

## [Spring MVC 教程,快速入门,深入分析](#)

资源下载:

[Spring MVC 教程 快速入门 深入分析 V1.1.pdf](#)

[SpringMVC 核心配置文件示例.rar](#)

作者: 赵磊

博客: <http://elf8848.iteye.com>

## 目录

- 一、前言
- 二、spring mvc 核心类与接口
- 三、spring mvc 核心流程图
- 四、spring mvc DispatcherServlet 说明
- 五、spring mvc 父子上下文的说明
- 六、springMVC-mvc.xml 配置文件片段讲解
- 七、spring mvc 如何访问到静态的文件, 如 jpg, js, css
- 八、spring mvc 请求如何映射到具体的 Action 中的方法
- 九、spring mvc 中的拦截器:
- 十、spring mvc 如何使用拦截器

- 十一、 spring mvc 如何实现全局的异常处理
- 十二、 spring mvc 如何把全局异常记录到日志中
- 十三、 如何给 spring3 MVC 中的 Action 做 JUnit 单元测试
- 十四、 spring mvc 转发与重定向 （带参数重定向）
- 十五、 spring mvc 处理 ajax 请求
- 十六、 spring mvc 关于写几个配置文件的说明
- 十七、 spring mvc 如何取得 Spring 管理的 bean
- 十八、 spring mvc 多视图控制器
- 十九、 <mvc:annotation-driven /> 到底做了什么工作
- 二十、 本文中 springMVC.xml 配置文件是核心，这里给一个下载地址

## 一、前言：

为开发团队选择一款优秀的 MVC 框架是件难事儿，在众多可行的方案中决择需要很高的经验和水平。你的一个决定会影响团队未来的几年。要考虑方面太多：

- 1、简单易用，以提高开发效率。使小部分的精力在框架上，大部分的精力放在业务上。
- 2、性能优秀，这是一个最能吸引眼球的话题。
- 3、尽量使用大众的框架（避免使用小众的、私有的框架），新招聘来的开发人员有一些这方面技术积累，减低人员流动再适应的影响。

如果你还在为这件事发愁，本文最适合你了。选择 Spring MVC 吧。

Spring MVC 是当前最优秀的 MVC 框架，自从 Spring 2.5 版本发布后，由于支持注解配置，易用性有了大幅度的提高。Spring 3.0 更加完善，实现了对 Struts 2 的超越。现在越来越多的开发团队选择了 Spring MVC。

Struts2 也是非常优秀的 MVC 构架，优点非常多比如良好的结构，拦截器的思想，丰富的功能。但这里想说的是缺点，Struts2 由于采用了值栈、OGNL 表达式、struts2 标签库等，会导致应用的性能下降，应避免使用这些功能。而 Struts2 的多层拦截器、多实例 action 性能都很好。可以参考我写的一篇关于 Spring MVC 与 Struts2 与 Servlet 比较的文章 [《Struts2、SpringMVC、Servlet\(Jsp\)性能对比 测试》](#)

Spring3 MVC 的优点：

- 1、Spring3 MVC 使用简单，学习成本低。学习难度小于 Struts2，Struts2 用不上的多余功能太多。呵呵，当然这不是决定因素。
- 2、Spring3 MVC 很容易就可以写出性能优秀的程序，Struts2 要处处小心才可以写出性能优秀的程序（指 MVC 部分）
- 3、Spring3 MVC 的灵活是你无法想像的，Spring 框架的扩展性有口皆碑，Spring3 MVC 当然也不会落后，不会因使用了 MVC 框架而感到有任何的限制。

Struts2 的众多优点：

- 1、老牌的知名框架，从 Struts1 起积累了大量用户群体。技术文档丰富。
- 2、其它方面略... （呵呵，是不是不公平？）

Spring 的官方下载网址是：<http://www.springsource.org/download> （本文使用是的 Spring 3.0.5 版本）

转载请注明出处：原文地址：<http://elf8848.iteye.com/blog/875830>

## 二、核心类与接口：

先来了解一下，几个重要的接口与类。现在不知道他们是干什么的没关系，先混个脸熟，为以后认识他们打个基础。

DispatcherServlet    -- 前置控制器

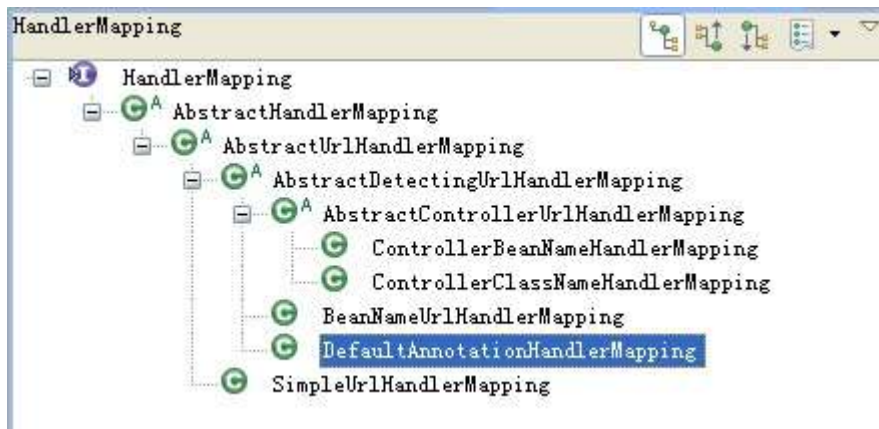


HandlerMapping 接口 -- 处理请求的映射

HandlerMapping 接口的实现类:

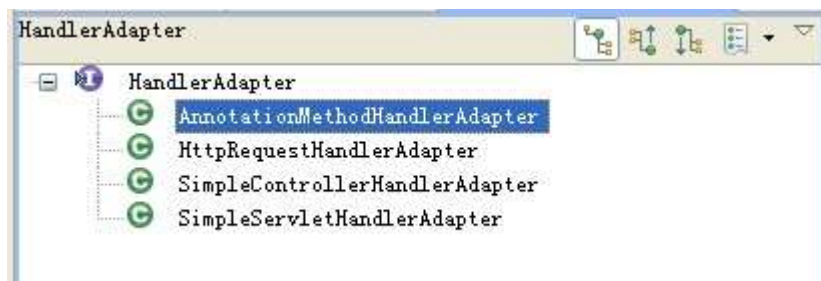
SimpleUrlHandlerMapping    通过配置文件，把一个 URL 映射到 Controller

DefaultAnnotationHandlerMapping    通过注解，把一个 URL 映射到 Controller 类上



HandlerAdapter 接口 -- 处理请求的映射

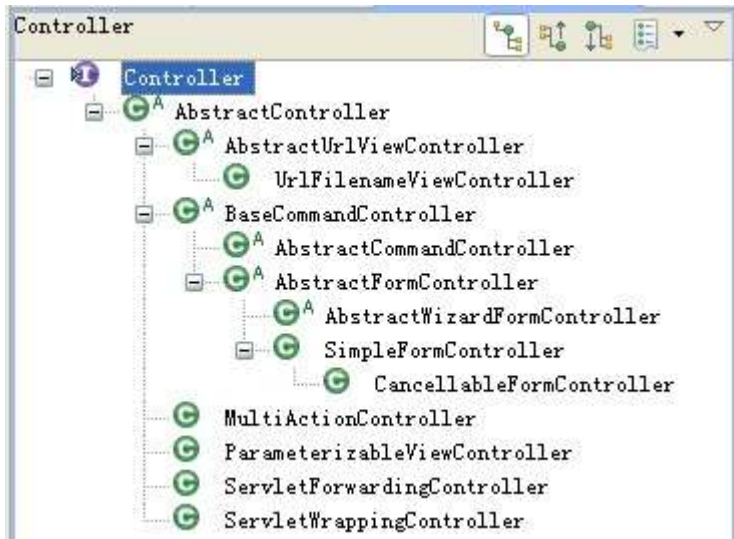
AnnotationMethodHandlerAdapter 类，通过注解，把一个 URL 映射到 Controller 类的方法上



