Spring Portlet MVC Seminar

John A. Lewis
Cris J. Holdorph
Unicon, Inc.

JA-SIG Spring 2008 Conference
27 April 2008

© Copyright Unicon, Inc., 2008. Some rights reserved. This work is licensed under a
Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License.
To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/us/

Agenda

- Introduction (Morning)
  - Portlet & Spring Review
  - The Spring MVC Framework
  - Configuring Spring Portlet MVC
  - Building & Deploying the Samples
  - Handler Mapping
  - View Resolver & Exception Resolver
  - Controllers & AbstractController

Agenda

- Advanced (Afternoon)
  - Handler Interceptors
  - Form Controllers
  - File Uploads
  - Redirects
  - Annotation-based Dispatching
  - Spring Security for Portlets

Portlet Review

A quick refresher

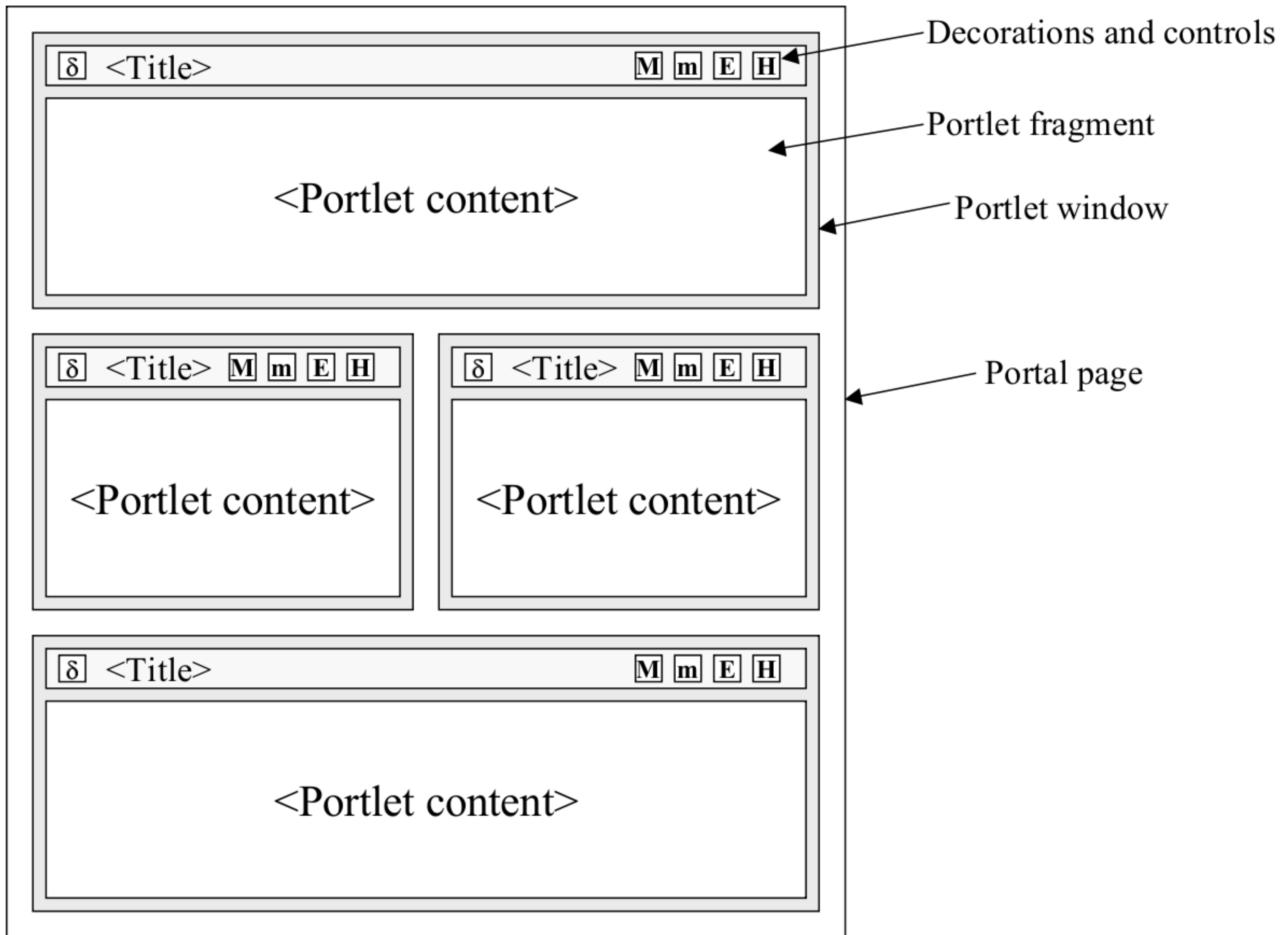


Diagram from Java™ Portlet Specification, Version 2.0 Public Draft

Java Portlet Standards

- **Java Portlet 1.0 Specification (JSR 168)**
  - **Started**: 29 January 2002
  - **Released**: 27 October 2003
  - **Reference Implementation**: Apache Pluto
  - Linked to WSRP 1.0 Specification
- **Java Portlet 2.0 Specification (JSR 286)**
  - **Started**: 29 November 2005
  - **Final Approval Ballot**: 3 March 2008 (Passed)
  - Waiting for Reference Implementation (Pluto 2.0)
  - Linked to WSRP 2.0 Specification

Portlets and Servlets

- Portlets and Servlets closely related, but no direct connection
- Portlets run in Portlet Container
- Portlet Container is an extension of a Servlet Container
- Portlet Application is an extension of a Web Application
  - `web.xml` & `portlet.xml` Deployment Descriptors
- Can have Portlets and Servlets together in the same Web App

Multiple Request Phases

- Action Requests
  - Executed only once
  - Used to change system state (e.g. form post)
  - No markup produced
- Render Requests
  - Executed at least once
  - May be executed repeated
  - Produces the fragment markup
  - Results can be cached
- Portlet 2.0 adds Event Requests and Resource Requests (now we have four!)

