line <! DOCTYPE>**
    - o It tells HTML validators which version of HTML you are using
    - o These validators are valuable debugging services, such as syntax errors in HTML
- **The two most popular online validators are**
    - o **World Wide Web Consortium (**http://validator.w3.org**)**
    - o **The Web Design Group** (http://www.htmlhelp.com/tools/validator/)

- They let you submit a URL, then they retrieve the page, check the syntax against the formal HTML specification, and report any errors to you. Since a servlet that generates HTML looks like a regular Web Page to visitors
- To generate HTML with ***println*** statements, especially long tedious lines like the DOCTYPE declaration. Some people address this problem by writing detailed HTML generation utilities in Java, then use them throughout their servlets. Still, have the problems listed below:
    - o Its inconvenience of generating HTML programmatically
    - o HTML generation routines can be cumbersome and tend not to support the full range of HTML attributes (CLASS and ID for style sheets, JavaScript event handlers, table cell background colors, and so forth).

1.4 Servlet Life cycle

- The life cycle of a servlet instance

    1. **Load Servlet class**
    2. **Create servlet instance**
    3. **Call to the init( ) method**
    4. **Call to the service( ) method**
    5. **Call to the destroy( ) method**

➢ **Load Servlet class:**
- o A Servlet class is loaded when first request for the servlet is received by the Web Container.

➢ **Create servlet instance**

- After the Servlet class is loaded, Web Container creates the instance of it. Servlet instance is created only once in the life cycle.

- **Call to the init( ) method**
  - init( ) method is called by the Web Container on servlet instance to initialize the servlet.
  - It is used for one-time initialization, just as in applets
  - There are two init( ) methods
    - init( )
    - init(SefvletConfig config)

  - init( )
    - init simply creates or loads some data that will be used throughout the life cycle.
    - The first version is used when the servlet does not need to read any settings that vary from server to server.
    - Syntax:

```
public void init() throws ServletException {
 // Initialization code...
}
```

    - The second version of init( ) is used when the servlet needs to *read server-specific settings* before it can complete the initialization.

    - Example: about database settings, password files, serialized cookies

```
public void init(ServletConfig config) throws
ServletException  {
     super.init(config);
     // Initialization code...
}
```

    - Notice two things about this code.
    - First, the **init** method takes a ServletConfig as an argument. *ServletConfig* has a *getInitParameter* method with which you can look up initialization parameters associated with the servlet.
    - Second thing is that the first line of the method body is a call to*super.init(config)*. This method has a parameter ServletConfig object and always call the init method of the superclass registers it where the servlet can find it later.

- **Call to the service() method**
  - Each time the server receives a request for a servlet, the server spawns a new thread and calls `service.`
  - The service method checks the HTTP request type (GET, POST, DELETE, PUT) and calls doGet, doPost, doPut, doDelete etc,

- **Syntax:**

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

 IOException {

     //servlet code

}
```

- **doGet method Syntax:**

```
public void doGet(HttpServletRequest request,
  HttpServletResponse response)throws ServletException,
IOException
{
     // Servlet Code
}
```

- **doPost method Syntax:**

```
public void doPost(HttpServletRequest request,
  HttpServletResponse response)
    throws ServletException, IOException
{
      // Servlet Code
}
```

➢ **Call to the destroy( ) method**
- The server calls a servlet's destroy() method when the servlet is about to be unloaded.
- In the destroy( ) method, a servlet should free any resources it has acquired that will not be garbage collected.
- The destroy( ) method also gives a servlet a chance to write out its unsaved cached information or any persistent information that should be read during the next call to init( ).
- **Syntax:**

```
public void destroy()
```

1.5Single Threaded Model

- It is an interface from **javax.servlet** package used with Servlets. It is marker interface having no methods. Few servlets may require this to implement.
- With multiple requests for the same servlet, in the Container multiple threads will be active within the process.
- If the Programmer would like to have only one thread active at a time (other threads, if exist, must be passivated or made inactive) then he implements the **SingleThreadModel interface**and it being marker interface no methods need to be overridden.
- The system makes single instance of your servlet and then creates a new thread for each user request with multiple concurrent threads running, if a new request comes in while a previous request still executing.
- This means that your doGet() and doPOst() methods must be careful to synchronize access to fields and other shared data, since multiple thread may access the data simultaneously
- **General form**:

```
public YourServlet extends HttpServlet implements SingleThreadModel {
    //……
}
```

- If a servlet implements this interface, you are *guaranteed* that no two threads will execute concurrently in the servlet's service method.
- It queue’s all the request and passing them one at a time to a single servlet instances
- The server is permitted to create a pool of multiple instances, each of which handles one request at a time.

Note: SingleThreadModel does not solve all thread safety issues

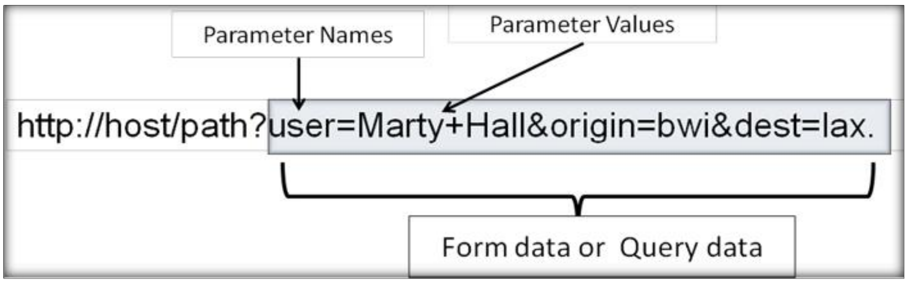
- o For example, session attributes and static variables can still be accessed by multiple requests on multiple threads at the same time, even when SingleThreadModel servlets are used.
- o It is recommended that a developer take other means to resolve those issues instead of implementing this interface, such as avoiding the usage of an instance variable or synchronizing the block of the code accessing those resources.

1.6 Handling Client Request

- Whenever we want to send an input to a servlet that input must be passed through html form.
- An html form is nothing but various controls are inherited to develop an application.
- Every form will accept client data end it must send to a servlet which resides in server side.
- Since html is a static language which cannot validate the client data. Hence, in real time applications client data will be accepted with the help of html tags by developing form and every form must call a servlet.

1.6.1 Form Data:

- URL's like



- The part after the question mark(i.e., user=Marty+Hall&origin=bwi&dest=lax) is known as *form data* (or query data)
- Form data can be attached to the end of the URL after a question mark for GET requests, or sent to the server on a separate line, for POST requests.
- Extracting the form data from CGI programming
  - *First*, read data one way for GET request and different way for POST request.
  - *Second*, you have to chop the pairs at the ampersands, then separate the parameter names from parameter values
  - *Third*, you have to URL-decode the values. Alphanumeric characters are sent unchanged
  - *Fourth*, reason that parsing form data is tedious is that values can be omitted. (e.g., "**param1=val1**&param2=val2&**param1=val3**")

1.6.2 Reading form data from Servlets

- One of the nice features of servlets is that all of this form parsing is handled automatically.
- Reading data from servlets:
  - getParameter( )
  - getParameterValues( )
  - getParameterNames( )

➢ getParameter()
  - `getParameter` exactly the same way when the data is sent by `GET` as you do when it is sent by `POST`.
  - An empty `String` is returned if the parameter exists but has no value, and `null` is returned if there was no such parameter.

