Spring Portlet MVC

John A. Lewis

2006 Winter JA-SIG Conference
Atlanta - Dec 3-5, 2006

UNICON

Topics:

1. Introduction to Portlets
2. Overview of Spring Web MVC
3. Introduction to Spring Portlet MVC
4. Configuration of Spring Portlets
5. The Spring Portlet API
6. Integration with Spring Web Flow
7. Summary
8. Resources

UNICON

Introduction To Portlets

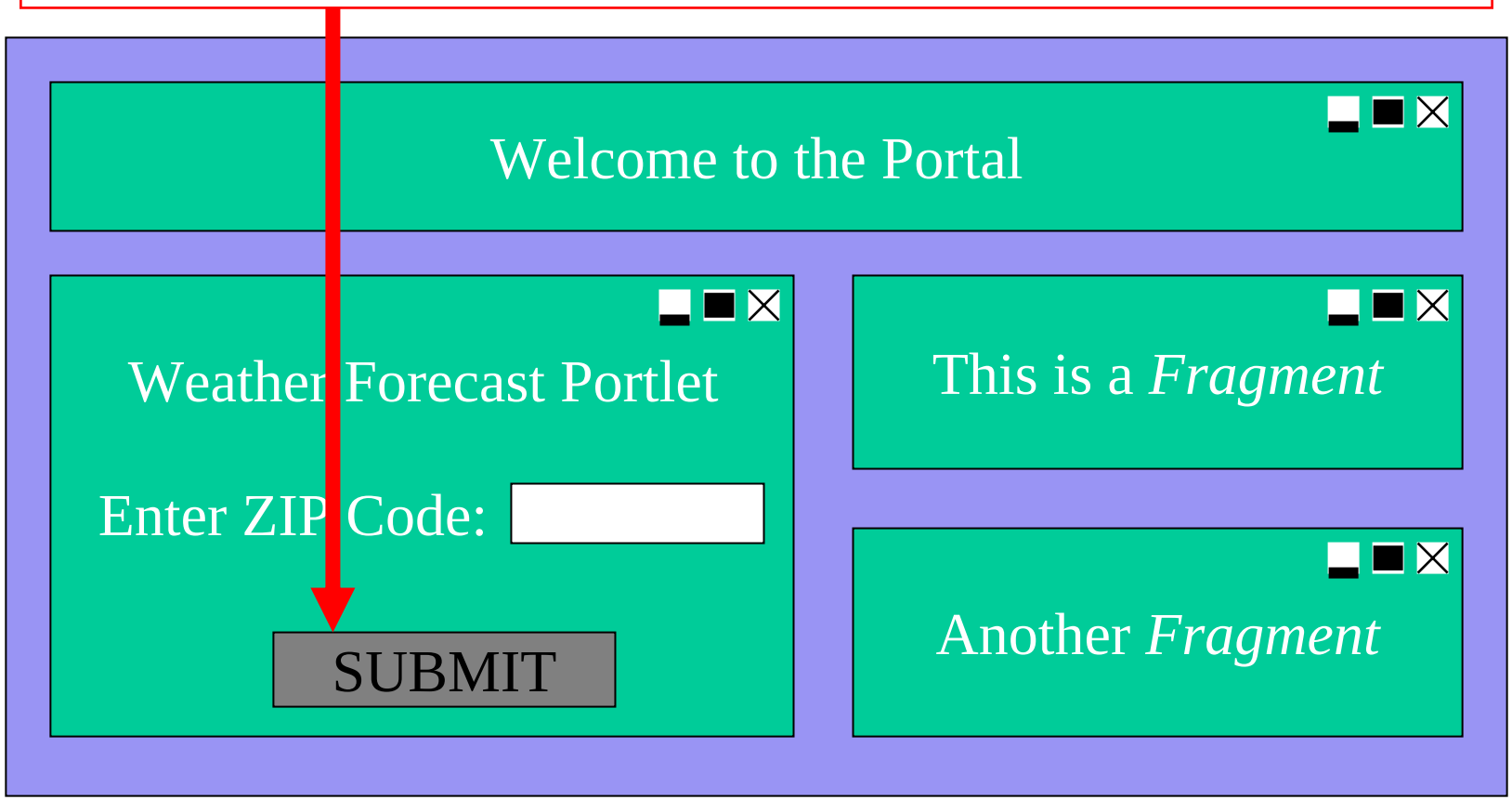
UNICON

The Portlet Specification: JSR-168

"Portlets are web components - ***like Servlets*** - specifically designed to be aggregated in the context of a ***composite page***. Usually, many Portlets are ***invoked in the single request*** of a Portal page. Each Portlet ***produces a fragment of markup*** that is combined with the markup of other Portlets, all within the Portal page markup."