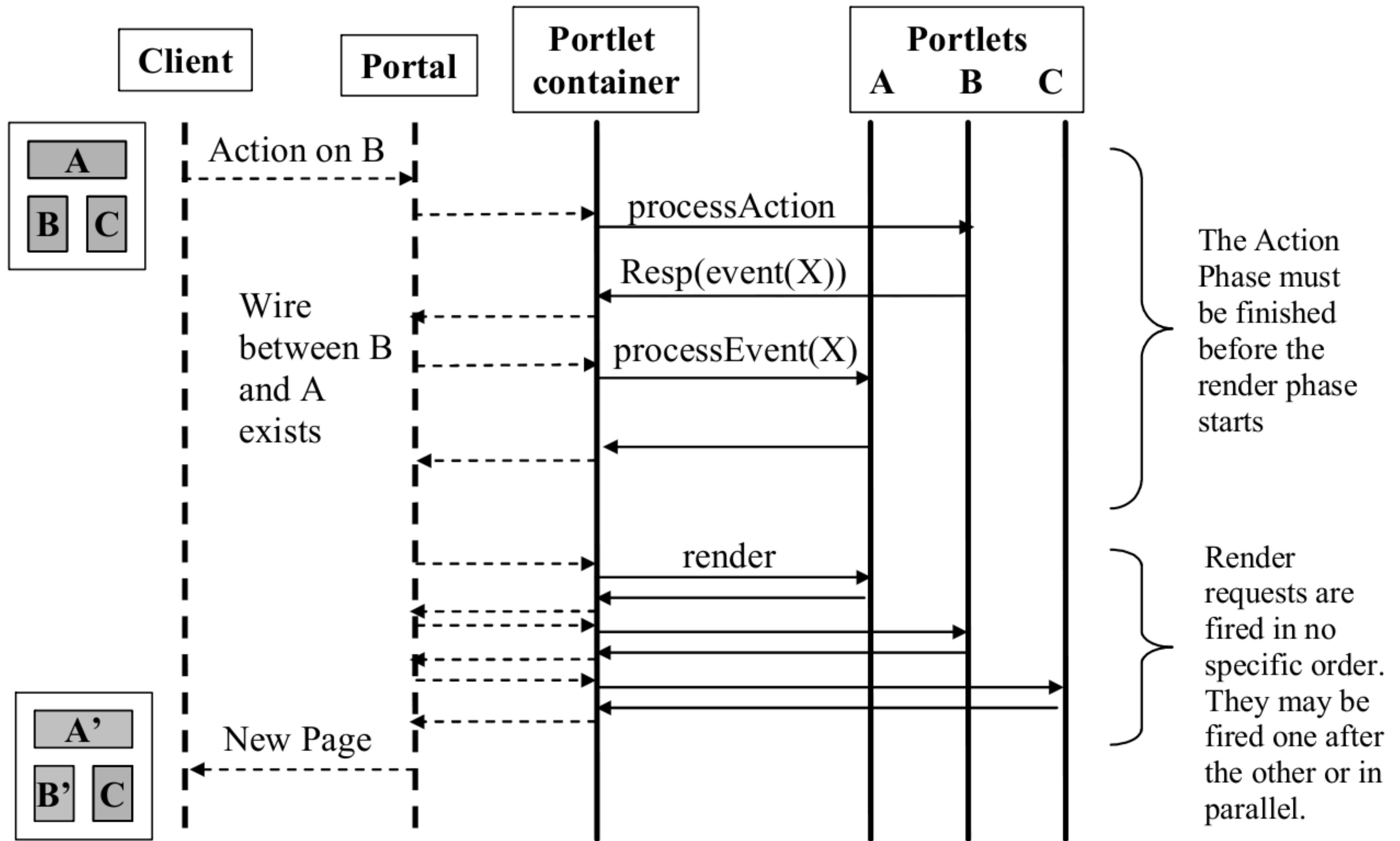


Diagram from Java™ Portlet Specification, Version 2.0 Public Draft

Portlet Modes

- Control state of portlet from portal-provided navigation controls
- Three standard modes:
  - **VIEW**: Normal display of Portlet
  - **EDIT**: Configure the Portlet (e.g. Preferences)
  - **HELP**: Show documentation about Portlet
- Portals can have additional custom modes (several suggested modes in the specs)
- Portlets can change their own mode

Portlet Window States

- Control level of detail of portlet from portal-provided navigation controls
- Three standard window states:
  - **NORMAL**: Standard view, probably combined with a number of other portlets in the page
  - **MAXIMIZED**: Largest view, likely the only portlet in the page or at least the primary one
  - **MINIMIZED**: Smallest view, either no content at all or a very small representation
- Portals can have additional custom states
- Portlets can change their own window state

Portlet URL Handling

- Portals are in control of actual URLs
- Portlets must use specific APIs for generating URLs and setting parameters
- Multiple types of URLs corresponding to request types (Action and Render)
- Must treat URLs as opaque Objects – don't think of them as Strings
- No concept of “path” for the portlet – must use Portlet Mode, Window State, and Request Parameters for navigation

Spring Review

Another quick refresher

What Is Spring?

- "Full-stack Java/JEE application framework"
- Lightweight
  - Born out of frustration with EJB
- Core focus is on Inversion of Control (IoC)
  - aka Dependency Injection (DI)
- Builds on top of core container to provide all needed application components / services

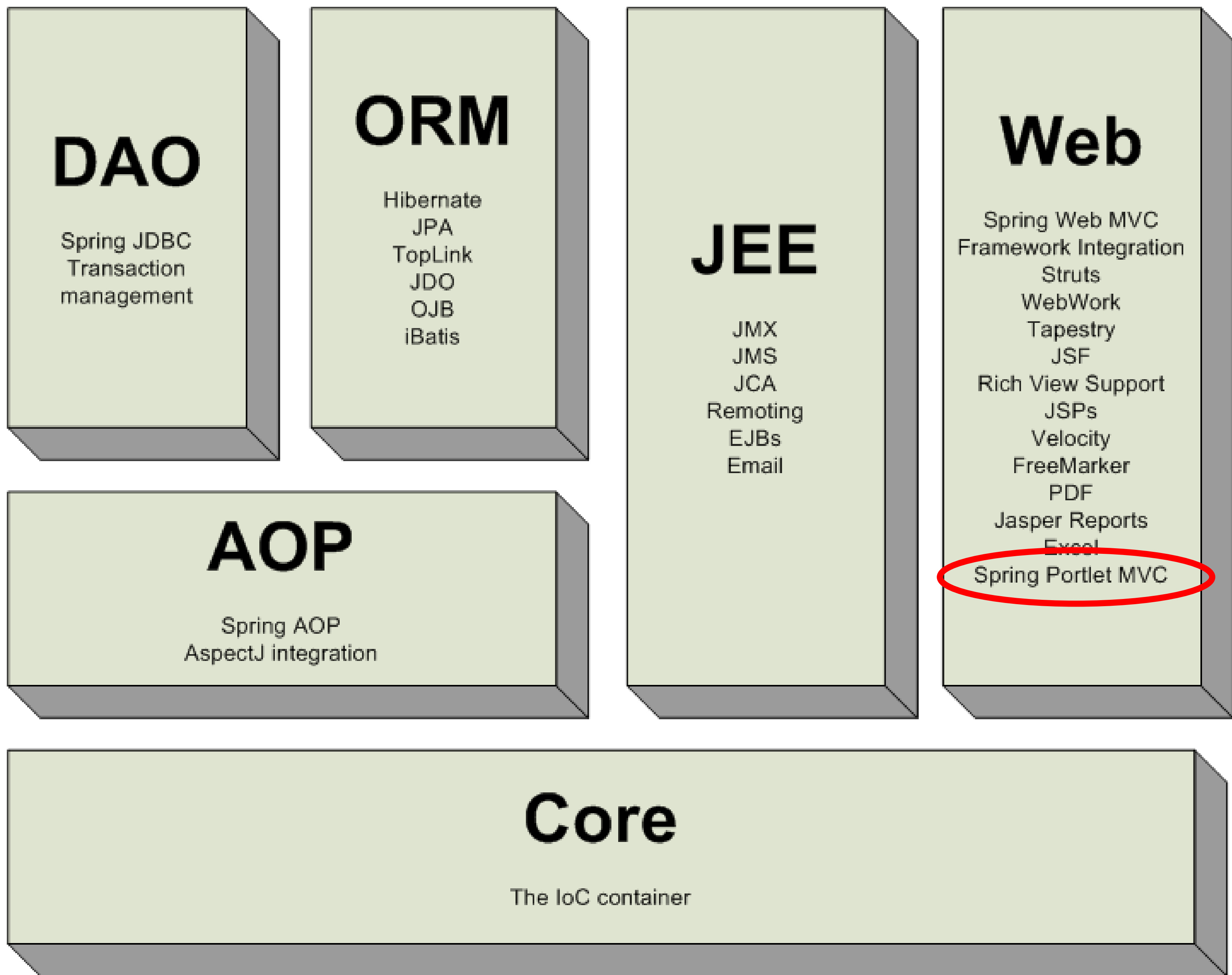


Diagram from Spring Framework Reference Documentation

Dependency Injection

- The core of Spring is based on techniques to externalize the creation and management of component dependencies
- This Inversion of Control principle has been defined as Dependency Injection

