Spring MVC

Authored by : Asfiya Khan ***Presented by: Asfiya Khan***

Agenda

- MVC Architecture
- Components in Spring MVC module
- Spring MVC

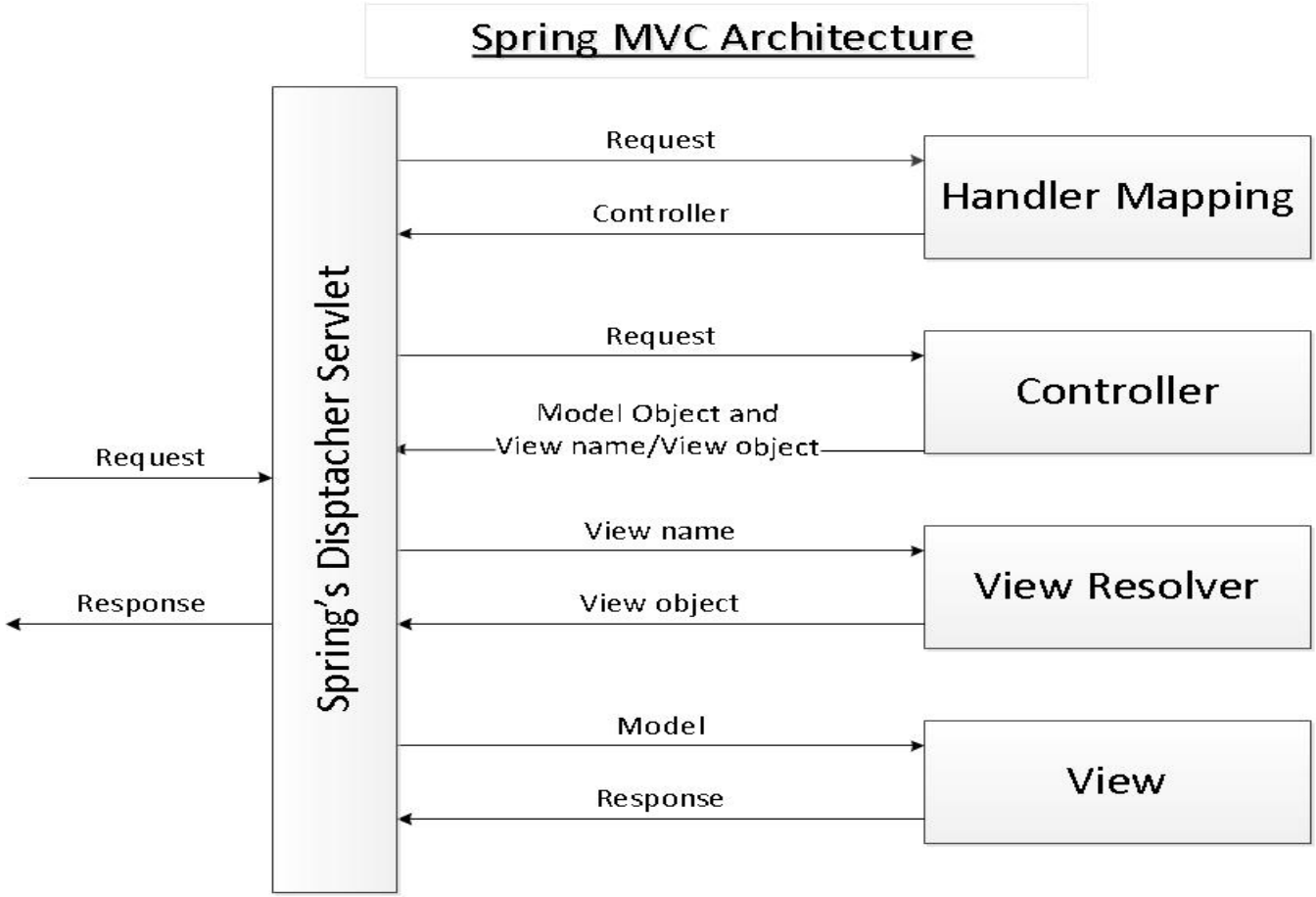
Spring MVC Framework

- Spring MVC framework is based on the Model - View - Controller (MVC) design pattern which helps in separating the business logic, presentation logic and navigation logic.
- Models are responsible for encapsulating the application data.
- The Views render response to the user with the help of the model object .
- Controllers are responsible for receiving the request from the user and calling the back-end services.

