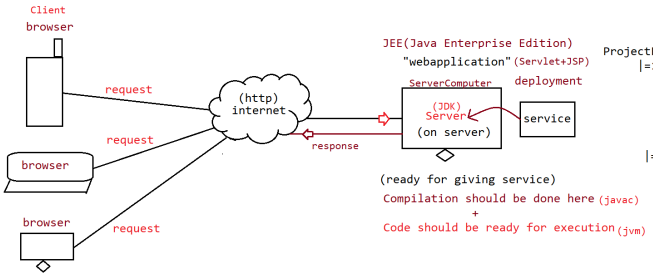


ProjectFolder (FirstApp)
|=> WEB-INF
|=>web.xml
|=> classes
*.class (code is compiled and ready for execution)
|=> lib (optional untill u use 3rd party api's)
*.jar
|=> src
|=>packages
|=> .java

Standalone(oneuser)
1 Thread(JVM creates)

JSE => CoreJava
ExceptionHandling,String,MultiThreading
Collection,Wrapperclasses,FileHandling,JDBC

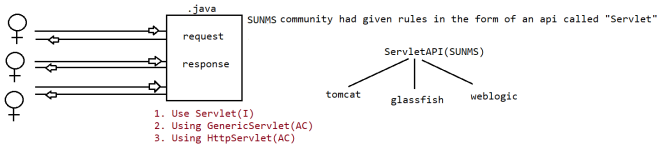
webapps(many user)
Server(ThreadPool)

JEE => AdvancedJava
Servlet,JSP,JSTL,Thymleaf,....

Assign one thread for every client and
give them the response
henceforth no main() in webapps

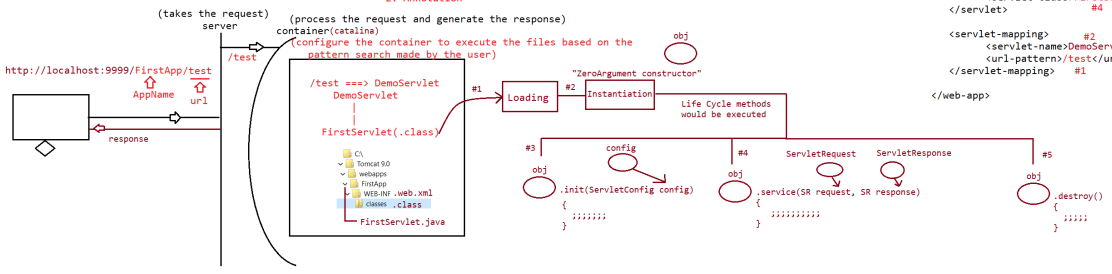
Structure of tomcat

- bin ⇦ .exe files
- conf ⇦ configuration files
- lib ⇦ *.jar
- logs ⇦ (request,response information of clients)
- temp ⇦ (temporary files)
- webapps ⇦ (deployment folder)
- work ⇦ (backup files)



1. Use Servlet(I)
2. Using GenericServlet(AC)
3. Using HttpServlet(AC)

1. XML
2. Annotation



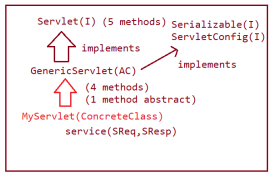
web.xml

```
<web-app>

   <servlet>                #3
         <servlet-name>DemoServlet</servlet-name>
         <servlet-class>FirstServlet</servlet-class>
   </servlet>               #4

   <servlet-mapping>        #2
         <servlet-name>DemoServlet</servlet-name>
         <url-pattern>/test</url-pattern>
   </servlet-mapping>       #1

</web-app>
```

Servlet(I)
|=> 5 methods

init(SC)
service(SReq,SRes) ⇦ request processing logic
destroy()

getServletConfig()
getServletInfo()



2nd Approach is better

D:\servletpgms\.metadata\.plugins\org.eclipse.wst.server.core

our project created location

inside this folder eclipse automatically
keeps the application.
this type of deployment is called "soft-deployment".

1. Servlet(I) ⇦ javax.servlet
   init(),service(sreq,sresp), destroy()===> life cycle methods
   getServletInfo(), getServletConfig()

   ⇧ implements

2. GenericServlet(AC) ⇦ javax.servlet
   All 4 methods of Servlet(I) implementation is available
   abstract public void service(Sreq,Sresp) throws SE,IOE

   ⇧ extends

3. HttpServlet(AC) ⇦ javax.servlet.http

GenericServlet====> meant for any protocol interaction
                    like http, smtp,.....

For a webapplication, the protocol used in "Http",so to work with
specific protocol SUNMS had come up with SRS called "HttpServlet".

SUNMS had come up with special package to promote
        http protocol interaction
        interface
        =========
        1.HttpServletRequest(I)
        2.HttpServletResponse(I)
        3.HttpSession(I)

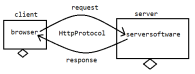
        class
        =====
        1.HttpServlet
        2.Cookie

public void init(ServletConfig config) throws ServletException;
        ⇧
This is used internally by webcontainer

1. loading
2. instantiation
3. initialization
        init(ServletConfig config)
        {
            init()   ⇧ local variable
        }
4. service
    public void service(Sreq,Sresp)throws SE,IOE
    {
        ServletConfig config =getServletConfig();
        System.out.println(config); // null
    }
5. deInstantiation

public void init() throws ServletException;
        ⇧
This is used by devleoper in his code

1. loading 2. instantiation
3. initialization
        init(ServletConfig config)
        init()
4. service
5. destroy



Structurue of HttpProtocol Request
==================================

RequestLine
RequestHeaders
RequestBody

RequestLine: Request Type | Requested resource | protocol version
GET | FirstApp/test | HTTP/1.1
FirstApp/test ⇧ configured in the server env

RequestHeaders ⇦ internally used to give a proper response
        |
Browser details and its supported language format and so on.....

RequestBody ⇦ "user supplied inputs will be available"
        |
It contains user information

Structure of HttpProtocol Response
==================================

StatusLine
ResponseHeaders
ResponseBody

StatusLine: Protocol version | status code | Description status code
HTTP/1.1 | 200 | OK
information

1XX => informational
2XX => Succesfull
3XX => Redirection
4XX => ClientError
5XX => ServerError

ResponseHeader ⇦ MIME(MultiPurpose Internet Mail Exchange)
        |
response information from the server
    a. content-type
    b. content-length
    c. modified date etc.....

ResponseBody ⇦ "Acutal information"
        |
Actual information sent by the server,this
information will be displayed to the user.

Type of HttpRequest Methods
===========================

1. GET
2. POST
3. HEAD
        ⇘ introduced in HTTP/1.0V
4. PUT
5. DELETE
6. OPTIONS
7. TRACE
        ⇗ HTTP/1.1V

RequestType used in Webapplications are
    a. GET
    b. POST

GET (it is also called as "idempotent request/safe request")
  => It is used to get the information from server
  => This request data will be visible in the address bar when we send the request,
     so it is not secured.
  => It can be book marked,and it also supports caching of data
  => since the data is visible in address bar, only limitied data can be sent.

eg:: http://localhost:9999/FirstApp/test ?username = sachin & password = tendulkar
                                        ⇧ QueryString

  1. Type url and hit enter key
  2. clicking on hyperlink
  3. submitting html form with method attribute as "GET"

RequestLine ⇦ GET(Query String is sent)
RequestHeaders ⇦ browser details would be available
RequestBody ⇦ empty

POST(It is not a safe request/idempotent request)
  => It is used for putting the data sent from client to server
  => This request data would not be visible in the address bar when we send the request,
     so it is secured.
  => It can't be book marked,and it won't supports caching of data
  => since the data is not visible in address bar, large volumen of data can be sent.

eg: http://localhost:9999/FirstApp/test ? username = sachin & password = tendulkar
                                        ⇧ QueryString

1. submitting html form with method attribute as "POST"

RequestLine ⇦ POST
RequestHeaders ⇦ browser details would be available
RequestBody ⇦ (Query String is sent)

HttpServlet(AC)

```
public void service(SReq req, SRes resp) throws SE,IOE
{
        HttpServletRequest request = (HSR) req; HttpServletResponse response = (HSR) resp;
        service(request,response);

}
protected void service(HSR req,HSR resp) throws SE,IOE
{
    String method = req.getMethod();

    if(method.equalsIgnoreCase("POST"))
      doPost(req,resp)

    else if(method.equalsIgnoreCase("GET"))
      doGet(req,resp);

    ;;;;;
    ;;;;;
    ;;;;;
    else
        return 501 status code saying http method not implemented

}
```

HttpServlet(AC)

```
protected void doPost(HSR req, HSR resp) throws SE,IOE
{
    return 405|400 status code saying HTTP Method POST not s upported by this URL
}

protected void  doGet(HSR req, HSR resp) throws SE,IOE
{
    return 405|400 status code saying HTTP Method GET not s upported by this URL
}
```

1. loading
      static block

2. instantiation
      constructor(public zero argument)

3. initialization
      init(ServletConfig config)
      init() ⇦ meant for devleoper

4. request processing
      public void service(SReq req,SRes) throws SE,IOE
      protected void service(HSR req,HSR res) throws SE,IOE
      ⇧
      user defined FirstServlet Object doXXX(HSR req,HSR res) throws SE,IOE should be called

5.deinstantiation
      public void destroy()

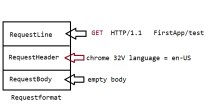
Structure of Project
====================

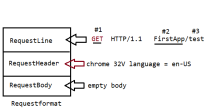
ProjectName
|
|====> WEB-INF
| |=======> classes
| |=======> lib *.class   private-area
| |=======> web.xml
|
|=====> .java
|=====> .html files   public area

⇧ The region/files which can't be accessed by the clients from address bar(sending the request)
This is solely meant for webcontainer only.

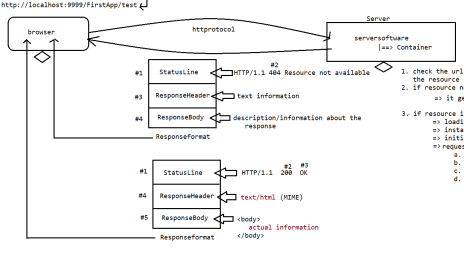
The region/files which can be accessed directly by the clients from address bar(sending the request)

http://localhost:9999/FirstApp/test
⇧ it is configured in annoatation style for container



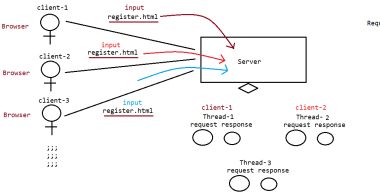


http://localhost:9999/FirstApp/test

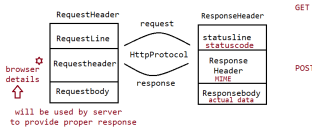


1. check the url pattern and find the resource
2. if resource not available
        => it generates clienterror and send it server

3. if resource is available
        => loading
        => instantiation
        => initialization
        => request processing
            a. content-type
            b. use response entry
            c. write the response
            d. close the writer

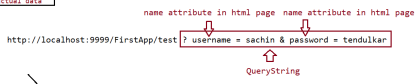



```
RequestApp
    |-> src
        |=> in.ineuron.controller
                => RequestServlet.java

    |-> webapp
        |====> register.html

    |=> WEB-INF
        |=> classes
            |=> RequestServlet.class
        |=> lib
            |=> *.jar

        |=> web.xml
            |=> mapping details in xml format
```

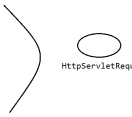


GET ====> 1. type url and hit enterkey
2. clicking on hyperlink
3. submiting form with method attribute as "get".
Note: data would be sent in RequestLine

POST ====> 1.submitting form with method attribute as "post".
Note: data would be sent in RequestBody

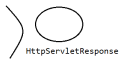


Servlet
=======
1. public String getParameter()
2. public String[] getParameterValues()
3. public Enumeration<String> getHeaderNames()
4. public Object getHeader(String headerName)
5. public String getRemoteHost()
6. public String getRemoteAddr()
7. public String getRemotePort()
8. public String getServerName()
9. public String getServerPort()



Note: load-on-startup
a. value should be 0 and any positive number
b. if 2 servlets has same value then we can't predict the order
3. less the load-on-startup for a value,that particular servlet loading,instantiation,initialziation will happen.