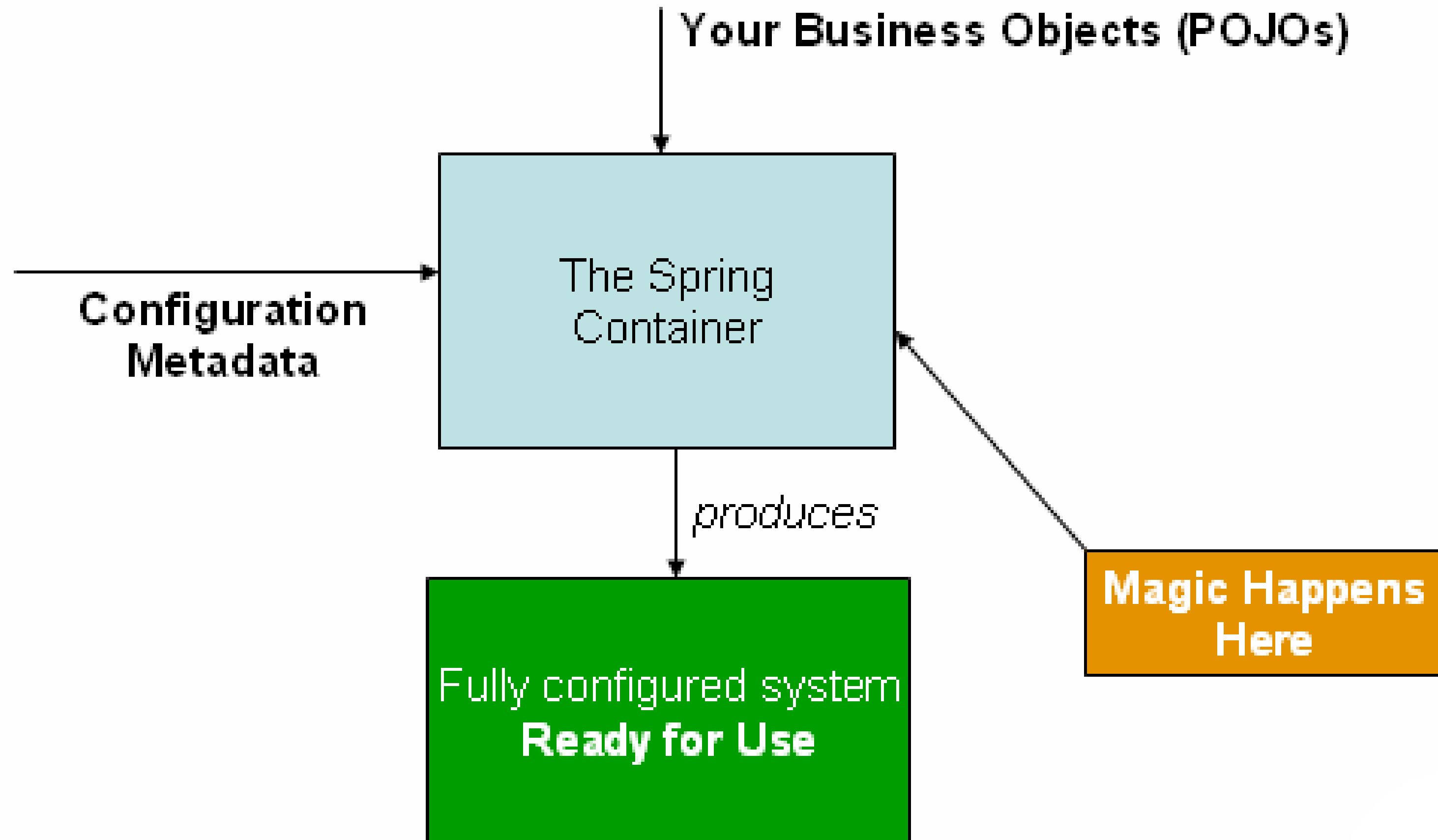


Diagram from Spring Framework Reference Documentation

Spring Beans

- The central part of Spring's IoC container is the BeanFactory / ApplicationContext
- Responsible for managing components and their dependencies
- In Spring the term "Bean" is used to refer to any component managed by the container
- The term "Bean" implies some conformance to the JavaBean standard
- The BeanFactory / ApplicationContext is typically configured with a configuration file

Spring Bean Definition

- Using XML Schema Definitions:

```
<beans
    xmlns="http://www.springframework.org/schema/beans"...>

    <bean id="logger"
        class="com.logging.StandardOutLogger"/>

    <bean id="doIt" class="com.do.DoIt">
        <property name="log">
            <ref local="logger"/>
        </property>
    </bean>

</beans>
```

The Spring MVC Framework

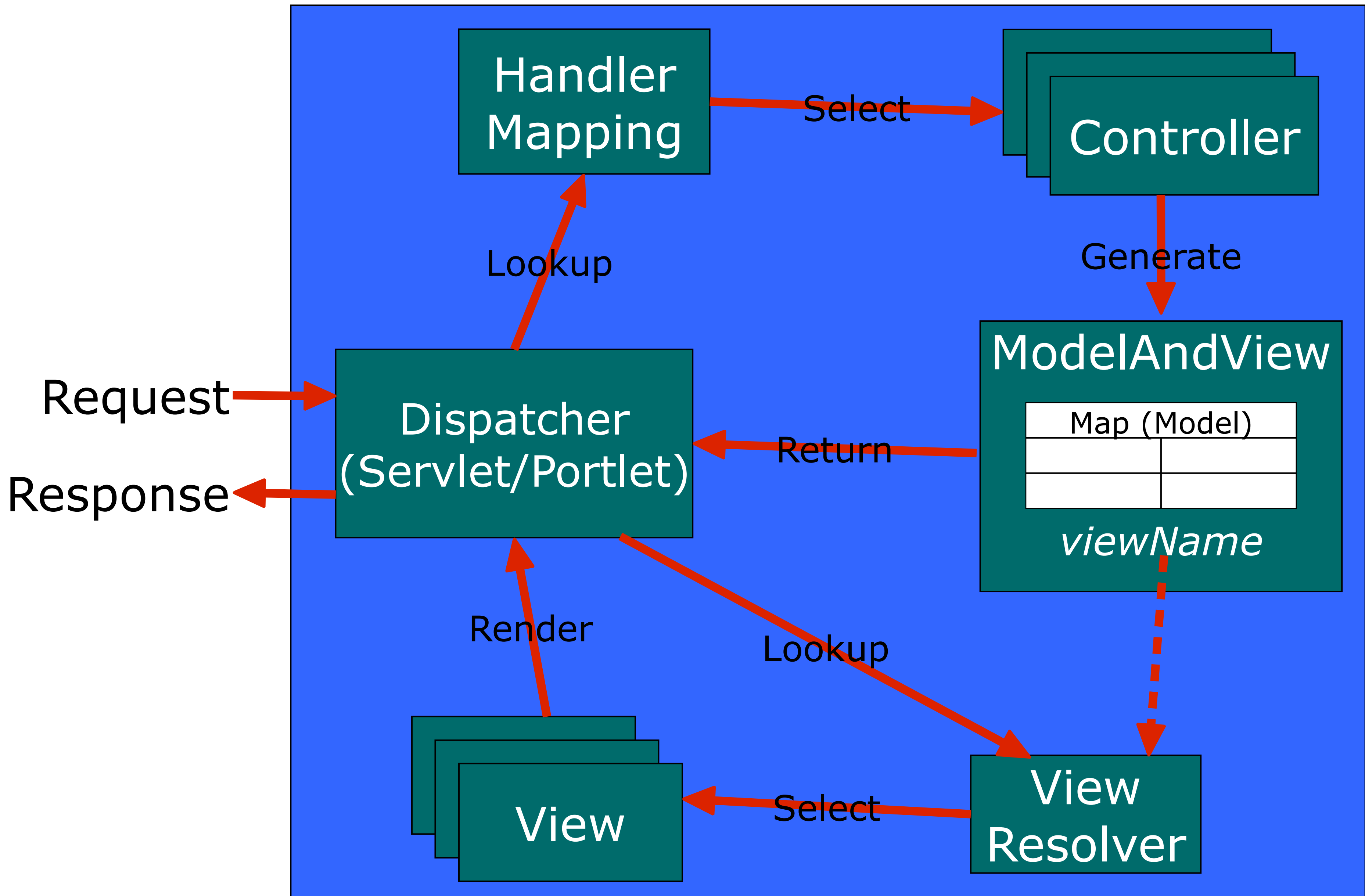
The Spring module for building web application

