What is HTTP?

HTTP stands for Hyper Text Transfer Protocol which is an application level protocol for transferring textual data between client and server.

HTTP is stateless protocol.

Web Client such as web browser sends the request to server, server responds and sends the requested data (such as a HTML document) or returns error code and closes the connection. Once the response is returned, server doesn t remember anything about the client.(even the very next request)

Http request methods and Headers

eg : URL -- http://www.server.com:8080/bank/login.jsp

Whenever the client sends the request to server for any resource such as a HTML document, it specifies

1. HTTP method (get/post/put/delete....)
2. Request URI (eg : /bank/login.jsp)
3. Protocol version
4. Optional Header information(eg : accept , cookies)

After the client sends the request, server processes the request and sends the response.

HTTP Response contains

1. Response status information(eg : 404/200)
2. Response headers (eg : cookies/resp cont type

eg : text/html, application/json image/gif.... /cont length)

3. Response data.

Following is an example of HTTP request which uses GET request method to ask for a HTML document named tutorial.html using HTTP protocol version 1.1

GET /tutorial.html HTTP/1.1

Following is the example of request header, which provides additional information about the request.

User-Agent: Mozilla/4.0 (compatible; MSIE 4.0; Windows 95)

Accept: image/gif, image/jpeg, text/*, */*

Above header specifies the information about the client software and what MIME(Multi-purpose Internet Mail Extension) type the client accepts in response, using User-Agent and Accept headers.

Http request methods

The request method indicates to the server what action to perform on the resource identified by the request-URI.

HTTP 1.1 specifies these request methods:

GET, POST , HEAD,OPTIONS, PUT, TRACE, DELETE, CONNECT.

Servlets can process any of the above requests.

But GET and POST methods are most commonly used.

The GET request method -- safe(doesn't change state of the resource) & idempotent (repeated reqs do not have any further side effects after the first request)

The GET request is used typically for getting static information from the server such as HTML document, images, and CSS files.

It can be used to retrieve dynamic information also by appending query string at the end of request URI.

Query String

Query string is used to pass additional request parameters along with the request.

Query string format

URL?name1=value1&name2=value&name3=value3 .

eg -- http://www.abc.com/test/login.jsp?userid=abc

The POST request method

The POST request method is typically used to access dynamic resources.

POST request is used to

1. send the large amount of data to the server.
2. upload files to servers
3. send serializable Java objects or raw bytes.

GET Vs POST

1. GET request sends the request parameters as query string appended at the end of the request URL, whereas POST request sends the request parameters as part of the HTTP request body.

2.Unlike GET, POST request sends all the data as part of the HTTP request body, so it doesn t appear in browser address bar and cannot be bookmarked.

3. GET -- non-secure, POST -secure

GET -- idempotent

POST---non-idempotent

Typically use GET -- to retrieve stuff from server.

use POST -- for changing some state on server(something like update)

4. Some web servers limit the length of the URL.

So if in GET request -- too many parameters are appended in query string and URL exceeds this length, some web servers might not accept the request.

For POST -- no such limitations.

A web server is the combination of computer and the program installed on it.

Web server interacts with the client through a web browser. It delivers the web pages to the client and to an application by using the web browser and the HTTP protocols respectively.

We can also define the web server as the package of large number of programs installed on a computer connected to Internet or intranet for downloading the requested files using File Transfer Protocol, serving e-mail and building and publishing web pages.

A web server works on a client server model. A computer connected to the Internet or intranet must have a server program. While talking about Java language then a web server is a server that is used to support the web component like the Servlet and JSP. Note that the web server does not support to EJB (business logic component) component.

A computer connected to the Internet for providing the services to a small company or a departmental store may contain the HTTP server (to access and store the web pages and files), SMTP server (to support mail services), FTP server (for files downloading) and NNTP server (for newsgroup). The computer containing all the above servers is called the web server.

Internet service providers and large companies may have all the servers like HTTP server, SMTP server, FTP server and many more on separate machines. In case of Java, a web server can be defined as the server that only supports to the web component like servlet and jsp. Notice that it does not support to the business component like EJB.

* The term web server can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver web content that can be accessed through the Internet.

* The most common use of web servers is to host websites, but there are other uses such as gaming, data storage or running enterprise applications.

* The primary function of a web server is to deliver web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to text content.

* A user agent, commonly a web browser or web crawler, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or an error message if unable to do so. The resource is typically a real file on the server's secondary storage, but this is not necessarily the case and depends on how the web server is implemented.

* While the primary function is to serve content, a full implementation of HTTP also includes ways of receiving content from clients. This feature is used for submitting web forms, including uploading of files.

* Many generic web servers also support server-side scripting using Active Server Pages (ASP), PHP, or other scripting languages. This means that the behaviour of the web server can be scripted in separate files, while the actual server software remains unchanged. Usually, this function is used to create HTML documents dynamically ("on-the-fly") as opposed to returning static documents. The former is primarily used for retrieving and/or modifying information from databases. The latter is typically much faster and more easily cached but cannot deliver dynamic content.

Content Type

Content Type is also known as MIME (Multipurpose internet Mail Extension) Type. It is a HTTP header that provides the description about what are you sending to the browser.

MIME is an internet standard that is used for extending the limited capabilities of email by allowing the insertion of sounds, images and text in a message.

List of Content Types

There are many content types. The commonly used content types are given below:

text/html, text/plain, application/msword, application/jar , application/pdf , mages/jpeg , images/png ,images/gif, audio/mp3, video/mp4

Version Java EE 7 (J2EE 1.7)

What is J2EE ?

Consists of specifications only .

Which specs ? (Rules or contract)

Specifications of services required for any enterprise application.

What is enterprise application ?

-An enterprise application (EA) is a large software system platform designed to operate in a corporate environment -It includes online shopping and payment processing, interactive product catalogs, computerized billing systems, security, content management, IT service management, business intelligence, human resource management, manufacturing, process automation, enterprise resource planning ....

These specifications include ---

Servlet API,JSP(Java server page) API,Security,Connection pooling ,EJB (Enterprise Java Bean), JNDI(Naming service -- Java naming & directory i/f),JPA(java persistence API),JMS(java messaging service),Java Mail, Java Server Faces , Java Transaction API, Webservices support etc...

* Vendor of J2EE specs -- Oracle / Sun.

Implementation -- left to vendors (J2EE server vendors)

J2EE compliant web server --- Apache -- Tomcat (web container)

Services implemented --- servlet API,JSP API,Security,Connection pooling,JNDI(naming service)

J2EE compliant web application folder structure
Tomcat ---<tomcat>/webapps --hot dep folder

<dac_web> --name of folder --context root
--or root of the web appln.
ctx=web appln

<src>
.java source
for servlets.

directly under root
.html,.jsp,.js,css,media,ap
plets
public , accessible to clnt
via URL

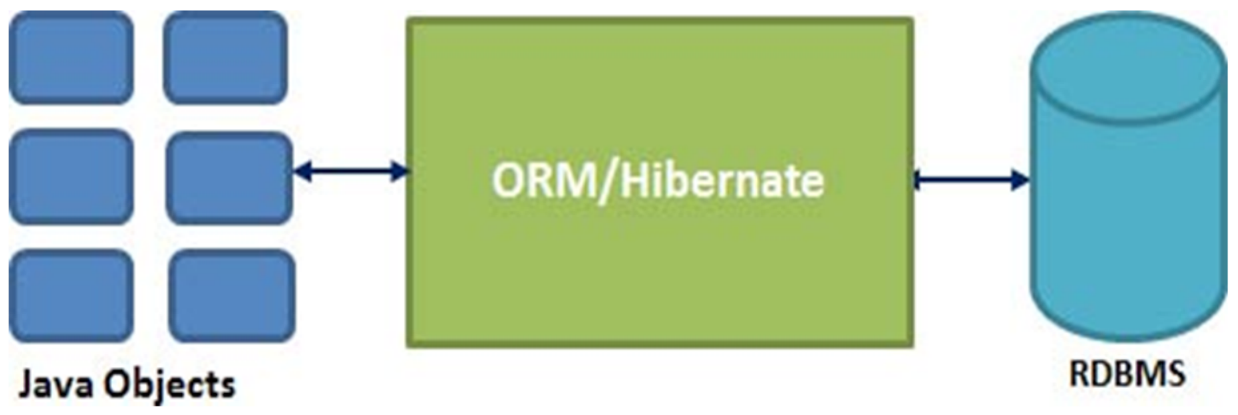
<WEB-INF> -- mandatory
private --not accessible to clnt
1. web.xml = Deployment desc
of the web appln

who -- Web container (=s.s JVM
within web server, run time env for
web comps)
when -- @ dep time
what ---xml tags ---

instrs meant for WC eg : session
tmout,welcome page,sec tags,servlet
dep tags.....

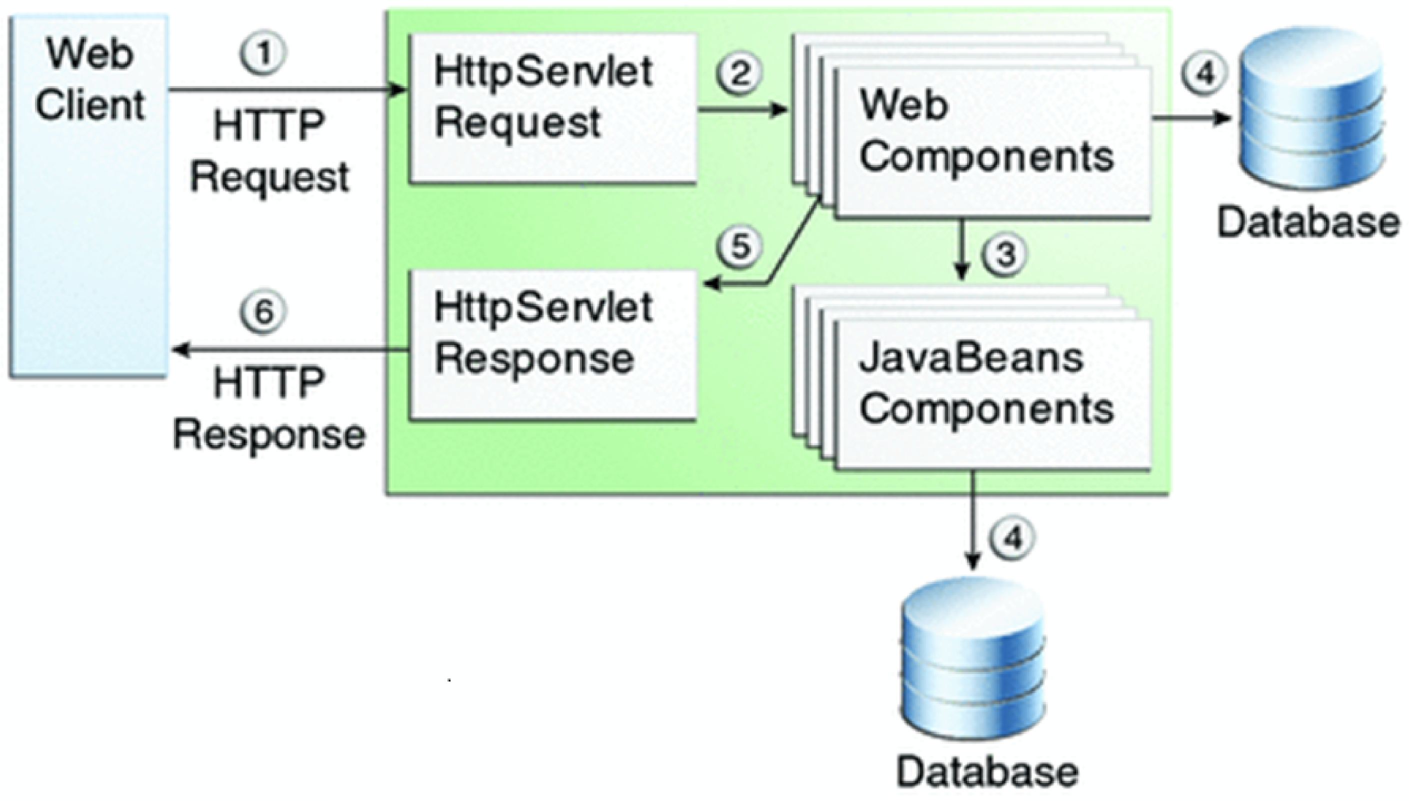
2. <classes> -- .class files (for
servlets)
3. <lib> -- .jar files (eg : JDBC
JAR,hib jars,struts jars,spring jars...)

ORM Overview (Object Relational Mapping)



Object World	Relational Database World
POJO classes	DB Tables
POJO properties	Table Columns
POJOs	Table rows
Unique ID Property	Primary Key

Web Request Handling



Typical Layers in DB programming

Tester -- main(..) based. --- UI & manage DAO layer.

DAO layer (Data access object layer)
DAO I/F --- to declare CRUD requirements
JDBC based DAO's Imple class
1. public & pkged class.
2. state -- Connection,STs/PSTs/CSTs
3. Constr -- init time jobs (cn,psts/csts)
4. clean up -- close PSTs/STs/CSTs/cn
5. CRUD methods

DBUtils
--get FIXED DB conn.

POJO/Entity/Model/Domain classes --object view of DB data
POJO class --- DB table, POJO property -- Table Column,
POJOs -- rows

DB Tier/Layer(EIS layer)
Tables/rows/cols....

Servlet API implementation classes in server provided JARs

Servlet API

javax.servlet.Servlet ---i/f --consists of mainly life cycle methods --init,service,destroy

javax.servlet.GenericServlet --abstract imple class of Servlet i/f
init & destory --concrete , service --abstract
represents pro tocol inde. servlet.
(not reco to be used with HTTP)

javax.servlet.http.HttpServlet -- abstract sub class of GenericServlet.
concrete service method.
public void service(ServletRequest rq,ServletResponse rs) throws ServletException,IOException

Checks for HTTP request method

get post put delete

protected void doGet(HttpServletRequest rq,HttpServletResponse rs) throws SE,IOExc
doPost /doPut/doDelete ---

For creating a servlet ---eg : HelloServlet
public class HelloServlet extends H.S
{
 //MUST override at least one of the doXXX methods --eg : doGet
 For complete understanding of life cycle
 override --init , doGet & destroy.
 doGet ---

 1. set response content type
 API of HttpServletResponse
 public void setContentType(String type)
 eg : rs.setContentType("text/html");
 2. To send response from servlet --- clnt browser , attach data strm.
 API of HttpServletResponse
 public PrintWriter getWriter()
 PW --char , buf , o/p strm conn from server to clnt for sending resp

 3. Send dyn contents.
 PW API -- print/write
 4. close PW

After creating servlet class -- supply deployment instrs to WC
2 ways.
1. Via annotation -- run time anno.
pkg -- javax.servlet.annotation
@WebServlet("/hi")
public class HelloServlet ....{...}
class level anno.

Who -- WC
When -- @ dep time of web app
Meaning ---If clnt send request ending in url-pattern /hi , use HelloServlet class for its servicing.

These specifications include ---

Servlet API,JSP(Java server page) API,Security,Connection pooling ,EJB (Enterprise Java Bean), JNDI(Naming service -- Java naming & directory i/f),JPA(java persistence API),JMS(java messaging service),Java Mail, Java Server Faces , Java Transaction API, Webservices support etc...

* Vendor of J2EE specs -- Oracle / Sun.

Implementation -- left to vendors (J2EE server vendors)

J2EE compliant web server --- Apache -- Tomcat (web container)

Services implemented --- servlet API,JSP API,Security,Connection pooling,JNDI(naming service)

WHY J2EE

1. Can support different types of clnts --- thin client(web clnt)

thick clnt --- application clnt

smart clnts -- mobile clnts

2. J2EE server independence --- Create & deploy server side appln --on ANY J2ee compliant server --- guaranteed to produce SAME results w/o touching or re-deploying on ANY other J2EE server

3. Ready made implementation of primary services(eg --- security, conn,pooling,email....)--- so that J2EE developer DOESn't have to worry about primary services ---rather can concentrate on actual business logic.

Layers involved in HTTP request-response flow

Web browser sends the request (URL)

eg : http://www.abc.com:8080/day1.1/index.html

/day1.1 --- root / context path /web app name

Host --Web server--Web Container(server side JVM)--Web application---HTML/JSP/Servlet....

What is dyn web application --- server side appln --deployed on web server --- meant for servicing typically web clnts(thin) -- using application layer protocol HTTP /HTTPS

(ref : diag request-resp flow)

Read --HTTP basics , request & response structure.

Objective ?: Creating & deploying dyn web appln on Tomcat -- For HTML content

IDE auto creates J2EE compliant web application folder structure .

Its details -- Refer to diag (J2EE compliant web app folder structure)

<u>**What is Web container --- (WC) & its jobs**</u>

1. Server side JVM residing within web server.

Its run-time env for dyn web components(Servlet & JSP,Filter) .

Jobs ---

1. Creating Http Request & Http response objects
2. Controlling life-cycle of dyn web comps (manages life cycle of servlet,JSP,Filters)
3. Giving ready-made support for services --- Naming,security,Conn pooling .
4. Handling concurrent request from multiple clients .
5. Managing session tracking...

What is web.xml --- Deployment descriptor one per web appln

created by -- dev

who reads it -- WC

when --- @ deployment

what --- dep instrs --- welcome page, servlet deployment tags, sess config, sec config......

Why servlets? --- To add dynamic nature to the web application

What is a servlet ?

-- Java class -- represents dynamic web component - whose life cycle will be managed by WC(web container)

no main method

life cycle methods --- init,service,destroy

Job list

1. Request processing **2.** B.L **3.** response generation

4. Data access logic(DAO class --managing DAO layer) 5. Page navigation

Servlet API details --refer to diag servlet-api

Creating & deploying Hello Servlet.

1. Using @WebServlet annotation

OR

2. Using XML tags

How to deploy a servlet w/o annotations --- via XML tags in web.xml

* At the time of web app deployment ---WC tries to populate map of url patterns , from XML tags (from web.xml). If not found --- will check for @WebServlet annotation.

How to read request parameters in Servlet ?

javax.servlet.ServletRequest i/f methods

1. public String getParameter(String paramName)
2. public String[] getParameterValues(String paramName)

Page Navigation Technique

Page Navigation=Taking user from 1 page to another page.

2 Ways

1. Client Pull

Taking the client to the next page in the NEXT request

1.1 User takes some action --eg : clicking on a button or link & then client browser generates new URL to take user to the next page.

1.2 Redirect Scenario

User doesn't take any action. Client browser automatically generates new URL to take user to the next page.(next page can be from same web appln , or diff web appln on same server or any web page on any srvr)

API of HttpServletResponse

public void sendRedirect(String redirectURL)

eg : For redirecting client from Servlet1 (/s1) to Servlet2 (/s2) , use

response.sendRedirect("s2");

Page Navigation Techniques

(Taking thin clnt (user) from one page to other page.)

Client Pull — Min scope reqd=session scope.

In this tech. user is navigated to next page in NEW request

- **Clnt Pull I**

  BOTH (client --person & clnt Browser) is involved.

  eg -- Button click,link .

- **Clnt Pull II**

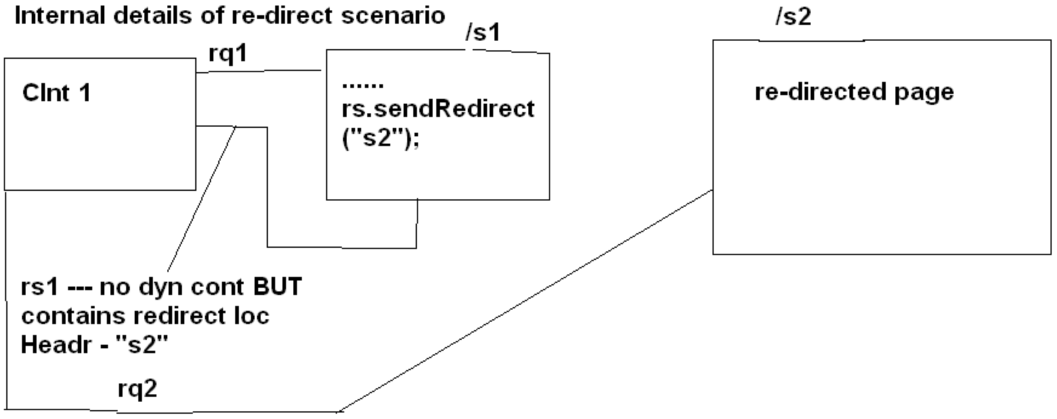
  Client (as a person) is NOT involved BUT clnt browser is involved.
  Re-direct scenarion.
  API of HttpServletResponse
  void sendRedirect(String redirectURL)

Server Pull

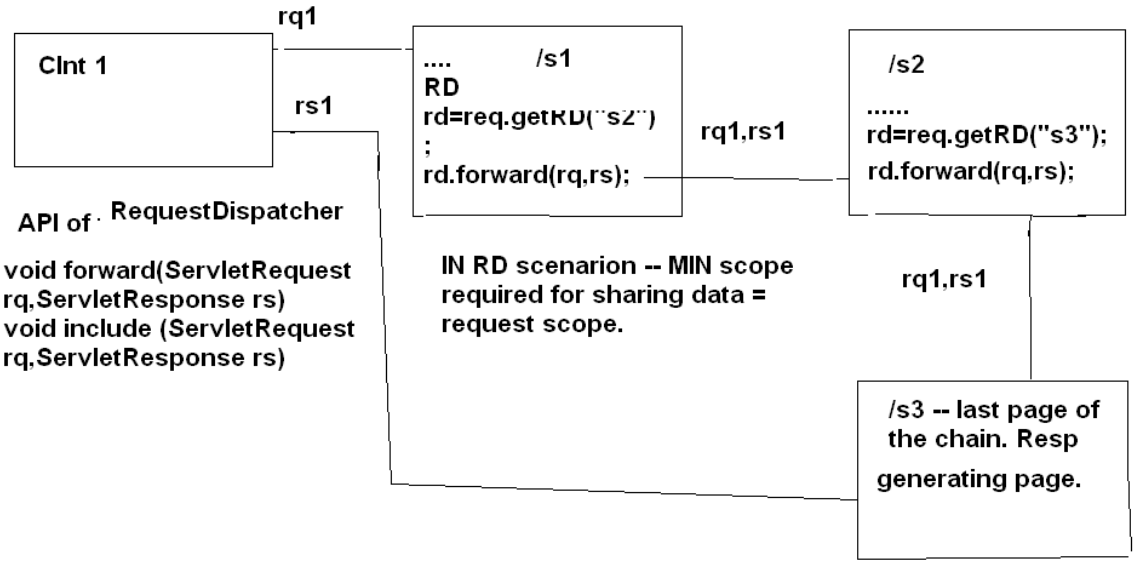
In this tech. user is navigated to next page in same request.

Resource chaining or request dispatching technique. --- NO round trip delay is involved since entire navigation takes place in server space.
API of ServletRequest
1. Get Request dispatcher object to chain resources(JSP,Servlet)
RequestDispatcher getRequestDispatcher(String dispatcherURL) 2. Use either forward or include scenario as appropriate.

Internal details of re-direct scenario



RequestDispatcher forward scenario. (Major MVC frameworks use this as a default.

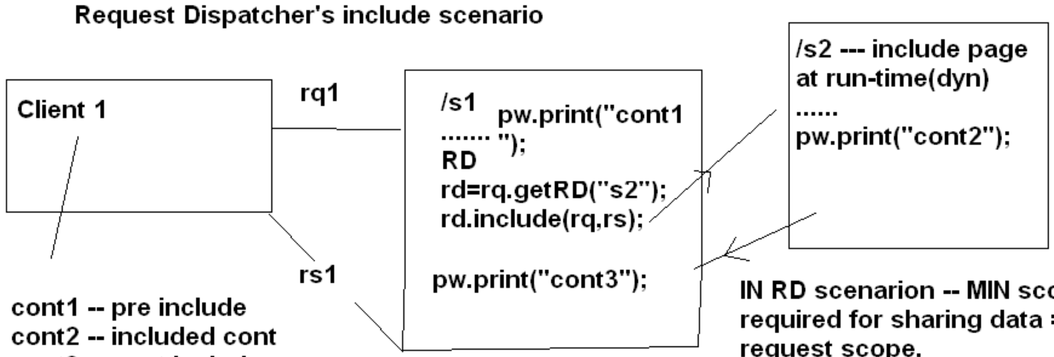


API of RequestDispatcher

void forward(ServletRequest rq,ServletResponse rs)
void include (ServletRequest rq,ServletResponse rs)

IN RD scenarion -- MIN scope required for sharing data = request scope.

Request Dispatcher's include scenario



cont1 -- pre include
cont2 -- included cont
cont3 -- post include

IN RD scenarion -- MIN scope required for sharing data = request scope.

2. Server Pull.

Taking the client to the next page in the same request.

Also known as resource chaining or request dispatching technique.

Client sends the request to the servlet / JSP. Same request can be chained to the next page for further servicing of the request.

Steps

1. Create Request Dispatcher object for wrapping the next page(resource --can be static or dynamic)

API of ServletRequest

javax.servlet.RequestDispatcher getRequestDispatcher(String path)

2.Forward scenario

API of RequestDispatcher

public void forward(ServletRequest rq,ServletResponse rs)

-This method allows one servlet to do initial processing of a request and another resource to generate the response. (i.e division of responsibility)

-Uncommitted output in the response buffer is automatically cleared before the forward.

-If the response already has been committed(pw flushed or closed) , this method throws an IllegalStateException.

-Limitation --only last page in the chain can generate dynamic response.

3. Include scenario

API of RequestDispatcher

-public void include(ServletRequest rq,ServletResponse rs)

-Includes the content of a resource @run time (servlet, JSP page, HTML file) in the response. -- server-side includes.

-Limitation -- The included servlet/JSP cannot change the response status code or set headers; any attempt to make a change is ignored.

What is a Session?

* Session is a conversional state between client and server and it can consists of multiple request and response between client and server. Since HTTP and Web Server both are stateless, the only way to maintain a session is when some unique information about the session is passed between server and client in every request and response.

* HTTP protocol and Web Servers are stateless, what it means is that for web server every request is a new request to process and they cant identify if its coming from client that has been sending request previously.

* But sometimes in web applications, we should know who the client is and process the request accordingly. For example, a shopping cart application should know who is sending the request to add an item and in which cart the item has to be added or who is sending checkout request so that it can charge the amount to correct client

What is the need of session tracking?

1. To identify the clnt among multiple clnts

2. To remember the conversational state of the clnt(eg : list of the purchased books/ shopping cart) throughout current session

session = Represents duration or time interval

* Consists of all requests/resps coming from/to same clnt from login to logout or till session expiration tmout.

(def tmout for Tomcat =30mins)

There are several techniques for session tracking.

1. Plain Cookie based scenario
2. HttpSession interface
3. HttpSession + URL rewriting

<u>Techniques</u>

1. Plain Cookie based scenario

What is a cookie?

Cookie is small amount of text data.

Created by -- server (servlet or JSP prog) & downloaded (sent) to clnt browser---within response header

Cookie represents data shared across multiple dyn pages from the SAME web appln.

Steps :

1. Create cookie/s instance/s

javax.servlet.http.Cookie(String cName,String cVal)

2.Add the cookie/s to the resp hdr.

HttpServletResponse API :

void addCookie(Cookie c)

3. To retrieve the cookies :

HttpServletRequest :

Cookie[] getCookies()

4.Cookie class methods :

String getName()

String getValue()

void setMaxAge(int ageInSeconds)

def age =-1 ---> browser stores cookie in cache

=0 ---> clnt browser should delete cookie

>0 --- persistent cookie --to be stored on clnt's hard disk.

int getMaxAge()

Disadvantages of pure cookie based scenario

0. Web dev (servlet prog) has to manage cookies.

1. Cookies can handle only text data : storing Java obj or bin data difficult.

2. As no of cookies inc., it will result into increased net traffic.

3. In cookie based approach : entire state of the clnt is saved on the clnt side. If the clnt browser rejects the cookies: state will be lost : session tracking fails.

How to redirect client automatically to next page ? (in the NEXT request)

API of HttpServletResponse

public void sendRedirect(String redirectLoc)

eg : resp.sendRedirect("s2");

IMPORTANT :

WC -- throws

java.lang.IllegalStateException: Cannot call sendRedirect() after the response has been committed(eg : pw.flush(),pw.close()...)

Technique # 2 : Session tracking based on HttpSession API

In this technique , entire state of the client is saved on the server side data structure (Http Sesion object)

BUT the key to this Http Session object is STILL sent to client in form of a cookie.

*Above mentioned , disadvantages ---0, 1 & 2 are reomved.

BUT entire session tracking again fails , if cookies are disabled.

Steps for javax.servlet.http.HttpSession i/f based session tracking.

1. Get Http Session object from WC

API of HttpServletRequest ---

HttpSession getSession()

Desc --- Servlet requests WC to either create n return a NEW HttpSession object(for new clnt) or ret the existing one from WC's heap.

* Above method creates either a NEW HTTP session obj for new user or returns existing HTTP session object for old user.

