JDBC - Java database connectivity

20-12-22

-Harshal Deore

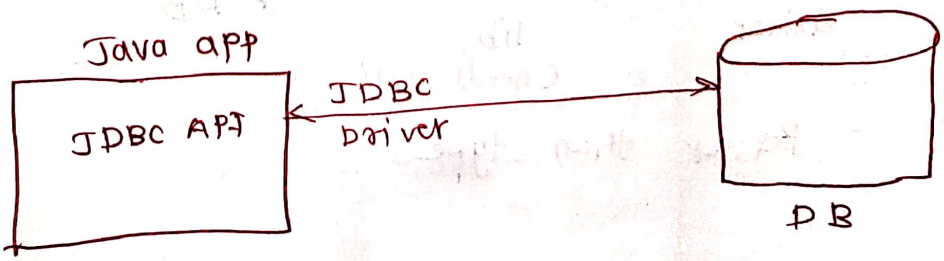
Data - Important part of any application on which it program is driven

- User Interface - Take/show data
- Application logic - Process data
- manage data - Data storage / CRUD

- RDBMS → MySQL
- ~~JBD~~ JDBC is a API which helps to bridge our java program/ application with the mySQL like database. (to transfer data)

(API - inbuilt interfaces, abstract/ concrete classes to interact with the app)

- DB can only understand the language of query
- JDBC executes query for java program to interact with database



- JDBC driver → used by JDBC API, this are developed by database entities.

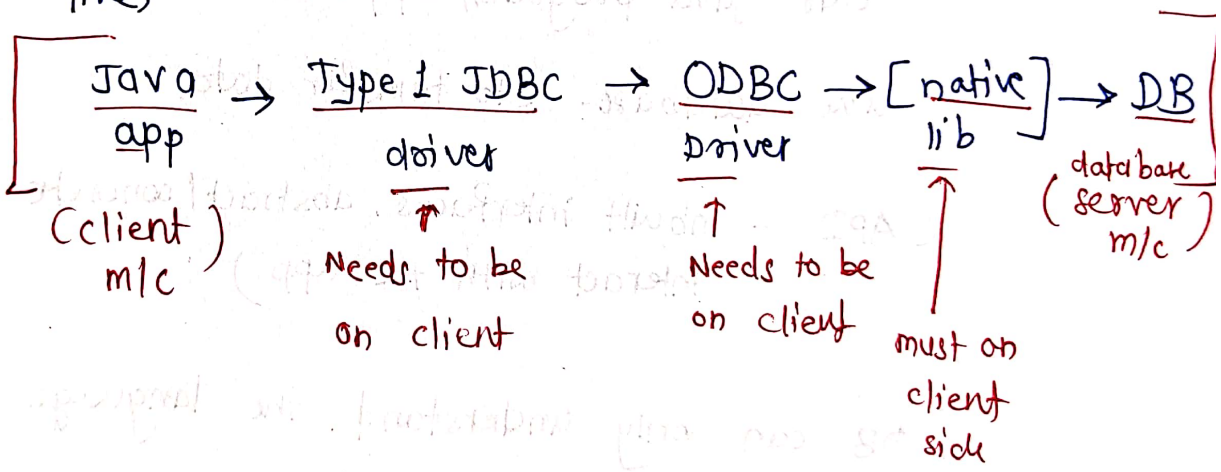
→ Intermediate software to generate real mySQL query generated by JDBC

→ They makes data transfer possible between app & database

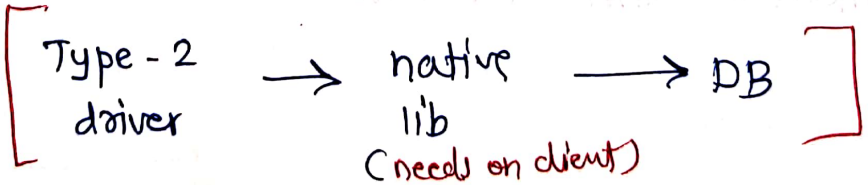
→ 4 types

1. Type 1 Driver [JDBC - Open database connectivity Bridge] ODBC

like,

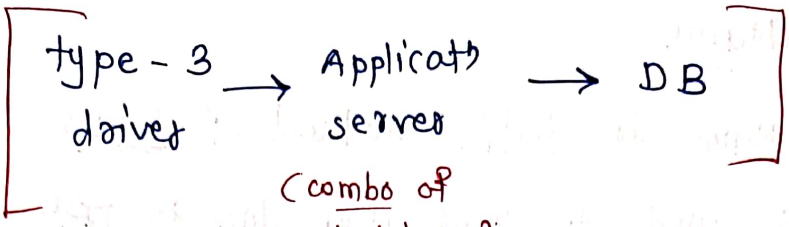


2. Type 2 Driver [java native diriver]



- Faster than type 1

(3) Type-3 Driver [Java network protocol driver]



- No native lib at client side is needed
- Application server enables to connect with different database with same driver
- Faster than type 2
- All databases need to be connected on one network

(4) Type-4 Driver [Pure java driver] *

- Everything is written only in java, it is fastest
- Driver for respective rdbms is different

- Some installation part
  - mysql connector jar file
  - Type 4 connectivity

So, Project → Build path → → configure build path → libr → class → Add ext jar

-Harshal Deore

Check for Eclipse

- Class.forName
  - forName is static method of Class (inbuilt class)
    it is used to load given class in JRE

Syntax - Class.forName("com.mysql.cj.jdbc.Driver");
Connn con = DriverManager.getConnection("jdbc:mysql://localhost:3306/ name of database", "root", "pass");

DriveManger.getconnection - inbuilt
("URL", 'username', 'password')

- Methods to execute query
  - execute() : boolean [for DDL query]
  - executeUpdate() : int [for DML query]
  - executeQuery() : ResultSet [for DQL query]

- CRUD of tables → Practise
- Revisit while ~~loop~~ part

- Drivermanager is a class with getConnection method in it
  ↓
  it gives username pass. database & url adress

21-12-22

- CRUD of SQL

- executequery → compilation & execution of query & result stored in rs

- Query statements

  - statement (Its obj is used to compile & execute query)
  - prepared statement

    (PreparedStatement s = con.prepareStatement ("select * From student)

    - Here compilation is done only once & can be execution for n times
    - AKA precompiled statement
    - using different methods CRUD can be performed

      (s.setInt(1) ← setter method
  - Callable statement

    - To call stored procedures or functions of database

      (CallableStatement s = con.prepareCall ("{ call my proc (?)}");
    - stored procedures

mysql

Funct^n → returns value

procedures ↳ fn with no value return

Metadata in DB

↳ To get the metadata of tables created using metadata method

Resultset (interface)

- selected items from query are stored in one variable
- moveable methods → next(), previous(), first(), last(), beforeFirst(), afterLast().
- getter methods → getInt(), getString(), getFloat(), getDate()