Spring MVC Features

- Spring's own MVC structure providing modularity
- Spring's own custom tag library used in JSP
- Configuration not tied to specific presentation technology
- Supports client side data format validations.
- IOC and AOP support can be wired in MVC configurations.
- Supports Internationalization

Spring MVC Architecture



Components

- DispatcherServlet
- A single FrontController Servlet

- Controller
- It processes the request, invoking the business layer

- HandlerMapper
- DispatcherServlet takes help of HandlerMapper to find out exact Controller for that request

- ModelAndView
- A controller bundles model data and name of a view inside the ModelAndView and sends back to Servlet

Components

- ViewResolver

  DispatcherServlet asks a ViewResolver to find the view such as actual JSP

- View

  It will use the model data to render a page that will be sent back to browser

Controller

- DispatcherServlet and Controllers are handling the responsibility of “Controller” in Spring MVC
- DispatcherServlet takes the input request and forwards it to an appropriate Controller
- It is provided by the Spring Framework to developers
- Controllers are responsible for delegating the task to actual Business components

  A controller is an interface to the application’s functionality provided by Spring framework

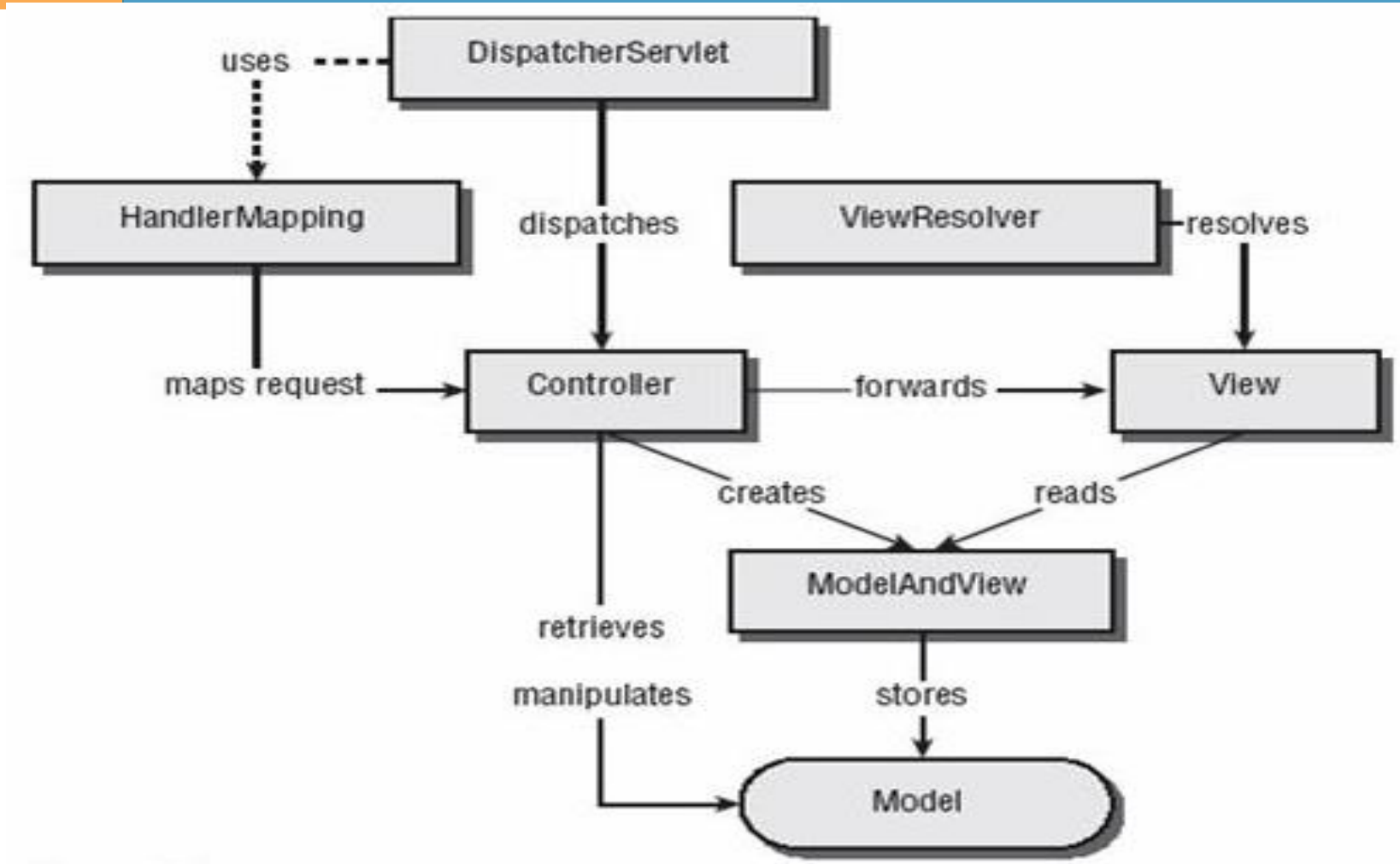
Model

- The Model is based on the Map interface that allows for the abstraction of the view technology
- The ModelAndView object encapsulates the relationship between view and model and is returned by corresponding Controller method
- The ModelAndView class uses a ModelMap class that is a custom Map implementation where data is stored in key-value format

