

Spring MVC 教程,快速入门,深入分析V1.1

作者: elf8848 <http://elf8848.iteye.com>

基于Spring 3.0的Spring MVC 教程，修正了1.0版的一些错误，作者：赵磊

目录

1. SPRING

1.1 Spring MVC 教程,快速入门,深入分析	3
-----------------------------------	---

1.1 Spring MVC 教程,快速入门,深入分析

发表时间: 2011-01-16 关键字: Spring MVC 教程, 快速入门

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 双亲上下文的说明
- 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
- 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

说明：本作者是文章的原创作者，转载请注明出处：本文地址：<http://elf8848.iteye.com/blog/875830>

有Q-q群共同交流，欢迎大家加入：34292021

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？）

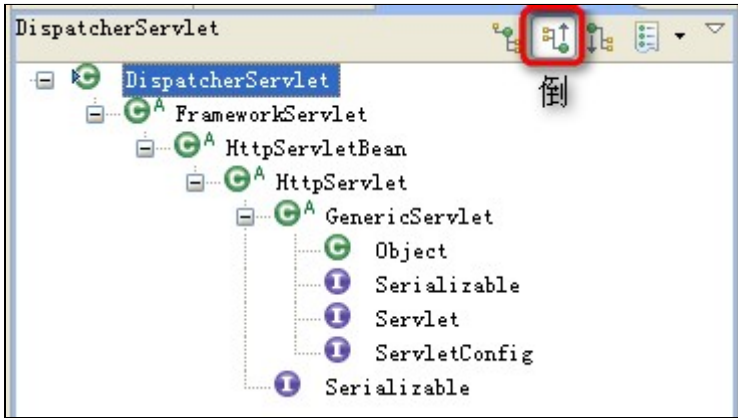
Spring的官方下载网址是：<http://www.springsource.org/download>（本文使用是的Spring 3.0.5版本）

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

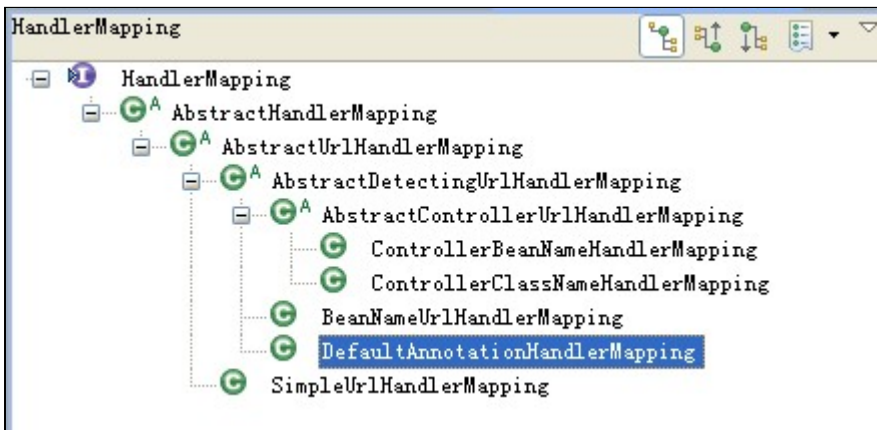


HandlerMapping接口 -- 处理请求的映射

HandlerMapping接口的实现类：

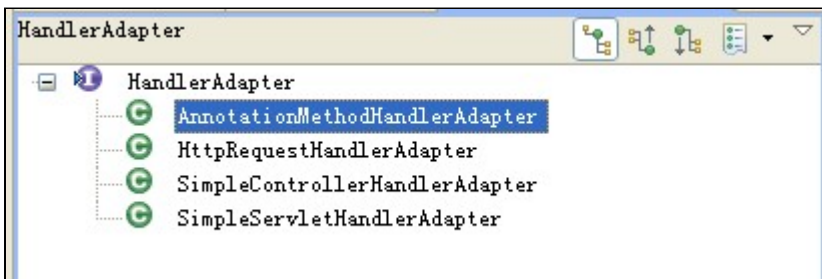
SimpleUrlHandlerMapping 通过配置文件，把一个URL映射到Controller