Controller 接口 -- 控制器

由于我们使用了@Controller 注解，添加了@Controller 注解的类就可以担任控制器（Action）的职责，

所以我们并没有用到这个接口。



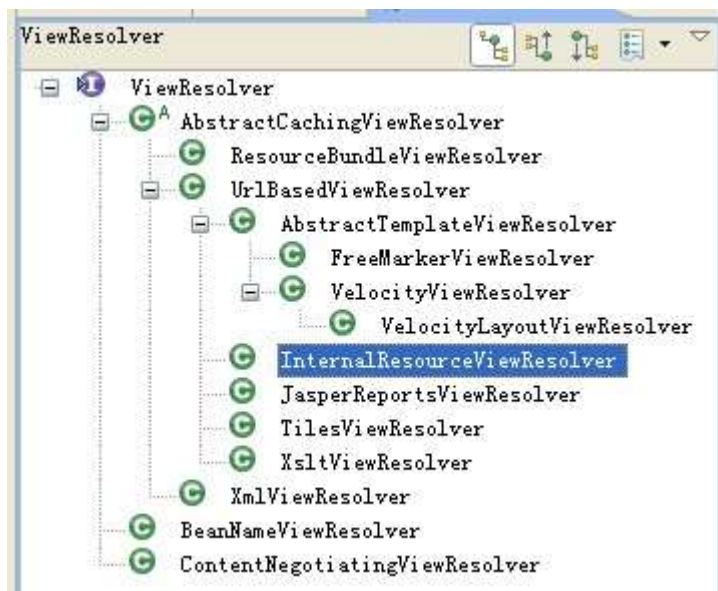
HandlerInterceptor 接口--拦截器

无图，我们自己实现这个接口，来完成拦截的器的工作。

ViewResolver 接口的实现类

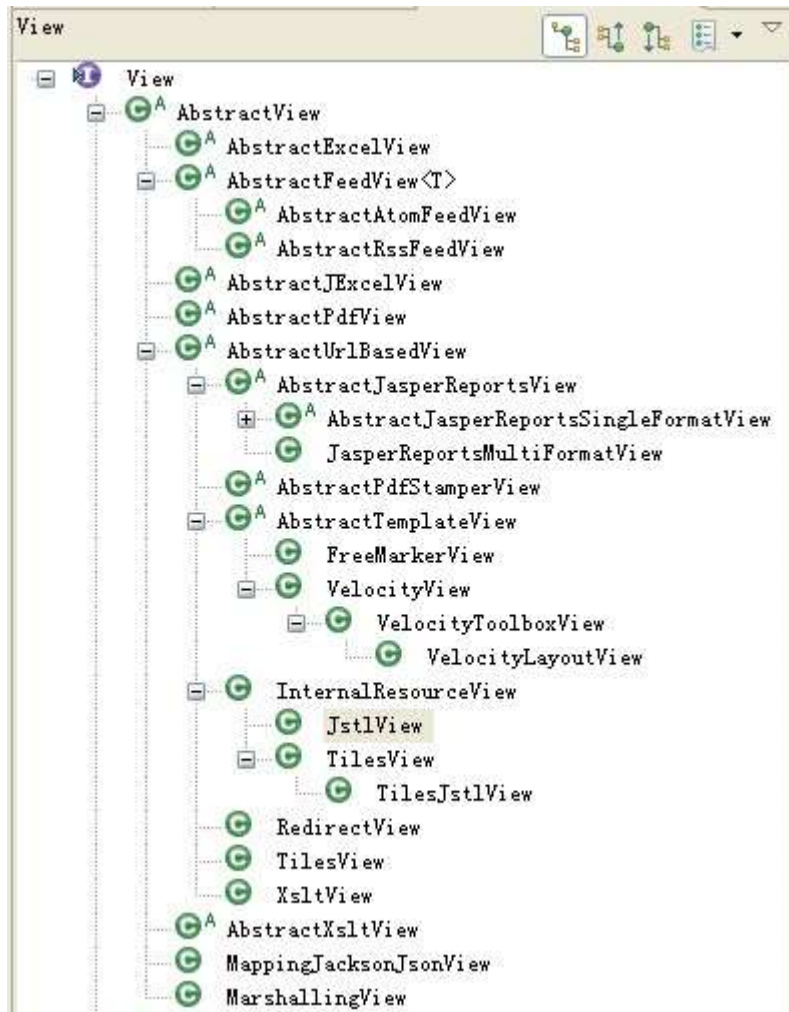
UrlBasedViewResolver 类 通过配置文件，把一个视图名交给到一个 View 来处理

