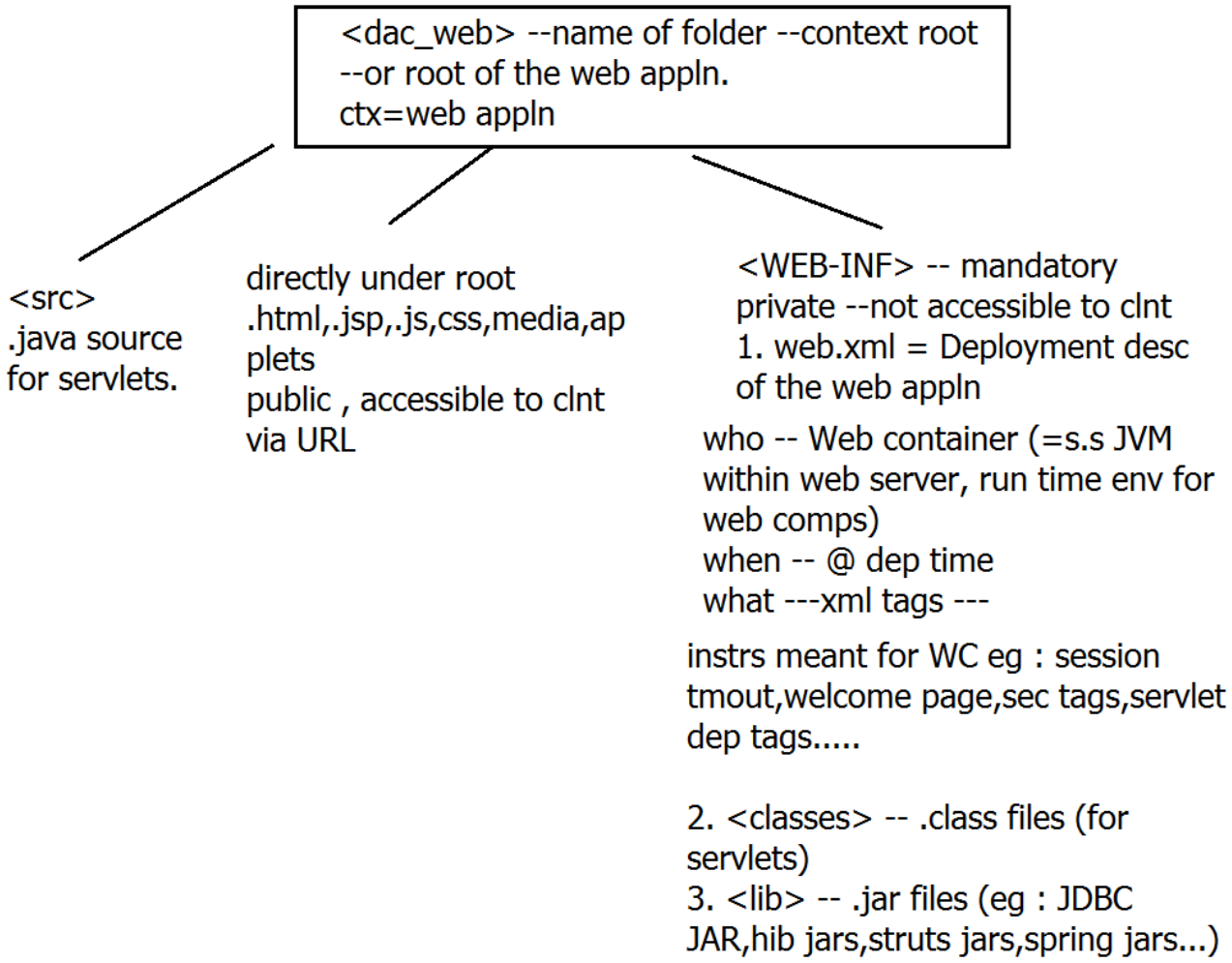
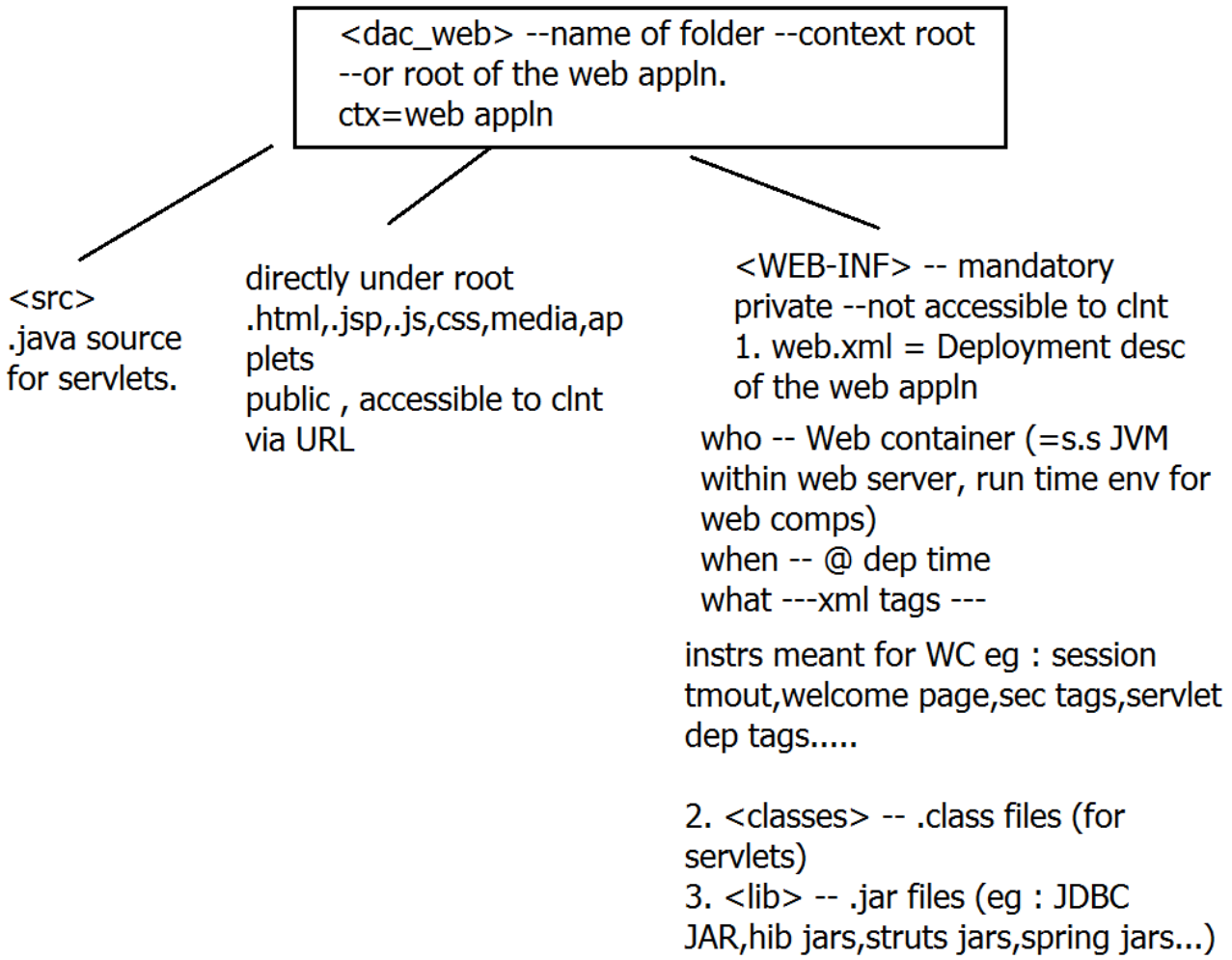
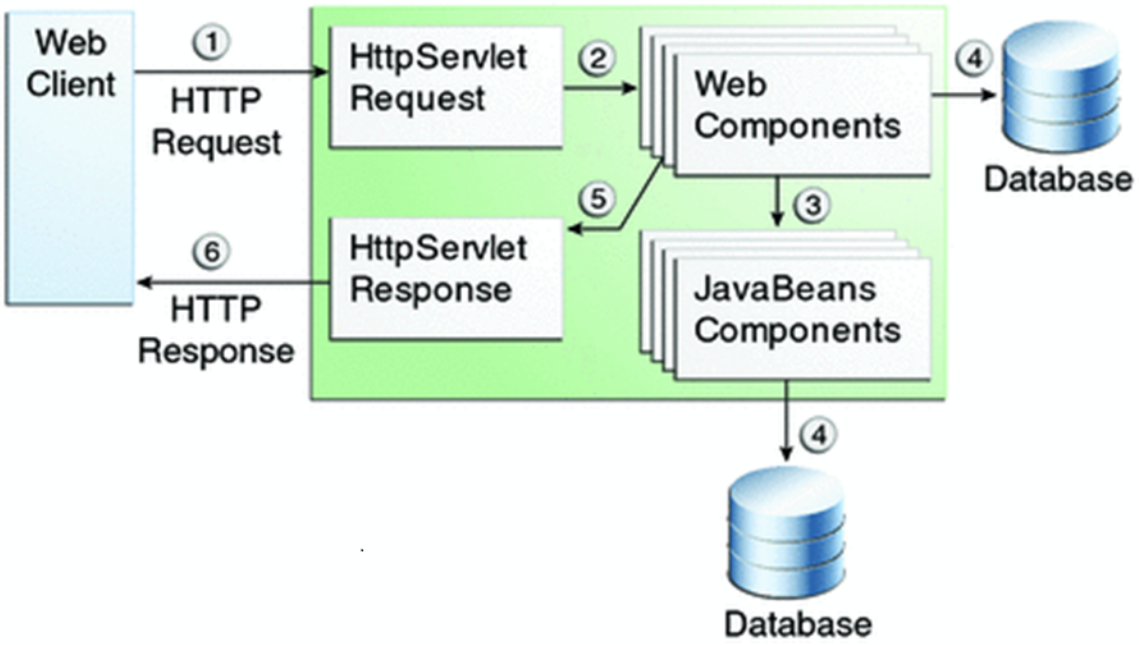
J2EE compliant web application folder structure
Tomcat ---<tomcat>/webapps --hot dep folder

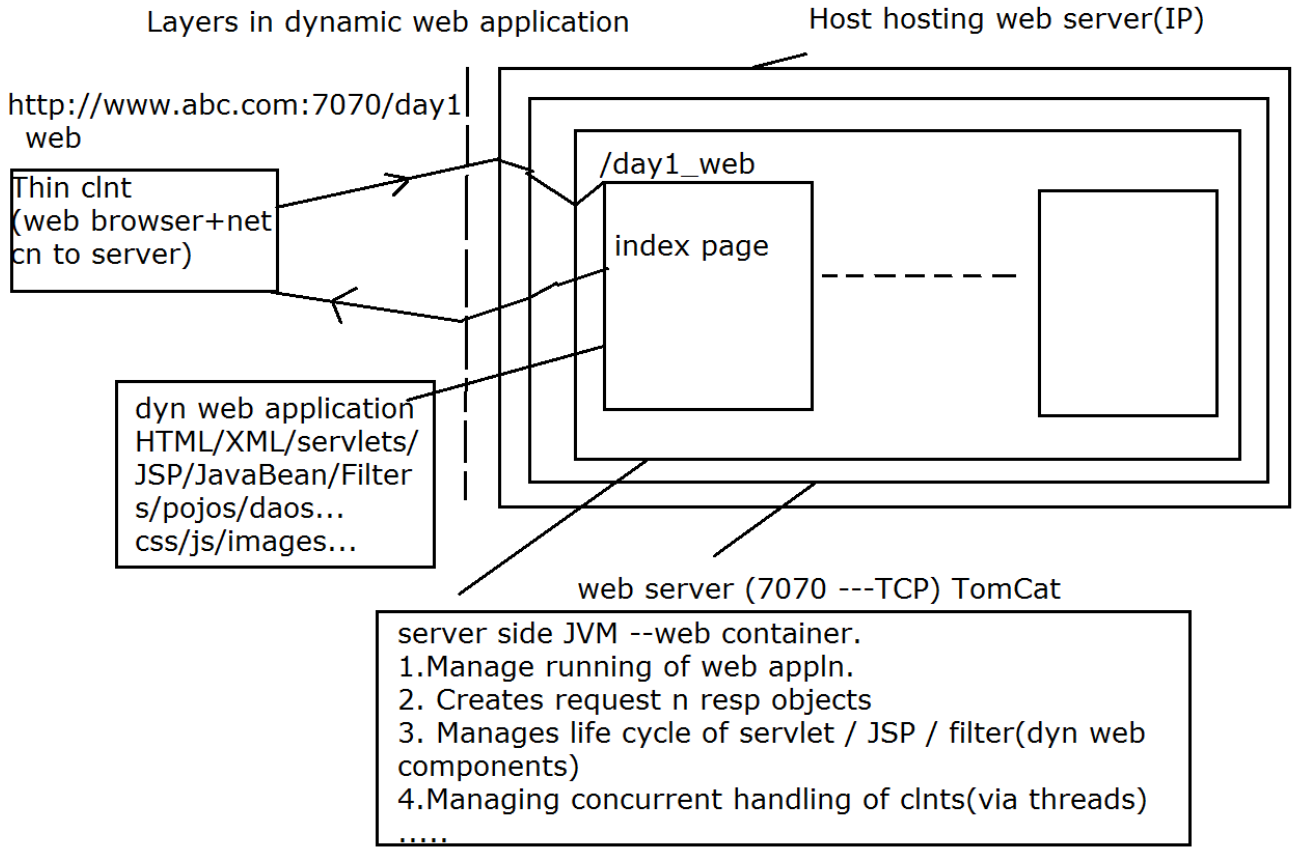


J2EE compliant web application folder structure
Tomcat ---<tomcat>/webapps --hot dep folder

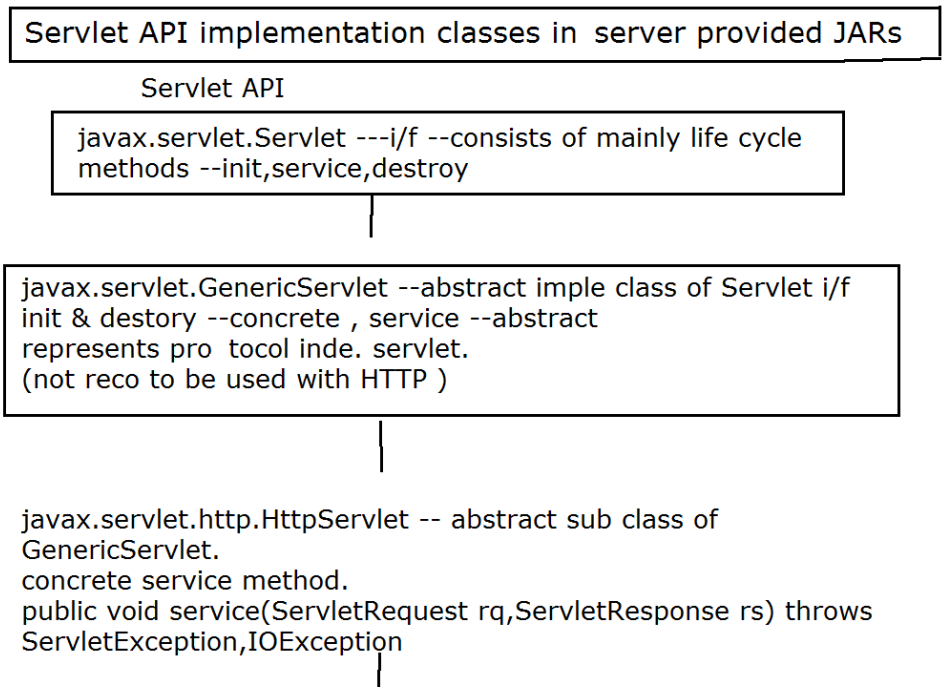


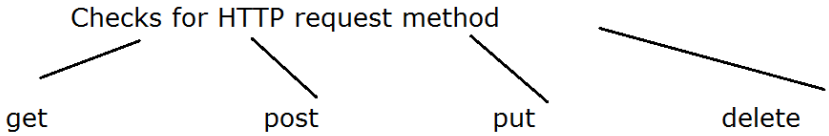
Web Request Handling





HTTP Resp Packet
Resp status code(200) | Header/s | response data





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 ---

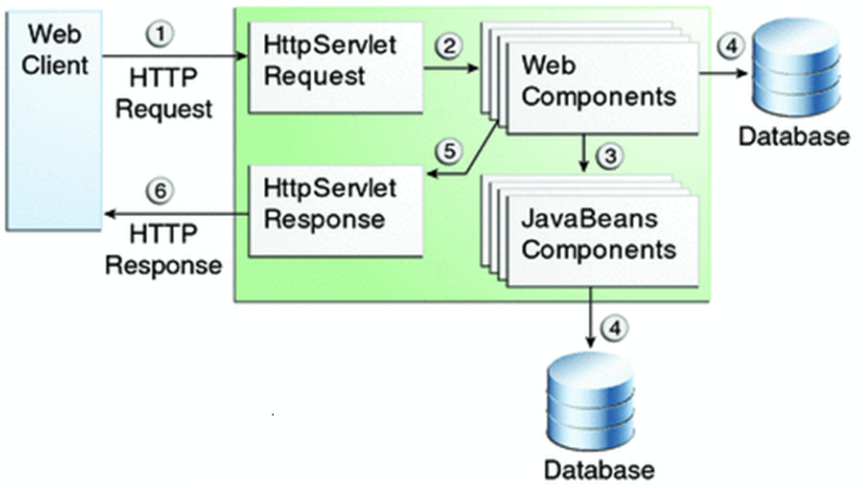
1. set response content type
API of HttpServletResponse
public void setContentType(String type)
eg : rs.setContentType("text/html");
2. To send response from servlet --- clnt browser , attach data strm.
API of HttpServletResponse
public PrintWriter getWriter()
PW --char , buf , o/p strm conn from server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

Web Request Handling



7

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

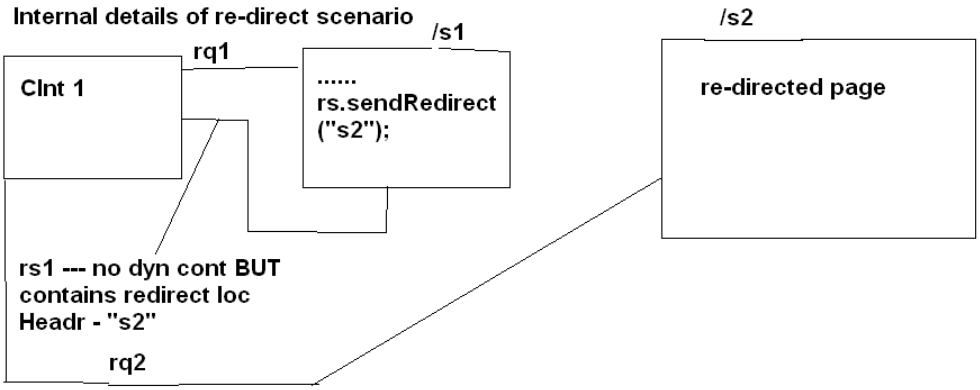
Client (as a person) is NOT involved BUT clnt browser is involved. Re-direct scenarion. API of HttpServletResponse void sendRedirect(String redirectURL)

Server Pull

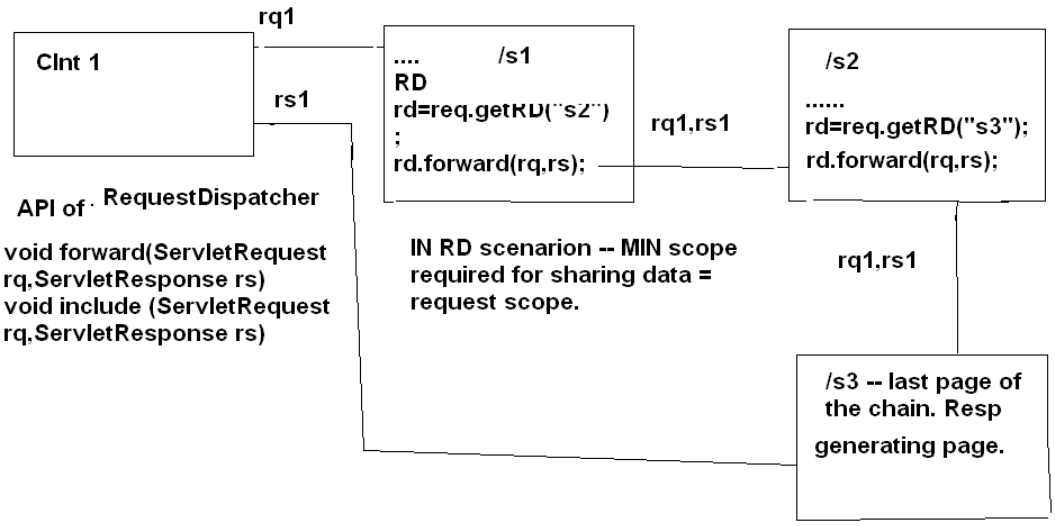
In this tech. user is navigated to next page in same request.