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



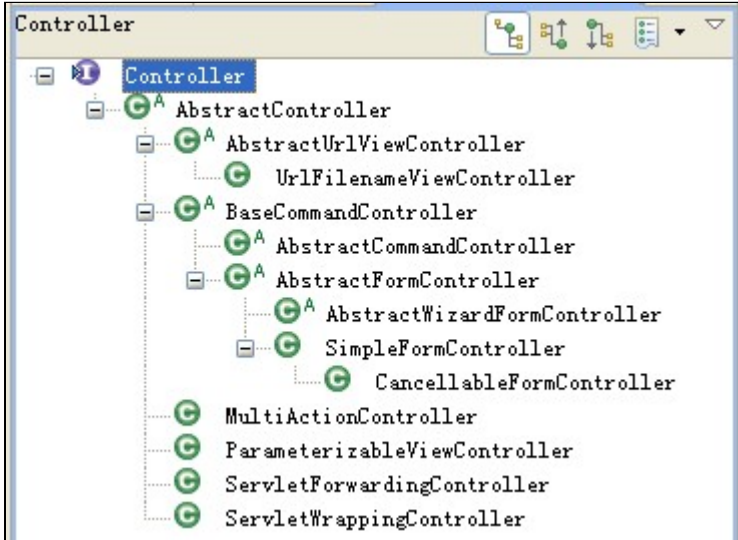
HandlerAdapter接口 -- 处理请求的映射

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。



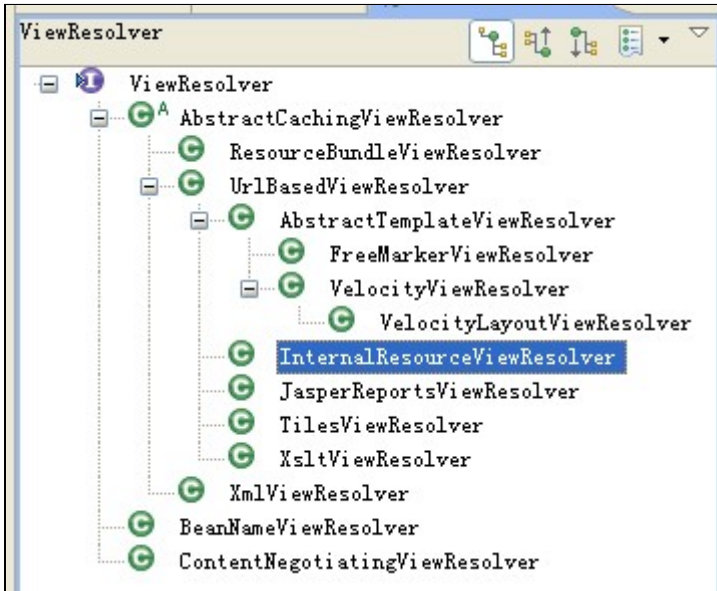
HandlerInterceptor 接口--拦截器

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

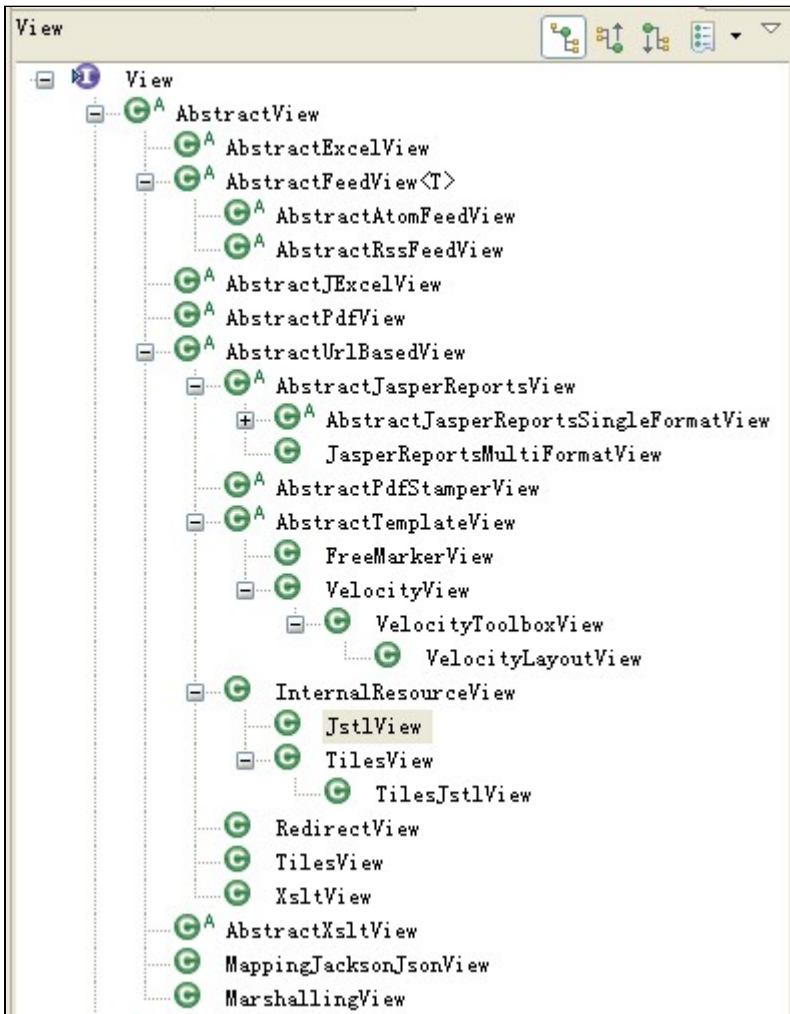
UrlBasedViewResolver类 通过配置文件，把一个视图名交给到一个View来处理

