

SCWCD

BY

DUGRA

MONOJ ENTERPRISES XEROX

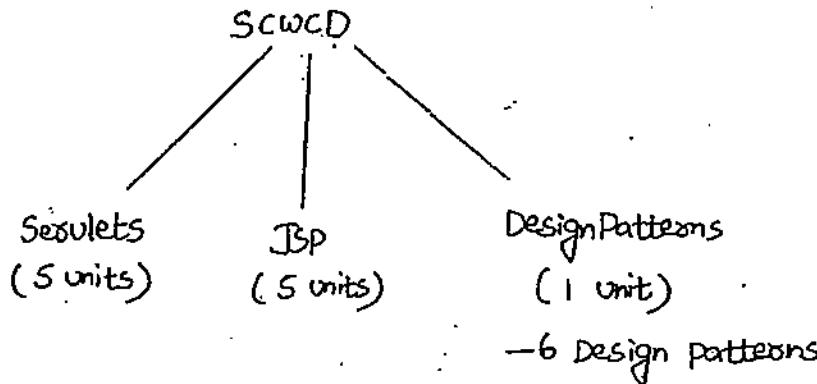
Plot No:40 Gayatri Nagar Behind HUDA Ameerpet, Hyd. Cell: 9908909131

Software institutes material available

Geometric series: $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots$

SCWCD

①



Servlets

① Servlet Technology model

- HTTP methods
- servlet Life cycle
- servlet Request
 - Form Parameters
 - Request Header
 - cookies
- Servlet Response
 - Set Content Type
 - Set headers
 - add cookies
 - getWriter => For sending Text Data
 - getOutputStream => For Binary Response
 - Send Redirection

② Structure of web Applications & Deployment Description

- Directory Structure of web Application
- * - web.xml
- war files

③ The web Container model

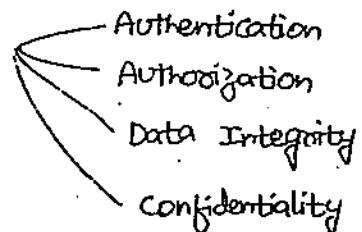
- Servlet Context Parameters
- Servlet Scopes & Attributes
- Request Dispatcher
- Filters
- Wrappers

④ Session Management

- Session API
- cookies
- URL Rewriting
- Listeners

⑤ web Security

- Terminology



- Types of Authentication
 - Basic Authentication
 - Digest "
 - Form Based "
 - client cert "

- web.xml declarations

- Programmatic Security

Jsp's

(2)

① JSP Technology model

- JSP Life cycle
- JSP Implicit Objects (9)
- JSP API
- JSP Documents
- Scripting Elements
- JSP scopes
- JSP Directives

② Developing Jsp's by Standard Actions

- | | |
|---------------------|-----------------|
| - Jsp : useBean | <u>Not used</u> |
| - Jsp : setProperty | Jsp : fallback |
| - Jsp : getProperty | Jsp : plugin |
| - Jsp : Include | Jsp : Params |
| - Jsp : Forward | |
| - Jsp : param | |

③ Building Jsp's by Expression Language (EL)

- EL implicit Objects (11)
- EL Operators
- EL Functions

④ JSTL

- Core Library

⑤ customTags

- └ classic Tags
- └ simple Tags
- └ tag Files

I) Introduction to Design patterns

- Front Controller
- MVC
- Intercepting Filter Design pattern
- Data Transfer Object
- Business Delegate
- Service Locators.

SCWCD 1.5V

J2EE 1.5V

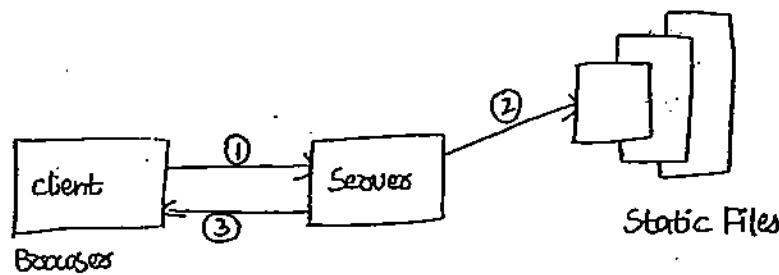
- Servlet 2.5V
- JSP's 2.1V
- JSTL 1.2V

UNIT - I

Servlet Technology model

(3)

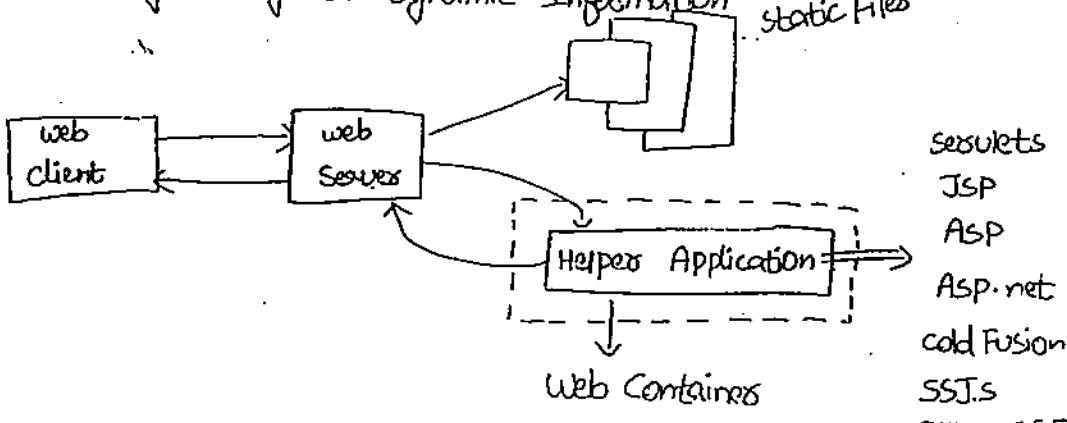
web Programming For Static Information -



- ① client Sends a request for a static files
- ② server Searches whether the request resource is there or not
- ③ If the request resource is available then it will return that resource as response.
- If it is not available, server Sends 404 Status code saying the requested resource is not available.

Note:- For Servicing Static Files, no Processing is required at server side. Hence web servers always loves to serve static Information.

Web Programming for Dynamic Information



- Browser sends a request to the web Server
- web Server checks whether the request is for Static or for Dynamic Information.
- If it is for Static, web Server Searches for that Static File in the Local File System.
 - If it is available, it returns that Static File otherwise it returns 404 Status Code.
- If the request is for Dynamic Information, the web Server forwards that request to Helper Application.
- Helper Application analyses & Process the request and generates the required Dynamic Information.
- Helper Application forwards that Dynamic Information to the web Server & web Server returns forwards that Dynamic Info to the browser.
- The following are various Helper Applications
 - Servlet - Cold Fusion
 - JSP - Server Side Java Script (SSJS)
 - ASP - PHP
 - ASP.net - CGI

Servlet

(4)

Servlet is a Server Side web Component managed by web containers for generation of Dynamic Information.

⇒ web Containers is the best Assistant for the Programmers. It maintains entire life cycle of the Servlet, so that the Programmers has to concentrate only on servicing logic.

Types of web Containers

There are 3 types of web Containers are possible

① Stand - Alone web Containers

② InProcess "

③ OutProcess "

Stand - Alone — Both web Servers & web Containers are available in a Single Integrated Component.

⇒ Tomcat.

Inprocess — If the web Container runs in the same address space (same machine) of web servers & it is available as plug-ins, such type of web Containers are Inprocess web Containers. → In this case both web servers & web Container need not be from the same vendor.

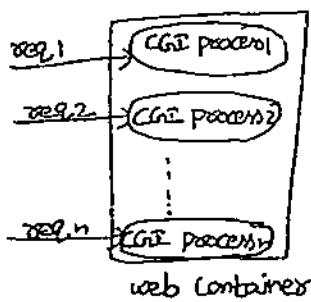
Out-process — web Servers & web Containers both are running on different machine. Web Container is attached to web servers externally.

- we can configure front End Apache Server to forward the request to the back End web logic Server.
- this type of web containers are most frequently used web containers.

Difference between CGI & Servlets

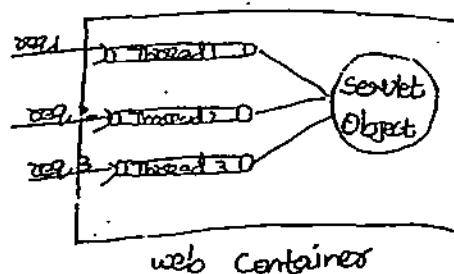
CGI

- It is process based i.e for every request a separate process will be created & it is responsible to generate response.



Servlet

- It is Thread based i.e for every request a separate new thread will be created and it is responsible to generate dynamic info.



- Creation & Destruction of new process for every request is costly.

→ If the number of request increases it will effect the Performance of the System. Hence CGI Technology fails to deliver Scalable Applications.

- creation & destruction of new thread for every Request is not costly.
→ There is no effect on the Performance even though number of requests increases. Hence Servlet Technology succeeds to deliver Scalable Applications.

Two processes never share common Address space. Hence there is no chance of concurrency problems.

CGI program can be written in multiple languages. But most commonly used language is perl.

most of the CGI Languages are not object oriented. Hence we are missing the benefits of oops.

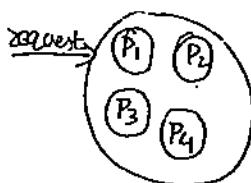
CGI Tech is Platform Dependent

Fast CGI

It improves the performance when compared with CGI.

In this case web container maintains a pool of processes

so that a single process can serve multiple requests one by one.

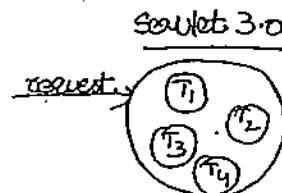


(3) All the threads shares the (5) common Address space. Hence there may be a chance of Concurrency problems.

(4) Servlets can be written only in Java.

(5) Java Language itself is Object oriented. Hence we can get all benefits of oops.

(6) Servlet Tech is platform Independent



Servlet API

- we can implement Servlet Programming by using the following 2 Packages:

- ① javax.Servlet (for generating Generic Servlet)
- ② javax.Servlet.http

javax.Servlet — This package contains several classes & Interfaces used for Developing Servlets irrespective of any protocol.

javax.Servlet.http — This package contains several classes & Interfaces for Developing http based Servlets.

javax.Servlet Package Interface

- ① Servlet
 - ② Servlet Request
 - ③ Servlet Response
 - ④ Servlet Config
 - ⑤ Servlet Context
 - ⑥ Request Dispatcher
 - ⑦ Single Thread model
 - ⑧ Filter
 - ⑨ Filter Config
 - ⑩ Filter chain
 - ⑪ Servlet Request Listener
 - ⑫ Servlet Request Attribute Listener
 - ⑬ Servlet Context Listener
 - ⑭ Servlet Context Attribute Listener
- } For implementing Filter concept
- } For Listener Concept

① **Sesulet** - Every Sesulet in Java should implement this interface either Directly or indirectly. (6)

→ This interface Defines the most common general methods which can be applied on any Sesulet.

→ Sesulet interface only contains Life Cycle methods.

② **Sesulet Request** - Sesulet Request Object contains the client information. It defines several methods to get client Info.

Ex: `getParameter("uname");`
`getCookies();`

③ **Sesulet Response** -

→ By using Response Object Sesulet can send required response to the End user (Browser).

→ This interface Defines several methods for Preparation of response

Ex: `getWriter(),` `sendRedirect`

④ **Sesulet Config** - For every Sesulet web Container creates Sesulet Config object to hold its Configuration Information.

By using Config object Sesulet can get its Configuration Information like Initialization Parameters, Logical name of Sesulet

⑤ **Sesulet Context** - For every web Application, web Container creates one Sesulet Context object to hold application level Configuration Information.

Note - Servlet Config Per Servlet where as Servlet Context is per web application.

⑤ Request Dispatcher By using RequestDispatcher Object we can dispatch the request from one web Component to another.

⇒ This interface contains only 2 methods

- ① Forward
- ② Include

3/69

⑥ Single Thread model

→ This interface is for Providing Thread Safety for the Servlets.

If a Servlet implement this interface, at a time the Servlet object can process only one client Request.

→ The advantage of this approach is, we can resolve Concurrency Problems.

→ The main limitation of this approach is, it may rise Performance Problems. Due to this reason Sun People Deprecated this interface with out any replacement in Servlet 2.4.

→ This Interface doesn't contain any methods. Hence it is a marker Interface.

→ It is recommended to use Synchronisation blocks & methods to provide Thread Safety instead of Single Thread Model.

Javax.Servlet Package Classes

(7)

Generic Servlet - This is for Developing Servlets irrespective of any Protocol. This class implements Servlet Interface.

ServletInputStream - This can be used for Reading Binary Data send by the client

ServletOutputStream - For Sending Binary Response (PDF, image files) to the client.

Servlet Request Wrapper } For Implementing wrapper classes
Servlet Response Wrapper } (for converting PDF to doc file)

Servlet Request Event
Servlet Request Attribute Event } For Defining Events.

Servlet Context Event

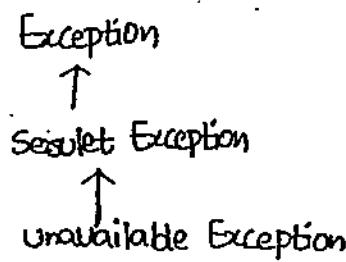
Servlet Context Attribute Event

Javax.Servlet Package Exceptions

This package defines the following 2 Exceptions :

① Servlet Exception - Servlet Can throw this Exception whenever it facing any problem while processing the request.

② Unavailable Exception - It is child class of Servlet Exception & it is Deprecated.



Note In total javax.Servlet package Contains

14 - Interfaces

9 - Classes

2 - Exceptions

Servlet Interface

→ This interface defines the following 5 methods.

- | | |
|-----------|--------------------|
| ① init | ④ getServletInfo |
| ② service | ⑤ getServletConfig |
| ③ destroy | |

① init :-

Public void init(ServletConfig config) throws ServletException

→ This method called only once by the web container to perform initialization activities immediately after instantiation.

→ Before servicing any client request, compulsory init() should be completed.

→ The web container won't place servlet in to service in
the following cases.

(8)

① If init() throws ServletException

② init() doesn't return within the time period specified
by web container.

→ In the above cases, web container makes that servlet
object eligible for Garbage Collection without calling
any other life cycle method.

```
init()  
{  
    ServletException  
    =  
    ServletException  
}  
}
```

```
init()  
{  
    }  
}  
} // web container internally creates another  
// init() and continues the process  
// until init() completes.
```

Service

For every request, web container executes this method
to provide response.

```
public void service(SevletRequest req, SevletResponse res)  
throws ServletException, IOException.
```

→ Entire servicing logic we have to define in this method only.

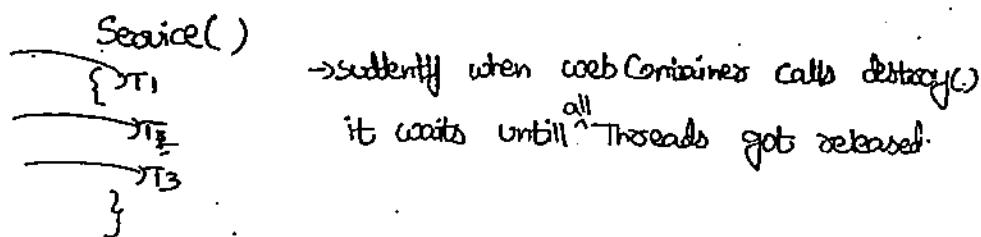
Destroy

```
public void destroy()
```

→ This method called only once by the web container to perform
clean-up activities whenever web container takes servlet

from out of Service. This usually happens at the time of Application undeployment. (or) Server Shutdown (or) whenever web Container requires some free memory.

→ whenever web Container calls destroy() it may not execute immediately. It will wait until the completion of all currently executing threads.



⇒ init, service & destroy also considered as Life Cycle methods of Servlet.

3) getServletConfig()

→ public ServletConfig getServletConfig()

- Returns Servlet Config of the Servlet Object.

→ Servlet can get its configuration information by using this its config object.

3) getServletInfo()

public String getServletInfo()

→ This method returns information about Servlet like author, version, copy write information etc.

- init, service(), destroy() are call back methods. web container internally calls these methods.
- Essentially we have to call getServletConfig(), getServletInfo().
- Program for writing Servlet that implements Servlet Interface directly.

```

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

public class FirstServlet implements Servlet, SingleThread model
{
    static
    {
        S.O.P ("Servlet Loading");
    }

    public FirstServlet()
    {
        S.O.P ("Servlet Instantiation");
    }

    public void init (ServletConfig config) throws ServletException
    {
        S.O.P ("Init method");
    }

    public void service (SR req, SR resp) throws SE, IOException
    {
        PrintWriter out = resp.getWriter();
        out.println ("

# welcome to SCWCD <h1>"); } }


```

```
public void destroy()
{
    System.out.println("Destroy method");
}

public ServletConfig getServletConfig()
{
    return null;
}

public String getServletInfo()
{
    return null; // or return "By Durga";
}
```

}

<web-app>

<Servlet>

<Servlet-name> FirstServlet <Servlet-name>
<Servlet-class> FirstServlet <Servlet-class>
<load-on-startup> 10 <load-on-startup>

</Servlet>

<Servlet-mapping>

<Servlet-name> FirstServlet </Servlet-name>
<url-pattern> /hello </url-pattern>
<url-pattern> /first </url-pattern>

</servlet-mapping>

(10)

</web-app>

without Load-on-Startup

1st Request

servlet loading
Instantiation
init()
service()

2nd Request

service()

with Load-on-Startup

⇒ At the time of Server Startup

Servlet Loading
Instantiation
init()

1st Request

service()

2nd Request

service()

Note

From Servlet 2.5 onwards a single Servlet can map with multiple URL-patterns.

Life Cycle of Servlet that implements Servlet Interface

Servlet class loading by class Loader

Servlet Instantiation by web container, for this web container always calls public no-argument Constructors. Hence every Servlet class should compulsorily contain public no-argument constructor otherwise instantiation problems will arise.

) Execution of init() by web Container.

Note

The above 3 steps will be executed generally at the time of First Request.

⇒ If load-on-startup is configured these will be executed at the time of Server Start-up (OS) at the time of Application Deployment.

) Execution of service() by web Container

) Execution of destroy() - by web container

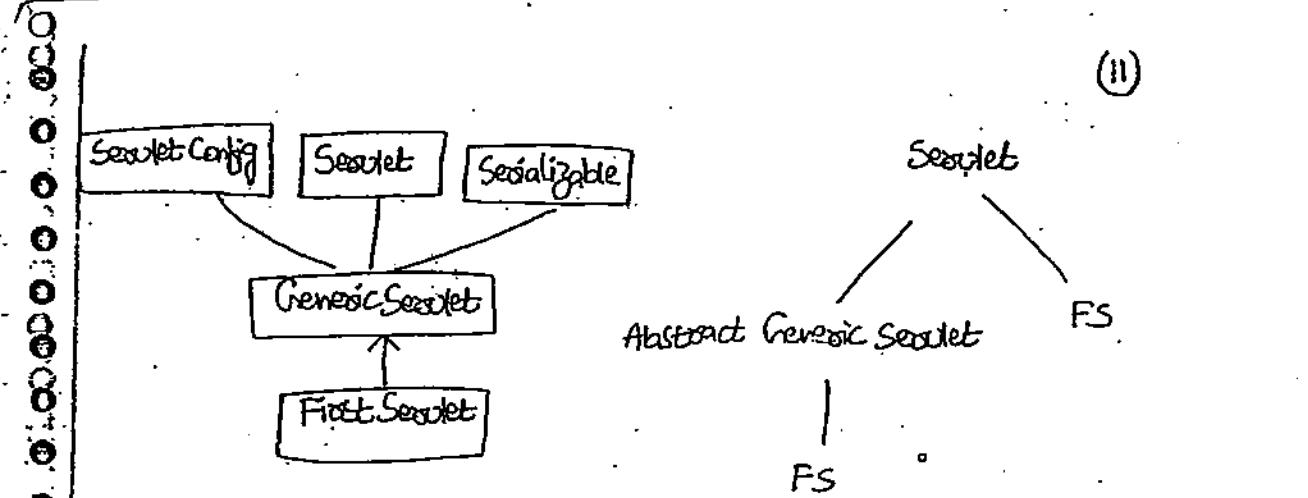
Generic Servlet

→ we can develop Servlet by implementing Servlet Interface directly. Here we should provide implementation for all 5 methods of Servlet Interface even though most of the times our concentration is on service() only.

→ This makes code ugly & unnecessarily complicated. we can resolve this problem by using GenericServlet.

→ GenericServlet already implemented Servlet Interface & provide implementation for all methods except service().

→ we can make our Servlet as child class of GenericServlet and provide implementation only for service().



- GenericServlet is an Abstract Class and implements
- ServletConfig , Servlet & Serializable Interfaces.
- Demo program for writing Servlet that extends GenericServlet
- Public class FirstServlet extends GenericServlet.

```

{
    public void service (SR req, SR resp) throws SE, IOE
    {
        PrintWriter out = resp.getWriter();
        out.println ("writing Servlet by Extending GenericServlet
                     is Very Easy ");
    }
}

```

Implementation of Generic Seawiki

```
public class GenericServlet implements Servlet, ServletConfig,  
                                Serializable.
```

£

Point transient Section big Config;

```
public void init(SeoulConfig config)
```

6

this.config = config; for web browser
purpose

```
init();
```

3

public void init() throws SE

۱

For Programmes Purpose

```
public SessionConfig getSessionConfig()
```

[

```
return Config();
```

3

\Rightarrow Generic Servlet Contains 2 init()’s.

\Rightarrow we can avoid init() in our Secret as follows

① Public void init(ServletConfig config) throws SE

5

// DataBase Operations

3

→ This approach is not recommended bcz we are not saving Config object Provided by the web container for the

future purpose. In this case, getServletConfig() returns null.

(12)

② Public void init(ServletConfig config) throws SE .

{

Super.init(config);

// DataBase operation.

}

→ This is Perfectly valid approach but not recommended to use bcz we are increasing Complexity and it may effect Performance.

③ Public void init() throws SE

{

// DataBase Operations.

}

→ This approach is valid and highly recommended

Q What is the purpose of 2 init()'s in Generic Servlets.

Q What is the necessity of Declaring Config Variable as transient in genericServlet.

javax.servlet.http package

→ this package defines several classes & interfaces which can be used for developing http based Servlets.

Interfaces of javax.servlet.http

1) HttpServletRequest - This is the child interface of ServletRequest. It contains client information.

2) HttpServletResponse - It is the child interface of ServletResponse. By using response object we can send the response to the client.

3) HttpSession - It is used for Session Management.

⇒ In addition to the above 3 interfaces javax.servlet.http package contains the following interfaces also:

4) HttpSessionListener

5) HttpSessionAttributeListener

6) HttpSessionBindingListener

7) HttpSessionActivationListener

8) HttpSessionContext - Deprecated in 1.5 U and not recommended to use

for implementing
Session Based Listener

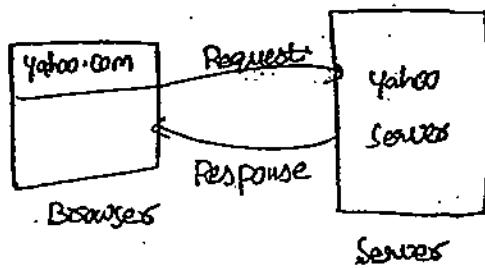
classes of javax.servlet.http Pkg.

(13)

- ① HttpServlet - It is the child class of GenericServlet. It can be used for developing Http Based Servlets.
- ② Cookie - used for Session Management.
- ③ HttpSessionEvent } For Defining Events in Session Listener
- ④ HttpSessionBindingEvent }
- ⑤ HttpServletRequestWrapper } For Defining Http Based Wrappers.
- ⑥ HttpServletResponseWrapper
- ⑦ HttpServletUtils - Deprecated in Servlet 2.3 Version.

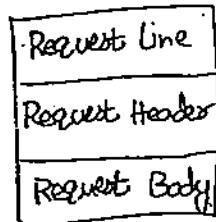
⇒ In total javax.servlet.http Pkg defines
8 - Interfaces
7 - Classes.

Common Terminology

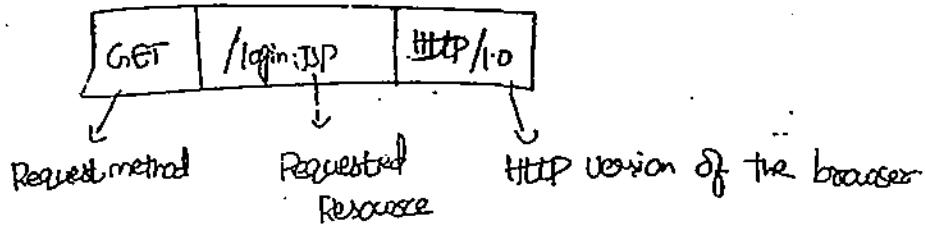


- web Client & web Server communicates by using some common language known as HTTP.
- HTTP defines standard structures for Request & Response.

Structure of HTTP Request



Request Line -



Request Headers -

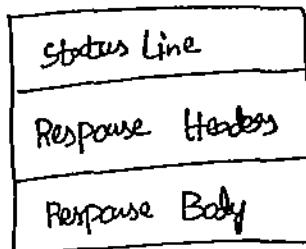
These provide ~~Browser's~~ Configuration information like media types accepted by browser, supporting Encoding types etc.

→ Server uses these request headers while preparing the Response. (14)

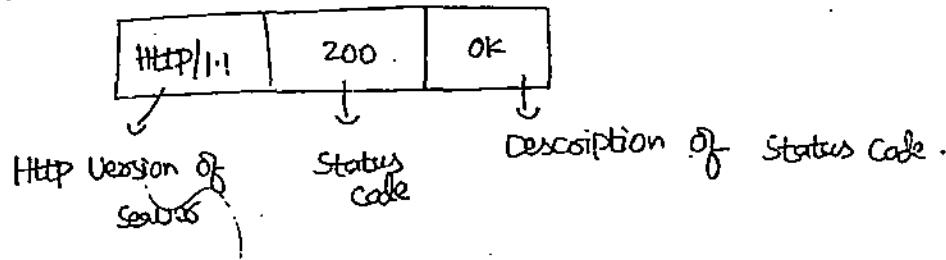
Request Body — It contains end-users provided information.

For the GET Request it is optional. But mandatory for the Post Request.

Structure of HTTP Response:



Status Line --



1xx → Information

2xx → Successfull

3xx → Redirection

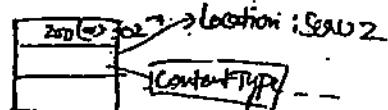
4xx → Sender Error

5xx → Server Error

Response Headers — These provides Configuration information of the server & the information abt the Response.

Like Content type, Content length, Last modified Date

etc.



→ The browsers use these Response Headers to Display Response Properly to the End user.

Response Body — It Contains the original Response provided by the Server.

Types of Http Request methods

→ Based on the type of information Requested by the browser all http methods are divided as follows.

① GET	} Introduced in Http 1.0 v	⑧ CONNECT
② POST		⑨ MOVE
③ HEAD		⑩ PROPFIND
④ OPTIONS	} Introduced in Http 1.1 v	⑪ LOCK
⑤ PUT		
⑥ DELETE		
⑦ TRACE		

→ until Servlet 2.4 version web server can provide the support for the first 7 HTTP methods. But from Servlet 2.5 ver onwards it provides the support for the remaining methods also.

① GET

→ If we require information from the Server then we should go for GET request.
usually GET requests are Read-Only.
→ In GET Request the client provided information appended to the URL as the part of Query String. & it is visible to the outside. Hence there is no security for the client Data & we can't send sensitive information.
By using get Request we can send only limited number of Information. we can't send Binary Data & we should send only Text Data.

Book marking of url's is possible for the GET Request.

Idempotent ($A^2 = A$)

By making the request either 1 time or several times if we are getting the same Response then that Request is said to be Idempotent.

GET Requests are Idempotent.

⇒ If we send a req for HeadFirst
Servlet book we get the same Response, irrespective of no of requests.

SAFE Request

By Repeating a Request Several times If it won't create any side effects at Servers Side , such type of Requests are Said to be Safe Request.

⇒ GET Requests are Safe but POST Requests are not Safe.

GET
⇒ Request for Head Part Searched

⇒ Response is same for any no of requests

POST

Durga l lack → Chiranjeevi

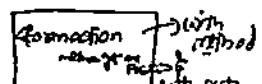
→ We want to send 1 hr to Chiru Acc

→ If he clicks for 3 or 4 times bcz of Process

Triggers For Sending GET Request time. At serverside problem arises.

① Typing url in the Address Bar & Submit is always GET Request

② Clicking Hyperlink



③ Submitting the form whose method Attribute is not Specified

④ Submitting the form whose method Attribute is Specified with
 `getvalue`

~~⑤~~
② Post

⇒ This method is usually for sending (or) Posting huge information to the Server.

Ex: Uploading Resume in Job portal.

→ In Post request the client information encapsulates in the request body instead of appending to the URL.

Hence we can send sensitive info by using Post method.

→ There is no limit on size of request body. Hence we can

Send large amount of information.

(16)

→ we can send binary information also in addition to text data.

→ we can't bookmark post url's.

→ Post Requests are not Idempotent & not safe bcz at servers side some updates will take place in post req.

Trigger for sending Post Request

→ To send Post Request Compulsory there should be form with method Attribute Value as post.

```
<form action = /web-app / test1 method = "POST">  
=   
</form>
```

Differences b/w GET & POST

HEAD

This method is for getting only Response Header Information like content type, content size, last modification.

For Head Request internally doGET() will execute.

Head Requests are Idempotent & Safe.

Options

This method is for getting supporting methods for retrieving a particular resource from the web server.

Options method is Idempotent & Safe.

3) PUT

- It is for placing a resource at the server side where the location is specified by URL.
- At the specified location if already another resource present then the old resource is replaced with provided new resource.
- By means of status code we can identify whether the replacement is happened or not.

Answers

200 : if Replacement Happen

201 : otherwise.

- Put Request are Idempotent but not Safe

3) DELETE

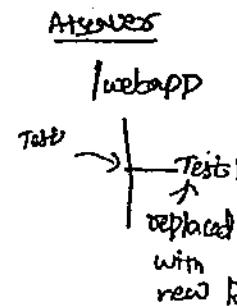
- This method is for deleting a particular resource at the server side. It is exactly counterpart (opp) to PUT.
- Delete method is Idempotent but not Safe.

Note

Put & Delete methods are not Safe. Hence most of the web servers doesn't provide the support.

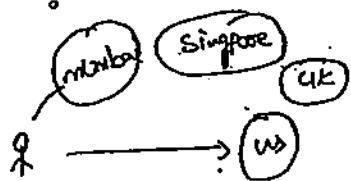
7) Trace

- we can use these methods for Debugging purposes. If we want to know what Request Server getting as Response then we should go for Trace method.



3 → TRACE method is Idempotent & Safe.

(17)



HTTP method	Idempotent	Safe
GET	Yes	Yes
POST	No	No
HEAD	Yes	Yes
OPTIONS	Yes	Yes
PUT	Yes	No
DELETE	Yes	No
TRACE	Yes	Yes

The only non-Idempotent method is "POST".

The following methods are not Safe

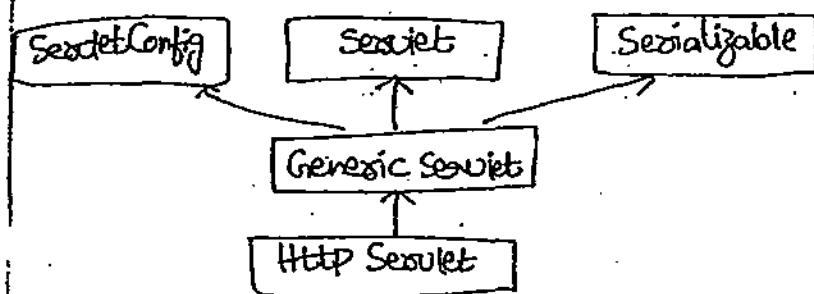
POST

PUT

DELETE.

Http Seoulet

- this class can be used for developing HTTP Based Seoulets.
- it is the child class of Generic Seoulet.



- For every HTTP method xxx , HttpServlet class Contains the Corresponding doxxx methods.

```
protected void doXXX (HttpServletRequest req,  
                      HttpServletResponse res) throws SE, IE
```

Ex: `protected void doGET (HSR req, HSR res) throws SE, IE`

- HttpServlet class contains 2 Service methods.

① `public void Service (SR req, SR res) throws SE, IE`

② `protected void Service (HSR req, HSR res) throws SE, IE.`

One Demo Program for HttpServlet

(18)

form.html

<html>

<body> This is Second class </body>

<form> action = "/Second/first" method="GET">

Enter Name : <input type="text" name="uname">

<input type="Submit">

</body>

</form>

</html>

import javax.servlet.*; (Servlet Exception)

import javax.servlet.http.*;

import java.io.*;

Public class FirstServlet extends HttpServlet

{
public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException

{
String name = req.getParameter("uname");

PrintWriter out = req.getWriter();

out.println("This is doGet()");

out.println("Hello " + name + " doGet() got executed");

```
public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
```

```
{
```

```
// same code as in doGet().
```

```
}
```

```
}
```

Life cycle of Http Request

- whenever we are submitting the form, Browser prepares HTTP Request & Sends to the Server.
- web Server checks whether the request is for static or Dynamic Information.
- If the request is for static Information web Server provides the required Response.
- If the request is for Dynamic Information, web server forwards it to the web Containers.
- web Containers identifies the Corresponding Servlet Class & checks whether the Servlet Object is already available or not.
- If the Servlet Object is not already available then web Containers loads the Corresponding Servlet class, performs Instantiation (creates object) and executes init()

- web Container Creates Servlet Request & Servlet Response Objects & invokes Public Service() by passing these objects as Arguments.

Public void Service (SReq req, SResp resp) throws SE, ISE

{

 HttpServletReq req = (HttpServletReq)req;

 HttpServletResp resp = (HttpServletResp)resp;

 service (req, resp);

}

- Public Service method Type Casts ServletRequest & Response Objects into HttpServletRequest & HttpServlet Response. Then it invokes Protected Service().

- The Protected Service() identifies the corresponding request method and invoke the corresponding doXXX().

- If our Servlet Contains that method it will be executed otherwise HttpServlet doXXX() will execute & provide the response 405 status code saying Requested method is not supported by us.

- If the Requested method is not a valid http method then we will get 501 status code saying Requested method is not a valid http method.

protected void Service (HCSR req, HCSR resp) throws ServletException

{

String method = req.getMethod();

if (method.equals("GET"))

{ doGET (req, resp); }

}

else if (method.equals("POST"))

{ doPOST (req, resp); }

}

else

{

return 501 Status code;

}

}

Case 1

If our Servlet Contain Service() with SR & SResp as arguments and another Service() with HSReq, HSResp as arguments then for any Request first Service() will be executed.

If our Servlet Contain Service() & doGET() then for any kind of Request including GET only Service() will be executed.

Case 3 If we are sending GET request but our Servlet
doesn't contain doGET() then HttpServlet's doGET() will be
executed and returns 405 Status Code saying HTTP method
GET is not supported by this URL.

Case-4

For Providing same Response for both get() & post() we have
to implement Servlet as follows.

```
public class HelloServlet extends HttpServlet
{
    public void doGET(HttpServletRequest, HttpServletResponse) throws ServletException
    {
        // Implementation.
    }

    public void doPost(HttpServletRequest, HttpServletResponse)
    {
        doGET(request, response);
    }
}
```

Case-5

For Providing Response to Head request we have to override
either doHEAD() or doGET() otherwise we will get
405 : saying http GET() is not supported by this.

```
doHEAD()
{
    doGET(); // not present, HttpServlet GET() will be
             // executed - 405.
}
```

- In our Servlet it is not Recommended to override the following methods.

Servlet Request

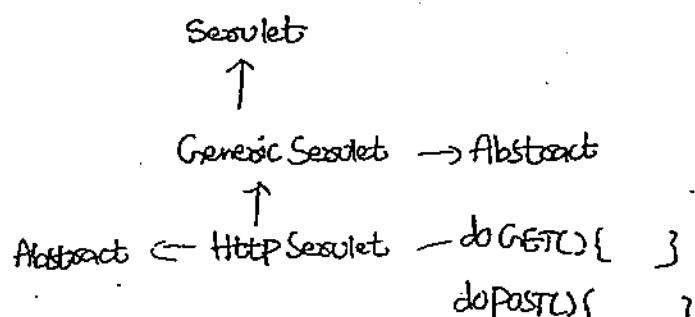
public Servlet(SR, SReq) HttpServlet Class Contains

protectedServlet(HSR, HSR) Proper Implementation.

doHEAD()

doOPTIONS()

doTRACE()



→ GenericServlet implements Servlet interface but doesn't provide implementation for service(). Hence GenericServlet is declared as Abstract.

→ All the methods are available in HttpServlet are Concrete methods and don't contain Single Abstract method.

But HttpServlet class declared as Abstract bcz most of the methods of HttpServlet are just for sending Error information.

→ creating an object of HttpServlet & calling doGET() explicitly

it is senseless. Bcz it will provide only error information as the response. To prevent instantiation **HttpServlet** is declared as abstract.

(21)

Http Servlet Request

using **HttpServletRequest** write code to

- ① Retrieve Form Parameters
- ② Retrieve Header Information
- ③ Retrieve Cookies.

Retrieving Form Parameters

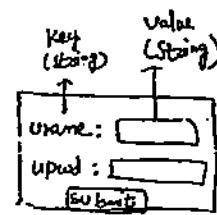
Form Parameters are key-value pairs where both key & value are String objects only.

A Form Parameter can be Associated with Single or multiple values.
(Selecting a course c, c++, java ---, we can select more than one course.)

Servlet Request Interface defines the following methods to retrieve form parameters from the request object.

- ① String getParameter(String parametername) - Returns the parameters value associated with specified parameter.
 - If the parameter Associated with multiple values then this method returns first value.
 - If the specified parameter is not available then this method returns "null".
 - The Argument is case sensitive.

Ex: `String course = req.getParameter("ucourse");`



2) String[] getParameterValues(String Parameter)

- Returns all the Values Associated with the Specified Parameter.
- If the Specified Parameter is not available, this method returns "null".
- Argument is Case-Sensitive.

Ex: String[] s = req.getParameterValues("ucourse");

```
for (String s1 : s)
{
    out.println(s1);
}
```

3) Enumeration getParameterNames()

- Returns all the Form parameter Names associated with Request.
- If the Request doesn't contain any form parameter, this method returns Empty Enumeration object but not null.

Ex: Enumeration e = req.getParameterNames();

```
while (e.hasMoreElements())
{
    String Parameter = (String) e.nextElement();
    String PValue = req.getParameter(Parameter);
    out.println(Parameter + " --- " + PValue);
}
```

(4) Map getParametersMap();

(22)

- Returns the Map Object Containing Parameter Names as Keys & Parameters Values as Map Values.
- Keys are String Objects but Values are String []
- If the request doesn't contain any form parameters then this method returns Empty map Object but not null.

<u>Keys</u> String	<u>Values</u> String []
uname	durga
upwd	anushka
ucourse	SCJP, SCWCD

```
map m1 = req.getParametersMap();
Iterator ito = m1.iterator();
while (ito.hasNext())
{
    Map.Entry m1 = (Map.Entry) ito.next();
    String name = (String) m1.getKey();
    String[] s = (String[]) m1.getValue();
    out.println(name);
    for (String s1 : s)
    {
        out.println(s1);
    }
}
```

```

map m1 = req.getParameterMap();
for(Map.Entry mg : m1)
{
    String pName = (String) mg.getKey();
    String[] s = (String[]) mg.getValue();
    out.println(pName);
    for(String[] s1 : s)
    {
        out.println(s1);
    }
}

```

> Demo Program

```

<html>
<body>
<form action="/ScwCDID/test">
    Enter Name: <input type="text" name="uname">
    Enter Contact No: <input type="text" name="ucontact">
    Enter Course: <select name="course" multiple>
        <option value="ScJP"></option>
        <option value="ScwCDP"></option>
    </select>
    <input type="submit" value="Submit" />
</form> </body> </html>

```

```

import javax.servlet.*; import javax.servlet.http.*; import java.util.*;
public class FormDemoServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        PrintWriter out = req.getWriter();
        out.println("GetParameters Demo");
        String uname = req.getParameter("uname");
        String ucontact = req.getParameter("ucontact");
        String ucourse = req.getParameter("ucourse");
        out.println("Student name : ", +uname);
        out.println("Student Contact : ", +ucontact);
        out.println("Student Course : ", +ucourse);
        out.println("Get Parameters Values ");
        String[] s = req.getParameterValues("course");
        for(String s1 : s)
        {
            o.println(s1);
        }
        o.println("Get Parameters Names ");
        Enumeration e = req.getParameterNames();
        while (e.hasMoreElements())
        {
            String s1 = (String) e.nextElement();
        }
    }
}

```

(23)

```
out.println( s1 )
```

```
out.println( "Get Parameters Map Demo" );
```

```
Map m = req.getParameterMap();
```

```
Set s = m.entrySet();
```

```
Iterator ito = s.iterator();
```

```
while ( ito.hasNext() )
```

```
{
```

```
Map.Entry mi = (Map.Entry) ito.next();
```

```
String pName = (String) mi.getKey();
```

```
String[] s1 = (String[]) mi.getValues();
```

```
o.println( pName + " -- " );
```

```
for( String[] s2 : s1 )
```

```
{
```

```
o.println( s2 );
```

```
}
```

```
}
```

(2) $\text{http://www.durgsoft.com/orderServlet.do?uname=durga \& item=1238}$
 $\& item=456 \& item=789$. (24)

① getParameters("uname"); \rightarrow DURGA

② getParameters("item"); \rightarrow 1238

③ getParametersValues("users"); { DURGA }

④ getParametersValues("item"); { 1238, 456, 789 }

⑤ getParametersNames() \rightarrow Enumeration containing uname & item

⑥ getParametersMap();

uname	{ DURGA }
item	{ 1238, 456, 789 }

⑦ getParameters("age") \rightarrow null

⑧ getParameters("Priority") \rightarrow null

Retrieving Request Headers

\rightarrow For every Request Browser Sends its Configuration Information in the form of Headers. These may include the media types accepted by Browser, Encoding types supported by browser, the type of Browser etc.

\rightarrow Server uses these Request Headers to send proper Response to the browser.

\rightarrow The following is some of important Request Header.

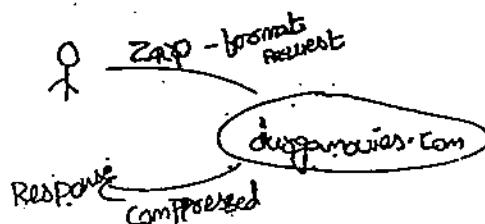
① Accept : media Types Accepted by Browser (PDF, ZIP, ZIP etc..)

② Accept-Encoding : Encoding Types supported by Browser (ZIP, GZip ..)

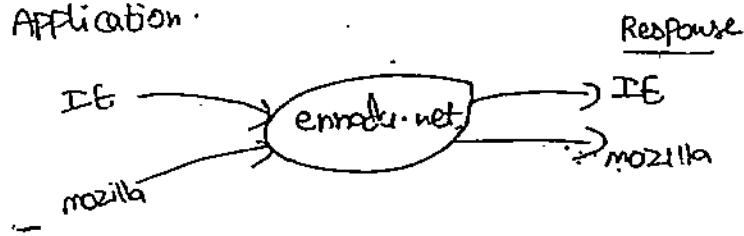
-) user-Agent: It represents the type of Browser (IE, mozilla)
-) Content-Length: It represents the length of request Body.
-) cookie: used to send cookies for Session Management.

Utilities of Request Headers

-) By using Accept-Encoding Request Header the Server can send compressed Data instead of Original Data. It reduces download time and improves performance



-) By using user-Agent Request Headers, the Server can send customized Response. we can increase hit-rate of the Application.



-) By using cookie Request Headers, we can send cookies to the Server so that we can achieve Session Management.
-) HttpServletRequest defines the following methods, to Retrive Headers Information at Servers side.

(25)

① String getHeader (String name)

- Returns the Value Associated with Specified Header.
- If the Specified Headers Associated with multiple Values then this method returns First Value.
- If the Specified Headers is not available it returns "null".
- Arguments are in Case-insensitive.

② Enumeration getHeaders (String headername)

- Returns all the Values Associated with Specified Headers.
- If the Specified Headers not available then this method returns Empty Enumeration Object but not null.
- Argument is case-insensitive.

③ Enumeration getHeadersNames()

→ HttpServletRequest Contains the following more convenient methods to retrieve int & Date Header Values.

① String s = req.getHeader ("Content-length");

{ int l = Integer.parseInt (s);
 replaced, start
 with }

→ int l = req.getIntHeader ("Content-length");

int getIntHeader (String name);

→ If the Specified Headers not Associated with int Value then this method raises NumberFormat Exception at RunTime.

2) long getDateHeader (String name)

- Returns the Date value associated with the header as the no. of milliseconds since Jan 1st 1970.
- If we can parse these millisec as the argument to Date constructor, then we will get Exact Date.

```
long l = req.getDateHeader
```

```
Date d = new Date (long l)
```

- If the specified header not representing Date value then this method rises Illegal Argument Exception.
- Demo program for displaying all Request Header sent by Browser

```
public class RequestHeadersDemoServlet extends HttpServlet
```

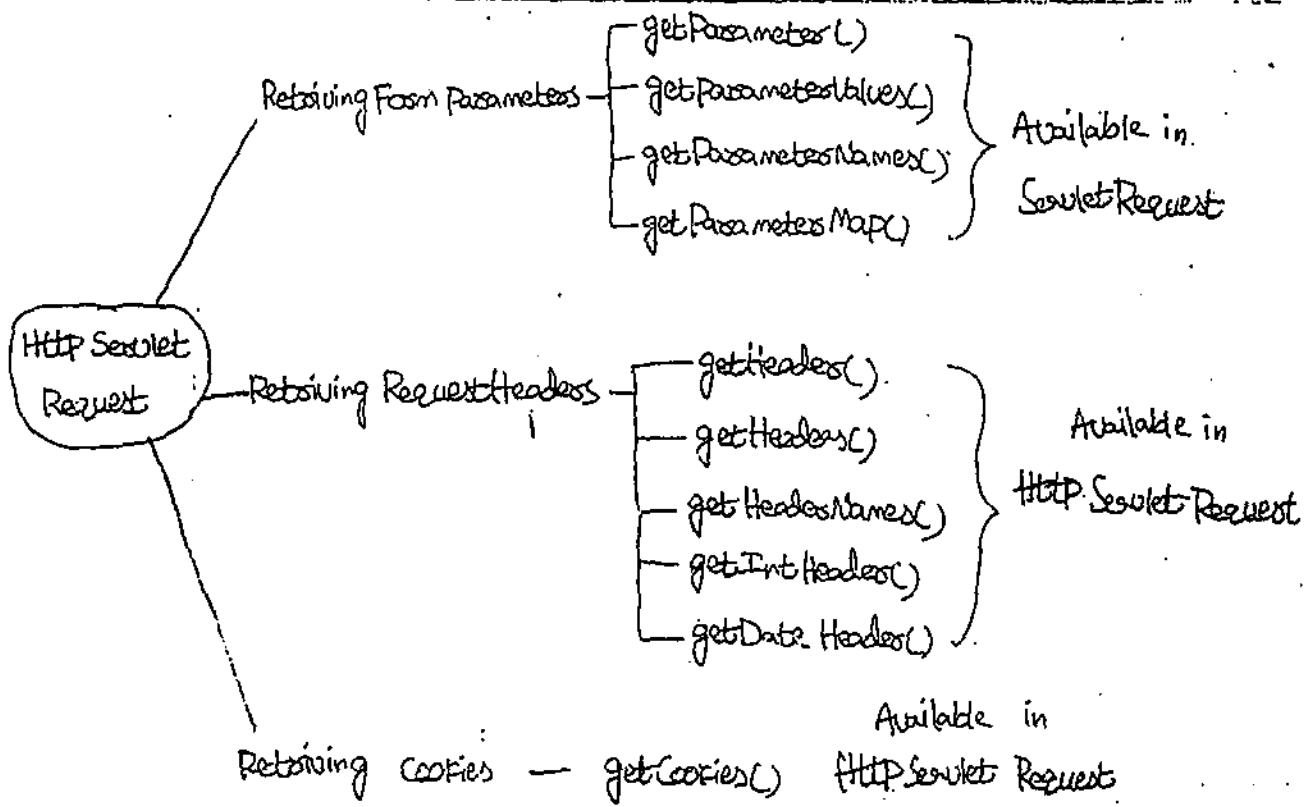
```
{  
    public void doGet(  
    {  
        PrintWriter out = resp.getWriter();  
        Enumeration e = req.getHeaderNames();  
        while (e.hasMoreElements())  
        {  
            String hname = (String)e.nextElement();  
            String hvalue = (String)req.getHeader(hname);  
            System.out.println (hname + " - - " + hvalue);  
        }  
    }  
}
```

Retaining Cookies From the Request

(26)

HTTP Servlet Request Contains the following methods to retrieve cookies from the Request object:

```
cookies[] c = req.getCookies();
```



Retrieving Client & Server Information from the Request

we can retrieve Client & Server Information from the Request by using the following methods of Servlet Request.

