Spring MVC

Topics

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

What is and Why Spring MVC?

What is Spring MVC?

- Web application framework that takes advantage of Spring design principles
  - Dependency Injection
  - Interface-driven design
  - POJO without being tied up with a framework

Why Spring MVC?

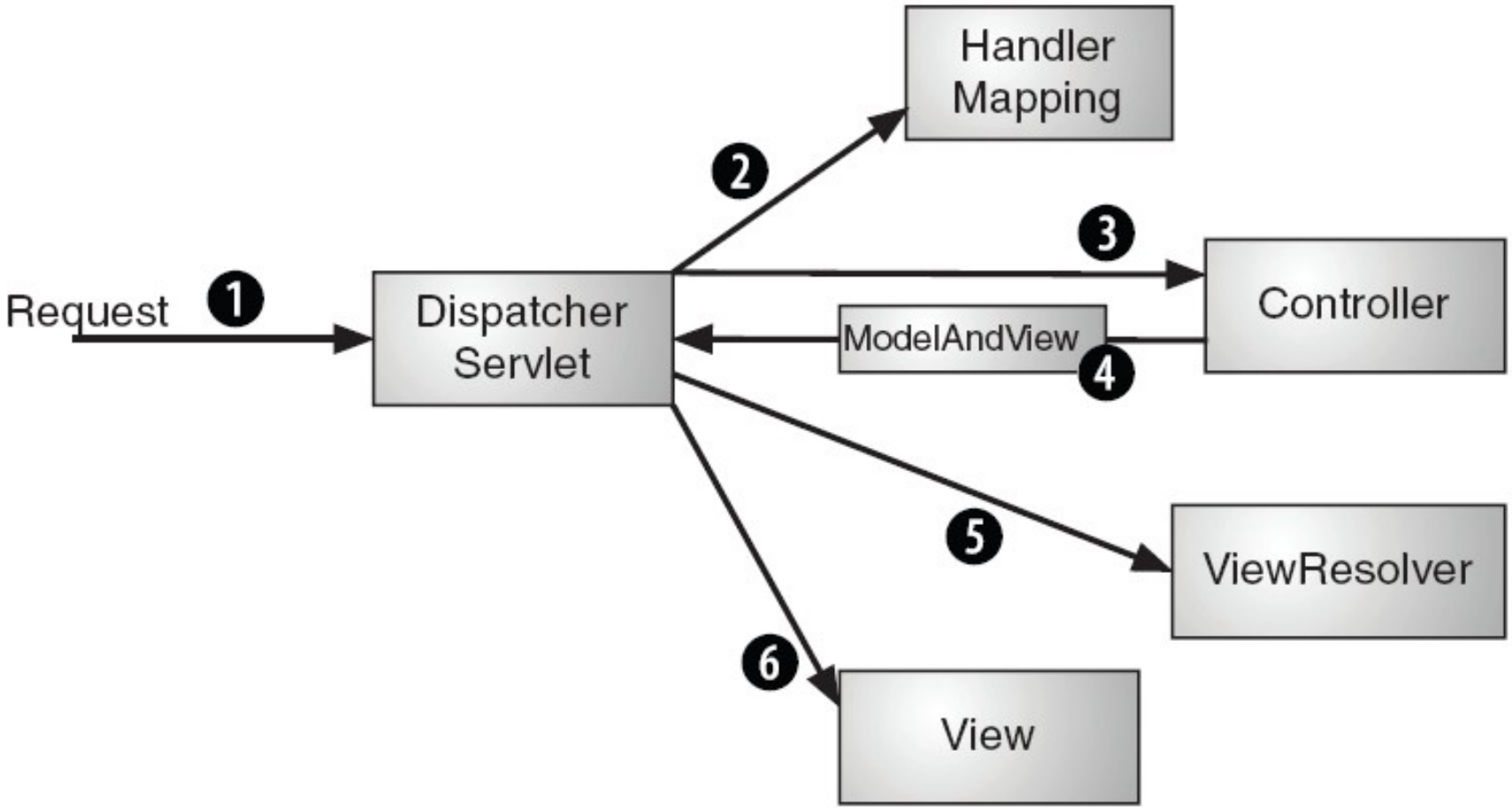
- Testing through Dependency Injection
- Binding of request data to domain objects
- Form validation
- Error handling
- Multiple view technologies
  - JSP, Velocity, Excel, PDF
- Page workflow

Request Life-cycle

Request Life-cycle

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

Request Life-cycle



DispatcherServlet

DispatcherServlet Configuration

- HandlerMapping
  - Routing of requests to handlers
- HandlerAdapter
  - Adapts to handler interface. Default utilizes *Controller*s
- HandlerExceptionResolver
  - Maps exceptions to error pages
  - Similar to standard Servlet, but more flexible
- ViewResolver
  - Maps symbolic name to view

DispatcherServlet Configuration

- MultipartResolver
  - Handling of file upload
- LocaleResolver
  - Default uses HTTP accept header, cookie, or session

