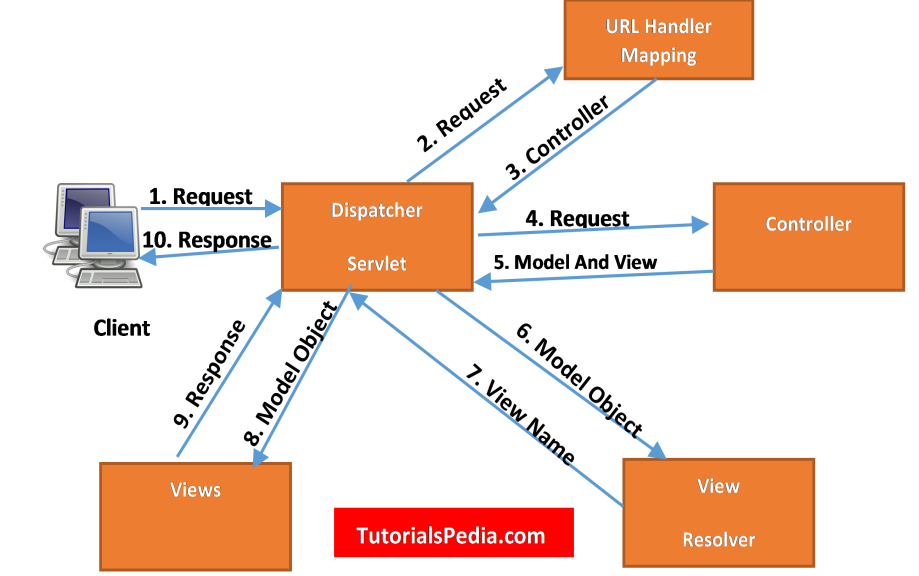
**Spring MVC(Model-View-Controller Design pattern)**

The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

* **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
* **Controller -** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.
* **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page.

**DispatcherServlet**/**Front Controller:**The Spring Web model-view-controller (MVC) framework is designed around a *DispatcherServlet* that handles all the HTTP requests and responses.

****

**1. Request**

The first step in the MVC flow is when a request is received by the Dispatcher Servlet.

**2. Dispatcher Servlet**

Now, the Dispatcher Servlet will with the help of Handler Mapping understand the Controller class name associated with the received request. Once the Dispatcher Servlet knows which Controller will be able to handle the request, it will transfer the request to it.

**3. Controller**

The Controller will process the request based on appropriate methods and will return it to Model Data and View Name.

**4. Model and View**

It will return the processed data to the Dispatcher Servlet.

**5. View Resolver**

Once Model and View receive the data, Dispatcher Servlet will transfer it to the View Resolver to get the actual view page.

**6. View**

Finally, the Dispatcher Servlet will pass the Model object (results) to the view page. This is the final step of the flow where the results will be displayed.

**Create Spring MVC project**

**Step1)File--New--maven project--next--select web archType 1.1--write groupId,artifactId,name etc finish.**

**Step 2)check JRE system library change—build—configure build path—libraries—remove previous JRE—add Library—JRE System library—bydefault selected—finish—apply and close**

**Step3)Update maven project forcefully**

**Step4)Attach tomcat webserver**

**Step4.1)check wheather tomcat sever path set for entire workspace if not then set it-🡪window—preferences—search server—Runtime environment—Add—search apache—select apache tomcat v9.0—next—copy the path till bin where you downloaded it**

**(D:\Future\_Perspective\Spring\Server\apache-tomcat-9.0.67) paste it in Tomcat installation library---finish**

**Step4.2)step4.1 is optional if already done don’t need to follow again then attach this tomcat to specific project—rightclick on project—proporties—Targeted runtime—select apache tomcat—apply and close**

**First time while deploying project on server may be you wont see tomcat server so deploy it manually—select manually define new server—search apache—select tomcat apache v9.0 –next—finish**

**Configuring spring mvc project**

**1.Configure dispatcher servlet/Front controller in web.xml**

<!-- step 1 Configure Dispatcher servlet/ front controller -->

<servlet>

<servlet-name>spring</servlet-name>

<servlet- class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/</url-pattern><!-- will handle all type of url -->