① Public String getRemoteHost()

— Returns Fully Qualified Name of the client which Sends the Request.

Ex: gmail.com, yahoo.com

② Public String getRemoteAddress()

— Returns the IP Address of the client which Sends the Request.

③ Public int getRemotePort()

— Returns the Port Number on which the client is Running.

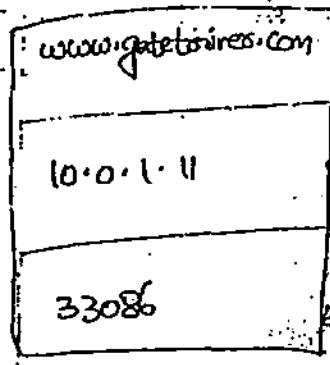
④ Public String getServerName()

— Returns the Name of the Server to which the request has Sent.

⑤ Public int getServerPort()

— Returns the Port Number at which the Server is Running.

Even a Request from www.gabetsainers.com with an IP Address 10.0.1.11 on Port 33086 place the appropriate Servlet Request methods on to their corresponding values.



getServerPort()
 getServerAddrs()
 getServerName()
 getRemotePort()
 getRemoteAddrs()
 getRemoteHost()

(27)

Http Servlet Response

→ By using `HttpServletResponse Interface` write code

- ① To Set Response Headers
- ② To Set Content Type of Response
- ③ To Acquire Text Stream for Response.
- ④ To Acquire Binary Stream for Response.
- ⑤ To Re-Direct Request to another URL.
- ⑥ To Add Cookies to the Response.

Setting Http Response Headers

→ `Http Response Headers` Describes the Configuration Information of the Server and Information abt the Response like Content Type, content length etc..

→ Browsers using these Response Headers to Display Response Body properly to the End user.

⇒ HttpServlet Response Defines the following methods to Add Headers to the Response.

- ① Public void addHeaders (String hname, String hvalue);
 - If the specified HeaderName is already available, to the existing values this new value will also be added.
 - Replacement will not occur.

~~x : z, y~~
~~(x, y)~~

- ② Public void setHeaders (String hname, String hvalue);
 - If the specified Header is already available then the old value will be replaced with new value.

~~x : z, y~~
~~(x, y)~~
w

⇒ sometimes Headers Associated with Int & Date Values.

HttpServlet Response Defines the following more convenient methods to add these methods.

- ③ Public void addIntHeader (String hname, int hvalue);
- ④ Public void setIntHeader (String hname, int hvalue);
- ⑤ Public void addDateHeader (String hname, long ms)
- ⑥ Public void setDateHeader (String hname, long ms)

(28)

Sets Content Type of Response

Content Type Headers Represents MIME Type (Multipurpose Internet Mail Extension) of the Response.

Common MIME Types

① text/html → HTML text as Response

② application/pdf → PDF file as Response

③ image/jpeg

④ Application/zip → Zip file as Response

We can set MIME Type by using the following 2 ways.

① By Servlet Response

- It contains the following method to set the Response

```
public void SetContentType (String mimeType);
```

Ex: resp.setContentType ("application/pdf");

② By HttpServletResponse

- It contains setHeaders()

Ex: resp.setHeaders ("Content-type", "text/html");

To Acquire Text Stream For the Response

We can send Text Data as the Response by using PrintWriter Object.

We can get PrintWriter Object by using getWriter() of ServletResponse Interface.

Public PrintWriter getWriter() throws IOException

Ex: PrintWriter out = resp.getWriter();

↳ To Acquire Binary Stream as Response

- we can send Binary Information (video files, image files etc)

by using ServletOutputStream Object

- By using getOutputStream() of ServletResponse we can get this object

Public ServletOutputStream getOutputStream() throws IOException

Ex: SDS os = resp.getOutputStream();

109 Demo program to Send Image file as Response

import java.io

Public class BinaryStreamDemo extends HttpServlet

{

 Public void doGET(--) ---

{

 resp.setContentType("image/jpeg");

 ServletOutputStream os = new ServletOutputStream();
 resp.get

```

String path = getServletContext().getRealPath("sunset.jpg");
file f = new File(path); (29)

FIS fis = new FIS(f);
byte[] b = new byte[(int)f.length()];
fis.read(b);
os.write(b);
os.flush();
os.close();

}

}

```

↓
 returns long
 so we have to
 convert it to int
 ⇒ [] accepts int

```

graph TD
    sunsetjpg[sunset.jpg] --> WEBINF[WEB-INF]
    sunsetjpg --> webxml[web.xml]
    sunsetjpg --> classes[classes]
    sunsetjpg --> BinaryStreamDemo[BinaryStreamDemo]
    WEBINF --> servlets[servlets]
  
```

Note:

At any Point of time we can get either PrintWriter object or ServletOutputStream Object but not both Simultaneously. otherwise we will get Runtime Exception Saying Illegal StateException.

Ex:

```

public void doGET(--) {
  PrintWriter out = resp.getWriter();
  SOS os = resp.getOutputStream();
}

```

(or)

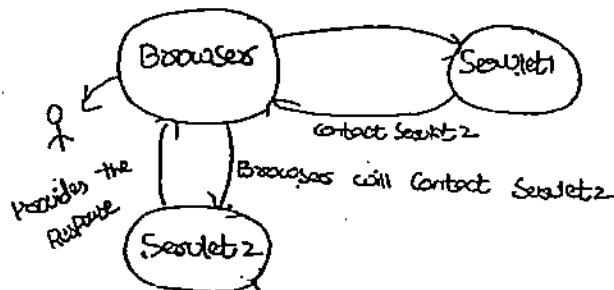
SOS os =
 PW out =
 ↓

Runtime Exception Saying - IllegalStateException
 getWriter() has already been called for this Response.

By using which of the following stream we can send both binary & Text data as response from the Servlet.

- ① PrintWriter
- ② ServletOutputStream
- ③ ServletInputStream
- ④ None of the above.

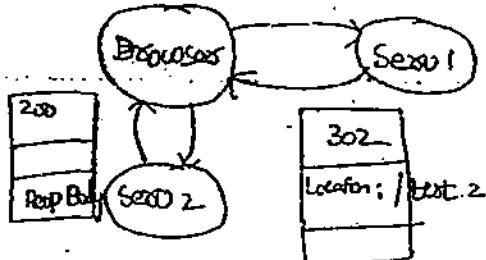
Re-Directing HTTP Request to Another URL



→ Sometimes we have to Re-Direct HTTP Request to another URL. We can achieve this by using `sendRedirect()` of `HttpServletResponse`.

`public void sendRedirect (String TargetPath) throws IOException`

Process of Send Re-direction



→ Here operation takes place at Browser (i.e. client side)

- Browser Sends a Request to Servlet 1.
- If Servlet 1 is not Responsible to provide Response, then it will update Browser to Contact Servlet 2. This can be informed by setting Status code 302 & Location Header with Servlet 2. 30
- Browser by seeing Status code 302 creates a new request object & sends to Servlet 2.
- Servlet 2 Provides Desired Response to the browser & the browser inton that Response to the End User.

Demo Program For Send-Redirect

import

```
public class FirstServlet extends HttpServlet
{
    public void doGet( -- ) -- -
    {
        resp.sendRedirect( "test2" );
        (or)
        resp.setStatus( "302" );
        resp.setHeaders( "Location", "http://test2" );
    }
}
```

public class SecondServlet extends HttpServlet

----- (--)
public void doGet(--)

PrintWriter out = resp.getWriter();

out.println("Hi This is second servlet");

else wherever we are sending Request to Frontend,
SecondServlet will provide the Response message.

Re-Direction.

→ After committing the Response we are not allow to response
Re-Direction. Otherwise we will get RuntimeException saying
IllegalStateException (Tomcat People doesn't provide Response
for this).

Public void doGet(--)

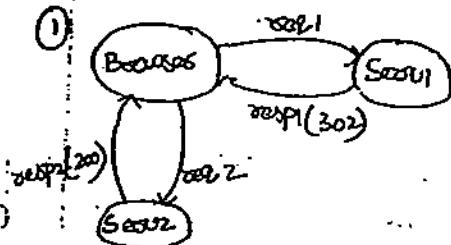
out.print("

else if (true)
{
 out.println("Hello
 world");
}

resp.sendRedirect("test2"); // IllegalState Exception

~~Q~~ 8) Differences b/w send-reDirection & Forward mechanism.

Send Re-Direct



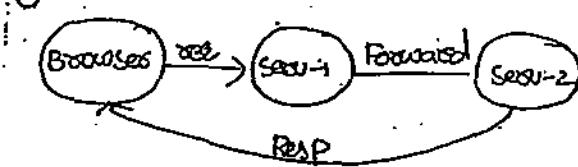
This mechanism will work at client-side. Hence client aware side and it is not visible to the client which servlet is providing the required response.

② In this approach an extra trip is required to the client side. Hence network traffic increases & effects the performance in/w traffic & performance problems of the system.

③ This is only possible mechanism to communicate with resources which are present outside of web container.

④ A separate new Request object will be created in send Re-Direction. Hence Information sharing b/w the components is not possible.

Forward



This mechanism will work at Server-side and it is not visible to the client. Hence client is not aware of which servlet is providing the required response.

② No extra trip is required to the client & hence there are no performance in/w traffic & performance problems of the system.

③ Forward mechanism will work with in the web container only & we can't use this mechanism to communicate with the resources present outside of web container.

④ The same Request object will be forwarded to the second servlet & hence information sharing b/w the components is possible in the form of attributes.

(5) By using HttpServlet Response Object we can achieve Send-Redirection.

(5) By using RequestDispatcher object we can achieve Forward mechanism.

(6) After committing the Response we are not allowed to perform Send-Redirection otherwise we will get RunTime Exception Saying Illegal State Exception.

(6) After committing Response we can't perform Forward otherwise IllegalStateException.

Adding Cookies to the Response

→ HttpServlet Response Contains the following method to add Cookie to the Response Object.

Public void addCookie(Cookie c)

Ex: cookie c = new Cookie(String name, String value);

resp.addCookie(c);

Forwarding a request to another URL

① Add Response Headers

```
addHeaders()  
setHeaders()  
addIntHeaders()  
setIntHeaders()  
addDateHeaders()  
setDateHeaders()
```

② Set ContentType

HTTP Servlet
Response

```
resp.setContentType("text/html");  
resp.setHeader("Content-Type", "text/html");
```

③ To Acquire Text Stream of Response

```
resp.getWriter();
```

④ To Acquire Binary Stream of Response

```
resp.getOutputStream();
```

⑤ To Perform Send Re-Direction

```
resp.sendRedirect("/test2");
```

```
resp.setStatus(302);
```

```
resp.setHeader("Location", "/test2");
```

⑥ Add Cookies

```
resp.addCookie(c);
```

UVII--

Structure and Deployment of Web Applications

- ① Directory Structure of web application
- ② web.xml
- ③ war file.

Directory Structure of web application

- (i) Context File & Directory Structure of web Application

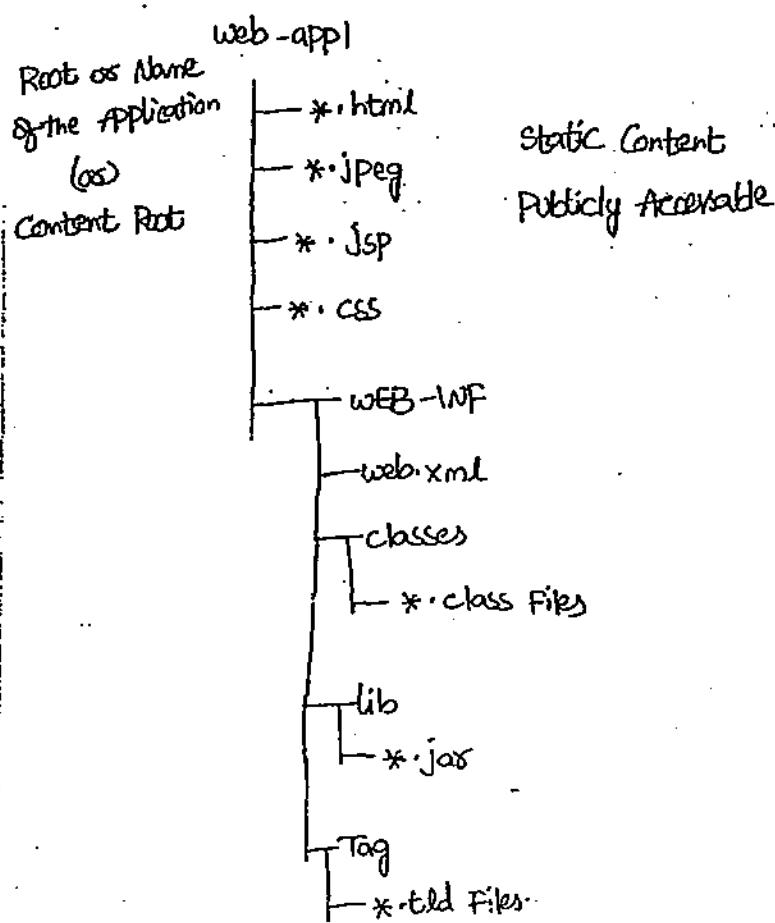
that may contain

- ① static Content
- ② JSP pages
- ③ servlet classes
- ④ Deployment Descriptor (DD)
- ⑤ Tag Libraries
- ⑥ zac Files
- ⑦ Java class Files.

→ servlet Specification Defines a Standard structure for web Application

so that every web servers can provide that Directory
structure, irrespective of vendor.

(32)



All the static Information we have to place with in the Context root Directly. All the Resources present in Context Root are Publicly Accessible Resource i.e any Person can Access these Resources Directly by its Name. Hence we can't place Secured Resource with in the Context Roots Directly.

For every web-Application we have to maintain WEB-INF folder. All the Secured Resources we have to place in WEB-INF. And these Resources are not Publicly Accessible. If any Person trying to Access these Resources directly by their name we

| will get 404 Status code.

http://localhost:8888/web-app1/WEB-INF/web.xml ↴

→ we have to place Deployment Descriptor (web.xml) in WEB-INF Directly.

→ web-Container will use this web.xml file to get Deployments Information.

→ with in the WEB-INF we have to place classes folder to Place all java.class files (both Servlet & General Utility classes).

→ web Container by Default will search for Required .class file in the classes folder only. Hence it is not required to set class path if we are placing .class files in the classes folder.

→ with in WEB-INF we have to maintain lib folder to Place all the Required jar files.

→ for the required .class file web Container first Search in the classes folder & If it is not available then only it will Search in the lib folder.

→ All the Tag libraries we have to place any where with in web-INF either Directly or Indirectly.

Web-Resource	Location to place	(34)
Static Content	① with in Context Root Directly.	
JSP Pages	② with in Context Root Directly. If JSP is secured we have to place inside web-INF.	
Servlet classes	③ Inside Classes folder	
Deployment Descriptors	④ with in web-INF Directly.	
Tag Libraries	⑤ Anywhere with in WEB-INF either Directly or Indirectly.	
Jar Files	⑥ Inside lib folder	
Java classes files	⑦ Inside classes folder.	

Deployment Descriptor

Describe the Purpose, Semantics & the correct structure of the following Deployment Descriptor Elements.

- ① Error Page
- ② Init-param
- ③ `MIME-mapping >`
- ④ `<Servlet>`
- ⑤ `<Servlet-mapping >`
- ⑥ `<Servlet-class >`
- ⑦ `<Servlet-name >`
- ⑧ `<url-mapping >`

- Deployment Descriptor is an xml file named with web.xml and should be placed in WEB-INF folder directly.
- Web Container uses this web.xml to get Web-Applications deployment information i.e. web.xml acts as a guide to the web-container.
- web.xml provides declarative mechanism for customising our web-application without touching the source-code.
(to change username, port, to change url patterns, Initialization parameters in web.xml).
- For every web application we have to maintain exactly one web.xml.

Anatomy of web.xml

```

<web-app>
    <description>
    <display-name>
    <servlet>
    <servlet-mapping>
    <welcome-file-list>
        <Error-page>
        <mime-mapping>
        <context-param>
        <Security-constraints>
        <jsp-config>
        <session-config>
    </web-app>
  
```

In total there are 27 top level child tags are possible

- until Servlet 2.3 V. web.xml is validated by using DTD's (35)
 - where the order is important. Hence until Servlet 2.3 V the order of these Top-Level Elements is important.
- But from Servlet 2.4 V. onwards web.xml is validated by using Schemas where the order is not important. Hence from Servlet 2.4 V onwards we can take these Top-Level Elements in any Order.
- Among all 27 Top Level Tags no tag is mandatory. Hence the following are also valid web.xml's.

`<web-app>
.....
</web-app>`

[or]

`<web-app />`

a) `<Servlet>`

b) we can use `<Servlet>` to declare a servlet in web.xml.

c) Servlet Tag Contains the following 9 child Tags.

```
<web-app>
  .
  • <servlet>
    <description>
    <display-name>
    <icon>
    <servlet-name>
    <servlet-class> [or] <jsp-file>
    <init-param>
    <load-on-startup>
    <run-as>
    <security-role-ref>
  </servlet>
  !
</web-app>
```

→ The order of these 9 tags is important and we should not change.

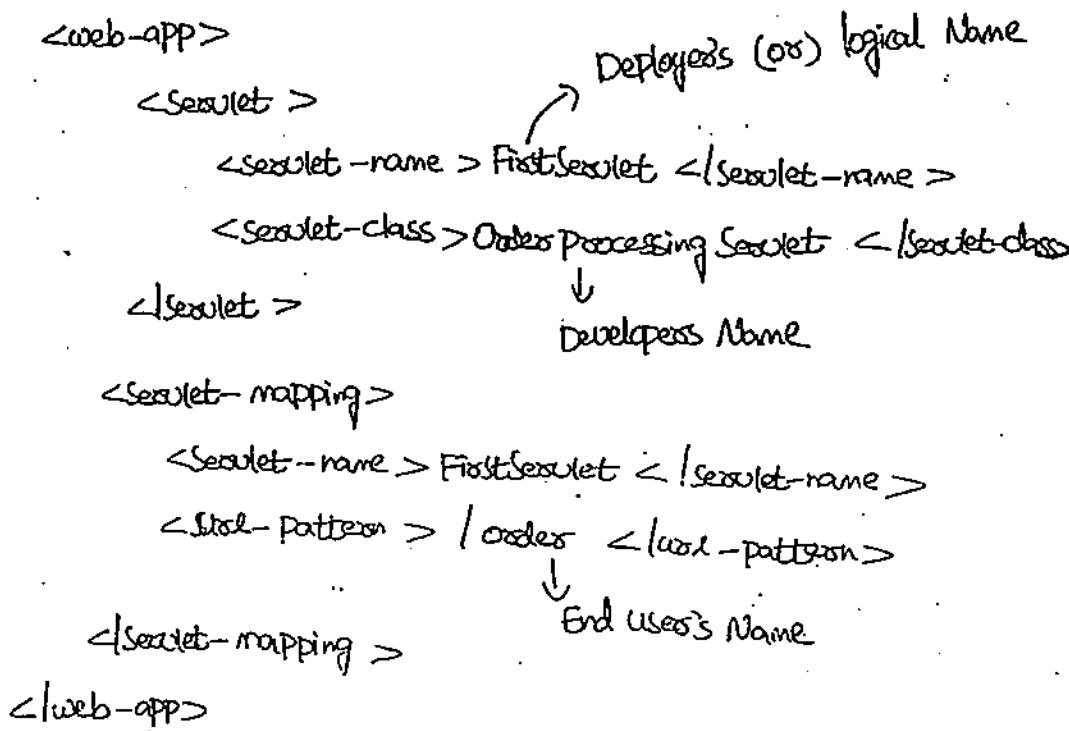
→ Among these 9 tags the following 2 are mandatory

- ① <Servlet-name>
 - ② <Servlet-class> [or] <jsp-file>

② <Soviet-name>

→ According to Servlet-Specification there are 3 names are possible for every Servlet.

- ① Developer's Name Specified by <Servlet-class>
- ② Deployer's Name [or] logical-name Specified by <Servlet-name>
- ③ End-user's Name Specified by <url-pattern>



→ The main Advantages of Logical-Name are

- ① we can achieve Security bcz we are not highlighting our Internal Naming Convention to the End-user.
- ② we can map the same Servlet for Different url-patterns.
- ③ without affecting End-user we can change our Internal-Naming Convention.

→ Within the web-application the logical name should be unique.

→ we can get logical-name of Servlet with in the Servlet class by using getServletName() of ServletConfig Interface.

```
public String getServletName()  
  
ex: public void doGet(--)--  
{  
    PrintWriter out = resp.getWriter();  
    String name = getServletName();  
    out.println("The Logical Name is," + name);  
}
```

<Servlet-class>

→ By using this Tag we have to specify the Original Name of Servlet.

→ For this class only web-Container performs Instantiation.

Output Configuring JSP in web.xml

→ Generally JSP's we can place in Context Root Directly.

In this case End-users can access directly by its name.

→ But if the JSP is secure then it is not recommended to place in the Context Root Directly. we have to place such type of Resources Inside WEB-INF and we have to provide access through URL-pattern

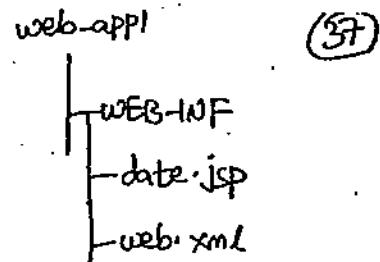
→ we can configure JSP by using <jsp-file> in web.xml.

Demo Program

```
<h1> The Server Time is :
```

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

```
</h1>
```



→ we can place JSP files in classes folder also. In that case we have to specify hole path in <jsp-file>

```
<servlet-name> JspDemo </servlet-name>
```

```
<servlet>
```

```
<jsp-file> /WEB-INF/date.jsp </jsp-file>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
<servlet-name> JspDemo </servlet-name>
```

```
<url-pattern> /test </url-pattern>
```

```
</servlet-mapping>
```

```
</web-app>
```

```
<init-param> (Servlet Initialization Parameters)
```

→ If the value of a variable will change frequently those values, it is not recommended to hard-code inside Servlet class.

→ Bcz for every modification it requires to recompile the Servlet class, Rebuild the application, Redeployment & Some times even require Server Restarts also.

→ Such type of variables we have to configure in web.xml

by using <init-param>. To reflect the changes Just Re-Deployment is enough.

<web-app>

<Servlet>

0 to many {

<init-param>

<param-name> user </param-name>

<param-value> Scott </param-value>

</init-param>

</Servlet>

</web-app>

→ we can configure any number of Initialization Parameters

but for each parameter one <init-param>

→ with in the Servlet we can access Servlet Initialization parameters by using ServletConfig Object.

→ ServletConfig Interface defines the following methods to access these parameters Inside Servlet.

① Public String getInitParameter (String name)

— Returns the value associated with Specified Initialization Parameters

— If the specified Parameters is not available then this method returns "Null".

(2) Public Enumeration getInitParameterNames()

(38)

→ If the Servlet doesn't contain any Initialization parameters then this method returns Empty Enumeration Object but not null.

→ Generic Servlet implements ServletConfig Interface, hence Generic Servlet provides implementation for the above 2 methods.

Demo Program for Displaying all Servlet Initialization Parameters.

```
<web-app>
  <Servlet>
    <s-n> DemoServlet </s-n>
    <s-c> InitializationParameters </s-c>
    <init-param>
      <param-name> Phone number </param-name>
      <param-value> 9848022338 </param-value>
    </init-param>
    <init-param>
      <param-name> MailID </param-name>
      <param-value> Info@diagsoft.com </param-value>
    </init-param>
    <init-param>
      <p-n> Username </p-n>
      <p-v> Scott </p-v>
    </init-param>
  </Servlet>
```

```
<Servlet-mapping>
  <s-n> DemoServlet </s-n>
    <url-pattern> /test </url-pattern>
```

```
</s-m>
```

```
</web-app>
```

```
import javax.servlet.*;
```

```
public class InitializationParameters extends HttpServlet
```

```
{
```

```
  public void doGet( -- )--
```

```
{
```

```
  PrintWriter out = resp.getWriter();
```

```
  Enumeration e = getInitParameterNames();
```

```
  while (e.hasMoreElements())
```

```
{
```

```
  String pName = (String) e.nextElement();
```

```
  String pValue = getInitParameter("pName");
```

```
  out.println(pName + " --- " + pValue);
```

```
}
```

```
}
```

```
}
```

⇒ Servlet Initialization Parameters are key-value pairs where both key & value are String Objects only.

From the Servlet we can access these parameters but we
can't modify them. i.e. we have only getter methods
but there are no setter methods. 39

Hence these Initialization Parameters are considered as
Deployment time Parameters.

Servlet-Config

For Every Servlet web-containers creates one Servlet-Config Object.

By using Servlet-Config Object Servlet can get its Configuration
Information.

Servlet-Config Interface defines the following 4 methods.

- ① public String getServletName();
- ② public String getInitParameter(String name);
- ③ public Enumeration getInitParameterNames();
- ④ public ServletContext getServletContext();

<load-on-startup>

usually Servlet class Loading, Instantiation, Execution of init() will
takes place at the time of first request. It Increases the
Response Time of the first Request.

If we are Configuring <load-on-startup>, these steps will
be performed at the time of either Server Startup [or]
Application Deployment.

```
<web-app>
  <Servlet>
    !
    <load-on-startup> 10 </load-on-startup>
  </Servlet>
</web-app>
```

- The main Advantage of load-on-startup is, we can maintain uniform Response Time for all the Requests
- The Disadvantage of this Tag is, Creating Servlet Object at the beginning may effect performance, cause memory problems.
- The Servlet whose <load-on-startup> Value is less will be loaded First.
- If 2 Servlets having the same load-on-startup value or if the load-on-startup Value is negative then we can't predict orders of loading. It is vendor dependent.

14/04/04 <Servlet-mapping>

(40)

- → By using this tag we can map a Servlet with the url-pattern
- → upto Servlet 2.4 V with in the <Servlet-mapping> a single Servlet can map to exactly one <url-pattern> but from Servlet 2.5 V onwards a single Servlet can map with multiple url-patterns i.e. we can take multiple <url-pattern> Tags inside Servlet mapping.

<Servlet-mapping>

```
<Servlet-name> FirstServlet </Servlet-name>
<url-pattern> /test </url-pattern>
<url-pattern> /hello </url-pattern>
<url-pattern> /order </url-pattern>
</Servlet-mapping>
```

- → This is valid in Servlet (2.5V) (Tomcat - 6). But Invalid in Servlet 2.4 V (5.x)

- → According to Servlet Specification these are 4 types of url-patterns are possible.

- ① Exact matching url-pattern

e.g. /test

- ② Longest Path Prefix url-pattern

e.g. /test/test/*

- ③ url-pattern by Extension

*.jsp e.g. *.do, *.jsp

- ④ Default url-pattern

e.g. /

Q1) Which of the following comparisons are valid?

- (a) /test
- (b) /
- (c) /test/test/* /test
- (d) /test/* .jsp
- (e) *.do or /*.durga.

→ web-Container always gives the precedence in the following order

- ① Exact match
- ② LongestPath Prefix
- ③ By Extension:
- ④ Default url-pattern.

→ If no other url-pattern matched then only default url-pattern will get a chance.

Demo program for url-patterns:

- ① /test — FS
- ② /test/test/* — SS
- ③ *.do — TS
- ④ / — Default Servlet

Http://localhost:8080/scwcd2c/test → First Servlet (41)

/test/test/durga.do → Second Servlet

/test/durga.do → Third Servlet

/durga.do → Third Servlet

/durga → Default Servlet

How we can Configure DefaultServlet in web.xml & in which cases Default Servlet will get the chance.

with "/" we can Configure DefaultServlet. If no Servlet url-pattern is matching the DefaultServlet will be executed.

⇒ Getting Information from the URL.

→ Servlet Request Interface defines the following methods for this.

- ① getHttpServletRequest();
- ② getServletContext();
- ③ getServletPath();
- ④ getPathInfo();
- ⑤ getQueryString();

```

public class FS extends HttpServlet
{
    public void doGET( -- )
    {
        PrintWriter out = resp.getWriter();
        out.println("<h1> RequestURI : " + req.getRequestURI() + "</h1>");
        out.println("<h1> ContextPath : " + req.getContextPath() + "</h1>");
        out.println("<h1> ServletPath : " + req.getServletPath() + "</h1>");
        out.println("<h1> PathInfo : " + req.getPathInfo() + "</h1>");
        out.println("<h1> QueryString : " + req.getQueryString() + "</h1>");
    }
}

```

<w-a>

FS



/test/test/* -

<w-a>

http://localhost:8080/ScwCD2CA/test/test/durga/software?

uname = Durga & value = DSS

RequestURI - /ScwCD2CA/test/test/durga/software

ContextPath - /ScwCD2CA

ServletPath : /test/test

(ii)

PathInfo : dooga/software

QueryString: uname=Douga & uvalue=DSS

http://localhost:8080/scwcd2ca/test/test/

RequestURI: /scwcd2ca/test/test

ContextPath: /scwcd2ca

ServletPath: /test/test

PathInfo: /

QueryString: null

http://localhost:8080/scwcd2ca/test/test

PathInfo: null

QueryString: null

10) Configuring Welcome Pages for web Application.

- It is recommended to configure welcome files in web-application
- we can configure welcome-files in web.xml as follows

```
<web-app>
    <welcome-file-list>
        <welcome-file> home.jsp </welcome-file>
        <welcome-file> login.jsp </welcome-file>
    </welcome-file-list>
</web-app>
```

> `<welcome-file-list>` is the Direct child Tag of `<web-app>` and we can take anywhere.

> we can Configure more than one welcome file but the order is important.

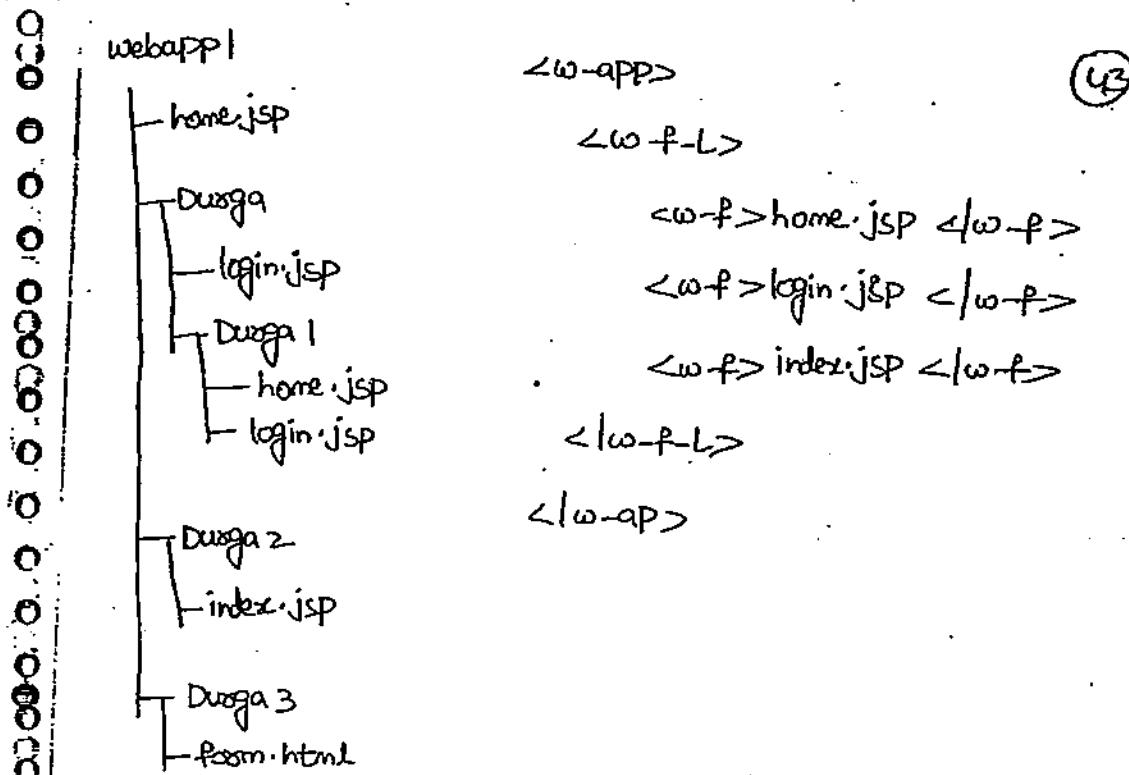
In any web-Application `Index.html` acts as Default welcome page. If it is not available then `Index.jsp` acts as Default welcome page.

> whenever we are configuring welcome-files explicitly in `web.xml` then this Default welcome-file is not applicable.

we can Configure welcome files even Folders wise also.

According to Servlet Specification leading or trailing back slashes are not allowed for the `<welcome-files>`.

Ex: `<welcome-file> /login.jsp </welcome-file>` Invalid



http://localhost:8080/web-app1 - home.jsp

/web-app1/durga-login.jsp
 /web-app1/durga/durga1 - home.jsp
 /web-app1/durga2 - Index.jsp
 /web-app1/durga3 - 404 status code forces.

If we have both index.html & index.jsp, Index.html will be executed.

If we are not using <w-f-l> then only Default index.html will be executed.

Default.html (or) JSP is not applicable for sub-folders. But only for context root folder.

Configuring Error pages in web.xml

It is not recommended to send Errors Information Directly to the End user. we have to Convert Java Errors Information in to End user understandable form.

we can achieve this by Configuring Errors pages in web.xml.

we can Configure Error page either based on Status Code or based on Exception Type.

Configuring Error Page based on Exception Type

```
<web-app>
```

```
    <error-page>
```

```
        <exception-type>java.lang.ArithException</exception-type>
```

```
        <location>/error.jsp</location>
```

```
    <error-page>
```

```
</web-app>
```

Configuring Error page based on Status Code.

```
<web-app>
```

```
    <error-page>
```

```
        <error-code>404</error-code>
```

```
        <location>/error404.jsp</location>
```

```
    <error-page>
```

```
</web-app>
```

Public class FS extends HttpServlet

(44)

```
{  
    public void doGet( -- ) --  
    {
```

```
        PW out = resp.getWriter();
```

```
        open (10/0);
```

```
}
```

Error.jsp

<hi> Your Provided Values

are not correct plz provide

valid Input. </hi>

```
<web-app>
```

```
    FS
```

```
    ↓
```

```
    /test
```

```
    <error-page>
```

```
        <exception-type> Java.lang.A-E </exception-type>
```

```
        <location> /error.jsp </location>
```

```
    </error-page>
```

```
</web-app>
```

⇒ If we are sending Request to the Servlet Instead of getting Arithmetic Exception we will get error page as response.

Note
→ We can set Error Code programmatically.

→ HttpServlet Response Contains the following method for this.

```
Public void sendError (int ErrorCode);
```

```
Ex: resp.sendError (503);
```

login.html

```
<form action = "/Scuod/test" >  
Enter name : <input type = "text" name = "uname" >  
<input type = "submit" >  
</form action>
```

DS.java

```
Public class DS extends HttpServlet
```

```
{  
    Public void doGet (--)--  
    {  
        PW out = resp.getWriter ();  
        String name = req.getParameter ("uname");  
        If (name.equals ("Durga"))  
        {  
            o.println ("

# Authentication Successfull

");  
        }  
        Else  
    }
```

```
    resp.sendRedirect( "503" );
```

(45)

```
}
```

```
}
```

web.xml

```
<web-app>
```

```
    <error-page>
```

```
        <error-code> 503 </error-code>
```

```
        <location> /error503.jsp </location>
```

```
    </error-page>
```

FirstServlet



/test

```
</web-app>
```

Error503.jsp

```
<h1> your authentication is fail ... Plz provide valid credentials
```

</h1>

(we can keep hyperlink to login page)

- ① We can Configure Error Page either based on ErrorCode or ExceptionType but not both Simultaneously in the same Error-Page Tag.

~~`<error-page>`~~ X → Invalid Configuration.

`<error-code> 503 </error-code>`

`<exception-type> java.lang.AE </exception-type>`

`<error-page>`

- ② The Exception-Type Tag Value Should be Fully Qualified Name of Exception.

~~Ex:~~

Fully Qualified Name Required

i.e. `java.io.IOException`

`<error-page>`



`<exception-type> IOException </exception-type>`

`<location> /error.jsp </location>`

`<error-page>`

- ③ The `<location>` value must be Compulsory Starts with "/".
otherwise we will get Deployment Problems.

~~Ex:~~

`<error-page>`

`<error-code> 503 </error-code>`

`<location> error.jsp </location>`

`<error-page>`

"/" must be Required

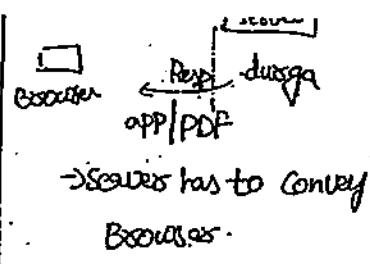
i.e. `/error.jsp`

Q) <mime-mapping>

→ we can use <mime-mapping> to map file

Extension with the corresponding <mime-type>

(46)



Ex:-

* * <mime-mapping>

<extension> duo ga </extension>

<mime-type> application/pdf </mime-type>

</mime-mapping>

→ <mime-mapping> is the Direct child Tag of <web-app> & Hence we can take any cohesive.

Wax File

Q) Explain the purpose of wax file & Describe the contents of wax file & the process of Construction.

→ wax file (web-Archive) provides a convenient way to store the resources of a web-application in to a single Component.

→ we can deliver, Transport & Deploy web-application very easily. If it is available in the Wax-File form.

→ Sun People provides a Standard Structure for the wax-file & every web-Servers provides support for that wax-file.

→ A wax-file Represents a web-Application whereas as ear-file represents an Enterprise Application.

→ Jar-file Represents a group of .class files.

Commands

- ① Creation of war-file

```
jar -cvf myapp.war *.*
```

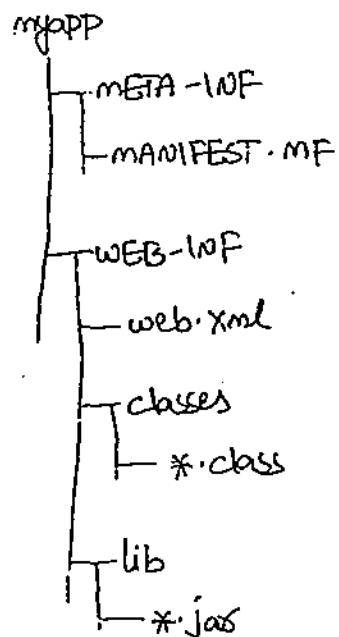
- ② Extraction of war-file

```
jar -xvf myapp.war *
```

- ③ Display Contents of war-file with Out Extraction

```
jar -tf myapp.war
```

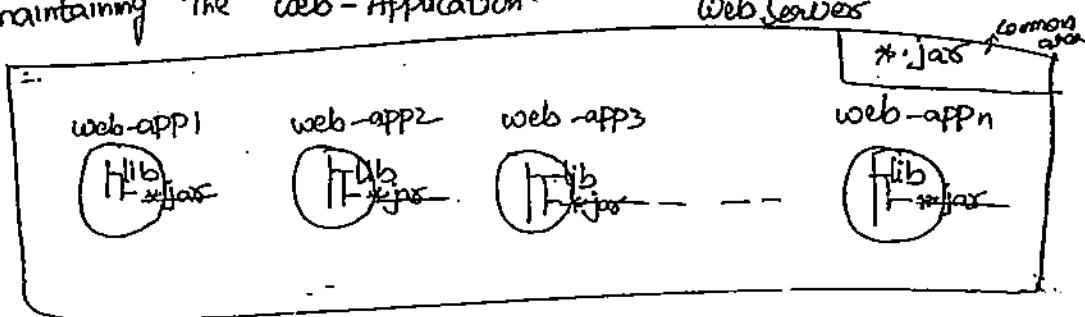
Structure of war-file



META-INF

(47)

- Every war-file should Compulsory Contain meta-inf folder which should contain MANIFEST.MF . i.e META-INF & MANIFEST.MF are mandatory for any war-file.
- META-INF folder contains Security Related Resources like signature files, Digital certificates which are necessary for maintaining the web-Application.



MANIFEST.MF

- The library Dependencies of a web-Application we can define in this file only.

- It contains the class Path of the common Jar Files stored outside of web-Application.

- If there is any Jar File which is common to Several web-Application then it is not recommended to place that Jar file at application level. we have to place such type of Jars at some Common location outside of web-application & we can define their paths in MANIFEST.MF.

- If we are trying to Access the Resources Present in META-INF

(or web-INFO) we will get 404 status code i.e these resources are not publicly Accessable.

Http://localhost:8080/myapp/mETA-INF/manifest.MF

↓
404 - Status Code Error?

UNIT-3

(48)

The web Container Model

Servlet Context

For the Servlet Context Initialization parameters

① Write Servlet Code to Access Initialization Parameters.

② Create the Deployment Descriptor Elements for the Initialization parameters.

For every Servlet, web Container will create one Servlet Config Object to maintain Servlet Level Initialization parameters. By using this object Servlet can get its Configuration Information.

Similarly for every web-Application, web container creates one Servlet Context object to maintain Application level Configuration Information.

Servlet can get Application level Configuration Info through this Context Object only.

Servlet Config per Servlet whereas Servlet Context per web-Application.

If Initialization Parameters are common for several Servlets then it's not recommended to declare those parameters at Servlet level. we have to declare such type of parameters at Application-level by using <Context-param>

```
<web-app>
```

```
<context-param>
```

```
  <param-name> username </param-name>
```

```
  <param-value> scott </param-value>
```

only

```
</context-param>
```

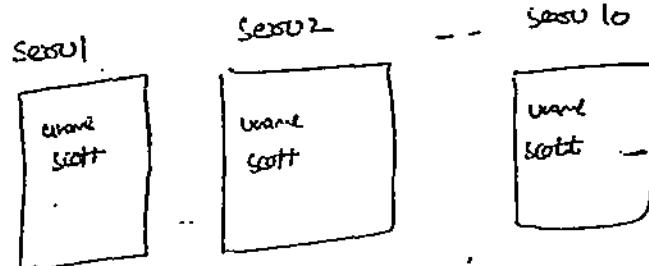
!

```
</web-app>
```

→ we can declare any number of context parameters but one `<Context-param>` for each parameter.

→ `<context-param>` is the direct child Tag of `web-app` & Hence we can declare anywhere within `<web-app>`

→ The Context Initialisation parameters are available through out the web application anywhere.



```
<init-param>
```

```
<p-n> un
```

```
<p-v> scott
```

```
<i-p>
```

to
servlet

```
<i-p>
```

```
<p-n>
```

```
<p-v>
```

```
<i-p>
```

→ If we want to use same init-parameters then we can declare at Context-level

as

```
<Context-param>
```

```
<p-n>
```

```
<p-v>
```

```
</context-param>
```

→ with in the Session we can access these Context Initialization parameters by using SessionContext Object. (49)

→ we can get SessionContext Object by using getSessionContext() of SessionConfig Interface.

```
SessionContext context = getSessionContext();
```

→ SessionContext Interface defines the following methods for accessing Context Initialization Parameters.

① String getInitParameter (String name);

② Enumeration getInitParameterNames();

Demo program to display Context Initialization Parameters.

```
Public class SDemoServlet extends HttpServlet
```

```
{
```

```
    Public void doGet(--) --
```

```
{
```

```
    PW out = resp.getWriter();
```

```
    SessionContext context = getSessionContext();
```

```
    Enumeration e = context.getInitParameterNames();
```

```
    while (e.hasMoreElements())
```

```
{
```

```
        String pName = (String) e.nextElement();
```

```
        String pValue = context.getInitParameter("pName");
```

```
        } .println(pName + "----" + pValue);
```

web.xml

<web-app>

<i-p>

<p-n> urine </p-n>

<p-v> scott </p-v>

<li-p>

<i-p>

<p-n> pwd </p-n>

<p-v> tigers </p-v>

<li-p>

ScDemoServlet



/test

<web-app>

top

Note:

→ we can access Servlet Initialization parameters in the following ways.

```
String value = getInitParameter("movie");
```

```
String value = getServletConfig().getInitParameter("movie");
```

→ we can access Context Initialization parameters in the following ways.

```
String value = getServletContext().getInitParameter("movie");
```

```
String value = getServletConfig().getServletContext().getInitParameter("movie");
```

Composition between Servlet & Context Initialization parameters.

Properties

① Deployment Descriptor Declaration

Servlet Init-param

By using `<init-param>`
with in `Servlet`

```
<Servlet>
  <init-param>
    <param-name>
      <param-value>
    </init-param>
```

```
</Servlet>
```

Context Init-param

By using `<Context-param>`
with in `<web-app>`

```
<web-app>
  <context-param>
    <param-name>
      <param-value>
    </context-param>
</web-app>
```

Composition between

Initialization parameters

→ By default Initialization parameters means

② Servlet code to access the parameters

```
String value = getInitParameters("pname");
```

[or]

```
String value = getServletConfig().
```

```
getInitParameters("pname");
```

```
String value = getServletConfig().getInitParameters("pname");
```

[or]

```
getServletConfig().getInitParameters("pname");
```

```
String value = getServletConfig().getInitParameters("pname");
```

⑤ Availability (scope)

Available only for a Particular Session in which <init-param> is declared.

Available for all Session & JSP's within in the web-application.

Note

whether Session or Context Init-Parameters these are Deploy-time Constants i.e from the Session we can Read their values but we are not allow to modify i.e. we have only getXXX()'s but not setXXX()'s.

Servlet Scopes (2 to 3)

(51)

Q) For the fundamental Servlet Attribute Scopes (Request, Session, Context).

a) Write ServletCode to add, Retrive & Remove Attributes

b) Given a usage Scenario Identify the proper Scope.

c) Identify multiThreading Issues Associated with each Scope.

Q) There are 3 types of parameters are possible.

① Form Parameters

② Servlet Initialization Parameters

③ Servlet Context Parameters

These parameters are Read only i.e from the Servlet we can perform Read Operation, we can't modify & Remove these values based on our requirement. Hence Parameters are not useful for sharing the data btw the components of a web-App.

