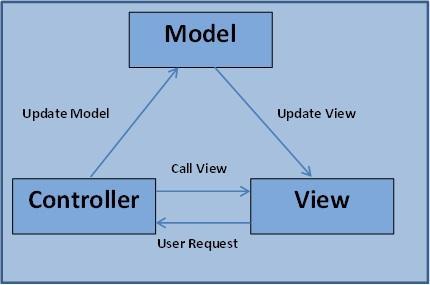
**SPRING MVC**

**Q: Question: Explain Spring MVC framework with an example ?**

**Ans:**

The MVC is a standard software architecture that aims to separate business logic from presentation logic, enabling the development, testing and maintenance of both isolated .



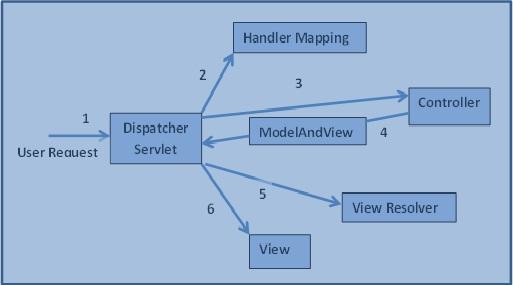
(1) The user triggers an event through the UI (click a button on the page or something).

(2) The controller receives the event and coordinates how things will happen on the server side, i.e. the flow goes to the objects required to perform the business rule.

(3) The model is used to represent the business information so it is called by the controller to perform the business rule that should be done. The model can also be used to send data to the view layer.

(4) After the controller performs business rule using the model it renders a new view and sends it to the user. Create an application using the MVC pattern is nothing more than to separate the logic in “packages” and make that information follow a certain path.

**Information flow diagram of Spring Web MVC :**



(1) A request from the user always arrives in a single servlet Dispatcher Servlet. This is a very common pattern in MVC frameworks called front controller (front controller) where a single servlet is responsible for receiving the request, delegate processing to other components of the application and return a response to the user.

(2) Once the Dispatcher servlet gets the request it must find out which controller the request will be sent. So it asks for help in the Handler Mapping. Based on the request URL Handler Mapping finds the controller.

(3) The request is then sent to the controller that will take care of the data contained therein. Once the controller receives the request it processes the data and performs some business rules of the application.

(4) Often rendered by the logic controller results in some information that should be taken back to the user (the model). Only sending information back is not enough, it must be formatted in a way that the user can understand, for that controller send it to the view. For this model and the view are encapsulated in a ModelAndView object and returned to the Dispatcher.

(5) As the controller is not stuck to just one view it sends a hint in the ModelAndView object to the Dispatcher to know which view should send the data. The Dispatcher passes this hint to viewResolver that returns the view which should be called.

(6) Finally, the Dispatcher sends the information (model) to the view (JSP page) who has just discovered. The page renders the information received and is returned to the user.

**Q. In Spring's Web MVC Architecture, is it required to have the Model class uses framework specific class or interface?**

**Ans.** No, it is not required to have the model implementation depend on Spring's Framework specific class or interface. Spring's Model can be a POJO with no dependency on Spring's Framework class files.

**Q. In Spring Web MVC, can the Controller implementation be able to write directly to the output/response stream?**

**Ans.** Yes, in Spring Web MVC, Controller can write content to the output/response stream.

**Q. What are the advantages of Spring MVC over Struts MVC ?**

**Ans:**

1. There is clear separation between models, views and controllers in Spring.

2. Spring’s MVC is very versatile and flexible based on interfaces but Struts forces Actions and Form object into concrete inheritance.

3. Spring provides both interceptors and controllers, thus helps to factor out common behavior to the handling of many requests.

4. Spring can be configured with different view technologies like Freemarker, JSP, Tiles, Velocity, XLST etc. and also you can create your own custom view mechanism by implementing Spring View interface.

5. In Spring MVC Controllers can be configured using DI (IOC) that makes its testing and integration easy.

6. Web tier of Spring MVC is easy to test than Struts web tier, because of the avoidance of forced concrete inheritance and explicit dependence of controllers on the dispatcher servlet.

7. Spring web layer built on top of a business object layer which is considered a good practice. In Struts framework you need to implement your business objects .

