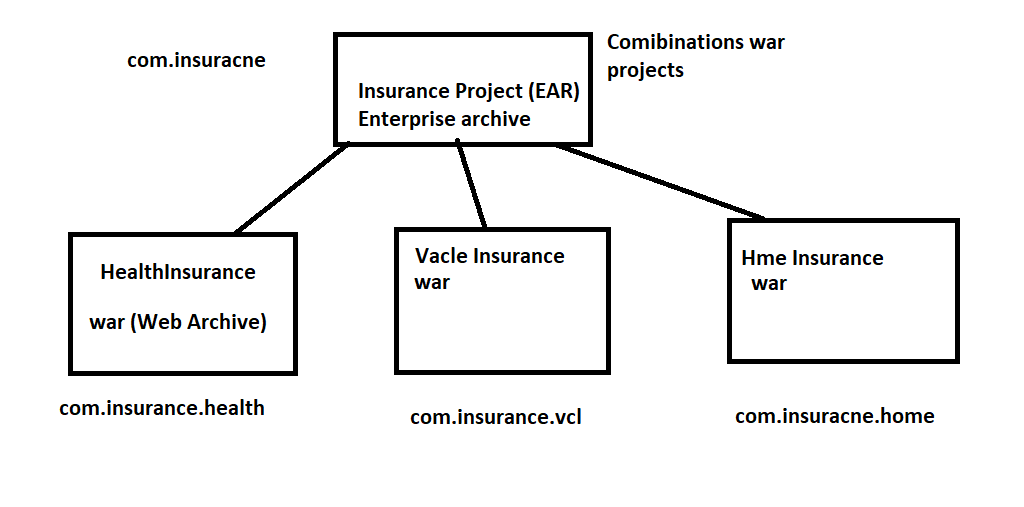
**Maven:**

* It is a Framework tool for building the project.
* Maven is a powerful **project management tool** that is based on POM (project object model). It is used for projects build, dependency and documentation.
* It simplifies the build process like ANT. But it is much more advanced than ANT.(build.xml), we need to provide start and stop of execution, jar location in local.
* WithOut Maven tools if you want to create a project , we have to add external jars for all api whatever you are developing it will be hard to search and download the jar files every time.
* example: if we want to work with Spring Framework for IOC we have almost 14 jars and Spring JDBC we have 5 jars like the same way for every structure. If we are downloading the jars and keeping them as external in the project it will be hard to deploy the project as well.
* What is Build Tool:

A build tool takes care of everything for building a process. It does following:

* Generates source code
* Generates documentation
* Compiles source code
* Packages compiled code into JAR of ZIP file
* Installs the packaged code in local repository, server repository, or central repository
* <https://mvnrepository.com/>
* the same repository of project how you cloned in your local workspace and doing work, the same way we have an maven repository where we can download all the related jars from that repo.
* we need to update the required jar file into pom.xml
* every jar file which you required it called as dependency.
* we can mention this dependency with in a **dependencies** in pom.xml by adding the groupId, ArtifactId, version etc.,
* groupId is nothing but a package name for that jar.
* artifactId is nothing but a project name of that jar.



* When you selecting maven structure please select webApp so that it will create web application module.
* Steps to setup the maven project in your local workspace.
* import through the maven existing project, once it was clear
* Update maven project forcefully.
* do maven clean, it will clean all the pom.xml dependency jars.
* do maven install, when you use this command it will download all the required jars from repository to local machine.
* dependency jars always located in local machine {user}/.m2/repository
* src/main/java is used fot java classes
* src/main/resources is for properties file can be stored.

**SpringFramework:**

* Spring is an framework , were we have all built in logics.
* Spring IOC: core of spring framework
* Spring JDBC: replacement of JDBC with framework templates.
* Spring MVC: Project Architecture.