To resolve this problem Sun people introduced Attributes Concept.

Based on our requirement we can create a new Attribute or we can modify & Remove existing Attributes. Hence Attributes are best suitable for storing & sharing the Data btw the components of web-Applications.



→ Based on our Requirements we have to save the Attributes in the Proper scopes.

→ There are 3 scopes are possible for the Servlets.

① Request Scope - ATM - credit card.

② Session Scope - login → name, pass for one user

③ Application / Context Scope Globally, for any number of users.

Request Scope

→ we can maintain this scope either by using `ServletRequest` or `HttpServletRequest` Object.

→ Request Scope will start at the time of request Object creation (i.e Just before starting Service method) & ends at the time of Request Object Destruction (i.e Just after completing service()).

→ The Data (or Attribute) stored in the Request Scope are available to all Components which are processing that Request.

→ Servlet Request Interface defines the following methods for Attribute management in the Request Scope.

① Public void setAttribute (String name, Object value).

- If the specified Attribute is already available then the old value is replaced with new value.

(52)

② Public Object getAttribute (String name).

- Returns the value associated with specified Attribute Name.

- If the specified Attribute is not available then this method returns "null".

③ Public void removeAttribute (String name)

④ Public Enumeration getAttributeNames();

~~example~~

→ The most common Area where we can use Request Scoped Attributes is Request Dispatcher Forward & Include mechanisms.

→ For every Request a separate new request object will be created which is responsible Accessed by only the current thread.

other Threads are not allowed to Access Request Scoped Attributes. Hence Request SCOPED Attributes are always

Thread safe.

Session Scope

→ Session scope is maintained by HttpSession Object

HttpSession hs = req.getSession();

Session scope will start at the time of Session Object creation

& will lost at the time of session object destruction. It

may be either by invalidate() or by TimeOut mechanism

→ The information stored in the Session Scope is available for

- all the components which are participating in that session.
- HttpSession Interface defines the following methods for Attribute management in Session Scope.

- ① Public void setAttribute(String name, Object value);
- ② Public ~~Object~~ getAttribute (String name);
- ③ Public void removeAttribute (String name);
- ④ Public EnumerationgetAttributeNames();

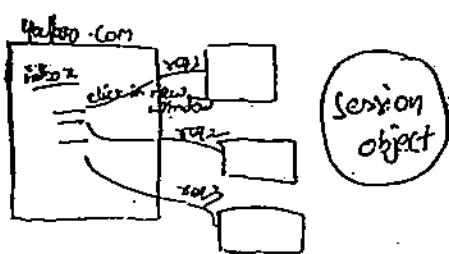
- once the Session Expires we are not allowed to call these methods violation leads to Runtime Exception Saying

IllegalStateException

Example

- login Information should be available for Entire Session.
Hence we have to store this information in the Session Scope.

- ⇒ within the same Session we can send multiple Requests simultaneously by opening new Browser windows. Hence Session Objects can be Accessed by multiple Threads simultaneously & Hence Session Scoped Attributes are not Thread Safe.



Application Scope

- Application Scope is maintained by Servlet Context Object.
- It Starts at the time of Context Object creation (ie at the time of Application Deployment or Server startup) & will be lost at the time of Context Object Destruction. (ie At the time of Application Un-deployment or Server Shutdown).
- The Data stored in the Application scope will be available to all the Components of web-application, irrespective of request & users.
- ServletContext Interface defines the following methods for Attribute management in the Application Scope.
 - ① Public void setAttribute (String name, Object value);
 - ② Public ~~void~~ ^{Object} getAttribute (String name);
 - ③ Public void removeAttribute (String name);
 - ④ Public EnumerationgetAttributeNames();
- Context Object can be Accessed Simultaneously by multiple Threads & hence Context Scoped Attributes are not Thread-Safe.

→ Instance & Static Variables can be accessed multiple threads

Simultaneously & Hence these are not Thread Safe

→ For every Thread a Separate Copy of Local Variables will be created & Hence Local Variables are always Thread Safe.

public class DS extends DS

{
 int x=0; — Not Thread Safe

 Static int y=0; — Not Thread Safe

 Public void Service () —

 {
 int z=0; Thread Safe

}

}

members

ThreadSafe

1) Requester Scope Attributes

Yes

2) Session Scope "

No

3) Context Scope "

No

4) Instance Vlbs

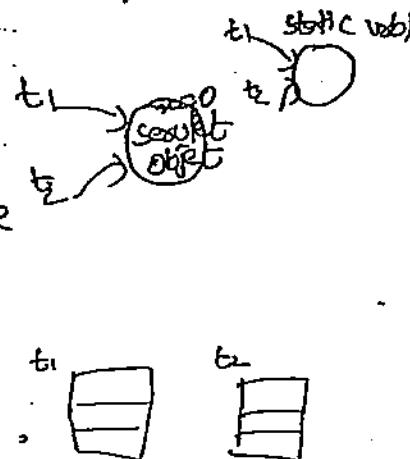
No

5) Static Vlbs

No

6) Local Vlbs

Yes



Parameters are key-value pairs where both key & value are String objects only. Hence at the time of Retrieval it is not required to perform any typecasting. We can assign directly parameter value to the String directly.

String value = request.getParameter("uname");

String value = request.getInitParameter("uname");

Attributes are also key-value pairs. But keys are String objects but values can be any type of object. Hence at the time of Retrieval compulsory we should perform typecasting.

X String value = request.getAttribute("uname"); → CTError

Incompatible Type

found : Object

required : String

✓ String value = (String) request.getAttribute("uname");

To access the value of RequestScoped Attribute uname which of the following is valid way.

String value = request.getParameter("uname");

String value = request.getInitParameter("uname");

3) String value = getInitParameter ("uname");

4) String value = request.getAttribute ("uname"); return type is object

5) String value = (String) request.getAttribute ("uname");

Differences b/w Parameters & Attributes:

Parameters :

- ① Parameters are Read only i.e. within the Servlet we can perform Read operation but we can't modify their values i.e. we have only getMethod's but not setMethod's.

- ② Parameters are Deployment Time Constants.

- ③ Parameters are key-value pairs & both key & value are String objects only.

Key - String

Value - String

- ④ At the time of Retrieval it is not necessary to perform Type Casting.

Attribute :

- ① Based on our requirement we can get & set the Attributes i.e. these are not Read only & we have both getMethod's & setMethod's.

- ② Attributes are not Deployment Time Constants.

- ③ Attributes are Key-value pairs where key is String but value can be any type of object.

Key - String

Value - Object. (any object)

- ④ At the time of Retrieval we should perform Type Casting.

Demo program to Display Hit-Count (no. of request.) of web-Application.

(55)

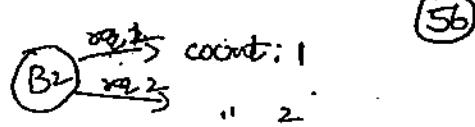
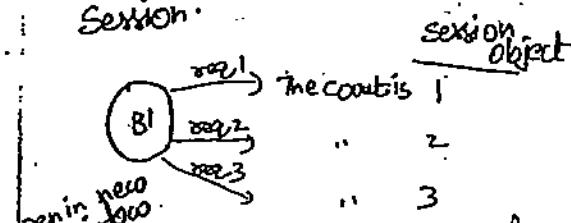
```
import java.util.*;  
public class HitCountDemo extends HttpServlet {  
    public void doGet(---) {  
        PrintWriter out = resp.getWriter();  
        ServletContext context = getServletContext();  
        Integer count = (Integer) context.getAttribute("hitcount");  
        if (count == null)  
            count = 1;  
        else  
            count++;  
        context.setAttribute("hitcount", count);  
        out.println("The hit count is " + count);  
    }  
}
```

Demo Program for Displaying the numbers of users currently login (who are in on-line).

```
import java.util.*;
```

```
public class userCount extends HttpServlet
{
    public void doGET(--)-
    {
        PrintWriter out = resp.getWriter();
        ServletContext context = getServletContext();
        Integer count = (Integer) context.getAttribute("userCount");
        HttpSession session = req.getSession();
        if(session.isNew())
        {
            if(count==null)
            {
                count=1;
            }
            else
            {
                count++;
            }
            context.setAttribute("userCount", count);
            → we can keep outside of if also.
            out.println("The no. of userCount Value", +count);
        }
    }
}
```

→ Demo program to display the total no. of request in each session.



open in new window
→ no session object will be created

import java.util.*;
Session

Public class RequestHitCount extends HttpServlet

{ public void doGet(..)

{

PrintWriter out = resp.getWriter();

HttpSession session = req.getSession();

Integer count = (Integer) session.getAttribute("requestCount");

if (count == null)

{

count = 1;

}

else

{

count++;

}

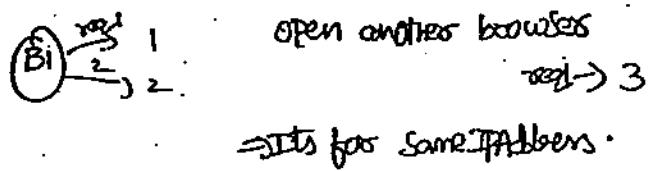
session.setAttribute("requestCount", count);

out.println("The no. of request in current session is", + count);

}

}

Demo program to Display no. of Requests from each IP address.

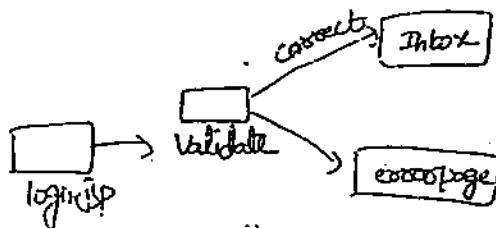


```
import java.util.*;
public class IRequestCount extends HttpServlet
{
    public void doGet(HttpServletRequest req)
    {
        PrintWriter out = resp.getWriter();
        ServletContext context = getServletContext();
        String IPAddress = req.getRemoteAddr();
        Integer count = (Integer) context.getAttribute(IPAddress);
        if (count == null)
        {
            count = 1;
        }
        else
        {
            count++;
        }
        context.setAttribute("IPAddress", count);
        out.println("The no. of request from " + IPAddress + " is " + count);
    }
}
```

Request Dispatcher

(57)

- ① Describe RequestDispatcher mechanism.
- ② write ServletCode to Create Request Dispatcher.
- ③ write ServletCode to Forward or Include TargetResource.
- ④ Identify & Describe the Attributes Added by web-container while Forwarding & Including.
- It is not recommended to Define entire functionality in a single Component. It has Several Serious Disadvantages.
 - ① without affecting Remaining Components we can't modify any component. Hence enhancement will become very complex & maintainability will be down.
 - ② It doesn't promote Re-usability.



→ we can resolve these problems by maintaining a separate component for each task.

→ The main advantages of this approach are without affecting remaining components we can modify any component. Hence enhancement will become very easy & improves maintainability.

→ It promotes Re-usability.

Ex: wherever validation is required, we can use the same validate servlet with out re-writing.

→ If entire functionality is distributed across several components, these components have to communicate with each other to provide response to end users. We can achieve this communication by using Request Dispatcher.

Servlet Code for getting Request Dispatcher

→ we can get Request Dispatcher either by using ServletRequest object or. Servlet Context Object.

① By Servlet Request Object:

→ Servlet Request Interface defines the following method for this.

public RequestDispatcher getRequestDispatcher (String TargetResource);

The Target Resource can be specified either by Absolute path or Relative path.

Ex: RequestDispatcher rd = req.getRequestDispatcher("/test2"); (58)

RequestDispatcher rd = req.getRequestDispatcher("test2");

If the target Resource is not available we will get 404-Status code Saying Required Requested Resource is not available.

⑦ By Servlet Context Object:

Servlet Context Interface defines the following 2 methods for getting RequestDispatcher Object.

(a) public RequestDispatcher getRequestDispatcher(String TargetResource)

→ The target Resource Should be specified by only Absolute Path

→ If we are using Relative path i.e if the path not Starts with "/" we will get Runtime Exception Saying Illegal Argument Exception.

Ex: RequestDispatcher rd = context.getRequestDispatcher("/test2");

RequestDispatcher rd = context.getRequestDispatcher("test2"), Error

If the target Resource is not available we will get 404 Status code Saying Requested Resource is not available.

(b) public RequestDispatcher getNamedDispatcher(String Servlet-name)

→ The Argument represents the Value Associated with the <Servlet-name> in web.xml.

→ If the specified Servlet is not available then we will get "null". On that null if we are trying to call forward we will get NullPointerException.

Ex: RD rd = context.getNamedDispatcher("First-Context");

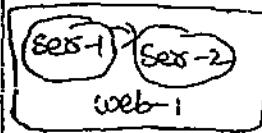
Difference b/w getting Request Dispatcher by
ServletRequest & by Servlet Context.

RD from ServletRequest

- 1) RD `rd=req.getRequestDispatcher("/test2");`

→ we can specify the target Resource either by Absolute or Relative Path.

- 2) we can't get Foreign Request Dispatcher by Request Object



- 3) we can't get RD by using `Servlet-name>`

methods of Request Dispatcher

- 1) Request Dispatcher defines the following 2 methods-

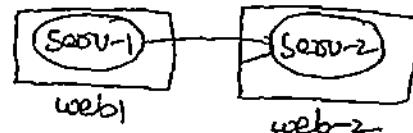
- ① `public void forward (SR req, SResp resp)` throws SE, IOException
② `public void include (SR req, SResp resp)` throws SE, IOException

RD from Servlet Context

- 1) RD `rd=context.getRequestDispatcher("/test2");`

→ The target Resource Path can be Specified only by Absolute Path. Otherwise we will get Illegal Argument Exception.

- 2) we can get foreign RD by Context object.

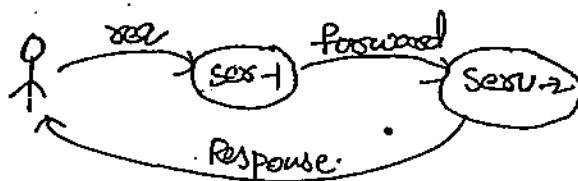


- 3) we can get RD by `<Servlet-name>` for this Servlet Context Defined `getNamedDispatcher();`

Forward mechanism

(59)

- If the First Servlet is responsible for some Preliminary Policies & 2nd Servlet is Responsible to provide Required Response then we should go for Forward mechanism.



- Just before forwarding, The Request Response Object will be created automatically by web-container. Hence if any Response added by the first Servlet won't be delivered to the End user.

- Second Servlet has Complete Control on the Response Object
It can change the Response Headers also.



- The same Request Object will be forwarded to the 2nd Servlet
Hence Information Sharing btw the Components in the form of Request-Scope Attributes.

~~Notes~~

- Once the Response Committed we are not allowed to call forward mechanism. Violation leads to Illegal State Exception.

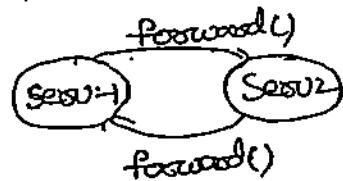
~~ex:~~ Pw.out = resp.getWriter();

out.println("Hello");

out.println("flush"); committed → Ready to send the Response Object.

rd.forward(req, resp); Error → Illegal State Exception.

→ Recursive forward call raises Stack Overflow Errors.



→ After forward call the control comes back for execution

of Remaining Starts. In those starts if we are trying to add any response those starts will be ignored by the web-container.

Example - 1

public class FS extends HttpServlet

{ P v doGET(--) -->

{ Pw.out = resp.getWriter();

out.println("FirstServlet");

RequestDispatcher rd = req.getRequestDispatcher("/test");

rd.forward(req, resp);

System.out.println("0/b"); [or] out.println("0/b"); Exception

}

Public class SecondServlet extends HttpServlet.

(6)

```
{ P v doGET( -- ) ---
```

```
{ out.println("SecondServlet");
```

```
}
```

⇒ whenever we are sending the Request to 2nd Servlet we will get ArithmeticException has the Response.

Example-2

FirstServlet.java

Public class FS extends HttpServlet

```
{ P v doGET( -- ) ---
```

```
{ PW out = res.getWriter();
```

```
out.println("This is FirstServlet");
```

RequestDispatcher rd = req.getRequestDispatcher("/test2");

```
rd.forward(req, res);
```

out.println("This is firstServlet again"); Ignored by web container

S.out.println("The control comes back again to FS");

SecondServlet.java

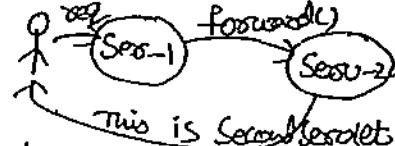
Public class SS extends HttpServlet

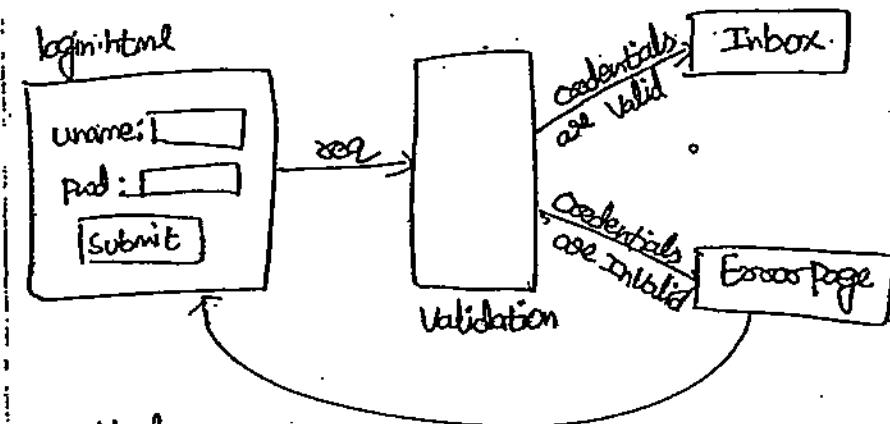
```
{ P v doGET( -- ) ---
```

```
{ PW out = res.getWriter();
```

```
out.println("This is SecondServlet");
```

```
}
```





login.html

```

<html> <body> This is forward Demo >
<form action = "scwcd /test1">
    Enter Name : <input type = text name = uname>
    Enter Pwd : <input type = text name = pwd>
    <input type = submit>
</form> </body> </html>

```

validation.java

```

public class validation extends HttpServlet
{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
    {
        PrintWriter out = response.getWriter();
        String name = request.getParameter("uname");
        String pwd = request.getParameter("pwd");
        if(name.equals("durga") && pwd.equals("scwcd"))
        {
            RequestDispatcher rd = getServletContext().getRD("/inbox.jsp");
            rd.forward(request, response);
        }
    }
}

```

(6)

```
else
{
    RD rd = ser.getRD("/error.jsp");
    rd.forward(req, resp);
}
}
```

Inbox.jsp

This is your Inbox.
Check your mails.

Error.jsp

This is Error page. Credentials
are invalid. Please login again
[Login](/scwcp/login.html)

Servlet

Attributes Added by web-Container while Forwarding
the Request:

while Forwarding the request from one Servlet to another Servlets

the Container adds some attributes in the Request Scope.

These attributes provides information to second Servlet about
original Request.

Attribute Name

Description

Equivalent Servlet method

① javax.servlet.forward.
request_uri

The URI of the
original Request

getRequestURI()

[/webapp/test/test/diego/
software]

② javax.servlet.forward.
context-path

The context path of
the application
[/mihir]

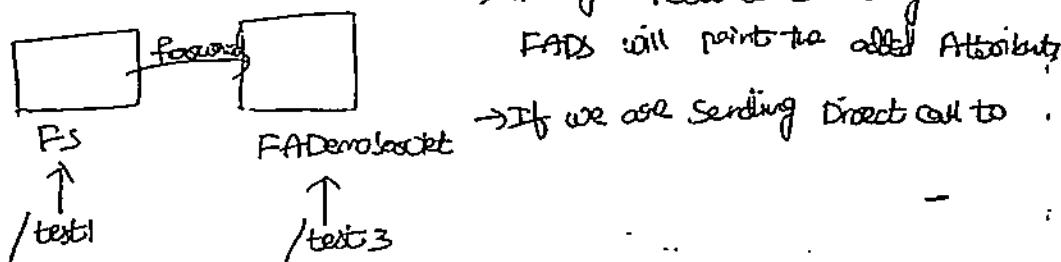
getContextPath()

3) javax.servlet.Forward
ServletPath The ServletPath of the
forwarding Servlet getServletPath();
[/test/test]

4) javax.servlet.Forward.
pathInfo Returns the extra Path
Info associated with
Request URI . i.e this
is the String which follows
Servlet path but precedes
QueryString.
[durga/software]

5) javax.servlet.Forward.
queryString The QueryString Associated
with the Request. getQueryString();

Demo Program



class
Public ^ FS extends HttpServlet

{

P. v doGET(--) -- context.getRD("test3");
{

RequestDispatcher rd = req.getRequestDispatcher("test3");

rd.forward (req, resp);

}

Forward Attribute Demo.java

(62)

```
public class FAD extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println(" Forward Request Attribute");
        Enumeration e = request.getAttributeNames();
        while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            Object value = request.getAttribute(name);
            out.println(name + " --- " + value);
        }
    }
}
```

If we are sending the Request to the forward DemoServlet through forward call then we will get all the Attributes which are added by web Container.

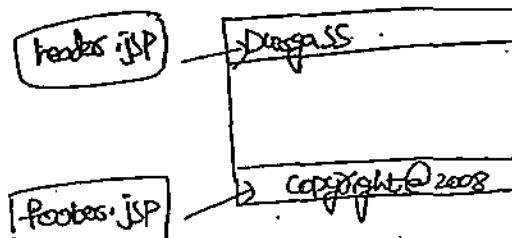
http://localhost:8080/scard/test1?uname=deepa&path=PR.

If we are sending the Request Directly to the Forward Attribute DemoServlet, then we won't get any Attribute.

If we are performing forward by the RequestDispatcher which is obtained by `getNamedDispatcher()` then WebContainer won't add any Attributes in the request Scope.

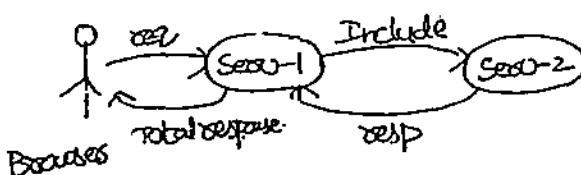
- RD rd = context.getNamedDispatcher("FDServlet");
- rd.forward(req, resp);
- In this web-container won't add any Attributes.
- After Forwarding the request if we are trying to forward in the same Servlet we will get IllegalStateException.

Include Mechanism



→ If we want DSS, Copyright in 2 JSP pages make it separate as header, footer wherever useful Just Include that JSPs.

→

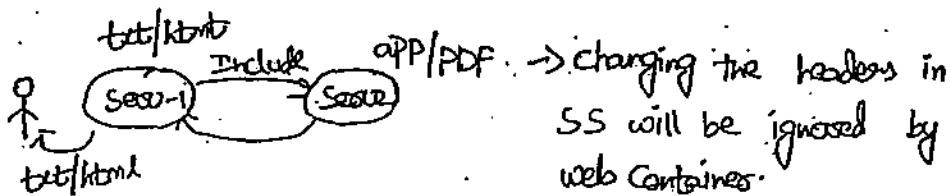


- This mechanism is helpful to include the Response of other Resources in the current Response.
- This Inclusion mechanism is best suitable for Including Banners Information like copyright, Logo etc.
- The Servlet who got the request Initially is responsible to provide Response.
- After committing the Response we can perform include() call

(62)

but we can't perform forward call.

- In the include, the second servlet doesn't have complete control on the response object. It is not allowed to change the response headers. If it is trying to perform any changes these will be ignored by the web-container.



FirstServlet.java

public class FS extends HttpServlet

```
    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
        PrintWriter out = resp.getWriter();  
        out.println("This is firstServlet");  
        RequestDispatcher rd = req.getRequestDispatcher("/test2");  
        rd.include(req, resp);  
        out.println("This is firstServlet Again");  
    }  
}
```

SecondServlet.java

```
    PrintWriter out = resp.getWriter();  
    out.println("This is ss");
```

→ whenever we are sending the request to the host server
the following is the O/P

This is FS

This is SS

This is FS again.

→ Attributes added by web container while include call-in
RequestScope.

① javax.servlet.include.request-URI

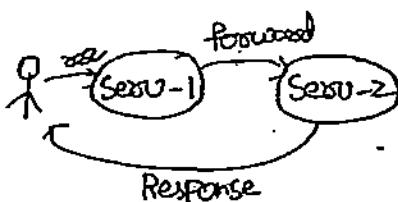
- ② " " Context-Path
- ③ " " Servlet-Path
- ④ " " Path-Info
- ⑤ " " Query-String

→ If we are getting RequestDispatcher by using getNamedDispatcher
then web container won't add any attributes in the
RequestScope.

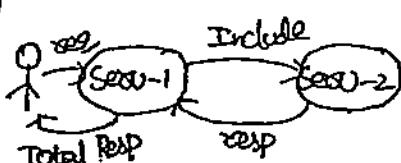
Difference b/w forward & include

(64)

Forward



Include



→ Once a Servlet forwards the Request to another Servlet then the forwarded Servlet (S-2) is completely responsible to provide required Response.

→ In the case of Include the Including Servlet (S-1) is responsible to provide the required Response.

Q) While performing forward() the response Object will be cleared by the web-container.

Q) In the Include web container won't clear the Response Object.

Q) After committing the Response we are not allowed to perform forward(). Otherwise - Illegal State Exception.

Q) After committing the Response we can perform Include.

Q) Within any Servlet we can call forward() only once & mostly as the last stat.

Q) We can call include() any no. of times in the Servlet.

Q) In the case of forward, the forwarded Servlet having the

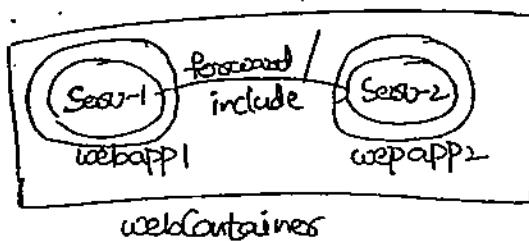
In the case of Include, the Included Servlet (S-2) doesn't have complete control on the

complete control on the response Response Object. It is not object. It is allowed to change allowed to change the the response headers.

Response headers.

- ⑤ Forward mechanism, we can use frequently in Servlets, bcz it is associated with processing mostly its associated with View Part.
- ⑥ Include mechanism, we can use frequently in JSP's, bcz mostly it is associated with View Part.

Foreign Request Dispatcher



```
ServletContext context = getServletContext();
```

```
ServletContext foreignContext = context.getServletContext("webapp2");
```

```
RequestDispatcher foreignRD = foreignContext.getRequestDispatcher("test2");
```

```
foreignRD.forward(req, resp);
```

→ Request Dispatcher mechanism will work within the same

web containers. Hence both web application should be deployed

in the Same Server.

(65)

- ⇒ most of the web servers won't provide support for cross Context Communication due to security reasons. In this case we will get "Null Pointer Exception".

Q) what is the purpose of RequestDispatcher?

Q) How many possible ways are there for getting RD? 3

Q) what is the diff b/w obtaining RD from Servlet Request & Servlet Context Object.

Q) what is the diff b/w forward & include.

Q) what is the diff b/w forward & sendRedirect.

Q) what is foreign Request Dispatcher & how we can get it?

Q) what are the attributes added by the web container while forwarding & including the request.

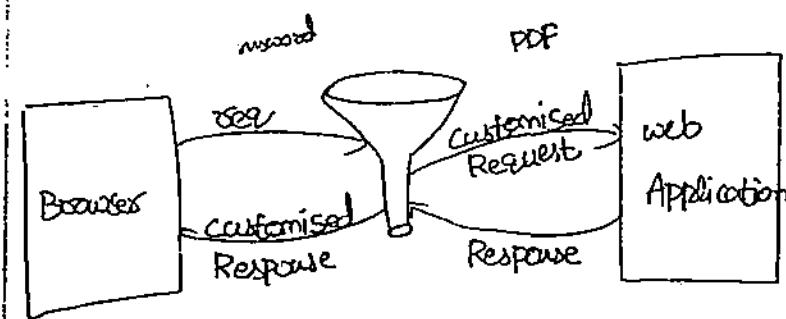
Q) what is the purpose of these attributes & explain the meaning.

~~108~~

Filters 3 to 4.

- (1) Describe the web Containers Request Processing Model
- (2) write & configure a filter.
- (3) Create a Request & Response Wrappers for the given Design Problem.
- (4) describe how to apply a filters or wrappers.

→ Filters are used for Pre-Processing of Request & Post processing of Response before they reach target Resource in the web application.



→ The most common application areas of filters are Logging, Security checks, Altering Request Information, Compressing the response, Encryption of Response, Authentication etc..

→ Filters concept has Introduced in Servlet 2.3 Version.

Filters API

(6)

→ Filters API Contains the following 3 Interfaces.

① Filter

② FilterConfig

③ FilterChain

Filter

→ Every Filter in Java has to implement Filter Interface either directly or indirectly.

→ Filter Interface Defines the following 3 methods.

① init()

 public void init(FilterConfig config) throws SE

 → This method will be executed only once to perform Initialization Activities.

② destroy();

 public void destroy();

 → This method will be executed only once to perform clean up activities just before taking the filter from out of service.

③ doFilter()

 public void doFilter(ServletRequest req, ServletResponse resp,
 FilterChain fchain) throws
 SE, IOException

- Entire filtering logic we have to define in this method only.
- This method will be executed for every Request.
- By using FilterChain Object we can forward the request to the next level. (It may be Servlet or another Filter).

FilterConfig

- For every Filter, web container creates a filter Config object and hand over it to the filter by init().
- Filter can get its configuration information by using this Filter config object.
- Filter Config Interface defines the following methods.
 - (1) public String getFilterName();
 - Returns the logical name of the Filter configured in web.xml by using <filter-name>
 - (2) public String getInitParameters(String name);
 - (3) public Enumeration getInitParameterNames();
 - (4) public ServletContext getServletContext();

Filter Chain

67

- Filter chain Object can be used for forwarding the request to the next level. (it may be filter or Servlet.)
 - Filterchain Interface contains the following method.

```
⑧ public void doFilter(ServletRequest, ServletResponse);
```

Demo program For Filter

o Demofilters.java

public class DemoFilter implements Filter

5

Public void init(FitterConfig config) throws SE, ~~DE~~

5

3

```
public void doFilters(ServletRequest request, ServletResponse response, FilterChain filterChain)
```

~~theoxos~~ SE, DoE

3

```
pw.out = esp.getWriter();
```

open ("This line is added by Demofitter before processing
the request");

fc.doFilters(req, resp);

O-Per ("This line is added by DemoFilter after processing
the request");

3

public void destroy()

۳۹

TargetServlet.java

```
public class TargetServlet extends HttpServlet  
{  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
    {  
        PrintWriter out = response.getWriter();  
        out.println("This is TargetServlet");  
    }  
}
```

web.xml

```
<web-app>
```

```
    <servlet>
```

```
        <servlet-name>TargetServlet </s-n>
```

```
        <servlet-class> " </s-c>
```

```
    </servlet>
```

```
    <s-m>
```

→ Should be
Same name.

```
        <s-n> TargetServlet </s-n>
```

```
        <url-pattern>/{test} </url-pattern>
```

```
    </s-m>
```

```
    <filter>
```

```
        <filter-name>DemoFilter </filter-name>
```

```
        <filter-class> DemoFilter </filter-class>
```

```
    </filter>
```

```
    <filter-mapping>
```

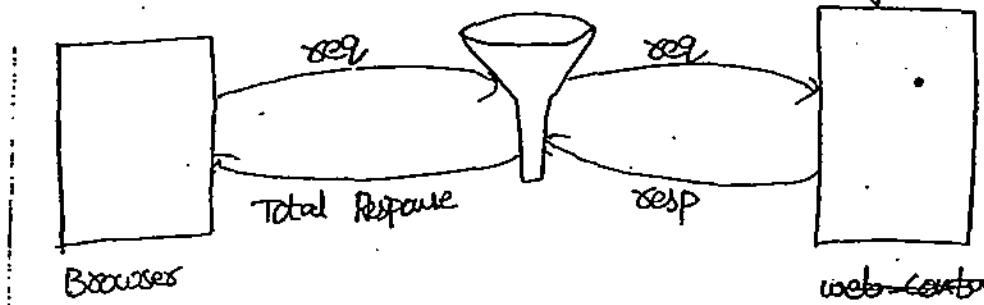
```
        <filter-name> DemoFilter </filter-name>
```

```
        <url-pattern>/{test} </url-pattern>
```

```
    </filter-mapping>
```

Analysis

(68) Target Servlet



- whenever we are sending Request to the Servlet, web container checks is there any Filter configured for this Servlet or not.
- If any Filter is configured, web container forwards that request to the Filter instead of Servlet.
- After completing Filtering logic, Filter forwards the request to the Target Servlet.
- After processing the request by Servlet, the response will be forwarded to Filter instead of Browser.
- After Executing Filtering logic, Filter forwards Total response to the End user.

Configuring Filter in web.xml

- we can configure Filter by using <filters>.

<web-app>

<filters>

<filter-name>
<filter-class>

~~<filter>~~ } mandatory.

```
<init-param>
  <param-name> } 0
  <param-value> } to
                many
</init-param>
```

```
</filters>
```

```
</web-app>
```

→ we can map Filters either for a Particular URL-Pattern
or to a particular Servlet or to the whole webApplication.

Filters-mapping to a particular URL-pattern

```
<filters-mapping>
  <filter-name> DemoFilter </filter-name>
  <url-pattern> /test </url-pattern>
```

```
<filters-mapping>
```

→ If the request is coming with the specified URL-pattern
automatically this Filter will execute.

Filters-mapping to a particular Servlet

```
<filters-mapping>
  <filter-name> DemoFilter </filter-name>
  <Servlet-name> TargetServlet </Servlet-name>
```

```
<filters-mapping>
```

Filter-mapping for entire web application.

(6)

<filter-mapping>

<filter-name> Demo Filter </filter-name>

<url-pattern> * </url-pattern>

→ For any Request to the web-application whether it is for Servlet or JSP, this filter will be executed.

→ mapping a Filter to whole web-application is possible from Servlet 2.5 version onwards.

<dispatchers>

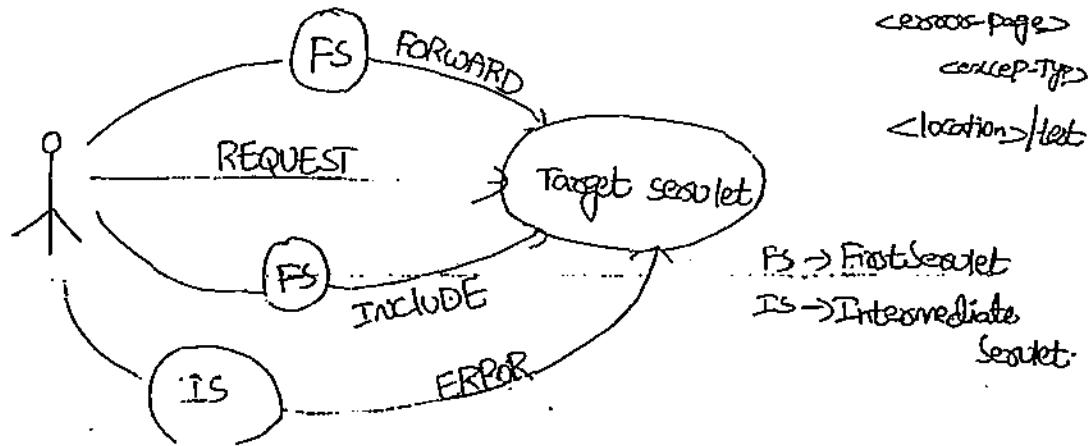
→ A servlet can get the Request in one of the following possibility

① A Request Directly from the browser. (REQUEST)

② By RequestDispatcher Forward Call (FORWARD).

③ By RequestDispatcher Include call (INCLUDE).

④ By ErrorPage call (ERROR)



- In Servlet 2.3 V Filter's concept is applicable only for Direct End-user Request. It is not applicable for Request Dispatcher forward, Include & Error page calls.
- But in the Servlet 2.4 V, Sun people introduced <dispatcher> to extend filter concept for the remaining cases also.
- For which type of Request the filter will be executed is decided by <dispatchers>. The allowed values for the Dispatcher Tag are

① REQUEST

- Filter will execute only for Direct end user Request.
- This is the Default value.

② FORWARD

- Filter will execute for Request Dispatcher's forward call.

③ INCLUDE

- Filter will execute for RD Include call.

④ ERROR

- Filter will execute for error page calls.

<error-page>

<exception-type>

<location> /test 2 <locations>

<error-page>

(1) If we want to execute Filter for Direct End-User Request & Request Dispatchers Forward call we have to configure filter Mapping as follows. (70)

```
<filter-mapping>
  <filter-name> DemoFilter </filter-name>
  <url-pattern> /test </url-pattern>
  <dispatcher> REQUEST </dispatcher>
  <dispatcher> FORWARD </dispatcher>
```

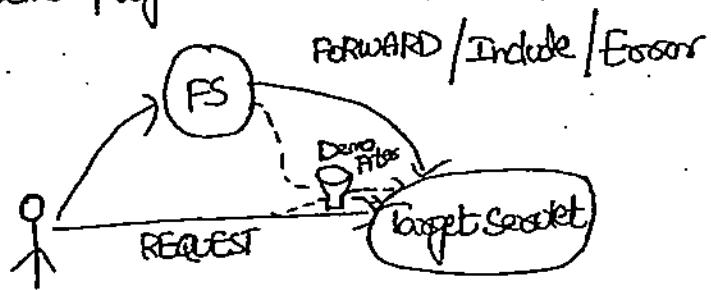
→ In this case filter ^{won't} be executed for the RD Include calls & Error pages.

(2)

```
<filter-mapping>
  <filter-name>
  <url-pattern>
  <dispatcher> INCLUDE </dispatcher>
</filter-mapping>
```

→ In this case filter will execute only for RD Include call.

Demo program two components



TargetServlet.java

```
pw.out = resp.getWriter();
open("This is TargetServlet");
```

FirstServlet.java

```
public class FirstServlet extends HttpServlet
```

```
{
```

```
    public void doGET(--)--
```

```
{
```

```
    pw.out = resp.getWriter();
```

```
    s.o.println("1/1");
```

```
    RD rd = seq.getRD("/test2");
```

```
    rd.forward(req, resp);
```

```
    rd.include(req, resp);
```

```
}
```

DemoFilter.java

<web-app>

(7)

FS
↑
test1

TargetServlet
↑
test2

<filters>

<filter-name> DemoFilter

<filter-class> "

</filters>

<filter-mapping>

<filter-name> DemoFilter

< servlet-name > TargetServlet

< dispatcher > Error

< dispatcher > REQUEST

<filter-mapping>

<error-page>

<exception-type> java.lang.ArithmaticException

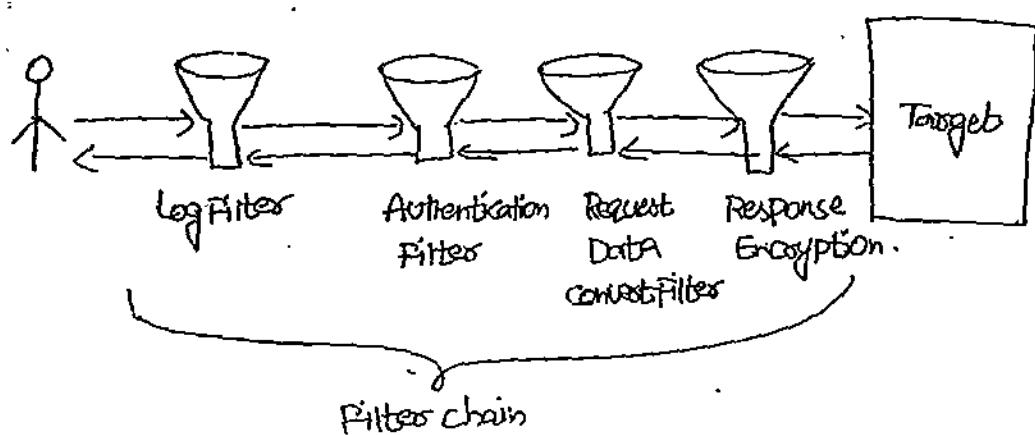
<location> /test2 <location>

<error-page>

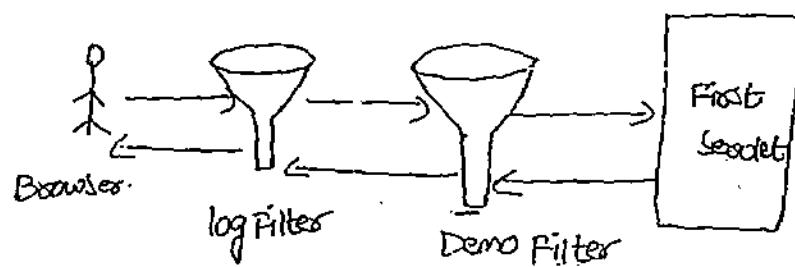
</web-app>

Filter Chain

→ we can configure more than one filter for a Target Resource and all these filters will be executed one by one and forms filterchain.



Demo Program for the filter chain



LogFilter.java

Public class LogFilter implements Filter

{

 FilterConfig conf;

 Public void init(FilterConfig conf) throws ServletException
 {
 this.conf = conf;
 }

```
public void doFilter(ServletRequest req, ServletResponse resp, FilterChain fc) throws  
    SE, IOException  
{  
    PrintWriter out = resp.getWriter();  
    open("This line is added by LogFilter before processing the  
    request");  
    HSR seq1 = (HSR)req;  
    ServletContext context = config.getServletContext();  
    context.log("A Request is coming from " + req.getRemoteAddr());  
    context.log("Request info to the log file as separate  
    lines");  
    for (" + seq1.getRemoteURL() + " at " +  
        seq1.getDate());  
    fc.doFilter(req, resp);  
    open("This line is added by LogFilter after Processing the  
    Request");  
}
```

```
public void destroy()
```

```
{  
    conf = null;  
}
```

DemoFilter.java

FristServlet.java

web.xml

<web-app>

c:\tomcats 6.0\log\localhost.

2009-04-07
tot

<filter>

<filter-name> LF </filter-name>

<filter-class> LF </filter-class>

</filter>

<filter>

<filter-name> DF </filter-name>

<filter-class> DF </filter-class>

</filter>

<filter-mapping>

<filter-name> Log Filter

<url-pattern> /test1 [or] <servlet-name> FS </servlet-name>

① DF execution

② LF

</filter-mapping>

<filter-mapping>

<filter-name> DemoFilter

<url-pattern> / test1

<filter-mapping> O/P

FS

This line added by LF before Proc Req

↑

" " " " DF " " "

/test1

This is TargetServlet

</web-app>

This line added by DF After Proc Req

" " " " LF " "

Containness Rule for Ordering of Filters in Filterchain. (73)

Identify all the filters which are configure according to url-pattern. & execute all these filters from TOP to Bottom.

Identify all Filters which are configure according to <Servlet-name> & execute all these filters from TOP to Bottom.

~~Step by step~~

Consider the following web.xml Declarations.

<filter-mapping>

<filter-name> Filter1 </filter-name>
<url-pattern> /Recipes/* </url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name> Filter2 </filter-name>
<Servlet-name> /Recipes /toplist.do </Servlet-name>

</filter-mapping>

<filter-mapping>

<filter-name> Filter3 </filter-name>
<url-pattern> /Recipes /Add/* </url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name> Filter4 </filter-name>

<Servlet-name> /Recipes /Modify /modify.do </Servlet-name>

<filters-mapping>

<f-n>filters </f-n>

<url-pattern> /* </url-pattern>

</f-n>

Request URI

Order of Filters Execution.

① /Recipes/TopRepost.do

Filter 1 → Filter 5

② /Recipes/TopList.do

① → ⑤ → ②
Dual-Pattern ↓ T-B
@seadet-name ↓ T-B

③ /TopList.do

⑤

④ /Recipes/modify/modify.do

① → ⑤ → ④

⑤ /Recipes/Add/AddRecipes.do

① → ③ → ⑤

→ Filters & Filter mapping Tags are the Direct child Tags of web-app
and we can take these tags anywhere with in <web-app>

doFilters(SR req, SR resp, fc fc)
call back method {
doFilters(SR req, SR resp)
Inline method }

Difference b/w FilterChain doFilter() & Filter's doFilter method.

Filter's doFilter()

FilterChain's doFilter()

① Public void doFilter(SR req,
SR resp; Fc fc) throws IOException

① Public void doFilter(SR req,
SR resp)

② For Defining Filtering Logic

② For Defining forwarding
Request to the next level
(it may be Servlet or Filter)

③ It is call back method bcz web-containers calls these method automatically.

③ It is inline method bcz programmer has to call these method explicitly. (74)

Wrappers

→ sometimes it is required to alter Request & Response Information in the filters. we can achieve this by using wrapper classes.

Exam-1 - within the filter we have to convert end users Resume from word format to PDF format.

Exam-2 - In the filter we have to compress the response and that compressed response we can send to the Browser so that we can reduce download times.

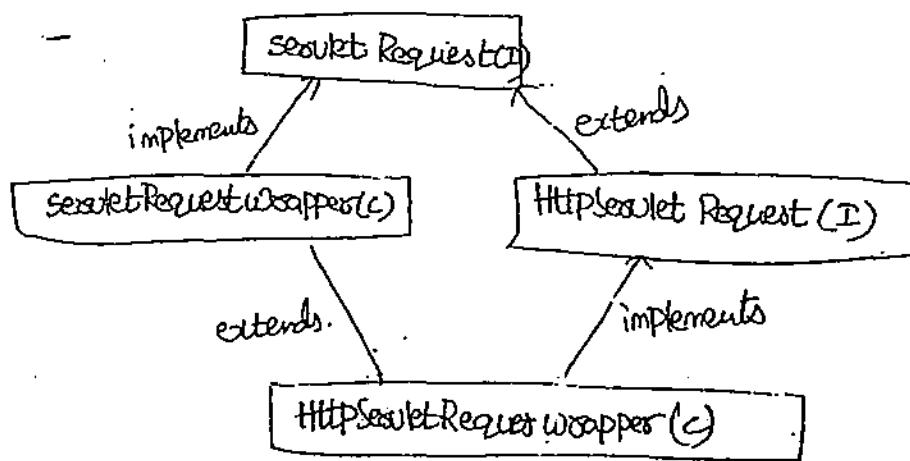
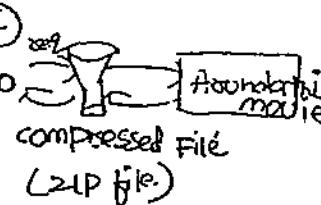


⇒ There are 2 types of wrapper classes.

