Java 2 Enterprise Edition (J2EE)

Reference Material on J2EE technology

J2EE Contents:

Chapter 1

XML

Chapter 2

HTTP
Servlets
JSP

Chapter 3

Java Beans
Struts
EJB

Chapter 4

JMS
Java Mail …

Chapter 5

Design Patterns

Chapter 6

Required Concepts

XML

When we create .rtf document apart from saving the actual info the tool saves additional info like start of a paragraph, bold, size of the font .. Etc. This info will be used by the tools to display the actual info.

In html also we use several tags to tell the browser how the data has to be display.

In html we use tags like H, B, FONT, TABLE… etc to tell how the data has to be formatted.

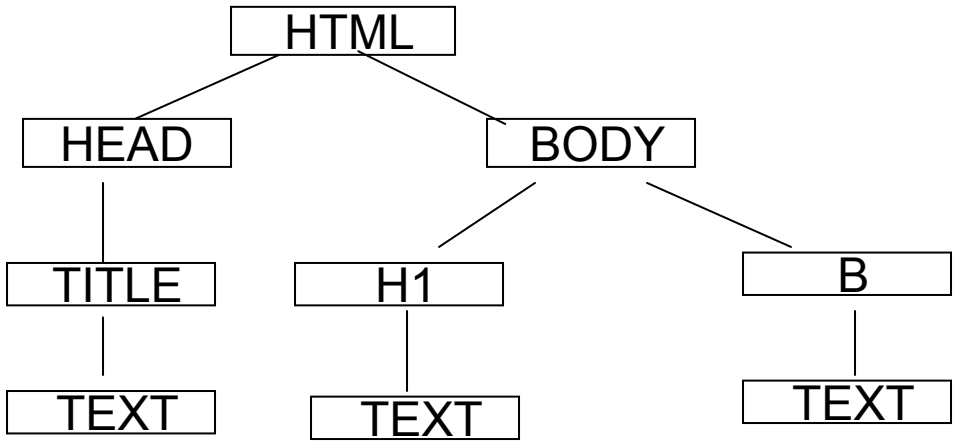
As part of html pages we can use an entity like nbsp, copy, lt, gt, reg .. Entities can be used for replacement.

By looking at html tags we can say how the data has to be displayed but we can’t understand the semantics of the data.

DOM TREE

Every html document can be represented as DOM tree (document object model)

DOM tree :



All the boxes are different objects representing various html elements. Object can be called as nodes.

Various scripting languages like java script s supports the DOM.

In xml we can use a DOM parser to read the document and create the objects representing a tree.

Validation of XML

There are so many tools available that can validate the html documents. www.w3c.org provides a facility to validate the html files.

Meta languages like sgml and xml can be used to define a new language like html.

Xml can be used to define our own language, we can define our own tags either by using DTD(document type definition) or xml schema standard.

A well form xml document is a document containing at least one element.

- For every start tag there must be an end tag.
- All the elements must be nested properly.
- All the attribute values must be placed in quotes.

A valid xml document is a well formed xml document and it is written according to xml schema or xml DTD.

An element in xml can have attributes, element content, simple content, empty content or mixed content.

An empty element can be written either as

<abc atr1="value"> </abc>

(or)

<abc atr1="value" />

Attributes are used to provide additional info about the data

Limitations on Attributes :

- Can’t contain multiple values
- Not easily expandable
- Can’t describe structures
- More difficult to manipulate programmatically
- Values are not easy to test against a DTD

Most of the xml authors uses attributes to store meta data

In xml documents we can use standard entities like <, > …etc

DTD

Xml DTD can be used to define our own language.

In a DTD we define

- Root of the document
- The content of an element
- The sequence in which the elements has to appear
- Define the entities and attributes of an element

Ex :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT page (title+, content, comment?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT content (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
```

+ one or more

? Zero or one

* Zero or more

PCDATA or CDATA contains characters or simple content

We use ELEMENT to define content of an element

<! ELEMENT element name (content)>

To define attributes we can use,

<! ATTLIST element name attribute name attribute type default value>

In ATTLIST we can specify list of valid values

We need to specify whether the document is valid or not according to DTD

Name Spaces

We can directly have DTD as part of an xml document called internal DTD.

A DTD can be external.

In the xml file we can refer to an external DTD using DOCTYPE as,

<! DOCTYPE root element system “file.dtd”>

A name space is a set of unique names.

In java we use a package to provide a name space ex :

Since both of them are two different name spaces we can have the same class name socket.

In above case also there will be ambiguity if we refer to names directly. To resolve the ambiguity we can use the fully qualified names ex : namespacename followed by name in namespace

com.inet.package.socket

While debugging the tags we can define the name of the namespace

In xml we can write the fully qualified name as,

xyz:Element where xyz is namespace prefix

According to xml specification the name of the namespace must be an URI

Instead of writing <http://www.w3.org/.....:author> we can define an alias for namespace and use it as,

<h : ELEMENT xmlns: h =“name of the namespace”>

Where h is prefix

Xml schemas

In xml DTDs we can’t specify the data types and can’t specify the restrict on the values.

Xml schema can be used in place of DTDs. The standard schema language supported by all the software vendors today is w3c xml schema language.

Why schemas :

- Extensible to future addition
- Richer and more useful than DTDs
- Written in xml
- It support data types and namespaces

Xml schema uses xml syntax

W3c has defined a set of tags as part of the name space www.w3.org/2001/xmlschema. (You can download the xsd file www.w3.org/2001/xmlschema.xsd or dtd file www.w3.org/2001/XMLSchema.dtd).

Some of these tags are ELEMENT, SIMPLE TYPE, COMPLEXTYPE, SEQUENCE, STRING, DATE…… Apart from this name space w3c has defined another name space www.w3.org/2001/xmlschema-instance (www.w3.org/2001/XMLSchema-instance.xsd)

When we have to define our own tags we have to use the above tags defined by W3C

According to xml schema if an element contains other elements or attributes the element is known as complex. You can see different types of elements at

http://www.cs.rpi.edu/~puninj/XMLJ/classes/class4/all.html

Parsers

We can produce the xml files directly without using any additional software. To consume xml we need to use a parser.

In our application for reading xml data we can use xml parser. Most of the parsers available in the market can validate the xml files against xml DTD as well as xml schema. These parsers are responsible for reading xml content, breaking that into multiple tokens, analyzing the content and gives the content to our program.

There are different types of parser soft wares available in the market. We can use these soft wares using JAXP.

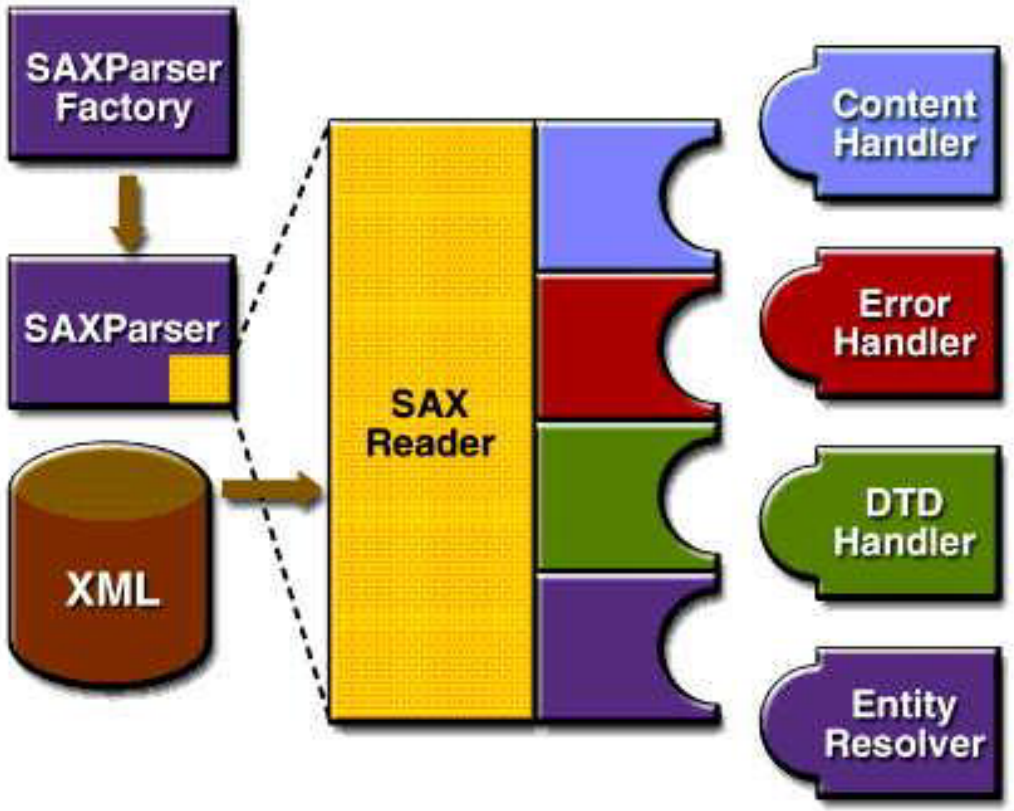
Mainly there are two types of parsers

a) Sax parser b) DOM parser

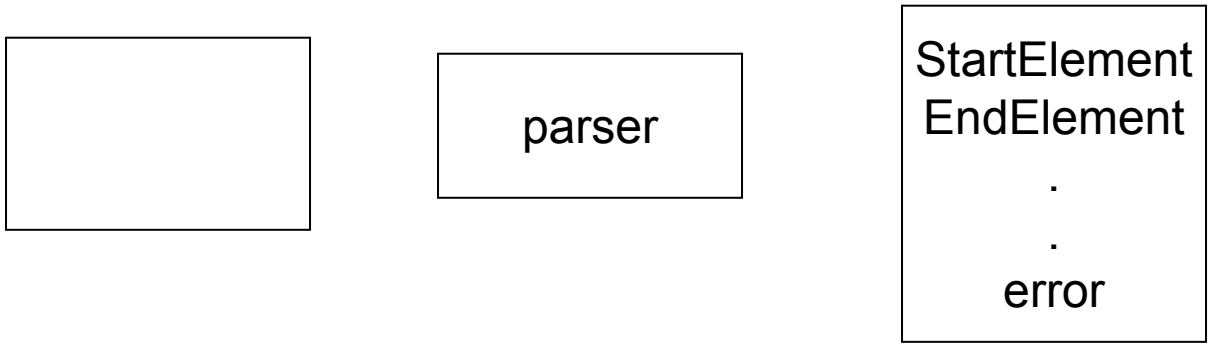
Sax parser

You can see details of sax parser at this site

http://www.saxproject.org/



Processing xml document through sax parser :

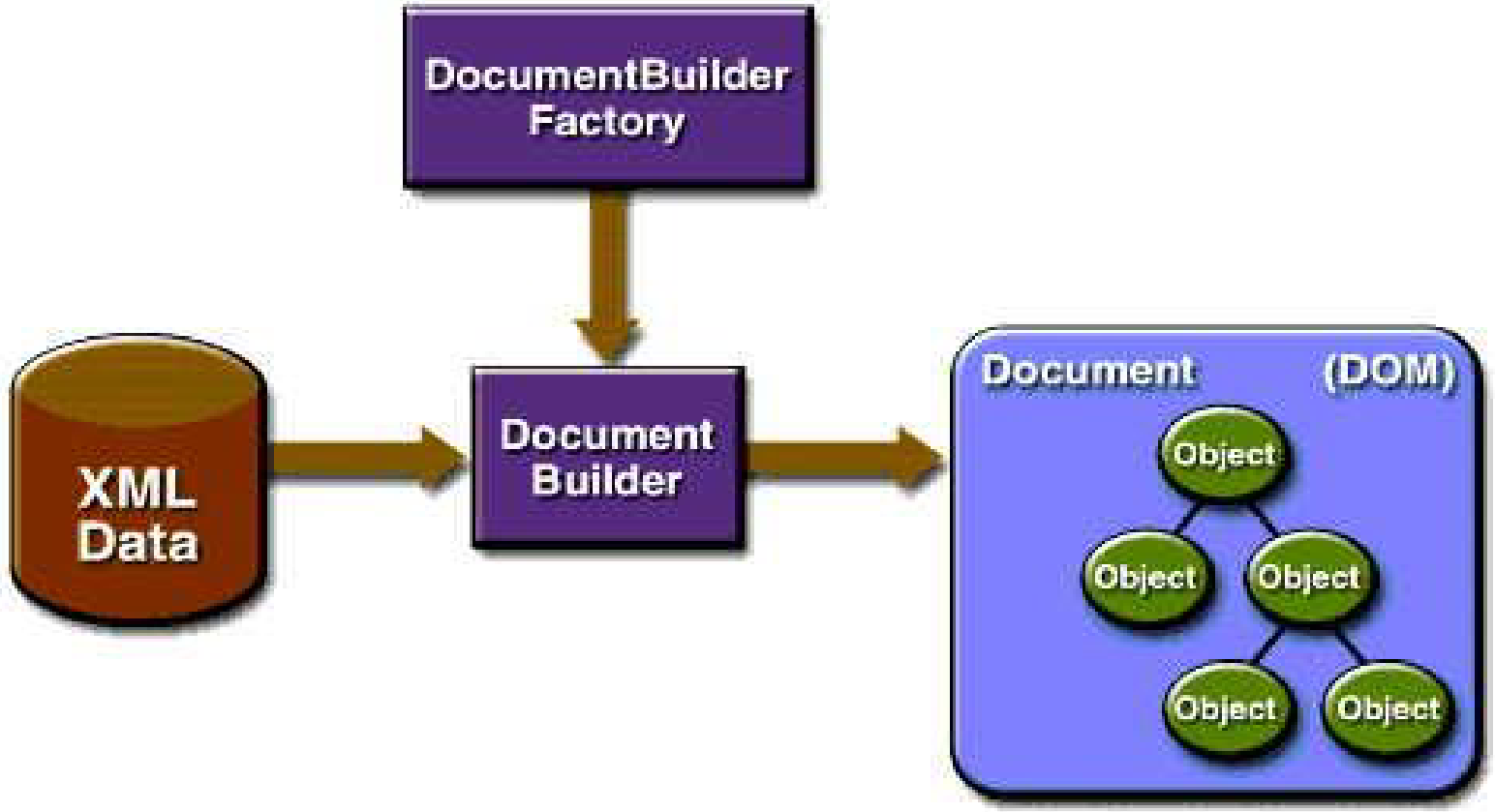


Xml document (handler) our program

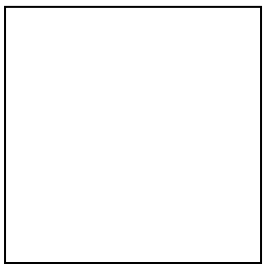
To use a SAX parser we need to create a handler with a set of methods startDocument, endDocument, startElement, endElement … and create a link between the handler and the parser.

Sax parser reads the XML content serially and it calls various methods for ex: when it sees the start of the element like Student, Name, Rno parser calling the method startElement, when it discover any error it calls Error.

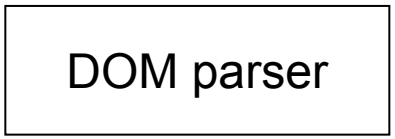
DOM parser

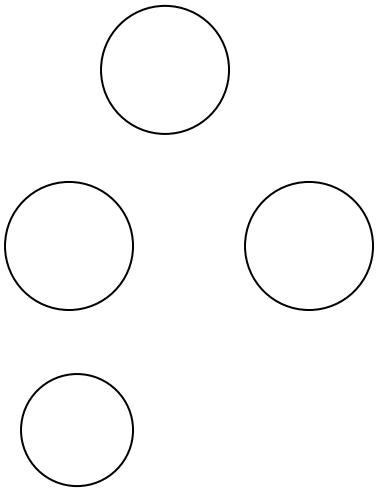

DocumentBuilder Factory
XML Data
Document Builder
Document (DOM)
Object
Object
Object
Object
Object

A DOM parser reads the XML file if there are no errors it produces a set of objects represents the DOM tree. Even though it is easy to program the applications will perform very slowly as the parser need to create too many objects.



xml file





DOM tree

We need to use SAX parser factory to create SAX parser.

If the document is not well formed the parser will detect the errors and calls the appropriate method in error handler.

To validate the XML file we need to set the property validating = true as,