1. public void setContentType(String name)
2. public PrintWriter getWriter()
3. public ServletOutputStream getOutputStream()



Note: Writing images,pdf's,..... as a response

```
response.setContentType("image/jpg");
   ;;;;
   ;;;;
ServletOutputStream os = response.getOutputStream();
```

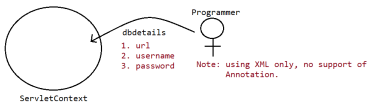
location of image file
==========
D:\servletpgms\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\ImageResponseApp\nitin.JPG
smooth deployment
ProjectName
Image

Note:

```
PrintWriter out = response.getWriter();
ServletOutputStream os = response.getOutputStream();
```

In Servlet Coding we can't open 2 streams for response, if we try to do it would result in
java.lang.IllegalStateException

ServletContext(I)
==========
=> When we do deployment(manual), server will scan "webapps" folder and identifies the projects which is deployed.
=> All the identified Projects will be kept in Meta-space of server.
=> For Every Project which is deployed in Meta-Space automatically "ServletContext" object will be created.



web.xml

```
<web-app>
   <context-param>
        <param-name>jdbcURL</param-name>
        <param-value>
             jdbc:mysql://localhost:3306/enterprisejavabatch
        </param-value>
   </context-param>

   <context-param>
        <param-name>user</param-name>
        <param-value>root</param-value>
   </context-param>

   <context-param>
        <param-name>password</param-name>
        <param-value>root123</param-value>
   </context-param>

</web-app>
```

context object can be accessed in the entire project, meaning in all the servlets of the project

1. public Enumeration getInitParameterNames()
2. public String getInitParameter(String name)
3. public Enumeration getAttributeNames()
4. public String getAttribute(String name)
5. public void setAttribute (String k,Object v)

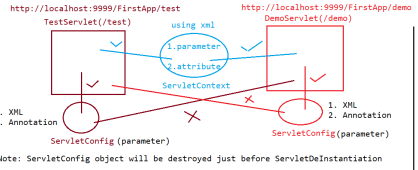
Note: In ServletContext Object, we can add parameter data as well as attribute data
from xml
during execution

Note: Context Object will be destroyed only when we undeploy the project.

ServletConfig
==========
=> This object is used to store the configuration details of a particular servlet like logical name of the servlet,initialization parameters, and so on...
=> Using ServletConfig we will get to know the complete view of a particular servlet.
=> Loading ===> static block
instantiation ===> public Zero argument constructor
ServletConfig
initialization ====> public void init(ServletConfig config) throws SE

RequestProcessing==> public void doXXXX(HSR request, HSR response) throws SE,IOE
DeInstantiation ==> public void destroy()

=> ServletConfig object is specific to a particular Servlet.



Note: ServletConfig object will be destroyed just before ServletDeInstantiation

1. public Enumeration getInitParameterNames()
2. public String getInitParameter(String name)

XML
===

```
<web-app>
   <context-param>
        <param-name></param-name>
        <param-value></param-value>
   </context-param>
        ;;;
        ;;;
   <servlet>
        <servlet-name></servlet-name>
        <servlet-class></serlvet-class>
        <init-param>
             <param-name></param-name>
             <parma-value></param-value>
        </init-param>
             ;;;
             ;;;
   </servlet>

</web-app>
```

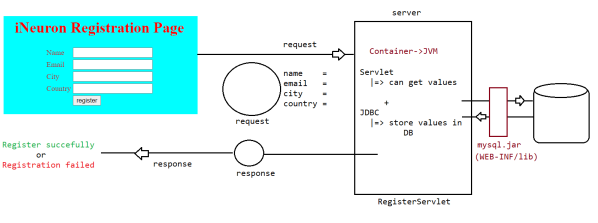
ServletContext
ServletConfig

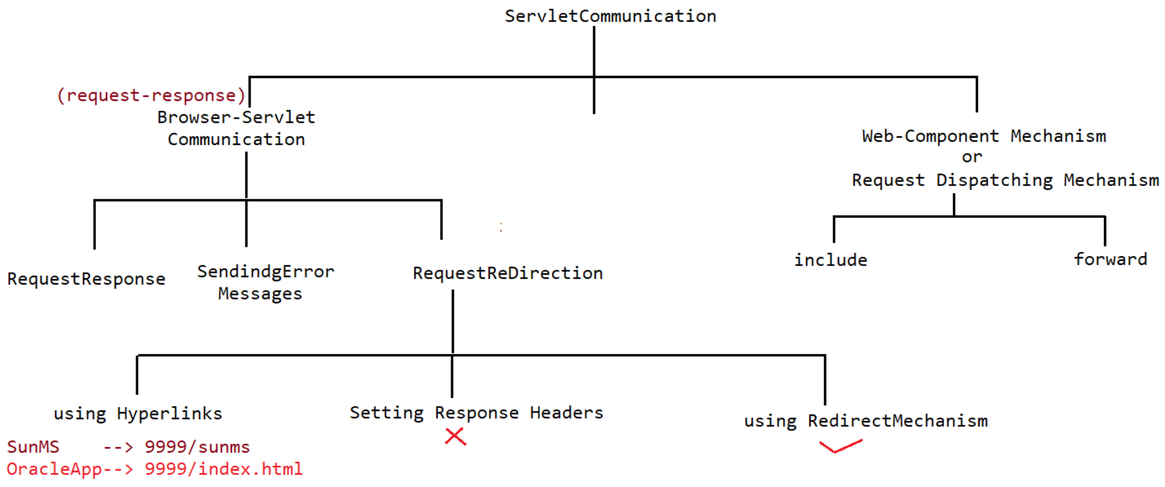
```
<servlet>
        <servlet-name>TestServlet</servlet-name>
        <servlet-class>in.ineuron.controller.TestServlet</servlet-class>
        <init-param>
            <param-name>jdbcurl</param-name>
            <param-value>jdbc:mysql:///enterprisejavabatch</param-value>
        </init-param>
        <init-param>
            <param-name>user</param-name>
            <param-value>root</param-value>
        </init-param>
        <init-param>
            <param-name>password</param-name>
            <param-value>root123</param-value>
        </init-param>
</servlet>
```

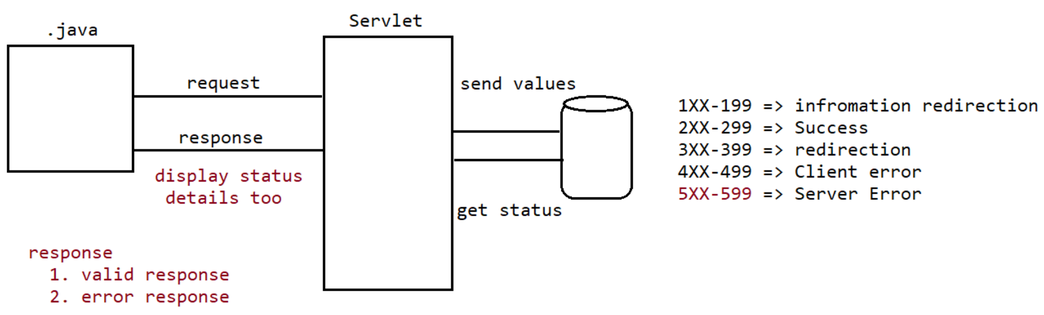

```
@WebServlet
(
   urlPatterns = { "/test" },
   initParams = {
      @WebInitParam(name = "jdbcUrl", value = "jdbc:mysql:///enterprisejavabatch"),
      @WebInitParam(name = "user", value = "root"),
      @WebInitParam(name = "password", value = "root123")
   }

)
```

```
<servlet>
        <servlet-name>DemoServlet</servlet-name>
        <servlet-class>in.ineuron.controller.DemoServlet</servlet-class>
        <init-param>
            <param-name>Company</param-name>
            <param-value>iNeuron</param-value>
        </init-param>
        <init-param>
            <param-name>CourseName</param-name>
            <param-value>EnterpriseJava</param-value>
        </init-param>
        <init-param>
            <param-name>Mentor</param-name>
            <param-value>NavinReddy</param-value>
        </init-param>
</servlet>
```


```
@WebServlet
(
   urlPatterns = { "/demo" },
   initParams = {
            @WebInitParam(name = "Company", value = "iNeuron"),
            @WebInitParam(name = "course", value = "Enterprisejava"),
            @WebInitParam(name = "mentor", value = "NavinReddy")
   }

)
```







1XX-199 => infromation redirection
2XX-299 => Success
3XX-399 => redirection
4XX-499 => Client error
5XX-599 => Server Error

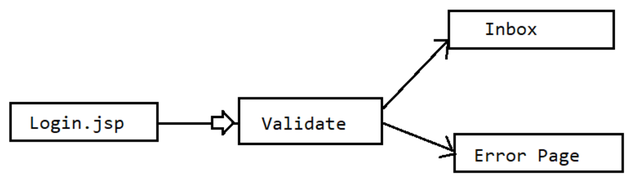
response
1. valid response
2. error response

Servlet

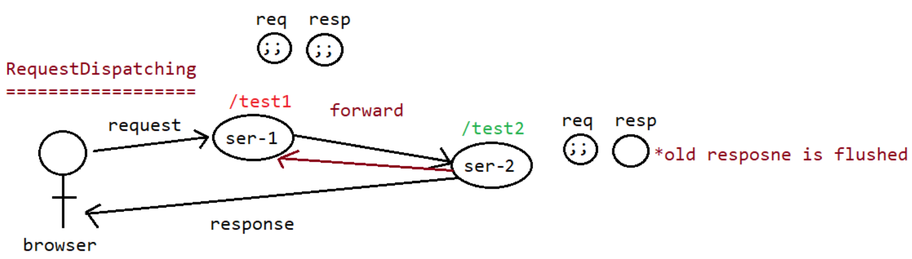
Communication happening between webcomponent

Login page
Validation
Inbox
ComposeMail
MailSending
ErrorPage
;;;;
;;;;
;;;;

Maintainence is not easy
debuging becomes difficult
enhancement becomes difficult



Note: For server the default landing page of the application is "index.html".
meaning if we send the request then this page would be loaded automatiaclly.
eg: http://localhost:9999/FirstApp/index.html
not required to explicitly write



RequestDispatcher Object can be obtained in 2 ways

a. ServletContext

```
ServletContext context = getServletContext();
RequestDispatcher rd   = context.getRequestDispatcher("/test2");//relative path only
```

b. ServletRequest

```
RequestDispatcher rd = request.getRequestDispatcher("/test2");// relative path
RequestDispatcher rd = request.getRequestDispatcher("test2");//  absolute path
```

Absolute path => It indicates we need to write from the root level.
Relative path => It indicates we need to write from the current directory(./ or /)

Note: Topics on RequestDispatcherMechanism

/test1

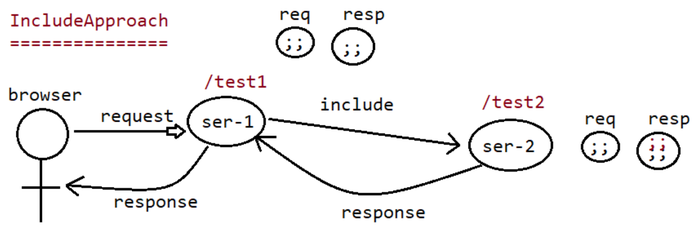
```
RequestDispatcher rd = request.getRequestDispatcher("/test2");
rd.forward(request,response);
```

/test2

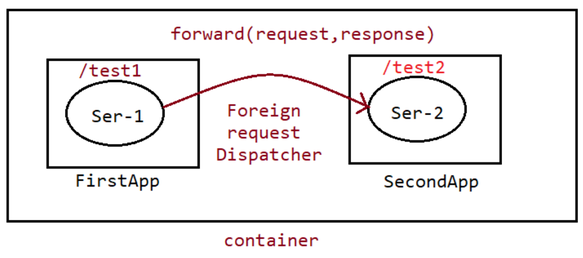
```
RequestDispatcher rd = request.getRequestDispatcher("/test1");
rd.forward(request,response);
```

StackOverFlowError

After commiting the response, we can't forward the request, if we try to do it would result in "IllegalStateExeption.