① Request Wrappers — To Alter Request Info

② ServletRequestWrapper

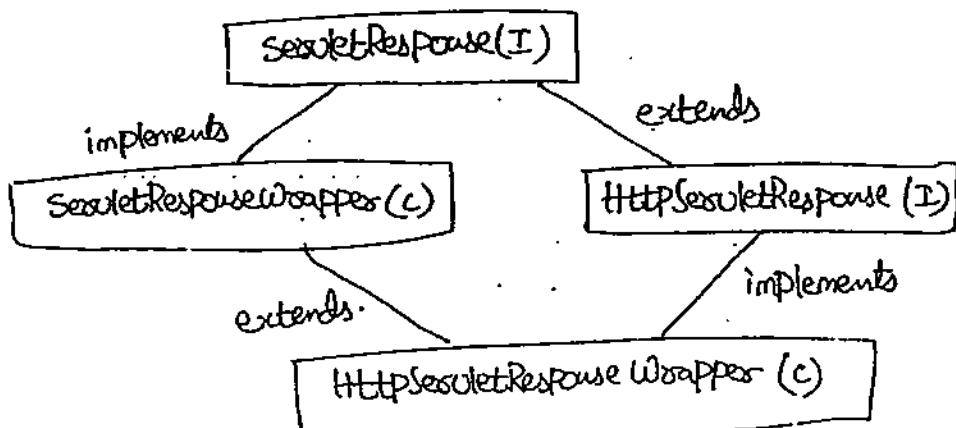
③ HttpServletRequestWrapper



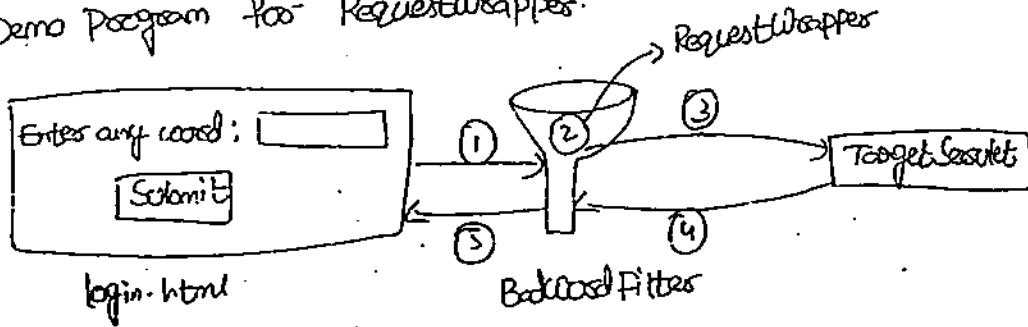
② Response Wrapper — To Alter Response Information.

(a) `ServletResponseWrapper`

(b) `HttpServletResponseWrapper`.



→ Demo Program for RequestWrappers.



Browser

- End user Entered the word & click the submit button.
- Filter creates a customized Request object by using `Wrapper class`.
- Filter forwards that customized request object to the Target `Servlet` instead of original Request
- with in the `Servlet` if we are applying any operation on the Request object our own customized behaviour will be reflected.

- TargetSelect Prepares the Response & forwards to Filter
- Filter forwards that Response intern to the End user. (76)

AB+ve Blood Group
99491 78512 - feoz than.

~~login.html~~

```
<html>
<form action = "/wrapper/test1">
Enter your word : <input type= text name=word>
<input type= submit >
```

(76)

BadwordFilter.java

public class BWF implements Filter

```
{  
    public void init(FilterConfig config) throws ServletException  
    {  
        //  
    }
```

```
    public void doFilter(ServletRequest request, ServletResponse response)  
    {  
        //  
    }
```

Customized Request req = new customized Request (HttpServlet
Request) req);

```
    fc.doFilter(req, resp);
```

```
} -
```

```
    public void destroy()  
    {  
        //  
    }
```

Customized Request.java

```
public class CR extends HttpServletRequestWrapper  
{  
    public CR (HttpServletRequest req)  
    {  
        super(req);  
    }  
  
    public String getParameter(String word)  
    {  
        String word1 = super.getParameter("word");  
        if(word1.equals("Java") || word1.equals("SCJP"))  
            return "Sleep";  
        else  
            return word;  
    }  
}
```

RequestServlet.java

```
P class TS extends HttpServlet  
{  
    P() {super();}  
  
    PW out = resp.getWriter();  
    String word = req.getParameter("word");  
    out.println("you entered word is " + word);  
}
```

web.xml

(77)

<w-a>

TargetServlet

/test

Badword Filter

/test

⇒ For wrappers not required to perform any configuration in web.xml.

</w-a>

Note

(web-6)

- ① Filter Object is created by the web Container, for this it is always calling Public no-arg constructor. Hence every filter class should compulsory contain Public no argument constructor
- ② usage of filters is nothing but following Intercepting Filter Design Pattern. Hence it is recommended to use filters concept.
- ③ usage of wrappers is nothing but following Decorator Design pattern.

UNIT-4

8 to 10 Ques

Session Management.

(1) Given a Scenario, Describe the Session API

(2) Explain the Process of creating a Session object.

(3) what are Various Different mechanisms to Invalidate a Session.

→ The basic limitation of HTTP is, it is stateless Protocol i.e. it is unable to remember client state across multiple Request.

→ Every Request to the Server is considered as a new Request. Hence some mechanism is required to remember client information across multiple Request. This mechanism is nothing but Session Management Mechanism.

→ The following are various Session Management mechanisms.

(1) Session API

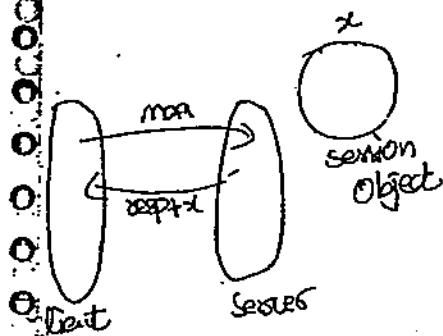
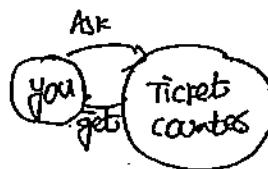
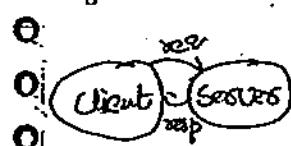
(2) Cookies

(3) url-Rewriting

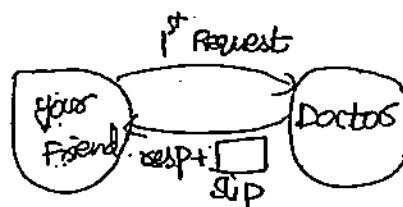
(4) hidden Form Field — Not official mechanism from son, it is just programmers trick to remembers client Information.

Q4) Creation of Session Object

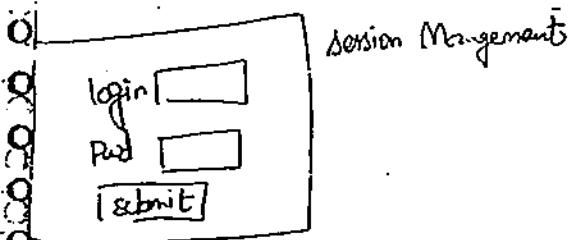
(78)



At Server side , Session Object

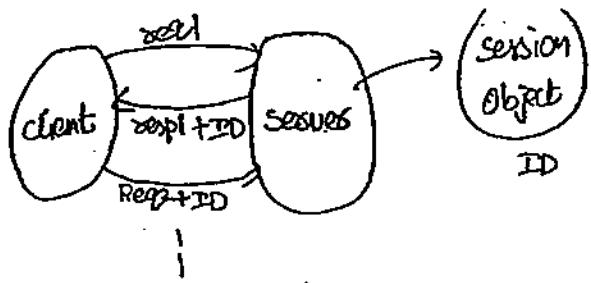


- Some after 2 days along with your slip
- This is Client Side Session object.
- This type is cookies.



whenever client sends the 1st Request, if the server wants to remember clients information then it will create a session object & sends the corresponding session ID to the Browser as the part of 1st Resp.

Browser sends that session ID as the part of consecutive request, By accessing that session ID & the corresponding Session Obj, servers able to remember client info across multiple Request.



⇒ HTTP Serviet Request defines the following methods for the creation of Session Object.

① public HttpSession getSession();

Ex: HttpSession Session = req.getSession();

⇒ If the request associated with Session Object already then existing session object will be returned.

⇒ If the request doesn't associated with any session, this method will create a new session Object & returns it.

⇒ There is a guarantee that this method will always return session Object. It may be newly created or existing object.

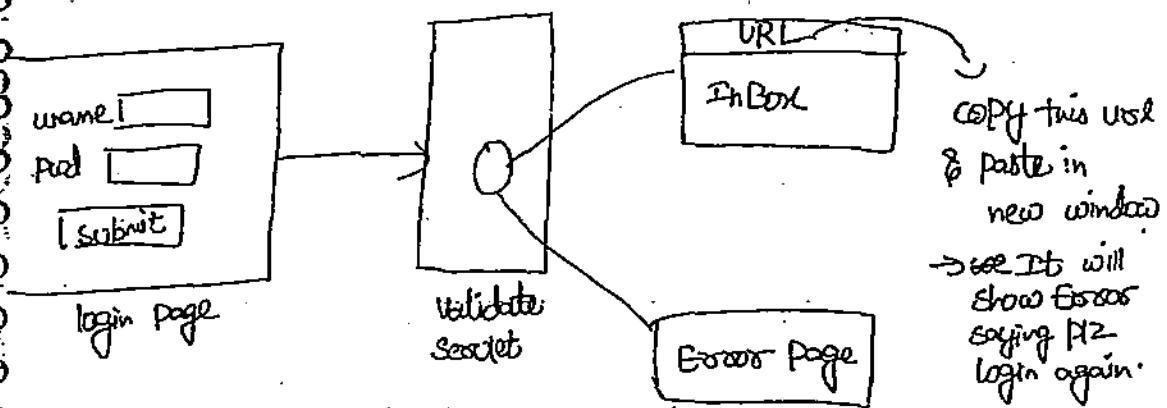
② public HttpSession getSession(Boolean b);

→ If the Argument is true then this method simple acts as getSession();

→ If the Argument is false then this method first checks whether the request associated with any session or not. If the request already associated with session then this method returns Existing session object.

- If the request doesn't associate with any session then
- this method returns "NULL" without creating any new Session Object.

(79)



- To Access Inbox Page compulsorily the request should contain session. If the request doesn't associated with any session we can't create new session Object & we have to forward that request to the login page. In this case we have to use getSession(Boolean b).

which of the following 2 statements are equal.

- ① session = `req.getSession();`
- ② session = `req.getSession(false);`
- ③ session = `req.getSession(true);`
- ④ session = `context.getSession(true);`

InValidating a Session.

→ we can invalidate a session by using the following 2 ways.

① By invalidate()

② By Time Out Mechanism.

invalidate()

→ HttpSession Contains this method to invalidate a Session -

public void invalidate()

e.g. whenever we are clicking **logout** intensionally this
method will be executed.

session.invalidate();

TimeOut Mechanism.

If we are not performing any operation on the Session object for a pre-defined amount of time then automatically session will be expired. This pre-defined amount of time is called Session-TimeOut.

→ we can configure Session Timeout either Server Level [00]
web-Application Level or a particular Session Object Level.

- ① Automatic Support from the web servers
- most of the web servers provide default support for session timeout. mostly it is 30 minutes.
- we are allowed to change this server-level session time out value based on our requirement.

② Configuring SessionTimeOut at Web Application Level.

- we can configure session timeout for entire web-application in web.xml as follows.

```
<web-app>
  <session-config>
    <session-timeout> 10 </session-timeout>
  </session-config>
  ...
</web-app>
```

- `<session-config>` is the direct child tag of `web-app` & we can configure it place anywhere within `web-app`.

- the unit to the `Session-timeout` value is minutes

- 0 or -ve value indicates that session never expires.

- this `Session-timeout` is applicable for all the sessions which are created in that web-application.

③ Individual setting SessionTimeOut for a particular Session Object.

- we can set `SessionTimeOut` for a particular session object by using the following method of `HttpSession`.

`Public void setMaxInactiveInterval (int Seconds)`

→ The Argument must be in Seconds.

→ -ve value indicates the session never expires

→ '0' indicates sessionExpires immediately.

→ This sessionTimeout is applicable only for a particular sessionObject on which this method has called.

Comparison b/w 2 sessionTimeout mechanisms-

Property

<session-timeout>

SetMaxInactiveInterval()

① Scope

It is applicable for all the sessions which are created in that web-application.

It is applicable only for a particular Session Object on which we called this method.

② units

minutes

seconds

③ 0-value

Session never Expires

Expires immediately.

④ -ve value

Session never Expires

Session never Expires.

(81)

Important methods of HttpSession

①) public boolean isNew()

- To check whether the SessionObject is newly created or not

②) public void invalidate()

- To expire a session forcefully

③) public void setMaxInactiveInterval (int Sec)

- To set Session-timeOut for a particular Session Object.

④) public int getMaxInactiveInterval();

- Returns Session-timeOut value in Seconds.

⑤) public String getId();

- Returns the Session-ID.

⑥) public long getCreationTime()

- Returns the time when the Session was created in milliseconds

since Jan 1st 1970.

1970 Jan 1st

↑ in millsec

If we are passing this long value to the Date constructor we will get exact date & time. may 1st 7:30 AM.

(Date d = new Date(l));

⑦) public long getLastAccessedTime();

- Returns the time when the client accessed recently the Session Object in milliseconds.

⑧) public ServletContext getServletContext();

- Returns the ServletContext Object to which this session belongs.

→ Once session expires, we are not allowed to call above methods. Violation leads to illegal state exception.

→ But this rule is not applicable for the following 3 methods.

- | | |
|-------------------------|-----------------------------------|
| ① getCreationTime() | HttpSession s = req.getSession(); |
| ② getLastAccessedTime() | s.invalidate(); |
| ③ getServletContext() | s.open(s.getCreateTime()); |

→ HttpSession defines the following methods to perform Attribute management in Session Scope.

- ① public void setAttribute(String name, Object value);
- ② public Object getAttribute(String name);
- ③ public EnumerationgetAttributeNames();
- ④ public void removeAttribute(String name);

~~109~~
Demo program for Session Management by using Session API.

login.html

(2)

```
<html>
<body>
<form action = "/session1/test1" >
<h1> Enter Session Information </h1>
    Enter Name : <input type = "text" name = "uname" >
    Enter Value : <input type = "text" name = "value" >
    <input type = "submit" >
</form>
<a href = "/session1/test2" > Session Information </a>
</body> </html>
```

Session Servlet1.java

public class extends HttpServlet

{

 @Override

{

 PrintWriter out = response.getWriter();

 HttpSession session = request.getSession();

 if (session.isNew())

{

 out.println("New session got created with Session ID: " +

 session.getId());

}

else

{

open("Existing session only using with SessionID" +
Session.getId());

}

String name = req.getParameter("uname");

String value = req.getParameter("value");

Session.setAttribute(name, value);

Session.setMaxInactiveInterval(120);

RD rd = req.getRequestDispatcher("login.html");

rd.include(req, resp);

}

}

SessionServlet.java

Public class SS extends HttpServlet

P v doGET(-->)

{

PW out = resp.getWriter();

HttpSession Session = req.getSession(false);

If (Session == null)

{

o.println("No session Information is available");

}

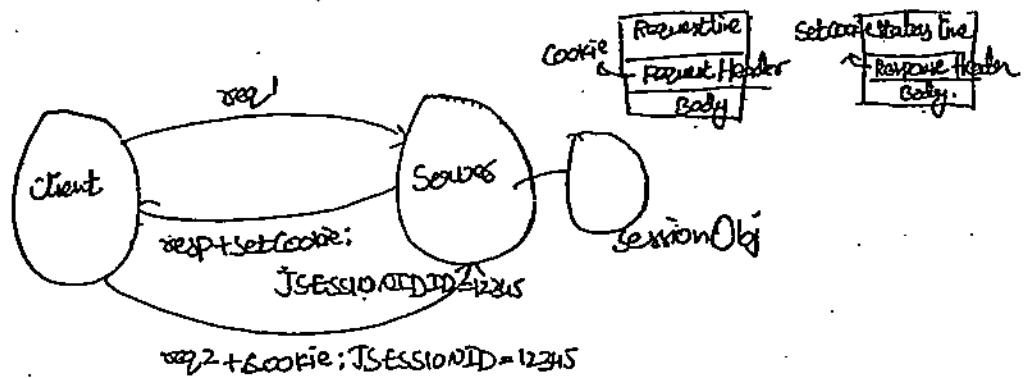
(83)

```
| else
{
    Enumeration e = hs.getAttributeNames();
    o.println("<table> <thead> <tr><td> SessionInformation Names
              </td> SessionInformation Values </td> </tr> </thead>
              <tbody> ");
    while (e.hasMoreElements())
    {
        String name =
        String value = (String) session.getAttribute(name);
        o.println(name + " --- " + value);
    }
    o.println("</tbody> ");
    long l1 = session.getCreationTime();
    long l2 = session.getLastAccessedTime();
    int l3 = session.getMaxInactiveInterval();
    o.println("<tr> <td> creation time is :" + new Date(l1) + "</td> ");
    o.println(" <td> " + new Date(l2) + " </td> ");
    o.println(" <td> maxInActiveInterval " + l3 + "</td> ");
}
```

web.xml

```
<!-->
<session-config> Not using
                  ↑ Defaults 1800 → 30 min
                  will be displayed
                  <session-timeout> 3 </s-to> /test1
                  </session-config>                                SS1
                                                               ↑
                                                               SS2
                                                               ↑
                  /test2 </testz>
```

How the SessionID tiechanging btw client & server



→ whenever Browser sends a Request, Server creates a Session Object.

If

→ The corresponding Session-ID Server will send as the part of Response for this Server uses SetCookie Response Header.

The Browser will receive that Session-ID & will send with every consecutive request. For this the browser uses cookie request Headers.

→ xahref = "Session / test3"

RequestHeader Script → for checking the SessionID.

Note: If the required session info is very less then creating a separate session object & maintaining that object at server side is not recommended bcz it impacts the performance of the system.

→ To resolve this we should go for cookie concept where required session info is maintained at client side & server is not responsible to maintain session info.

Cookie

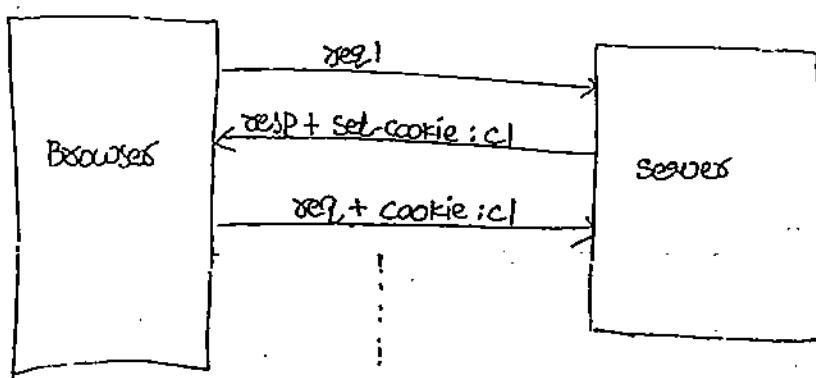
→ cookie is a small amount of information (key-value pairs) which is created by server & send to the browser.

The browser stores that cookies in the local file system & sends with every consecutive request to the server.

By accessing that cookie server able to remember client information.

→ Server sends cookies by using setCookie Response Headers & Browser sends those cookies by using cookie Request Header.

This is exactly similar to exchanging Session-ID b/w client & server.



→ we can create a cookie object by using cookie class constructor.

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

→ After creating cookie object we have to add that object to the response by using addCookie();

```
resp.addCookie(c);
```

→ At Server side we can retrieve cookies by using getCookies()

```
Cookie[] c = req.getCookies();
```

→ If the request doesn't associated with any cookies this method returns "null".

→ Important methods of cookie class.

① public String getName();

 -- Returns name of the cookie

② public String getValue();

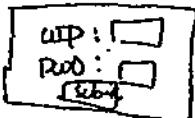
③ public int getMaxAge();

 -- Returns the maximum age of the cookie in seconds.

④ public void setMaxAge(int seconds);

→ Setting MaxAge as "-1" cookies will be disabled automatically whenever Browser's window closed.

→ Gmail



Keep login for 15 days.

(85)

c.setMaxAge(15*24*60*60);

Cookie.html

```
<form action = "/session1/test1">  
<h1>Enter cookie information </h1>  
Name : <input type = text name = uname >  
Value : <input type = text name = value >  
<input type = submit >  
</form>  
<a href = "Session1/test2" >Cookie Information </a>
```

CookieDemoServlet-1.java

```
public class CDS1 extends HttpServlet {  
    public void doGet(HttpServletRequest req, HttpServletResponse res) {  
        PrintWriter out = res.getWriter();  
        String name = req.getParameter("uname");  
        String value = req.getParameter("value");  
        Cookie c = new Cookie(name, value);  
        c.setMaxAge(300);  
    }  
}
```

```
res.addCookie(c);
o.println("Cookie Added Successfully");
RD rd = oeq.getRD("cookie.html");
rd.include(oeq, resp);
```

CookieDemoServlet.java

```
public class CDS2 extends HttpServlet
{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Cookie[] c = request.getCookies();
        if (c == null)
        {
            o.println("No Cookies are Associated with the request.");
        }
        else
        {
            for (Cookie c1 : c)
            {
                String name = c1.getName();
                String value = c1.getValue();
```

`o.open("name" -- "value");`

(86)

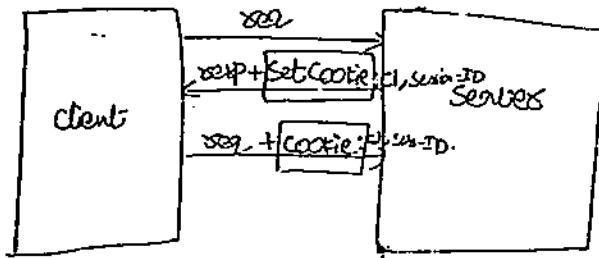
3

Advantages of Cookies

- ① Very easy to implement.
- ② Persist across Browsers shutdowns also.
- ③ All web servers provides automatic support for cookies.

Disadvantages of Cookies

- ④ Cookies can be enable or disable at client side to meet security constraints.
 - If the cookies are disabled then session management by using cookies is not possible.
 - The no. of cookies supported by any browser is always fixed.
 - The size of cookies is also fixed, hence we can't store huge amount of information by using cookies.



→ Session ID & cookies are exchanged b/w client & server by using Set-cookie Response Headers & cookie Request Headers.

→ If the cookies are disabled at client side then the Browser is unable to see setcookie Response Headers. Hence Browser won't get any cookies or session-ID to sent by Server.

→ If the cookies are disabled at client-side then the Browser unable to send cookie RequestHeaders. Hence server won't get any cookies or Session-ID from the Request & Every Request is treated as New Request. Hence Entire Session management fails.

→ To resolve this problem Sun People introduced the most powerful & painful Technique - URL-Rewriting.

Comparison b/w Session API & cookies

Session-API

cookies

① Session Information will be maintained at Server side

① Session Info will be maintained at client side.

② Best suitable if we want to store huge amount of Info

② Best suitable if we want to store very less amount of Info.

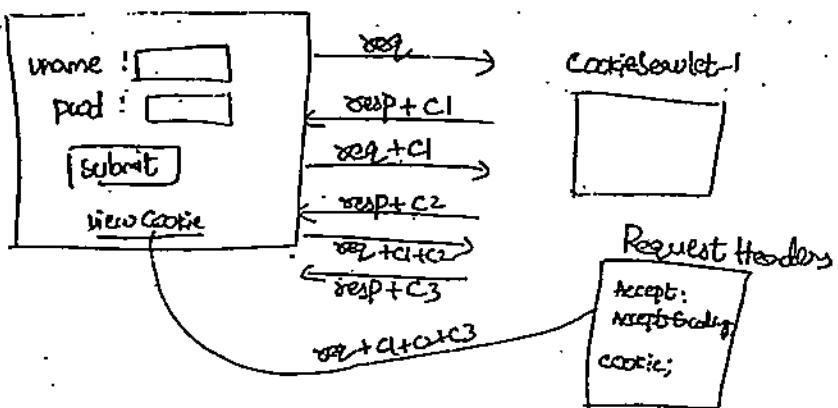
③ The Session Information can be any type i.e need not be String Info.

③ The Session Info should be String Info (both cookie name, Value should be String objects only).

④ Security is High & won't create any into traffic problems.

④ Security is less & may face into traffic problems.

(87)



Note

- If we are setting MaxAge to the cookie Such type of cookies are called Persistent cookies & these are available in the Local file System.
- If we are not setting any maxage such type of cookies are called Temporary cookies. Once the Browser closes automatically the cookies will be expired. These cookies will be stored in the Browser's cache & not visible in the Local file System.

URL-Rewriting

- URLs can be Re-Routed or Encoded to include Session Information. This Technique is called URL-Rewriting.

Ex: URL; JSESSIONID = 12345

i.e URL Rewriting = URL + Session Info

- HttpServlet Response Defines the following methods to append sessionID to the URL.

- ① Public String encodeURL (String URL);
 - Returns URL by appending JSession ID.

- 2) public String encodeRedirectURL (String url)
- Returns url by appending session-ID.
- This can be used as an Argument to send-Redirect().

⇒ The above 2 methods will append Jsession-ID if & only if
cookies are disable.

⇒ If the cookies are enable, these methods returns the same
url's with out appending Jsession-ID.

⇒ At Server side we can identify whether SessionID is coming
as the part of url or from the cookie Request Headers
by using the following methods of HttpServletRequest.

- (1) boolean isRequestedSessionIdFromURL();
- (2) boolean isRequestedSessionIdFromCookie();

By using these methods we can identify underlying
Session management Technique.

Advantages of URL-Rewriting

(38)

- There is no chance of Disabling URL-Rewriting Technique. Hence it is universally supported.

Limitations of URL-Rewriting

- It is very difficult to Rewrite all url's to append Session Information.
- URL-Rewriting works only for Dynamic Documents.
- It is common trend to use both cookies & URL-Rewriting for session management.
- Cookies will work if they are Enabled otherwise immediately URL-Rewriting will work so that we can maintain client state always across multiple requests.

Demo program for URL-Rewriting

login.html

```
<html>
```

```
<body>
```

```
<form action = "session3/test1">
```

```
<h1> Enter session Information
```

```
Name : <input type = "text" name = "uname">
```

```
<input type = "submit" /> </h1>
```

```
</form>
```

```
</body>
```

```
</html>
```

sessionServlet1.java → Retrieve this name & adding to the second servlet.

Public class SS1 extends HttpServlet

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

```
        PrintWriter out = response.getWriter();
```

```
        String name = request.getParameter("uname");
```

```
        out.println("<html> welcome to Durga soft school </html>");
```

```
        out.println("<a href = "/session3/test2?name=" + name + ">");
```

```
            click here to get ur name </a>");
```

```
}
```

```
}}
```

sessionServlet2.java - Accepting that name & Print the name

Public class SS2 extends HttpServlet

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

```
        PrintWriter out = response.getWriter();
```

```
        String name1 = request.getParameter("name");
```

```
        out.println("<html> hi " + name1 + " Good Morning. </html>");
```

```
}
```

```
}
```

blank space

Listeners 2 to 3 Ques

Q1) Describe the web Container Event Lifecycle model for the Request, Session & web-Application.

Q2) Create & Configure Listener class for each scope.

Q3) Create & Configure Attribute Listeners for each scope.

Q4) For the given Scenario Identify Proper Attribute Listener.

In the web-application there may be a chance of occurring several Events like

- ① Request Object Creation
- ② Session Object Destruction.
- ③ Context Object Creation
- ④ Attribute addition in Request Scope
- ⑤ Removal of Attribute in sessionScope.

we can configure listeners classes to listen these Events & they can do appropriate things whenever that Event occurs.

All the listeners are divided in to 3 Groups

① Request Listeners - These listen the Events Related to Request. There are 2 types of Request Listeners:

- ① Servlet Request Listeners.
- ② Servlet Request Attribute Listeners.

Q2) Session listeners - Listen the events related to session.

These are 4 types of Session listeners.

(9D)

- ① HttpSessionListeners
- ② HttpSessionAttributeListeners
- ③ HttpSessionBindingListeners
- ④ HttpSessionActivationListeners

Context listeners - Listen the events related to Context.

These are 2 types of Context listeners

- ① ServletContextListeners
- ② ServletContextAttributeListeners

Servlet Request Listener

This Listener listens the life cycle Events of Request Object

Like creation & destruction.

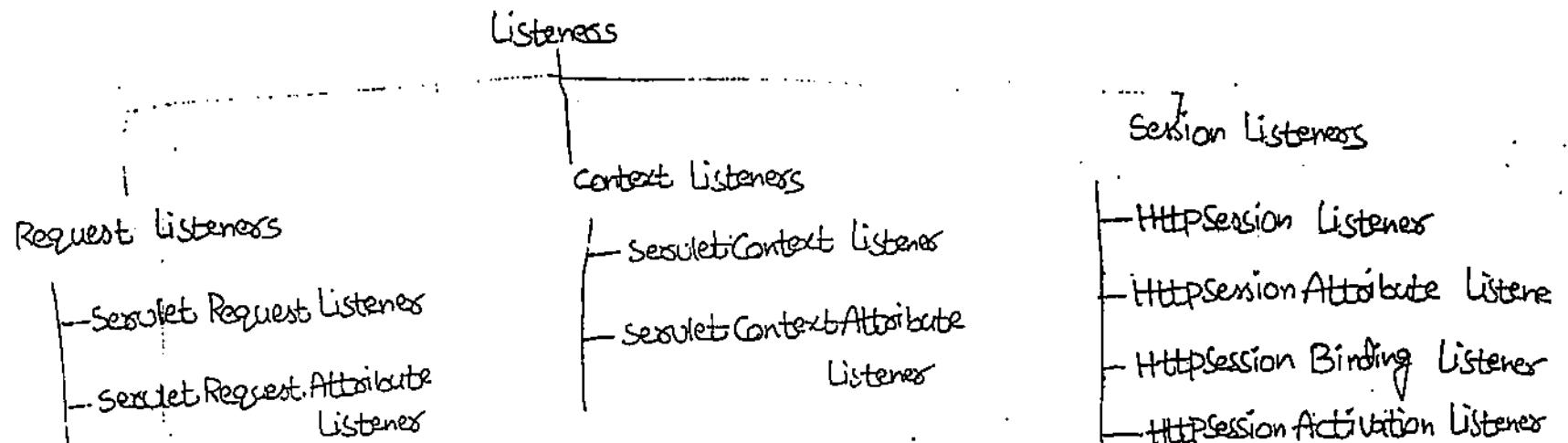
This interface defines the following 2 methods:

① Public void requestInitialized (ServletRequestEvent e)

- This method will be executed automatically by the web container at the time of Request Object creation.
i.e Just before Starting Service method.

② Public void requestDestroyed (ServletRequestEvent e)

- This method will be executed automatically by the web container at the time of Request Object Destruction
i.e Just After completing Service method.



ServletRequestListener

(91)

- This class contains the following methods to return Request & context Objects.

① Public ServletRequest getServletRequest();

② Public ServletContext getServletContext();

→ ServletRequestEvent is the child class of java.util.EventObject.

→ EventObject class contains one method getSource();

Public Object getSource();

- It returns the source which causes the event. In this case web-application is the source of event & hence we will get ServletContext object.

java.util.EventObject — getSource()

↑
ServletRequestEvent — getServletRequest()
getServletContext()

Demo Program for ServletRequestListener

→ This listener maintains counter variable to represent hit count of the web-application.

SesuletRequestDemo Listener.java

```
public class SRDL implements SesuletRequestListener  
{  
    static int count=0;  
  
    public void requestInitialized (SREvent e)  
    {  
        count++;  
        System.out.println ("A new Request came at :" + new Date());  
    }  
  
    public void requestDestroyed (SREvent e)  
    {  
        System.out.println ("A Request Destroyed at :" + new Date());  
    }  
}
```

5.9 Configuring a Listener in web.xml

→ we can configure a listener in web.xml by using
<listeners>.

```
<web-app>  
    <listeners>  
        <listener-class> SRDL </listener-class>  
    </listeners>  
</web-app>
```

- ⇒ <listeners> is the child tag of <web-app> & we can place anywhere with in <web-app> 92
- ⇒ we can configure more than one listener of same type, the order of execution of these listeners is depends on the order of <listeners> in web.xml.

First Servlet.java

```

public class FirstServlet extends HttpServlet
{
    public void doGET(--) {
        static void
        PrintWriter out = resp.getWriter();
        so we can call
        by class name
        out.println("The hit count of the web application is:" +
                    SRDL.count);
    }
}

```

⇒ web container is responsible for the creation of Listener class obj for this Web-Container always calls public no-argument constructor. Hence every Listener class should compulsory contain public no argument constructor.

⇒ Instantiation of the Listener should happen at either server start-up or At the time of Application Deployment.

Servlet Request Attribute Listener

- This Listener listens the Events Related to Request Scoped Attributes like Attribute Addition, Attribute Replacement & Attribute Removal.
- This interface defines the following 3 methods.
 - ① Public void attributeAdded (ServletRequestAttributeEvent e);
- This method will execute automatically by the Web-Container whenever we are adding an Attribute in the Request Scope.
 - ② Public void attributeRemoved (SRAE e);
 - ③ Public void attributeReplaced (SRAE e);

Servlet Request Attribute Event

- This class contains the following 2 methods.
 - ① Public String getName();
- Returns the Name of the Attribute which is added (or) Replaced (or) Removed in the Request Scope.
 - ② Public Object getValue();
- Returns the Value of the Attribute which is added (or) Replaced (or) Removed in the Request Scope.

- ⇒ In the case of Attribute Addition & Removal this method returns the corresponding attribute value. But in the case

of replacement into newer version Old Value of the Attribute.

(93)

java.util.EventObject → getSource()

Servlet Request Event

getServletRequest()



Servlet RequestAttribute Event

getServletContext()

getName()

getValues()

⇒ SRAE is the child class of SRE

Demo Program

Public class SRADL implements SRAL

{

 Public void attributeAdded (ServletRequestAttributeEvent e)

{

 S.o.println (e.getName() + " -- Attribute Added");

}

 Public void attributeRemoved (SRAE e)

{

 S.o.println (e.getName() + " -- Attribute Removed");

}

 Public void attributeReplaced (SRAE e)

{

 S.o.println (e.getName() + " -- Attribute Replaced");

}

3

```
<coeb-app>
  <listeners>
    <listener-class> SRADL </listener-class>
  </listeners>
  FrostServlet
  ↑
  /test
```

```
<web-app>
```

FrostServlet.java

public class FS extends HttpServlet

{ public void doGet(--) ---

```

  PW out = seq.getWriter();
  out.println("This is SRADL");
  out.println(seq.getAttribute("attr1"), "SCJP");
  seq.setAttribute("attr2", "SCWCD");
  seq.setAttribute("attr1", "SCBCD");
  seq.removeAttribute("attr1");
}
```

}

SeruletContext Listener

(94)

→ This listener listens the life cycle Events of Serulet Context like creation & destruction.

→ This interface defines the following 2 methods.

(1) public void contextInitialized (SeruletContextEvent e)

- This method will be executed automatically by the web container at the time of Context Object creation i.e At the time of Application Deployment.

(2) public void contextDestroyed (SeruletContextEvent e)

- This method will be executed automatically by the web container at the time of Context Object destruction i.e At the time of Application undeployment.

~~Serulet Context Event~~

→ It is the child class of java.util.EventObject & contains only one method getSeruletContext();

Public SeruletContext getSeruletContext();

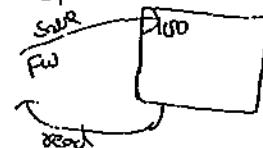
java.util.EventObject - getSource();



SeruletContextEvent - getSeruletContext();

Persistent Counter Util

Static int Count



To save the count value permanently (i.e At the time of Restart,

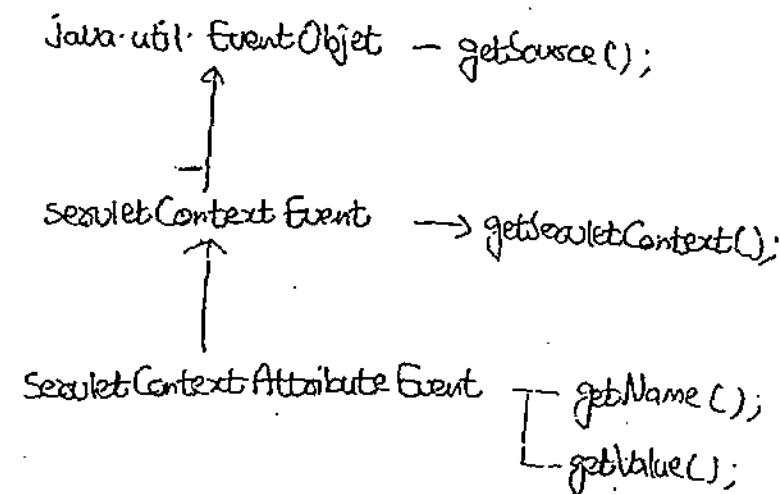
Deployment, undeployment,
Reload the value will gone

Servlet Context Attribute Listener

- This listener listens the Events Related to context scoped Attributes like Addition, Removal, Replacement.
- This Interface defines the following 3 methods.
 - 1) Public void attributeAdded (ServletContextAttributeEvent e);
 - 2) public void attributeRemoved (SCAE e);
 - 3) public void attributeReplaced (SCAE e);

Servlet Context Attribute Event

- It is the child class of ServletContextEvent.
- It contains the following 2 method.
 - 1) Public String getName();
 - 2) Public Object getValue();



HttpSession Listener

(95)

- This Listener listens the lifecycle Events of HttpSession Object like creation & destruction.
- This interface defines the following 2 methods.

- ① public void sessionCreated(HttpSessionEvent e);
- ② public void sessionDestroyed(HttpSessionEvent e);

HttpSessionEvent

- This class defines only 1 method getSession();

public HttpSession getSession();

java.util.EventObject - getSource();

HttpSessionEvent - getSession();

Demo Program to Display the number of users currently online.
(i.e to display the no. of active sessions at the server side).

SessionCounter.java

```
public class SessionCounter implements HttpSessionListener
{
    static int count = 0;

    public void sessionCreated (HttpSessionEvent e)
    {
        count++;
        System.out.println("A new session created with id : " + e.getSession().getId());
    }
}
```

```
public void sessionDestroyed ( HttpSession e )
{
    count--;
    System.out.println ("An existing session Destroyed with Id :" +
        e.getSession().getId ());
}
```

FirstServlet.java

```
public class FS extends HttpServlet
{
    PrintWriter out = resp.getWriter ();
    HttpSession session = req.getSession ();
    session.setMaxInactiveInterval (120);Session Obj expires after 2 minutes
    out.println ("The no. of users online is :" + SessionCounter.count );
}
```

web.xml

```
<web-app>
    <listeners>
        <listener-class> SessionCounter </listener-class>
    </listeners>
    <!-- FS -->
    <!-- test -->
</web-app>
```

HttpSessionAttributeListener

(96)

→ This Listener listens the Events Related to session Scoped Attributes like Addition | Removal | Replacement.

→ This interface defines the following 3 methods.

① public void attributeAdded (HttpSessionBindingEvent e)

② public void attributeRemoved (HSB E e)

③ public void attributeReplaced (HSB E e)

→ There is no Event class Named with HttpSessionAttributeEvent
For this Equivalent Event class is HttpSessionBindingEvent.

HttpSessionBindingEvent

It is the child class of HttpSessionEvent & contains 2 methods.

① public String getName();

② public Object getValue();

java.util.EventObject — getsource();



HttpSessionEvent — getSession();



HttpSessionBindingEvent —
 |
 | getName()
 | getValue()

05 M

HttpSession Binding Listener

→ whenever a particular type of object Adding / Removing / Replacing in Session Scope. If we want to perform certain activities then we should go for HttpSession Binding listeners.

→ This listeners list defines the following 2 methods:

① Public void valueBound (HttpSessionBindingEvent e)

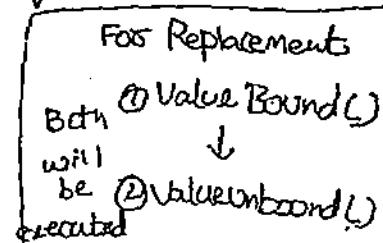
→ Adding an Attribute

② Public void valueUnbound (HSB E e)

→ Remove an Attribute

Note

→ whenever it is not required to configure HttpSessionBinding listeners in web.xml. whenever we are adding an attribute the web container will check the corresponding class implements HttpSessionBinding Listener or not. If it implements, the corresponding method will be executed automatically.



→ If both Attribute & Binding listeners are configured then Binding Listener will be executed first followed by

Attribute Listener.

Attribute Listener (attributeAdded())

↳ Session.setAttribute("attr", "scjp")

HsAL → General Attribute

↳ Session.setAttribute("attr",

HsBL → Dof

Value Bound()

attributeAdded();

new Dof());

Public class FS extends HttpServlet

(97)

```
{  
    Public void doGet(--) --  
{
```

PW out = resp.getWriter();

HttpSession session = req.getSession();

See on
Server console

session.setAttribute("a1", "SCJP"); attributeAdded()

" " ("a1", "SCJP"); attributeReplaced()

" " ("a2", new Dog()); valueBound()
 attributeAdded()

" " ("a3", new Dog()); valueBound()
 attributeAdded()

" " ("a2", new Dog()); valueBound()
 valueUnBound()
 attributeReplaced()

session.removeAttribute("a3");

Output <h1>This is Session Binding Listener Demo</h1>.

3

SessionAttributeListener.java (For Attribute Listener)

Public class SAL implements HttpSessionBindingListener -

```
{  
    Public void attributeAdded (HttpSessionBindingEvent e)
```

```
{  
    System.out.println("Attribute Added");  
}
```

```
    Public void attributeRemoved (HttpSessionBindingEvent e)
```

```
{  
    System.out.println("Attribute Removed");  
}
```

```
    public void attributeReplaced (HSBE e)
    {
        s.open ("Attribute Replaced");
    }
}
```

Dog.java (For Binding Listener)

public class Dog implements HttpSessionBindingListener

```
{
    public void valueBound (HttpSessionBindingEvent e)
    {
        s.open ("Dog Object has added to Session Scope");
    }
}
```

```
    public void valueUnBound (HSBE e)
    {
        s.open ("Dog object has removed from SessionScope");
    }
}
```

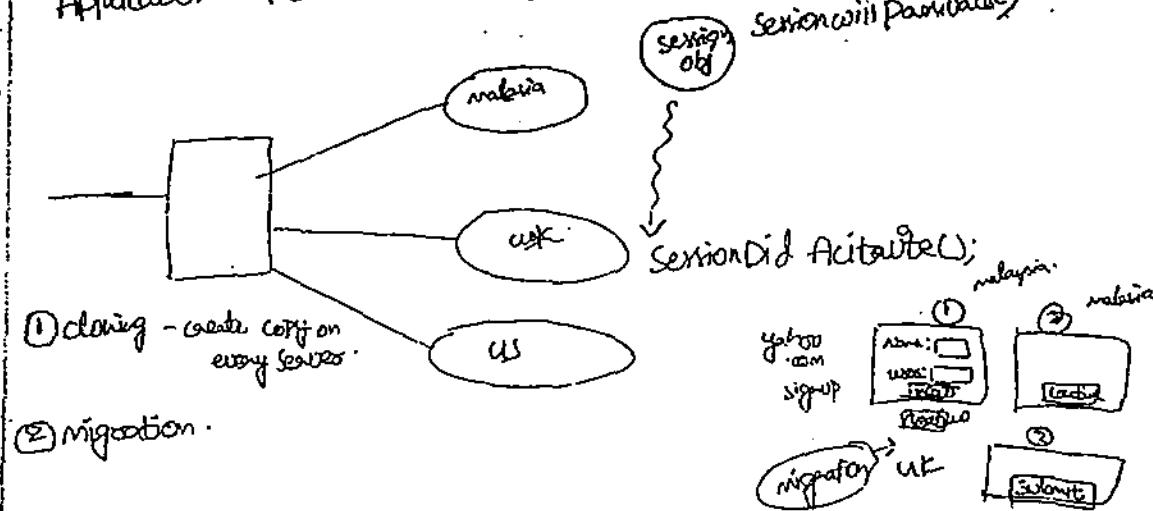
web.xml → It is not Required to configure
HTTP Session Binding Listener in web.xml.

```
<web-app>
    <listeners>
        <listener-class>SessionAttributeListener</listener-class>
    <listeners>
        <filter>
            <filter-name>DogFilter</filter-name>
            <filter-class>com.DogFilter</filter-class>
            <!--> This filter will check whether the Dog class implements
            <!--> HttpSessionBindingListener or not.
        </filter>
    </filters>
</web-app>
```

HttpSessionActivationListener

(98)

- If a web Application is distributed across several JUM's such type of web Application is called Distributed web Applications.
- The main Advantages of Distributed web Application are.
 - ① By Load Balancing we can improve Performance of Application.
 - ② By handling Fail over situations, we can keep our web Application Robust. (Fail of chances are less).



- In the Distributed web Application the Session Object will be migrated from one JUM to another.
- when ever a SessionObject is migrated from one JUM to another JUM , compulsory the corresponding ^{Session}Attributes should be migrated across the network. Hence every Session Attribute Compulsory should be Serializable.
- At the time of Session Object migration if we want to perform certain Activities then we should go for

HttpSessionActivation Listener

→ This interface defines the following 2 methods.

① Public void sessionWillPassivate (HttpSessionEvent e)

-This method is called on each implementing object bound to the session just before serialization.

② Public void SessionDidActivate (HttpSessionEvent e)

-This is called on each implementing object bound to the session just after Deserialization.

Web Security (6 to 8 Ques)

99

- ① Terminology
- ② Types of Authentication
- ③ web.xml Declarations
- ④ Demo program
 - Basic
 - Form
- ⑤ Programmatic Security
- (b) Based on Servlet Specification compare & contrast the following security mechanisms.
 - ① Authentication
 - ② Authorization
 - ③ Data Integrity
 - ④ Confidentiality.

Authentication

→ It is the process of validating the user. usually we can implement Authentication by using username & password combination.

Ex: providing Username & Pwd to login in to the Bank site is called Authentication.

Authorisation

→ It is the process of validating access permissions of a user i.e. the process of checking whether the user is allowed to access particular resource or not.

→ After Authentication we have to perform Authorisation.

→ usually we can implement Authorization by using Access Control List (ACL)

Ex: we are not allowed to access others account info even though we are member of the bank. i.e. we are not authorized to access others account info.

Data Integrity

→ It is the process of ensuring that data should not be changed in Trans (Transformation) from client to server

→ we can implement Data Integrity by using

Secure Socket Layer (SSL)

Ex: If we are sending a request to transfer 1000Rs from our account to others, the bank should get Transfer

Required for 100ms but not for 5000s

(100)

Confidentiality (Data Privacy)

- It is the process of ensuring that no one except intended user is able to understand our information.
- we can achieve Confidentiality by using Encryption Mechanism
- Difference b/w Authorization & Confidentiality.
- Authorization prevents the Information from reaching unintended users in the first place.
- whereas Confidentiality Ensures that, Even the Information falls in the wrong hands it remains unreadable.

Example :- Bees Site web Application.

(1) members should have username & pass to access site — Authentication

(2) Only Premium users can get 10% special Discount - Authorization.

(3) when a user places an order Some sort of confirmation is required - Data Integrity.

(4) when a customer makes a purchase no one allow to use our credit card Information - Confidentiality.

Types of Authentication

(a) Describe the Authentication Types & Explain their process for the given Design Problem identify appropriate Authentication mechanism.

→ Servlet Specification defines the following 4 types of Authentication mechanisms.