```
<html>
    <head>
        <title>Collecting Three Parameters</title>
    </head>
      <body>
           <form action="ParmeterServlet" method="Get">

           First Parameter:<input type="text" name="param1"/><br>
      Second Parameter:<input type="text"name="param2"/><br>
          Third Parameter: <input type="text" name="param3"/><br>

           <input type="submit" value="Click Here">
           </form>
      </body>
</html>
```

ParameterServlet.java

```
public class ParmeterServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request,
  HttpServletResponse response)throws ServletException, IOException {
        PrintWriter out=response.getWriter();
        String title = "Reading Three Request Parameters";

            out.println("Parameter1:+request.getParameter(“param1”));
      out.println("Parameter1:+request.getParameter(“param2”));
      out.println("Parameter1:+request.getParameter(“param3”));

      } }
```

- getParameterValues( )
  - If the parameter have more than one value, eg: checkbox
    - which returns an array of strings
    - The return value of `getParameterValues` is `null`, for nonexistent parameter names and is a one-element array when the parameter has only a single value.
  - **General form:**

```
String[] values = getParameterValues(“Input Parameter”);
```

- Example:

Index.html

```
     <form action="ongetParameterVlaues" method="post">
     Habits :
        <input type="checkbox" name="habits" value="Reading">Reading
        <input type="checkbox" name="habits" value="Movies">Movies
        <input type="checkbox" name="habits" value="Writing">Writing
        <input type="checkbox" name="habits" value="Singing">Singing
        <input type="submit" value="Submit">
     </form>
```

OngetParameterValues.java

```
public class OngetParameterValues extends HttpServlet
{
  protected void doPost(HttpServletRequest request, HttpServletResponse
  response)throws ServletException,IOException
    {
        PrintWriter out=res.getWriter();
        response.setContentType("text/html");

        String[] values=request.getParameterValues("habits");
        out.println("Selected Values...");

            for(int i=0;i<values.length;i++)
            {
              out.println("<li>"+values[i]+"</li>");
            }
          out.close();
    }
}
```

- getParameterNames( )
  - to get a full list
  - to get this list in the form of an Enumeration, each entry of which can be cast to a String and used in a getParameter or getParameterValues call.
  - Example:

Index.html

```
    <form action="onPM" method="post">
        Name:<input type="text" name="name">
        Country:<input type="text" name="country">
        <input type="submit" value="Submit">
     </form>
```

OngetParameterNames.java

```
public class OngetParameterNames extends HttpServlet
{
    protected void doPost(HttpServletRequest request,
HttpServletResponse response)throws ServletException,IOException
 {
     PrintWriter out=response.getWriter();
     response.setContentType("text/html");

      Enumeration en=request.getParameterNames( );

      while(en.hasMoreElements())
      {
           String parameterName =(String)en.nextElement();
           out.println("Parameter = "+parameterName);
     }
      out.close();
     }
}
```

Note: Parameter names are case sensitive so, for example, `request.getParameter("Param1")` and `request.getParameter("param1")` are *not* interchangeable.

1.7 Handling Client Request: HTTP Request Headers

- The HTTP information that is sent from the browser to the server in the form of request headers.
- HTTP request headers are distinct from the form data.
- Form data results directly from user input and is sent as part of the URL for `GET` requests and on a separate line for `POST` requests.
- Request headers, on the other hand, are indirectly set by the browser and are sent immediately following the initial `GET` or `POST` request line.

1.7.1 Reading Request Headers from Servlets

- Header names are not case sensitive
- The list of headers that are generally used

Header names	Description
`getCookies`	The getCookies method returns the contents of the Cookie header, parsed and stored in an array of Cookie objects.
getAuthType() and getRemoteUser()	break the Authorization header into its component pieces.
getContentLength()	returns int value of the Content-Length header
getContentType()	Returns string value of the Content-Type header
getDateHeader() getIntHeader()	read the specified header and then convert them to Date and int values
getHeaderNames()	to get an Enumeration of the values of all occurrences of the header names received on this particular request.
getHeaders()	If a header name is repeated in the request, • version 2.1 servlets cannot access: returns the value of the first occurrence of the header only • version 2.2, returns an Enumeration of the values of all occurrences of the header.
getMethod()	returns the main request method(Get, Post . . . .)
getRequestURI()	returns the part of the URL that comes after the host and port but before the form data
getProtocol()	returns the third part of the request line, [HTTP/1.0 or HTTP/1.1]

1.7.2 Printing all headers

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

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

      String title = "Servlet Example: Showing Request Headers";
      out.println(ServletUtilities.headWithTitle(title) + "<BODY>\n" +
          "<H1>" + title + "</H1>\n" +
          "<B>Request Method: </B>"+request.getMethod() +"<BR>\n" +
          “<B>Request URI: </B>"+request.getRequestURI()+"<BR>\n" +
           “<B>Request Protocol: </B>" +request.getProtocol());

          Enumeration <String> headerNames= request.getHeaderNames();
          while(headerNames.hasMoreElements()){
               String headerName = headerNames.nextElement();
               out.println(“Header Name: <em>”+headerName);
               String headerValue = request.getHeader(headerName);
               out.print(“</em>, Header Value: <em>”+ headerValue);
               out.println(“</em><br/>”);
               out.println("</BODY></HTML>");} }
```

1.7.2 HTTP 1.1 Request Headers

- Access to the request headers permits servlets to perform a number of optimizations and to provide a number of features not otherwise possible.