View

- The View page can be explicitly returned as a part of ModelAndView object by the Controller
- The View name can be independent of view technology (i.e without using .jsp in a controller) and resolved to specific technology by using **ViewResolver** and rendered by the **View**
- Spring enables to use JSPs, XSLT, PDFs, Excels, Tiles as the different view types

Spring Architecture



Configuring the DispatcherServlet

Configure the DispatcherServlet in web.xml

<servlet> <servlet-name>empServlet</servlet-name>

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

</servlet-class>

</servlet>

DispatcherServlet will try to load the application context from a file named empServlet-servlet.xml

Add the servlet mapping entry for above in web.xml

<servlet-mapping>

<servlet-name>empServlet</servlet-name>

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

</servlet-mapping>

Configuring the Controller

- Configure the bean for Controller in Spring's configuration file

```
<bean name="/emp.htm“  class="com.emp.EmpListController">
      <property name=“empService" ref=“empService" />
</bean>
```

Any service implementation can be injected in Controller bean

- Write a Java Bean extending from *AbstractController* and override the *handleRequestInternal* method returning the *ModelAndView*

```
return new ModelAndView(“list”, “empl” , empList );
```

list = logical name of a view

empl = name of the Model object

empList = Actual Model object

Declaring the View Resolver

- Configure the View Resolver to resolve logical view name such as “list” returned in ModelAndView to an actual JSP file

```
<bean id="viewResolver“
    class="org.springframework.web.servlet.view.
                        InternalResourceViewResolver">
  <property name="prefix"> <value>/WEB-INF/</value>
  </property>
  <property name="suffix"><value>.jsp</value></property>
</bean>
```

- If ModelAndView returns logical view name as list, then *InternalResourceViewResolver* will try to find the view at /WEB-INF/list.jsp

Mapping Requests to Controllers

- Handler mappings help DispatcherServlet figure out which controller the request should be sent to
- It typically maps a specific controller bean to a URL pattern
- If No handler mapping is found in the context DispatcherServlet creates a default BeanNameUrlHandlerMapping
- Spring MVC's handler mappings implement the org.springframework.web.servlet.HandlerMapping interface

Types Of Controllers

- Core Controllers

  The base interface *Controller* gives the basic functionality . AbstractController class implements this interface
- Command Controllers
  - Accept one or more parameters from the request and binds them to the object
  - Capable of performing parameter validations.
- Form Controllers

  Used in cases where entry form is to be displayed to the User and processing of the entered data needs to be done

Types Of Controllers

- View Controllers
  Used when controller needs to display a static view where no processing or retrieval of data is needed

- Multiaction Controller
  Used where application has several actions that perform similar logic

Command Controllers in Spring

- Command controllers automatically bind HTTP request parameters to a command object
  - A command Object is like an ActionForm in Struts but it is a POJO without extending from any Spring specific class
  - A controller should extend from *AbstractCommandController,* overriding the *handle* method and setting the name of Command class in the controller's constructor

Command Controllers in Spring

- Spring attempts to match request parameters to properties in Command object and if name matches parameter values are automatically bound to the properties.
- Then Spring calls handle method of the Controller

Form Controllers

- Form Controllers extend from Command Controllers and add functionality to display a Form when GET request is received and process the Form when POST is received
- A controller should extend from *SimpleFormController*, overriding the *onSubmit* method.
- If dependent objects need to be added to Command object then override *formBackingObject* method
- SimpleFormController keeps view details in config file, out of java code wiring the following properties inside Controller

  formView

  The logical name of a view to display when the controller receives an HTTP GET request or when any errors are encountered

  successView

  The logical name of a view to display when the form has been submitted successfully

  Validator

  class responsible for validating the input

Validations using *Validator* interface

- Write custom class extending from *Validator* interface implementing validate and supports methods
  - supports

    It is used by Spring to determine whether validator can be used for the given class
  - validate