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.

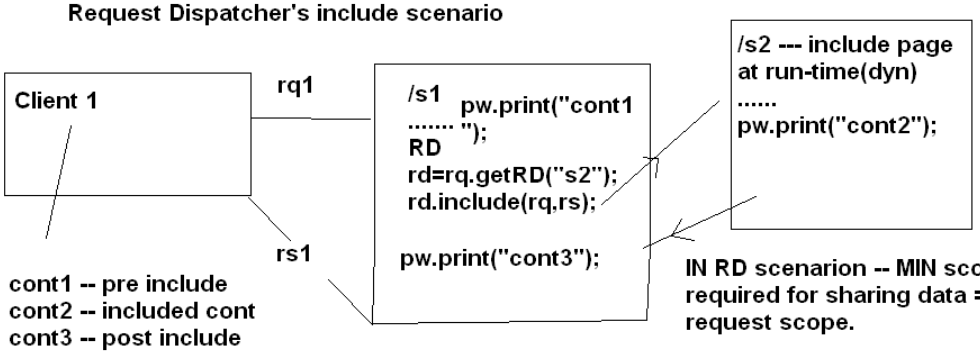


API of RequestDispatcher

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

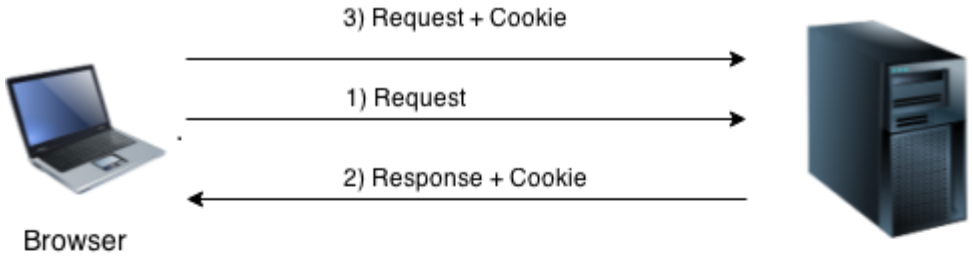
IN RD scenarion -- MIN scope required for sharing data = request scope.

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.



1. Create a Cookie
API --javax.servlet.http.Cookie(String nm,String val)
eg : Cookie c1=new Cookie("user_dtls",u.toString());
2. Add cookie/s to resp header
API HttpServletResponse
public void addCookie(Cookie c1);

3. Client browser chks privacy settings
Cookies disabled -- can't remember clnt
Cookies enabled --- browser chks cookie age (expiry)
-1 -- default -- cookie is stored in browser cache.
0 -- delete the cookie
> 0 --save it on clnt's hard disk (persistent)
4. When clnt send a request to same web appln
Clnt browser --sends --req +cookies(hdr)

5. How to read cookie data ?
API of HttpServletRequest
Cookie[] getCookies()

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

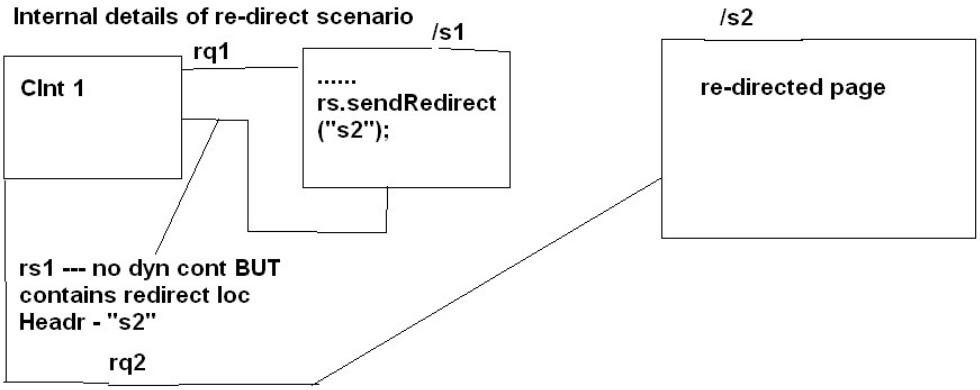
Client (as a person) is NOT involved BUT clnt browser is involved. Re-direct scenarion. API of HttpServletResponse void sendRedirect(String redirectURL)

Server Pull

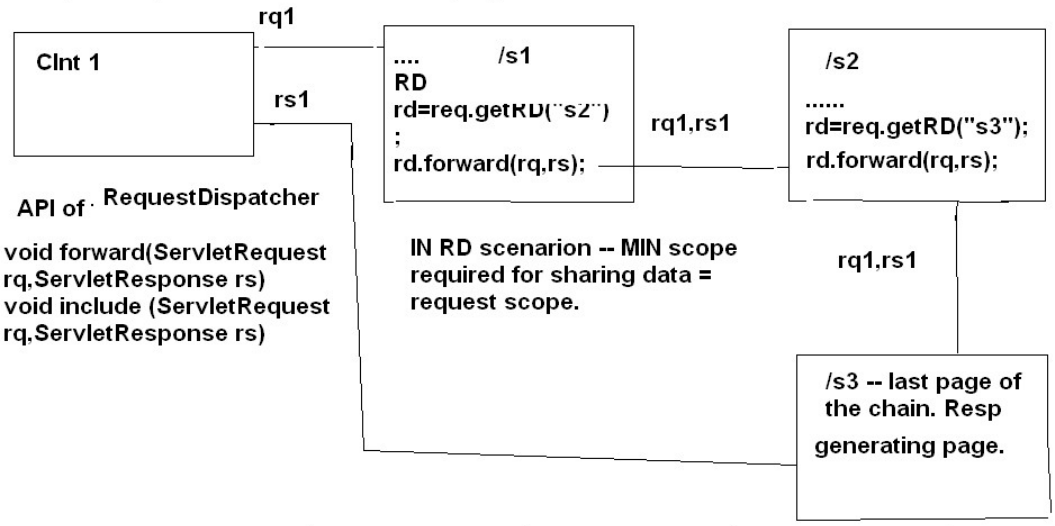
In this tech. user is navigated to next page in same request.

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.

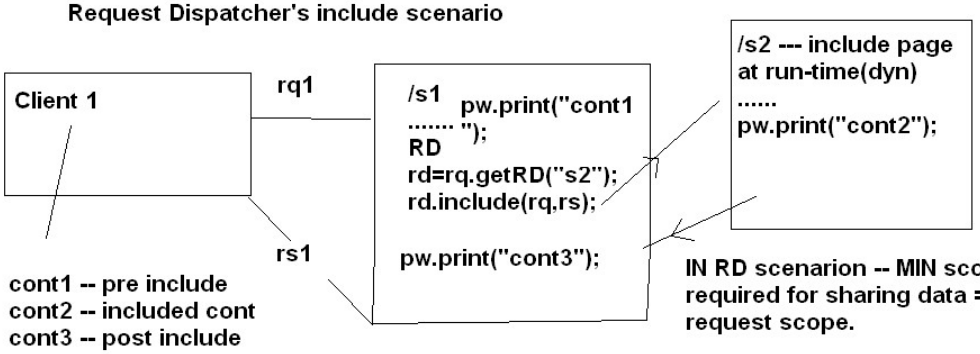


API of RequestDispatcher

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.

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.

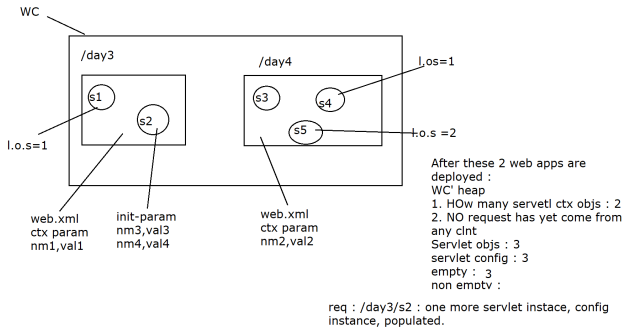
WC has already created a thread pool @ WC startup (Using Executor Framework)
Any time client sends HTTP request to the Servlet , WC simply pools out ANY thread from the pool.

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.



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

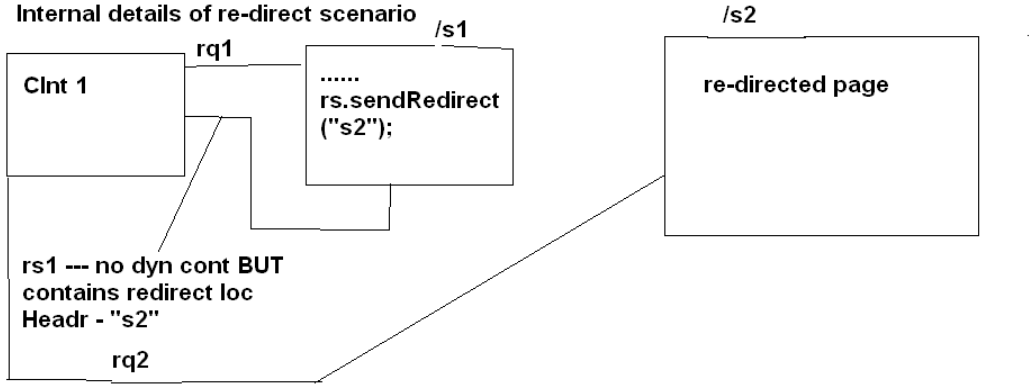
Client (as a person) is NOT involved BUT clnt browser is involved.
Re-direct scenarion.
API of HttpServletResponse
void sendRedirect(String redirectURL)

Server Pull

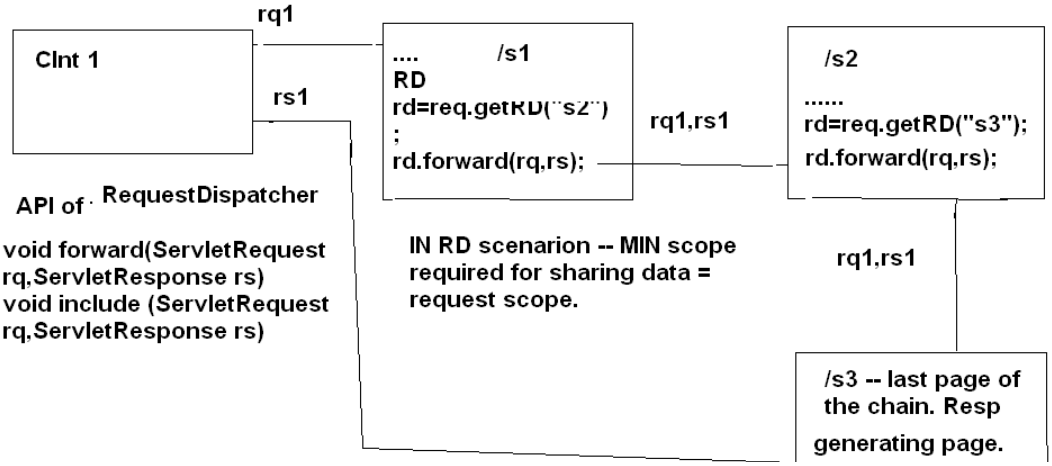
In this tech. user is navigated to next page in same request.

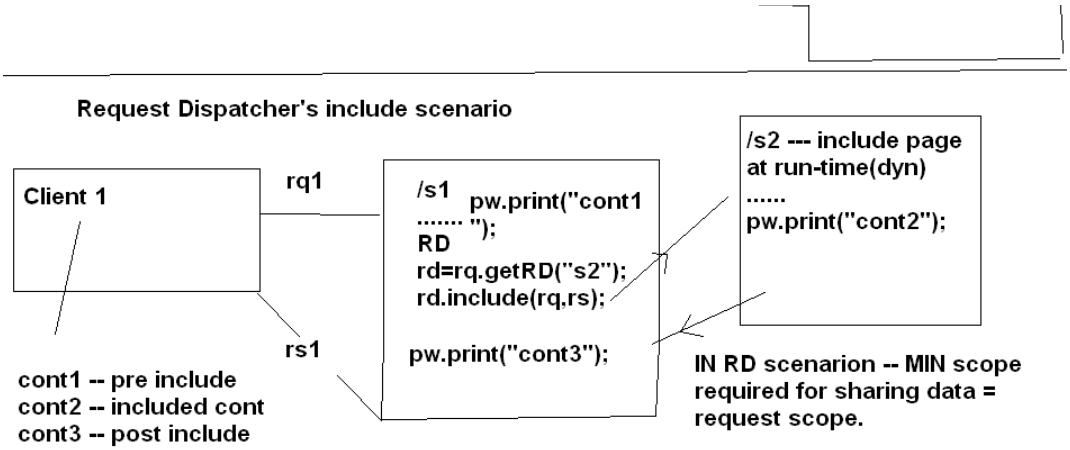
Resource chaining or request dispatching technique. --- NO round trip delay is involved since entire navigation takes place in server space.
API of ServletRequest
1. Get Request dispatcher object to chain resources(JSP,Servlet)
RequestDispatcher getRequestDispatcher(String dispatcherURL) 2. Use either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

Internal details of re-direct scenario



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





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

sendRedirect Vs Request Dispatcher forward
common --both of above are page navigation techniques.

1. Navigates the user to next page in NEXT request

2.Client browser is involved (round trip delay involved)

3. Min scope required for sharing attributes = session scope

4. URL changes to the next(redirected) page.

5. Usage --
To navigate client outside current web application.
Double submit guard.

1. Navigates the user to next page in SAME request.

2. Client browser is NOT involved (no round trip delay)

3. Min scope required for sharing attributes =request scope

4.URL seen by the client is that of 1st page(in the chain of resources)

5. Usage --
In MVC applications, to navigate the client from controller to the view layer.
(Used as default navigation by all major MVC frmworks)

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.

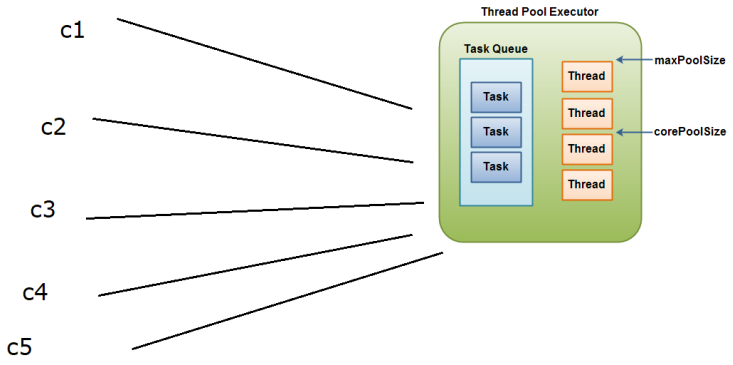
WC has already created a thread pool @ WC startup (Using Executor Framework)
Any time client sends HTTP request to the Servlet , WC simply pools out ANY thread from the pool.

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.



WC uses : Executor framework for managing concurrency.
It's java.util.concurrent
i/f Executor
sub i/f ExecutorService

Various impl classes :
Thread pool based executor
Created by WC @ server startup (server.xml --specific to Tomcat)
thread pool size , queue size.

eg : core pool size 2
max pool size 4

'j2ee'

Java Bean ——— **Attribute**

id / name ——— **attribute name**

instance ——— **attribute value**

scope=page|request|session|application ——— **scope=page|request|session|application**

Via JSP using JB related actions

via java API --- setAttribute & getAttribute.

