# 官网地址

https://docs.spring.io/spring/docs/4.3.20.RELEASE/spring-framework-reference/htmlsingle/#mvc

# 常见的MVC框架比较

## （1）运行性能上

JSP+servlet>struts1>springMVC>struts2+freemarker>struts2,ognl,值栈。

## （2）开发效率

struts2,ognl,值栈>struts2+freemarker>struts2>springMVC>struts1>JSP+servlet

说明：spring mvc开发效率和struts2不相上下。Struts2的性能低的原因是因为OGNL和值栈造成的。所以，如果你的系统并发量高，可以使用freemaker进行显示，而不是采用OGNL和值栈。这样在性能上会有相当大得提高。

# Spring MVC特性

Spring Web Flow（SWF）旨在成为Web应用程序页面流管理的最佳解决方案, SWF在Servlet和Portlet环境中与Spring MVC和JSF等现有框架集成。 如果您有一个业务流程（或流程）可以从会话模型中受益而不是纯粹的请求模型，那么SWF可能是解决方案SWF允许您将逻辑页面流捕获为可在不同情况下重用的自包含模块，因此非常适合构建Web应用程序模块，以引导用户完成驱动业务流程的受控导航。

Spring的Web模块包含许多独特的Web支持功能：

（1）明确分离角色。每个角色--controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver等都可以由专用对象来完成。

（2）像JavaBeans一样强大而直接地配置框架和应用程序类。此配置功能包括跨上下文轻松引用，例如从Web控制器到业务对象和验证器。

（3）适应性，非侵入性和灵活性。定义您需要的任何控制器方法签名，可能使用给定方案的参数注释之一（例如@RequestParam，@ RequestHeader，@ PathVariable等）。

（4）可重复使用的业务代码，无需重复。 将现有业务对象用作命令或表单对象，而不是镜像它们以扩展特定的框架基类。

（5）可定制的绑定和验证。 键入不匹配作为应用程序级验证错误，保留违规值，本地化日期和数字绑定等，而不是仅包含字符串的表单对象，具有手动解析和转换为业务对象。

（6）可定制的处理程序映射和视图解析。 处理程序映射和视图解析策略的范围从简单的基于URL的配置到复杂的，专用的解析策略。 Spring比授权特定技术的Web MVC框架更灵活。

（7）灵活的模型转移。 具有名称/值Map的模型传输支持与任何视图技术的轻松集成

（8）可自定义的区域设置，时区和主题解析，支持带或不带Spring标记库的JSP，支持JSTL，支持Velocity而无需额外的桥接等等。

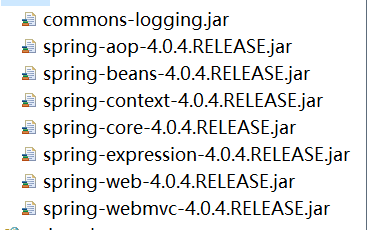
（9）一个简单但功能强大的JSP标记库，称为Spring标记库，为数据绑定和主题等功能提供支持。 自定义标记允许在标记代码方面具有最大的灵活性。

（10）Spring 2.0中引入的JSP表单标记库，使得在JSP页面中编写表单变得更加容易。

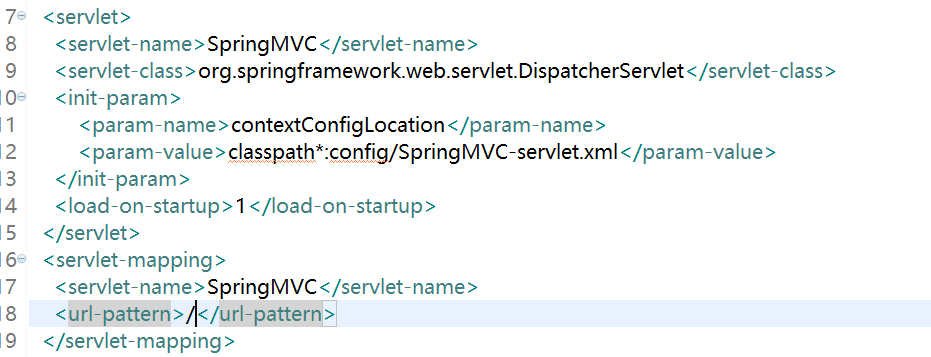
（11）生命周期范围限定为当前HTTP请求或HTTP会话的Bean。 这不是Spring MVC本身的特定功能，而是Spring MVC使用的WebApplicationContext容器。

