# Spring MVC

1. A spring MVC is sub framework of spring framework which is used to build a **web application.**
2. It is built on the top of **Servlet API**.
3. If follows **Model-View-Controller** design pattern.
4. It implementation all the basic feature a core spring framework like **IOC, Dependency Injection**.

**Why Spring MVC**

* Separate role model, view , controller etc.
* Power configuration (XML, Java Based, Annotation).
* It is sub framework of spring framework. Use of spring core features like IOC, DI, Loosely coupling.
* One time you configure the project you can development rapidly.
* Spring MVC is flexible, easy to test and much features.

**MVC Design Pattern:-**

* The **Model** encapsulates the application data and in general they will consist of POJO.
* The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
* The **Controller** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.

User

**1.HttpsReq 5**

**HttpsRes.**

View resolver

Dispatcher Servlet

**4**

**2.Hander mapping 3.View model object**

Handler

* After receiving an HTTP request, *DispatcherServlet* consults the *HandlerMapping* to call the appropriate *Controller*.
* The *Controller* takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the *DispatcherServlet*.
* The *DispatcherServlet* will take help from *ViewResolver* to pickup the defined view for the request.
* Once view is finalized, The *DispatcherServlet* passes the model data to the view which is finally rendered on the browser.

**Spring MVC Application Creation Steps:-**

1. Configure Dispatcher Servlet
2. Create Spring Configuration File
3. Configure View Resolver
4. Create Controller
5. Create a View to show(Page)

**Dynamic Returning value in Jsp Page by through controller:-**

**@Controller: -** It is indicates that a particular class serves the role of a controller.

**@RequestMapping: -** It is used to map a URL to either an entire class or a particular handler method.

1. **By the Help of Model**

**addAttributes(String Key ,Object Value)**

1. **By the help of ModelAndView**

**addObject((String Key ,Object Value)**

**Example:-**

@RequestMapping("/home")

public String home(Model model) {

System.*out*.println("This is home page");

model.addAttribute("name","Mirza Faisal ");

model.addAttribute("name1","Mirza Faisal Ibrar Baig ");

List<String> friends=new ArrayList<>();

friends.add("Satya");

friends.add("Soubhagya");

friends.add("Mukesh");

model.addAttribute("f",friends);

return "index";

}

@RequestMapping("/about")

public String about() {

System.*out*.println("This is about pages");

return "about";

}

@RequestMapping("/help")

public ModelAndView help() {

System.*out*.println("This is about help");

ModelAndView modelAndView=new ModelAndView();

modelAndView.addObject("name","Mirza Faisla Ibara Baig");

modelAndView.setViewName("help");

return modelAndView;

}

**Printing Values by the help of JSTL Expression:-**

<h1>

Name is :

<%-- <%=name%> --%>

${name }

</h1>

<h1>

Name is :

<%-- <%=name1%> --%>

${name1 }

</h1>

<c:forEach var=*"friends"* items=*"*${f}*"*>

<h1>${friends}</h1>

</c:forEach>

1)Redirect Prefix

public String handler()

{

return "redirect:/url"

}

2)RedirectView Class

public RedirecView handler()

{

RedirectView redirectView=new RedirectViw();

redirectView.setUrl("two");

return redirectView;

}