InternalResourceViewResolver 类，比上面的类，加入了 JSTL 的支持



View 接口

JstlView 类



LocalResolver 接口





HandlerExceptionResolver 接口 --异常处理

SimpleMappingExceptionHandler 实现类

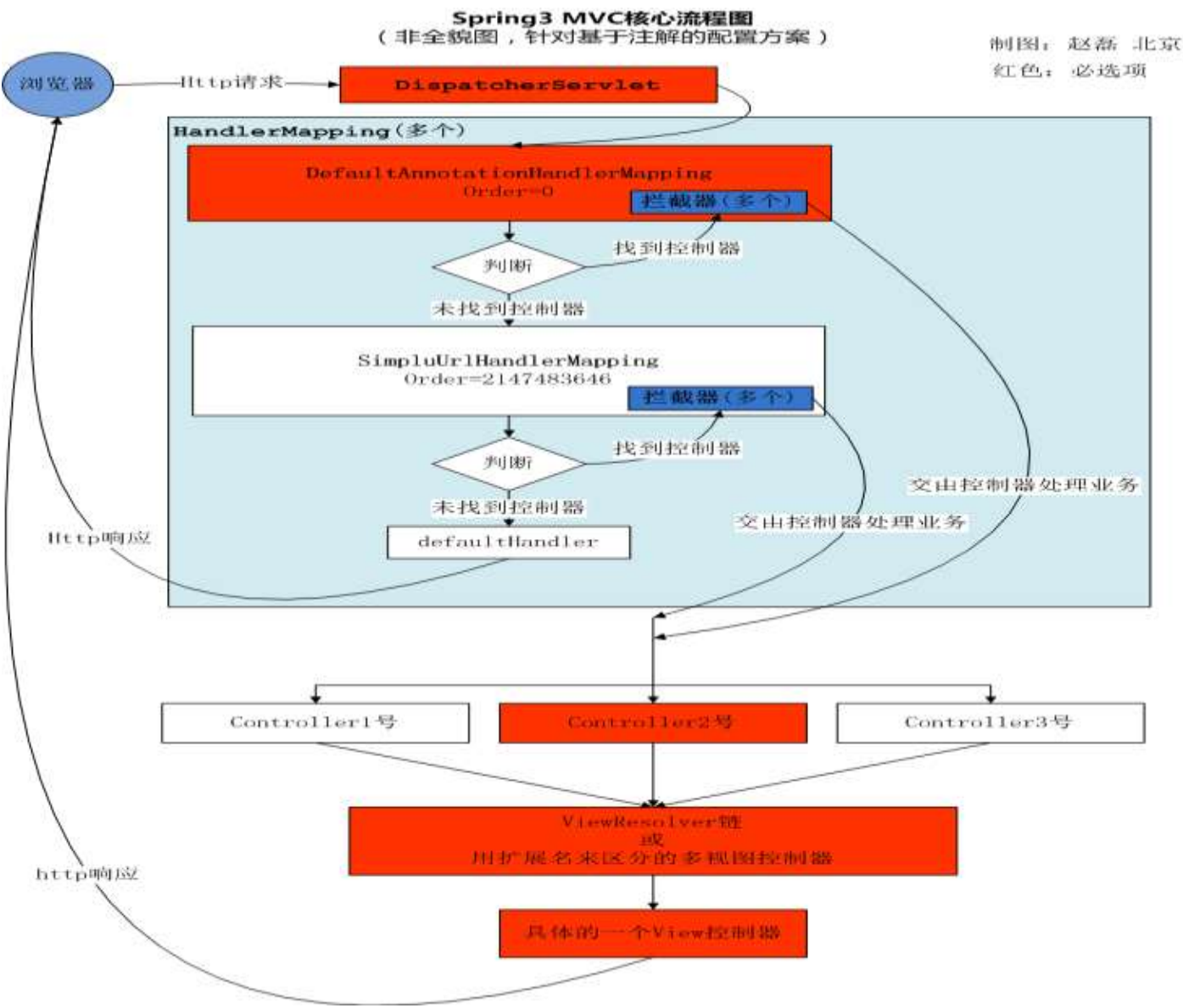


ModelAndView 类

无图。

三、核心流程图

本图是我个人画的，有不严谨的地方，大家对付看吧。总比没的看强。



转载请注明出处：本文地址：<http://elf8848.iteye.com/blog/875830>

## 四、DispatcherServlet 说明

使用 Spring MVC, 配置 DispatcherServlet 是第一步。

DispatcherServlet 是一个 Servlet, 所以可以配置多个 DispatcherServlet。

DispatcherServlet 是前置控制器, 配置在 web.xml 文件中的。拦截匹配的请求, Servlet 拦截匹配规则要自己定义, 把拦截下来的请求, 依据某某规则分发到目标 Controller(我们写的 Action) 来处理。

“某某规则”：是根据你使用了哪个 HandlerMapping 接口的实现类的不同而不同。

先来看第一个例子：

Xml 代码   

```
1. <web-app>
2.     <servlet>
3.         <servlet-name>example</servlet-name>
4.         <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
5.         <load-on-startup>1</load-on-startup>
6.     </servlet>
7.     <servlet-mapping>
8.         <servlet-name>example</servlet-name>
9.         <url-pattern>*.form</url-pattern>
10.    </servlet-mapping>
11.</web-app>
```

<load-on-startup>1</load-on-startup>是启动顺序，让这个 Servlet 随 Servletp 容器一起启动。

<url-pattern>\*.form</url-pattern> 会拦截\*.form 结尾的请求。

<servlet-name>example</servlet-name>这个 Servlet 的名字是 example，可以有多个 DispatcherServlet，是通过名字来区分的。每一个 DispatcherServlet 有自己的 WebApplicationContext 上下文对象。同时保存的 ServletContext 中和 Request 对象中，关于 key，以后说明。

在 DispatcherServlet 的初始化过程中，框架会在 web 应用的 WEB-INF 文件夹下寻找名为[servlet-name]-servlet.xml 的配置文件，生成文件中定义的 bean。

第二个例子：

Xml 代码   

```
1. <servlet>
2.     <servlet-name>springMVC</servlet-name>
3.     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
4.     <init-param>
5.         <param-name>contextConfigLocation</param-name>
6.         <param-value>classpath*:./springMVC.xml</param-value>
7.     </init-param>
8.     <load-on-startup>1</load-on-startup>
9. </servlet>
10. <servlet-mapping>
11.     <servlet-name>springMVC</servlet-name>
12.     <url-pattern>/</url-pattern>
```

### 13. </servlet-mapping>

指明了配置文件的文件名，不使用默认配置文件名，而使用 springMVC.xml 配置文件。

其中<param-value>\*\*.xml</param-value> 这里可以使用多种写法

- 1、不写, 使用默认值:/WEB-INF/<servlet-name>-servlet.xml
- 2、<param-value>/WEB-INF/classes/springMVC.xml</param-value>
- 3、<param-value>classpath\*:springMVC-mvc.xml</param-value>
- 4、多个值用逗号分隔

**Servlet 拦截匹配规则可以自己定义，拦截哪种 URL 合适？**

当映射为@RequestMapping("/user/add")时，为例：

- 1、拦截\*.do、\*.htm， 例如： /user/add.do

这是最传统的方式，最简单也最实用。不会导致静态文件（jpg, js, css）被拦截。

- 2、拦截/，例如： /user/add

可以实现现在很流行的 REST 风格。很多互联网类型的应用很喜欢这种风格的 URL。

弊端：会导致静态文件（jpg, js, css）被拦截后不能正常显示。想实现 REST 风格，事情就是麻烦一些。后面有解决办法还算简单。

- 3、拦截/\*，这是一个错误的方式，请求可以走到 Action 中，但转到 jsp 时再次被拦截，不能访问到 jsp。

转载请注明出处：本文地址：<http://elf8848.iteye.com/blog/875830>

## 五、父子上下文(WebApplicationContext)

如果你使用了 listener 监听器来加载配置，一般在 Struts+Spring+Hibernate 的项目中都是使用 listener 监听器的。如下

Java 代码   

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

Spring 会创建一个 WebApplicationContext 上下文，称为父上下文（父容器），保存在 ServletContext 中，key 是 WebApplicationContext.ROOT\_WEB\_APPLICATION\_CONTEXT\_ATTRIBUTE 的值。

可以使用 Spring 提供的工具类取出上下文对象：WebApplicationContextUtils.getWebApplicationContext(ServletContext);

DispatcherServlet 是一个 Servlet, 可以同时配置多个，每个 DispatcherServlet 有一个自己的上下文对象（WebApplicationContext），称为子上下文（子容器），子上下文可以访问父上下文中的内容，但父上下文不能访问子上下文中的内容。它也保存在 ServletContext 中，key 是 "org.springframework.web.servlet.FrameworkServlet.CONTEXT"+Servlet 名称。当一个 Request 对象产生时，会把这个子上下文对象（WebApplicationContext）保存在 Request 对象中，key 是 DispatcherServlet.class.getName() + ".CONTEXT"。

可以使用工具类取出上下文对象：RequestContextUtils.getWebApplicationContext(request);

说明：Spring 并没有限制我们，必须使用父子上下文。我们可以自己决定如何使用。

### 方案一，传统型：

父上下文容器中保存数据源、服务层、DAO 层、事务的 Bean。

子上下文容器中保存 Mvc 相关的 Action 的 Bean。

事务控制在服务层。

由于父上下文容器不能访问子上下文容器中内容，事务的 Bean 在父上下文容器中，无法访问子上下文容器中内容，就无法对子上下文容器中 Action 进行 AOP（事务）。

当然，做为“传统型”方案，也没有必要这要做。

### 方案二，激进型：

Java 世界的“面向接口编程”的思想是正确的，但在增删改查为主业务的系统里，Dao 层接口，Dao 层实现类，Service 层接口，Service 层实现类，Action 父类，Action。再加上众多的 0(vo\po\bo) 和 jsp 页面。写一个小功能 7、8 个类就写出来了。开发者说我就是想接点私活儿，和 PHP，ASP 抢抢饭碗，但我又是 Java 程序员。最好的结果是大项目能做好，小项目能做快。所以“激进型”方案就出现了——没有接口、没有 Service 层、还可以没有众多的 0(vo\po\bo)。那没有 Service 层事务控制在哪一层？只好上升的 Action 层。

