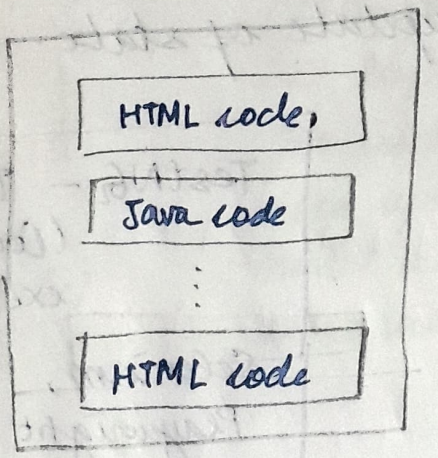
Java Server Pages (JSP)

JSP file: HTML page with some Java code
Dynamic content from Java code



JSP Process:
- JSP is processed on the server side
- Java code is processed and it's results are returned to browser as plain HTML

Where to place JSP file?

▽ Web Content
  ▽ META-INF
  ▽ WEB-INF
    helloworld.jsp (~ to multiple html files in a directory)

Code Example:

```
<html>
<body>
    <h3> Hello World! </h3>

    <p> Time on the server is: <%= new java.util.Date() %>.
    </p>

</body> </html>
```

calls toString() of the object

Output included in HTML file.

JSP scripting Elements:

JSP Expression : <%= some Java Expression %>

JSP Scriplet : <% some Java code : 1 or many lines %>

JSP Declaration : <%! variable or method declaration %>

JSP Implicit Objects:

They are a set of implicit Java objects that every JSP container automatically creates and makes it available on every JSP page. These objects can be directly accessed without any declarations.

9 implicit objects provided by JSP

1) request : instance of HttpServletRequest (contains request headers and form data)
2) response : instance of HttpServletResponse
3) config : instance of ServletConfig
4) application : instance of ServletContext (App-level common data)
5) session : instance of HttpSession (unique session for each user of the web App)
6) page context : instance of PageContext
7) page : reference to the current JSP page object
8) exception : instance of Throwable (used in error page)
9) out : instance of JspWriter (includes content in HTML page)

Difference between Scriplet and Expression tags

* Scriplet tags don't insert the evaluated Java code into HTML page. To include content in the page use : out.println()
* Expression tags evaluates Java expressions and automatically inserts the result into the HTML page (w/o use of out.println())

Best practices with scriplets :

* Minimise amount of Java code inside HTML file
* Refactor the Java code into Java class ... make use of MVC

Scriptlet tag: Any java code written between <% --- %> is converted into HTML / plain text and sent to client browser when translating a JSP page.

Other JSP scripting Elements:

JSP comment <%-- Set of comment statements --%>

JSP Directive <%@ directive %>

JSP Declarations

* Declare a method in the JSP page
* Call the method in the same JSP page.

Best practices:

* Minimise the number of JSP declarations
* Refactor them into Java classes.

Calling Java class from JSP

1. Create Java class (contains the heavy business logic) (under Java resources)
2. Call java class from JSP

JSP Directives:

* During JSP lifecycle, there is a phase where JSP is translated into servlets.
* Directives are commands or instructions to JSP container while converting JSP to servlets.
* Contain many attributes that are comma - separated and act as key - value pairs.

Syntax: <%@ directive attribute = " " %>

Types: 1. page directive
2. Include directive
3. Taglib directive.

Page directive: <%@ page attribute = "attribute_value" %>

XML equivalent = <jsp:directive.page attribute = "attribute_value" />

Defines properties that is applied on the entire JSP page.

Ex: imported packages
classes/interfaces
allocated buffer, etc.

(1) import: <%@ page import = "value" %>
the container
Tells what and all packages/classes must be imported into the JSP page.

(2) contentType: <%@ page contentType = "value" %>
The format of data exchange between server and client.

(3) info: <%@ page info = "value" %>
Defines the string to be printed when using getServletInfo() method.

(4) buffer: <%@ page buffer = "1kb" %>
JSP outputs are generally stored in buffer (RAM) before sending them to clients

If buffer = "none" it means JSP outputs are immediately sent to clients as and when outputs are generated. So, JSP exceptions cannot be thrown

If output generated < buffer when exception is thrown ⇒ full error page

If output generated > buffer, partial HTML page is displayed.