8. Struts forces your Controllers to extend a Struts class but Spring doesn’t, there are many convenience Controller implementations that you can choose to extend.

9. In struts a variety of Struts specific tags are used to assure that request parameters are bound to ActionForm fields and show binding/validation errors. In SpringMVC, there is one simple bind tag that handles all of this. Your JSP’s remain smaller and have more pure HTML content.

10. In Struts, Actions are coupled to the view by defining ActionForwards within a ActionMapping or globally. SpringMVC has HandlerMapping interface to support this functionality.

11. With Struts, validation is usually performed (implemented) in the validate method of an ActionForm. In SpringMVC, validators are business objects that are NOT dependent on the Servlet API which makes these validators to be reused in your business logic before persisting a domain object to a database.

**Q. What is role of Controller in Spring MVC framework?**

**Ans:** In a general way, Controller (‘c’ in mvc ) delivers access to the behavior of application which is usually defined by a service interface and acts as glue between core application and the web. It processes/interprets client data and transforms it to a domain object called model, which is finally represented to the client by the view. Spring framework provides wide range of controllers.

In a specific way, Spring controllers are based on “org.springframework.web.servlet.mvc.Controller” Interface. Controller interface provides a single method that handles a request and returns a particular model and view as given in code snippet below.

**public interface Controller {**

**ModelAndView handleRequest( HttpServletRequest request,**

**HttpServletResponse response) throws Exception;**

**}**

**Q. What is a front controller in Spring MVC ?**

**Ans:**

A front controller is defined as “a controller which handles all requests for a Web Application.” DispatcherServlet (actually a servlet) is the front controller in Spring MVC that intercepts every request and then dispatches/forwards requests to an appropriate controller. The DispatcherServlet is declared in the web.xml file of web application. Requests which we want to be handled by DispatcherServlet should be mapped using URL mapping. For example all requests ending with \*.htm will be handled by the DispatcherServlet.

*<web-app>*

*<servlet>*

*<servlet-name>example</servlet-name>*

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

*<load-on-startup>1</load-on-startup>*

*</servlet>*

*<servlet-mapping>*

*<servlet-name>example</servlet-name>*

*<url-pattern>\*.htm</url-pattern>*

*</servlet-mapping>*

*</web-app>*

**Q. SimpleFormController example Spring MVC?**

**Ans:**

Use of SimpleFormController Example :

It offers you form submission support. This can help in modeling forms and populating them with model/command object returned by the controller. After filling the form, it binds the fields, validates the model/command object, and passes the object back to the controller so that the controller can take appropriate action. It supports features like invalid form submission, validating form data, and normal form workflow.

SimpleFormController let’s you specify a model/command object, a viewname for your form, a viewname for the page you want to display the user when form submission is successful.

In this tutorial user is first showed the welcome page (index.html) where on clicking the given link it takes the user to Application form (fillForm.jsp). User fills in the details, submits the form and finally he is shown another page containing the details that he filled previously.

Things you will learn in this tutorials :

* Spring MVC basic flow (data flow from view to controller and rendering model data back to view)
* Use of SimpleFormController
* Use of SimpleUrlHandlerMapping
* Use of InternalResourceViewResolver
* Use of referenceData(),Spring API
* Use of formBackingObject(),Spring API

**Q. AbstractController Example In Spring MVC?**

**Ans:** **AbstractController :** It provides a basic infrastructure and all of Spring’s Controller inherit from AbstractController. It offers caching support, setting of the mime-type etc. It has an abstract method ‘handleRequestInternal(HttpServletRequest, HttpServletResponse)’ which should be overridden by subclass. For example a class extending AbstractController and its declaration in the web application context give below.

This TestController will tell the client to cache things for 2 minutes before rechecking. This controller also returns a ModelAndView object having ‘Test’ as view, ‘message’ as model and ‘Hello World!’ as model value.

public class TestController extends AbstractController {

public ModelAndView handleRequestInternal(

HttpServletRequest request,

HttpServletResponse response) throws Exception {

ModelAndView mav = new ModelAndView("Test"); //hard coded view name “Test”

mav.addObject("message", "Hello World!");

return mav;

}

}

-----------------------

<bean id="testController" class="TestController">

<property name="cacheSeconds" value="120"/>

</bean>

**Q. ParameterizableViewController Example In Spring MVC?**

**Ans:** ParameterizableViewController : ParameterizableViewController is one of the concrete Spring’s Controller. This controller returns a view name specified in Spring configuration xml file thus eliminates the need of hard coding view name in the controller class as in case of AbstractController. ParameterizableViewController is used when you don’t need to implement any logic in controller and just want to redirect to another view.