Total response => response from ser-1 + response from ser-2



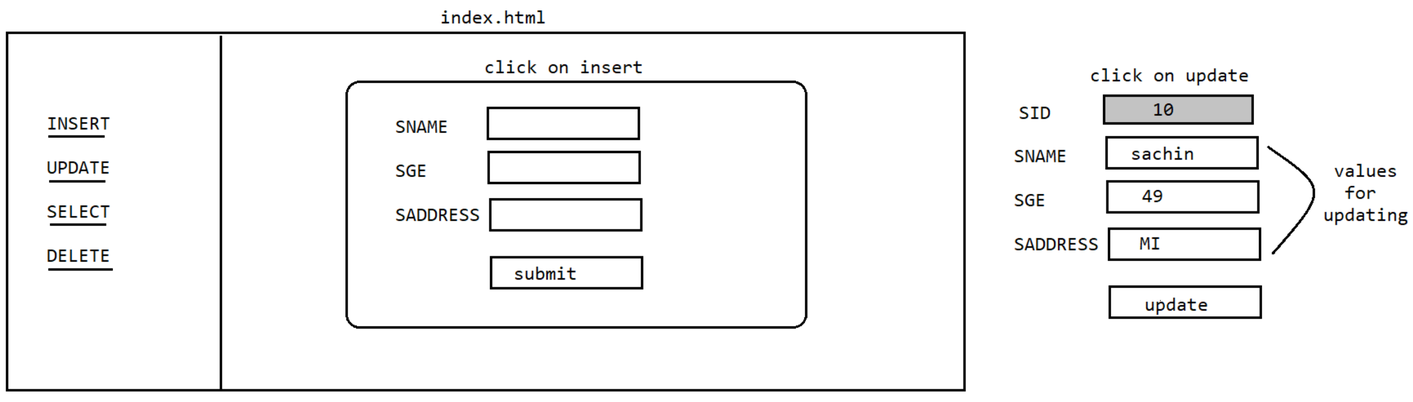
1.sendRedirect(request,response)
different app and those applications can run in different server also.

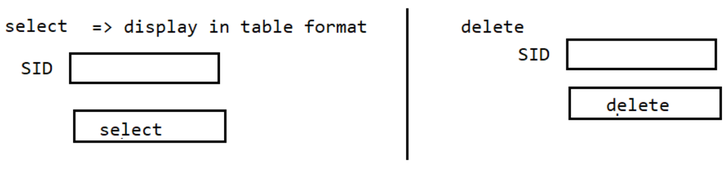
2.forward(request,response)
different app and those application should run compulsorily in the same server.
Note: different application means use Context object only.

By default Foreign Request Dispatcher support is not available in eclipse
To see the effect just make the following changes
open tomcat/conf folder
a. open context.xml file
b. update as shown below

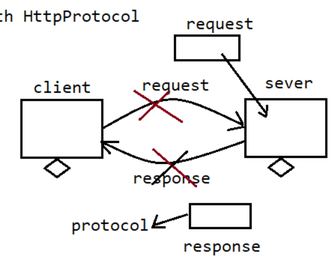
```
<Context crossContext = "true">
      ;;;;;;;
</Context>
```

CRUD Application





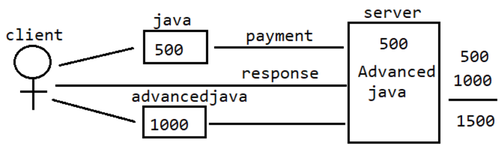
Working with HttpProtocol



1. once the server sends the response to the protocol, server will forget the client data because the Http protocol is "Stateless".

It would assume every request as the new request

Assume we are working with Flipkart application



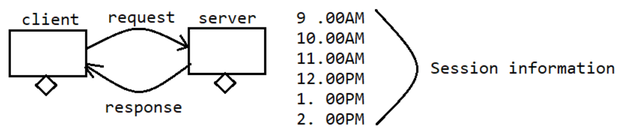
1. At the time of processing the later request, if we want to get the data of previous request, then we need to go for "Session Tracking"
2. Every server side technology should support session tracking because the protocol used is http and it is "stateless" in nature

Session Tracking Mechanism

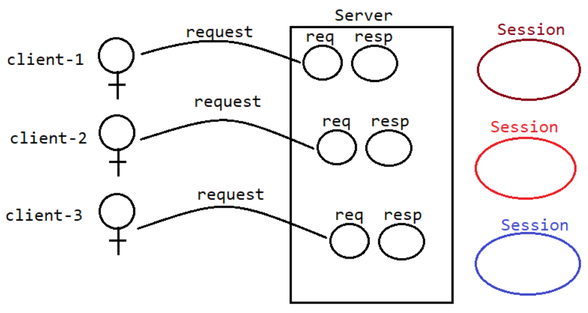
a. HttpSession Tracking mechanism
b. Cookies Session Tracking mechanism
c. URL-ReWriting Session Tracking mechanism
d. Hidden form field session Tracking mechanism

What is Session?

Sesison is a time duration, it will start from starting point of the client conversation with server and will terminate at the ending point of client conversation with the server



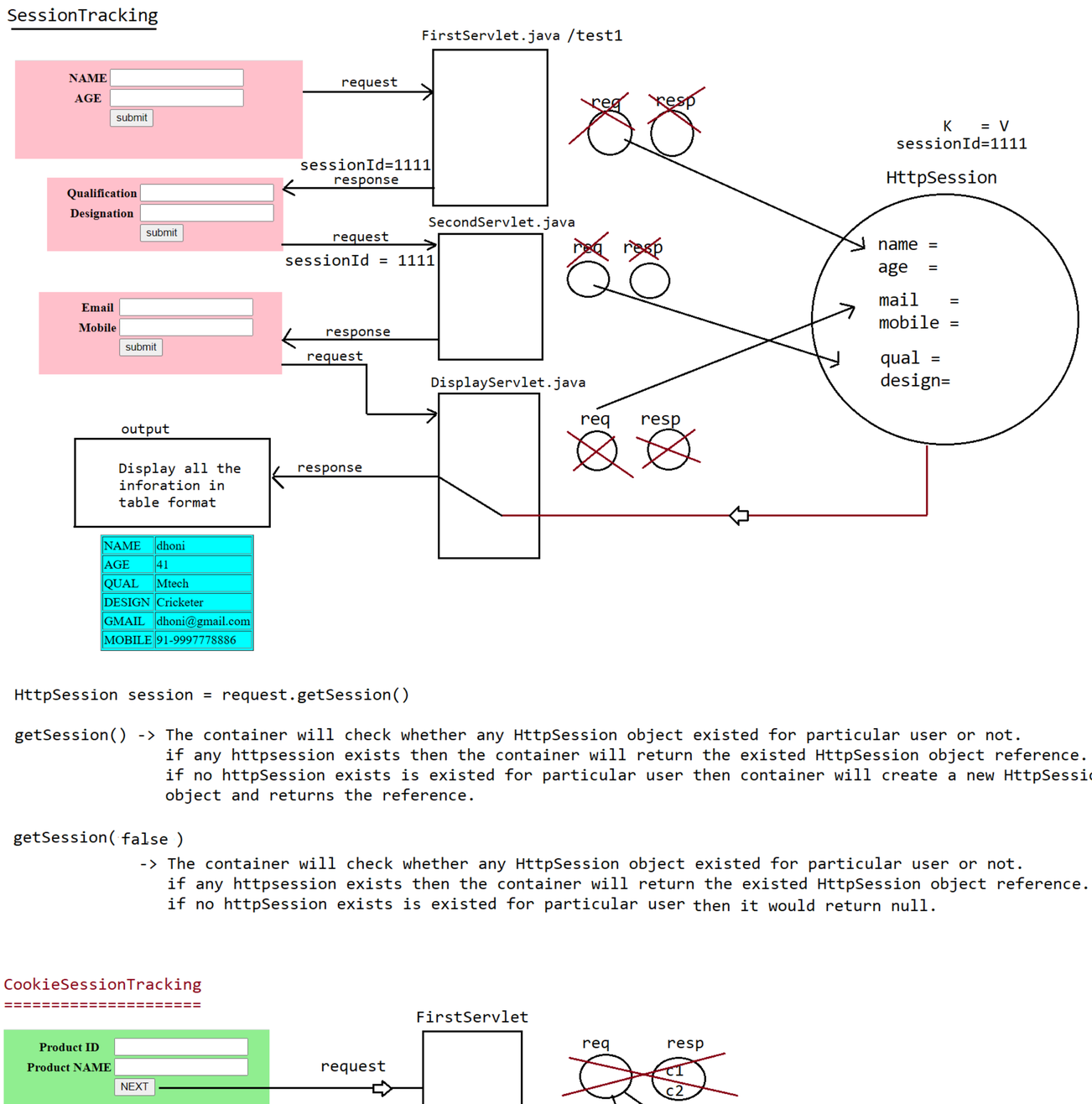
The data which is transferred b/w client to server through multiple request during a particular session then that data is called "State of a session".



To keep track of all the session objects at the server machine, we need a set of explicit mechanisms called as "Session Tracking Mechanism".

a. HttpSession Tracking mechanism
b. Cookies Session Tracking mechanism
c. URL-ReWriting Session Tracking mechanism — Servlet-api level
d. Hidden form field session Tracking mechanism ⇦ purely devloper level

SessionTracking



NAME	dhoni
AGE	41
QUAL	Mtech
DESIGN	Cricketer
GMAIL	dhoni@gmail.com
MOBILE	91-9997778886

Disadvantages

1. More the no of users, more would be the session object
2. More the session objects, those object will be in the server memory.
3. More the session objects, maintainence would be difficult at the server side.

To resolve this problem we use "CookieSessionTracking" mechanism.

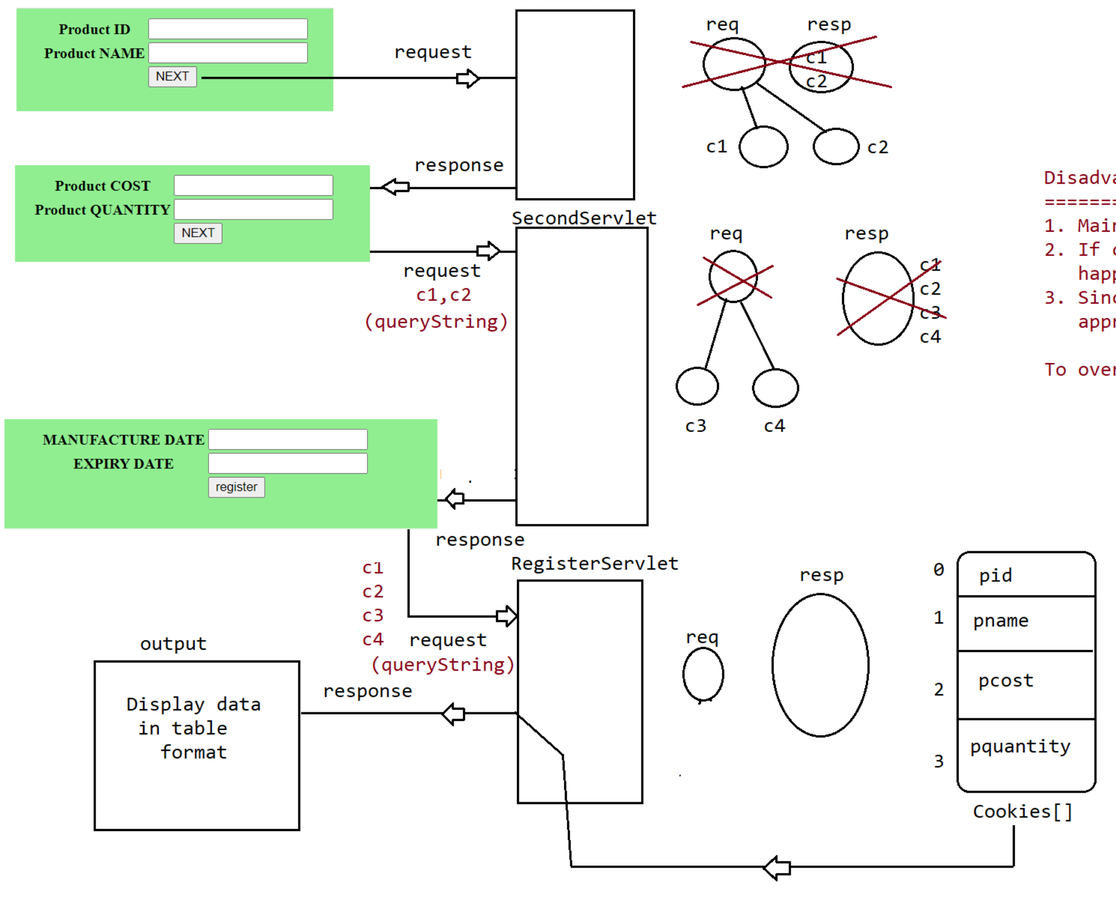
Note: In case of HttpSessionTracking mechanism, if the client disables cookies then HttpSessionTracking mechanism won't work.