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



View接口

JstlView类



LocaleResolver接口



HandlerExceptionResolver接口 --异常处理

SimpleMappingExceptionHandler实现类



ModelAndView类

无图。

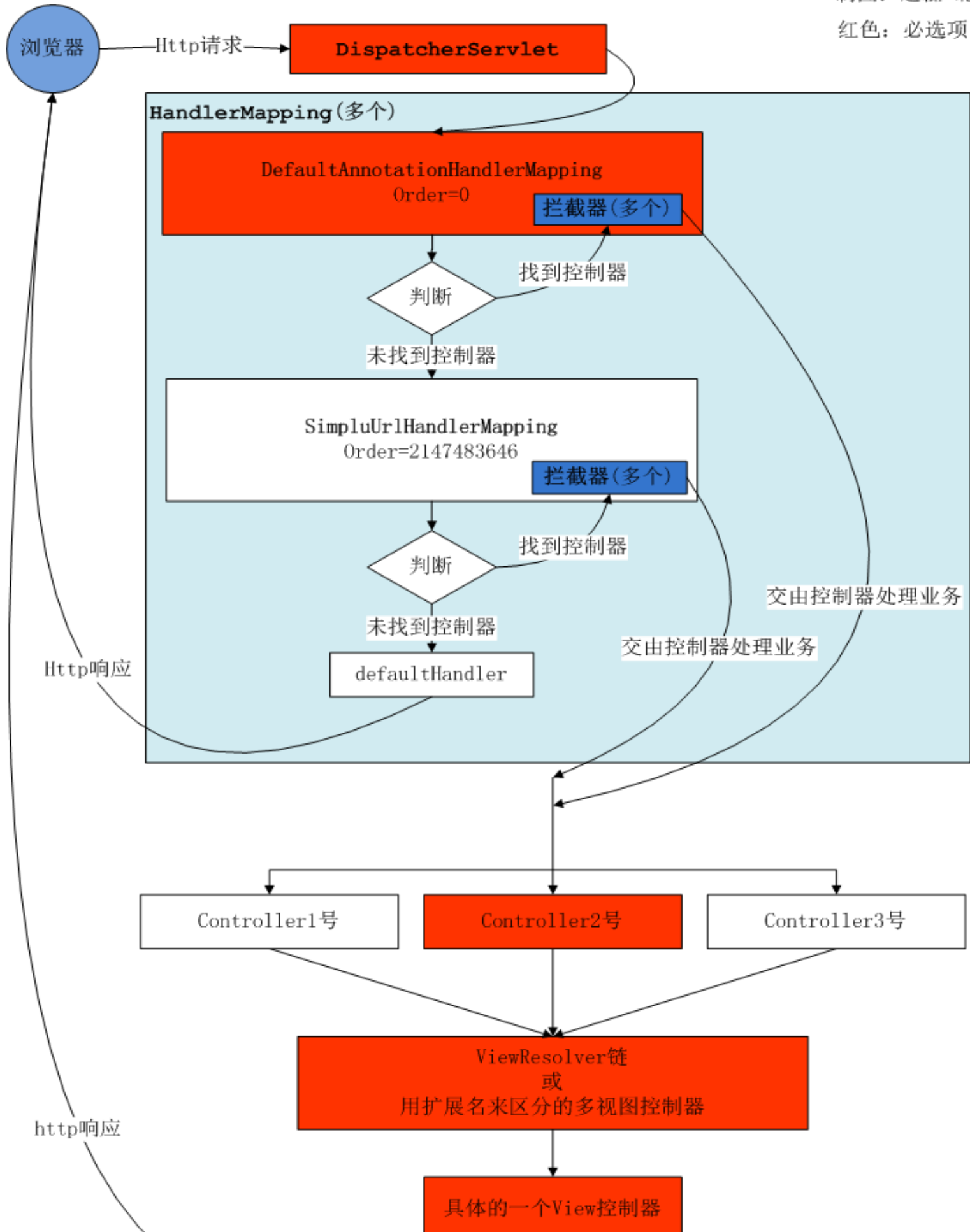
三、核心流程图

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。

Spring3 MVC核心流程图 (非全貌图, 针对基于注解的配置方案)

制图: 赵磊 北

红色: 必选项



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：

```
<web-app>
  <servlet>
    <servlet-name>example</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>example</servlet-name>
    <url-pattern>*.form</url-pattern>
  </servlet-mapping>
</web-app>
```

<load-on-startup>1</load-on-startup>是启动顺序，让这个Servlet随Servlet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：

```
<servlet>
    <servlet-name>springMVC</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath*/springMVC.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>springMVC</servlet-name>
    <url-pattern>/</url-pattern>
</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

```
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</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O(vo\po\bo)和jsp页面。写一个

小功能 7、8个类就写出来了。开发者说我不是做企业级开发的，我就是想和PHP，ASP抢抢市场，但我又是Java程序员。理想的目标是大项目能做好，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。哪没有Service层事务控制在哪一层？只好上升的Action层。

本文不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 version="1.0" encoding="UTF-8"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
```



```
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

<!-- 自动扫描的包名 -->
<context:component-scan base-package="com.app,com.core,JUnit4" ></context:component-scan>

<!-- 默认的注解映射的支持 -->
<mvc:annotation-driven />

<!-- 视图解释类 -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <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
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />
</bean>

<!-- 拦截器 -->
<mvc:interceptors>
    <bean class="com.core.mvc.MyInteceptor" />
</mvc:interceptors>

<!-- 对静态资源文件的访问 方案一 (二选一) -->
<mvc:default-servlet-handler/>

<!-- 对静态资源文件的访问 方案二 (二选一) -->
<mvc:resources mapping="/images/**" location="/images/" cache-period="31556926"/>
<mvc:resources mapping="/js/**" location="/js/" cache-period="31556926"/>
<mvc:resources mapping="/css/**" location="/css/" cache-period="31556926"/>

</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，(spring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

```
<servlet-mapping>
  <servlet-name>default</servlet-name>
  <url-pattern>*.jpg</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>default</servlet-name>
  <url-pattern>*.js</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>default</servlet-name>
  <url-pattern>*.css</url-pattern>
</servlet-mapping>
```