```
parserFactory.setValidating(true);
```

In a standalone file we can't make reference to external entities. While creating XML documents to improve the readability we will be using the spaces, tabs, new line characters – these are called as ignorable white spaces.

We need to implement org.xml.Sax.DocumentHandler interface that has the method setDocumentLocator, startDocument, startElement …. If errors have to handle we need to implement error handler.

Before the parser starts parsing a document it calls the method stLocator() by passing an object Locator.

When the methods in error handler are called, the parser parses SaxParseException object. We can execute the methods like getLinenumber, getSystemID, getMessage.

As soon as the pasing is started the parser calls the method setLocator. After it calls the method startDocument – after this depending upon what it has seen (start of element/ end of element…) it will call the appropriate methods.

By using the locator object we are able to find the location where the parser is currently parsing.

When the characters is called the parser gives us a buffer which contains the character that it has parsed plus some other data.

In characters method we need to start reading from offset up to length.

We can create our handler classes by extending org.xml.sax.helpers.DefaultHandler. This class provides the implementation of the handler interfaces.

Similar to setValidator = true we can use setNamespaceAware to true indicating that the parser must using the namespace.

To validate an xml file against xml schema we need to specify (i) Schema language (ii) Schema source. Even though we can specify schema file in xml file it is recommended to programmatically set the values.

Code we develop for xml files using xml schema as well as xml DTDs is same except

- Set the schema language
- Set schema source
- We need to setup NamespaceAware to true.

To use the DOM parser:

• Creates a document builder factory

• Using factory creates a document builder

• Ask the document builder to parse

• If there is no problem in parsing the parser returns an object of type Document.

org.w3c.dom.Document is an interface and it is implemented by the parser vendors. In our code we will not creating objects directly.

Once the document is created we can execute the getDocumentElement method which returns the root element. Using root element we can get child nodes using root.getChildNodes(return type nodelist)

For every node we can find out its type, name, value, parent, child nodes using hasChildNode, hasAttribute, getAttribute ….

Servlets

Introduction to web applications

HTTP is used by the browsers to communicate with the web servers. In place of browser we can develop our own programs using Socket API to communicate with a web server.

According to HTTP the user agent/http client/browser sends a request for a resource and gets back response. According to the protocol the web server/http server sends the response with a status code.

Most of the web servers are configured not to list the contents of a directory. Almost every web server provides an option of defining a list of files as welcome files.

TOMCAT :

Tomcat_home is a directory in which TOMCAT is installed. Ex: C:\Tomcat\Tomcat5.0 (most it look like C:\ jakarta-tomcat-5.0.19 /jakarta-tomcat-5.0.19 when you extracted zip file to C drive)

Starting and Stopping Tomcat :

To start Tomcat run Tomcat_Home\bin\startup.bat
To stop Tomcat run Tomcat_Home\bin\shutdown.bat

Tomcat requires the location in which java is installed for this we need to add set JAVA_HOME="path where jdk is installed" and add set CATALINA_HOME=.. in both startup.bat and shutdown.bat.

If you are not installed the tomcat, for accessing Admin and Manager we need to add

<role rolename="manager"/>
<role rolename="admin"/>
<user username="uname" password="pwd"
roles="admin,manager,tomcat"/>
 to Tomcat_Home\conf\tomcat-users.xml

Tomcat is configured to listen at port no 8080. We can change the port no using configuration files.(Tomcat_Home\conf\server.xml file change <Connector port="8080" … >).

Request and Response formats :

Request format:

Initial request line
header1
header2
header3
.
.
.

Body (optional)

Most of the browsers either implements http 1.0 or 1.1 protocol.

According to http initial request line contains

- Request method
- Requested resource
- Protocol version

Header is split into (1)Header name (2)Header value
Ex: User-Agent : MSIE

Response Format:

Initial response line
header1
header2
header3
.
.
.
Body

In the response the server sets several heads like

- Server ----- Name of the server
- Content length ----- Length of the content

MIME is standard format in which different types of contents can be placed sdtandard format used by e-mail clients

Status codes:

1xx	Informational message only
2xx	Success of some find
3xx	Redirects the client to another URL
4xx	Error on clients part
5xx	Error on servers part

Standard Request Methods:

- GET Used to request a resource
- HEAD Used to request only header part
- POST Used to send the data from client to server ex: when we will fill the form the browser may send the data
- OPTIONS Used to identify the options supported by the browser
- PUT Used by the clients to ask the server to store the content on it
- DELETE Used to ask the server to delete a file

Diff Btw Get and Post :

In our programs most of the time we will use GET or POST methods.

In HTML forms we can specify the method as POST or GET

When we use GET method the browser appends the form data to the URL but when we use POST method the data will not appended to the URL. Data will be placed as part of request body.

In order to upload files from browser to web server we can use <INPUT Type= “file”> for these types of forms we need to use POST instead of GET.

Web-Containers:

Java soft has released Servlet and JSP specifications publicly. There are so many vendors who has provided the implementation of these specifications. All these products are known as web containers.

A web application developer develops a web application using different resources like html pages, image files, servlets, jsp pages, ……..

Configuring web applications:

Java soft has defined a standard approach for configuring web applications. Once the web application is configured we can deploy it on any of the available web containers.

Steps:

- Create WEB_APP_ROOT directory
- Under WEB_APP_ROOT create WEB_INF
- Create classes and lib directories under WEB_INF (place java classes in classes directory and .jar files in lib directory)
- Create web.xml and place it under WEB_INF
- Create a WAR (Web Application Archive) file using java tool

for this move to WEB_APP_ROOT dir and use

jar cvf webapp.war .

Above command places all files in webapp.war

We use content paths to identify the web applications. Ex: http://localhost:8080/myweebapp1/index.html

By default index.html is consider as a welcomefile. According to our application we can configure a different set of welcome files. Ex:

```
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    </welcome-file-list>
```

The web containers will not be sending the information about WEB_INF directory to the user agent.

Even though we can place all the files under the root it is not recommended.
It is not recommended to use the absolute paths of the resource in a web application. We have to use relative paths.

Managing web applications in web containers:

Most of the vendors provides the tools to manage the web applications. We can use these tools to upload our war files to the server and deploy them.

Ex: In Tomcat

- Use the URL http://localhost:8080
- Choose Tomcat Manager (by using uname and pwd given in config file)
- Choose the war file to be uploaded choose install

In Web Logic

- Login to web logic server using admin uname and pwd
- Choose deployments web applications node
- Choose configure a new application.
- Choose upload the war files
- Select the war file place it on server , configure and deploy it

Today most of the websites generates the content dynamically (when a request is send the web server, the content is generated)

First generation web servers are designed to serve the static content. Later on the web servers provided an option of extending its capability using CGI

A CGI program or script is the one that runs external to a web server.

Earlier most of the programmers has used CGI programs to generate content dynamically. The main problems with this approach are

- Server has to spend some time in starting a new process
- As web server gets too many requests the server has to create more no of processes and this reduces the performance.

As scripting languages are easier than programming languages most of the web developers has started writing web applications using scripting languages.

To run the perl scripts we require a program called perl interpreter. Even it is easy to develop perl scripts we need an interpreter to run them.

Later on most of the web server vendors started supporting different scripting languages directly.

Java soft designed an alternative for CGI programs as well as script getting executed in a web server called as a Servlet.

In servlets the request send by the browser and the response send by the server will be represented as objects.

While compiling the java servlets we need to set the classpath point to a jar file that contains javax.servlet and javax.servlet.http packages.(in most cases the jar file will be servlet.jar or servlet-api.jar or ….)

Every container vendor supplies the jar files that contains javax.servlet and javax.servlet.http packages.

Deploying servlets:

After developing the servlet classes to deploy the servlets,

- Copy the servlet classes under WEB_APP_ROOT/WEB-INF/classes
- Modify web.xml by adding the information about the servlet element as