Header Names	Description
Accept	specifies the MIME types that the browser or other client can handle. Returns more than one format
Accept-Charset	indicates the character sets (e.g., ISO-8859-1) the browser can use.
Accept-Encoding	designates the types of encodings that the client knows how to handle. If it receives this header, the server is free to encode the page by using the format, sending the *Content-Encoding* response header to indicate that it has done so
Accept-Language	specifies the client's preferred languages, in case the servlet can produce results in more than one language. The value of the header should be one of the standard language codes such as en, en-us, da, etc.
Authorization	is used by clients to identify themselves when accessing password-protected Web pages.
Cache-Control	used by the client to specify a number of options for how pages should be cached by proxy servers.
Connection	tells whether or not the client can handle persistent HTTP Connections.
Content-Length	only applicable to POST requests and gives the size of the POST data in bytes. Simply use request.getContentLength().
Content-Type	It is used in responses *from* the server, it can also be part of client requests when the client attaches a document as the POST data or when making PUT requests. You can access this header with the shorthand getContentType method of HttpServletRequest.
Cookie	This header is used to return cookies to servers that previously sent them to the browser.
Expect	This rarely used header lets the client tell the server what kinds of behaviors it expects.
From	This header gives the e-mail address of the person responsible for the HTTP request.
Host	Browsers are required to specify this header, which indicates the host and port as given in the *original* URL.
If-Match	This rarely used header applies primarily to PUT requests. The client can supply a list of entity tags as returned by the ETag response header, and the operation is performed only if one of them matches.
If-Modified-Since	This header indicates that the client wants the page only if it has been changed after the specified date. This option is very useful because it lets browsers cache documents and reload them over the network only when they've changed.
Pragma	A Pragma header with a value of no-cache indicates that a servlet that is acting as a proxy should forward the request even if it has a local copy.
Proxy-Authorization	This header lets clients identify themselves to proxies that require it. Servlets typically ignore this header, using Authorization instead.
Range	This rarely used header lets a client that has a partial copy of a document ask for only the parts it is missing.

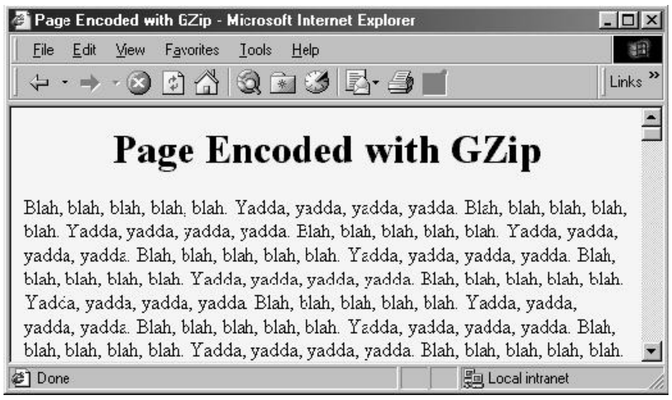
Referer	This header indicates the URL of the referring Web page. it is a useful way of tracking where requests came from
Upgrade	The Upgrade header lets the browser or other client specify a communication protocol it prefers over HTTP 1.1
User-Agent	This header identifies the browser or other client making the request.
Via	This header is set by gateways and proxies to show the intermediate sites the request passed through.
Warning	This rarely used catchall header lets clients warn about caching or content Transformation errors.

1.7.3 Sending Compressed Web pages

- Gzip is a text compression scheme that can dramatically reduce the size of HTML pages.
- Several recent browsers know how to handle gzipped content, so the server can compress the document and send the smaller document over the network, after which the browser will automatically reverse the compression and treat the result in the normal manner
- Sending such compressed content can be a real timesaver, since the time required to compress the document on the server and then uncompress it on the client is typically dwarfed by the savings in download time, especially when dialup connections are used.
- To send compressed content, a servlet must send a Content-Encoding header to tell the client scheme by which the content has to be encoded
    - Browsers that support this feature indicate that they do so by setting the `Accept-Encoding` request header.
- **Implementing compression**
    - `java.util.zip:` package contains classes that support reading and writing the GZIP and ZIP compression formats.
    - The servlet first checks the `Accept-Encoding` header to see if it contains an entry for gzip.
        - If so, it uses a `GZIPOutputStream` to generate the page, specifying `gzip` as the value of the `Content-Encoding` header.
        - explicitly call `close` when using a `GZIPOutputStream`.
        - If gzip is not supported, the servlet uses the normal `PrintWriter` to send the page
    - Compression could be suppressed by including `?encoding=none` at the end of the URL.

```
public class EncodedPage extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
          response.setContentType("text/html");

      String encodings = request.getHeader("Accept-Encoding");
      String encodeFlag = request.getParameter("encoding");

      PrintWriter out;

      String title;

      if ((encodings != null) && (encodings.indexOf("gzip") != -1) &&
      !"none".equals(encodeFlag)) {

            title = "Page Encoded with GZip";
            OutputStream out1 = response.getOutputStream();
            out = new PrintWriter(new GZIPOutputStream(out1), false);

            response.setHeader("Content-Encoding", "gzip");
            } else {
                  title = "Unencoded Page";
                  out = response.getWriter();
            }

      out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n");

      String line = "Blah, blah, blah, blah, blah. " +
                  "Yadda, yadda, yadda, yadda.";

          for(int i=0; i<10000; i++) {
                  out.println(line);
      }
     out.println("</BODY></HTML>");
    out.close();
  }
}
```

Output:


Page Encoded with GZip - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Done
Local intranet

Page Encoded with GZip

Blah, blah, blah, blah, blah. Yadda, yadda, yadda, yadda. Blah, blah, blah, blah, blah. Yadda, yadda, yadda, yadda. Blah, blah, blah, blah, blah. Yadda, yadda, yadda, yadda. Blah, blah, blah, blah, blah. Yadda, yadda, yadda, yadda. Blah, blah, blah, blah, blah. Yadda, yadda, yadda, yadda. Blah, blah, blah, blah, blah. Yadda, yadda, yadda, yadda. Blah, blah, blah, blah, blah. Yadda, yadda, yadda, yadda. Blah, blah, blah, blah, blah. Yadda, yadda, yadda, yadda. Blah, blah, blah, blah, blah.

1.7.4 Restricting Access to Web

- Many Web servers support standard mechanisms for limiting access to designated Web pages.
- These mechanisms can apply to static pages as well as those generated by servlets, so many authors use their server-specific mechanisms for restricting access to servlets.
- Once a servlet that uses form-based access grants initial access to a user, it would use session tracking to give the user access to other pages that require the same level of authorization.
- Form-based access control requires more effort on the part of the servlet developer, and HTTP-based authorization is sufficient for many simple applications.
- The steps involved for “basic” authorization.
- **Step1:**
    - Check whether there is an Authorization header. If there is no such header, go to Step 2.
    - If there is, skip over the word “basic” and reverse the base64 encoding of the remaining part.
    - This results in a string of the form **username:password**. Check the username and password against some stored set. If it matches, return the page. If not, go to Step 2.
- **Step 2:**
    - Return a 401 (Unauthorized) response code and a header of the following form:
    - WWW-Authenticate: BASIC realm="some-name" This response instructs the browser to pop up a dialog box telling the user to enter a name and password for some-name, then to reconnect with that username and password embedded in a single base64 string inside the Authorization header.