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：

```
<!-- 对静态资源文件的访问 -->
<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/>

```
<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。

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

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：

```
public class MyInteceptor implements HandlerInterceptor {
    略。。。
}
```

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

```
<mvc:interceptors>
    <bean class="com.app.mvc.MyInteceptor" />
</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。

```
<mvc:interceptors >
  <mvc:interceptor>
    <mvc:mapping path="/user/*" /> <!-- /user/* -->
    <bean class="com.mvc.MyInteceptor"></bean>
  </mvc:interceptor>
</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。

```
<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
  <property name="interceptors">
    <list>
      <bean class="com.mvc.MyInteceptor"></bean>
    </list>
  </property>
</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：

```
<!-- 总错误处理-->
<bean id="exceptionResolver" class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
    <property name="defaultErrorView">
        <value>/error/error</value>
    </property>
    <property name="defaultStatusCode">
        <value>500</value>
    </property>
    <property name="warnLogCategory">
        <value>org.springframework.web.servlet.handler.SimpleMappingExceptionResolver</value>
    </property>
</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：

```
<%@ page language="java" contentType="text/html; charset=GBK"
    pageEncoding="GBK"%>
<%@ page import="java.lang.Exception"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loos
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=GBK">
<title>错误页面</title>
</head>
<body>
<h1>出错了</h1>
<%
Exception e = (Exception)request.getAttribute("exception");
out.print(e.getMessage());
%>
</body>
</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.SimpleMappingExceptionResolver”，是“SimpleMappingExceptionResolver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.SimpleMappingExceptionResolver=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

```
package test;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.junit.BeforeClass;
import org.springframework.mock.web.MockServletContext;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.support.XmlWebApplicationContext;
import org.springframework.web.servlet.HandlerAdapter;
import org.springframework.web.servlet.HandlerExecutionChain;
import org.springframework.web.servlet.HandlerMapping;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter;
import org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping;
/**
 * 说明： JUnit测试action时使用的基类
 *

```

```
* @author 赵磊
* @version 创建时间：2011-2-2 下午10:27:03
*/

public class JUnitActionBase {
    private static HandlerMapping handlerMapping;
    private static HandlerAdapter handlerAdapter;
    /**
     * 读取spring3 MVC配置文件
     */
    @BeforeClass
    public static void setUp() {
        if (handlerMapping == null) {
            String[] configs = { "file:src/springConfig/springMVC.xml" };
            XmlWebApplicationContext context = new XmlWebApplicationContext();
            context.setConfigLocations(configs);
            MockServletContext msc = new MockServletContext();
            context.setServletContext(msc);                context.refresh();
            msc.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE, context);
            handlerMapping = (HandlerMapping) context
                .getBean(DefaultAnnotationHandlerMapping.class);
            handlerAdapter = (HandlerAdapter) context.getBean(context.getBeanNamesForType(HandlerAdapter.class));
        }
    }

    /**
     * 执行request对象请求的action
     *
     * @param request
     * @param response
     * @return
     * @throws Exception
     */
    public ModelAndView excuteAction(HttpServletRequest request, HttpServletResponse response)
        throws Exception {
        HandlerExecutionChain chain = handlerMapping.getHandler(request);
        final ModelAndView model = handlerAdapter.handle(request, response,
            chain.getHandler());
    }
}
```

```
        return model;
    }
}
```

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。

```
package test.com.app.user;
import org.junit.Assert;
import org.junit.Test;
import org.springframework.mock.web.MockHttpServletRequest;
import org.springframework.mock.web.MockHttpServletResponse;
import org.springframework.web.servlet.ModelAndView;

import test.JUnitActionBase;

/**
 * 说明：测试OrderAction的例子
 *
 * @author 赵磊
 * @version 创建时间：2011-2-2 下午10:26:55
 */

public class TestOrderAction extends JUnitActionBase {
    @Test
    public void testAdd() throws Exception {
        MockHttpServletRequest request = new MockHttpServletRequest();
        MockHttpServletResponse response = new MockHttpServletResponse();
        request.setServletPath("/order/add");
        request.addParameter("id", "1002");
        request.addParameter("date", "2010-12-30");
        request.setMethod("POST");
    }
}
```