```
<servlet>
 <servlet-name>servlet1</servlet-name>
 <servlet-class>class of servlet</servlet-class>
</servlet>
<!-- mapping our servlet -->
<servlet-mapping>
 <servlet-name>servlet1</servlet-name>
 <url-pattern>/serv1</url-pattern>
</servlet-mapping>
```

We can use any url pattern like * or *.do or *.xyz etc…

ex: <url-pattern>/*</url-pattern>

To get the name of the user agent we can use

stringbuffer = request.getHeader("USER_AGENT");

We can get the list of all headers using

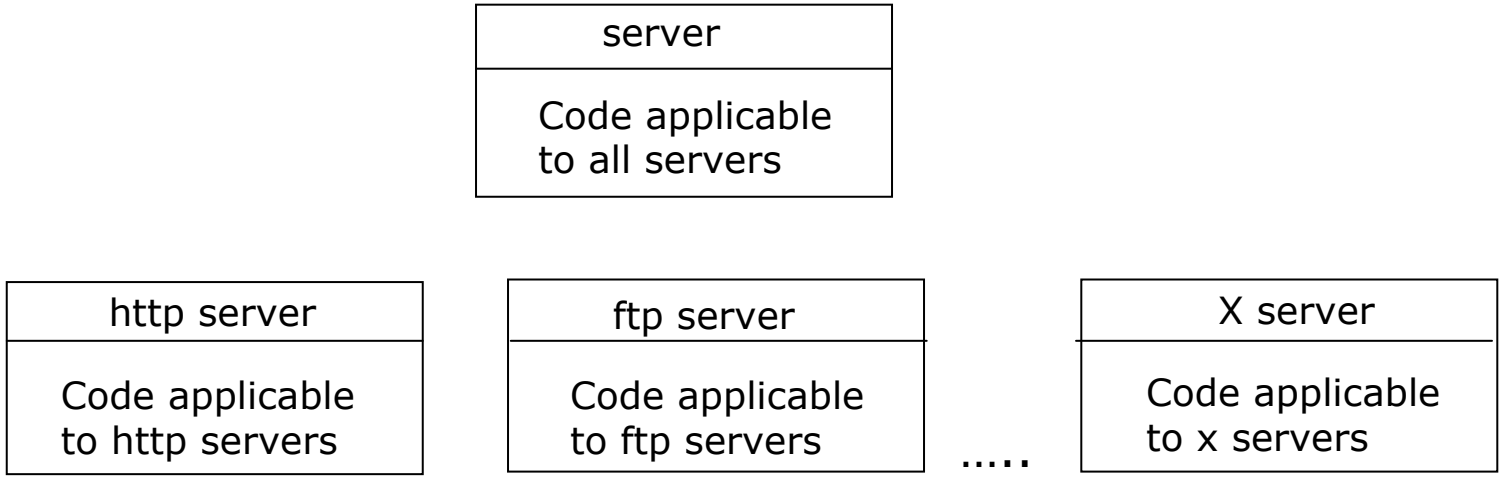
Java.util.Enumuration list = request.getHeaderNames

To invoke a servlet from our applications (standalone java apps/applets/servlet/ejb …)

- Establish a socket connection with web server
- Send http request to the server
- Read the response

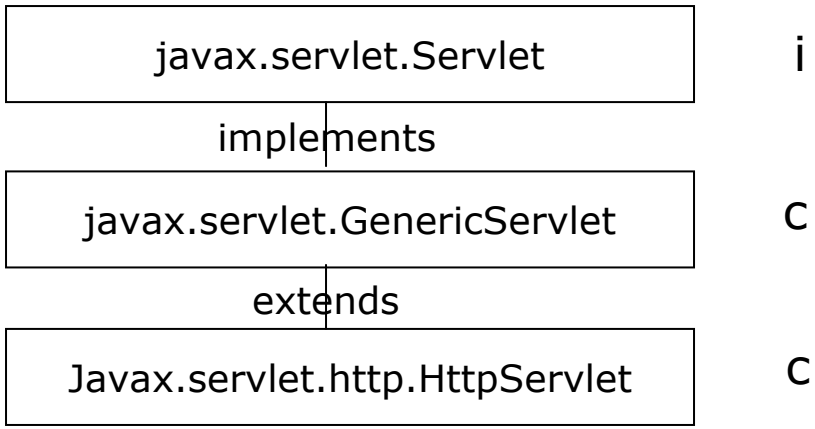
Servlets are initially designed to extend any type of server (web server/ ftp server / smtp …) but almost all the companies has implemented this technology to extend http servers.

In object oriented projects to eliminate redundancy we will be using class hierarchy as,

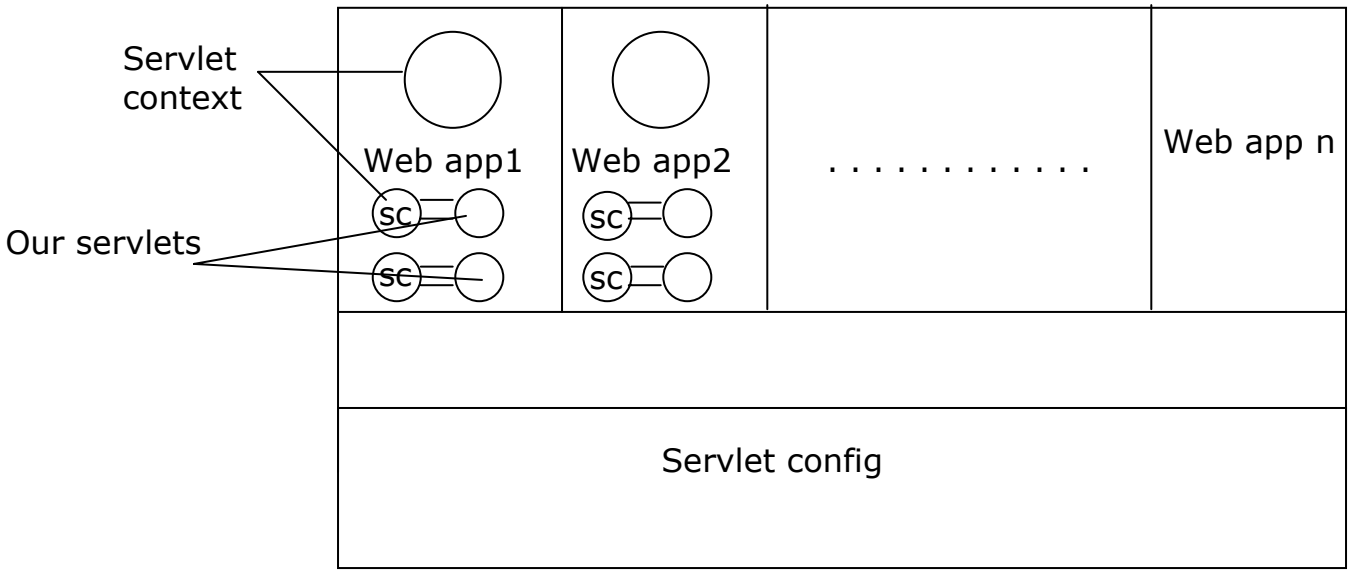


In above ex, code specific to web server in http server class, code specific to ftp server in ftp server class … and code that is common to all the servers is implemented in server class.

Servlet class hierarchy:



A servlet is an object that provides the implementation of javax.servlet.Servlet interface.



Web container

Web container must provide one ServletConfig object to a servlet.

ServletContext is also called as application object.

javax.servlet.Servlet has the following methods

- init – called by the web container after our servlet object is created.
- service – called when the client sends the request.
- destroy – called before our servlet object is removed.

A servlet life cycle has 3 stages, (a) servlet is created (b) servlet will provide the service (c) servlet will be destroyed.

getServletInfo() must return a string describing the purpose of our servlet.

getServletConfig() method must return the servlet config associated with the servlet.

It is the responsibility of servlet containers to create the servlet objects and destroy.

<load-on-startup>10</load-on-startup> where 10 is the priority.

If we use it, the web container will be created our servlet object when our application is deployed.

A servlet object is created

- At the startup if we use load-on-startup
- When the initial request is send and the servlet is not yet available or
- Whenever it is required.

Servlet object will be destroyed

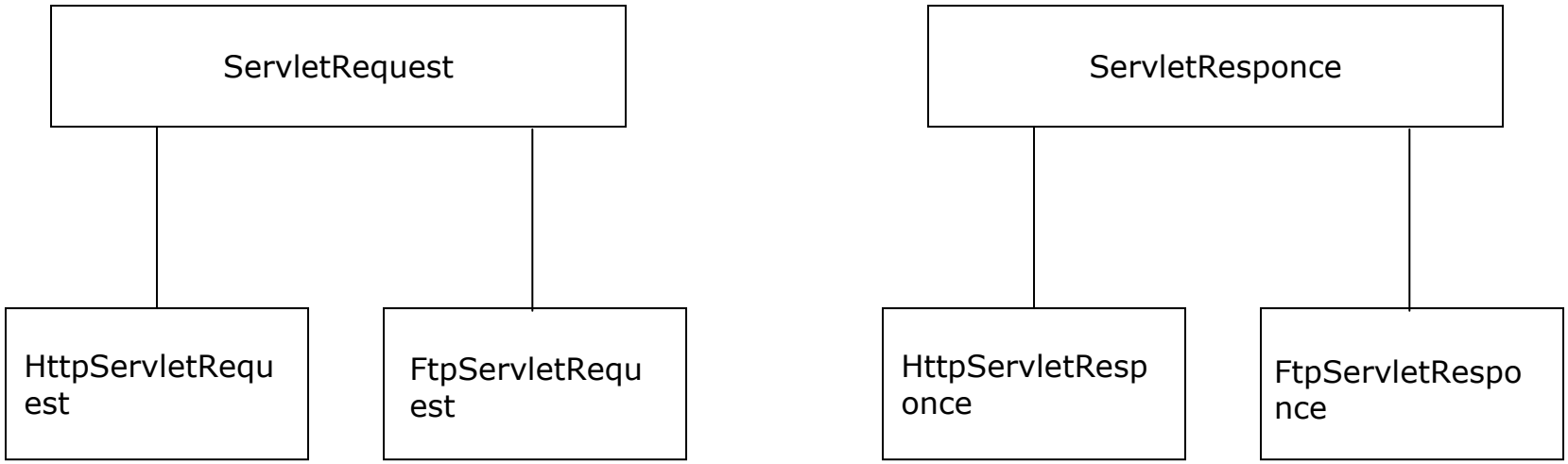
- When the web application is undeployed
- Whenever the server wants to remove.

The *GenericServlet* implemented as part of *javax.servlet* package provides the implementation of *init* method and *getServletConfig* method that fits almost all the servlets.

javax.servlet.http.HttpServlet provides the implementation of *service* method.

We can implement a servlet as a sub class of GenericServlet.

service method of *GenericServlet* takes *ServletRequest* and *ServletResponce* as parameters.



HttpServletRequest and *HttpServletResponce* contains methods specific to http and same as with ftp.

If we create a servlet as subclass of *GenericServlet* we may need to write common code like converting the reference of type *ServletRequest*, *ServletResponce* to *HttpServletRequest* and *HttpServletResponce*.

This can be eliminated by creating our servlet as a subclass of *javax.servlet.http.HttpServlet*.

Inside the servlet containers our servlets are pointing by using reference of type *javax.servlet.Servlet*.

★ Whenever a method is called on an object the JVM will start searching for the method from the class based on which the object is created. If it is not available then the method will be searched in the super class.

The containers always calls the methods exposed by Servlet interface.

```
init(ServletConfig)      {
                           init()   }

service(ServletRequest, ServletResponce) {
          service(HttpServletRequest,HttpServletResponce)       }

service(HttpServletRequest,HttpServletResponce)  {
                           doXXX()          }
```

The implementation of doXXX methods of HttpServlet sends an error to the user agent.

To report an error from our servlets we can use response.SendError(“status code”)

Before we start development of any solution we need to understand the business requirements and functional specifications.

We can use request.getParameter to get values sumbitted using a form.

Instead of using System.out for debugging its better to use log method. The data will be stored in a destination like in a log file or as database.

To know the location of log file we need to read the documentation of the container.

Typical sequence of steps we carry out in a servlet to process the form (a) Read the i/p params (b) validate the i/p (c) process the i/p (d) generate the view.

Instead of hard coding we can place the information as servlet initialization parameters or context parameters.

driverManager.registerDriver(driver);
con=DriverManager.getConnection(url,dbuser,dbpass);
In above ex, instead of hard code we have used variables. Values of these variables can be picked up from web.xml file.

ServletConfig provides the methods getInitparameter() and getInitParameterNames(). As this is implementation of GenericServlet we can call these methods in our servlet.

String driver = getInitParameter("driver");
String url = getInitParameter("dburl");

Above code reads the initialization parameters.

In web.xml we can add the names of initialization parameters,

```
<init-param>
<param-name>driver</param-name>
<param-value>oracle.jdbc.driver.OracleDriver</param-value>
</init-param>
<init-param>
<param-name>dburl</param-name>
<param-value>jdbc:oracle:thin:@localhost:1521:orcl</param-
value>
</init-param>
```

If multiple servlets uses the same set of parameters instead of placing them as servlet initialization parameters we can use context parameters.

```
<context-param>
        <param-name>name1</param-name>
        <param-value>value1</param-value>
    </context-param>
```

getInitparameter() available in ServletContext will be getting the initialization parameters configured as context parameters.

ServletContext scon = getServletContext();

url = scon.getInitParameter(“dburl”)

1st line gets the reference of servlet context

2nd line gets context parameter.

We can add any no of context parameters in web.xml file.

The web container is responsible for reading context parameters from web.xml and place them in ServletContext. For every servlet the container is responsible for creating ServletConfig and store the init parameters in this object.

```
public class init extends HttpServlet {

public void doGet(…) {

        HttpServletRequest req;

        HttpServletResponce res;

        out.getInitParameters(“name”);

}
```

```
public  void init(ServletConfig config) throws
ServletException{

System.out.println(ServletConfig.getInitParameter("name");