**SpringIoc:**

* Spring IOC is a core module.
* IOC is nothing but inversion of control, which is nothing but a container.
* container is used to store the object instance.
* whatever the object you’re injected all the objects will be holding in IOC container.
* Injection, object configuration
* Type of injection are:
* **Setter Injection**
* **Constructor Injection**
* **As per core java whatever the object you’re creating it was with new keywork.**

**Example:**

Class Address{

}

class Employee{

Address address;

}

my class employee always dependent on address, so whenever you want to create the employee object we need to passs the address object, so at the time of employee object creation it was dependent on address object.

the above scenario we can call it as tightly coupled.

* with the help of DI (Dependency Injection) we are making objects as loosely coupled.
* by injecting the object with setter or constructor.
* we are going to inject with applicationContext.xml
* if we are going for setter injection, we will use property tag to inject the properties values.
* if we are going for constructor injection we will pass the property values in constructor arguments.

To Read the injected object from the container(IOC) we need to create a container object creation.

Two types of container:

BeanFactory

ApplicationContext

If we want to read the objects from the beanfactory we need to create extra resource object.

for applicationcontext there is no need of creating the resources, it will directly read the object from the container.

Steps to inject the object in XML:

-> we need to inject the object with the help of bean tag.

<bean id="employeeRef" class="com.vtalent.ioc.Employee">

id attribute is nothing but the object reference.

Employee employeeRef = new Employee();

<property name="employeeId" value="100"></property>

name attribute indicates the property name of the class.

this property tag is used to call setter method of the name attribute while creating the object.

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

-> above container object will be created by reading all the bean Id’s in the xml and it will load in IOC container.

-> we need a bean class with setter and getter methods, then we need to inject that in application context. xml with bean tag.

-> if we are going for setter use property tag.

SpringJDBC:

* Spring JDBC is another module in Spring Framework.
* It was designed on top of JDBC.
* whatever the JDBC modules we have like Connection, Statement, PreparedStatement , ResultSet everything we will go for injecting in springJdbc.
* In SpringJdbc connection will be injected through DI (Dependency Injection).
* Statement and PreparedStatement query will be executed with the help of a template called JDBCTemplate.
* JdbcTemplate was designed based on Modules in springJdbc.
* to execute statement queries we need to called jdbcTempalate.update() method, now update method will able to execute all insert, update, delete queries.
* to execute PreparedStatement queries we need to call jdbcTemplate.execute() method, now execute method is able to execute all insert, update, delete queries.
* But the difference between Statement and preparedstatement will be same as we discussed in JDBC.
* Now to execute the select queries we need an resultSet right, so to execute that queries we need to call jdbcTemplate.query() method.
* let’s discuss the modules in spring Jdbc Framework

1. Connection
2. JDBCTemplate
3. Statement
4. PreparedStatementCallBack -> doInPreparedStatement()
5. ResultSetExtractor-> extractData()

**Connection:**

**<bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">**

**<property name="driverClassName" value="com.mysql.jdbc.Driver" />**

**<property name="url" value="jdbc:mysql://localhost:3306/springdb" />**

**<property name="username" value="root" />**

**<property name="password" value="root" />**

**</bean>**

Whatever the objects you’re inserting in to table with the help of prepared statement that object should be an final object.

jdbcTemplate.execute(ref of PreparedStatementCallBack, query);