Portlets within a Portal layout

When the button is pressed, an *ACTION* is handled by that Portlet only, but each of the Portlets will *RENDER*.



Portlet Modes

- View

  Render data or show a form for user interaction.

- Edit

  Modify user preferences.

- Help

  Display information to assist the user.

Window States

- Normal

  Portlets share the screen space according to the configuration of layouts in the Portal environment.

- Maximized

  Optionally display more information.

- Minimized

  Minimal or no rendering is necessary.

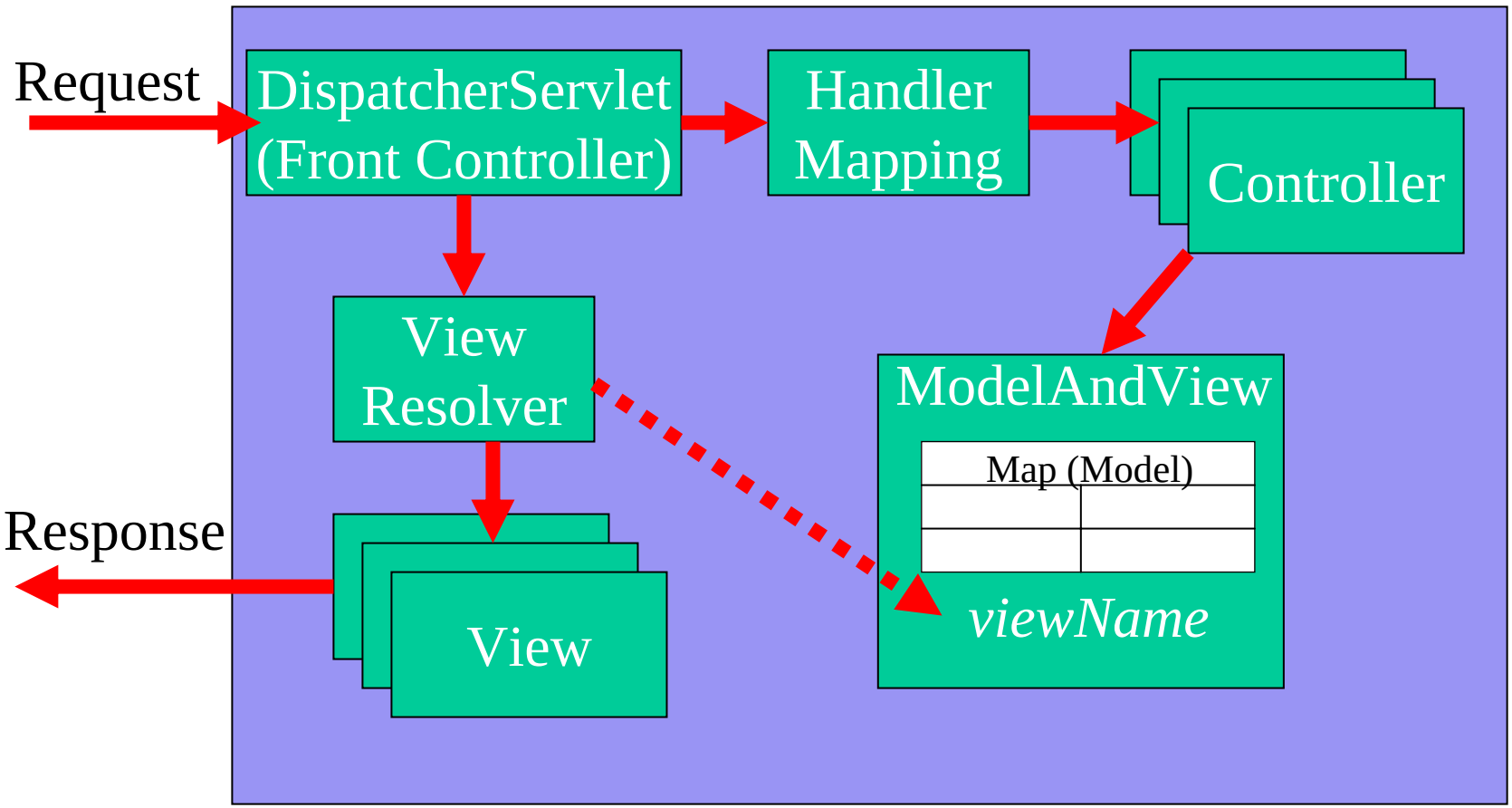
Overview of Spring Web MVC

UNICON

Spring Web MVC Basics

- Model
  - A `java.util.Map` containing domain objects
  - The *contract* between `Controller` and `View`
- View
  - Definition used to render the `Model` data
- Controller
  - Handles the Request
  - Delegates to the Service Layer
  - Prepares the `Model`
  - Chooses a *logical view name*

Spring Web MVC Architecture



Spring Web MVC Controllers

The `Controller` interface defines a handle method:

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

Implement the interface or extend a base class:

- `AbstractController`
- `MultiActionController`
- `AbstractCommandController`
- `SimpleFormController`
- … *and more*

Data Binding, Validation, and Forms

Spring Web MVC's *Command* Controllers enable:

- Powerful data-binding to graphs of domain objects
  - Using Spring's `ServletRequestDataBinder`
  - Extensible via Property Editors for converting between Strings and Objects
- Pluggable validation with a simple `Validator` interface that is *not* web-specific.

The `SimpleFormController` builds on this functionality and adds workflow (display, bind+validate, process)

Spring Web MVC Views

The `View` interface defines a method for rendering:

```
public void render(Map model,
           HttpServletRequest request,
           HttpServletResponse response)
  throws Exception;
```

Implement the interface or use one of these implementations:

- `JstlView`
- `FreeMarkerView`
- `VelocityView`
- `AbstractExcelView`
- `AbstractPdfView`
- `XsltView`
- … *and more*

Other Features of Spring Web MVC

- Handler Interceptors
  - `preHandle(request, response, handler)`
  - `postHandle(request, response, handler, modelAndView)`
  - `afterCompletion(request, response, handler, exception)`
- Handler Exception Resolvers
  - `resolveException(request, response, handler, exception)`
  - Returns a `ModelAndView`
- Multipart Resolvers
  - If a Multipart is present, wraps the request
  - Provides access to the File(s)