1.8 Generating server Response: HTTP Status codes

- When a web server responds to a request from a browser or other wen client, the response typically consists of a status line, some response headers, a blank line, and the document
- **Example:**

```
HTTP/1.1 200 OK
Content-Type: text/plain

Hello World
```

1.8.1 Specifying Status codes

- The HTTP response status line consists of an HTTP version, a status code, and an associated message.
- The message is directly associated with the status code and the HTTP version is determined by the server, all the servlet needs to do is to set the status code.
- The way to do this is by the setStatus method of HttpServletResponse.
- If your response includes a special status code and a document, be sure to call setStatus before actually returning any of the content via the PrintWriter.

- The setStatus method takes an int (the status code) as an argument, but instead of using explicit numbers, it is clearer and more reliable to use the constants defined in HttpServletResponse.
- There are two common cases where a shortcut method in `HttpServletResponse` is provided. Just be aware that both of these methods throw `IOException`, whereas `setStatus` doesn't.

    - **`public void sendError(int code, String message)`**
        - The `sendError` method sends a status code (usually 404) along with a short message that is automatically formatted inside an HTML document and sent to the client.

    - **`public void sendRedirect(String url)`**
        - The `sendRedirect` method generates a 302 response along with a `Location` header giving the URL of the new document. With servlets version 2.1, this must be an absolute URL. In version 2.2, either an absolute or a relative URL is permitted and the system automatically translates relative URLs into absolute ones before putting them in the `Location` header.
- Setting a status code does not necessarily mean that you don't need to return a document. For example, although most servers automatically generate a small "File Not Found" message for 404 responses, a servlet might want to customize this response. Remember that if you do send output, you have to call `setStatus` or `sendError` *first*.

- The following sections describe each of the **status codes** available for use in servlets talking to HTTP 1.1 clients, along with the standard message associated with each code.

- These codes fall into five general categories:

Status Code	**Description**
100-199	Codes in the 100s are informational, indicating that the client should respond with some other action.
200-299	Values in the 200s signify that the request was successful
300-399	Values in the 300s are used for files that have moved and usually include a `Location` header indicating the new address.
400-499	Values in the 400s indicate an error by the client.
500-599	Codes in the 500s signify an error by the server.

- You should only send the new codes to clients that support HTTP 1.1, as verified by checking **request.getRequestProtocol.**

HTTP/1.1 status code

Status code & Message	**Constant Name**	**Description**
100 (Continue)	`SC_CONTINUE`	If the server receives an `Expect` request header with a value of `100-continue`, it means that the

		client is asking if it can send an attached document in a follow-up request.
101 (Switching Protocols)	`SC_SWITCHING_PROTOCOLS`	status indicates that the server will comply with the `Upgrade` header and change to a different protocol. This status code is new in HTTP 1.1.
200 (OK)	`SC_OK`	A value of 200 means that everything is fine.
201(Created)	`SC_CREATED`	signifies that the server created a new document in response to the request; the `Location` header should give its URL.
202 (Accepted)	`SC_ACCEPTED`	tells the client that the request is being acted upon, but processing is not yet complete.
203 (Non-Authoritative Information)	`SC_NON_AUTHORITATIVE_INFORMATION`	status signifies that the document is being returned normally, but some of the response headers might be incorrect since a document copy is being used
204 (No Content)	`SC_NO_CONTENT`	stipulates that the browser should continue to display the previous document because no new document is available.
205 (Reset Content)	`SC_RESET_CONTENT`	means that there is no new document, but the browser should reset the document view.
400 (Bad Request)	`SC_BAD_REQUEST`	status indicates bad syntax in the client request.
401 (Unauthorized)	`SC_UNAUTHORIZED`	signifies that the client tried to access a password-protected page without proper identifying information in the `Authorization` header. The response must include a `WWW-Authenticate` header.
403(Forbidden)	`SC_FORBIDDEN`	means that the server refuses to supply the resource, regardless of authorization
404 (Not Found)	`SC_NOT_FOUND`	status tells the client that no resource could be found at that address. This value is the standard “no such page” response.
502 (Bad Gateway)	`SC_BAD_GATEWAY`	used by servers that act as proxies or gateways; it indicates that the initial server got a bad response from the remote server.
503 (Service Unavailable)	`SC_SERVICE_UNAVAILABLE`	signifies that the server cannot respond because of maintenance or overloading
504 (Gateway Timeout)	`SC_GATEWAY_TIMEOUT`	is used by servers that act as proxies or gateways; it indicates that the initial server didn’t get a timely response from the remote server.

1.9 Generating server Response: HTTP Response Headers

- The most general way to specify headers is to use the *setHeader* method of HttpServletResponse. This method takes two strings: *the header name* and *the header value.*
- **Syntax:**

```
public void setHeader(string headername, int  headervalue)
```

- setHeader method, HttpServletResponse also has two specialized methods to set headers that contain dates and integers
  - **setDateHeader(String header, long milliseconds)**
  - **setIntHeader(String header, int headerValue)**

- `HttpServletResponse` also supplies a number of convenience methods for specifying common headers. These methods are summarized as follows.

Methods	Meaning
setContentType	This method sets the Content-Type header and is used by the majority of servlets.
setContentLength	This method sets the Content-Length header, which is useful if the browser supports persistent (keep-alive) HTTP connections.
addCookie	This method inserts a cookie into the Set-Cookie header. There is no corresponding setCookie method, since it is normal to have multiple Set-Cookie lines.
sendRedirect	The sendRedirect method sets the Location header as well as setting the status code to 302.

1.9.1 HTTP 1.1 Response Headers and their Meaning

The possible HTTP1.1 response headers along with a brief summary of how servlets can make use of them.