HttpSession session = request.getSession()

getSession() -> The container will check whether any HttpSession object existed for particular user or not.
if any httpsession exists then the container will return the existed HttpSession object reference.
if no httpSession exists is existed for particular user then container will create a new HttpSession object and returns the reference.

getSession(false)
-> The container will check whether any HttpSession object existed for particular user or not.
if any httpsession exists then the container will return the existed HttpSession object reference.
if no httpSession exists is existed for particular user then it would return null.

CookieSessionTracking



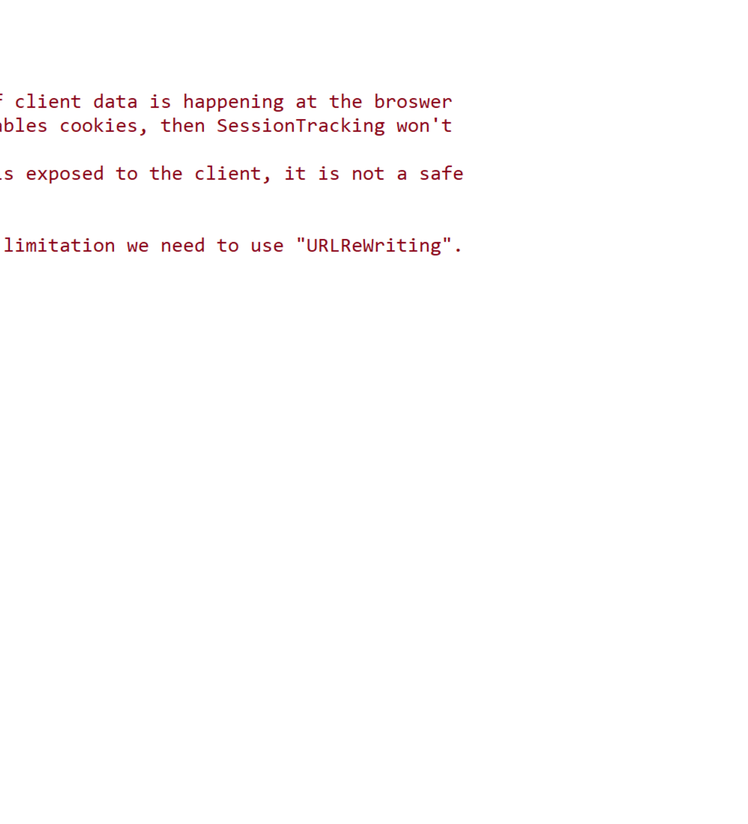
Disadvantage

1. Maintainece of client data is happening at the broswer
2. If client disables cookies, then SessionTracking won't happen.
3. Since cookie is exposed to the client, it is not a safe approach.

To overcome this limitation we need to use "URLReWriting".

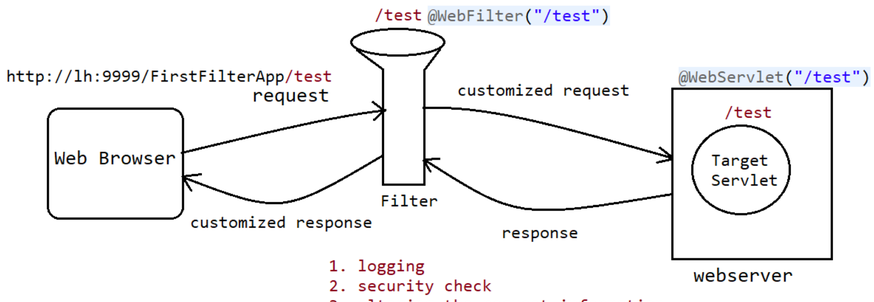
URLReWriting Mechanism

Hidden Form Field



Filter

It can be used for PreProcessing the request and Postprocessing of request before they reach the target resource of the webapplication.



1. logging
2. security check
3. altering the request information
4. compressing the response
5. encryption of response
6. authentication..

Note:

Whenever we are sending the request to Target Servlet, container will check whether any filter is configured for this servlet or not.
if filter is configured then container will forward the request to filter instead of servlet

```
void doFilter(req,resp,chain) throws SE,IOE
```

After executing the above logic, filter forwards the request to TargetServlet.
After processing the request by TargetServlet the response will be forwarded to the Filter instead of browser.
After executing the Filter logic, Filter forwards the total response to the browser.

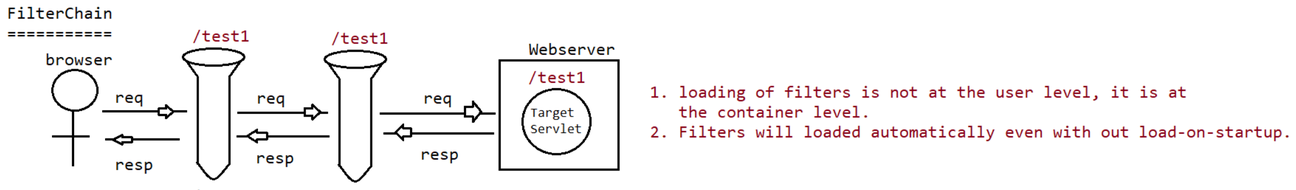
http://localhost:9999/FirstFilterApp/test

This line is added by Demo Filter before procesing the request...

This is the response from Target Servlet

This line is added by Demo Filter after procesing the request...

FilterChain
===========



1. loading of filters is not at the user level, it is at the container level.
2. Filters will loaded automatically even with out load-on-startup.

Log Filter Demo Filter

Order of Execution of Filters in XML
====================================
1. identify all the filters which are configured in url-pattern & execute all these filters from top to bottom
2. identify all the filters which are configured according to servlet-name and execute all these filters from top to bottom

Order of Execution of Filters w.r.t Annotations
===
1. Filters are executed based on the Alphabetical names of the Filter names.

url pattern: http://localhost:9999/FilterChaingApp/test1

This line is added by DemoFilter before processing the request....

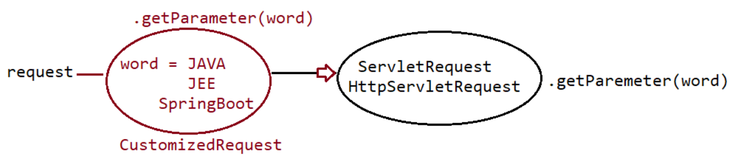
This line is added by the Log filter before processing the request

This is the response from Target Servlet...

This line is added by Log filter after processing the request....

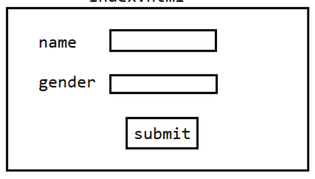
This line is added by DemoFilter after processing the request....

HttpServletRequestWRapper



Listeners
=========

1. Event



Something which happens on a webpage is called as "Event".
It can be mouseclick,typing some text inside textbox,hovering on the page,.....

1. RequestListener
2. ContextListener
3. SessionListener
4. ServletRequestAttributeListener

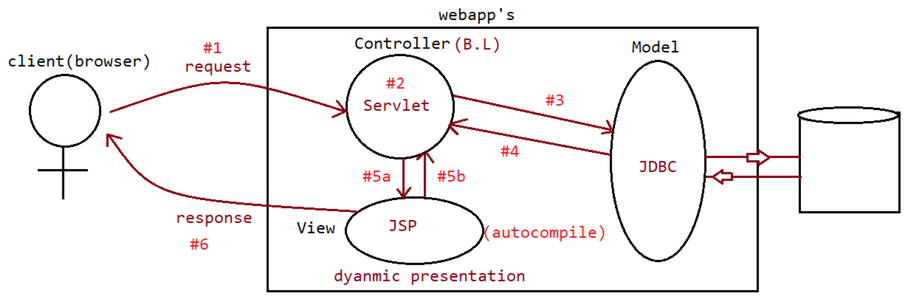
Event =========> class
Listeners======> Interface

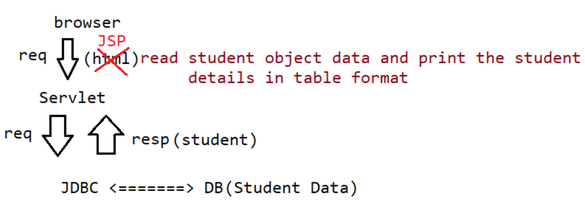
request ============> to particular servlet
SRE
any servlet u send request object should be created.

Do we need mapping?
Ans. no.

ServletReqeustListener

requestInitialized => At the time of request Object creation
requestDestory => At the time of request object destruction

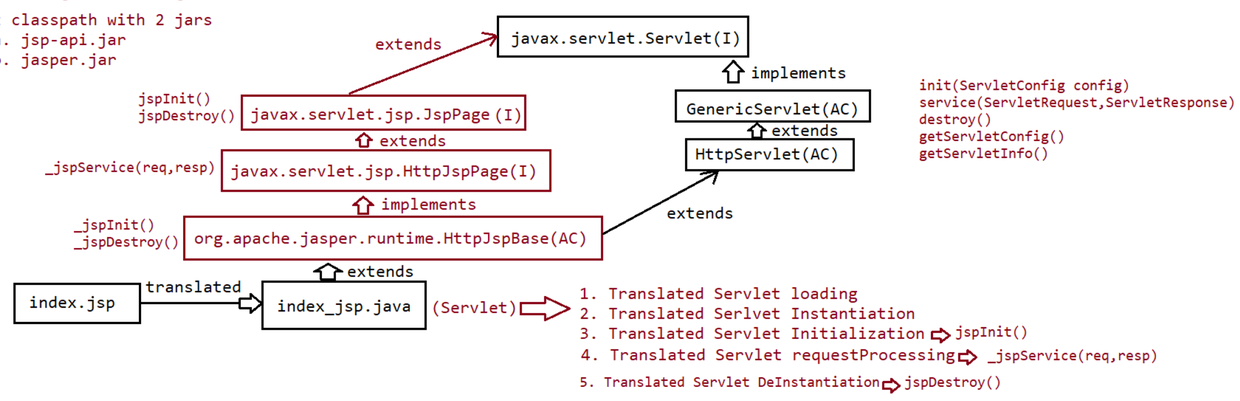




JDBC <=======> DB(Student Data)

How JSP Programs would get executed?

set classpath with 2 jars
a. jsp-api.jar
b. jasper.jar



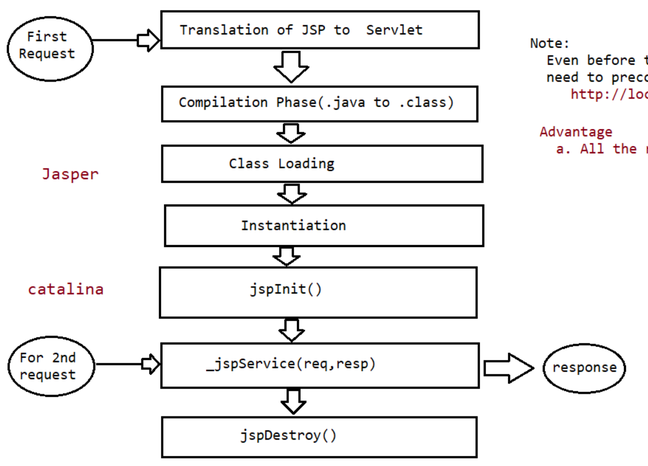
init(ServletConfig config)
service(ServletRequest,ServletResponse)
destroy()
getServletConfig()
getServletInfo()

