Introduction to Spring MVC

> Dmitry Sklyut
> Matt Swartley

About Chariot Solutions

- Small, high-powered consulting firm
- Focused on Java and open source
- Services include:
  - Architecture: Strategy, infrastructure planning, technology selection, enterprise and application architecture
  - Development: Full lifecycle application development, integration of existing systems, extension and/or rehabilitation of existing Java applications
  - Rapid Response: Quick resolutions to time-critical issues (performance and stability problems, missing features, critical defects, security flaws...)
- If you don’t work for us, you should!
- We're hiring! Email: emulder@chariotsolutions.com

Agenda

- Overview of Spring MVC
- Request Lifecycle
- Form Processing
- New Spring MVC Features in Spring 2.0
- Overview of JSR-168
- Portlet Request Lifecycle
- Spring Web MVC vs. Spring Portlet MVC

Overview of Spring MVC

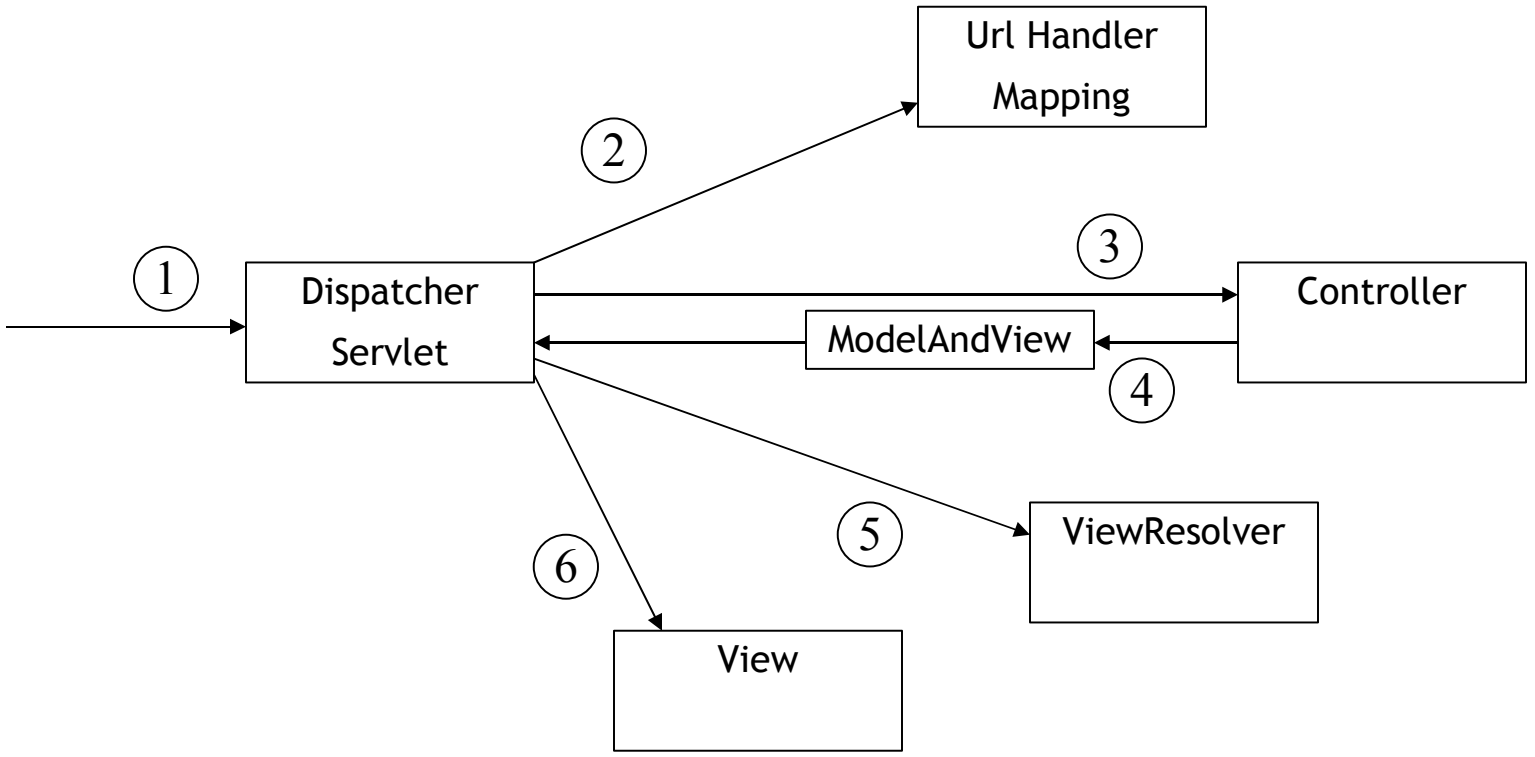
Overview

Spring MVC is a framework that *simplifies* development of the Web Tier

- Simplifies testing through dependency injection
- Simplifies binding request data to domain objects
- Simplifies form validation and error handling
- Simplifies implementation of multiple view technologies (JSP, Velocity, Excel, PDF, …)
- Simplifies multiple page workflow