Consider the following configuration. Spring returns a view name by looking the value of property ‘viewName’ of ParameterizableViewController bean. In this case when testController is called, this testController (which is ParameterizableViewController) will return view name as ‘Test’.

*<bean name="testController"*

*class="org.springframework.web.servlet.mvc.ParameterizableViewController">*

*<property name="viewName" value="Test" />*

*</bean>*

**Q. Why to override formBackingObject method in Spring MVC ?**

**Ans:** You should override formBackingObject(HttpServletRequest request) if you want to provide view with model object data so that view can be initialized with some default values e.g. Consider in your form view there are three text boxes along with other components like text boxes, radio buttons and you want when this form is shown to user two of three check boxes get checked/selected by default. For this you need to override the formBackingObject method. In this method you can create model object with default values and then return model object containing default values to the view.

**Q. What is the use of overriding referenceData(HttpServletRequest request)?**

**Ans:** Overridden referenceData() method is used to provide some additional information to the formView (fillForm.jsp) to construct and display the view e.g.

In your form view you want to give various options to user say in the form of checkboxes and user can select multiple options. For this You can create multiple checkboxes in your view and corresponding multiple instance variables in your command/model object. This would make you view page (jsp usually) and command/model class lengthy. In such a case overriding referenceData(HttpServletRequest request) is very useful. It returns a Map containing list of options depending on these options multiple checkboxes can be created in the view and just one instance variable (array or list probably) needs to be declared in command class. In view we specify the key of the Map which is mapped with list of options. See code fragments given below..

**In view :**

*<td><form:checkboxes items="${availableOptions}" path="options" delimiter=" " /> </td>*

**In SpringFormController :**

@Override

protected Map<String, List<String>> referenceData(HttpServletRequest request) throws Exception {

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

listOfOptions.add("Modem");

listOfOptions.add("Extra cable");

listOfOptions.add("Landline set");

Map<String, List<String>> map = new HashMap<String, List<String>>();

map.put("availableOptions", listOfOptions);

return map;

}

**Command class:**

Private String [] options;

**Q. When and how to use MultiActionController in Spring MVC?**

**Ans:**

**MultiActionController** supports the aggregation of multiple request-handling methods into one controller, so allows you to group related functionality together. It is capable of mapping requests to method names and then invoking the correct method to handle a particular request.

Using the MultiActionController is especially useful when you have a lot of related functionality that would perhaps be appropriate to define all in a single class without having to implement one Controller for each bit of functionality.

There are two ways to use MultiActionController.

• Subclass the MultiActionController and specify the methods that will be resolved by the MethodNameResolver.

• You need to define a delegate object, on which methods resolved by the MethodNameResolver will be invoked. You will need to inject delegate object into the MultiActionController bean as a collaborator.

**Q. Which are the different method name resolvers in MultiActionController ?**

**Ans:** MultiActionController needs some way to resolve which method to call when handling an incoming http request. Spring provides MethodNameResolver interface to achieve this. The MultiActionController class provides a property named ‘methodNameResolver’ so that you can inject a MethodNameResolver.

There are three different types of MethodNameResolver listed below

1) [InternalPathMethodNameResolver](http:///h)

2) [ParameterMethodNameResolver](http:///h)

3) [PropertiesMethodNameResolver](http:///h)

**Q. Which are the different method name resolvers in MultiActionController ?**

**Ans:** MultiActionController needs some way to resolve which method to call when handling an incoming http request. Spring provides MethodNameResolver interface to achieve this. The MultiActionController class provides a property named ‘methodNameResolver’ so that you can inject a MethodNameResolver.

There are three different types of MethodNameResolver listed below

1) [InternalPathMethodNameResolver](http:///h)

2) [ParameterMethodNameResolver](http:///h)

3) [PropertiesMethodNameResolver](http:///h)