[Spring MVC中的视图解析ViewResolver](http://blog.csdn.net/prince2270/article/details/5891085)

标签： [spring](http://www.csdn.net/tag/spring)[mvc](http://www.csdn.net/tag/mvc)[exception](http://www.csdn.net/tag/exception)[jfreechart](http://www.csdn.net/tag/jfreechart)[object](http://www.csdn.net/tag/object)[bean](http://www.csdn.net/tag/bean)

2010-09-17 15:41 26188人阅读 [评论](http://blog.csdn.net/prince2270/article/details/5891085#comments)(5) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/prince2270/article/details/5891085#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

spring mvc学习笔记（3） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

        在Spring MVC中，当Controller将请求处理结果放入到ModelAndView中以后，DispatcherServlet会根据ModelAndView选择合适的视图进行渲染。那么在Spring MVC中是如何选择合适的View呢？View对象是是如何创建的呢？答案就在ViewResolver中，ViewResolver接口定义了resolverViewName方法，根据viewName创建合适类型的View实现。

        那么，如何配置ViewResolver呢？在Spring中，ViewResolver作为Spring Bean存在，可以在Spring配置文件中进行配置，例如下面的代码，配置了jsp相关的viewResolver。

**[xhtml]** [view plain](http://blog.csdn.net/prince2270/article/details/5891085) [copy](http://blog.csdn.net/prince2270/article/details/5891085)

1. <!-- Resolves view names to protected .jsp resources within the /WEB-INF/views directory -->
2. **<bean** class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>**
3. **<property** name="prefix" value="/WEB-INF/views/"**/>**
4. **<property** name="suffix" value=".jsp"**/>**
5. **</bean>**

       在Spring MVC中，因为ViewResolver是使用bean来配置的，所以扩展起来非常的容易，可以根据自己的需要定制ViewResolver，然后在配置文件中进行相关的配置即可。

       ViewResolver接口声明了resolverViewName方法，这个方法的主要功能是根据ModelAndView中给定的viewName信息，再结合相关的配置，创建出合适类型的View对象。

       ViewResolver接口是在DispatcherServlet中进行调用的，当DispatcherServlet调用完Controller后，会得到一个ModelAndView对象，然后DispatcherServlet会调用render方法进行视图渲染。

**[java]** [view plain](http://blog.csdn.net/prince2270/article/details/5891085) [copy](http://blog.csdn.net/prince2270/article/details/5891085)

1. **protected** **void** render(ModelAndView mv, HttpServletRequest request, HttpServletResponse response) **throws** Exception {
2. // Determine locale for request and apply it to the response.
3. Locale locale = **this**.localeResolver.resolveLocale(request);
4. response.setLocale(locale);
6. View view;
7. **if** (mv.isReference()) {
8. // We need to resolve the view name.
9. view = resolveViewName(mv.getViewName(), mv.getModelInternal(), locale, request);
10. **if** (view == **null**) {
11. **throw** **new** ServletException(
12. "Could not resolve view with name '" + mv.getViewName() + "' in servlet with name '" +
13. getServletName() + "'");
14. }
15. }
16. **else** {
17. // No need to lookup: the ModelAndView object contains the actual View object.
18. view = mv.getView();
19. **if** (view == **null**) {
20. **throw** **new** ServletException("ModelAndView [" + mv + "] neither contains a view name nor a " +
21. "View object in servlet with name '" + getServletName() + "'");
22. }
23. }
25. // Delegate to the View object for rendering.
26. **if** (logger.isDebugEnabled()) {
27. logger.debug("Rendering view [" + view + "] in DispatcherServlet with name '" + getServletName() + "'");
28. }
29. view.render(mv.getModelInternal(), request, response);
30. }