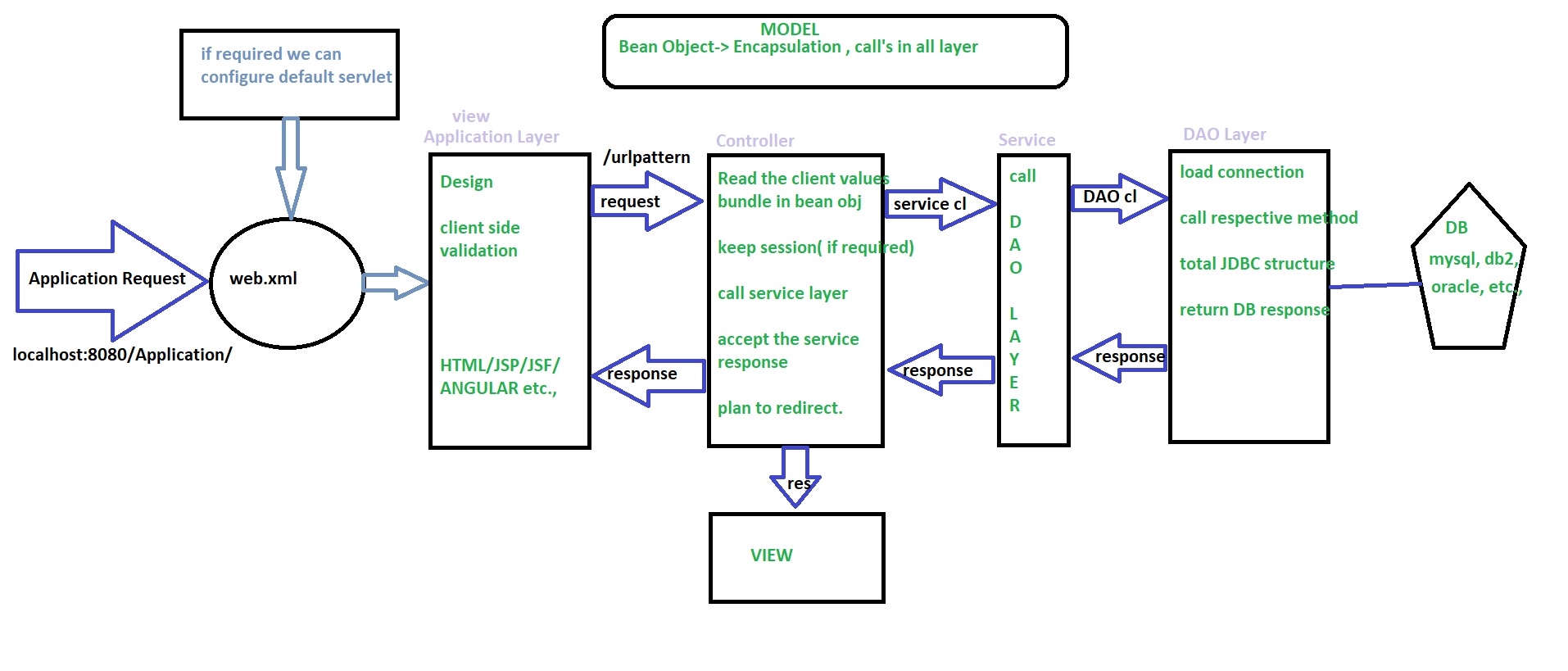
to create an interface object we have an anonymous Inner Class.

doInPreparedStatement method is invoking preparedStatement of JDBC.

to execute the select queries we nned to call jdbcTemplate.query method which is accepting two params called, qurey and ref of ResultSetExtractor

extractData method is invoking ResultSet of JDBC.

**SpringMVC:**

****

**Explanation of Spring MVC Architecture:**

* **Point1:** As I mentioned in the previous J2EE project MVC flow, in any of the we project every request when you hit browser level it will scan through the web.xml which is nothing but the deployment descriptor.
* **Point2:** For that reason we will configure the DispatcherServlet in web.xml for every load.
* When we load the **DispatcherServlet** class at the same time we need to load the config file related to that, the config file name should be servletName-Servlet.xml or default file name will be spring-servlet.xml, where you’ll configure all the spring mvc project related configuration.
* That file should create the default web location in maven project. (WebApp)
* **Point3:** DispatcherServlet (DS) will act as a front controller in spring mvc, where every request and response will transform from here.
* **Point4:** Now the request will pass to HandlerMapping to check the url is in correct format or not then the response will return back to DS.
* **Point5:** From the DS the request will pass to call the request method in a controller class.
* In Spring mvc we are not going to create any servlet except the DS.
* For every request to be accepted on the server side we are going to create a controller class, now the controller class will act as a servlet.
* The only difference in j2ee structure is that every request we need creates a servlet to accept and do the logic.
* But in the controller it will accept a number of requests with help of method level.
* We can make our class as a controller with the help of @Controller annotation which is a stereotype package.
* The method level request we can accept with help of @RequestMapping annotation.
* @RequestMapping is an annotation where we can use it in our class level and method level also.
* From the controller class we can call the service layer then after the response will return back to DS again.
* **Point6:** Service layer structure remains the same to call the DAO layer or any other SI call’s, then most of the implementation we will write in the service layer.
* **Point7:** DAO layer is an data access object to implement DB related logic, either it can be JDBC, SpringJDBC, Hibernate etc.,
* **Point8:** DB can be anything it can be an MySQL, Oracle, DB2 etc.,
* Now the response type will be an ModelAndView object, where we can map the model object nothing but the session object to display in UI and the view page to which page should open on the browser.
* **Point9:** From the DS the modelObject will pass to InternalResourceViewResolver to map the view page to open by adding the prefix as an location of the page and suffix as an extension of that page, then the view object will return to DS again.
* /WEB-INF/viewpages/index.jsp- view object
* **Point10:** Then the view will be displayed on the browser.
* For every request and response the DS is taken care becoz of that we will call it as an FrontController.