}
```

Above code in doGet() will be failing to get the initial parameters. To avoid this failure we need to add super.init(config) if init method is overridden.

Methods like getInitParameter, getServletConfig, getServletContext, log available in GenericServlet will fail if we write code with calling super.init().

Instead of overriding init(ServletConfig) we can override init()

We can store the strings as resource bundles instead of hard coding the strings in source code.

Its very common to submit the information using multiple forms ex: user registration form can be split-up into multiple forms.

In above ex information is gathered from the user using 3 different forms. Once the user fills the 1st page he gets the 2nd page after it is filled he gets 3rd page. Once the 3rd page filled, information will be saved in database.

If the protocol is designed to remember what is done earlier by the client then it is called as stateful protocol otherwise it is called as stateless protocol.

★ Http 1.0 as well as Http 1.1 are stateless protocols.

If we have application with multiple forms the web server will be able to give the information submitted in the current form, it will not be able to give us the info submitted earlier.

A web application can use following techniques to remember the state of the client.

(a) Hidden Fields (b) Cookies (c) Sessions using cookies (d) Sessions using URL rewriting.

We can use <input type= hidden name="n1" value="v1"> as part of our html forms. Browser will not be developing the info about the hidden fields.

If there are too many no of forms more amount of N/W traffic will be generated if we use hidden fields.

Cookie is a piece of info set by the server on the client using http.

We can use *response.setCookie()* to set a cookie on a browser, to get the cookies we can use *request.getCookie()*

Steps to set cookie:

- Create a cookie object *cookie c1 = new cookie(name, value)*
- Add the cookie *response.addCookie(name)*

Cookies are mainly used to serve personalized content according to user requirement.

Problems:

- If used heavily this generate more N/W traffic
- There are some limitations in some browsers in maximum no of cookies per domain.

It is not advisable to store sensitive info using cookie.

When we execute *response.addCookie(),* it adds set-cookie header to the response.

Browser sends back the cookie as part of request header as, cookies name1=value1&name2=value2

Most of the browsers provides an option of allowing or denying the cookies.

Most of web application developers displays a message saying that their web application works properly if the cookies are allowed.

The cookie class provides the methods *setMaxAge*, *setDomain*, *setPath* … In most of he cases we may not use these methods.

We can store any java object using *session.setAttribute(key, java object)* , *request.setAttribute, ServletContext.setAttribute(key, jo)…*

We can retrieve the objects by using *session.getAttribute(key) request.getAttribute(key) ServletContext.getAttribute(key)*

To remove the java object we can use *xxx.removeAttribute(key)*

The web containers can create the session objects on behalf of a client. For accessing the session object we use,

request.getSession(true/false). For every session object that is created a session id will be generated.

request.getSession(true) creates a session object if it is not already exists.

We can remove the session objects by using *session.invalidate().* As the server can't detect the closure of browser, the server deletes the session object after inactivatetimeout which can be configured in web.xml or we can use *session.setMaxInactiveInterval(int)*

session.setAttribute("key1","value1");
session.setAttribute("key1","value2");
session.getAttribute("key1");
session.getAttribute("key2");

When 1st stmt executed value1 will be stored using key1 and when 2nd stmt executed value1 is removed and value2 is stored in key1. 3rd stmt gets value2 and 4th line gets null.

A web application based on session using cookie to carry session id may failed(if user doesn't accept cookies). To overcome this we can use sessions maintain using URL rewriting technique.

To use URL rewriting in our servlets, we need to use *response.encodeURL(string)* instead of directly writing the URL.

When we use *response.encodeURL,* the URL will rewritten as, /xyz/abc + session id

This technique places a bit of burden on server (server has to spend some time in writing URLs with session id).

To upload the files we have to use *post* method. In the form we can use any no of inputs. For files the input type must be specified as file.

There are so many file uploading components available in the market one such component is javazoom file upload component. You can see details at http://www.javazoom.net/jzservlets/uploadbean/uploadbean.html

To deals with upload files

- Create a newreq object using the class javazoom
- To get the list of the files we can use *getFiles()* – uses a list of files in a hash table.

To store the files:

- Create uploadBean
- Set the folder
- Set override policy
- Execute *store* method with newreq as parameter.

To read the parameter submitted as part of the form we can use *newreq.getParameter()*

Ex: book store provided by java soft

We can consider deploying a web application, un-deploying a web application, creating a session, adding an attribute, removing an attribute from ServletContext, HttpSession as events. We can write the code by implementing the listeners interface provided as part of javax.servlet.http package.

Names of the listener interface,

ServletContextListener, HttpSessionListener, ServletContextAttributeListener and HttpSessionAttributeListener

ContextInitialized is called when the application is deployed.

ContextDestroyed is called when the application is undeployed.

We can implement the above listeners and configure them as listeners in *web.xml* file as,

<listener><listener-class> Class Name</listener-class> </listener>

SessionListener interface has (a) *sessionCreated* – called when the session is created (b) *sessionDestroyed* – called when session is destroyed.

Whenever we need to add an object as soon as the session object is created. We can write the code in *sessionCreated* of *HttpSessionListener.*

Along with the cookie we can pass a comment indicating the purpose of the cookei using *cookie.setComment()*

Netscape 4.75 (old versions) stores the cookies in cookies.txt file and IE uses multiple files.

When we specify the age of the cookie the browser stores the info about the cookie in a file, (a) Name of the cookie
(b) Value of the cookie (c) Path of cookie (d) Domain of the cookie and (e) Expire time of the cookie.

In most of the applications the parameters will not setting the path explicitly using *setPath()* In such case path of the cookie will be set to the path of resource.

It is very rare to use *setDomain()* on a cookie. By default domain value will be set to the domain name where the cookie is set ex: http://localhost or www.yahoo.com

A servlet running at www.domain1.com can't set a cookie specifying the domain name of the cookie as www.domain2.com.

If the cookies are set by hotmail.com the browser will send back to hotmail.com only. This is why the browser stores the domain of a cookie.

While sending back the cookies, browser will check whether the path of requested resource is matching with the path of the cookie or not.

If we are running 2 web applications in same server one servlet in application1 can set a cookie that can be reuse by servlet in application2.

We can remove the cookie that is set earlier by setting its MaxAge to 0, if we have not used setMaxAge -1 will be used as default.

If MaxAge is -1 the browser will not storing the cookie in file, cookie will held in memory and its get deleted when the browser is closed.

We need to pass the no of seconds the cookie has to live. Ex: setMaxAge(600) – cookie will live 10 min.

We can assign multiple names like a.com, b.com, www.a.com … to the same machine. On the global internet the N/W administrators will be configuring the DNS servers with this mapping.

We can place the mapping between the name and ip address in C:/winnt/system32/drivers/etc (in 2000/NT)
C:/windows/system32/drivers/etc (in win Xp).
Ex: 102.54.94.97 xyz.com
abc.com

Http 1.1 supports virtual hosting – hosting the web site with different domain names on a single sever. This is supported by using the *Host* header.

According to Http 1.1 Host header is mandatory.

Using this header the web server will be able to identify the resource that has to be served.

It is very common to refer images, applets, activex controls in a webpage.

In older Http protocol the browsers used to establish a connection, sends the request, get the response, disconnect the connection. If we have a html file with 10 images embedded inside it. The browser has to repeat the same steps for 11 times. This may reduce the performance of browser as establishing a connection is time consuming process.

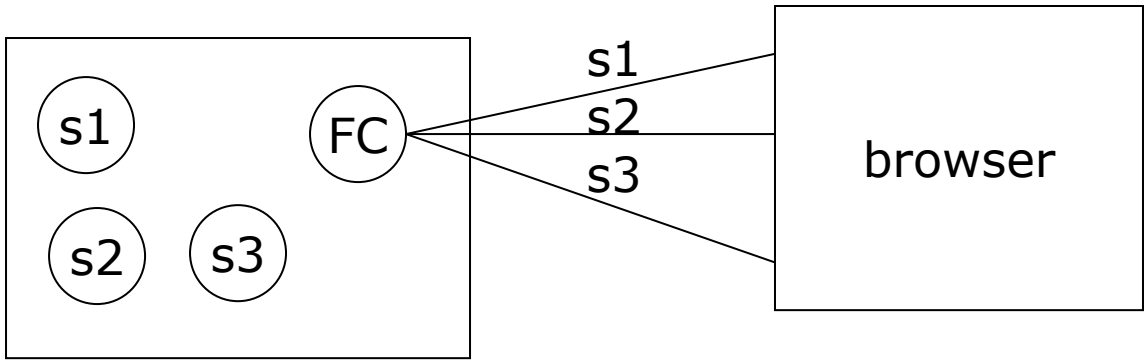
Browsers can send connection keep alive header asking the web browser to keep the connection alive. The server may or may not keep the connection alive.

Most of the web administrators either disable keep alive request or disable persistent connection or set the connection timeout to a small value.

If we expect too many concurrent users then to improve the performance of web server. We use small value for no of connections that are kept alive or we can total disable persistent connection.

★ Irrespective of the type of connection http is stateless.

Front Controller:



The above web application uses Front Controller design pattern

In this design pattern we use one servlet to act as a controller sitting in front of other resources. In this design pattern all the requests will be first handled by FC then it dispatches the request to the appropriate resource.

The servlet engine internally uses *RequestDispatcher* to dispatch the request to the appropriate servlet or jsp. We can use *ServletContext.getRequestDispatcher()* to get the request dispatcher.

There are two methods (a) forward (b) include

Its very common to develop a web application with more servlets processing the same request.

★ We can use both include and forward to dispatch the request to other servlets. When we use include the outputs generated by both the servlets will be sent to the client when forward is used the output generated by 1st servlet will be discarded and only 2nd servlet output will be send to the client.

★ **Difference between Request Forward and Redirect** :

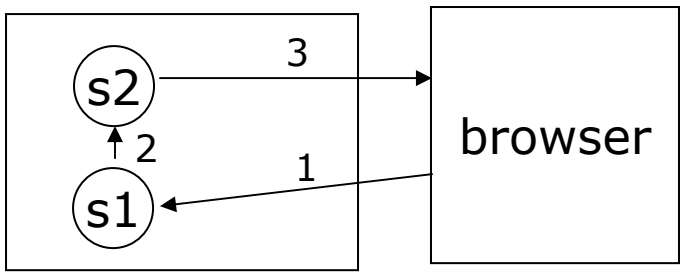
When we forward method only one request will be send by the browser and it is processed by two servlets/jsps. But when we SendRedirect method two requests will be send by the browser.

Ex:

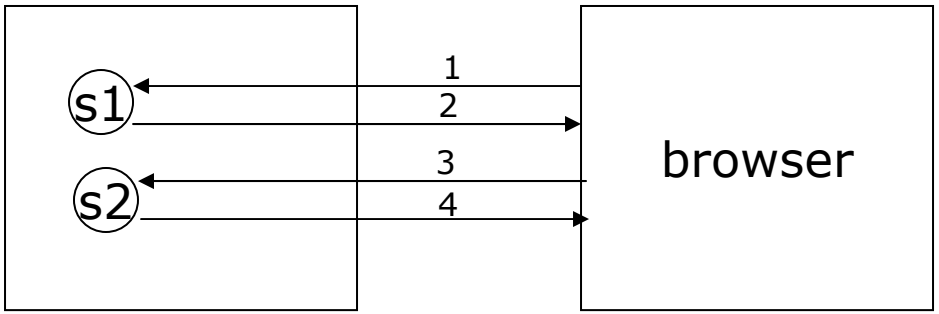
```
class s2 {    service()  { …….. }
          }

class s1 {   service() {
rd= sc.getRequestDispatcher("/two");                    ……
rd.forward("req,resp);    }
```

In this case browser sends the request to the resource s1, this will forward to s2 finally the browser will getting the response.



```
class s2 {
        service() { …… }
        }
class s1 {
        service() {
           resp.sendRedirect(http://..../two); …..
           }
```



In this case two different requests will send by the browser and it will get two responses.

Filters:

We can setup the files to perform the operations like logging, gathering performance statistics, user authentication, authorization of user to carryout some operation, compressing the output generated by servlet. We can also configure chain of filters.

A filter is a class that provides the implementation of *javax.servlet.Filter.* Life cycle methods of a filter are defined in this interface.

- When filter object is created by web container it calls *init()*

- Whenever there is a request that ha to be processed by the filter the container will call the method *doFilter()*

- When web container decides to remove the filter object then it will call *destroy()*

Filter chain object holds the details of objects (filter & resources like servlet/jsp/…)

We need to configure the web application with the info about the filters in *web.xml* as,

<filter>
<filter-name> name </filter-name>
<filter-class> class name <filter-class>
</filter>

<filter-mapping>
<filter-name> name </filter-name>
<url-pattern> / </url-pattern>*
</filter-mapping>

Most of the web containers creates the filter object and calls *init()* when the web application is deployed. When we undeploy the web application *destroy()* will be called and the filter object will be deleted.

See the URLs for more info on filters,

http://www.onjava.com/pub/a/onjava/2003/11/19/filters.html
http://www.informit.com/guides/content.asp?g=java&seqNum=95

In most of the applications which generates content dynamically we need to set the expiry time. If the expiry time is not set, the browser assumes that the content will never expire. To set the expiry time we can use the,

Date d= new Date();

long l = d+6000;

response.setDateHeader("Expires",l);

When we use *response.setDateHeader("Expires",l)* a response header with name Expires will be added. It looks like,

Expires: Tue, 15 Nov 1994 08:22:31 GMT

Date: Tue, 15 Nov 1994 08:12:31 GMT

Every database driver sets a value for *Fetchsize*. In our programs we can use *getFetchSize* and *setFetchSize.* You can see more details at http://javaalmanac.com/egs/java.sql/GetFetchSize.html

JSP

Most of the web developers deploying web applications using servlets mixes the presentation logic and business logic.

Separation of business logic from presentation logic helps us in simplifying the development of application(web authors can concentrates on developing the presentation logic and other developers in the team who has good knowledge of java can concentrate on business logic in (1)Java Beans (2)EJB and (3)Tag Handlers)

Java Server Page is designed to simplify the process of developing a web application on the top of servlet technology. Writing and configuring the application using JSP is easy

Javax.servlet.jsp and javax.servlet.jsp.tagext contains the interfaces to support JSP technology and Tag libraries.

When we send a request to JSP file if there is no servlet representing the JSP file the web container runs JSP compiler that generates the servlet code. Whenever we modify the JSP file the servlet will be regenerated. Some of the web containers provides an option of pre-compiling of JSP.

We can place our JSP files directly under WEB_APP_ROOT or any other sub directory except WEB-INF.

We need not add an entry in web.xml file for most of JSPs. When we need to pass initialization parameters to JSPs then we have to make an entry in web.xml as,

```
<servlet>
<servlet-name>servlet1</servlet-name>
<jsp-file>one.jsp</jsp-file>
</servlet>
<!-- mapping our servlet -->
<servlet-mapping>
<servlet-name>servlet1</servlet-name>
<url-pattern>/jsp1</url-pattern>
</servlet-mapping>
```

If there are any errors in JSP file the jsp compiler fails to compile the java source code. We can check the log files for more information on session for failure.

ASP allows us to mix html with a scripting language like jscript, vb script… JSP also allows us to mix html with java language (scriptlet) or scripting languages.

Most of the web containers support only java language code be mixed with html. Resin supports jscript as part of JSP.

In JSP files we can use Page directives like Language, Import, Include … These directives very much similar to preprocessor directives we use in C programs.

It is very common to have the same header and footer for multiple web pages, to simplify the development of such applications we develop header.jsp – responsible for generating only the header portion and footer.jsp to generate the footer. In all other JSP pages we write the code as,

Page1.jsp :	Page2.jsp:
Include header.jsp	Include header.jsp
Content1	Content1
Include footer.jsp	Include footer.jsp

Ex: www.oracle.com (same header and footer for all pages)

When we send the request for page1.jsp the JSP compiler internally generates Page1.jsp by merging header.jsp, footer.jsp with Page1 and this JSP page will be converted as a servlet.

According to JSP specification the container need not regenerates the servlet whenever there is a change in the files that are included using include directive. Tomcat regenerates the servlet even if there is a change in the included file.

When we used Info directive JSP compiler generates getServletInfo method. Language directive can be used to specify the language that is used as part of our JSP. Ex:

Page Directive:
<%@ page
  [language="**java**"]
  [extends="*package*.*class*"]
  [import= "{ *package*.*class* | *package*.* }, ..."]
  [session="**true**|false"]
  [buffer="none|**8kb**|*size*kb"]
  [autoFlush="**true**|false"]

[isThreadSafe="**true**|false"]
[info="*text*"]
[errorPage="*relativeURL*"]
[contentType="*mimeType* [
;charset=*characterSet*]" |
"text/html ; charset=ISO-8859-1"]
[isErrorPage="true|**false**"]
%>

For more information on JSP page directives see the URL
http://www.developer.com/java/other/article.php/626391

Whenever we use an object in System.out.println internally toString() will be executed. Ex:

```
Date d = new Date();
System.out.println(d);
System.out.println(d.toString());
```

} these two are same

If we need to use a class in a package other than the default package we can use import package or fully qualified name of the class in our code

In our JSP pages to import the classes we can use

- <%@ page import = “java.util.Date” %>

  <%@ page import = “java.io.* %> (**or**)

- <%@ page import = “java.util.Date, java.io.*” %>

We can include the java code directly as part of JSP files ,

<% Java Code **%>** called as scriptlet. When the JSP compiler compiles the JSP file it copies the scriptlet as it is in jspService method.

Html content
<%= new java.util.Date() %>
<% int i=10; out.println(“i=“,+i); %>
Html content

In above code we have declared a variable ‘i’ and used in print stmt but we are using a variable ‘out’ without declaring it but it is perfectly valid in JSP scriptlets.

Implicit Variables:

JSP compiler generates JSP code with a set of variable like
(i) out ii) request iii) response iv) pageContext v) session vi) application vii) config and viii) page. All the variables can be used in the scriptlets without declaring them. These variables are called as well known variable or implicit variables.

For more info on implicit variable see the following link
http://javaboutique.internet.com/tutorials/JSP/part11/page07.html

page refers to this – current object and pageContext is an object (javax.servlet.jsp.pageContext) that is created for every JSP page and it provides access to the other objects like out, session ….

We have methods like forward and include to perform operations like RequestDispatcher.forward and RequestDispatcher.include

In JSP pages we can use declarative statements to define the variables or implement the methods as,

<%! int i; %>

We can write our own methods, static variables and instance variables in declarative statement as,

```
<%! Static int x =10;
int add (int a, int b) { return a+b; }

%>
```

It is highly recommend to use tag libraries instead of using scriptlets and declarative statement.

In java we use try catch blocks to separate out the actual code that performs the action and code the code that deals with error. In JSP we can provide the implementation of code that handles the error in a separate page. This JSP page is known as error page.

For supporting the above concept JSP provides isErrorPage and errorPage directives.

```
<%@ isErrorPage="True" %>
```

errpage.jsp

```
<%@ isErrorPage="True" %>
<%@ errorPage="errpage.jsp"%>
```

one.jsp

When an exception is thrown while executing one.jsp file the container will start executing errpage.jsp. We can access the exception variable without declaring it in error pages.

Instead of sending the data byte by byte to the browser the web container collects the data in a buffer. If the size of the buffer is enough to accommodate the whole o/p of JSP page the o/p is collected and send at once to the browser. This reduces the amount of Network traffic.

If the jsp generates more amount of data than the size of the buffer the container frees the buffer once it is filled (this is done if autoFlush is set to true)

<%@ page buffer="1kb"%>
<%@ page autoFlush="false"%> (default is true)

When we encounter JSP buffer overflow exception we can solve it either by increasing the size of the buffer or by setting the value of autoFlush to true.

Action Tags:

In JSP pages we can use action tags like include, forward, useBean, setProperty, getProperty, param… these tags are supported by all the web containers.
See url http://javaboutique.internet.com/tutorials/JSP/part11/page08.html

<jsp:include page="page2.html" flush="true" /> It includes the content of page2 (it may be of type jsp/html/img ….) in the o/p sent to the client (internally it uses requestDispatcher.include) similarly we can use jsp:forward tag to forward the request from one page to other page. This is equivalent to requestDispatcher.forward.

If the data is already committed before the execution of jsp:forward then forward will be failed. We can solve the problem by increasing the buffer.

There will be no difference in the o/p produced by using jsp:include and include directive. When we use include directive the contents will merge together by jsp compiler and generates a single servlet but when we use action tag include two servlets will be generated and in first servlet code similar to requestDispatcher.include will be added.

There may be slight benefit in terms of performance using include directive.

When we use jsp:include a change in main file and change in sub file causes regeneration of the servlets.

According to jsp specification the container need not regenerates the servlet if there is a change in included file when we use include directive

We can write the functions as part of declarations and in these function we can't access implicit variables. Implicit variables are local to jspService and can be accessed from the scriptlets.

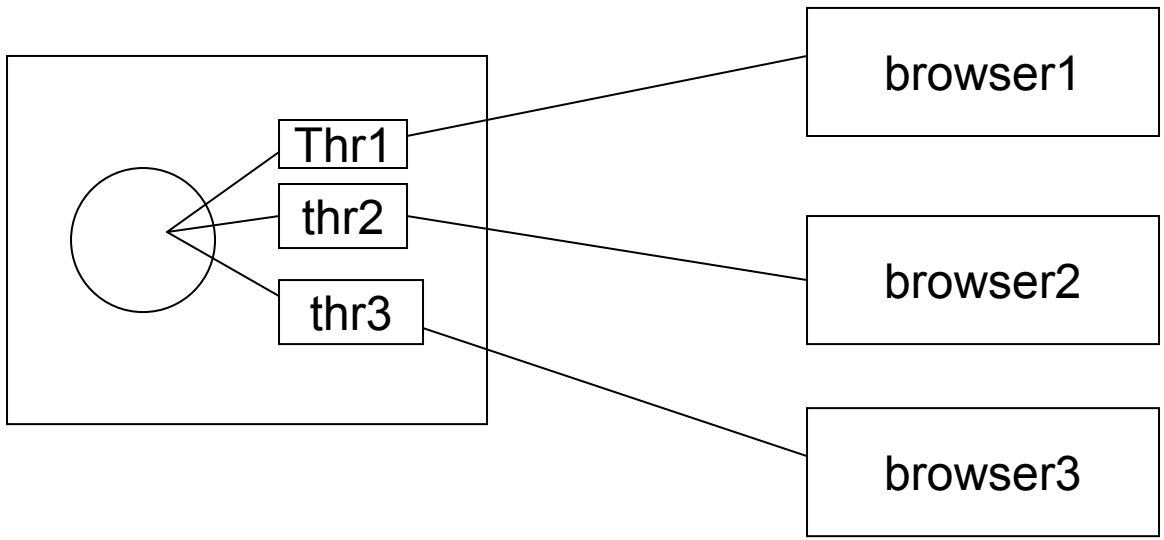
We can provide the implementation of jspInit and jspDestory which are equivalent to init() and destroy() methods in the servlets.

As part of JSP files we can use jsp expression as,
<%= Java Expression **%>**
ex: Current time: <%= new java.util.Date() %>

In jsp 2.0 we have support for **EL – Expression Language**

Tomcat as well as other web containers internally creates the threads and access the service method from the threads.

A servlet container creates multiple threads and access the service method from these threads concurrently as shown below same point of time it sends the response.



Since the servlets are accessing from multiple threads the code in servlets must be **ThreadSafe**. Code will be thread safe if it can be executed concurrently from multiple threads.
All the objects we will use from our servlets using instance variables must be thread safe. Not every java class is thread safe ex: AWT, JFC classes

If our servlet uses only locals variables we can claim that our servlet is thread safe.

We can develop a servlet by implementing single thread model. This is a marker interface or tagged interface. From servlets 2.4 this interface is deprecated and it is not recommended. By implementing this interface we are telling the web container that my service method can't executed concurrently from multiple threads.
Most of the web container vendors creates multiple servlet objects based on same class if the servlet implements single thread model. In this case more amount of memory space is required.

Java Beans:

We can assemble a computer or fan very easily by choosing different components manufactured by different vendors. We can take a screw from company one and use it to fit the Mother board to cabinet as they are manufactured according to a standard. Observing to this point to simplify the process of developing software, different software companies has proposed different component technologies. Ex: java soft java bean component tech, EJB component tech, Microsoft COM, BONOBO component model.

Java Bean and EJB are two different specifications from java soft.

EJB can be used to implement business logic on the server side. Most of the developers uses to assume Java Bean components are for developing GUI components and they can be used only on the client side but we can develop any kind of software using Java Bean standard (GUI/ non GUI). Java Bean can be used either on the client side or on the server side.

AWT, JFC components are implemented according to Java Bean standard.

According to Java Bean standard a Bean component can support a set of properties, set of events, any number of additional methods.

A property can be read-write or it can be just read only property. For read write property we need to provide setXXX and getXXX methods (isXXX if the property is Boolean)

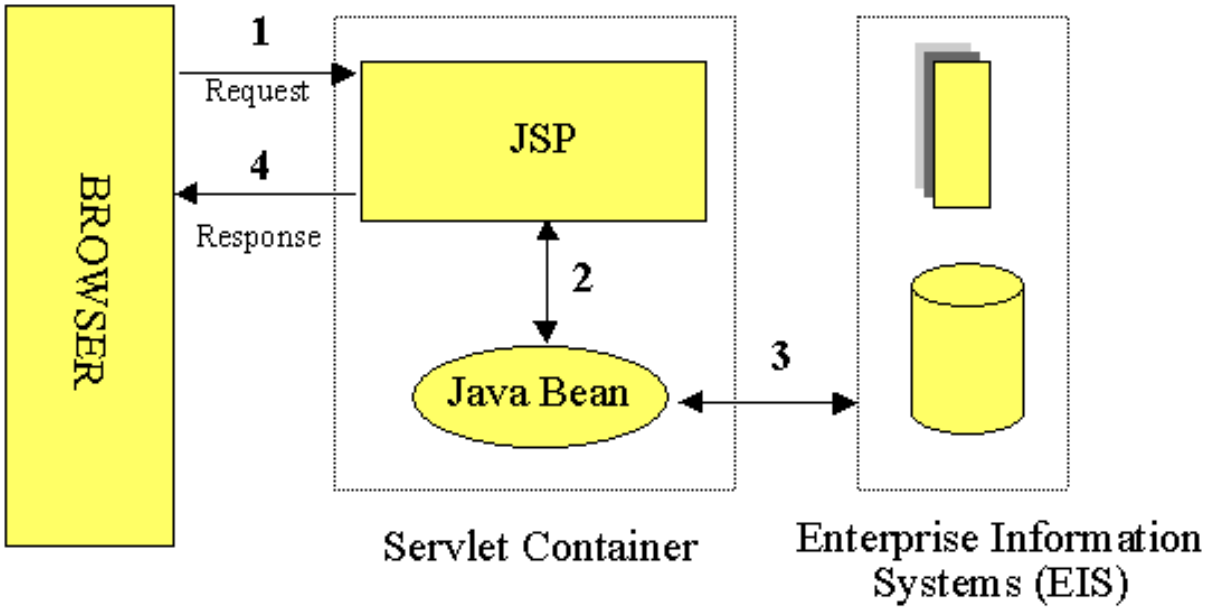
To support the following properties (i) uname (ii) email (iii) age according to Java bean standard we need to write the code as,

