Views

Cognizant
Passion for building stronger businesses

About the Author

Created By:	Ramesh C.P. (161646)
Credential Information:	8+ years of experience in technical training
Version and Date:	Spring MVC/PPT/1011/3.0

Cognizant Certified Official Curriculum

Icons Used

Questions

Tools

Hands on Exercise

Coding Standards

Test Your Understanding

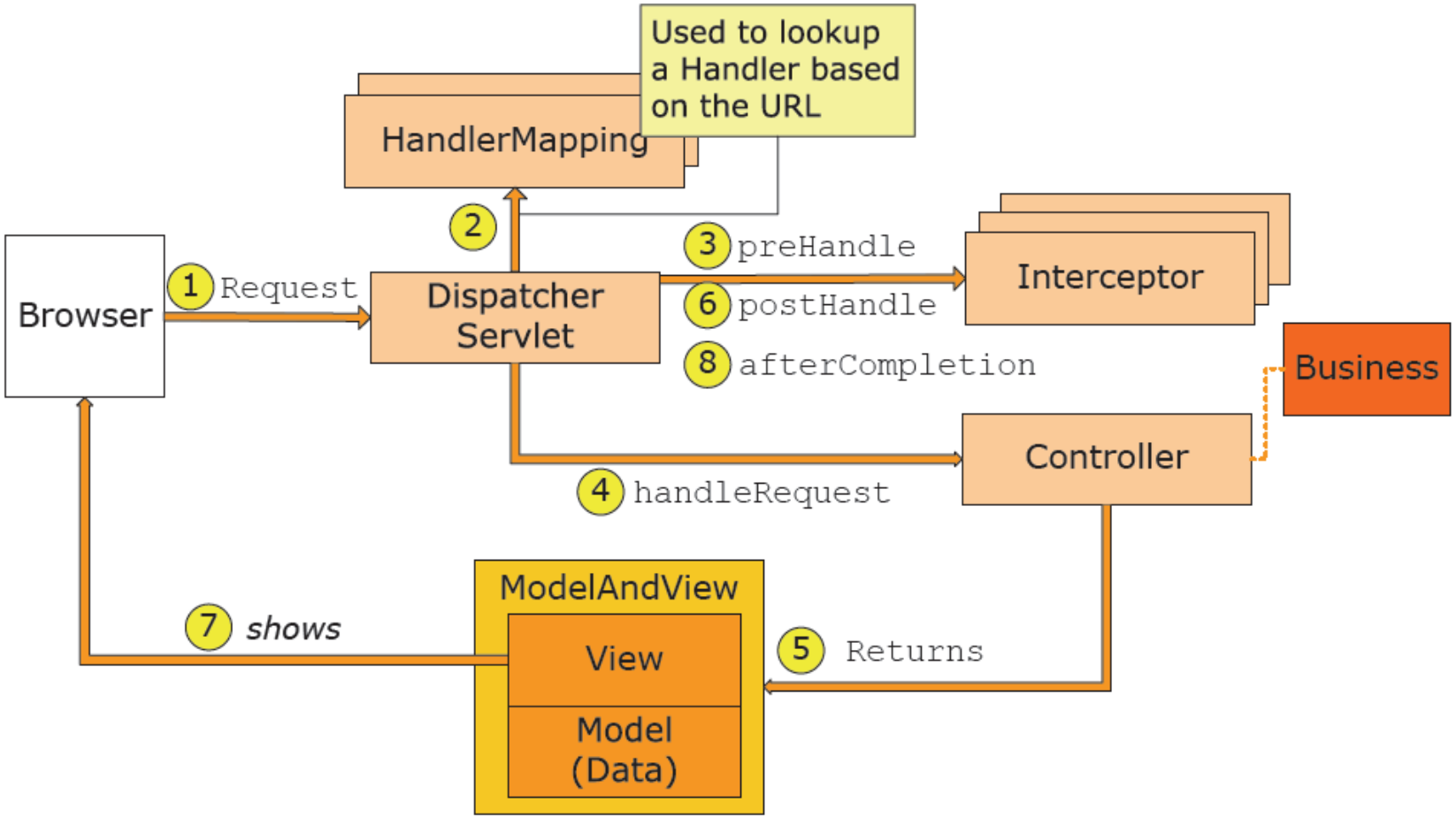
Reference

Demonstration

A Welcome Break

Contacts

Review of Architecture



Views: Overview

- Introduction:
  - One of the areas in which Spring excels is in the separation of view technologies from the rest of the MVC framework.
    - For example, deciding to use Velocity or XSLT in place of an existing JSP is primarily a matter of configuration.

Views: Objectives

- Objective:

After completing this chapter you will be able to:

- Configure different view resolvers
- Integrate with different view technologies

ViewResolvers

- Normally a View is not returned programmatically but uses a `ViewResolver`
  - Allows you to use named and (pre) configured View objects
  - The `ModelAndView` class has overloaded constructors to support looking up a View using a `ViewResolver`:

```
public ModelAndView handleRequest(HttpServletRequest request,
                HttpServletResponse response) throws Exception {
  String dvdID = request.getParameter("dvdID");
  DVD dvd = manager.getProductInfo(dvdID);
  return new ModelAndView("dvd-details","dvd",dvd);
}
```

ViewResolvers (Contd.)

- Standard `ViewResolver` implementations:

Implementation	Description
`BeanNameViewResolver`	Maps views to bean names (difficult in a large application)
`ResourceBundleViewResolver`	Views are defined in a properties file (supports `I`18N)
`UrlBasedViewResolver`	Views are found based on a Url
`XmlViewResolver`	Similar to the ResourceBundle variant except uses an XML file (no `I`18N support)

Resolving Views

- ViewResolver- An interface that need to be implemented by objects that can resolve views by name.

```
public  interface ViewResolver{
   View resolveViewName(String viewName, Locale locale)
            throws Exception;
}
```

ViewResolvers Example

- WebApplication Context

```
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
  <property name="basenames">
    <list><value>inventory</value></list>
  </property>
</bean>
```