# 环境搭建（重点）

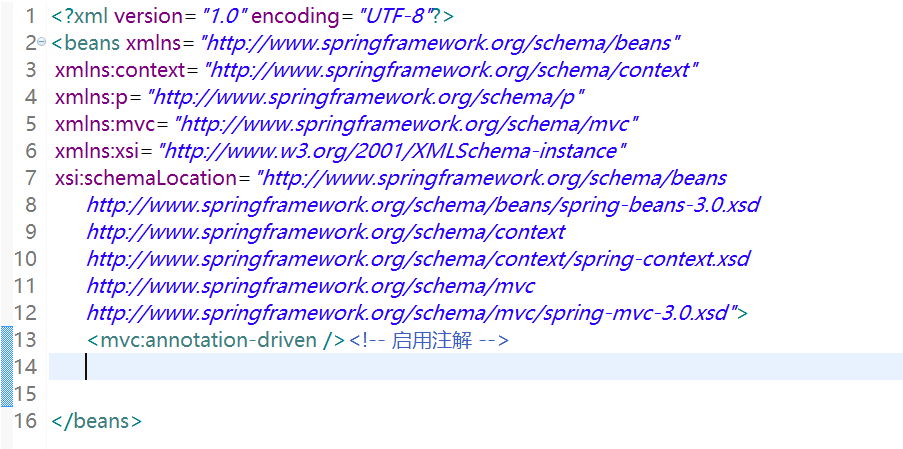
## （1）加载jar包



## （2）在web.xml文件中配置servlet



## （3）创建Spring MVC的配置文件SpringMVC-servlet.xml



<mvc:annotation-driven />

说明：SpringMVC会帮我们自动做一些注册组件之类的事情。

## （4）创建Controller和View

# Spring MVC基础（4.x版本）

## 基本概念

MVC：Model+View+Controller（数据模型+视图+控制器）

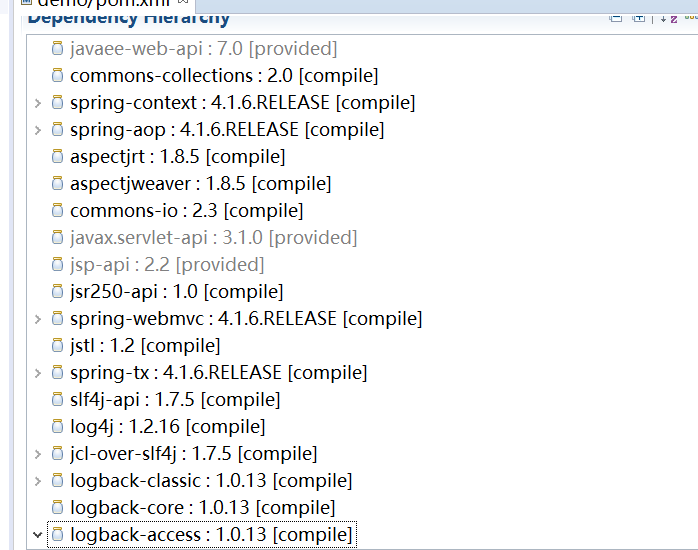
三层架构：展现层+应用层+数据访问层

实际上MVC只存在三层架构的展现层，M实际上是数据模型，是包含数据的对象，在Spring MVC里，有一个专门的类叫Model，用来和V之间的数据交互传值，V指的是视图页面，包含JSP、freeMarker、Velocity、Thymeleaf、Tile等，C当然就是控制器（Spring MVC的注解@Controller的类）。而三层架构是整个应用的架构，是由Spring框架负责管理的，一般项目结构中都有Service层、DAO层，这两个反馈在应用层和数据访问层。

Spring MVC使我们可以简单的开发灵活且松耦合的Web项目，我们主要关注基于注解和Java配置的零配置（无xml文件配置）的Spring MVC开发。

## Spring MVC项目快速搭建

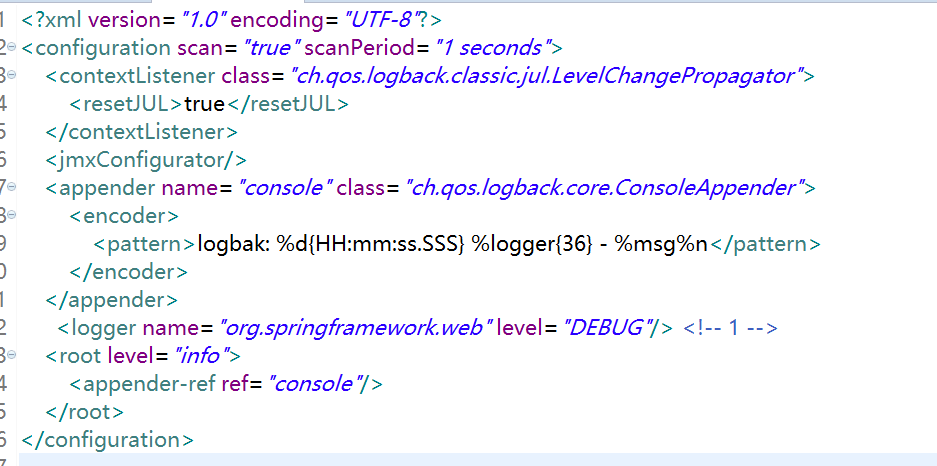
### 1、首先修改pom.xml文件



目的：配置这些jar包。

### 2、日志配置（这个不是必须的）

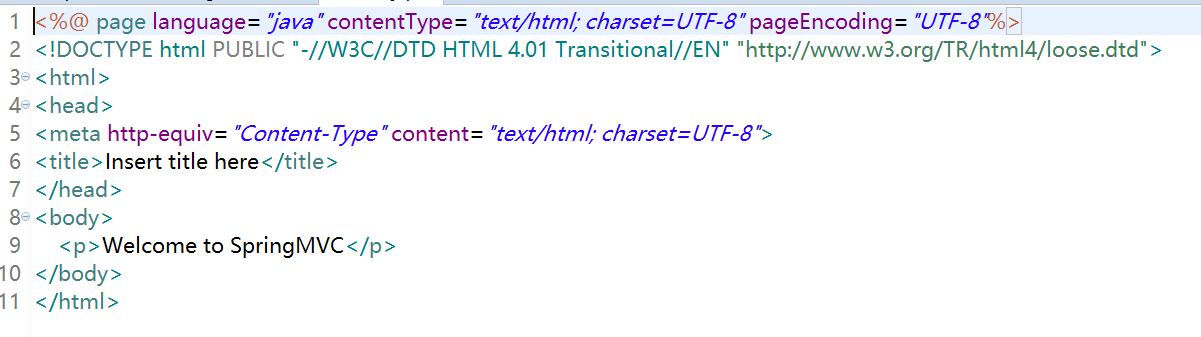
在src/main/resources目录下，新建logback.xml用来配置日志，如下：



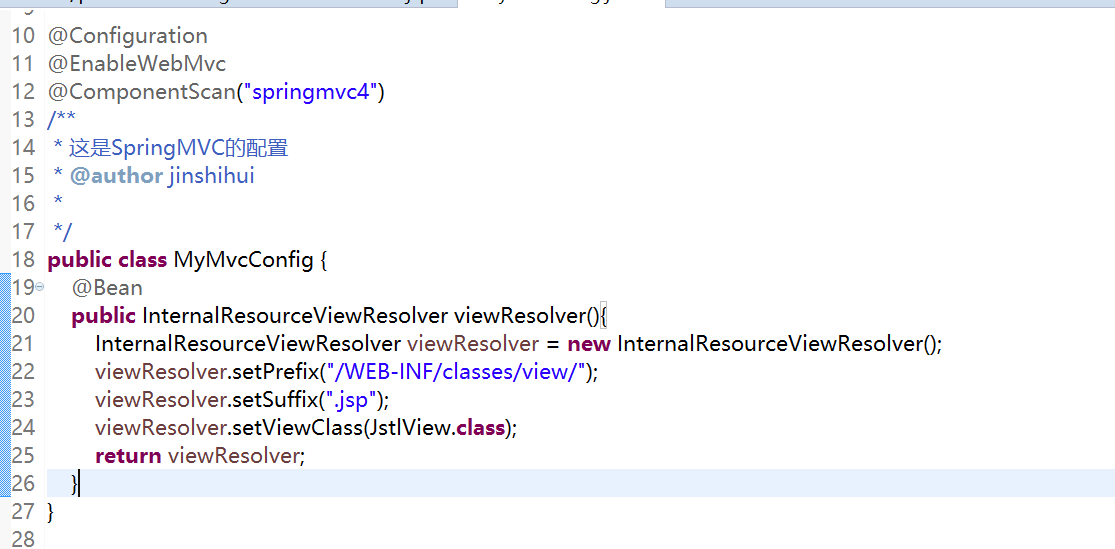
说明：将org.springframework.web包下的类的日志级别设置为DEBUG，我们开发Spring MVC经常出现和参数类型相关的4XX错误，设置此项我们会看到更详细的错误信息。