1. Translated Servlet loading
2. Translated Serlvet Instantiation
3. Translated Servlet Initialization ⇨ jspInit()
4. Translated Servlet requestProcessing ⇨ _jspService(req,resp)
5. Translated Servlet DeInstantiation ⇨ jspDestroy()

location of translated servlet
C:\Tomcat 9.0\work\Catalina\localhost\SecondApp\org\apache\jsp\index_jsp.java

Note: _symbol represents that this method is generated automatically by JSP Engine and we cannot write explicitly

Life Cycle of JSP



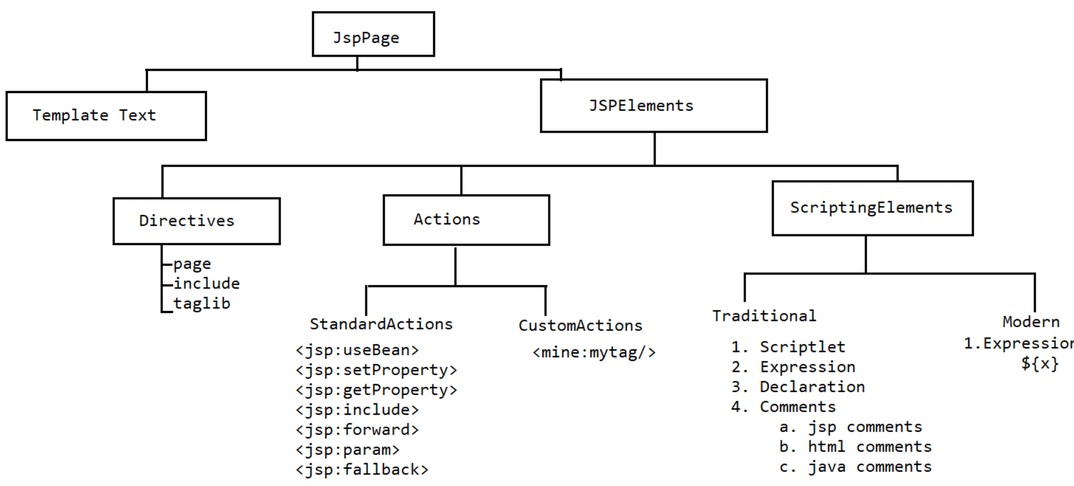
Jasper

catalina

Note:
Even before the request comes, if we want the translation phase to complete then we need to precompile the jsp page as shown below.
http://localhost:9999/SecondApp/index.jsp?jsp_precompile=true

Advantage
a. All the request will be processed with uniform response time.

JSP Elements



<jsp:useBean>
<jsp:setProperty>
<jsp:getProperty>
<jsp:include>
<jsp:forward>
<jsp:param>
<jsp:fallback>

<mine:mytag/>

1. Scriptlet
2. Expression
3. Declaration
4. Comments
   a. jsp comments
   b. html comments
   c. java comments

1.ExpressionLanguage
${x}

Template Text
==========
It contains plain text data and html tags
For template text no processing is requried and it will become argument to write() method in
_jspService(req,res) method

index.jsp
=========
<h1>The server time is <%=new java.util.Date()%></h1>

⇩ Translated servlet

```
public final class index_jsp extends .....
{
    _jspService(request,response){
             out.write("The server time is ");
             out.println(new java.util.Date())
      }
}
```

write() => it takes only character data
println() => it can take any type of argument

Directives
==========
1. page directive

In the current jsp page if we want to define import statements,present jsp page characterstics then we need to go for page directive.
syntax:
<%@ page [attribute-list]%>

a. language ='java'
This is the default value of language attribute

b. contentType = 'text/html'
This is the default value of contentType attribute
As per the page requirement we can change the values too.

c. import = ''
This is the only attribute which can be repeated in the following ways
eg: <%@ page import = '' import = '' %>
<%@ page import = 'java.io.*,java.util.Date'%>
or
<%@ page import = 'java.io.*'%>
<%@ page import = 'java.util.*'%>

The default values of import is
java.lang.*
javax.servlet.*;
javax.servlet.http.*;
javax.servlet.jsp.*;

d. extends
The default value of extends in "HttpJspBase".

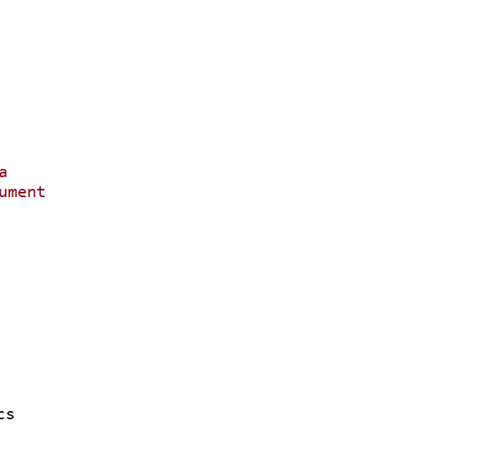
e. info
The default value of this attribute is "Jasper JSP2.3 Engine"
This value can be obtained inside the servlet by making a call to method called "getServletInfo()".

f. buffer, autoFlush

buffer => jsp internally maintains a buffer to write the data to the console as the response.
The default size of the buffer is 8kb.

autoFlush => It is a boolean attribute which is used to indicate to the container whether to flush the response to client automatically or not.
if autoFlush is true, then container will flush the complete response to the client from the buffer.
if autoFlush is false, then container will raise an Exception when the Buffer is filled with response.

java.io.IOException: Error: JSP Buffer overflow



```
<% page  [attributeName = attributeValue] %>
```

```
 isErrorPage = true
  errorPage  = name of jsp file

errorPage => This attribute is used to specify if exception occurs to which page the exception
             object should be delegated.

isErrorPage => This attribute takes a default value as false, meaning the exception object would not be
               available inside this page to handle, to make the exception object available to jsp page
               we need to set the boolean value to true.
```

```
 Servlet ====>  Processing logic

 JSP     ====>  Presentation logic(in dynamic fashion)
                          ⇧
                         Java
```

JSP ===> frontend
Main intention of JSP is to bring frontend developers(HTML/CSS) also to build webapplication using Java(with zero knowledge of Java).

JSP => No Java, but to bring dynamic nature we need to write Java code
To resolve this problem they used a new approach called "ExpressionLanguage"

```
isELIgnored = "true"  => syntax won't be processed rather it treats as Template Text.
isElIgnored = "false" => syntax will be processed and it prints the value
```

Note: default value is false.

```
<%@ page isELIgnored = "false"%>

<h1>
     The userName is :: ${param.user}<br/><br/>
     The password is :: ${param.password}<br/><br/>
</h1>
```

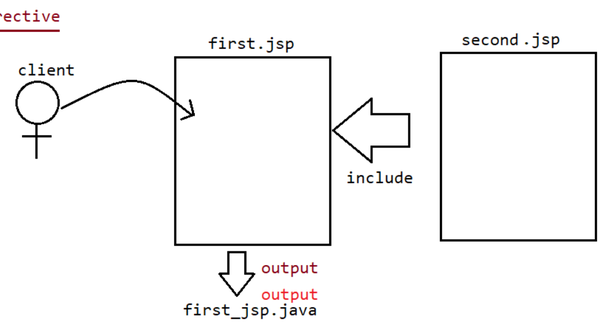
```
http://localhost:9999/SecondApp/index.jsp?user=sachin&password=tendulkar
 output
     The userName is sachin
     The password is tendulkar
```

session

⇩ false means in the current jsp page session object is not accesible,
default value is true

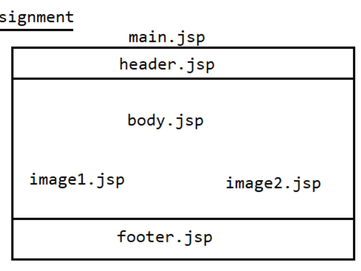
```
<%@ page session = "false" %>
<%
     session.setAttribute("Name","iNeuron");
     session.setAttribute("Java","NavinReddy");
%>
<h1>The name of the company  is :: <%= session.getAttribute("Name") %></h1><br/>
<h1>The trainer name of java is :: <%= session.getAttribute("Java") %></h1>
```

include directive



The content of second.jsp will be included in the current jsp during translation phase.
since the inclusion is happening at the translation phase we call such inclusion as "static inclusion".

Assignment



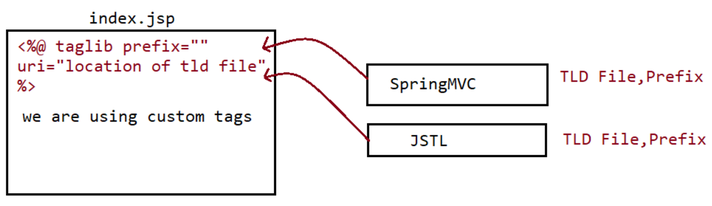
taglib directive

customization

```
<mine:iNeuron>
    //body of the tag
</mine:iNeuron>
```

1. TLD file(Tag library descriptor)
2. Prefix

For few operations to be made easy for developers, the customized tags are already been coded by 3rd party vendor like SpringPivotal team,JSTL libraries by 3rd parties and so on....



Scripting Elements
  a. Declartive tag
  b. Scriptlet tag
  c. Expression tag
  d. comments
       a. html comments
       b. jsp comments
       c. java based comments

Comments
  a. JSP comments <%--JSP COMMENTS --%>
  b. HTML comments <!-- HTML COMMENTS -->
  c. Java comments
       1. // single line
       2. /*
             multiline
          */
       3. /**
             java doc
          **/

Comments visiblity

1. JSP Comments
Visible only in jsp page
Translated servlet not Visilbe

2. HTML Comments
Visible in JSP page
Visible in Translated servlet also

3. Java Comments
Visible in JSP page
Visible in Translated servlet also

Expression Tag

```
Syntax:
<%= x %>   ⇨   _jspservice()
               {
                    out.print(x);
               }
```

Scriptlet Tag

```
Syntax:
  <%
     //write java code   ⇨   _jspservice()
  %>                          {
                                 //java code
                              }
```

Delcartive Tag

```
Syntax:
  <%!
    //Any java declarations  ⇨  The logic will be placed inside the servlet
                                 but outside  _jspservice(,,,)
  %>
```

Note: Inside JSP, there are 9 implicit objects available and these objects are local to _jspService() so these variables directly can't be used inside Declarative Tag.