本文不想说这是不是正确的思想，我想说的是 Spring 不会限制你这样做。

由于有了父子上下文，你将无法实现这一目标。解决方案是**只使用子上下文容器，不要父上下文容器**。所以数据源、服务层、DAO 层、事务的 Bean、Action 的 Bean 都放在子上下文容器中。就可以实现了，事务（注解事务）就正常工作了。这样才够激进。




总结：不使用 listener 监听器来加载 spring 的配置文件，只使用 DispatcherServlet 来加载 spring 的配置，不要父子上下文，只使用一个 DispatcherServlet，事情就简单了，什么麻烦事儿也没有了。

Java--大项目能做好--按传统方式做，规规矩矩的做，好扩展，好维护。

Java--小项目能做快--按激进方式做，一周时间就可以出一个版本，先上线接受市场(用户)的反馈，再改进，再反馈，时间就是生命(成本)。

转载请注明出处：原文地址：<http://elf8848.iteye.com/blog/875830>

## 六、springMVC-mvc.xml 配置文件片段讲解（未使用默认配置文件名）

Xml 代码   

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3.     xmlns="http://www.springframework.org/schema/beans"
4.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5.     xmlns:tx="http://www.springframework.org/schema/tx"
6.     xmlns:context="http://www.springframework.org/schema/context"
7.     xmlns:mvc="http://www.springframework.org/schema/mvc"
8.     xsi:schemaLocation="http://www.springframework.org/schema/beans
```



```
9.      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
10.     http://www.springframework.org/schema/tx
11.     http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
12.     http://www.springframework.org/schema/context
13.     http://www.springframework.org/schema/context/spring-context-3.0.xsd
14.     http://www.springframework.org/schema/mvc
15.     http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">
16.
17.
18.     <!-- 自动扫描的包名 -->
19.     <context:component-scan base-package="com.app,com.core,JUnit4" ></context:component-scan>
20.
21.     <!-- 默认注解映射的支持 -->
22.     <mvc:annotation-driven />
23.
24.     <!-- 视图解释类 -->
25.     <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
26.         <property name="prefix" value="/WEB-INF/jsp/" />
27.         <property name="suffix" value=".jsp"/><!--可为空,方便实现自己的依据扩展名来选择视图解释类的逻辑
编辑 -->
28.         <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />
29.     </bean>
30.
31.     <!-- 拦截器 -->
32.     <mvc:interceptors>
33.         <bean class="com.core.mvc.MyInteceptor" />
34.     </mvc:interceptors>
35.
36.     <!-- 对静态资源文件的访问 方案一 (二选一) -->
37.     <mvc:default-servlet-handler/>
38.
39.     <!-- 对静态资源文件的访问 方案二 (二选一) -->
40.     <mvc:resources mapping="/images/**" location="/images/" cache-period="31556926"/>
```

```
41.      <mvc:resources mapping="/js/**" location="/js/" cache-period="31556926"/>
42.      <mvc:resources mapping="/css/**" location="/css/" cache-period="31556926"/>
43.
44. </beans>
```

<context:component-scan/> 扫描指定的包中的类上的注解，常用的注解有：

@Controller 声明 Action 组件

@Service 声明 Service 组件            @Service("myMovieLister")

@Repository 声明 Dao 组件

@Component 泛指组件，当不好归类时。

@RequestMapping("/menu") 请求映射

@Resource 用于注入，( j2ee 提供的 ) 默认按名称装配，@Resource(name="beanName")

@Autowired 用于注入，(srping 提供的) 默认按类型装配

@Transactional( rollbackFor={Exception.class}) 事务管理

@ResponseBody

@Scope("prototype") 设定 bean 的作用域

<mvc:annotation-driven /> 是一种简写形式，完全可以手动配置替代这种简写形式，简写形式可以让初学都快速应用默认配置方案。

<mvc:annotation-driven /> 会自动注册 DefaultAnnotationHandlerMapping 与 AnnotationMethodHandlerAdapter 两个 bean, 是 spring MVC 为 @Controllers 分发请求所必须的。

并提供了:数据绑定支持,@NumberFormatannotation支持,@DateTimeFormat支持,@Valid支持,读写XML的支持(JAXB),读写JSON的支持(Jackson)。后面，我们处理响应 ajax 请求时，就使用到了对 json 的支持。

后面，对 action 写 JUnit 单元测试时，要从 spring IOC 容器中取 DefaultAnnotationHandlerMapping 与 AnnotationMethodHandlerAdapter 两个 bean，来完成测试，取的时候要知道是<mvc:annotation-driven />这一句注册的这两个 bean。

如何替换 <mvc:annotation-driven />? 他到底做了什么工作，请看，最后面的 十九节 <mvc:annotation-driven /> 到底做了什么工作。

`<mvc:interceptors/>` 是一种简写形式。通过看前面的大图，知道，我们可以配置多个 `HandlerMapping`。`<mvc:interceptors/>` 会为每一个 `HandlerMapping`，注入一个拦截器。其实我们也可以手动配置为每个 `HandlerMapping` 注入一个拦截器。

`<mvc:default-servlet-handler/>` 使用默认的 `Servlet` 来响应静态文件。

`<mvc:resources mapping="/images/**" location="/images/" cache-period="31556926"/>` 匹配 URL `/images/**` 的 URL 被当做静态资源，由 Spring 读出到内存中再响应 http。

转载请注明出处：本文地址：<http://elf8848.iteye.com/blog/875830>

## 七、如何访问到静态的文件，如 jpg, js, css?

如何你的 `DispatcherServlet` 拦截 `*.do` 这样的有后缀的 URL，就不存在访问不到静态资源的问题。

如果你的 `DispatcherServlet` 拦截 `/`，为了实现 REST 风格，拦截了所有的请求，那么同时对 `*.js`, `*.jpg` 等静态文件的访问也就被拦截了。

我们要解决这个问题。

目的：可以正常访问静态文件，不可以找不到静态文件报 404。

方案一：激活 Tomcat 的 `defaultServlet` 来处理静态文件

Xml 代码   

1. <servlet-mapping>
2.       <servlet-name>default</servlet-name>
3.       <url-pattern>\*.jpg</url-pattern>
4. </servlet-mapping>
5. <servlet-mapping>
6.       <servlet-name>default</servlet-name>
7.       <url-pattern>\*.js</url-pattern>
8. </servlet-mapping>
9. <servlet-mapping>
10.      <servlet-name>default</servlet-name>
11.      <url-pattern>\*.css</url-pattern>
12. </servlet-mapping>
13. 要配置多个，每种文件配置一个

要写在 DispatcherServlet 的前面， 让 defaultServlet 先拦截请求，这样请求就不会进入 Spring 了，我想性能是最好的吧。

Tomcat, Jetty, JBoss, and GlassFish 自带的默认 Servlet 的名字 -- "default"

Google App Engine 自带的 默认 Servlet 的名字 -- "\_ah\_default"

Resin 自带的 默认 Servlet 的名字 -- "resin-file"

WebLogic 自带的 默认 Servlet 的名字 -- "FileServlet"

WebSphere 自带的 默认 Servlet 的名字 -- "SimpleFileServlet"

方案二： 在 spring3.0.4 以后版本提供了 mvc:resources ，      使用方法：

Xml 代码   

1. `<!-- 对静态资源文件的访问 -->`
2. `<mvc:resources mapping="/images/**" location="/images/" />`

/images/\*\*映射到 ResourceHttpRequestHandler 进行处理, location 指定静态资源的位置. 可以是 web application 根目录下、jar 包里面, 这样可以把静态资源压缩到 jar 包中。cache-period 可以使得静态资源进行 web cache

如果出现下面的错误, 可能是没有配置`<mvc:annotation-driven />`的原因。

报错 WARNING: No mapping found for HTTP request with URI [/mvc/user/findUser/lisi/770] in DispatcherServlet with name 'springMVC'

使用`<mvc:resources/>`元素, 把 mapping 的 URI 注册到 SimpleUrlHandlerMapping 的 urlMap 中, key 为 mapping 的 URI pattern 值, 而 value 为 ResourceHttpRequestHandler, 这样就巧妙的把对静态资源的访问由 HandlerMapping 转到 ResourceHttpRequestHandler 处理并返回, 所以就支持 classpath 目录, jar 包内静态资源的访问。

另外需要注意的一点是, 不要对 SimpleUrlHandlerMapping 设置 defaultHandler. 因为对 static uri 的 defaultHandler 就是 ResourceHttpRequestHandler, 否则无法处理 static resources request.

方案三 , 使用`<mvc:default-servlet-handler/>`

Xml 代码   

1. `<mvc:default-servlet-handler/>`

会把"/\*\*" url, 注册到 SimpleUrlHandlerMapping 的 urlMap 中, 把对静态资源的访问由 HandlerMapping 转到 org.springframework.web.servlet.resource.DefaultServletHttpRequestHandler 处理并返回. DefaultServletHttpRequestHandler 使用就是各个 Servlet 容器自己的默认 Servlet.

补充说明：多个 HandlerMapping 的执行顺序问题：

DefaultAnnotationHandlerMapping 的 order 属性值是：0

< mvc:resources/> 自动注册的 SimpleUrlHandlerMapping 的 order 属性值是： 2147483646

<mvc:default-servlet-handler/>自动注册 的 SimpleUrlHandlerMapping 的 order 属性值是： 2147483647

spring 会先执行 order 值比较小的。当访问一个 a.jpg 图片文件时，先通过 DefaultAnnotationHandlerMapping 来找处理器，一定是找不到的，因为我们没有叫 a.jpg 的 Action。然后再按 order 值升序找，由于最后一个 SimpleUrlHandlerMapping 是匹配 “/\*\*”的，所以一定会匹配上，就可以响应图片。

*访问一个图片，还要走层层匹配。不知性能如何？*

最后再说明一下，方案二、方案三 在访问静态资源时，如果有匹配的(近似)总拦截器，就会走拦截器。如果你在拦截中实现权限检查，要注意过滤这些对静态文件的请求。

如何你的 DispatcherServlet 拦截 \*.do 这样的 URL 后缀，就不存上述问题了。还是有后缀方便。

转载请注明出处：原文地址：<http://elf8848.iteye.com/blog/875830>

## 八、请求如何映射到具体的 Action 中的方法？

方案一：基于 xml 配置映射，可以利用 SimpleUrlHandlerMapping、BeanNameUrlHandlerMapping 进行 Url 映射和拦截请求。配置方法略。

方案二：基于注解映射，可以使用 DefaultAnnotationHandlerMapping。

Xml 代码   