- Properties

  ResultSet.TYPE_SCROLL_SENSITIVE

  ResultSet.CONCUR_UPDATABLE

  - concurrently updating in database

22/12/22

Types of http request

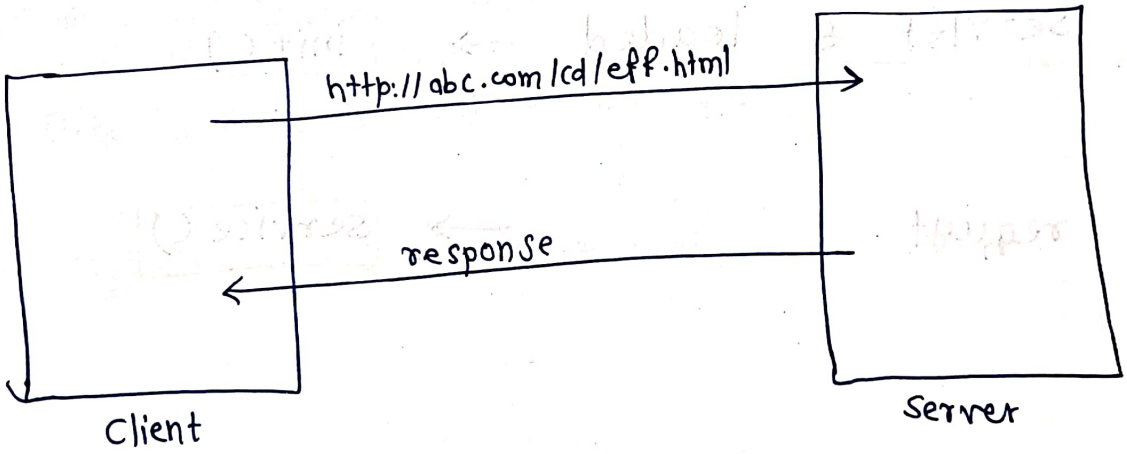
↳ get ↳ post ↳ put ↳ delete ↳ trace

↳ header ↳ options

get → to get data (data is along with URL) from server

post → to give data (data is given seperately) to server

↳ get type of request is catche able



- Java, php, python, are few server side language/ tech.

-Harshal Deore

Java web development

- **Servlet** : To execute the functions on server to provide dynamic webserver
  - A server side java component
  - It enhance functionality of webserver

- making servlet
  1. extend HttpServlet (To handle only http protocol)
  2. extend GenericServlet (To handle diff protocol like http,.....)

utility	methods
Servlet is loaded →	init()
request →	service()
Servlet is unloaded ←	destroy()

Fig: servlet lifecycle methods

- When servlet is loaded it init() method is called

- Each request is served by the calling service() method
  - execution code shall be written in service() method (request processing code)

- destroy() method called before unloading if it is called only once.
  - It shall contain resource releasing code

→ making webapp using eclipse
- Integrating server into workspace
- File → new → server → Apache → Tomcat v8/8.5/9
- making dynamic web app
  File → new → web → Dynamic web project
- myWebApp → src → main → webapp
  - new → HTML file
- servleapp → Java resource → package → class
- servletLifeCycleApp

- Container (J2EE container)

↳ responsible to manage servlet operations, creating operations, objects

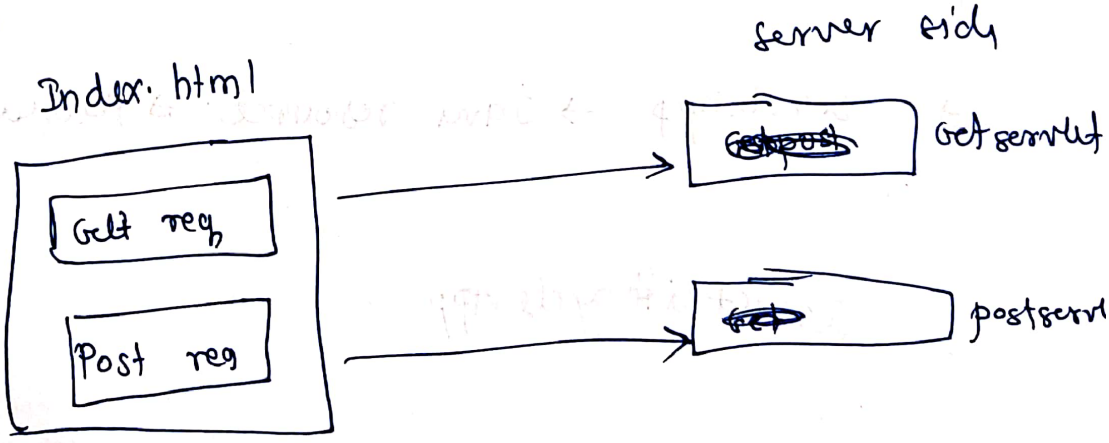
- Designing servlet to handle only particular type of request

↳ Using request processing of HttpServlet (inbuilt interface)

- service() - for any request
- doGet() - for get type req
- doPost() - for post
- doPut() - for put
- doDelete() - for delete
- doTrace() - for trace
- doOptions() - for options
- doHeader() - for headers

use method according to need

- program



- Servlet Lifecycle → java → ~~src~~ Key → main → webapp

↳ create HTML page

↳ Servlet → Getservlet.java

—

Servlet wizard → Java resource → s

— servlet lifecycle

↳ myservlet.java → extends HttpServlet

→ override service(), init()

destroy()

→ check with sout

→ refer program comment

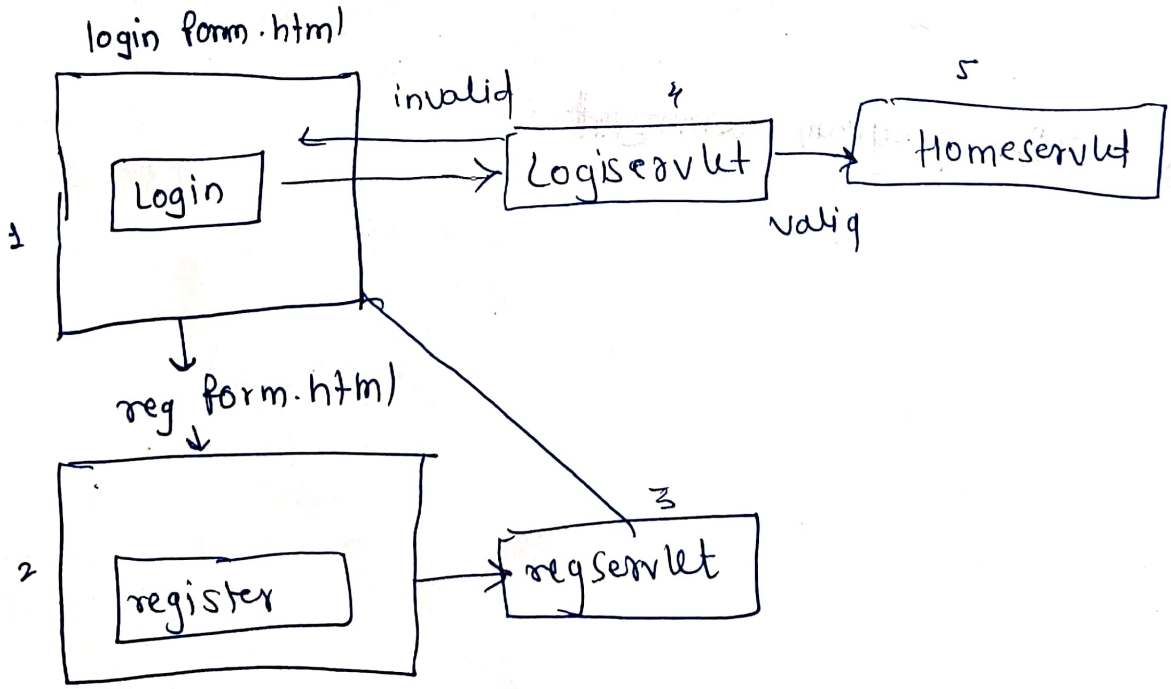
23-12-22

HttpServlet object → created by container & all the inform of request stored in HttpServlet ~~Response~~ Request object