Header Name	Meaning
Accept-Ranges	tells the client whether or not you accept Range request headers.
Age	is used by **proxies** to indicate how long ago the document was generated by the original server.
Allow	specifies the request methods (GET, POST, etc.)
Cache-Control & Pragma	• the circumstances in which the response document can safely be cached. • It can have values **public, private** or **no-cache.**
	• **Private** means document is for a single user and can only be stored in private caches • **no-cache** means document should never be cached. • `response.setHeader("Cache-Control","no-cache");` • `response.setHeader("Pragma",  "no-cache");`
Connection	instructs the browser whether to use persistent in HTTP connections or not. `Connection: keep-alive`
Content-Encoding	indicates the way in which the page was encoded during transmission.
Content-Language	This header signifies the language in which the document is written. Example: en, en-us, ru, etc.
Content-Length	indicates the number of bytes in the response.
Content-Location	supplies an alternative address for the requested document. Content-Location is **informational**;
Content-Range	is sent with partial-document responses and specifies how much of the total document was sent
Content-Type	• gives the MIME type of the response document. • The default MIME type for servlets is text/plain

	Example: application/zip:- Zip archive image/gif:- GIF image text/html:- HTML document video/mpeg:- MPEG video clip
Date	specifies the current date in GMT format. Example: `response.setHeader("Date", "...");`
ETag	gives names to returned documents so that they can be referred to by the client later

Expires	• The time at which document should be considered out-of-date and thus should no longer be cached.
	• Use setDateHeader() to set this header
	Example:
	`long currentTime = System.currentTimeMillis();`
	`long tenMinutes = 10*60*1000; // In milliseconds`
	`response.setDateHeader("Expires",`
	`currentTime`
	`+tenMinutes);`

Last-Modified	When time document was last changed.
Location	should be included with all responses that have a status code in the 300s. The URL to which browser should reconnect. Use sendRedirect instead of setting this directly

Refresh	The number of seconds until browser should reload page. Can also include URL to connect to.
	`response.setIntHeader("Refresh", 30)`
	`response.setHeader("Refresh","5; URL=http://host/path")`

Set-Cookie	This header specifies a cookie associated with the page.
Server, Retry –After, Trailer, Transfer- Encoding, WWW-Authenticate	

1.10 Handling Cookies

- Cookies are small bits of textual information that a web server sends to a browser and that browser returns unchanged when later visiting the same website or domain.
- **Cookies** are text files stored on the client computer and they are kept for various information tracking purpose.
- Java Servlets transparently supports HTTP cookies.

1.10.1 Benefits of cookies

- Four typical ways in which cookies can add the value

➢ **Identifying a user during an e-commerce**
  - o By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we identify the user as the old user.

➢ **Avoiding username and password**
  - o Many large sites require you to register in order to use their services, but it is inconvenient to remember and enter the username and password each time you visit. Cookies are a good alternative for low-security sites. When a user registers, a cookie containing a unique user ID is sent to him. When the client reconnects at a later date, the user ID is returned, the server looks it up,

determines it belongs to a registered user, and permits access without an explicit username and password.

- **Customizing a site**
  - Many “portal” sites let you customize the look of the main page. They might let you pick which weather report you want to see, what stock and sports results you care about, how search results should be displayed, and so forth. Since it would be inconvenient for you to have to set up your page each time you visit their site, they use cookies to remember what you wanted. For simple settings, this customization could be accomplished by storing the page settings directly in the cookies.

- **Focusing advertising**
  - Most advertiser-funded Web sites charge their advertisers much more for displaying “directed” ads than “random” ads. Advertisers are generally willing to pay much more to have their ads shown to people that are known to have some interest in the general product category. For example, if you go to a search engine and do a search on “Java Servlets,” the search site can charge an advertiser much more for showing you an ad for a servlet development environment than for an ad for an on-line travel agent specializing in Indonesia.

1.10.2 Problems with cookies

- Cookies are not a serious security threat
  - they can present a significant threat to *privacy*
- Cookies are never interpreted or executed in any way and thus cannot be used to insert viruses or attack your system.
- Since browsers generally only accept 20 cookies per site and 300 cookies total and since each cookie can be limited to 4 kilobytes, cookies cannot be used to fill up someone’s disk or launch other denial of service attacks.
- ***Problems:***
  - **First**, Some people don’t like the fact that search engines can remember that they’re the user who usually does searches on certain topics. Example: search for job openings or sensitive health data and don’t want some banner ad tipping off their coworkers next time they do a search.
  - **Second**, privacy problem occurs when sites rely on cookies for overly sensitive data. For example, some of the big on-line bookstores use cookies to remember users and let you order without reentering much of your personal information.

1.10.3 Cookie API

Creating Cookie

- Call the Cookie constructor with a cookie name and a cookie value, both of which are strings
- **javax.servlet.http.Cookie** class provides the functionality of using cookies.
- **Constructor of cookie class**

Constructors	**Description**
Cookie( )	Construct cookie
Cookie(String name, String value)	Constructs a cookie with a specified name and value

- **Syntax**:
```
Cookie obj_name = new Cookie(“name”, “value”);
```
- Neither the name nor the value should contain white space or any of the following characters: [ ] ( ) = , " / ? @ : ;

Cookie Attribute

- Before adding the cookie to the outgoing headers, you can set various characteristics of the cookie by using one of the following
  - set***Xxx*** methods, where ***Xxx*** is the name of the attribute you want to specify.
  - get***Xxx*** method to retrieve the attribute value.

Methods Name	**Description**
public String getComment( ) **public void setComment(String comment)**	These methods look up or specify a comment associated with the cookie. The comment is used purely for informational purposes on the server; it is not sent to the client
public String getDomain( ) **public void setDomain(String domainPattern)**	These methods get or set the domain to which the cookie applies. the browser only returns cookies to the exact same hostname that sent them.
public int getMaxAge( ) **public void setMaxAge(int lifetime)**	These methods tell how much time (in seconds) should elapse before the cookie expires. A negative value, which is the default, indicates that the cookie will last only for the current session and will not be stored on disk. Specifying a value of 0 instructs the browser to delete the cookie.
public String getName( ) **public void setName(String cookieName)**	This pair of methods gets or sets the name of the cookie. The name and the value are the two pieces you virtually *always* care about.
public String getPath( ) **public void setPath(String path)**	These methods get or set the path to which the cookie applies. If you don’t specify a path, the browser returns the cookie only to URLs in or below the directory containing the page that sent the cookie.
public boolean getSecure( ) **public void setSecure(boolean secureFlag)**	This pair of methods gets or sets the Boolean value indicating whether the cookie should only be sent over encrypted (i.e., SSL) connections. The default is false; the cookie should apply to all connections.
public String getValue( ) **public void setValue(String cookieValue)**	The getValue method looks up the value associated with the cookie; The setValue method specifies it.
public int getVersion( ) **public void setVersion(int version)**	These methods get/set the cookie protocol version the cookie complies with. Version 0, the default,

	follows the original Netscape specification Version 1, not yet widely supported

Creating cookie and placing in response headers

```
Cookie usercookie =new Cookie("user","123l");    //creating cookie object
usercookie.setMaxAge(60*60*24*365);   //setting for 1 year
response.addCookie(ck); //adding cookie in the response
```