    It examines fields of object passed to it and rejects any invalid values via Errors object
- Configure the controller wiring *validator* property in it along with the formView and successView

```
<bean id="empController" class="com.emp.EmployeeController">
<property name="formView" value="addRant" />
<property name="successView" value="rantAdded" />
<property name="validator">
<bean class="com.emp.EmployeeValidator" />
</property>
</bean>
```

Validating with Commons Validator

Configure the *DefaultBeanValidator*, which is the Spring module's implementation of Validator. Wire the validatorFactory property

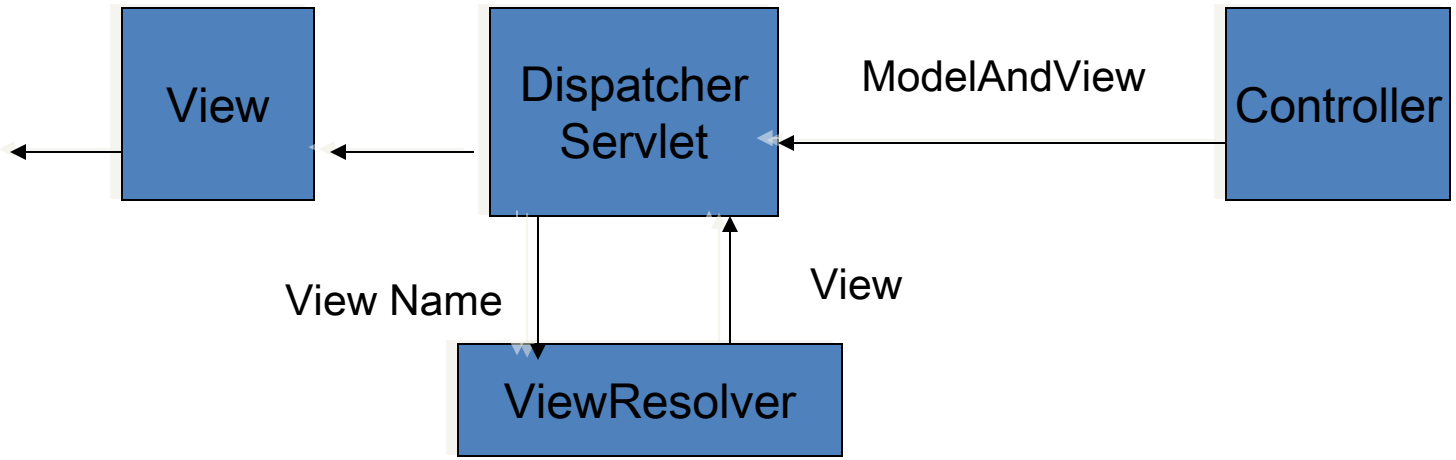
```
<bean id="beanValidator" class=
"org.springmodules.commons.validator.DefaultBeanValidator">
<property name="validatorFactory" ref="validatorFactory" />
</bean>
```

Configure validatorFactory

```
<bean id="validatorFactory" class=
"org.springmodules.commons.validator.DefaultValidatorFactory">
    <property name="validationConfigLocations">
        <list><value>WEB-INF/validator-rules.xml</value>
            <value>WEB-INF/validation.xml</value>
        </list>
    </property>
</bean>
```

View Resolvers in Spring

- In Spring, a View is a bean that renders results to the user
- A logical view name given to ModelAndView object is resolved to a View bean by the ViewResolvers
- A View Resolver is any bean that implements org.springframework.web.servlet.ViewResolver



Types of View Resolvers

View Resolver	How it Works
InternalResource ViewResolver	Resolves logical view names into View objects that are rendered using template file resources such as JSPs, Velocity templates.
BeanName ViewResolver	Finds View interface implementation as a bean in Spring Context, assuming that bean name is logical view name.
ResourceBundle ViewResolver	Uses property file that maps logical view name to the View interface implementation.
XmlViewResolver	Resolves View beans from an XML file that is defined separately from the application context definition files.

Multiple View Resolvers in single Context

```
<bean id="viewResolver" class=
"org.springframework.web.servlet.view. InternalResourceViewResolver">
   <property name="prefix"><value>/WEB-INF/jsp/</value></property>
   <property name="suffix"><value>.jsp</value></property>
</bean>
<bean id="beanNameViewResolver" class=
 "org.springframework.web.servlet.view.BeanNameViewResolver">
   <property name="order"><value>1</value></property>
</bean>
<bean id="xmlFileViewResolver" class=
 "org.springframework.web.servlet.view.XmlFileViewResolver">
   <property name="location">
            <value>/WEB-INF/views.xml</value>
   </property>
   <property name="order"><value>2</value></property>
</bean>
```

Any Questions?