Configuring DispatcherServlet

```
<servlet>
    <servlet-name>roadrantz</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet
                                                    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
.....
<servlet-mapping>
        <servlet-name>roadrantz</servlet-name>
                <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

Loding 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/roadrantz-service.xml
/WEB-INF/roadrantz-data.xml
/WEB-INF/roadrantz-security.xml
</param-value>
</context-param>
```

Spring MVC at Glance

1 Write the controller class that performs the logic behind the homepage.

2 Configure the controller in the DispatcherServlet's context configuration file

3 Configure a view resolver to tie the controller to the JSP.

4 Write the JSP that will render the homepage to the user

URL Handler Mapping

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
- *BeanNameUrlHanlderMapping*
- *SimpleUrlHandlerMapping*

SimpleUrlHanlderMapping

- Map requests to controllers
- Supports direct matchs and wildcards
  - given "/test" -> registered "/test" (direct match)
  - given "/test" -> registered "/t*" (wildcards)

Example:SimpleUrlHanIderMapping

```
<bean id="urlMapping"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"
    >
    <property name="mappings">
        <props>
            <prop key="/welcome.htm">clinicController</prop>
            <prop key="/vets.htm">clinicController</prop>
            <prop key="/findOwners.htm">findOwnersForm</prop>
            <prop key="/owner.htm">clinicController</prop>
            <prop key="/addOwner.htm">addOwnerForm</prop>
            <prop key="/editOwner.htm">editOwnerForm</prop>
            <prop key="/addPet.htm">addPetForm</prop>
            <prop key="/editPet.htm">editPetForm</prop>
            <prop key="/addVisit.htm">addVisitForm</prop>
        </props>
    </property>
</bean>
```

Controllers

Controllers

- Receives requests from DispatcherServlet and interacts with business tier
- Implements the Controller interface

```
ModelAndView
    handleRequest(HttpServletRequest req,
    HttpServletResponse resp) throws Exception
```

  - Returns ModelAndView object
  - ModelAndView contains the model (a Map) and either a logical view name, or implementation of View interface

Controller Classes

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

AbstractController

- Convenient superclass for controller implementations
- Workflow
  - *handleRequest*() of the controller will be called by the *DispatcherServlet*
  - the located Controller is then responsible for handling the actual request and - if applicable - returning an appropriate *ModelAndView*
  - handleRequest() method calls abstract method *handleRequestInternal*() , which should be implemented by extending classes to provide actual functionality to return *ModelAndView* objects.

BaseCommandController

- Controller implementation which creates an object (the command object) on receipt of a request and attempts to populate this object with request parameters
- Workflow
  - Since this class is an abstract base class for more specific implementation, it does not override the *handleRequestInternal*() method and also has no actual workflow

AbstractFormController

- Form controller that auto-populates a form bean from the request. This, either using a new bean instance per request, or using the same bean when the *sessionForm* property has been set to true.
- This class is the base class for both framework subclasses like *SimpleFormController* and *AbstractWizardFormController*, and custom form controllers you can provide yourself

SimpleFormController

- Handles single page from input
- Most commonly used command controller
- Split into two workflows
  - Form request
- Load form backing object and reference data
- Show form view
  - Form submission
- Load from backing object
- Bind and validate from backing object
- Execute submission logic
- Show success view

SimpleFormController: Form View

- Controller receives a request for a new form (typically a GET)
- formBackingObject()
  - to load or create an object edited by the form
- initBinder()
  - to register custom editors for fields in the command object
- showForm()
  - to return a view to be rendered
- referenceData()
  - to add data needed by the form (select list) to the model

SimpleFormController: Form Submission

- Controller receives a form submission (typically a POST)
- formBackingObject()
  - to load or create an object edited by the form
- Request data is bound to the form backing object
- onBind()
  - to perform custom processing after binding but before validation
- Validator is invoked
- onBindAndValidate()
  - to do custom processing after binding and validation
- onSubmit()
  - to do custom submission processing

SimpleFormController

```
public class LoginBankController extends
    SimpleFormController {

protected ModelAndView onSubmit(Object command)
    throws Exception{

LoginCommand loginCommand = (LoginCommand)
    command;

authenticationService.authenticate(loginCommand);

AccountDetail accountdetail =

accountServices.getAccountSummary(loginCommand.get
    UserId());

return new

ModelAndView(getSuccessView(),"accountdetail",accoun
    tdetail);

}
```

SimpleFormController Configuration

```
<bean id="loginBankController"
class="springexample.contoller.LoginBankController">
<property name="sessionForm">
<value>true</value>
</property>
<property name="commandName">
<value>loginCommand</value>
</property>
<property name="commandClass">
<value>springexample.commands.LoginCommand</value>
</property>
```

SimpleFormController Configuration

```
<property name="authenticationService">
<ref bean="authenticationService" />
</property>
<property name="accountServices">
<ref bean="accountServices" />
</property>
<property name="formView">
<value>login</value>
</property>
<property name="successView">
<value>accountdetail</value>
</property>
</bean>
```