HttpServletResponse object (contains header info, user data, req related info)

- contains all method used to generate response
- sending & recieving response (all data)
- setting response characteristics (type/length)

- HttpServletResponse & request are inbuilt interfaces
- Integrating jdbc & servlet



To use jdbc in servlet driver jar file need to be connected

↳ src → main → webapp → webINF → lib

① HTML file → login → Regi

② servlet RegServlet → using wizard

③ Code in dopost method of regservlet

↳ unm, upass variable

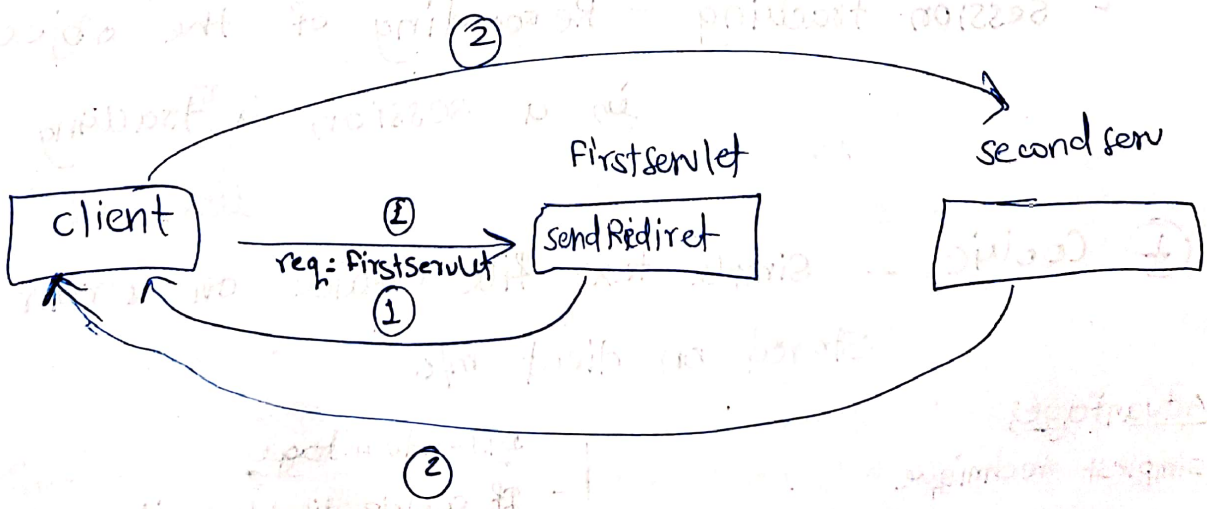
↳ jdbc conn & query running

↳ set string method (1, unm)
(2, pass)

↳ calling int i = s.execute update methode

then response. method

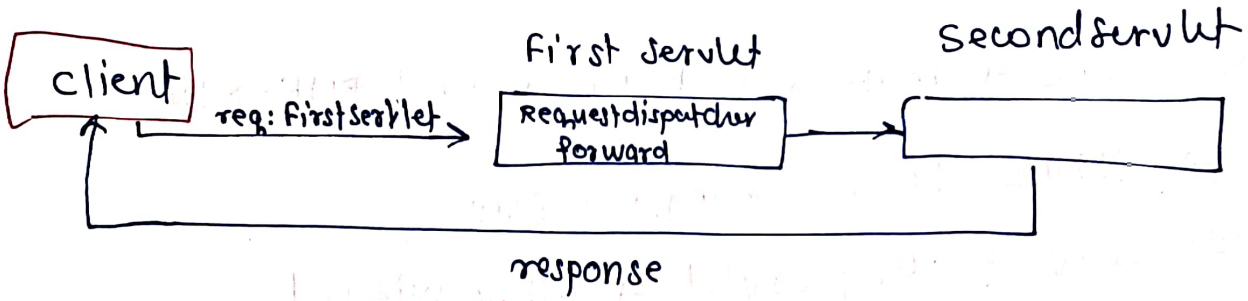
- Senddirect approach



request object is created

-Harshal Deore

Requestdispatcher



Ex: Enter a password then access site

- According to need we can use requestdispatcher or redirect approach

Servlet Session Handling

- Http is stateless protocol (Each request considered as new request)
- session means a particular interval of time
- In session mgt we maintain state (data) of an user
- session tracking - Recording of the object in a session is tracking

① Coolie - Simple text file created on server & stored on client m/c

Advantages
- simplest technique
- maintained at client side

Dis-advantages
- If cookie disabled - it wont work
- only textual info

24-12-22

(2) Http Session

- An object which is one per client
- It has one unique ID known as session ID
- can store data in key value pair
- getSession(boolean flag) : method to get session object

getsession(true) : existing session is provided & if object dont exists it returns null

- Copy of cookie app → SessionApp
  - keeping electric as it is
  - removing cookie part from other 3
  - using Http session object
  - using setattribute method & get attribute method
- key value is assigned to session ID

request.getSession(false/true)

- false: Allow user to start from particular page (keep it for login)
- true: to allow user to start from any page (true for login)

-Harshal Deore

- It is more secure method to achieve tracking

Servlet Scopes

1. request (HttpServletRequest)
2. HttpSession
3. servletcontext

1. HttpServletRequest

- one per request
- created & destroyed when request is recieved & responded

2. HttpSession

- one per client
- create & destroy using code

3. servlet context

- one per application
- when app is deployed
- destroyed when an app is un-deployed

- WEB-INF

↳ It is private folder, can not be accessed by client directly, it contains web.xml file

JSP

↳ Java server pages (It is server side java component)

↳ view code & java code are written seperately (java code written inside view code)

↳ Document has two components → ① Text document (to write code)

② jsp component (to write java code)

view

- JSP elements

26/12/22

why JSP ?

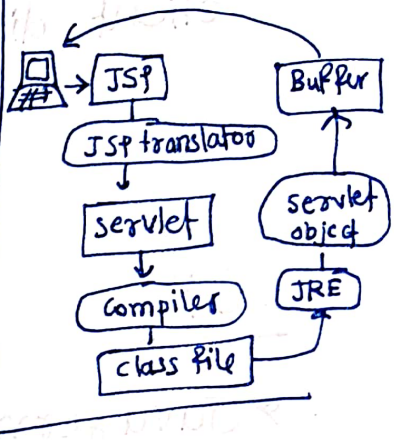
↳ In servlet view code writing is complex thing, in JSP view code can be written seperately along with java code.