Association between JSP Using JBs actions & what gets invoked on JB class?

jsp:useBean ——— **invokes JB constr once per scope.**

jsp:setProperty ——— **invokes MATCHING setters**

jsp:getProperty or EL syntax ——— **Invokes matching getters.**

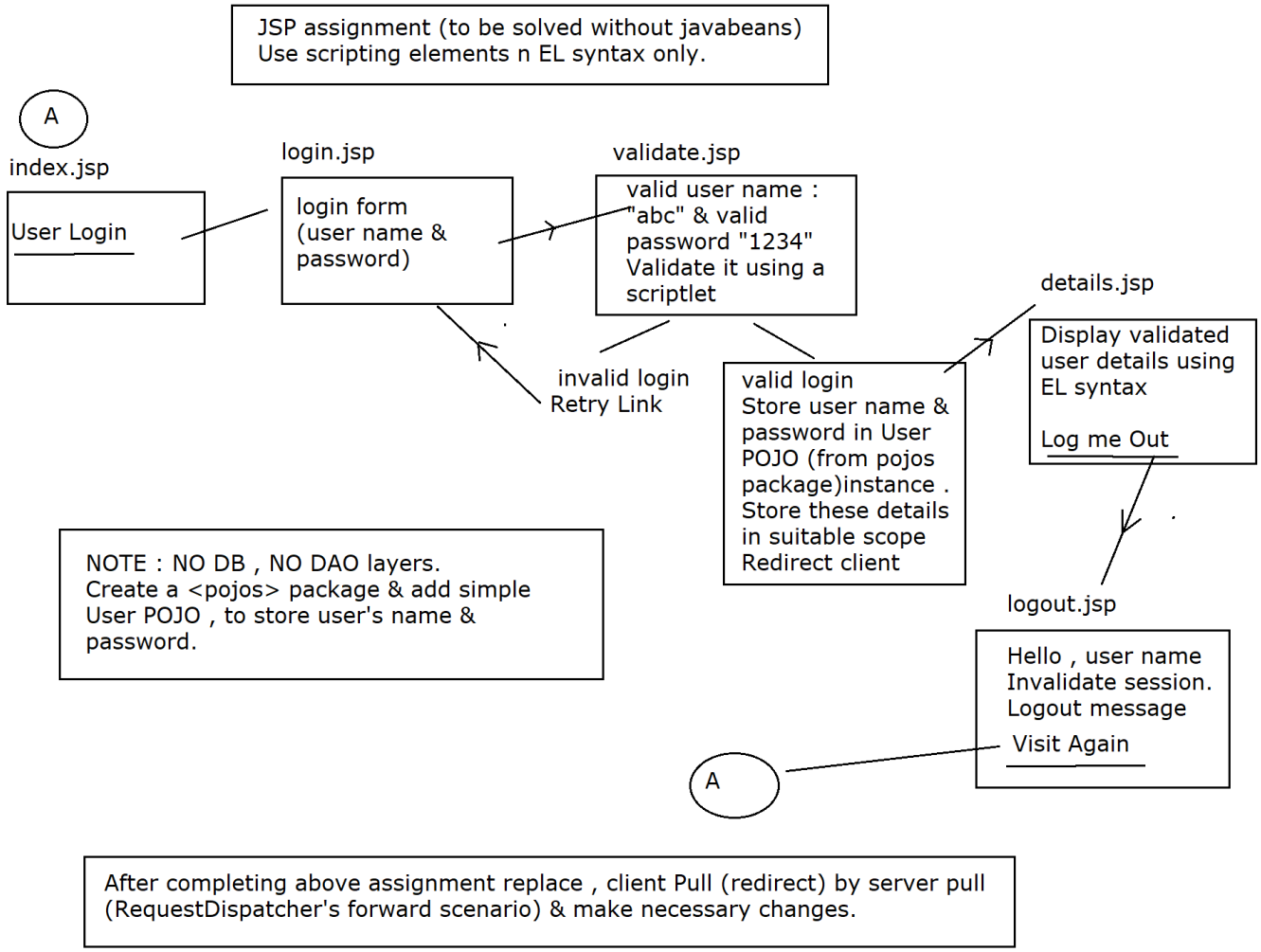
Business logic methods via scriptlets or pseudo-getters ——— **Invokes matching methods from JB class.**

JSP API --- javax.servlet.jsp
Imple classes <tomcat>/lib/jsp-api.jar (contains specifications) actual imple classes provided by Tomcat are bundled into jasper.jar

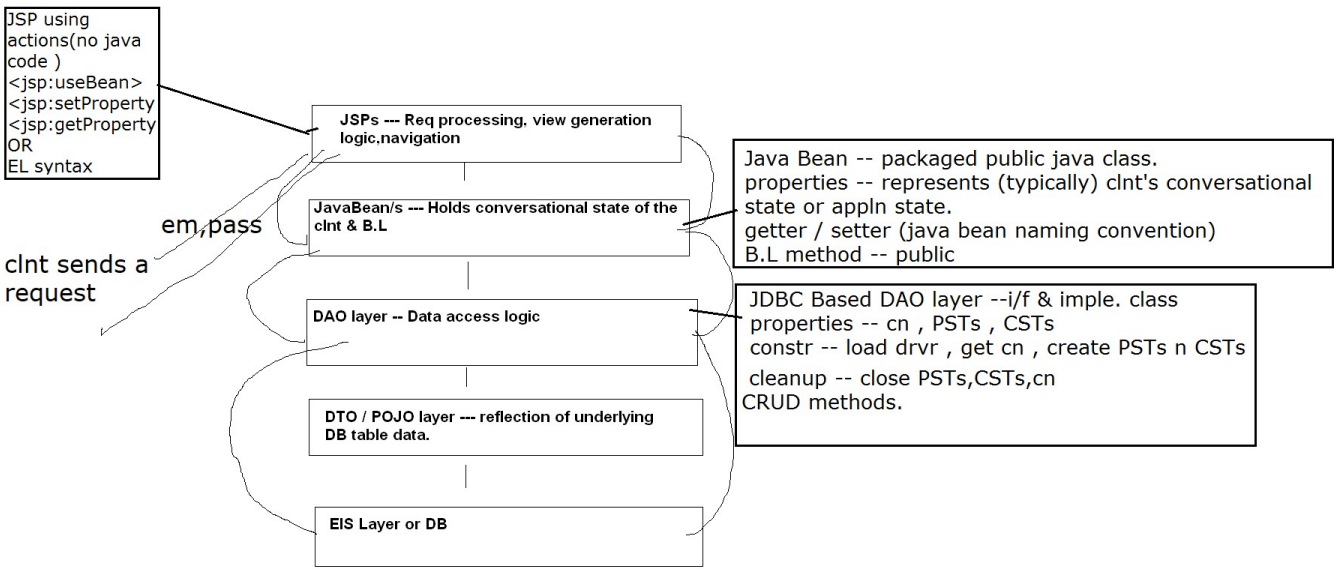
javax.servlet.Servlet ---starting point

javax.servlet.jsp.JspPage --i/f
public void jspInit()
public void jspDestroy()

javax.servlet.jsp.HttpJspPage --sub i/f
public void _jspService(HttpServletRequest rq,HttpServletResponse rs) throws SE,IOExc
---Should not be overridden by JSP page author.



Typical layers in JSP using JavaBean scenario



JSP life cycle

any clnt sending rq1 to JSP page

test.jsp

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.

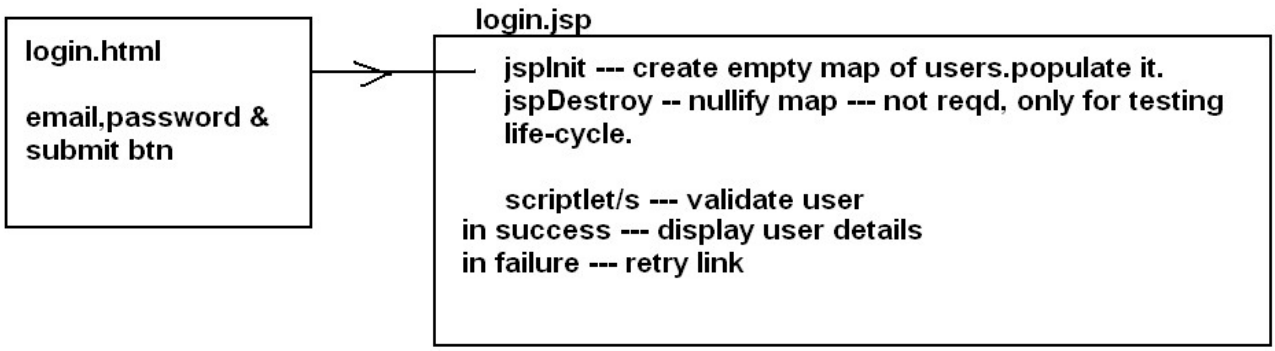
Compilation phase --- o/p --page translation class (.class) called by WC --- once per life-cycle

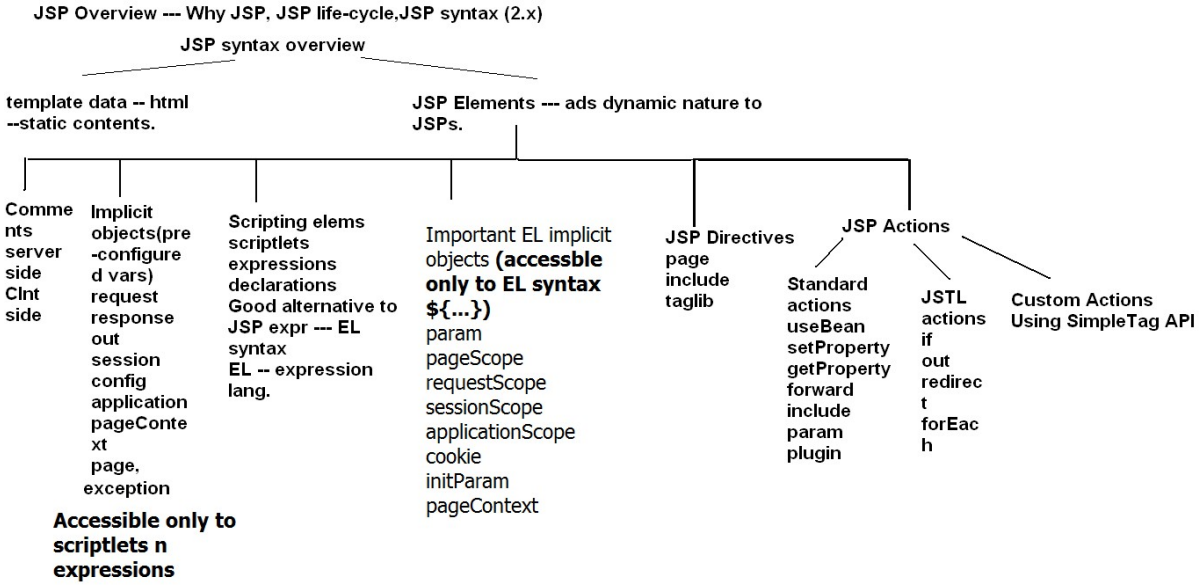
locate .class (from server specific folder) , loaded using Class.forName(...) & instantiated using def. constructor.

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.


login.html
email,password &
submit btn
login.jsp
jspInit --- create empty map of users.populate it.
jspDestroy -- nullify map --- not reqd, only for testing life-cycle.
scriptlet/s --- validate user
in success --- display user details
in failure --- retry link


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

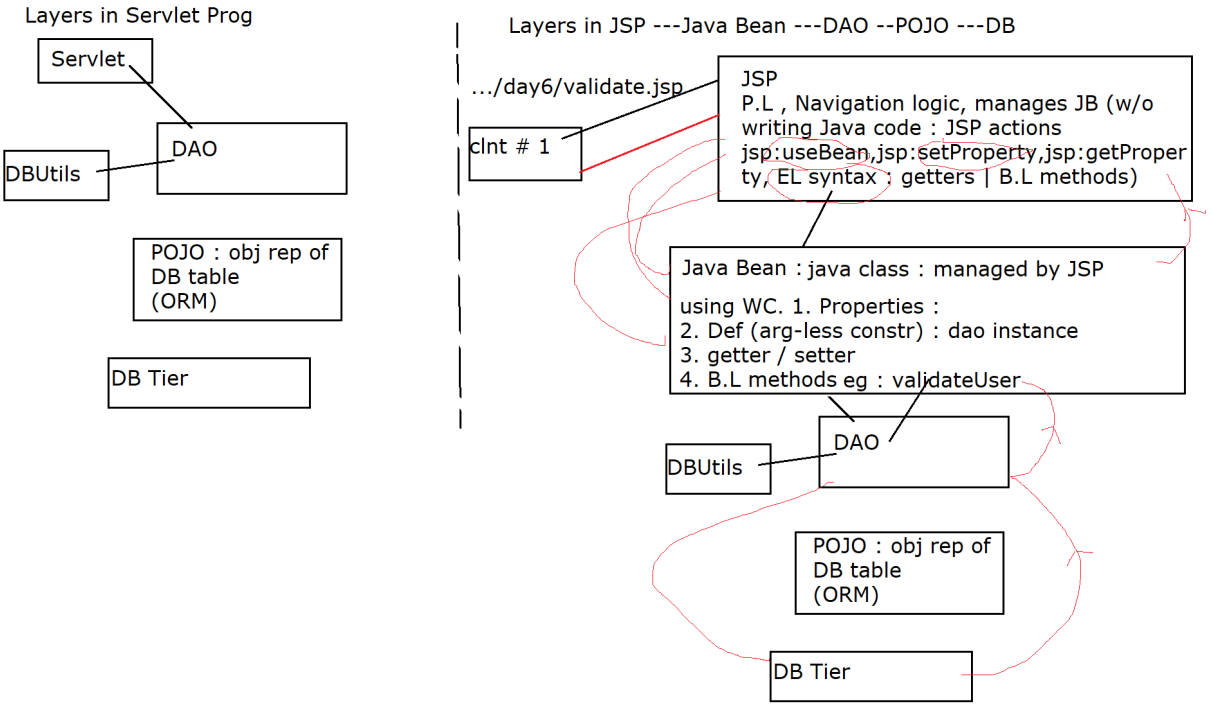
<jsp:useBean id="shop" class="beans.BookShopBean" scope="session"/>
WC invokes --- WC checks if specified bean exists under specified scope ?
session.getAttribute("shop")

null
1. Create an attribute loc(web-inf/classes)-load(Class.forName ("beans.BookShoBean")-inst def constr.
BookShopBean ref=new BookShopBean();

2. Save it under specified scope.
session.setAttribute("shop",ref);

not-null

Tag rets.



<jsp:useBean id="voter" class="beans.VoterBean"
scope="session"/>

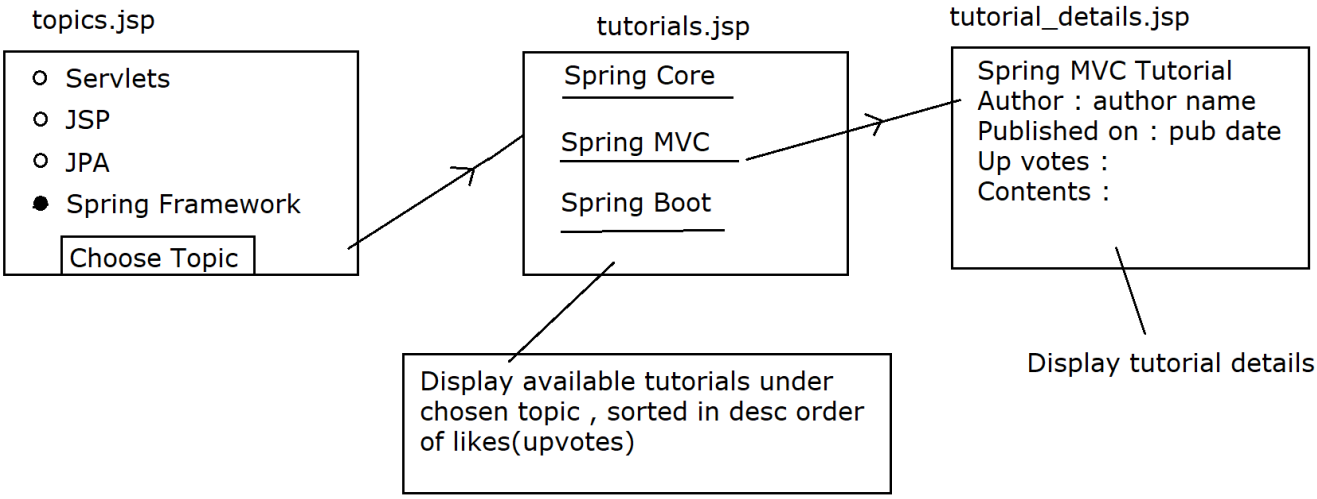
WC checks if this bean(=attribute) exists already in the given scope ?
session.getAttribute("voter")

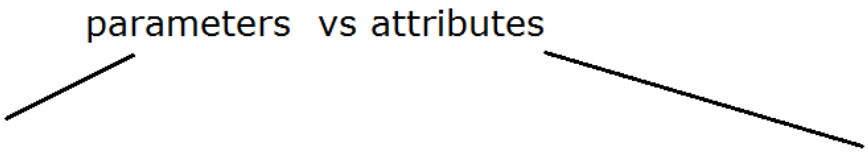
null => bean doesn't exist
WC
session.setAttribute("voter",new VoterBean());

not null => bean alrdy exists

<jsp:useBean> tag rets

JSP--JavaBean --DAO--POJO --DB based assignment





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}

JSP actions for managin Javabean
1. <jsp:useBean id="user" class="beans.BankUserBean"
scope="session" />

WC checks if specified bean exists under specified scope?
session.getAttribute("user")

null

1. locate(WEB-INF/classes)
loads (Class.forName)
inst : def constr
eg : BankUserBean ref=new BankUserBean();

2. WC adds this bean instance under specified scope
session.setAttribute("user",ref);

not null

bean instance is alredy created by WC
(in earlier req coming from SAME clnt)

If there are 10 clnts sending 5 requests each , how many times WC will call Bean's constr :

<jsp:useBean id="test" class="beans.TestBean" />
how many times WC will call Bean's constr :

<jsp:useBean id="test" class="beans.TestBean" scope="request"/>
how many times WC will call Bean's constr :

<jsp:useBean id="test" class="beans.TestBean" scope="application"/>
how many times WC will call Bean's constr :

Centralized error handling in JSP

web site (dyn web app)

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

JSP

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

JSP

serlvet

In case of any errors : Tell WC to
forward the client to common err page

err_handler.jsp

How to tell WC : about error page
<%@ page isErrorPage="true" %>

Typical Layers in Hibernate Based Java SE application

hibernate n JDBC JARs

main(...) based Tester -- For UI.
Sets up Hibernate frmwork(creates SessionFactory)

hibernnate.cfg.xml

HibernateUtils

1. StandardServiceRegistry via its builder

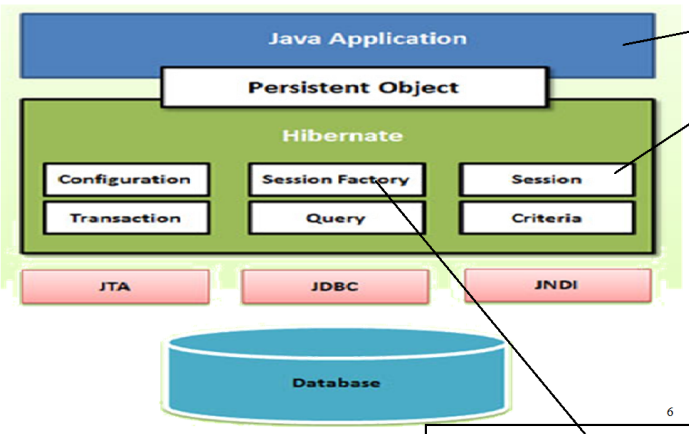
2. Create Metadata from MetadataSources & build singleton n immutable SessionFactory