```
public class UserBean
{String uname;
 String email;
 int age;
 public void setUsername( String value ) {uname = value; }
 public void setEmail( String value ) { email = value; }
 public void setAge( int value ) { age = value; }
 public String getUsername() { return uname; }
 public String getEmail() { return email; }
 public int getAge() { return age; }
}
```

Java Beans like JButton supports the events by providing the methods with naming patterns (i) addXXXListener (ii) removeXXXListener

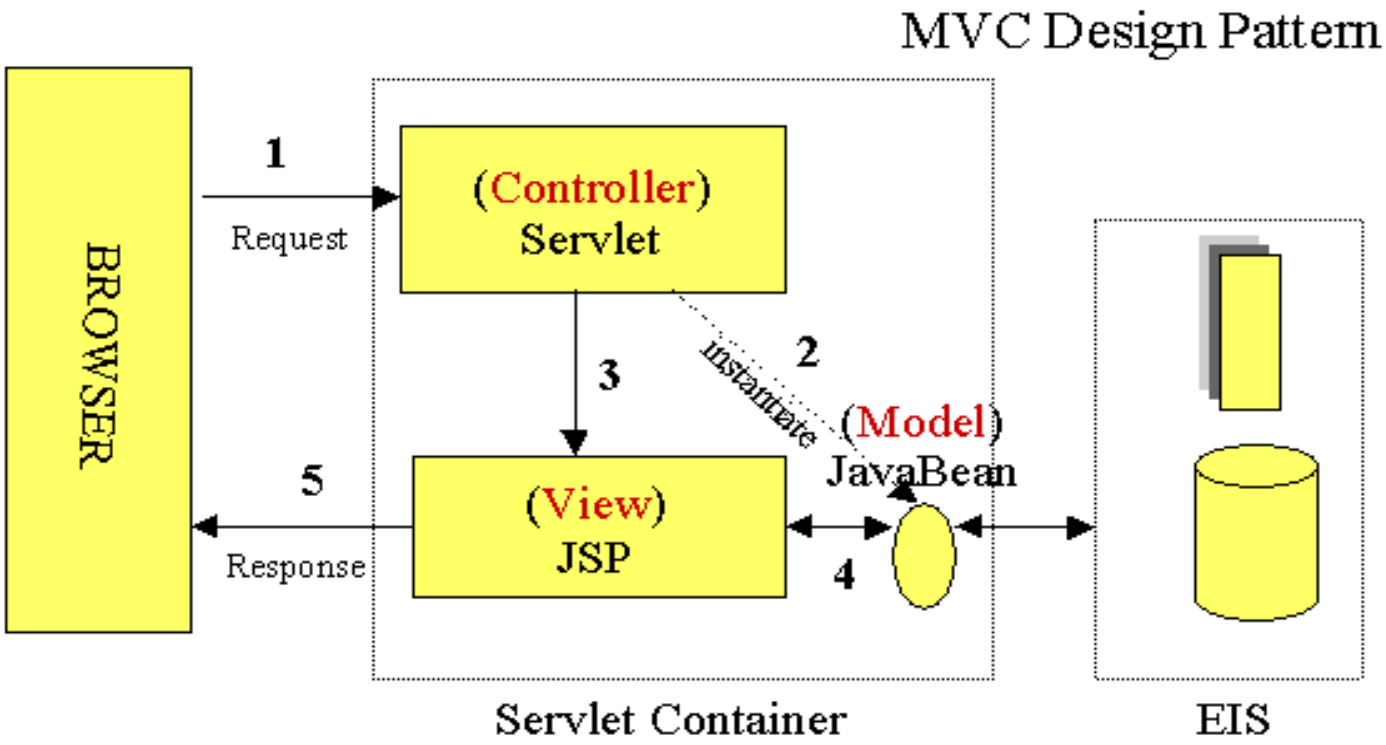
Apart from developing Java bean class we can also provide BeanInfo class. In this class we can provide (i) Information about properties (ii) Information about the events and (iii) Information about the icon that represents our bean.

According to JSP model1 we can develop the application as,



According to above model the presentation logic has to be implemented in JSP page and the business logic has to be implemented as part of Java bean This model help us in separating the presentation and business logic.

For a large scale projects instead of using model1 it is better to use model2 (MVC). Stuts frame work is based on model2.



JSP compiler generates the following code equivalent to

<jsp:useBean id =“var1” scope =“session” class =“class name” />

as,

```
ourpack.ourbean var1 = null;
var1 = session.getAttribute(“var1”);
if(var1 == null) {
var1 =  (ourpack.ourbean)class.forName(“ourpack.ourbean”).new
instance();
Session.setAttribute(“var1”, var1);    }
```

JSP compiler generates the following code equivalent to
<jsp:useBean id =“var1” scope =“application” class =“class name” />
as,

```
ourpack.ourbean var1 = null;
var1 = application.getAttribute(“var1”);
if(var1 == null) {
var1 =  (ourpack.ourbean)class.forName(“ourpack.ourbean”).new
instance();
application.setAttribute(“var1”, var1);    }
```

In useBean tag we can specify the following scopes,

• Session -- Bean object will be stored in session object. We will be using this scope if we have to store data specific to the client.

• Application – Bean object will be stored in application object. We will use this scope when global data has to be stored.

• Page – Used to store the bean in pageContext object. Uses when we need to hold the data only while processing the request. The bean object is accessible only in that page.

• Request – Bean object will be stored in request object. Bean object can be accessible by multiple pages that process the request. This can be used if we need to hold the data only during the request processing using multiple JSP/ servlets.

We can use the tags jsp: setProperty and jsp:getProperty to set and get the properties of a bean.

To access additional methods available in java bean we can write var1.somemethod in a scriptlet.

We can automate the process of setting bean properties with the values submitted by the user in a form.

Most of the application developers develops a JSP/ servlet for
(i) Generating the form and (ii) Process the form.

If we need to process a form as follows,

Ename [ ] (ename)

Salary [ ] (sal)

Dept no [ ] (dept)

Submit

We can create a java bean with property names ename, sal, dept so that in our JSP we can use,

<jsp:setProperty name = “emp” property = “*” />

Let us assume that we are developing an application that has to deployed at different customer sides, for this application we can store the preferences of the customer in a xml file or in a database table as,

App_prof

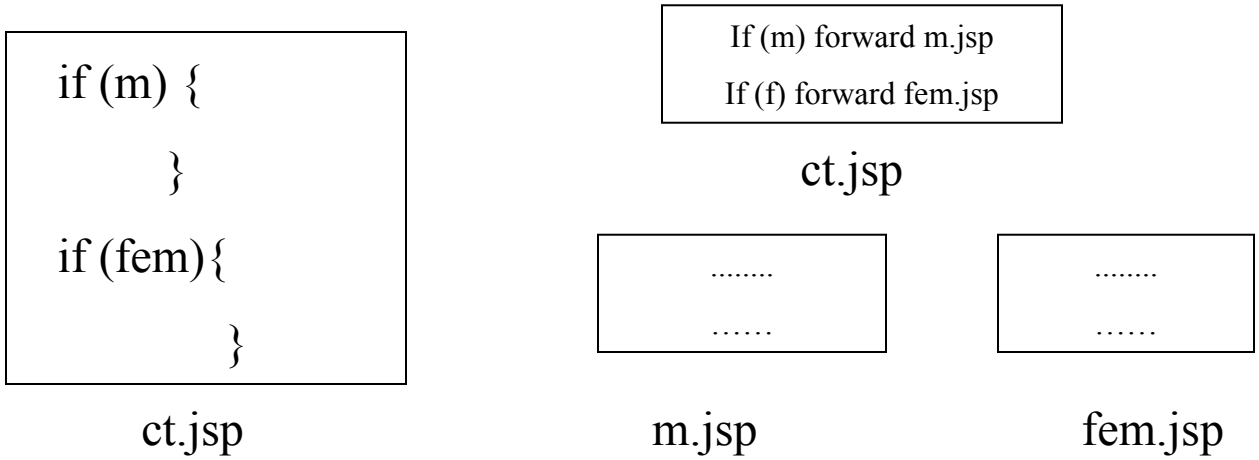
Prof_name	prof_value
Logo	one.jpg
Name	comp1
bgcolor	white

In this project we can create a java bean that holds this information. In this above data is not specific to the client and it need to be accessed by multiple jsps we can create a java bean and store it inside Application obj. The data that is specific to the client has to be stored inside a session object if multiple jsp/servlets need to access the same data.

When we use page scope, object will be available only in that page.

Multiple jsps processing same request need to access same data then we need to store the data in request scope.

Let us assume we need to compute the tags of a application and generates different views for male & female applicants. For this we can develop a single jsp like CT.jsp and write the whole code in a single file or we can write multiple JSP files.



In our earlier ex we establishes the connection with DB in our servlets whenever a connection is required and closes the connection when the task is completed. This reduces the performance as it takes some amount of time in establishing the connection.

To improve performance of servlet/jsp/EJB we can use connection pools. A connection pool pre creates a set of connections to DB. In our code whenever we require a connection we will take it from the pool and when the task is completed instead of really closing the connection we will be putting back the connection in the connection pool.

If we need to use connection pool we can store it inside application scope and access the connection pool from all the JSPs. Today every web container provides an action of creates a connection pool and accessing it using javax.sql.DataSource.

To use connection pool with Tomcat server,

- Copy the JDBC drivers under Catalina_HOME/Commom/lib
- Open the browser and login into Admin page.
- Choose Data Sources and use create new Data Source by feeding the information like name of the jdbc driver class, DB Uname, pwd and jdbc URL by giving a name to this. In our programs we can use this name to access the connection pool.

We can write the following code to detect the jsp version currently supported by the container.

```
JspEngineInfo  ei = jspf.getEngineInfo();

out.println(ei.getSpecificationVersion() );
```

We can use the code to check the jsp version and display error message if our web application deployed on a engine that is not enough to support our web application.

Most of the shell scripting languages supports $ {varname} for getting value of a variable. Same kind of syntax is supported in EL that is part of JSP 2.0

EL supports basic arithmetic operations (+,-,*,/,%)

To display ${1} in jsp we need to write \${1}. Similar to any other programming language in an expression we can use a variable or a constant.

Java soft has initially used EL as part of JSTL (Java Standard Tag Libraries). As most of the developers started using this language Java soft started supporting it directly in jsp.

EL supports basic comparison operations like <,> …

EL provides support for implicit objects like pageContext, pagescope, sessionscope, applicationscope, param, paramvalue, cookie and initparam.

Tag Libraries:

With tag libraries…

< arth:add param1 = 5 param2 = 10 />

Without tag libraries…

<% int a =5, b =10, c = 0;

c = a+b ; %>

Above two ex shows the jsp page performs the same operation but the first one is written without using java code. Any html developer can very easily learn and use tags like this. A jsp tag library provides the support for multiple tags or at least one tag.

By using tag libraries we can eliminate the scriptlets this help a non java programmer to develop a web application using jsp technology.

Tag library implementer provides a tld file and jar file containing java classes. These classes are called as tag handlers.

When a tag is encounter in a jsp file the web container internally executes the methods on tag handlers.

Java soft has identified a set of commonly used tags in almost all jsp applications and released a specification called JSTL.

Any one can provide the implementation on JSTL tags. One such implementation is available at http://www.jakarta.apache.org

Every tag library will be identified uniquely by URI.

The JSTL tags are divided into different categories

- Core Tags : out, if, forEach, forTokens, redirect, import, set. Remove
- SQL tags : used to perform DB operations – transaction, query, update …
- XML tags : used to deal with xml content – parse, other core tags like set,

… that deals with xml.

• For developing internationalized applications we can use i18n tags – setLocale, timezone, bundle, setbundle, message, formatnumber, formatdate

Procedure for using a Tag Lib :

• Copy tag lib jar files under WEB-INF/lib (if tag handler supplied as classes without package a jar file then we need to copy the classes under WEB_INF/classes.

• Copy the tld files files under WEB-INF or create a directory under WEB-INF and copy tld files under this directory.

• We need to give the information about the tag libraries in web.xml by adding the following lines to it as,

