javax.servlet.http package:

The javax.servlet.http package contains a number of classes and interfaces that describe and define the contracts between a Servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming Servlet container.

Interface *Summary*

HttpServletRequest *It extends the ServletRequest interface to provide request information for HTTP Servlets.*

HttpServletResponse *It extends the ServletResponse interface to provide HTTP-specific functionality in sending a response.*

HttpSession *Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.*

HttpSessionActivationListener *Objects that are bound to a session may listen to container events notifying them that sessions will be passivated and that session will be activated.*

HttpSessionAttributeListener *This listener interface can be implemented in order to get notifications of changes to the attribute lists of sessions within this web application.*

HttpSessionBindingListener *Causes an object to be notified when it is bound to or unbound from a session.*

HttpSessionListener *Implementations of this interface are notified of changes to the list of active sessions in a web application.*

Class	Description
Cookie	Allows state information to be stored on a client machine
HttpServlet	Provides methods to handle HTTP requests and responses

Following are the interfaces and their methods description

HttpServletRequest Interface:

Cookie[ ] getCookies	Returns an array of the cookies in this request
String getMethod()	Returns the HTTP method for this request
String getQueryString()	Returns any query string in the URL
String getRemoteUser()	Returns the name of the user who issued this request.
String getRequestedSessionId()	Returns the ID of the session
String getServletPath()	Returns the part of the URL that identifies the servlet

HttpServletResponse Interface:

Void addCookie(Cookie cookie)	Adds cookie to the HTTP response.
Void sendError(int c)	Send the error code c to the client
Void sendError(int c , String s)	Send the error code c and the message s
Void sendRedirect(String url)	Redirects the client to url

Cookie class:

The Cookie class encapsulates a cookie. A cookie is stored on a client and contains state information. Cookies are valuable for tracking user activities. A servlet can write a cookie to a user's machine via the addCookie() method of the HttpServletResponse interface. The names

and values of cookies are stored on the user's machine. Some of the information that is used saved includes the cookie's

- Name
- Value
- Expiration date
- Domain and path

Following are the methods that are used by the Cookie class

String getComment()	Returns the comment
String getDomain()	Returns the domain
Int getMaxAge()	Returns the age
String getName()	Returns the name
String getPath()	Returns the path
Boolean getSecure()	Returns true if the cookie is secure
Int getVersion()	Returns the version
Void setComment(String c)	Sets the comment to c
Void setDomain(String d)	Sets the domain to d
Void setPath(String p)	Sets the path to p
Void setSecure(boolean secure)	Sets the security flag to secure

HttpServlet Class:

The HttpServlet class extends GenericServlet. It is commonly used when developing Servlets that receive and process HTTP requests. Following are the methods used by HttpServlet class

Void doGet(HttpServletRequest req, HttpServletResponse res)	Performs an HTTP GET
Void doPost(HSR req, HSR res)	Performs and HTTP POST
Void service(HSR req, HSR res)	Called by the server when HTTP request arrives for this servlet.

Handling HTTP Requests and Responses:

The following example shows a Servlet program handling ***HTTP GET*** requests

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
 public class GetServlet extends HttpServlet
{
public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
{
String name=req.getParameter("name");
res.setContentType("text/html");
PrintWriter pw=res.getWriter();
pw.println("The Name is ");
pw.println(name);
pw.close();
}
```

The following example shows a Servlet program handling ***HTTP POST*** requests

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
 public class GetServlet extends HttpServlet
{
public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
{
String name=req.getParameter("name");
res.setContentType("text/html");
PrintWriter pw=res.getWriter();
pw.println("The Name is ");
pw.println(name);
pw.close();
}
```

Using Cookies – security issues:

Cookies are small bits of textual information that a Web server sends to a browser and that the browser returns unchanged when visiting the same Web site or domain later. By having the server read information it sent the client previously, the site can provide visitors with a number of conveniences:

1. *Identifying a user during an e-commerce session.* Many on-line stores use a "shopping cart" metaphor in which the user selects an item, adds it to his shopping cart, then continues shopping. Since the HTTP connection is closed after each page is sent, when the user selects a new item for his cart, how does the store know that he is the same user that put the previous item in his cart? Cookies are a good way of accomplishing this. In fact, this is so useful that Servlets have an API specifically for this, and servlet authors don't need to manipulate cookies directly to make use of it.
2. *Avoiding username and password.* Many large sites require you to register in order to use their services, but it is inconvenient to remember the username and password. Cookies are a good alternative for low-security sites. When a user registers, a cookie is sent with a unique user ID. 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 doesn't require an explicit username and password.
3. *Customizing a site.* Many portal sites let you customize the look of the main page. They use cookies to remember what you wanted, so that you get that result initially next time.