- inventory.properties

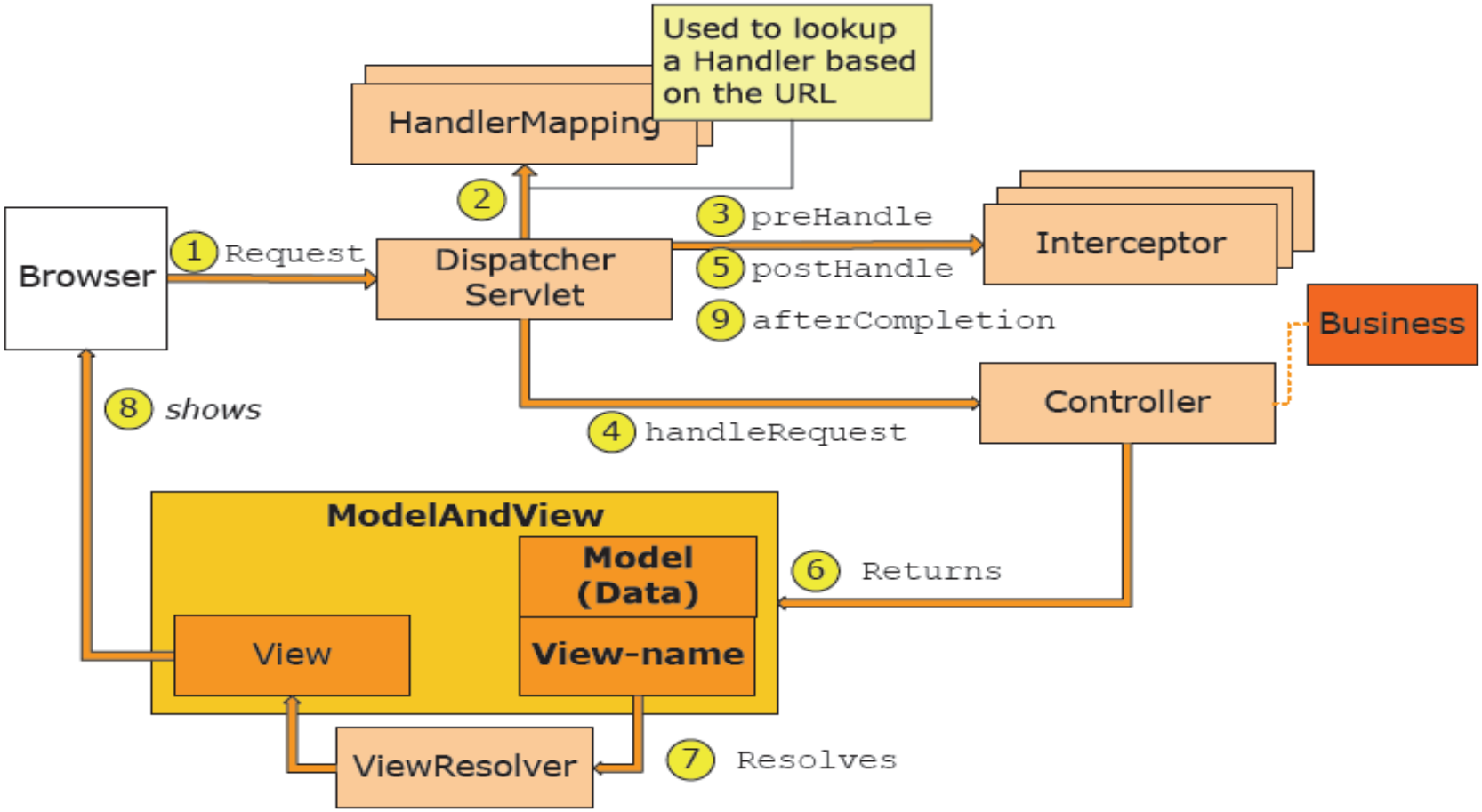
```
DVDlist.class=org.springframework.web.servlet.view.JstlView
DVDlist.url=/WEB-INF/views/inventory/ListAll.jsp
```