```
        // 执行URI对应的action
        final ModelAndView mav = this.executeAction(request, response);
        // Assert logic
        Assert.assertEquals("order/add", mav.getViewName());
        String msg=(String)request.getAttribute("msg");
        System.out.println(msg);
    }
}
```

需要说明一下：由于当前最稳定版本的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>

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

```
/**
 * ajax测试
 * http://127.0.0.1/mvc/order/ajax
 */

@RequestMapping("/ajax")
@ResponseBody
public Object ajax(HttpServletRequest request){
    List<String> list=new ArrayList<String>();
    list.add("电视");
    list.add("洗衣机");
    list.add("冰箱");
    list.add("电脑");
    list.add("汽车");
    list.add("空调");
    list.add("自行车");
    list.add("饮水机");
}
```



```
list.add("热水器");  
return list;  
}
```

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spring会扫描两次。

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

如果写两个配置文件，先执行springMVC.xml（xxx-servlet.xml），后执行applicationContext.xml。

先执行springMVC.xml（xxx-servlet.xml）时，由于数据源、事务都配置在applicationContext.xml中，还没有执行（未准备好），所以导致了上面的问题。

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】

```
<servlet>
<servlet-name>springMVC</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<init-param>
<param-name>contextConfigLocation</param-name>
<param-value>classpath*/springMVC.xml</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</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。

```
ServletContext sc=略
WebApplicationContext attr = (WebApplicationContext)sc.getAttribute("org.springframework.web.se
```

2、listener方式加载时：

【web.xml】

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext</param-value>
</context-param>

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

【jsp/servlet】可以这样取得

```
ServletContext context = getServletContext();
WebApplicationContext applicationContext = WebApplicationContextUtils .getWebApplicationContex
```

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：

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

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
/**
 * 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.
 *
 */
public class SpringContextHolder implements ApplicationContextAware {
    private static ApplicationContext applicationContext;

    /**
     * 实现ApplicationContextAware接口的context注入函数,将其存入静态变量.
     */
    public void setApplicationContext(ApplicationContext applicationContext) {
        SpringContextHolder.applicationContext = applicationContext; // NOSONAR
    }

    /**
     * 取得存储在静态变量中的ApplicationContext.
     */
    public static ApplicationContext getApplicationContext() {
        checkApplicationContext();
        return applicationContext;
    }

    /**
     * 从静态变量ApplicationContext中取得Bean,自动转型为所赋值对象的类型.
     */
    @SuppressWarnings("unchecked")
    public static <T> T getBean(String name) {
```

```
checkApplicationContext();
return (T) applicationContext.getBean(name);
}

/**
 * 从静态变量ApplicationContext中取得Bean, 自动转型为所赋值对象的类型.
 */
@SuppressWarnings("unchecked")
public static <T> T getBean(Class<T> clazz) {
checkApplicationContext();
return (T) applicationContext.getBeansOfType(clazz);
}

/**
 * 清除applicationContext静态变量.
 */
public static void cleanApplicationContext() {
applicationContext = null;
}

private static void checkApplicationContext() {
if (applicationContext == null) {
throw new IllegalStateException("applicaitonContext未注入,请在applicationContext.xml中定义Spring
}
}
}
```

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,ftl (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,ftl (模板)两种页面生成展示方式，你中以自己添加，支持更多。

```
<!-- 多视图处理器 -->
<bean class="com.xxx.core.web.MixedViewResolver">
    <property name="resolvers">
        <map>
            <entry key="jsp">
                <bean class="org.springframework.web.servlet.view.Inter
                    <property name="prefix" value="/WEB-INF/jsp/" />
                    <property name="viewClass" value="org.springfra
                </bean>
            </entry>
            <entry key="ftl">
                <bean class="org.springframework.web.servlet.view.freem
                    <property name="cache" value="true" />
                    <property name="contentType" value="text/html;ch
                    <!-- 宏命令的支持 -->
                    <property name="exposeSpringMacroHelpers" value
                    <property name="viewClass" value="org.springfra
                    <property name="requestContextAttribute" value=
                </bean>
            </entry>
        </map>
    </property>
</bean>