Request Lifecycle

Request Lifecycle



DispatcherServlet

- Spring MVC's front controller
- Coordinates the entire request lifecycle
- Configured in web.xml

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

- Loads Spring application context from XML file (default is <servlet-name>-servlet.xml) that usually contains <bean> definitions for the Spring MVC components

Url Handler Mappings

- Tell the **`DispatcherServlet`** which controller to invoke for a request
- Implement the **`HandlerMapping`** interface
- Spring provides several implementations
- **`BeanNameUrlHandlerMapping`**

```
<bean name="/index.htm"
    class="beer.controllers.WelcomeController"> …
<bean name="/manageBeers.htm"
    class="beer.controllers.ManageBeersController"> …
<bean name="/editBeer.htm"
    class="beer.controllers.BeerFormController"> …
```

Url Handler Mappings (contd.)

- **SimpleUrlHandlerMapping**

```
<bean id="urlMapping"
  class="org.sfw.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
      <props>
        <prop key="/index.htm">welcomeController</prop>
        <prop key="/manageBeers.htm">manageBeersController</prop>
        <prop key="/editBeer.htm">beerFormController</prop>
      </props>
    </property>
</bean>
```

- **CommonsPathHandlerMapping**
  - Uses source level meta-data (a `PathMap` attribute) compiled into the controller using the Jakarta Commons Atttributes compiler

Controllers

- Receive requests from **`DispatcherServlet`** and coordinate business functionality
- Implement the Controller interface

```
public ModelAndView handleRequest(HttpServletRequest request,
   HttpServletResponse response) throws Exception;
```

- Return instance of **`ModelAndView`** to **`DispatcherServlet`**
- **`ModelAndView`** contains the model (a **`Map`**) and either a logical view name, or an implementation of the **`View`** interface

Controllers (contd.)

- Spring provides many implementations:
  - `AbstractController`
  - `AbstractCommandController`
  - `ParameterizableViewController`
  - `MultiActionController`
  - `SimpleFormController`
  - `AbstractWizardController`
- Can handle requests as commands (as in WebWork) with the `ThrowawayController`

View Resolvers

- Resolve logical view names returned from controllers into View objects
- Implement the **`ViewResolver`** interface
- Spring provides several implementations

View Resolvers (contd.)

- InternalResourceViewResolver

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

View Resolvers (contd.)

- BeanNameViewResolver

```
<bean id="viewResolver"
  class="org.sfw.web.servlet.view.BeanNameViewResolver"/>

<bean id="manageBeers"
  class="org.springframework.web.servlet.view.JstlView">
    <property name="url">
      <value>/WEB-INF/jsp/manageBeers.jsp</value>
    </property>
</bean>

<bean id="manageBeersAsExcel"
  class="beer.views.ManageBeersExcelView"/>
```

View Resolvers (contd.)

- ResourceBundleViewResolver

```
<bean id="viewResolver"
   class="org.sfw.web.servlet.view.ResourceBundleViewResolver">
    <property name="basename"><value>views</value></property>
</bean>
```

views.properties

```
welcome.(class)=org.springframework.web.servlet.view.JstlView
welcome.url=/WEB-INF/jsp/welcome.jsp

manageBeers.(class)=org.springframework.web.servlet.view.JstlView
manageBeers.url=/WEB-INF/jsp/manageBeers.jsp

manageBeersAsExcel.(class)=beer.views.ManageBeersExcelView
```

View Resolvers (contd.)