- Controller Class

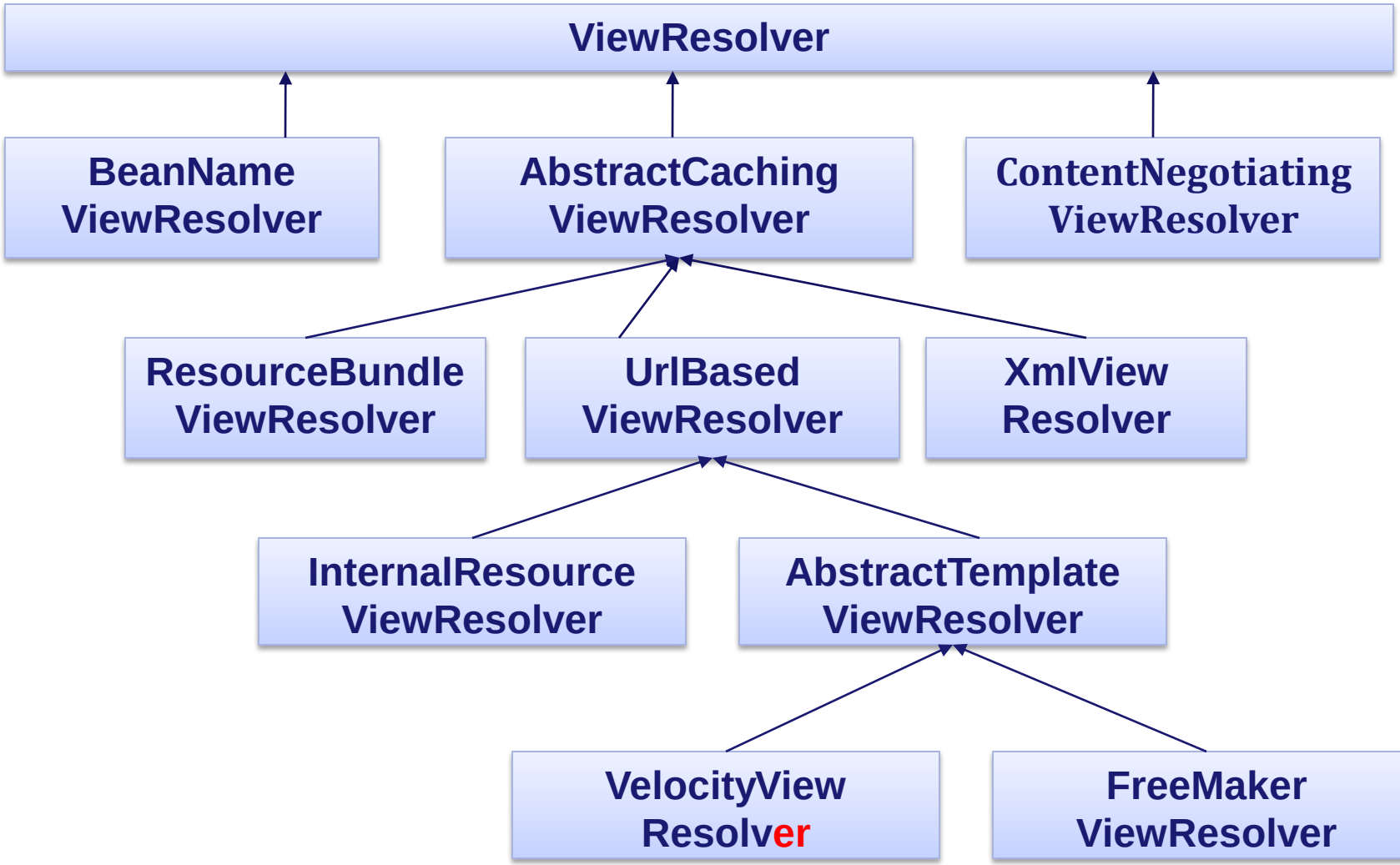
```
public ModelAndView viewAll(HttpServletRequest request,
                            HttpServletResponse response) throws Exception {
   Collection<DVDInfo> all = manager.getAll();
   return new ModelAndView("DVDlist", "catalog", all);
}
```

ViewResolvers (Contd.)

- Architecture incorporating `ViewResolvers`



ViewResolver : Hierarchy



BeanNameViewResolver

- Simple implementation of ViewResolver that interprets a view name as bean name in the current application context

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

Note: The BeanNameViewResolver queries only the BeanFactory in which it was declared.

XmlViewResolver

- Implementation of ViewResolver that uses bean definitions in an XML file, specified by resource location.
- The file will typically be located in the WEB-INF directory;
  - default is "/WEB-INF/views.xml“

```
<bean id="excelViewResolver"
      class="org.springframework.web.servlet.view.XmlViewResolver">
  <property name="order" value="1"/>
  <property name="location" value="/WEB-INF/simpleviews.xml"/>
</bean>
```

and in simpleviews.xml

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

ResourceBundleViewResolver

- ViewResolver implementation that uses bean definitions in a ResourceBundle , specified by the bundle basename.
- The bundle is typically defined in a properties file, located in the class path. The default bundle basename is "views".
- It allows for localized views based on a locale from the user's request.
- For example, the basename "views" will be resolved as class path resources "views_de_AT.properties", "views_de.properties", "views.properties" - for a given Locale "de_AT".