Reading cookies from the client

- getCookies used to read the cookies.
    - This call returns an array of Cookie objects corresponding to the values that came in on the Cookie HTTP request header
    - Returns null if there are no cookies

```
Cookie[ ] cookies = request.getCookies();
```

- Now you can iterate through the array of cookies and find the cookies you need. Unfortunately there is no way to obtain a cookie with a specific name. The only way to find that cookie again is to iterate the Cookie[ ] array and check each cookie name. Here is an example:

```
Cookie[ ] cookies = request.getCookies( );
for(int i=0;i<ck.length;i++){
 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());
}
```

Removing Cookies

- The simple code to delete cookie

```
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
```

1.10. 4 Examples of setting and Reading Cookies

index.html

```
<form method="post" action="MyServlet">
       Name:<input type="text" name="user" /><br/>
       Password:<input type="text" name="pass" ><br/>
       <input type="submit" value="submit">
</form>
```

MyServlet.java

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

public class MyServlet extends HttpServlet {

  protected void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        String name = request.getParameter("user");
        String pass = request.getParameter("pass");

        if(pass.equals("1234"))
        {
            Cookie ck = new Cookie("username",name);
            response.addCookie(ck);
            response.sendRedirect("First");
        }
    }
}
```

First.java

First.java

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

public class First extends HttpServlet {

  protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

       Cookie[] cks = request.getCookies();
        out.println("Welcome "+cks[0].getValue());
    }
}
```

1.10.5 Basic Cookie Utilities

Finding Cookies with specified names

- Simplifies the retrieval of a cookie or cookie value, given a cookie name. The getCookieValue method loops through the array of available Cookie objects, returning the value of any Cookie whose name matches the input. If there is no match, the designated default value is returned.

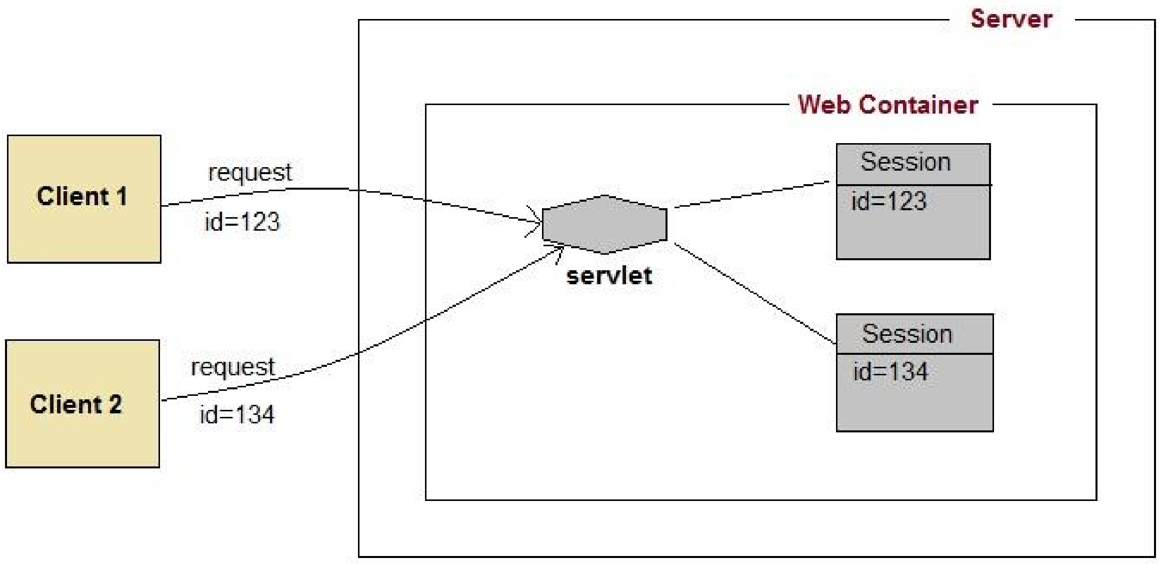
```
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletUtilities extends HttpServlet {

  public static String getCookieValue(Cookie[] cookies,
                                      String cookieName,
                                      String defaultValue) {
    for(int i=0; i<cookies.length; i++) {
      Cookie cookie = cookies[i];
      if (cookieName.equals(cookie.getName()))
        return(cookie.getValue());
    }
    return(defaultValue);
  }
  public static Cookie getCookie(Cookie[] cookies,
                                 String cookieName) {
    for(int i=0; i<cookies.length; i++) {
      Cookie cookie = cookies[i];
      if (cookieName.equals(cookie.getName()))
        return(cookie);
    }
    return(null);
  }
```

1.11 Session

- **Session:** interval of time
- **Session Tracking** is a way to maintain state (data) of an user

1.11.1 Session Tracking

- We all know that **HTTP** is a stateless protocol. All requests and responses are independent. But sometimes you need to keep track of client's activity across multiple requests. For eg. When a User logs into your website, not matter on which web page he visits after logging in, his credentials will be with the server, until he logs out. So this is managed by creating a session.
- Session Management is a mechanism used by the Web container to store session information for a particular user.



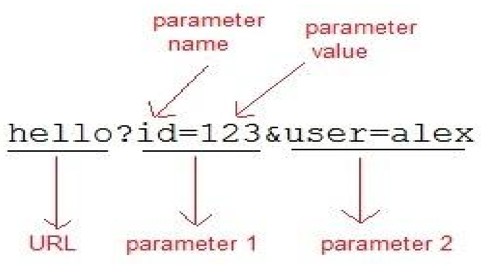
- There are four different techniques used by Servlet application for session management. They are as follows:
  1. Cookies
  2. URL-rewriting
  3. Hidden form fields
  4. HttpSession

Cookies

- **Cookies** are small pieces of information that are sent in response from the web server to the client.
- **Cookies** are stored on client's computer. They have a lifespan and are destroyed by the client browser at the end of that lifespan.
- **Advantage of cookie**
  - Simplest technique of maintaining the state
  - Cookies are maintained at client side.
- **Disadvantage**
  - It will not work if cookie is disabled from the browser.
  - Only textual information can be set in Cookie object

URL-rewriting

- If the client has disabled cookies in the browser then session management using cookie wont work. In that case **URL Rewriting** can be used as a backup. **URL rewriting** will always work.
- In URL rewriting, a token(parameter) is added at the end of the URL. The token consist of name/value pair separated by an equal(=) sign.