Hibernate Based DAO layer -- uses org.hibernate API
Main i/f org.hibernate.Session --has CRUD methods.
Wraps JDBC connection, so that developer doesn't have to go to JDBC layer

Hibernate Based POJO / Entity --annotated POJO class .
Using annotations from JPA
javax.persistence package

DB / EIS Tier

Hibernate Architecture



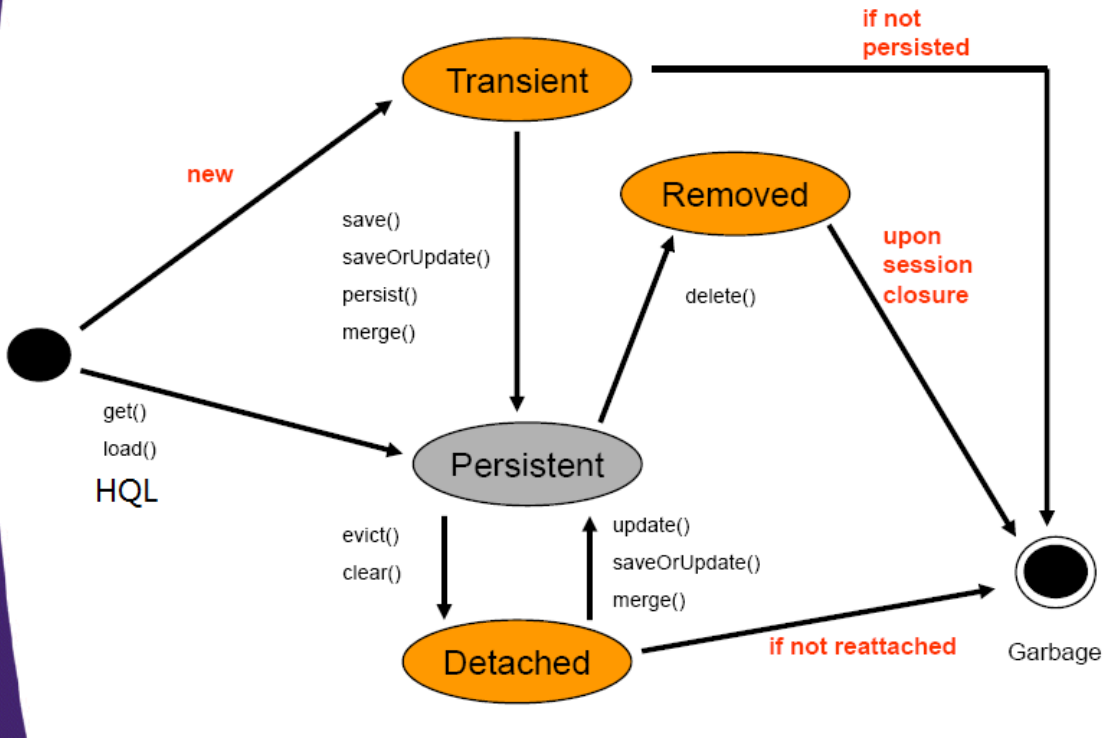
Core Java (standalone/distributed app : desktop based) or web app

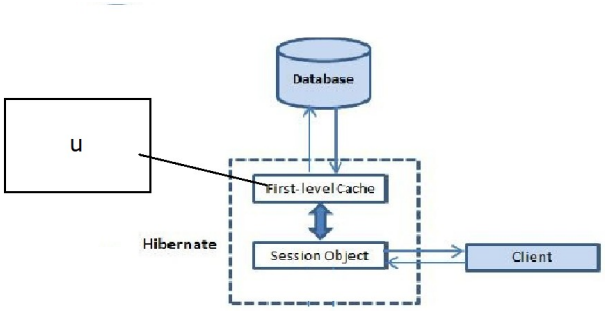
org.hibernate.Session i/f
1. Persistence manager
It has methods :
save : insert
update : update
get : select with PK
delete : delete .....
JPQL : select with different options...
2. Represents L1 (session level cache)
3. Represents a wrapper around DB connection , pooled from connection pool.
session objs created / request, light weight.unsafe

org.hibernate.SessionFactory : i/f
Provider of hibernate session objects.
singleton(single instance created in entire app / DB), immutable , inherently thrd safe) , creating SF object : time consuming (heavy weight)
--typically should be done @ app starting time/dep time

SF is associted with L2 cache. Must be explicitly enabled.

Hibernate Lifecycle



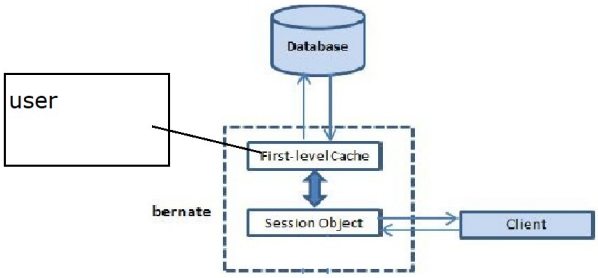


DAO :
u : TRANSIENT (exists only in heap)

Session hs=getSf().openSession();
begin tx
try
hs.save(u);//u : PERSISTENT : part of L1 cache
not yet part of DB

tx.commit();//hibernate performs auto dirty checking : insert query
(transient ----save/persist/merge/saveOrUpdate---
persistent : gains DB identity upon commit : insert)

finally : hs.close() :
L1 cache is destroyed , db cn rets to pool
u :state POJO : transient / persistent
DETACHED : from L1 cache.



Session hs=getSF().getCurrentSesison();
tx
try
user=hs.get(User.class,1);

hib chks if the corresponding id rec exists in L1 cache ?

doesn't exist
select with id(?)

rets existing
PERSISTENT ref
to the caller.

rec exists

rec does't exist

rets null

1. User u=new User();
2. all setters
3. adds this populated pojo ref to L1 cache (persistent)
4. ref is reted to the caller.

tx.commit();

user : DETACHED (from L1 cache)

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.

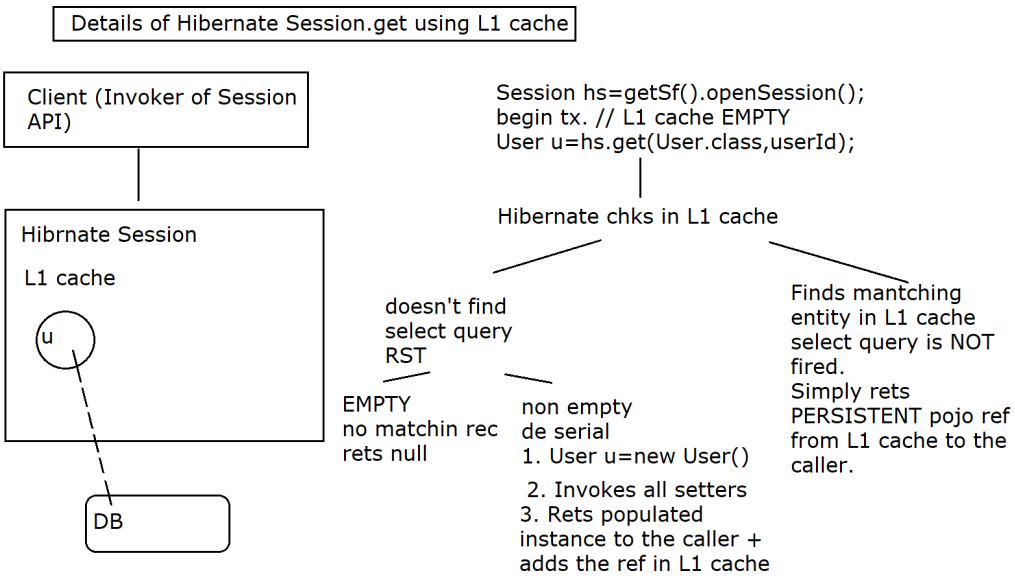
getCurrentSession

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.

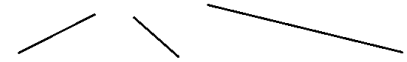
Hibernate POJO / Entity States

	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



save vs persist

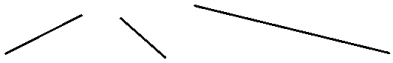
session.save(ref)



def val of id (null) | non null not existing in DB | non null existing rec in DB

common ans : Hibernate ignores passed id value, creates its own id n inserts a record in DB.

session. persist(ref)



def val of id (null) | non null not existing in DB | non null existing rec in DB

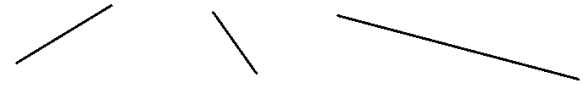
rec inserted no exc

common resp : PersistentObjExc : no insertion of a record.

Hibernate Session API

Assumptions : 1. id property : Integer/Long
2. @GeneratedValue(strategy=GenerationType.IDENTITY)

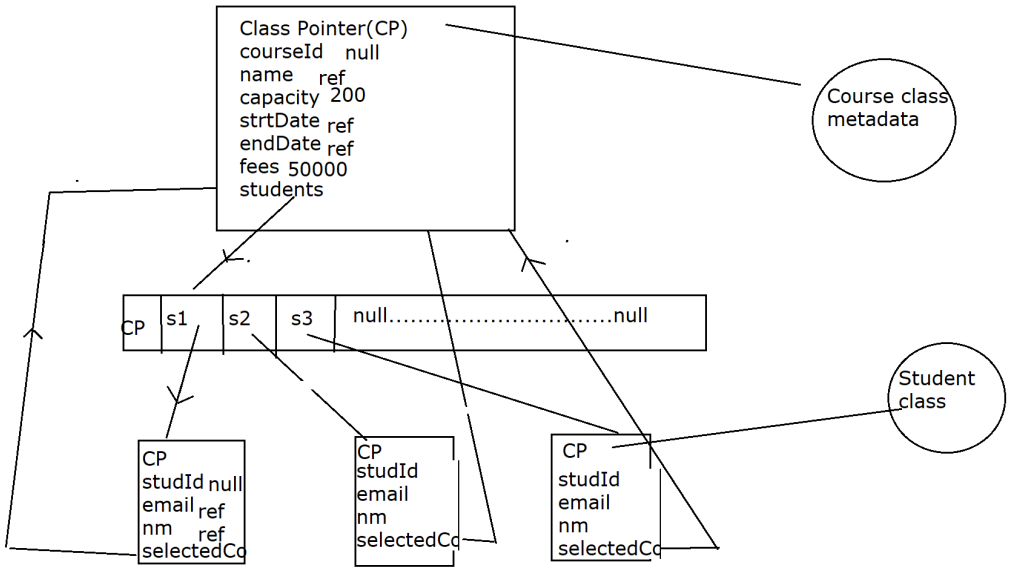
id= session.save(ref)



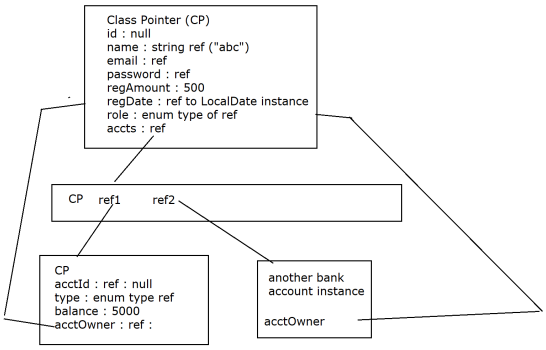
id=null (default) | id : not default(not null) : exists in DB | id : not null : doesn't exist in DB

No exception is raised : insertion of rec takes place.

```
Course c1 = new Course(sc.next(), sc.nextInt(), parse(sc.next()),
                        parse(sc.next()), sc.nextDouble());//c1 : transient
                //accept 3 student details , who want to enroll in this course
                for(int i=0;i<3;i++)
                {
                     System.out.println("Enter student details  email,  name");
                     Student s=new Student(sc.next(), sc.next());
                }
```




```
Vendor v = new Vendor(sc.next(), sc.next(), sc.next(), sc.nextDouble(),
                        LocalDate.parse(sc.next()),
                        Role.valueOf(sc.next().toUpperCase()));
                //accept 2 acct details from user
                for(int i=0;i<2;i++)
                {
                     System.out.println("Enter a/c details type n  balance");
                     BankAccount a=new
BankAccount(AcType.valueOf(sc.next().toUpperCase()), sc.nextDouble());
                }
```



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

1. Has its own DB identity -- Primary key (mapped by @Id)

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

3. Supports shared references.
eg Item & category. 2 Items can share the same category.

4. eg Item,Category,User,Order.

1. Doesn't have any DB identity (no @Id annotation)

2. Owned by Entity, no independent life-cycle. Life cycle bounded by the life cycle of owning entity.
3. Doesn't support shared references.

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.

get vs load -- Session API
public <T> T load/get (T class , Serilizable id)

get -- eager fetching policy -- i.e select query will be fired if POJO doesn't exists alrdy in L1 cache.

if id exists --- rets fully inited
object loaded with DB data
(doesn't return proxy)

Such **loaded object** can be directly accessed in detached manner--DOESn't result into LazyInitializationExc.

if id doesn't exist -- no exc ---null is reted.

load -- default fetching policy = lazy. i.e Hib frmwork will NOT fire select query (hit DB) unless & until -- non - id getter is invoked in persistent state.

If non-id getter is not accessed --- Hlb rets un-inited proxy(i.e only wrapper BUT no data from DB)

When such a un-inited proxy is accessed in detached state -- throws LazyInitExc, even when id exists or doesn't exist.

If non-id getter is accessed in persistent state & id doesn't exist--- throws ObjectNotFoundExc.

hs.load(.....)

existing id

invoked non-id getter

inited proxy (select)

--No exc.

not invoked non-id getter (within sess scope)

un-inited proxy

session closed -- sop(b);

LazyInitExc ---accessed not-yet-fetched(un-inited proxy) data in detached manner

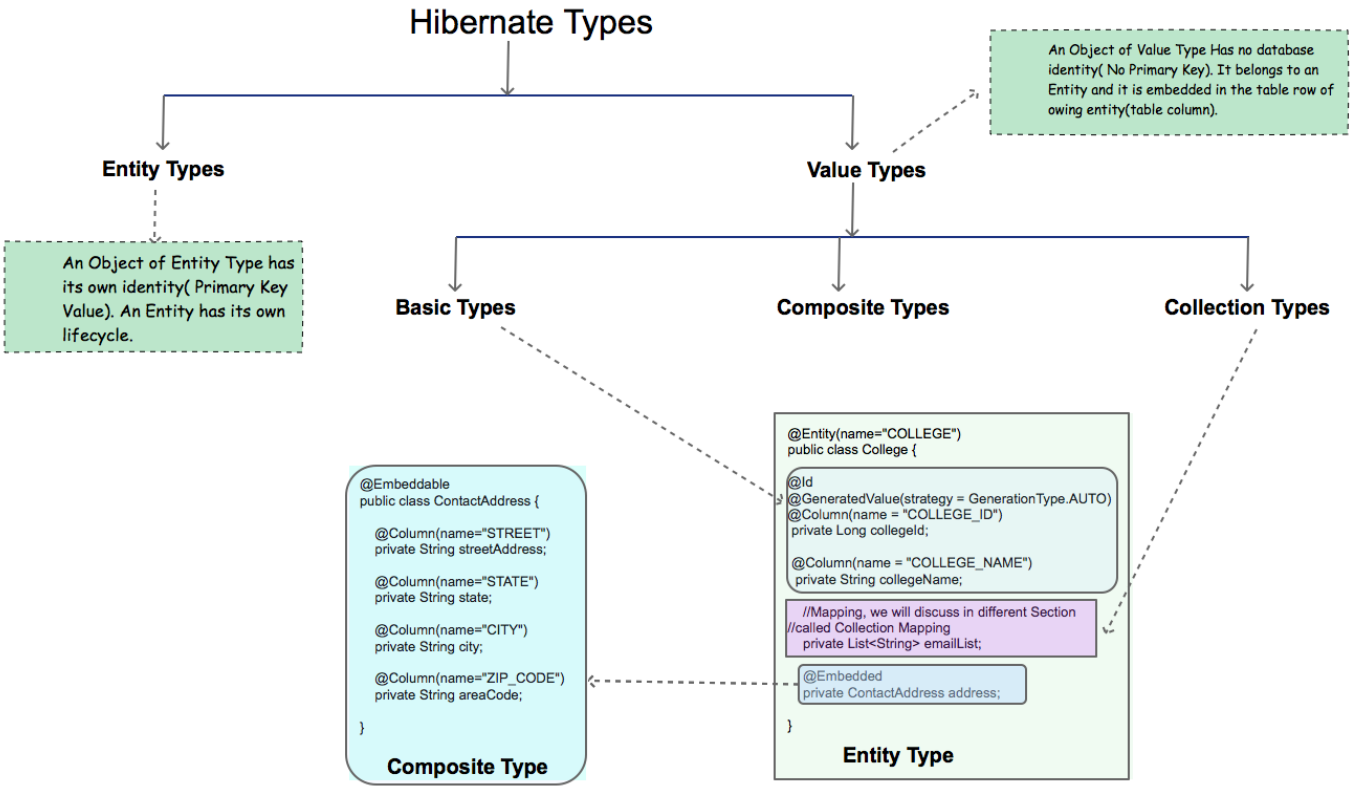
non existing id

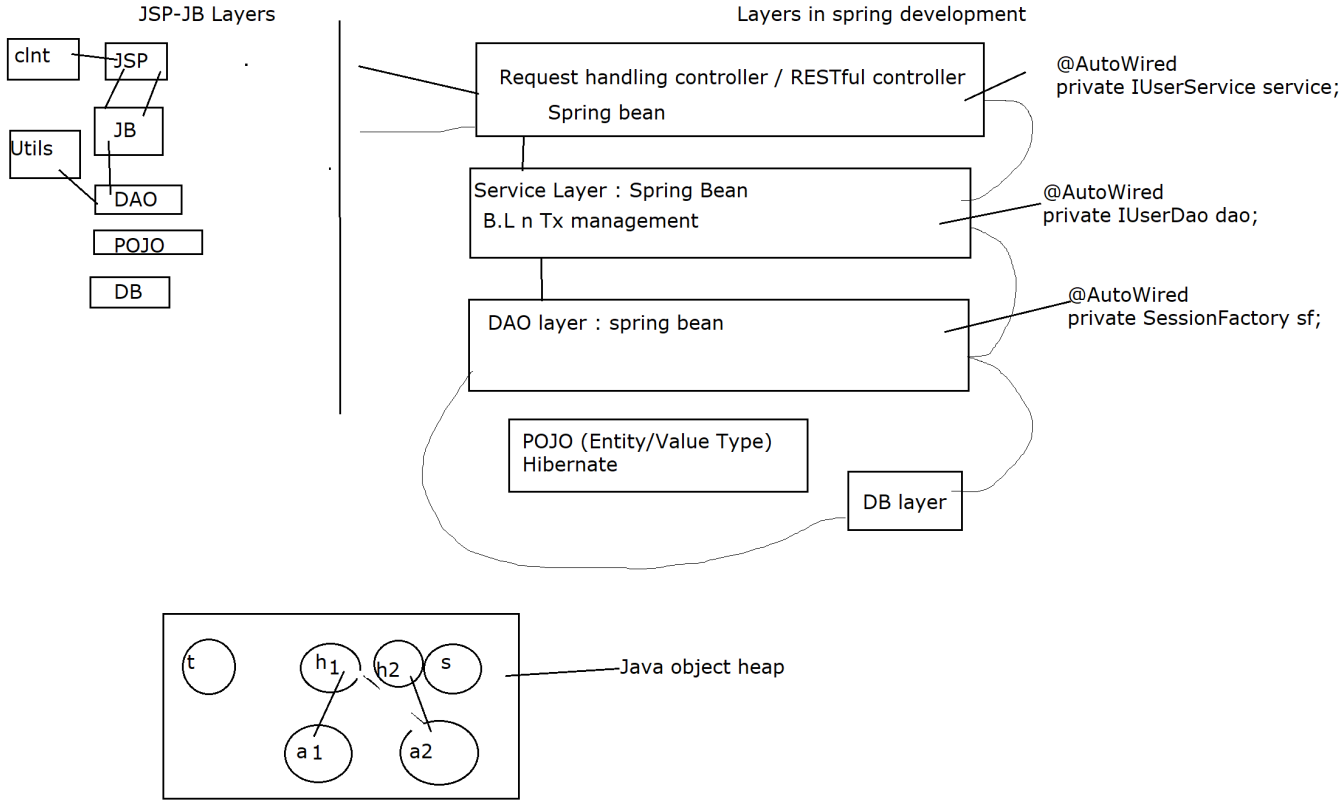
non -id getter accessed(eg : b.getPrice())