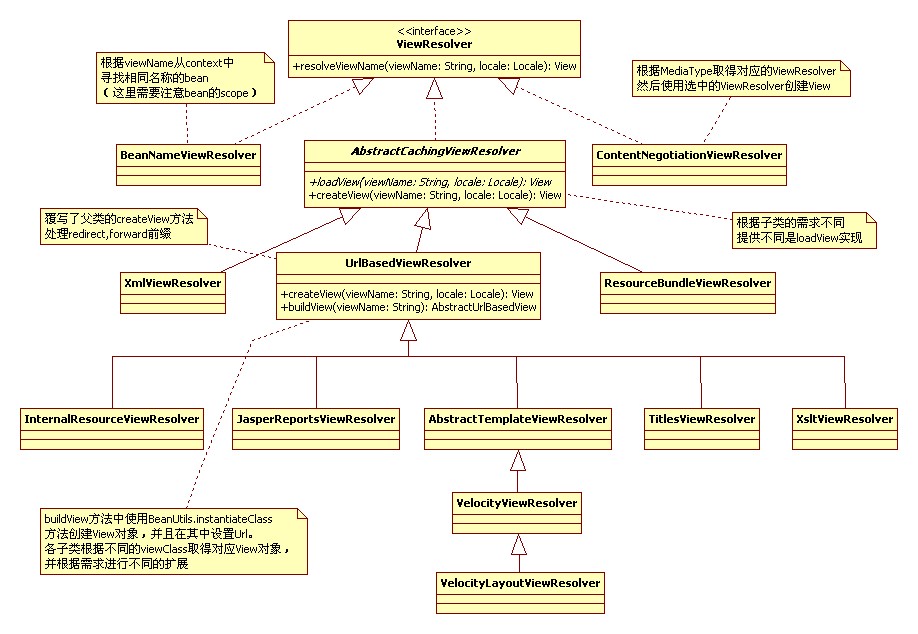
        在DispatcherServlet类中，init方法中已经进行了相关的初始化，配置的ViewResolver信息都存放在viewResolvers中。在render方法中调用resolverViewName方法，在这个方法中逐一调用ViewResolver去取得View对象。

**[java]** [view plain](http://blog.csdn.net/prince2270/article/details/5891085) [copy](http://blog.csdn.net/prince2270/article/details/5891085)

1. **protected** View resolveViewName(String viewName, Map<String, Object> model, Locale locale,
2. HttpServletRequest request) **throws** Exception {
4. **for** (ViewResolver viewResolver : **this**.viewResolvers) {
5. View view = viewResolver.resolveViewName(viewName, locale);
6. **if** (view != **null**) {
7. **return** view;
8. }
9. }
10. **return** **null**;
11. }

        在这里需要关于ViewResolver的选择是通过循环进行的，只选择第一个符合要求的，因此在定义ViewResolver时，需要注意定义其优先级。

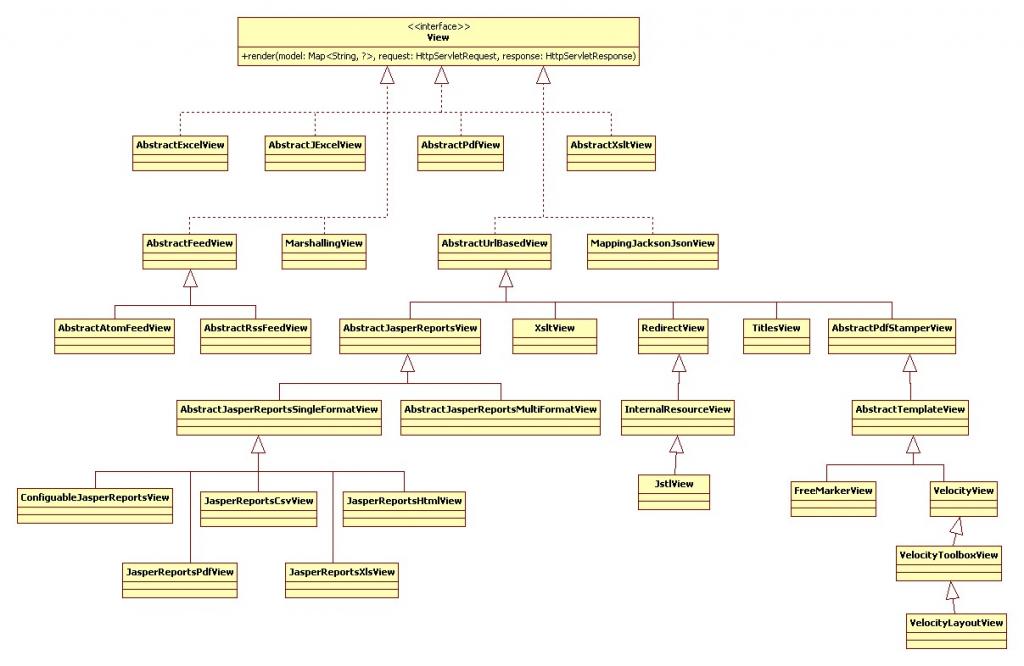
        下面就来着重关注一下ViewResolver的类结构。



       关于View对象的创建，不同的ViewResolver的解决方法是各部相同的。如BeanNameViewResolver是根据viewName选择相应名称的bean（这里需要注意bean的scope，是否需要线程安全），而UrlBasedViewResolver则是使用反射机制，根据viewClass信息创建view对象，因此这个view不受IoC容器的管理。ContentNegotiationViewResolver中可以嵌套ViewResolver，根据不同的的请求类型选择合适的ViewResolver。