Using A Framework

- Why use a framework to write Portlets?
- Do you write Servlets from scratch?
  Why not?
- Frameworks take care of infrastructure and let you focus on *your unique functionality*
- Coding Portlets from scratch is of course an option, but let's go to the frameworks...

Spring MVC

- Flexible and Lightweight
- Request-Oriented Web Framework
- Implements Classic MVC Pattern
  - **Model**
    - Information to be presented
    - Contract between Controller and View
  - **View**
    - User interface definition
    - Used to render the Model for display
  - **Controller**
    - Handles the request and assembles the Model
    - Delegates to service layer for business logic


Handler Mapping
Select
Controller
Lookup
Generate
ModelAndView
Map (Model)
viewName
Request
Dispatcher (Servlet/Portlet)
Return
Response
Render
Lookup
View
Select
View Resolver

Spring Views

- Includes support for numerous common view technologies:
  - JSP & JSTL, XSLT, Velocity, FreeMarker, Tiles, PDF Docs, Excel Docs, JasperReports
- Easy to implement new View technologies
- View technology all usable in both Servlets and Portlets
  - Although only ones capable of producing HTML markup fragments generally useful in Portlets
- JSP & JSTL is the most common View technology for Portlets

Spring Controllers

- Basic interfaces handle requests and potentially return a *ModelAndView*
- Many useful abstract classes for common Controller patterns
- All easily extensible for your custom handling
- (Stay tuned for information about Annotation-based Controllers in Spring 2.5)

Other MVC Features

- Interceptors for wrapping other concerns around Controller execution
- Exception Resolvers to catch Exceptions coming out of Controllers and mapping to appropriate Views
- Data Binding to take request properties and bind them directly to Domain Objects
- Data Validation to test validity of bound Domain Objects
- Multipart Handling to bind file uploads

Spring Web MVC Resources

- Spring Framework Reference Manual
  Chapter 13: Web MVC Framework
  http://static.springframework.org/spring/docs/2.5.x/reference/mvc.html
- Spring Framework Java Docs
  Package org.springframework.web
  http://static.springframework.org/spring/docs/2.5.x/api/
- Expert Spring MVC and Web Flow
  Apress book by Seth Ladd
  http://www.springframework.org/node/235
- Community Support Forums
  http://forum.springframework.org/
- Spring MVC Step-By-Step Tutorial
  http://www.springframework.org/docs/Spring-MVC-step-by-step/

Spring Portlet MVC Resources

- Spring Framework Reference Manual

  Chapter 16: Portlet MVC Framework

  http://static.springframework.org/spring/docs/2.5.x/reference/portlet.html

- Spring Framework Java Docs

  Package org.springframework.web.portlet

  http://static.springframework.org/spring/docs/2.5.x/api/

- Spring Portlet Wiki Site

  News, Downloads, Sample Apps, FAQs, etc.

  http://opensource.atlassian.com/confluence/spring/display/JSR168/

- Community Support Forums

  http://forum.springframework.org/

Configuring Spring Portlet MVC

What you do to your web application

web.xml: ContextLoaderLister

- Load the parent *ApplicationContext* with *ContextLoaderListener* in *web.xml*
- Shared by all Portlets (and Servlets) within the Web Application / Portlet Application

```
<listener>

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

</listener>
```

No different from Servlet Spring Web MVC

web.xml: contextConfigLocation

- Also in *web.xml*, set *contextConfigLocation* parameter to list bean definition file(s) for *ContextLoaderListener*

```
<context-param>

   <param-name>contextConfigLocation</param-name>

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

</context-param>
```

No different from Servlet Spring Web MVC

web.xml: ViewRendererServlet

- Add the *ViewRendererServlet* to *web.xml*:

```
<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>
```

ViewRendererServlet

- *DispatcherPortlet* uses this to dispatch the actual view rendering into a Servlet context
- Acts as a bridge between a Portlet request and a Servlet request
- Allows Portlet application to leverage the full capabilities of Spring MVC for creating, defining, resolving, and rendering views
- Therefore, you are able to use the same *ViewResolver* and *View* implementations for both Portlets and Servlets

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>
```

DispatcherPortlet

- Front controller for each Portlet
- Portlet-specific bean definitions specified in an individual application context
- Bean definitions shared between Portlets or with other Servlets go in the parent application context
- Uses *HandlerMappings* to determine which *Controller* should handle each request
- Autodetects certain bean definitions:
  - *HandlerMappings*
  - *HandlerExceptionResolvers*
  - *MultipartResolvers*

Building & Deploying the Samples

Let's get our hands dirty

First Code Sample

- Things we need to do:
  - Get development environment installed
  - Setup Pluto as a Server in Eclipse & start it
  - Import 'sample1' application into Eclipse
  - Create Maven task to build & deploy
  - Build & deploy the sample application
  - Verify that it works in Pluto

    http://localhost:8080/pluto/portal
  - Explore the *web.xml* and *portlet.xml* files

Handler Mapping

Where should the request go?

HandlerMapping

- *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*

Most of these become obsolete in Spring 2.5 because the Annotation-based Mapping is now preferred

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

- Can optionally set the *parameterName* property – the default value is ‘action’