- **XmlViewResolver**

```
<bean id="viewResolver"
   class="org.sfw.web.servlet.view.XmlViewResolver">
     <property name="location"><value>views</value></property>
</bean>
```

views.xml

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

Views

- Renders the output of the request to the client
- Implement the `View` interface
- Built in support for:
  - **JSP,** XSLT**,** Velocity, Freemarker
  - **Excel,** PDF, JasperReports
- And of course, you can write your own custom implementation

Sample Application (Part 1)

Form Processing

Form Processing

- `SimpleFormController` handles single page form input
- `AbstractWizardFormController` handles multi-page form input
- Flexible Binding System
  - Many default binders (Date, Number, etc.)
  - Easy to create custom binders (`PropertyEditor` implementations)
  - Binding support (in Views) for JSP, Velocity and FreeMarker
- Flexible Validation Framework
  - Injected into Controllers
  - Applied automatically during form processing
  - Text of error messages resolved from ResourceBundle
  - Field level error information available in Views

Some Definitions

- Form Backing Object
  - The object being modified by the form
  - Can be any POJO
  - Typically a class created for a specific form or a domain object
  - Sometimes called the Command object
- Reference Data
  - Supporting (read-only) data needed to render the form
  - Often used to populate drop down lists

SimpleFormController

- Split into two distinct workflows
- Form Request
  - Load form backing object and reference data
  - Show form view
- Form Submission
  - Load form backing object
  - Bind and validate form backing object (show form view on error)
  - Execute submission logic
  - Show success view

SimpleFormController – Form View

- Controller receives a request for a new form (typically a GET)
- **`formBackingObject()`** – Allows you to load or create the object edited by the form
- **`initBinder()`** – Allows you to register custom editors for fields in the command object
- **`showForm()`** – Returns the View to be rendered
- **`referenceData()`** – Allows you to add data needed by the form (select lists, etc.) to the model
- Model gets exposed and view gets rendered

SimpleFormController – Form Submission

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

Custom Binding

- Spring uses PropertyEditors to bind request data to form backing object
- Provides a convenient superclass class called **`PropertyEditorSupport`** for custom property editors
- Need to provide implementations for **`setAsText(String text)`** and **`getAsText()`**

Validators

- Used to validate POJOs – typically form objects and domain objects
- Usually injected into controllers for validation during form processing
- Implement the **`Validator`** interface

```
boolean supports(Class clazz);
void validate(Object obj, Errors errors);
```

- **`Validator`** implementations store details about the errors in the Spring **`Errors`** object
- Spring makes the **`Errors`** object available to views for displaying error information to the user

Sample Form Backing Object

```
public class Beer {
    private long id;
    private String name;
    private Country country;
    private Money price;
    …
}
```

Sample Application (Part 2)

New Spring MVC Features in 2.0

- Expanded tag library
  - Form tags (<spring:form>, <spring:input>, <spring:select>, <spring:textarea>, etc.)
- UrlFilenameViewController
  - Transforms the virtual filename at the end of a URL into a view name and returns that view
  - Can optionally prepend a prefix and/or append a suffix to build the view name
  - /index.htm would return a view name of "index
- ControllerClassNameHandlerMapping
  - WelcomeController gets mapped to /welcome.htm
  - HomeController gets mapped to /home.htm

Thank You!

mswartley@chariotsolutions.com

Spring Portlet MVC

Build Portals the Spring Way

Agenda

- Overview of JSR 168 (Portlets)
- Overview of Spring Portlet MVC
- Basic request lifecycle
- DispatcherPortlet
- Handler Mapping
- Controllers
- View Resolvers
- Form processing

Terminology

- *A portal is a web based application that provides personalization, single sign on and content aggregation*
- *A portlet is a Java technology based web component, managed by a portlet container, which processes requests and generates dynamic content.*

Overview of JSR 168

- JSR 168 is a specification from the Java Community Process for the standardization of portlets. The specification was developed to provide interoperability for running portlets on any vendor's implementation of the JSR 168 portlet container. The specification was approved in October of 2003.
- For more information about the Java Portlet Specification, see the JSR 168 specification: http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/

Overview of JSR 168 (cont…)

- Very similar to the Servlet API
- Rendering is asynchronous
- Session attributes are scoped (Portlet or Application)
- Provides persistent storage for user preferences
- Portlets have Modes and Window States
- Request life cycle is different from Servlets

Portlet Request Life Cycle

- Two types of requests – action and render
- Action request – asking portlet to perform some operation
- Render request – asking portlet to display its content based on the current state
- API:
  - processAction (ActionRequest, ActionResponse)
  - render (RenderRequest, RenderResponse)

Spring Portlet MVC

- Complete MVC framework for JSR 168 Portlets.
- Preserves JSR 168 unique lifecycle, i.e. RenderRequest and ActionRequest separation
- Most of the API is same or very close to Spring Servlet MVC
- Same benefits as Spring Servlet MVC
  - Simplifies testing
  - Simplifies binding form/request data to objects
  - Simplifies handling of multiple view technologies
  - And it is just fun to work with