↳ Easy to main,

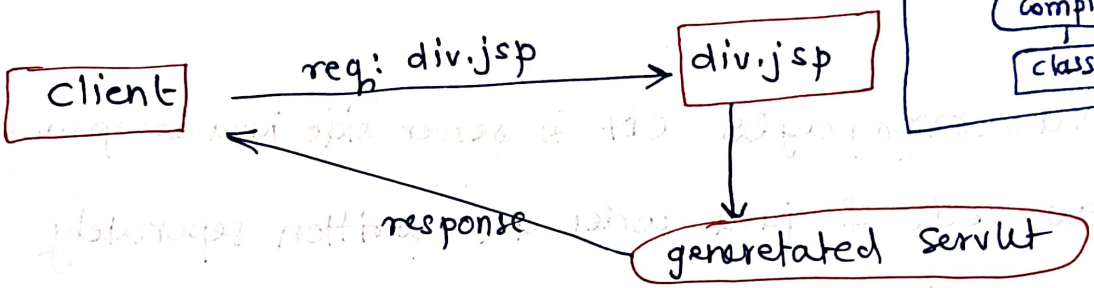
↳ fast developement

↳ Applicat is simple to use

- Writing JSP
  1. Description
  2. scriptlet i.e. method
  3. output



- Process Flow (JSP Lifecycle)



↳ JSP engine converts JSP request into servlet, generated servlet creates response

- JSP implicit objects (This can be directly used)
  1. out : JSP writer
  2. request : HttpServletRequest
  3. response : ——— response
  4. session : HttpSession
  5. Application : ServletContext
  6. page : represents current page ('this' type)
  7. config : ServletConfig
  8. pageContext : PageContext (get scope)
  9. exception : Throwable [only in jsp error]

JSP directives (messages that tell container how to translate JSP page into respective servlets)

syntax : <%@ directive-name attribute = "value" %>

3 Directives - 1. Page
2. include
3. taglib

→ Page (It defines attribute that apply to an entire JSP page)

- buffer in jsp → it specifies the buffering characteristics for the server output response object

  Ex: IF buffer = null, servlet output will be immediately directed to response object

- Error page → For except handling
- session → For availing session object

Include directive

↳ To include the content from certain page in my page

- taglib directive

↳ Used to include tag libraries, providing external tag libraries used in jsp except html.

Java bean

- Simple java class having private properties & public setter / getter methods for every property
- A bean encapsulates many objects into one object so that we can access this object from multiple places. It provides easy maintenance

1. Usebean

- used to keep the bean object in any jsp scope and use that bean object in jsp

Syntax : `< jsp : useBean id = "bean-id" class = "bean-class" scope = "jsp-scope" > < /jsp : use Bean >`

Eg: `<jsp : useBean id = "emp" class= "model. Employee" scope = "session" > < /jsp : useBean>`

- If session scope contains Employee object having key (id) emp then it (bean object) is given in jsp. But if the bean object is not present in scope for given key then new bean object is created, stored in the given session & provided in JSP.

2. Set Property

- this jsp action stores request parameter data in bean properties

Syntax: <jsp : setProperty name = "bean-id" param = "reqst_param"
property = "bean_property" > </jsp:setProperty>

3. getProperty

- this jsp action is used to show the bean property values

<jsp:getProperty name = "bean_id" propert = "bean-propert"
</ >

27/12/22

- Design pattern → A way of writing the code

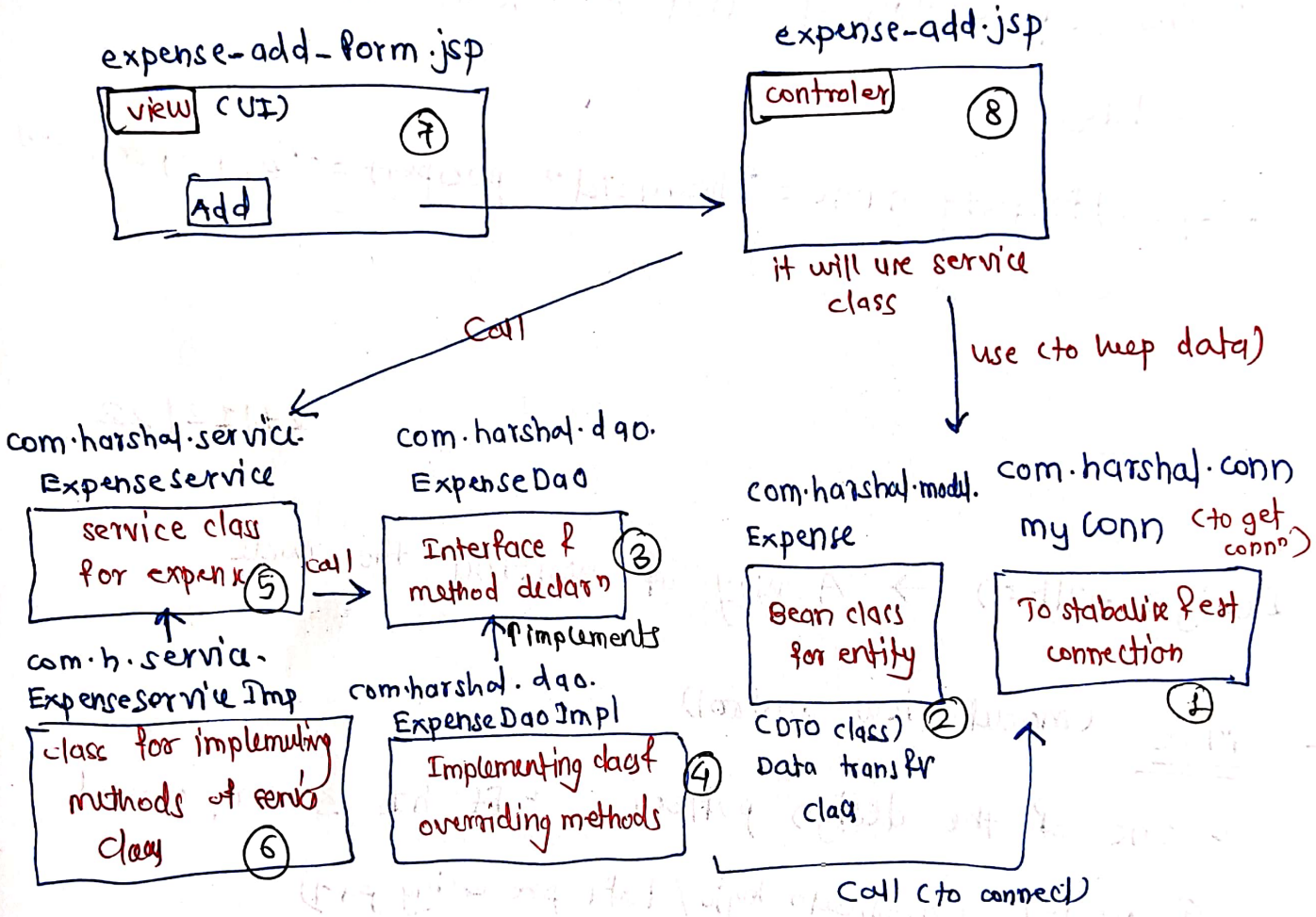
- MVC (model view control)
  - one of the design pattern
  - It has 3 component

  (1) Model (Applicatn logic / data processing part)
  - It contains only application data, not includes concern of UI

  (2) View (UI)
  - Presents model data to user, does not include manipulatn part

  (3) Controller (flow of data betn model & view)
  - It listens to triggered events and executes appropriate reaction

DAO (data access object)



Steps

- Following numbers, adding conn file to lib folder
- 1) Jdbc connectn core (Required- Driver class name, u, pass, URL)
  - ↳ getCon() method
    - ↳ for creating connection

2) Expense bean class

- defining variably & generating getter & setters method
- tags- usbean, set property.

3) Dao Interface (Data access object

- creating Dao interface
- most jdbc code is written here
- For CRUD of database table
- creating implementation class with methods required according different required queries.