```
<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>
```

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 *HandlerMapping*

- Can use multiple *HandlerMappings*, controlled by the *order* property to set the chain
- Can apply *HandlerInterceptors* to requests by including them in the mapping definition – very useful since Portlets don't have Filters

Mapping and Portlet Lifecycle

- 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

Handler Mapping Sample

Let's go look at 'sample1' again:

- *src/main/webapp/*
  - WEB-INF/context/portlet/mode.xml
  - WEB-INF/jsp/include.jsp
  - WEB-INF/jsp/view.jsp

View Resolver & Exception Resolver

Finding view definitions and dealing with exceptions

Resolving Views

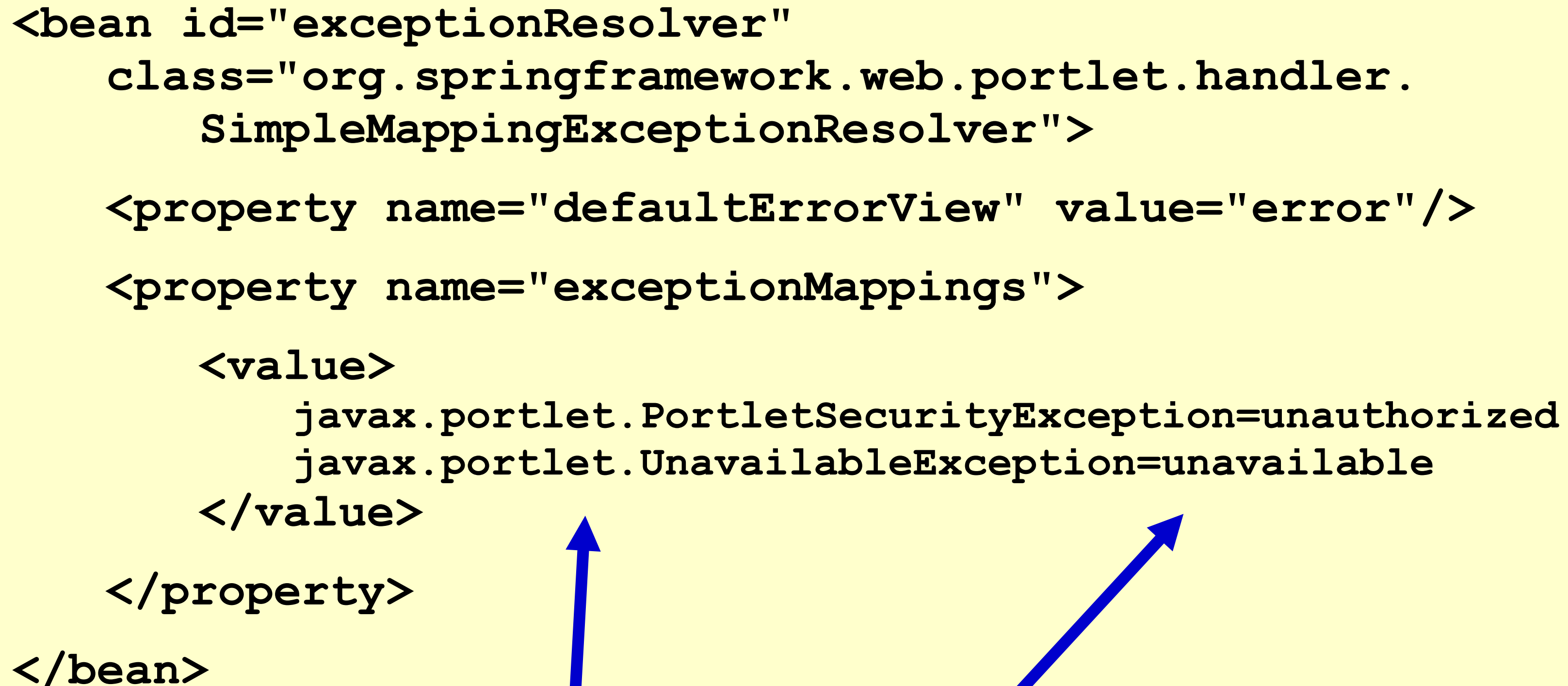
- Instead of building Views ourselves, refer to them by name and have them loaded for us

```
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.
    InternalResourceViewResolver">

    <property name="cache" value="false" />

    <property name="viewClass"
    value="org.springframework.web.servlet.view.JstlView" />

    <property name="prefix" value="/WEB-INF/jsp/" />

    <property name="suffix" value=".jsp" />

</bean>
```

Can be shared between multiple portlets & servlets

Resolving Exceptions

- Manage Exceptions escaping out of Handlers



```
<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 **View Names** (used by View Resolver)

More on Resolvers

- Can use multiple *ViewResolver*s and *ExceptionResolver*s
- *DispatcherPortlet* finds them by type, so the name of the beans doesn't matter
- Priority is controlled by the *Ordered* interface and the 'order' property

Resolver Sample

Let's go look at 'sample1' again:

- *src/main/webapp/*
  - WEB-INF/context/applicationContext.xml
  - WEB-INF/context/portlet/mode.xml

Internationalization & Localization

Reaching a wider audience

Internationalization

- Put all user visible text into a properties file:

```
button.home = Home
button.edit = Edit
button.next = Next
button.previous = Previous
button.finish = Finish
button.cancel = Cancel

exception.notAuthorized.title = Access Not Permitted
exception.notAuthorized.message = You do not have
permission to access this area.
```

MessageSource

- Define a Spring MessageSource that uses Resource Bundles by giving it the basename of the bundle (i.e. the basename of your properties file):

```
<bean id="messageSource"
    class="org.springframework.context.support.
        ResourceBundleMessageSource">
    <property name="basenames">
        <list>
            <value>messages</value>
        </list>
    </property>
</bean>
```

Using Messages in Views

- In your Views, use the appropriate mechanism to retrieve the messages from the MessageSource by their identifier.
- JSP example:

```
<%@ taglib prefix="spring"
   uri="http://www.springframework.org/tags" %>

...

<p><spring:message code="exception.contactAdmin"/></p>
```

Localization

- After creating the default file (e.g. “messages.properties”), create files for each supported Locale and translate contents accordingly:
  - messages_de.properties (German)
  - messages_fr.properties (French)
  - messages_fr_CA.properties (French – Canadian)
  - ...

I18n & L10n Sample

Let's move on to 'sample2':

- *src/main/webapp/*
  - WEB-INF/context/applicationContext.xml
  - WEB-INF/jsp/myBooks.jsp
- *src/main/resources/*
  - messages.properties
  - messages_de.properties

Controllers & AbstractController

Where to put the logic

Controllers

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

Most of these become obsolete in Spring 2.5 because the Annotation-based Mapping is now preferred

The Controller Interface

```
public interface Controller {
    ModelAndView handleRenderRequest (
        RenderRequest request, RenderResponse response)
        throws Exception;
    void handleActionRequest (
        ActionRequest request, ActionResponse response)
        throws Exception;
}
```

PortletModeNameViewController

- 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 *InternalResourceViewResolver* can 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!

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)

AbstractController Sample

Let's go look at 'sample2' again:

- *src/main/webapp/*
  - WEB-INF/context/portlet/myBooks.xml
  - WEB-INF/jsp/myBooks.jsp
  - WEB-INF/jsp/myBooksEdit.jsp
- *src/main/java/*
  - sample/portlet/MyBooksController.java
  - sample/portlet/MyBooksControllerEdit.java

Handler Interceptors

Pre/post processing
the requests and responses

HandlerInterceptor

- *HandlerInterceptor* opportunity to pre-process and post-process the request and response as it flows through the *HandlerMapping*
- Critical for portlets since we don't have Portlet Filters in JSR 168
- Can also use any *WebRequestInterceptor* in Spring (shared between Portlet and Servlet)
  - Includes “Open Session In View” Interceptors for JPA, JDO, and Hibernate so that you can lazily access persistent objects during view rendering

HandlerInterceptor Interface

```
public interface HandlerInterceptor {

    boolean preHandleAction(
        ActionRequest request, ActionResponse response,
        Object handler) throws Exception;

    void afterActionCompletion(
        ActionRequest request, ActionResponse response,
        Object handler, Exception ex) throws Exception;

    boolean preHandleRender(
        RenderRequest request, RenderResponse response,
    Object handler) throws Exception;

    void postHandleRender(
        RenderRequest request, RenderResponse response,
        Object handler, ModelAndView modelAndView) throws Exception;

    void afterRenderCompletion(
        RenderRequest request, RenderResponse response,
        Object handler, Exception ex) throws Exception;

}
```