<%@ page language="java" %>

(6) isELIgnored : Tells if the page supports expression language

Default : false (evaluates expressions ${...} ~~allows expression tags~~)
true (displays ${...} as plain text ~~disables~~)

(7) errorPage : <%@ page errorPage="value" %>

Defines which error page to redirect to in case the current page encounters an error.

(8) isErrorPage : * classifies a page as error page or not.
* If error page, can use implicit object "exception" to display exceptions

<%@ page isErrorPage = "true/false" %>

Include directive:

* Directive for including other files into current JSP page.
* Other files = html files, other jsp files.
* Allows code re-usability.

<%@ include file = "filename" %>

Taglib directive

* Used to mention library of custom-defined tags that can be used in current JSP page.

<%@ taglib uri = "value" prefix = "value" %>

value to identify tags of the library mentioned in uri

JSP Actions.

* Action tags are special elements that give instructions to the JSP container to handle various runtime tasks
* They can be added anywhere in the JSP page where dynamic behaviour is intended.
* Can handle tasks like forwarding requests, including external resources, interact with JavaBeans and plugins.
* Enclosed within <jsp:...>

(1) <jsp:include> tag

Includes content from another resource (JSP, HTML, servlet) into the current page at runtime.

Syntax: <jsp:include page="filename"/>

Commonly used for re-using header, footer and navigation

Differences between include action tag and include directive.

Directive	Action Tag.
Compile Time	Runtime.
Any changes to included file, need to recompile	Any changes to included file, reflected immediately
Static merge	Dynamic include
Mostly used in templates	Mostly used for displaying dynamic content.

(2) <jsp:forward> tag

* Forwards current request to another resource (JSP/HTML/servlet). The client is unaware of the forwarding (confirm this under Network Tab of DevTools)
* The rest of current page is skipped.

<jsp: forward destination = "filename.jsp" />

(3) <jsp: useBean> tag

* Instantiates or locates an existing JavaBean (basically a Java class) to be used within a JSP page.

<jsp: useBean id = "beanName" class = "package" />

Ex: <jsp: useBean id = "emp" class = "com.luv2code.dto.Employee" />

* If bean can't be found in the requested scope, a new instance is created automatically.

(4) <jsp: setProperty> tag

* Set properties of the JavaBean

<jsp: setProperty name = "emp" property = "name" value = "Aparna" />

* We can directly retrieve values of the parameters from the request using property = "*"

(5) <jsp: getProperty> tag

* Retrieves and displays the value of a JavaBean property

<jsp: getProperty name = "emp" property = "name" />

(6) <jsp: plugin> tag (commonly used for Applets)

* Generates HTML code for browser-specific Plugins such as Java applets

<jsp:plugin type="applet" code="MyApplet.class" />

(9) <jsp:attribute> tag

Defines attribute values (custom) for a JSP page

<jsp:attribute name="color" value="blue" />

(8) <jsp:body> tag

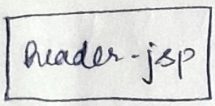
Dynamically include some content using this custom JSP tag

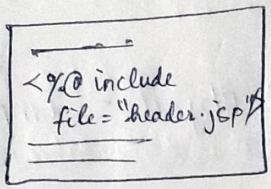
<jsp:body>
Body content
</jsp:body>

Ex:
<jsp:body>
 This is custom content
</jsp:body>

Working difference between directive and action

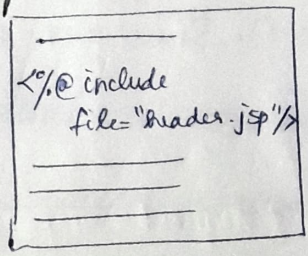
Files involved:



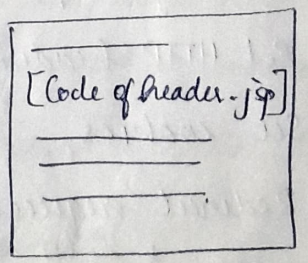


main.jsp

Working of directive



inserts code of Header.jsp

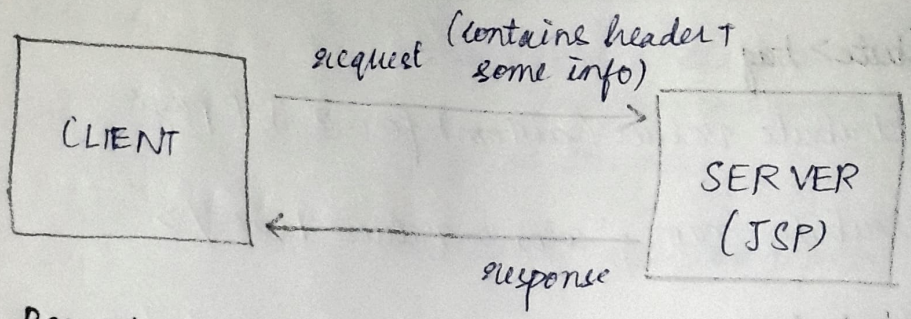


Then compiles as a single servulet → output to user

Working of action tag.

main.jsp → separate servulet → output
Header.jsp → separate servulet → output
→ combine outputs at runtime.

JSP Implicit Objects (Continued)



Request Object:

★ Request object is created for every client request

★ Used for:

a) Read form data

b) Access request parameters

c) Retrieve request headers

d) Obtain session information

Response Object

★ Similar to request object - implicitly created

★ Used for

a) Sending output to client

b) Set HTTP headers

c) Set cookies

d) Redirect requests.

Difference between scriptlet tag and declaration

test.jsp

```
<%! int x = 10 %>
<% int y = 20 %>
<% = x+y %>
```

This is converted as:

```
public class test_jsp extends HttpJspBase {

    int x = 10; // declaration in class level
    public void _jspService(HttpServletRequest
        request, HttpServletResponse response) {

        int y = 10; // scriptlet is service/
                    method level
        out.print(x+y);
                    declaration
    }
}
```

x cannot be accessed here (so we cannot use request in <%! ... %> tag)

Cheat sheet:

Scriptlet - <% ... %> - method scope.

Declaration - <%! ... %> - class scope.

JSTL and EL [EL = ${...}].

* JSTL = JSP Standard Tag Library (Java code in the form of HTML)
* convert Java code to HTML code.
* Expression language (EL) can be used to print anything stored in java lang. EL can be used instead of Java code.
* EL was developed so that it can be used within JSTL tags.

Difference between forward and redirect:

Forward (server-side): client → servlet → JSP (no round-about trip to client)

Redirect (client-side): client → servlet → redirect (302) → ~~servlet~~ → client → new URL

Scope Explanation:

1) Request scope: 1 request only
2) Session scope: exists until browser / logout
3) application scope: until server shutdown.

JSTL - jar file to be added in

(1) Java Resources > Libraries > Web App Libraries

(2) Web Content > WEB-INF > lib.

Common JSTL core tags:

1) <c:if>

2) <c:choose> + <c:when> + <c:otherwise>

3) <c:forEach>

4) <c:remove>

5) <c:set>

6) <c:import>.

Contd: Implicit objects.

pageContext:

Scopes in HTTP Session:

1) page scope: ★ only available till JSP execution (i.e., compilation of JSP, until response is sent to browser)

★ NOT available in:

↳ new tab

↳ new request

↳ refresh.

2) request scope: ★ only available for 1 request-response cycle.

★ during that 1 cycle, it can survive forward and include

★ cannot survive:

↳ redirects

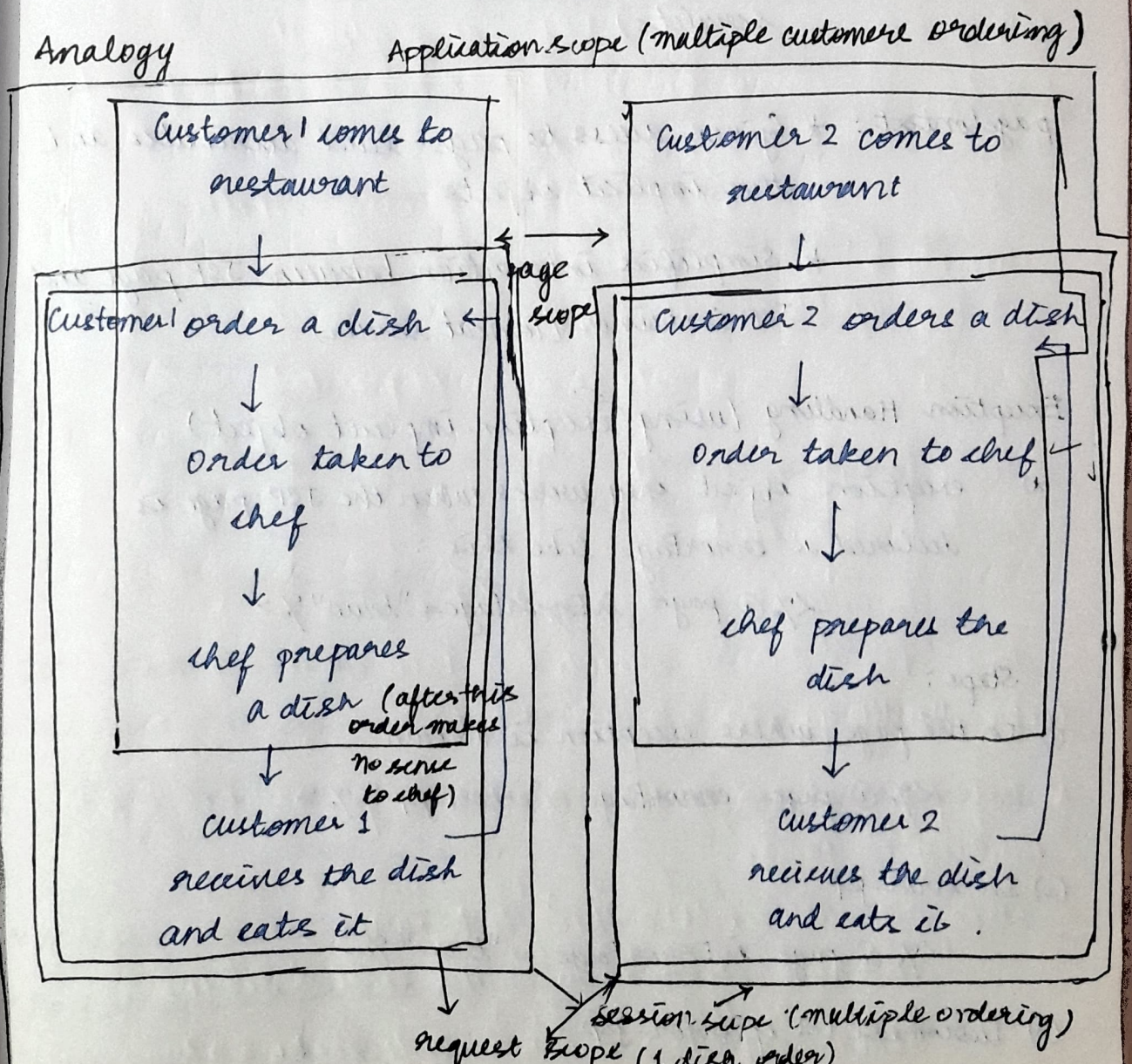
↳ new tabs

↳ new request

3) session scope: ★ Available for a browser session

* Available across:
  - ↳ pages
  - ↳ tabs
  - ↳ navigation
  - ↳ refresh
* Not available on
  - ↳ browser close
  - ↳ logout
  - ↳ session timeout

1) Application scope: * available entire web application
   * shared across all users, all tabs, all sessions



ServletConfig: ★ This object holds info about initialization parameters for a specific servlet when the servlet is deployed

★ Common parameters: credentials, file paths, default values (like Locale), API key, feature switches (like language, theme for viewing) specific for that servlet.

★ Params are read-only after initialisatn

★ Not shared across servlets

ServletContext: app-wide settings. (common across all servlets)

pageContext: ★ Gives access to page-level attributes and other implicit objects

★ Simplifies interaction between JSP page and JSP Container (Tomcat server)

Exception Handling (using exception implicit object)

(*) exception object only works when the JSP page is declared as errorPage like this:

<%@ page isErrorPage="true" %>

Steps:

(1) In the page where exception is thrown

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

(2) In error.jsp

<%@ page isErrorPage="true" %>

(3) Customise UI of error.jsp.

