

SPRING MVC

Outline

- What is and Why Spring MVC?
- Request life-cycle
- DispatcherServlet
- URL Handler mapping
- Controllers
- View & View Resolvers

What is Spring MVC?

- Spring MVC is a request-based Web application framework. The framework defines strategy interfaces for all of the responsibilities which must be handled by a modern request-based framework
- It takes advantage of Spring design principles
 - Dependency Injection
 - Interface-driven design
 - POJO without being tied up with a framework

Why Spring MVC?

- Spring provides a very clean division between controllers, JavaBean models, and views
- *Powerful and straightforward configuration of both framework and application classes as JavaBeans*
- *Adaptability, non-intrusiveness, and flexibility*
- *Reusable business code, no need for duplication*
- *Customizable handler mapping and view resolution*
- *Flexible model transfer*

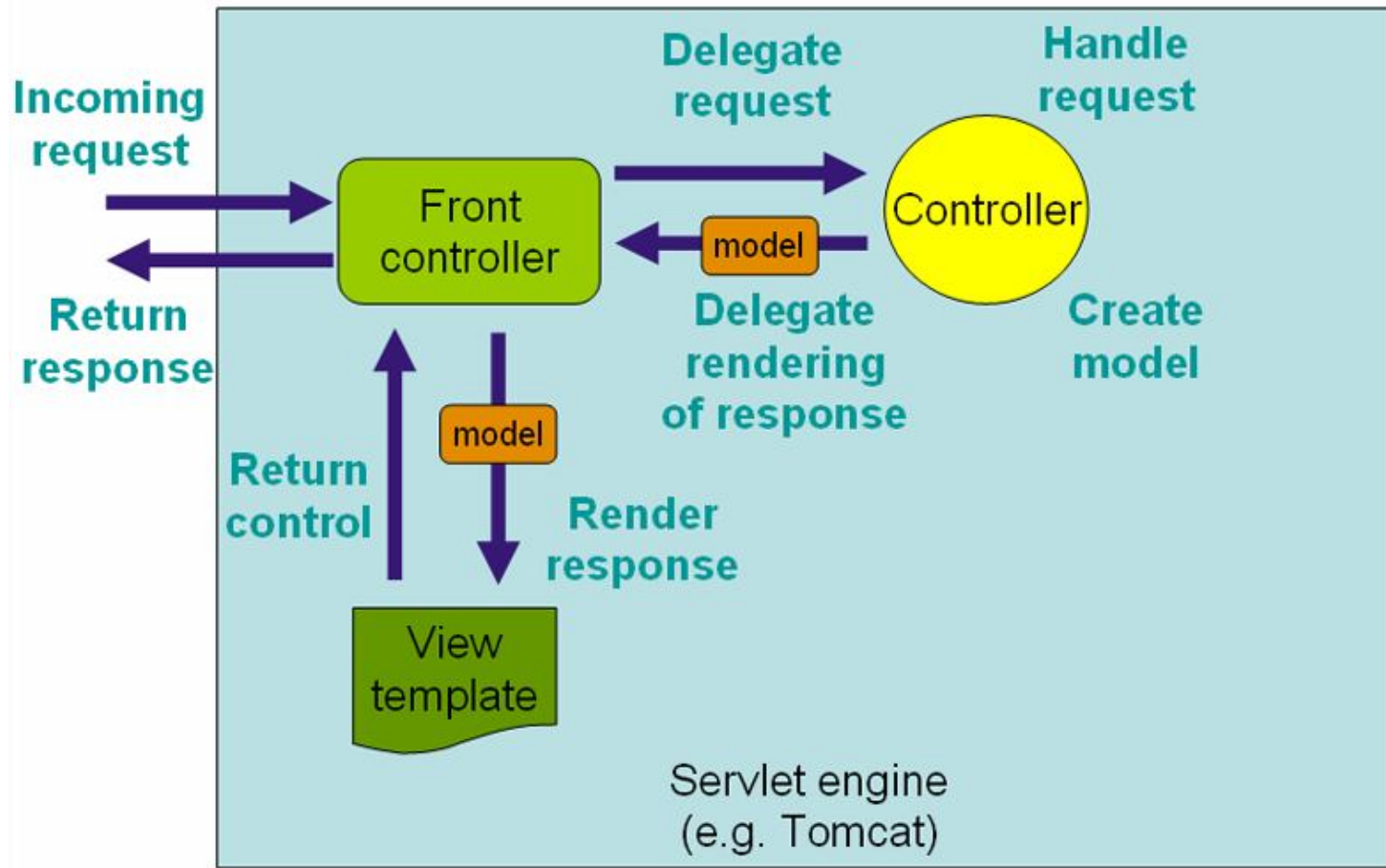
Why Spring MVC?