<!-- freemarker config -->
<bean id="freeMarkerConfigurer" class="org.springframework.web.servlet.view.freemarker.Free
```

```
<property name="templateLoaderPath" value="/WEB-INF/ftl/" />
<property name="freemarkerSettings">
  <props>
    <prop key="template_update_delay">5</prop>
    <prop key="default_encoding">UTF-8</prop>
    <prop key="locale">zh_CN</prop>
  </props>
</property>
</bean>
```

```
import java.util.Locale;
import java.util.Map;
import org.springframework.web.servlet.View;
import org.springframework.web.servlet.ViewResolver;

/**
 * 说明：多视图处理器
 *
 * @author 赵磊
 * @version 创建时间：2011-8-19 上午09:41:09
 */
public class MixedViewResolver implements ViewResolver{
    private Map<String,ViewResolver> resolvers;

    public void setResolvers(Map<String, ViewResolver> resolvers) {
        this.resolvers = resolvers;
    }

    public View resolveViewName(String viewName,Locale locale) throws Exception{
        int n=viewName.lastIndexOf(".");
        if(n!=-1){
            //取出扩展名
            String suffix=viewName.substring(n+1);
```

```
        //取出对应的ViewResolver
        ViewResolver resolver=resolvers.get(suffix);
        if(resolver==null){
            throw new RuntimeException("No ViewResolver for "+suffix);
        }
        return  resolver.resolveViewName(viewName, locale);
    }else{
        ViewResolver resolver=resolvers.get("jsp");
        return  resolver.resolveViewName(viewName, locale);
    }
}
}
```

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。

```
<!-- 注解请求映射 -->
<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"
    <property name="interceptors">
        <list>
            <ref bean="logNDCInteceptor"/> <!-- 日志拦截器，这是你自定义的拦截器 -->
            <ref bean="myRequestHelperInteceptor"/> <!-- RequestHelper拦截器，这是
            <ref bean="myPermissionsInteceptor"/> <!-- 权限拦截器，这是你自定义的拦截
            <ref bean="myUserInfoInteceptor"/> <!-- 用户信息拦截器，这是你自定义的拦截
        </list>
    </property>
</bean>
```

```
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter"
    <property name="messageConverters">
        <list>
            <ref bean="byteArray_hmc" />
            <ref bean="string_hmc" />
            <ref bean="resource_hmc" />
            <ref bean="source_hmc" />
            <ref bean="xmlAwareForm_hmc" />
            <ref bean="jaxb2RootElement_hmc" />
            <ref bean="jackson_hmc" />
        </list>
    </property>
</bean>
<bean id="byteArray_hmc" class="org.springframework.http.converter.ByteArrayHttpMessageConverter" />
<bean id="string_hmc" class="org.springframework.http.converter.StringHttpMessageConverter" />
<bean id="resource_hmc" class="org.springframework.http.converter.ResourceHttpMessageConverter" />
<bean id="source_hmc" class="org.springframework.http.converter.xml.SourceHttpMessageConverter" />
<bean id="xmlAwareForm_hmc" class="org.springframework.http.converter.xml.XmlAwareFormHttpMessageConverter" />
<bean id="jaxb2RootElement_hmc" class="org.springframework.http.converter.xml.Jaxb2RootElementHttpMessageConverter" />
<bean id="jackson_hmc" class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter" />
```

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

<http://dl.iteye.com/topics/download/f7c02d0b-7c8f-3124-aaea-9b66a620e550>

附件下载:

- [sprintMVC.rar \(2.2 KB\)](#)
- dl.iteye.com/topics/download/f7c02d0b-7c8f-3124-aaea-9b66a620e550