### 3、编写页面index.jsp

在src/main/resources下新建views目录，并在此目录下新建index.jsp页面。



### 4、编写Spring MVC配置类（重点）



说明：

1、@EnableWebMvc注解会开启一些默认配置，如一些ViewResolver或者MessageConverter等。

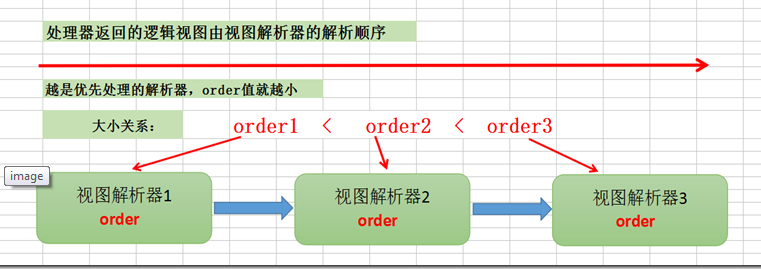
2、Spring MVC的ViewResolver，这是Spring MVC视图渲染的核心机制。SpringMVC里有一个接口叫做ViewResolver，实现这个接口要重写resolveViewName()方法，这个方法的返回值是接口View。View的职责就是使用model、request、response对象，并将渲染的视图（不一定是html，可能是json、xml、pdf）返回给浏览器。

3、InternalResourceViewResolver这个类是最常用的视图解析器类， springmvc在处理器方法中通常返回的是逻辑视图，通过视图解析器定位到真正的页面。

#### 视图解析器工作原理

当处理器返回逻辑视图时（也就是return “string”），要经过视图解析器链，前面的解析器能处理的，就不会继续往下传播。如果不能处理就要沿着解析器链继续寻找，直到找到合适的视图解析器（概括为：能解析的，不继续往下找，不能解析的，要继续往下找解析器）。如图：

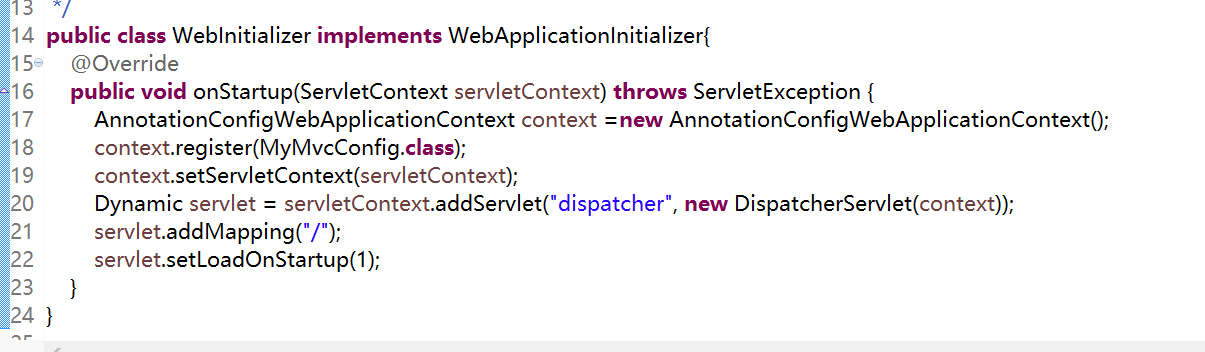
假如我们有几个视图解析器：视图解析器1、视图解析器2、视图解析器3：



当处理器（@controller）返回逻辑视图解析过程：

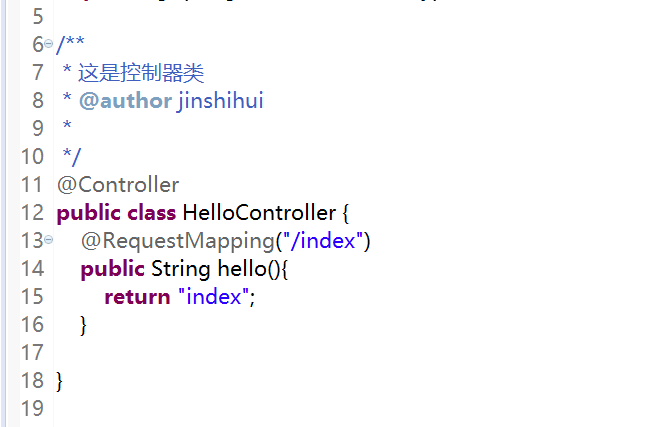
当经过视图解析器1时，如果能解析就解析而且不会再继续往下。如果不能执行就返回null，这样下面的解析器才能处理。但是对于解析器InternalResourceViewResolver来说，不管能不能解析它都不会返回null，也就是说它拦截了所有的逻辑视图，让后续的解析器得不到执行，所以InternalResourceViewResolver必须放在最后。记住一点：不能解析就返回null，这样后续解析器才能解析。

### 5、编写Web配置类（重点，相当于web.xml文件）



说明：WebApplicationInitializer是Spring提供用来配置Servlet3.0+配置的接口，从而实现了替代web.xml文件的位置，实现这个接口将会自动被SpringServletContainerInitializer类（用来启动Servlet3.0容器）获取到。新建AnnotationConfigWebApplicationContext类，注册配置类，并将其和当前ServletContext关联，注册Spring MVC的DispatcherServlet。

### 6、编写控制器类

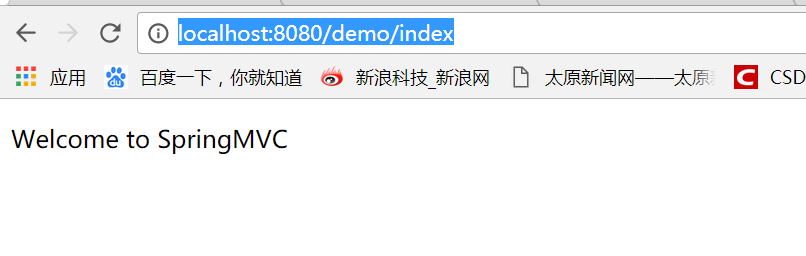


说明：@Controller注解声明是一个控制器，利用@RequestMapping注解配置URL和方法之间的映射。通过上面ViewResolver的Bean配置，返回值是index,说明我们的页面放置的路径为/WEB-INF/classes/view/index.jsp。