- When the User clicks on the URL having parameters, the request goes to the **Web Container** with extra bit of information at the end of URL. The **Web Container** will fetch the extra part of the requested URL and use it for session management.
- The `getParameter()` method is used to get the parameter value at the server side.
- **Advantage of URL-Rewriting**
    - It will always work whether cookie is disabled or not (browser independent).
    - Extra form submission is not required on each pages.
- **Disadvantage of URL-Rewriting**
    - It will work only with links.
    - It can send Only textual information.

Hidden form fields

- Hidden form field can also be used to store session information for a particular client.
- User information is stored in hidden field value and retrieved from another servlet.

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

- **Advantage**
    - Does not have to depend on browser whether the cookie is disabled or not.
    - Inserting a simple HTML Input field of type hidden is required. Hence, its easier to implement.
- **Disadvantage**
    - Extra form submission is required on every page. This is a big overhead.

HttpSession

- Servlets provide an outstanding technical solution: the HttpSession API.
- This high-level interface is built on top of cookies or URL-rewriting

1.11.2 The Session Tracking API

- Using sessions in servlets is straightforward and involves looking up the session object associated with the current request, creating a new session object when necessary, looking up information associated with a session, storing information in a session, and discarding completed or abandoned sessions.
- **HttpSession** object is used to store entire session with a specific client. We can store, retrieve and remove attribute from **HttpSession** object.
- Any servlet can have access to **HttpSession** object throughout the `getSession()` method of the **HttpServletRequest** object.

- **Creating a new session**

Creating a new session

1. `HttpSession session =request.getSession();`
   // getsession() method returns a session. If the session already exists, it returns the existing session else create a new session

2. `HttpSession session = request.getSession(true);`
   // getsession(true) always returns new session

Getting a pre-existing session

```
HttpSession session = request.getSession(false);
```
//getSession(false)will check existence of session, If session exists, then it returns the reference of that session object, if not, this methods will return null

Destroying a session

```
session.invalidate( ); //destroy a session
```

The methods available in HttpSession class

1	**public Object getValue(String name)** **public Object getAttribute(String name)** These methods extract a previously stored value from a session object. They return `null` if there is no value associated with the given name. `getAttribute` is preferred and `getValue` is deprecated.
2	**public void putValue(String name, Object value)** **public void setAttribute(String name, Object value)** These methods associate a value with a name. Use `putValue` with servlets and either `setAttribute` (preferred) or `putValue` (deprecated) with version 2.2 servlets.
3	**public void removeValue(String name)** **public void removeAttribute(String name)**

	These methods remove any values associated with the designated name. If the value being removed implements `HttpSessionBindingListener`, its `value Unbound` method is called.
4	**public String[] getValueNames()** **public Enumeration getAttributeNames()** These methods return the names of all attributes in the session. Use `getValueNames` in version 2.1 of the servlet specification. In version 2.2, `getValueNames` is supported but deprecated; use `getAttributeNames` instead.
5	**public String getId()** This method returns the unique identifier generated for each session. It is sometimes used as the key name when only a single value is associated with a session, or when information about sessions is being logged.
6	**public boolean isNew()** This method returns `true` if the client (browser) has never seen the session, usually because it was just created rather than being referenced by an incoming client request. It returns `false` for preexisting sessions.
7	**public long getCreationTime()** This method returns the time in milliseconds since midnight, January 1,1970 (GMT) at which the session was first built. To get a value useful for printing out, pass the value to the `Date` constructor or the `setTimeInMillis` method of `GregorianCalendar`.
8	**public long getLastAccessedTime()** This method returns the time in milliseconds since midnight, January1970 (GMT) at which the session was last sent from the client.
9	**public int getMaxInactiveInterval()** **public void setMaxInactiveInterval(int seconds)** These methods get or set the amount of time, in seconds, that a session should go without access before being automatically invalidated. A negative value indicates that the session should never time out.

Example for creating session

Index.html

```
<html>
    <head>
        <title>TODO supply a title</title> </head>
    <body>
      <form method="post" action="Validate">
                <h3>User: <input type="text" name="user" /></h3>
                <h3> Password: <input type="password" name="pass"></h3>
                <input type="submit" value="submit">
    </form>
    </body>
</html>
```

Validate.java

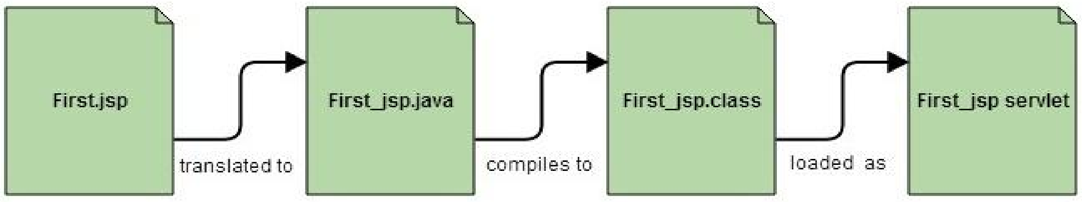
```
public class Validate extends HttpServlet {
    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");

        String name = request.getParameter("user");
        String pass = request.getParameter("pass");

        if(pass.equals("1234"))
        {
            //creating a session
            HttpSession session = request.getSession();
            session.setAttribute("user", name);
            response.sendRedirect("Welcome");
        }
    }
}
```

Welcome.java

```
public class Welcome extends HttpServlet {

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

         response.setContentType("text/html;charset=UTF-8");

        PrintWriter out = response.getWriter();

        HttpSession session = request.getSession();
        String user = (String)session.getAttribute("user");
        out.println("Hello "+user);
    }
}
```

