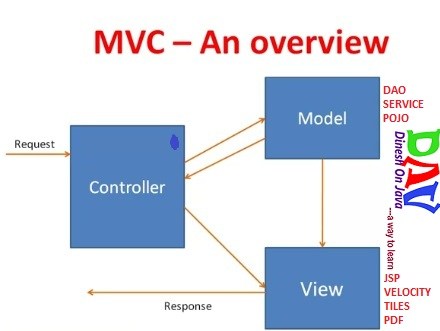
**Spring MVC (Model view controller)** is based on the MVC design pattern, it is a software architecture design pattern.  It provides solution to layer an application by separating three concerns business, presentation and control flow

* The **Model** can be some DAO layer or some Service Layers which give some information about request or requested information or Model can be a POJO which encapsulates the application data given by the controller.
* 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 appropriate model and passes it to the view for rendering.



**Advantages of Spring MVC Framework-**

* Supports [RESTful URLs](http://en.wikipedia.org/wiki/Representational_state_transfer).
* Annotation based configuration(i.e. you may reduce the metadata file or less of configuration).
* Supports to plug with other MVC frameworks like Struts, Struts2, WebWorks etc.
* Flexible in supporting different view types like JSP, velocity, XML, PDF, Tiles etc.

December 21, 2012

[**17 COMMENTS**](https://www.dineshonjava.com/spring-web-mvc-framework-chapter-38/#comments)

# Spring MVC Framework Tutorial with Example

## Spring MVC

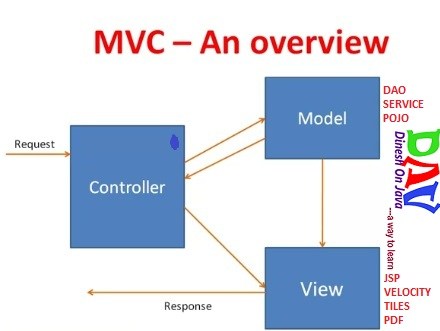
**Spring MVC (Model view controller)** is based on the MVC design pattern, it is a software architecture design pattern.  It provides solution to layer an application by separating three concerns business, presentation and control flow.

## Spring 5 Design Pattern Book

You could purchase my **Spring 5 book** that is with title name “**Spring 5 Design Pattern**“. This book is available on the [**Amazon**](https://www.amazon.in/Spring-Design-Patterns-Dinesh-Rajput/dp/1788299450/ref=sr_1_1?ie=UTF8&qid=1507925340&sr=8-1&keywords=spring+5+design+pattern) and **[Packt](https://www.packtpub.com/application-development/spring-5-design-patterns" \t "_blank)** publisher website. Learn various **design patterns** and **best practices** in Spring 5 and use them to solve common design problems. You could use author discount to purchase this book by using code- “**AUTHDIS40**“.

[](https://www.packtpub.com/application-development/spring-5-design-patterns)

* The **Model** can be some DAO layer or some Service Layers which give some information about request or requested information or Model can be a POJO which encapsulates the application data given by the controller.
* 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 appropriate model and passes it to the view for rendering.



**Popular Tutorials**

* [***Spring Tutorial***](https://www.dineshonjava.com/spring-tutorial/)
* [**Spring MVC Web Tutorial**](https://www.dineshonjava.com/spring-web-mvc-framework-chapter-38/)
* [**Spring Boot Tutorial <**](https://www.dineshonjava.com/introduction-to-spring-boot-a-spring-boot-complete-guide/)
* [**Spring Security Tutorial**](https://www.dineshonjava.com/spring-security-take-baby-step-to-secure/)
* [**Spring AOP Tutorial**](https://www.dineshonjava.com/spring-aop-tutorial-with-example-aspect-advice-pointcut-joinpoint/)
* [**Spring JDBC Tutorial**](https://www.dineshonjava.com/using-spring-jdbc-framework-chapter-32/)
* [***Spring HATEOAS***](https://www.dineshonjava.com/spring-hateoas-hypermedia-driven-restful-web-service/)
* [***Microservices with Spring Boot***](https://www.dineshonjava.com/microservices-with-spring-boot/)
* [**REST Webservice**](https://www.dineshonjava.com/jax-rs-web-service-tutorial/)
* [***Core Java***](https://www.dineshonjava.com/core-java-baby-step-to-be-best-java-ian/)
* [***Hibernate Tutorial***](https://www.dineshonjava.com/hibernate-3-on-baby-steps/)
* [**Spring Batch**](https://www.dineshonjava.com/spring-batch-process-with-example/)

## Advantages of Spring MVC Framework-

* Supports [RESTful URLs](http://en.wikipedia.org/wiki/Representational_state_transfer).
* Annotation based configuration(i.e. you may reduce the metadata file or less of configuration).
* Supports to plug with other MVC frameworks like Struts, Struts2, WebWorks etc.
* Flexible in supporting different view types like JSP, velocity, XML, PDF, Tiles etc.