**Staring With SpringMVC project:**

Creating the maven Project: where it contains webapp(is the root folder for all UI related things)

creating the groupId and ArtifactId in application level:

groupId: is nothing but package ( com.vtalent.springmvc)

ArtifactId: VtalentSpringMVC

On the WebApp maven structure: It should contains four source folder

src/main/java -> we are creating all java classes

src/main/resource -> we are creating the confg files like .xml

src/test/java -> test classes nothing JUnit

src/test/resource -> test class related confg files like .xml (levl of JUnit)

Note: because of running test classes on your main method we need a seperate config files.

**When the source folders are missing in the Application:**

* Go to the build path, see the below screenshot where your in source tab.
* Remove the missing folder in the source tab.
* Add the new source folder in your java resource.

Add the dependency jar in pom.xml

* Add all spring related IOC, JDBC, MVC dependency jar’s from maven repository.
* POM : Project Object Model
* After adding all the dependencies, you need to update and clean the maven structure.

**SetUp the workspace:**

* Please check in the build path where it contains any unbounded jar are there, please add or modify your local workspace related jars.
* For the maven first we need to update the project.
* Clean the maven project, runAs -> maven clean
* Install the maven project , runas -> maven install
* After the clean and install, maven will download all dependency jars in your local drive, the path will be C:users/.m2/repository/
* Please make sure that the maven dependency is added in your project or application level.

@) in Build path

@) In Deployment Assembly

* Maven install error:

No compiler is provided in this environment. Perhaps you are running on a JRE rather than a JDK?

[INFO] 1 error

The error indicates a change to the JDK environment.

* Because of the above point if you miss you’ll get the class not found exception.

**Setp by Step procedure for spring mvc application configuration:**

* Created the Home.jsp page, to open this we need to go for Spring Artech level.
* We need to open the Home.jsp page on the initial of project running.
* Initial URL:<http://localhost:8080/VtalentSpringMVC/>

Localhost: local IP machine 127.0.0.1

8080: Tomcat port number

VtalentSpringMVC : context name or application name

/: config name or request name

/ is also called as default request

* Configure the spring mvc application.
* Configuring the DispatcherServlet in web.xml and dispatcher cfg file also we need to load in web.xml
* Make configuration of dispatcher with controller and view resolver and all class related injections, UI related injections, Hibernate or springJDBC connection related injections etc.,
* In configuration first we need to import all the schema related to the frameworks.
* We need to inject a controller and to scan the controller with the help of **contextComponentScan** tag.
* Let me create a controller package and create the controller class to accept the request from the DispatcherServlet.
* @Controller is used to make our class as controllers.
* @RequestMapping is used to handle the request or config value from the URL and it should be used at the method level.
* Please make sure that all the @RequestMapping values are handling the unique request.
* We need to configure the ViewResolver in spring config file to display the view page on the browser.
* Let me create a registration page where I inject on Hom.jsp to open my registration form.
* <http://localhost:8080/VtalentSpringMVC/openingAnEmployeeReg>
* **openingAnEmployeeReg** is an config name or request name for that form
* To Read the form values and those values to be inserted into the DB, we will configure Hibernate.

**Hibernate:**

Hibernate is the framework and it was built on top of JDBC and SpringJDBC.