Implicit Objects

1. request   => HttpServletRequest
2. resposne   => HttpServletResponse
3. out   => JspWriter(AC)
4. config   => ServletConfig
5. application   => ServletContext
6. exception   => java.lang.Throwable
7. session   => HttpSession
8. page   => java.lang.Object
9. pageContext   => java.servlet.jsp.PageContext(AC)

Note:

```
<h1>
     <%!
          int i = 0;
     %>

     <%
          i++;
          out.println(i);
     %>
</h1>
```

ouput

i = 1,2,3,4,5,....

```
<h1>
     <%
          int i =0;
          i++;
          out.println(i);
     %>
</h1>
```

output
i = 1 (always)

Program to demonstrate the usage of all Scripting elements in JSP

```
<%@ page import = "java.util.Date" %>

<%!
     Date d = null;
     String date = "";
%>

<%
     d =new Date();
     date = d.toString();
%>
<h1 style= 'color:red;'>
     Today date is <%= date%>
</h1>
```

declartive tag
scriptlet
expression

```
import java.util.Date;
public class index_jsp extends .....
{
   Date d =null;
   String date ="";

  public void _jspService(,,,)
  {
       d = new Date();
       date = d.toString();

      out.write("<h1 style='color:red;>");
      out.write("Today date is ");
      out.print(date);
       out.write("</h1>");

   }
}
```

web.xml

```
<web-app>
     <display-name>JSP Implicit Object Application</display-name>
     <context-param>
          <param-name>User</param-name>
          <param-value>root</param-value>
     </context-param>
     <context-param>
          <param-name>Password</param-name>
          <param-value>root123</param-value>
     </context-param>
</web-app>
```

Context Object data

index.jsp

```
<h1>
     The context parameter username is ::
          <%= application.getInitParameter("User")%><br/>
     The context parameter password is ::
          <%= application.getInitParameter("Password")%><br/>
     The Application name is ::
          <%= application.getServletContextName()%>
</h1>
```

web.xml

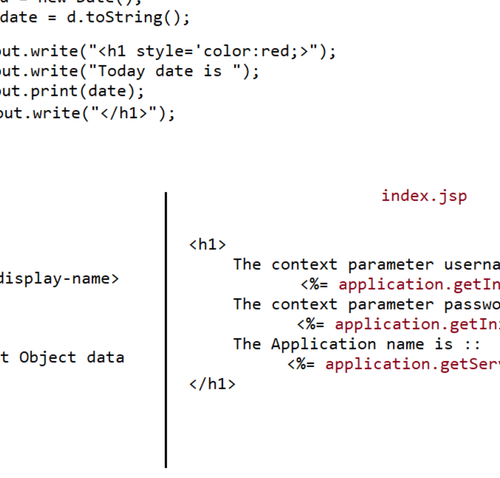
```
<web-app>
     <servlet>
          <servlet-name>Demo</servlet-name>
          <jsp-file>/config.jsp</jsp-file>
          <init-param>
               <param-name>IPL</param-name>
               <param-value>BCCI</param-value>
          </init-param>
     </servlet>
     <servlet-mapping>
          <servlet-name>Demo</servlet-name>
          <url-pattern>/test</url-pattern>
     </servlet-mapping>

</web-app>
```

```
<h1>
     The Init param is :: <%= config.getInitParameter("IPL")%><br/>
     The logical name of Servlet is :: <%= config.getServletName()%><br/>

     <%
          java.util.Enumeration<String> params=config.getInitParameterNames();
          while(params.hasMoreElements()){
                   String data = (String)params.nextElement();
                   System.out.println(data +"::" + config.getInitParameter(data));
          }
     %>
     Context Name is <%=config.getServletContext()%>
</h1>
```

pageContext
==========
  1. We can get remaining implicit object using pageContext object
  2. To perform requestDispatching mechanisim
       a. include(req,resp)
       b. forward(req,resp)
  3. To perform attribute management in any scope

Note:

```
request      = pageContext.getRequest()
response     = pageContext.getResponse()
config       = pageContext.getServletConfig()
application  = pageContext.getServletContext()
session      = pageContext.getSesion()
out          = pageContext.getOut()
exception    = pageContext.getException()
page         = pageContext.getPage()
```



Scopes in JSP

1. Request Scope
2. Session Scope
3. Application Scope
4. Page Scope

⇦ Servlet Scope

Request Scope

In Servlet this scope is mainitained by ServletRequest Object,in jsp it is maintained by "request" implicit object.
The information stored in request scope is available for all components which are processing that request.
Request Scope => it will start at the time of request object creation.(before calling service())
it would end at the time of request object destruction(after completing service()).

Session Scope

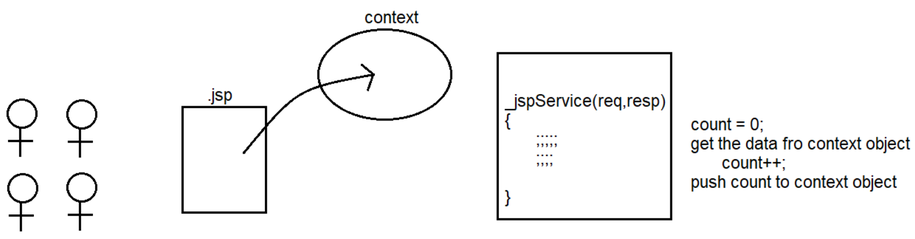
In Servlet this scope is maintianed by "HttpSession" object, in jsp it is maintained by "session" implicit object.
The information stored in session scope is available for all components which are participating in session.
Session Scope => It will start at the time of session object creation
It will end at the time of session expires(logout or invalidate()) or timeout mechanism.

Application Scope

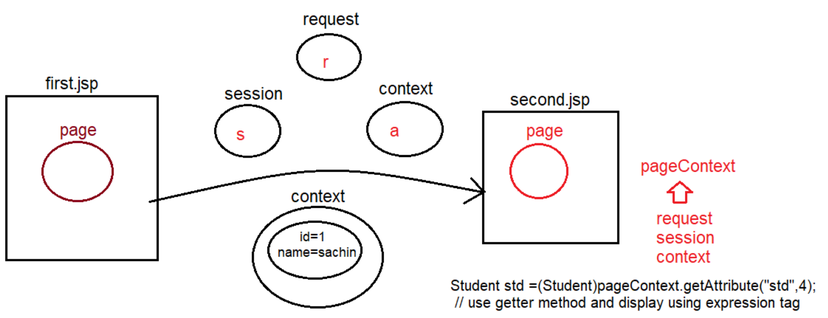
In Servlet this scope is maintained by "ServletContext" object,but in jsp it is maintained by "application" implicit object.
The information stored in aplicaiton scope is available for all the components of the webapplication.
Application Scope = > it will start at the time of ServletContext Object(at the time of server startup) construction
it will end at the time of Servlet Context Object destruction(at the time of undeployment or server shutdown)

Page Scope

This scope is not applicable for Servlet, it is applicable only for JSP.
In JSP this scope is managed by "pageContext" implicit object
The information stored in pagescope is available only in the current jsp page, and not available for other jsp's.



Usage of pageContext Object



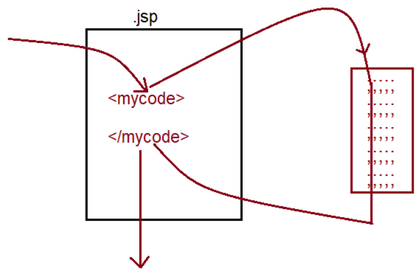
Difference b/w findAttribute(name) vs getAttribute(name)

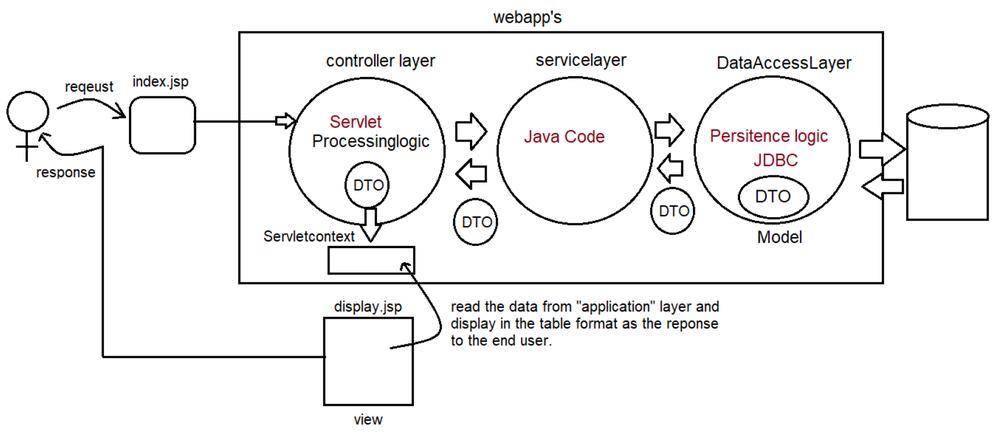
getAttribute(name) => by default it will check in page Scope, unitll explicitly we tell through SCOPE.

getAttribute(name,scope) =>it will check in the respective scope, and if it is availale it would return the value otherwise it would return null.

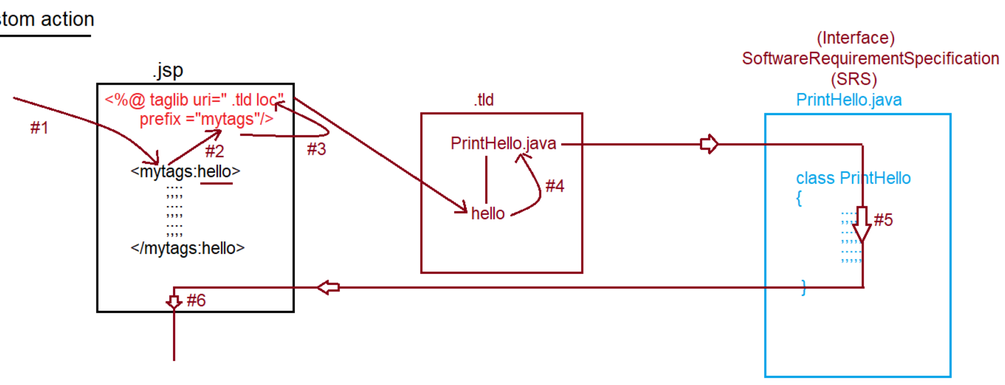
findAttribute(name) => first it will check in page scope, followed by request scope, sessionscope and application scope in any one of this scope
if the object is avaialbe it would return the value otherwise it would return null.

JSP Actions





Custom action



To implement simple if else, loop statements we have to provide lot of java code internally w.r.t interfaces supplied by SUNMS for custom actions to take palce in jsp code.

To Overcome this JSP technology has provided a seperate tag library for simple java syntax implementation and frequently used operations.
JSTL is an abstraction given by SUNMS and its implementation is given by Server vendors.

⇧
tomcat implemented jars in the following location
⇧
C:\Tomcat 9.0\webapps\examples\WEB-INF\lib

Application scope
==========
Write a code using applicationscope to print hit count of the application

```
<%
            Integer count=(Integer)application.getAttribute("hitcount");
            if(count == null)
                  count = 1;
            else
                  count++;

            application.setAttribute("hitcount", count);
      %>
<h1 style = 'color:red;'>Hit count of the application is :: <%=count %></h1>
```

Write a code using application scope to count no of users login to the application
     users => track through session.

```
<%@ page  session="true"%>
<%
      Integer count = (Integer) application.getAttribute("usercount");
      if (session.isNew()) {
            if (count == null)
                  count = 1;
            else
                  count++;
      }
      application.setAttribute("usercount", count);
%>
<h1 style='color: red;'>Hit count of the application is ::<%=count%></h1>
```

Write a code to display the no of requests in current session?
      session -> hold the data uniquely w.r.t user, so keep it in session object.

```
<%
      Integer count = (Integer) session.getAttribute("sessionRequestCount");
      if (count == null)
            count = 1;
      else
            count++;
      session.setAttribute("sessionRequestCount", count);
%>
<h1 style='color: red;'>Hit count of the application is ::<%=count%></h1>
```

Note: In all the above programs,initally the all the variables would not be available in the respective object, so null value will
          be returned, based on the condition the variable would be created with the respective values and stored back in
          the respective objects.