```
1. <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">    </bean>
```

但前面我们配置了<mvc:annotation-driven />，他会自动注册这个 bean, 就不须要我们显示的注册这个 bean 了。

如何替换 <mvc:annotation-driven />？他到底做了什么工作，请看，最后面的 十九节 <mvc:annotation-driven /> 到底做了什么工作。

以上都可以注入 interceptors，实现权限控制等前置工作。

我们使用第 2 种，基于注解来使用 spring MVC

并在 action 类上使用：  
@Controller  
@RequestMapping("/user")

转载请注明出处：原文地址：<http://elf8848.iteye.com/blog/875830>

## 九、Spring 中的拦截器：

Spring 为我们提供了：  
org.springframework.web.servlet.HandlerInterceptor 接口，

org.springframework.web.servlet.handler.HandlerInterceptorAdapter 适配器，  
实现这个接口或继承此类，可以非常方便的实现自己的拦截器。

有以下三个方法：

Action 之前执行：

```
public boolean preHandle(HttpServletRequest request,  
    HttpServletResponse response, Object handler);
```

生成视图之前执行

```
public void postHandle(HttpServletRequest request,  
    HttpServletResponse response, Object handler,  
    ModelAndView modelAndView);
```

最后执行，可用于释放资源

```
public void afterCompletion(HttpServletRequest request,  
    HttpServletResponse response, Object handler, Exception ex)
```

分别实现预处理、后处理（调用了 Service 并返回 ModelAndView，但未进行页面渲染）、返回处理（已经渲染了页面）  
在 preHandle 中，可以进行编码、安全控制等处理；



在 postHandle 中，有机会修改 ModelAndView;

在 afterCompletion 中，可以根据 ex 是否为 null 判断是否发生了异常，进行日志记录。

参数中的 Object handler 是下一个拦截器。

转载请注明出处：原文地址：<http://elf8848.iteye.com/blog/875830>

## 十、如何使用拦截器？

自定义一个拦截器，要实现 HandlerInterceptor 接口：

Java 代码   

```
1. public class MyInteceptor implements HandlerInterceptor {  
2.     略。。。   
3. }
```

Spring MVC 并没有总的拦截器，不能对所有的请求进行前后拦截。

Spring MVC 的拦截器，是属于 HandlerMapping 级别的，可以有多个 HandlerMapping，每个 HandlerMapping 可以有自己的拦截器。




当一个请求按 Order 值从小到大，顺序执行 HandlerMapping 接口的实现类时，哪一个先有返回，那就可以结束了，后面的 HandlerMapping 就不走了，本道工序就完成了。就转到下一道工序了。

拦截器会在什么时候执行呢？一个请求交给一个 HandlerMapping 时，这个 HandlerMapping 先找有没有处理器来处理这个请求，如何找到了，就执行拦截器，执行完拦截后，交给目标处理器。

如果没有找到处理器，那么这个拦截器就不会被执行。

在 spring MVC 的配置文件中配置有三种方法：

方案一，（近似）总拦截器，拦截所有 url

Java 代码   

```
1.      <mvc:interceptors>
2.          <bean class="com.app.mvc.MyInteceptor" />
3. </mvc:interceptors>
```

为什么叫“近似”，前面说了，Spring 没有总的拦截器。

<mvc:interceptors/>会为每一个 HandlerMapping，注入一个拦截器。总有一个 HandlerMapping 是可以找到处理器的，最多也只找到一个处理器，所以这个拦截器总会被执行的。起到了总拦截器的作用。

如果是 REST 风格的 URL，静态资源也会被拦截。

方案二，（近似）总拦截器，拦截匹配的 URL。

Xml 代码   

```
1. <mvc:interceptors >
2.     <mvc:interceptor>
3.         <mvc:mapping path="/user/*" /> <!-- /user/* -->
4.         <bean class="com.mvc.MyInteceptor"></bean>
5.     </mvc:interceptor>
6. </mvc:interceptors>
```

就是比 方案一多了一个 URL 匹配。

如果是 REST 风格的 URL，静态资源也会被拦截。

方案三, HandlerMappint 上的拦截器。

如果是 REST 风格的 URL, 静态资源就不会被拦截。因为我们精准的注入了拦截器。

Xml 代码   