</servlet-mapping>

**2.Create spring-servlet.xml file where we declare beans**

Create a xml file in web-inf folder name is imp syntax :ServletName-servlet.xml

Here spring is servlet name so spring-servlet.xml and copy all required content of spring-config.xml paste it in spring-servlet.cml

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:util=*"http://www.springframework.org/schema/util"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.0.xsd*

*http://www.springframework.org/schema/util*

*http://www.springframework.org/schema/util/spring-util-3.0.xsd"*>

**3.configure view resolver in spring-config.xml**

<!-- Step 3 configure view resolver in spring-servlet.xml -->

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"* name=*"viewResolver"*>

<!-- All response page in the form view present inthis folder to handle request -->

<property name=*"prefix"* value=*"/WEB-INF/views/"*></property>

<!-- Response page ending with .jsp extention ex-view name= page\_name.jsp -->

<!-- Fully Qualified name=/WEB-INF/views/hello.jsp -->

<property name=*"suffix"* value=*".jsp"*></property>

</bean>

**4.Create controller**

Create simple java class mark class as @Controller annotation

**5.create a view to show**

Create a vies in views folder in WEB-INF to handle request

**Sending data from controller to view**

**1.Model**

addAttribute(String key,Object value);

**2.ModelandView**

addObject(String key,Object value);

**Get data in view---scriptlet tag**

Object ob=(Object)request.getAttribute(“key”);

Controller

HttpServletRequest

HttpServletRequest object use to get data send from view page to controller or viseversa

View

Html form

Data send through html form

**Spring MVC-ORM flow**

Dao

DataAccess

Layer

Database

Layer

Service

Layer

Controller

Layer

View Layer/present

**Some Important Annotations :**

**@Controller :**This annotation used to mark particular class as controller class which handle or mapping request and method declared inside class handle particular type request return view pages

**@RequestMapping:**This is class or method level annotation generally used to mark or identify request

Syntax : @RequestMapping(“/home”)---------------home request through URL

If it is used with class level then it means whatever request handler method inside class handle specific request belongs from specific task

**@GetMapping**: It maps the HTTP GET requests on the specific handler method. It is used to create a web service endpoint that fetches It is used instead of using: @RequestMapping(method = RequestMethod.GET)

**@PostMapping:** It maps the HTTP POST requests on the specific handler method. It is used to create a web service endpoint that creates It is used instead of using: @RequestMapping(method = RequestMethod.POST)

**@PutMapping:** It maps the HTTP PUT requests on the specific handler method. It is used to create a web service endpoint that creates or updates It is used instead of using: @RequestMapping(method = RequestMethod.PUT)

**@DeleteMapping:** It maps the HTTP DELETE requests on the specific handler method. It is used to create a web service endpoint that deletes a resource. It is used instead of using: @RequestMapping(method = RequestMethod.DELETE)

Note :@RequestMapping is more generic type while all above are more specific type annotation

**@RequestParam** :This annotation used to extract pararmeter from URL

**@ModelAttribute :**This annotation is used to binding data from request param in key-value pairs,it used to inject object in to model object i.e it binds method parameter or method return value to named model attribute

**@Repository :**This is class level annotation usually marks that class is DAO(Data Access Object) class which directly access database This class does all database related operation

**@Service :**This is also class level annotation mark that class contain all business logic and

**@Transactional** :It is class or method level annotation mark that class or method perform transaction from java application to database

If it is declare with class then all method inside class are transaction class or it can be method specific

**@PathVariable :** Mostly used when you build REST API,Used to bind method paratmeter to URI template variable #http://localhost:8080/project/home/2----🡪id

@RequestMapping(“/home/{id}”)

Public String Handler(@PathVariable(“id”) int id) {

**Redirect in Spring MVC :Transfer http request to another page ,it simply redirect your request url to another url either because of success or fail**

**1.Redirect Prefix**

return “redirect:/new\_request\_name”;

return “redirect: <http://www.google.com>”;

**2.Redirect View**

@RequestMapping(“/home”)

Public RedirectView method(){

RedirectView obj=new RedirectView();

Obj.setUrl(“<http://www.google.com>”);

return obj;

}

**Exception Handling In Spring MVC**

**@ExceptionHandler(Exception.class) :This annotation is used to detect certain runtime-exception and send response according to that exception so user can get error and validate it,it handle the specific exception and sending the custome response to client**