ResourceBundleViewResolver (Contd.)

```
# /WEB-INF/classes/views.properties
index.class=org.springframework.web.servlet.view.JstlView
index.url=/WEB-INF/jsps/index.jsp
welcome.class=org.springframework.web.servlet.view.JstlView
welcome.url=/WEB-INF/jsps/store.jsp

# /WEB-INF/classes/views_nl.properties
Welcome.class=org.springframework.web.servlet.view.JstlView
welcome.url=/WEB-INF/jsps/nl/welcome.jsp

<bean class="org.springframework.web.servlet.view.ResourceBundleView">
    <property name="baseName"><value>views</value></property
</bean>
```

UrlBasedViewResolvers

- Maps logical view names directly to URLs that it hands over to the view class specified.
- The view class in turn uses the URL to render the response.
- The URL can, for instance, point to a JavaServer Page, a Velocity template, or an XSLT style sheet.

InternalResourceViewResolver

- A convenience subclass of UrlBasedViewResolver that supports InternalResourceView (i.e. Servlets and JSPs).

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

Note: The default view class is InternalResourceView

ContentNegotiatingViewResolver

- The ContentNegotiatingViewResolver does not resolve views itself but rather delegates to other view resolvers.
- This view supports multiple representations of a resource to different type of views like JSP, XML, RSS, JSON etc based on the file extension or Accept header of the HTTP request.
- Here is an example configuration of a ContentNegotiatingViewResolver.

```
<bean class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver"> <property
    name="mediaTypes">
    <map>
            <entry key="atom" value="application/atom+xml"/>
            <entry key="html" value="text/html"/>
            <entry key="json" value="application/json"/>
    </map>
    </property>
     <property name="viewResolvers">
    ...
    </property>
</bean>
```

ModelAndView *and* View

- The `ModelAndView` consists of:
  - A *model* which is basically a `Map` with named objects (similar to session or request attributes) to be used by the `view`
- The `View` in the `ModelAndView`:
  - Is an implementation of the `View` interface
  - Contains one method:

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

The View *in* ModelAndView

- It is supplied with the `Model`
- Some standard implementations:

Example of standard `View` Implementations	
`AbstractJExcelView`	`JasperReportsHtmlView`
`AbstractPdfView`	`JasperReportsPdfView`
`RedirectView`	`AbstractXsltView`
`VelocityView`	`FreeMarkerView`
`JstlView`	…

ModelAndView

- Holder for both Model and View in the web MVC framework.
- This class merely holds both to make it possible for a controller to return both model and view in a single return value.
- The view can take the form of a String view name which will need to be resolved by a ViewResolver object(alternatively a View object can be specified directly)
- Alternatively a View object can be specified directly.
- The model is a Map, allowing the use of multiple objects keyed by name.
- While rendering the view, the map will be available specific to the view technology used (for example, in the VelocityContext when using Velocity or in the request context when using JSPs).

ModelAndView (Contd.)