JSTL Sql tags

Steps:

1) Add some data in Database (create DB in mySQL/Postgres)
2) Download the jar file for the DB you are using (download from Maven Central)
3) Add the jar file in WebContent > WEB-INF > lib. This automatically adds
4) Add the URL for JSTL - SQL tags using taglib directive)
5) Use <sql:setDatasource> tag in the following manner:

```
<sql:setDataSource
    var = "db" (like connection Object)
    driver = "org.postgresql.Driver"
    url = " " → url to your database
    user = " "          (Ex: jdbc:postgresql://localhost:5432/<db name>)
    password = ""       → username set during installation
                        → password set during installation
/>.
```

6) To run a query:

```
<sql:query
    var = " " → name of variable storing the result set
    datasource = "${db}"
> Query </sql:query>
```

JSTL Function tags:

Add this URL for function tags:

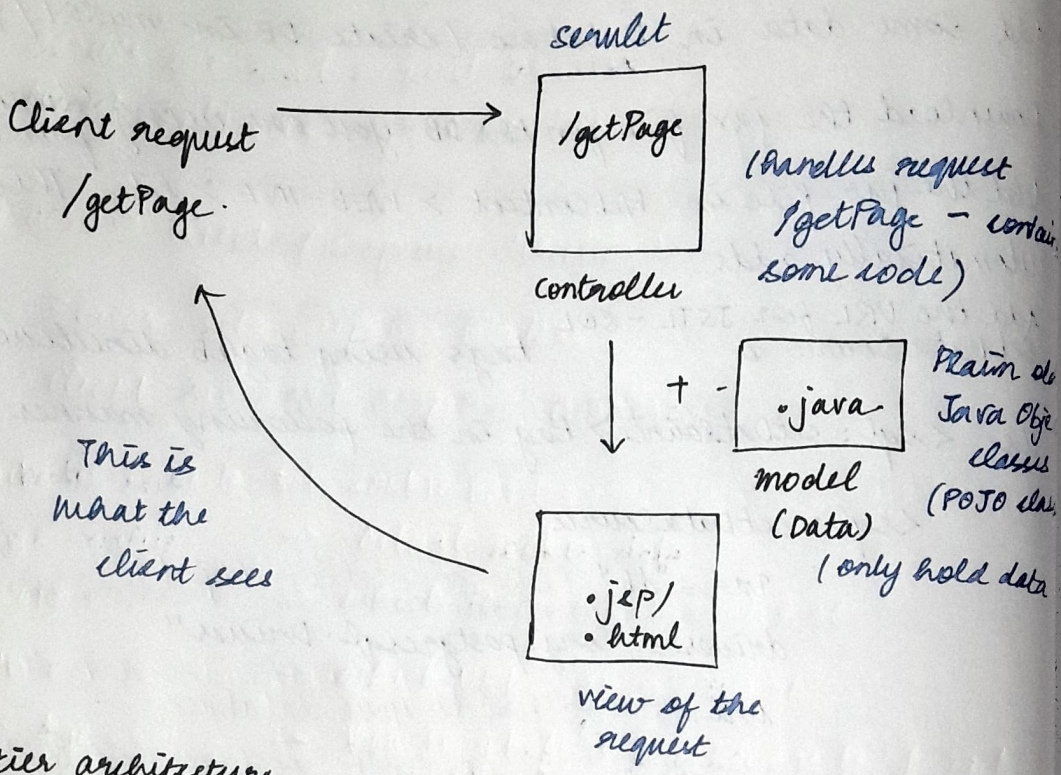
```
<%@ taglib url = "http://java.sun.com/jsp/jstl/functions"
    prefix = "fn"%>
```

MVC using Servlet and JSP

↳ Do not write business logic inside servlets.