HttpSession --- i/f from javax.servlet.http

NEW --- HttpSession<String,Object> --empty map

String,Object ---- (entry)= attribute

OR

HttpSession getSession(boolean create)

2. : How to save data in HttpSession?(scope=entire session)

API of HttpSession i/f

public void setAttribute(String attrName,Object attrVal)

equivalent to map.put(k,v)

eg : hs.setAttribute("cart",l1);

3. For retrieving session data(getting attributes)

public Object getAttribute(String attrName)

4. To get session ID (value of the cookie whose name is jsessionid -- unique per client)

String getId()

4.5 How to remove attribute from the session scope?

public void removeAttribute(String attrName

5. How to invalidate session?

HttpSession API

public void invalidate()

(WC marks HS object on the server side for GC ---BUT cookie is NOT deleted from clnt browser)

6. HttpSession API

public boolean isNew()

7.How to find all attr names from the session ?

public Enumeration<String> getAttributeNames()

--rets java.util.Enumeration of attr names.

What is an attribute ?

*** Attribute** = server side object(entry/mapping=key value pair)

*** Who creates server side attrs ?** -- web developer (servlet or JSP prog)

Each attribute has --- attr name(String) & attr value (java.lang.Object)

Attributes can exist in one of 3 scopes --- req. scope,session scope or application scope

1. Meaning of req scoped attr = attribute is visible for current req.

2. Meaning of session scoped attr = attribute is visible for current session.(shared across multiple reqs coming from SAME clnt)

3. Meaning of application scoped attr = attribute is visible for current web appln.(shared across multiple reqs from ANY clnt BUT for the SAME web application)

Why J2EE

1. can support diff clnts **2.** server inde. (How -- separation bet specs & imple)

3. rdy imple of prim services

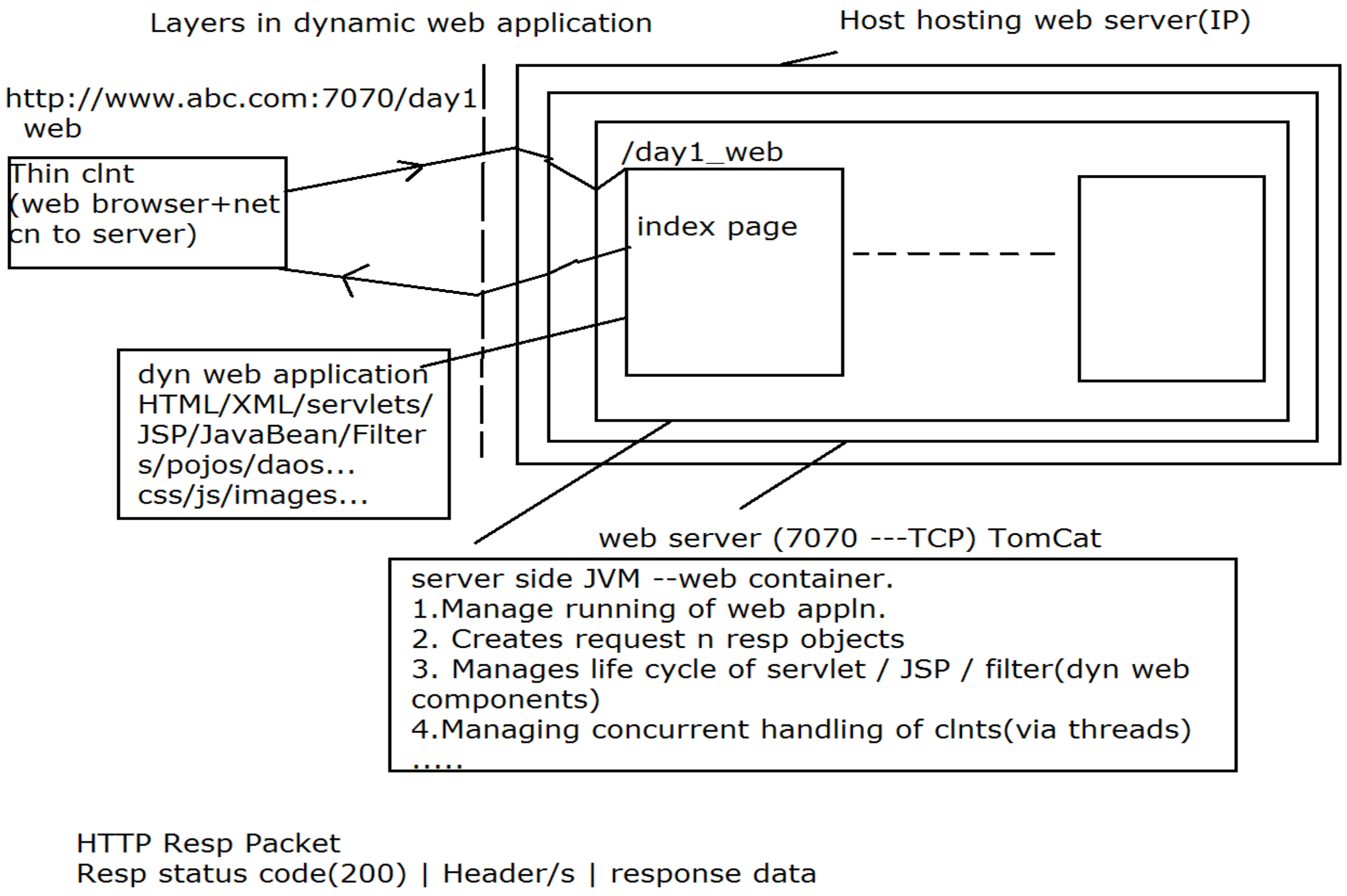
what is it ?

specs --- rules/contract

prim services/ enterprise services

eg : servlet / JSP /JB,conn pooling , security,JTA..... * vendor -- Oracle/sun

Request response flow(Layers)


Layers in dynamic web application
Host hosting web server(IP)
http://www.abc.com:7070/day1 web
Thin clnt (web browser+net cn to server)
/day1_web
index page
dyn web application HTML/XML/servlets/ JSP/JavaBean/Filter s/pojos/daos... css/js/images...
web server (7070 ---TCP) TomCat
server side JVM --web container.
1.Manage running of web appln.
2. Creates request n resp objects
3. Manages life cycle of servlet / JSP / filter(dyn web components)
4.Managing concurrent handling of clnts(via threads)
.....
HTTP Resp Packet
Resp status code(200) | Header/s | response data

URL --- http://www.abc.com:8080/day2_web/validate?em=abc&pass=1234

Layers --- Host(IP) ---Web Server(Tomcat --TCP)--WC(s.s JVM , run time env dyn web comps --servlets,JSP,Filter)---web app (/day2_web)--

URI ---/ctx path /url-pattern ? query string

J2EE compliant web app folder structure

<day2_web>

<WEB-INF> -- web.xml , <classes> , <lib>

private from clnt

root (WebContent) --public

html/JSP/css/js/images...

<src> -- java source files

What is a Servlet ?

Its jobs

1. rq processing **2.** B.L **3.** page navigation

(taking clnt from 1 pgae to another ---clnt pull / server pull)

4. Manage DAO **5.** resp gen.

How do you create n deploy a servlet ?

```
public class Test exetnds H.S{
            @Override
            protected/public void doGet(HSReq rq,HSResp rs) throws SE,IOExc
             {  }
           }
```

javax.servlet.ServeltException

```
public ServletException(String message,
              Throwable rootCause)
```

<u>Centralized err handling in Servlets</u>

How ?

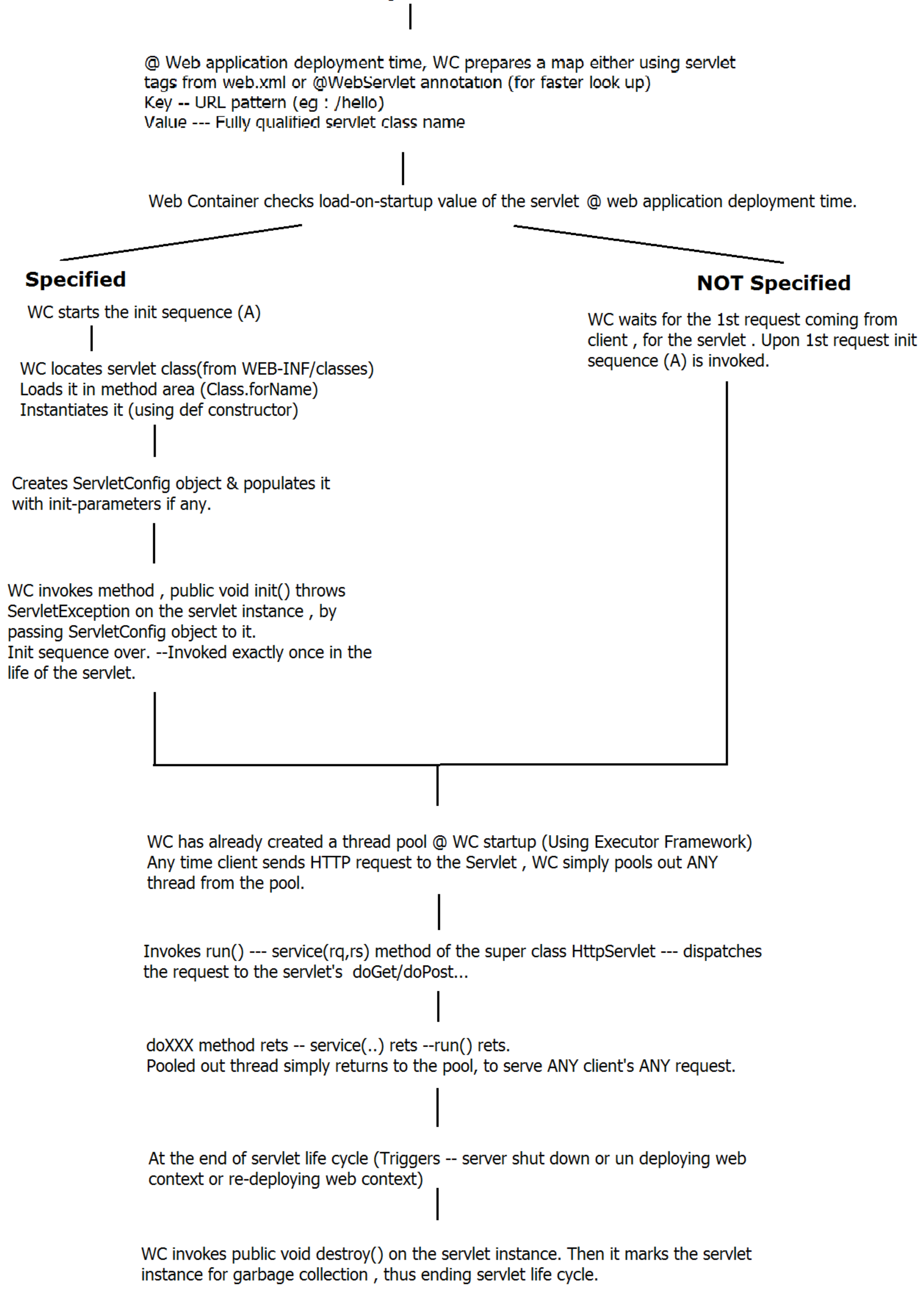
```
@Override
public void init() throws ServletException
{
try {
 // create dao inst
} catch(Exception e)
{
  throw new ServletException("err in init" +getClass().getName(),e);
}
}
```

eg : @WebServlet("/validate)

public class LoginServlet extends H.S {....}

OR xml tags

Servlet Life Cycle Managed by Web Container



@ Web application deployment time, WC prepares a map either using servlet tags from web.xml or @WebServlet annotation (for faster look up)
Key -- URL pattern (eg : /hello)
Value --- Fully qualified servlet class name

Web Container checks load-on-startup value of the servlet @ web application deployment time.

Specified

WC starts the init sequence (A)

WC locates servlet class(from WEB-INF/classes)
Loads it in method area (Class.forName)
Instantiates it (using def constructor)

Creates ServletConfig object & populates it with init-parameters if any.

WC invokes method , public void init() throws ServletException on the servlet instance , by passing ServletConfig object to it.
Init sequence over. --Invoked exactly once in the life of the servlet.

NOT Specified

WC waits for the 1st request coming from client , for the servlet . Upon 1st request init sequence (A) is invoked.

WC has already created a thread pool @ WC startup (Using Executor Framework)
Any time client sends HTTP request to the Servlet , WC simply pools out ANY thread from the pool.

Invokes run() --- service(rq,rs) method of the super class HttpServlet --- dispatches the request to the servlet's doGet/doPost...

doXXX method rets -- service(..) rets --run() rets.
Pooled out thread simply returns to the pool, to serve ANY client's ANY request.

At the end of servlet life cycle (Triggers -- server shut down or un deploying web context or re-deploying web context)

WC invokes public void destroy() on the servlet instance. Then it marks the servlet instance for garbage collection , thus ending servlet life cycle.

1. What will happen if u don't add / in url pattern ? --error(web app doesn't get deployed --- invalid url pattern (IllegalArgumentException))

2. What will happen --if u add "/url-pattern" in form action or href ? --- HTTP 404 (uses absolute url eg : http://host:port/hi)

load-on-startup

@WebServlet annotation

load-on startup --def value =-1

eg : @WebServlet(value={"/hi","/hello"},loadOnStartup=1)

public class Servlet1 extends H.S {...}

@WebServlet("/test")

public class Servlet2 extends H.S {...}

use case of loadOnStartup

--in case of time consuming init methods(eg : DAO inst DispatcherServlet of Spring MVC)

How to read request params sent from the clnt ?

javax.servlet.ServletRequest i/f methods

1. public String getParameter(String paramName)

2. public String[] getParameterValues(String paramName)

Servlet--JDBC integration

Layers involved

Servlet --DAO(DBUtils) -- POJO -- DB

LoginServlet

D.M -- dao

init --- DAO inst

destroy --- Dao's clean up

doGet

1. set cont type

2. pw

3. read rq params

4. invoke Dao's --validate

null --- Invalid login --retry link

not null ---mesg , display customer dtls

Objective

Complete login--logout flow

(login.html -- LoginServlet --successful login --redirect to DetailsServlet --logout link --LogOutServlet

How to redirect client ?

Page navigation --client pull

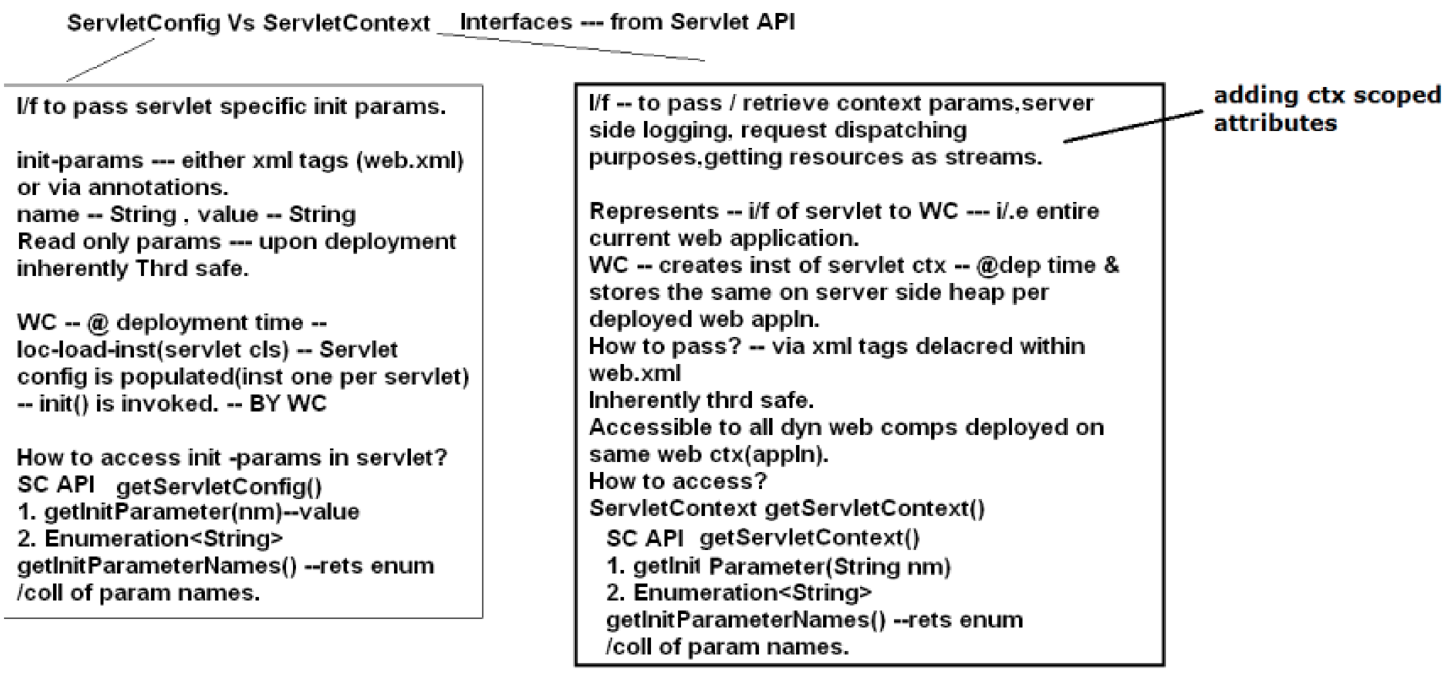
Refer to page navigation techniques.

What will happen if u explicitely flush/close PrintWriter & then invoke sendRedirect ?

WC throws ---java.lang.IllegalStateException --

reason --Can't redirect the clnt (empty the buffer's contents) after committing the response.

*** IMPORTANT** --- copy context.xml from "j2ee_help\day1_help\config files" in your <tomcat>/conf


ServletConfig Vs ServletContext Interfaces --- from Servlet API

I/f to pass servlet specific init params.

init-params --- either xml tags (web.xml) or via annotations.
name -- String , value -- String
Read only params --- upon deployment inherently Thrd safe.

WC -- @ deployment time -- loc-load-inst(servlet cls) -- Servlet config is populated(inst one per servlet) -- init() is invoked. -- BY WC

How to access init -params in servlet?
SC API getServletConfig()
1. getInitParameter(nm)--value
2. Enumeration<String> getInitParameterNames() --rets enum /coll of param names.

I/f -- to pass / retrieve context params,server side logging, request dispatching purposes,getting resources as streams.

adding ctx scoped attributes

Represents -- i/f of servlet to WC --- i/.e entire current web application.
WC -- creates inst of servlet ctx -- @dep time & stores the same on server side heap per deployed web appln.
How to pass? -- via xml tags delacred within web.xml
Inherently thrd safe.
Accessible to all dyn web comps deployed on same web ctx(appln).
How to access?
ServletContext getServletContext()
SC API getServletContext()
1. getInit Parameter(String nm)
2. Enumeration<String> getInitParameterNames() --rets enum /coll of param names.

Regarding SERVLET CONFIG

A servlet specific configuration object used by a servlet container to pass information to a servlet during initialization.

1. Represents Servlet specific configuration.

Defined in javax.servlet.ServletConfig -- interface.

2. Who creates its instance ?

Web container(WC)

3. When ?

After WC creates servlet instance, ServletConfig instance is created & then it invokes init() method of the servlet.

4. Usage

To store servlet specific init parameters.

(i.e the init-param is accessible to one servlet only or you can say that the init-param data is private for a particular servlet.)

5. Where to add servlet specific init parameters?

Can be added either in web.xml or @WebServlet annotation.

6. How to access servlet specific init params from a servlet ?

6.1 Override init() method

6.2 Get ServletConfig

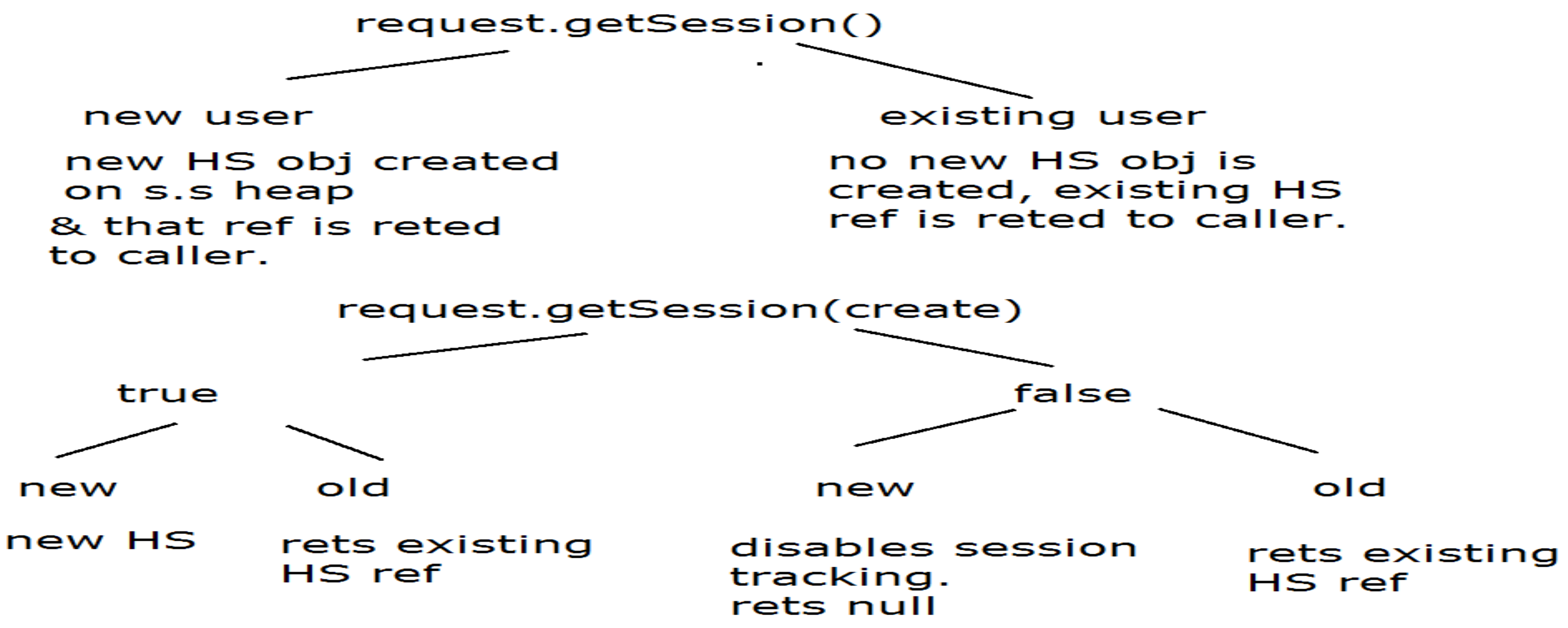
Method of Servlet i/f

public ServletConfig getServletConfig()

6.3 Get the init params from ServletConfig

Method of ServletConfig i/f

String getInitparameter(String paramName) : rets the param value.


request.getSession()
new user
new HS obj created on s.s heap & that ref is reted to caller.
existing user
no new HS obj is created, existing HS ref is reted to caller.
request.getSession(create)
true
new
new HS
old
rets existing HS ref
false
new
disables session tracking. rets null
old
rets existing HS ref

Regarding javax.servlet.ServletContext(i/f)

1. Defined in javax.servlet package.

2. Who creates its instance -- WC

3. When -- @ Web application (=context) deployment time

NOTE : The ServletContext object is contained within the ServletConfig object, which the WC provides the servlet when the servlet is initialized.

4. How many instances ? --one per web application

5. Usages

5.1 Server side logging

API public void log(String mesg)

5.2 To create context scoped attributes

API public void setAttribute(String nm,Object val)

NOTE : Access them always in thread safe manner

5.3 To access global(scope=entire web application) parameters

How to add context parameters ?

In web.xml

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

How to access ctx params in a Servlet ?

(can be accessed from init method onwards)

1. Get ServletContext

API of GenericServlet

ServletContext getServletContext() --method inherited from GenericServlet

2. ServletContext API

String getInitparameter(String paramName) : rets the param value.

What is a Servlet Listener(or web application listener)?

* During the lifetime of a typical web application, a number of events take place.

eg : requests are created or destroyed.

* sessions are created & destroyed * Contexts(web apps) are created & destroyed.

request or session or context attributes are added, removed, or modified etc.

* The Servlet API provides a number of listener interfaces that one can implement in order to react to these events.

eg : Event Listener i/f

1. ServletRequestListener **2.** HttpSessionListener **3.** ServletContextListener

Steps

1. Create a class , implementing from Listener i/f.

2. Register it with WC

2.1 @WebListener annotaio OR

2.2 XML tags in web.xml

```
<listener>
 <listener-class>Fully qualified listener  cls name </listener-class>
</listener>
```

sendRedirect Vs Request Dispatcher forward

common --both of above are page navigation techniques.

1. Navigates the user to next page in NEXT request	1. Navigates the user to next page in SAME request.
2.Client browser is involved (round trip delay involved)	2. Client browser is NOT involved (no round trip delay)
3. Min scope required for sharing attributes = session scope	3. Min scope required for sharing attributes =request scope
4. URL changes to the next(redirected) page.	4.URL seen by the client is that of 1st page(in the chain of resources)
5. Usage -- To navigate client outside current web application. Double submit guard.	5. Usage -- In MVC applications, to navigate the client from controller to the view layer. (Used as default navigation by all major MVC frmworks)

****What will happen if you invoke sendRedirect after committing the response(pw.flush/pw.close)?**

ANS-----WC throws java.lang.IllegalStateException

* Standard practise : Don't access PW or set content type in case of redirect.

* HttpServletRequest ---getSession vs getSession(create)

atributes --- server side objs (nm(string), value(Object)) --- can be added to different scopes. --- page|request|session|application (typically created by servlet or JSP programmer)

Invoker	Common Block API	significance
ServletRequest	void setAttribute(String nm,Object val) Object getAttribute(String nm) void removeAttribute(String nm) Enumeration<String> getAtrributeNames()	scope=current request only, GCed after current resp is comitted.
HttpSession		scope=cuurent session;shared across all web pages from SAME web appln for the same clnt.
ServletContext		scope=entire web appln;shared across all web pages from SAME web appln for any clnt. inherently thrd un-safe MUST be accessed in thrd safe (synch block) manner

HttpSession Internals
Prog invokes --- HttpSession hs=request.getSession();

WC checks -- for old user (alrdy exising --participated in session tracking) or new user. Ⓐ

old user

rets alrdy existing HS obj ref to the caller.

New User

1. Empty HttpSession Object is created on heap.
eg -- HttpSession hs1=new HttpSession();
2. WC creates 1 cookie.
Cookie c1=new Cookie("JSESSIONID","abc123456"); --- value is generated uniquely per clnt.
3. WC adds cookie to resp hdr
resp.addCookie(c1);
4. Rets hs1 to the caller(P)

Prog invokes --- hs.setAttribute("user_info",ref);

WC simply adds entry(key -- attr name ---user_info, value --attr value --user pojo ref. ---clnt's state. --- into HttpSession object(hm.put(k,v))

Ⓐ

How does WC know ---old user or new user.
WC invokes
Cookie[] cookies=req.getCookies();

null ---NEW

WC invokes
```
for(Cookie c : cookies)
  if (c.getName().equals("JSESSIONID")) ---
    c.getValue() --- "abc123456"  --
```
From this value --- WC retrieves HS object associated with cookie value --- HS1

OLD user -- alrdy visited clnt.

JSP-Java Servlet Pages

What is JSP? (Java server pages)

Dynamic Web page (having typically HTML markup) , can embed Java code directly.

Dynamic web component , whose life-cycle is managed by WC(JSP container/Servlet container/Servlet engine)

WHY JSP?

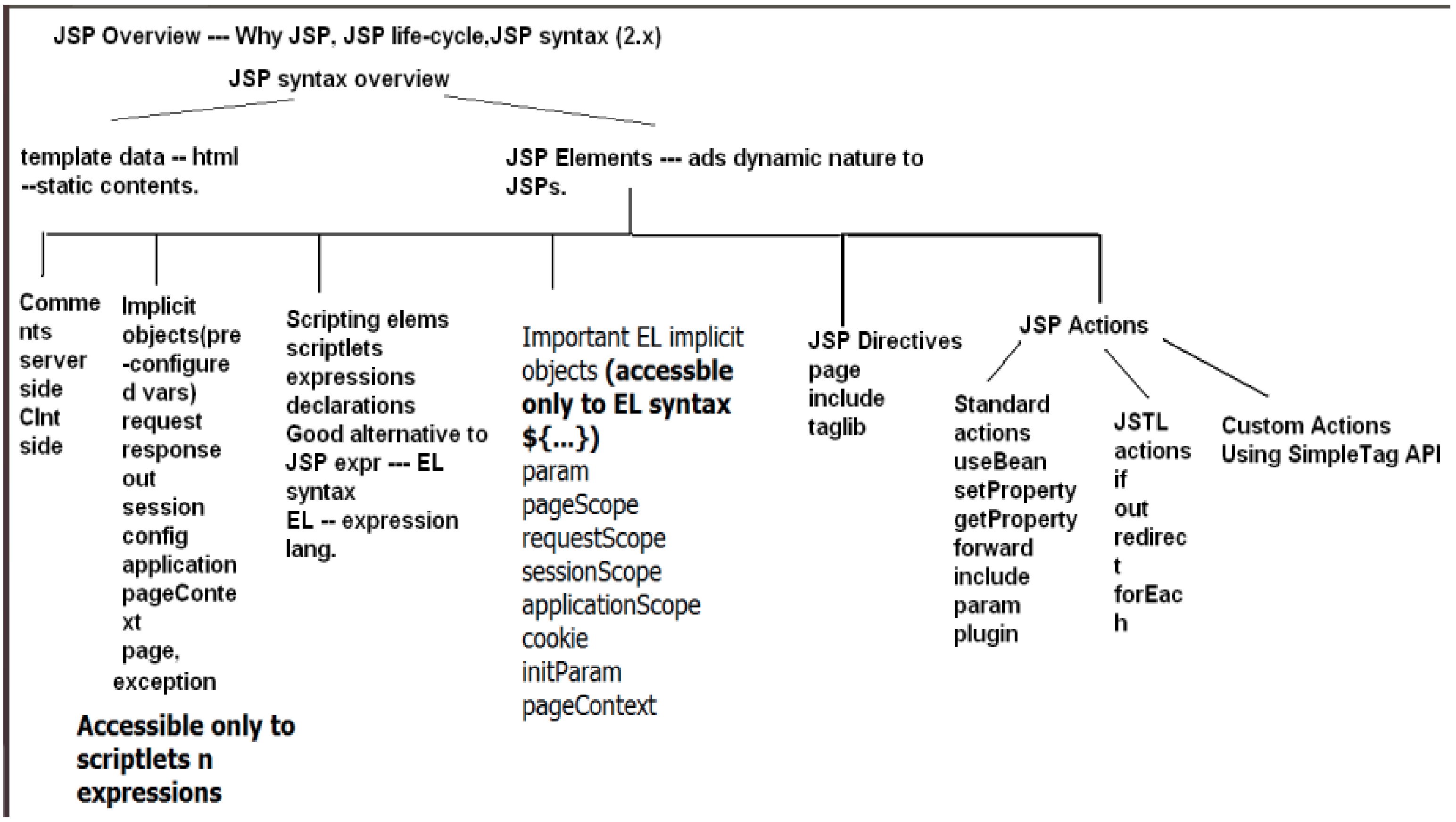
1. JSP allows developer to separate presentation logic(dyn resp generation) from Business logic or data manipulation logic.