  - Property Editors available for binding to String or byte array

Introduction to Spring Portlet MVC

UNICON

Similarities to Web MVC

- *Mostly* parallel with Spring's Servlet-based Web MVC framework:
  - `DispatcherPortlet`
  - `HandlerMapping`
  - `HandlerInterceptor`
  - `Controller`
  - `PortletRequestDataBinder`
  - `HandlerExceptionResolver`
  - `MultipartResolver`

Differences in Portlet MVC

However, there are a few significant differences…

- 2 Phases of Request: *Action* and *Render*
  - **One** Portlet may perform an action, **All** will render
  - Instead of *`handleRequest`*`(..)` in Controllers:
    - `handleActionRequest(..)`
    - `handleRenderRequest(..)`
- To pass parameters from the action phase to the render phase call:

```
actionResponse.setRenderParameter(name, value)
```

Differences in Portlet MVC (cont)

- URL is controlled by the Portlet Container:

  "The API will provide a URL-rewriting mechanism for creating links to trigger actions within a Portlet without requiring knowledge of how URLs are structured in the particular web application."

  – *JSR-168 Specification*

- What are the implications?
  - Unable to provide meaning in the URL's path
  - Therefore no equivalent of `BeanNameUrlHandler`:

```
<bean name="/search.html" class="SearchController"/>
```

Portlet Modes, Windows States and Request Parameters are used to determine navigation instead

Configuration of Spring Portlets

UNICON

Configuring `web.xml` (1)

Set the *parent* `ApplicationContext`

- Shared by all portlets within the WebApp
- Use `ContextLoaderListener` to load the parent context

(Same as in Spring Web MVC)

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

Configuring `web.xml` (2)

Set `contextConfigLocation` parameter to list bean definition file(s) for `ContextLoaderListener`

(Again same as in Spring Web MVC)

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

Configuring web.xml (3)

Add the ViewRendererServlet:

```
<servlet>
    <servlet-name>view-servlet</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.ViewRendererServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>view-servlet</servlet-name>
    <url-pattern>/WEB-INF/servlet/view</url-pattern>
</servlet-mapping>
```

The `ViewRendererServlet`

- **`ViewRendererServlet`** acts as a bridge between a Portlet request and a Servlet request.
- It allows a Spring Portlet MVC application to leverage the full capabilities of Spring Web MVC for creating, defining, resolving, and rendering views.
- Therefore, you are able to use the same `ViewResolver` and `View` implementations.