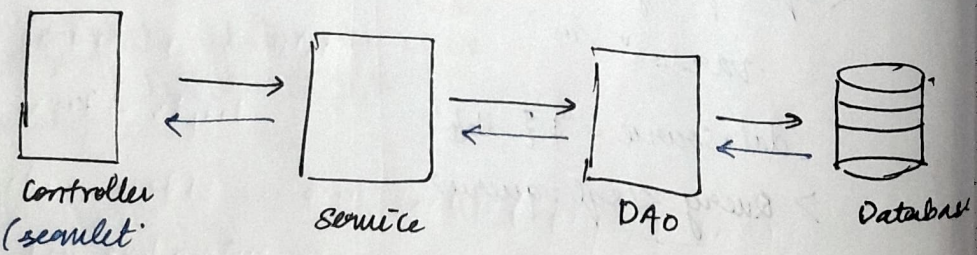
View → static (HTML)
     → dynamic (JSP)

MVC in diagram:



N-tier architecture

* Service class containing business logics like connection with database.
* To map DB results to java class use Data Access Object classes.



Cookie Handling:

* Text files stored in the client side for various informal tracking purposes.

Steps: 1) Server sends cookies to browser

2) Browser stores them in it's local machine

3) Future requests - browser sends cookies along with the requests

4) Server uses this info to identify the user.

Anatomy of cookie:- How server sends cookies:

(protocol version)
HTTP/1.1 200 OK (request-code - request successfully processed by server)
Date: ~~Fri~~ Mon, 26 Jan 2026 21:03:38 GMT
↳ (date at which response was sent)
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
↳ what server software handled the request
Set-Cookie: name = xyz; expires = Mon, 26-Jan-26 22:07:00 GMT;
↓ here is where server sends browser (client) the cookies
cookies = key-value pair
↓ time until which the browser has to cookies
path = /; domain = ~~expo~~ example.com
↳ the path within the domain
Connection: close ← After sending response the server has closed TCP connection, if
Content-Type: text/html.
↓ tells browser how to interpret the response body sent by the server
Connection: keep-alive ← the server keeps the connection for sending future responses

How browser sends the cookie:

If the user points the browser that matches the path and domain of the cookie, the browser resends the cookie to server.

→ GET request → path
GET / HTTP/1.0 → HTTP protocol version
Connection: Keep-alive (Client wants to keep TCP connection open after this request so as to save time for subsequent requests)
User-Agent: Mozilla/4.76 (X11; I; Linux 2.2.6-15apmac ppc)
Browser OS (server may adapt response based on the client's architecture, OS, browser)
Host: zink.demon.co.uk:1126
domain port (non-standard HTTP port) (standard - 80#)
Accept: image/gif, */*
Accept-Encoding: gzip → browser accepts compressed response using gzip
Accept-Language: en → browser accepts English content
Accept-Charset: iso-8859-1, *, utf-8 → accepts characters from these charsets
Cookie: name=xyz. (cookies sent by browser → server)
↓ multiple cookies
Cookie: A=1; B=2; C=3

Accepts an format of response (*/*) but prefers (image/gif)

NOTE:

Client → server (request direction)

server can read request headers
Client only sets request headers.

server → client (response direction)

Server sets response headers
Client can read response headers.

Attributes inside set-cookie header:

name = value	→	Actual cookie data
Domain =	→	which domains can receive it
Path =	→	which URLs can receive it
Max-Age =	→	how long before browser has to "forget" the cookie
Expires =	→	expiry date
Secure	→	sent only over HTTPS connections
Httponly	→	JS cannot read the cookie
SameSite =	→	CSRF rule
Comment =	→	note to users (deprecated)

Setting Cookies in servlet

1) Create cookie object

Cookie cookie = new Cookie("key", "value");

NOTE: key and value shouldn't contain white spaces or any of these characters: [], (), =, ;, /, ", ?, @,

2) Set Maximum age (seconds)

If no age is set, cookie is stored until browser is shutdo

cookie.setMaxAge(3600)

3) Send the cookie into the Set-Cookie HTTP response header

response.addCookie(cookie)

Reading cookies with Servlet

1) Get Cookie [ ] using getCookies() method

request.getCookies()

2) Iterate through the array

3) Use getters and setters of each cookie attribute to access each cookie and it's attributes

Delete a Cookie with Servlet

1) Read the required cookie and store it in a Cookie object

2) cookie.setMaxAge(0) ⇒ deletes the existing cookie

3) response.addCookie(cookie).