4) DAO implementing class

- Implement above DAO file
- override method using eclipse
- making constructor for myconn object

5) Creating Service Interface

- service class are used to take service of any other class
- To insert record here DAO class method are used
- To call them add() method will be used

6) Implementing class of Servia Interface

- DAO object created here to work on add() method

7) webapp jsp file

- form for expense_add_form
- HTML UI

8) Controller JSP file

- Expense_add.jsp
- by default scope is session
- setProperty of JSP
- tags → useBean, setproperty
- calling jsp bean
- calling service method

```
<%
obj → = new ExpenseService
      method → add
%>
```

↑ here

- It is taking data from view & given to service

Selecting Record

→ DAO & service class depends upon entities, per new entity both DAO & services ~~out~~ class will be created

- New method to existing entity can be generated using method to DAO & Service classes

28-12-22

- Update op^n

(1) when update button clicked → request for expense-upd.jsp generated

(2) control goes to exp-up.jsp → to store data in bean obj
< usebean tag used

↳ Expense obj generated with modify method service
& passing object

↳

(3) control goes to modify() method of servic
↳ then update() method of dao is called

(4) control goes to update() method of DAO
↳ Connection ↳ executing query
↳ setting methods of preparedstatem.

↳ Record will be updated here

(5) It will go to service class & then to update class

(6) response.sendirect() send data from here to expense_list.jsp

(7) All records will be selected & displayed

Login / Logout mechanism

- Only authenticated user shall have access to particular page

- Registration

Entity : User

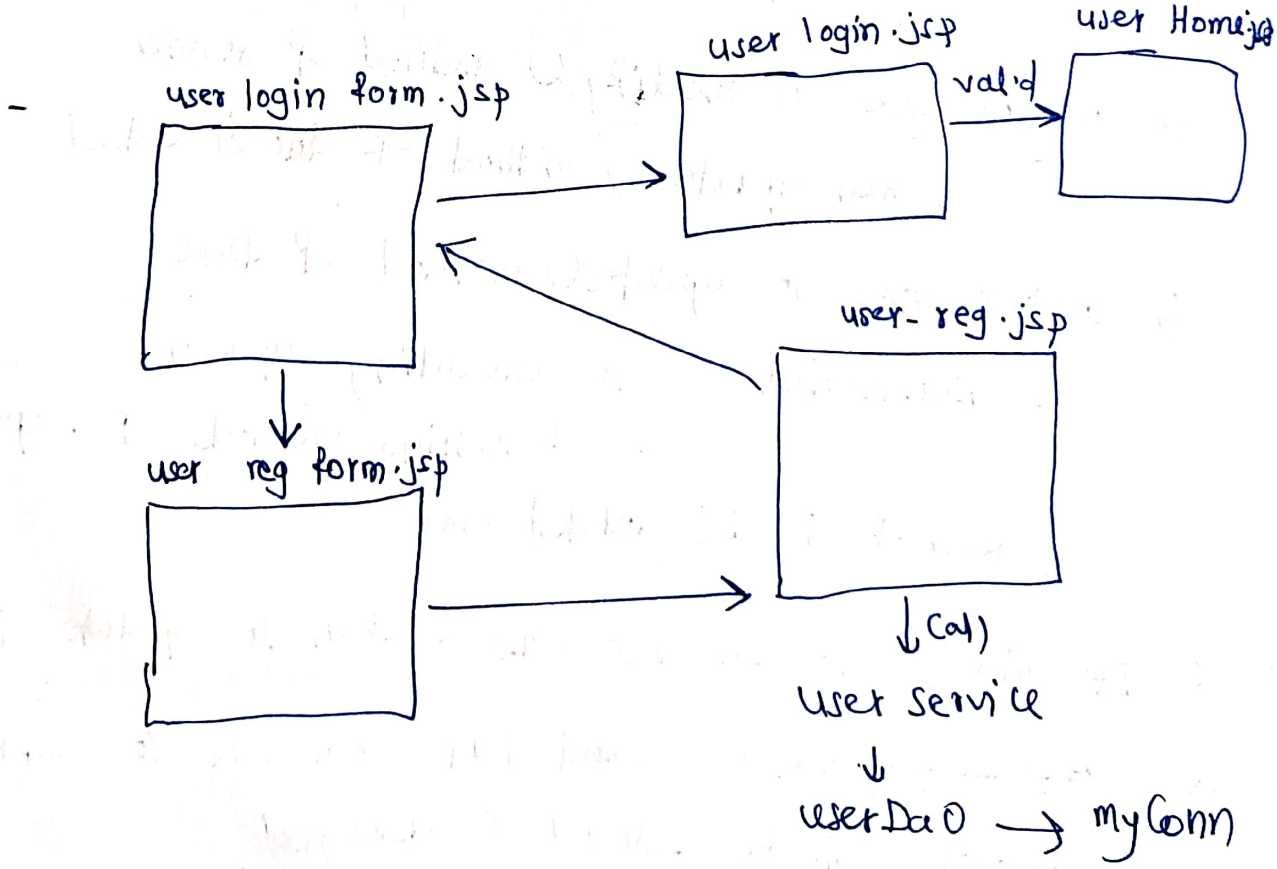
can : Register, login, CRUD , logout

Entity : user

can : uid : int , upass : string

uname : string

- (DAO & service + Bean class) done



29/12/22

- Taglib directive

- It is used to include library which has tags written
- Ex: JSTL - Jsp standard Tag Library

ORM (Object Relational mapping Model)

- A programming technique that transfers data between oop's language & RDBMS. This transaction is object transaction.
- Class is mapped with dbtable
- ORM tools → hibernate, iBatis, topLink

Hibernate

- It is an ORM tool
- Features

1. Database portable → same app can work with diff db
2. 3 fully featured query language
   1) HQL - Hibernate QL
   2) Criteria API - Hib writing query for us using inbuilt meth
   3) Native Query → method Dbs