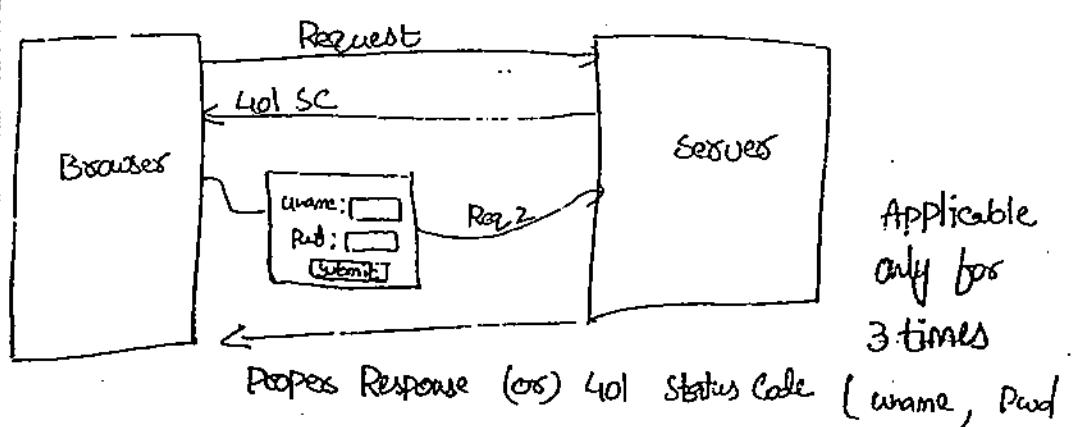
- ① HTTP Basic Authentication
- ② HTTP Form Based Authentication
- ③ HTTP Digest Authentication
- ④ HTTPS Client-Cert Authentication

HTTP Basic Authentication

→ It is the simplest & most commonly used Authentication.

→ This basic Authentication has introduced in HTTP 1.1 specification.

Process



After 3rd time we will get as 401 status code as (Name, Pwd) for Tomcat
— unauthorised person.

- Browser sends a request to the server. At this time (10) browsers don't know whether requested resource is protected or not. Hence it sends a normal HTTP Request.
- Server observes that the requested resource is secure. Hence instead of sending the requested response it will send 401 Status code saying it requires Authentication.
- By receiving 401 Status code, Browser opens a dialogue box prompting for username & password.
- Once the user enters username & password, browser resends the request with user credentials.
- When the server receives the request it validates the username & password.
- If they are valid, server sends proper required response otherwise server sends again 401 Status code.

Advantages of Basic Authentication

- (1) It is very easy to implement & set-up.
- (2) All browsers can provide support for this.

Limitations

- (1) Username & password is sending in plain text form from client to server. Hence the security is very less in this authentication.
- This authentication follows Base-64 Encoding. (not Encryption)

② we can't change look & feel of Dialogue Box.

HTTP Digest Authentication

- This mechanism is exactly similar to basic authentication, except that password is sending in Encrypted form.

→ This encryption makes more secure.

Advantage

- ① when compared with basic authentication, Digest Authentication is More Secure.

Limitations

- ① only few browsers provide the support for Digest Authentication.
(bcz Browser is responsible to perform Encryption.)

- ② most of web-servers doesn't provide support for Digest Authentication bcz Servlet Specification doesn't tell it is mandatory.

- ③ we can't change look & feel of the Dialogue Box.

HTTP Form Based Authentication

- This mechanism is exactly similar to Basic Authentication, except, instead of depending on Browsers Dialogue Box we can provide our own login html form.

- Developers is responsible to provide login & Error pages so that we can customise look & feel based on our requirement.

The only requirement of the login form is (102)

The value of Action attribute should be j-security-check.

The form should compulsorily contain 2 text fields with the names j_username & j_Password.

Except this all the remaining are customisable (we can add anything).

Advantages

It is very easy to set up.

All the Browsers should provide the support.

We can customize the login form.

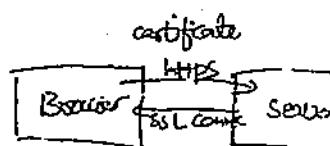
Limitation

Here also we are sending username & password in plain text form hence it is not secure.

`<form action=j-security-check>`

username : <input type="text"/>	input type = name=j_username
Pass : <input type="password"/>	name=j_password
<input type="submit"/>	

~~Ques 10~~ HTTPS Client-Cert Authentication



HTTPS means HTTP over Secure Socket Layer.

↑
Data integrity &
confidentiality

SSL is a protocol to ensure the privacy of sensitive Data

Transmitted over the Internet.

In this mechanism authentication is performed when SSL connection is established b/w Browser & Server.

All the Data is Transmitted in Encryption form by using

Public key cryptography which is handled by both Browser & Server.

Advantage

→ It is the most Secure Type of Authentication.

Disadvantage

→ It is costly to implement & maintain

→ It require a certificate from 3rd Party Certificate Authority like VeriSign.

Q2) Declarative Security. (2 to 3 Ques)

→ In the Deployment Descriptor declare

① <security-constraint>

② <web-resource>

③ <transport-guarantee>

④ <login-config>

⑤ <security-role>

<Security-Constraint>

→ we can implement web-Security mainly by using the following 3 Tags.

① <Security-Constraint>

- It defines the resources which have to be Protected, which roles are allow to access & how the resource is transported across the network.

② <login-config>

- It defines the Type of Authentication what we are using.

(103)

③ <security-role>

- It defines the Security roles which are allowed in the web Application.

⇒ All the above 3 Tags are Direct child Tags of web-app

& Hence we can place anywhere with in <web-app>

<Security-constraint>

→ This Tag defines the following 3 child Tags.

① <web-resource-collection>

- Defines the resource which is to be protected.

② <auth-constraint>

- Authorization constraints which determines what roles are allowed to access the resource.

③ <user-data-constraint>

- It specifies what type of protection is required when transporting the resource across the network.

<web-resource-collection>

It contains the following 4 child Tags:

<web-resource-name>

<description>

<url-pattern>

<http-method>

- It specifies the http-method to which the security



constraint is applicable.

→ If we are not specifying this Tag, then security constraint is applicable for all http-methods.

<auth-constraint>

→ It specifies which security roles are allowed to access the protected resource.

→ It contains the following 2 child Tags.

① <description>

② <role-name>

→ If the security constraint is applicable for all the roles we have to specify as follows:

<role-name> *

<user-data-constraint>

→ This Tag contains the following 2 child Tags.

① <description>

② <transport-guarantee>

→ This Tag specifies what type of guarantee we are providing while transporting the resource across the n/w.

⇒ The allowed values for this Tag are

① NONE - It means the Data is transported in plain text form. It is default value.

- (2) Integral - It means the Data should not be changed in Trans.
(104)
- (3) CONFIDENTIAL - It means the Data is transported in Encryption Form.

⇒ The required Priority order is confidential, Integral & None.

<login-config>

→ This Tag specifies the Type of Authentication we are using.

→ It contains the following Sub-Tags.

<auth-method>

- It specifies Authentication method.

- The allowed Values are

① BASIC

② DIGEST

③ FORM

④ CLIENT-CERT

<realm-name>

- It specifies the location where we are storing Authentication Information.

- It is required only for basic authentication.

<form-login-config>

- This Tag required to specify login Page url & Errors Page url in the case of FORM-Based authentication.

- This Tag Contains the following 2 child Tags:

① <form-login-page> login.html </f-l-p>

② <form-login Error - Page> (x) <form-^{Error}
/error.jsp JSP - Page>

<Security-sole>

→ It can be used to define Security roles in the web Application.

→ This Tag Contains the following 2 child Tags.

① <description>

② <code-name>

Demo Program for Basic Authentication.

Fist Seuket · Java

public class FS extends HS

5

public void doGET(--)--

6

Pw cut = resp.getWaiter();

a.php ("GET : After Authentication only we can Access this
Content").

3

```
public void doPost( ... ) --
```

1

Pw ad = esp.getWriters();

OpenID POST : After authentication only we can access this "secret".

3

3

I

<security-constraint>

<web-resource-collection>

<web-resource-name>

<description>

<url-pattern>

<http-method>

<web-resource-collection>

<auth-constraint>

<description>

<role-name>

<auth-constraint>

<user-data-constraint>

<description>

<transport-guarantee>

</user-data-constraint>

</security-constraint>

II

<login-config>

<auth-method>

<realm-name>

<form-login-config>

<form-login-page>

<form-error-pages>

<form-login-config>

</login-config>

III

<security-role>

<description>

<role-name>

</security-role>

5

<web-app>

First Servlet
↑
/test

<security-constraint>

<web-resource-collection>

<web-resource-name> check Servlet <

<url-pattern> /test

<http-method> POST

<http-method> GET

<web-resource-collection>

<auth-constraint>

<role-name> designer

</auth-constraint>

<security-constraint>

<login-config>

<auth-method> BASIC <!-->

<login-config>

<security-role>

<role-name> designer <!-->

</s->

<web-app>

tomcat-users.xml

(16)

- Q) Add the following entries in the tomcat-users.xml.

```
<tomcat-users>
```

```
    <role rolename="dugoprole" />
```

```
    <user username="durga" password="d8" role="dugoprole" />
```

```
    <user username="divu" password="P@P" role="dugoprole" />
```

```
</tomcat-users>
```

Demo Program for Form-Based

FirstServlet.java

② `<web.xml>`

```
<web-app>
```

```
    <login-config>
```

```
        <auth-method> FORM </auth-method>
```

```
        <form-login-config> /login.html </f-l-c>
```

```
        <form-error-page> /error.jsp </f-e-p>
```

```
    </login-config>
```

```
</web-app>
```

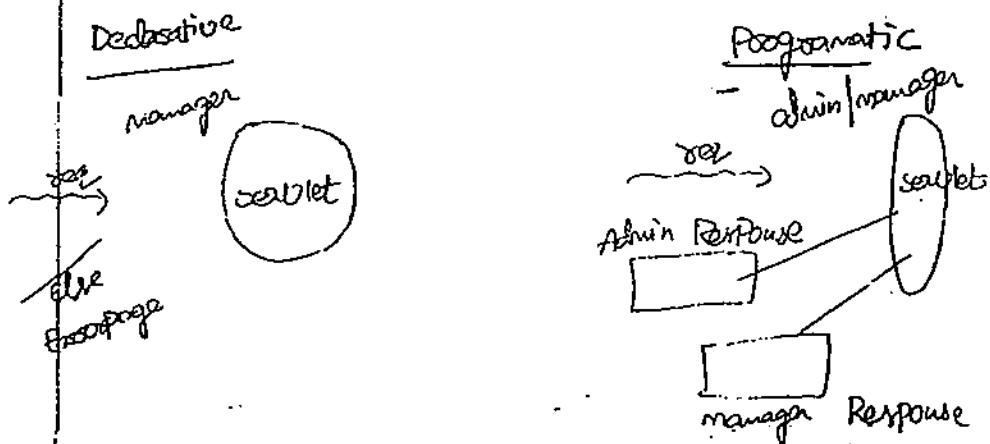
③ `<tomcat-users.xml>`

④ | login.htm

```
<html>
<body> <h1> welcome to DSS </h1>
<h2>Plz login to avail the facilities &gt;
<form action = "j_security-check">
    Enter Name : <input type = "text" name = "j_username" >
    Enter Pass : <input type = "password" name = "j_password" >
    <input type = "submit" >
</body>
</html>
```

⑤ Errors.html

<hi> your credentials are not correct. Plz provide valid
credentials </hi>



- Sometimes Declarative Security is not enough compulsory we should go for Programmatic Security.
- Based on the user's role we have to provide the corresponding Response. If the user is Admin then Admin related response & if the user is Manager then Manager related response we have to generate.
- For this type of requirements compulsory we should go for programmatic security.
- We can achieve Programmatic Security by using the following methods of HttpServletRequest.

① boolean isUserInRole(String roleName)

- If the Authenticated user belongs to the specified role then this method returns "true".
- If the Authenticated user not belongs to specified role or if the user not Authenticated then this method returns "false".

② String getRemoteUser();

admin
user: durga

- Returns the authenticated username (login - Name).

- If the user has not been Authenticated then this method returns "Null".

3) Principal getuserprincipal(),

- Returns java.security.Principal object which contains Authenticated user name.
- Returns null if the user has not been Authenticated.

Example

```
if (req.isUserInRole("admin"))  
{  
    o.println("This is Admins Home Page");  
}  
else  
{  
    o.println("This is general Home page");  
}
```

⇒ The main problem in this approach is we are hard-coding the role names in the Servlet.

→ If there is any change in the role-name modifying Servlet code is costly & creates maintenance problems.

→ To resolve this we have <security-role-ref>. By using this Tag we can map Hard-coded role names with original Role name.

```
< servlet >
```

```
!
```

```
< security-role-ref >
```

(dummy name)

This is logical role name which

is hard coded in Servlet

```
< role-name > Admin < /role-name >
```

link

```
< role-link > Dsa@Adwin < /role-link >
```

```
< /security-role-ref >
```

original Role name.

FirstServlet.java

(108)

Public class FirstServlet extends HttpServlet

{

 public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException

{

 PrintWriter out = resp.getWriter();

 String name = req.getRemoteUser();

 out.println("Hi " + name);

 if (req.isUserInRole("admin"))

{

 out.println("This is Admin's Home Page");

}

 else

{

 out.println("This is Others Home Page");

}

}

 public void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException

{

 doGet(req, resp);

}

}

web.xml

<web-app>

<Servlet>

<s-n> FS

<s-c> FS

<security-role>

<role-name> admin </role-name>

<role-link> DurgaAdmin </role-link>

</security-role>

</seawlet>

firstSeawlet

/test

<security-constraint>

<web-resource-collection>

<web-resource-name> check Seawlet </web-resource-name>

<url-pattern> /test

<http-method> GET

</web-resource>

<login-config>

<auth-constraint>

<role-name> DurgaAdmin

<role-name> Durga manager

</auth-constraint>

</s-c>

<login-config>

<auth-method> BASIC

</login-config>

<security-role>

<role-name> Durga Admin

</s-r>

<Security-role>

<role-name> DurgaManager

(109)

</Security-role>

tomcat-users.xml

<tomcat-users>

<role rolename="durgaAdmin"/>

<role rolename="durgaManager"/>

<user username="durga" password="scjp" role="durgaAdmin"/>

<user username="pallavi" password="scwcd" role="durgaManager"/>

</tomcat-users>

卷之三

3/05/14

JSP

UNIT-1

①

Q) Describe the purpose & Event Sequence of JSP page Life cycle

A) Translation Phase

b) JSP Page compilation

② Load the class

③ Create an Instance

④ call Jsp init()

⑤ call -JSP Service()

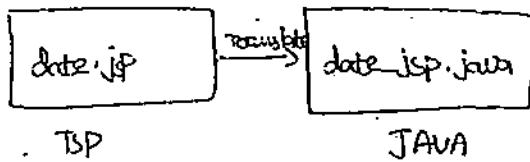
⑥ call -JSP Destroy()

→ The process of translating JSP Page (.jsp file) into corresponding Servlet (.java file) is called Translation phase

→ This can be done by the JSP containers. In Tomcat this component is called JASPER.

The Server Time is:

<.= new java.util.Date();>

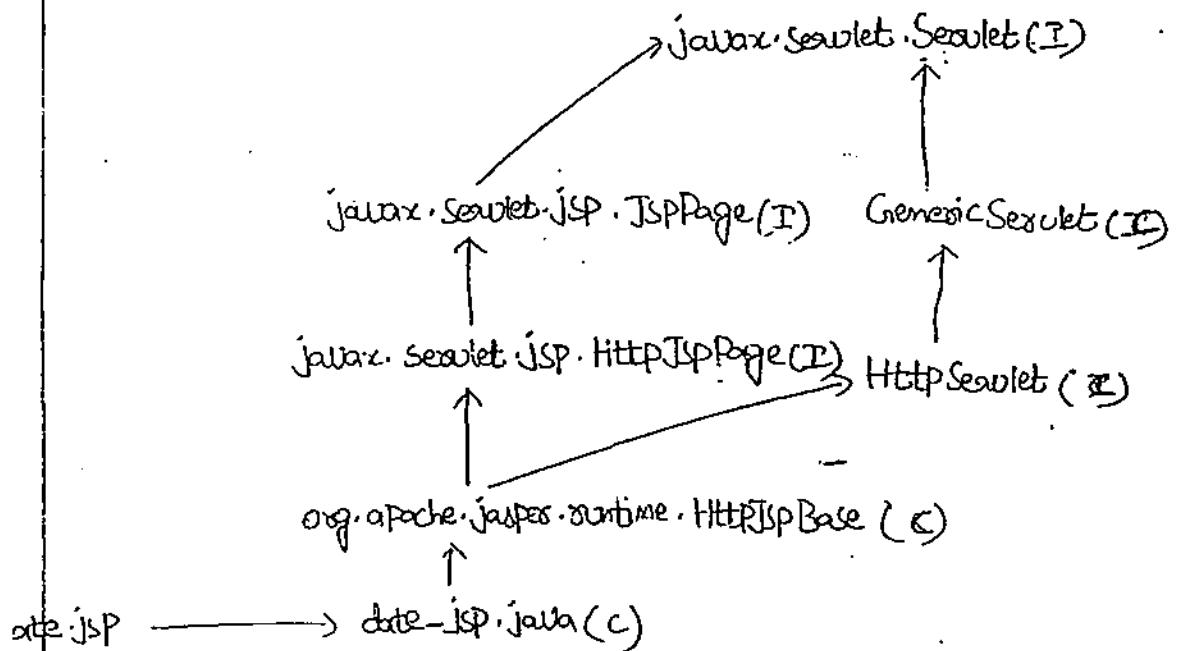


→ The generated .java file is available in the following location in Tomcat.

c:\Tomcat\work\catalina\localhost\{Name of the Application}\apache\jsp\

JSP API

- Every class which is generated for the Jsp must implement javax.servlet.jsp.JSPPage (I) or javax.servlet.jsp.HttpJspPage either directly or indirectly.
 - , Tomcat people provided a Base class org.apache.jasper.runtime.HttpJspBase for implementing the above 2 interfaces. Hence any servlet which is generated by Tomcat is extending this HttpJspBase class.



java.lang.Serulet.JSP.JSPPage (I)

→ This interface Defines the following 2 Life Cycle methods.

(2)

(1) JSPInit()

(2) JSPDestroy()

JSPInit()

Public void JSPInit()

→ This method will be executed whenever we are sending first Request to the JSP to perform Initialization Activities.

→ web Containers always calls init(ServletConfig config) of HttpServletBase class which internally calls JSPInit().

→ If we have any Initialization activities we have to override JSPInit() in the JSPPage.

```
Class HttpServletBase {
    final void init(ServletConfig config)
    {
        Public void JSPInit()
        {
            Empty Implementation
        }
    }
}
```

<? = new java.util.Date(); ?>

How to override jspInit() in our class.

<?!

```
public void jspInit()
{
    System.out.println("my Init method");
}
```

?>

) we can't place init(SeadletConfig config) in the Jsp. Because it is declared as final in HttpJspBase class.

jspDestroy()

```
public void jspDestroy()
```

→ This method will be executed only once to perform clean-up activities just before taking Jsp from out of service.

→ web container always calls destroy() available in HttpJSPBase which internally calls jspDestroy().

→ we can override jspDestroy() in our JSP.

) we can't override destroy() in the JSP bcz it is declared as final in HttpJspBase class.

```
class HttpJSPBase
```

```
{
    final new destroy()
    {
        new jspDestroy();
    }
}
```

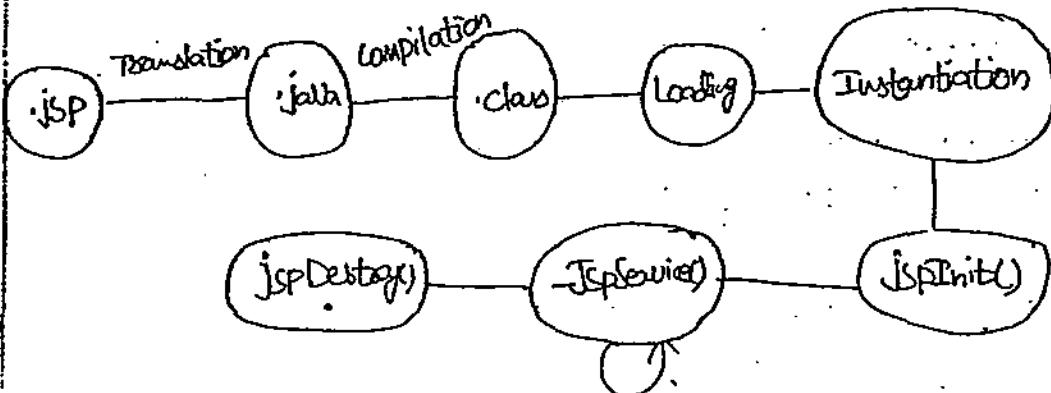
```
    jspDestroy();
```

```
}
```

1/option

- jspservice() . Public void -jspservice (HttpServletRequest req, HttpServletResponse resp) throws ServletException
- web container always calls service() available in HttpServletBase class ③ which intern calls -jspservice() method.
- At the time of Translation JSP Engine always generates -jspservice() in generated Servlet class.
- If we are placing -jspservice() explicitly in Jsp the generated Servlet contains 2 -jspservice(). If a class contain more than one method with the same signature we will get Compile-time Errors. Hence if we are placing -jspservice() explicitly we will also get CompileTime Errors.
- In the Jsp we are not allow to override service() explicitly. bcz it is declared as final in Parent class (i.e. HttpServletBase).
- In the Jsp which of the following methods are allow to place explicitly.
- (1) JspInit()
(2) init()
(3) JspDestroy()
(4) destroy()
(5) -jspservice()
(6) service()
- ②, ④, ⑥ are final class methods
-jspservice() is automatically generated by JspEngine.

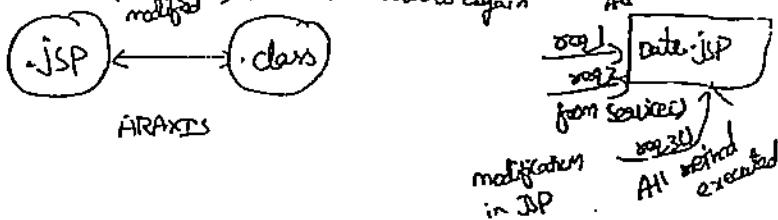
Life Cycle of the JSP



The JSP Page will participate in Translation phase in following cases-

At the Time of First Request

If the source code of JSP got modified when compared with earlier request. For this JspEngine use ARAXIS Tool to compare TimeStamps of .class & .JSP files.



For Compilation Of JSP

We can initiate Translation Phase Explicitly by using Pre-Compilation Process.

JSP Specification Provides a special request parameter

* `jsp-precompiler`. This parameter can be used to hit the JSP for compilation without processing any request.

we can invoke pre-compilation has follows. (4)

http://localhost:8080/date.jsp?JSP_Precompile=true.

⇒ till JSPInit() will be executed.

If we remove `nn` then service() will be executed

Example Date.jsp → P V JSPInit() { }

JSP Scripting Elements

(b) Identify, describe, write JSP Code for following Elements.

Template Text

Scripting Elements

Standard & Custom Actions

Expression Language Elements.

JSP Page

Template Text

Jsp Elements

Scripting Elements

Action

Actions

Page

Standard Actions

Include

jsp:include

Taglib

jsp:forward

jsp:param

jsp:useBean

jsp:setProperty

jsp:getProperty

Custom Tags

<mime:Mytag />

Traditional Model

Expression

Scriptlet

Declaration

Comments

JSP Comments

HTML Comments

Java Comments

EL

Element

eg:

\$[]

Template Text

It contains HTML & XML Tags and raw Data.

For the Template Text no Translation is required & it will pass directly as Argument to `out.write()` in `JspService()`

```
<hi>
```

The Server Time is : <%= new java.util.Date() %>

```
</hi>
```

Template Text

It will become argument to
`out.write()`

JSP Expression.

→ It will become argument to
`out.print()`.

The generated servlet is

```
public final class dateJSP
```

```
{
```

```
    public void JspService( -- )
```

```
{
```

```
    out.write ("<hi>The Server Time is " ); no proceeding is  
    out.print(new java.util.Date()); Required.
```

```
    out.write ("</hi>");
```

```
}
```

```
}
```

`write()` can take only character Data as Argument and Template Text is always characters Data. Hence Template Text will become argument to the `write()`.

→ Expression value can be any Data Type Hence we can't use `<@` write() compulsory we should go for print() which can take any type of Argument.

Directives

→ These can provide general information about JSP Page to the JSP Engine like Session Tracking is required or not, Is current JSP is EchosPage or not, which ^{scripting} language used in the JSP etc--.

`<%@ name attributeName = attributeValue %>`
↓
page|include|taglib

→ These are 3 types of Directives are possible.

① Page Directive

`<%@ page attributeName = attributeValue %>`

→ can be used to define page specific attributes.

② Include Directive

`<%@ include file="headers.html" %>`

→ To include the content of headers.html in current JSP at Translation time.

③ Taglib

`<%@ taglib prefix="mine" uri="www.java2s.com" %>`

→ To make custom Tags available to the JSP.

5) `<:= 24*36 %.>`

6) `<:= 5>3 %.>`

7) `<:= new Student() %.>` → toString() will be called.

⇒ Expression values will become argument to out.println() inside jspService().

Scriptlet

⇒ we can use Scriptlet to place java code in the JSP.

```
<:           Public void jspService(--)  
Any Java code;    => {  
/:>          }  
               }
```

→ java code inside scriptlet will be placed directly inside jspService()

→ All Java stmts inside scriptlet should compulsory ends with semicolon (;)

→ write a program to display hit counts of the JSP

```
<:  
int count=0;      It is Invalid & Generates always '1' as  
count++;        Response.  
open(count);     → Local Variable in jspService()  
/:>
```

```
<%!
    int count=0;
```

Valid & Generates 1, 2, 3 ... as output

(7)

```
%>
```

```
<%!
    count++;
    out.print(count);
```

```
%>
```

Instance Variable

```
int count=0;
{
    count++;
    out.print(count);
```

Difference b/w

```
<%!
    int i=0;
```

```
%>
```

Local Variable
in -jpservice()

<%!

→ Even we can take

static vars also;

```
int i=0;
```

```
%>
```

Instance Variable.

in -jpservice();

→ Its not recommended to use scriplets in the Jsp.

Declaration Tags

This can be used for Declaring Instance members, static members,
Instance or static Blocks, Inner classes etc.

```
<%!
```

Declarations.

```
%>
```

These Declarations will be placed Directly in the generated
Servlet but outside of -jpservice().

<?!

```
int i=10;  
static int j=20;  
int[] a={10,20,30};  
public void m1()  
{  
    s.open("hi");  
}
```

?>

⇒ Every Java Stmt inside Declaration Tag should compulsory
Ends with Semicolon (;) .

Note

→ All the Implicit objects available in JSP are Declared as Local
Variables Inside `-jspService()`.

→ Declaration Tag Code will be placed outside of `-JSPService()`
in the generated Servlet. Hence implicit objects we are
not allowed to use inside Declaration Tag otherwise we
will get CompileTimeError.

hello.jsp

CTError

out cannot be resolved.

<?!

```
public void m1()  
{  
    out.println("Hi");  
    }  
    s.println();✓
```

?>

```
<? . m1();  
?>
```

`-jspService()`

{ `out`
`out` is local to this method.

}

→ If we use `out` of this method we
will get CTError.

(8)

9: Comments

These are 3 Types of Comments allow to use in the JSP

JSP Comment:

<%-- This is JSPComment --%>

Also known as Hidden Comments bcz these are not visible in the Remaining phases of JSP Execution.

It is Highly Recommended to use JSP Comments in the JSP.

HTML Comments:

<!-- This is HTML Comment -->

Generated Servlet
class
Response.

Also known as Template Text Comments.

These are visible to the End user as the part of Generated Response Source Code.

Hence these are not recommended to use in the JSP.

JavaComments:

</

// single Line Java Comment

/* multiple lines Java Comment

; >

Generated
Servlet

Also known as Scripting Comments.

Also visible in the generated Servlet Source Code but in all other phases these are not visible.

Comment Type	In JSP Page	Unenhanced Servlet	To user Response Source code
① JSP Comments	Visible	No	No
② HTML Comments	Visible	Visible	Visible
③ Java Comments	Visible	Visible	No

⇒ Among Expressions, Scriptlets, Declarations & Jsp Comments we can't use one Inside Another i.e Nesting of these Scripting Elements is not possible. Otherwise we will get Compile Time Errors.

sample:-

- ④ `<%`
- `<% out.println("Hi"); %>`
- `<%= new java.util.Date(); %>`
- `<%>`
- `↑ unable to understand by Java compiler.`
- ① `<%`
- `out.println(<% = new java.util.Date(); %>);`
- `<%>`
- `↑ This is Very Important Line -->`
- `out.println("Hi");`
- `<%>`
- ③ `<% !`
- `<% -- This is JSPComment -->`
- `public double getBalance()`
- `{`
- `return 1000;`
- `}`
- `<%>`

Comprehension Table for JSP Scripting Elements

(9)

Scripting Elements	Syntax	Ends with ;	Is code generated Inside <code>JspService()</code>
① Expression	<code><: = %></code>	No	Yes
② Scriptlet	<code><% %></code>	Yes	Yes
③ Declaration	<code><%! %></code>	Yes	No
④ Comments	<code><%-- --%></code>	NA	Not Applicable

JSP Directives

Q6) write JspCode that uses the following Directives.

① Page ② Include ③ TagLib

Q7) Directives provides general Information about our Jsp to the Jsp Engine. These are Translation Time Instructions to the Jsp Engine.

`<%@ directiveName attributename = attributeValue %>`

↓ ↓

Page | include | Taglib

Compulsory we Should
Take Single or Double
Quotes.

page Directive

→ page Directive specifies the overall properties of Jsp Page to the Jsp Engine.

Ex:

① <%@ Page language="java" %>

-specifies to the Jsp Engine the Scripting Language used in current Jsp is java.

⇒ we can use Page Directive any no. of Times, Any where in the JSP.

② <%@ Page session="true" %>

⇒ the following is the list of all possible 13 Attributes in the Page Directive.

① import ⑧ language

② session ⑨ Extends

③ contentType ⑩ buffer

④ isELIgnored ⑪ AutoFlush

⑤ isThreadSafe ⑫ Info

⑥ isErrorPage ⑬ PageEncoding

⑦ errorPage

import

(10)

→ we can use import attribute for importing classes & interfaces in the JSP.

→ this is similar to import Stmt in Case java.

with out import Attribute

<h1> the Server Time is:

<%= new java.util.Date() %>

</h1>

with import Attribute

<%@ page import="java.util.*" %>

<h1> the Server Time is :

<%= new Date() %>

</h1>

→ To import multiple Packages the following are various possibilities.

① <%@ page import="java.util.*" %>

<%@ page import = "java.sql.*" %>

② <%@ page import = "java.util.*" import="java.sql.*" %>

③ <%@ page import = "java.util.* , java.sql.*" %>

* Within in the same JSP we are not allowed to take any attribute except import of page directive multiple times with diff values.

→ But we can take same attribute multiple times with same value.

→ Which of the following Page Directives are valid.

- (1) <%@ page session="true" Session ="true" %>
- (2) <%@ page session="true" Session ="false" %>
- (3) <%@ page Session ="true" %>
- (4) <%@ page import="java.util.*" impor="java.sql.*" %>
- (5) <%@ page import =java.util.* %> code quote Expected

In (2) we will get Translation Time Error saying
Page Directive: illegal to have multiple occurrences of
Session with diff values (old : true, new : false).

⇒ Inside JSP there is no need to import the following packages
bcz these are available by default.

- : javax.servlet.*;
- : javax.servlet.http.*;
- : javax.servlet.jsp.*;
- : java.lang.*;

Session

(11)

- we have to use this attribute to make Session object unavailable to the JSP.
- By default Session object is available in every JSP. If we don't want Session object we have to declare page directive as follows.

<1.2 Page session="false" />

- In this case Session implicit object not available to the JSP.
- If we are not declaring Session attribute explicitly or declaring Session attribute with "true" value, the equivalent generated Servlet code is

HttpSession session = null;

session = PageContext.getSession();

if false

→ These stmts will not be present in Generated Servlet Code.

- The allowed values for the Session Attribute are.

"true" TRUE True ~~dusga X~~

"false" FALSE FALSE

i.e Case Insensitive String of True or false.

- If we are taking any other we will get Translation Time Error.

<1.2 Page session="dusga" /> X Invalid value for session.

ContentType

→ we have to use this attribute to specify content type
(MIME type of the response).

```
<@ page contentType="image/jpeg" %>
```

→ the default value of the ContentType attribute is text/html.

isELIgnored

→

```
<@= session.getId() %> ${session.id}
```

```
<@= param.getParameters("uname") %> ${param.uname}
```

→ In JSP 1.2+ Expression Language has introduced the main objective of EL is to eliminate Java code from the JSP.

→ isELIgnored = "true"

EL Syntax Just Treated as Template Text (Plain Text)

with out any processing.

→ isELIgnored = "false"

EL Syntax Should be Evaluated & Print its value.

Example

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

```
<hi> Session ID : ${pageContext.session.id}
```

O/p Session ID : \${pageContext.session.id}

`<@ Page isThreadSafe = "false" %>`

(12)

`<hi> Session ID : ${PageContext.session.id}`

SessionID : 1456789

→ The Default Value in JSP 1.2 is true. But from JSP 2.0 onwards the Default Value is false.

18/05/09

isThreadSafe

→ we can use this Attribute to provide Thread Safety to the JSP.

(a) `isThreadSafe = "true"` ;

It means the JSP Page is already ThreadSafe & it is not required to generate a Servlet to implement Single Thread Model.

→ JSP Page can Process any no. of Client Request Simultaneously.

(b) `isThreadSafe = "false"`.

The current JSP is not ThreadSafe. Hence to provide ThreadSafety Generated Servlet will implement SingleThreadModel.

→ In this Case JSP Can Process only one Client Request at a Time.

→ The Default Value of `isThreadSafe` attribute is "true".

→ To provide Thread Safety which of the arrangement we have to take in the JSP.

(1) `<@ Page isThreadSafe = "true" %>`

(2) `</@ Page isThreadSafe = "true" %>`

3) Not Required any arrangement bcz every JSP is default

ThreadSafe.

Summary of Page Attributes

<u>Attribute Name</u>	<u>Purpose</u>	<u>Default Value</u>
1) import	To import classes & Interface.	javax.servlet.*; javax.servlet.http.*; javax.servlet.jsp.*; javax.servlet.jsp.jstl.*;
2) session	To make SessionObject unavailable TO JSP Page	true
3) contentType	To specify mime Type of the Response	text/html
4) isELIgnored	To Disable expression language in the JSP	false.
5) isThreadsafe	To Provide Thread Safety to the JSP for resolving Data Inconsistency Problems.	true.

Include Directive

(13)

- Inclusion
 - Static \Rightarrow Include Directive
 - Dynamic \Rightarrow Include Action \rightarrow Dynamic Inclusion.
- If several JSP contains same code it is recommended to separate that common code in separate file. whenever that common file is required just we have to include that file.

- The main Advantages of this approach are

- ① Promotes Reusability.
- ② It improves maintainability.
- ③ Enhancement will become very easy.

- ⇒ we can achieve this inclusion either by using include Directive or Include Action.

Include Directive

```
<%@ include file="header.html" %>
```

- The inclusion will happen at Translation Time Only. Hence this Inclusion is also called Static Include or Translation Time Inclusion.

Include action

```
<jsp:include page="header.html" flush="true" />
```

- The Inclusion will happen at Request Processing Time. Hence this inclusion also called Dynamic Include or RunTime Inclusion.

items in Java Certification

headers.jsp →

<@ include file="header.jsp" />

welcome TO DSS

<jsp:include page="Footer.jsp" />

main.jsp

Learning Java is not enough.
Be certified professional

Footer.jsp

~~10~~ b/w Include Directive & Include Action.

Include Directive

- ① `<%@ include file="header.jsp"`
Contains only one Attribute file
- ② Inclusion will happen at Translation Time Hence it is considered as Static Include
- ③ For both Including & Included JSP's a single Servlet will be generated Hence code sharing b/w the components is possible.
- ④ Performance is High
- ⑤ There is no guarantee for inclusion of latest version of Included JSP. It is vendor dependent.

- ⑥ If the Target Resource won't change frequently then it is recommended to use Static Include.

Include Action

(14)

Contains 2 Attributes Page, flush.

- ① `<jsp:include page="header.jsp flush="true"/>`
- ② Inclusion will happen at Request Processing Time Hence it is considered as Dynamic Inclusion.
- ③ Separate Servlets will be generated for both Including & Included JSP's Hence code sharing b/w the components is not possible.
- ④ Performance is low.
- ⑤ Always Latest Version of Included Page will be Included.
- ⑥ If the Target Resource will change frequently then it is recommended to use Dynamic Include.

→ To include header.jsp at Translation Time. The required code is.

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

→ To include header.jsp at Request Processing Time. The required code is

<jsp:include page="header.jsp" /> .

→ Which of the following inclusions are valid.

1) <%.@ include page="header.jsp" %>

2) <%.@ include file="header.jsp" %>

3) <jsp:include file="header.jsp" />

4) <jsp:include page="header.jsp" />

5) <jsp:include page="header.jsp" flush="true" />

6) <%.@ include fil="header.jsp" flush="true" %>

⇒ Page is the attribute applicable for Include Action whereas file Attribute is applicable for Include Directive.

⇒ Flush Attribute is applicable only for Include Action but not for Include Directive.

Taglib Directive

15

- It makes custom Tags available to the Jsp.
 - ```
<%@ taglib prefix="mimeType" uri="www.durgasoft.com" %>
```
  - The Taglib Directive contains 2 Attributes prefix & uri.
  - Every custumtag we are using in the Jsp must have a separate Taglib Directive.
  - uri represents the location of the tld file.

## Jsp Implicit Objects

- Q) Given a design goal write the JSP Code using appropriate implicit objects.

→ when composed with Servlet Programming Developing JSP's is very easy bcz the required mandatory stuff automatically available in every JSP. Implicit objects also one of such Area which are by default available for every JSP.

⇒ The following is the list of ISP Implicit Objects.

|   | <u>Implicit Object</u> | <u>Declared Type</u>                          |
|---|------------------------|-----------------------------------------------|
| ① | request                | javax.servlet.http.HttpServletRequest<br>(I)  |
| ② | response               | javax.servlet.http.HttpServletResponse<br>(I) |
| ③ | config                 | javax.servlet.ServletConfig (I)               |
| ④ | application            | javax.servlet.ServletContext (I)              |

⑤ session

javax.servlet.Session (I)

⑥ out

javax.servlet.jsp.JspWriter (Abstract class)

⑦ page

java.lang.Object (class)

⑧ pageContext

javax.servlet.jsp.PageContext (Abstract class)

⑨ exception

java.lang.Throwable (class)

⑩ ~~log~~

Request & Response Object.

→ Request & Response Implicit objects are available as arguments to service method.

→ All the methods of HttpServletRequest & HttpServletResponse can be applicable on these request & response implicit objects.

<h1>

The Request method is : <!,= request.getMethod() %>

The Username is : <!,= request.getParameter("uname"); %>

The cookie array is : <!,= request.getCookies(); %>

The Request Host is : <!,= request.getRemoteAddr(); %>

response.setContentType ("text/html");

response.setHeader ("douga", "SCJP");

!>

→ In browser if we give  
uname=45

</h1>

## Q Application

(16)

- ⇒ This implicit object is of type `ServletContext` which represents the environment of the application.

Q `<hi>`

Q The application name is : `<%= application.getServletContextName() %>`  
Q The Init Parameter `uname` : `<%= application.getInitParameter("uname") %>`

Q `</hi>`

## Q Session

- Q In Every JSP by default Session Object is available. It is of type `HttpSession`.

Session is already available  
by default

Q `<hi>`

Q The Session ID is : `<%= session.getId() %>`

Q `</hi>`

Q If we can make explicitly Session Object unavailable by using.

Q Page Directive:

Session is not available

Q `<%@ Page session="false" %>`

Q `<hi>`

Q The Session ID is : `<%= session.getId() %>`

Q `</hi>`

Q CTE - Session Cannot be Resolved.

Q Which of the following makes Session Object available to the JSP

Q `<%@ Page session="true" %>`

Q `<%@ Page session="false" %>`

Q `<%@ Page session="false" session="true" %>` Translation Time Error

Q) <1.2> Page contentType = "text/html" /> " " available.

### Page

→ The Page implicit Variable is always pointing to current Servlet Object.

→ In the Generated Servlet this Varibl is declared as follows.

Object page=this;

Parent class ref can be used for holding child class Obj.

⇒ which of the following are valid. → with parent ref we can call only parent class methods but not child class methods.

① <1.= page.getServletInfo() />

② <1.= this.getServletInfo() />

③ <1.= getServletInfo() />

④ <1.= ((HttpServlet)page).getServletInfo() />

① CTE - The method getServletInfo() is undefined for the type object.

→ Page implicit Object is declared as Object type Hence we are allow to call Object Class methods only but not Servlet Specific methods. Hence Page Implicit Varibl is the most rarely used Varibl to the JSP.

## Config

(17)

- Config Implicit Object is of Type ServletConfig.
- whatever the methods we can apply on the ServletConfig object, the same we can apply on config also.
- the following is the list of applicable methods.

① getInitParameter()

② getInitParameterNames()

③ getServletName() servlet logical Name.

④ getServletContext()

## date.jsp

<hi>

The Init Parameters is : <%= config.getInitParameter("uname") %>

The Logical Name is : <%= config.getServletName() %>

</hi>

## web.xml

<web-app>

<servlet>

<s-n> DemoServlet </s-n>

<jsp-file>/date.jsp </j-f>

<init-param>

<param-name> uname </p-n>

<param-value> SCOTT </p-v>

```
</servlet>
<servlet-mapping>
 <s-> DemoServlet </s->
 <url-pattern> /test </url-pattern>
```

if s-m>

</web->

http://localhost:8080 / jsp / date.jsp → Directly calling JSP.

The Init Parameters is : null

The Logical Name is : JSP (Default Value)

http://localhost:8080 / jsp / test → Through url-pattern.

The Init Parameters is : SCOTT

The Logical Name is : DemoServlet.

⇒ To reflect Servlet Level web.xml configurations in the JSP  
compulsory we should access by using url-pattern only.

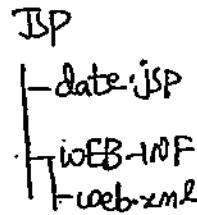
### PageContext

⇒ The Page Context Implicit Object is of Type javax.Servlet.jsp.

### PageContext

→ It is an abstract class & Vendor is responsible to provide implementation.

→ PageContext is the child class of JSP Context.



JspContext → Simple Custom Tag JSP 2.0 V

↑  
↓  
PageContext → Custom Custom Tag JSP 1.1 V

(18)

→ we can use PageContext Object for the following purposes.

① To get All other JSP Implicit Objects.

② For Request Dispatching Purposes.

③ For Attribute Management in any Scope.

Getting JSP Implicit Objects from PageContext

→ By using PageContext Implicit Object we can get all other JSP Implicit Object i.e PageContext acts as Single Stop for all other Implicit Objects.

→ Page Context class contains the following methods for this

① request — getRequest();

② response — getResponse();

③ config — getServletConfig();

④ application — getServletContext();

⑤ session — getSession();

⑥ out — getOut();

⑦ page — getPage();

⑧ exception — getException();

These methods are not usefull with in the JSP but these are useful outside of JSP mostly in Custom Tag handlers.

### Request Dispatching by using PageContext

You can perform Request Dispatching by using the following methods of PageContext.

① Public void forward (String Target)

② Public void include (String Target)

The Target Resource can be specified either by Absolute Path or by Relative Path.

#### Example

##### first.jsp

<h1>

This is First JSP

</h1>

<!

PageContext.forward ("second.jsp");

>

Output  
forward - This is Second JSP

include - This is First JSP

This is Second JSP

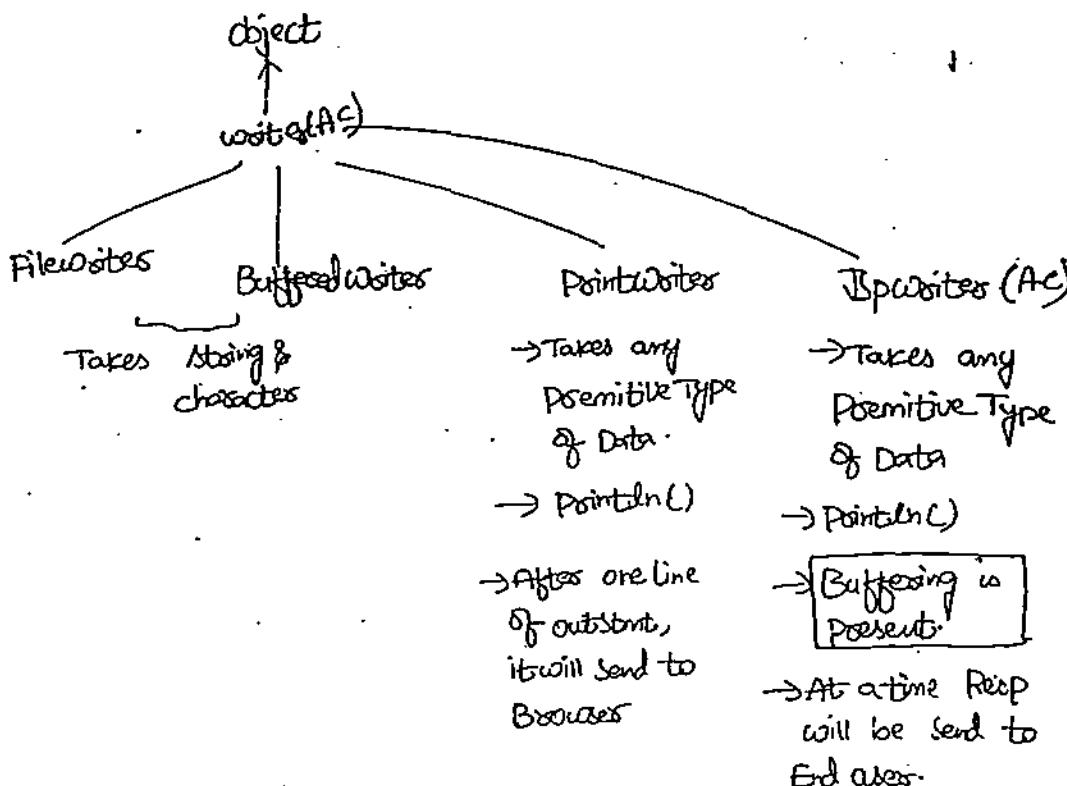
##### Second.jsp

<h1>This is Second JSP </h1>

## Attribute Management In Any Scope

(19)

→ we can use PageContext Object to perform Attribute Management in any scope. (will be covered in JSP Scopes).



### Out

→ This implicit object is of type javax.servlet.JspWriter. This class is specially designed for JSP's to write character data to the response.

→ The main diff b/w JspWriter & PrintWriter is, in the case of PrintWriter no buffering is available. Hence whenever we are adding character data to the response it will be delivered immediately to the browser with out waiting until committing the response.

- But in the case of buffering Concept is available.
- whenever we are adding Data to the response, it will store in the buffer without Delivering immediately. once the response is committed then only entire Response will be delivered to the Browser.
- Except this Buffering Diff there is no others Diff b/w JspWriter & PrintWriter.

JspWriter = PrintWriter + Buffering

### Example

#### Demo.jsp

<%

```
PointWriters outl = response.getWriter();
outl.println("Hello QSR Good Morning");
outl.println("Hello Babu Good Morning");
outl.println("Hello QSR Good Morning");
outl.println("Hello Babu Good Morning");
outl.println("Hello QSR Good Morning");
outl.println("Hello Babu Good Morning");
```

%>

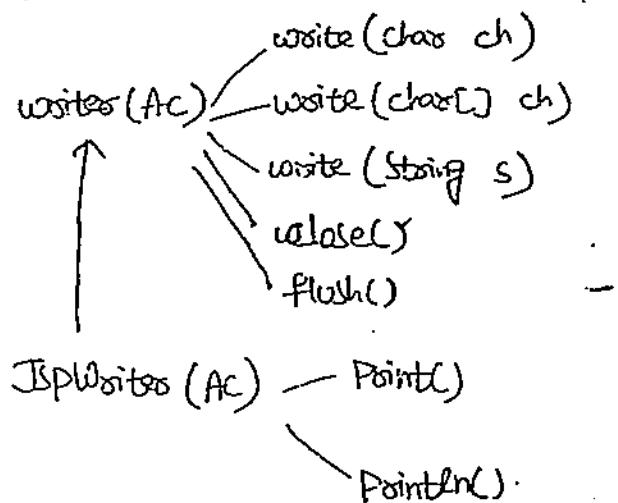
PointWriter-out:  
JspWriter-out

up: Hello Baby Good Morning  
 Hello Baby " " } PointWriter  
 Hello Baby " " } PointWriter  
 Hello USR Good Morning } JspWriter  
 Hello USR " " } JspWriter  
 Hello USR " " } JspWriter

(20)

- o with in the Jsp we can use either PointWriter (or) JspWriter but not recommended to use both simultaneously
- o JspWriter is the child class of writer (Ac). Hence all the methods available in writer or by default available to the JspWriter through Inheritance.