To retrieve the value from the PageContext object w.r.t to the scope we need to use the following methods
      a. pageContext.getAttribute(String name, int scope);

Scope levels
==========
PAGE_SCOPE = 1
REQUEST_SCOPE = 2
SESSION_SCOPE = 3
APPLICATION_SCOPE = 4

Demonstrate the need of pageContext object

```
====================================
first.jsp
=====
<%
      pageContext.setAttribute("p", "page");
      request.setAttribute("r", "request");
      session.setAttribute("s", "session");
      application.setAttribute("a", "application");

      pageContext.forward("second.jsp");
%>

second.jsp
=========
Page Scope attribute :: <%= pageContext.getAttribute("p",1)%><br/>
Request Scope attribute ::<%= pageContext.getAttribute("r",2)%><br/>
Session Scope attribute :: <%= pageContext.getAttribute("s",3)%><br/>
Application Scope atribute :: <%= pageContext.getAttribute("a",4) %>

Output
Page Scope attribute :: null
Request Scope attribute :: request
Session Scope attribute :: session
Application Scope atribute :: application

usage of findAttribute(string name)
============================
<%
      pageContext.setAttribute("page", "page");
      request.setAttribute("request", "request");
      session.setAttribute("session", "session");
      application.setAttribute("application", "application");
%>
<h1>Find Attribute ::<%=pageContext.findAttribute("a")%></h1>


JSP Actions
=========
In JSP technology, using scripting elements we are able to provide java code inside
jsp pages.
As per the theme of JSP writing java code is not allowed.
=> To eleminate java code from jsp pages we need to use "JSP Actions".
=> In JSP actions we provide Scripting tag in jsp page and we provide java code
w.rt. Scripting tag.

Note:
Whenever container encounters the Scripting tag, then container will execute
respective code by this an action will be
performed which  is called as "JSP Actions".

In JSP we have 2 types of Actions
      a. Standard Actions(supplied by jsp technology only)
      b. Custom Actions(as per the user needs by taking the support of SRS we can
define our own)

Standard Actions
==============
1. <jsp:useBean>
2. <jsp:setProperty>
```

3. <jsp:getProperty>
4. <jsp:include>
5. <jsp:forward>
6. <jsp:scriptlet>
7. <jsp:expression>
8. <jsp:delcaration>

What is java bean?
   It is a normal java class with setters, getters defined for private variables of a class.
   To promote serialziation for a java bean we use an interface called "Serializable".
   It is also called as "POJO".

Standard Actions
===============
```
<jsp:useBean id = "name of the reference "  scope="[scopes of jsp]"
                        class="name of the class for which object should be created"/>
      X idvalue=(X)Class.forName([supplied value in class]).newInstance();


<jsp:setProperty property ="" name = "" value = ""/>
            name.setPropertyValue(value supplied);
<jsp:getProperty property="" name = "" />
            name.getPropertyValue()

eg:
<jsp:useBean id="student" class="in.ineuron.bean.Student" scope="page">
      <jsp:setProperty property="id" name="student" value="10" />
      <jsp:setProperty property="name" name="student" value="sachin" />
      <jsp:setProperty property="address" name="student" value="MI" />
      <jsp:setProperty property="age" name="student" value="49" />
</jsp:useBean>
<jsp:getProperty property="id" name="student"/>

input.html
--------------
<table>
                  <tr>
                              <th>ID</th>
                              <td><input type='text' name='id' /></td>
                  </tr>
                  <tr>
                              <th>NAME</th>
                              <td><input type='text' name='name' /></td>
                  </tr>
                  <tr>
                              <th>AGE</th>
                              <td><input type='text' name='age' /></td>
                  </tr>
                  <tr>
                              <th>ADDRESS</th>
                              <td><input type='text' name='address' /></td>
                  </tr>
                  <tr>
                              <th></th>
                              <td><input type='submit' value='reg' /></td>
                  </tr>
```

```
</table>

As notice above inside html page name attribute values and in the bean instance
variable(fieldname) are same
      <input type ='text' name = 'id'/>
      <input type ='text' name = 'name'/>
      <input type ='text' name = 'age'/>
      <input type ='text' name = 'address'/>

class Student
{
            Integer id;
            String name;
            Integer age;
            String address;
}
If both variable names are same, then instead of binding the value to each variable
explicitly  as shown below
<jsp:setProperty property="id" name="student" value='<%=id%>' />
<jsp:setProperty property="name" name="student" value='<%=name%>' />
<jsp:setProperty property="address" name="student" value='<%=address%>' />
<jsp:setProperty property="age" name="student" value='<%=age%>' />

we can bind it automatically using "*"
<jsp:setProperty property="*" name="student" />

2. <jsp:include>=> include request Dispatching mechanism.
    <jsp:forward>=> forward request Dispatching mechanism.
    <jsp:param name = '' value=''">  => To add new values to request object and
send it to the respective page we use
                                                          <jsp:param name = ''
value=''/>.

3. deprecated jsp actions are
      <jsp:plugin>
      <jsp:fallback>
      <jsp:params>

Customactions
============
    These are the actions which are developed by developers as per their
application requirements.
    In jsp two types of tags are available
            a. standardactions -> predefined tags known to container
            b. customactions   -> inform explicitly to the container.

To prepare custom tags in Jsp pages we use the following syntax.
      <prefix_name:tag_name>
                  ;;;;;;
                  ;;;;;;
                  ;;;;;;
                  ;;;;;;
      </prefix_name:tag_name>
If we want to design custom tags in our jsp application, then we use the following
3 elements
      a. jsp page with taglib directive
      b. TLD file(Tag library descriptor)
      c. Tag Handler class.
```

1. JSP => to write view(presentation layer) and to make page dynamic.
   Avoid writing java code as much as possible.

2. EL and JSP Actions
   EL => we can avoid java code,but not able to replace all the functionality of java.
   JSP Actions
   a. standard actions
      tags are limited in no,not all functionalities of java can be promoted.
   b. custom actions
      user should write the tag and its working which is difficult and lengthy for programmer
      a. Inside jsp we need to <%@ taglib uri = "" prefix = "" %>
      b. TLD file
      c. For every tag equivalent "Tag Handler Class".

3. JSTL
   1. Core library(commonly used)
   2. SQL library
   3. Functional library
   4. FMT library ⇦ i18N(internationalization)
   5. XML library(not useful)

Core library

1. General purpose tags	2. Conditional tags	3. Iteration tags	4. URL related tags
<c:out>	<c:if>	<c:forEach>	<c:import>
<c:set>	<c:choose>	<c:forTokens>	<c:url>
<c:remove>	<c:when>		<c:redirect>
<c:catch>	<c:otherwise>		<c:param>

programs pasted in notes

Note:
      To use jstl jar supplied by tomcat vendor we refer to the following location
                  C:\Tomcat 9.0\webapps\examples\WEB-INF\lib

Core Libray
=========
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
   <c:out>
            It is used for writing Template text data and expression to the JSP.
    c=> prefixName
   out => tagName

<c:out value="WELCOME TO JSTL CODING...."/>
The user name is :: <c:out value = "${param.user}"/>
The password is :: <c:out value = "${param.password }" default="Guest"/>

input
 http://localhost:9999/JSTLApp-01/index.jsp?user=Hyder&password=iNeuron

output
      WELCOME TO JSTL CODING....
      The user name is :: Hyder
      The password is :: iNeuron

2. <c:set>
            We can use to set attributes in any scope and to set map and bean
properties also.
eg:
<c:set var="x" value="10" scope="request"/>
<c:set var="y" value="20" scope="request"/>
<c:set var="sum" value="${x+y}" scope="session"/>
<h1 style='color:red; text-align:center;'>
The result is :: <c:out value="${sum}"/>

3. <c:remove>
            To remove attributes in the specified scope we can use this tag.
            if the scope is not specified for removing, by default it will search
in
                  a. page scope
                  b. request scope
                  c. session scope
                  d. application scope

eg::
<c:set var = "x" value="10" scope="page"/>
<c:set var = "y" value="20" scope="page"/>
<c:set var = "z" value="${x+y}" scope="session"/>
<h1 style='color:blue; text-align:center;'>
            The result is :: <c:out value="${z}"/>
</h1>
<c:remove var="x"/>
<c:remove var="y"/>
<c:remove var="z"/>
<h1 style='color:red; text-align:center;'>
      The result is :: <c:out value="${z}" default="1000"/>
</h1>

4.
<c:catch var="">

```
      //risky code
</c:catch>
```
      If any exception occurs, then that exception object is collected inside var attribute vaiable which is page scope.
      if any exception is raised inside risky code, then this tag suppress that exception and rest of the jsp wil be executed
       normally.

eg:
```
<h1 style='color: blue; text-align: center;'>
            UserName is :: ${param.userName}<br />
            <c:catch var="e">
                  <%
                        int age =
Integer.parseInt(request.getParameter("userAge"));
                  %>
                  UserAge  is   :: ${param.userAge }<br />
            </c:catch>

            <c:if test="${e!=null}">
                  oops... Exception raised .... : ${e}<br/>
            </c:if>
            UserHeight is :: ${param.userHeight }
</h1>
```

input
      http://localhost:9999/JSTLApp-01/index.jsp?userName=sachin&userAge=ten&userHeight=5.5
output
      UserName is :: sachin
            oops... Exception raised .... : java.lang.NumberFormatException: For input string: "ten"
      UserHeight is :: 5.5

input
      http://localhost:9999/JSTLApp-01/index.jsp?userName=sachin&userAge=49&userHeight=5.5
output
      UserName is :: sachin
      UserAge is :: 49
      UserHeight is :: 5.5

Conditional Tags
==========
1. <c:if>
       It is used to implement core java if statement

```
      <c:if test="" scope="" var="">
            //body of if
      </c:if>
```
      if the condition evaluates to true only then body of if will be executed,otherwise the remaining statement present
      in jsp page will be executed.

eg:
```
<c:set var="x" value="10"/>
<c:set var="y" value="20"/>
<c:if test="${x<y }" var="result">
```

```
      X value is ${x}<br/>
      Result is ${result}
</c:if>
<c:if test="${x eq 10 }">
      X is  equal to 10
</c:if>

output
X value is 10
Reslut is true
X is equal to 10

2.  <c:choose>, <c:when> and <c:otherwise>
            We can use these tags for implementing if else and switch statements.

Implementing if-else
=================
<c:choose>
      <c:when test = "condition">
            ACTION-1
      </c:when>
      <c:otherwise>
            ACTION-2
      </c:otherwise>
</c:choose>
            if condition evaluates to tree then ACTION-1 otherwise ACTION-2

Implementing switch
=================
<c:choose>
      <c:when test = "test_condition1">
            ACTION-1
      </c:when>
      <c:when test = "test_condition2">
            ACTION-2
      </c:when>
      <c:when test = "test_condition2">
            ACTION-3
      </c:when>
            ;;;;
      <c:when test = "test_conditionN">
            ACTION-2
      </c:when>
      <c:otherwise>
            Default Action
      </c:otherwise>
</c:choose>

Note:
 1. <c:when> tag explicitlly contains break statement, so no chance of fall through
in switch.
 2. <c:otherwise> should always be last case only
 3. <c:choose> should compulsorily contain one <c:when> tag, but <c:otherwise> is
optional.


eg:
<h1>
            Select one number
```

```
            <form action="./index.jsp">
                  <select name="combo">
                        <option value='1'>1</option>
                        <option value='2'>2</option>
                        <option value='3'>3</option>
                        <option value='4'>4</option>
                        <option value='5'>5</option>
                        <option value='6'>6</option>
                        <option value='7'>7</option>
                        <option value='8'>8</option>
                        <option value='9'>9</option>
                  </select> <input type='submit' />
            </form>

            <c:set var='day' value='${param.combo }' />
            <c:choose>
                  <c:when test="${day==1 }">
                        SUNDAY
                  </c:when>
                  <c:when test="${day==2 }">
                        MONDAY
                  </c:when>
                  <c:when test="${day==3 }">
                        TUESDAY
                  </c:when>
                  <c:when test="${day==4 }">
                        WEDNESDAY
                  </c:when>
                  <c:when test="${day==5 }">
                        THURSDAY
                  </c:when>
                  <c:when test="${day==6 }">
                        FRIDAY
                  </c:when>
                  <c:when test="${day==7 }">
                        SATURDAY
                  </c:when>
                  <c:otherwise>
                        SELECT NUMBER BETWEEN 1 to 7
                  </c:otherwise>
      </c:choose>
</h1>

Iteration tags
===========
1. <c:forEach  begin="" end="" step="">
            It would ressamble general purpose for loop.
            default value of step is "1", it gets incremented automatially.
            The loop body will be executed w.r.t  "begin<=end".
eg:
<c:forEach begin="1" end="10" step="2" var="count">
      <h1>Learning JSTL is very easy..${count}</h1>
</c:forEach>

eg:
<%
      String[] names = {"sachin","saurav","dhoni","kohli"};
      pageContext.setAttribute("names", names);
%>
```

```
<c:forEach  items="${names}" var="obj">
      <h1>The data is :: ${obj }<br/></h1>
</c:forEach>

output
The data is :: sachin
The data is :: saurav
The data is :: dhoni
The data is :: kohli

Note:
This is similar to
      for(String name: names)
            System.out.println(name);

2. <c:forTokens>
            It is a specialized version of forEach to perform StringTokenization
based on some delimitor.

syntax
      <c:forTokens items = "" delims="" var="" begin="" end="" step="">
                  //body
      </c:forTokens>

eg:
<c:forTokens items="Sachin,Saurav,Dhoni,Dravid" delims="," var="name">
            <h1>The name is :: ${name}</h1><br/>
</c:forTokens>

eg:
<c:forTokens items="One,Two,Three,Four,Five,Six,Seven" delims="," var="data"
begin="2" end="5" step ='2'>
            <h1>The result :: ${data}</h1><br/>
</c:forTokens>

output
The result :: Three
The result :: Five

eg:
<%
            ArrayList<String> al = new ArrayList<String>();
            al.add("sachin");
            al.add("dhoni");
            al.add("kohli");
            al.add("dravid");
            al.add("rahul");
            pageContext.setAttribute("names", al);
%>
<c:forEach  items="${names}" var="name">
            <h1>${name }</h1>
</c:forEach>

Note:
      <c:forTokens> items attribute should be string only.
      <c:forEach> items attributes can be String,Collection object,Map etc.

URL related tags
----------------------
```