### 7、运行

启动服务器，在浏览器中输入：http://localhost:8080/demo/index

如图：



返回index.jsp页面

## Spring MVC常用注解

### （1）@Controller

@Controller注解在类上，说明这个类是Spring MVC里的Controller，将其声明为Spring的一个Bean，DispatcherServlet会自动扫描注解了这个注解的类，并将web请求映射到@RequestMapping的方法上。这里需要指出的是在声明普通Bean的时候，使用@Component、@Service、@Repository和@Controller是等同的，因为@Service、@Repository和@Controller都组合了@Component元注解，但是在Spring MVC声明控制器Bean的时候，只能使用@Controller。

说明：这个注解在类上使用。

### （2）@RequestMapping

@RequestMapping注解是用来映射Web请求（访问路径和参数）、处理类和方法的。@RequestMapping可注解在类或方法上，注解在方法上的@RequestMapping路径会继承注解在类上的路径，@RequestMapping支持Servlet的request和response作为参数，也支持对request和response的媒体类型进行配置。

说明：这个注解在**类和方法上**使用。

#### 属性

value：用于将指定请求的实际地址映射到方法上。

name：给映射地址指定一个别名。

method：请求的方法类型，包括GET、POST、HEAD、OPTIONS、PUT、PATCH、DELETE、TRACE。

consumes：指定处理请求的提交内容类型(Content-Type),例如application/json、text/html等。

produces：指定返回的内容类型，返回的内容类型必须是request请求头(Accept)中所包含的类型。

params：指定request中必须包含某些参数值时，才让该方法处理。

headers：指定request中必须包含某些指定的header值，才能让该方法处理请求。

### （3）@ResponseBody

@ResponseBody支持将返回值放在response体内，而不是返回一个页面，我们在很多基于Ajax程序的时候，可以以此注解返回数据而不是页面，此注解可放置在返回值前或者方法上。

说明：这个注解在**返回值前或方法上**。

### （4）@RequestBody

@RequestBody允许request的参数在request体中，而不是直接链接在地址后面，此注解放置在参数前。

说明：这个注解在**参数前**使用。

### （5）@PathVariable

@PathVariable用来接收路径参数，如/news/001,可接收001作为参数，此注解放置在参数前。

说明：这个注解在**参数前**使用。

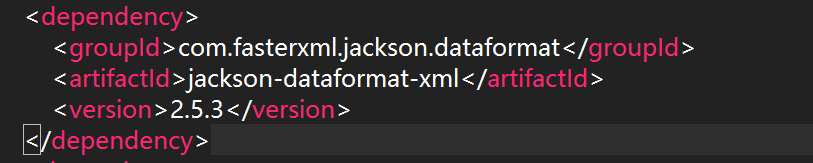
### （6）@RestController

@RestController是一个组合注解，组合了@Controller和@ResponseBody，这就意味着当你只开发一个和页面交互数据控制的时候，需要使用此注解。若没有此注解，要想实现上述功能，则需自己在代码中加@Controller和@ResponseBody两个注解。