- o In addition to these methods, JspWriter Contains its own set of Point() & Pointln() to add any type of Data to the Response.



### Example

<%

out.println("The PI Value is"); String

out.println(3.1415); float

out.println("This is"); characters

out.println(true); Boolean

%>

→ How we can write Response in Jsp's:

A) By using JspWriter.

→ what is the diff b/w PrintWriter & JspWriter.

JspWriter = PrintWriter + Buffering

Configuring ErrorPages in Jsp's (Exception)

→ we can configure Error Pages in Jsp's by using the following 2 approaches.

① Declarative Approach

② Programmatic Approach

### Declarative Approach

→ we can configure Error Pages according to a particular exception or according to error-code in web.xml as follows.

(21)

```
<web-app>
 <error-page>
 <exception-type>java.lang.AE</e-t>
 <location>. /errors.jsp</l>
 </error-page>
 <error-page>
 <error-code>500</e-c>
 <location>/errors.jsp</l>
 </error-page>
</web-app>
```

The error Pages Configured in this approach are applicable to the entire web application.

### Programmatic Approach

We can Configure ErrorPage for a Particular JSP by using `errorPage` Attribute of Page Directive.

### Demo.jsp

```
<%@ page errorPage="errors.jsp" %>
```

The Result is : <h1= 10/0 %> </h1>

In Demo.jsp if any Exception or Error occurs then errors.jsp is responsible to display this Error.

This way of Configuring Error Page is applicable for a Particular JSP.

- we can declare a JSP as exception by using isException attribute of Page Directive.
- In the Exception pages only exception implicit object is available.

### Exception JSP

<@ page isException="true" %>

<%! we are facing some Technical Problems. Plz Try later.

The Problem we are facing is : <%= exception %>

- If JSP is not declared as exception but if we are trying to access Exception Implicit Object we will get CompileTimeError Saying - Exception can not be resolved.

- If we are accessing exception directly without any exception, then exception implicit object refers "null" which approach is recommended.

Declaration Approach is recommended bcz we can customise Exception Pages based on Type of Exception or exception-code.

### Note

- ① All JSP Implicit Objects are available as local Variables of `jspService()` in Generated Servlet. Hence we are allowed to use these implicit objects in Scriptlet & Expression Tag.

- o Bcz this code will be placed in `JspService()`. 22
- o But we are not allowed to use in Declarative Tag bcz  
its code will be placed outside of `service()`.
- o
- o ② Among all Jsp Implicit Objects Except Session & Exception
  - o all the remaining Objects always available in every Jsp.
  - o we can't make them unavailable.
- o
- o ③ session Object by Default available in every Jsp. If we  
don't want we can make it unavailable by using `PageDirective`
  - o `<%@ Page session="false" %>`
- o
- o ④ Exception Implicit Object is by default not available in  
every Jsp. we can make it available by using `PageDirective`
  - o `<%@ page isErrorPage="true" %>`

S10)

## JSP Scopes

→ In Servlets we have the following 3 scopes for storing information in the form of Attributes.

① Request Scope

② Session scope

③ Application scope.

→ In addition to these 3 scopes in JSP's we have pageScope also.

### Request Scope

→ In Servlets it is maintained by Servlet Request Object but in JSP's it is maintained by request Implicit Object.

→ The information stored in request Scope is available for all the components which are processing that request.

→ requestScope will be start at the time of request object creation i.e. before starting Service method & will be lost at the time request object Destruction i.e. just after completing Service method.

→ we can perform attribute Management in request Scope by using the following methods:

① Public void setAttribute (String name, Object value);

② Public Object getAttribute (String name);

⑥ ⑦ public void removeAttribute (String name);

(23)

⑧ ⑨ public Enumeration getAttributeNames();

### Session Scope

→ In Servlets Session Scope is maintained by HttpSession Object but in Jsp's it is maintained by session implicit Object.

→ Session Scope will be Started at the time of Session Object creation (i.e. at the time of executing get session method).

& will be lost once session Expires either by invalidate().

or TimeOut.

→ The information stored in the Session Scope will be available for all components which are participating in that Session.

HttpSession Contains the following methods for Attribute Management

⑩ public void setAttribute (String name, Object value)

⑪ public Object getAttribute (String name)

⑫ public void removeAttribute (String name)

⑬ public Enumeration getAttribute

### Application Scope :-

→ we can maintain application scope by using ServletContext

Objects in Servlets & by application implicit object in Jsp's

The Information stored in application scope is available for

- all components of the web application.
- Application Scope will be started at the time of `ServletContext` object creation (i.e. at the time of application deployment) & will be lost at the time of Context Object Destruction (i.e. at the time of application undeployment).
- `ServletContext` defines the following methods for application scope attribute mgmt.
- `setAttribute`, `getAttribute`, `removeAttribute`, `getAttributesNames`

### Page Scope

- This scope is applicable only for JSP's but not for Servlets.
- This scope is maintained by Page Context implicit Object.
- Information stored in PageScope is by default available within the Translation unit only.
- Page scope is most commonly used scope in custom tags to share information b/w Tag Handler classes.
- PageContext Class defines the following methods for attribute mgmt in PageScope:

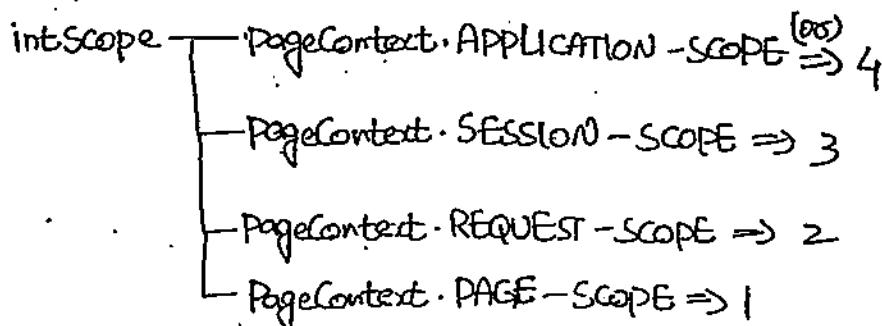
- ① Public void `SetAttribute (String name, Object value);`
- ② Public void `removeAttribute (String name);`
- ③ Public Object `getAttribute (String name);`

## Extra methods in PageContext

(24)

→ PageContext Class contain the following methods to perform Attribute Mgmt in any Scope.

① Public void setAttribute (String name, Object value, int scope);



② Public Object getAttribute (String name, int scope)

③ Public void removeAttribute (String name, int scope).

→ which of the following is valid way to store an attribute  
in Page Scope.

① page.setAttribute ("duoga", "SCJP");

② PageContext.setAttribute ("duoga", "SCJP");

③ PageContext.setAttribute ("duoga", "SCJP", 4); Applic SCP but we need PageScope

④ PageContext.setAttribute ("duoga", "SCJP", 1);

⑤ PageContext.setAttribute ("duoga", "SCJP", PageContext.PAGE\_SCOPE)

→ PageContext defines the following extra methods also

① Public Enumeration getAttributeNamesInScope (int scope)

② Public Object findAttribute (String name);

→ First it will Searches in the pageScope If it is not available then it will Search request , Session & application Scopes respectively. If it is not available it will say that the attribute is not available.

→ getAttribute (String name) → Searches in only page Scope If not it will say that , attribute is not available.

→ what is the diff btw getAttribute (String name) & getAttribute (String name , int scope)

→ what is the diff btw getAttribute (String name) & findAttribute (String name)

write a JSP to display the no. of Session Objects Created at Server side.

⇒ <%@ Page %> isELIgnored = "false" %>

<%

    Integer count = (Integer) application.getAttribute ("sessionCount");

    if (session.isNewt())

    { if (count == null)

        count = 1;

else

count++;

(25)

PageContext.setAttribute("sessionScope", count);

}

The No. of Session Objects : \${sessionScope}.

(or) <%= count %>.

### JSP Standard Actions

Q) for the given Design Problem write code by using the  
following Standard actions.

① <jsp:useBean> with Attributes - Id, Type, Name, Scope.

② <jsp:getProperty>

③ <jsp:setProperty>

⇒ we are generally using Scripting Elements in the JSP. This approach is very easy approach at beginners level.

#### Demo.jsp

```
<%! public int squareIt(int i)
{
 return i*i;
}%>
```

<h1> The Square of 4 is : <%= squareIt(4) %>

The Square of 5 is : <%= squareIt(5) %>

</h1>

→ This approach has several serious Disadvantages.

(1) There is no clear separation of Presentation & Business Logic. The person who is writing this JSP Should have Compulsory the knowledge of both java & html. which may not possible always.

(2) This approach doesn't promote Reusability.

(3) It reduces Readability of code also.

⇒ we can resolve these problems by Encapsulating entire Business logic inside JavaBean.

CalculatorBean.java

Package Pack1;

Public class CalculatorBean

{ public int squareIt(int i)

{

return i \* i;

}

3

Demo.jsp

<jsp:useBean id="calc" class="Pack1.CalculatorBean" />

<hi> The Square of 4 is <:= calc.squareIt(4) /.>

The Square of 5 is <:= calc.squareIt(5) /.>

</hi>

BeanFix

demo.jsp

WEB-INF

classes

Pack1

CalculatorBean.java

.class

→ The advantages of this approach are (4)

① Separation of Presentation & Business Logic (Presentation logic is available in JSP & Business logic is available in Bean class), so that readability of code has improved.

② Separation of Responsibilities - Java Developers can concentrate on business logic whereas as HTML Page Designers can concentrate on Presentation logic. As both can work simultaneously so that we can reduce Project Development Time.

③ It Promotes Reusability of the Code. whenever specific functionality is required we can use the same Bean without re-writing.

→ we can purchase a bean for File Uploading & we can start uploading of files within minutes.

### Java Bean

→ Java Bean is a simple Java class. to use Bean inside JSP the Bean class has to follow the following rules.

① The class should contain Public no argument Constructors. Bcz JSP Engine always calling this Constructors only for Instantiating the Bean.

② For every property, Bean class should contain Public getter & Setter methods. JSP Engine calls these methods to access or mutate (changing) Bean Properties.

① <jsp:useBean />

→ we have to use this Tag to make Bean Object available to the Jsp.

→ There are 2 forms of JSP useBean.

① without Body

```
<jsp:useBean id="calc" class="Pack1.CalculatorBean" />
```

② with Body

```
<jsp:useBean attribute list >
```

≡ Body

```
</jsp:useBean>
```

→ The main objective of Body is to perform initialization for the newly created Bean Object.

→ If the Bean object is already available then jsp:useBean won't create any new object, it will use existing object only. At this time Body won't be executed.

Attribute list of <jsp:useBean />

→ <jsp:useBean > can contain the following 5 Attributes.

① Id      ④ Scope

② Class    ⑤ beanName

③ Type

- ① **Id** - This attribute is the name of the Reference Void of the Bean Object.
- By means of this Id only we can access Bean in rest of JSP.
- The Id attribute is mandatory.

```
<jsp:useBean id="calc" class="pack1.CalculatorBean" />
```

The equivalent Java code is

```
calculatorBean calc = new CalculatorBean();
Id → refers Bean Object.
```

- ② **Class** - This attribute specifies the Fully Qualified Name for the Java Bean class.

→ JSP Engine, to perform Instantiation the Value of class Attribute should be Concrete class. Abstract <sup>classes</sup> & Interfaces are not allowed (bcz creation of obj is not possible). new CBL:

→ The Bean class should compulsorily contain public No-Argument constructor otherwise we will get Instantiation Problem.

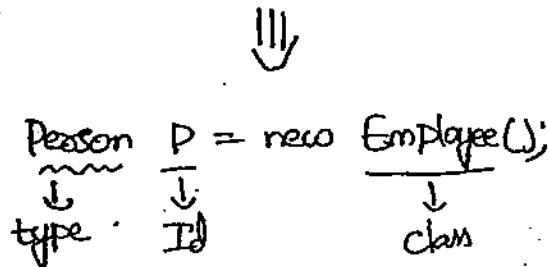
→ This attribute is optional & whenever we are not using Class Attribute compulsorily we should use type Attribute.

③ **Type** - This attribute can be used to specify the type of Reference Void.

→ The value of type Attribute can be Concrete class or Abstract class or Interface.

→ The type Attribute is optional but at that time compulsory we should specify class Attribute.

```
<jsp:useBean id="P" class="Employee" type="person" />
```



- ④ Scope - This attribute specifies in which scope the JSP Engine has to search for the required Bean Object.
    - In that scope if the bean object is not already available then JSP Engine search will create a new bean object & stores that object in the scope for the future purpose.
    - The allowed values for the scope attribute are page, request, session & application.
    - The scope attribute is optional & default value is page-scope.

```
<jsp:useBean id="calc" class="CustomerBean" scope="request" />
```

```
CustomerBean calc = null;
```

```
Synchronized
method
performing
should not.
calc = (CustomerBean) PageContext.getAttribute ("calc",
PageContext.REQUEST_SCOPE);
```

↳ listen all  
some changes  
use the obj at  
the same time.  
 $\text{calc} = \text{new}$

```
calc = new CustomerBean();
```

```
PageContext.setAttribute("calc", calc, PageContext.REQUEST_SCOPE)
```

→ To use session scope compulsory session object should be available in the JSP. (28)

<%@ Page Session="false" %>

<jsp:useBean id="calc" class="CustomerBean" Scope="Session"/>  
CTE → HTTP StatusCode 500

—Illegal for useBean to use session scope when JSP page declare (via page directive) that it does not participate in session.

⑤ beanName — There may be a chance of using Serialized bean object from local file system. In that case we have to use beanName Attribute.

#### Various Possible Combinations

① Id Attribute is mandatory.

② scope is optional & Default Scope → PAGE-Scope.

③ The Attributes class, type, beanName can be used in the following combinations.

④ class

⑤ type

⑥ class, type

⑦ type, beanName (beanName associate with type but not class)

- If class Attribute is used whether we are using type Attribute or not, it should be Concrete class & should contain Public no-argument Constructor.
- If we are using only type Attribute with out class Attribute Compulsory Bean Object should be available in specified scope. otherwise this Tag won't create any new Bean Object & raises Instantiation Exception.

Ex: `<jsp:useBean id="calc" type="java.lang.Thread" />`

In this case, compulsory Thread object should be available in pageScope otherwise we will get `java.lang.InstantiationException, bean calc not found` with in scope.

→ Consider the class Package pack1;

```
public class customersBean
{ . . .
}
```

→ Assume that no customersBean Object is already created. which of the following Standard Action creates a new Instance of this & store in the RequestScope.

- ① `<jsp:useBean name="cust" type="Pack1.customersBean" />`
- ② `<jsp:makeBean name="cust" type="Pack1.customersBean" />`
- ③ `<jsp:useBean id="cust" class="Pack1.customersBean
 scope='request' />`

② <jsp:useBean id="cust" type="pack1.CustomerBean" scope="request"/>  
object should be available already (ans → obj  
not found)

### JSP: getProperty

(29)

This Tag can be used to get the Properties of bean Object.

</>

Customer c=new Customer();

c.out.println(c.getName()); Accessing & Pointing

</>



<jsp:useBean id="c" class="CustomerBean" />

<jsp:getProperty name="c" property="name" /> getting &  
Pointing

<jsp:getProperty> contains the following 2 Attributes.

name - The name of the bean Instance from which the required  
property is obtained.

It is exactly equal to ID Attribute Value of the <jsp:useBean>

property - The Name of the Java Bean Property which has to  
retrieve.

Both attributes are mandatory.

### CustomerBean.java

```
Package Pack1;
public class CustomerBean
{
 private String name = "Durga";
 private String mail = "info@dugisoft.com";
 public String getName()
 {
 return name;
 }
 public String getMail()
 {
 return mail;
 }
}
```

### Demo.jsp

```
<jsp:useBean id="c" class="Pack1.CustomerBean" />
```

```
<h1>
```

```
 <jsp:getProperty name="c" property="name" />

 <jsp:getProperty name="c" property="mail" />
```

```
</h1>
```

Durga

info@dugisoft.com.

## JSP: SetProperty

30

→ This Standard Action can be used to set properties of a bean object:

→ we can use <jsp:setProperty> in the following forms:

```
① <jsp:useBean id="c" class="Customer" />
```

```
<jsp:setProperty name="c" property="name" value="durga" />
```

14

C.SetName("deoga");

```
<jsp:setProperty name="c" property="name" param="uname" />
```

11

Name: \_\_\_\_\_  
Id: \_\_\_\_\_

```
c.setUname(request.getParameter("uname"));
```

- It receives the value of Specified request Parameters & assign its value to the Specified Bean Property.

③ If the request parameters Name matches with bean Property Name  
then no need to use param Attribute.

`<jsp:setProperty name="c" Property="name" />`      Name:   
                  |||  
                  ..      Id:  id

④ <jsp:setProperty name='c' Property="\*" /> Name:  Age:   
→ \* specifies all Properties of c

④ \* specifies all properties of Bean

It iterates through all request Parameters & if any parameter

- name matched with bean property then it assigns request parameter value to the bean property.

#### Attributes of <jsp:setProperty>

- ① name - It refers name of the Bean Object whose property has to set.

→ This is exactly same as Id Attribute of <jsp:useBean>

→ It is mandatory Attribute.

- ② Property - The name of the Java Bean Property which has to be set.

→ It is Mandatory Attribute

- ③ Value - It specifies the value which has to set to the Java Bean Property.

→ It is optional Attribute & should not come in conjunction (combination) with "Param" Attribute.

- ④ Param - This attribute specifies the name of request Parameter whose value has to set to the Bean Property.

→ It is optional Attribute & should not come in conjunction with "Value" Attribute.

• login.html

    <html>  
        <form action="test.jsp">  
            Name : <input type="text" name="name"> <br>  
            mail : <input type="text" name="mail"> <br>  
            age : <input type="text" name="age"> <br>  
            <input type="submit" >  
        </form>  
    </html>

(31)

test.jsp

Page Scope  
→ Changes Reflected

    <jsp:useBean id="c1" class="Pack1.customersBean" scope="Session" />  
    <jsp:setProperty name="c1" property="\*"/>      → no effect  
    in same browser  
    <jsp:setProperty name="c1" property="name"/>      overriding with new value  
    </jsp:useBean>  
    <h1> Please check your values & confirm <br>  
    <jsp:getProperty name="c1" property="name"/> <br>  
    <jsp:getProperty name="c1" property="mail"/> <br>  
    <jsp:getProperty name="c1" property="age"/> <br>  
    </h1>

```

package pack1;
public class CustomerBean
{
 private String name;
 private String mail;
 private String age;
 public String getName()
 {
 return name;
 }
 public String getMail()
 {
 return mail;
 }
 public int getAge()
 {
 return age;
 }
 public void setName(String name)
 {
 this.name = name;
 }
 public void setMail(String mail)
 {
 this.mail = mail;
 }
 public void setAge(int age)
 {
 this.age = age;
 }
}

```

### Note

→ All the required type conversion

(String → int or int → String)

will take care by Bean

Related Tags.

Name:   
 mail:  → from type as  
 age:  string.

→ Taken care by  
 Bean Related Tags.

## Developing Reliable Web Components

(32)

→ we can develop reusable web components by using the following standard Actions.

① <jsp:include>

② <jsp:forward>

③ <jsp:param>

<jsp:include>

<jsp:include page="headers.jsp" flush="true" />

→ The response of included page will be placed in including page at Request Processing Time. Hence it is Dynamic Include.

→ This is the Tag Representation of PageContext.include()

→ This standard Action contains the following 2 Attributes.

① Page - Represents name of the included page & mandatory

② Flush - It determines whether the response will be flushed before inclusion or not.

It is optional Attribute. & default value is "false".

header.jsp

Master in JCT

<jsp:include page="headers.jsp" />

welcome to DSS

footer.jsp

<jsp:include page="footer.jsp" />

Learning Java is not enough

make use of that Subject is important.

→ In JSP's we can perform Include by following ways.

#### ① Include Directive

```
<%@ include file="header.jsp" %>
```

#### ② Include Action

```
<jsp:include page="header.jsp" flush="true" />
```

```
<%, script>
③ pageContext.include("header.jsp");
<%>
```

#### ④ By RequestDispatcher

```
script
<%,
RD rd=request.getRequestDispatcher("header.jsp");
rd.include(req, resp);
<%>
```

last <jsp:forward>

→ If first JSP is responsible for some Preliminary Process & second JSP is responsible for providing complete response, then we should go for forward mechanism.

```
<jsp:forward page="second.jsp" />
```

first JSP

```
<h1> This is first JSP </h1>
```

```
<jsp:forward page="second.jsp" />
```

second.jsp

```
<h1> This is Second JSP </h1>
```

(33)

→ <jsp:param>

→ while Forwarding or Including we can allow to send parameters to the Target JSP. we can achieve this by using <jsp:param>

<jsp:param name="c1" value="scjp" />

→ <jsp:param> contains the following 2 mandatory Attributes

① name - It represents name of the parameter

② value - " " value " " "

→ The parameters which are sending by using JSP:Param Tag are available as form parameters in the Target JSP.

### first JSP

<h1> This is first JSP </h1>

<jsp:forward page="second.jsp">

<jsp:param name="c1" value="scjp" />

<jsp:param name="c2" value="scwcd" />

<jsp:forward>

=> send as Query String

second.jsp? c1=scjp &

c2=scwcd

Second.jsp

<!--@ Page IsELIgnored="false" -->

<h1> This is Second JSP . The offered courses are

`$param.c1}` and `$param.c2}`

(Ex)

<--= request.getParameter("c1");>

<--= request.getParameter("c2");>

⇒ In JSP's the following are various Techniques for forwarding a request from one JSP to another JSP.

① forward Standard Action

```
<jsp:forward Page="Second.jsp" />
```

② By PageContext implicit Object:

```
<% PageContext.forward("Second.jsp"); %>
```

③ By RequestDispatcher

```
<%
 RD rd = request.getRequestDispatcher("Second.jsp");
 rd.forward(request, response);
>
```

### Conclusions}

#### case-1

```
<jsp:forward Page="http://localhost:8080/JSP/Second.jsp" />
X
<jsp:include file="http --- /Second.jsp" />
```

→ The value of file & page Attributes should be Relative Paths.  
only we are not allow to use Server Port, Protocol etc. otherwise we will get 404 - Status code.

### case-2

(24)

✓ <jsp:forward page="/test" />

✓ <jsp:include page="/test" />

✗ <%@ include file="/test" %>

→ In the case of Forward & Include actions the attribute Page can Pointing to a Servlet.

→ But in case of include Directive file attribute can't Pointing to a Servlet. It can Point to JSP, HTML, XML, XHTML etc.

### case-3

✓ <jsp:forward page="Second.jsp?cl=scjp" />

✓ <jsp:include page="Second.jsp?cl=scjp" />

✗ <%@ include file="Second.jsp?cl=scjp" %>

→ In the case of forward & Include Actions we are allow to Pass Query String.

→ But in the case of Include Directive we are not allow to Pass Query String.

## Summary of JSP Standard Action

<u>Standard Action</u>	<u>Description</u>	<u>Attribute List</u>
① <jsp:useBean>	To make Bean Object available to the JSP	id, class, type, scope beanName.
② <jsp:getProperty>	To get & Print property of Bean	name, property
③ <jsp:setProperty>	To Set Properties of the Bean	name, property, value, param
④ <jsp:include>	For including the response of Target Jsp in current Jsp Response at request processing time.	page, flush
⑤ <jsp:forward>	To Forward a request from one JSP to another JSP	Page
⑥ <jsp:param>	To send Parameters to the target JSP while Forwarding or Including	name, value.
⑦ <jsp:plugin>		
⑧ <jsp:Params>		
⑨ <jsp:fallback>		

## JSP Document

(35)

→ These are 2 types of Syntaxes are possible to write JSP

① JSP Standard Syntax

② XML Based Syntax

→ If we are writing JSP by using JSP Standard Syntax, such type of JSP's are called JSP-Pages.

→ If we are Developing JSP's Based on XML Syntax Such type of JSP's are called JSP-Documents.

① Scriptlet

Standard

<%,  
=/  
%>

XML Syntax

<jsp:scriptlet>  
=/  
</jsp:scriptlet>

② Expression

<:=  
=/  
%>

✓ <:= 2+3 %>

<jsp:expression>

=/  
</jsp:expression>

Parts of Expression  
only

X <j:el> @ z+3 </j:el>

✓ <j:el> z+3 </j:el>

③ Declaration

<%!

=

%>

<jsp:declaration>

=

</jsp:declaration>

### Directive

① <%@ Page import="java.util.\*" %>

✓ xml Syntax , <jsp:directive.page import="java.util.\*"/>

② <%@ include file="header.jsp" %>

✓ xml <jsp:directive.include file="header.jsp" />

③ <%@ taglib prefix="mime" uri="www.durgsoft.com" %>

✗ xml <jsp:directive.taglib prefix="mime" uri="www.durgsoft.com"/>

→ For the taglib directive there is no equivalent syntax in xml. we can provide its purpose by using <jsp:root> Tag.

<jsp:root xmlns:mime="www.durgsoft.com" version="21">

≡  
<jsp:root>

### Standard Actions

→ There is no difference in Standard Actions representation

btw standard & xml based Syntaxes

Ex - <jsp:useBean id="calc" class="Pack1.calculatorBean"/>

### Comments

<!-- xml Comment -->

→ JSP Specification does not provide any specific Syntax for writing comment. Hence we can use normal xml comment syntax.

Template Text - JSP Standard Syntax doesn't provide any specific way to write template text. But, xml based Syntax provides a std tag, <jsp:text> Tag      <jsp:text> TemplateText </jsp:text>

ElementStandard Syntaxxml Based Syntax

(36)

① Scriptlet

&lt;%

Any Java code

%&gt;

&lt;jsp:scriptlet&gt;

Any Java code

&lt;/jsp:scriptlet&gt;

② Expression

&lt;%=

Any Java Expression

%&gt;

&lt;jsp:expression&gt;

2+3

&lt;/jsp:expression&gt;

③ Declaration

&lt;%!

Any Java Declaration

e.g:- int i=10;

%&gt;

&lt;jsp:declaration&gt;

int i=10;

&lt;/jsp:declaration&gt;

④ Directives

Page

&lt;@ page import="java.util.\*" %&gt;

&lt;jsp:directive.

page

import="java.util.\*"

/

Include

&lt;@ include file="header.jsp" %&gt;

&lt;jsp:directive.include

Taglib

&lt;@ taglib prefix="mime" uri="dugasoft.com" %&gt;

file="header.jsp" /&gt;

No Equivalent Tag

⇒ But we can achieve this  
by using <jsp:script>

⑤	Standard Actions	<code>&lt;jsp:forward     page = "Second.jsp" /&gt;</code>	<code>&lt;jsp:forward     page = "Second.jsp" /&gt;</code>	⇒ No Difference in Syntax
⑥	Comments	<code>&lt;!-- JSP Comments --&gt;</code>	<code>&lt;!-- XML comment --&gt;</code>	
⑦	TemplateText	No specific Syntax	<code>&lt;jsp:text&gt;     Template Data &lt;/jsp:text&gt;</code>	

### How web Container Identifies JSP Document

- ① Save the JSP file with .JSPX extension
- ② Enclose entire JSP in `<jsp:root>` Tag
- ③ use `jsp-config` Element in `web.xml` -

```

<web-app>
 <jsp-config>
 <jsp-property-group>
 <url-pattern>/jspx/* </url-pattern>
 <jsp-xml> true </jsp-xml>
 </jsp-property-group>
 </jsp-config>

```

Note

(37)

- ⇒ From Servlet 2.4 Version, onwards Web Containers can identify JSP Document automatically. No special arrangement is required.

→ write a JSP-Document to point Hit count of JSP

JSP Page

```
<%!
int count=0;
%>
<%
count++;
%>
The Hit Count is:
<%= count %>
```

.JSP  
(as)  
.jspx

JSP Document

```
<jsp:root version="2.1">
<jsp:declaration>
 int count=0;
</jsp-declaration>
<jsp-scriptlet>
 count++;
</jsp-scriptlet>
<jsp-text>
 The hit count is:
</jsp-text>
<jsp-expression>count
</jsp-expression>
</jsp:root>
```

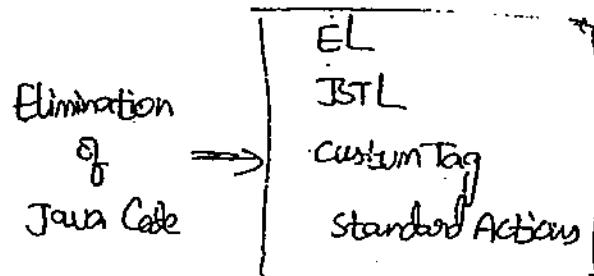
→ It's not recommended to use both Standard & XML based Syntaxes simultaneously.

→ The main Advantage of XML-based Syntax is, we can use XML editors like XML-Spy for writing & debugging JSPs.

→ In XML-based Syntax, all Tags, Attributes are case-sensitive. Attributes values must be enclosed either in single quotes or in double quotes.

⇒ which of the following is the valid way of importing util package in the JSP.

- ① <jsp:page import="java.util.\*" />
- ② <jsp:directive page import="" />
- ③ <jsp:page.directive import="" />
- ④ <jsp:directive.page import="" />



● ● ● ●

ବେଳେ ରାତି ଦିନରେ କାହିଁ ପାଇଁ କାହିଁ କାହିଁ

Digitized by srujanika@gmail.com

## UNIT-3

(38)

### Expression Language (EL)

→ It has been introduced in JSP 2.0 version. The main objective of EL is to eliminate Java code from the JSP.

→ In general we can use EL with JSTL & custom tags for complete elimination of Java code.

#### Example

(1) To point the value of request parameter "uname"

Standard : `<%= request.getParameter("uname"); %>`

EL : `${param.uname}`.

(2) To point the value of x Attribute

Standard : `<%= pageContext.findAttribute("x"); %>`

EL : `${x}`

→ To use any variable "x" in EL compulsory it should be an attribute of some scope.

`${x}` → It points the value of "x" attribute & if there is no such attribute, we will get blank space (in standard syntax tags we will get NULL but EL suppresses NULL & does not point anything (i.e. blank space)).

#### EL Implicit Objects

EL Operations

EL functions

10 to 12 Questions.

## EL Implicit Objects

EL contains 11 Implicit Objects. The power of EL is just because of these implicit objects only.

The following is the list of all <sup>Implicit</sup> EL Objects

String	String or String[]
key	value
x	SETP.
y	{SETP, SCO}

1) PageScope

2) Application Scope

3) Request Scope

4) Session Scope

5) param

6) paramValues

7) headers

8) headersValues

9) cookie — map of cookies.

10) initParam → A map of Context Initialization parameters

(but not servlet Initialization parameters)

11) pageContext

A map of Scoped Attributes

Map of request Parameters

Map of request Headers

Page Scope, Request Scope, Session Scope, Application Scope (39)

→ By using these implicit Objects we can retrieve the attributes of a particular scope.

PageScope ⇒ PageContext.getAttribute()

RequestScope ⇒ request.getAttribute()

SessionScope ⇒ session.getAttribute()

ApplicationScope ⇒ application.getAttribute()

To Access the value of Session Scoped Attribute x.

`$[sessionScope.x]`

`$[requestScope.x]` - It Points the value of request-scoped attribute 'x'. In request scope if there is no such attribute we will get blank space

`$[x]`

→ JSP Container first checks in PageScope for Attribute "x". If it is available it points its value. If it is not available it will check in request scope followed by session & application scopes. It simply acts as `PageContext.getAttribute()`.

`</@ page isELIgnored="false" %>`

`<%`

`PageContext.setAttribute("x", "SGP", 3);`

`PageContext.setAttribute("x", "SCWCD", 1);`

`%>`

The value of attribute is : \${x} ScwCD

### Param & ParamValues

By using these implicit Object we can retrieve request parameters.

param => request.getParameter()

paramValues => request.getParameterValues()

$\${\text{Param.x}}$  - It Points the Value of formparameter "x".

If the specified parameter is not already available then we will get blank Space.

If Parameters associated with multiple Values then we will get

first Value.

Pname	PValue
uname	Durga
uaddr	SRINagar

Pname	PValue
uname	{durga}
uaddr	[SRINagar, Amnespet]

$\${\text{paramvalues.x}}$

On String[] to String() will be called.

- {paramValues.x[0]} — Points first value of x
- {paramValues.x[1]} — Points second value of x.
- {paramValues.x[10]} Points blank space

Name:	<input type="text" value="duesga"/>	name
mail :	<input type="text" value="duesgatutor@gmail.com"/>	mail
food1:	<input type="text" value="Dasha"/>	food
food2:	<input type="text" value="Puri"/>	food
<input type="button" value="submit"/>		

login.html

```
<html>
 <form action="test.jsp">
 Enter Name : <input type="text" name="name">

 Enter mail : <input type="text" name="mail">

 Enter Food1 : <input type="text" name="food1">

 Enter Food2 : <input type="text" name="food2">

 <input type="submit">
 </form>
</html>
```

### test.jsp

<%@ page isELIgnored = "false" %>

`$ {param.name}` dusg a

`$ {param.mail}` dusgatac@gmail.com

`$ {param.food}` Dasha

`$ {param.age}` blank space

`$ {paramValues.name}` (java.lang @ 12345)

`$ {paramValues.name[0]}` dusg a

`$ {paramValues.name[1]}` blank space

`$ {paramValues.food[0]}` Dasha

`$ {paramValues.food[1]}` Puri

`$ {paramValues.food}` java.lang @ 15698

EL handles "null" & ArrayIndexOutOfBoundsException very nicely. It suppresses them & prints blank space for the QueryString

? fname = Balwan & lname = Kalyan. What is the O/P for the following code.

`$ {param.fname}` Balwan

`$ {param.mname}` blank space

`$ {param.lname}` Kalyan

`${paramValues.lname[0]} Kalyan`

(41)

`${paramValues.lname[1]} blankSpace`

`${paramValues.lname}`  toString() will be called  
java.lang @ 156789.

headers & headersValues

These are exactly similar to param & paramValues, except  
these are for receiving requestHeaders.

headers  $\Rightarrow$  request.getHeaders();

headersValues  $\Rightarrow$  request.getHeader();

`${headers.accept} */*`

`${headersValues.accept} toString()`

`${headers.accept[2]}`  blankSpace

Cookie

By using this implicit object, we can access cookies associated  
with the request object.

cookie  $\Rightarrow$  request.getCookie();

name	value
JSESSIONID	

`${cookie.JSESSIONID.name}`

`${cookie.JSESSIONID.value}`

`${cookie.JSESSIONID}`  toString() got called

java.lang.Servlet.http.cookie @ 15947

`initParam`

- By using this implicit object we can access Servlet Context initParameters.

`initParam => getServletContext.getInitParameters();`

`> ${initParam.mail}`

`${initParam.name}`

<web.xml>

<web-app>

<context-param>

`<?>`

`mail`

`durgsoft.com`

If the specified initParam is not available then we will get blank space bcz EL supersedes null.

`pageContext`

This is the only one EL implicit object which matches with JSP Implicit Objects.

This is the only one EL Implicit Object which is a non-map object.

By using this implicit object we can access all other JSP implicit objects in EL.

- ~~request.getMethod();~~  
~~→ \${request.method}~~  
~~✓ \${pageContext.request.method} GET~~  
~~✓ \${pageContext.session.id} 12345abcE8~~  
~~✗ \${session.id}~~

request, session implicit objects are (42)  
not available in EL

### EL Operators

→ EL contains its own specific operators. The following is the list of all possible operators.

- ① Property Access Operator (.)
  - ② Collection Access Operator ([ ])
  - ③ Arithmetic operators
  - ④ Relational operators
  - ⑤ Logical operators.
- Property Access operator (.)

$\$\{ \text{left Variable} . \text{right Variable} \}$

Map (or) Bean

MapKey (or) Property } should be a  
Valid Java Identifier.

`${custom.name}`

`${pageContext.request.method}`

`${initParam.mail}`  
 Bean → map name of map

~~`${customers.1234}`~~  
 ~~~ should be identified

### Collection Access Operator ( [ ] )

`${leftVariable[rightVariable]}`

| key     | value    |
|---------|----------|
| student | Roll.no. |

map --- mapkey } should be enclosed either  
Bean --- Property } in Single or Double Quotes

Array --- Index }  
List --- Index } Quotes are optional.

→ Map

`${initParam['mail']}`

`${initParam["mail"]}`

`${initParam[mail]}` blank space.

→ If we are not keeping quotes EL assumes that the key is an attribute stored in some scope. If that attribute is not available then we will get blank space as O/p.

→ Which of the following is the valid way of retrieving Accept Request Headers.

1) \${ header.accept }

(43)

2) \${ header[accept] } blank space as o/p

3) \${ header['accept'] }

4) \${ header ['accept'] }

Bean

1) \${ customers['name'] }

2) \${ customers["name"] }

3) \${ customers [name] } blank space as o/p.

Arrays

<!

String[] s = {"A", "B", "C"};

PageContext.setAttribute ("s", s);

. >

\${ s[0] } A

\${ s[1] } B

. \${ s[2] } C

\${ s[10] } blank space

List

<@ page import="java.util.\*" .>

<!

LinkedList l = new LinkedList();

```

l.add("congress")
l.add("LS");
l.add("TDP");
l.add("POP");

pageContext.setAttribute("list", l, 3);
/> pageContext.setAttribute("index", 3);

${list[2]} TDP ${list[0]} congress
${list[3]} POP ${list[index]} POP

```

⇒ wherever property access operator is required then we can use collection access operators. But wherever collection access operator is required we may not use property access operator.

### Arithmetic Operators

#### ① + Operator

→ EL doesn't support overloading & hence there is no string concatenation operator. "+" always acts as addition operator only.

|                    |   |                    |                                      |
|--------------------|---|--------------------|--------------------------------------|
| $\$ \{ 2+3 \}$     | 5 | $\$ \{ "abc"+3 \}$ | NumberFormat<br>NullPointerException |
| $\$ \{ "2"+3 \}$   | 5 | $\$ \{ " "+3 \}$   | NumberFormat<br>NullPointerException |
| $\$ \{ "2"+'3' \}$ | 5 | $\$ \{ null+3 \}$  | 3                                    |
| $\$ \{ abc+3 \}$   | 3 |                    | $0+3=3$                              |

(u4)

② "-" operator       $\$[a - b]$

→ All rules are exactly similar to '+' operator.

$\$["0" - 3]$

$\$["abc" - 3]$

③ '\*' operator       $\$[a * b]$

→ All rules are exactly similar to '+' operator.

$\$["0" * 3] \Rightarrow 30$

④ "/" operator (∞) div

→ All rules are exactly similar to '+' operator & Division operator always follows floating point arithmetic.

$\$[0/0]$  Infinity

$\$[0/0]$  NAN

$\$[0/2] \infty \$[0 \text{ div } 2]$  5.0

⑤ '%' operator (∞) mod

→ In the case of mod operators both floating point & integral arithmetic are possible.

$\$[10 \% 0]$  ArithmeticException.

$\$[10 \% 3]$  1

$\$[0.0 \% 0]$  NaN

## Relational Operators

|               |                    |                    |
|---------------|--------------------|--------------------|
| $> \infty$ gt | $= (\infty)$ eq    | $\leq (\infty)$ le |
| $< \infty$ lt | $\geq (\infty)$ ge | $\neq (\infty)$ ne |

$\$[ 3 < 4 ]$  true

$\$[ abcd == abcd ]$  true

## Logical Operators

→ There are 3 logical operators.

$\& (\infty) \&$  and       $\$[ T \& \& \text{true} ]$  true

$| (\infty) || \quad \infty$        $\$[ !\text{true} ]$  false

$! (\infty) !$  not

## Conditional Operators

$\$[ (3 < 4) ? "yes" : "no" ]$  yes

$\$[ (\text{true} == \text{false}) ? \text{right} : \text{wrong} ]$  wrong

## Empty Operators

$\$[ \text{empty object} ]$

true → if object doesn't exist ( $\infty$ )

object is an empty array ( $\infty$ )

" " " " String ( $\infty$ )

" " " " collection

otherwise returns false.

(45)

`$empty test` true

`$empty "durga"` false

`$empty null` true

### EL Operators Precedence

① Unary operators (`! & empty`)

② `*`, `/`, `%`

`+`, `-`

③ Relational operators

④ Logical operators - `&&`, `||`

⑤ Conditional operators.

### EL Reserved words

|       |            |     |
|-------|------------|-----|
| true  | instanceof | and |
| false | gt         | or  |
| null  | lt         | mod |
| empty | ge         | div |
|       | le         |     |
|       | ne         | not |
|       | eq         |     |

EL uses null

EL behaves very nicely with null.

In Arithmetic operations null is treated as "zero".

With in String Evaluations null is treated as "empty string".

→ with in logical operators null is evaluated as false.

## EL Functions

→ The main objective of EL is to separate Java code from the JSP. If we are having any business functionality we can separate it to a Java class & we can invoke its functionality through EL Syntax. The Page designer has to know only function name & tld file use to get its functionality. Hence EL is almost considered as alternative to custom tags.

Designing EL function Application Contains the following Stages

- ① writing a Java class with required functionality.
- ② writing tld file to map Java class to JSP.
- ③ write a taglib Directive to make business functionality available to JSP.
- ④ write EL function call.

Writing a Java class

→ Any Java class can simply acts as repository for EL Function. The only requirement of a method that acts as ELfunction is it should be declared as public & static.

→ method can take parameters.

→ void return Type methods are not recommended to use as EL functions even though it is legal. (for closing DB operations).

46

```
0 | package Pack1;
0 |
0 | public class DiceRoller
0 | {
0 | public static int rollDice()
0 | {
0 | return (int) (Math.Random() * 6 + 1);
0 | }
0 | }
0 | }
```

## Writing Taglibrary Descriptor File

- For EL functions Tld File provides Mapping b/w Java class (whose functionality is available) & JSP (which calls the functionality)
- we can configure EL function by using <function> tag in the tld.

`0` it contains 4 subTags.

① <description>

**02** <name> - By means of this name only we can call EL functionality in JSP.

Q. 3) <function-class> The fully qualified name of Java class where the function is defined. -

④ <function-signature> The signature of EL Function which is defined in Java class.

→ Here we have to provide return type; method-name & Arguments List.

: tld file MyFunction.tld

→ JSP Version

<taglib version="2.1" >

→ Developers Version (we can give  
any version)

<tlib-version> 1.2 </tlib-version>

<uri> Dice Functions </uri>

<function>

<name> rollIt </name>

<function-class> Packt::DiceRoller <

<function-signature> int rollDice() <

</function>

</taglib>

→ By means of uri, we can make EL functionality  
available in JSP.

Writing Taglib Directive.

<%@ taglib prefix="mime" uri="Dice Functions" %>

Invoking EL Function

\$ { mime : rollIt( ) }

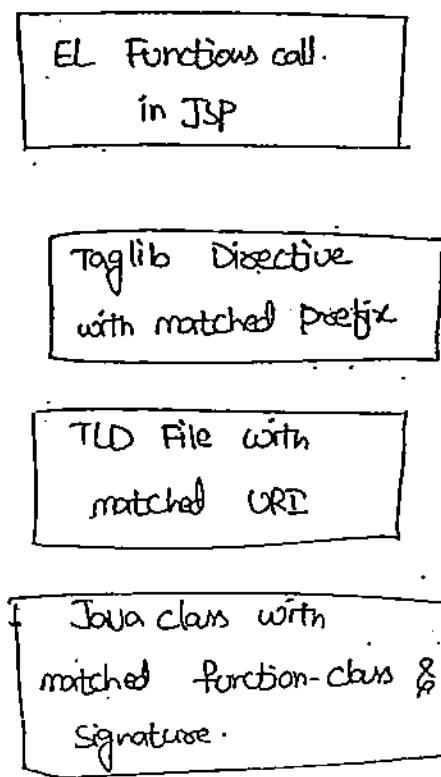
text.jsp <%@ page isELIgnored="false" %>

<%@ taglib prefix="mime" uri="Dice Functions" %>

<h1> your 3-digit lucky Number is :

$\$\{ \text{mime:} \text{convert} \}$   $\$\{ \text{mime:} \text{convert} \}$   $\$\{ \text{mime:} \text{convert} \}$  • (47)

### WorkFlow of EL Function Execution.



whenever JSP Engine encounters EL function call with prefix & EL function name then it checks for corresponding Taglib Directive with matched prefix.

From Taglib Directive JSP Engine identifies uri & checks for tld file with matched uri.

From the tld file JSP Engine checks the corresponding class & required method.

JSP Engine executes that method & return its result to JSP.

write a program to invoke math class Square root() as ELFunction call with method name m1().

### test.jsp

```
<%@ page isELIgnored="false" %>
```

```
<%@ taglib prefix="mme" uri=mathFunctions %>
```

The Square Root of 4 is : \${mme:m1(4)}

The square Root of 5 is : \${mme:m1(5)}

### myfunction.tld

```
<taglib version="2.1">
```

```
 <tlib-version> 1.0 </tlib-version>
```

```
 <uri> mathFunctions </uri>
```

```
 <function>
```

```
 <name> m1 </name>
```

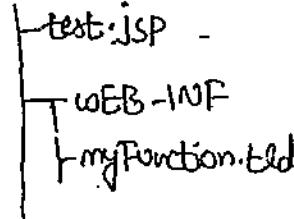
```
 <function-class> java.lang.Math </function-class>
```

```
 <function-signature> double sqrt(double) </f-s>
```

```
 </function>
```

```
</taglib>
```

ELI



## JSTL ( JSP Standard Tag Library )

(48)

- o Sun People Encapsulates the core functionality which is common to many web applications in the form of JSTL. Programmer can use this Predefined Library with out writing on his own.
- o The main objective of EL is removing Java code from the JSP. But it fails to replace Java code which processes some functionality. we can recover this by using JSTL. Hence the main objective of JSTL also for removing Java code from the JSP.

- o Entire JSTL Library Dividing in to 5 sub parts.

- o Core Library - It defines Some Standard Actions to perform Programming general stuff like Conditions & loops. It can also perform JSP Fundamental Tasks such as setting Attributes, writing output & Redirecting the request to other Pages.

- o xml - Defines several Standard Actions useful for writing & Formating xml Data.

- o fmt (formatting) - Defines several Standard Actions which can be used for Formating Numbers & Date in the process of Internationalization.

- o sql - Defines several Standard Actions can be used for

## Data Base operations.

Functional - defines several standard actions can be used for manipulating collection & String object.

### Core Library

JSTL Core Library Divided in to the following 4 parts based on functionality.

#### ① General Purpose Tags

<c:out>

<c:set>

<c:remove>

<c:catch>

#### ③ Iteration

<c:foreach>

<c:forTokens>

#### ② Conditional Tags

<c:if>

<c:choose>

<c:when>

<c:otherwise>

#### ④ URL Related Tags

<c:import>

<c:url>

<c:redirect>

<c:param>

}

## Installing JSTL

By default JSTL functionality is not available to the Jsp. we can provide JSTL functionality by placing the following jar files in web Application lib folder:

① jstl.jar - Defines Several API classes which are <sup>(49)</sup> defined by Sun.

② standard.jar - Provides library implementation classes by vendor.

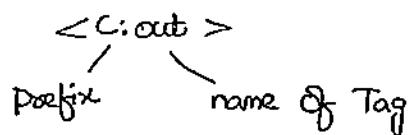
⇒ It is recommended to place these Jar Files At Server Level ( c:\Tomcat\lib folder ) instead of Application Level.

⇒ To make core library available to JSP we have to declare taglib directive as follows:

```
<%@ taglib prefix="c" uri="http://java.sun.com/JSP/jstl/core" %>
```

.. General Purpose Tags

<c:out>

  
Prefix      name of Tag

This Tag can be used for writing Template Data & Expressions to the JSP.

Form -1

```
<c:out value="Durga" />
```

It points Durga to JSP.

#### Topic-4

<c:out value="<\${param.username}"/>

→ It Points the Value of request parameter uname to the Jsp.

→ If the main value is not available or it evaluates to null  
then we can provide default value by using default attribute.

#### Example

→ If we give both `param` [Jsp /date.jsp ?uname=durga] Query String  
& default ⇒ `param` will be placed in Jsp

<c:out value="<\${param.username}" default="Guest"/>

→ If we are not passing request parameter uname then we  
will get default value guest.

→ <c:out> defines the following 3 Attributes

(1) value - It is mandatory attribute & it is for providing  
the value.

→ It should be either Literal or Runtime Expression

(2) default - It is optional Attribute & it is to provide  
default value.

→ JSP Engine considers its value if & only if the value attribute  
evaluates to "null".

(3) escapeXml.

`<c:set>`

(5)

Form-1

we can use `<c:set>` to set Attribute in any scope.