Useful Portlet Interceptors

- *ParameterMappingInterceptor* – Used to forward a request parameter from the Action request to the Render request – helps w/ HandlerMapping based on request params
- *UserRoleAuthorizationInterceptor* – Simple security mechanism to enforce roles from PortletRequest.isUserInRole

Configuring Interceptors

```
<bean id="parameterMappingInterceptor"
   class="org.springframework.web.portlet.handler.
   ParameterMappingInterceptor" />

<bean id="portletModeParameterHandlerMapping"
   class="org.springframework.web.portlet.handler.
   PortletModeParameterHandlerMapping">

   <property name="interceptors">
      <list>
         <ref bean="parameterMappingInterceptor"/>
      </list>
   </property>

   <property name="portletModeParameterMap">
      ...
   </property>

</bean>
```

HandlerInterceptor Sample

Let's move on to 'sample3':

- *src/main/webapp/*
  - WEB-INF/context/portlet/books.xml

Form Controllers

Getting some input from the user

Command Controllers

- All start with *BaseCommandController*
- Powerful data-binding to graphs of domain objects
  - Uses *PortletRequestDataBinder*
  - Extensible via Property Editors for converting between Strings and Objects
- Pluggable validation with a simple *Validator* interface that is not web-specific
- The Form Controllers build on this functionality and add workflow (display, bind+validate, process)

SimpleFormController

- Handles form processing workflow:
  - display of the formView
  - binding and validation of submitted data
  - handle successfully validated form submission
- By defining the command class, a form view and a success view, no code is required except to customize behavior

SimpleFormController Form

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 Submit

Some methods for controlling processing of the form submission:

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

Several others, including ones inherited from AbstractFormController and from BaseCommandController

SimpleFormController Sample

Let's look at 'sample3' again:

- *src/main/webapp/*
  - WEB-INF/context/portlet/books.xml
  - WEB-INF/jsp/bookEdit.jsp
- *src/main/java/*
  - sample/portlet/BookEditController.java

AbstractWizardFormController

- Wizard-style workflow with multiple form views and multiple actions:
  - **finish**: trying to leave the wizard performing final action – must pass complete validation
  - **cancel**: leave the wizard without performing final action – ignore validity of current state
  - **page change**: show another wizard page (next, previous, etc.)
- Specify action via submit parameter names (e.g. HTML button): *_finish*, *_cancel*, or *_targetX* (with X as desired page number)

More AbstractWizardFormController

- Most of the same methods as *SimpleFormController* for controlling the form and the submission.
- A few additional methods:
  - *validatePage(...)* - perform partial validation of the command object based on what page was submitted
  - *processFinish(...)* - perform the final action based on a successful submit
  - processCancel(...) - cleanup after a cancel

AbstractWizardFormController Sample

Let's look at 'sample3' again:

- *src/main/webapp/*
  - WEB-INF/context/portlet/books.xml
  - WEB-INF/jsp/bookAdd.jsp
- *src/main/java/*
  - sample/portlet/BookAddController.java

File Uploads

Pre/post processing
the requests and responses

Handling File Uploads

- Just specify a *MultipartResolver* bean and *DispatcherPortlet* will automatically detect it

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

- Two ways to use this:
  - *ActionRequest* wrapped as *MultipartActionRequest*, which has methods for accessing the files
  - Bind directly to Command objects using *PropertyEditors* for *MultipartFiles*:

    ByteArrayMultipartFileEditor, StringMultipartFileEditor

MultipartResolver Sample

Let's move on to 'sample4':

- *src/main/webapp/*
  - WEB-INF/context/portlet/books.xml
  - WEB-INF/jsp/bookAdd.jsp
  - WEB-INF/jsp/bookEdit.jsp
- *src/main/java/*
  - sample/portlet/BookAddController.java
  - sample/portlet/BookEditController.java

Redirects

Going to a new website via HTTP redirect

Performing Redirects

- We can perform an HTTP redirect during the using *ActionResponse.sendRedirect(...)*
- Have to make sure we **haven't** set any *renderParameters*, the *portletMode*, or the *windowState* before we call it
  - This includes *HandlerInterceptor*s like *ParameterMappingInterceptor*

Redirect Sample

Let's look at 'sample4' again:

- *src/main/webapp/*
  - WEB-INF/context/portlet/books.xml
  - WEB-INF/jsp/bookView.jsp
- *src/main/java/*
  - sample/portlet/BookAddController.java
  - sample/portlet/BookWebsiteRedirectController.java

Annotation-based Dispatching

A whole new approach to Controllers

Annotation Based Dispatching

- New in Spring Framework 2.5!
- Eliminates need for complex *HandlerMapping* configuration to deal with navigation via Portlet Modes and Request Parameters
- Allows related logic to be combined into a single Controller class
- Likely to replace the entire *Controller* hierarchy – most capability already supported

Spring MVC Annotations

- **@Controller** – class stereotype for controller classes so they can be found and mapped
- **@SessionAttributes** – list model attributes to be stored in the session (command object)
- **@RequestMapping** – class/method mapping to requests (mode, parameters, etc.)
- **@RequestParam** – bind method params to request params
- **@ModelAttribute** – bind method params or return values to model attributes
- **@InitBinder** – method to setup binder for putting form submission into command obj

Annotation Bean Definitions

```
<context:annotation-config/>

<bean class="org.springframework.web.portlet.mvc.
    annotation.DefaultAnnotationHandlerMapping">

    <property name="interceptors">

        <bean class="org.springframework.web.portlet.handler.
            ParameterMappingInterceptor"/>

    </property>

</bean>

<bean class="org.sample.MyViewController"/>

<bean class="org.sample.MyEditController"/>

<bean class="org.sample.MyHelpController"/>
```

Dispatching Annotation Examples

```
@Controller
@RequestMapping("VIEW")
@SessionAttributes("item")
public class MyViewController {

    @RequestMapping
    public String listItems(Model model) {
        model.addAttribute("items",
            this.itemService.getAllItems());
        return "itemList";
    }

    @RequestMapping(params="action=view")
    public String viewPet(
        @RequestParam("item") int itemId, Model model) {
        model.addAttribute("item",
            this.itemService.getItem(itemId));
        return "itemDetails";
    }   ...
```

Dispatching Annotation Examples

```
...

@ModelAttribute("dateFormat")
protected String dateFormat(PortletPreferences prefs) {
    return preferences.getValue("dateFormat",
        itemService.DEFAULT_DATE_FORMAT);
}

@InitBinder
public void initBinder(PortletRequestDataBinder binder,
        PortletPreferences preferences) {
    String format = preferences.getValue("dateFormat",
        ItemService.DEFAULT_DATE_FORMAT);
    SimpleDateFormat dateFormat =
        new SimpleDateFormat(formatString);
    binder.registerCustomEditor(Date.class,
        new CustomDateEditor(dateFormat, true));
}

...
```

Annotation Dispatching Sample

Let's move on to 'sample5':

- *src/main/webapp/*
  - WEB-INF/context/portlet/myBooks.xml
  - WEB-INF/context/portlet/books.xml
- *src/main/java/*
  - sample/portlet/MyBooksController.java
  - sample/portlet/BooksController.java

Spring Security for Portlets

Controlling access within the portlet

Portal Authentication

- The portal is completely responsible for authentication
  - This means we just use what it gives us – we don't redirect for authentication purpose
- The JSR 168 *PortletRequest* class provides two methods for getting user identity (the same ones as the Servlet spec)

```
String getRemoteUser()

Principal getUserPrincipal()
```

Portal Authorization

- Portals generally provide the ability to assign a set of "Roles" to the User
- The JSR 168 *PortletRequest* class provides a method for getting at these roles (the same ones as the Servlet spec)

```
boolean isUserInRole(String)
```

Declaring Portal Roles

- Same as declaring roles for Servlet container-based security
- Include all portal roles that may be used in *web.xml*:

```
...
<security-role>
   <role-name>manager</role-name>
</security-role>
...
```

Mapping Portal Roles To Portlet Roles

- In *portlet.xml*:

```
<portlet>
    <portlet-name>books</portlet-name>
    ...
    <security-role-ref>
        <role-name>ADMINISTRATOR</role-name>
        <role-link>manager</role-link>
    </security-role-ref>
</portlet>
```

Portlet Role

Portal Role

Warning!