Typically JSPs -- used for P.L(presentation logic)

Java Beans or Custom Tags(actions) --- will contain Business logic.

2. Ease of development --- JSP pages are auto. translated by W.C in to servlet & compiled & deployed.

3. Can use web design tools -- for faster development (RAD --rapid application development) tools.


JSP Overview --- Why JSP, JSP life-cycle,JSP syntax (2.x)
JSP syntax overview
template data -- html --static contents.
JSP Elements --- ads dynamic nature to JSPs.
Comments server side Clnt side
Implicit objects(pre-configured vars) request response out session config application pageContext page, exception
Accessible only to scriptlets n expressions
Scripting elems scriptlets expressions declarations Good alternative to JSP expr --- EL syntax EL -- expression lang.
Important EL implicit objects (accessble only to EL syntax ${...}) param pageScope requestScope sessionScope applicationScope cookie initParam pageContext
JSP Directives page include taglib
JSP Actions
Standard actions useBean setProperty getProperty forward include param plugin
JSTL actions if out redirect forEach
Custom Actions Using SimpleTag API

JSP API

jsp-api.jar --- <tomcat>/lib

Contains JSP API implementation classses.

0. javax.servlet.Servlet -- super i/f

1. javax.servlet.jsp.JspPage -- extends Servlet i/f

1.1 public void jspInit()

1.2 public void jspDestroy()

Can be overridden by JSP page author

2. Further extended by javax.servlet.jsp.HttpJspPage

2.1 public void _jspService(HttpServletRequest rq,HttpServletResponse rs) throws ServletExc,IOExc.

Never override _jspService ---JSP container auto translates JSP tags (body) into _jspService.

JSP life-cycle

1. Clnt sends the 1st request to the JSP (test.jsp)

2. Web-container invokes the life cycle for JSP

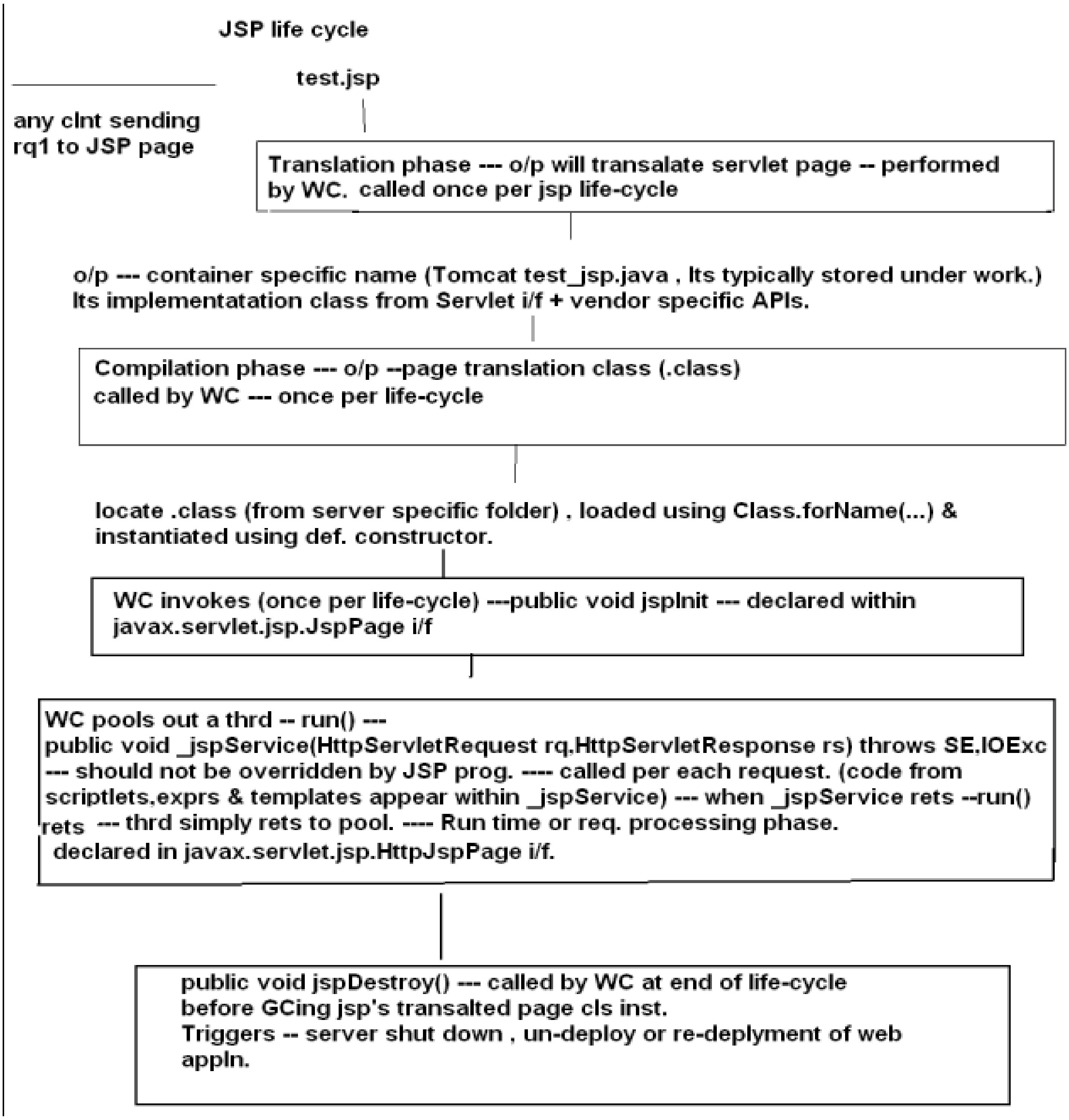
3. Translation Phase : handled by the JSP container.

I/p : test.jsp O/p : test_jsp.java (name : specific to the Tomcat container)

Meaning : .jsp is translated into corresponding servlet page(.java)

Translation time errs : syntactical errs in using JSP syntax.

In case of errs : life-cycle is aborted.



JSP life cycle
test.jsp
any clnt sending rq1 to JSP page
Translation phase --- o/p will transalate servlet page -- performed by WC. called once per jsp life-cycle
o/p --- container specific name (Tomcat test_jsp.java , Its typically stored under work.) Its implementatation class from Servlet i/f + vendor specific APIs.
Compilation phase --- o/p --page translation class (.class) called by WC --- once per life-cycle
locate .class (from server specific folder) , loaded using Class.forName(...) & instantiated using def. constructor.
WC invokes (once per life-cycle) ---public void jspInit --- declared within javax.servlet.jsp.JspPage i/f
WC pools out a thrd -- run() --- public void _jspService(HttpServletRequest rq,HttpServletResponse rs) throws SE,IOExc --- should not be overridden by JSP prog. ---- called per each request. (code from scriptlets,exprs & templates appear within _jspService) --- when _jspService rets --run() rets --- thrd simply rets to pool. ---- Run time or req. processing phase. declared in javax.servlet.jsp.HttpJspPage i/f.
public void jspDestroy() --- called by WC at end of life-cycle before GCing jsp's transalted page cls inst. Triggers -- server shut down , un-deploy or re-deplyment of web appln.

4. Compilation Phase : handled by the JSP container.

I/p : Translated servlet page(.java)   O/p : Page Translation class(.class)

Meaning : servlet page auto. compiled into .class file

Compilation time errs: syntacticle errs in generated Java syntax.

5. Request processing phase / Run time phase. : typically handled by the Servlet Container.

6. S.C : will try to locate,load,instantiate the generated servlet class.

7. The 1st it calls : public void jspInit() : one time inits can be performed.(jspInit availble from javax.servlet.jsp.JspPage)

8. Then it will call follwing method using thrd created per clnt request :

public void _jspService(HttpServlet Rq,HttpServletResponse) throws ServletException,IOException(API avlble from javax.servlet.jsp.HttpJspPage)

When _jspService rets , thread's run method is over & thrd rets to the pool, where it can be used for servicing some other or same clnt's req.

9.. At the end ...(server shutting down or re-deployment of the context) : the S.C calls

public void jspDestroy()

After this : translated servlet page class inst. will be GCEd....

10 For 2nd req onwards ...... : SC will invoke step 8 onwards.

--

JSP 2.0/2.1/2.2 syntax

1. JSP comments

1.1 server side comment

syntax : <%-- comment text --%>

significance : JSP translator & compiler ignores the commented text.

1.2 clnt side comment

syntax : <!-- comment text -->

significance : JSP translator & compiler does not ignore the commented text BUT clnt browser will ignore it.

2. JSP's implicit objects (available only to _jspService) -- avlable to scriptlets,exprs

2.1 out - javax.servlet.jsp.JspWriter : represents the buffered writer stream connected to the clnt via HttpServletResponse(similar to your PW in servlets)

Has the same API as PW(except printf)

usage eg : out.print("some text sent to clnt");

2.2 request : HttpServletRequest (same API)

2.3 response : HttpServletResponse

2.4 config : ServletConfig (used for passing init params)

2.4 session : HttpSession (By def. all JSPs participate in session tracking i.e session obj is created)

2.5 exception : java.lang.Throwable (available only to err handling pages)

2.6 pageContext : current page environment : javax.servlet.jsp.PageContext(this class stores references to page specific objects viz -- exception,out,config,session)

2.7 application : ServletContext(used for Request dispatching, server side logging, for creating context listeners,to avail context params, to add/get context scoped attrs)

2.8 page --- current translated page class instance created for 'this' JSP

3. Scripting elements : To include the java content within JSP : to make it dynamic.

3.1 Scriptlets : can add the java code directly . AVOID scriptlets . (till Javabeans & custom tags or JSTL, we will use use the scriptlets to add : Req. processing logic, B.L & P.L)

syntax : <% java code...... %>

location inside the translated page : within _jspService

usage : till JBs or cust. tags are introduced : scriptlets used for control flow/B.L/req. proc. logic

3.2 JSP expressions :

syntax : <%= expr to evaluate %>

--Evaluates an expression --converts it to string --send it to clnt browser.

eg : <%= new Date() %>

expr to evaluate : java method invocation which rets a value OR

const expr or attributes(getAttribute) or variables(instance vars or method local)

location inside the translated page : within _jspService

significance : the expr gets evaluated---> to string -> automatically sent to clnt browser.

    eg <%= new Date() %>
    eg <%= request.getAttribute("user_dtls") %>
    <%= 12*34*456 %>
    <%= session.getAttribute("user_dtls") %>
    <%= session.setAttribute("nm",val) %>
    <%= session.getId() %>

Better alternative to JSP Expressions : <u>EL syntax</u> (Expression Lang. : avlble from JSP 1.2 onwards)

syntax : ${expr to evaluate} (to be added directly in body tag)

EL syntax will evaluate the expr ---toString --sends it clnt browser.

JSP implicit object --- request,response,session....---accessible from scriptlets & JSP exprs. --- accessible to scriptlets/exprs

EL implicit objects --- can be accessible only via EL syntax

param = map of request parameters

pageScope=map of page scoped attrs

requestScope=map of request scoped attrs

sessionScope=map of session scoped attrs

applicationScope=map of application(=context) scoped attrs

pageContext --- instance of PageContext's sub class

---avlable ONLY to EL syntax ${...}

---to be added directly within <body> ...</body>

eg : ${param.user_nm} ---to string ---clnt browser

***request.getParameter("user_nm") ---to string --sent to clnt browser.

-*- ${requestScope.abc} ---request.getAttribute("abc") ---to string --sent to clnt browser.

eg : suppose ctx scoped attr --- loan_scheme

${applicationScope.loan_scheme} --- getServletContext().getAttirbute("loan_scheme") ---to string --sent to clnt

${user_dtls}   --- null -- blank
${abc} ---
pageContext.getAttribute("abc") ---not null -- to string -clnt
 null
--request.getAttribute("abc") -- not null -- to string -clnt
null
session.getAttribute("abc") ---
null
getServletContext().getAttirbute("abc") --not null -- to string -clnt
null ---BLANK to clnt browser.

eg : ${sessionScope.nm}
${pageContext.session.id}
--pageContext.getSession().getId() --- val of JessionId cookie w/o java code.
--${pageContext.request.contextPath} ---/day5_web
--${pageContext.session.maxInactiveInterval}

${param}
{user_nm=asdf, user_pass=123456}
eg : ${param.f1} ---> request.getParameter("f1").toString()---> sent to browser
param ----map of req parameters.
* param : req. param map
--${requestScope.abc} ----- out.print(request.getAttribute("abc").toString())
**${abc} -----pageCotext.getAttribute("abc")----null ---request ---session---application ---null ---EL prints blank.

3.3 JSP declarations (private members of the translated servlet class)
syntax : <%! JSP declaration block %>
Usage : 1. for creating page scoped java variables & methods (instance vars & methods/static members)
2. Also can be used for overriding life cycle methods (jspInit,jspDestroy)
** location inside the translated page : outside any of life-cycle meths & within the translated servlet class.

JSP Directives --- commands/messages for JSP Engine(=JSP container=WC) -- to be used @Translation time.
Syntax ---
<%@ Directive name attrList %>
1. page directive
--- all commands applicable to current page only.
Syntax
<%@ page import="comma separated list of pkgs" contentType="text/html" %>

eg -- <%@ page import="java.util.*,java.text.SimpleDateFormat" contentType="text/html" %>

Imp page directive attributes

1. import --- comma separated list of pkgs

2. session --- boolean attribute. def=true.

To disable session tracking, spectify session="false"

3. errorPage="URI of err handling page" ---

tells WC to forward user to err handler page.

4. isErrorPage="true|false" def = false

If u enable this to true--- one can access 'exception' implicit object from this page.

--This exception obj is stored under current page ---i.e under pageContext (type=javax.servlet.jsp.PageContext -- class which represents curnt JSP)

EL expresssion to display error mesg

${pageContext.exception.message}

-- evals to pageContext.getException().getMessage()

Additional EL syntax

EL syntax to be used in error handling pages

ERR causing URI : ${pageContext.errorData.requestURI }<br/>

ERR code : ${pageContext.errorData.statusCode}<br/>

ERR Mesg : ${pageContext.exception.message} <br/>

Throwable : ${pageContext.errorData.throwable}<br/>

Throwable Root cause: ${pageContext.errorData.throwable.cause}

5. isThreadSafe="true|false" default=true. "true" is recommended

true=>informing WC--- JSP is already written in thrd -safe manner ---- DONT apply thrd safety.

false=>informing WC --- apply thrd safety.

(NOT recommended) ---WC typically marks entire service(servlet scenario) or _jspService in JSP scenarion --- synchronized. --- this removes concurrent handling of multiple client request --so not recommended.

What is reco? --- isThreadSafe=true(def.) --- identify critical code--wrap it in synchronized block.

eg ---Context scoped attrs are inherently thrd -un safe. So access them always from within synched block.

Equivalent step in Servlet

Servlet class can imple. tag i/f -- javax.servlet.SingleThreadModel(DEPRECATED) -- WC ensures only 1thread (representing clnt request) can invoke service method. --NOT NOT recommended.

2. include directive

<%@ include file="URI of the page to be included" %>

Via include directive ---- contents are included @ Translation time.--- indicates page scope(continuation of the same page).

Typically used -- for including static content (can be used to include dyn conts)

eg ---one.jsp

<%@ include file="two.jsp" %>

two.jsp.....

JSP actions ---- commands/mesgs meant for WC

to be interpreted @ translation time & applied @ req. processing time.(run time)

Syntax ---standard actions --implementation classes are present in jsp-api.jar.

<jsp:actionName attr list>Body of the tag/action

</jsp:actionName>

JSP Using Java beans

Why Java Beans?

1. allows prog to seperate B.L in JBs.(Req processing logic, Page navigation & resp generation will be still part of JSP)

JBs can store conversational state of clnt(JB 's properties will reflect clnt state) + supplies Business logic methods.

2. simple sharing of JBS across multiple web pages---gives rise to re-usability.

3. Auto. translation between req. params & JB props(string--->primitive data types auto. done by WC)

What is JB?

1. pkged public Java class

2. Must have def constr.(MUST in JSP using JB scenario)

3. Properties of JBs --- private, non-static , non-transient Data members --- equivalent to request params sent by clnt.(Prop names MUST match with req params for easy usage)

In proper words --- Java bean props reflect the conversational state of the clnt.

4. per property -- if RW

naming conventions of JB

supply getter & setter.

Rules for setter

public void setPropertyName(Type val)

Type -- prop type.

eg -- private double regAmount;

public void setRegAmount(double val)

{...}

Rules for getter

public Type getPropertyName()

Type -- prop type.

eg -- public double getRegAmount(){...}

5. Business Logic --- methods

public methods --- no other restrictions

Using Java Beans from JSP Via standard actions

1. <jsp:useBean id="BeanRef name" class="F.Q. Bean class name" scope="page|request|session|application/>

def = page scope.

pre-requisite --- JB class exists under <WEB-INF>/classes.

JB = server side obj (attribute), attr name --- bean id,attr val -- bean inst.,can be added to any scope using scope atribute.

eg :

eg --- beans.Userbean

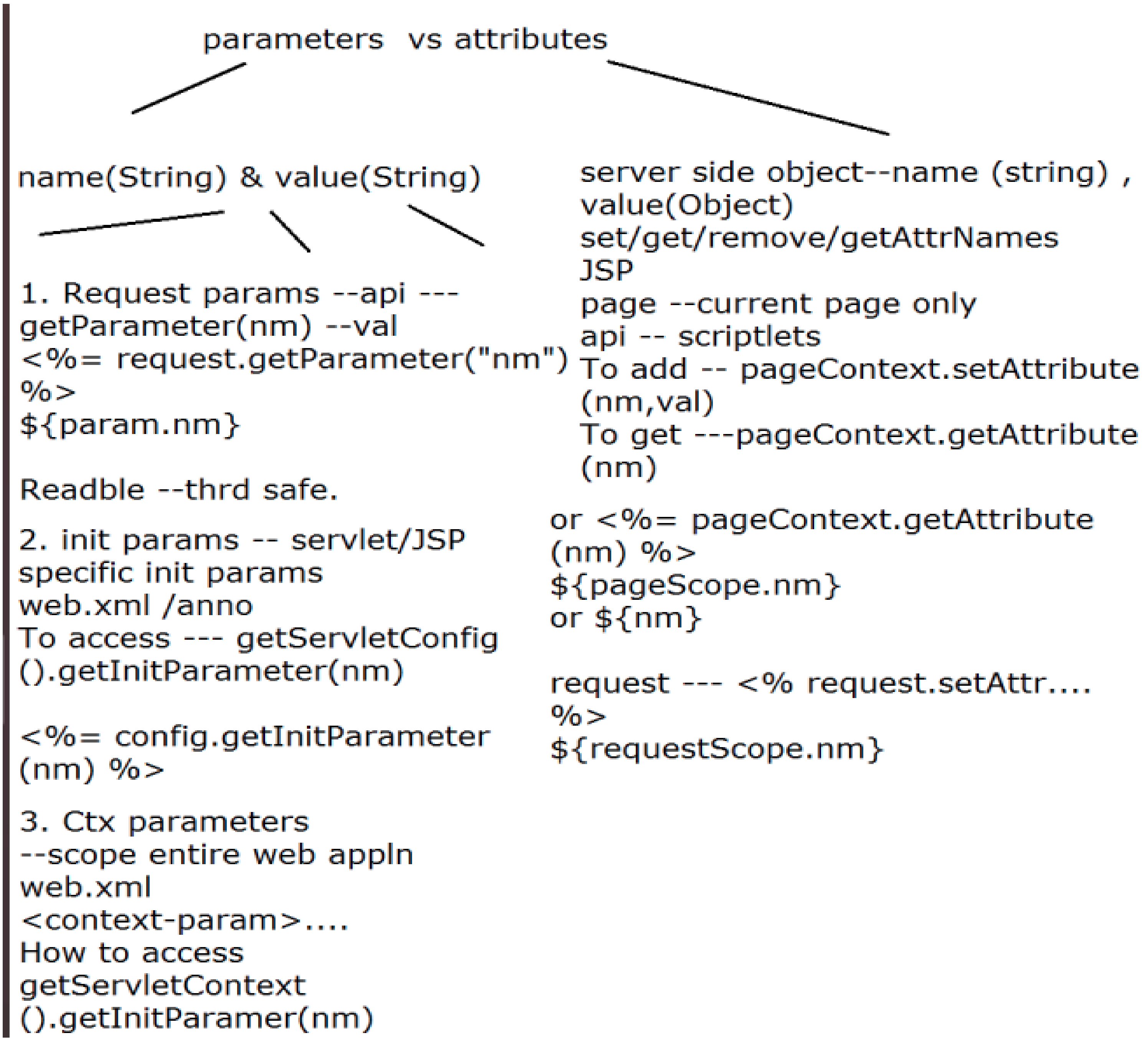
props --- email,pass

setters/getters

B.L mehod -- for validation

Usage ---

<jsp:useBean id="user" class="beans.UserBean" scope="session"/>



parameters vs attributes

name(String) & value(String)

1. Request params --api ---
getParameter(nm) --val
<%= request.getParameter("nm")
%>
${param.nm}

Readble --thrd safe.

2. init params -- servlet/JSP
specific init params
web.xml /anno
To access --- getServletConfig
().getInitParameter(nm)

<%= config.getInitParameter
(nm) %>

3. Ctx parameters
--scope entire web appln
web.xml
<context-param>....
How to access
getServletContext
().getInitParamer(nm)

server side object--name (string) ,
value(Object)
set/get/remove/getAttrNames
JSP
page --current page only
api -- scriptlets
To add -- pageContext.setAttribute
(nm,val)
To get ---pageContext.getAttribute
(nm)

or <%= pageContext.getAttribute
(nm) %>
${pageScope.nm}
or ${nm}

request --- <% request.setAttr....
%>
${requestScope.nm}

W.C invokes JB life-cycle

1. WC chks if specified Bean inst alrdy exists in specified scope

java api --- request.getAttribute("user")

---null=>JB doesn't exist

---loc/load/inst JB class

UserBean u1=new UserBean();

--add JB inst to the specified scope

java api -- request.setAttribute("user",u1);

--- not-null -- WC continues....

2. JSP using JB action

2.1 <jsp:setProperty name="Bean ref Name" property="propName" value="propVal---static/dyn" />

Usage--

<jsp:setProperty name="user" property="email"

value="a@b"/>

WC invokes --- session.getAttribute("user").setEmail("a@b");

<jsp:setProperty name="user" property="email"

value="<%= request.getParameter("f1") %>"/>

OR via EL

<jsp:setProperty name="user" property="email"

value="${param.f1}"/>

**WC invokes ---

session.getAttribute("user").setEmail(request.getParameter("f1"));

2.2 <jsp:setProperty name="Bean ref Name" property="propName" param="rq. param name"/>

Usage eg -

<jsp:setProperty name="user" property="email" param="f1"/>

WC invokes --- ((Userbean)request.getAttribute("user")).setEmail(request.getParameter("f1"));

2.3 <jsp:setProperty name="Bean ref Name" property="*"/>

usage

<jsp:setProperty name="user" property="*"/>

eg -- If rq. param names are email & password(i.e matching with JB prop names) then ---matching setters(2) will get called

3. <jsp:getProperty name="Bean ref name" property="propName"/>

Usage -- <jsp:getProperty name="user" property="email"/>

WC --- session.getAttribute("user").getEmail()--- toString --- sent to clnt browser.

Better equivalent -- EL syntax

* ${sessionScope.user.email} ---

* session.getAttribute("user").getEmail()--- toString --- sent to clnt browser.

* ${requestScope.user.validUser.email}

* request.getAttribute("user").getValidUser().getEmail()

* ${pageContext.exception.message}

4.JSP std actions related to Request Dispatcher

RD's forward scenario

<jsp:forward page="dispatcher URI" />

eg : In one.jsp

<jsp:forward page="two.jsp"/>

WC invokes ---RD rd=reuqest.getRD("two.jsp");

rd.forward(request,response);

RD's include scenario

    <jsp:include page="dispatcher URI" />

Why JSTL ? JSP standard tag library

* When JSP std actions are in-sufficient to solve B.L

* w/o writing scriptlets --- use additional std actions --- supplied as JSTL actions

** JSP standard Tag Library

--- has become std part of J2EE 1.5 onwards.

---support exists in form JAR ---

1. jstl-1.2.jar

For using JSTL steps

1.Copy above JAR into ur run-time classpath(copy jars either in <tomcat_home>/lib OR <web-inf>/lib

2. Use taglib directive to include JSTL tag library into ur JSP pages.

    tag=action
    tag library=collection of tags

supplier = JSTL vendor(spec vendor=Sun, JAR vendor=Sun/any J2EE compliant web/app server)

jstl.jar --- consist of Tag implementation classes

Tag libr-   TLD -- Tag library descriptor -- desc of tags -- how to use tags

    <%@ taglib uri="URI of JSTL tag lib" prefix="tag prefix" %>

3. Invoke JSTL tag/action

3.1 eg

    <c:set var="abc" value="${param.f1}" />

---WC

pageContext.setAttribute("abc",request.getParameter("f1"))

WC invokes --- session.setAttribute("abc",request.getparameter("f1"));

    menaing of <c:set> sets the specified attr to specified scope.
    <c:set var="details" value="${sessionScope.abc}" />

WC**-----** pageContext.setAttribute("details",session.getAttribute("abc"));

2. <c:remove var="abc" scope="request"/>

WC ---request.removeAttribute("abc") ---removes the attr from req scope.

3.2 JB --- ShopBean -- property --

    private AL<Category> categories; --g & s
    <c:forEach var="cat" items="${sessionScope.shop.listCategories()}">
    ${cat}<br/>
    </c:forEach>

WC invokes ---

```
for(Category cat : session.getAttribute("shop").listCategories())
  out.print(cat);
```

eg : <c:forEach var="acct" items="${sessionScope.my_bank.acctSummary}">
${acct.acctID} ${acct.type} ${acct.balance} <br/>
</c:forEach>

****** http://localhost:8080/day6_web/close_acct.jsp?acId=101

```
<input type="submit" name="btn" value="Withdraw"
                                    formaction="transactions.jsp" /></td>
                              <td><input type="submit" name="btn" value="Deposit"
                                    formaction="transactions.jsp" /></td>
<%
  request.getPrameter("btn").equals("Deposit") ---
%>
<c:if test="boolean val">
```

..

```
</c:if>
<c:if test="${param.btn eq 'Deposit'}">
 in deposit
</c:if>
<c:if test="${param.btn eq 'Withdraw'}">
 in withdraw
</c:if>
```

http://localhost:8080/day6_web/transactions.jsp?acId=102&amount=500&btn=Deposit

<c:redirect url="${sessionScope.my_bank.closeAccount()}"/>

WC --- response.sendRedirect(session.getAttribute("my_bank").closeAccount());

3.3 JSTL action --- for URL rewriting

<c:url var="attr Name" value="URL to be encoded" scope="page|request|session|application"/>

eg : <c:url var="abc" value="next.jsp" />

WC invokes --- pageContext.setAttribute("abc",resp.encodeURL("next.jsp"));

<a href="${abc}">Next</a>

var -- loop var

items -- any JB 's prop --- array based,coll based (List or set) map based

How to set session tm out ?

1. programmatically --- using Java API

From HttpSession --- setMaxInactiveInterval(int secs)

2. declarativally -- either using Java annotations OR using XML config files (web.xml)

Note : when u dont specify form action , its submitted to the same page.

Expression Language implicit variables(case sensitive)

1. pageContext : PageContext object (javax.servlet.jsp.PageContext) asso. with current page.

2. pageScope - a Map that contains page-scoped attribute names and their values.

3. requestScope - a Map that contains request-scoped attribute names and their values.

4. sessionScope - a Map that contains session-scoped attribute names and their values.

5. applicationScope - a Map that contains application-scoped attribute names and their values.