```
1. <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
2.   <property name="interceptors">
3.     <list>
4.       <bean class="com.mvc.MyInteceptor"></bean>
5.     </list>
6.   </property>
7. </bean>
```

如果使用了<mvc:annotation-driven />, 它会自动注册DefaultAnnotationHandlerMapping 与 AnnotationMethodHandlerAdapter 这两个bean, 所以就没有机会再给它注入 interceptors 属性, 就无法指定拦截器。

当然我们可以通过人工配置上面的两个 Bean, 不使用 <mvc:annotation-driven />, 就可以 给 interceptors 属性 注入拦截器了。

其实我也不建议使用 <mvc:annotation-driven />, 而建议手动写详细的配置文件, 来替代 <mvc:annotation-driven />, 这就控制力就强了。

如何替换 <mvc:annotation-driven />? 他到底做了什么工作, 请看, 最后面的 十九节 <mvc:annotation-driven /> 到底做了什么工作。

转载请注明出处: 原文地址: <http://elf8848.iteye.com/blog/875830>

## 十一、如何实现全局的异常处理？

在 spring MVC 的配置文件中：

Xml 代码   

```
1. <!-- 总错误处理-->
2. <bean id="exceptionResolver" class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
3.     <property name="defaultErrorView">
4.         <value>/error/error</value>
5.     </property>
6.     <property name="defaultStatusCode">
7.         <value>500</value>
8.     </property>
9. <property name="warnLogCategory">
10.     <value>org.springframework.web.servlet.handler.SimpleMappingExceptionResolver</value>
11. </property>
12.</bean>
```

这里主要的类是 SimpleMappingExceptionResolver 类，和他的父类 AbstractHandlerExceptionResolver 类。

具体可以配置哪些属性，我是通过查看源码知道的。




你也可以实现 HandlerExceptionResolver 接口，写一个自己的异常处理程序。spring 的扩展性是很好的。

通过 SimpleMappingExceptionResolver 我们可以将不同的异常映射到不同的 jsp 页面（通过 exceptionMappings 属性的配置）。

同时我们也可以为所有的异常指定一个默认的异常提示页面（通过 `defaultErrorView` 属性的配置），如果所抛出的异常在 `exceptionMappings` 中没有对应的映射，则 Spring 将用此默认配置显示异常信息。

注意这里配置的异常显示界面均仅包括主文件名，至于文件路径和后缀已经在 `viewResolver` 中指定。如 `/error/error` 表示 `/error/error.jsp`

显示错误的 jsp 页面：

Html 代码   

```
1. <%@ page language="java" contentType="text/html; charset=GBK"
2.         pageEncoding="GBK"%>
3. <%@ page import="java.lang.Exception"%>
4. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
5. <html>
6. <head>
7. <meta http-equiv="Content-Type" content="text/html; charset=GBK">
8. <title>错误页面</title>
9. </head>
10. <body>
11. <h1>出错了</h1>
12. <%
13. Exception e = (Exception)request.getAttribute("exception");
14. out.print(e.getMessage());
15. %>
16. </body>
17. </html>
```

其中一句：`request.getAttribute("exception")`，key 是 `exception`，也是在 `SimpleMappingExceptionHandlerResolver` 类默认指定的，是可能通过配置文件修改这个值的，大家可以去看源码。

## 十二、如何把全局异常记录到日志中？

在前的配置中，其中有一个属性 `warnLogCategory`，值是“`SimpleMappingExceptionHandlerResolver` 类的全限定名”。我是在 `SimpleMappingExceptionHandlerResolver` 类父类 `AbstractHandlerExceptionHandlerResolver` 类中找到这个属性的。查看源码后得知：如果 `warnLogCategory` 不为空，spring 就会使用 apache 的 `org.apache.commons.logging.Log` 日志工具，记录这个异常，级别是 `warn`。

值：“`org.springframework.web.servlet.handler.SimpleMappingExceptionHandlerResolver`”，是“`SimpleMappingExceptionHandlerResolver` 类的全限定名”。这个值不是随便写的。因为我在 `log4j` 的配置文件中还要加入 `log4j.logger.org.springframework.web.servlet.handler.SimpleMappingExceptionHandlerResolver=WARN`，保证这个级别是 `warn` 的日志一定会被记录，即使 `log4j` 的根日志级别是 `ERROR`。




转载请注明出处：原文地址：<http://elf8848.iteye.com/blog/875830>

## 十三、如何给 spring3 MVC 中的 Action 做 JUnit 单元测试？

使用了 spring3 MVC 后，给 action 做单元测试变得很方便，我以前从来不给 action 写单元测试的，现在可以根据情况写一些了。

不用给每个 Action 都写单元测试吧，自己把握吧。

JUnitActionBase 类是所有 JUnit 的测试类的父类

Java 代码   

```
1. package test;
2. import javax.servlet.http.HttpServletRequest;
3. import javax.servlet.http.HttpServletResponse;
4. import org.junit.BeforeClass;
5. import org.springframework.mock.web.MockServletContext;
6. import org.springframework.web.context.WebApplicationContext;
7. import org.springframework.web.context.support.XmlWebApplicationContext;
8. import org.springframework.web.servlet.HandlerAdapter;
9. import org.springframework.web.servlet.HandlerExecutionChain;
10. import org.springframework.web.servlet.HandlerMapping;
11. import org.springframework.web.servlet.ModelAndView;
12. import org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter;
13. import org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping;
14. /**
15. * 说明： JUnit 测试 action 时使用的基类
16. *
17. * @author 赵磊
18. * @version 创建时间：2011-2-2 下午 10:27:03
19. */
20. public class JUnitActionBase {
21.     private static HandlerMapping handlerMapping;
22.     private static HandlerAdapter handlerAdapter;
23.     /**
24.      * 读取 spring3 MVC 配置文件
25.      */
```

```

26.         @BeforeClass
27.     public static void setUp() {
28.         if (handlerMapping == null) {
29.             String[] configs = { "file:src/springConfig/springMVC.xml" };
30.             XmlWebApplicationContext context = new XmlWebApplicationContext();
31.             context.setConfigLocations(configs);
32.             MockServletContext msc = new MockServletContext();
33.             context.setServletContext(msc);                context.refresh();
34.             msc.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE, context);
35.             handlerMapping = (HandlerMapping) context
36.                 .getBean(DefaultAnnotationHandlerMapping.class);
37.             handlerAdapter = (HandlerAdapter) context.getBean(context.getBeanNamesForType(AnnotationMethodH
andlerAdapter.class)[0]);
38.         }
39.     }
40.
41.     /**
42.      * 执行 request 对象请求的 action
43.      *
44.      * @param request
45.      * @param response
46.      * @return
47.      * @throws Exception
48.      */
49.     public ModelAndView excuteAction(HttpServletRequest request, HttpServletResponse response)
50.     throws Exception {
51.         HandlerExecutionChain chain = handlerMapping.getHandler(request);
52.         final ModelAndView model = handlerAdapter.handle(request, response,
53.             chain.getHandler());
54.         return model;
55.     }
56. }

```



这是个 JUnit 测试类，我们可以 new Request 对象，来参与测试，太方便了。给 request 指定访问的 URL，就可以请求目标 Action 了。

Java 代码   

```
1. package test.com.app.user;
2. import org.junit.Assert;
3. import org.junit.Test;
4. import org.springframework.mock.web.MockHttpServletRequest;
5. import org.springframework.mock.web.MockHttpServletResponse;
6. import org.springframework.web.servlet.ModelAndView;
7.
8. import test.JUnitActionBase;
9.
10. /**
11. * 说明： 测试 OrderAction 的例子
12. *
13. * @author 赵磊
14. * @version 创建时间：2011-2-2 下午 10:26:55
15. */
16.
17. public class TestOrderAction extends JUnitActionBase {
18.     @Test
19.     public void testAdd() throws Exception {
20.         MockHttpServletRequest request = new MockHttpServletRequest();
21.         MockHttpServletResponse response = new MockHttpServletResponse();
22.         request.setServletPath("/order/add");
23.         request.addParameter("id", "1002");
24.         request.addParameter("date", "2010-12-30");
```