If you are storing your *SecurityContext* in the *PortletSession* with *APPLICATION_SCOPE* (more on this later), make sure these are the same in all your **<portlet>** declarations – the first one to be invoked on a page will determine the mapping for all portlets in your webapp.

Security Constraints

- Require a secure transport in *portlet.xml*:

```
<portlet-app>
    ...
    <portlet>
        <portlet-name>accountSummary</portlet-name>
        ...
    </portlet>
    ...
    <security-constraint>
         <display-name>Secure Portlets</display-name>
         <portlet-collection>
             <portlet-name>accountSummary</portlet-name>
        </portlet-collection>
        <user-data-constraint/>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    ...
</portlet-app>
```

Other Portlet Security Info

- *PortletRequest* has a couple other key security-related methods:

```
String getAuthType()
```

Returns name of authentication scheme used (BASIC_AUTH, CLIENT_CERT_AUTH, custom) or null if user is not authenticated.

```
boolean isSecure()
```

Returns true if the request was made over a secure channel (such as HTTPS)

Portlet User Attributes

- Can also use the **`USER_INFO`** Map available as a *PortletRequest* attribute.
- May contain arbitrary user information:
  - **`user.name.given`**
  - **`user.bdate`**
  - **`user.gender`**
  - etc.
- Some portals expose security-related information here, but this mechanism should be avoided if possible

What Is Spring Security?

- Powerful, flexible security framework for enterprise software
- Emphasis on applications using Spring
- Comprehensive authentication, authorization, and instance-based access control
- Avoids security code in your business logic – treats security as a cross-cutting concern
- Built-in support for a wide variety of authentication and integration standards

Spring Security Releases

- Acegi Security (the old name)
  - Current Version: 1.0.7
  - Initial GA Release: May 2006
  - Portlet support in Sandbox
- Spring Security (the new name)
  - Current Version: 2.0.0
  - Initial GA Release: April 2008
  - Portlet support Included
  - Changes packaging from **`org.acegisecurity`** to **`org.springframework.security`**

Applications Are Like Onions

- Spring Security can be applied at multiple layers in your application:
  - Apply security as markup is constructed in the Rendering Layer using the supplied JSP taglib
  - Restrict access to areas of web application in the Dispatch Layer based on URL pattern-matching
  - Secure method invocation on the Service Layer to ensure calls are from properly authorized user
  - Provide Access Control Lists (ACLs) for individual objects in the Domain Layer

Portlet Challenges

- Portlets have some key differences from Servlets:
  - No Filters
  - Can't treat URLs like Paths
  - Multiple Request Phases
- These create some challenges in applying the normal Spring Security patterns
- So we need some different infrastructure for wiring Spring Security into our portlet application

Six Main Portlet Security Beans

- PortletProcessingInterceptor
- AuthenticationManager
- AuthenticationDetailsSource
- AuthenticationProvider
- UserDetailsService
- PortletSessionContextIntegrationInterceptor

PortletProcessingInterceptor

- Interceptor that processes portlet requests for authentication by invoking the configured *AuthenticationManager*
- Creates the initial *AuthenticationToken* from the *PortletRequest* security methods

```
<bean id="portletProcessingInterceptor"
    class="org.springframework.security.ui.portlet.
        PortletProcessingInterceptor">
    <property name="authenticationManager"
        ref="authenticationManager" />
    <property name="authenticationDetailsSource"
        ref="portletAuthenticationDetailsSource" />
</bean>
```

Portlet equivalent of *AuthenticationProcessingFilter* used for traditional servlet web applications

AuthenticationManager

- Use normal provider-based *AuthenticationManager* bean
- Declared via special namespace schema:

```
<sec:authentication-manager
   alias="authenticationManager" />
```

Can use multiple providers if you are authenticating from Portlets and Servlets

AuthenticationDetailsSource

- Can be used to check isUserInRole(...) to get list of Portal Roles into the Authentication Request:

```
<bean name="portletAuthenticationDetailsSource"
    class="org.springframework.security.ui.portlet.
    PortletPreAuthenticatedAuthenticationDetailsSource">
    <property name="mappableRolesRetriever">
        <bean class="org.springframework.security.
        authoritymapping.SimpleMappableAttributesRetriever">
            <property name="mappableAttributes">
                <list>
                    <value>ADMIN</value>
                </list>
            </property>
        </bean>
    </property>
</bean>
```

Only needed if we are using Portal Roles for our security decisions

AuthenticationProvider

- PreAuthenticatedAuthenticationProvider processes pre-authenticated authentication request (from *PortletProcessingInterceptor*)
- A valid *PreAuthenticatedAuthenticationToken* with non-null principal & credentials will succeed

```
<bean id="portletAuthenticationProvider"
    class="org.springframework.security.providers.preauth.
        PreAuthenticatedAuthenticationProvider">
    <sec:custom-authentication-provider />
    <property name="preAuthenticatedUserDetailsService"
        ref="preAuthenticatedUserDetailsService" />
</bean>
```

UserDetailsService

- Bean that knows how to populate user details (including *GrantedAuthorities*) for the authenticated user
  - *PreAuthenticatedGrantedAuthoritiesUserDetailsService* will use purely data contained in the *PreAuthenticatedAuthenticationToken*

```
<bean name="preAuthenticatedUserDetailsService"
    class="org.springframework.security.providers.preauth.
        PreAuthenticatedGrantedAuthoritiesUserDetailsService" />
```

Can also use any other *UserDetailsService* that can populate *UserDetails* by username, such as *JdbcUserDetailsManager* or *LdapUserDetailsManager*

PortletSessionContextIntegrationInterceptor

- Interceptor that retrieves/stores the contents of the *SecurityContextHolder* in the active *PortletSession*
- Without this, every request would trigger a full authentication cycle
- Default is to use **APPLICATION_SCOPE**

```
<bean id="portletSessionContextIntegrationInterceptor"
    class="org.springframework.security.context.
        PortletSessionContextIntegrationInterceptor" />
```

Portlet equivalent of *HttpSessionContextIntegrationFilter*, used for traditional servlet web applications

Using The Two Security Interceptors

- Add them to our Portlet's *HandlerMapping*:

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

Warning! This ordering is critical – they will not work correctly if they are reversed!