说明：这个注解在类上使用。

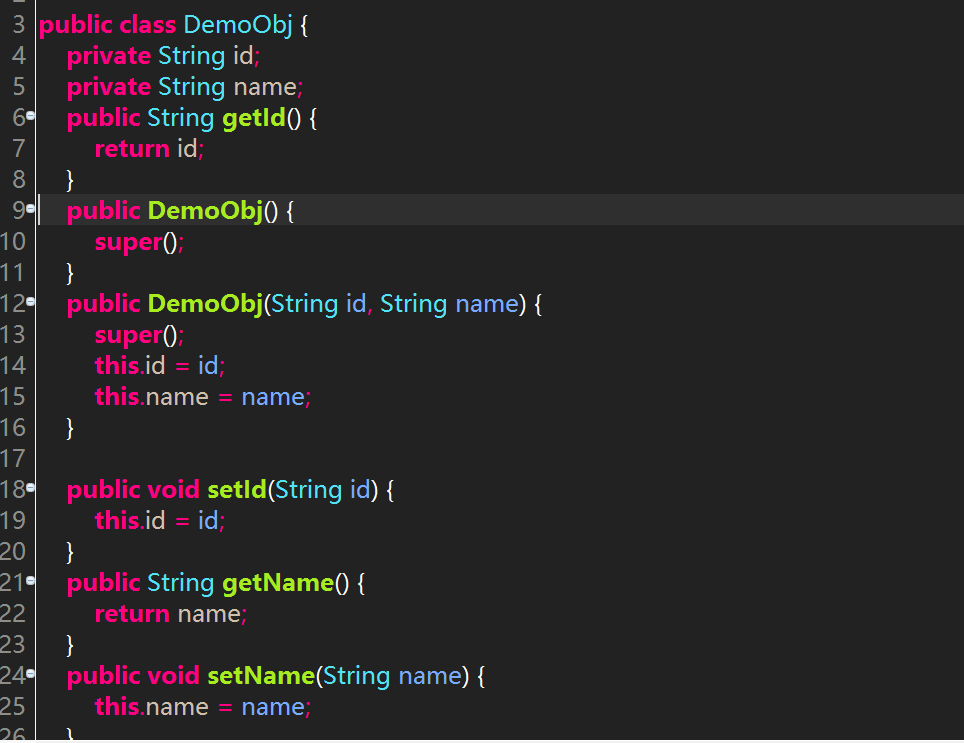
### 代码示例

（1）添加jackson及其相关依赖，获得对象和json或xml之间的转换。



说明：在实际项目开发中，我们主要支持json数据格式，没有必要同时支持json和xml，因为json比xml简洁，由于js的广泛使用，json成为最推荐的格式。

（2）编写一个Java Bean

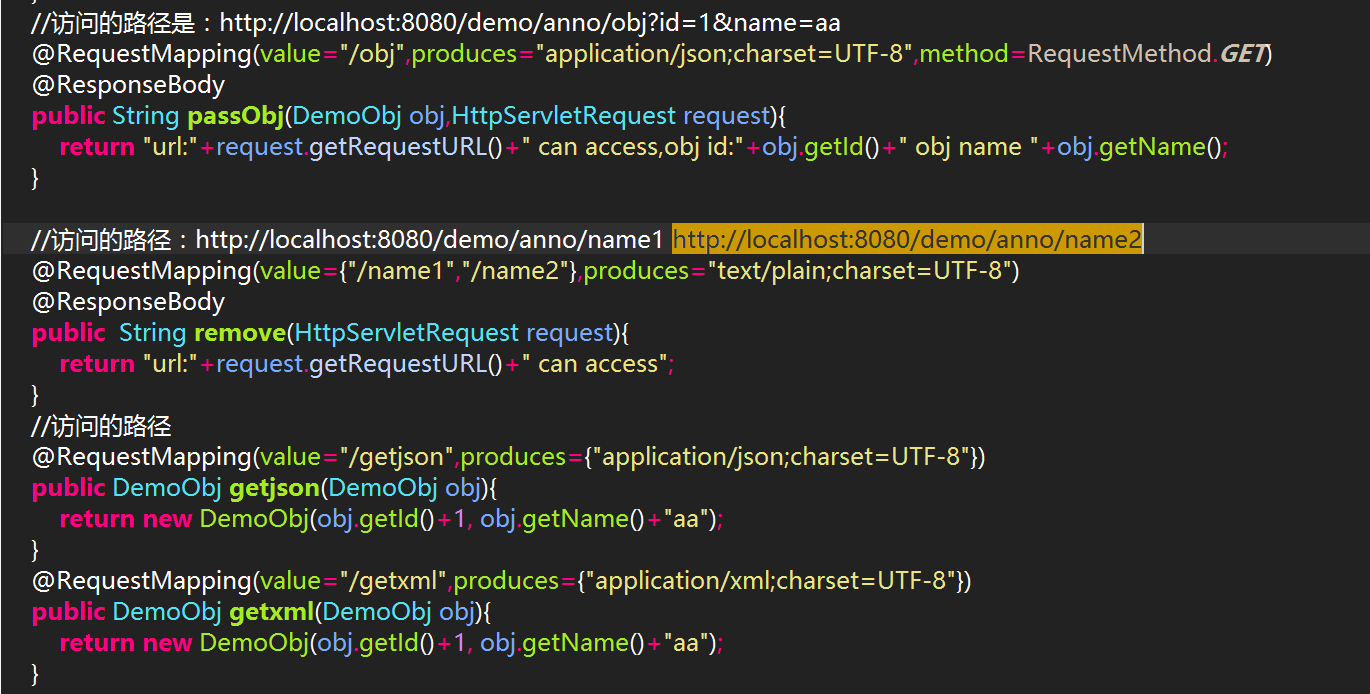


说明：这个类是用来获取request对象参数和返回此对象到response中。

jackson对对象和json做转换时一定需要空的构造函数。

（3）配置控制器Controller





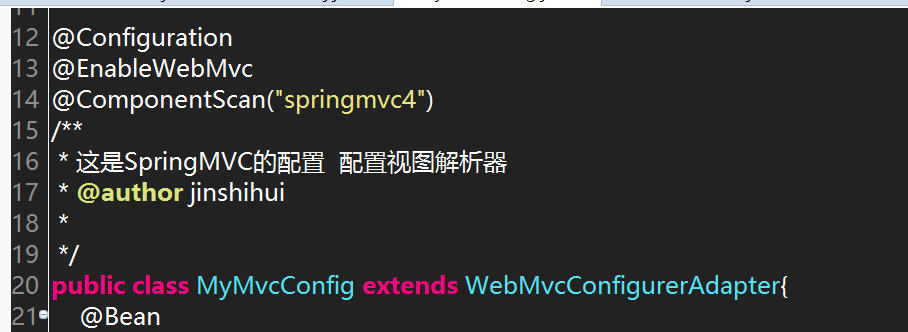
## Spring MVC基本配置

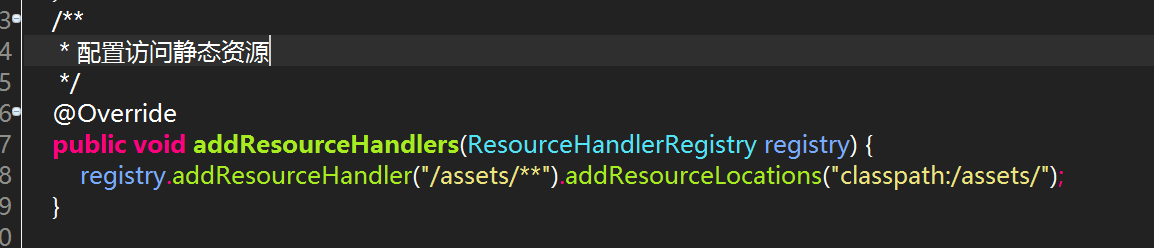
Spring MVC的定制配置需要我们的配置类继承一个WebMvcConfigurerAdapter类，并在此类使用@EnableWebMvc注解，来开启对Spring MVC的配置支持，这样我们就可以重写这个类的方法，完成我们的常用配置。