```
25.         request.setMethod("POST");
26.         // 执行 URI 对应的 action
27.         final ModelAndView mav = this.excuteAction(request, response);
28.         // Assert logic
29.         Assert.assertEquals("order/add", mav.getViewName());
30.         String msg=(String)request.getAttribute("msg");
31.         System.out.println(msg);
32.     }
33. }
```

**需要说明一下**：由于当前最想版本的 Spring(Test) 3.0.5 还不支持@ContextConfiguration 的注解式 context file 注入，所以还需要写个 setUp 处理下，否则类似于 Tiles 的加载过程会有错误，因为没有 ServletContext。3.1 的版本应该有更好的解决方案，

参见：<https://jira.springsource.org/browse/SPR-5243>。

参考：<http://www.iteye.com/topic/828513>

## 十四、转发与重定向

可以通过 redirect/forward:url 方式转到另一个 Action 进行连续的处理。

可以通过 redirect:url 防止表单重复提交。

写法如下：




```
return "forward:/order/add";
```

```
return "redirect:/index.jsp";
```

转载请注明出处：原文地址：<http://elf8848.iteye.com/blog/875830>

## 带参数重定向—RedirectAttributes

用户保存或修改后，为了防止用户刷新浏览器(F5)导致表单重复提交，一般在保存或修改操作之后会 redirect 到一个结果页面（不是 forward），同时携带参数，如操作成功的提示信息。因为是 Redirect，Request 里的 attribute 不会传递过去。Spring 在 3.1 才提供了这个能力——RedirectAttributes。反复按 F5，操作成功的提示信息也不会再次出来（总共只出现一次），效果很理想。

Java 代码   

```
1. public String save(@ModelAttribute("group") Group group, RedirectAttributes redirectAttributes) {  
2.     accountManager.saveGroup(group);  
3.     redirectAttributes.addFlashAttribute("message", "操作成功");  
4.     return "redirect:/account/group/";  
5. }
```

## 十五、处理 ajax 请求

1、引入下面两个 jar 包，我用的是 1.7.2，好像 1.4.2 版本以上都可以，下载地址：<http://wiki.fasterxml.com/JacksonDownload>

jackson-core-asl-1.7.2.jar

jackson-mapper-asl-1.7.2.jar

2、spring 的配置文件中要有这一行，才能使用到 spring 内置支持的 json 转换。如果你手工把 POJO 转成 json 就可以不须要使用 spring 内置支持的 json 转换。

```
<mvc:annotation-driven />
```

3、使用@ResponseBody 注解

Java 代码   

```
1. /**
2.  * ajax 测试
3.  * http://127.0.0.1/mvc/order/ajax
4.  */
5.
6. @RequestMapping("/ajax")
7. @ResponseBody
8. public Object ajax(HttpServletRequest request){
9.     List<String> list=new ArrayList<String>();
10.    list.add("电视");
11.    list.add("洗衣机");
12.    list.add("冰箱");
13.    list.add("电脑");
```

```
14.         list.add("汽车");
15.         list.add("空调");
16.         list.add("自行车");
17.         list.add("饮水机");
18.         list.add("热水器");
19.         return list;
20. }
```

转载请注明出处：原文地址：<http://elf8848.iteye.com/blog/875830>

## 十六、关于写几个配置文件的说明

我看到有的人把配置文件写两份：

一个是原有的 applicationContext.xml，这个文件从 spring2.0-2.5 时一直在使用。

别一个是新加的 spring MVC 的配置文件。

其实这两个文件是可以写成一个文件的，springMVC 相关的配置，数据源，事务相关配置可以都写再一个配置文件中。

本例子中只使用了一个 spring 配置文件叫 “springMVC.xml”。

就不要再多配置一个 applicationContext.xml 文件了。

web.xml 文件中也不要再配置 org.springframework.web.context.ContextLoaderListener 的 listener 了。

写两个配置文件一般就会导致扫描两次，一定要精确控制扫描的包名，做到不重复扫描。

写两个配置文件还出现事务不好使的现象，是当把@Transactional 写有 Action 层时出现的。

是因为父子上下文的原因，请参看前的 第五节 父子上下文，里面有说明。原因是父上下文不能访问子上下文。

## 十七、如何取得 Spring 管理的 bean （请用第 3 种方法）

1、servlet 方式加载时，

【web.xml】

Xml 代码   

```
1. <servlet>
2. <servlet-name>springMVC</servlet-name>
3. <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
4. <init-param>
5. <param-name>contextConfigLocation</param-name>
6. <param-value>classpath*/springMVC.xml</param-value>
7. </init-param>
8. <load-on-startup>1</load-on-startup>
9. </servlet>
```

spring 容器放在 ServletContext 中的 key 是 org.springframework.web.servlet.FrameworkServlet.CONTEXT.springMVC  
注意后面的 springMVC，是你的 servlet-name 配置的值，注意适时修改。

Java 代码   

```
1. ServletContext sc=略
2. WebApplicationContext attr = (WebApplicationContext)sc.getAttribute("org.springframework.web.servlet.FrameworkServlet.CONTEXT.springMVC");
```




2、listener 方式加载时：

【web.xml】

Xml 代码   

```
1. <context-param>
2.     <param-name>contextConfigLocation</param-name>
3.     <param-value>/WEB-INF/applicationContext</param-value>
4. </context-param>
5.
6. <listener>
7.     <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
8. </listener>
```

【jsp/servlet】可以这样取得

Java 代码   

```
1. ServletContext context = getServletContext();
2. WebApplicationContext applicationContext = WebApplicationContextUtils .getWebApplicationContext(context);
```

3、通用的方法来了，神器啊，前的 1、2 两种方法并不通用，可以抛弃了。

在配置文件中加入：

Xml 代码   

```
1. <!-- 用于持有 ApplicationContext, 可以使用 SpringContextHolder.getBean('xxxx') 的静态方法得到 spring bean 对象 -->
2. <bean class="com.xxxxx.SpringContextHolder" lazy-init="false" />
```

Java 代码   

```
1. import org.springframework.context.ApplicationContext;
2. import org.springframework.context.ApplicationContextAware;
3. /**
4.  * 以静态变量保存 Spring ApplicationContext, 可在任何代码任何地方任何时候中取出 ApplicaitonContext.
5.  *
6.  */
7. public class SpringContextHolder implements ApplicationContextAware {
8. private static ApplicationContext applicationContext;
9.
10. /**
11. * 实现 ApplicationContextAware 接口的 context 注入函数, 将其存入静态变量.
12. */
13. public void setApplicationContext(ApplicationContext applicationContext) {
14. SpringContextHolder.applicationContext = applicationContext; // NOSONAR
15. }
16.
17. /**
18. * 取得存储在静态变量中的 ApplicationContext.
19. */
20. public static ApplicationContext getApplicationContext() {
21. checkApplicationContext();
22. return applicationContext;
23. }
24.
25. /**
26. * 从静态变量 ApplicationContext 中取得 Bean, 自动转型为所赋值对象的类型.
27. */
28. @SuppressWarnings("unchecked")
29. public static <T> T getBean(String name) {
30. checkApplicationContext();
31. return (T) applicationContext.getBean(name);
32. }
33.
```