The Servlet Cookie API:

To send cookies to the client, a servlet would create one or more cookies with the appropriate names and values via new Cookie(name, value), set any desired optional attributes via cookie.setXxx, and add the cookies to the response headers via response.addCookie(cookie).

To read incoming cookies, call request.getCookies(), which returns an array of Cookie objects. In most cases, you loop down this array until you find the one whose name (getName) matches the name you have in mind, then call getValue on that Cookie to see the value associated with that name.

Creating Cookies:

A Cookie is created by calling the Cookie constructor, which takes two strings: the cookie name and the cookie value. Neither the name nor the value should contain whitespace or any of:

[ ] ( ) = , " / ? @ : ;

Reading and Specifying Cookie Attributes:

Before adding the cookie to the outgoing headers, you can look up or set attributes of the cookie. Here's a summary:

1. *getComment/setComment* – Gets/sets a comment associated with this cookie.
2. *getDomain/setDomain* – Gets/sets the domain to which cookie applies. Normally, cookies are returned only to the exact host name that sent them. You can use this method to instruct the browser to return them to other hosts within the same domain.
3. *getMaxAge/setMaxAge* – Gets/sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session (i.e. until the user quits the browser), and will not be stored on disk. See the ***LongLivedCookie*** class below, which defines a subclass of Cookie with a maximum age automatically set one year in the future.

LongLivedCookie.java

```
import javax.servlet.http.*;
public class LongLivedCookie extends Cookie {
public static final int SECONDS_PER_YEAR = 60*60*24*365;
public LongLivedCookie(String name, String value) {
super(name, value);
setMaxAge(SECONDS_PER_YEAR);
}
}
```

4. *getName/setName* – Gets/sets the name of the cookie. The name and the value are the two pieces you virtually always care about. Since the getCookies method of HttpServletRequest returns an array of Cookie objects, it is common to loop down this array until you have a particular name, then check the value with getValue.
5. *getPath/setPath* – Gets/sets the path to which this cookie applies. If you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories. This method can be used to specify something more general. For example, someCookie.setPath("/") specifies that all pages on the server should receive the cookie. Note that the path specified must include the current directory.
6. *getSecure/setSecure* – Gets/sets the boolean value indicating whether the cookie should only be sent over encrypted (i.e. SSL) connections.
7. *getValue/setValue* – Gets/sets the value associated with the cookie. Again, the name and the value are the two parts of a cookie that you almost always care about, although in a few cases a name is used as a Boolean flag, and its value is ignored (i.e. the existence of the name means true).
8. *getVersion/setVersion* – Gets/sets the cookie protocol version this cookie complies with. Version 0, the default, adheres to the original Netscape specification. Version 1, not yet widely supported, adheres to RFC 2109.

Placing Cookies in the Response Headers:

The cookie is added to the Set-Cookie response header by means of the addCookie method of HttpServletResponse. Here's an example:

```
Cookie userCookie = new Cookie("user", "uid1234");
response.addCookie(userCookie);
```

Reading Cookies from the Client:

To read the cookies that come back from the client, you call getCookies on the HttpServletRequest. This returns an array of Cookie objects corresponding to the values that came in on the Cookie HTTP request header.

Once you have this array, you typically loop down it, calling getName on each Cookie until you find one matching the name you have in mind. You then call getValue on the matching Cookie, doing some processing specific to the resultant value. This is such a common process that the following section presents a simple getCookieValue method that, given the array of cookies, a name, and a default value, returns the value of the cookie matching the name, or, if there is no such cookie, the designated default value.