```
<c:set var="name of Attribute" value="AttributeValue"
 scope="page" />
```

Scope Attribute is optional & default scope is pageScope.

```
</@ taglib prefix="c" uri="http://java.sun.com/JSP/jstl/core"/>
```

```
<c:set var="x" value="10" scope="session" />
```

```
<c:set var="y" value="20" scope="session" />
```

```
<c:set var="result" value="${x+y}" scope="application" />
```

```
<hi>The result is : <c:out value="${result}" />
```

The result is : 30

Form-2

we can set Map or Bean property by using `<c:set>`

we can specify Bean or map Object by using TargetAttribute.

& Property name by using Property Attribute.

```
<c:set target="customers" property="name" value="Pawan" />
```

The attributes for the `<c:set>` are

① Var      ③ Scope      ⑤ Property.

Value      ④ Target

<c:remove>

→ This Tag can be used for Removing Attributes in the Specified Scope.

→ This Tag contains the following 2 Attributes.

i) Var - name of Attribute.

ii) scope - the scope in which Attribute present.

```
<c:remove var="x" scope="session" />
```

```
<c:remove var="x" />
```

If the scope is not specified then the containers will search in the page scope first for the required Attribute. If the Attribute is not available then it will search request scope followed by session & application scopes.

```
<@ taglib prefix="c" uri="
```

```
<@ page isELIgnored="false" />
```

```
<c:set var="x" value="10" scope="session" />
```

```
<c:set var="y" value="20" scope="session" />
```

```
<c:set var="result" value="${x * y}" scope="application" />
```

<@> The result is : <c:out value="\${result}" />

```
<c:remove var="x" />
```

```
<c:remove var="y" />
```

(5)

<c:remove value="result" />

<hi> The result is : <c:out value="\${result}" default="8888" />

The result is : 20

The result is : 8888

<c:catch>

This can be used to catch an Exception within in the JSP instead of forwarding to the Error page.

the Risky code we have to place as the body of <c:catch>

```
<c:catch>
 =
 Risky code
 =
</c:catch>
```

If an Exception is raised in the Risky code this Tag suppresses that exception & rest of the JSP will be executed normally.

If an Exception is raised we can hold that by using var attribute which is page scoped attribute.

<%@ Page isELIgnored="false" %>

<%@ taglib prefix="c" uri="

<hi> UserName is : \${param.uname} <br>

<c:catch var="e" >

<-->

```
int age = Integer.parseInt(request.getParameter("uname"));
```

<-->

→ only one catch tag

```
Age : ${param uname}
 is enough.
```

</c:catch>

```
<c:if test = "${e != null}">
```

```
oops -- Exception raised : ${e}

```

```
Height : ${param uheight}.
```

http://localhost:8080/jsp1/catch1.jsp ? uname=durga & uage=20 &

uheight=5.3

UserName : durga

Age : 20

Height : 5.3

http://localhost ----- uname=durga & uage=ten &

userName : durga

uheight=5.3

oops --- Exception raised : java.lang.NumberFormatException;

For input string = "ten".

Height = 5.3

## Summary of General Purpose Tag.

(52)

| Tag          | Description                                                     | Attribute                          |
|--------------|-----------------------------------------------------------------|------------------------------------|
| ① <c:out>    | For writing Expression & Template Text to JSP                   | Value, default, escapeXml.         |
| ② <c:set>    | To Set Attributes, Bean or Map Properties                       | Var, Value, Scope target, property |
| ③ <c:remove> | For removing Attributes                                         | Var, Scope                         |
| ④ <c:catch>  | For Supressing an Exception & to continue rest of JSP Normally. | Var.                               |

## Conditional Tags

### ① <c:if>

→ This is for implementing case Java if Statement.

→ There are 2 forms available for <c:if>

#### withOut Body

```
<c:if test = "test-condition" Var = "x" Scope = "request" />
```

In this case test-condition will be evaluated & store its result into Variable 'x'. In the rest of the page wherever this test

Condition is required we can use directly its value without evaluating once again.

→ In this case both test & var attributes are mandatory. scope attribute is optional & Default value is page.

with body

for future purpose we can save.



<c:if test="test-condition" var="x" scope="request">  
    ≡ Body

</c:if>

→ If test-condition is true then body will be executed otherwise without executing body rest of the JSP will be continued.

→ In this case also we can store test result inside Var Attribute

→ Here both var & scope attributes are optional but Test attribute is mandatory.

<@ Page isELIgnored

<@ taglib prefix="c" uri= -

<c:set var="x" value="10" scope="request" />

compulsory single quote

<c:if test="\${x eq '10'}" var="y" scope="session"/>

<hi> x value is : \${x} .      xValue is : \${y}

Test Result is : \${y}      Test Result is : true

`<c:choose> &lt;c:when> &lt;c:otherwise>`

(52)

We can use these Tags for implementing if-else & switch Stmt.

### Implementing if-else

JSTL doesn't contain any Tag for else we can implement if-else Stmt by using above Tags.

```
<c:choose>
 <c:when test="test-condition">
 if { ≡ Action 1
 </c:when>
 }
 <c:otherwise>
 else { ≡ Action 2
 </c:otherwise>
 }
 </c:choose>
```

If test-condition is true, Action-1 will be executed otherwise Action-2 will be executed.

### Implementing Switch Stmt

<c:choose>

<c:when test = "test-condition 1">

Action 1 implicit break Stmt; available.

<|c:when>

<c:when test = "test-condition 2">

≡ Action 2

<|c:when>

<c:when test = "test-condition 3">

≡ Action 3

<|c:when>

<c:otherwise>

default-Action

<|c:otherwise>

<|c:choose>

→ <c:choose> should compulsorily contain atleast one <c:when>,

<c:otherwise> is optional.

→ note Every <c:when> implicitly contains break Stmt.

Hence there is no chance of fall through inside switch.

→ we have to place <c:otherwise> as last case only.

→ <c:choose> & <c:otherwise> won't take any Attributes.

→ <c:when> contains only one mandatory Attribute which is  
text.

(5)

### Switch.jsp

<%@ page isELIgnored

<%@ taglib prefix="c" uri=

<h1> Select the Number :

<form action="/jspl/switch.jsp">

<select name="combo">

<option value="1"> 1 </option>

<"> value="2"> 2 "

<"> value="3"> 3 "

<"> value="4"> 4 "

<"> value="5"> 5 "

<"> value="6"> 6 "

<"> value="7"> 7 "

</select>

<input type="submit" />

</form>

```
<c:set var="s" value="${param.combo}" />
```

Today is:

```
<c:choose>
```

```
<c:when test="${s==1}"> Sunday </c:when>
```

```
<" test="${s==2}"> Monday "
```

```
<" test="${s==3}"> Tuesday "
```

```
<" test="${s==4}"> Wednesday "
```

```
<" test="${s==5}"> Thursday "
```

```
<c:otherwise> Select between 1 & 5 </c:otherwise>
```

```
</c:choose>
```

```
<h1>
```

## Summary of Conditional Tag

|   | Tag                                       | Description                               | Attributes                                                                                                                  |
|---|-------------------------------------------|-------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| ① | <c:if>                                    | To implement Cose<br>Java if-stmt.        | test ; var, scope                                                                                                           |
| ② | <c:choose>,<br><c:when>,<br><c:otherwise> | To implement,<br>if-else &<br>switch stat | <c:choose> & <c:otherwise><br>doesn't contain any Attribute<br>but <c:when> should contain<br>only one Attribute i.e. test. |

## Iteration Tags

(55)

<c:forEach>

There are 2 Iteration Actions available in JSTL.

<c:forEach> - General Purpose for loop.

<c:forTokens> - Specialized for splitting strings similar to String Tokenizer.

<c:ForEach>

from=1 </@ Page isELIgnored="false" />

</.@ taglib prefix="c" uri=

<c:forEach begin="0" end="4" step="1" >

<hi> Learning JSTL is very Easy </hi>

</c:forEach>

Output - Learning JSTL is very Easy will be printed 5 times

→ begin="0" end="4" step="1" 5 times will be executed

→ begin="4" end="0" step="-1"

The begin Attribute Specifies the Index where the loop has to start.

End Index Specifies the index where the loop has to terminate.

This loop internally maintains Counter Variable which is incremented

by Step Attribute Value.

The Default Value for Step Attribute is "1" & it is optional

Attribute.

### Form-2 <c:foreach> with Var Attribute

- <c:foreach> internally maintains a counter variable which can be accessed by using Var Attribute.
- This variable is local Variable , from outside of forloop we can't Access.

```
<c:foreach begin="0" end="7" step="2" Var="counter">
 <h1>Learning JSTL is very Easy --- ${counter} </h1>
</c:foreach>
```

Output ->  
0  
2  
4  
6

### Form-3 <c:foreach> for Iterating through Arrays & Collections.

- <c:foreach items="collection Object" Var="current Object" >  
    ≡ Body

```
</c:foreach>
```

- items Attribute should contain either Collection Object or Array the Action will iterate over each item in the collection until all elements completion.

- we can represent current "Object" by using Var.

Type of items  
Attribute

corresponding Var Attribute

(56)

① Primitive Array

corresponding Wrapper class

Ex: int[] ————— Integer

double[] ————— Double.

② Object type Array

corresponding Declared Type

Ex: Student[] ————— Student

③ Collection

Object

④ Map

map. Entry

⑤ List of Strings

String.

Separated by :

Example

<!

String[] s = {"A", "B", "C", "D"};

The current object is : A

" " : B

" " : C

" " : D

/>

<c:forEach items="\${s}" var="obj" >

<hi> The Current Object is : \${obj} </hi>

</c:forEach>

~~QUESTION~~  
write a program to display all request headers by using <c:foreach>

```
<h1>
<table border="2">
 <c:foreach items="#{headers}" var="hds" >
 <tr>
 <td> ${hds.key} </td>
 <td> ${hds.value} </td>
 </tr>
 </c:foreach>
</table> </h1>
```

F V  
map.Entry → getKey  
get value  
set value.

→ To Display all Cookies Associated with the Request

```
<h1>
<table border="2">
 <c:foreach items="#{cookie}" >
 <tr>
 <td> ${x.value.name} </td>
 <td> ${x.value.value} </td>
 </tr>
 </c:foreach> </table> </h1>
```

key Value  
cookie name cookie object

on this particular  
object we have  
to get cookie  
name & cookie  
value.

→ if we call  
x.key    x.value  
SessionID    toString()  
will be called.

### Point 4 <c:foreach> with VarStatus Attribute

(57)

→ This Attribute describes the Status of Iteration like current iteration Number, is it first Iteration or not etc.

→ This Attribute is of type javax.servlet.jsp.core.TagLoopStatus

→ This class contains several methods which are useful during Iteration.

① Object getCurrent()

— Returns the current Item.

② int getIndex()

— Returns the current Index (counter value)

③ int getCount()

— Returns the no. of iterations that have already performed.

④ boolean isFirst()

— Returns true if the current Iteration is first Iteration.

⑤ boolean isLast()

— Returns true if the current Iteration is last Iteration.

⑥ Integer getBegin()

— Returns Start Index.

⑦ Integer getEnd()

— Returns Last Index.

⑧ Integer getStep()

— Returns Incremented value.

`<c:foreach items="durga, Parvati, xavi, shiva" varStatus="status">`

`<hi> Is it The First Iteration : ${status.first} </hi>`

`The Current Object is : ${status.current} </hi>`

`</c:foreach>`

`o/p: Is it First Iteration : true → false`

`The Current Object is : durga xavi`

`Is it First Iteration : false → false`

`The Current Object is : Parvati shiva`

509 `<c:forTokens>` It is a specialised version of foreach to perform string Tokenization based on some delimiters. It behaves exactly same as String Tokenization.

syntax `<c:forTokens items="list of strings" delims="separators.sequence" </c:forTokens>`

`<c:forTokens items="List of Strings" delims="separators.sequence">`

`<c:forTokens items="one, two, three, four" delims="," var="x" >`

`<hi> The currentToken is : ${x} </hi>`

`</c:forTokens>`

We can store the current Token by using Var attribute.

`<c:forTokens>` can Access (or) Accept the following extra Attributes also:

begin - Specifies the Index at which the iteration should start  
The Index of first Token is zero.

58

End - Specifies the Index where the iteration should terminate.

Step - Counter Increment Value b/w iterations.

varStatus - to specify the status of Iteration.

Ex-1

<c:forEach items="congress;TDP;PR;TRS" delims=";">

No Party is good.

O/P - No Party is good → 4 times

</c:forEach>

<c:forEach items="chiru,ntr,Pawan,roja" delims="," var=>

Cine Industry is too good then Politics : \${x}

</c:forEach>

<c:forEach items="one,two,three,four,five" delims="," var=>

begin="0" end="3" step="2" >

currentToken is : \${x}

.. O/P: currentToken is : one

current Token is : three.

</c:forEach>

In the case of <c:forEach> items Attribute should be

String only . But in the case of <c:forEach> the items Attribute can be collection, Array , Map or String . Hence <c:forEach> is considered as Specialized Version of <c:forEach>

## Summary of Iteration Tags

| <u>Tag</u>    | <u>Description</u>                     | <u>Attributes</u>                                    |
|---------------|----------------------------------------|------------------------------------------------------|
| <c:forEach>   | General purpose<br>for loop            | items, begin, end, step,<br>vars, VarStatus          |
| <c:forTokens> | Specialized for<br>String Tokenization | items, delims, begin,<br>end, step, vars, VarStatus. |

## URL Related Tags

### i) <c:import>

→ This is used for importing the response of other pages in to current page at Request Processing Time (i.e Dynamic Include)

#### Form-1

```
<c:import url="second.jsp" />
 ↓
 absolute | Relative.
```

c: import ⇒  
c: include.

#### first.jsp

```
<%@ taglib prefix="c" uri= ... %>
```

```
<%@ page isELIgnored = "false" %>
```

```
<c:import url="Second.jsp" />
```

```
<h1>This is First JSP </h1>
```

#### Second.jsp

```
<h1>This is Second JSP </h1>
```

O/P - This is Second JSP

This is First JSP

### Form-2

(59)

- we can import the resources from outside of current Application also (i.e cross context import is also possible).

```
<c:import url="/second.jsp" context="/webapplic_2"/>
 ↓
 Not Possible in Tomcat
 Absolute Path ↓
 Absolute Path.
```

First.jsp      JSP1      JSP2  
                  second.jsp    context.jsp

### Form-3

- we can store the result of imported page in to a variable specified by var Attribute. In the rest of the page where ever that result is required we can use directly the variable with out performing import once again.

```
<c:import url="Second.jsp" var="x" scope="page"/>
```

<%

```
String s = (String) pageContext.getAttribute("x");
```

```
String s1 = s.substring(0, 10);
```

%>

This is Second Jsp

This

```
<h1> The Result is : ${x};
```

The part of substring is : <%= s1 %>

### Form-4

- the more convenient way to store the result of <c:import> is to use Reader Object. It is alternative to Var Attribute.

```
<c:import url="second.jsp" var="Reader" myReader="myReader"
 scope="page" />
```

4.

```
Reader r = (Reader) PageContext.getAttribute("myReader");
int i = r.read();
while (i != -1)
{
 // Perform some operation on i
 o.out(i);
 i = r.read();
}
//
```

#### Form-5

While performing import we can send Form Parameters also to the se Target JSP. For this we have to use `<c:param>` Tag. These parameters are available as Form Parameters in the Target JSP.

Target JSP:

first.jsp

```
</@ taglib prefix="c" uri="---" %>
```

```
<h1> welcome to DSS </h1>
```

```
<c:import url="second.jsp" >
```

```
 <c:param name="P1" value="SCJP" />
```

```
 <c:param name="P2" value="SCWCD" />
```

```
</c:import>
```

## Second.jsp

(60)

<?@ page isELIgnored = "false" ?>

<h> The courses offered are : \${param.P1} and \${param.P2}

<c:redirect>

→ This action can be used to redirect the request to another page.

This is exactly similar to `SendRedirect()` of `ServletResponse`.

Form-1

<c:redirect url="Second.jsp" />

↓

Absolute or relative

Form-2

we can redirect the request to some other web applications resource also.

<c:redirect url="/Second.jsp" context="/webapplic2" />

↓

Compulsory Absolute Path

↓

Compulsory Absolute Path

Form-3

→ while performing redirection we can pass parameters also to Target Resource.

<c:redirect url="Second.jsp" >

<c:param name="x" value="10"/>

<c:param name="y" value="20"/>

</c:redirect>

<c:out>

we can use two standard Action to rewrite url's to append session information & from parameters.

form-1

```
<c:out value="Second.jsp" var="x" scope="request" />
→ modified url will be stored in x.
```

form-2

```
<c:out value="/Second.jsp" context="/webapp2" var="x"
 scope="request" />
```

form-3

```
<c:out value="Second.jsp" var="x" scope="request">
 <c:param name="p1" value="10" />
 <c:param name="p2" value="20" />
```

</c:out>

First.jsp

```
<c:out value="Second.jsp" var="x" >
 <c:param name="p1" value="10" />
 <c:param name="p2" value="20" />
```

</c:out>

<h1> The modified url is : \${x} </h1>

<a href="\${x}"> Click here to go to next page </a>

## Second.jsp

→ Disable cookies :

<?@ Page isEligible = "false" ?>

<hi> This is Second JSP

The Parameters are : \${param.p1} and \${param.p2}.

~~Summary of web related Tags.~~

| Tag          | Description                                                           | Attributes                          |
|--------------|-----------------------------------------------------------------------|-------------------------------------|
| <c:import>   | For importing the Response of other pages at Request processing Time. | url, var, scope, varReader, context |
| <c:redirect> | To Redirect the request to other web components.                      | url, context,                       |
| <c:out>      | To Rewrite url for appending session information & parameters (form). | value, var, scope context.          |
| <c:param>    | To send Parameters while importing & redirecting                      | name, value.                        |

## Design Pattern

A software Design Pattern is a repeatable solution for a commonly occurring software problem. It provides a standard process to design Application.

The following are most commonly used Design Patterns.

- 1) Transfer Object Design Pattern
- 2) Intercepting Filter
- 3) Front - Controllers
- 4) MVC
- 5) Business Delegate
- 6) Service Locators

### Data Transfer Object Design Pattern

Context - In Distributed Applications client & servers are located at remote locations and they have to communicate via network.

problem - Every call b/w Client & servers is a <sup>Remote</sup> method call.

If client Application calls individual getters & setters methods to receive & update Attribute Values. Then the no. of Remote calls required is as many as the no. of Attributes.

→ These individual remote calls generate lot of network traffic & Degrade Performance of the System.

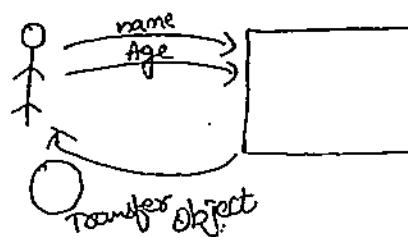
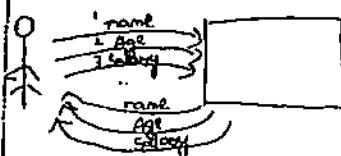
• Solution - create an Object to encapsulate all Attribute Values. that are required by Client Application. This Object is called Data Transfer Object.

(62)

→ When the Client request Data from Server, Server Side Component gathers Data Values & construct Transfer Object by setting its Data Values. This Object is then sent to the client.

→ Client use this Object to Query all the required Data Values.

→ The Transfer Object acts as Proxy for the Properties of Remote Object.



#### Consequences & Implications

→ Transfer Object DP reduces the network Traffic b/w Client & Server.

→ It minimises coupling (dependencies)

→ The main Advantage of this Design Pattern is we can prevent network overhead problems. & we can improve performance of the System.

→ The main limitation of this Design pattern is upto Date Information may not be available at client side.

## keyWords

1) Reduce Network Traffic

Improve Response Time

Transfer Data Across Tiers

Group Information

Intercepting Filter Design Pattern

Context — client Request may have many diff Processing units & System must be able to meet them.

1) The System has to receive the request by multiple protocols like http, ftp, SMTP ---

2) The System must Authorize & Authenticate each request.

3) The System needs to Add or Remove Info to request before further processing.

Problem — Request & Response needs to be Pre & Post Processed in a simple Organised Manner.

This Preprocessing needs to be performed with out affecting rest of the System i.e it should be pluggable.

Solution — Use Filters to Pre-Process The request & Post Processing of Response.

Configuring or Removing Filter takes places without affecting rest of the system.

### Consequences & Implications

(63)

- ① Intercepting Filters is based on cohesion, loose coupling, increase declarative control.
- ② Declarative Control allows Filters to be easily implemented either on temporary or permanent bases.
- ③ Declarative Control allows the sequence of invocations to be easily updatable.

### Key Words

Central Object for PreProcessing Request & Post-Processing Response.

Authentication of each request.

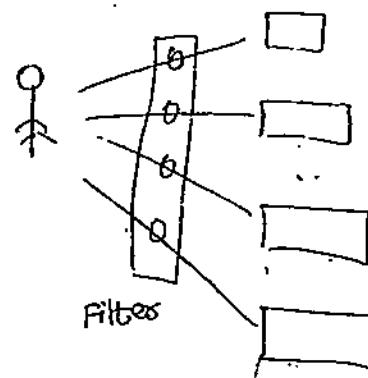
Authorization of each request.

Logging Request Information.

Composition of Response.

Encryption of Response.

Altering Request & Response Information.



Q9

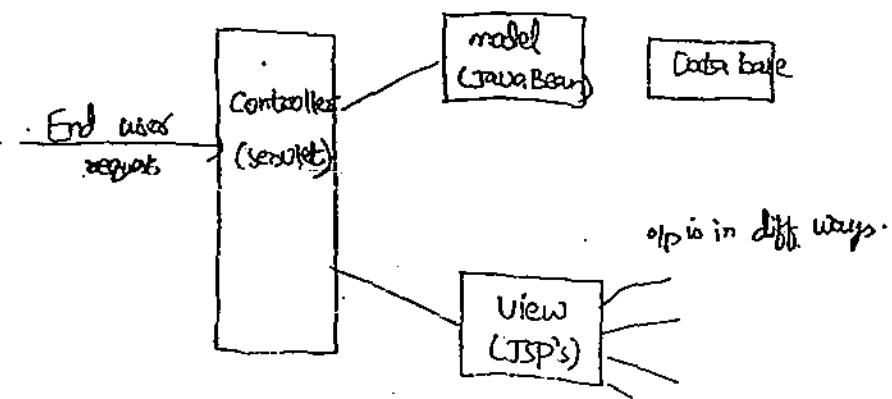
## MVC Design Pattern (Model View <sup>Controller</sup> Design Pattern)

Context — In the Systems involving user Interfaces, the following situation typically arises.

- ① The system has to accept the Data from the user
- ② Update Database and return the same Data to the user at later time. There are several ways in which the Data can be accepted and several ways to present Data to the user.
- ③ Data stored in the System in one form should be retrievable in another form.

Problem — If the System Deploys a Single Component that interacts with user and maintains Database, then a requirement to support a new type of Display or View will requires Re-Design of entire component.

Solution — The Solution is to separate Data Presentation from Data maintenance and have a third component that co-ordinate first two. These 3 components are called model, view & controller.



model - It is responsible to keeping the Data or state of the application. It also manages the storage and retrieval of Data from Data Source.

(64)

View - It contains presentation logic. It displays Data from the model to the end user. It also allows the user to interact with the system.

Controller - It manages whole show. It instantiates the model & view, and associates view with the model. Depending on application requirements it may instantiate multiple views for the same model.

#### Consequences or implications

It provides Job Separation. Hence without affecting remaining components we can modify any component.

It improves code maintainability.

It supports loose coupling & high cohesion.

for the same Data we can provide multiple views.

It promotes Re-usability of model components.

The main limitation of this Design Pattern is it increases the overall complexity of the application & it is not suitable for small scale applications.

#### Key words

Separation Of Data Presentation from Data Maintenance.

Provide services to different clients, web clients, wap client

; multiple views such as .html, .xml

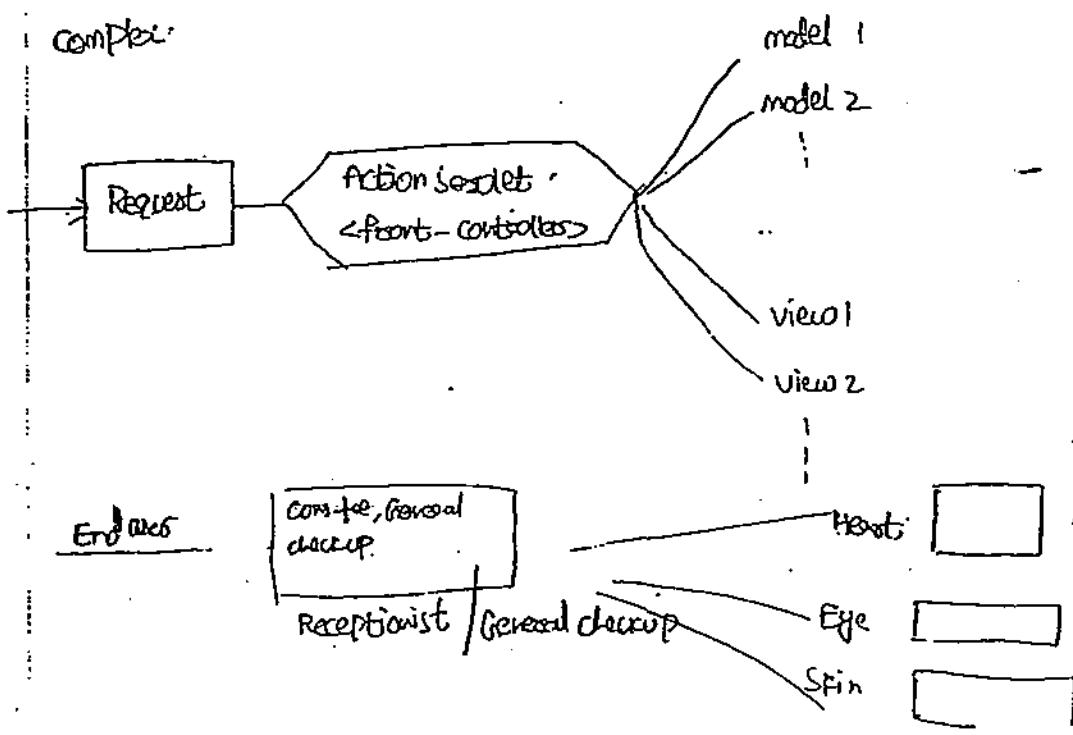
) Single Controllers

### Front Controller Design Pattern

Context — The presentation tier request handling mechanism must control and coordinate processing of each user across multiple requests. Such control mechanism may be managed either in centralized or de-centralized manner.

Problem — When the user is accessing view component directly without going through a centralized mechanism each view is required to provide its own system services which may result in duplicate code.

Solution — Use Front Controller Design Pattern to gather common redundant request processing code in to a single component. This allows the application to be more cohesive and less complex.

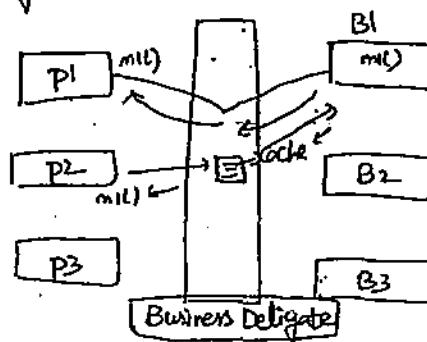
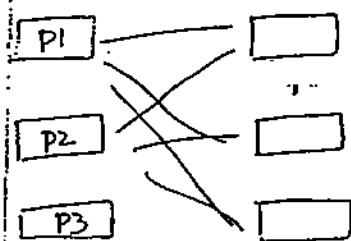


Consequences, implications keywords.

(65)

- ① Centralized control of request processing.
- ② improves maintainability
- ③ promotes Reusability.

### Business Delegate Design Pattern



Context - A multi-tiered Distributed System requires remote method invocations to send and receive Data across Tiers. clients are exposed to the complexity of dealing with distributed components.

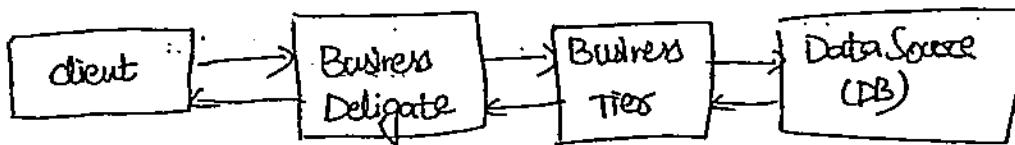
Problem - Presentation Tiers components interacts directly with Business Services. This direct interaction exposes the underlying implementation details of Business service to the presentation tiers. As a result if there is any change in business service it requires to modify all presentation tier components.

- There may be network overheads bcz Presentation Tiers Components make too many invocations over the network and presentation tier components won't maintain any caching.
- Presentation Tiers components are responsible for network issues also.

plation - we use a business Delegate to reduce coupling b/w presentation Tiers and business Tiers.

Business Delegate hides implementation Details of Business Service

→ Business Delegate acts as client-side business proxy



If there is any change in business Service its not require to perform any change in presentation-Tiers components.  
we have to perform that change only in Business Delegate.

Presentation Tiers components are not responsible to handle any network issues. Bcz it is the responsibility of Business Delegate.

Business-Delegate can maintain <sup>re-instances</sup> caching so that we can reduce network traffic and improves performances.

It adds complexity bcz of extra Tiers Business Delegate

It decreases flexibility.

The client is restricted to the options provided by Business Delegate.

## Service Locator Design Pattern.

(6)

- Context - Service look up and creation involves complex interfaces and network operations.
- EJB clients needs to use JNDI API to look-up EJB Home object
- JMS clients needs to use JNDI API to look-up JMS Components such as connection factories, Queues and Topics.
- look-up and creation of service components could be complex and may use repeatedly in multiple client components.
- Problem - Every client component has to contain code for look-up and creation of service component. As the same code present in several places it is difficult to change everywhere if there is any change in service component.
- Every call to look-up process is an individual network call. It may increases network traffic and affects performance.  
bcz at client side we are not maintaining any caching.
- Solutions - use a service locator object to abstract all JNDI usage and to hide complexities of initial context creation, EJB Home object look-up and EJB object creation.
- multiple clients can reuse the same Service Locator object to reduce complexity.
- Providing a single point of control for look-up processes is possible through Service Locator Object.

It can reduce network overhead and improve performance by maintaining cache.



- 1) Client invokes singleton instance of locator.
- 2) locator creates and initial context for service look-up
- 3) Initial context obtain EJB Service Object.
- 4) locator returns that Service Object to client.

#### Consequences or implications and keywords

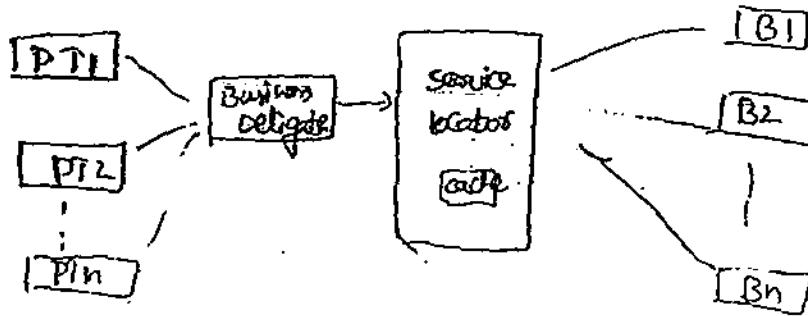
It abstracts complexity

Provides uniform Service Access for all clients.

Facilitates adding new business components.

Improves performance and reduces network overhead.

It supports loosely coupling by reducing the dependency b/w presentation tiers and business tier components.



## Design Pattern

## Application Tier

(67)

① Data Transfer Object

Business Tiers (Remote Host sources)

② Intercepting Filter

Presentation Tiers (Servlets, JSP)

③ Front controllers

"

④ MVC

Presentation & Business

⑤ Business Delegate

Business Tier

⑥ Service Locator

Business Tier

web.xml Configurations

memoising EJB Related Tags:

Reference to a local Bean

<ejb-local-ref>

<ejb-ref-name> ejb/customers </ejb-ref-name>

<ejb-ref-type> Entity </ejb-ref-type>

local <local-home> com.dusqa.CustomerHome </local-home>

Remote <local> com.dusqa.Customer </local>

<ejb-local-ref>

JNDI look-up name

<ejb-ref-name>

<ejb-ref-type>

<local-home>

</local>

These must be fully qualified names

of beans exposed interfaces.

<ejb-ref>

<ejb-ref-name>

<ejb-ref> but not

<ejb-ref-type>

<ejb-remote-ref>

<home>

<home> but not <remote-home>

<remote>

<ejb-ref>

memorising JNDI <env-entry> Tag

look-up name we will use in code

<env-entry>

↑

<env-entry-name> rates / discountRate <env-entry-names>

<env-entry-type> java.lang.Integer <env-entry-type>

<env-entry-value> 10 <env-entry-value>

<env-entry>

, Standard Actions

① <jsp:useBean>

<jsp:include> <RD id=xxx>.getRD()

<jsp:forward> id.forward();

) EL

② <%>

String context = getServletContext().getInitParameter("prod")

) JSTL

<%>

<c:if test="#{initParam.prod} >

in Java and make that

{initParam.prod}

functionality available to JSP.

~~Object~~

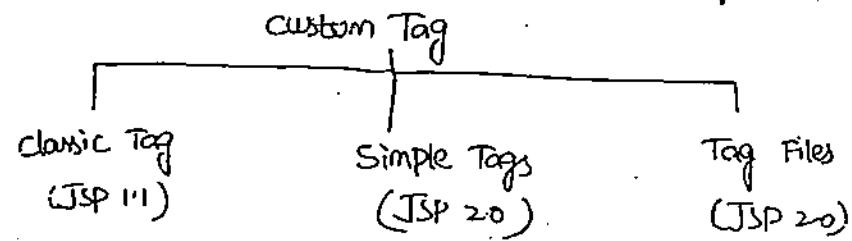
## Custom Tags

(68)

→ Standard Actions, EL & JSTL are not sufficient completely to separate Java code from the JSP.

Ex: sometimes our requirement has to communicate with DB or with EJB. There is no standard Action define for this requirement. we can provide our own custom Tags to meet our requirement from JSP 1.1 version onwards. i.e we can define our own Tags to provide required functionality which are nothing but custom Tags.

→ All Custom Actions are divided into 3 categories



### Components Of Custom Tag Application

→ Custom Tag Application Contains the following 3 components.

① Tag Handler - It is a simple Java class which defines entire required functionality.

→ Every Tag Handler class should compulsorily implement Tag interface either directly or indirectly.

→ web container is responsible for creation of Tag Handler object. For this it always invokes Public no-Argument constructor. Hence every Tag Handler class should compulsorily

contain public no-Argument constructor.

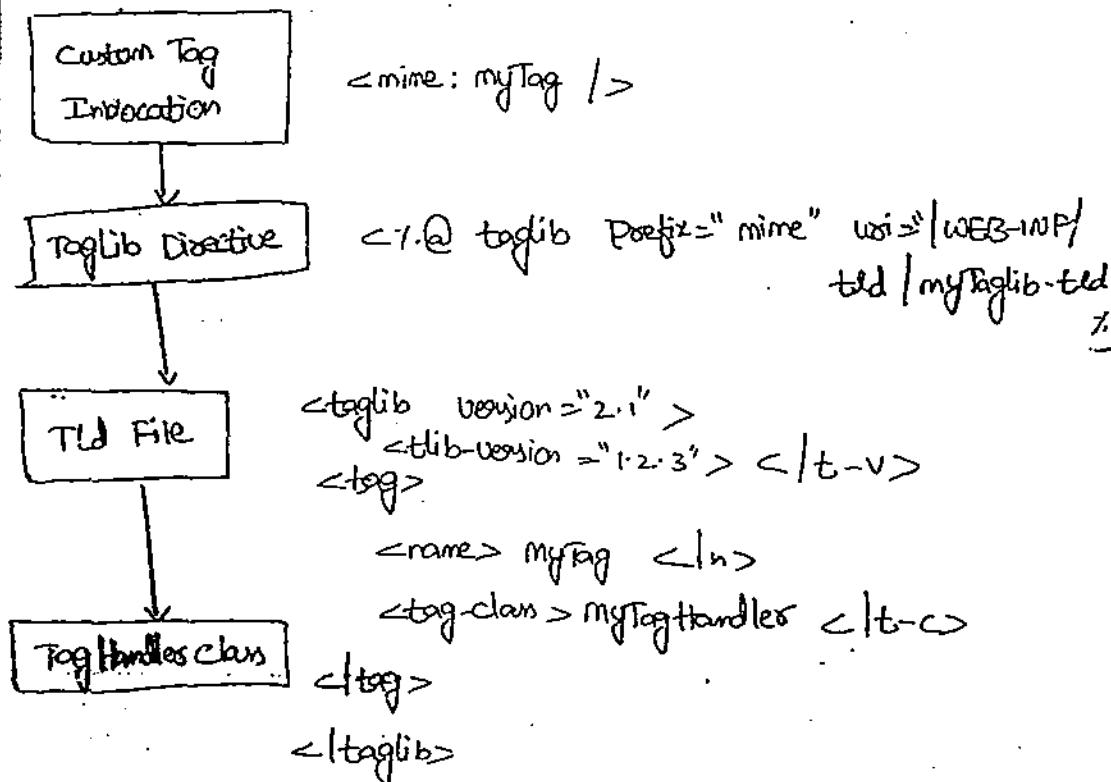
### 1) Tld file - ( Tag Library Descriptor File)

It is an xml file which provides mapping b/w JSP (where customTag functionality is required) and Tag Handler class (where customTag functionality is available).

### 2) Taglib Directive

It makes customTag functionality available to the JSP. It defines the location of the Tld File.

Work Flow of Custom Tag Application



- ① whenever JSP Engine encounters a customTag , it identifies 69  
Prefix and checks for the corresponding Taglib Directive with  
matched prefix.
- ② From the Taglib Directive JSP Engine Identifies the location of  
TLD File.
- ③ From TLD File JSP Engine identifies the corresponding Tag-Handler  
class.
- ④ JSP Engine Executes Tag Handler class and provides required  
functionality to the JSP.

#### Tag Extension API

- we can build customTags by using only one package  
`javax.servlet.jsp.tagext`.
- This package contains the following Interfaces and classes to  
build customTags.
- ① Tag :- It is the base Interface for all customTag Handlers.  
Every Tag Handler class should compulsorily implement this interface  
either directly or indirectly.
  - This interface Defines 6 methods which can be applicable for  
any Tag Handler Object.
  - we should go for this interface if we are not manipulating Tag  
body and iteration is not required.

### Iteration Tag

- It is the child Interface of Tag. we should go for this interface whenever iteration is required.
- It contains one extra method i.e doAfterBody().

### Body Tag

- It is the child Interface of Iteration Tag. we should go for Body Tag whenever Tag Body manipulation is required.
- This interface defines the following 2 extra methods.  
setBodyContent() and doInitBody().

### i) TagSupport class

- It implements Iteration Tag Interface and provides default implementation for all its methods.
- It acts as a base class to Provide (or) Develop Iteration & Simple Tags.

- It acts as Adapter for Tag and Iteration Tag Interfaces.

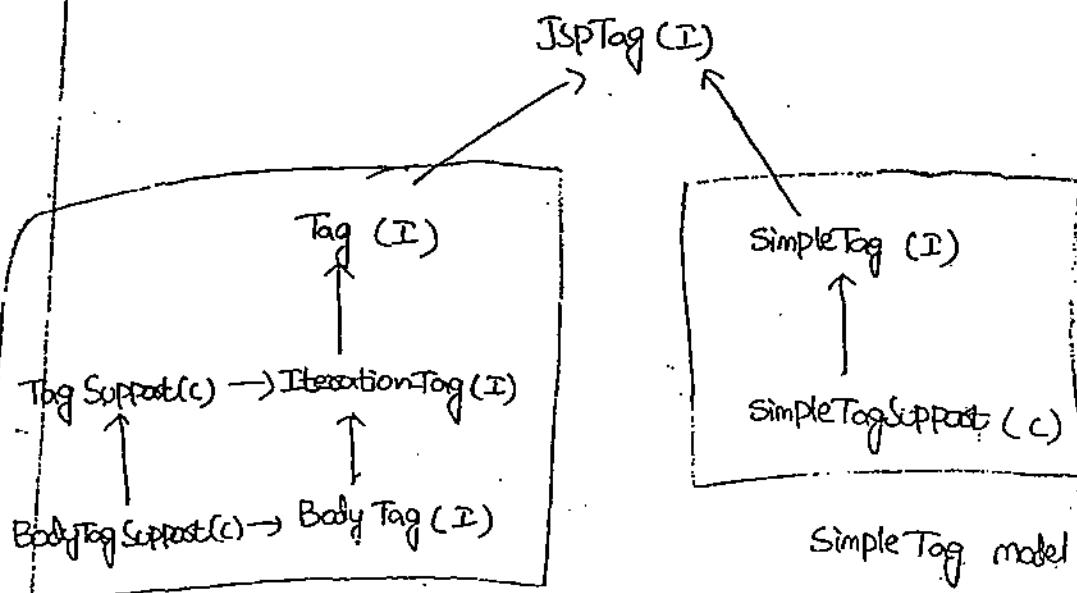
### ii) BodyTagSupport class

- This class implements BodyTag Interface and provides default implementation for all its methods.
- It acts as a Adapter class for BodyTag.
- we can use this as Base class for implementing Custom Tag that processes Tag Body.

(70)

### ⑥ Body Content class

- Body Content Object acts as a buffer for Tag body.
- It extends JspWriter.
- we have to use BodyContent class only in BodyTag Interface and BodyTagSupport class.



classic Tag model

JSP 1.1

Simple Tag model

JSP 2.0

- JSPTag Interface is just for Polymorphism purpose and doesn't contain any methods.

## Implementing Tag (I)

It defines the following 6 methods

setPageContext()

setParent()

doStartTag()

doEndTag()

release()

getParent()

Tag Interface Defines the following Constants also

EVAL\_BODY\_INCLUDE

SKIP\_BODY

EVAL\_PAGE

SKIP\_PAGE

Lifecycle of Tag Handler that implements Tag Interface.

Whenever JSP Engine Encounters a CustomTag it will identify the corresponding Tag Handler class through Tag lib directive and tld files.

web Container creates an Instance of Tag Handler by executing Public no argument Constructor (if it is not already available).

JSP Engine calls setPageContext() to make Page Context of Jsp available to Tag Handler class.

Public void setPageContext(PageContext p)

→ JSP Engine calls setParent() to make ParentTag object available to TagHandlers. This method is helpful in nested Tags. (71)

Public void setParent(Tag t)

→ setting Attributes - Attributes in custom Tags are exactly similar to Properties of Java Beans. If a custom Tag has an Attribute then compulsory TagHandler class should contain one Instance Variable and the corresponding Setters methods.

→ JSP Engine calls Setters methods for each Attribute.

→ JSP Engine calls doStartTag()

Public int doStartTag() throws JSPException.

→ we can define entire Tag functionality in this method only.

→ doStartTag() can return either EVAL-BODY-INCLUDE or SKIP-BODY.

→ If it returns EVAL-BODY-INCLUDE, the tag body will be included in result.

→ If it returns SKIP-BODY, JSP Engine doesn't consider Tag Body.

→ JSP Engine calls doEndTag()

Public int doEndTag() throws JSPException.

→ doEndTag() can return either EVAL-PAGE or SKIP-PAGE.

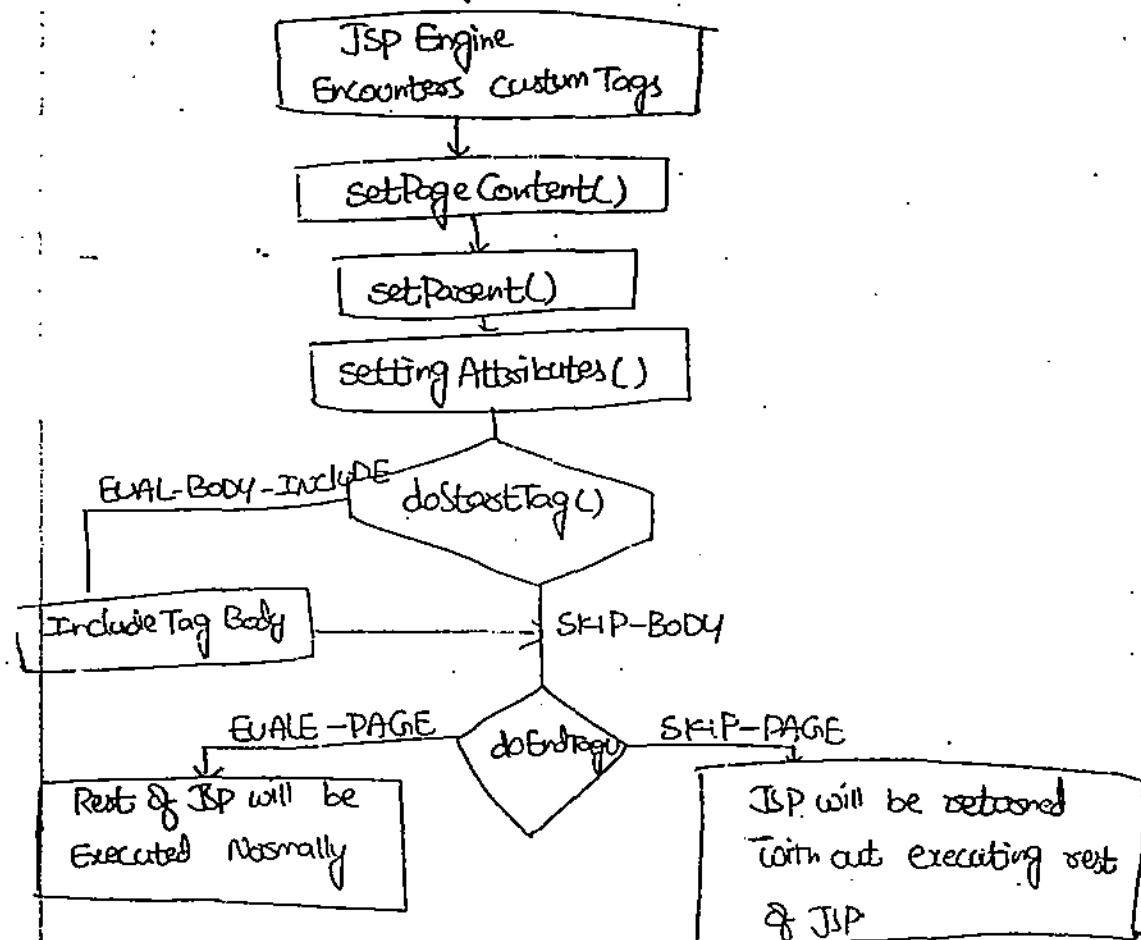
→ If it returns EVAL-PAGE, rest of the JSP will be executed normally.

→ If it returns SKIP-PAGE, JSP Page will be returned with out executing rest of the JSP.

Finally JSP Engine calls `release()` to perform clean-up Activities when ever Tag Handler Object no longer required.