exc --- ObjectNotFoundExc

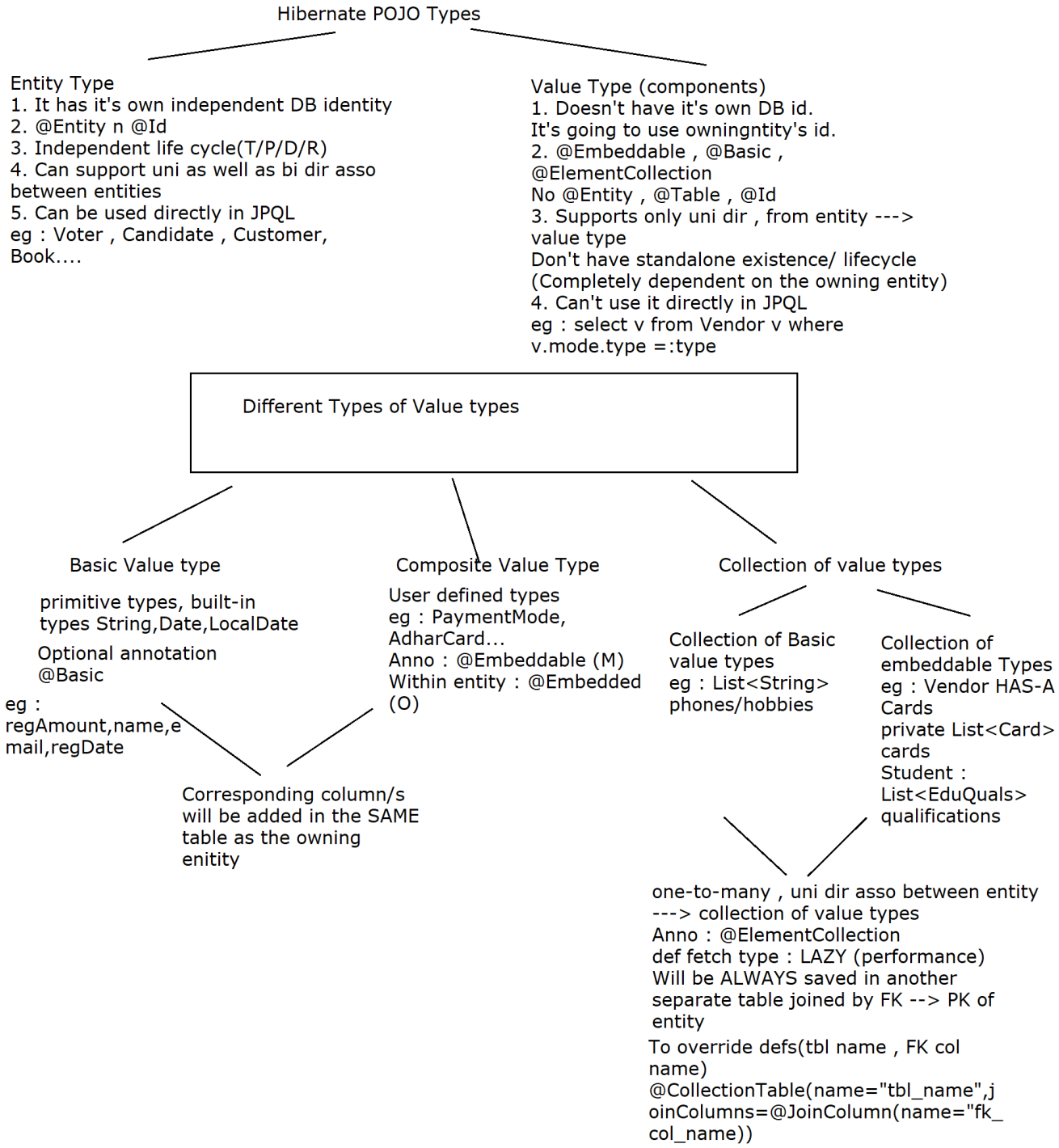
non-id getter not accessed in session scope

exc ---LazyInitExc




JSP-JB Layers
clnt
JSP
JB
Utils
DAO
POJO
DB
Layers in spring development
Request handling controller / RESTful controller
Spring bean
@AutoWired
private IUserService service;
Service Layer : Spring Bean
B.L n Tx management
@AutoWired
private IUserDao dao;
DAO layer : spring bean
@AutoWired
private SessionFactory sf;
POJO (Entity/Value Type)
Hibernate
DB layer

t
h1
h2
s
a 1
a2
Java object heap


Hibernate POJO Types
Entity Type
1. It has it's own independent DB identity
2. @Entity n @Id
3. Independent life cycle(T/P/D/R)
4. Can support uni as well as bi dir asso between entities
5. Can be used directly in JPQL
eg : Voter , Candidate , Customer, Book....
Value Type (components)
1. Doesn't have it's own DB id.
It's going to use owningntity's id.
2. @Embeddable , @Basic , @ElementCollection
No @Entity , @Table , @Id
3. Supports only uni dir , from entity ---> value type
Don't have standalone existence/ lifecycle (Completely dependent on the owning entity)
4. Can't use it directly in JPQL
eg : select v from Vendor v where v.mode.type =:type
Different Types of Value types
Basic Value type
primitive types, built-in types String,Date,LocalDate
Optional annotation @Basic
eg : regAmount,name,email,regDate
Composite Value Type
User defined types
eg : PaymentMode, AdharCard...
Anno : @Embeddable (M)
Within entity : @Embedded (O)
Collection of value types
Collection of Basic value types
eg : List<String> phones/hobbies
Collection of embeddable Types
eg : Vendor HAS-A Cards
private List<Card> cards
Student : List<EduQuals> qualifications
Corresponding column/s will be added in the SAME table as the owning enitity
one-to-many , uni dir asso between entity ---> collection of value types
Anno : @ElementCollection
def fetch type : LAZY (performance)
Will be ALWAYS saved in another separate table joined by FK --> PK of entity
To override defs(tbl name , FK col name)
@CollectionTable(name="tbl_name",joinColumns=@JoinColumn(name="fk_col_name))

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.

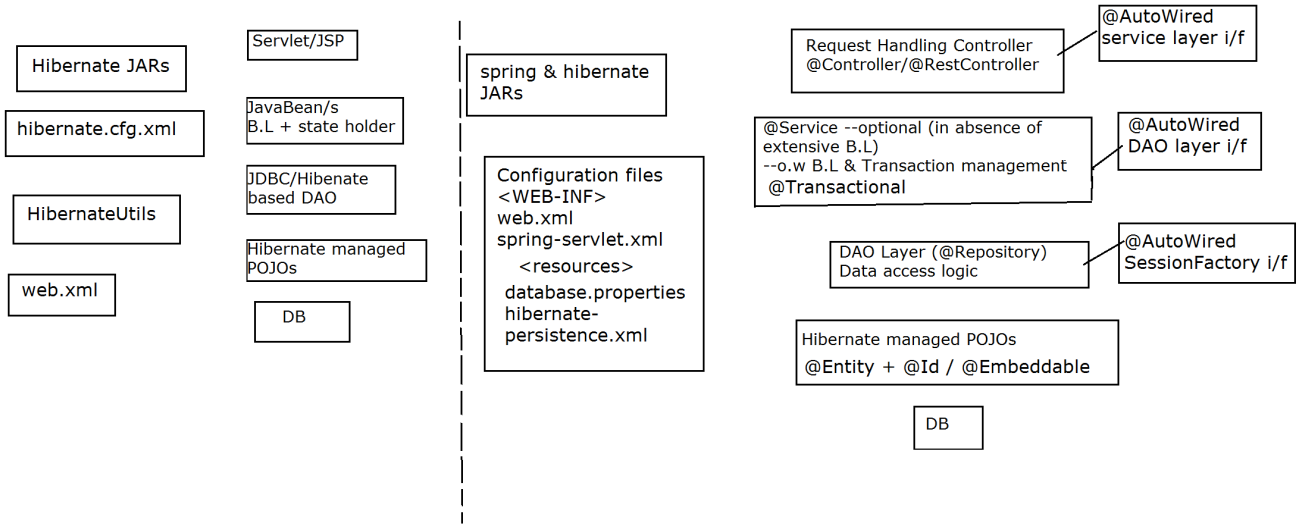
Implicit Wiring (auto wiring) --Must supply setters or constrs . No XML config. Just choose autowire mode

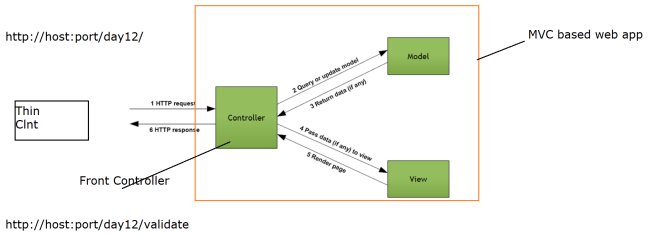
autowire=byName
1. Provide setters in dependent bean class.
2. No <property> tags required in xml config files.
3.Must match property setter names to dependency bean id.
4. No exception thrown if match not found.

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

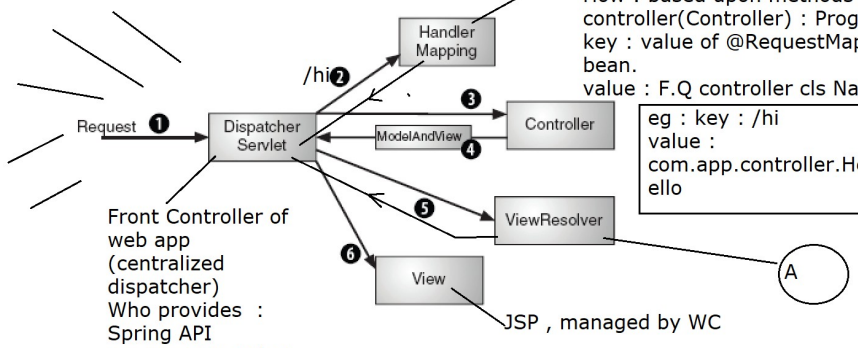
Servlet/JSP Using JavaBean Layers Vs Spring MVC Layers





http://host:port/spring_mvc/hi

Spring MVC Overview



Spring managed Bean
Map<K,V>
Who creates this Map : SC
How : based upon methods in request handling controller(Controller) : Prog supplied
key : value of @RequestMapping anno in Contrller bean.
value : F.Q controller cls Name + method name

eg : key : /hi
value :
com.app.controller.HelloController.sayHello

Front Controller of web app (centralized dispatcher)
Who provides :
Spring API
o.s.w.s.Dispatcher Servlet

How to configure ?
web.xml servlet dep. tags
load-on-startup=1
servlet-name
spring

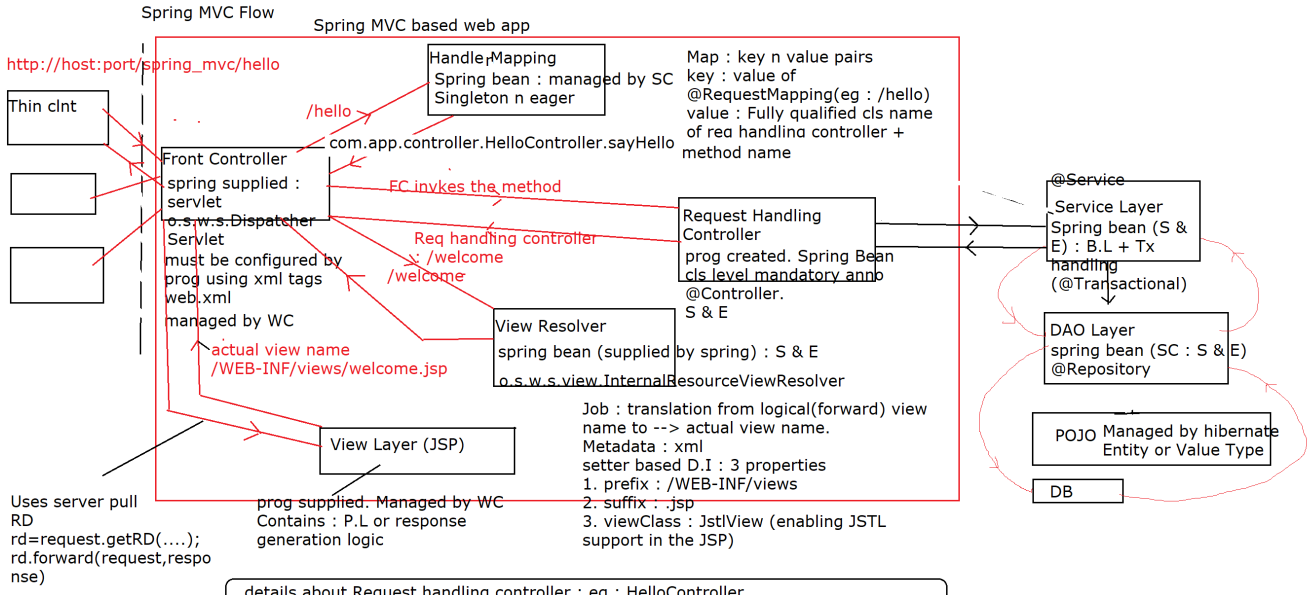
How to ensure that any clnt sending any req is intercepted by D.S ?
url-pattern /

init() of D.S :
start SC (WebApplicationContext i/f)

Hybrid approach (less XML + majority annotations)
XML
name : spring-servlet.xml
location : WEB-INF
Consists of instrs meant for SC

A

View resolver : spring managed bean
singleton n eager : loc-loaded-inst --D.I --init
Job : translation from logical view name to actual view name
How
prefix : /WEB-INF/views
suffix : .jsp



details about Request handling controller : eg : HelloController
spring bean (S & E)
@Controller : class level annotation. Contains multiple request handling methods.
How to tell SC about request handling method
eg : @RequestMapping("/hello")
public String sayHello() { return "/welcome";}

After adding above HelloController , what will be the entry in HandlerMapping bean?
key : /hello
value : com.app.controller.HelloController.sayHello

who's going to start SC in web app ?
Front Controller : supplied by Spring : DispatcherServlet.(D.S)
web.xml : servlet tags
servlet-name=spring
load-on-startup =1

D.S --- uses hybrid approach (lesser xml n more annotaions) :
By default : spring-servlet.xml
location : <WEB-INF>
contains : metadata to start SC

'j2ee'

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

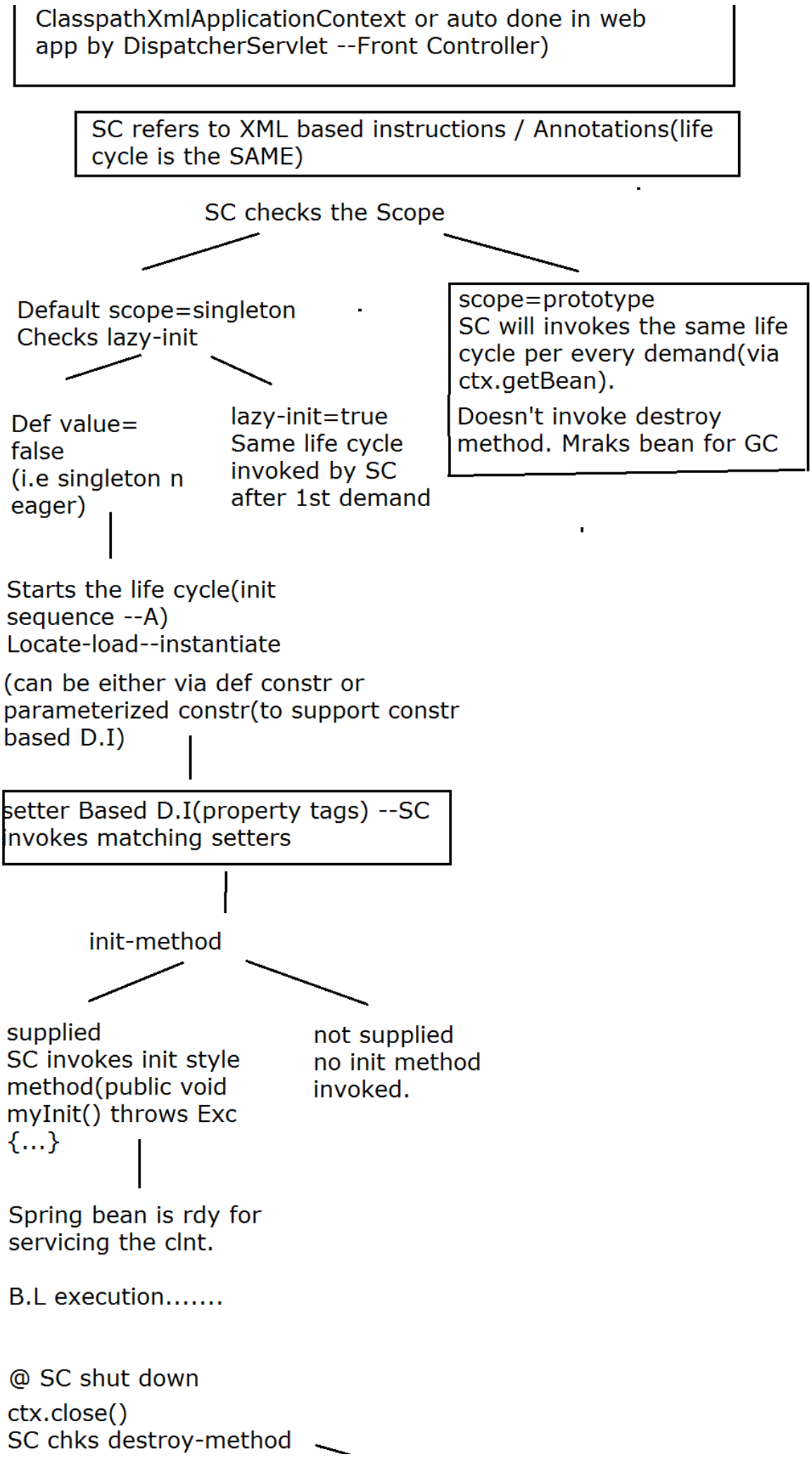
autowire=byName
1. Provide setters in dependent bean class.
2. No <property> tags required in xml config files.
3.Must match property setter names to dependency bean id.
4. No exception thrown if match not found.