6. param - a Map that contains rq. parameter names to a single String parameter value (obtained by calling ServletRequest.getParameter(String name)).

7. paramValues - a Map that contains rq. param name to a String[] of all values for that parameter (similar to calling ServletRequest.getParameterValues(name)

8. initParam - a Map that contains context initialization parameter names and their String value (obtained by calling ServletContext.getInitParameter(String name)).

eg : ${initParam.db_drvr}

9. cookie : Map.Entry of cookies. (entrySet of cookies)

eg : ${cookie.cookiename.value}

key ---cookie name

value ---javax.servlet.http.Cookie

${cookie.JSESSIONID.value}

---cookie.get("JSESSIOIND").getValue()

10. To retrieve err details from Error handling page.

ERR causing URI : ${pageContext.errorData.requestURI }

ERR code : ${pageContext.errorData.statusCode}

ERR Mesg : ${pageContext.exception.message }

Throwable : ${pageContext.errorData.throwable}

Throwable Root cause: ${pageContext.errorData.throwable.cause}

eg :

<c:set var="abc" scope="session" value="Hello User...."/>

${sessionScope.abc}

Why Java Bean ?

JSP standard actions to manage java bean

1. <jsp:useBean id="bean id" class="F.Q bean class name" scope="page|request|session|application"/>

def scope=page (i.e by default , bean instance will be added under page scope.)

eg : <jsp:useBean id="cust" class="beans.CustomerBean" scope="session"/>

What will WC invoke?

eg : refer to diag.

2. <jsp:setProperty name="bean id" property="*"/>

eg : <jsp:setProperty name="cust" property="*"/>

What will WC invoke?

WC tries call ALL MACHING setters.

Request param names MUST MATCH with JB porperty setters

eg : rq param names --regAmt , name

3. <jsp:getProperty name="cust" property="details"/>

WC -- session.getAttribute("cust").getDetails() -- to string --sent to clnt.

OR EL syntax

${sessionScope.cust.details}

4. How to invoke B.L method of JB , w/o java code from JSP? ---EL syntax

${sessionScope.cust.authenticateCustomer()}

WC ---session.getAttribute("cust").authenticateCustomer()

<jsp:forward page="${sessinScope.cust.authenticateCustomer()}.jsp"/>

WC --- RD rd=request.getRD(session.getAttribute("cust").

authenticateCustomer().concat(".jsp"));

rd.forward(request,response);

What is Hibernate ?

0. Complete solution to the problem of managing persistence in Java.

1. ORM tool.(Object Relational Mapping) used mainly in data access layer or DAO layer.

2. Provides automatic & transperent persistence.

3. JPA(Java Persistence API) implementor

JPA vs Hibernate

JPA ---part of J2EE specification --vendor --J2EE (sun)

Implementation classes -- JAR ---hibenrate core JARs(implementor of JPA)

* Provides automatic & transparent persistence framework to store & retrieve data from database.

* Open Source Java based framework founded by Gavin King in 2001, hosted on hibernate.org

* Currently hosted on sourceforge.net

Java Persistence API (JPA) compliant

Current version Hibernate 5.x

Other popular ORM Frameworks

EclipseLink,iBATIS,Kodo etc.

WHY Hibernate?

* It mediates the applications interaction with a relational database, leaving the developer free to concentrate on the business problem at hand.

* J2EE developer does not have to use JDBC API & manage data persistence at RDBMS level.

* No need to go to Table/Query/Column level.

* One has to bootstrap Hibernate framework , create transient(=not yet persistent) POJOs & then rely entirely on Hibernate frmwork to manage persistence

There is huge mismatch between Object & Relational world.

Formally referred as -- Object-Relational Impedance Mismatch' (sometimes called the 'paradigm mismatch)

Important Mismatch Points

1. Granularity **2.** Sub Types or inheritance n polymorphism

3. Identity **4.** Associations **5.** Data Navigation

Cost of Mismatch

1.SQL queries in Java code **2.**Iterating through ResultSet & mapping it to POJOs or entities.

3.SQL Exception handling. **4.** Transaction management **5.** Caching

6. Connection pooling **7.** Boiler plate code

Hibernate Frmwork --- popular ORM Tool ---JPA (Java perssitence API) provider

***** Hibernate 4.x --- JPA compliant --- Java persistence API --- Its part of J2EE specifications. ---Is fully JPA compliant

***** BUT it also has additional services / annotations --- specific to Hibernate.

***** Dev MUST add hibernate JARs ---while deploying appln on web server. Need not add JPA provider JARs , while working on appln server.

***** Transparent persistence provider.(As POJOs or Entities are not bound to any Persistence API --- its written completely independent of Persistence Provider.)

--Fully supports OOP features --- association,inheritance & polymorphism

--can persist object graphs , consisting of asso. objects

--caches data which is fetched repeatedly (via L1 & L2 cache) -- thus reduces DB traffic(L1 cache - at session level -- built in. L2 cache - pluggable) (More on caching at end of document)

--supports lazy loading -- thus increases DB performance.

(**Meaning** --- Lazy fetchingThe associated object or collection is fetched lazily, when its first accessed. This results in a new request to the database (unless the associated object is cached). Eager fetchingThe associated object or collection is fetched together with the owning object, using an SQL outer join, and no further database request is required.

--supports Objectified version of SQL -- HQL --works on objects & properties

--Hibernate usually obtains exactly the right lock level automatically . so developer need not worry about applying Read/Write lock.

Some basics

1. Hibernate uses runtime reflection to determine the persistent properties of a class.

2. The objects to be persisted(called as POJO or Entity) are defined in a mapping document or marked with annotations.

Either these HBM XML docs or annotations serves to describe the persistent fields and associations, as well as any subclasses or proxies of the persistent object.

3. The mapping documents or annotations are compiled at application startup time and provide the framework with necessary information for a persistent class.

4. What is Hibernate config.?

* An instance of Hib Configuration allows the application to specify properties and mapping documents to be used at the frmwork start-up.

The Configuration : initialization-time object.

5. SessionFactory is created from the compiled collection of mapping documents .

The SessionFactory provides the mechanism for managing persistent classes, the Session interface.

6. A web application or Java SE apllication will create a single Configuration, build a single instance of SessionFactory and then instantiate multiple Sessions in threads servicing client requests.

SessionFactory : immutable and does not reflect any changes done later to the Configuration.

7. The Session class provides the interface between the persistent data store and the application.

The Session interface wraps a JDBC connection, which can be user-managed or controlled by Hibernate.

Hibernate Session

A Hibernate Session is a set of managed entity instances that exist in a particular data store.

*****Managing an Entity Instances Life Cycle**

You manage entity instances(or POJOs) by invoking operations on the entity/POJO using EntityManager/Session instance.

Entity instances are in one of four states (2 imp aspects of it : its asso. with the hibernate session & sync of its state with the underlying DB)

States : new or transient , managed or persistent, detached, removed.

* New entity instances have no persistent identity and are not yet associated with a hib. session (transient)

* Managed entity instances have a persistent identity and are associated with a hib. session.(persistent : via save() or saveOrUpdate()) Changes to DB will be done when tx is commited.

* Detached entity instances have a persistent identity and are not currently associated with a persistence context/Hib session.

* Removed entity instances have a persistent identity, are associated with a persistent context and are scheduled for removal from the data store.(removed via session.delete(obj))

Introduction to Hibernate Caching

* While working with Hibernate web applications we will face so many problems in its performance due to database traffic. That too when the database traffic is very heavy . Actually hibernate is well used just because of its high performance only. So some techniques are necessary to maintain its performance.

* Caching is the best technique to solve this problem.

* The performance of Hibernate web applications is improved using caching by optimizing the database applications.

* The cache actually stores the data already loaded from the database, so that the traffic between our application and the database will be reduced when the application want to access that data again.

* At maximum the application will work with the data in the cache only. Whenever some another data is needed, the database will be accessed. Because the time needed to access the database is more when compared with the time needed to access the cache. So obviously the access time and traffic will be reduced between the application and the database.

* Here the cache stores only the data related to current running application. In order to do that, the cache must be cleared time to time whenever the applications are changing.

Difference in get & load

1. Both use common API (i.e load or get(Class c,Serializable id))

Ret type = T

In get --- if id doesn't exist --- rets null

In load --- if id doesn't exist & u are accessing it from within hib session --- throws ObjectNotFoundExc

2. In get --- Hibernate uses eager fetching policy ---- meaning will generate select query always & load the state from DB in persistent POJO ref. --- so even if u access the same from within the session(persistent pojo) or outside (detached) the hib session --- NO EXCEPTION(proxy + state)

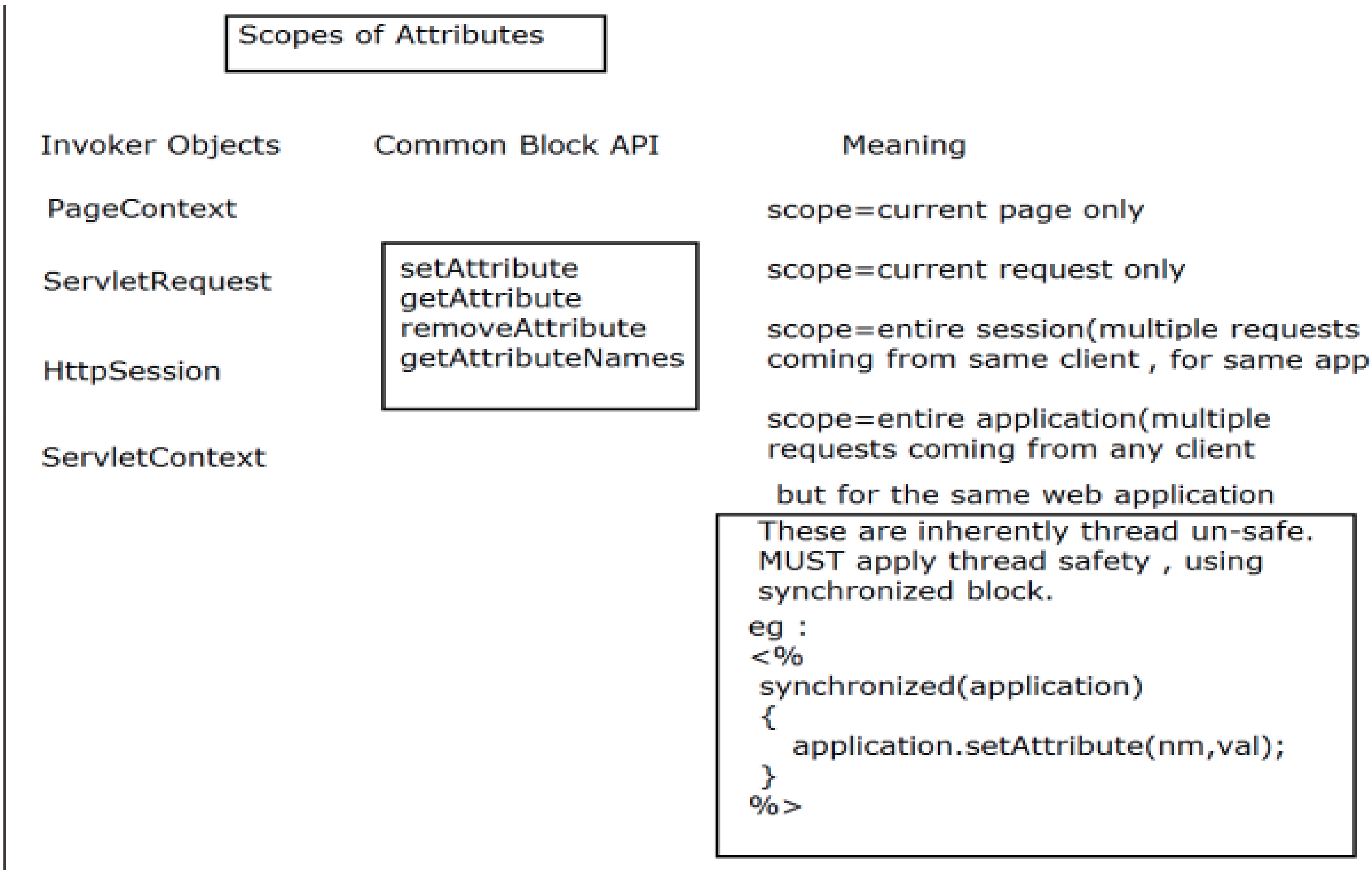
3. In load --- Hib uses lazy fetching policy ---- meaning it will , by default NOT generate any select query --- so what u have is ONLY PROXY(wrapper ---with no state loaded from DB) --- on such a proxy --- if u access anything outside the hib session(detached) ----

U WILL GET ---LazyInitializationExc

Fix --- 1. Change fetch type --- to eager (NOT AT ALL reco.=> no caching , disabling L1 cache)

2. If u want to access any POJO in detached manner(i.e outside hib session scope) -

fire non-id get method from within session & then hib has to load entire state from DB ---NO LazyInitializationExc



Scopes of Attributes

Invoker Objects	Common Block API	Meaning
PageContext		scope=current page only
ServletRequest	setAttribute getAttribute removeAttribute getAttributeNames	scope=current request only
HttpSession		scope=entire session(multiple requests coming from same client , for same app
ServletContext		scope=entire application(multiple requests coming from any client but for the same web application

These are inherently thread un-safe. MUST apply thread safety , using synchronized block.

```
eg :
<%
 synchronized(application)
 {
   application.setAttribute(nm,val);
 }
%>
```

Session API update Vs merge

* Both methods transition detached object to persistent state.

Update():- if you are sure that the session does not contain an already persistent instance with the same identifier then use update to save the data in hibernate. If session has such object with same id , then it throws --- org.hibernate.NonUniqueObjectException: a different object with the same identifier value was already associated with the session:

Merge():-if you want to save your modificatiions at any time with out knowing about the state of an session then use merge() in hibernate.

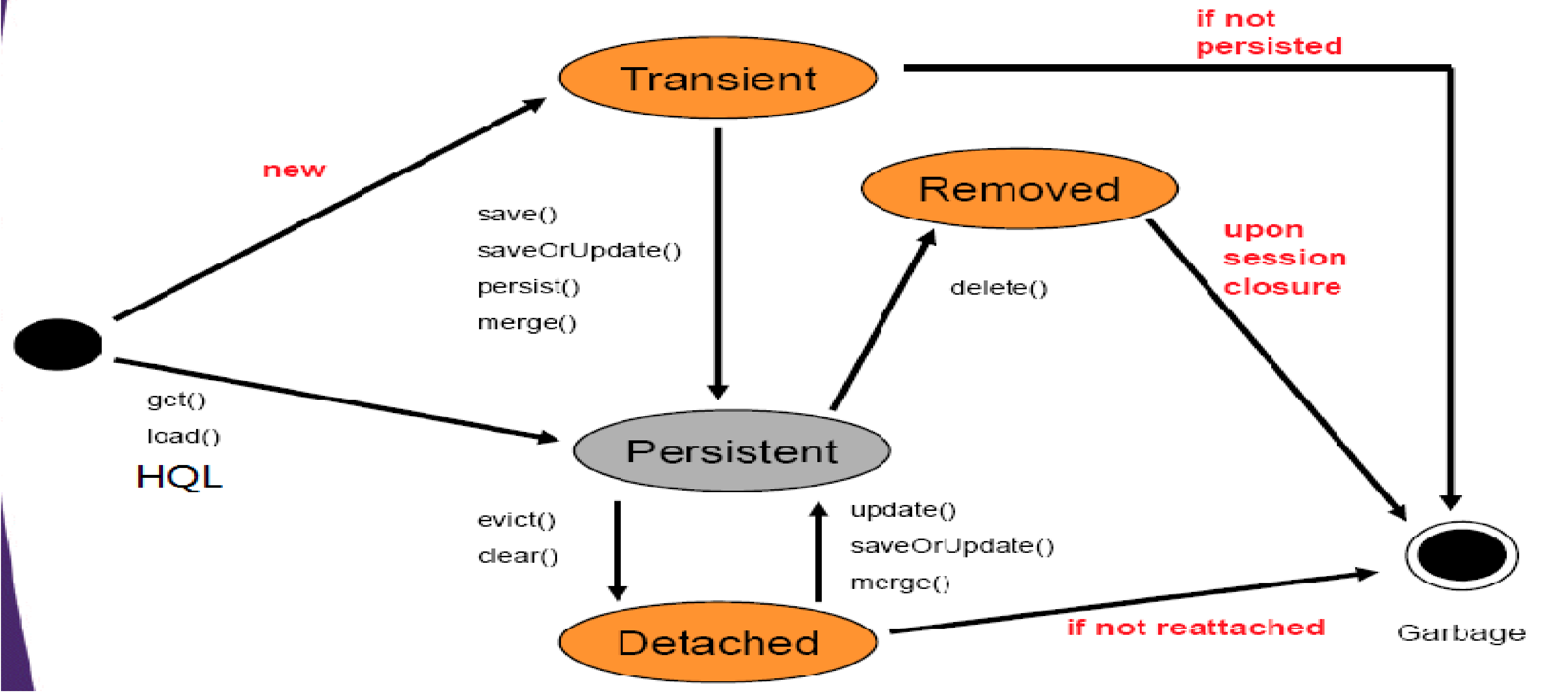
***** Lazy fetching (becomes important in relationships or in Load Vs get)

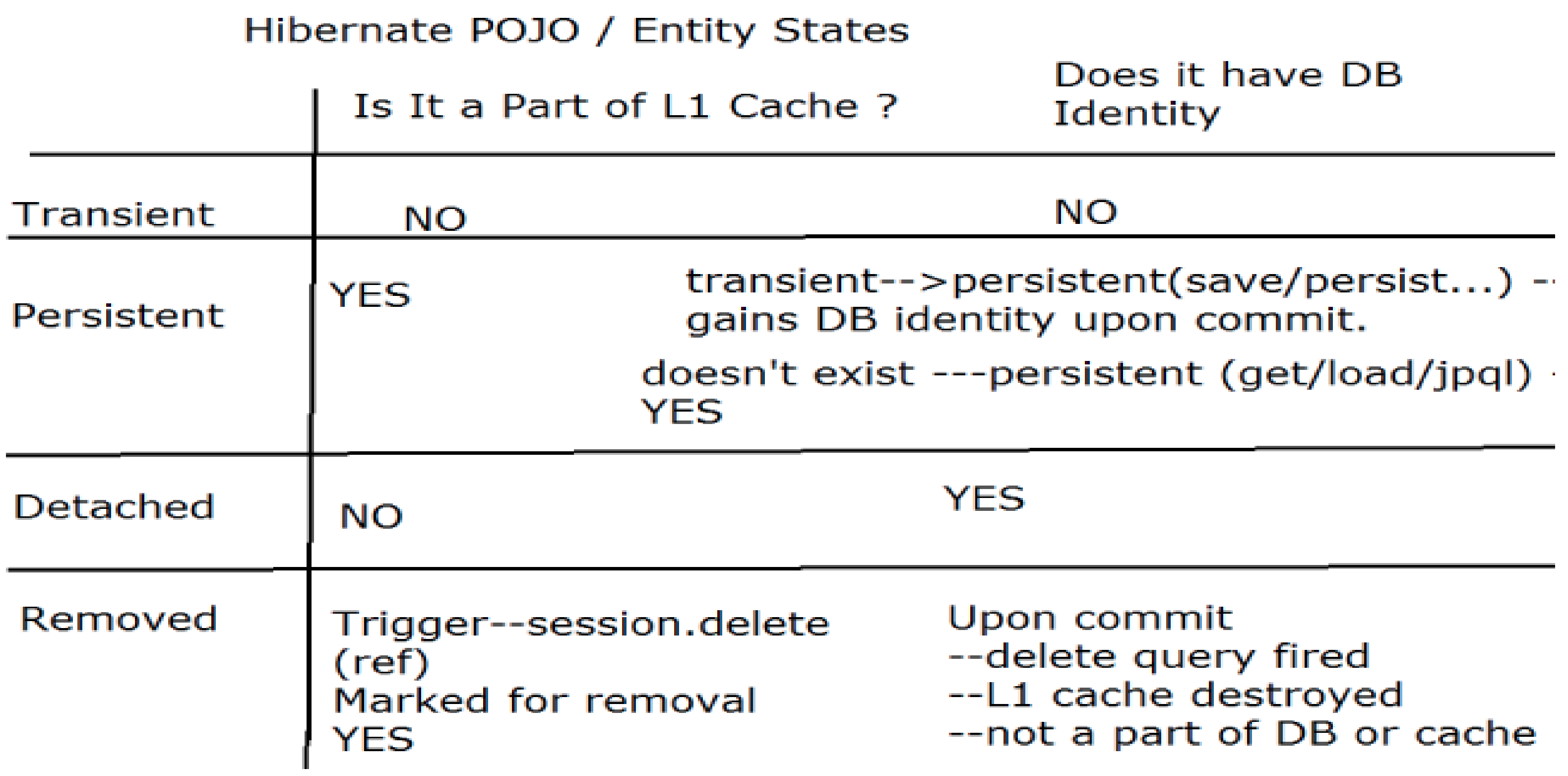
***** When a client requests an entity(eg - Course POJO) and its associated graph of objects(eg -Student POJO) from the database, it isnt usually necessary to retrieve the whole graph of every (indirectly) associated

object. You wouldnt want to load the whole database into memory at once;

eg: loading a single Category shouldnt trigger the loading of all Items in that category(one-->many)

Hibernate Lifecycle





Hibernate POJO / Entity States

	Is It a Part of L1 Cache ?	Does it have DB Identity
Transient	NO	NO
Persistent	YES	transient-->persistent(save/persist...) -- gains DB identity upon commit. doesn't exist ---persistent (get/load/jpql) -- YES
Detached	NO	YES
Removed	Trigger--session.delete (ref) Marked for removal YES	Upon commit --delete query fired --L1 cache destroyed --not a part of DB or cache

What is Session?

* Represents a wrapper around pooled out jdbc connection.

* Session object is persistance manager for the hibernate application

* Session object is the abstraction of hibernate engine for the Hibernate application

* Session object provides methods to perform CRUD operations

* Session is associated implicitely with L1 cache (having same scope as the session lifetime) , referred as Persistence context.

Example

save()	-	Inserting the record
get() / load()	-	Retrieveing the record
update()	-	Updating the record
delete()	-	Deleting the record

What is SessionFactory?

* It is a factory(provider) of session objects.

* we use sessionfactory object to create session object

* It is a heavy weight object, therefore it has to be created only once for an application(typically @ appln start up time) -- typically one per DB per web application.

* Its immutable --- Once SF is created , changes made to hibernate.cfg.xml will not be auto reflected in SF.

* It is associtated with L2 cache(must be explicitely enabled)

What is Configuration Object ? (org.hibernate.cfg.Configuration)

* Configuration object is used to create the SessionFactory object.

* Object Oriented Representation of Hibernate configuration file and mapping files(or annotations) is nothing but Configuration object.

*When we call configure() method on configuration object ,hibernate configuration file(hibernate.cfg.xml from run time classpath) and mapping files (or resources) are loaded in the memory.

Why connection pooling?

Java applications should use connection pools because :

* Acquiring a new connection is too expensive
* Maintaining many idle connections is expensive
* Creating prepared statements is expensive

* Hibernate provides basic or primitive connection pool -- useful only for classroom testing.

* Replace it by 3rd party vendor supplied connection pools(eg Apache or C3P0 or hikari in spring boot) for production grade applications.

Natural Key Vs Surrogate Key

*If u have User reg system -- then u have a business rule that --- user email must be distinct. So if u want to make this as a prim key --then user will have to supply this during regsitration.

* This is called as natural key. Since its value will be user supplied , u cant tell hibernate to generate it for u---i.e cant use @GeneratedValue at all.

* Where as -- if u say I will reserve user id only for mapping purposes(similar to serial no), it need not come from user at all & can definitely use hib. to auto generate it for u---this is ur surrogate key & can then use @GeneratedValue.

There are many advantages of Hibernate Framework over JDBC

1) Opensource , Lightweight : Hibernate framework is opensource & lightweight.

2) Fast performance: The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled bydefault.

Third type of cache is --query level cache.(not implicitely enabled)

3) Database Independent query: HQL (Hibernate Query Language) / JPQL (Java persistence query language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, If database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

4) Automatic table creation: Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

5) Simplifies complex join: To fetch data form multiple tables is easy in hibernate framework.

eg : To display the course names ordered by desc no of participants (many-to-many)

select c.name from dac_courses c inner join course_studs cs on c.id = cs.

c_id inner join dac_students s on cs.s_id = s.stud_id group by c.id order by count(*) desc;

JPQL -- select c from Course c join fetch c.students group by c.id order by count(*) desc

6) Provides query statistics and database status: Hibernate supports Query cache and provide statistics about query and database status.

7. Hibernate translates checked SQLException to un checked org.hibernate.HibernateException(super cls of all hibernate related errs)

---so that prog doesn't have to handle excs.

Advantages of hibernates:

1. Hibernate supports Inheritance, Associations, Collections.

2. In hibernate if we save the derived class object, then its base class object will also be stored into the database, it means hibernate supporting inheritance

3. Hibernate supports relationships like One-To-Many,One-To-One, Many-To-Many-to-Many, Many-To-One

4. This will also supports collections like List,Set,Map (Only new collections)

5. In jdbc all exceptions are checked exceptions, so we must write code in try, catch and throws, but in hibernate we only have Un-checked exceptions, so no need to write try, catch, or no need to write throws. Actually in hibernate we have the translator which converts checked to Un-checked ;)

6. Hibernate has capability to generate primary keys automatically while we are storing the records into database

7. Hibernate has its own query language, i.e hibernate query language which is database independent

So if we change the database, then also our application will works as HQL is database independent

HQL contains database independent commands

8. While we are inserting any record, if we dont have any particular table in the database, JDBC will rises an error like View not exist, and throws exception, but in case of hibernate, if it not found any table in the database this will create the table for us ;)

9. Hibernate supports caching mechanism by this, the number of round trips between an application and the database will be reduced, by using this caching technique an application performance will be increased automatically.

Hibernate supports annotations, apart from XML

10. Hibernate provided Dialect classes, so we no need to write sql queries in hibernate, instead we use the methods provided by that API.

11. Getting pagination in hibernate is quite simple.

Persistent Object Life cycle

1.Transient State

* An object is said to be in transient state if it is not associated with the session,and has no matching record

in the database table.

* Account account=new Account();

account.setAccno(101);

2.Persistent State

* An object is said to be in persistent state if it is associated with session

object (L1 cache) and will result into a matching record in the databse table.(i.e upon commit)

* session.save(account);tx.commit();

or Account account=session.get(Account.class,102);

***** When the object is in persistent state it will be in synchronization with the matching

record i.e if we make any changes to the state of persistent object it will be

reflected in the database.(after commiting tx) -- i.e automatic dirty checking will be performed.

3.Detached state

Object is not associated with session but has matching record in the database table. If we make any changes to the state of detached object it will NOT be reflected in the database.

***** session.clear(); *****session.evict(Object); *****session.close();

Note : By calling update method on session object it will go from detached state to persistent state.

By calling delete method on session object it will go from persistenet state to transient state.

* Explain the following methods of Session API

public void persist(Object ref) -- Persists specified transient POJO on underlying DB , upon comitting the transaction.

void clear()

When clear() is called on session object all the objects associated with the session object become detached.

But Databse Connection is not closed. (Completely clears the session. Evicts all loaded instances and cancel all pending saves, updates and deletions)

void close()

When close() is called on session object all the objects associated with the session object become detached and

also closes the Database Connection.

***** public void evict(Object ref)

* It detaches a particular persistent object detached or disassociates from the session. (Remove this instance from the session cache. Changes to the instance will not be synchronized with the database.)

void flush()

* When the object is in persistent state ,whatever changes we made to the object state will be reflected in the databse only at the end of transaction.

* If we want to reflect the changes before the end of transaction (i.e before commiting the transaction)

call the flush method. (Flushing is the process of synchronizing the underlying DB state with persistable state of session cache)

*** boolean contains(Object ref)**

The method indicates whethere the object is associated with session or not.

void refresh(Object ref) -- ref --persistent or detached

This method is used to get the latest data from database and make corresponding modifications to the persistent object state. (Re-read the state of the given instance from the underlying database)

public void update(Object ref)

If object is in persistent state no need of calling the update method .As the object is in sync with the database whatever changes made to the object will be reflect to database at the end of transaction.

***** When the object is in detached state record is present in the table but object is not in sync with database, therefore update() method can be called to update the record in the table

Which exceptions update method can raise?

1. StaleStateException -- If u are trying to update a record (using session.update(ref)), whose id doesn't exist.

i.e update can't transition from transient --->persistent

***** It can only transition from detached --->persistent.

eg -- update_book.jsp -- supply updated details + id which doesn't exists on db.

2. NonUniqueObjectException -- If there is already persistence instance with same id in session.

eg -- UpdateContactAddress.java

public Object merge(Object ref)

Can Transition from transient -->persistent & detached --->persistent.

Regarding Hibernate merge

1. The state of a transient or detached instance may also be made persistent as a new persistent instance by calling merge().

2. API of Session

Object merge(Object object)

3. Copies the state of the given object(can be passed as transient or detached) onto the persistent object with the same identifier.

3.If there is no persistent instance currently associated with the session, it will be loaded.

4.Return the persistent instance. If the given instance is unsaved, save a copy of and return it as a newly persistent instance. The given instance does not become associated with the session.

5. will not throw NonUniqueObjectException --Even If there is already persistence instance with same id in session.