[](https://certification-questions.com/buy-mock-exams.html?affiliateCode=4f5a35d8-0022-4c9c-b8d3-265ed0b825a2&utm_source=Dineshh&utm_medium=affiliate&utm_campaign=affiliate)

## Front Controller

## The front controllera [controller](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) that handles all requests for a [website](https://en.wikipedia.org/wiki/Website)",[[1]](https://en.wikipedia.org/wiki/Front_controller" \l "cite_note-:0-1) which is a useful structure for web application developers to achieve the flexibility and reuse without code redundancy.

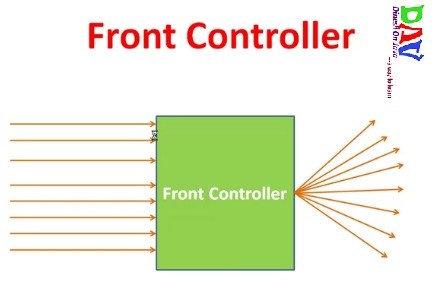
 Is a initial level of contract point for handling a request. The front controller provides a centralized entry point for that controls and manages web request handling. By centralizing decision point and controls.  
  
The front controller also reduce your java code and business logic by promoting code reuse ability across the requests. The front controller coordinates with dispatcher components. Dispatcher are responsible to view management only, the one who bring the view components to the user is called dispatcher.  
There are lot of things about the front controller design pattern but we are not going here into the deep, we will see about front controller design pattern in detail in another post.

Ad by Google

In Spring MVC **org.springframework.web.servlet.DispatcherServlet** is a front controller who handles all the user request and process the request as per there mapping.  
  
**For Example suppose you are going for an interview with reputed company once you entered inside the premisses, you will not directly entered in interview panel, someone who will process your CV and bring you to the front of interview panel. Each and every candidate first reached on common place and few HR person were there who took your CV and bring you to the front of interviewer. Here HR persons are the front controller who process each and every request and based your experience level they will send to the interviewer for the same.**

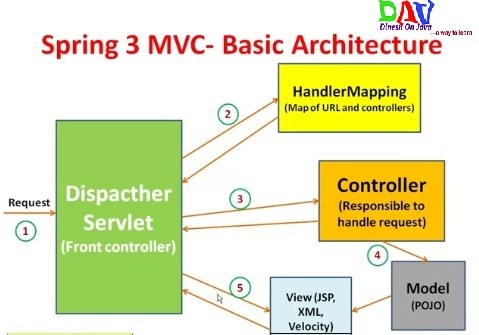
Front Controller is very important component one which route the all the requests into framework control that means when ever requests land on different controllers it queues that request to the controller of framework without this MVC framework will not may be able to take control of the request at landing at the application. So front controller is not only capture the request but also the following responsibility-

* It initialize the framework to cater to the requests.
* Load the map of all URLs and the components responsible to handle the request.
* Prepare the map for the views.



## Spring MVC Basic Architecture

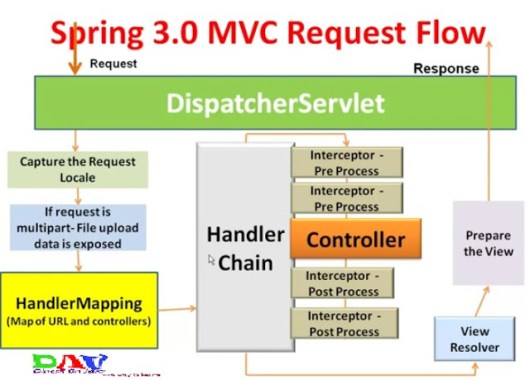
The **Spring web MVC framework** provides **model-view-controller** architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic),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



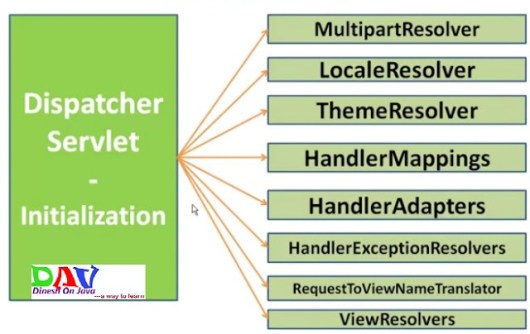
In Spring MVC framework **Dispatcher Servlet** access Front Controller which handles all coming requests and queses for forward to the different controller.  
**1.** Whenever request lands the dispatcher servlet consult with HandlerMapping  
(**HandlerMapping**– is a component which have the map of URL and Controller which need to be invoked for that particular request which lands with URL)  
**2.** then Dispatcher servlet has information about which is controller need to be invoked  
**3.** then that controller will be invoked  
**4.** and Controller can request the model for some information (about some DAO, Service layer or Data in POJO, or data in database using business logic)  
**5.** once process has been done then dispatcher servlet get the response then dispatcher servlet will get view resolver to build the view and view resolver look out what view has being configured it has been JSP, Velocity, XML etc. based this configuratin view has been prepared and the information from model i.e. POJO it will be put on the view and response will be send back to browser.