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

Spring Bean life cycle maanged by Spring Container

Start SC (in Java SE by creating instance of


ClasspathXmlApplicationContext or auto done in web app by DispatcherServlet --Front Controller)
SC refers to XML based instructions / Annotations(life cycle is the SAME)
SC checks the Scope
Default scope=singleton
Checks lazy-init
scope=prototype
SC will invokes the same life cycle per every demand(via ctx.getBean).
Doesn't invoke destroy method. Mraks bean for GC
Def value= false
(i.e singleton n eager)
lazy-init=true
Same life cycle invoked by SC after 1st demand
Starts the life cycle(init sequence --A)
Locate-load--instantiate
(can be either via def constr or parameterized constr(to support constr based D.I)
setter Based D.I(property tags) --SC invokes matching setters
init-method
supplied
SC invokes init style method(public void myInit() throws Exc {...}
not supplied
no init method invoked.
Spring bean is rdy for servicing the clnt.
B.L execution.....
@ SC shut down
ctx.close()
SC chks destroy-method

supplied (public void anyName() throws Exc {...}

SC invokes the same.
Marks bean for GC
End of life cycle.

not supplied
--doesn't invoke destroy method callback.

Marks the bean for GC

Singleton Vs Prototype

1. Single instance of bean instance is shared across multiple demands made to Spring Container(SC)

2. Default loading policy = eager. Can be changed to lazy (using lazy-init)

3. SC will automatically invoke destroy style method , upon ctx closing.

4. Created as stateless beans (Can't contain the conversational state of the client)

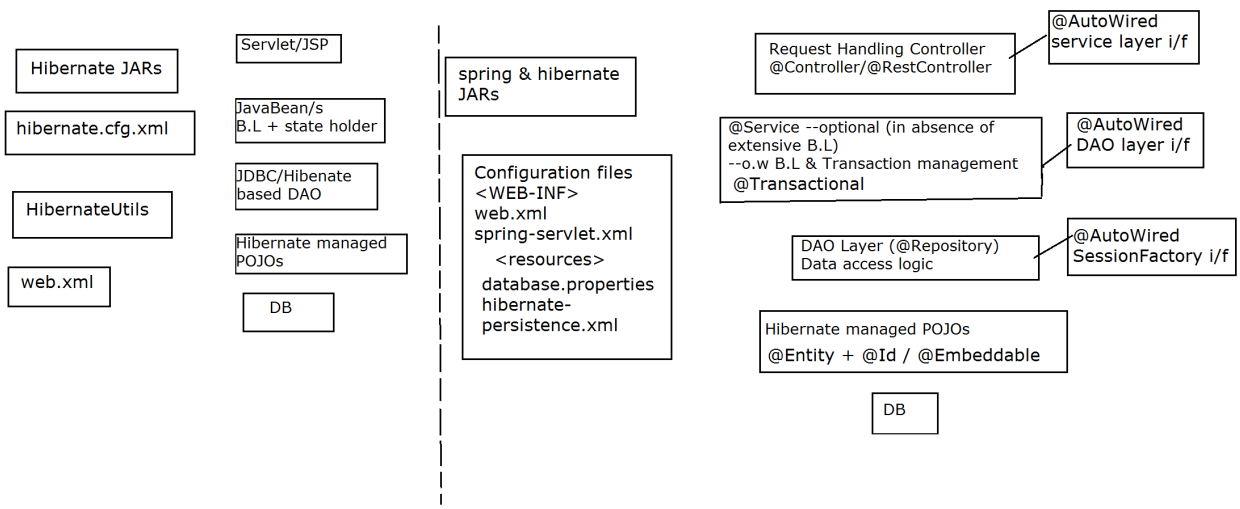
1. SC will create a separate bean instance per demand(getBean)

2. Prototype beans are always loaded lazily (upon demand). lazy-init is not applicable

3. SC doesn't invoke destroy style method.

4. Stateful Beans. Can hold conversational state of the client.

Servlet/JSP Using JavaBean Layers Vs Spring MVC Layers



Spring Hibernate Integration development Blocks

WEB-INF

web.xml
DispatcherServlet
url-pattern "/"

Request Handling Controller
@Controller/@RestController
@RequestMapping --class level anno.

url pattern "/
servlet name
"spring"
load-on-startup=1

spring-servlet.xml
master bean config file
who --D.S
what -- SC
<context-anno-confg>
<context:comp-scan>
<mvc:anno-driven>
Config view resolver-
prefix n suffix
<import hib persistence xml>

<resources> --run
time classpath --
<web-inf>/classes

database.properties
drvr class,db url,user
name,pass , db dialect
(optional in hibernate 5)

hibernate-persistence.xml
1. location of db properties.
2. spring beans
2.1 Connection Pool
vendor --Apache
class --BasicDataSource
(imple class of
javax.sql.DataSource)
dependencies --db config
details)
2.2 SF Bean
i/f --org.hibernate.SF
imple class --Spring
supplied
o.s.orm.hib5.LocalSessionFa
ctoryBean
dependencies -- CP
packages-to-scan --pkg
containing POJOs

Method level -- req handling methods.
@GetMapping/@PostMapping...
Important anno/dependencies
Model -- map of model attrs
RedirectAttributes --map of flash attrs
@RequestParam -- req param
@PathVariable --URI template vars.
HttpSession,HttpServletRequest,POJO
(empty POJO gets in Model map)
Typically --2 phases --@GetMapping
showForm n @PostMapping processForm
Dependency of controller layer --Service
@AutoWired
private Service service;

Service Layer --B.L methods + Tx
management. i/f + implementation class
Class level anno -- @Service @Transactional
@Transactional -- for auto tx handling.
properties
readOnly --def --false
timeout --in secs (if tmout elapses --tx auto
rolled back.
tx propagation rules --
supports,mandatory,never,new
tx isolation level --to avoid dirty/non
repeatable/phantom
rollback rules --def --Tx mgr bean will auto
rollabck tx --if @Transactional method raises
un-checked exception.
can be overridden by supplying your own
rollback rules.
Dependency of service layer --DAO
@AutoWired
private DAO dao;

DAO Layer --
i/f & implementation class
class level anno --@Repository
Dependency -- SF
@AutoWired
private SF sf;
No constr/clenup. Complete reduction of
boilerplate code.
eg :

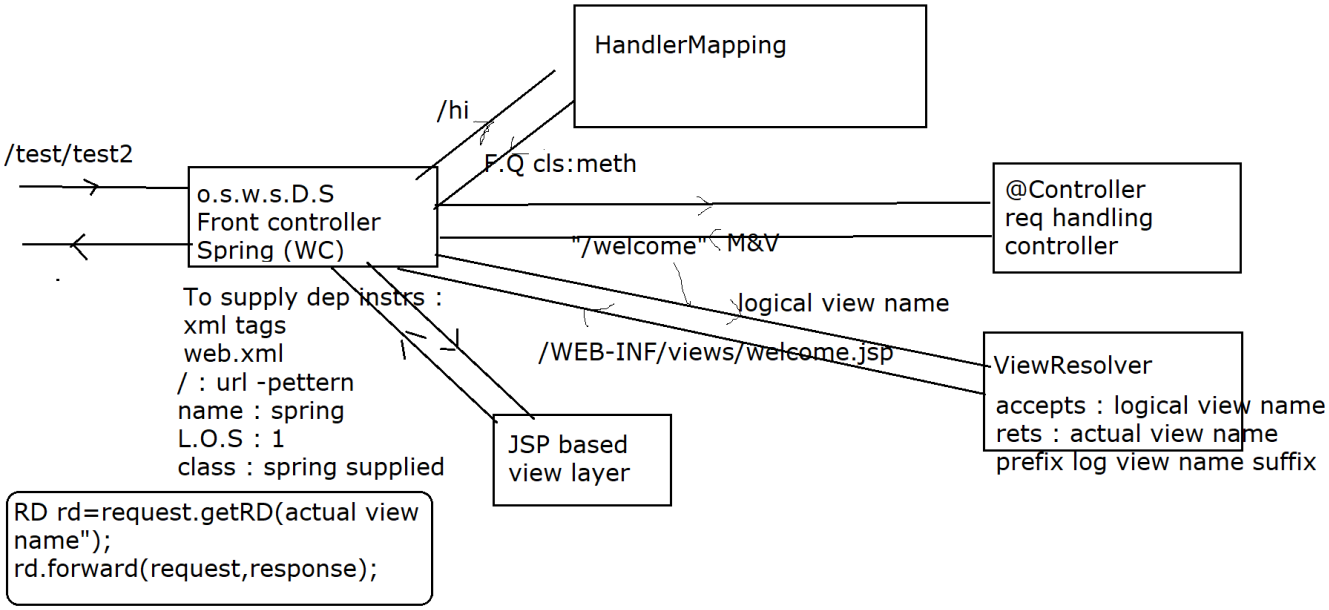
```
public Customer validateCustomer(em,pass)
{
  return sf.getCurrentSession().createQuery
(jpql,Customer.class).setParameter
(...).setParamter(..).getSingleResult();
}
```

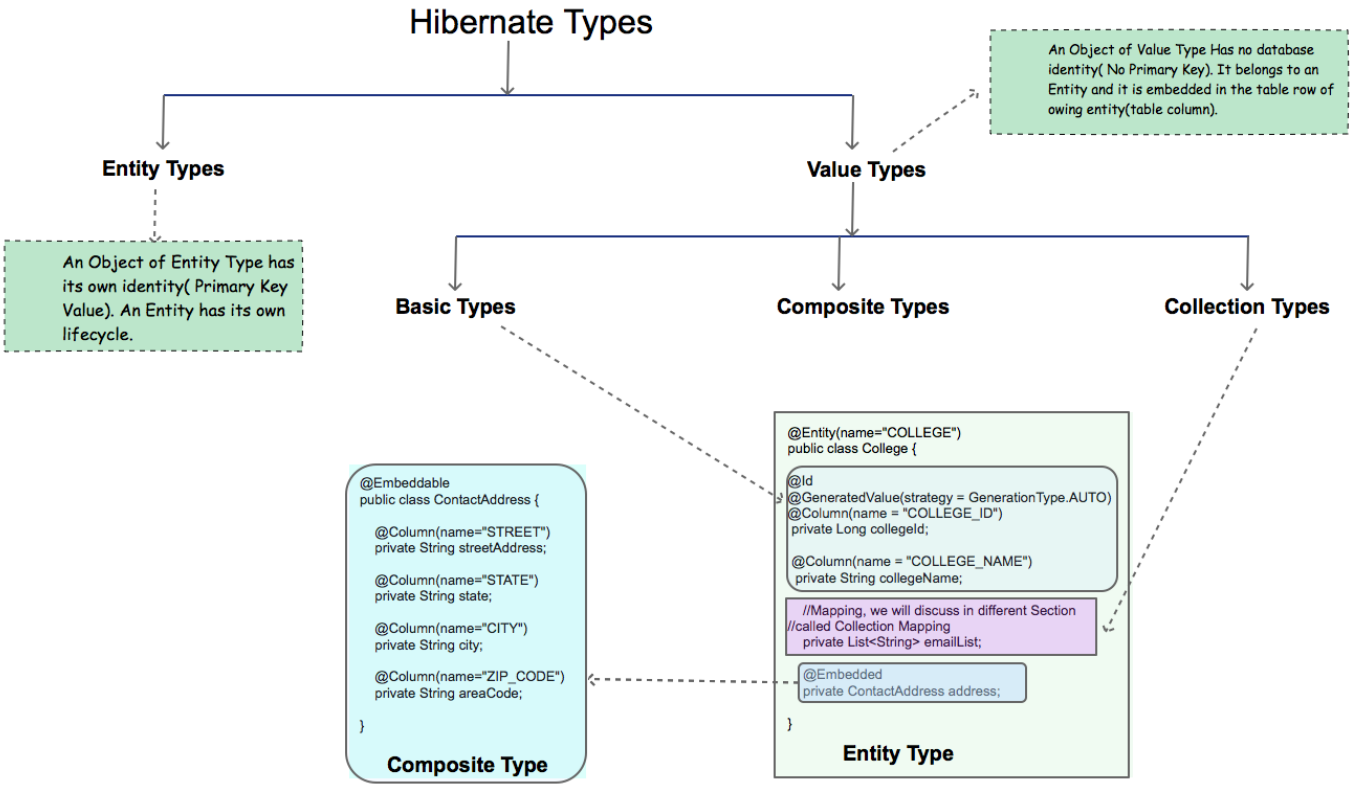
POJO -- typically represent ORM
Annotation -- @Entity/@Embeddable (In case of DTO

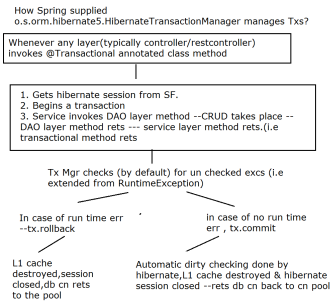
--to xchange info between REST server n clnt --No annos required)
POJO as ORM
POJO props --duel role --represents
1. DB columns
2. conversation state of the clnt(req param/path)

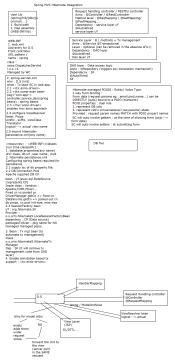
For ensuring form binding with POJO -- MUST MATCH request param names(path) with POJO prop names (reflected via setters)
Annotaions
1. Field level -- anno for Presentation logic validations
eg : @NotEmpty,@NotNull,@Pattern,@Email, @DateTimeFormat,@Range...

2. Property level --JPA annotations
@Column, @Id,@OneToMany...




Hibernate Types
An Object of Value Type Has no database identity(No Primary Key). It belongs to an Entity and it is embedded in the table row of owing entity(table column).
Entity Types
Value Types
An Object of Entity Type has its own identity(Primary Key Value). An Entity has its own lifecycle.
Basic Types
Composite Types
Collection Types
@Embeddable
public class ContactAddress {
@Column(name="STREET")
private String streetAddress;
@Column(name="STATE")
private String state;
@Column(name="CITY")
private String city;
@Column(name="ZIP_CODE")
private String areaCode;
}
Composite Type
@Entity(name="COLLEGE")
public class College {
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
@Column(name = "COLLEGE_ID")
private Long collegeId;
@Column(name = "COLLEGE_NAME")
private String collegeName;
//Mapping, we will discuss in different Section
//called Collection Mapping
private List<String> emailList;
@Embedded
private ContactAddress address;
}
Entity Type


How Spring supplied o.s.orm.hibernate5.HibernateTransactionManager manages Txs?
Whenever any layer(typically controller/restcontroller) invokes @Transactional annotated class method
1. Gets hibernate session from SF.
2. Begins a transaction
3. Service invokes DAO layer method --CRUD takes place -- DAO layer method rets --- service layer method rets.(i.e transactional method rets
Tx Mgr checks (by default) for un checked excs (i.e extended from RuntimeException)
In case of run time err --tx.rollback
in case of no run time err , tx.commit
L1 cache destroyed,session closed,db cn rets to the pool
Automatic dirty checking done by hibernate,L1 cache destroyed & hibernate session closed --rets db cn back to cn pool



Solution to identify scope

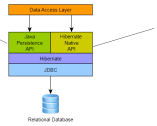
To remember the details in curnt req only
Model map
(model attrs)
map.addAttribute(nm,value);

To remember details only till next req coming from same clnt
RedirectAttributes
addFlashAttribute(nm,value)