Getting the Value of a Cookie with a Specified Name:

The following example shows how to retrieve a cookie value given a cookie name by looping 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.

```
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);
}
```

Session Tracking – security issues:

An HttpSession class was introduced in the Servlet API. Instances of this class can hold information for one user session between requests. You start a new session by requesting an HttpSession object from the HttpServletRequest in your doGet or doPost method:

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

This method takes a boolean argument. true means a new session shall be started if none exist, while false only returns an existing session. The HttpSession object is unique for one user session.

The Servlet API supports two ways to associate multiple requests with a session: cookies and URL rewriting. If cookies are used, a cookie with a unique session ID is sent to the client when the session is established. The client then includes the cookie in all subsequent requests so the servlet engine can figure out which session the request is associated with. URL rewriting is intended for clients that don't support cookies or when the user has disabled cookies. With URL rewriting the session ID is encoded in the URLs your servlet sends to the client. When the user

clicks on an encoded URL, the session ID is sent to the server where it can be extracted and the request associated with the correct session as above.

To use URL rewriting you must make sure all URLs that you send to the client are encoded with the *encodeURL* or *encodeRedirectURL* methods in *HttpServletResponse*.

An HttpSession can store any type of object. A typical example is a database connection allowing multiple requests to be part of the same database transaction, or information about purchased products in a shopping cart application so the user can add items to the cart while browsing through the site. To save an object in an HttpSession you use the *putValue* method:

```
...
Connection con = driver.getConnection(databaseURL, user, password);
session.putValue("myappl.connection", con);
...
```

In another servlet, or the same servlet processing another request, you can get the object with the getValue method:

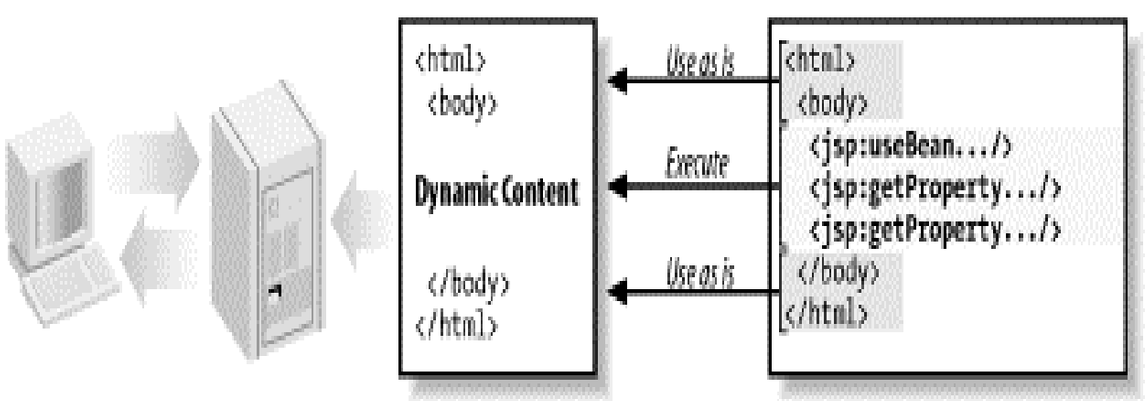
```
...
HttpSession session = request.getSession(true);
Connection con = (Connection) session.getValue("myappl.connection");
if (con != null) {
// Continue the database transaction
...
```

You can explicitly terminate (invalidate) a session with the invalidate method or let it be timed-out by the servlet engine. The session times out if no request associated with the session is received within a specified interval. Most servlet engines allow you to specify the length of the interval through a configuration option. In the Servlet API there's also a *setMaxInactiveInterval* so you can adjust the interval to meet the needs of each individual application.

Introduction to JSP:

Java Server Pages is a technology for developing web pages that include dynamic content. Unlike HTML page which contains static content, JSP page contains dynamic content which can change based on any number of variable items including identity of user, user's browser type, information provided by the user and so on.

JSP not only contains standard markup language elements like HTML tags but also contains special JSP elements which allow the server to insert the dynamic content in the page. JSP elements are used for various purposes like retrieving information from database, registering user preferences. The following figure contains JSP elements like jsp:useBean, jsp:getProperty.



When a user requests a JSP page, server executes JSP elements and merges the results with static parts of the page and sends the dynamically composed page back to the browser. JSP is the latest technology for web application development which is based on servlet technology.