```
public void release()
```

Flowchart for Tag Handler LifeCycle.



72

## Demo Program

### test.jsp

```
<!--@ taglib prefix="mine" uri="/WEB-INF/myTld.tld" /-->
<hi> Hello ! This is Demo Jsp </hi>
<mine:mytag>
 This is body of customTag
</mine:mytag>
<hi> This is After Custom Tag </hi>
```

### myTld.tld

```
<taglib version='2.1'>
<tlib-version>1.2.3</tlib-version>
 <uri> www.durgalab.com </uri>
 <tag>
 <name> mytag </name>
 <tag-class>tags.myTagHandler </tag-class>
 </tag>
</taglib>
```

### MyTagHandler.java

```
Package tags;
Import javax.servlet.jsp.tagext.*;
Import javax.servlet.jsp.*;
Import java.io.*;
```

Public class MyTagHandler implements Tag

Private PageContext Pcontext;

Public void setPageContext (PageContext Pcontext)

{ this.Pcontext = Pcontext;

}

Public void setParent (Tag t) { }

Public int doStartTag () throws JSPException

{ tag =>

{ JspWriter out = Pcontext.getOut();

out.println ("Hello! This is from Tag Handler");

}

catch (IOException e) { }

return EVAL\_BODY\_INCLUDE;

}

Public int doEndTag () throws JSPException

{

return EVAL\_PAGE;

}

Public void getParent()

{ return null;

}

Public void release() { }

→ If doStartTag() returns SKIP\_BODY & doEndTag() returns SKIP\_PAGE then the output is

(73)

Hello ! This is Demo Jsp

Hello ! This is from Tag Handler class

→ Instead of hard coding the location of tld file in jsp we can specify through web.xml also

Example

```
<!--@ taglib prefix="mime" uri="www.duogalsoft.com" /-->
```

web.xml

```
<web-app>
```

```
 <jsp-config>
```

```
 <taglib>
```

```
 <taglib-uri> www.duogalsoft.com </taglib-uri>
```

```
 <taglib-location> /WEB-INF/myTLD.tld </taglib-
```

```
 </taglib>
```

location

```
 </jsp-config>
```

```
 </web-app>
```

We can map Tag lib directly to the tld File by using uri Attribute.

```
<!--@ taglib prefix="mime" uri="www.duogalsoft.com" /-->
```

tld file

```
 <taglib version="2.1" >
```

<tlib-version> 1.2.3 </tlib-version>

<uri> www.durgsoft.com </uri>

<tag>

<name>

Directly place in WEB-INF

<tag-class>

<tag>

<taglib>

This feature may not be supported by all web servers.

Structure of Tld File

<taglib version="2.1">

<tlib-version> 1.2.3 </tlib-version>

<uri> www.durgsoft.com </uri>

<tag>

<description>

<name> mytag </name> Fully Qualified Name

<tag-class> tags.MyTagHandler </tag-class>

<body-content> XXXX </body-content>

↳ empty / tag dependent / scriptlets

<attribute>

⇒ Default value is JSP

JSP

<name>

<required> true/false

<exposure>

<attribute>

</tag>

body-content - It describes the type of content allowed inside tag body. The allowed values are

(74)

① empty - The body of the Tag Should be empty. we can't take any tag body. In this case we can invoke customTag as follows.

• <mime:myTag /> or <mime:myTag>  
</mime:myTag>

② tagdependent - Entire Tag body will be Treated as plain text.

JSP Engine Sends TagBody to the Handles class with out any Processing.

③ scriptless - Tag Body Should not Contain any Scripting Elements (i.e scriptlet, Expression) But Standard Actions & EL Expressions are allowed.

④ JSP - No restrictions on Tag Body. whatever is allowed in JSP is by default allowed in TagBody also.

⇒ The Default Value is JSP

Attributes

A Tag can contain any no. of Attributes we can declare these Attributes by using <Attribute> tag in tld.

<Attribute> Contains the following child Tags.

① <name> - name of the Attribute.

② <required> - true - if Attribute is mandatory  
- false - " " " optional - false  
Default value is

### 3) <@expeditable> runtime Expansion Value

true - RuntimeExpressions also allowed.

Default  
Value  
is  
false

Ex: <mime:mytag color="\${param.color}" />

false - RuntimeExpressions are not allowed and we have to provide only literals.

Ex: <mime:mytag color="red" />

Demo Program for empty custom tag with <sup>mandatory</sup> Attribute

- A tag can contain Attributes also. For each Attribute we have to do the following things.
  - (1) we have to declare that Attribute is tag by using <attribute> Tag.
  - (2) For each Attribute in TagHandlers class we have to define one instance variable and corresponding setter method.
  - (3) In the case of optional Attributes there may be a chance of null pointer exceptions. we have to handle carefully.

### test.jsp

```
<?@ taglib prefix="mime" uri="/WEB-INF/myTLD-file.tld" ?>
```

```
<mime:myTag1 numbers="2" />
<mime:myTag1 numbers="5" />
```

(75)

### myTLD.tld

```
<taglib version="2.1" >
```

```
 <tlib-version> 1.2.3 </tlib-version>
```

```
 <tag >
```

```
 <name> myTag </name>
```

```
 <tag-class> tags.MyCustomTags </tag-class>
```

```
 <attribute >
```

```
 <name> numbers </name>
```

```
 <required> true </required>
```

```
 </attribute >
```

```
 </tag >
```

```
</taglib>
```

### MyCustomTag.java

package tags;

```
import javax.servlet.jsp.tagext.*;
```

```
import javax.servlet.jsp.*;
```

public class MyCustomTag implements Tag

{ private int numbers;

PageContext pcontext;

public void setPageContext (PageContext pContext)

{ this.pContext = pContext;

}

public void setParent (Tag t) { }

public void setNumber (int numbers)

{ this.numbers = numbers;

}

public int doStartTag () throws JspException

{ tag

{ JspWriter out = pContext.getOut();

out.print ("<hi> Double of " + numbers + " is: " +

(2 \* numbers) "</hi>");

}

catch (IOException e) { }

return EVAL\_BODY\_INCLUDE;

}

public int doEndTag () throws JspException

{ return EVAL\_PAGE;

Public void getPresent()

{  
    return null;  
}

Public void release()

Demo program for Empty Tag with optional Attribute

test.jsp

<!,@ taglib

<mime:mapping greetings name="durga" />  
<mime:greetings />

tld

<attribute>

<name> greetings </name>

<required> false </required>

<attribute>

myCustomTag.java

Public class MyCustomTag implements Tag

{  
    Private String name;

    Public void setName (String name)

{  
    this.name = name;

}

(76)

```
public int doStartTag() throws JspException
{
 try
 {
 JspWriter out = pContext.getOut();
 if(name==null)
 {
 o.open("Hello Guest Good Morning");
 }
 else
 {
 o.open("Good Morning " + name);
 }
 }
 catch(IOException e) { }
 return EVAL_BODY_INCLUDE;
}
}
```

## Iteration Tag

(77)

- Iteration Tag is the child interface of Tag. If we want to include Tag Body multiple times then we should go for Iteration Tag.

→ It contains only one extra method doAfterBody() and one extra constant EVAL-BODY-AGAIN.

doAfterBody()

Public int doAfterBody() throws JSPException

1) This method is invoked after doStartTag() and can return either SKIP-BODY or EVAL-BODY-AGAIN

2) If it returns SKIP-BODY the body will be skipped and JSP Engine calls doEndtag().

3) If it returns EVAL-BODY-AGAIN then the body will be considered once again followed by execution of doAfterBody().

Flowchart for

JSP Engine Encounters Custom Tag

↓  
setPageContext()

↓  
setParent()

↓  
setting Attributes

EVAL-BODY-INCLUDE

Evaluate body

doStartTag()

SKIP-BODY

EVAL-BODY-AGAIN

doAfterBody(): SKIP-BODY

↓  
EVAL-PAGE

doEndTag()

SKIP-PAGE

↓  
Rest of JSP will be  
Processed Normally

↓  
JSP return without  
executing rest of Page

Java Program for Iteration Tag

<!,@ taglib prefix="mime" uri="/" />

count=0  
count=-2 } no o/p

<mime:loop count="9">

<h> learning CustomTags is very Easy -- </h>

</mime:loop>

tld

<taglib version="2.1">

<tlib-version>1.2.3 </tlib-version>

<tag>

<name> loop </name>

<tag-class>tags.myCustomTag </tag-class>

<attribute>

<name> count </name>

<required> true </required>

<attribute>

</tag> -

</taglib>

myCustomTag.java

Package tags;

import

import

(78)

```
Public class MyCustomTags implements IterationTag
{
 Private int count;
 PageContext pcontext;
 Public void SetCount(int count)
 {
 this.count = count;
 }
 Public void setPageContext(PageContext pcontext)
 {
 this.pageContext = pcontext;
 }
 Public void setParent(Tag t)
 {
 }
 Public int doStartTag() throws JspException
 {
 If (count > 0)
 return EVAL_BODY_INCLUDE;
 Else
 return SKIP_BODY;
 }
 Public int doAftertag() throws JspException
 {
 If (--count > 0)
 return EVAL_BODY_AGAIN;
 Else
 return SKIP_BODY;
 }
}
```

```
public void doEndTag() throws JSPException
{
 return EVAL_PAGE;
}

public void getParent()
{
 return null;
}

public void release()
{
}
```

### Tag Support Class

The main draw back of implementing Tag and Iteration Tag Interfaces is we have to provide implementation for all methods even though most of times we have to consider doStartTag(), doEndTag() and doAfterBody().

- > we can resolve this Problem by using Tag Support class.
- , TagSupport class implements Iteration Tag and provides default implementation for all its methods.
- > It is easy to extend TagSupport class and override required methods instead of implementing all methods.

Implementation of TagSupport class

(79)

public class TagSupport implements IterationTag, Serializable

{

    public PageContext PageContext;

    private Tag t;

    public void setPageContext(PageContext p)

    {

        PageContext = p;

    }

    public void setParent(Tag t)

    {

        this.t = t;

    }

    public int doStartTag() throws JSPException

    {

        return SKIP\_BODY;

    }

    public int doAfterBody() throws JSPException

    {

        return SKIP\_BODY;

    }

    public int doEndtag() throws JSPException

    {

        return EVAL\_PAGE;

    }

    public Tag getParent()

    {

        return t;

    }

```
public void release()
{
 t=null;
 pageContext=null;
}
```

#### Note

- 1) The default return type of doStartTag() and doAfterBody()  
is SKIP-BODY.
- 2) The default return type of doEndTag() is EVAL-PAGE.
- 3) pageContext variable is by default available to the child class  
Hence we can use this var directly in our Tag Handler  
class.

#### Example

```
public class MyCustomTag extends TagSupport
{
 public int doStartTag() throws JSPException
 {
 try
 {
 JSPWriter out = pageContext.getOut();
 out.println("<hi>This is from 'CustomTag Handler class' </hi>");
 }
 catch (IOException e) { }
 }

 public void doAfterBody()
 {
 pageContext
 by default
 available
 through
 inheritance
 of TagSupport
 return EVAL-BODY-INCLUDE;
 }
}
```

## last JSP

(30)

↳ (a) taglib prefix = "mime" uri = " one time will be executed  
<mime:mytag />

## Body Tag Interface

→ It is the child Interface of Iteration Tag. If we need to manipulate Tag Body then we should go for Body Tag Interface.

→ Body Tag Interface defines the following 2 extra methods.

- ① public void setBodyContent (BodyContent b)
- ② public void doInitBody () throws JSPException

→ Body Tag Interface defines the following 2 extra constants.

① EVAL\_BODY\_BUFFERED

② EVAL\_BODY\_TAG ⇒ Deprecated in JSP 1.2 V  
(JSP 1.1)

## Body Content

→ It is an Abstract class and extends JSPWriter

→ BodyContent Object represents the body of the Tag.

→ BodyContent class defines the following methods to retrieve

## Tag Body

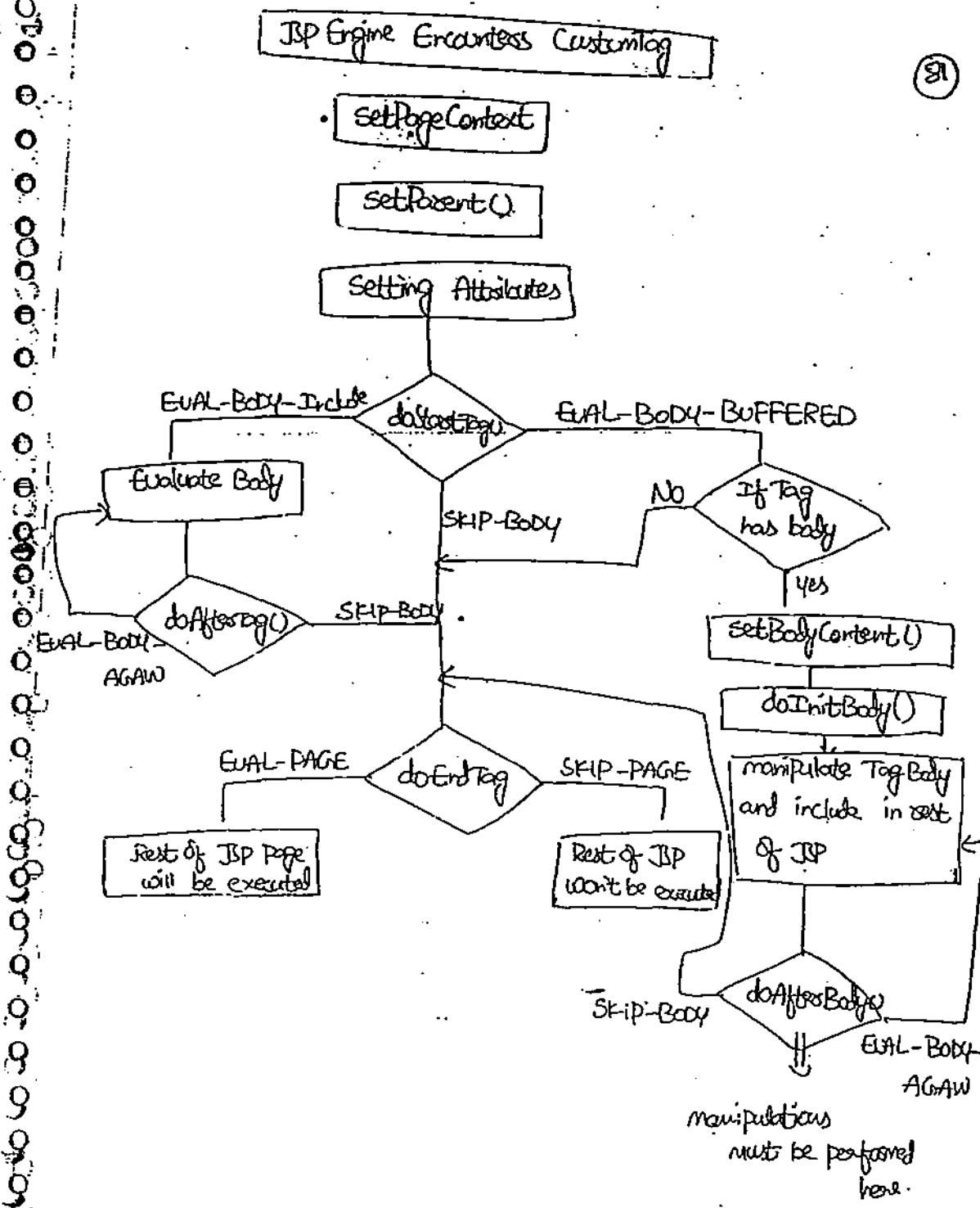
- ① public String getString () — returns body content as String.
- ② public Reader getReader () — returns a Reader Object to extract Tag Body.

- | JSPWriter getEnclosingWriters()
  - Returns JSPWriter Object of Parent class.
  - If there is no parent then it returns current JSP out object.

- | Public void clearBody()
  - It clears body Content Object i.e. the data present in body content object will be removed.

### LifeCycle of Body Tag

- The Life Cycle JSP of Body Tag Handler is exactly similar to Iteration Tag Handler. The difference is doStartTag() can return EVAL-BODY-BUFFERED in addition to EVAL-BODY-INCLUDE and SKIP-BODY.
- If the method returns EVAL-BODY-INCLUDE then body will be evaluated exactly similar to Iteration Tag.
- If method returns EVAL-BODY-BUFFERED and Tag contains body then JSP Engine creates BodyContent Object and call setBodyContent() followed by doInitBody().
- The setBodyContent() and doInitBody() won't be executed in following cases:
  - If doStartTag() returns either EVAL-BODY-INCLUDE or SKIP-BODY or if the doesn't have the Body.



## BodyTag Support class

This class extends Tag Support class and implements BodyTag

Interface.

This class provides default implementation for all 9 methods available in Body Tag Interface.

It is very easy to implement Tag Handler by Extending BodyTag Support class instead of implementing BodyTag interface directly. In this case we have to override only required methods instead of implementing all 9 methods.

public class BodyTagSupport extends TagSupport implements BodyTag

{

    public PageContext pageContext;

    public BodyContent bodyContent;

    public Tag t;

    public void setPageContext (PageContext p)

{

        pageContext = p;

}

    public void setParent (Tag b)

{

        this.t = b;

}

    public void init () throws JspException

{

        return EVAL\_BODY\_BUFFERED;

```
Public int doAfterBody() throws JSPException (82)
{
 return SKIP_BODY;
}

Public int doEndTag() throws JSPException
{
 return EVAL_PAGE;
}

Public void setBodyContent (BodyContent bodyContent)
{
 this.bodyContent = bodyContent;
}

Public void doInitBody() throws JSPException { }

Public Tag getParent()
{
 return t;
}

Public void release() { }

}

PageContext p, BodyContent c are by default available to
the Tag Handler class.

The default return type of doStartTag() is EVAL_BODY_BUFFERED
doAfterBody() is SKIP_BODY, doEndTag() is EVAL_PAGE.
```

```
<@ taglib prefix="mine" uri="/" />
<mine:body>
<h1> Learning custom Tags is very Easy -- <h1>
</mine:body>
<mine:body>
<h1> Durga Software Solutions <h1>
</mine:body>
```

### tld

```
<tld>
 <tag>
 <name>
 <tagClass>
 <bodyContent> JSP </bodyContent>
 </tag>
```

### myCustomTag.java

```
public class myCustomTag extends BodyTagSupport
{
 public int doAfterBody() throws JSPException
 {
 try
 {
 String s = bodyContent.getString();
 s = s.toUpperCase();
 JspWriter out = bodyContent.getEnclosingWriter();
 out.println(s);
 }
 }
}
```

```
 catch (IOException e) { }
```

```
 return SKIP_BODY;
```

```
}
```

(23)

Comparison b/w TagSupport and BodyTagSupport classes.

method      TagSupport

BodyTagSupport

(1) doStartTag()      EVAL-BODY-INCLUDE

EVAL-BODY-INCLUDE

(2) Possible Return  
Values

SKIP-BODY

SKIP-BODY

EVAL-BODY-BUFFERED

(3) Default Value  
from implementation  
class

SKIP-BODY

EVAL-BODY-BUFFERED

(4) no. of times it can  
be called Per Tag

only once

only once

(5) doAfterBody()

EVAL-BODY-AGAW

EVAL-BODY-AGAW

(6) Possible Return  
values

SKIP-BODY

SKIP-BODY

(7) default value  
from implementation  
class

SKIP-BODY

SKIP-BODY

(8) no. of times it can be  
called Per Tag

0 or more

0 or more

|                                                                        |                        |                                                                                                  |
|------------------------------------------------------------------------|------------------------|--------------------------------------------------------------------------------------------------|
| ③ doEndTag()                                                           |                        |                                                                                                  |
| ④ Possible return values.                                              | EVAL-PAGE<br>SKIP-PAGE | EVAL-PAGE<br>SKIP-PAGE                                                                           |
| ⑤ Default Value from implementation class                              | EVAL-PAGE              | EVAL-PAGE                                                                                        |
| ⑥ no of times it can be called per Tag                                 | once                   | one                                                                                              |
| ⑦ doInitBody () & setBodyContent.                                      | Not Applicable         | Executed only once if and only if doStartTag() returns EVAL-Body-Buffered and Tag contains Body. |
| → circumstances underwhich these can be called and no of times per Tag |                        |                                                                                                  |
| Invocation                                                             |                        |                                                                                                  |

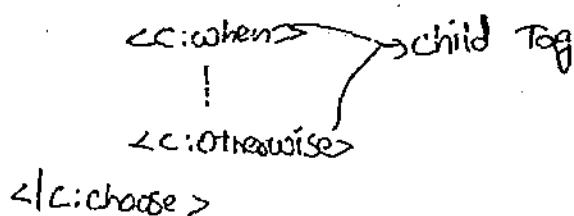
### Cooperative or Nested Tags

→ Sometimes a group of tags work together to perform required functionality. Such type of tags are called co-operative or nested tags.

Ex: In JSTL `<c:choose>`, `<c:when>` and `<c:otherwise>` tags work together to implement Java's switch stmt.

These tags are called co-operative tags.

• `<c:choose>` Parent Tag



~~83~~ <mine:outerTag>  
    <mine:InnerTag />  
    </mine:OuterTag>

(84)

From InnerTag (child Tag) we can get Parent Tag reference by  
using getParent().

Tag Parent = getParent();

Demo Program for nested Tags

first JSP

<? taglib prefix="mine"  
    <mine:myTag> — 0  
        <mine:myTag> — 1  
            <mine:MyTag> - 2  
                <mine:MyTag /> - 3  
        </mine:myTag>  
    </mine:myTag>  
    <mine:myTag>

tld File

<taglib version="2.1">  
    <tlib-version>1.2 </t-v>  
    <tag>  
        <name> MyTag  
        <tag-class> tags.myCustomTag </t-c>  
        <body-content> JSP </b-c>  
    </tag>  
</taglib>

### MyCustomTag.java

```
public class Mycustomtag extends TagSupport
{
 public int doStartTag() throws JspException
 {
 int nestedLevel = 0; Nested level is : 0
 Tag t = getParent(); " " : 1
 while (t != null) " " : 2
 {
 nestedLevel++; " " : 3
 t = t.getParent();
 }
 tag
 {
 PageContext pageContext = getOut();
 pageContext.getOut().println("<hi> Nested level is:
 " + nestedLevel + "</hi>");
 }
 catch (JspException e) {}
 return EVAL_BODY_INCLUDE;
 }
}
```

### Example-2

#### test.jsp

```
<%@ taglib prefix="mime" uri=%> />
< mime : menu >
 < mime : menuItem item="chicken" />
 < mime : menuItem item="mutton" />
</mime:menu> < mime : menuItem item="fish" />
```

<taglib>  
</taglib>

<attribute>

<taglib> true </taglib>

<name> item </name>

<attribute>  
<!-->

<body-content> JSP </b-c>

<tag-class> tags.MenuItemTag </t-c>

<name> MenuItem </n>

<tag>

</tag>

<body-content> JSP </b-c>

<tag-class> tags.MenuItem </t-c>

<name> menu </n>

<tag>

<file> 1.2.3 </f-u>

<taglib version="2.1">

file

<mime:menu>

<item = "SAB" />

<item = "SCB" />

<item = "SCAC" />

<item = "SGP" />

<mime:menu>

(58)

```
public class MenuTag extends TagSupport
{
 private ArrayList l=null;
 public int doStartTag() throws JspException
 {
 l=new ArrayList();
 return EVAL_BODY_INCLUDE;
 }
 public void addMenuItem(String s)
 {
 l.add(s);
 }
 public int doEndTag() throws JspException
 {
 try
 {
 JspWriter out=PageContext.getOut();
 out.println("Ch1> Items are : " +l);
 }
 catch (IOException e) { }
 return EVAL_PAGE;
 }
}
import java.util.*;
public class MenuItemTag extends TagSupport
{
 private String item;
 public void setItem (String item)
 {
 this.item=item;
 }
}
```

menu items are : chicken, mutton, fish  
menu items are : SCJP, SCWCD,  
SCBCD, SCJD.

```

public int doStartTag()
{
 menuTag parent = (menuTag).getParent();
 parent.addMenuItem(item);
 return EVAL_BODY_INCLUDE;
}

```

(26)

Getting an arbitrary Ancestors class  
we can get immediate Parent by using getParent().

TagSupport class contains the following method to get an arbitrary  
ancestors class.

Public static Tag findAncestorsWithClass( Tag t, Class c )  
(t, class)

<sup>t1</sup>  
<sup>t2</sup>      required parent  
<sup>t3</sup>      class name

Accessing JSP Implicit Objects & Attributes in TagHandler class

using PageContext API write Tag Handler code to access JSP  
Implicit Web and Access web Application Attributes -

Tag Handler class can get PageContext Object by setPageContext()  
By using this PageContext Object Tag Handler can get all other  
JSP Implicit objects.

JSP Implicit Object

PageContext class method

- |               |                     |
|---------------|---------------------|
| ① request     | getRequest()        |
| ② response    | getResponse()       |
| ③ config      | getServletConfig()  |
| ④ application | getServletContext() |

```
} session getSession()
} out getOut()
} Page getPage()
} Exception getException()
```

If the Enclosing JSP is not from Page Then getException()

returns null

Package tags

Public class mycustomTag extends TagSupport

```
{ public int doStartTag() throws JspException
```

```
{
```

```
 ServletRequest req = PageContext.getRequest();
```

```
 String s1 = req.getServerName() + ":" + req.getServerPort();
```

```
 ServletResponse resp = PageContext.getResponse();
```

```
 String s2 = resp.getContentType();
```

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

```
 String s3 = session.getId();
```

```
 Throwable e = PageContext.getException();
```

```
tag
```

```
{ JspWriter out = PageContext.getOut();
```

```
 out.println("<h1>" + s1);
```

```
 out.println(s2);
```

```
 out.println("</>");
```

```
 open(e + "hi");
}
catch(IoE e) {
}
return E-B-I
}
```

(87)

Getting and setting Attributes by using PageContext API.

PageContext defines the following methods to perform Attribute management.

public void setAttribute (String name, Object obj);

public void setAttribute (String name, Object obj, int scope);

public Object getAttribute (String name)

I-H  
P, S, A

public Object getAttribute (String name, int scope)

public void remove (String name)

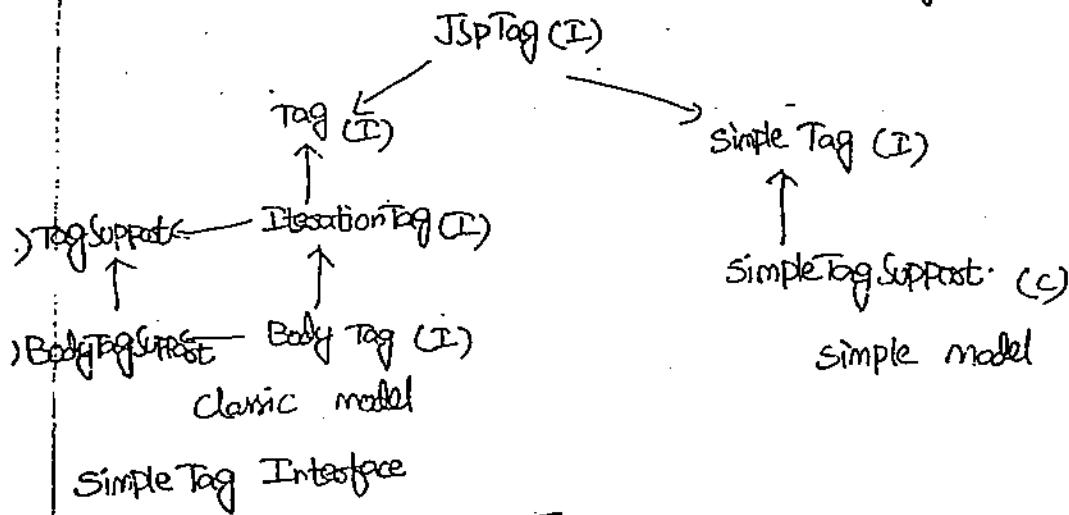
public void remove (String name, int scope)

public Object findAttribute (String name)

public Enumeration getAttributeNames (int scope).

## Simple Tag Model

- implementing CustomTags by using classic Tag model (Tag, Iteration Tag, Body Tag, TagSupport, BodyTag Support) is very complex bcz each Tag has its own life cycle and different possible return types for every method.
- To Resolve this complexity Sun People introduced Simple Tag model in JSP 2.0 v.
- In this model we can build customTags by using SimpleTag Interface and its implementation class SimpleTagSupport.



- This is child Interface of JSPTag and contains the following 5 methods.

- ① Public void setJspContext(JspContext c)
- ② Public void setParent(JSPTag t)
- ③ Public void setJspBody(JspFragment f)
- ④ Public void doTag() throws JspException, IOException.
- ⑤ Public JSPTag getParent();

## Lifecycle of Simple Tag Handler

(28)

① when JSP Engine Encounters a custom tag in JSP, it will identify the corresponding Tag Handler class by using taglib directive and tld file. It creates a new instance of Tag Handler class by executing Public no argument constructor. Simple Tag objects are never reused by web containers. So for each Tag Invocation a new Tag Handler object will be created.

② web Containers executes setJspContext() to make JspContext object available to Tag Handler class. By using this JSP Context Object Tag Handler class can get all JSP Implicit Objects and Attributes.

③ web Containers executes setParent() to make Parent Tag Object available to Tag Handler class. This method is usefull in the case of nested Tags.

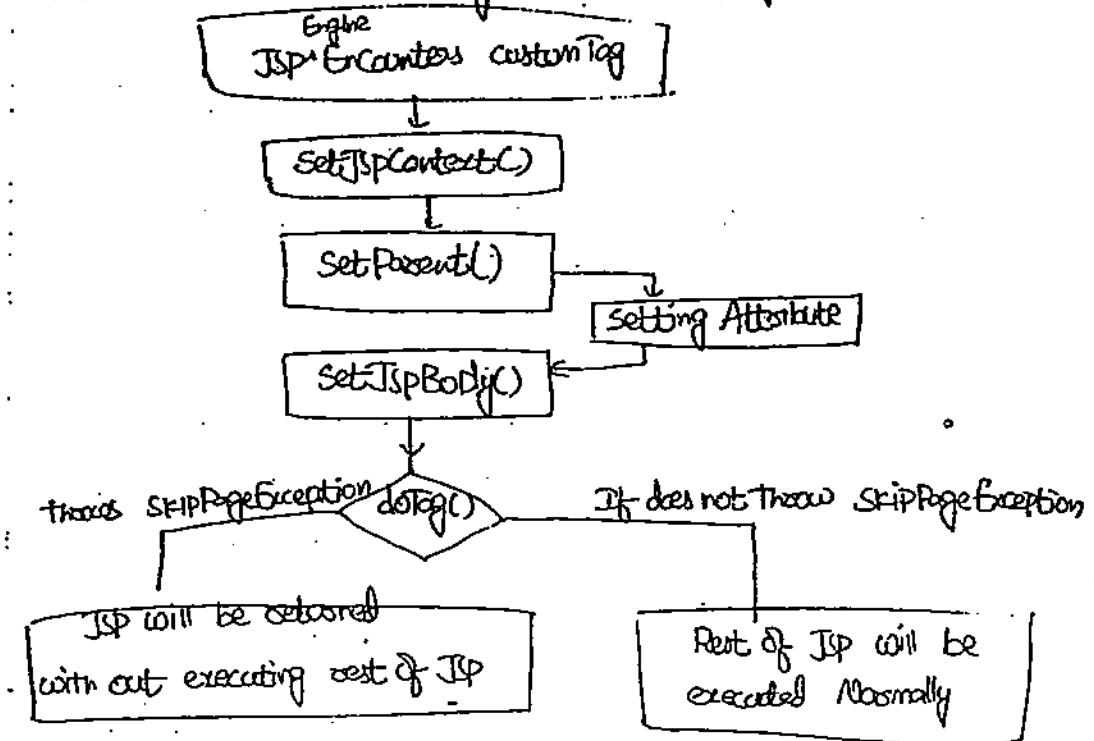
④ If the Tag is invoked with a body then set JSPBody() will be executed by taking JSPFragment Object as argument.

JSPFragment represents Tag Body in Simple Tag model. TagBody should not contain any Scripting Elements.

⑤ finally doTag() will be executed which is responsible to provide required functionality. This method is equivalent to doStartTag(), doEndTag() & doAfterBody().

Once doTag() completes, the TagHandler object will be destroyed by web container.

### Flowchart for SimpleTag Handler Lifecycle



### SimpleTagSupport Class

- This class provides default implementation for all methods of SimpleTag Interface.
- This class also provides the following additional methods:

- ① `JSPContext getJspContext()`
- ② `JSPFragment getJspBody()`
- ③ `JSPTag findAncestorWithClass(JSPTag t, class c)`.

first.jsp

<1.0 taglib prefix="mine" uri= "

<mine:myTag> This is tag Body </mine:myTag>

<hi> This is rest of the JSP </hi>

tld file

<tag>

<name> myTag </name>

<tag-class> tags.myCustomTag </tag-class>

<body-content> tagdependent </body-content>

consider as Plain Text

Public class MyCustomTag extends SimpleTagSupport

{ Public void doTag() throws JspException, IOException

{ JspWriter out = getJspContext().getOut();

out.println "<hi> Hello this is simple Tag </hi>";

throws new SKIPPageException();

}

}

If doTag() throws SKIP-Page Exception Hello this is simple Tag

If doTag() doesn't throw SKIP-Page Exception then O/P is

Hello this is simple Tag

This is rest of JSP.

What is the Diff b/w classic & Simple Tag w.r.t Tag Body.

In classic Tag model we are allow to take Scripting Elements in

Tag body hence the possible values for Body Content Tag are

empty, tagdependent, scriptlets, JSP.

→ But in Simple Tag Model Scripting Elements are not allowed in tag body. Hence the allowed values for <Body-Content> tag are empty, tagdependent, scriptlets but not JSP.

Processing Body Content in SimpleTags

→ we can Access Tag Body inside a Simple Tag Handler by using getJSPBody(). This method returns JspFragment Object.

```
public JspFragment getJSPBody()
```

→ JSP Fragment object represents Tag Body on this object we can call the following 2 methods.

① JspContext getJSPContext();

② void invoke(Writer w); - It causes evaluation of TagBody and return to supplied writer.

→ use "null" Argument to write directly to the JSP OutputStream

test-JSP

```
<@ taglib
 chs> This is Before Tag Invocation </ch>
<mime:myTag> This is Tag Body </m-m>
```

<ch> This is after Tag Invocation </ch>

tld file

```
<tag>
 <name>
 <tag-class>
 <body-content> tagdependent </b-c>
```

```

public class MyCustomTag extends SimpleTagSupport (90)
{
 public void doTag() throws JspException, IOException
 {
 JspWriter out = getJSPBody().getOut();
 out.println("<hi>Hello This is from Simple Tag </hi>");
 getJSPBody().invoke(null);
 throw new SkipPageException();
 }
}

```

This is Before Tag Invokation  
Hello This is from simple tag  
This is Tag Body

### Key Diff b/w Simple and classic Tags

| <u>Property</u>                                    | <u>Simple Tags</u>                          | <u>Classic Tags.</u>                                                                 |
|----------------------------------------------------|---------------------------------------------|--------------------------------------------------------------------------------------|
| Tag Interfaces                                     | Simple Tag                                  |  |
| Supporting Implementation classes                  | SimpleTagSupport                            | TagSupport<br>BodyTagSupport                                                         |
| Key - life cycle methods that we have to implement | doTag()                                     | doStartTag()<br>doEndTag()<br>doAfterBody()                                          |
| How to write response to JSP OutputStream          | getJSPContext().getOut().<br>println();     | PageContext.getOut();<br>println();                                                  |
|                                                    | → no need to use try, catch for IOException | → compulsory we should enclose by using try, catch for IOException                   |

- 5) How to Access JSP Implicit Objects & Attributes.  
By using JSPContext object
- 6) How to cause the body to be Processed  
getJspBody().invoke(null) Returns EAL-Body-Marker from doStartTag()
- 7) How to cause the current Page Evaluation to Stop.  
throw SkipPageException from doTag()  
in doEndBody()  
body tag <sup>if present</sup> from doStartTag()  
Return SKIP-PAGE from doEndTag()

## Dynamic Attributes

(91)

In general, Tags can Contain Attributes for which we can have declarations in the tld file by `<attribute>` tag.

For these Attributes we have to maintain Instance's variables and corresponding Setter methods in Tag Handler class. Such Type of Attributes are called Static Attributes.

But we can use Attributes, even though Tld file doesn't contain any Attribute Tag Declarations. Such type of Attributes are called Dynamic Attributes.

Dynamic Attributes are applicable for both classic & Simple Tags and Introduced in JSP 2.0 Versions.

To support Dynamic Attributes we have to do following Arrangements.

In the Tld File we have to use Dynamic Attributes Tag

```
<taglib <taglib ---- >
 |
 <tag>
 |
 <dynamic-attribute>true </dynamic-
 attributes>
 |
 </tag>
</taglib>
```

② The corresponding Tag Handler class should Implement Dynamic Attributes Interface.

⇒ This interface introduced in Jsp 2.0 version and contains only one method setDynamicAttribute().

```
public void setDynamicAttribute (String namespace,
 String name, Object value)
```

⇒ web container calls this method for each Dynamic Attribute.

### test.jsp

```
<%@ taglib prefix="mime" uri="-----" %>
<mime:MyTag num="2" min="4" max="5", pow="3" />
<mime:MyTag num="4" min="15" />
```

### tld file

```
<taglib version="2.1">
 <lib-version> 1.2.3 </lib-version>
 <tag>
 <name> MyTag </name>
 <tag-class> tags.MyCustomTag </tag-class>
 <body-content> empty </body-content>
```

<attribute>

<name> num </n>

<required> true </r>

</attribute>

<dynamic-attribute> true </dynamic-attribute>

<tag>

</taglib>

### myCustomTag.java

```
public class myCustomTag extends SimpleTagSupport
 implements DynamicAttribute
```

```
{
 int num;
```

```
 String result = " ";
```

```
 public void setNum(int num)
```

```
 {
 this.num = num;
```

```
}
```

```
 public void setDynamicAttribute(String ns, String name,
 Object value)
```

```
 int i = Integer.parseInt((String) value);
```

(92)

```
if(name == "min")
{ result = result + "The minimum value is :" +
 math . min (num , i);
}

else if(name == "max")
{ result = result + "The maximum value is :" +
 math . max (num , i);
}

else if(name == "pow")
{ result = result + "The power of number is :" +
 math . max (num , d);
}
```

```
3
Public void doTag() throws JspException , IOException
{
 getJspContext().getOut().Println ("

" + The result + "

");
}
3
Output : The minimum value is : 2
 The maximum value is : 5
 The Power of numbers is : 8
```

## Tag Files

(Q3)

- Q Describe the Semantics of Tag File model.
- Q Describe Application Structure for Tag Files.
- Q Write a Tag File and Explain the constraints on JspContent in body of Tag.
- Q Tag Files concept has introduced in JSP 2.0 Version.
- Q Tag File is a simple JSP Page (or) JSP Document designed to be used as custom Action.
- Q The main Advantage of Tag files is, we can build very easily when compared with classic & simple Tags.
- Q The main Limitation of Tag Files is, it won't suggestable for doing much processing.

### Building and using a Simple Tag File

- Q Write a JSP Page as Document and save it with ".tag" extension.
- Q Place this .tag file in /WEB-INF/tags.
- Q Put a taglib Directive in the JSP with "tagdir" Attribute.
- Q <%@ taglib prefix="mime" tagdir="/WEB-INF/tags" %>

## Demo Program.

### test.jsp

```
<%@ taglib prefix="mime" tagdir="/WEB-INF/tags" %>
```

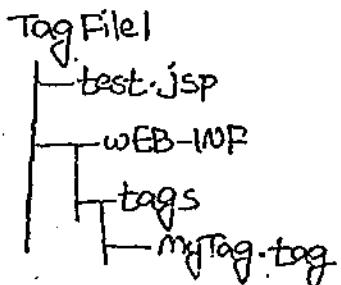
<h1> This is Tag File Demo Example </h1>

<mime : mytag />

<mime : mytag />

myTag.tag

This 2 should be  
same name



<h1> Hello !! This is from Tagfile </h1>

### Note

⇒ Tag Files internally converted in to Simple Tag Handles  
(available in Work folder)

⇒ Not required to write tld file.

### Declaring a TagFile with Attribute

⇒ we can define Attribute for Tag Files by using  
attribute Directive.

<%@ attribute name="num" required="true"

value="true" %>

## test.jsp

(Q)

```
<%@ taglib prefix="mime" tagdir="/WEB-INF/tags" %>
<h1> This is Tagfile Demo Example </h1>
<mime:myTag title="Kick" />
```

## myTag.tag

```
<%@ attribute name="title" required="true"
 %>
```

```
<h1> Hello --- ${title} is a big hit movie </h1>
```

Output: This is Tagfile DemoExample

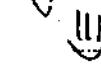
Hello --- Kick is a big hit movie.

## Declaring Body Content for a Tag File

Tag File Invocation can contain body but we are not allow to use scripting Elements.

We can declare body-content in tag file by using tag directive.

```
<%@ tag body-content="tagdependent" %>
```



allowed values are — empty, tagdependent,  
scriptless  
Default Value — scriptless.

→ In the Tag File we can Access Tag body by using.

```
<jsp:doBody />
```

### test.jsp

```
<%@ taglib prefix = "mime" tagdir = "/WEB-INF/tags" %>
```

```
<mime:headers color = "yellow" >
```

This is body of Tag File

```
</mime:headers>
```

### headers.tag

```
<%@ attribute name = "colours" required = "true"
 stepValue = "true" %>
```

```
<%@ tag body-content = "tagdependent" %>
```

```
<hi> Hello -- This is from Tag file </hi>
```

```

```

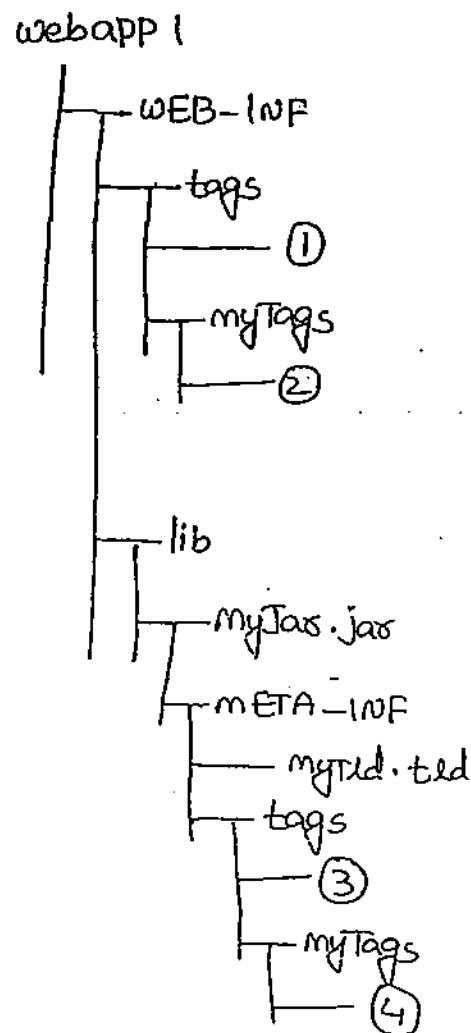
```
 <jsp:doBody />
```

```

```

where web Container looks for Tag Files.

(95)



Either Directly or Indirectly in WEB-INF / tags folder.

Either Directly or Indirectly in META-INF / tags

folders present in jar file of WEB-INF / lib.

whenever we are Deploying tag file in some jar

then Compulsory we have to write tld file.

```
<taglib version="2.1">
 <tlib-version>1.2.3 </t-v>
 <uri>www.duogsoft.com </uri>
 <tag-file>
 <name> MyTag </name>
 <path>/META-INF/tags/mytag.tag
 </tag-file> </path>
</taglib>
```

### test.jsp

```
<%@ taglib prefix="mime" uri="www.duogsoft.com" %>
<mime :mytagfile />
<mime :myTagfile />
```

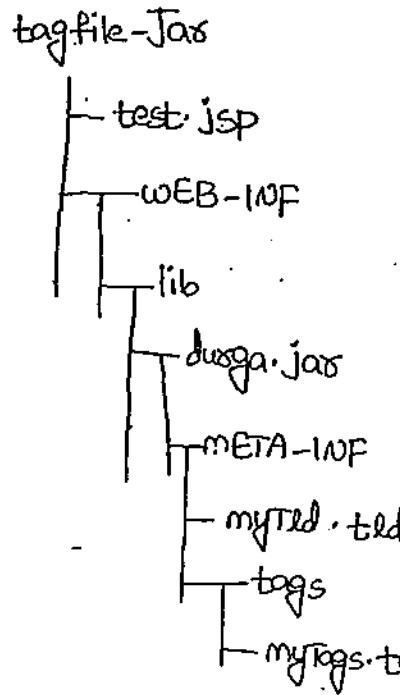
tld file - myTld.tld

```
<taglib version="2.1">
 <tlib-version>1.2.3 </t-v>
 <uri>www.duogsoft.com </uri>
 <tag-file>
 <name>MyTagfile </name>
 <path>/META-INF/tags/myTag.tag </path>
 </tag-file>
</taglib>
```

## myTags.tag

(q6)

<hi> This is from Tag File Deployed in Jars. </hi>



→ first create

dugsja

└ META-INF

└ myTld.tld

└ tags

└ myTags.tag

→ create Jar file

for dugsja folder

→ Place that <sup>Tag</sup> folders  
in lib.

→ jars -cvf dugsja.jar \*

\*.\*

## Disabling Scripting Language.

we can disable Scripting language in JSP's by using  
<scripting-invalid> Tag.

<web-app>

<jsp-config>

<jsp-property-group>

<url-pattern> \*.jsp </url-pattern>

<scripting-invalid> true </scripting-invalid>

<jsp-property-group>

</jsp-config>

</web-app>

→ once the Scripting Language is Invalidate, we are not allowed to use Scripting Elements in JSP. Violation leads to Translation Errors. (Tomcat doesn't provide support).

### Disabling Expression Language Globally

→ we can disable EL for a particular JSP by using "isELIgnored" Attribute of Page Directive.

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

→ we can disable EL at Application Level as follows.

```
<web-app>
```

```
 <jsp-config>
```

```
 <jsp-property-group>
```

```
 <url-pattern> *.jsp </url-pattern>
```

```
 <el-ignored> true </el-ignored>
```

```
 <jsp-property-group>
```

```
 <jsp-config>
```

```
 </web-app>
```