### 静态资源映射

程序的静态文件（js、css、图片）等需要直接访问，这时我们可以在配置里重写addResourceHandlers方法来实现。

#### 代码示例





说明：

1、@EnableWebMvc注解开启SpringMvc支持，若没有这个注解，重写WebMvcConfigurerAdapter类中的方法无效。

2、继承WebMvcConfigurerAdapter类，重写方法addResourceHandlers()方法对Spring MVC进行配置。

3、addResourceLocations指的是文件放置的目录，addResourceHandler指的是对外暴露的访问路径。

运行：

http://localhost:8080/demo/assets/js/jquery-3.0.0.min.js

http://localhost:8080/demo/assets/pdf/one.pdf

### 拦截器设置

拦截器实现对每一个请求处理前后进行相关的业务处理，类似于Servlet的Filter。

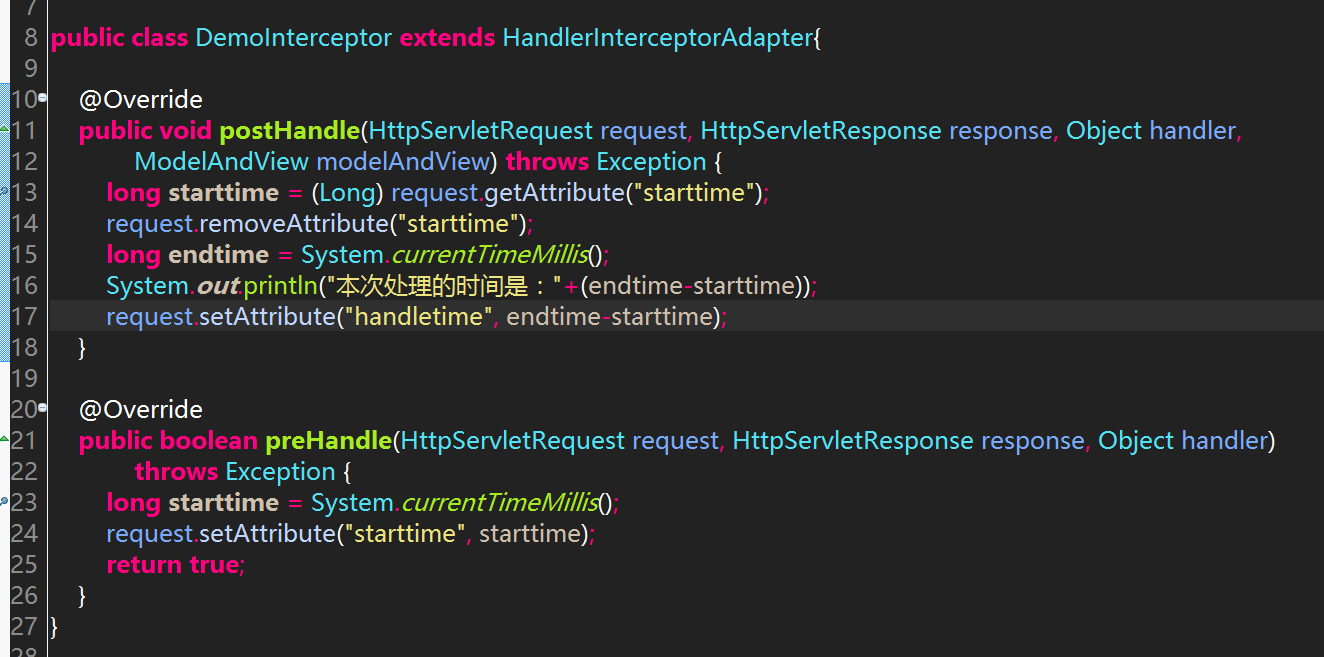
实现方法：

（1）可让普通的Bean实现HandlerInterceptor接口或者继承HandlerInterceptorAdapter类来实现自定义拦截器。

（2）在配置类中，通过重写WebMvcConfigurerAdapter的addInterceptor方法来注册自定义的拦截器。

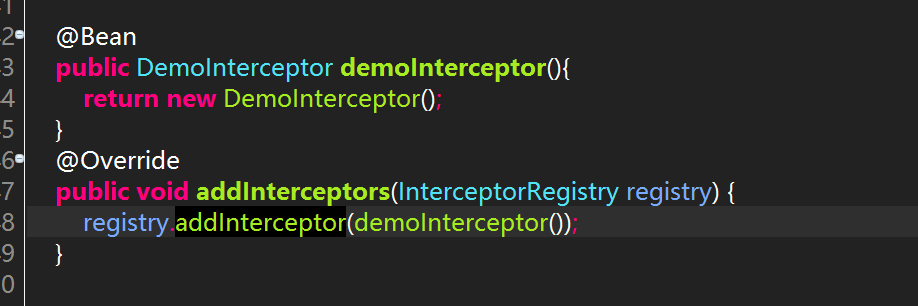
#### 代码示例

（1）编写一个类，继承HandlerInterceptorAdapter类。



说明：重写preHandler()方法在请求发生前执行，重写postHandler()方法，在请求完成后处理。

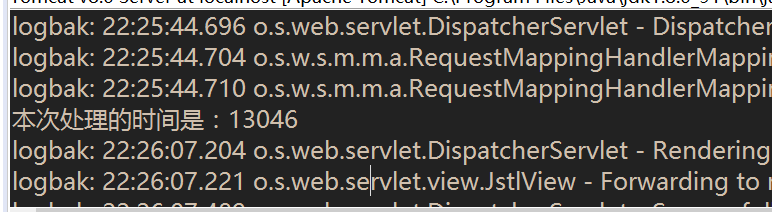
（2）编写Springmvc配置类，继承WebMvcConfigurerAdapter类，重写addInterceptors()方法。



说明：demoInterceptor()方法是配置拦截器的Bean，重写addInterceptors()方法，注册拦截器。

（3）运行

输入：http://localhost:8080/demo/index，查看控制台：



### @ControllerAdvice

通过@ControllerAdvice注解，我们可以将对于控制器的全局配置放置在同一个位置，注解了@Controller类的方法可使用@ExceptionHandler、@InitBinder、@ModelAttribute注解到方法上，这对所有注解在@RequestMapping的控制器内的方法有效。

@ExceptionHandler：用于全局处理控制器里的异常。

@InitBinder：用来设置WebDataBinder，WebDataBinder用来自动绑定前台请求参数到Model中。

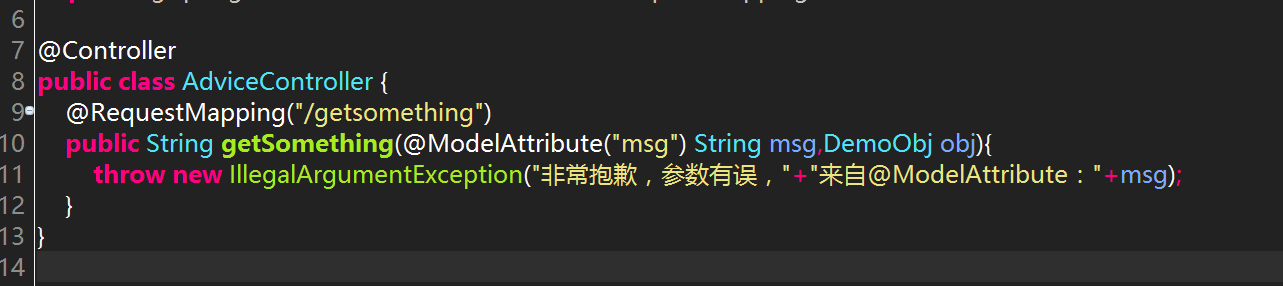
@ModelAttribute：这个注解本来的作用是绑定键值对到Model中，此处是让全局的@RequestMapping能够获取在此处设置的键值对。