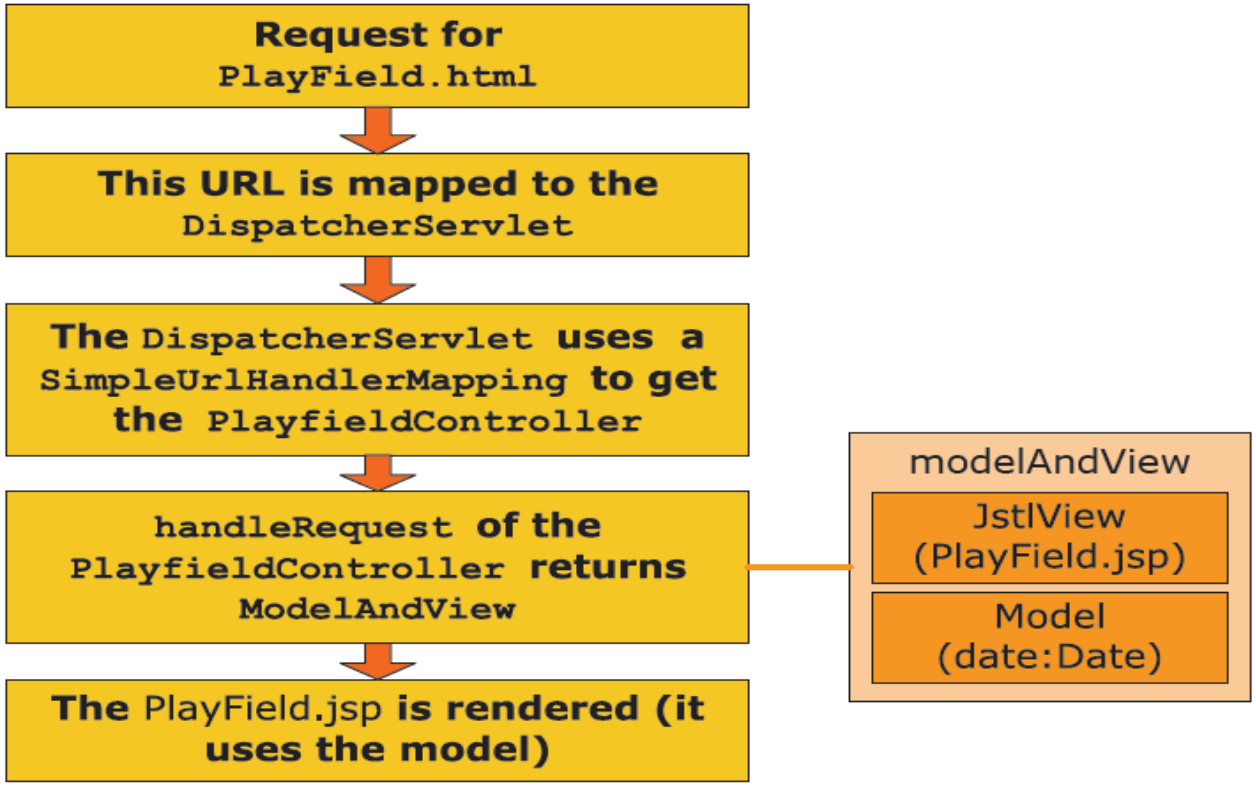
- Constructors:
  - public **ModelAndView**(String viewName)
    - Convenient constructor when there is no model data to expose
  - public **ModelAndView**(String viewName, Map model)
    - Creates new ModelAndView given a view name and a model.
  - public **ModelAndView**(String viewName, String modelName, Object modelObject)
    - Convenient constructor to take a single model object.

ModelAndView (Contd.)

- Methods:
  - **addAllObjects**(Map modelMap)
  - **addObject**(String modelName, Object modelObject)
  - **clear**()
  - **getModel**()
  - **getViewName**()

View Processing

- When a request is made the following happens:



ModelAndView Example

```
public class PlayfieldController implements Controller {
  public ModelAndView handleRequest(HttpServletRequest request,
                      HttpServletResponse response) throws Exception {
    JstlView view = new JstlView();
    view.setUrl("/WEB-INF/views/playfield/PlayField.jsp");
    return new ModelAndView(view, "date", new Date());
  }
}
```

ModelAndView Example (Contd.)

PlayField.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <body>
    <p>It is ${date} now</p>
  </body>
</html>
```

web.xml

```
<!-- in the web.xml -->
<servlet-mapping>
  <servlet-name>DispatcherSample</servlet-name>
  <url-pattern>/PlayField.html</url-pattern>
</servlet-mapping>
```

ModelAndView Example (Contd.)

DispatcherSample-servlet.xml

```
<bean id="handlerMapping"
      class="....SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <prop key="/PlayField.html">playfieldController</prop>
    </props>
  </property>
</bean>
<bean id="playfieldController" class="demos.PlayfieldController"/>
```

A Custom `View` Example

- An `AbstractExcelView` implementation