Configuring `portlet.xml`

```
<portlet>
   <portlet-name>example</portlet-name>
   <portlet-class>
        org.springframework.web.portlet.DispatcherPortlet
   </portlet-class>
   <init-param>
      <name>contextConfigLocation</name>
      <value>/WEB-INF/context/example-portlet.xml</value>
   </init-param>
   <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>view</portlet-mode>
      <portlet-mode>edit</portlet-mode>
      <portlet-mode>help</portlet-mode>
   </supports>
   <portlet-info>
      <title>Example Portlet</title>
   </portlet-info>
</portlet>
```

A “Front Controller” for *this* Portlet

Bean definitions for *this* Portlet’s own `ApplicationContext`

The Spring Portlet API

UNICON

The `DispatcherPortlet` (1)

- *Each Portlet* will use a single `DispatcherPortlet`.
- It will play a *Front Controller* role as with Spring MVC's `DispatcherServlet`.
- The portlet-specific bean definitions to be used by the `DispatcherPortlet` should be specified in an individual application context file *per Portlet*.
- Bean definitions that are shared between Portlets or with other Servlets, etc. should be in the parent application context file.

The `DispatcherPortlet` (2)

- The `DispatcherPortlet` uses `HandlerMappings` to determine which `Controller` should handle each `PortletRequest`.

- The DispatcherPortlet automatically detects certain bean definitions, such as the HandlerMappings, HandlerExceptionResolvers, and MultipartResolvers.

Handler Mappings

- `PortletModeHandlerMapping`
  - Map to a Controller based on current PortletMode
- `ParameterHandlerMapping`
  - Map to a Controller based on a Parameter value
- `PortletModeParameterHandlerMapping`
  - Map to a Controller based on current PortletMode *and* a Parameter value
- Or create your own custom `HandlerMapping` …

PortletModeHandlerMapping

```
<bean id="portletModeHandlerMapping"
   class="org.springframework.web.portlet.handler.
   PortletModeHandlerMapping">
    <property name="portletModeMap">
       <map>
          <entry key="view" value-ref="viewController"/>
          <entry key="edit" value-ref="editController"/>
          <entry key="help" value-ref="helpController"/>
       </map>
    </property>
</bean>
<bean id="viewController" class="ViewController"/>
...
```

ParameterHandlerMapping

```
<bean id="handlerMapping“
 class="org.springframework.web.portlet.handler.
   ParameterHandlerMapping">
   <property name="parameterMap">
      <map>
         <entry key="add" value-ref="addHandler"/>
         <entry key="remove" value-ref="removeHandler"/>
      </map>
   </property>
</bean>
```

(can optionally set the `parameterName` property – the default value is ‘action’)

PortletModeParameterHandlerMapping

```
<bean id="handlerMapping"
    class="…PortletModeParameterHandlerMapping">
 <property name="portletModeParameterMap">
  <map>
   <entry key="view">
    <map>
     <entry key="add" value-ref="addHandler"/>
     <entry key="remove" value-ref="removeHandler"/>
    </map>
   </entry>
   <entry key="edit">
    <map><entry key="prefs” value-ref="prefsHandler"/></map>
   </entry>
  </map>
 </property>
</bean>
```

More on `HandlerMappings` (1)

- As with Spring's Servlet-based Web MVC framework, a `DispatcherPortlet` can use multiple `HandlerMappings`.
- The *`order`* property can be set to create a chain, and the first mapping to find a handler wins.
- For example, you can use a `PortletModeParameterHandlerMapping` to detect an optional parameter followed by a `PortletModeHandlerMapping` with default handlers for each mode.

More on HandlerMappings (2)

Interceptors can be assigned to the HandlerMapping in the same way as Spring Web MVC:

```
<property name="interceptors">
   <list>
      <ref bean="someInterceptor"/>
      <ref bean="anotherInterceptor"/>
   </list>
</property>
```

More on `HandlerMappings` (3)

- For an Action Request, the handler mapping will be consulted **twice** – once for the *action phase* and again for the *render phase*.
- During the action phase, you can manipulate the criteria used for mapping (such as a request parameter).
- This can result in the render phase getting mapped to a **different Controller** – a great technique since there is no portlet redirect.