There will be two cache levels on hibernate:

1) SessionFactory

2) Session

**SessionFactory:**

* This is an application level sessionFactory object.
* We are creating a DB connection with DriverManagerDataSource object.
* We are injecting Entity classes.
* Entity class: it is nothing but an encapsulation class where we can make class as an Entity with the help of @Entity annotation.
* Hibernate queries are not dependent on your DB (MySql).
* Hibernate will depend on Entity classes.
* Configuration of Entity classes.
* Entity class we will map to DB table with the help of @Table annotation.
* What are all the fields we have in entity classes? All we are mapping to table columns with help of @column annotation, this one mainly used when the class properties and table fields are different.
* These queries will return as HQL queries, Hibernate Query Language.
* Now all the entity classes will be injected in SessionFactory.

**Session:**

* Session objects will be opened from sessionfactory and this reference will be used in module level.
* Number of session objects can be created with the help of session factory.
* We will close the session once the module execution is done.

**Note:**  SessionFactory object will destroy once application is down or stopped.

**Hibernate Core steps:**

1) injecting the DB in your applicationContext.xml

2) to load the configuration file we need to use **Configuration** class.

3) With reference to configuration we need to create the sessionfactory object.

4) With the reference of sessionfactory we are creating the session object.

Ex: Session session = sessionFactory.openSession.

5) With the help of the session we will begin the transaction.

@AutoWired: this annotation is used to get a reference from your container and it will inject into the class whatever you mapped.

Ex: @AutoWired Employee emp;

**Spring form tags:**

Contains an additional attribute to read the form values to server level.

**modelAttribute**: it is an attribute in your spring form where we will give the ref, so that all the form values will scan the inputs to inject into my modelAttribute object. It will be done by internally calling the setter methods of that bean class.

How the setter methods will be called: if your employee class property and form level input name attribute should be the same.

è To read the modelAttribute object from your form level, we need to use @ModelAttribute annotation in your method level.

ModelAndView is the object where I can combine the view page and the object value to pass to that page.

HQL queries:

HQL: From entityclassName: it will pull all the data from that class which is mapped to the table.

Ex: from employeeBean; (select \* from employeebean)

HQL: From EmployeeBean emp where emp.employeeId:employeeId;

(select \* from table where employeeid=?)

(select employeename, employeedob from table where employeeid=?)

HQL: select emp.employeeName, emp.employeeDOB from EmployeeBean emp where emp.employeeId:employeeId;

**SpringMVC Annotation based Configuration:**

Git: <https://github.com/narsinghraom/USA_Stores.git>

Branch: develop

* WebApplicationInitializer will act as web.xml
* WebMvcConfigurerAdapter will act as spring-servlet.xml config fileS

|  |  |  |
| --- | --- | --- |
| XML Tag | Annotation | Description |
| <context:component-scan/> | @ComponentScan() | Scan starts from base package and registers all controllers, repositories, service, beans, etc. |
| <mvc:annotation-driven/> | @EnableWebMvc | Enable Spring MVC-specific annotations like @Controller |
| Spring config file | @Configuration | Treat as the configuration file for Spring MVC-enabled applications. |

if we want to inject DB related classes, make those classes as @Repository and service implemented class as @Service. so that these objects can be available in container

They are very different in that Services don't typically know how to access data from persistence, and repositories typically *only* access data/objects for any services you may have.

@**Component** is a class level annotation whereas @**Bean** is a method level annotation and name of the method serves as the **bean** name. @**Component** need not to be used with the @Configuration annotation whereas @**Bean** annotation has to be used within the class which is annotated with @Configuration.

@Component: annotation is used to make your class as a bean injected class and it will available in container too.

We can read form input values in three ways:

1. **ModelAttribute**: if you want to read all the form input values we can use @ModelAttribute to scan all the values by calling setter method of that property it will create an object.