1. <c:import>
we can use this tag for importing the response of the other pages in the current page response at the time of
request processing.(ie dynamic include)

eg:
first.jsp
```
<h1>Welcome to iNeuron+Physics Wallah</h1><br/>
<c:import url="second.jsp" />
```

second.jsp
```
      <h1>The free videos are available in www.youtube.com/navinreddy</h1>
```

eg:
As noticed below the ouput of <c:import> is copied into the variable, so in the current jsp where ever the ouput is required
we can just refer to that variable.

first.jsp
```
<h1>Welcome to iNeuron+Physics Wallah</h1><br/>
      <c:import url="second.jsp" var = "x" scope="request" />
      ${x} <br/>
      ${x} <br/>
      ${x} <br/>
      ${x} <br/>
```

eg:
first.jsp
```
<h1>Welcome to iNeuron+Physics Wallah</h1><br/>
      <c:import url="second.jsp" >
            <c:param name="java" value="hyder"/>
            <c:param name="jee" value="nitin"/>
            <c:param name="spring" value="navinreddy"/>
      </c:import>
```

second.jsp
```
<h1>The free videos are available in www.youtube.com/navinreddy</h1><br/>
<h1>Trainer name for java is :: ${param.java }</h1>
<h1>Trainer name for Jee is  :: ${param.jee }</h1>
<h1>Trainer name for spring is  :: ${param.spring }</h1>
```

2. <c:redirect>
This is similar to sendRedirect() method of ServletResponse.

eg:
```
<h1>Welcome to iNeuron+Physics Wallah</h1><br/>
      <c:redirect url="second.jsp" >
            <c:param name="java" value="hyder"/>
            <c:param name="jee" value="nitin"/>
            <c:param name="spring" value="navinreddy"/>
      </c:redirect>
```

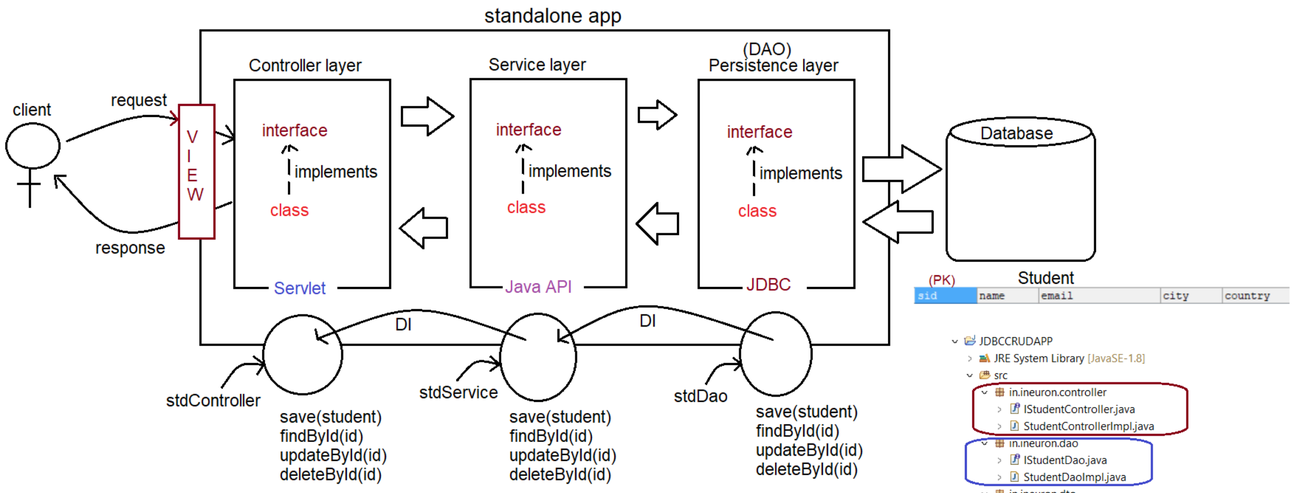
second.jsp
```
<h1>The free videos are available in www.youtube.com/navinreddy</h1><br/>
<h1>Trainer name for java is :: ${param.java }</h1>
<h1>Trainer name for Jee is  :: ${param.jee }</h1>
<h1>Trainer name for spring is  :: ${param.spring }</h1>
```

input

```
      http://localhost:9999/JSTLApp-01/second.jsp?
java=hyder&jee=nitin&spring=navinreddy

output
The free videos are available in www.youtube.com/navinreddy
Trainer name for java is :: hyder
Trainer name for Jee is :: nitin
Trainer name for spring is :: navinreddy

3. <c:url>
   This would attach jsessionid and the query parameters to the url.

first.jsp
======
<c:url value="second.jsp" var="x" scope='request'>
            <c:param name="java" value="hyder" />
            <c:param name="jee" value="nitin" />
            <c:param name="spring" value="navinreddy" />
</c:url>
<h1>The modified url is :: ${x}</h1>
<a href="${x }">Click here to go to Next Page...</a>

second.jsp
=========
<h1>The free videos are available in www.youtube.com/navinreddy</h1><br/>
<h1>Trainer name for java is :: ${param.java }</h1>
<h1>Trainer name for Jee is  :: ${param.jee }</h1>
<h1>Trainer name for spring is  :: ${param.spring }</h1>

SQLTags
=======
      Code related to JDBC.

1. <sql:setDataSource>-> to create datasource object
2. <sql:query> -> to perform select operations
3. <sql:update>-> to perform non select operations(insert,update,delete)
4. <sql:param> -> to inject the values for preparedStatement object
5. <sql:dateParam> -> to inject data values in case of preparedstatement.

Code to perform select operation using jstl
===================================
<sql:setDataSource var="ds" url="jdbc:mysql:///enterprisejavabatch"
            driver="com.mysql.cj.jdbc.Driver" user="root" password="root123" />
<sql:query var="result" dataSource="${ds}">
            select * from student
</sql:query>
<h1>
      <c:forEach items="${result.rows}" var="row">
            ${row.sid }||${row.name }||${row.email }||${row.city }||$
{row.country}<br/>
      </c:forEach>
</h1>

Code to perform insert operation using jstl
================================
<sql:setDataSource var="ds" url="jdbc:mysql:///enterprisejavabatch"
            driver="com.mysql.cj.jdbc.Driver" user="root" password="root123" />
      <sql:update dataSource="${ds}" var="count">
            insert into student(`name`,`email`,`city`,`country`)values(?,?,?,?)
```

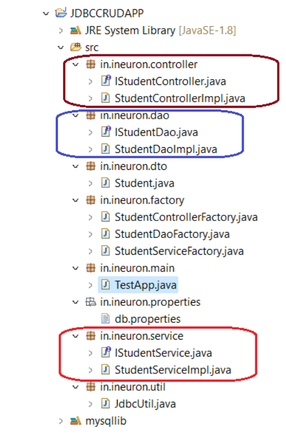
```
            <sql:param value="pandya" />
            <sql:param value="pandya@gmail.com" />
            <sql:param value="GT" />
            <sql:param value="IND" />
      </sql:update>
<h1>The no of rows affected is :: ${count}</h1>
```

Note:
  It is not a good practise to write persistence logic(jdbc code) inside jsp using jstl library,becoz jsp is meant for view part that
  is presentation purpose.

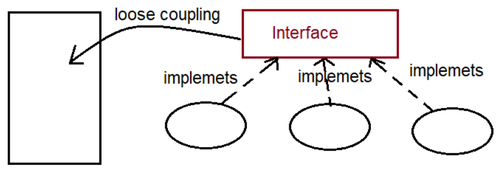


(PK) Student

sid	name	email	city	country

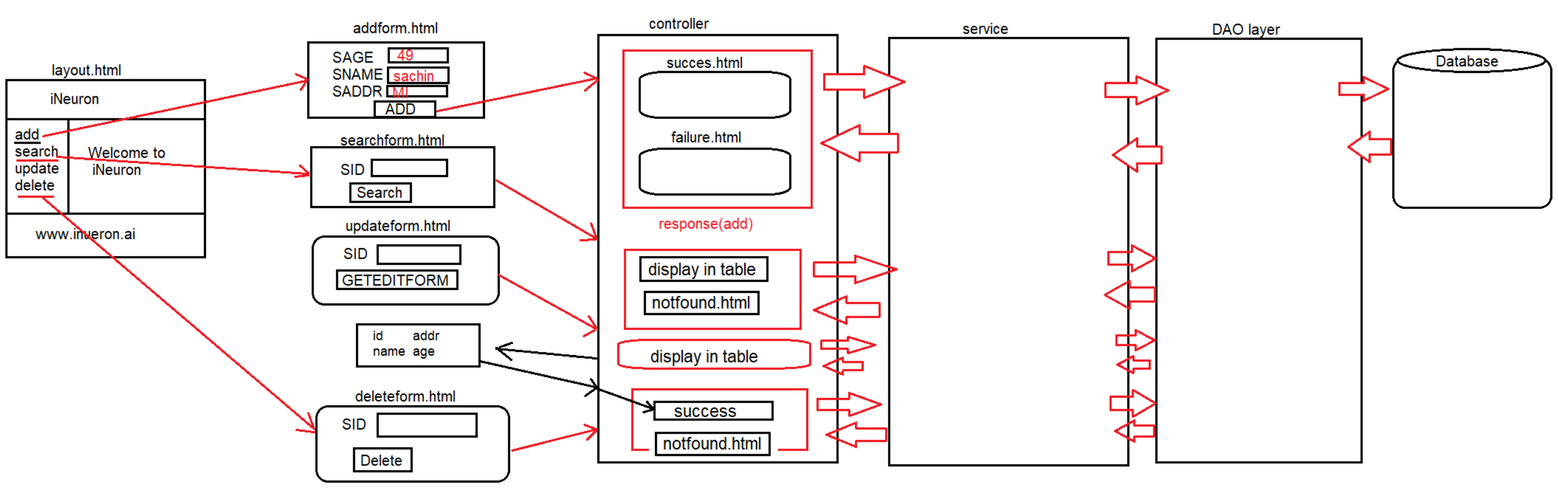


Creation of an object for a class implementing an interface is done through "Factory Design pattern".
Using this we are achieving "Abstraction".



PROJECT-2

CRUD OPERATION using DesignPattern and layerd approach
a. use hikaricp connection pool.
b. use statement object to perform operations.
c. use DTO to transfer the object b/w layers.
d. follow POJI-POJO implementation



. => current location searching
.. => go one step backward and start to search

./* ======> any type of request
eg: http://lh:9999/App/test
http://lh:9999/App/demo
http://lh:9999/App/disp

/controller/*===> request for any type under contoller
eg: http://lh:9999/App/controller/test
http://lh:9999/App/controller/demo
http://lh:9999/App/controller/disp