***** public void saveOrUpdate(Object ref)

The method persists the object (insert) if matching record is not found (& id inited to default value) or fires update query If u supply Object , with non-existing ID -- Fires StaleStateException.

lock()

when lock() method is called on the session object for a persistent object ,untill the transaction is commited in the hibernate application , externally the matching record in the table cannot be modified.

    session.lock(object,LockMode);

    eg - session.lock(account,LockMode.UPGRADE);

Annotation support from pkg : javax.persistence (JPA)

1. Mark the entity or POJO class with @Entity annotation

@Entity : class level ---mandatory

2. @Table(name="table name") : class level annotation ---optional

3. @Id : can be field level OR getter method level. ---mandatory

4. optional

@GeneratedValue(strategy=GenerationType.AUTO) --> id generator supplied by persistence provider(app srvr's or ORM frmwork i.e hibernate) & not by DB

Mandatory Rule for Identifier property type---must be Serializable

5. @Column(name="col name ") --> not mandatory if same names

@Column(columnDefinition= "double(10,2)")

private double price;

3,4,5 : applicable to Entity id property

6.@Column(name="upper-cased col name") : for rest of prop to col names mapping(optional)

eg : for additional @Column annotations (method level annotation)

NOTE : Annotations like -- @Id,@GeneratedValue,@Column,@ElementCollection,@Temporal

--can either be applied @ field level(before data member) or property level(before getter)

@Temporal --- can be applied to --java.util.Date,Calendar, GregorianCalendar

How to get Scrollable Result from Query?

* ScrollableResults scroll(ScrollMode scrollMode) throws HibernateException

* Return the query results as ScrollableResults. The scrollability of the returned results depends upon JDBC driver support for scrollable ResultSets.

* Then can use methods of ScrollableResults ---first,next,last,scroll(n) .

7. How to create Named query from Session i/f?

What is a named query ?

Its a technique to group the HQL statements in single location(typically in POJOS) and lately refer them by some name whenever need to use them. It helps largely in code cleanup because these HQL statements are no longer scattered in whole code.

Fail fast: Their syntax is checked when the session factory is created, making the application fail fast in case of an error.

Reusable: They can be accessed and used from several places which increase re-usability.

eg : In POJO class, at class level , one can declare Named Queries

8. How to invoke native sql from hibernate?

Query q=hs.createSQLQuery("select * from books").addEntity(BookPOJO.class);

 l1 = q.list();

For composite primary key

Rules on prim key class

* Annotation -- @Embeddable (& NOT @Entity)

* Must be Serializable.

* Must implement hashCode & equals as per general contract.

In Owning Entity class

*Add usual annotation -- @Id.

Annotations related to relationships(associations) between entities

0. one -----> one : unidirectional relationship.

In Customer Entity class ---

@OneToOne(cascade=CascadeType.ALL)

@JoinColumn(name="addr_id")

private Address adr;

---owning side

In Address Entity --- no need of additional annotations.

1. one 1 <----> one : bidirectional relationship.

```
    @OneToOne(cascade=CascadeType.ALL,mappedBy="cust")
    private Address adr;
    ---owning side

    @OneToOne
    @JoinColumn(name="cust_id")
    private Customer cust;
```

2. one 1 <---->* many : bi-directional

At one side : field level annotaion @OneToMany(cascade=CascadeType.ALL,mappedBy="propertyName in many side")

NOTE -- cascade is optional attribute. can be skipped .

At many side :

```
    @ManyToOne
    @JoinColumn(name="prim key column name of one side")
```

Meaning - Acts as Foreign key column referred from one side

eg -- Course 1----* Students

Table structure for understanding ---

    Course table --- course_id(PK),name,start_date,end_date,fees

Students table --- id(PK),name,addr,course_id(FK)

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
private int courseId;
@OneToMany(cascade=CascadeType.ALL,mappedBy="myCourse")
private List<Student> students;
```

In Student POJO

```
    @ManyToOne
    @JoinColumn(name="courseId")
    private Course myCourse;
```

eg One User having multiple Vehicles.

In User class

```
    @OneToMany(mappedBy="user")
    private Collection<Vehicle> vehicles=new ArrayList<>();
```

---mappedBy attribute tells hibernate that instead of having a separate join table --map vehicles by user i.e user = name of the property appearing in Vehicle class , anno by @ManyToOne

***** In Vehicle class -- to supply our own name for the join col (o.w it will take def name)

```
    @ManyToOne
    @JoinColumn(name="USER_ID")
    private User user;
```

Annotation for Date

```
@Temporal(TemporalType.DATE)
private Date startDate;
```

Creates date type of column in underlying DB(java.util.Date or java.util.Calendar or GC)

Annotation for Time

```
@Temporal(TemporalType.TIME)
private Date openingTime;
```

Creates time type of column in underlying DB

Annotation for TimeStamp

```
@Temporal(TemporalType.TIMESTAMP)
private Date closingDateTime;
```

Creates datetime type of column in underlying DB

More details on one ---many

* The association may be bidirectional. In a bidirectional relationship, only one of the sides has to be the owner: --- the owner is responsible for the association column(s) update.

* To declare a side as not responsible for the relationship, the attribute mappedBy is used. mappedBy refers to the property name of the association on the owner side. You MUST NOT declare the join column since it has already been declared on the owners side.

Some more annotations

```
@Entity
@Table(name = "stock", catalog = "scott", uniqueConstraints = {
@UniqueConstraint(columnNames = "STOCK_NAME"),
@UniqueConstraint(columnNames = "STOCK_CODE") })
```

For Identity generator

```
@GeneratedValue(strategy=GenerationType.IDENTITY)
@Column(name="b_id")
private int bookId;
```

The two rules, for bidirectional one-to-one associations

1 The @JoinColumn annotation goes on the mapping of the entity that is mapped tothe table containing the join column, or the owner of the relationship. This might be on either side of the association.

2 The mappedBy element should be specified in the @OneToOne annotation in theentity that does not define a join column, or the inverse side of the relationship.

* Illegal to have a bidirectional association that had mappedBy on both sides.

* Incorrect not have it on either side.

* Hibernate will assume each side was the owner and each will have a join column.

* When an entity is associated with a Collection of other entities, it is in form of a one-to-many mapping.

* When a relationship is bidirectional, there are actually two mappings, one for each direction.

* A bidirectional one-to-many relationship always implies a many-to-one mapping back to the source.

eg : Course & Students

In this , there is a one-to-many mapping from Course -----> Student and a many-to-one mapping from Student -----> Course.

2. Many to Many association

In owning side of the association

@ManyToMany // mandatory

@JoinTable(name = "orders", joinColumns = @JoinColumn(name = "cust_id"), inverseJoinColumns = @JoinColumn(name = "prduct_id"))

public Set<Product> getProducts() {..}

Note:-DON'T add CascadeType.REMOVE -- in this type of association.

In the inverse side of the asso

@ManyToMany(mappedBy="nm of the prop appearing on the owning side")

public Set<Customer> getCustomers(){..}

Which table must have a foreign key ?

* When a Course entity has an number of Student entities stored in its collection, there is no definite way to store those references in the database table.

* Instead Student table MUST have foreign keys back to the Course

* So the one-to-many association is almost always bidirectional and never the owning side.

* In Course entity u need to map the Students with @OneToMany annotation.

* This doesn't have foreign key , so its an inverse side of relationship.

* Since this is the inverse side of the relationship, MUST include the mappedBy attribute.

eg : In Course Entity

mappedBy -- must refer to the prop name in the associated table --to specify ownership of the asso.

@OneToMany(cascade = CascadeType.ALL,mappedBy = "course")

students getter. (getStudents)

---mappedBy tells hibernate that instead of having a separate join table --map students by course i.e course = name of the property appearing in Student class , anno by @ManyToOne

Rules

1. The many-to-one side is the owning side, so the join column is defined on that side.

2. The one-to-many mapping is the inverse side, so the mappedBy element must be used.

Collection of 3 types --- entities , embeddables & basic types.

In Course 1<----->n Student the Course entity has a collection of Student instances.It is called a multivalued relationship. BUT collections of embeddable and basic types are not relationships; they are simply collections of elements .They are called element collections. Relationships define associations between independent entities , whereas element collections contain objects that are dependent upon the referencing entity, and can be retrieved only through the entity that contains them.

Annotation Used -- @ElementCollection --mandatory

BUT then hibernate creates its own table to store these embeddables. If u want to name the table , optional anno is @CollectionTable

@ElementCollection

@CollectionTable(name = "table_name", joinColumns = @JoinColumn(name = "join_col_name"))

Followed by getter of embeddables or basic types.

Hibernate Caching

* Caching is a facility provided by ORM frameworks which help users to get fast running web application, while help framework itself to reduce number of queries made to database in a single transaction.

* Hibernate achieves this by implementing first level cache.

* First level cache in hibernate is enabled by default and you do not need to do anything to get this functionality working. In fact, you can not disable it even forcefully.(typically)

* Its associated with Session object. Session object is created on demand from session factory and it is lost, once the session is closed.

* Similarly, first level cache associated with session object is available only till session object is live. It is available to session object only and is not accessible to any other session object in any other part of application.

NOTE

1. First level cache is associated with session object and other session objects in application can not see it.

2. The scope of cache objects is of session. Once session is closed, cached objects are gone forever.

3. First level cache is enabled by default and you can not disable it.

4. When we query an entity first time, it is retrieved from database and stored in first level cache associated with hibernate session.

5. If we query same object again with same session object, it will be loaded from cache and no sql query will be executed.(completely true only for get or load)

6. The loaded entity can be removed from session using evict(Object ref) method. The next loading of this entity will again make a database call if it has been removed using evict() method.

7. The whole session cache can be removed using clear() method. It will remove all the entities stored in cache.

***)** Caching is facility provided by ORM frameworks which help users to get fast running web application, while help framework itself to reduce number of queries made to database in a single transaction. Hibernate also provide this caching functionality, in two layers.

Fist level cache: This is enabled by default and works in session scope. Read more about hibernate first level cache.

Second level cache: This is apart from first level cache which is available to be used globally in session factory scope.

Above statement means, second level cache is created in session factory scope and is available to be used in all sessions which are created using that particular session factory.

It also means that once session factory is closed, all cache associated with it die and cache manager also closed down.

How second level cache works?

1. Whenever hibernate session try to load an entity, the very first place it look for cached copy of entity in first level cache (associated with particular hibernate session).

2. If cached copy of entity is present in first level cache, it is returned as result of load/get method.

3.If there is no cached entity in first level cache, then second level cache is looked up for cached entity.

4.If second level cache has cached entity, it is returned as result of load/get method. But, before returning the entity, it is stored in first level cache also so that next invocation to load method for entity will return the entity from first level cache itself, and there will not be need to go to second level cache again.

If entity is not found in first level cache and second level cache also, then database query is executed and entity is stored in both cache levels, before returning as response of load() method.

Entity Types :

1. If an object has its own database identity (primary key value) then it s type is Entity Type.

2. An entity has its own lifecycle. It may exist independently of any other entity.

3. An object reference to an entity instance is persisted as a reference in the database (a foreign key value).

eg : College is an Entity Type. It has it s own database identity (It has primary key).

Value Types :

1. If an object don t have its own database identity (no primary key value) then it s type is Value Type.

2. Value Type object belongs to an Entity Type Object.

3. It s embedded in the owning entity and it represents the table column in the database.

4. The lifespan of a value type instance is bounded by the lifespan of the owning entity instance.

Different types of Value Types

Basic, Composite, Collection Value Types :

1. Basic Value Types :

Basic value types are : they map a single database value (column) to a single, non-aggregated Java type.

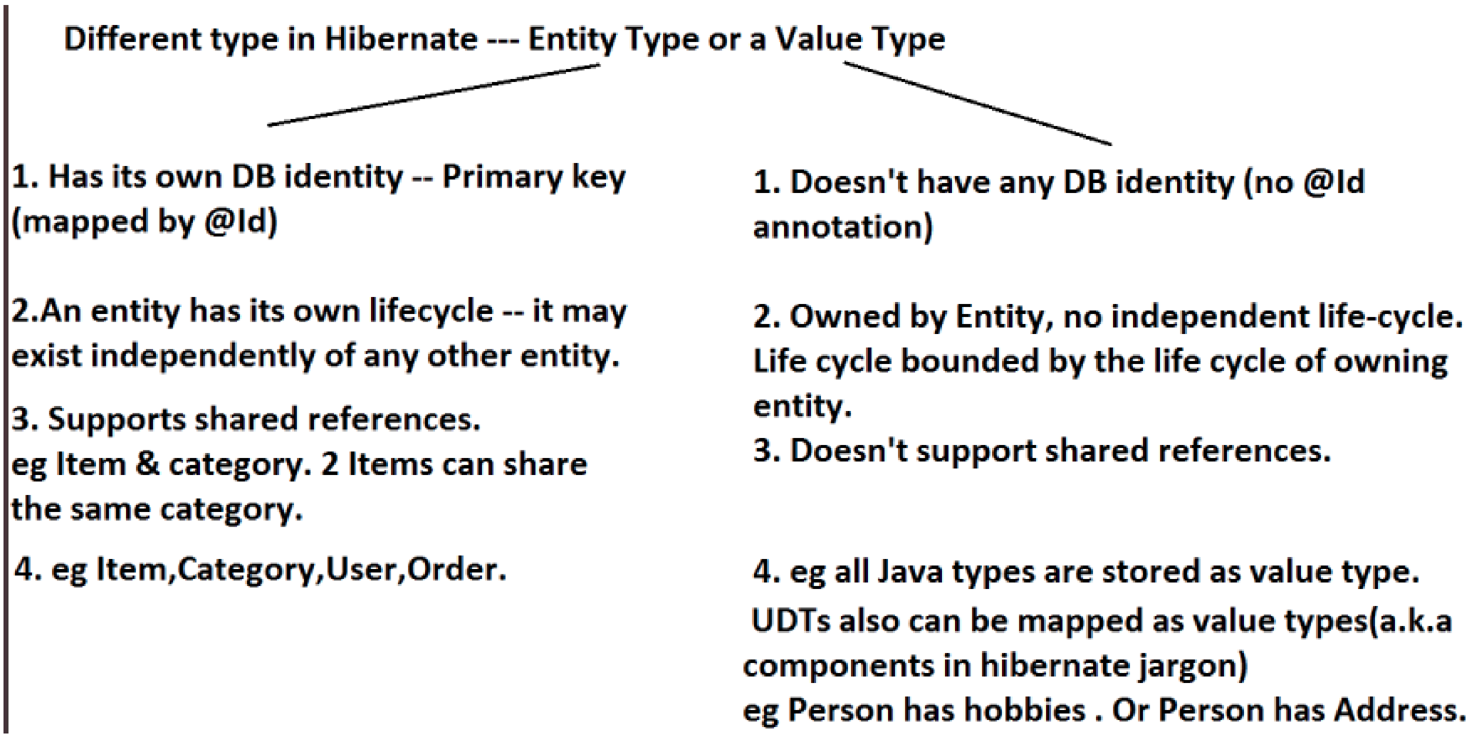
Hibernate provides a number of built-in basic types.

String, Character, Boolean, Integer, Long, Byte, etc.

2.Composite Value Types :

In JPA composite types also called Embedded Types. Hibernate traditionally called them Components.

2.1 Composite Value type looks like exactly an Entity, but does not own lifecycle and identifier.



Different type in Hibernate --- Entity Type or a Value Type

Entity Type	Value Type
1. Has its own DB identity -- Primary key (mapped by @Id)	1. Doesn't have any DB identity (no @Id annotation)
2.An entity has its own lifecycle -- it may exist independently of any other entity.	2. Owned by Entity, no independent life-cycle. Life cycle bounded by the life cycle of owning entity.
3. Supports shared references. eg Item & category. 2 Items can share the same category.	3. Doesn't support shared references.
4. eg Item,Category,User,Order.	4. eg all Java types are stored as value type. UDTs also can be mapped as value types(a.k.a components in hibernate jargon) eg Person has hobbies . Or Person has Address.

Annotations Used

1. @Embeddable : Defines a class whose instances are stored as an intrinsic part of an owning entity and share the identity of the entity. Each of the persistent properties or fields of the embedded object is mapped to the database table for the entity. It doesn't have own identifier.

eg : Address is eg of Embeddable

Student HAS-A Address(eg of Composition --i.e Address can't exist w/o its owning Entity i.e Student)

College HAS-A Address (eg of Composition --i.e Address can't exist w/o its owning Entity i.e College)

BUT Student will have its own copy of Address & so will College(i.e Value Types don't support shared reference)

2. @Embedded : Specifies a persistent field or property of an entity whose value is an instance of an embeddable class. The embeddable class must be annotated as Embeddable.

eg : Address is embedded in College and User Objects.

3. @AttributesOverride : Used to override the mapping of a Basic (whether explicit or default) property or field or Id property or field.

***** In Database tables observe the column names. Student table having STREET_ADDRESS column and College table having STREET column. These two columns should map with same Address field streetAddress. @AttributeOverride gives solution for this.

To override multiple column names for the same field use @AtributeOverrides annotation.

3. Collection Value Types :

Hibernate allows to persist collections.

But Collection value Types can be either collection of Basic value types, Composite types and custom types.

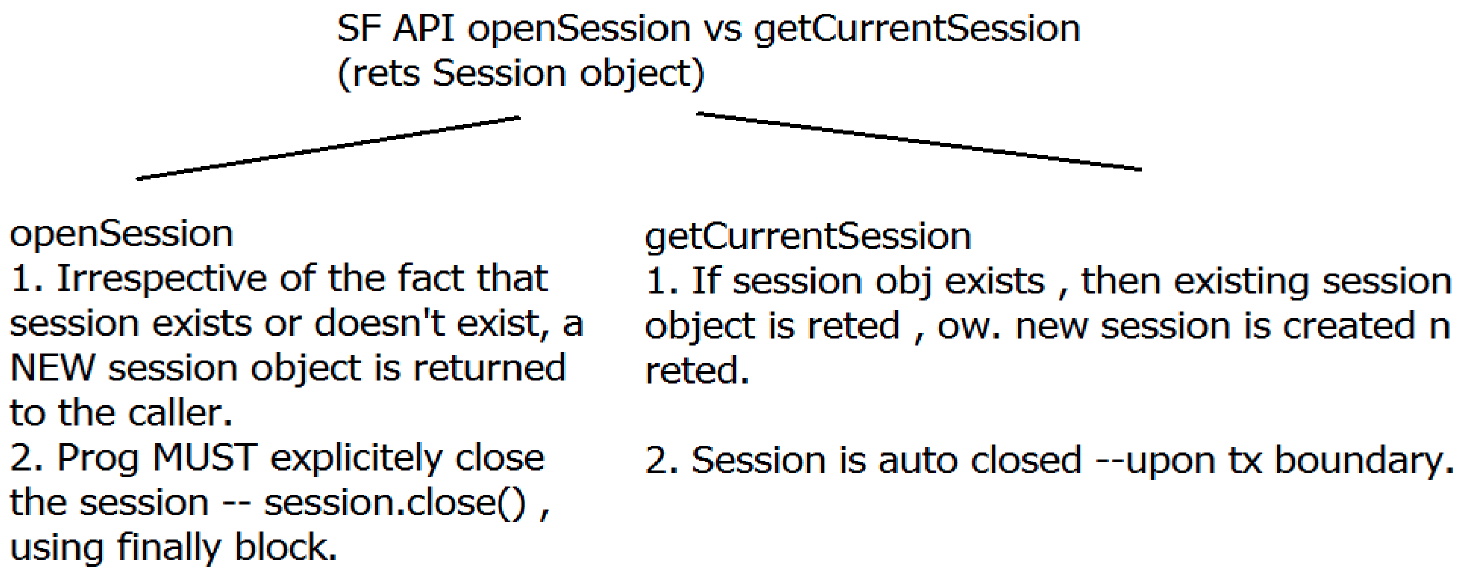
Eg : Collection mapping means mapping group of values to the single field or property. But we can   t store list of values in single table column in database. It has to be done in a separate table.

eg : Collection of embeddables

```
@ElementCollection
        @CollectionTable(name="CONTACT_ADDRESS", joinColumns=@JoinColumn(name="USER_ID"))
        @AttributeOverride(name="streetAddress", column=@Column(name="STREET_ADDRESS"))
        private List<ContactAddress> address;
```

eg : collection of basic type

```
        @ElementCollection
        @CollectionTable(name="Contacts", joinColumns=@JoinColumn(name="ID"))
        @Column(name="CONTACT_NO")
        private Collection<String> contacts;
```



SF API openSession vs getCurrentSession
(rets Session object)

openSession
1. Irrespective of the fact that session exists or doesn't exist, a NEW session object is returned to the caller.
2. Prog MUST explicitely close the session -- session.close() , using finally block.

getCurrentSession
1. If session obj exists , then existing session object is reted , ow. new session is created n reted.

2. Session is auto closed --upon tx boundary.

Session Tracking tchnique :

 HttpSession + URL rewriting

Why ????

To develop a web app , independent of cookies , for session tracking.

For tracking the clnt (clnt's session) : the only information, WC needs from the clnt browser is JSessionID value. If clnt browser is not sending it using cookie : Servlet/JSP prog can embed the JSessionID info in each outgoing URL .

What is URL Rewriting : Encoding the URL to contain the JSEssionID info.

W.C always 1st chks if JsessionID is coming from cookie, if not ---> then it will chk in URL : if it finds JseesionID from the encoded URL : extracts its value & proceeds in the same manner as earlier.

Regarding ID generators

* Each entity must have a primary key, which you annotate on the class with the @Id annotation. Typically, the primary key will be a single field, though it can also be a composite of multiple fields .

* The placement of the @Id annotation determines the default access strategy that Hibernate will use for the mapping. If the annotation is applied to a field as shown below, then   field access   will be used.

```
    @Id
    private Integer employeeId;
```

If, instead, the annotation is applied to the accessor for the field then property access will be used.