- *Customizable locale and theme resolution, support for JSPs with or without Spring tag library, support for JSTL, support for Velocity without the need for extra bridges, and so on*
- *A simple yet powerful JSP tag library known as the Spring tag library*
- *A JSP form tag library, introduced in Spring 2.0, that makes writing forms in JSP pages much easier*
- *Beans whose lifecycle is scoped to the current HTTP request or HTTP Session*

Request Life-cycle

- *DispatcherServlet receives the HTTP request*
- URL Handler mapping
 - Controller is invoked
 - Controller returns *ModelAndView object*
- *ViewResolver selects a view*

Request Life-cycle



DispatcherServlet

- The Spring Web MVC framework is designed around a DispatcherServlet
- DispatcherServlet is an expression of the “Front Controller” design pattern
- Dispatches requests to Handler Mapping which invokes the appropriate controller
- DispatcherServlet completely integrated with the Spring IoC container and as such allows you to use every other feature that Spring has.

Configuring DispatcherServlet

- The DispatcherServlet is an actual Servlet (it inherits from the HttpServlet base class), and is declared in the web.xml

```
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

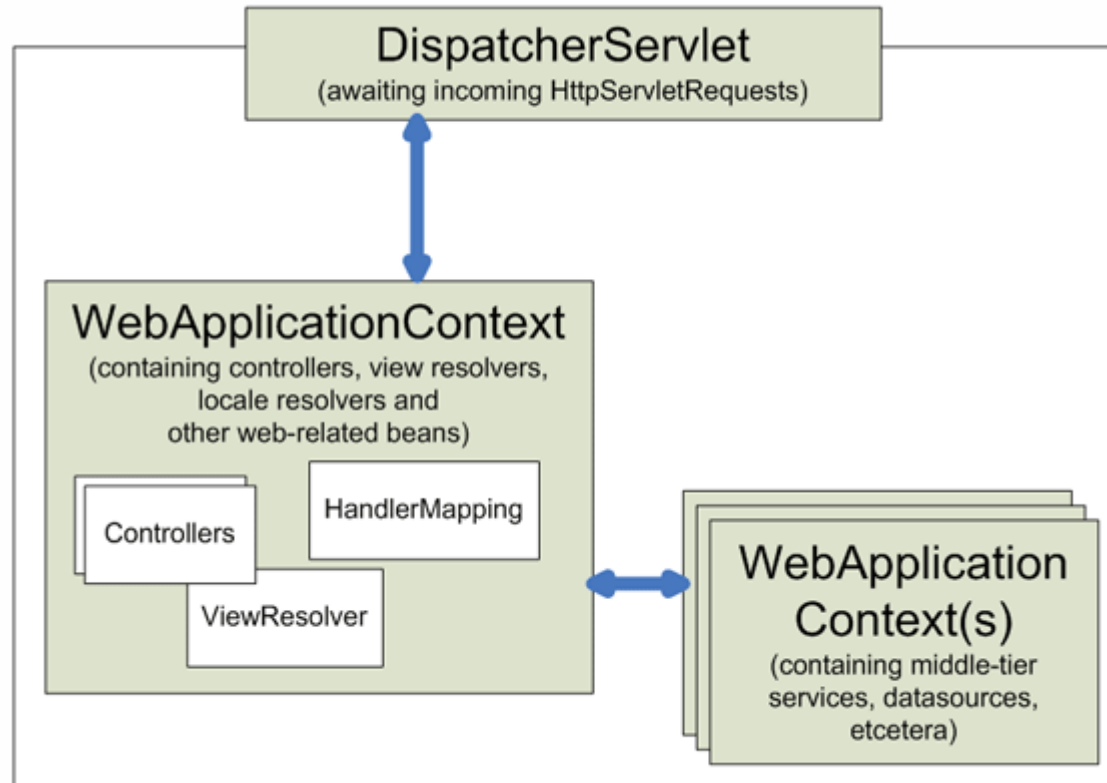
.....

<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

Web ApplicationContext

- Each DispatcherServlet has its own WebApplicationContext
- Inherits all the beans already defined in the root WebApplicationContext
- Upon initialization of a DispatcherServlet, the framework *looks for a file named [servlet-name]-servlet.xml* in the WEB-INF

Web ApplicationContext



Web ApplicationContext

- The `WebApplicationContext` is an extension of the plain `ApplicationContext`
- It has some extra features necessary for web applications.
- It differs from a normal `ApplicationContext` in that it is capable of resolving themes , and that it knows which servlet it is associated with
- The `WebApplicationContext` is bound in the `ServletContext`

Loading More than One Context File

- By default Dispatcher servlet will load only one context configuration file
- For additional context configuration file you need to configure 'Context Loader'