```
public class CatalogWorksheet extends AbstractExcelView {
  protected void buildExcelDocument(Map model,
        HSSFWorkbook workbook, HttpServletRequest request,
        HttpServletResponse response) throws Exception {
    HSSFSheet sheet = workbook.createSheet("Spring");
    HSSFCell cell = getCell(sheet, 0, 0);
    setText(cell, "Spring-Excel test");
    Collection<DVD> dvds = (Collection<DVD>) model.get("dvds");
    setText(cell, "Export of our DVDs");
    setText(getCell(sheet,1,0), "ID");
    setText(getCell(sheet,1,1), "Title");
    int i = 0;
    for (DVD dvd : dvds) {
      int row = 2 + i;
      setText(getCell(sheet, row, 0), dvd.getId());
      setText(getCell(sheet, row, 1),dvd.getTitle());
      i++;
    }
  }
}
```

A Custom `View` Example (Contd.)

- The corresponding controller implementation

```
Collection<DVD> dvds = manager.getAll();
View v = getApplicationContext().getBean("catalogWorksheet", View.class);
return new ModelAndView(v,"dvds",dvds);
```

- Note it uses getBean as opposed to instantiation because it is stateless (which scales better)

```
<bean id="catalogWorksheet"
      class="demos.springmvc.CatalogWorksheet"/>
```

Chaining ViewResolvers

- Spring supports more than just one view resolver.
- This allows you to chain resolvers and, for example, override specific views in certain circumstances.
- Chaining view resolvers is pretty straightforward - just add more than one resolver to your application context.
- If necessary, set the order property to specify an order. Remember, the higher the order property, the later the view resolver will be positioned in the chain.

Chaining ViewResolvers (Contd.)

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

Note: If a specific view resolver does not result in a view, Spring will inspect the context to see if other view resolvers are configured. If there are additional view resolvers, it will continue to inspect them. If not, it will throw an Exception.

Integrating view technologies

- One of the areas in which Spring excels is in the separation of view technologies from the rest of the MVC framework.
- The various views that are supported are:
  - JSP and JSTL
  - Tiles
  - Velocity and FreeMarker
  - XSLT
  - Document Views(PDF/Excel)
  - JasperReports

JSP and JSTL

- Spring provides out-of-the-box solutions for JSP and JSTL views.
- Using JSP or JSTL is done using a normal view resolver defined in the WebApplicationContext.
- The most commonly used view resolvers when developing with JSPs are the InternalResourceViewResolver and the ResourceBundleViewResolver(Both are declared in the WebApplicationContext)

JSP and JSTL (Contd.)

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

# And a sample properties file is uses (views.properties in WEB-INF/classes):
welcome.class=org.springframework.web.servlet.view.JstlView
welcome.url=/WEB-INF/jsp/welcome.jsp
productList.class=org.springframework.web.servlet.view.JstlView
productList.url=/WEB-INF/jsp/productlist.jsp
```

Note: With a **ResourceBundleViewResolver** you can mix different types of views using only one resolver.

JSP and JSTL (Contd.)

```
<!– the InternalResourceViewResolver  -- >
<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>
```

XSLT - Controller

```
@Controller
Public class XMLAbstractController{
protected ModelAndView handleRequestInternal( HttpServletRequest request,
                                              HttpServletResponse response) throws Exception {
   Map map = new HashMap();
   List employeeNames = new ArrayList();

   employeeNames.add(“Johnson");
   employeeNames.add(“Steve");
   …

   map.put(“empNames", employeeNames);

   return new ModelAndView("xl", map);
}
```

XSLT - view

```
public class XMLView extends AbstractXsltView {
   protected Source createXsltSource(Map model, String rootName, HttpServletRequest
      request, HttpServletResponse response) throws Exception {
      Document document =
      DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();
      Element root = document.createElement(rootName);
      List  empNames = (List) model.get("empNames");
      for (Iterator it = empNames.iterator(); it.hasNext();) {
         String nextName = (String) it.next();
         Element nameNode = document.createElement("Name");
         Text textNode = document.createTextNode(nextName);
         nameNode.appendChild(textNode);
         root.appendChild(nameNode);
      }
        return new DOMSource(root);
   }
```