       DispatcherServlet得到View对象后，即调用View的render方法，执行真正的渲染工作。

       最后，看一下View的类结构图。



       有上述的View类结构图可知，Spring已经为我们提供了一系列可用的View。同时，如果当前提供的View不能满足我们的要求时，可以通过实现View接口进行扩展。如需要根据model中的数据使用JFreeChart绘图，或者将这些数据作为文件下载时，我们可以扩展出JFreeChartView和FileDownloadView等，这样就能更灵活的将同一份数据用不同的方式展现出来。

# [Spring MVC 4之ViewResolver视图解析器](http://blog.csdn.net/shehun1/article/details/43984877)

2015-02-28 13:53 2962人阅读 [评论](http://blog.csdn.net/shehun1/article/details/43984877#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/shehun1/article/details/43984877#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

爪蛙Java（28） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

    所有的We MVC框架都有一套它自己的解析视图的机制，Spring MVC也不例外，它使用ViewResolver进行视图解析，让用户在浏览器中渲染模型。ViewResolver是一种开箱即用的技术，能够解析JSP、Velocity模板和XSLT等多种视图。

    Spring处理视图最重要的两个接口是ViewResolver和View。ViewResolver接口在视图名称和真正的视图之间提供映射； 而View接口则处理请求将真正的视图呈现给用户。

    1.几种常见的ViewResolver视图解析器

    在Spring MVC 4控制器中，所有的处理方法必须返回一个逻辑视图名称，无论是显式的（返回String，View或ModelAndView）还是隐式的。Spring中的视图由视图解析器处理这个逻辑视图名称，Spring有以下几种视图解析器：

  AbstractCachingViewResolver：用来缓存视图的抽象视图解析器。通常情况下，视图在使用前就准备好了。继承改解析器就能够使用视图缓存。

  XmlViewResolver ：XML视图解析器。它实现了ViewResolver接口，接受相同DTD定义的XML配置文件作为Spring的XML bean工厂。

  ResourceBundleViewResolver：它使用了ResourceBundle定义下的bean，实现了ViewResolver接口，指定了绑定包的名称。通常情况下，配置文件会定义在classpath下的properties文件中，默认的文件名字是views.properties。

    UrlBasedViewResolver：它简单实现了ViewResolver接口，它不用显式定义，直接影响逻辑视图到URL的映射。它让你不用任何映射就能通过逻辑视图名称访问资源。

    InternalResourceViewResolver：国际化视图解析器。

    VelocityViewResolver /FreeMarkerViewResolver：Velocity或FreeMarker视图解析器。

    ContentNegotiatingViewResolver：内容谈判视图解析器

    在JSP视图技术中，Spring MVC经常会使用 UrlBasedViewResolver视图解析器，该解析器会将视图名称翻译成URL并通过RequestDispatcher处理请求后渲染视图。

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

    假如我们配置了如上所示的URL视图解析器，我们返回了一个叫“favmvc”的视图名称，视图解析器就会将请求转发到RequestDispatcher，然后跳转到/WEB-INF/views/favmvc.jsp页面。

    假如我们想要在应用中使用不同的视图技术，我们就应该使用 ResourceBundleViewResolver。

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

    ResourceBundleViewResolver对于每个要处理的视图，都会检查 ResourceBundle中basename的唯一性，它使用 [viewname].(class)作为视图类，[viewname].url作为视图的url。

    2. 链式视图解析器（Chaining ViewResolvers）

    Spring支持同时配置多个视图解析器，也就是链式视图解析器。这样，在某些情况下，就能够重写某些视图。如果我们配置了多个视图解析器，并想要给视图解析器排序的话，设定 order 属性就可以指定解析器执行的顺序。order的值越高，解析器执行的顺序越晚。

    下面代码所示的例子由两个视图解析器组成。 InternalResourceViewResolver总是最后一个执行，而 XmlViewResolver则指定解析XML视图（InternalResourceViewResolver不支持Excel视图）。