```
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>

.....

<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/appServlet-service.xml
    /WEB-INF/appServlet-data.xml
    /WEB-INF/appServlet-security.xml
  </param-value>
</context-param>
```

DispatcherServlet Configuration

- The Spring DispatcherServlet uses special beans to process requests and render the appropriate views
- These beans are part of Spring Framework
- You can configure them in the `WebApplicationContext`, just as you configure any other bean, however, for most beans, sensible defaults are provided so you initially do not need to configure them

DispatcherServlet Configuration

- **HandlerMapping**
 - Routing of requests to handlers
- **HandlerExceptionResolver**
 - Maps exceptions to error pages
 - Similar to standard Servlet, but more flexible
- **ViewResolver**
 - Maps symbolic name to view
- **MultipartResolver**
 - Handling of file upload
- **LocaleResolver**
 - Default uses HTTP accept header, cookie, or session

Spring MVC at Glance

- Write the controller class that performs the logic behind the homepage.
- Configure the controller in the DispatcherServlet's context configuration file
- Configure a view resolver to tie the controller to the view.
- Write the view that will render the homepage to the user

Url Handler Mappings

- Instructs *DispatcherServlet* which *Controller* to invoke for a request
 - Dependency Injection
- Implements *HandlerMapping* interface
- Spring MVC comes with two implementation classes of *HandlerMapping* interface
 - *SimpleUrlHandlerMapping*
 - *BeanNameUrlHanlderMapping*

SimpleUrlHanlderMapping

```
<bean id="urlMapping"  
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">  
  <property name="mappings">  
    <props>  
      <prop key="/login">loginController</prop>  
      <prop key="/employee">employeeController</prop>  
    </props>  
  </property>  
</bean>
```

BeanNameUrlHanlderMapping

```
<bean id="defaultHandlerMapping"  
      class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>
```

```
<bean name="/login" class="loginController"/>
```

```
<bean name="/employee" class="employeeController"/>
```

DefaultAnnotationHandlerMapping

- With the introduction of Spring 2.5, the DispatcherServlet enables the DefaultAnnotationHandlerMapping, which looks for @RequestMapping annotations on @Controllers.

```
<beans>
```

```
    <bean id="handlerMapping"  
        class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"/>
```

```
</beans>
```

- To enable autodetection of such annotated controllers

```
<beans...>
```

```
    <context:component-scan base-package=  
        "com.examples.spring.web.mvc"/>
```

```
    // ...
```

```
</beans>
```

Controllers

- Receives requests from DispatcherServlet and interacts with business tier
- Implements the Controller interface
- Returns ModelAndView object
- ModelAndView contains the model (a Map) and either a logical view name, or implementation of View interface

Controller Classes

- AbstractController
 - BaseCommandController
- AbstractCommandController
- AbstractFormController
 - SimpleFormController
 - AbstractWizardController
 - MultiActionController
 - ParameterizableViewController

Annotation-based Controller

- Uses annotations such as `@Controller`, `@RequestMapping`, `@RequestParam`, `@ModelAttribute`
 - Controllers implemented in this style do not have to extend specific base classes or implement specific interfaces.
 - Furthermore, they do not usually have direct dependencies on Servlet APIs, although you can easily configure access to Servlet facilities.

Example

- Controller Class

@Controller