XSLT – views.properties

home.class=XMLView

home.stylesheetLocation=/WEB-INF/xsl/name.xslt

home.root=EmpNames

Excel - View

- For Excel, we write a subclass of org.springframework.web.servlet.view.document.AbstractExcelView (for Excel files generated by POI)

```
public class AbstractEXcelView extends AbstractExcelView {
   protected void buildExcelDocument( Map model,   HSSFWorkbook wb,
      HttpServletRequest req, HttpServletResponse resp) throws Exception {
      HSSFSheet sheet;
      HSSFRow sheetRow;
     HSSFCell cell;
```

Excel – View (Contd.)

```
        // Go to the first sheet
        sheet = wb.createSheet("Spring");
        sheet.setDefaultColumnWidth((short)12);
        // write a text at A1
        cell = getCell( sheet, 0, 0 );
        setText(cell,"Spring-Excel test");
        List empNames = (List ) model.get("empNames");
        for (int i=0; i < empNames.size(); i++) {
            cell = getCell( sheet, 2+i, 0 );
            setText(cell, (String) empNames.get(i));
        }
    }
}
```

PDF - View

```
public class PDFPage extends AbstractPdfView {

   protected void buildPdfDocument( Map model, Document doc,  PdfWriter writer,     `
                                                HttpServletRequest req, HttpServletResponse resp)
                                               throws Exception {
      List empNames = (List) model.get("empNames");

      for (int i=0; i<empNames.size(); i++)
         doc.add( new Paragraph((String) empNames.get(i)));

   }
}
```

Locales

- DispatcherServlet enables you to automatically resolve messages using the client's locale. This is done with *LocaleResolver* objects.
- When a request comes in, the DispatcherServlet looks for a locale resolver and if it finds one it tries to use it to set the locale.
- Besides the automatic locale resolution, you can also change the locale under specific circumstances, based on a parameter in the request.

LocaleChangeInterceptor

- This interceptor needs to be added to one of the handler mappings .
- It will detect a parameter in the request and change the locale on the LocaleResolver that also exists in the context.

```
<bean id="localeChangeInterceptor"
        class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
   <property name="paramName" value="siteLanguage"/>
</bean>

<bean id="localeResolver"
        class="org.springframework.web.servlet.i18n.CookieLocaleResolver"/>
```

LocaleChangeInterceptor (Contd.)

```
<bean id="urlMapping"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="interceptors">
    <list>
      <ref bean="localeChangeInterceptor"/>
    </list>
  </property>
  <property name="mappings">
    <props>
        <prop key="/login.htm">loginForm</prop>
    <prop>
  </property>
</bean>
```

AcceptHeaderLocaleResolver

- This locale resolver inspects the accept-language header in the request that was sent by the browser of the client.
- Usually this header field contains the locale of the client's operating system.

CookieLocaleResolver

- This locale resolver inspects a Cookie that might exist on the client, to see if a locale is specified. If so, it uses that specific locale.
- Using the properties of this locale resolver, you can specify the name of the cookie, as well as the maximum age.

```
<bean id="localeResolver"
    class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
  <property name="cookieName" value="clientlanguage"/>

  <!-- in seconds. If set to -1, the cookie is not persisted (deleted when browser shuts
    down) -->
  <property name="cookieMaxAge" value="100000">
</bean>
```

SessionLocaleResolver

- The SessionLocaleResolver allows you to retrieve locales from the session that might be associated with the user's request

Views

Time for a Break !

Views

- Questions from participants

Test Your Understanding

1) Which of the following takes the last priority by default
   a) BeanNameViewResolver
   b) XmlViewResolver
   c) InternalResourceViewResolver

Views: Summary

- Spring provides different kind of view resolvers like:
  - BeanNameViewResolver
  - XmlViewResolver
  - InternalResourceViewResolver
- Spring provides a seamless integration with the following view technologies:
  - JSP and JSTL
  - Velocity and FreeMarker
  - Tiles
  - XSLT
  - Excel and PDF

Views: Source

- http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/view.html

You have successfully completed Views

Click here to proceed

Cognizant
Passion for building stronger businesses