Problems with Servlets:

Servlet: A servlet is basically a small Java program that runs within a Web server. It can receive requests from clients and return responses. The whole life cycle of a servlet breaks up into 3 phases:

1. ***Initialization***: A servlet is first loaded and initialized usually when it is requested by the corresponding clients. Some websites allow the users to load and initialize Servlets when the server is started up so that the first request will get responded more quickly.
2. ***Service***: After initialization, the Servlets serve clients on request, implementing the application logic of the web application they belong to.
3. ***Destruction***: When all pending requests are processed and the Servlets have been idle for a specific amount of time, they may be destroyed by the server and release all the resources they occupy.

Problems with Servlets:

In many java Servlets based applications, processing the request and generating the response are both handled by a single servlet class. A servlet class contains request processing, business logic implemented by the methods like “isOrderInfoValid()” and “saveOrderInfo()” and generating the responses and also a HTML code is embedded in servlet code using println() call.

Java programming knowledge is needed to develop and maintain all the aspects of an application, since the processing code and HTML elements are lumped together. Changing the look and feel of an application or adding support for a new type of client requires servlet code to be updated and recompiled.

It is time taking to design an application interface using web page development tools because HTML elements are embedded in to the servlet code manually. The problem with servlet can be solved by separating request processing, business logic and presentation (HTML code).

Separation of request processing and business logic from presentation makes it possible to divide the development and designing of a webpage. This means that java programmers can implement request processing and business logic (development), whereas webpage authors implement the user interface (designing).

Anatomy of JSP page:-

A JSP page is similar to a regular webpage with JSP elements. Which generates the parts differed from each request.

Example: Regular webpage with JSP elements (JSP page) is given as follows:

```
<%@ page language=“java” contentType=“text/html” %> /* JSP element */
<!-- template text -->
<html>
<body bgcolor=“yellow”>
<!-- JSP Element -->
<jsp:useBean id=“userInfo” class=“com.ora.jsp.beans.userInfo.UserInfoBean”>
<jsp:setProperty name=“userInfo” property=“*”/>
</jsp:useBean>
```

```
<ul>
<li> User Name:
<jsp:getProperty name="userInfo" property="userName"/> /* JSP Element */
<li> Email Address:                                    /* Template text */
<jsp:getProperty name="userInfo" property="emailAddr"/> /* JSP Element */
</ul> </body> </html>                  /* template text */
```

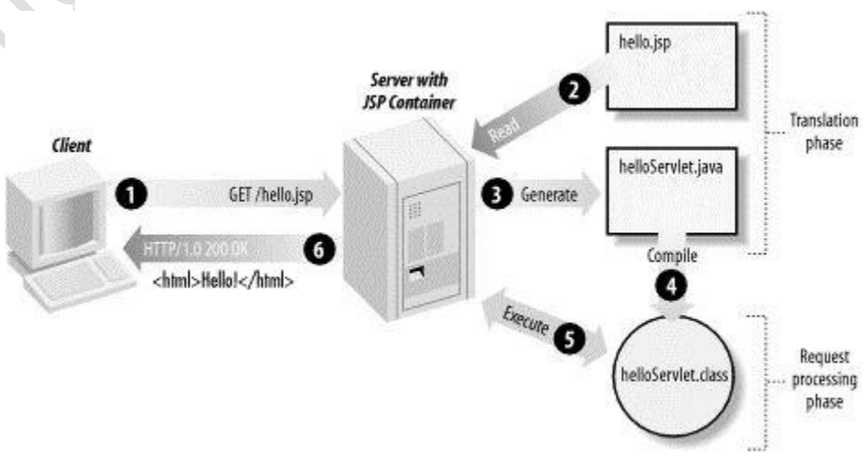
In the above example of JSP page everything except JSP element is called as template text. Template text may be HTML or WML or XML or even plain text. When a JSP page request is processed, the template text and dynamic content generated by JSP elements are merged and the result is sent as response to the browser.

JSP Processing:

The following steps explain how the web server creates the web page using JSP:

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of .html.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.

All the steps mentioned above can be shown below in the following diagram:



Typically the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated

servlet still matches the JSP's contents. This makes the process more efficient than with other scripting languages and therefore faster.