```
@Id
    public Integer getEmployeeId()
    {   return employeeId; }
```

* Property access means that Hibernate will call the mutator/setter instead of actually setting the field directly, what it does in case of field access. This gives flexibility to alter the value of actual value set in id field if needed. Additionally, you can apply extra logic on setting of   id   field in mutator for other fields as well.

* By default, the @Id annotation will not create a primary key generation strategy, which means that you, , need to determine what valid primary keys are, by setting them explicitly calling setter methods.

OR you can use @GeneratedValue annotation.

The strategy attribute must be a value from the javax.persistence.GeneratorType enumeration. If you do not specify a generator type, the default is AUTO. There are four different types of primary key generators on GeneratorType, as follows:

AUTO: Hibernate decides which generator type to use, based on the database   s support for primary key generation.

IDENTITY: The database is responsible for determining and assigning the next primary key.

SEQUENCE: Some databases support a SEQUENCE column type. It uses @SequenceGenerator.

TABLE: This type keeps a separate table with the primary key values. It uses @TableGenerator.

Regarding org.hibernate.LazyInitializationException

1. In JPA or hibernate --default fetching policy for Any--To--Many is -- LAZY

i.e if u try to fetch (using get/load/jpql) Course details(one side details) --hibernate WON't fetch student(many side) details.(Confirm it by looking at select query)

So in Course object --in place of actual student data from db , its collection of PROXY (=un fetched data from db) is kept.

When will hibernate throw this exception ?

If u try to access any un-fetched data from DB (represented by un-initilized proxy) from outside the session scope(in detached manner) --- hibernate throws LazyInitExc

Triggers -- any-to-many association

session's load method.

Fix

1. POJO layer soln.

Change fetching policy of one-to-many to eager

@OneToMany(......,fetch=FetchType.EAGER)

public List<Student> getStudents(){...}

---not recommended (since it may cause a performance hit)

Use case --if the size of many is small (few)

Even if user wants ONLY course details , hib will fetch all associated student details --causing performance hit.

2. Soln in DAO layer

--Access the size of collection --from inside session scope.

Disadv --- complete data fetched using multiple queries(select n+1 problem)

3. Better alternative --- join fetch

String jpql="select c from Course c join fetch c.students where c.name=:nm"

WHY Spring ?

* To simplify Java development.

* It supports loose coupling of java objects using dependency injection & AOP(aspect oriented programming)

* Reduces boilerplate code.

* Spring eco system keeps evolving every day , to include multiple spring projects eg : Spring social,Cloud support,microservices,spring boot etc.

* Suppports excellent integration support for ORM,JDBC(templates),MVC , RESTful web services...

What is spring?

* Its a container & a framework both.

Why Container --It manages life cycle of spring beans.

(spring bean --- java objects whose life cycle completely managed by SC(spring container)

eg : controller, service,DAO.

Why Framework --Supplies readymade implementation of standard patterns(eg :MVC,Proxy,singleton,factory, ORM ...)

*Spring is modular n extensive framework. * Spring is an open source framework since February 2003.

* Created by Rod Johnson ,earlier published under Apache license. * Currently hosted on pivotal.

* **One line answer --- Spring does not directly implement any of the J2EE specification BUT its created to make developing complex J2EE applications easier.**

*__Why learn one more frmwork ?__ --- when u already have EJB,Struts,Hibernate etc....

Spring helps you to

1.Build applications from plain old Java objects (POJOs) (known as spring beans)

2. Apply enterprise services non-invasively.

(w/o invasion means --- POJOs DONT implement or extend from spring APIs)   This capability applies to the Java SE programming model and to full and partial Java EE.

Examples of how you, as an application developer, can use the Spring platform advantage:

Make a Java method execute in a database transaction without having to deal with transaction APIs.

Make a local Java method a remote procedure without having to deal with remote APIs.

Make a local Java method an ORM operation without having to deal with overheads of ORM set up.

Make a local Java method a web service end point , without having to deal with JAX WS or JAX RS setups.

Simple answer to WHY Spring

Spring simplifies Java development.

Since above is a bold stmt -- to justify it --- there are main 4 reasons

Reasons ---it applies 4 key strategies

1. lightweight & min intrusive(POJOs not tied to spring)   development with POJOs

2. Loose coupling thro' DI/IoC & with usage of i/f

3.Declarative prog(XML or anno or java config) + thro' Aspects & common conventions

4. Boilerplate code reduction thro aspects & templates.

7. Def. of Spring ----

Spring is a lightweight , dependency injection and aspect-oriented container and framework. works on Ioc(Inversion of control)

Meaning

7.1 Lightweight Lesser no of JARs

JavaBeans / POJOs in Spring-enabled application often have no dependencies on Spring-specific classes.

7.2 Dependency Injection Spring allows loose coupling through dependency injection (DI).

DI =instead of an object looking up dependencies from a container(in EJB from EJB container or in RMI from RMIRegistry) or creating its own dependency(in Fixed JDBC , conn = DM.getCn(....)) , the container gives the dependencies to the object at instantiation without waiting to be asked.

*You can think of D.I as JNDI(Java naming & directory iterface -- Naming service) in reverse.

7.3 Aspect-oriented Spring supports aspect-oriented programming(AOP)

(AOP) allows separating application business logic from system services (such as auditing and transaction management,logging , security,transactions).

* Application objects perform only business logic and nothing more.

* They are not responsible for (or even aware of) other system concerns, such as eg : logging , transactions, or security

7.4 Why Spring is a Container ?

Spring is a container which manages the lifecycle and configuration of application objects.(spring beans)

In Spring, using config XMLs or annotations or java config, you can declare - how to create each of your application objects(spring beans)

- how to configure them

- how they should be associated with each other.(collaboration/wiring/coupling=connecting dependencies with dependent objs)

7.5 Why Spring is a framework ? Spring allows you to configure and compose complex applications from simpler components. In Spring, application objects are composed declaratively, typically in an XML file or using annotations

Spring also provides you with ready made implemetations of services like - transaction management, persistence framework,web mvc etc.

Spring is unique, for several reasons:

1. It helps you in important areas that many other popular frameworks don't. eg : readymade Hibernate templates.

2. Provides a way to manage your business objects - based on Dependeny injection(DI)

3.Spring is both comprehensive and modular.

Offers you lot many features, yet gives you choice to integrate layers one by one, test it & then add new features via new layers.

4 Spring is an ideal framework for test driven projects.

5. It is basically a one-stop shop, addressing most of the concerns of typical enterprise applications.

6.Using Spring one can centrally describe collaborating objects. From the earliest versions of Spring, there was an XML file that was used to describe the object graph.

Contents of XML ---- Consists of beans. Each bean element describes an object that will be created and given an id. Each property element describes a setter method on the object and the value that should be given to it. These setters are called for you by the Spring application container.

What is Spring ?

1. An open source framework since February 2003.

Created by Rod Johnson and described in his book Expert One-on-One: J2EE Design and Development.

Allows us to give capability of EJBs to plain JavaBeans without using an application server.

Any Java SE application can also use Spring to get simplicity, testability, and loose coupling.

2.Spring has been hosted on Pivotal

3. Spring is a lightweight framework. Most of your Java classes will have no dependency on Spring.   This means that you can easily transition your application from the Spring framework to any other frmwork. (Framework independence)

4. All Java applications that consist of multiple classes have inter-dependencies or coupling between classes. Spring helps us develop applications that minimize the negative effects of coupling and encourages the use of interfaces in application development.

5. Using interfaces in our applications to specify type helps make our applications easier to maintain and enhance later.

6. The Spring framework helps developers for separation of responsibilities.

eg scenario --

***** Think of a situation   Your manager tells you to do your normal development work(eg - write stock trading appln) + write down everything you do and how long it takes you.

***** A better situation would be you do your normal work, but another person observes what you re doing and records it and measures how long it took.

***** Even better would be if you were totally unaware of that other person and that other person was able to also observe and record , not just yours but any other people s work and time.

* That s separation of responsibilities.   --- This is what spring offers u thro AOP

8 Spring framework Modules

9. Advantages of ApplicationContext over BeanFactory

9.1Application contexts resolve text messages, including support for internationalization (I18N).

9.2 Application contexts provide a generic way to load file resources, such as images.

9.3Application contexts can publish events to beans that are registered as listeners.

IOC -- rather a generic term

Inversion of Control (IoC) is an object-oriented programming practice where the object coupling(dependent obj bound with dependency) is bound at run time by an assembler object(eg spring container) and is typically not known at compile time using static analysis.

* Unlike in traditional prog -- where dependent obj creates dependencies leading to tight coupling , container sets the dependencies (not US --not a prog or not a dependent obj) ---so its inversion of control

* Dependency. injection=Ioc+dependency inversion

Why IoC or Dependency Injection (advantages of IoC)

* There is a decoupling of the execution of a certain task from implementation.
* Every module can focus on what it is designed for.
* Modules make no assumptions about what other systems do but rely on their contracts/specs (=i/f)
* Replacing modules has no side effect on other modules.

Inversion of Control is sometimes referred to as the "Hollywood Principle: Don't call us, we'll call you",

Dependency injection (DI) is the ability to inject dependencies. DI can help make your code architecturally pure. It aids in using a design by interface approach as well as test driven development by providing a consistent way to inject dependencies. For example a data access object (DAO) may need a database connection. Thus the DAO depends on the database connection. Instead of looking up the database connection with JNDI, you could inject it. or another eg is JMS -- conn factory or destination

One way to think about a DI container like Spring is to think of JNDI turned inside out. Instead of the objects looking up other objects that it needs to get its job done (dependencies), with DI the container injects those dependent objects. This is the so-called Hollywood principle, you don't call us (lookup objects), we will call you (inject objects).

More on ApplicationContext

The instantiation of the ApplicationContext creates the container that consists of the objects defined in that XML file.

The purpose of this XML file is to create the beans and their relationship.

This XML file is then provided to the ApplicationContext instance, which creates a container with these beans and their object graphs along with relationships. The Spring container is simply a holder of the bean instances that were created from the XML file.

An API (getBean) is provided to query these beans and use them accordingly from our client application.

IoC is also known as dependency injection (DI).

It is a process whereby objects define their dependencies,that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method.

The container then injects those dependencies when it creates the bean.

This process is fundamentally the inverse,(hence the name Inversion of Control (IoC)), of the bean itself controlling the instantiation or location of its dependencies by using direct construction of classes, or a mechanism such as the Service Locator pattern.

The org.springframework.beans and org.springframework.context packages are the

basis for Spring Framework's IoC container. The BeanFactory interface provides an advanced

configuration mechanism capable of managing any type of object. ApplicationContext is a

sub-interface of BeanFactory. It adds easier integration with Spring's AOP features; message resource handling (for use in internationalization), event publication; and application-layer specific contexts such as the WebApplicationContext for use in web applications.

What is spring ?

container --manages life cycle of spring beans (spring bean --- java objects whose life cycle completely managed by SC(spring container)

eg : controller, service,DAO.

framework --rdymade implementation of std patterns(eg :MVC,Proxy,singleton,factory, ORM ...)

Spring is modular n extensive framework.

What is dependency injection ?

In JSP---JB---DAO -- POJO layers

Dependent Objs -- JavaBean , Hibernate based DAO, JDBC Based DAO

Dependencies --- DAO, SessionFactory , DB connection

All of above are examples of tight coupling.

Why --Any time the nature of the dependency changes , dependent obj is affected(i.e u will have to make changes in dependent obj)

* Tight coupling --strongly un desirable.

Why -- difficult to maintain or extend.

In above layers , Java bean creates the instance of DAO.

Hibernate based DAO , looks up SF from HibUtils.

JDBC based DAO , looks up db connection from DBUtils.

i.e dependent objects are managing their dependencies. ---traditional/conventional programming model.

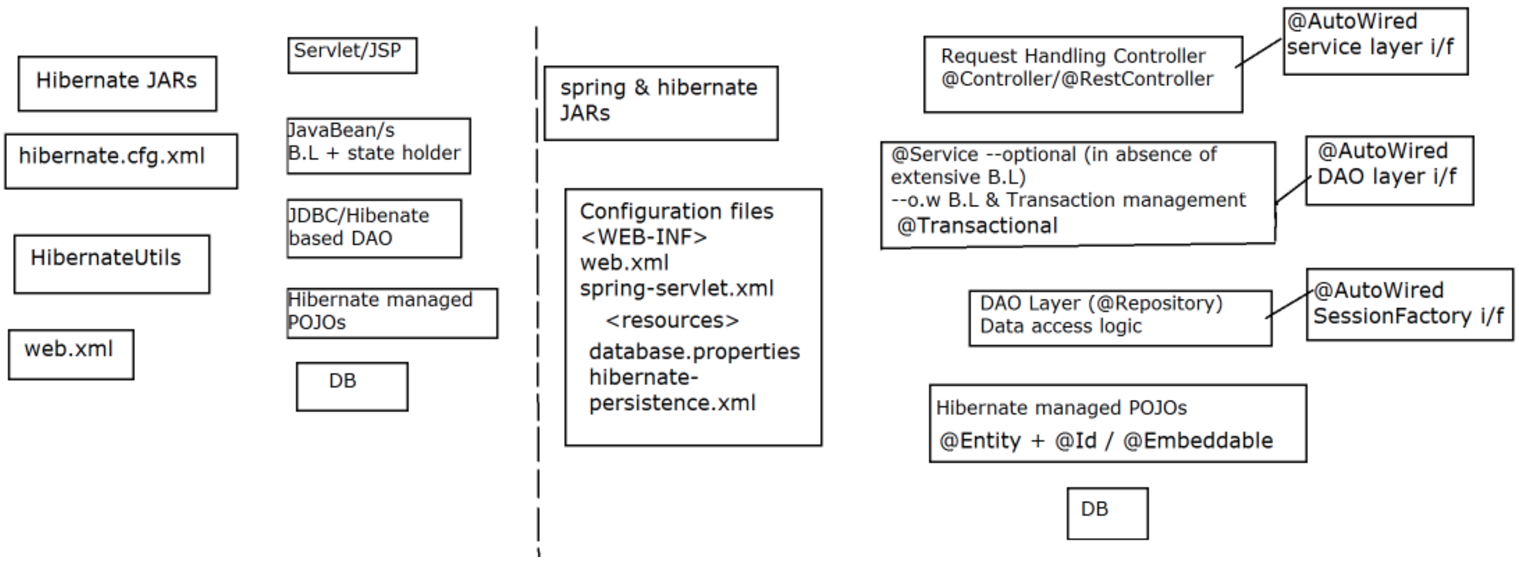
What is D.I ?(Dependency injection=wiring=collaboration between dependent & dependency)

Instead of dependent objs managing their dependencies , 3rd party containers(eg : Angular / Spring/ EJB) will auto create the dependecies & make it available to dependents, directly @ run time.

Since control of managing dependencies is no longer with dependent objects(BUT lies with container)--its called as IoC ---Inversion of control

** Hollywood principle --You don't call us , we will call you....

Servlet/JSP Using JavaBean Layers Vs Spring MVC Layers



Spring 4 DB Transactions

Required JARS ---org.springframework.transaction & aop jars

Benefits of Spring Transaction Management

* Very easy to use, does not require any underlying transaction API knowledge
* Your transaction management code will be independent of the transaction technology
* Both annotation- and XML-based configuration
* It does not require to run on a server - no server needed

What is a Transaction?

A Transaction is a unit of work performed on the database and treated in a reliable way independent of other transaction. In database transaction processing ACID property refers to the Atomicity, Consistency, Isolation, Durability respectively.

Atomicity- This property says that all the changes to the data is performed as if they form single operation. For example suppose in a bank application if a fund transfer from one account to another account the atomicity property ensures that is a debit is made successfully in one account the corresponding credit would be made in other account.

Consistency- The consistency property of transaction says that the data remains in the consistence state when the transaction starts and ends. for example suppose in the same bank account, the fund transfer from one account to another account, the consistency property ensures that the total value(sum of both account) value remains the same after the transaction ends.

Isolation- This property says that, the intermediate state of transaction are hidden/ invisible to another transaction process.

Durability- The Durability says that when the transaction is completed successfully, the changes to the data persist and are not un-done, even in the event of system failure.A transaction is not considered durable until it commits. A system failure entails a database recovery, which includes a rollback procedure for all uncommitted transactions, ultimately leaving the database in a consistent state.

Transaction Handling

Now, in Java you can handle transactions with plain SQL, with plain JDBC (a bit higher level), using Hibernate (or any other ORM library), or on an even higher level - with EJB or, finally, Spring!

EJBs require an application server, but spring based jdbc application doesn't.

Ways of Transaction Handling

Programmatic vs. Declarative

* Spring offers two ways of handling transactions: programmatic and declarative. If you are familiar with EJB transaction handling, this corresponds to bean-managed and container-managed transaction management.

* Programmatic means you have transaction management code surrounding your business code. That gives you extreme flexibility, but is difficult to maintain and too much of boilerplate code.

* Declarative means you separate transaction management from the business code. You only use annotations or XML based configuration.

As a summary

* programmatic management is more flexible during development time but less flexible during application life
* declarative management is less flexible during development time but more flexible during application life

Global transactions Vs Local Transactions

Global transactions enable you to work with multiple transactional resources, typically multiple relational databases . The application server manages global transactions through the JTA. (complex to use through UserTransaction object)

Local Transactions

Local transactions are resource-specific, such as a transaction associated with a JDBC connection. Local transactions may be easier to use, but have significant disadvantages as they cannot work across multiple transactional resources.

Spring's solution

Spring resolves the disadvantages of global and local transactions. It enables application developers to use a consistent programming model in any environment. You write your code once, and it can benefit from different transaction management strategies in different environments. It supports both declarative and programmatic transaction management. Most users prefer declarative transaction management.

API details

1. The key to the Spring transaction abstraction is the notion of a transaction strategy. Its the central interface in Spring's transaction infrastructure. A transaction strategy is defined by the org.springframework.transaction.PlatformTransactionManager

interface: which has TransactionStatus getTransaction(TransactionDefinition td) throws TransactionExc

TransactionException --- As in Spring's philosophy, the TransactionException that is thrown by any of the PlatformTransactionManager interface's methods , is unchecked Transaction infrastructure failures are generally fatal , managed by spring Tx frmwork & developer is NOT forced to handle this.

The TransactionDefinition interface specifies:

Isolation: The degree to which this transaction is isolated from the work of other transactions.

Concurrent transactions cause problems that might be difficult to investigate.

* Lost update - two transactions both update a row, the second transaction aborts, both changes are lost
* Dirty read - reading changes that are not yet committed
* Unrepeatable read - a transactions reads twice the same row, getting different data each time
* Phantom read - similar to the previous one, except that the number of rows changed

Now, the perfect solution to these problems is maximum isolation, but in reality this would cost too much resources and could lead to deadlocks. So, instead, you will set one of five isolation levels (where the fifth one is actually the maximum isolation level):

Supported levels

ISOLATION_DEFAULT -- Use the default isolation level of the underlying datastore.

ISOLATION_READ_UNCOMMITTED --- Indicates that dirty reads, non-repeatable reads and phantom reads can occur.

ISOLATION_READ_COMMITTED --- Indicates that dirty reads are prevented; non-repeatable reads and phantom reads can occur.

ISOLATION_REPEATABLE_READ -- Indicates that dirty reads and non-repeatable reads are prevented; phantom reads can occur.

ISOLATION_SERIALIZABLE -- Indicates that dirty reads, non-repeatable reads and phantom reads are prevented.

Transaction Propagation

Whenever a one transactional method calls other transactional mehod , a decision is made - what to do with the transaction. Create a new one? Use an existing one if it exists, otherwise create a new one? Use an existing one only if it exists, otherwise fail?

Supported behaviors ---

MANDATORY --Supports a current transaction; throws an exception if no current transaction exists.

REQUIRED -- default behavior.

Supports a current transaction; creates a new one if none exists.

NESTED ---Executes within a nested transaction if a current transaction exists, otherwise same as REQUIRED

SUPPORTS Supports a current transaction; executes non-transactionally if none exists.

REQUIRES_NEW Creates a new transaction, suspending the current transaction if one exists.

NEVER Does not support a current transaction; throws an exception if a current transaction exists.

NOT_SUPPORTED Does not support a current transaction; always executes non-transactionally.

Timeout: in seconds .How long this transaction runs before timing out and being rolled back automatically by the underlying transaction infrastructure.

Default value = -1 , indefinite w/o time out

Otherwise specify value

eg

@Transactional(timeout=100)

Read-only status: A read-only transaction can be used when your code reads but does not modify data.

Read-only transactions can be a useful optimization in some cases, such as when you are using Hibernate.

eg -- @Transactional(readOnly = true)

default is false.

Rollback behavior

With Spring transaction management the default behavior for automatic rollback is this: Only unchecked exceptions cause a rollback. Unchecked exceptions are RuntimeExceptions and Errors.

But can be changed.

eg --

```
@Transactional(rollbackFor = IOException.class, noRollbackFor = RuntimeException.class)
public void doSomething(...)
```

* Implementation steps & concept for annotation based declarative transaction management.

1. For plain JDBC implementations of PlatformTransactionManager --- use DataSourceTransactionManager --- implementation class for a single JDBC DataSource.

2. Declare the same in spring configuration xml file.

eg --

```
<!-- tx manager bean -->
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
p:dataSource-ref="dataSource">
</bean>
```

3. To enable annotated transaction support , add transaction namespace(tx) & add following

```
<tx:annotation-driven transaction-manager="transactionManager" />
```

Note : can even skip attribute transaction-manager, if id of Tx Mgr bean is transactionManager

This completes configuration steps

4. In service layer beans (typically annotated with @Service) , add method level annotation @Transactional along with suitable properties.

Spring-mvc-req-flow

1. Thin clnt sends request along with rq. params.(*.htm)

2. First it reaches Spring   s DispatcherServlet which acts like front controller in MVC model II . It is a common web-application pattern where a single servlet delegates responsibility for a request to other components of an application to perform the actual processing.

3. The DispatcherServlet   s job is to send the request on to a Spring MVC controller(prog supplied)

4. As typical appln may have several controllers, DispatcherServlet consults Handler mapping to select controller.

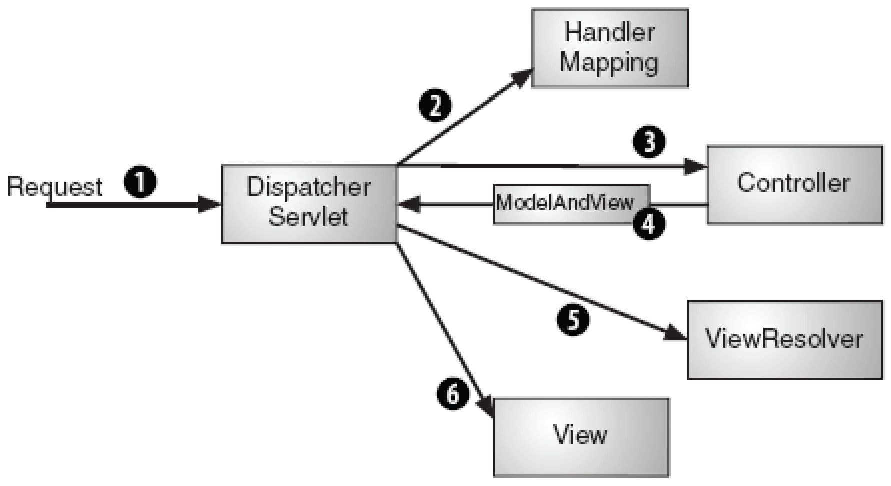
5. The Handler mapping will choose a controller based on rq. URL.

6. Req thus reaches the controller. Controller may use one or more service layer objs for exec of B.L

The result of B.L needs to be carried back to the user and displayed in the browser. This info. is the model. It has to be sent to JSP(or any other view template) for converting it to HTML like format.

7. So the controller will package up the model data and the name of a view into a ModelAndView object. It rets the request+ ModelAndView back to the DispatcherServlet. (M&V doesn   t carry a reference to the actual JSP but only carries a logical result name that will be used to look up the actual view that will produce the resulting HTML.

8. DispatcherServlet now consults a view resolver to find the actual JSP. & delivers the model data to view JSP



9. The view will use the model data to render a page that will be sent as dyn resp back to the clnt browser.

object.

The ViewResolver provides a mapping between view names and actual views.

UrlBasedViewResolver Simple implementation of the ViewResolver interface thateffects the direct resolution of logical view names to URLs, without an explicit mapping definition. This is appropriate if your logicalnames match the names of your view resources in a straightforward manner, without the need for arbitrary mappings.

XML config   Annotation

1. <bean id="abc" class="beans.TestBean"/>

class level annotations --stereotype annotations

@Component --ordinary spring bean

@Controller -- req handling spring bean

@Service ---B.L supplying bean

@Repository --DAO layer sping bean

@RestController -- RESTful service end point spring bean

2. scope="singleton|prototype"

@Scope --class level annotation

3. lazy-init="false|true"

@Lazy -- class level annotation

4. init-method

@PostConstruct --method level annotation

5. destroy-method

@PreDestroy --method level annotation

6. factory-method --no support in annotations

7. autowire=byType | constructor

@AutoWired ---It can appear before a setter / paramterized constr OR directly

Field level injection --- no setters , no paramed constrs ,add @Autowired directly to a field

eg : In VendorController

@AutoWired //required=true ---mandatory

private IVendorService service;

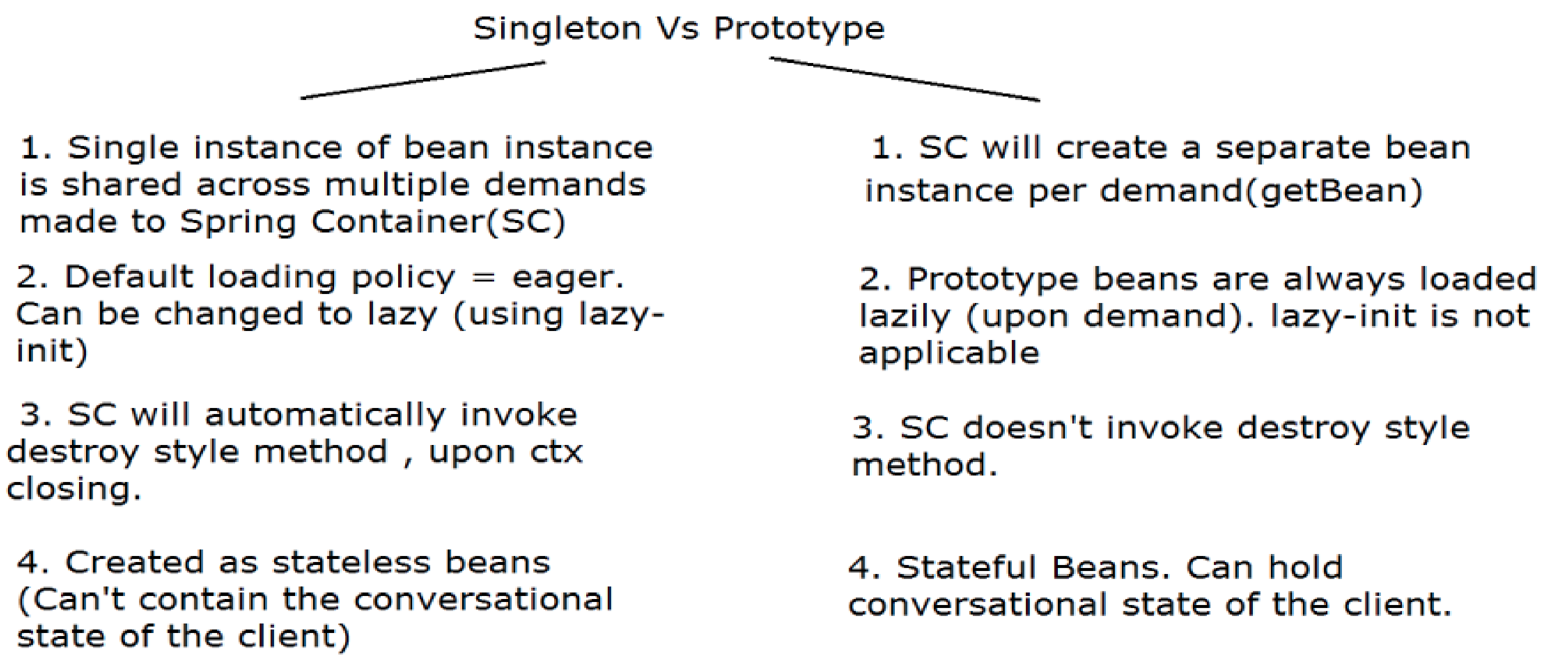
8. autowire=byName

eg : In VendorController bean

@AutoWired

@Qualifier("abc")

private IVendorService service;



Singleton Vs Prototype

1. Single instance of bean instance is shared across multiple demands made to Spring Container(SC)	1. SC will create a separate bean instance per demand(getBean)
2. Default loading policy = eager. Can be changed to lazy (using lazy-init)	2. Prototype beans are always loaded lazily (upon demand). lazy-init is not applicable
3. SC will automatically invoke destroy style method , upon ctx closing.	3. SC doesn't invoke destroy style method.
4. Created as stateless beans (Can't contain the conversational state of the client)	4. Stateful Beans. Can hold conversational state of the client.

Enter MVC --Model-View-Controller (pattern)

Model I architecture --No specific separation

(any web comp servlet/JSP can execute any functionality --req processing,B.L, Data access logic,navigation , resp generation)

Suitable for smaller web apps.

Model II architecture -- MVC pattern based architecture, based upon clear cut separation of roles(func/concerns/ tasks)

* Model -- Java bean --holds conversational state/data + B.L methods

* View -- JSP --P.L (UI)

* Controller -- Servlet / (Servlet Filter --used in Struts 2 architecture) -- Navigation based upon incoming request,manages Java beans, request processing

Front Controller based model II(MVC) architecture

Any client sending any request to web app -- its intercepted by a common entry point --front controller element(can be implemented by servlet/filter). = centralized dispatcher

It will dispatch clnt request to further components , based upon nature of the req.

How does spring achieve loose coupling ?

1. IoC -- IoC is achieved using Dependency Injection(D.I)

2. Aspect Oriented Programming(AOP)

1.What is o.s.w.s.ModelAndView?

-- A class -- holder of logical view name + model(=map of model attributes)

What is a model attribute ?

An attribute(=key & value pair --attrName & value)

which represents results of B.L or some data, to be shared with view layer.

Who creates it ? --Req handling controller.

Sends it to –D.S. D.S will save them under request scope & push it to view layer.

How to read them in JSP(view)?

    Using EL syntax.

  Constr of ModelAndView(String logicalView,String attrName,Object attrValue)

2. Understand concept of Model

* Concept of Model --o.s.ui.Model (i/f) = map of model attributes

* Type of Map -- Map<String,Object>

* API of Model --to create model attrs

    public Model addAttribute(String name,Object value)

* who cretes empty map -- SC

* How do u(Controller) access it --can be passed as one of the params of request handling methods. (as Dependency)

* who populates it -- SC / Controller(prog)

* why ? -- to share the results available in controller layer with View.

* Model attr map is sent from controller --> F.C Front Controller (Dispatcher Servlet)

F.C adds it in request scope & pushes it(sends it) to JSP (view layer)

So this is called push MVC architecture.

3. How to add model attr in Model map?

public Model addAttribute(String nm,Object value);

*** How to get Model map ?**

Using D.I -- just tell SC that your req handling method needs a model map(how ? -- by adding it in the method arg) & SC will create Model map & inject it in your code.

4. **How to access request params in Controller method?**

Ans : use @RequestParam annotation , on method args.

eg : @RequestParam int id

SC invokes -- int id=Integer.parseInt(request.getParameter("id"));

1. If there multiple request params(use case -- register/update) --- bind POJO directly to a form.(2 way form binding technique)

How ?

1.1 For loading the form (in showForm method of the controller) , bind empty POJO (using def constr) in model map

How ?

Explicit --add Model as dependency & u add

map.addAttribute(nm,val)

OR better way

implicit -- add POJO as a dependency

eg : User registration

@GetMapping("/reg")

public String showForm(User u) {...}

What will SC do ?

SC --- User u=new User();

chks --- Are there any req params coming from client ? --- typically --no --- only getters will be called --

adds pojo as model attr (in Model map)

map.addAttribute("user",new User());

1.2 In form (view ---jsp) -use spring form tags along with modelAttribute

Steps

1. import spring supplied form tag lib

2. Specify the name of modelAttribute under which form data will be exposed.(name of model attr mentioned in the controller)