HandlerInterceptor

```
public interface HandlerInterceptor {

    boolean preHandle(PortletRequest request,
                      PortletResponse response,
                      Object handler) throws Exception;

    void postHandle(RenderRequest request,
                    RenderResponse response,
                    Object handler,
                    ModelAndView mav) throws Exception;

    void afterCompletion(PortletRequest request,
                         PortletResponse response,
                         Object handler,
                         Exception ex) throws Exception;
}
```

The Controllers

- `Controller` (The Interface)
- `AbstractController`
- `SimpleFormController`
- `PortletWrappingController`
- `PortletModeNameViewController`
- Several others!

The Controller Interface

```
public interface Controller {

    ModelAndView handleRenderRequest (
        RenderRequest request,
        RenderResponse response)
        throws Exception;

    void handleActionRequest (
        ActionRequest request,
        ActionResponse response)
        throws Exception;
}
```

AbstractController

An example of the *Template Method* pattern

Implement one or both of:

- `handleActionRequestInternal(..)`
- `handleRenderRequestInternal(..)`

Provides common properties (with defaults):

- `requiresSession` (false)
- `cacheSeconds` (-1, uses container settings)
- `renderWhenMinimized` (false)

SimpleFormController (1)

- Very similar to its Spring Web MVC counterpart.
- Handles the form workflow including display of the *formView*, binding and validation of submitted data, and a chain of methods for handling a successfully validated form submission.
- Due to the two phases of a portlet request, the *onSubmit(..)* methods each have two versions: *onSubmitAction(..)* and *onSubmitRender(..)*.
- In most cases, the default *onSubmitRender(..)* will be sufficient as it simply renders the configured *successView*.
- By defining the command class, a form view and a success view, no code is required except to customize behavior

`SimpleFormController` (2)

Some methods for controlling the form:

- `formBackingObject(..)` – the default implementation simply creates a new instance of the *command* Class
- `initBinder(..)` – register custom property editors
- `referenceData(..)` – provide additional data to the model for use in the form
- `showForm(..)` – the default implementation renders the *formView*

SimpleFormController (3)

Some methods for controlling processing of the form submission:

- `onBind(..)` & `onBindAndValidate(..)` – callback for post-processing after binding and validating
- `onSubmitAction(..)` & `onSubmitRender(..)` – callbacks for successful submit with no binding or validation errors

Several others, including ones inherited from AbstractFormController, BaseCommandController

`PortletWrappingController` (1)

A `Controller` implementation for managing a JSR-168 compliant Portlet's lifecycle within a Spring environment.

Example Uses:

- Apply `Interceptors` to the wrapped `Portlet`
- Use dependency injection for init-parameters

PortletWrappingController (2)

```
<bean id="wrappedPortlet"
  class="org.springframework.web.portlet.mvc.
  PortletWrappingController">
   <property name="portletClass"
  value="xyz.SomePortlet"/>
   <property name="useSharedPortletConfig"
  value="false"/>
   <property name="portletName" value="wrapped-portlet"/>
   <property name="initParameters">
      <props>
         <prop key="someParam">some value</prop>
      </props>
   </property>
</bean>
```

`PortletModeNameViewController` (1)

- This `Controller` simply returns the current `PortletMode` as the view name so that a view can be resolved and rendered.
- Example: *PortletMode.HELP* would result in a *viewName* of "help" and an `InternalResourceViewResolver` may use /WEB-INF/jsp/help.jsp as the View.
- This means you can use JSP in a portlet with no Java classes to write at all!

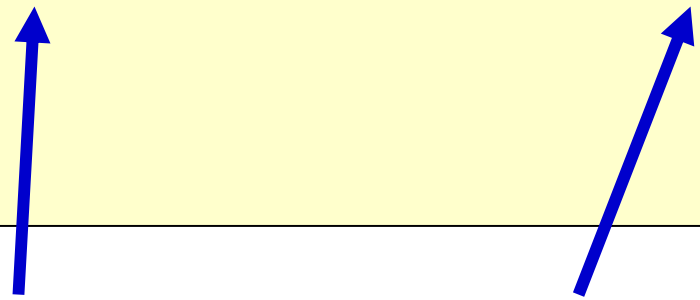
PortletModeNameViewController (2)

```
<bean id="modeNameViewController"
  class="org.springframework.web.portlet.mvc.
PortletModeNameViewController"/>


<bean id="viewResolver"
  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>
```