JSP Application Design with MVC:

JSP technology can play a part in everything from the simplest web application, such as an online phone list or an employee vacation planner, to full-fledged enterprise applications, such as a human-resource application or a sophisticated online shopping site. How large a part JSP plays differs in each case, of course. In this section, I introduce a design model called Model-View-Controller (MVC), suitable for both simple and complex applications.

The key point of using MVC is to separate logic into three distinct units: the Model, the View, and the Controller. In a server application, we commonly classify the parts of the application as business logic, presentation, and request processing. Business logic is the term used for the manipulation of an application's data, such as customer, product, and order information. Presentation refers to how the application data is displayed to the user, for example, position, font, and size. And finally, request processing is what ties the business logic and presentation parts together. In MVC terms, the Model corresponds to business logic and data, the View to the presentation, and the Controller to the request processing.

The reason to use MVC design with JSP lies primarily in the first two elements. Remember that an application data structure and logic (the Model) is typically the most stable part of an application, while the presentation of that data (the View) changes fairly often. Just look at all the face-lifts many web sites go through to keep up with the latest fashion in web design. Yet, the data they present remains the same.

Another common example of why presentation should be separated from the business logic is that you may want to present the data in different languages or present different subsets of the data to internal and external users. Access to the data through new types of devices, such as cell phones and personal digital assistants (PDAs), is the latest trend. Each client type requires its own presentation format. It should come as no surprise, then, that separating business logic from the presentation makes it easier to evolve an application as the requirements change; new presentation interfaces can be developed without touching the business logic.

This MVC model is used for most of the examples. JSP pages are used as both the Controller and the View, and JavaBeans components are used as the Model. Many types of real-world applications can be developed this way, but what's more important is that this approach allows you to examine all the JSP features without getting distracted by other technologies.

The model is responsible for managing the data of the application and it responds to the request from the view and it also responds to instructions from the controller to update itself. It is the lowest level of pattern which is responsible for maintaining data. The model represents the application core for example a list of data base records. It is also called the domain layer.

The view displays the data, it request information from the model that it needs to generate the output representation. It represents data in a particular format like JSP. MVC is often seen in web applications where the view is the HTML page.

The controller is the part of the application that handles user interaction. Typically controllers read data from a view, control user input, and send data to the model. It handles the input, typically user actions and may invoke changes on the model and view.

AJAX:

AJAX is a web development technique for creating interactive web applications. AJAX meant to increase the web page's interactivity, speed, and usability. AJAX-based applications can be difficult to debug, test and maintain.

- AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS and Java Script.
- Ajax uses XHTML for content and CSS for presentation, as well as the Document Object Model and JavaScript for dynamic content display.
- Conventional web application transmit information to and from the sever using synchronous requests. This means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX when submit is pressed, JavaScript will make a request to the server, interpret the results and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background
- Intuitive and natural user interaction. No clicking required only Mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven

AJAX is based on the following open standards:

- Browser-based presentation using HTML and Cascading Style Sheets (CSS)
- Data stored in XML format and fetched from the server
- Behind-the-scenes data fetches using XMLHttpRequest objects in the browser
- JavaScript to make everything happen

Technologies used in AJAX are:

JavaScript

- Loosely typed scripting language
- JavaScript function is called when an event in a page occurs
- Glue for the whole AJAX operation

DOM

- API for accessing and manipulating structured documents
- Represents the structure of XML and HTML documents

CSS

- Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript

XMLHttpRequest

- JavaScript object that performs asynchronous interaction with the server. The XMLHttpRequest object is used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Create an XMLHttpRequest Object

All modern browsers (IE7+, Firefox, Chrome, Safari, and Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

variable=new XMLHttpRequest();

An old version of Internet Explorer (IE5 and IE6) uses an ActiveX Object:

variable=new ActiveXObject("Microsoft.XMLHTTP");

Here is the list of famous web applications which are using AJAX

Google Maps: A user can drag the entire map by using the mouse instead of clicking on a button or something.

Google Suggest: As you type, Google will offer suggestions. Use the arrow keys to navigate the results.

Gmail: Gmail is a new kind of webmail, built on the idea that email can be more intuitive, efficient and useful.