1.12 Overview of JSP: JSP Technology

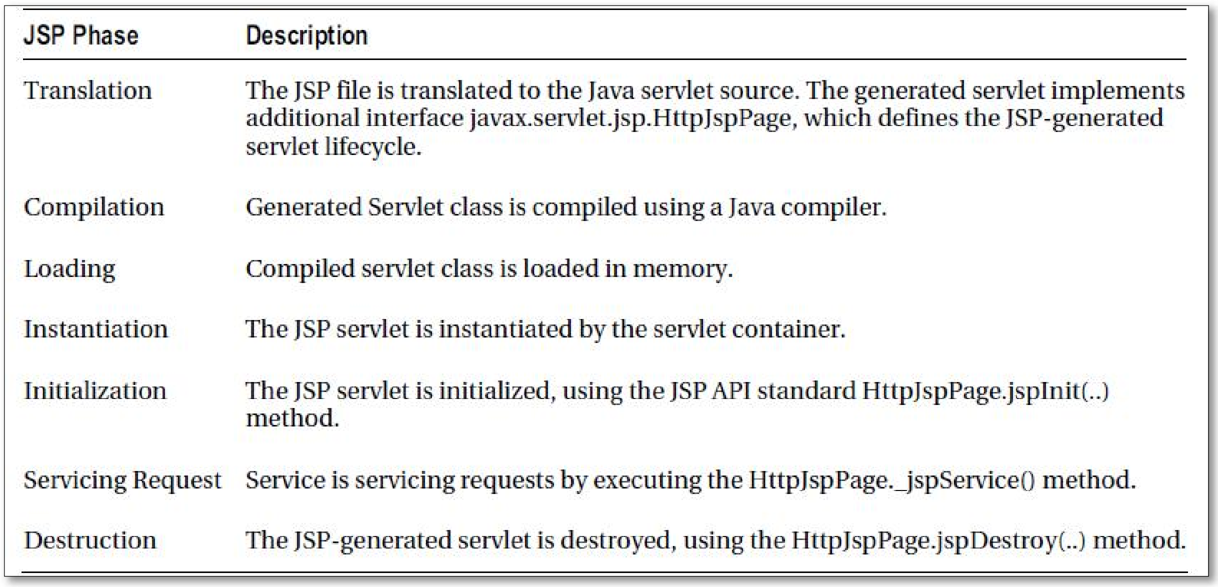
- **JSP** technology is used to create dynamic web applications. **JSP** pages are easier to maintain then a **Servlet**.
- JSP pages are opposite of Servlets as a servlet adds HTML code inside Java code, while JSP adds Java code inside HTML using JSP tags.
- JSP enables us to write HTML pages containing tags, inside which we can include powerful Java programs.
- **JSP** pages are converted into **Servlet** by the Web Container. The Container translates a JSP page into servlet **class source(.java)** file and then compiles into a Java Servlet class.



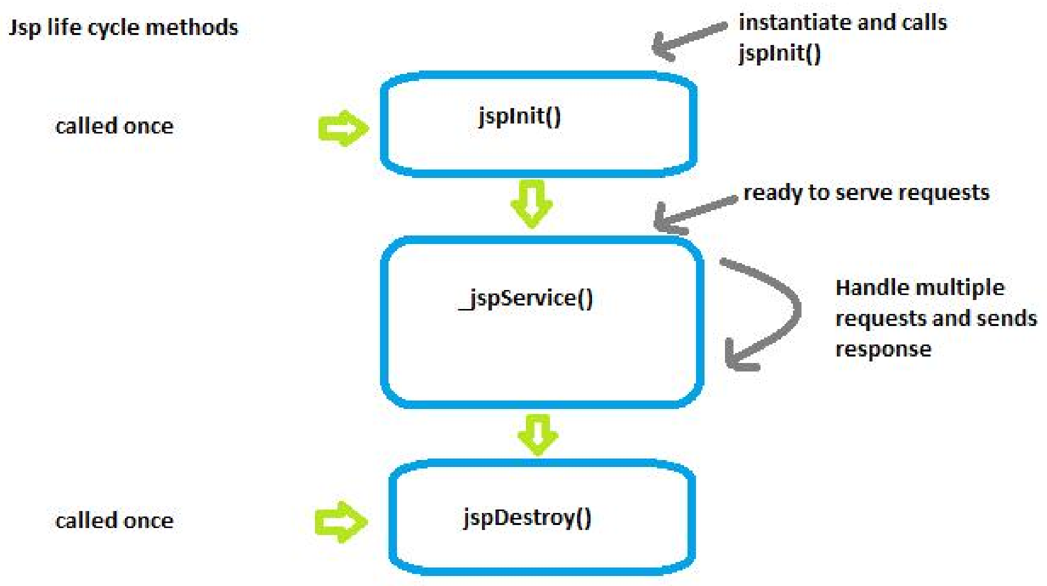
- **There are three main types of JSP constructs that you embed in a page:**
  - o scripting elements
  - o directives
  - o actions

1.12.1 lifecycle of JSP

JSP pages are saved with **.jsp** extension which lets the server know that this is a JSP page and needs to go through JSP life cycle stages.



JSP Phase	Description
Translation	The JSP file is translated to the Java servlet source. The generated servlet implements additional interface javax.servlet.jsp.HttpJspPage, which defines the JSP-generated servlet lifecycle.
Compilation	Generated Servlet class is compiled using a Java compiler.
Loading	Compiled servlet class is loaded in memory.
Instantiation	The JSP servlet is instantiated by the servlet container.
Initialization	The JSP servlet is initialized, using the JSP API standard HttpJspPage.jspInit(..) method.
Servicing Request	Service is servicing requests by executing the HttpJspPage._jspService() method.
Destruction	The JSP-generated servlet is destroyed, using the HttpJspPage.jspDestroy(..) method.



1.13 Need of JSP

- JSP provides an easier way to code dynamic web pages.
- JSP does not require additional files like, java class files, web.xml etc
- Any change in the JSP code is handled by Web Container (Application server like tomcat), and doesn't require re-compilation.
- JSP pages can be directly accessed, and web.xml mapping is not required like in servlets.

1.14 Advantages of JSP

- *Extension to Servlet*
  - JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
- *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 a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc

1.15 Basic syntax

There are four different types of elements you can use in JSP.

- Scripting elements
- Comments
- Directives
- Actions

Scripting elements

- There are three types of scripting elements in JSP
  - **Scriptlets** - A scriptlet tag is used to execute java source code in JSP.
    - General form:

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

    - Example:

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

  - **Declarations**—is used *to declare fields and methods*.
    - General form:

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

    - Example

```
<html>
<body>
<%! int data=50 %>
</body>
</html>
```

  - **Expressions**—Contains a Java expression that the server evaluates. The result of the expression is inserted into the Web page.

- General form:

```
<%= Java Expression %>
```

- Example:

```
Current time: <%= new java.util.Date() %>
```

- **Comments or *Template text***
  - In many cases, a large percentage of your JSP page just consists of static HTML, known as *template text*. In almost all respects, this HTML looks just like normal HTML, follows all the same syntax rules, and is simply “passed through” to the client by the servlet created to handle the page.
  - If you want a comment to appear in the JSP page use the below form

```
<%-- JSP Comment --%>
```

- **Directives**
  - JSP directives let you give directions to the server on how a page should be processed. There are three directives in JSP.

Directive	**Description**
<%@page...%>	defines page dependent properties such as language, session, errorPage etc.
<%@ include…%>	defines file to be included.
<%@ taglib…%>	declares tag library used in the page

- **Actions**
  - The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

JSP Action Tags	**Description**
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.