```
<taglib>
    <taglib-uri>http://www.inet.com/taglibs/bodytags</taglib-uri>
    <taglib-location>/WEB-INF/tlib/tlib1.tld </taglib-location>
</taglib>
```

To use tags in jsp file we need to add taglib directive as ,

<%@ taglib uri="http://www.inet.com/taglibs/samptags" prefix=“abc"%>

Every tag can support zero or more number of attributes and a tag may or may not support body content.

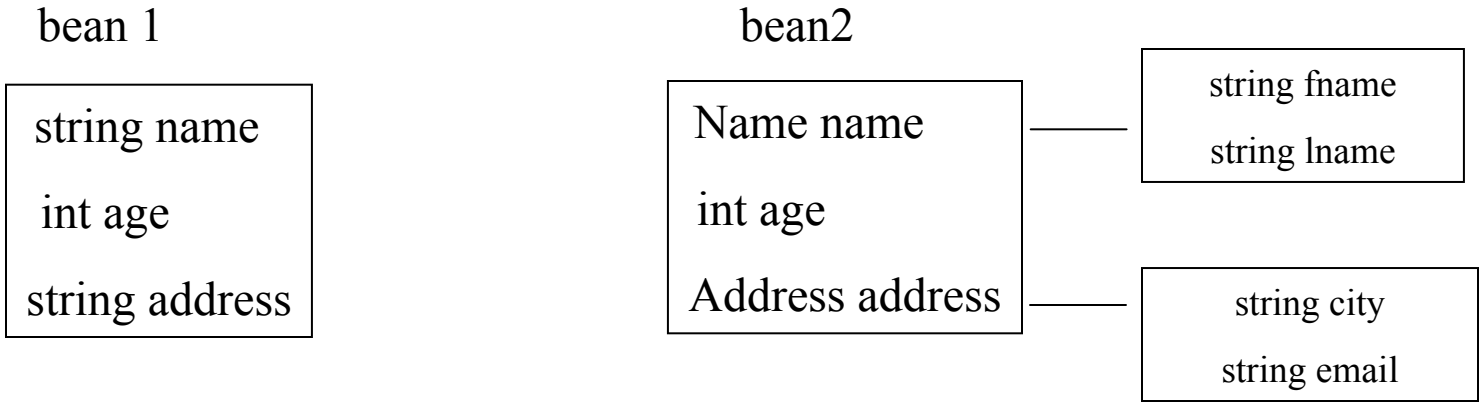
We can use setDataSource tag with driver, url, uname, pwd as attributes to the tag. We need to specify the name of the variable using var tag. This variable name has to be used in query, update tags.

We can execute sql queries using the query tag by specifying the datasource, select statement. The result set can be accessed using the variable name that is defined in var attribute.

Before running jdbc code from the servlets we need to copy jdbc driver classes under Catalina_Home/common/lib

We can run multiple sql stmts as part of a single transaction. For this we need to group update and query tags as part of transaction tag.

Using EL we can access the properties of a customer bema using
${cust.name}



In the second bean not all properties are of type java primitive or java strings. In this case we can access different properties using expression
${cust.Name.fname} …

As part of EL expressions we can use logical AND and logical OR.

In EL expression it is better to write xxxScope.beanname.property instead of using beanname.property.

paramvallues, headervalues are implicit objects. These objects are collections

In html forms we can use different input fields with same name. ex : http://localhost:8080/1.jsp?p1=val1&p2=val2&p3=val3&p3=val4&p3=val5

In above URL even though we have five parameters there are three parameters with same name.

We can use forTokens to split-up a string into multiple tokens separated by a delimiter. If the delimiter is a ‘, ‘ we can use forTokens or forEach.

To store multiple objects we can use classes like vector, arraylist, hashmap, hashtable … Whenever we need to iterate though the list of objects using forEach tag we can store the objects inside list, linkedlist, array, stack, set, hashmap, hashtable, property…

Instead of writing the loops for copying elements from one array to another array we can use System.ArrayCopy

Internationalization

When we develop application targeting the customers of multiple locale we need to take care about (i) Data format (ii) Time zone (iii) Number format (iv) Messages we display (v) Currency symbol. These kind of applications are called as internationalized applications.

Once we develop i18n applications we can localize it to different locale.

When we install OS we choose the locale settings. We can modify these settings at later of time. Ex: regional settings in windows

To represent a locale in java we can use java.util.locale class.

If we develop the code targeting one locale as,

JButton jb1 = new JButton (“save”);

Localizing into other locale takes lot of time. Instead of hard coding the strings that will be used as labels or messages in the code as shown above. We can separate the strings and store them in property files.

In the resource files the label of delete button is

button.delete.label = delete – app_en_US.property

button.delete.label = loschen – app_de_DE.property

If we create resource files in one locale in future we can add a new resource file to support a new locale. In this case we need not change the code.

In order to read resource files we can use java.util.ResourceBundle. In a single program we can use multiple resource bundles.

We can use ResourceBundle.getBundle(basename, locale) -- this loads the resources from the files specific to locale if it is not available the default resource bundle will be loaded.

errors.invalid = {0} is invalid

errors.maxlenght = {0} can’t be greater than {1} characters

In above ex at the run time we need to supply the parameter values to the messageFormatter.

In internationalized applications we can use java.text.DateFormat, java.text.NumberFormat, java.text.MessageFormat to format date, time, numbers, parameterized messages.

We can support <pre:tag without body and attributes />

<pre:tag without body attr1="val1" />

<pre:tag without body and attributes>body content<pre:tag />

<pre:tag with body > body content <pre:tag />

To support tag library Java soft has provided the package javax.servlet.jsp.tagext. As part of this package Tag, iterationTag, bodytag interfaces are defined. As part of same package we have classes TagSupport and BodyTagSupport.

Tag handler is a class that provides the implementation of TagSupport to support a tag without body and BodyTagSupport to support a tag with body.

The most important methods as part of Tag interface are doStartTag and doEndTag.

Similar to servlet specification in which java soft define the purpose of each and every method and when the container has to call these methods. In jsp tag specification java soft has defined the purpose of methods in Tag and BodyTag interfaces and it has defined the sequence in which the methods has to be called by the container.

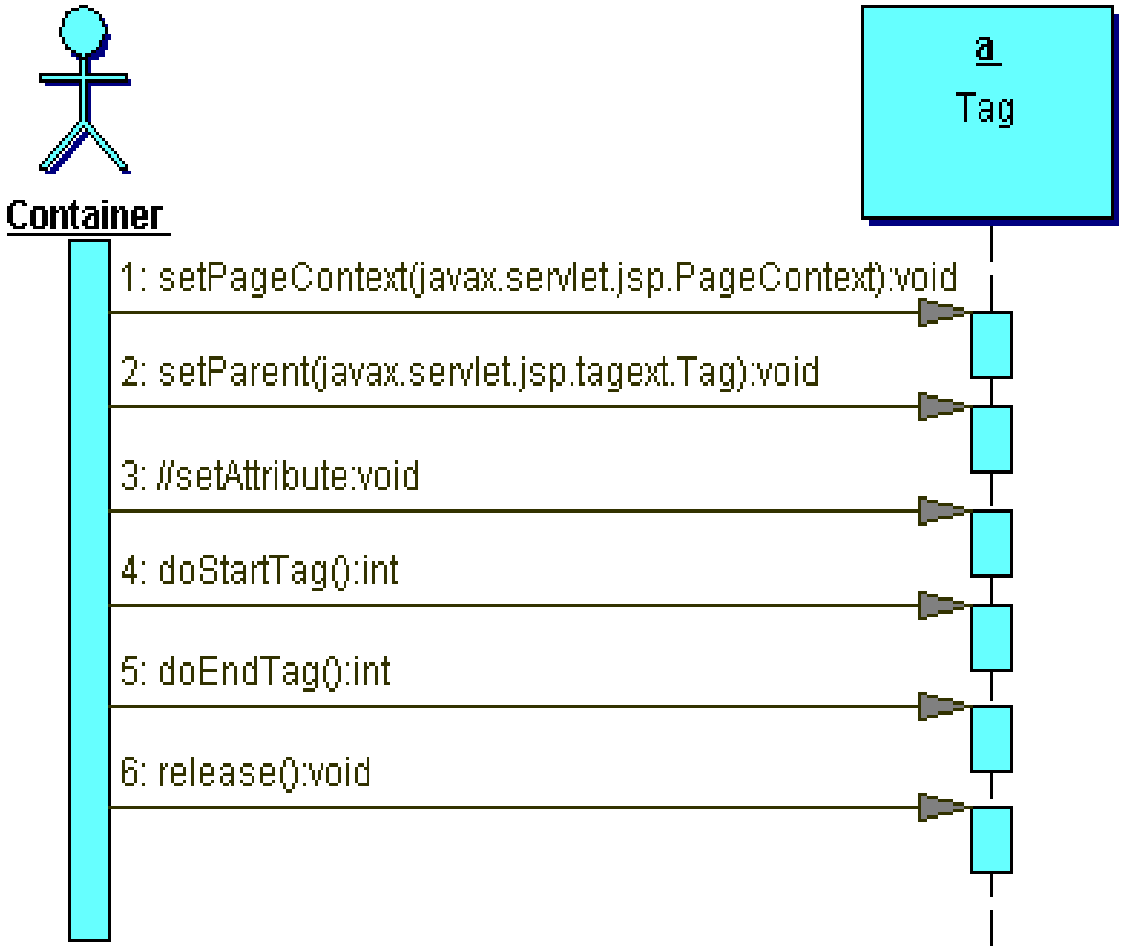
We can use a tag inside another tag like otherwise, when inside choose tag. In this case choose tag is known as parent tag.

Javax.servlet.jsp.PageContext provides the methods like setAttribute, getAttribute, getOut, getSession … we can use PageContext object to access session, application … objects.

setParent will be added by the container to give the information about the parent.

Since the code for setPageContext, setParent, getParent is common for most of the tags these are implemented in TagSupport.

Variables like id, pageContext are provided by TagSupport class. id is used to hold an attribute and pageContext variable is used to hold PageConext object.

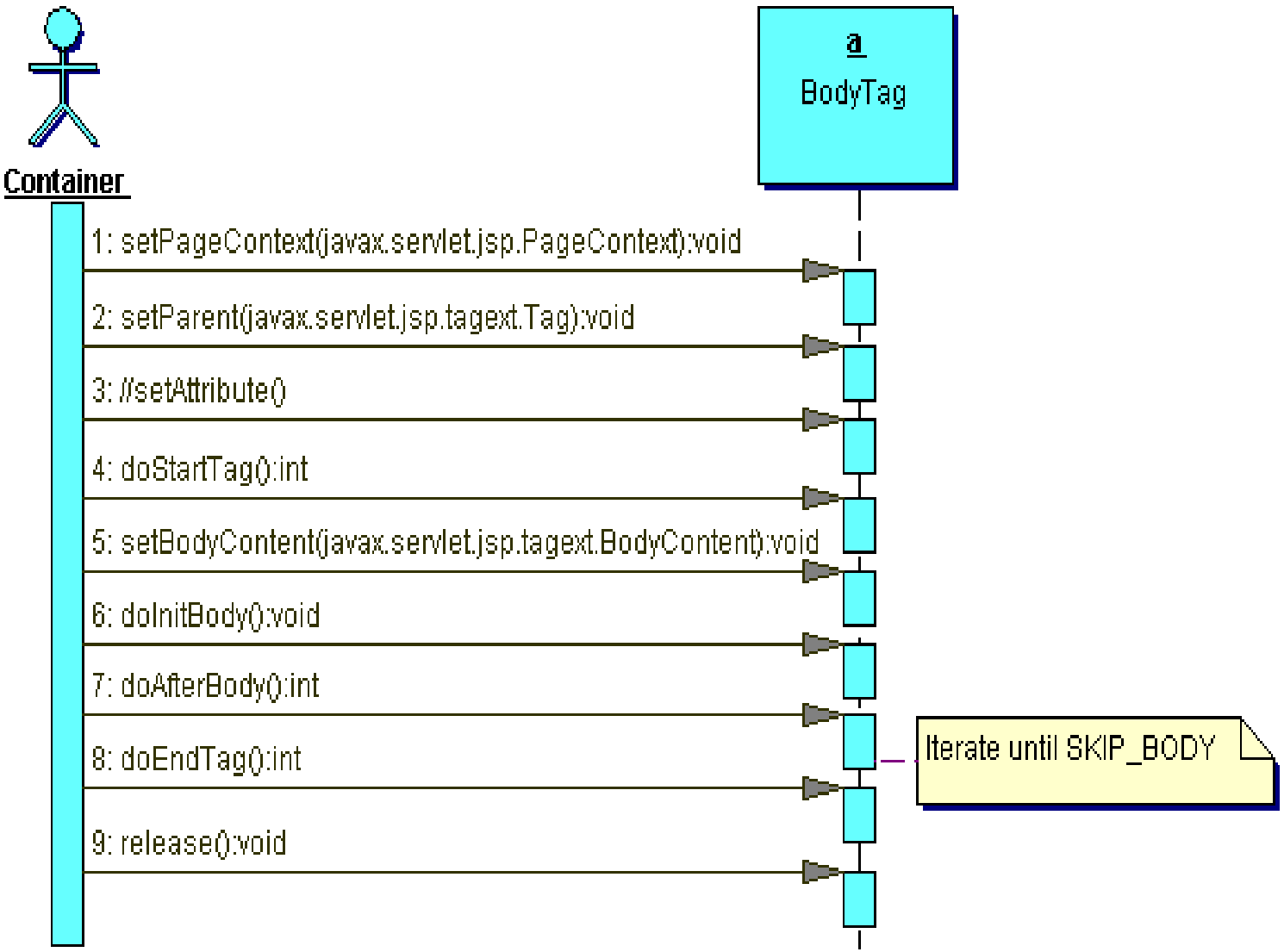

Container
a
Tag
1: setPageContext(javax.servlet.jsp.PageContext):void
2: setParent(javax.servlet.jsp.tagext.Tag):void
3: //setAttribute:void
4: doStartTag():int
5: doEndTag():int
6: release():void

Most of the 1st generation web containers used to create a tag handler object when a tag is encounter and call the method as shown above. Some web containers has implemented techniques like pooling the tag handlers and reusing the tag hadlers to improve the performance.

When we write a tag handler for a tag like <tag1 /> in the doStartTag method after carrying out initialization we have to return SKIP_BODY. If we have to support <tag2> Body </tag2> then we need to return EVAL_BODY_INCLUDE.

doEndTag will be called when the container sees the end of tag. In this method if we encounter a problem it is advise to return SKIP_PAGE other wise EVAL_PAGE. When the SKIP_PAGE is returned the remaining page will be skipped if EVAL_PAGE is returned web container evaluate the remaining part of the page.

To support a tag that accepts body we need to provide the methods defined in BodyTag interface. Some of these tag handlers (forEach) has to support evaluation of body for multiple times. Instead of directly implementing BodyTag interface we can create a sub class of BodyTagSupport class.


Container
a
BodyTag
1: setPageContext(javax.servlet.jsp.PageContext):void
2: setParent(javax.servlet.jsp.tagext.Tag):void
3: //setAttribute()
4: doStartTag():int
5: setBodyContent(javax.servlet.jsp.tagext.BodyContent):void
6: doInitBody():void
7: doAfterBody():int
8: doEndTag():int
9: release():void
Iterate until SKIP_BODY

doInitBody will be called only once but doAfterBody will be called multiple times till the method returns SKIP_BODY.

JSTL tag libraries provides a pair of tlds for every tag library. Ex: For core tag library we have c.tld and c-rt.tld. In c.tld for attributes we have,

<rtexprvalue>false</rtexprvalue> and in c-rt.tld we have ,

<rtexprvalue>true</rtexprvalue> for all attributes.

<inettag:mathops op1 =“14” op2=“10” oper=“+” />

<inettag:mathops op1 =“14” op2=“10” />

While designing the tags depending upon the requirement developers can stress to support set of attributes ex: op1, op2 and specify whether an attribute is required or not in the tld file. Ex: op1, op2 are required and oper is not required.

In order to support an attribute x as part of tag handler we need to provide setX method. For above tags we need to provide setop1, setop2 and setoper methods.

As most of the tags support an attribute ID, setID is provided.

We need not provide the implementation of general purpose tags like for, switch, forward… as these are available in JSTL. We need to implement application specific tags.

STRUTS:

A frame work like struts provides a set of classes implemented using an architecture. Frame work provides a proven procedure for implementing a project. Struts is based on *MVC* architecture. It contains a set of tag libraries like struts-bean, struts-html, struts-logic …

To simplify the development of struts based projects struts team has provided struts-blank.war. This can be used as starting point for the development of a web application using struts frame work.

Struts-bean tag library provides functionality similar to i18n tag library of *JSTL*.

In the project using struts frame work we need to copy struts related jar file in *WEB-INF/lib* and tld files of struts tag lib has to copied under *WEB-INF*.

To develop a struts based application we need to add struts tag library description to *web.xml* plus we need to add,