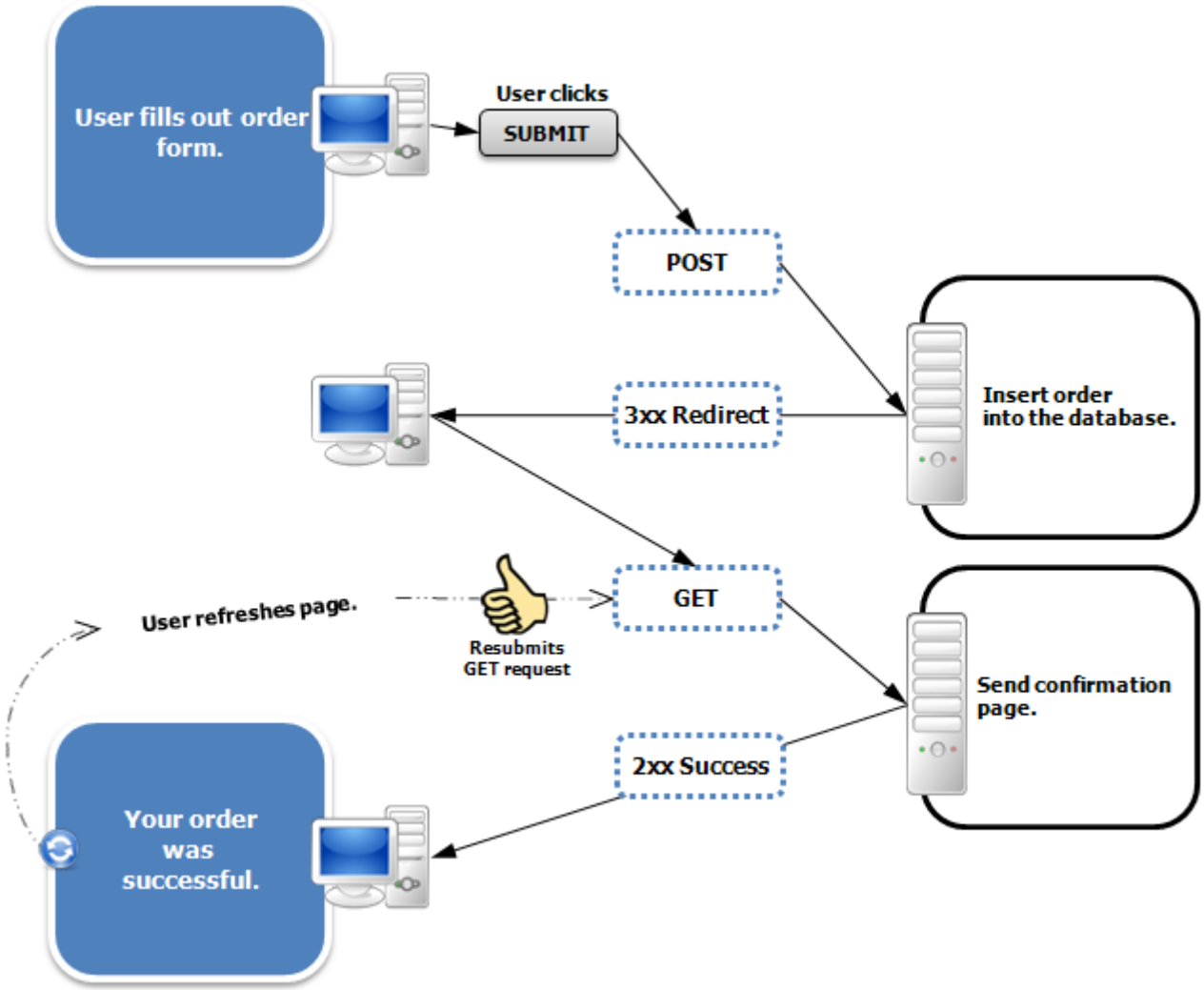
To remember details in entire session(from login --logout)
HttpSession
setAttribute(nm,val);

DAO : javax.persistence

EntityManagerFactory
@PersistenceContext
EntityManager
API : CRUD
persist
merge
find / JPQL
remove
createQuery



Hibernate Native API (DAO)
org.hibernate
@AutoWired
SessionFactory
Session :
getCurrentSession
Session CRUD :
save/persist
update
get/JPQL
delete
createQuery


User fills out order form.
User clicks
SUBMIT
POST
Insert order into the database.
3xx Redirect
GET
User refreshes page.
Resubmits GET request
Send confirmation page.
2xx Success
Your order was successful.

Validations in web applications

Presentation Logic validations
clnt side(JS) or server side --
after form is submitted

B.L validation -- server side
typically by DB
eg : DAO layer / service layer
eg : validateUser via DB

Spring MVC
1. Add validation rules in POJO
eg : @NotNull,@NotEmpty,
@DateTimeFormat,@Pattern,
@Length,@Range....
can be added field / property
level.

2. In Controller
--typically in processForm
To tell SC , to apply validation rules , use @Valid anno on POJO

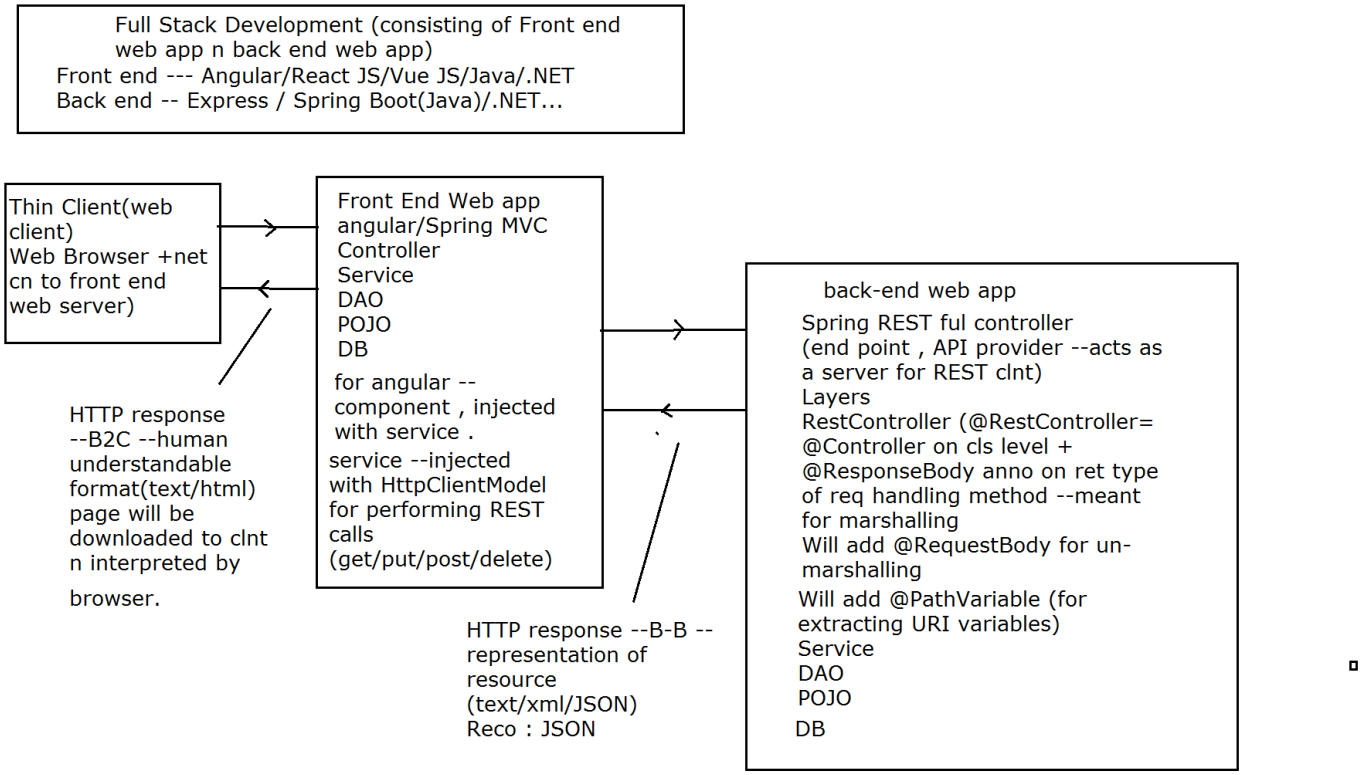
3. SC stores conversion n validation errors in an i/f
o.s.validation.BindingResult --
Reps binding between --- string based rq params --- pojo props

RULE -- BindingResult MUST appear immediately after @Valid POJO
o.w spring throws exc.
eg : @PostMapping("/register")
public String processForm(@Valid Customer c,BindingResult res,....)
{SC ---empty pojo , setters --- conv n validation rules invoked-res of
bininding(errs) is stored under BindingRes.
//P.L chks -- use API --boolean hasErrors(),boolean hasFieldErrors
(String pojoPropName)

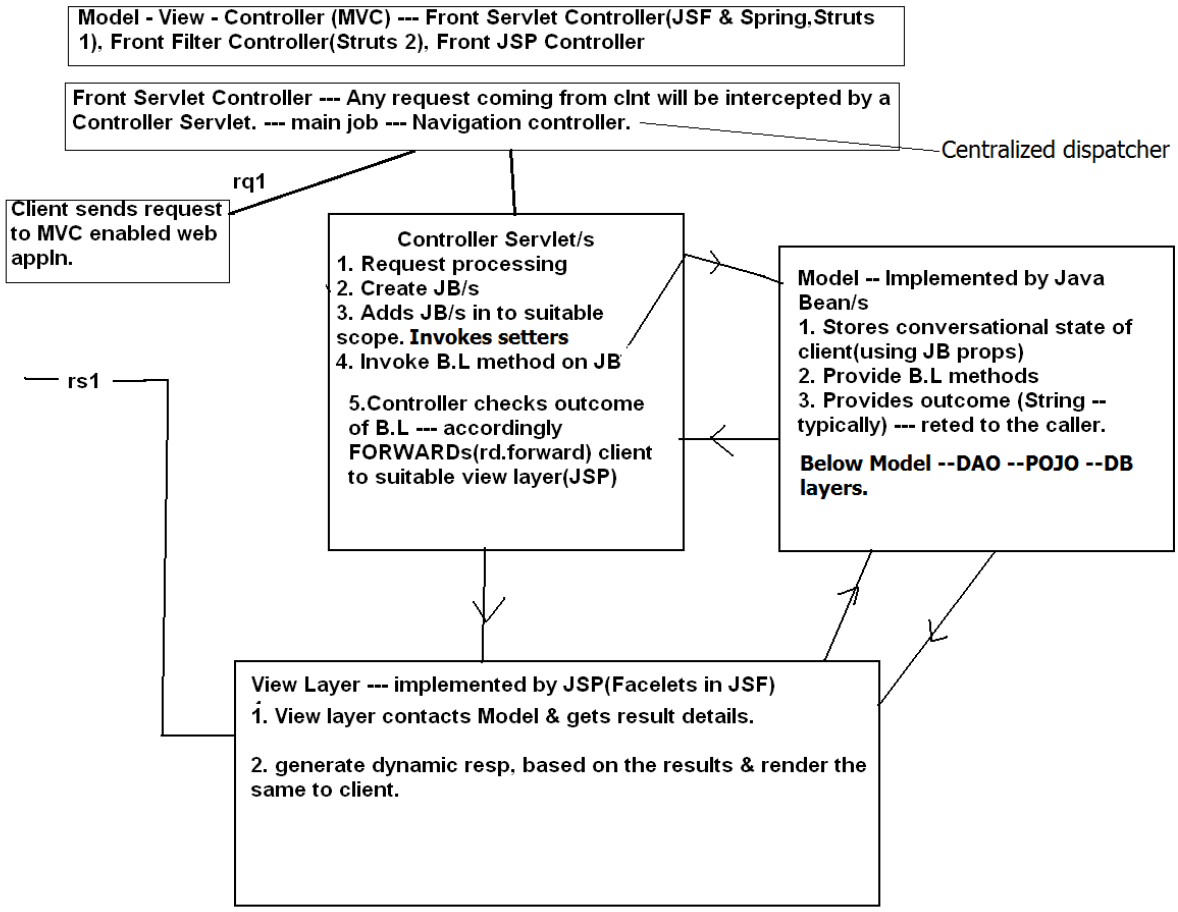
errs -- forward clnt
to view layer(reg
form)

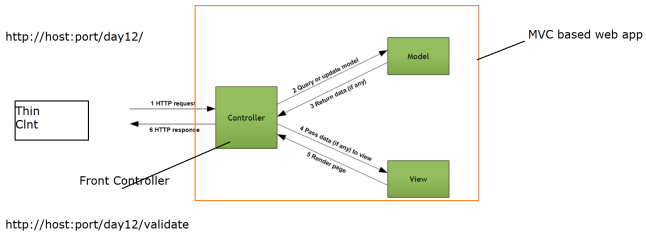
no errs---invoke service layer method &
proceed to B.L validations.

4. In view layer (JSP)
<form:errors path="pojoPropName"/>



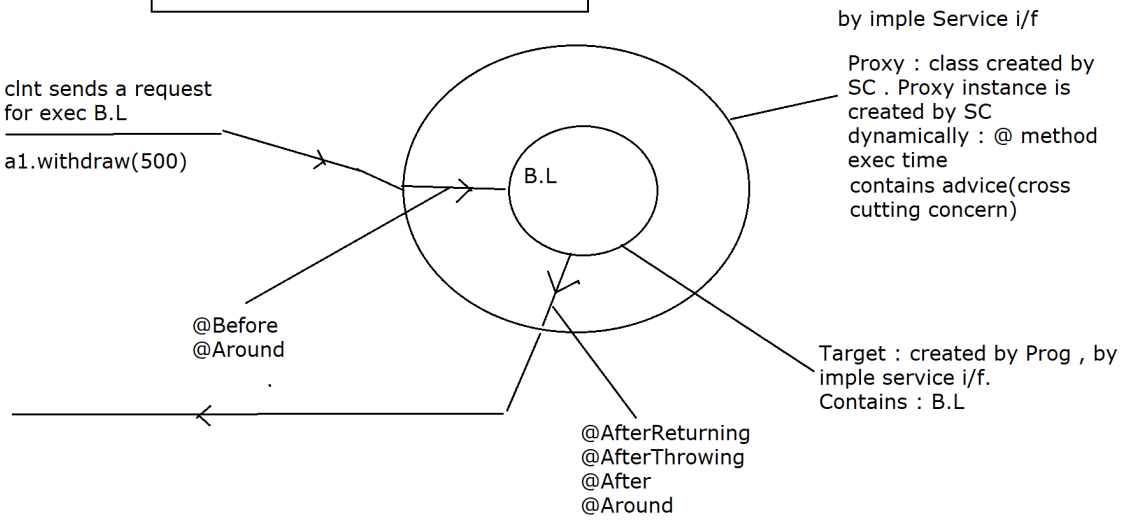
Implementation of MVC : using Front Servlet Controller Pattern
1.Thin clnt sends the HTTP rq. to a servlet controller.
2.Servlet/s reads the request params & performs initial processing if reqd.
3. Servlet controller will instantiate the Model componet/s (Java Beans)
2 ways : can directly invoke the 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.

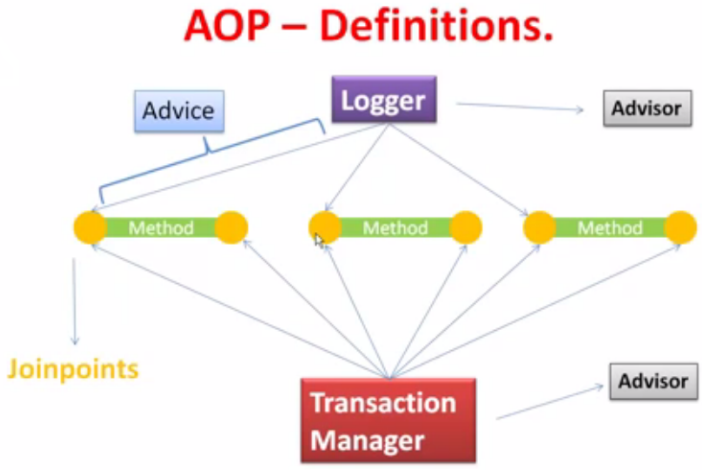

Model - View - Controller (MVC) --- Front Servlet Controller(JSF & Spring,Struts 1), Front Filter Controller(Struts 2), Front JSP Controller
Front Servlet Controller --- Any request coming from clnt will be intercepted by a Controller Servlet. --- main job --- Navigation controller.
Centralized dispatcher
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 of B.L --- accordingly FORWARDs(rd.forward) client to suitable view layer(JSP)
Model -- Implemented by Java Bean/s
1. Stores conversational state of client(using JB props)
2. Provide B.L methods
3. Provides outcome (String -- typically) --- reted to the caller.
Below Model --DAO --POJO --DB layers.
rs1
View Layer --- implemented by JSP(Facelets in JSF)
1. View layer contacts Model & gets result details.
2. generate dynamic resp, based on the results & render the same to client.



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.

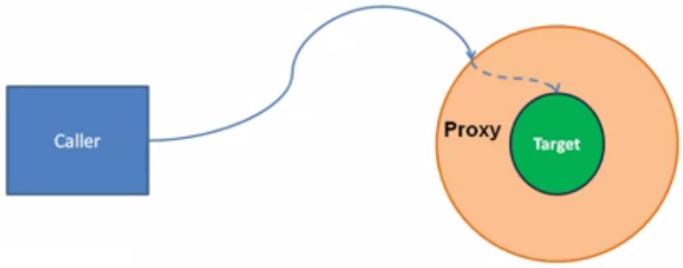
```
Regarding AOP
o/p
sc started....
com.sun.proxy.$Proxy22
Logging details
beginning tx
in withdraw
rolling back tx
err acct overdrawn
```





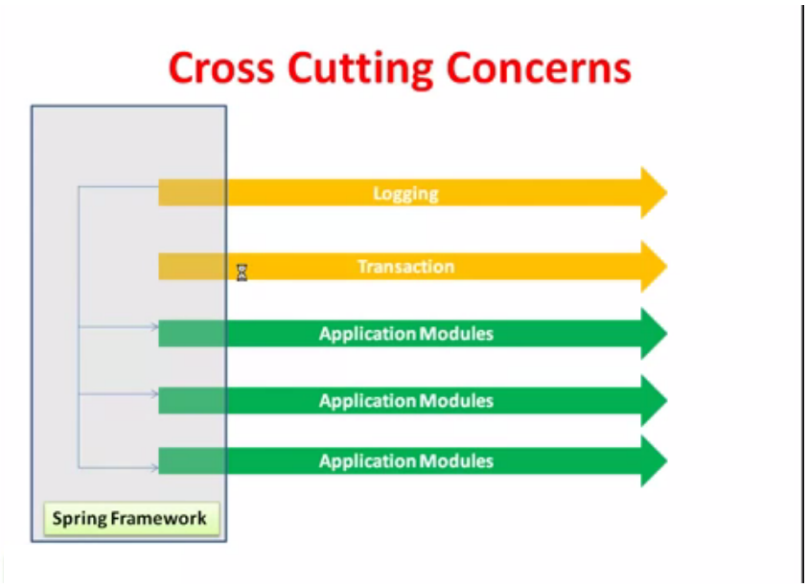
AOP - Weaving

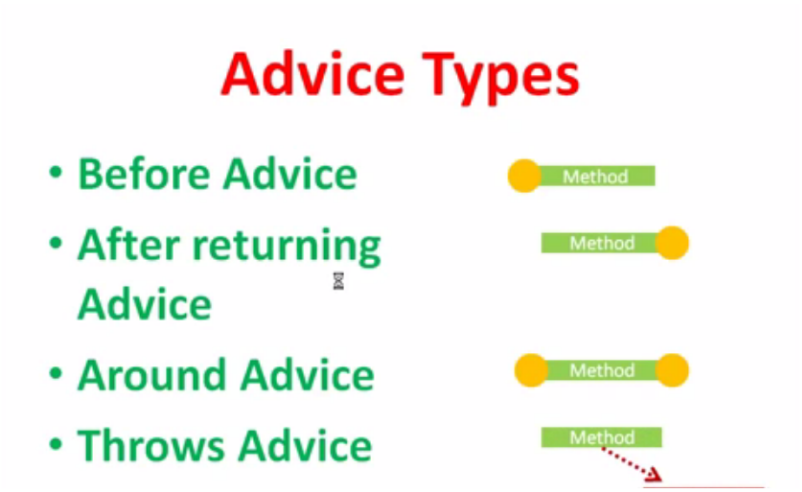
- Compile time
- Class Load Time
- Runtime – Springs way



Cross Cutting Concerns

```
class Bank{
   private int balance;
  public void withdraw(int amount){
   bankLogger.info("Withdraw –"+amount);
   tx.begin();
   balance = this.balance-amount;
   accountDao.saveBalance(balance);
   tx.commit();
   }
}
```


Cross Cutting Concerns
Logging
Transaction
Application Modules
Application Modules
Application Modules
Spring Framework


Advice Types
• Before Advice
• After returning Advice
• Around Advice
• Throws Advice
Method
Method
Method
Method

Steps for creating Custom Tag --- using javax.servlet.jsp.SimpleTag i/f

Identify tag exec logic ---prints hello with changing time stamp
create public pkged class --- extends SimpleTagSupport (impl class of SimpleTag i/f)
MUST override public void doTag() throws JspExc , IOExc --- to supply exec logic.
Much before --- WC invokes doTag --- WC invokes public void setJspContext(JspContext ctx) to xfer entire page env(PageContext) --- to the tag handler class.
How to get PC fro doTag ? --- getJspContext().getOut().print("hello....");

Create TLD (Tag lib descriptor) --- to describe custom tags(tag name, class ,attr details,body content details) --- to WC -- so that WC can invoke life-cycle of tag properly.
---.tld , loc --- <WEB-INF>/tlds/example.tld --- xml content --- xsd -- j2ee compliant
<uri> /WEB-INF/tlds/example </uri>
Per custom tag --- w/o body & w/o attributes
<tag> <name> tag suffix</name> <tag-class>F.Q tag handler cls name </tag-class>
<body-content>empty</body-content></tag>

use custom tag in .jsp
1. Import TLD
<%@ taglib uri="...." prefix="my"%>
2. <my:hello/>

Custom Tgas Vs Java Beans

Custom tags
1. Completely tied to JSP API(can't exist w/o WC) -- typically extends from SimpleTagSupport.