Dispatcher Portlet

DispatcherPortlet

- Front Controller for the Portlet Application
- Configured in portlet.xml

```
<portlet id="beerPortlet">
    <portlet-name>beer</portlet-name>
    <portlet-class>
      org.springframework.web.portlet.DispatcherPortlet
    </portlet-class>
      <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>VIEW</portlet-mode>
        <portlet-mode>HELP</portlet-mode>
      </supports>
      <portlet-info>
        <title>Beer Portlet</title>
      </portlet-info>
</portlet>
```

DispatcherPortlet (cont…)

- Loads configuration from <portlet-name>-portlet.xml file (configurable with an init-param “namespace”).
- <portlet-name>-portlet.xml contains configuration of the controllers and handler mappings for a given portlet
- Each portlet in the portlet web application has a unique namespace – no co-mingling of beans.

Handler Mapping

- A way to map each request to a controller.
- Can't use URLs for mapping - Portal server is in charge of the URL generation.
- ParameterHandlerMapping
  - Maps based on the request attribute name ("action" by default)
- PortletModeHandlerMapping
  - Maps based on the portlet mode name ("view", "edit", "help", etc)
- PortletModeParameterHandlerMapping
  - Maps based on the combination of the two scenarios, first by mode and second "action"

PortletModeHandlerMapping

```
<!-- Fallback mapper - only defines controllers based on the mode -->
<bean id="portletModeHandlerMapping"
    class="org.s...web.portlet.handler.PortletModeHandlerMapping">      <property
    name="order" value="20"/>
    <property name="portletModeMap">
        <map>
            <entry key="view">
                <ref bean="welcomeController"/>
            </entry>
            <entry key="help">
                <ref bean="helpController"/>
            </entry>
        </map>
    </property>
</bean>
```

PortletModeParameterHandlerMappin

```
<!-- In case of an ActionRequest pass request attribute to the render API -->
<bean id="parameterMappingInterceptor"
    class="org.springframework.web.portlet.handler.ParameterMappingInterceptor"/>
<!-- mapping based on the mode and the request parameteter -->
<bean id="portletModeParameterHandlerMapping“
    class="org.s.web.portlet.handler.PortletModeParameterHandlerMapping">
    <property name="order" value="10"/>
    <property name="interceptors">
        <list><ref bean="parameterMappingInterceptor"/></list>
    </property>
    <property name="portletModeParameterMap">
        <map><entry key="view"><map>
        <entry key="manageBeers"><ref bean="manageBeersController"/></entry>
                <entry key="editBeer" value-ref="beerFormController" />
        </map></entry></map>
    </property>
</bean>
```

Controllers

- Most controllers are similar to the Servlet MVC counterpart
  - Controller (interface)
  - BaseCommandController
  - AbstractCommandController
  - AbstractFormController
  - SimpleFormController
  - AbstractWizardFormController
- And new one (similar to ParameterizableViewController)
  - PortletModeNameViewController - determines view name based on the current portlet mode (“view”, “edit”, “help”, or some custom mode)

Controllers (continue)

- Controllers have two request handling APIs from Controller interfaces
  - void handleActionRequest (ActionRequest, ActionResponse)
  - ModelAndView handleRenderRequest (RenderRequest, RenderResponse)
- State of complex objects is shared with the PortletSession
- Use ActionResponse.setRenderParameter for simple String or String arrays

SimpleFormController

- Same workflow as Servlet counterpart.
- Follow onSubmitAction or doSubmitAction chain for db updates.
- onSubmitRender methods may be executed repeatedly by the portal

Controller Configuration Example

- There is no change in the way controllers are configured!
- Replace Servlet specific controllers with Portlet counterpart
- Modify controller code to account for Portlet API

View

- Portlet MVC reuses the View contracts from Servlet MVC
- *ViewRendererServlet* is used to convert portlet request to servlet request

```
<servlet>
    <servlet-name>ViewRendererServlet</servlet-name>
    <servlet-class>
      o.s.web.servlet.ViewRendererServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>ViewRendererServlet</servlet-name>
    <!- URL is important. Modified with viewRendererUrl on DispatcherPortlet →
    <url-pattern>/WEB-INF/servlet/view</url-pattern>
</servlet-mapping>
```

Demo

Q&A