<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><!-- in views.xml --><beans>  
    <bean name="report" class="org.springframework.example.ReportExcelView"/></beans>

    如果视图解析器没有指定视图的话，Spring就会检查其它的视图解析器，直到有一个完整的包含视图的视图解析器。如果一个完整的视图解析器也没有找到的话，Spring就会抛出 ServletException。

    视图解析器规定如果没有找到视图就返回null。但不是所有的视图解析器都这么做，因为在某些情况下，解析器并不能检测是否存在视图。比如InternalResourceViewResolver在内部使用RequestDispatcher时，调度转发是判断JSP是否存在的唯一方法，但该action却只能执行一次。VelocityViewResolver和其它的一些解析器也这样，区分这些视图解析器能否在找不到视图的情况下返回null，最好的方法就是看官方文档中它是否支持了。由于 InternalResourceViewResolver总会返回视图，在视图链中使用InternalResourceViewResolver就能让你避免这些问题。

    3.  重定向视图

    在controller控制器中强制重定向的方法就是创建并返回Spring的RedirectView实例。在这种情况下，DispatcherServlet不再使用正常的视图机制，因为它已经返回了重定向视图，DispatcherServlet只是告诉视图去显示。

    RedirectView会调用 HttpServletResponse.sendRedirect()方法，然后它就作为HTTP重定向返回给客户端浏览器。默认情况下，所有的模板属性变量都认为是重定向URL，其余的属性自动附加为查询参数。

    redirect前缀

    尽管RedirectView工作机制很好，但如果控制器本身创建RedirectView时，毫无疑问控制器本身知道该如何重定向。这样做并不好，控制器不应该关心响应如何处理，它只是处理被注入的视图名称。

    当我们在返回视图名称时，如果使用了“redirect：”前缀（如“redirect : /login”），UrlBasedViewResolver视图控制器会识别这是一次特殊的重定向，并把redirect后面的视图名称当做重定向的地址。

    假如我们在http://favccxx.com应用中返回“redirect:/favboy”视图时， 系统会重定向到http://favccxx.com/favboy。但如果我们返回了“redirect:http://favsoft.me”这样的视图时，系统会重定向到http://favsoft.me的视图上。

    forward前缀

    forward前缀视图是另一种通过URLBasedViewResolver机制处理到的转向机制，它在视图名称周围创建InternalResourceView，因此这个前缀跟InternalResourceViewResolver和InternalResourceView无关。但这个前缀在你想使用其它视图技术但想强制通过Servlet/JSP引擎处理资源进行转向时是有用的。

    redirect与forward的区别

    redirect方式相当 于"response.sendRedirect()".这种方式外部特征就是浏览器地址栏最后显示的路径是转发后的新的路径.工作方式是这样的，服务器端会首先发一个response给浏览器，然后浏览器收到这个response后再发一个requeset给服务器，然后服务器发新的response给浏览器。这时页面收到的request是一个新从浏览器发来的.这种方式的结果是：

    A.在转发前后有两个不同的request对象,转发前后的两个控制器在request上的参数(request.getParameter())和request属性(request.getAttribute())不能共享。

    B.如果转发前后的两个控制器都配置在spring 拦截器范围内,这样拦截器会拦截前后两个request,即会拦截两次。

    C.最后返回到浏览器后,因为地址栏显示的是转发后的url，所以刷新页面时只会执行后面的url映射的控制器。

    forward方式相当于 request.getRequestDispatcher().forward(request,response) .这种方式的外部特征是浏览器地址显示的路径是转发前的路径。工作方式是这样，forward 发生在服务器内部,在前一个控制器处理完毕后,直接进入下一个控制器处理，并将最后的response发给浏览器。这种方式的结果是：

    A.转发前后是同一个request,后一个控制器可共享前一个控制器的参数与属性。

    B.因为是同一个request，拦截器只会拦截前一个url，如果前一个url在映射时未配置到拦截器拦截，则拦截后一个url，即只拦截一次。

    C.最后返回到浏览器后,因为地址栏显示的是转发前的url，所以刷新页面时会依次执行前后两个控制器。

   总结：本文介绍了Spring MVC 4中常见的几种视图解析器，以及如何配置视图解析器包括链式视图解析器，最后除了正常视图映射机制外，我们还能够通过redirect/forward视图转向机制完成视图的重定向。本文偏向原理性介绍，关于视图解析机制在工作中的使用，并没有做过多的介绍，目的是让读者能够了解ViewResolver的工作机制。