3. Hibernate supports oop's features like inheritance, aggregation, polymorphism and java collection fw.

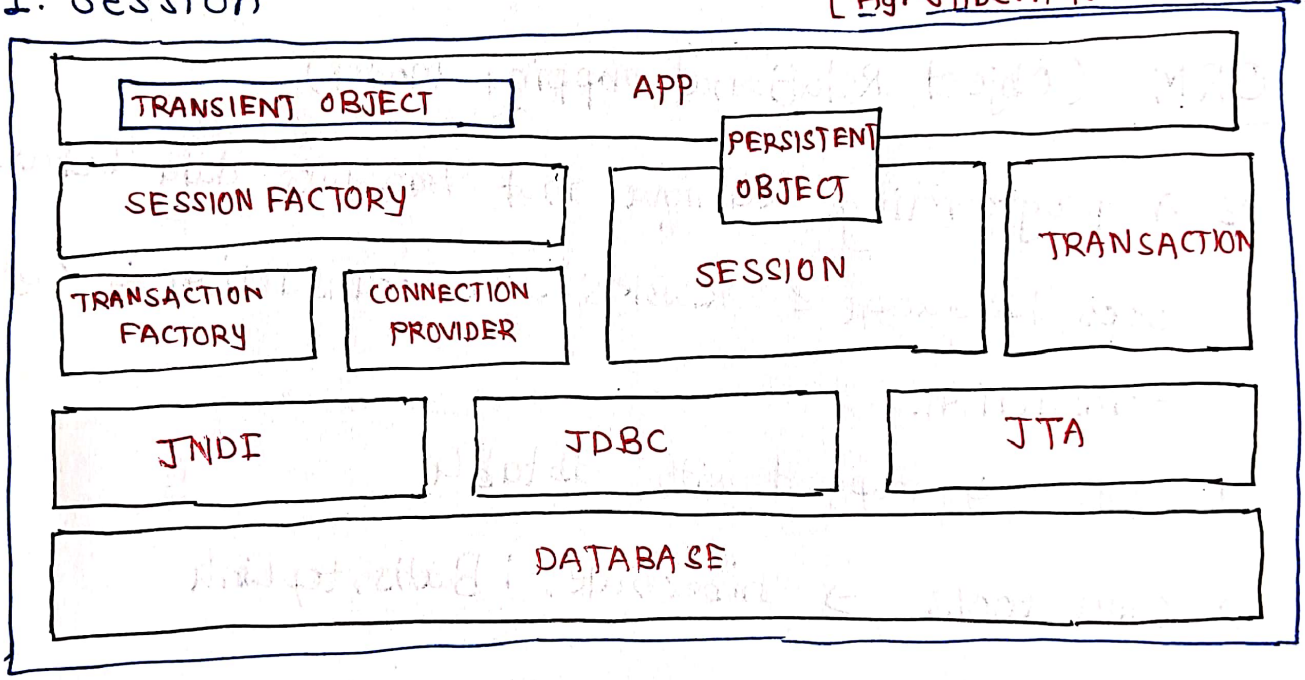
4. Free / open source

5. Key values can be generated automatically using automatic key generatn classes

Hibernate Architecture

1. Session

[Fig: Hibernate Architecture]



Application → Java program developed by us for data transaction

Session → It is widely used in WebApp due of data transfer is stateless
→ server shall identify user for time being to send / recieve particular data
→ Java → Client Database → Server

Ex: Http session

Steps to create hibernate

1. create java bean class (entity) (Hibernate entity)
2. write hibernate annotation in bean class for mapping it to db table
3. make hibernate configuration file - hibernate.cfg.xml
4. java main class

Continued ....

- use of session to particular object for particular period
- It is single threaded hibernate object (autogenerated which represents time period for data transaction

Transaction (similar to TCL)

- In jdbc trans. is autocommit, here it is manual
- It is hibernate object which represent unit of work (like Insert, update, delete quer) which is saved to db
- single threaded
- (org.hibernate.Transaction)
- explicitally need to be commited

-Harshal Deore

SessionFactory

↳ This is factory / pool for the hibernate sessn object

↳

Transactn factory

↳ It is a pool of transaction objects

↳ created & provided from here

Connection Provider

↳ It provides connection object to session / hiberate

↳ creates jdbc connnectn object & provide to session factory it further conner

JNDI (java naming, directory Interface)

↳ Data source are used to stabalise connection

↳ It is available in servers inbuilt

- Naming → Naming of diff methods, obj stored
- Directory → Name along with attribute

JDBC (java database connectivity)

JTA (Java transaction api)

↳ for transactn mgt

Transient object
(bean class obj)

- Object not attached with hibernate session

Persistent object
(bean class obj)

- objected attached with hibernate session

MAVEN developement Env

- Normal java cycle → app development → java → include jar file → compile source file → create jar/war (bytecode is here) → run program

- Different commands for diff ops in command prompt usually
- In IDE file created/edited & directly Run
  - These commands are driven by development envi/tool
- like MAVEN, ANT, GRADLE ← provident by IDE (Integrated Dev. Environment)

Dev. Environments

- External jar files → In MAVEN i.e. maven dependencies

↳ MAVEN is used widely for java dev.

↳ GRADLE is mostly used in Android ~~development~~ developement

– Creating MAVEN Project

① New → New MAVEN project → Tick the Create a simple project (skip archetype select)

② Group ID (to identify project uniquely)

③ Artifact ID (to identify jar file / Applicatn / module

↳ pom.xml (Project object model)

↳ MAVEN dependencies are written here & they are added to our project from here

④ Bean Class → Constructor & getter setters

↳ Importing hibernate @ Entity (persistent)

@Entity ↳ to qualify class as hibernate bean class

@Table ↳ to map bean class with particular database tables

@Id ↳ to identify property i.e. mapped with primary key of table

@Temporal ↳ for date type of db mapping

⑤ Creating cfg file (for hibernate configuration)
in src/main/resource

↳ creating hiberate cfg.xml → Hibernate version 4.3 selecting

↳ Database Dialect - mysql (configuring jdbc)

↳ Then properties → hibernate → show sql : True

→ Hbm2ddl Auto : create

↳ mappings → Add → Class → Employee.

↳ Source → delete name = " "

⑥ main Class

↳ creating config Obj

30/12/22

↳ Dialect → Allows hibernate to generate SQL optimised for particular relational db
→ mechanism which converts hibernate call into database qury

↳ hbm2ddl → create → drop & create
update →
create-drop → drop-create-drop
validate → existing table will be validated with given schema

-Harshal Deore

<mapping → to specify bean class

- All classes as per entities need to be mentioned

HQL : Hibernate Query Language

- Similar to SQL
- bean name is used instead of table name & bean properties instead of table
- bean & property name are case sensitive

hibernate.cfg.xml

- we are providing inform to hibernate as a developer
- connection object will be created automatically

standard service Registry Builder

- Builds all the objects needed like jdbc, sessn factor for the program
- and all this object will be available in standard service Registry object

[Hibernate is just like alternative to jdbc]

-Harshal Deore

get() & Load()

↳ both are used to create object for ~~the~~ row item

get() → returns null if data doesn't exist

load() → will throw objectNotFound objection

- CRUD ops finished (main1 - 5)

HQL (Hibernate Query Language)

Criteria

↳ Hibernate provide criteria interface to write the query.
Criteria methods are called for any query.

Native Query

↳ Queries written in used database

Named Queries

↳ Queries can be ~~written~~ given so that it can be used multiple times in program.

↳ Types → 1) Named HQL
2) Named Native Query

Inheritance in hibernate

- Hibernate offers 3 inheritance strategies
  1. table per class heirarchy
  2. table per subclass
  3. table per concrete class

1. table per class heirarchy

- only one table for all bean classes in inheritance
- columns for all properties of all classes will be in one table

2. Table per subclass

- In this table is created for every subclass
- This tables are attached using primary keys & foreign keys
- @Inheritance (strategy = InheritanceType.JOINED)

3. table per concrete class

- It is used when super type is interface / abstract class
- Tables are created for concrete subclasses

y records are managed in completely diff tables

Association Mapping

y Tables will be designed in relationship according to need

y In hibernate everything is the form of object, object should be related. For this classes of this objected shall be related in hibernate

Need?

y Storing multiple entities in single DB having some problems ⟶ 1) Data Reduadeny (Duplicath of data)
2) Data maintainance problem

- Associative property y mention in ~~one~~ bean classes
- @ OneToOne annotation y ured
- @ Join column (unique = "true")

Hibernate cache strategies

1. First level cache [session level cache]
   - For same id ~~object~~ data is selected from database only 1st time then for other call same created objected is given, not selected again
   - object is maintained throughout the session

2. Second level cache
   - Record selected (object created) in one session is available in other sessions. (not selected again)
   - to enable the second level cache, cache providers are required

---

IOC technique (Inversion of Control)

- In this instead of calling method to get dependencies, our method is called to provide dependency.
- Normal flow of program is inverted

DI - It is process of injecting dependency

- 3 types of DI — 1) setter DI
  2) Constructor DI
  3) Interface

IOC container

- Dependency creation & supply

Spring

- It is framework for making bean based application that works on IOC programming technique
- If we want food, we dont need to cook, it will be provide 😊