2. WC passes PageContext object to Custom Tag --- so can access entire page env from within doTag of the custom tag.

3. Custom tags developed using SimpleTag API -- are cacheless --- can't be added to any scope.--entire life-cycle (loc/load/inst ---GC) repeats per tag invocation.

Java Beans
1. Completely de-coupled from JSP API --- do not use any JSP specific super classes or dont need to imple. any JSP/Servlet specific i/f

2. JavaBeans do not have PageContext --passed by WC ---so CANT directly access anything from JSP page env.

3. JBs can be added to page|req|session|appln & can be shared across multiple dyn web pages.

Custom tag life cycle --- SimpleTag --- i/f

<test:welcome/> --- WC checks prefix --- test --- identifies -- uri of TLD --- location of TLD & opens the same.--- suffix --- from TLD resolves --- tag name---tag class --- loc(web-inf/classes), load -- inst -- def constr.

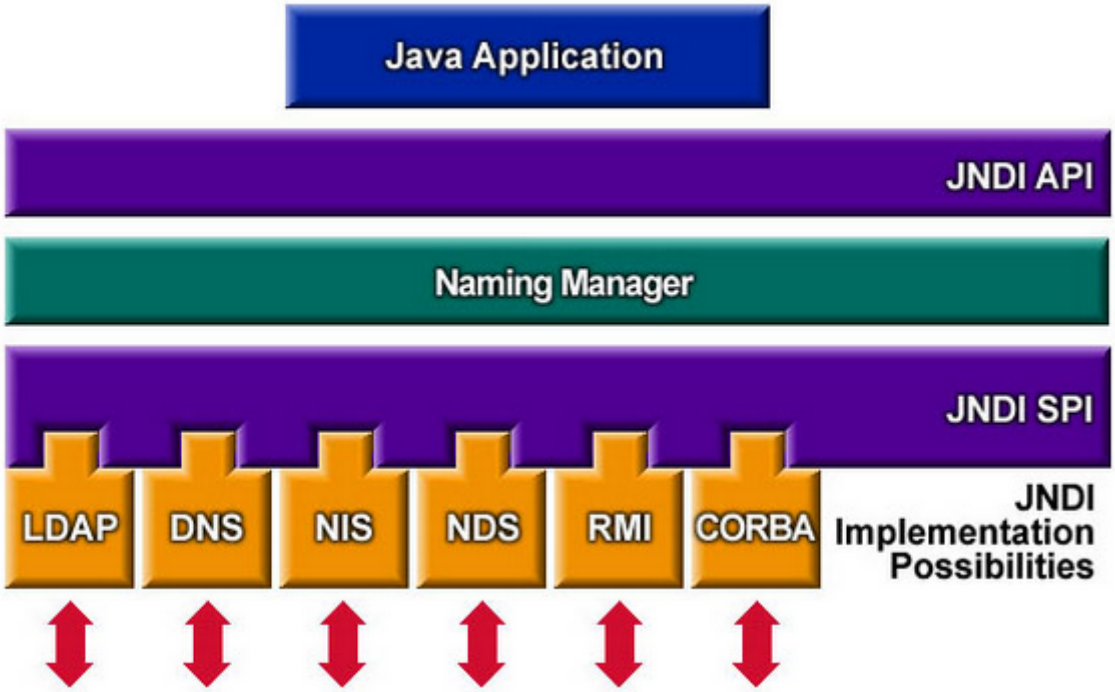
WC invokes setJspContext(JspContext ctx) --- to xfer entire page env to tag handler class . --- Mandatory

WC invokes setParent() --- ONLY if current tag is nested tag -- skipped otherwise.

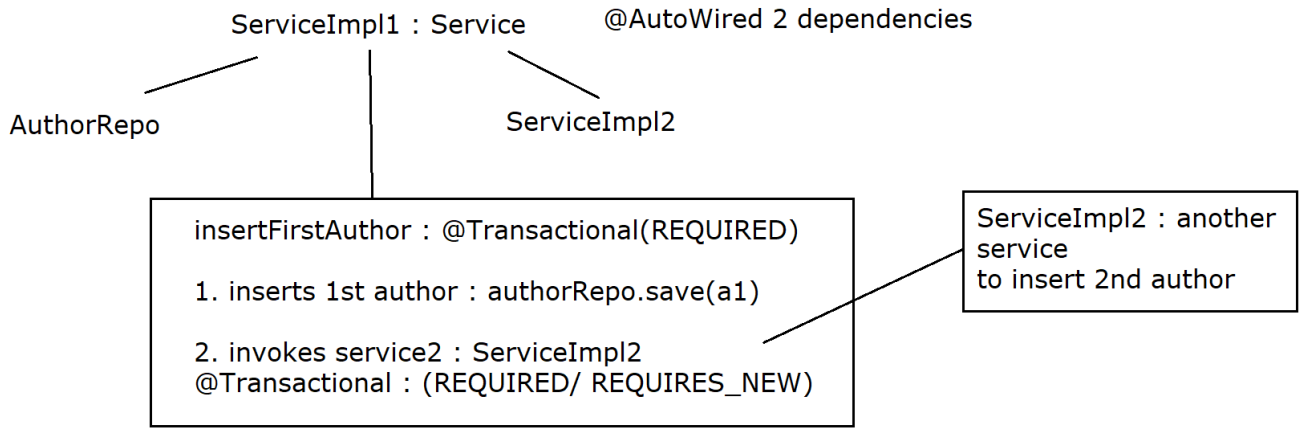
WC invokes setters for mandatory or supplied attributes from tag handler class. (Prog jobs --- must supply setter per each attr-- syntax same as JB) ---skipped in case of no attrs.

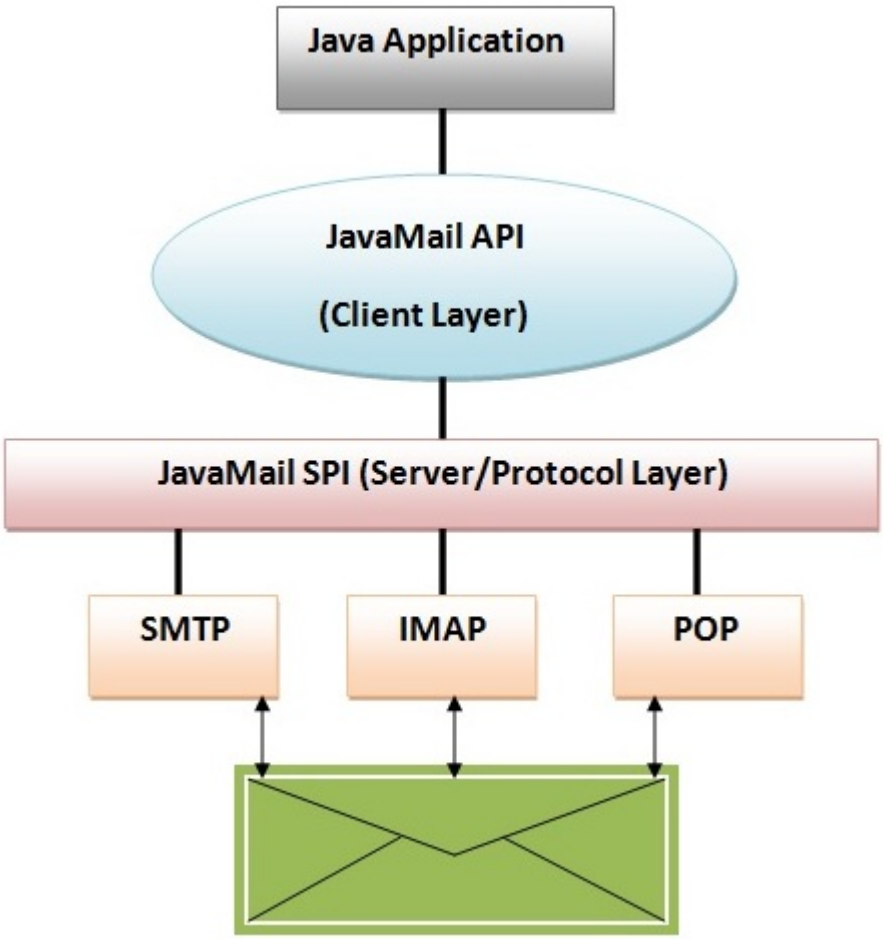
WC invokes setJspBody(JspFragment jspf) --- to xfer body content from page to T.H class --- skipped when body-content empty

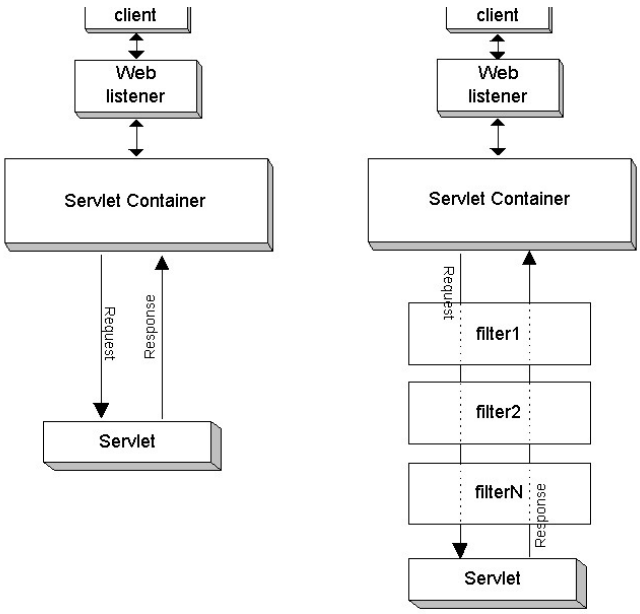
WC invokes doTag() --- for exec B.L ----- doTag rets ---- T.H cls inst GCed.

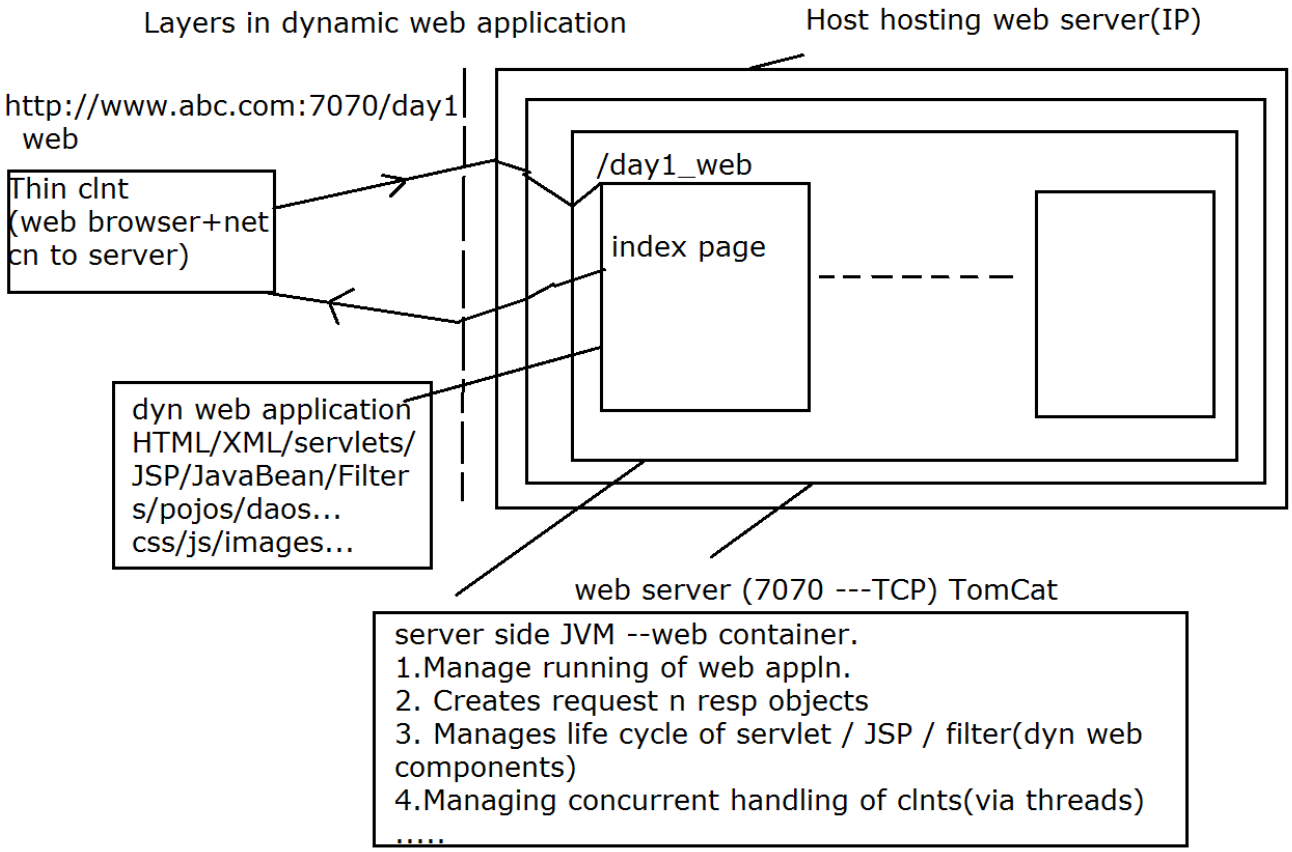


Tx Propagation Scenario









HTTP Resp Packet
Resp status code(200) | Header/s | response data

6.3.2. Query Creation

Generally, the query creation mechanism for JPA works as described in "Query Methods". The following example shows what a JPA query method translates into:

Example 57. Query creation from method names

```
public interface UserRepository extends Repository<User, Long> {

  List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);
}
```

We create a query using the JPA criteria API from this, but, essentially, this translates into the following query: `select u from User u where u.emailAddress = ?1 and u.lastname = ?2`. Spring Data JPA does a property check and traverses nested properties, as described in "Property Expressions".

The following table describes the keywords supported for JPA and what a method containing that keyword translates to:

Table 3. Supported keywords inside method names

Keyword	Sample	JPQL snippet
`Distinct`	`findDistinctByLastnameAndFirstname`	`select distinct … where x.lastname = ?1 and x.firstname = ?2`
`And`	`findByLastnameAndFirstname`	`… where x.lastname = ?1 and x.firstname = ?2`
`Or`	`findByLastnameOrFirstname`	`… where x.lastname = ?1 or x.firstname = ?2`
`Is`, `Equals`	`findByFirstname`, `findByFirstnameIs`, `findByFirstnameEquals`	`… where x.firstname = ?1`
`Between`	`findByStartDateBetween`	`… where x.startDate between ?1 and ?2`
`LessThan`	`findByAgeLessThan`	`… where x.age < ?1`
`LessThanEqual`	`findByAgeLessThanEqual`	`… where x.age <= ?1`
`GreaterThan`	`findByAgeGreaterThan`	`… where x.age > ?1`
`GreaterThanEqual`	`findByAgeGreaterThanEqual`	`… where x.age >= ?1`
`After`	`findByStartDateAfter`	`… where x.startDate > ?1`
`Before`	`findByStartDateBefore`	`… where x.startDate < ?1`
`IsNull`, `Null`	`findByAge(Is)Null`	`… where x.age is null`
`IsNotNull`, `NotNull`	`findByAge(Is)NotNull`	`… where x.age not null`

`Like`	`findByFirstnameLike`	`… where x.firstname like ?1`
`NotLike`	`findByFirstnameNotLike`	`… where x.firstname not like ?1`
`StartingWith`	`findByFirstnameStartingWith`	`… where x.firstname like ?1` (parameter bound with appended `%`)
`EndingWith`	`findByFirstnameEndingWith`	`… where x.firstname like ?1` (parameter bound with prepended `%`)
`Containing`	`findByFirstnameContaining`	`… where x.firstname like ?1` (parameter bound wrapped in `%`)
`OrderBy`	`findByAgeOrderByLastnameDesc`	`… where x.age = ?1 order by x.lastname desc`
`Not`	`findByLastnameNot`	`… where x.lastname <> ?1`
`In`	`findByAgeIn(Collection<Age> ages)`	`… where x.age in ?1`
`NotIn`	`findByAgeNotIn(Collection<Age> ages)`	`… where x.age not in ?1`
`True`	`findByActiveTrue()`	`… where x.active = true`
`False`	`findByActiveFalse()`	`… where x.active = false`
`IgnoreCase`	`findByFirstnameIgnoreCase`	`… where UPPER(x.firstname) = UPPER(?1)`

`In` and `NotIn` also take any subclass of `Collection` as a parameter as well as arrays or varargs. For other syntactical versions of the same logical operator, check “Repository query keywords”.