```
<servlet>
<servlet-name>action</servlet-name>
   <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
</servlet>
```

We need to provide servlet mapping as,

```
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
 </servlet-mapping>
```

The above servlet handles all the requests that matches with *.do. Action servlet is called as struts controller. This servlet has to be loaded when the application is deployed. Action servlet reads the information provided in configuration file *struts-config.xml.*

When a struts based application is deployed web container creates the controller servlet and this servlet reads the information available in *struts-config.xml*. We can manually edit *struts-config.xml* or use a product struts console to create the file.

It is highly recommended to use the resource files instead of hard coding the messages in JSP file.

Html tag library is used to generate the html tags.

The bean tag library provides the tags like *message* – to get the message from the resource bundle, *cookie*, *include*, *resource*, *write* – to write the value of bean property, *logic tag library* provides the tags like *present*, *equal*, *greaterequal*, ….

Struts automatically the process of getting the form data, validating form data and processing the form data. It is always advisable to use

<html:form action="xyz.do" /> instead of using form tag (normal html tag)

In struts frame work to handle the form data we need to create a form bean.

Procedure:

• Identify the names of the form fields ex: assume the form contains fields like empno, ename, desg, sal …

• Create a class xxxform as a subclass of *org.apache.struts.action.ActionForm* and provide the setters and getters corresponding to form fileds.

• Provide the *reset()* method in formbean class. In this method set the default values for the properties.

• If required provide the method *validate()*.

In validate method we need to provide the code that validates the input and returns a null if there are no errors. If there are any errors we need to return the errors.

As part of struts frame work we have *actionError* class that can be used to represent a single error. We can store multiple such errors inside an object of type *ActionErrors*.

To process an action request we need to provide an Action object

Procedure:

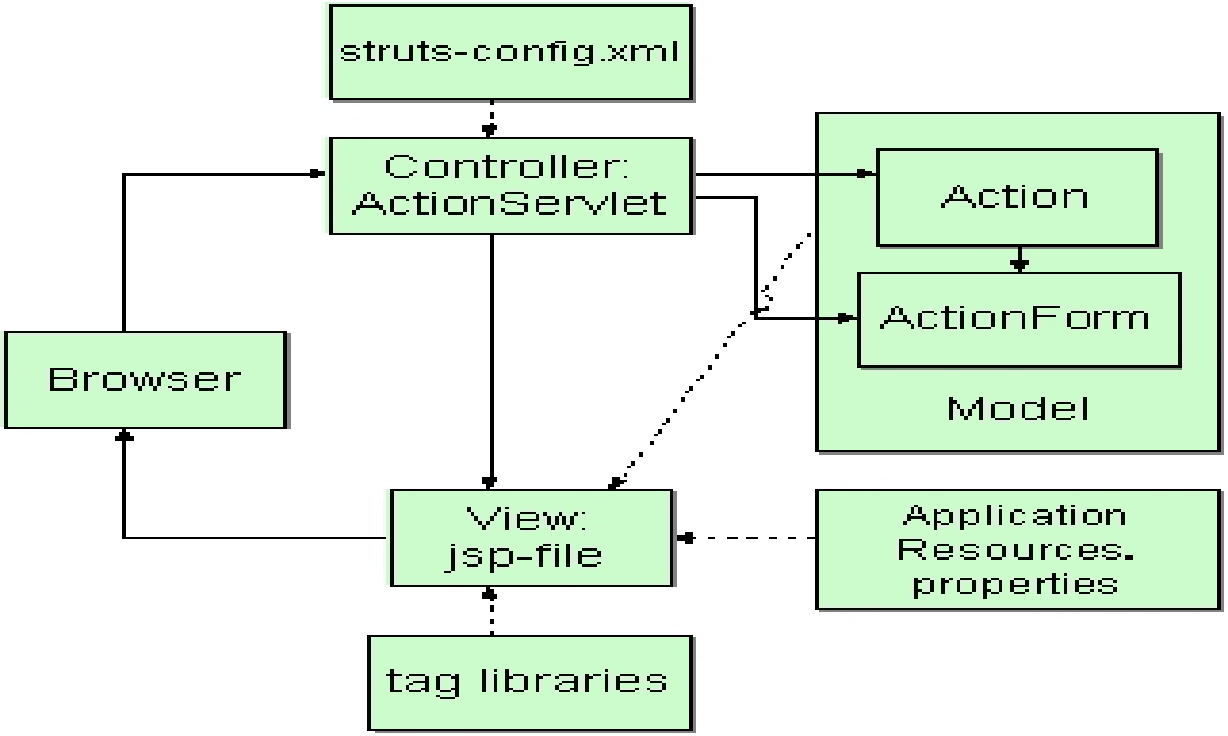
• Create an action class by extending *Action* class

• Provide the method *execute()*

In almost all web applications we need to carry out the following steps when user submits the data

• Get the data

• Validate the data if data is valid process it otherwise redisplay input form with the list of errors.



For struts UML diags visit the site

http://rollerjm.free.fr/pro/Struts11.html

When a request is send to perform an action like addemp.do struts framework creates formBean and sets the properties of formBean using the data submitted by user. It validates the data if the data is not valid it resends the form with errors. If there are no errors it executes the action. Depending upon the result of action struts framework picks up the view and send the result to the browser.

See the link for more info http://www.exadel.com/products_tutorials.htm (struts framework tutorials)

The developers of web applications using struts framework need to provide the *formBean, Action* classes and provide the information about these classes in *struts-config* file.

When we start struts bases applications the framework loads the resource that is listed in *message-resources* element in *struts-config* file as
<message-resources parameter="ApplicationResources" null="false"/>
<message-resources parameter="inetRes" null="false" key="Inet-key" />

In struts 1.1 we can specify multiple resource files as shown above, 1st resource will be loaded and stored inside servlet context using the key *org.apache.struts.actionmessage* and 2nd resource will be loaded by struts framework and it will be stored in the servlet context using the key *Inet-key*.

We can access the messages using *bean* tag as

<bean:message bundle="Inet-key" key="index.title">

Above tag picks up the value of index.title from the bundle *Inet-key*

If we use *null="True"* in *message-resource* element the framework throws an exception saying that the resource is not available. If we set *null="false"* instead of throwing an exception it returns *?? locale key ??*

We can access the parameterized messages using *bean* tag. In struts framework there is a limitation on no of parameters we pass to a parameterized message
arg{0} … arg{4}

As part of struts some facilities are provided if we want to extend those capabilities of struts framework we can provide our own plugins. Ex: struts framework can't validate our inputs automatically. A plug-in called struts-validator is provided to take care of validation.

To use a validator plug-in we need to add the *plug-in* element as

<plug-in className="org.apache.struts.validator.ValidatorPlugIn">

In order to handle the form we need not develop a formbean class on our own. In place of our own formBean we can use *dynamic action form* supplied as part of struts. In order to use the *dynamicformbean* we need to add entries in struts-config file as

```
<form-bean name="userinfo" type="org.apache.struts.action.DynaActionForm">
 <form-property name="fname" type="java.lang.String"/>
 <form-property name="lname" type="java.lang.String"/>
</form-bean>
```

We can avoid writing the form beans by using *DynaAactionForm*

To access the properties of the bean we need to use formbean.get("name of property");

To use validator framework

- Add validator plug-in info to struts-config.xml
- Add errorxxx entries to application resources file (these resources will be used by struts framework)
- Design the form and place validation rules in *validation.xml* file

It is better not to modify validator-rules.xml file. This contains the information about the java scripts that are required for validations on client side and the classes that are used for validation.

To use validator framework we need to create a formbean by extending *ValidatorForm* and provides *reset*, *getxxx*, *setxxx* methods. If we need to provide additional validations that are not supported by the validator framework we have to implement *validate()* method.

We need to define the validation rules specific to our application in *validation.xml*

```
<formset>
    <form    name="vform">
      <field    property="age" depends="required,integer"> </field>
      <field    property="e-mail" depends="required,email"> </field>
</formset>
```

Developing a web application that validate a form on the browser is easy with struts.

Procedure for generating java script with validator framework

• In jsp file that generates the form we need to add
<*html:javascript formName="vform"*> script uses a variable bcancel if the value of the bcancel is true validator will not be carried out.

• In html:form tag add, *<html:form action="/tv.do" onsubmit="return validateVform(this);">*

• Before we call validatevform we need to set bcancel to false. This can be done by adding, *<html:submit property="submit" onclick="bCancel=false;">*

The validator framework uses the java script functions that are part of *validator-rules.xml*

Let us assume there is a form field with the name UserName, from this field we want to accept the inputs like (i) Abc (ii) Xyz but not (iii) 12xy. For this in struts framework we can use the mask validator. For this validator we need to provide regular expression.

Let us assume we want to accept 5 chars from the user and 1st character must be capital and remaining 4 chars must be small letters. For this requirement we can use Regular Expression as shown below,

/w stands for a word character

Some times we may need to specify an option like A or X or Z for that similar to the range we can specify as, *[AXZ]*

We can fix the value for a char position by directly writing the char as, *[A]x[Z]*

We can define a constant in validation.xml as,

<constant>
<constant-name>zip</constant-name>
<constant-value>^\d{5}\d$</constant-value>*
</constant>

In RE ^ stands for start with, $ - end with

/W – non word char
/d – digit
/D – non digit
** -- zero or more + -- one or more ? – zero or one*
we can use |(or) as [A|Z]

When ^ is used as *[^abc]* it accepts the char other than abc

In some programs if we need to accept RE we can use regexp package available at jakarta site.

In a typical strut based application some of the URLs will be pointing to jsp files ex: form.jsp and some of the URLs pointing to strut actions ex: process.do. By using the Action class *org.apache.struts.actions.Forward* we can map a struts action to a jsp file.

For using this we need to add the following mapping.
<action input="/index.jsp" path="/abc" type="
org.apache.struts.ForwardAction" parameter="/two.jsp"> </action>

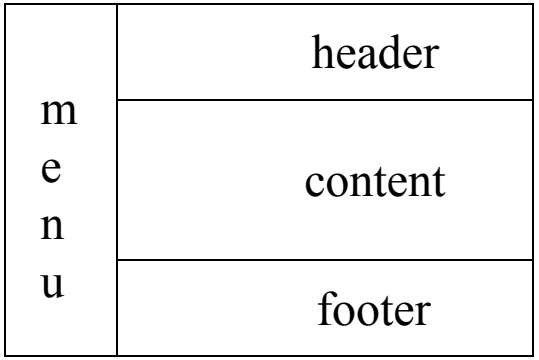
With above mapping struts framework will execute two.jp when a request is send for abc.do

In most of the cases to process a form and automatically validate it we did not use our own form beans. For this we can make use of DynaValidatorForm.

Most of the IDEs supports struts framework. These IDEs automatically generate the configuration files plus generate the skeleton code. We can also use struts console to edit config file.

TILES:

Before we actually develop the web application we need to decide about the layout of the pages. Most of the websites uses the layout. Once the layout is designed, the designers create the template according to the layout.

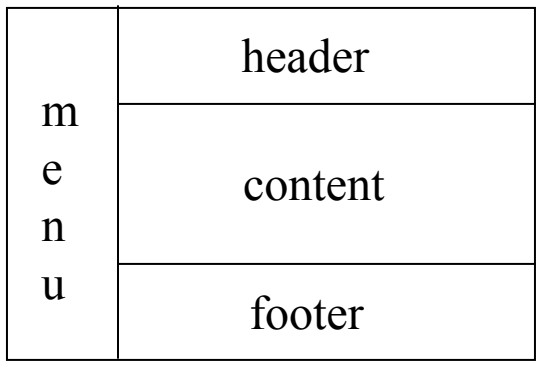


In above ex the webpage split-up into multiple pieces we can treat these pieces as tiles. Tiles framework can be used to design the web pages with a consistent look.

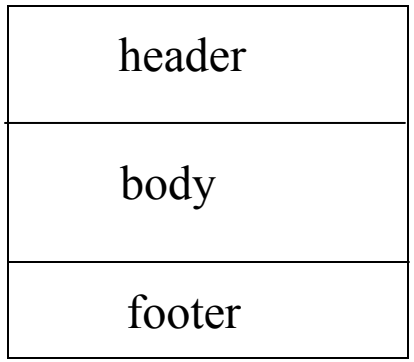
As part of tiles plug-in we have support for a set of tags, as part of tiles tag lib we have the support importattribute, useattribute, getasstrin, get, beanname, add, put, insert…

To use struts tiles framework we need to define a layout or use the layout files that are provided as part of the tiles. Once the layout is defined we can use the same layout as part of multiple pages.

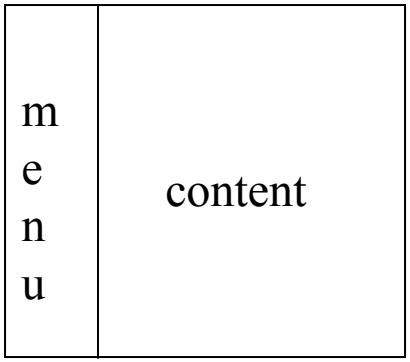
We can design the websites using the layout shown below, for this we will define a layout that splits the page into 3 parts.



L



L1



L2

As part of tiles we get a set of standard layouts these layouts are available in \layout.

Instead of defining the layouts in jsp files we can create a layout definition. We store the definition of layout in xml file.