#### 代码示例

（1）编写一个ExceptionHandlerAdvice类。



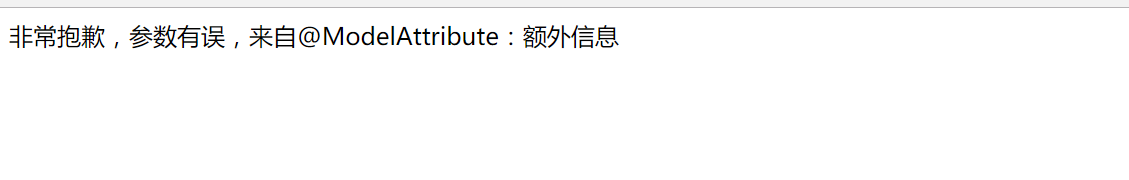
（2）编写控制器类



（3）运行

输入：http://localhost:8080/demo/getsomething

结果：



### 其他配置

#### 1、快捷的ViewController

如果只是单纯的要实现页面的跳转，可以在SpringMvc的配置类中重写addViewControllers方法来简化配置。

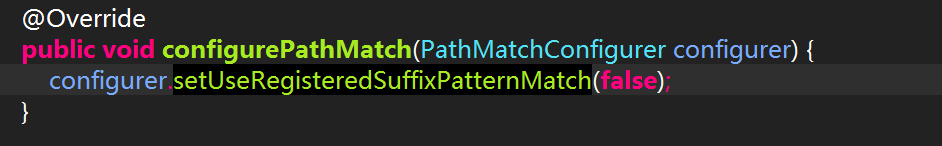


这样实现的代码更简洁，管理更集中。

#### 2、路径匹配参数配置

在Spring MVC中，路径参数如果带点.的话，.后面的值将被忽略。通过重写configurePathMatch方法可不忽略.后面的参数。

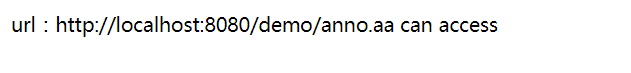
代码：



示例：

输入：http://localhost:8080/demo/anno.aa

结果：



#### 3、更多配置

更多配置查看WebMvcConfigurerAdapter类或者WebMvcConfigurer接口的API。

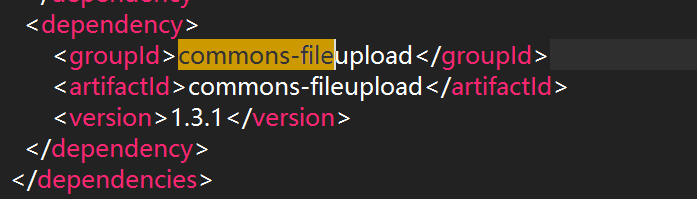
## Spring MVC高级配置

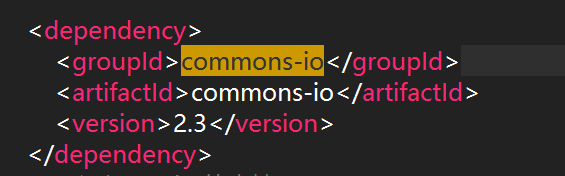
### 文件上传配置

Spring MVC通过配置MultipartResolver来上传文件。在Spring的控制器中，通过MultipartFile file来接收文件，通过MultipartFile[] files接收多个文件上传。