```
<s:form method="post" modelAttribute="user">
  <s:input path="email"/>.....
</s:form>
```

1.3 Upon form submission (clnt pull I)

clnt sends a new req --- containing req params

@PostMapping("/reg")

public String processForm(User u,RedirectAttributes flashMap,HttpSession hs) {

//SC calls

User u=new User();

SC invokes MATCHING (req param names --POJO prop setters)

setters. -- conversational state is transferred to Controller.

adds pojo as model attr (in Model map)

map.addAttribute("user",u)

Thus you get a populated POJO directly in controller w/o calling <jsp:setProperty> & w/o using any java bean.

PRG pattern(Post-redirect-get pattern)

--- to avoid multiple submission issue in a web app.

Replace forward view(server pull) by redirect view (clnt pull) --a.k.a double submit guard.

How to replace default forward view by redirect view in spring MVC ?

Ans -- use redirect keyword.

eg : return "redirect:/vendor/details";

D.S invokes response.sendRedirect(response.encodeRedirectURL("/vendor/details"));

Next request from clnt --- ..../vendor/details

How to remember user details till logout?

Ans : add them in session scope.

How to access HttpSession in Spring?

Using D.I

How -- Simply add HttpSession as method argument of request handling method.

How to remember the details(attributes) till the next request (typically required in PRG --redirect view)

Ans -- Add the attributes under flash scope.

(They will be visible till the next request from the same clnt)

How to add ?

Use i/f -- o.s.w.s.mvc.support.RedirectAttributes

Method

public RedirectAttributes addFlashAttribute(String attrName,Object value)

How to access them in view layer in the next request?

via request scope attributes.

eg : In case of successful login --save user details under session scope(till user log out) & retain status mesg only till the next request.

In case of invalid login --save status under request scope.

How to take care of links(href)/form actions + add URL rewriting support ?

1. Import spring supplied JSP tag lib.

(via taglib directive)

prefix ="spring"

2. Use the tag.

<a href="<spring:url value='/user/logout'/>">Log Out</a>

/ --- root of curnt web app.

What will be the URL if cookies are enabled ?

http://host:port/spring_mvc/user/logout

What will be the URL if cookies are disabled ?

http://host:port/spring_mvc/user/logout;jsessionid=egD5462754

OR form action example

eg : <form action="<spring:url value='/admin/list'/>"> .....

</form>

From Logout

1. Discard session
2. Forward the client to logout.jsp

How to auto navigate the clnt to home page after logging out after some dly ?

Ans : By setting refresh header of HTTP response.

API of HttpServletResponse

public void setHeader(String name,String value)

name --- refresh

value --- 10;url=home page url (root of web app)

How to get the root of curnt web app ?

API of HttpServletRequest

String getContextPath()

What will hapeen if any controller returns redirect view name to D.S ?

eg : UserController -- return "redirect:/admin/list"

D.S skips the V.R & sends temp redirect response to the clnt browser.

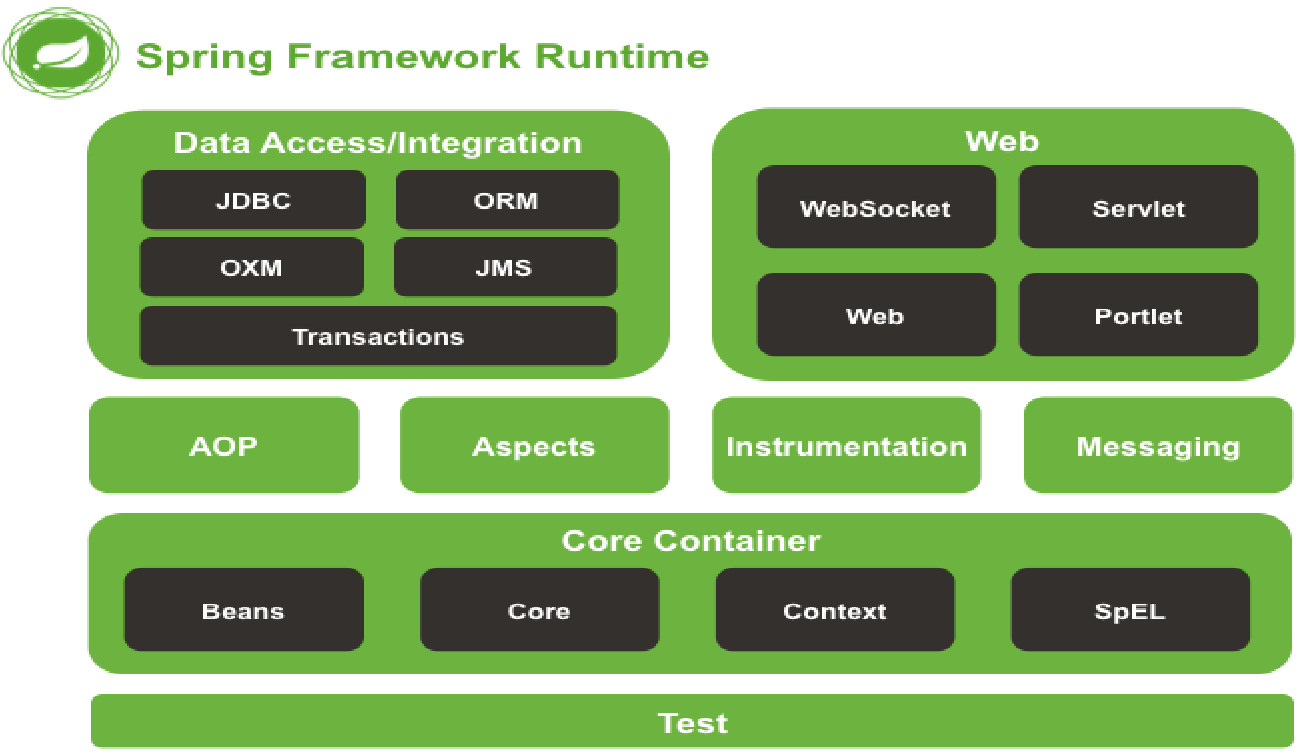
How ?

D.S invokes --- response.sendRedirec(response.encodeRedirectURL(".../admin/list");

So clnt browser will send a next request ---with method=get

URL --

http://host:port/spring_mvc/admin/list



POJO layer --new JPA annotation

1. **@MappedSuperclass**

To be applied at the common super class for all the entities, where one can define commonly required anotations.

No table created for this class.

Typically add --id & version field with suitable annotaitons.

Enhances re usability.

2. **@Version** -- can be applied to int/Integer/TimeStamp.. type of data members.

Automatically supports optimistic locking feature.

(i.e prevents lost update problem)

Hibernate throws OptismisticLockingException(root cause --StaleStateException) , in case of concurrent updates.

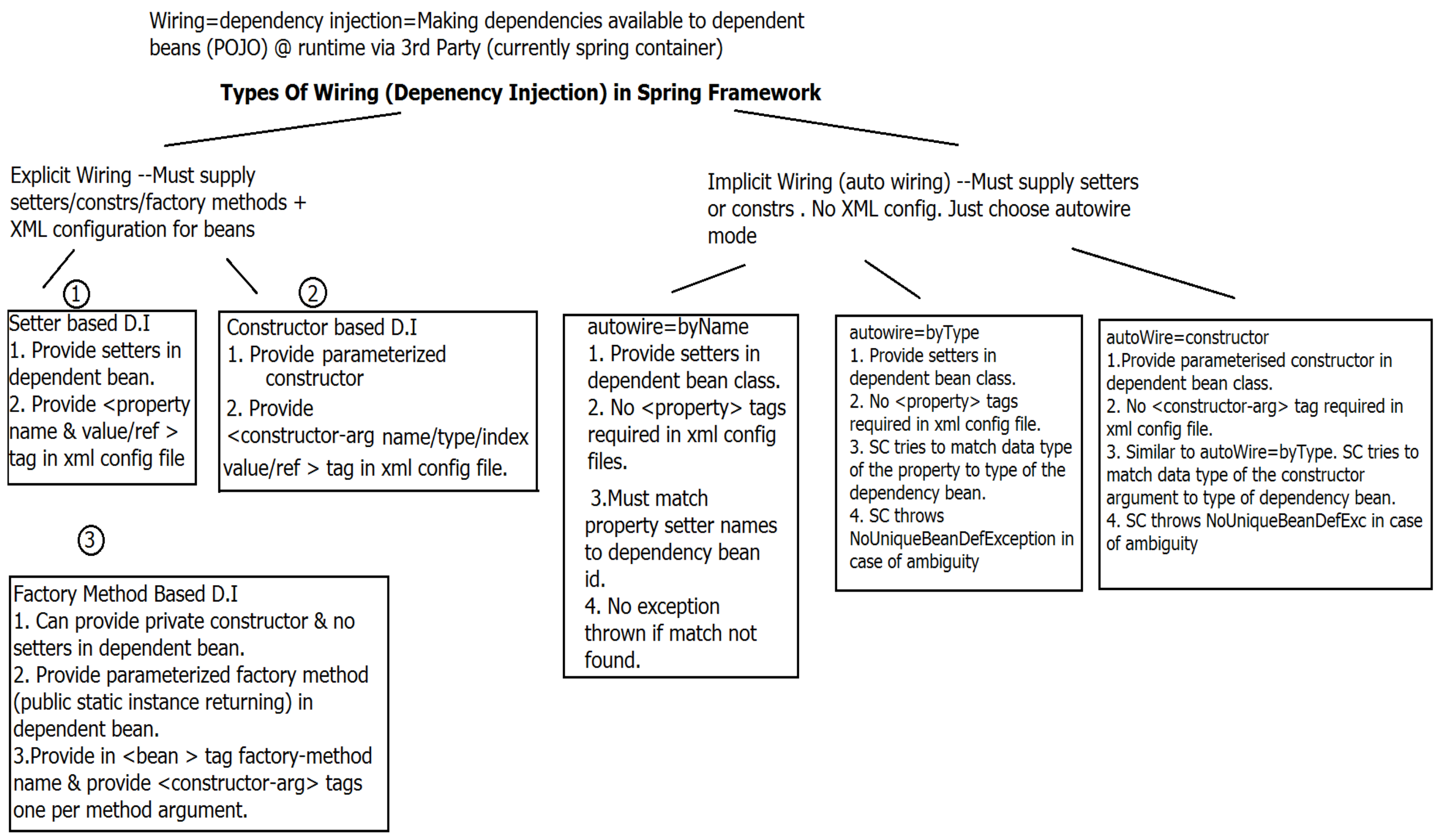
Preferred over pessimistic locikng --in web apps , for the scalability.

Import Spring web MVC annotation

@ModelAttrbute

Annotation that binds a method parameter or method return value to a named model attribute, available to a view layer. To be added in controller classes with @RequestMapping methods.

Eg : @ModelAttribute(name = "ac_types")



Wiring=dependency injection=Making dependencies available to dependent beans (POJO) @ runtime via 3rd Party (currently spring container)

Types Of Wiring (Depenency Injection) in Spring Framework

Explicit Wiring --Must supply setters/constrs/factory methods + XML configuration for beans

① Setter based D.I
1. Provide setters in dependent bean.
2. Provide <property name & value/ref > tag in xml config file

② Constructor based D.I
1. Provide parameterized constructor
2. Provide <constructor-arg name/type/index value/ref > tag in xml config file.

③ Factory Method Based D.I
1. Can provide private constructor & no setters in dependent bean.
2. Provide parameterized factory method (public static instance returning) in dependent bean.
3.Provide in <bean > tag factory-method name & provide <constructor-arg> tags one per method argument.

Implicit Wiring (auto wiring) --Must supply setters or constrs . No XML config. Just choose autowire mode

autowire=byName
1. Provide setters in dependent bean class.
2. No <property> tags required in xml config files.
3.Must match property setter names to dependency bean id.
4. No exception thrown if match not found.

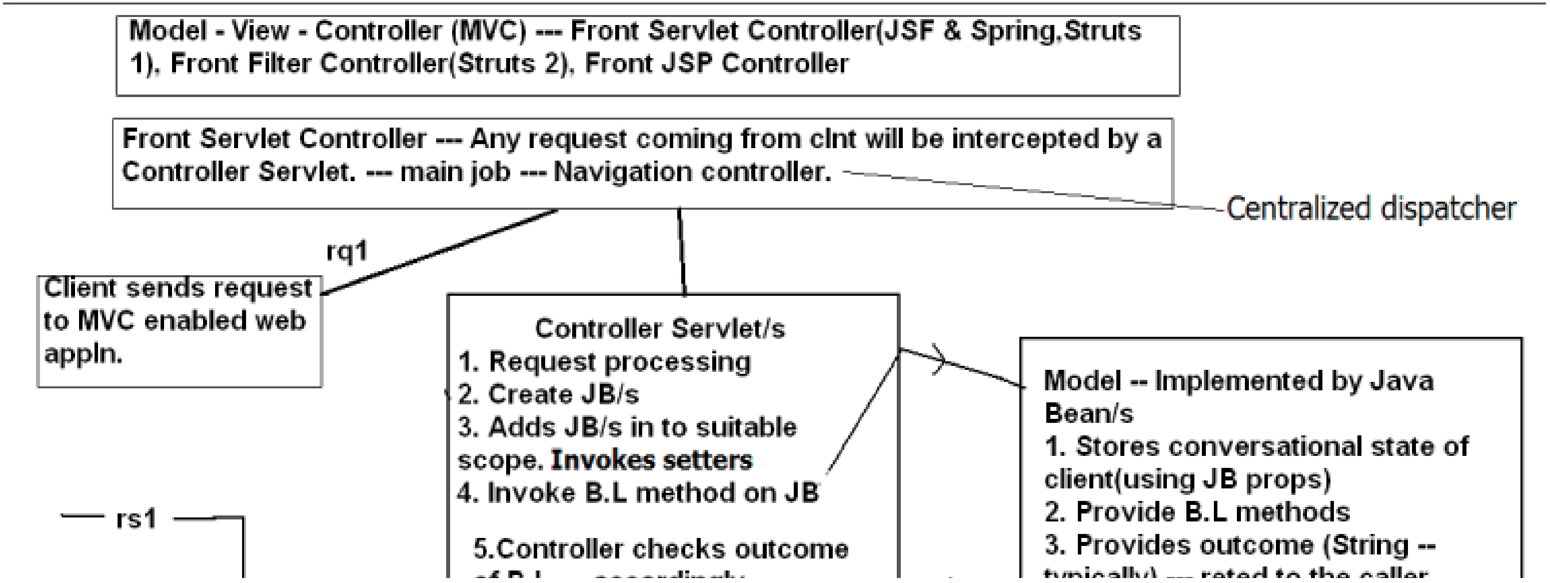
autowire=byType
1. Provide setters in dependent bean class.
2. No <property> tags required in xml config file.
3. SC tries to match data type of the property to type of the dependency bean.
4. SC throws NoUniqueBeanDefException in case of ambiguity

autoWire=constructor
1.Provide parameterised constructor in dependent bean class.
2. No <constructor-arg> tag required in xml config file.
3. Similar to autoWire=byType. SC tries to match data type of the constructor argument to type of dependency bean.
4. SC throws NoUniqueBeanDefExc in case of ambiguity

WHY MVC?

Implementation of MVC : using Front Servlet Controller Pattern
1.Thin clnt sends the HTTP rq. to a servlet controller.
2.Servlet/s reads the request params & performs initial processing if reqd.
3. Servlet controller will instantiate the Model componet/s (Java Beans)
2 ways : can directly invoke the parameterised constr. to instantiate & load the state of the JBs.
OR
Can invoke the def. constr & then invoke setters similar to ur JSP scenario.
Now JBs hold the data/state of the web appln reflecting the clnt state.
4. Servlet controller sends rq. to JBs to execute B.L
As a result of B.L , the servlet controller dyn. forwards/redirects the client to view (consisting of 1 or more JSPs)
5. JSPs contact JBs to load the state(using <jsp:getProperty> or EL syntax) .
6. Using the state info. JSPs genrate dyn contents & send the resp to the clnt.

MVC FLOW


Model - View - Controller (MVC) --- Front Servlet Controller(JSF & Spring,Struts 1), Front Filter Controller(Struts 2), Front JSP Controller

Front Servlet Controller --- Any request coming from clnt will be intercepted by a Controller Servlet. --- main job --- Navigation controller.

Centralized dispatcher

rq1

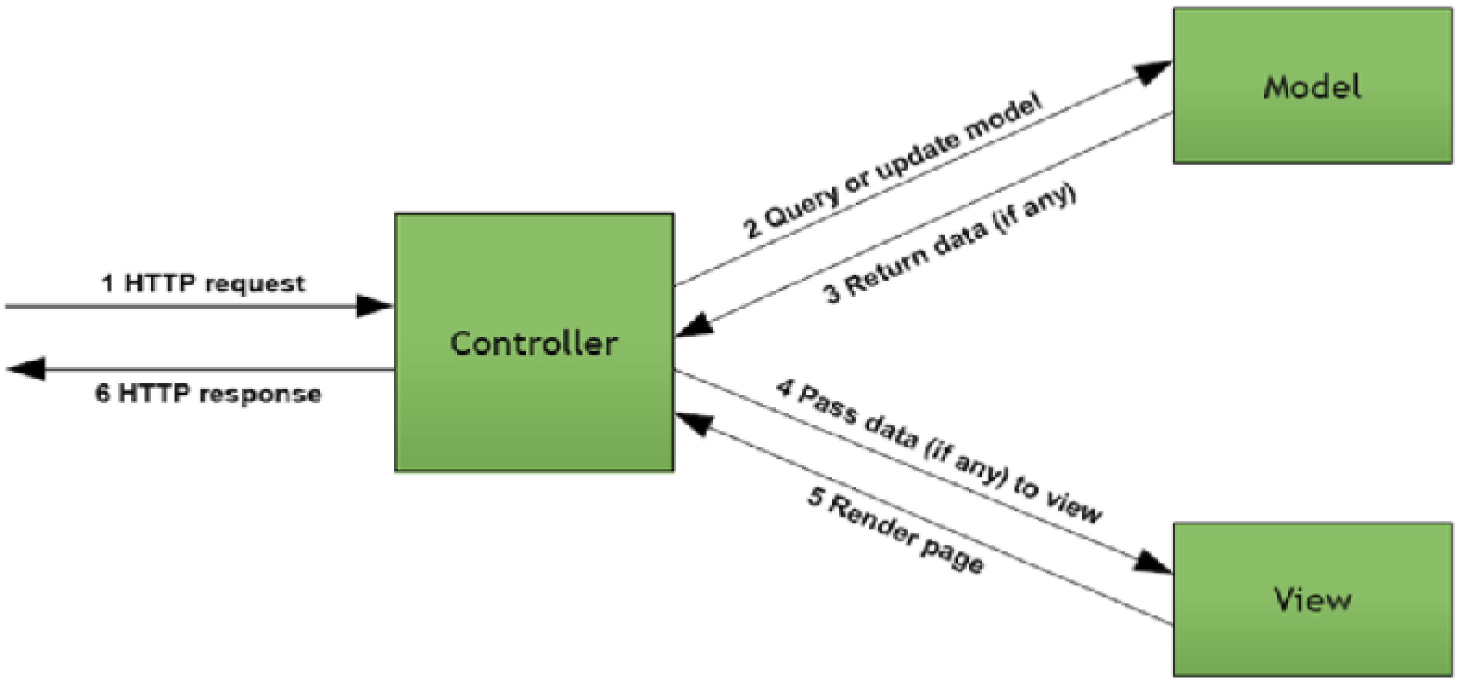
Client sends request to MVC enabled web appln.

Controller Servlet/s
1. Request processing
2. Create JB/s
3. Adds JB/s in to suitable scope. Invokes setters
4. Invoke B.L method on JB

5.Controller checks outcome

Model -- Implemented by Java Bean/s
1. Stores conversational state of client(using JB props)
2. Provide B.L methods
3. Provides outcome (String --

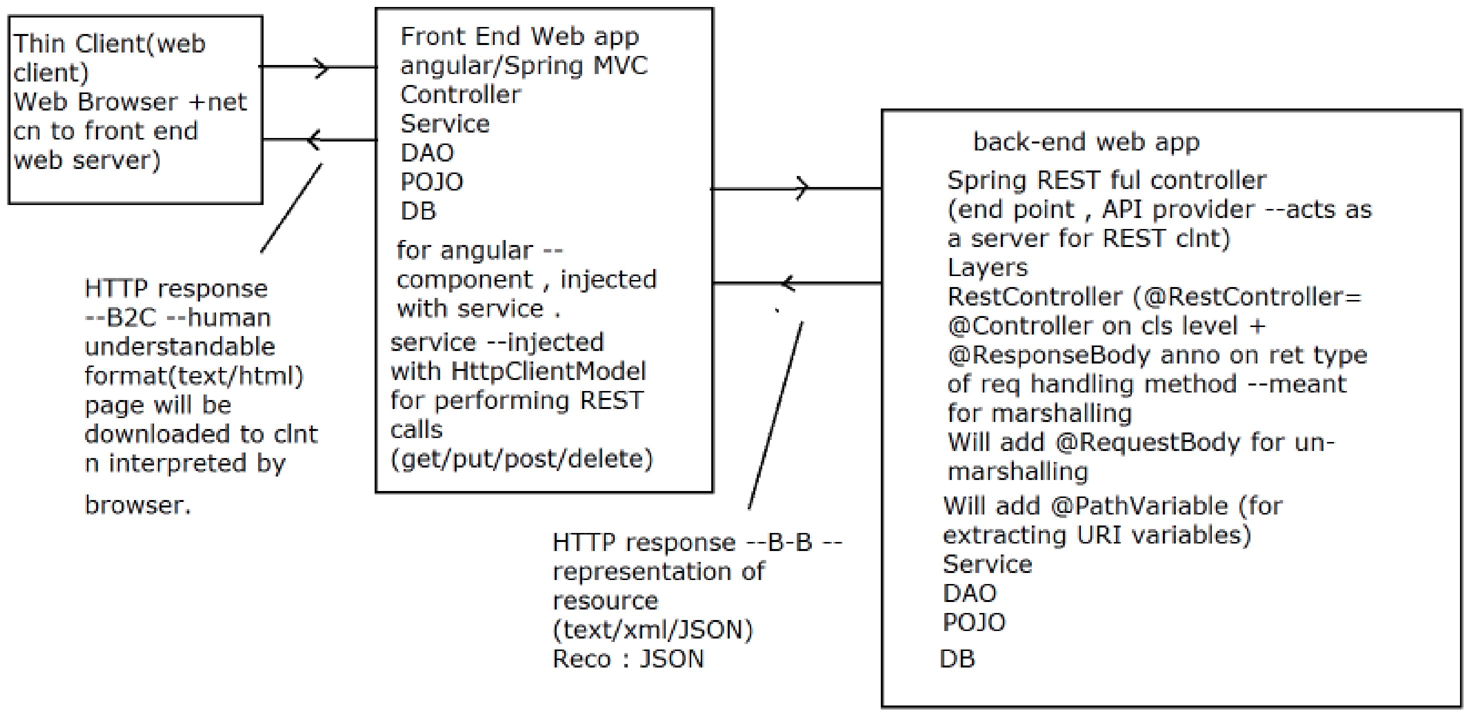
rs1



MVC advantages

1. Division of responsibilities among the various components.
2.One comonent is not TOO MUCH burdened with TOO many jobs.
3. Cleaner separation between request processing, navigation, business logic and presentation logic.
4. Reusability of Business logic components across various envs.
5. Each comp. can be implemnted completely inde. of others.
eg . Can generate fresh/attractive/new set of views(display renderers) keeping same B.L & navigation logic.
6. Single model can be made to support multiple views & which view to select can be decided dyn.

Full Stack Development (consisting of Front end web app n back end web app)
Front end --- Angular/React JS/Vue JS/Java/.NET
Back end -- Express / Spring Boot(Java)/.NET...


Thin Client(web client)
Web Browser +net cn to front end web server)
HTTP response --B2C --human understandable format(text/html) page will be downloaded to clnt n interpreted by browser.
Front End Web app
angular/Spring MVC
Controller
Service
DAO
POJO
DB
for angular -- component , injected with service .
service --injected with HttpClientModel for performing REST calls (get/put/post/delete)
HTTP response --B-B -- representation of resource (text/xml/JSON)
Reco : JSON
back-end web app
Spring REST ful controller (end point , API provider --acts as a server for REST clnt)
Layers
RestController (@RestController= @Controller on cls level + @ResponseBody anno on ret type of req handling method --meant for marshalling
Will add @RequestBody for un-marshalling
Will add @PathVariable (for extracting URI variables)
Service
DAO
POJO
DB

P.L validations ---

Understanding hibernate-persistence.xml & tx management

1. Supply the location of DB property file .

<context:property-placeholder

location="classpath:/database.properties" />

2.Configure a spring bean , to create Apache supplied connection pool.

I/F --javax.sql.DataSource (represents Connection pool)

Imple class --Apache supplied --org.apache.commons.dbcp2.BasicDataSource

Inject CP properties via setter Based D.I

3. Configure SessionFactory bean , supplied by Spring.

i/f --org.hibernate.SessionFactory (hibernate supplied)

SF provider -- o.s.orm.hibernate5.LocalSessionFactoryBean

Inject SF properties via setter Based D.I

eg : packgesToScan, show_sql , hbm2ddl.auto

Inject the ref of CP bean into SF

4. Configure spring supplied tx manager bean , to automate tx management(using @Transactional)

I/F : o.s.transaction.PlatformTransactionManager

Implementation class --o.s.orm.hibernate5.HibernateTransactionManager

5. Enable annotation based tx supprt

<tx:annotationDriven/>

4. Understanding Transaction Management in Spring

How to automate Tx management in spring?

1. Add spring supplied tx manager bean in config file

```
<bean id="transactionManager"
        class="org.springframework.orm.hibernate5.HibernateTransactionManager"
p:sessionFactory-ref="sessionFactory">
</bean>
```

2. Enable tx annotation support

```
        <tx:annotation-driven />
```

3. Use @Transactional attribute typically in Service or DAO Layer.

4. How to customize tx management -- using @Transactional attributes

4.1 timeout

eg : @Transactional(timeout=100)

service/dao layer method

4.2 readOnly --

def value --false;

eg : @Transactional(readOnly=true)

4.3 @Transactional(rollbackFor = IOException.class, noRollbackFor = RuntimeException.class)

public void doSomething(...)

SOAP	REST
• SOAP stands for Simple Object Access Protocol	• REST stands for Representational State Transfer
• SOAP is a protocol. SOAP was designed with a specification. It includes a WSDL file which has the required information on what the web service does in addition to the location of the web service.	• REST is an Architectural style in which a web service can only be treated as a RESTful service if it follows the constraints of being 1. Client Server 2. Stateless

	3. Cacheable 4. Layered System 5. Uniform Interface
• SOAP cannot make use of REST since SOAP is a protocol and REST is an architectural pattern.	• REST can make use of SOAP as the underlying protocol for web services, because in the end it is just an architectural pattern.
• SOAP uses service interfaces to expose its functionality to client applications. In SOAP, the WSDL file provides the client with the necessary information which can be used to understand what services the web service can offer.	• REST use Uniform Service locators to access to the components on the hardware device. For example, if there is an object which represents the data of an employee hosted on a URL as http://demo.guru99 , the below are some of URI that can exist to access them http://demo.guru99.com/Employee http://demo.guru99.com/Employee/1

• SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot.
 	
`<?xml version="1.0"?>`
`<SOAP-ENV:Envelope`
`xmlns:SOAP-ENV`
`="http://www.w3.org/2001/12/soap-envelope"`
`SOAP-ENV:encodingStyle`
`=" http://www.w3.org/2001/12/soap-encoding">`
`<soap:Body>`
`  <Demo.guru99WebService`
`  xmlns="http://tempuri.org/">`
`     <EmployeeID>int</EmployeeID>`
`     </Demo.guru99WebService>`
`  </soap:Body>`
`</SOAP-ENV:Envelope>`
	• REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. Below is an example of a JSON message passed to a web server. You can see that the size of the message is comparatively smaller to SOAP.
	
	{"city":"Mumbai","state":"Maharastra"}

• SOAP can only work with XML format. As seen from SOAP messages, all data passed is in XML format.	• REST permits different data format such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON.

Optimistic Locking

When we talk about locking we are often referring to optimistic locking. The optimistic locking model subscribes to the philosophy that there is a good chance that the transaction in which changes are made to an entity will be the only one that actually changes the entity during that interval. This translates into the decision to not acquire a lock on the entity until the change is actually made to the database, usually at the end of the transaction.

* When the data actually does get sent to the database to get updated at flush time or at the end of the transaction, the entity lock is acquired and a check is made on the data in the database. The flushing transaction must see whether any other transaction has committed a change to the entity in the intervening time since this transaction read it in and changed it. If a change occurred, it means that the flushing transaction has data that does not include those changes and should not write its own changes to the database lest it overwrite the changes from the

intervening transaction. At this stage, it must roll back the transaction and throw a special exception called OptimisticLockException.

Transactions & Locking

1. Change tx isolation level --to REPEATABLE_READ

OR

2 Use "select for update" Query

Both of above approach applies a write lock

Better approach --Optismistic Locking

* JPA 2 supports both optimistic locking and pessimistic locking. Locking is essential to avoid update collisions resulting from simultaneous updates to the same data by two concurrent users. Locking in JPA) is always at the database object level, i.e. each database object is locked separately.

* Optimistic locking is applied on transaction commit. Any database object that has to be updated or deleted is checked. An exception is thrown if it is found out that an update is being performed on an old version of a database object, for which another update has already been committed by another transaction.

* Optimistic locking should be the first choice for most applications, since compared to pessimistic locking it is easier to use and more efficient.

* In the rare cases in which update collision must be revealed earlier (before transaction commit) pessimistic locking can be used. When using pessimistic locking, database objects are locked during the transaction and lock conflicts, if they happen, are detected earlier.

* Optismistic Locking

Add @Version annotated property in hibernate POJO.(data type Integer)

* The initial version of a new entity object (when it is stored in the database for the first time) is 1. In every transaction in which an entity object is modified its version number is automatically increased by one.

* During commit , hibernate checks every database object that has to be updated or deleted, and compares the version number of that object in the database to the version number of the in-memory object being updated. The transaction fails and an OptimisticLockException is thrown if the version numbers do not match, indicating that the object has been modified by another user (using another transaction) since it was retrieved by the current updater.

Pessimistic Locking

The main supported pessimistic lock modes are:

PESSIMISTIC_READ - which represents a shared lock.

PESSIMISTIC_WRITE - which represents an exclusive lock.

Setting a Pessimistic Lock

An entity object can be locked explicitly by the lock method: org.hibernate.Session API

void lock(Object object,LockMode lockMode)

* Obtain the specified lock level upon the given object. This may be used to perform a version check (LockMode.READ) or to upgrade to a pessimistic lock (LockMode.PESSIMISTIC_WRITE)

eg :   sf.getCurrentSession().lock(employee, LockMode.PESSIMISTIC_WRITE);

What is REST ?

REST stands for REpresentational State Transfer.

Why the name ?

* It's the representation of the resource (eg : image/student/customer/invoice/bill....) using known data exchange formats(text/xml/json) --which gets transferred between client & server .

* REST is web standards based architecture and uses HTTP Protocol for data communication.

* It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. (ROA)

* REST was first introduced by Roy Fielding in 2000.

* In REST architecture, a REST Server simply provides access to resources and REST client accesses and presents the resources.

* Here each resource is identified by URIs

* REST uses various representations to represent a resource like text, JSON and XML. Most popular light weight data exchange format used in web services = JSON

HTTP Methods

Following well known HTTP methods are commonly used in REST based architecture.

GET - Provides a read only access to a resource.

POST - Used to create a new resource.

DELETE - Used to remove a resource.

PUT - Used to update a existing resource or create a new resource.

***RESTFul Web Services**

A web service is a collection of open protocols and standards used for exchanging data between applications or systems.

Platform & technology independent solution.

Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer.

This interoperability (e.g., between Java and Python, or Windows and Linux applications or java & .net) is due to the use of open standards.

Web services based on REST Architecture are known as RESTful web services.

These web services use HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation such as JSON and set of HTTP Methods.

Annotations

1. @PathVariable --- handles URL templates.

eg : URL -- http://host:port/test_web/products/10

Method of Controller

@GetMapping("/{pid}")

public Product getDetails(@PathVariable int pid)

{...}

In the above URL , the path variable {pid} is mapped to an int . Therefore all of the URIs such as /products/1 or /products/10 will map to the same method in the controller.

2. The @ResponseBody annotation is used to marshall(serialize) the return value into the HTTP response body. Spring comes with converters that convert the Java object into a format understandable for a client.

3.The @RequestBody annotation, instead, unmarshalls the HTTP request body into a Java object injected in the method.

* The RestTemplate is similar to other Spring templates such as JmsTemplate and JdbcTemplate in that Spring eliminates a lot of boot strap code and thus makes your code much cleaner. When applications use the RestTemplate they do not need to worry about HTTP connections, that is all encapsulated by the template. They also get a range of APIs from the RestTemplate which correspond to the well know HTTP methods (GET, PUT, POST, DELETE, HEAD, OPTIONS). These APIs are overloaded to cater for things like different ways of passing parameters to the actual REST API.

* Create resful service provider project & then develop its client

For spring restful web service --actually only spring web mvc , hibernate validator & json jars are sufficient.

Server side steps

1. Create Spring Web MVC application

2. The layers --service --dao--pojo --DB are the same.

3. In controller layer , replace @Controller by @RestController annotation, at class level.

4. Request Handling methods will respond to different HTTP methods

(get/post/put/delete)

5. For method=get

* Can directly return either a resource eg : Person,BankAccount or Customer or still better is can return entire HTTP response encapsulated in ResponseEntity<T>

* What is org.springframework.http.ResponseEntity<T>

* Represents HTTP response entity, consisting of status,headers and body.

@RestController : Spring 4′s new @RestController annotation.

Its a combination of @Controller and @ResponseBody.

@RequestBody : If a method parameter is annotated with @RequestBody, Spring will bind the incoming HTTP request body(for the URL mentioned in @RequestMapping for that method) to that parameter. While doing that, Spring will [behind the scenes] use HTTP Message converters to convert the HTTP request body into domain object [deserialize request body to domain object], based on ACCEPT or Content-Type header present in request.

@ResponseBody : If a method is annotated with @ResponseBody, Spring will bind the return value to outgoing HTTP response body. While doing that, Spring will [behind the scenes] use HTTP Message converters to convert the return value to HTTP response body [serialize the object to response body], based on Content-Type present in request HTTP header. As already mentioned, in Spring 4, no need to use this annotation.

ResponseEntity is a real deal. It represents the entire HTTP response. Good thing about it is that you can control anything that goes into it. You can specify status code, headers, and body. It comes with several constructors to carry the information you want to sent in HTTP Response.

@PathVariable This annotation indicates that a method parameter should be bound to a URI template variable [the one in '{}'].(binding between request handling method parametere & URI template variable)

MediaType : With @RequestMapping annotation, you can additionally, specify the MediaType to be produced or consumed (using produces or consumes attributes) by that particular controller method, to further narrow down the mapping.

URI Template Patterns

* URI templates can be used for convenient access to selected parts of a URL in a @RequestMapping

method.

* A URI Template is a URI-like string, containing one or more variable names. When you substitute values for these variables, the template becomes a URI. The proposed RFC for URI Templates defines how a URI is parameterized. For example, the URI Template http://www.example.com/users/

{userId} contains the variable userId. Assigning the value fred to the variable yields http://
www.example.com/users/fred.

* In Spring MVC you can use the @PathVariable annotation on a method argument to bind it to the value of a URI template variable:

```
@RequestMapping(value="/owners/{ownerId}", method=RequestMethod.GET)
public String findOwner(@PathVariable String ownerId, Model model) {
Owner owner = ownerService.findOwner(ownerId);
model.addAttribute("owner", owner);
return "displayOwner";
}
```

* The URI Template " /owners/{ownerId}" specifies the variable name ownerId. When the controller handles this request, the value of ownerId is set to the value found in the appropriate part of the URI.

* For example, when a request comes in for /owners/abc, the value of ownerId is abc.

* To process the @PathVariable annotation, Spring MVC needs to find the matching URI template variable by name. You can specify it in the annotation:

```
@RequestMapping(value="/owners/{ownerId}", method=RequestMethod.GET)
public String findOwner(@PathVariable("ownerId") String theOwner, Model model) {
// some implementation
}
```

Or if the URI template variable name matches the method argument name you can omit that detail. As long as your code is not compiled without debugging information, Spring MVC will match the method argument name to the URI template variable name:

Background

* REST is a way to access resources which lie in a particular environment. For example, you could have a server that could be hosting important documents or pictures or videos. All of these are an example of resources. If a client, say a web browser needs any of these resources, it has to send a request to the server to access these resources. Now REST defines a way on how these resources can be accessed.

eg of a web application which has a requirement to talk to other applications such Facebook, Twitter, and Google.

* Now if a client application had to work with sites such as Facebook, Twitter, etc. they would probably have to know what is the language Facebook, Google and Twitter are built on, and also on what platform they are built on.

Based on this, we can write the interfacing code for our web application, but this could prove to be a nightmare.

* So instead , Facebook, Twitter, and Google expose their functionality in the form of Restful web services. This allows any client application to call these web services via REST.

What is Restful Web Service?

* REST is used to build Web services that are lightweight, maintainable, and scalable in nature. A service which is built on the REST architecture is called a RESTful service. The underlying protocol for REST is HTTP, which is the basic web protocol. REST stands for REpresentational State Transfer

* The key elements of a RESTful implementation are as follows:

1. Resources The first key element is the resource itself. Let assume that a web application on a server has records of several employees. Let's assume the URL of the web application is http://www.server.com. Now in order to access an employee record resource via REST, one can issue the command http://www.server.com/employee/1 - This command tells the web server to please provide the details of the employee whose employee number is 1.

2. Request Verbs - These describe what you want to do with the resource. A browser issues a GET verb to instruct the endpoint it wants to get data. However, there are many other verbs available including things like POST, PUT, and DELETE. So in the case of the example http://www.server.com/employee/1 , the web browser is actually issuing a GET Verb because it wants to get the details of the employee record.

3. Request Headers These are additional instructions sent with the request. These might define the type of response required or the authorization details.

4. Request Body - Data is sent with the request. Data is normally sent in the request when a POST request is made to the REST web service. In a POST call, the client actually tells the web service that it wants to add a resource to the server. Hence, the request body would have the details of the resource which is required to be added to the server.

5. Response Body This is the main body of the response. So in our example, if we were to query the web server via the request http://www.server.com/employee/1 , the web server might return an XML document with all the details of the employee in the Response Body.

6. Response Status codes These codes are the general codes which are returned along with the response from the web server. An example is the code 200 which is normally returned if there is no error when returning a response to the client.

Restful Methods

* The below diagram shows mostly all the verbs (POST, GET, PUT, and DELETE) and an example of what they would mean.

* Let's assume that we have a RESTful web service is defined at the location. http://www.server.com/employee . When the client makes any request to this web service, it can specify any of the normal HTTP verbs of GET, POST, DELETE and PUT. Below is what would happen If the respective verbs were sent by the client.

POST This would be used to create a new employee using the RESTful web service

GET - This would be used to get a list of all employee using the RESTful web service

PUT - This would be used to update all employee using the RESTful web service

DELETE - This would be used to delete all employee using the RESTful web service

Annotations

1. @PathVariable --- handles URL templates. In the above code, the path variable {name} is mapped to a String object (@PathVariable("name") String name). Therefore all of the URI such as /books/xx or /books/yy will map to the methods in the controller.

eg : URI http://www.example.com/spring_mvc/users/123

Above contains the variable userId. Assigning the value "123" to the variable yields

http://www.example.com/users/123

How to acces URI variable ?

In Spring MVC you can use the @PathVariable annotation on a method argument to bind it to the value of a URI template variable:

eg : URL --- http://host:port/web_app/users/10

```
@GetMapping(value="/users/{userId}")
public User findUser(@PathVariable int userId) {
 User user = userService.findUser(userId);
 return user;
}
```

The URI Template "/users/{userId}" specifies the variable name userId. When the controller handles this request, the value of userId is set to the value found in the appropriate part of the URI.

2. The @ResponseBody annotation is used to marshall(serialize) the return value into the HTTP response body. Spring comes with converters that convert the Java object into a format understandable for a client(text/xml/json)

where -- on the ret type of request handling methods

eg :   @Controller

```
@RequestMapping("/employee")
public class EmpController
{
  @GetMapping(....)
  public @ResponseBody Emp fetchEmpDetails(....int empId)
  {
    //get emp dtls from DB
    return e;
  }
}
OR
```

@RestController = @Controller (at the cls level) + @ResponseBody auto added on the ret type of request handling methods

3.The @RequestBody annotation, instead, unmarshalls the HTTP request body into a Java object injected in the method.

Spring Boot

Why Spring Boot ?

Spring projects require quite a bit of configuration. Some of this configuration is boilerplate code related to persistence , file upload etc . It has nothing to do with business requirements. Spring Boot is a technology that will set up much of this configuration automatically, helping you get your project up and running as quickly as possible.

* Spring Boot = Production ready spring project + embedded web server - configuration files

Benefits

1. Automatic configuration of an application uses intelligent defaults based on the classpath.

2. When creating a Spring Boot Starter project, you select the features that your application needs and Spring Boot will manage the dependencies for you.(w/o explicitely compiling pom.xml)

3. A Spring Boot application can be packaged as a JAR file. The application can be run as a standalone Java application.

4. When developing a web application, Spring Boot configures an embedded Tomcat server so that it can be run as a standalone application.

You can package the application as a WAR file and deploy it to an external servlet container too.

Spring Boot Tutorial

Spring Boot uses completely new development model to make Java Development very easy by avoiding some tedious development steps and boilerplate code and configuration.

What is Spring Boot?

Spring Boot is a Framework from   The Spring Team   to ease the bootstrapping and development of new Spring Applications.

* It provides defaults for code and annotation configuration to quick start new Spring projects within no time. It follows   Opinionated Defaults Configuration   Approach to avoid lot of boilerplate code and configuration to improve Development, Unit Test and Integration Test Process.

What is NOT Spring Boot?

Spring Boot Framework is not implemented from the scratch by The Spring Team, rather than implemented on top of existing Spring Framework (Spring IO Platform).

* It is not used for solving any new problems. It is used to solve same problems like Spring Framework.

Why Spring Boot?

* To ease the Java-based applications Development, Unit Test and Integration Test Process.

* To reduce Development, Unit Test and Integration Test time by providing some defaults.

* To increase Productivity.

Advantages of Spring Boot:

*It is very easy to develop Spring Based applications with Java

* It reduces lots of development time and increases productivity.

* It avoids writing lots of boilerplate Code, Annotations and XML Configuration.

* It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.

* It follows Opinionated Defaults Configuration Approach to reduce Developer effort

* It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.

* It provides CLI (Command Line Interface) tool to develop and test Spring Boot(Java or Groovy) Applications from command prompt very easily and quickly.

* It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle

* It provides lots of plugins to work with embedded and in-memory Databases very easily.

* Automatic configuration of an application uses intelligent defaults based on the classpath and the application context, but they can be overridden to suit the developer s requirements as needed.

* When creating a Spring Boot Starter project, you select the features that your application needs and Spring Boot will manage the dependencies for you.

* A Spring Boot application can be packaged as a JAR file. The application can be run as a standalone Java application from the command line using the java -jar command.

* When developing a web application, Spring Boot configures an embedded Tomcat server so that it can be run as a standalone application. (Tomcat is the default, but you can configure Jetty or Undertow instead.) You can package the application as a WAR file and deploy it to an external servlet container if you prefer

* Spring Boot includes many useful non-functional features (such as security and health checks) right out of the box.

* Although Spring Boot will autoconfigure the application for you, it also gives you a lot of flexibility in terms of giving you leeway to make customizations. Consequently, Spring Boot gives you the best of both worlds.

* That means Spring Boot is nothing but existing Spring Framework + Some Embedded HTTP Servers (Tomcat/Jetty etc.) XML or Annotations Configurations.

* Here minus means we don t need to write any XML Configuration and few Annotations only.

Main Goal of Spring Boot:

* The main goal of Spring Boot Framework is to reduce Development, Unit Test and Integration Test time and to ease the development of Production ready web applications very easily compared to existing Spring Framework, which really takes more time.

* To avoid XML Configuration completely

* To avoid defining more Annotation Configuration(It combined some existing Spring Framework Annotations to a simple and single Annotation)

* To avoid writing lots of import statements

* To provide some defaults to quick start new projects within no time.

* To provide Opinionated Development approach.

* By providing or avoiding these things, Spring Boot Framework reduces Development time, Developer Effort and increases productivity.

Limitation/Drawback of Spring Boot:

Spring Boot Framework has one limitation.

It is some what bit time consuming process to convert existing or legacy Spring Framework projects into Spring Boot Applications but we can convert all kinds of projects into Spring Boot Applications. It is very easy to create brand new/Greenfield Projects using Spring Boot.

Aspect Oriented Programming(AOP)

WHY

Separating cross-cutting concerns(=repeatative tasks) from the business logic .

IMPORTANT

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Dependency Injection helps you decouple your application objects from each other(eg : Controller separate from Service or DAO layers) and AOP helps you decouple cross-cutting concerns from the objects that they affect.(eg : transactional code or security related code can be kept separate from B.L)

eg scenario --

* Think of a situation Your manager tells you to do your normal development work(eg - write stock trading appln) + write down everything you do and how long it takes you.

* A better situation would be you do your normal work, but another person observes what youre doing and records it and measures how long it took.

* Even better would be if you were totally unaware of that other person and that other person was able to also observe and record , not just yours but any other peoples work and time.

*Thats separation of responsibilities.   --- This is what spring offers u thro AOP.

* It is NOT an alternative to OOP BUT it complements OOP.

* The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect.

* Aspects enable the modularization of concerns.(concern=task/responsibility) such as transaction management,logging,security --- that cut across multiple types and objects. (Such concerns are often termed crosscutting concerns in AOP jargon)

Enables the modularization of cross cutting concerns(=task)

Eg : Logging,Security,Transaction management

Similar in functionality to ---In EJB framework -- EJBObject

Struts 2 -- interceptors

Servlet -- filters.

RMI -- stubs

Hibernate --- proxy (hib frmwork -- lazy --- load or any--many associations --rets typically un-inited proxy/proxies)

AOP with Spring Framework

One of the key components of Spring Framework is the Aspect oriented programming (AOP) framework.

Like DI, AOP supports loose coupling of application objects.

The functionalities that span multiple points of an application are called cross-cutting concerns.

With AOP, applicationwide concerns(common concerns-responsibilities or cross-cutting concerns like eg - declarative transactions , security,logging,monitoring,auditing,authentication....) are decoupled from the objects to which they are applied.

Its better for application objects(service layer) to focus on the business domain problems theyre designed for and leave certain ASPECTS to be handled by someone else.

Job of AOP framework is --- Separating these cross-cutting concerns(repeatative tasks) from the core business logic

AOP is like triggers in programming languages such as Perl, .NET.

Spring AOP module provides interceptors to intercept an application, for example, when a method is executed, you can add extra functionality before or after the method execution.

Key Terms of AOP

Advice : Action(=cross cutting concern) to take either before/after or around the method (B.L) execution. eg: transactional logic(begin tx,commit,rollback)

Advice describes WHAT is to be done & WHEN it's to be done.

eg : logging. It is sure that each object will be using the logging framework to log the event happenings , by calling the log methods. So, each object will have its own code for logging. i.e the logging functionality requirement is spread across multiple objects (call this as Cross cutting Concern, since it cuts across multiple objects). Wont it be nice to have some mechanism to automatically execute the logging code, before executing the methods of several objects?

2. Join Point : Place in application WHERE advice should be applied.(i.e which B.L methods should be advised)

(Spring AOP, supports only method execution join point)

3. Pointcut : Collection of join points.

Eg : @Pointcut("execution (Vendor com.app.bank.*.*(double))")

4. Advisor Group of Advice and Pointcut into a single unit.

5. Aspect : class representing advisor(advice logic + point cut definition)-- @Asepct -- class level annotation.

6. Target : Application Object containing Business logic.(To which advice gets applied at specified join points) -- supplied by Prog

7. Proxy : Object created after applying advice to the target object(created by SC dynamically by implementing typically service layer i/f) ---consists of cross cutting concern(repeatative jobs , eg : tx management,security)

8.Weaving -- meshing(integration) cross cutting concern around B.L

(3 ways --- compile time, class loading time or spring supported --dynamic --method exec time or run time)

Examples of readymade aspects :

    Transaction management & security.

Types of Advice --appear in Aspect class

    @Before Executes only before method execution.

    @AfterReturning Executes only after method returns in successful manner

    @AfterThrowing - Executes only after method throws exception

    @After Executes always after method execution(in case of success or failure)

    @Around Most powerful, executes before & after.

Regarding pointcuts

Sometimes we have to use same Pointcut expression at multiple places, we can create an empty method with @Pointcut annotation and then use it as expression in advices.

```
    eg of PointCut annotation syntax
    @Before("execution (* com.app.bank.*.*(..))")
    @Pointcut("execution (* com.app.bank.*.*(double))")
    // point cut expression
    @Pointcut("execution (* com.app.service.*.add*(..))")
    // point cut signature -- empty method .
    public void test() {
    }
    eg of Applying point cut
```

1. @Before(value = "test()")
public void logBefore(JoinPoint p) {.........}

2. @Pointcut("within(com.app.service.*)")
   public void allMethodsPointcut(){}
@Before("allMethodsPointcut()")
   public void allServiceMethodsAdvice(){...}

3.@Before("execution(public void com.app.model..set*(*))")
 public void loggingAdvice(JoinPoint joinPoint){pre processing logic ....}

4. //Advice arguments, will be applied to bean methods with single String argument
   @Before("args(name)")
   public void logStringArguments(String name){....}

5. //Pointcut to execute on all the methods of classes in a package
   @Pointcut("within(com.app.service.*)")
   public void allMethodsPointcut(){}

6.@Pointcut("execution(* com.core.app.service.*.*(..))") // expression
private void meth1() {} // signature

7.@Pointcut("execution(* com.app.core.Student.getName(..))")
private void test() {}

****** Eg of Point cut definition.

```
@PointCut("execution (* com.aop.service.Account.add*(..))")
        public void test() {}
```

OR

```
@Before("execution (* com.aop.service.Account.add*(..))")
        public void logIt()
        {
          //logging advice code
        }
```

***** Any advice method may declare as its first parameter, a parameter of type org.aspectj.lang.JoinPoint (In around advice this is replaced by ProceedingJoinPoint, which is a subclass of JoinPoint.)

***** The org.aspectj.lang.JoinPointJoinPoint interface methods

1. Object[] getArgs() -- returns the method arguments.

2. Object getThis() --returns the proxy object

3 Object getTarget() --returns the target object

4. Signature getSignature() -- returns a description of the method that is being advised

5. String toString() -- description of the method being advised

---

Why Custom Tags?

* When JSP 2.x std actions or JSTL actions are in-sufficient to solve B.L w/o writing scriptlets --- create your own tags & use them

* For separation --- Supply B.L in custom tags & JSP can invoke the custom tags using WC

Steps for creation of Custom Tags

1. Identify the Business logic(tag execution logic) & encapsulate the same in TagHandler class.

Custom Tag Handlers can be created using javax.servlet.jsp.tagext.SimpleTag ---i/f

+ API gives u the impl. class : SimpleTagSupport .

+ As a tag dev : extend from SimpleTagSupport .

+ Must override :

    public void doTag() throws IOExc,JSPExc : represents the tag exec. logic.

+ Current objective : Generate a Hello msg from the custom tag. (1st tag will be w/o attrs & without body)

How to get the JSPWriter inst : connected to clnt browser : from inside the Tag class?

+ API : inside : doTag....

getJspContext() ---> JsPContext (env. of the JSP : impl. objs avlbl to JSP , whichever has invoked this tag)

On JSPContext : to get JSPWriter

+ JspWriter getOut()

& then invoke : out.println(dyn cont.)

2. Describe the tag to W.C : so that W.C can manage the life-cycle of the tag.

Creating : TLD (Tag library descriptor : .tld : xml syntax)

copy the template : ur web-appln's : web-inf\tlds\example.tld

eg :

```
<tag>
  <name>welcome</name> : tag suffix
  <tag-class>cust_tags.MytagHandler</tag-class> : F.Q class name
  <body-content>empty</body-content> : no body supported by the tag + no attrs.
 </tag>
```

3. Invoking the custom tag from the JSP

3.1 Import the TLD : which contains the description of custom tags

```
<%@ taglib uri="URI of the TLD" prefix="tag prefix" %>
```

3.2 Invoke the tag

```
eg : <my:hello/>
```

Life-cycle of the custom Tag

+ Ref to JEE -- apidoc---javax.servlet.jsp.tagext.SimpleTag ---i/f

+ SimpleTag : i/f ----contains the desc of the life cycle.

1. W.C will invoke tag life cycle -- when JSP invoke tag.(eg : <ex:hello/>

2.WC locates TLD (using taglib directive)

3. From TLD --- searches for matching tag suffix (under tag name)

4. From tag name -- gets tag class --- locates/loads/instantiates tag class in WC's mem.

5. WC invokes setJspContext(JspContext ctx) --- to pass the entire JSP page env(PageContext obj containing all impl objs) to the tag handler class.(mandatory)

6.WC will invoke the setters for attributes (optional)

7. WC will invoke setJspBody(JspFragment jspf) iff body content is non-empty.(optional)

8.Finally WC invokes --- doTag()

9. Upon returninng from doTag() --- WC GCs T.H class inst.

+ body-content = empty =>W.C will skip handling of body-content.

+ body-content = scriptless => WC will invoke setJspBody(JspFragment jspf) to pass tag body content to T.H class.

+ allowed content types ---- plain text/HTML/std action/custom action

How to retrieve & invoke body content from T.H class?

From doTag ---

1. get JspFragment (API --- getJspBody())

2. To invoke JspFragment ----use API

public abstract void invoke(Writer w) throws JspException,IOException

if w=null --- o/p(contents) will be auto. sent to clnt browser.

Why Filters ?

1. Provides re-usability.

Meaning --- They provide the ability to encapsulate recurring tasks(=cross cutting concerns) in reusable units.

They provide clear cut separation between B.L & cross cutting concerns.

2. Can dynamically intercept req/resp to dyn or static content

What is Filter?

Dynamic web component just like servlet or

JSP. Resides within web-appln.(WC)

Filter life-cycle managed by WC

* It performs filtering tasks on either the request to a resource (a servlet,JSP or static content), or on the response from a resource, or both.

* It can dynamically intercepts requests and responses .

Usage of Filters

1. Authentication Filters
2. Logging Filters
3. Image conversion Filters
4. Data compression Filters

5. Encryption Filters

6. Session Check filter

Detailed Description ---

* Filters typically do not themselves create responses, but instead provide universal functions that can be "attached" to any type of servlet or JSP page.

* Filters are important for a number of reasons. First, they provide the ability to encapsulate recurring tasks in reusable units. Organized developers are constantly on the lookout for ways to modularize their code. Modular code is more manageable and documentable, is easier to debug, and if done well, can be reused in another setting.

* Second, filters can be used to transform the response from a servlet or a JSP page. A common task for the web application is to format data sent back to the client. Increasingly the clients require formats (for example, WML) other than just HTML. To accommodate these clients, there is usually a strong component of transformation or filtering in a fully featured web application. Many servlet and JSP containers have introduced proprietary filter mechanisms, resulting in a gain for the developer that deploys on that container, but reducing the reusability of such code. With the introduction of filters as part of the Java Servlet specification, developers now have the opportunity to write reusable transformation components that are portable across containers.

Java Annotation

* Annotation is a tag that represents the metadata.

* Meta data meant for java tools

* It can be present at different levels.

class, interface, methods or fields

* It indicates some additional information that can be used by java compiler and JVM.

2 Types

1. Built-In Annotations 2. Custom Annotations

There are several built-in annoations.

Built in annotations meant for java code

Eg: @Override

@SuppressWarnings

@Deprecated

@FunctionalInterface

Built-In Annotations that are applied to other annotations (to supply more information of the annotation itself)

1. @Target -- target of the annotation

Can have typically these values --CONSTRUCTOR,LOCAL_VARIABLE,FIELD,METHOD,PARAMETER etc.

2. @Retention -- Specifies Retention policy of the annotation.

Can have typically these values

SOURCE : - Annotations are to be discarded by the compiler (i.e don't appear in .class file).

eg : @Override

CLASS --Annotations are kept in the class file by the compiler but not retained by the VM at run time. This is the default behavior.

RUNTIME -- Annotations kept in the class file by the compiler and retained by the JVM at run time, so as to read using reflection

eg : @Deprecated,@FunctionalInterface

@Inherited --

Indicates that an annotation type is automatically inherited.

When you apply this annotation to any other annotation i.e. @MyCustomAnnotation; and @MyCustomAnnotation is applied of any class MySuperClass then @MyCustomAnnotation will be available to all sub classes of MySuperClass as well.

@Documented

This annotation indicates that new annotation should be included into java documents generated by java document generator tools.

Meaning of annotations

@Override annotation assures that the subclass or implementation class method is overriding/implementing the parent class / interface method. If it is not so, compile time error occurs.

@SuppressWarnings annotation: is used to suppress warnings issued by the compiler.

eg : @SuppressWarnings("serial")

or

@SuppressWarnings("unchecked")

@Deprecated annoation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.

Custom Annotations

* Java Custom Annotation

+ Java Custom annotations or Java User-defined annotations are easy to create and use. The @interface element is used to declare an annotation. For example:

@interface MyAnnotation{}

+ Here, MyAnnotation is the custom annotation name.

+ Points to remember for java custom annotation signature (methods in i/f)

+ Method should not have any throws clauses

+ Method should return one of the following: primitive data types, String, Class, enum or array of these data types.

+ Method should not have any parameter.

+ We should attach @ just before interface keyword to define annotation.

+ It may assign a default value to the method.

Types of Annotation

There are three types of annotations.

+ Marker Annotation + Single-Value Annotation + Multi-Value Annotation

1) Marker Annotation

An annotation that has no method, is called marker annotation. For example:

@interface MyAnnotation{}

eg : @Override , @Deprecated are marker annotations.

2) Single-Value Annotation

An annotation that has one method, is called single-value annotation. For example:

@interface MyAnnotation{

int value();

}
We can provide the default value also. For example:
@interface MyAnnotation{
int value() default 0;
}
How to apply Single-Value Annotation
eg :
@MyAnnotation(value=10)
The value can be anything.
3) Mulit-Value Annotation
An annotation that has more than one method, is called Multi-Value annotation.
For example:
@interface MyAnnotation{
 int value1();
 String value2();
 String value3();
}
We can provide the default value also. For example:
@interface MyAnnotation{
int value1() default 1;
String value2() default "";
String value3() default "xyz";
}
How to apply Multi-Value Annotation
@MyAnnotation(value1=10,value2="abc",value3="xyz")
Built-in Annotations used in custom annotations in java
@Target   @Retention   @Inherited   @Documented
Example to specify annoation for a class (i.e before a class definition)
@Target(ElementType.TYPE)
@interface MyAnnotation{
int value1();
String value2();
}
Example to specify annoation for a class, methods or fields

@Target({ElementType.TYPE, ElementType.FIELD, ElementType.METHOD})
@interface MyAnnotation{
int value1();
String value2();

}

@Retention annotation is used to specify to what level annotation will be available.

+ RetentionPolicy	Availability
+ RetentionPolicy.SOURCE	refers to the source code, discarded during compilation. It will not be available in the compiled class.
+ RetentionPolicy.CLASS	refers to the .class file, available to java compiler but not to JVM . It is included in the class file.
+ RetentionPolicy.RUNTIME	refers to the runtime, available to java compiler and JVM .

Example to specify the RetentionPolicy

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@interface MyAnnotation{
int value1();
String value2();
}
```

How built-in annotations are used in real scenario?

In real scenario, java programmer only needs to apply annotation. You don't need to create and access annotation. Creating and Accessing annotation is performed by the implementation provider. On behalf of the annotation, java compiler or JVM performs some additional operations.

Use Cases for Annotations

Annotations are very powerful and Frameworks like spring and Hibernate use Annotations very extensively for logging and validations. Annotations can be used in places where marker interfaces are used. Marker interfaces are for the complete class but you can define annotation which could be used on individual methods for example whether a certain method is exposed as service method or not.

What is JNDI ?

+ Java Naming and Directory Interface (JNDI)

+ J2EE API --- that provides naming and directory functionality to applications written using the Java. It is independent of any specific directory service implementation.

+ Thus variety of directories(new, emerging, and already deployed) can be accessed in a common way.

What is its basic use?

JNDI allows distributed applications to look up services in an abstract, resource-independent way.

When it is used?

The most common use case is to set up a database connection pool on a Java EE application server. Any application that's deployed on that server can gain access to the connections they need using the JNDI name java:comp/env/myPool without having to know the details about the connection.

Advantage

+ Applications don't have to change as they migrate between environments.

So basically you create objects and register them on the directory services which you can later do lookup and execute operation on.

JNDI Overview

JNDI is an API specified in Java technology that provides naming and directory functionality to applications written in the Java programming language. It is designed especially for the Java platform using Java's object model. Using JNDI, applications based on Java technology can store and retrieve named Java objects of any type. In addition, JNDI provides methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes.

+ JNDI is also defined independent of any specific naming or directory service implementation. It enables applications to access different, possibly multiple, naming and directory services using a common API. Different naming and directory service providers can be plugged in seamlessly behind this common API. This enables Java technology-based applications to take advantage of information in a variety of existing naming and directory services, such as LDAP, NDS, DNS, and NIS(YP), as well as enabling the applications to coexist with legacy software and systems.

What is Maven ?

Build automation tool for overall project management.

It helps in

1. checking a build status
2. generating reports (basically javadocs)
3. setting up the automated build process and monitors the same.

Why Maven ?

It helps in source code compilation, distribution(JAR/WAR--web app archive), documentation, collaboration with different teams .

Maven tries to describe

1. How a software is built.

2. The dependencies, plug-ins & profiles that the project is associated in a standalone or a distributed environment.

Ant disadvantages

1. While using ant , project structure had to be defined in build.xml. Maven has a convention to place source code, compiled code etc. So no need to provide information about the project structure in pom.xml file.

2. Maven is declarative, everything you define in the pom.xml file.

No such support in ant.

3. There is no life cycle in Ant, where as life cycle exists in Maven.

Maven advantages

4. Managing dependencies

5. Uses Convention over configuration - configuration is very minimal , in pom.xml

6. Multiple/Repeated builds can be achieved.

7. Plugin management.

8. Testing - ability to run JUnit and other integration test suites.

What is POM? (Project Object Model)

It is the core element of any maven project.

Any maven project consists of one configuration file called pom.xml.

Location --In the root directory of any maven project.

It contains the details of the build life cycle of a project.

Contents

+ Dependencies used in the projects (Jar files)
+ Plugins used
+ Project version
+ Developers involved in the project
+ Build profiles etc.

Maven reads the pom.xml file, then executes the goal. eg : clean , install