MultiActionController

- Controller implementation that allows multiple request types to be handled by the same class.
- Subclasses of this class can handle several different types of request with methods of the form
  - *(ModelAndView | Map | void) actionName(HttpServletRequest request, HttpServletResponse response);*
- Request to *actionName* mapping is resolved via *methodNameResolver* property in the configuration file

MethodNameResolver Implementations

- *InternalPathMethodNameResolver*
  - The method name is taken from the last part of the path /servlet/foo.html -> foo(...)
  - Default behavior
- *ParameterMethodNameResolver*
  - The method name is taken from the specified request parameter
  - The default parameter name is *action*
- *PropertiesMethodNameResolver*
  - The method name is resolved via <prop>

PropertiesMethodNameResolver

```
<!-- This bean is a MultiActionController that manages general View
rendering. It uses the "clinicControllerResolver" bean below for methodname resolution.-->
<bean id="clinicController"
class="org.springframework.samples.petclinic.web.ClinicController">
<property name="methodNameResolver" ref="clinicControllerResolver"/>
<property name="clinic" ref="clinic"/>
</bean>
<!-- This bean is a MethodNameResolver definition for a MultiActionController. It maps URLs
     to methods for the "clinicController" bean.-->
<bean id="clinicControllerResolver"
class="org.springframework.web.servlet.mvc.multiaction.PropertiesMethod NameResolver">
<property name="mappings">
<props>
<prop key="/welcome.htm">welcomeHandler</prop>
<prop key="/vets.htm">vetsHandler</prop>
<prop key="/owner.htm">ownerHandler</prop>
</props>
</property>
</bean>
```

View & View Resolvers

View

- Renders the output of the request to the client
- Implements the *View* interface
- 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
  - *InternalResourceViewResolver*
  - *BeanNameViewResolver*
  - *ResourceBundleViewResolver*
  - *XmlViewResolver*

ResourceBundleViewResolver

- The View definitions are kept in a separate configuration file
  - You do not have to configure view beans in the application context file
- Supports internationalization (I18N)

ResourceBundleViewResolver

```
<!-- This bean provides explicit View mappings in a resource bundle
instead of the default InternalResourceViewResolver. It fetches
the view mappings from localized "views_xx" classpath files, i.e.
"/WEB-INF/classes/views.properties" or "/WEBINF/
classes/views_de.properties". Symbolic view names returned by
Controllers will be resolved by this bean using the respective
properties file, which defines arbitrary mappings between view
names and resources. -->
<bean id="viewResolver" class="org.springframework.web.servlet.view.ResourceBundleViewRes
olver">
    <property name="basename" value="views"/>
</bean>
```

Example: views.properties

```
# This is from petclinic sample application

welcomeView.(class)=org.springframework.web.servlet.view.JstlView

welcomeView.url=/WEB-INF/jsp/welcome.jsp

vetsView.(class)=org.springframework.web.servlet.view.JstlView

vetsView.url=/WEB-INF/jsp/vets.jsp

# A lot more are defined
```

Example: Returning a View

```
public class ClinicController extends MultiActionController
implements InitializingBean {
public ModelAndView welcomeHandler(HttpServletRequest
    request,
HttpServletResponse response) throws ServletException {
return new ModelAndView("welcomeView");
}
public ModelAndView vetsHandler(HttpServletRequest request,
HttpServletResponse response) throws ServletException {
return new ModelAndView("vetsView", "vets",
this.clinic.getVets());
}
```

Validation

Validation Configuration

- <!--Validator for visit forms, implementing Spring's Validator interface. Could also reside in the root application context, as it is generic, but is currently just used within PetClinic's web tier. -->

```
<bean id="visitValidator"

class="org.springframework.samples.petclinic.validation.VisitValidator"/>
```

Validation Class

```
public class VisitValidator implements Validator {

public boolean supports(Class clazz) {

return Visit.class.isAssignableFrom(clazz);

}

public void validate(Object obj, Errors errors) {

ValidationUtils.rejectIfEmpty(errors, "description",

"required", "required");

}

}
```

Important Slide (1)

- Front Controller
  - DispatcherServlet
- Spring Controllers
  - Implements Controllers
  - Has many implementations for form based and non form based applications
  - Use form controllers if you have a form
- HandlerMappings
  - Implements *HandlerMapping* interface
  - Spring MVC comes with two implementation classes of *HandlerMapping* interface
    - *BeanNameUrlHanlderMapping*
    - *SimpleUrlHandlerMapping*

Important Slide (2)

- 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
    - *InternalResourceViewResolver*
    - *BeanNameViewResolver*
    - *ResourceBundleViewResolver*
    - *XmlViewResolver*

Important Slide (3)

- View
  - Renders the output of the request to the client
  - Implements the *View* interface
  - Built-in support for
    - JSP, XSLT, Velocity, Freemaker
    - Excel, PDF, JasperReports