```
public class HelloWorldController {  
    @RequestMapping("/helloWorld")  
    public ModelAndView helloWorld() {  
        ModelAndView mav = new ModelAndView();  
        mav.setViewName("helloWorld");  
        mav.addObject("message", "Hello World!");  
        return mav;  
    }  
}
```


Annotation Used

- **@Controller**

Indicates that a particular class serves the role of a controller

- **@RequestMapping**

To map URLs such onto an entire class or a particular handler method

- **@RequestParam**

Use the @RequestParam annotation to bind

request parameters to a method parameter

Annotation Used

- `@ModelAttribute`
Used controller gets a reference to the object holding the data entered in the form
- `@SessionAttributes`
Declares session attributes used by a specific handler
- `@CookieValue`
Allows a method parameter to be bound to the value of an HTTP cookie
- `@RequestHeader`
Allows a method parameter to be bound to a request header

Advanced @RequestMapping

- @PathVariable

Method parameter to indicate that a method parameter should be bound to the value of a URI template variable

@Controller

@RequestMapping("/owners/{ownerId}")

public class RelativePathUriTemplateController {

 @RequestMapping(value = "/pets/{petId}")

 public void findPet(@PathVariable String ownerId,
 @PathVariable String petId, Model model) {

 // implementation omitted

 }

 }

Advanced @ModelAttribute

- @ModelAttribute has two usage scenarios in controllers.
- When you place it on a method parameter, it maps a model attribute to the specific, annotated method parameter

```
public String processSubmit(@ModelAttribute("pet")  
    Pet pet, BindingResult result, SessionStatus status)
```

- You can also use it at the method level to provide *reference data* for the model

```
@ModelAttribute("types")  
public Collection<PetType> populatePetTypes() {  
    return this.clinic.getPetTypes();  
}
```

View & View Resolvers

- The two interfaces that are important to the way Spring handles views are **ViewResolver** and **View**.
 - The ViewResolver provides a mapping between view names and actual views.
 - The View interface renders the output of the request to the client

View

- Implements the *View interface*
- The View interface addresses the preparation of the request and hands the request over to one of the view technologies
- Built-in support for
 - JSP, XSLT, Velocity, Freemaker
 - Excel, PDF, JasperReports

View Resolvers

- Resolves logical view names returned from controllers into *View objects*
- Implements *ViewResolver interface*
 - *View resolveViewName(String viewName, Locale locale) throws Exception*
- Spring provides several implementations
 - *UrlBasedViewResolver*
 - *BeanNameViewResolver*
 - *ResourceBundleViewResolver*
 - *XmlViewResolver*

ResourceBundleViewResolver

- When you combine different view technologies in a web application, you can use the ResourceBundleViewResolver.

```
<bean id="viewResolver"  
class="org.springframework.web.servlet.view.ResourceBundleViewResolv  
er">  
    <property name="basename" value="views"/>  
    <property name="defaultParentView" value="parentView"/>  
</bean>
```

- The ResourceBundleViewResolver inspects the ResourceBundle identified by the basename, and for each view it is supposed to resolve, it uses the value of the property [viewname].(class) as the view class and the value of the property [viewname].url as the view url.

UrlBasedViewResolver

- With JSP as a view technology, you can use the UrlBasedViewResolver.
- This view resolver translates a view name to a URL and hands the request over to the RequestDispatcher to render the view.

```
<bean id="viewResolver"
class="org.springframework.web.servlet.view.UrlBasedViewResolver"
>
    <property name="viewClass"

value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>
```

- When returning test as a logical view name, this view resolver forwards the request to the RequestDispatcher that will send the request to /WEB-INF/jsp/test.jsp.

Chaining View Resolver

- Spring supports multiple view resolvers.
- Chain resolvers and override specific views in certain circumstances.
- This is done by adding more than one resolver to your application context
- Set the order property to specify ordering. Remember, the higher the order property, the later the view resolver is positioned in the chain

Chaining View Resolver

```
<bean id="jspViewResolver" class=
"org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass"
        value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>
<bean id="excelViewResolver"
class="org.springframework.web.servlet.view.XmlViewResolver">
    <property name="order" value="1" />
    <property name="location" value="/WEB-INF/views.xml" /> </bean>
```

<!-- in views.xml -->

```
<bean name="report" class="org.springframework.example.ReportExcelView"/>
```

Thank You!