```
34. /**
35. * 从静态变量 ApplicationContext 中取得 Bean, 自动转型为所赋值对象的类型.
36. */
37. @SuppressWarnings("unchecked")
38. public static <T> T getBean(Class<T> clazz) {
39. checkApplicationContext();
40. return (T) applicationContext.getBeansOfType(clazz);
41. }
42.
43. /**
44. * 清除 applicationContext 静态变量.
45. */
46. public static void cleanApplicationContext() {
47. applicationContext = null;
48. }
49.
50. private static void checkApplicationContext() {
51. if (applicationContext == null) {
52. throw new IllegalStateException("applicaitonContext 未注入, 请在 applicationContext.xml 中定义 SpringContextHolder");
53. }
54. }
55. }
```

转载请注明出处: 原文地址: <http://elf8848.iteye.com/blog/875830>

## 十八、多视图控制器

当有 jsp, flt (模板)等多种页面生成展示方式时, spring 默认使用的是“视图解析器链”。真是一个链, 所以性能不好, spring 会在“视图解析器链”中顺序的查找, 直到找到对应的“视图解析器”。jsp 视图解析器一定要写在最后面, 因为一旦调用 jsp, 就向浏览器发出数据了, Spring 就没有机会再尝试下一个了。

所以自己写一个“多视图解析器”, 依靠扩展名来区分, 可一次准确的选中一个 视图解析器, 提高性能 (会有多少提高呢? 没测试过)。

下面的例子支持 jsp, flt (模板)两种页面生成展示方式, 你中以自己添加, 支持更多。

Xml 代码   

```
1.      <!-- 多视图处理器 -->
2.      <bean class="com.xxx.core.web.MixedViewResolver">
3.          <property name="resolvers">
4.              <map>
5.                  <entry key="jsp">
6.                      <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
7.                          <property name="prefix" value="/WEB-INF/jsp/" />
8.                          <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"
9.                      </bean>
10.                  </entry>
11.                  <entry key="ftl">
12.                      <bean class="org.springframework.web.servlet.view.freemarker.FreeMarkerViewResolver">
13.                          <property name="cache" value="true" />
14.                          <property name="contentType" value="text/html; charset=UTF-8" />
15.                          <!-- 宏命令的支持 -->
16.                          <property name="exposeSpringMacroHelpers" value="true" />
17.                          <property name="viewClass" value="org.springframework.web.servlet.view.freemarke
18.                          r.FreeMarkerView" />
19.                          <property name="requestContextAttribute" value="rc" />
20.                      </bean>
21.                  </entry>
22.              </map>
23.          </property>
24.      </bean>
```

```

19.                </bean>
20.            </entry>
21.        </map>
22.    </property>
23.</bean>
24.
25.<!-- freemarker config -->
26.    <bean id="freeMarkerConfigurer" class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer">
27.
28.        <property name="templateLoaderPath" value="/WEB-INF/ftl/" />
29.        <property name="freemarkerSettings">
30.            <props>
31.                <prop key="template_update_delay">5</prop>
32.                <prop key="default_encoding">UTF-8</prop>
33.                <prop key="locale">zh_CN</prop>
34.            </props>
35.        </property>
36.    </bean>

```

Java 代码   

```

1. import java.util.Locale;
2. import java.util.Map;
3. import org.springframework.web.servlet.View;
4. import org.springframework.web.servlet.ViewResolver;
5.
6. /**
7.  * 说明： 多视图处理器
8.  */

```


```
9. * @author 赵磊
10.* @version 创建时间: 2011-8-19 上午 09:41:09
11.*/
12.public class MixedViewResolver implements ViewResolver{
13.    private Map<String,ViewResolver> resolvers;
14.
15.    public void setResolvers(Map<String, ViewResolver> resolvers) {
16.        this.resolvers = resolvers;
17.    }
18.
19.    public View resolveViewName(String viewName,Locale locale) throws Exception{
20.        int n=viewName.lastIndexOf(".");
21.        if(n!=-1){
22.            //取出扩展名
23.            String suffix=viewName.substring(n+1);
24.            //取出对应的 ViewResolver
25.            ViewResolver resolver=resolvers.get(suffix);
26.            if(resolver==null){
27.                throw new RuntimeException("No ViewResolver for "+suffix);
28.            }
29.            return resolver.resolveViewName(viewName, locale);
30.        }else{
31.            ViewResolver resolver=resolvers.get("jsp");
32.            return resolver.resolveViewName(viewName, locale);
33.        }
34.    }
35.}
```

转载请注明出处: 原文地址: <http://elf8848.iteye.com/blog/875830>

## 十九、<mvc:annotation-driven /> 到底做了什么工作

一句 <mvc:annotation-driven /> 实际做了以下工作：（不包括添加自己定义的拦截器）

我们了解这些之后，对 Spring3 MVC 的控制力就更强大了，想改哪就改哪里。

Xml 代码   

```
1.      <!-- 注解请求映射 -->
2.      <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
3.          <property name="interceptors">
4.              <list>
5.                  <ref bean="logNDCInteceptor"/>      <!-- 日志拦截器，这是你自定义的拦截器 -->
6.                  <ref bean="myRequestHelperInteceptor"/>      <!-- RequestHelper 拦截器，这是你自定义的拦截器 -->
7.                  <ref bean="myPermissionsInteceptor"/>      <!-- 权限拦截器，这是你自定义的拦截器-->
8.                  <ref bean="myUserInfoInteceptor"/>      <!-- 用户信息拦截器，这是你自定义的拦截器-->
9.              </list>
10.          </property>
11. </bean>
12. <bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
13.     <property name="messageConverters">
14.         <list>
15.             <ref bean="byteArray_hmc" />
16.             <ref bean="string_hmc" />
17.             <ref bean="resource_hmc" />
18.             <ref bean="source_hmc" />
19.             <ref bean="xmlAwareForm_hmc" />
20.             <ref bean="jaxb2RootElement_hmc" />
21.             <ref bean="jackson_hmc" />
22.         </list>
23.     </property>
```

```
24.</bean>
25.<bean id="byteArray_hmc" class="org.springframework.http.converter.ByteArrayHttpMessageConverter" /><!-- 处理.. -->
26.<bean id="string_hmc" class="org.springframework.http.converter.StringHttpMessageConverter" /><!-- 处理.. -->
27.<bean id="resource_hmc" class="org.springframework.http.converter.ResourceHttpMessageConverter" /><!-- 处理.. -->
28.<bean id="source_hmc" class="org.springframework.http.converter.xml.SourceHttpMessageConverter" /><!-- 处理.. -->
29.<bean id="xmlAwareForm_hmc" class="org.springframework.http.converter.xml.XmlAwareFormHttpMessageConverter" /><!-- 处理.. -->
30.<bean id="jaxb2RootElement_hmc" class="org.springframework.http.converter.xml.Jaxb2RootElementHttpMessageConverter" /><!-- 处理.. -->
31.<bean id="jackson_hmc" class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter" /><!-- 处理json-->
```

转载请注明出处：原文地址：<http://elf8848.iteye.com/blog/875830>

## 二十、本文中 springMVC.xml 配置文件是核心，这里给一个下载地址

要在 <http://www.iteye.com/> 网站有注册帐号才能下载(这不能怪我)

[Spring MVC 教程 快速入门 深入分析 V1.1.pdf](#)

[SpringMVC 核心配置文件示例.rar](#)

- [Spring MVC 教程 快速入门 深入分析 V1.1.pdf](#) (706.2 KB)
- 下载次数：11559
- [SpringMVC 核心配置文件示例.rar](#) (2.2 KB)
- 下载次数：6669
- [查看图片附件](#)