Framework

- Refere framework diagram from phone/internet

Modules

Core container module

- Beans & core - submodules responsible for bean creation, initialisation, instantiation, assembling the bean

- Context & Expression language (EL)
  - context works for beans & core to provide the dependency in more framework way
- EL is used to navigate a spring components [spring beans]

Data Access / Integration

- components - JDBC, ORM, OXM, JMS, Transactions
  - spring provides layer to ehnance jdbc etc
- OXM - mapping data transaction betn xml & application obj. (obj. r/s mapping)
  - for data connecting
  - Tags can be defined
- JMS - Java messaging service
  - message based data transfer
- ORM → To use hibernate at much ease & efficiency

Web

components → web, servlet, struts, Portlet

- Strut - Framework like spring

AOP

Aspect oriented programming (will be discussed further)

- used to decouple program

Test

- for testing tools

creating project

- MAVEN Project → complete POM file by adding dependency
- Bean class created (to map db with class)

03/01/23

Spring bean → any object in spring fw that we initialise through spring container

Spring Bean Scope

- when object is created & how many objects

1. Singleton Scope
   - bean object is created
   - Single object is created & every getBean() call will get same object

2. Prototype Scope
   - bean object is created when getBean() is called
   - every getBean() call will get the new object

3. request
   - object is created for http request
   - each http request will get new object
   - this scope is only for web app

4. Session

- object is created for http session
- each http session will get new object

Hello Spring program

<property tag - ApplicationContext → Subtype of context

- This is IOC container (spring IOC container)
- So cfg.xml will be stored here
- spring will read cfg.xml file

- ApplicationContext appCntx = new ClassPathXmlApplicationContext("cfg.xml");
  - creating IOC container

myBean mb = (myBean) appCntx.getBean("my");

- To get object from IOC container
- getBean is method of IOC container
- appCntx is container object

Autowire

- Injecting dependency automatically
- Types ① byName ② byType ③ Constructor
- If is written in <bean ____ autowire = "byName" >

- If tools are not working Copy cfg.xml file to resources

spr - setter - di - app

- cfg.xml file thing
- Then creating bean class
- Then cfg.xml → adding spring bean class entry here
  - using bean tag
- property tag → Spring IoC container will call setmessage method / to inject dependencies

setter - di - 2

- In property tag, instead of value, ref shall be written when dependency is of another bean class
- 2 bean entries → for myDao object will be created & saved with id = dao in IOC conter
  - for 2nd bean tag myservice object will be created & with id = serv will be saved in IOC container
  - so IoC container have id = dao & serv.
  - here myDao is created by IOC container
  - service obje is also created by — spring

-Harshal Deore

Setter DI

- setter methods are called to inject the dependency
- <property> tag is used for setter DI

Constructor DI

- parameterised constructors are called to inject the dependencies
- <constructor-arg> tag is used here

Interface DI

-

p-Name space

- Another way to write Setter DI
- property tag is not used instead attribute p:<property> is used

Autowire

types ↓

1. byName : property name & bean id must be same so that dependency can be injected by matching name. Here setter method is called to inject dep.

2. byType : property type & bean object type must be same so type is matched. setter method injects

3. constructor - Parameterised constructor is called

Annotations in Spring

@ Component → For any spring bean class

@ Service → For service bean class

@ Repository → for dao bean class

@ Controller → for spring web component

04/01/23

Spr-jdbc-app

- Here we are using DAO class for Implementn
- No need to write jdbc code → we will use
  - @autowired jdbcTemplate object
  - using this object will do CRUD
    - jdbcTemplate.update(Query)
      - save/update/delete
    - jdbcTemplate.query(Query)
      - for select
      - using resultset here

using @repository annotation

- Service Interface & then implementing class using @service annotn
  - @autowire carDao for dependency injection
- main method
  - getting carservice object

-Harshal Deore

- ApplicationContext appCntx ← spring IOC container object created
  - here in cfg.xml
    - bean entry of driverManager is done
    - jdbc template is created for jdbc functlty
- CarService object is taken by getBean method

Required dependencies

Class		Dependency
CarService	→	CarDao
CarDao	→	JdbcTemplate
JdbcTemplate	→	DataSource

spr-jdbc-app 2

- same project using preparedStatement
- RSet extractor
- value will be given as array of objects

spr-jdbc-app 3

- select query in different way
- using row mapper approach

Paramameter of query() method of j.template

↓

ResultSetExtractor

- A functional interface has a method that takes one callback function to provide the result set object for the given query.
- It returns the type specified by developer

↓

RowMapper

- It returns the list of type specified by developer

Spring ORM

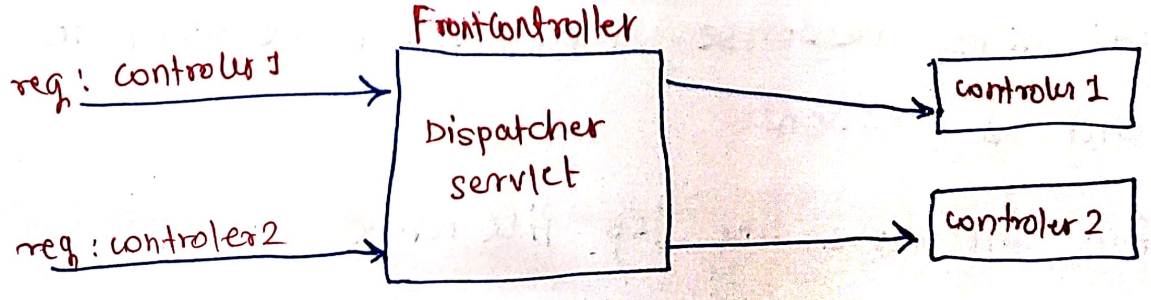
- to use ORM tools in application like hibernate,
- hibernate functionalities are given through hibernate template class
- ORM is programming technique that transfers data bet rdbms & java app