## Spring  MVC Request Flow

**1.** Request lands to Front Controller i.e. **DispatcherServlet**  
**2.** Capture the Request **Locale**i.e use for internationalization i.e Read **.properties** files  
**3.** Check for multipart-file(**MIME type header** or not) upload data from distributed application  
**4.** Consult with **HandlerMapping**for which Controller to be invoked  
**5.** and Then responsibility is given to the **Handler Chain**  
**6.** This **Handler Chain** is responsible to be invoked some of the **interceptors**that needs to be invoked before of a controller and after the controller that means interceptors are here like very to similar to the filters that help to separate the **pre-process logic** and **post-process logic**.  
**7.** After process of pre-process interceptor return to the controller process the post-process logic.  
**8.** Then return to the view resolver prepared the view based on your configuration decide the which configuration (JSP, Velocity, PDF etc.) to be invoked.  
**9.** After choosing view technology prepare the view and return the response back to the client.



## Spring MVC Framework- Initialization



**MultipartResolver:**Interface to handle the file uploads  
**LocaleResolver:**Helps to resolve the locale from the request  
**ThemeResolver:**Resolve a theme for a request(CSS)  
**HandlerMapping:**Maps the Request to Handlers (Controllers)  
**HandlerAdapter:**Plugs the other frameworks handlers  
**HandlerExceptionResolver:**Mapping of the exceptions to handlers and views  
**ViewResolver:**Maps the view names to view instances

All the above mentioned components ie. **HandlerMapping**, **Controller**and **ViewResolver**are parts of ***WebApplicationContext***which is an extension of the plain ***ApplicationContext***with

## Controllers are used to provide access to the application behavior that is defined through a service interface. Controllers are the ones that interpret user input and transform it into a model that is represented to the user by the view.

## The role of a controller in Spring MVC

In Spring MVC, controller methods are the final destination point that a web request can reach. After being invoked, the controller method starts to process the web request by interacting with the service layer to complete the work that needs to be done. Usually, the service layer executes some business operations on domain objects and calls the persistence layer to update the domain objects. After the processing has been completed by the service layer object, the controller is responsible for updating and building up the model object and chooses a view for the user to see next as a response.

Remember that Spring MVC always keeps the controllers unaware of any view technology used. That's why the controller returns only a logical view name; later, DispatcherServlet consults with ViewResolver to find out the exact view to be rendered. According to the controller, Model is a collection of arbitrary objects and View is specified with a logical name.

In all our previous exercises, the controllers used to return the logical view name and update the model via the model parameter available in the...

## Handler mapping

We have learned that DispatcherServlet is the one that dispatches the request to the handler methods based on the request mapping; however, in order to interpret the mappings defined in the request mapping, DispatcherServlet needs a HandlerMapping implementation (org.springframework.web.servlet.HandlerMapping). The DispatcherServlet consults with one or more HandlerMapping implementations to find out which controller (handler) can handle the request. So, HandlerMapping determines which controller to call.

The HandlerMapping interface provides the abstraction for mapping requests to handlers. The HandlerMappingimplementations are capable of inspecting the request and coming up with an appropriate controller. Spring MVC provides many HandlerMapping implementations, and the one we are using to detect and interpret mappings from the @RequestMapping annotation is the RequestMappingHandlerMapping class (org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping). To start using RequestMappingHandlerMapping, we have to add the <mvc:annotation-driven> element in our web application context...

No, they are pretty different from each other.

Both are different specializations of **@Component** annotation (in practice, they're two different implementations of the same interface) so both can be discovered by the classpath scanning (if you declare it in your XML configuration)

**@Service** annotation is used in your service layer and annotates classes that perform service tasks, often you don't use it but in many case you use this annotation to represent a best practice. For example, you could directly call a DAO class to persist an object to your database but this is horrible. It is pretty good to call a service class that calls a DAO. This is a good thing to perform the separation of concerns pattern.

**@Controller** annotation is an annotation used in Spring MVC framework (the component of Spring Framework used to implement Web Application). The @Controller annotation indicates that a particular class serves the role of a controller. The @Controller annotation acts as a stereotype for the annotated class, indicating its role. The dispatcher scans such annotated classes for mapped methods and detects @RequestMapping annotations.

So looking at the Spring MVC architecture you have a DispatcherServlet class (that you declare in your XML configuration) that represent a front controller that dispatch all the HTTP Request towards the appropriate controller classes (annotated by @Controller). This class perform the business logic (and can call the services) by its method. These classes (or its methods) are typically annotated also with **@RequestMapping** annotation that specify what HTTP Request is handled by the controller and by its method.