Resolving Exceptions

```
<bean id="exceptionResolver"
   class="org.springframework.web.portlet.handler.
   SimpleMappingExceptionResolver">
  <property name="defaultErrorView" value="error"/>
  <property name="exceptionMappings">
    <value>
        javax.portlet.PortletSecurityException=unauthorized
        javax.portlet.UnavailableException=unavailable
    </value>
  </property>
</bean>
```



Map Exceptions to viewNames

Handling File Uploads (1)

- Just specify a `MultipartResolver` bean
- `DispatcherPortlet` will automatically detect it

```
<bean id="portletMultipartResolver"
   class="org.springframework.web.portlet.multipart.
CommonsPortletMultipartResolver">
    <property name="maxUploadSize“ value=“2048”/>
</bean>
```

Handling File Uploads (2)

If a multipart file is detected, the portlet request will be wrapped:

```
public void onSubmitAction(ActionRequest request,
        ActionResponse response, Object command,
        BindException errors) throws Exception {

        if (request instanceof MultipartActionRequest) {

                MultipartActionRequest multipartRequest =
                        (MultipartActionRequest) request;

                MultipartFile multipartFile =
                        multipartRequest.getFile("file");

                byte[] fileBytes = multipartFile.getBytes();

                ...
}
```

Handling File Uploads (3)

- Spring also provides 2 `PropertyEditors` for working with `MultipartFiles`:
  - `ByteArrayMultipartFileEditor`
  - `StringMultipartFileEditor`
- These allow multipart content to be directly bound to `ByteArray` or `String` attributes of a command Class in `SimpleFormController` or `AbstractFormController`

Integration with Spring Web Flow

UNICON

Introduction to `PortletFlowController`

- The `PortletFlowController` is a Spring Web Flow Front Controller for use within a Portlet environment.
- Portlet requests (in view mode) can be mapped to the `PortletFlowController` to create or participate in an existing Flow execution.
- Flow definitions are not tied in any way to the Portlet environment. They can be reused in any supported environment - such as Spring Web MVC, Struts, or JSF.

Configuring PortletFlowController

```
<bean id="portletModeControllerMapping"
    class="org.springframework.web.portlet.handler.
PortletModeHandlerMapping">
  <property name="portletModeMap">
    <map>
      <entry key="view" value-ref="flowController"/>
    </map>
  </property>
</bean>
<bean id="flowController"
        class="org.springframework.webflow.executor.mvc.
PortletFlowController">
  <property name="flowExecutor" ref="flowExecutor"/>
  <property name="defaultFlowId" value="search-flow"/>
</bean>
```

Summary

UNICON

Summary (1)

- As much as possible, Spring's Portlet support mirrors the Servlet-based Spring Web MVC framework.
- The most significant differences result from the two-phase nature of Portlet requests.
- The handler mapping is also quite different, because the Portlet container has complete control over the formation of and meaning associated with URLs.

Summary (2)

- The actual view rendering is delegated to the Spring MVC `ViewResolver` and View implementations via the `ViewRendererServlet` which acts as a bridge from Portlet requests to Servlet requests.
- Several Controller base classes are provided - mostly parallel to Spring MVC.
- There are also some Portlet-specific Controllers such as `PortletModeNameViewController` and `PortletWrappingController`

Summary (3)

- Because they are so similar, porting code between Spring Web MVC and Spring Portlet MVC is pretty simple.
- Spring Portlet MVC preserves the dual phases of portlet requests -- one of the real strengths of the JSR-168 spec (example: dynamic search results)
  - Most other portlet MVC frameworks hide the phases (such as Apache Portal Bridges) – losing this key feature

Resources

UNICON

Resources

- Spring Framework Reference Manual
  - Chapter 16: Portlet MVC Framework
  - http://static.springframework.org/spring/docs/2.0.x/reference/portlet.html
- Spring Framework Java Docs
  - Package org.springframework.web.portlet
  - http://static.springframework.org/spring/docs/2.0.x/api/index.html
- Spring Portlet MVC Wiki Site
  - News, Downloads, Sample Apps, FAQs, etc.
  - http://opensource.atlassian.com/confluence/spring/display/JSR168/

Questions?

John A. Lewis

jlewis@unicon.net

www.unicon.net