-Harshal Deore

Spring - orm - app

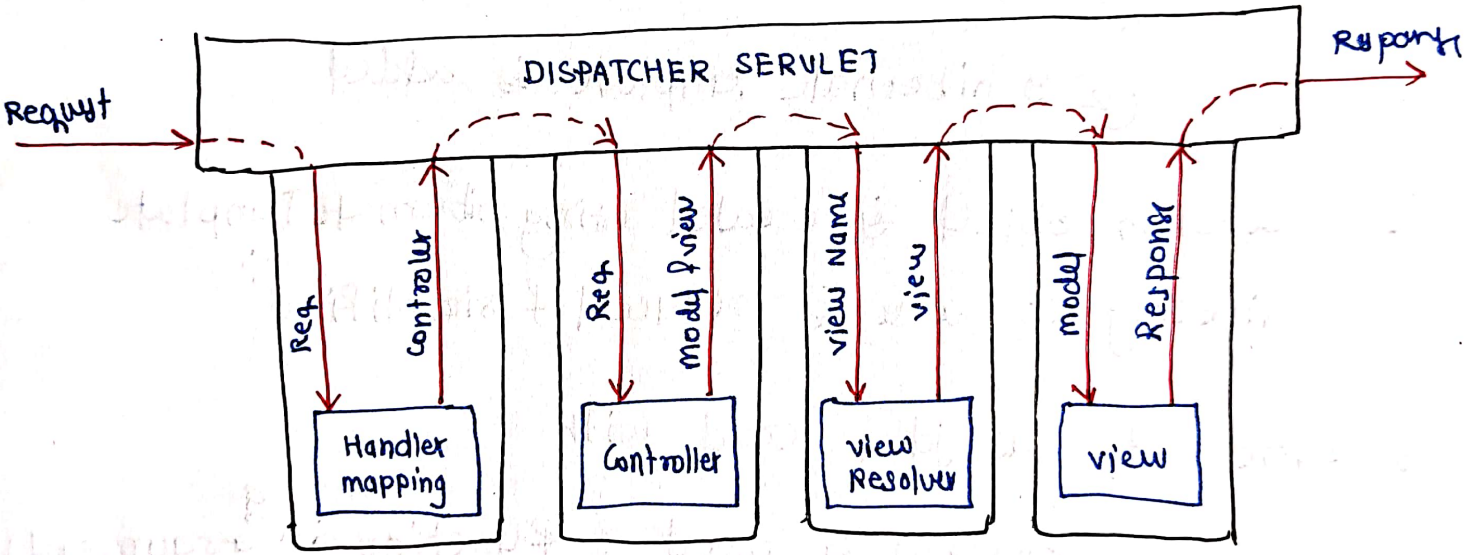
- Complete pom file by putting hibernate dependency
- In cfg.xml → jdbc template class + other bean classes
  1. sessionfactory → hibernate 4 autowiring it by type
     - properties for mapping classes'
  2. hibernate template is added
- session object is created using hibernateTemplate directly so code is reduced & simplified
- Same code as jdbc coud with
- lambda expressn is used (Function as argument)
- In Funcn body hibernate code is written

Spring MVC



-Harshal Deore

- Request will be recieved by Dispatcher servlet (DS)
- Then DS will forward it to Handler mapping.
- Handler mapping will see if URL is ~~matche~~ mapped with which controler
- Then DS will know that coming request will be handled by which controler



- Controller will be designed by us / developer
- Controller will process code & returns model And View (MAV)
- MAV means response data / object we want to send as response
- view will show like jsp file name

- View resolver will tell according to view name to which jsp it is linked
- Then DS will provide it to view & view will generate response.

Spr - hello - mvc - app

- Dynamic web project, check .xml file generator
- configure → convert to MAVEN project
- servlet entry int web.xml → Dispatcher servlet
- dispatcher-servlet.xml → mentioning base package
  - bean tag for view resolver
- creating controller class → annotation @Controller
  - method to print
- jsp page for view

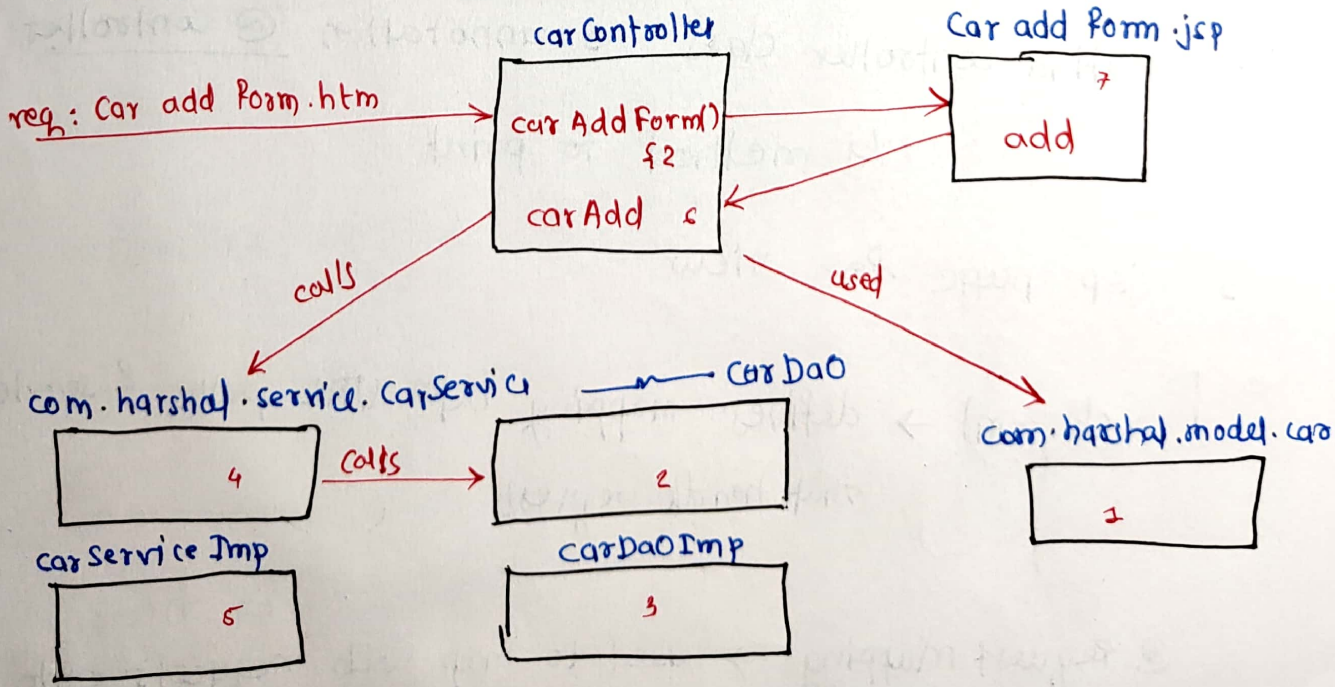
web.xml → defines mapping betn URL paths & servlets that handle requests

@RequestMapping → used to map web requests onto specific handler classes / methods

Flow → request generated from hello.htm, then it is recieved by Dispatcher servlet

- DS will find out controller for it, @controller tag will be recognised
- In controller method is checked which is tagged with @RequestMapping - hello.htm
- In this method to return something in response parameter of ModelMap type created (inbuilt class)
- In model data is stored in key-value pair
- return info : info is view name

Spring - crud - mvc - app



-Harshal Deore

05/01/2022

Thumb Rule while devl. appln

- Req → view [page is not using spring tags]

- Req → Controller → view [If view page is with spring tags / need any data from spring]

crud - App

↳ UI - car-add-form.jsp → <spr : form tag used here

↳ carController method created in controller
↳ @ controller annotn used
↳ It is mapped to URL /car_add_form.htm
↳ control goes to car_add_form.jsp → To maintain data from form

↳ Spring will provide modelMap object

↳ path → will do a job of name & value attribute

[↳ 1 entity → 1 dao → 1 service → 1 controller]

↳ @requestmapping → Specifying requestmethod.POST
↳ carAdd() ← method for create
↳ (Car car , modelMap model)
↳ main class obj

↳ @autowired service dependency will be there
↳ add(car) method ← dao will be called & data entry

-Harshal Deore

- model.put ("car", car) ← to keep same data
  - model object
  - going back to car_add.htm
- return car_add_form.jsp ← control goes back here
  So same page is shown to user

Read

- req: car list.htm → carController, carList(){}, cmethod in() → car list.jsp (view)
- @RequestMapping ——— "/car-list.htm" ——— post
  public string CarList (ModelMap model){
  ↑ No car object needed as not receiving any data
  List.carService.getAll();
  lst
  model.put ("cars", ~~car~~ lst);
  - select list will be kept here
  - using cars key keeping the list
  return "Car_list";
- generate car-list.jsp
  - first get data as list of car using req.getAttribute
  - display the record in html
  - html table with car.get method to fetch data

-Harshal Deore