Ex: @ModelAttribute(“nameOfTheAttribute”) Object object;

1. **RequestParam**: if you want to read the limited form values, we can use @RequestParam(inputfieldName) String fieldName
2. **PathVaribale**: if you want to read the value from the URL, we can use this annotation @PathVariable(“name”) String pathValue

**Log4J:**

* log4j is highly configurable through external configuration files at runtime.
* Logging is an important component of the software development
* Easily debug the application.
* log4j has three main components:
* **loggers**: Responsible for capturing logging information.
* **appenders**: Responsible for publishing logging information to various preferred destinations.
* **layouts**: Responsible for formatting logging information in different styles.

## Pattern Conversion Characters

The following table explains the characters used in the above pattern and all other characters that you can use in your custom pattern:

|  |  |
| --- | --- |
| **Conversion Character** | **Meaning** |
| c | Used to output the category of the logging event. For example, for the category name "a.b.c" the pattern %c{2} will output "b.c". |
| C | Used to output the fully qualified class name of the caller issuing the logging request. For example, for the class name "org.apache.xyz.SomeClass", the pattern %C{1} will output "SomeClass". |
| d | Used to output the date of the logging event. For example, %d{HH:mm:ss,SSS} or %d{dd MMM yyyy HH:mm:ss,SSS}. |
| F | Used to output the file name where the logging request was issued. |
| l | Used to output location information of the caller which generated the logging event. |
| L | Used to output the line number from where the logging request was issued. |
| m | Used to output the application supplied message associated with the logging event. |
| M | Used to output the method name where the logging request was issued. |
| n | Outputs the platform dependent line separator character or characters. |
| p | Used to output the priority of the logging event. |
| r | Used to output the number of milliseconds elapsed from the construction of the layout until the creation of the logging event. |
| t | Used to output the name of the thread that generated the logging event. |
| x | Used to output the NDC (nested diagnostic context) associated with the thread that generated the logging event. |
| X | The X conversion character is followed by the key for the MDC. For example, X{clientIP} will print the information stored in the MDC against the key clientIP. |
| % | The literal percent sign. %% will print a % sign. |

<param name="File" value="E://usastores.log" />

> it will create the file in the location which you mention in value attribute.

<param name="MaxBackupIndex" value="5" />

<param name="MaxFileSize" value="5000000" />

> the file size will go to maximum 5 MB.

> once the file reaches to 5 mb, it will backup to usastores.1.log and it will clear the usastores.log

> the above step will repeat till 5 backups, becoz the maxBackUpIndex was mentioned as 5.

> if it was reaches the size of file 6th time, it will clear usastore.1.log and it will copy the newly usastore.log

info()-> just to print the information

debug()-> it will print when the application is in debug mode

error()-> is used to print the exceptions in catch block

trace()-> enable or disable the trace, if enable only the loggers will generate.

**JQuery With Ajax:**

GitLink: <https://github.com/narsinghraom/InsuranceBySS.git>

Branch: develop

* First of all we will see how we can access the resources files in spring MVC architecture .

<bean id="multipartResolver"

class="org.springframework.web.multipart.commons.CommonsMultipartResolver" />

* The above class needs to inject in spring configuration file, where the commons Multipart Resolver is used to read all the resources type files from the project location.
* The below mvc:resource tag is used to map the mapping value to be used in code to access the location which we configured.
* we need to configure for all the resource like images, javascript, css and jquery ext.,
* The mapping value can be any unique name we can mention to access in the UI end.
* The location value should be the exact path where that file was located and their subfolder needs to mention.

<mvc:resources mapping="/images/\*\*" location="/WEB-INF/images/" />

<mvc:resources mapping="/js/\*\*" location="/WEB-INF/js/" />

<mvc:resources mapping="/styles/\*\*" location="/WEB-INF/styles/" />

<mvc:resources mapping="/javascript/\*\*" location="/resources/javascript/" />

If you see this type of URL: //InsuranceBySS/HomeAjax/getZips

InsuranceByss : Application Name

HomeAjax: class level request mapping

getZips: method level request mapping

**SpringBoot Annotation:**

@RequestMapping(value = “/”, RequestMethod.GET)

@RequestMapping(value = “/”, RequestMethod.POST)

> the above two changes we can use in spring boot with different annotation

GetMapping(“”)

PostMapping(“”)

@Controller -> @RestController