Applying Security To The Portlet

- View Layer
  - JSP TagLib
- Dispatch Layer
  - HandlerInterceptor
  - Annotations
- Service Layer
  - MethodSecurityInterceptor

Spring Security JSP TagLib

- Allows us to access authentication information and to check authorizations
- Useful for showing/hiding information or navigation controls based on security info

```
<%@ taglib prefix="sec"
    uri="http://www.springframework.org/security/tags" %>

<p>Username: <sec:authentication property="principal.username"/></p>

<sec:authorize ifAllGranted="ROLE_USER">
    <p>You are an authorized user of this system.</p>
</sec:authorize>

<sec:authorize ifAllGranted="ROLE_ADMINISTRATOR">
    <p>You are an administrator of this system.</p>
</sec:authorize>
```

Warning: Don't rely on this to restrict access to areas of the application. Just because navigation doesn't appear in the markup doesn't mean a clever hacker can't generate a GET/POST that will still get there.

Secure Portlet Request Dispatching

- Portlet Requests don't have a path structure, so we can't use the path-based patterns of *FilterSecurityInterceptor* to control access
- Something standard may be added in the future – perhaps a *ConfigAttributeDefinition* for various aspects of Portlet Requests that we can use as an *ObjectDefinitionSource*

Using a *HandlerInterceptor*

- Best practice in Spring 2.0 is to build a custom *HandlerInterceptor* for your Portlet
- Compare contents of *SecurityContextHolder. getContext(). getAuthentication()* with Portlet Mode, Window State, Render Parameters – whatever you want to use to determine permission
- Throw a *PortletSecurityException* if access is not permitted, otherwise allow processing to proceed

Using Security Annotations

- If using Spring 2.5 Annotation-based Dispatching, use Security Annotations as well
  - ApplicationContext entry:

```
<sec:global-method-security secured-annotations="enabled" />
```

  - Annotated method:

```
import org.springframework.security.annotation.Secured;
...
    @Secured({"ROLE_ADMIN"})
    @RequestMapping(params="action=view")
    public String deleteItems(RequestParam("item") int itemId) {
        ...
```

Using MethodSecurityInterceptor

```
<bean id="myService" class="sample.service.MyService">
    <sec:intercept-methods
        access-decision-manager-ref="accessDecisionManager">
        <sec:protect method="sample.service.MyService.*"
            access="IS_AUTHENTICATED_FULLY" />
        <sec:protect method="sample.service.MyService.add*"
            access="ROLE_ADMINISTRATOR" />
        <sec:protect method="sample.service.MyService.del*"
            access="ROLE_ADMINISTRATOR" />
        <sec:protect method="sample.service.MyService.save*"
            access="ROLE_ADMINISTRATOR" />
    </sec:intercept-methods>
</bean>
```

AccessDecisionManager

- Standard Spring Security bean for making decisions about access to resources

```
<bean id="accessDecisionManager"
    class="org.springframework.security.vote.
        AffirmativeBased">
    <property name="decisionVoters">
        <list>
            <bean class="org.springframework.security.
                vote.RoleVoter" />
            <bean class="org.springframework.security.
                vote.AuthenticatedVoter" />
        </list>
    </property>
</bean>
```

Portlet Spring Security Sample

Let's move on to 'sample6':

- *src/main/webapp/*
  - WEB-INF/context/applicationContext.xml
  - WEB-INF/context/portlet/books.xml
  - WEB-INF/jsp/books.jsp

Applying Portlet Security to Servlets

- We can reuse the Portlet *SecurityContext* in getting resources from Servlets in the same web application
- Useful for securing:
  - AJAX Calls
  - Dynamic Images
  - PDF Reports
- Need to get Portlets and Servlets to share session data to do this

Portlets & Servlets Sharing Session

- Possible according to JSR 168 (PLT 15.4)
  - Must be in the same webapp
  - Portlet must use **`APPLICATION_SCOPE`**
- Sometime tricky in practice
  - Portlet requests go thru Portal webapp URL
  - Servlet requests go thru Portlet webapp URL
  - Session tracking via **`JSESSIONID`** Cookie usually uses URL path to webapp – not shared!

Tomcat 5.5.4 +

On **`<Connector>`** element set **`emptySessionPath=true`**

Apply Servlet Filter Chain

- In *web.xml*:

```
<filter>
    <filter-name>securityFilterChainProxy</filter-name>
    <filter-class>org.springframework.web.filter.
        DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
    <filter-name>securityFilterChainProxy</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

FilterChainProxy

- Since the portal handles authentication, you only need a few entries in this bean:

```
<bean id="servletSecurityFilterChainProxy"
    class="org.springframework.security.util.
        FilterChainProxy">
    <sec:filter-chain-map path-type="ant">
        <sec:filter-chain pattern="/**"
            filters="httpSessionContextIntegrationFilter,
                exceptionTranslationFilter,
                filterSecurityInterceptor" />
    </sec:filter-chain-map>
</bean>
```

HttpSessionContextIntegrationFilter

- If session sharing is working properly, it will populate the SecurityContextHolder using the same SecurityContext as the Portlet side

```
<bean id="httpSessionContextIntegrationFilter"
    class="org.springframework.security.context.
        HttpSessionContextIntegrationFilter" />
```

This will only work if *PortletSessionContextIntegrationInterceptor* is storing in the **APPLICATION_SCOPE** of the *PortletSession* (which is the default)

ExceptionTranslationFilter

- Since we are relying on the Portal for authentication, then an Exception means that authentication has already failed
- *PreAuthenticatedProcessingFilterEntryPoint* returns **SC_FORBIDDEN** (HTTP 403 error)

```
<bean id="exceptionTranslationFilter"
   class="org.springframework.security.ui.
      ExceptionTranslationFilter">
   <property name="authenticationEntryPoint">
      <bean class="org.springframework.security.ui.preauth.
         PreAuthenticatedProcessingFilterEntryPoint" />
   </property>
</bean>
```

FilterSecurityInterceptor

- Secure resource URLs accordingly
- Use the same *AuthenticationManager* and *AccessDecisionManager* as in the portlet

```
<bean id="filterSecurityInterceptor"
    class="org.springframework.security.intercept.web.
        FilterSecurityInterceptor">
    <property name="authenticationManager"
        ref="authenticationManager" />
    <property name="accessDecisionManager"
        ref="accessDecisionManager" />
    <property name="objectDefinitionSource">
        <sec:filter-invocation-definition-source>
            <sec:intercept-url pattern="/resources/**"
                access="IS_AUTHENTICATED_FULLY" />
        </sec:filter-invocation-definition-source>
    </property>
</bean>
```

Servlet Spring Security Sample

Let's go back to 'sample6':

- *src/main/webapp/*
  - WEB-INF/context/applicationContext.xml
  - WEB-INF/context/servlet/resources.xml

Adapting Other Frameworks

Using Spring MVC as a bridge

Adapting Other Frameworks

- Spring Web MVC can adapt other frameworks
- *DispatcherPortlet/DispatcherServlet* and *HandlerMapping* can dispatch requests to any Class (that's why we call them *Handlers*)
- Simply create implementation of *HandlerAdapter* interface that adapts requests to the given framework
- Or use Annotations to create a Controller
- Allows framework objects to be created as Spring Beans and inject dependencies

Reuse Existing Portlets

- Two mechanisms for using existing Portlets as Handlers inside Spring MVC:
  - *SimplePortletHandlerAdapter* adapts a Portlet into a Handler for use with HandlerMappings
  - *PortletWrappingController* wraps a Portlet as a Spring MVC Controller – allows for specifying Portlet *init-parameters*
- Useful for:
  - Applying Interceptors to existing Portlets
  - Use dependency injection for initialization

Spring Web Flow

SWF In a Portlet Context

PortletFlowController

- Spring Web Flow Front Controller for use within a Portlet environment
- Portlet requests 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 (Spring Web MVC, Spring Portlet MVC, Struts, 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>
```