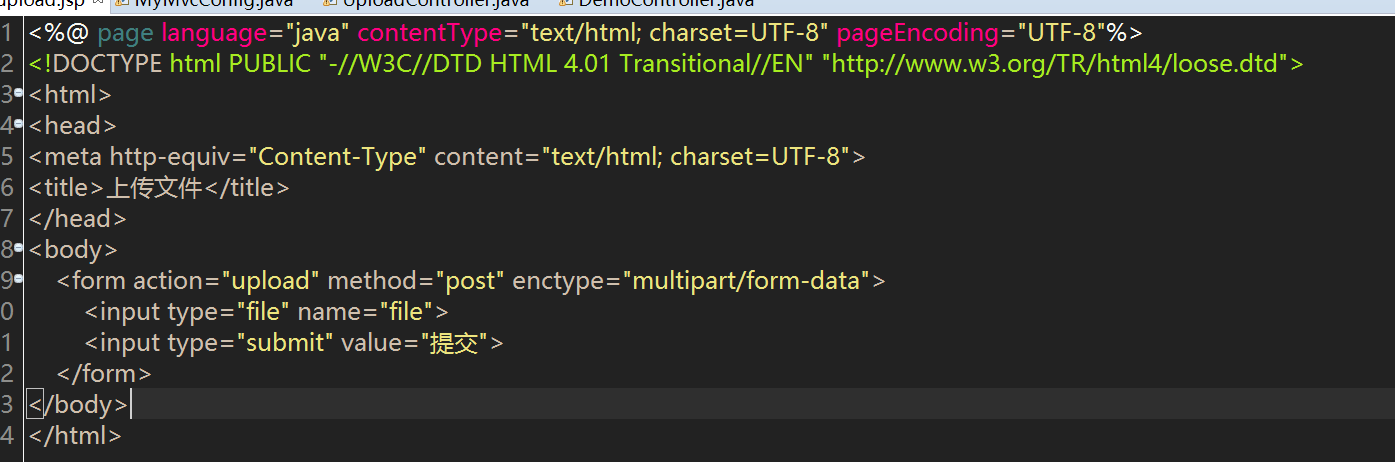
#### 代码示例

##### （1）在pom.xml文件中添加依赖

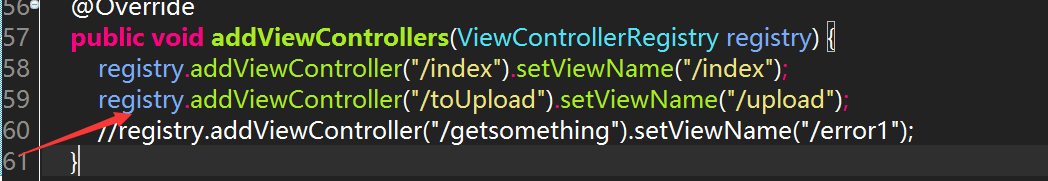




##### （2）定义上传文件页面upload.jsp

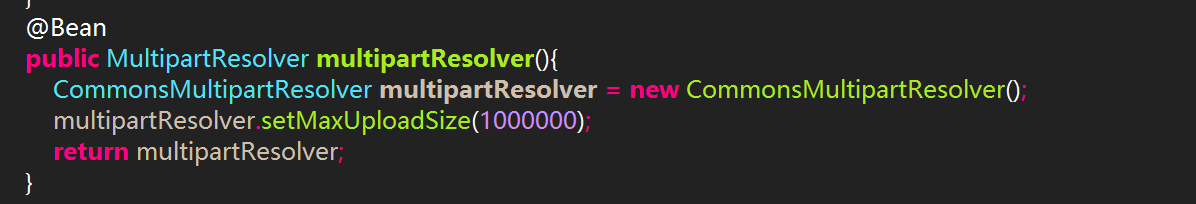


##### （3）添加跳转到上传页面的ViewController

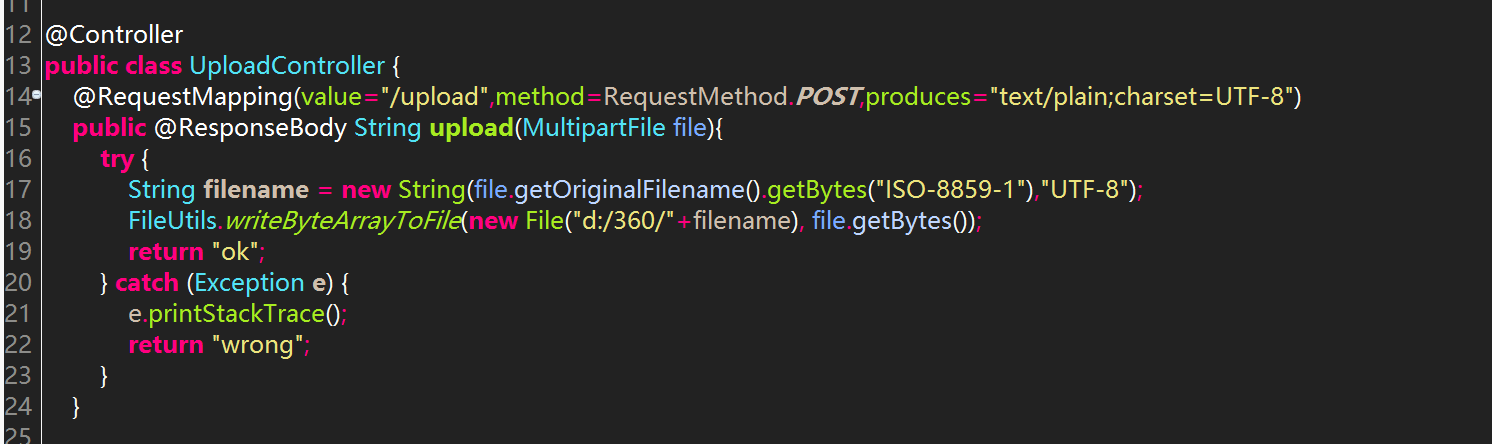


##### （4）MultipartResolver配置

在Spring MVC的配置类中进行配置



##### （5）编写上传文件的控制器



说明：

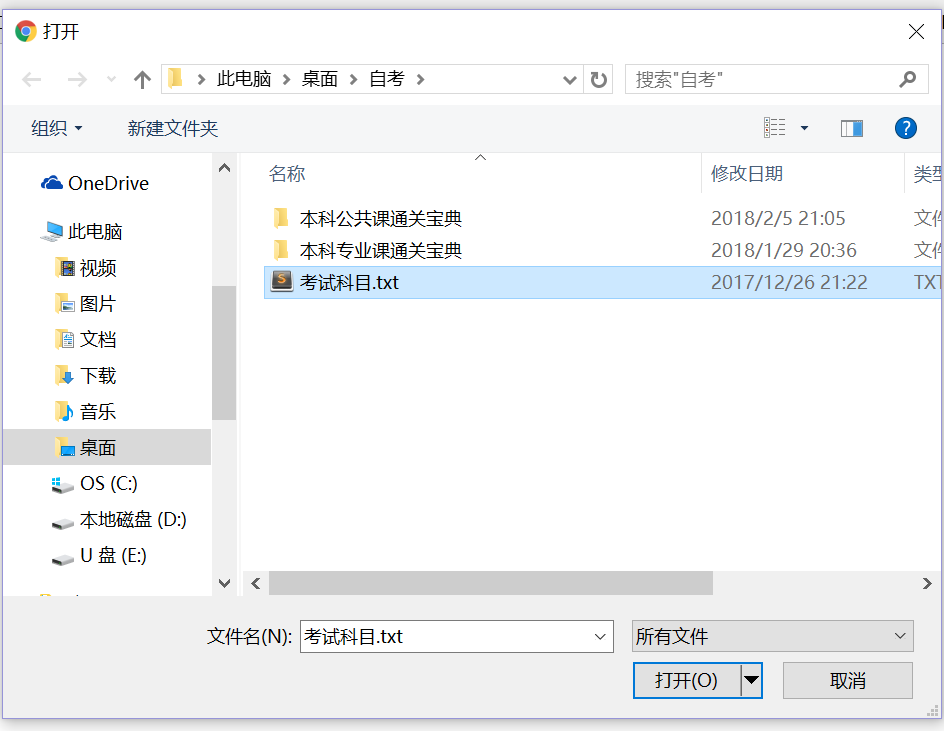
**new** String(file.getOriginalFilename().getBytes("ISO-8859-1"),"UTF-8");解决了文件名是中文的乱码。

##### （6）运行

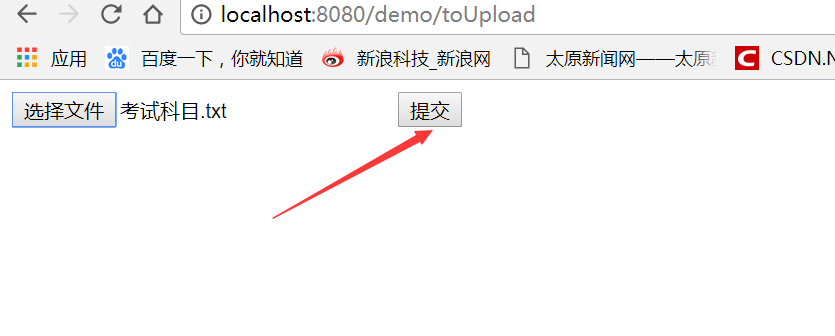
输入：http://localhost:8080/demo/toUpload



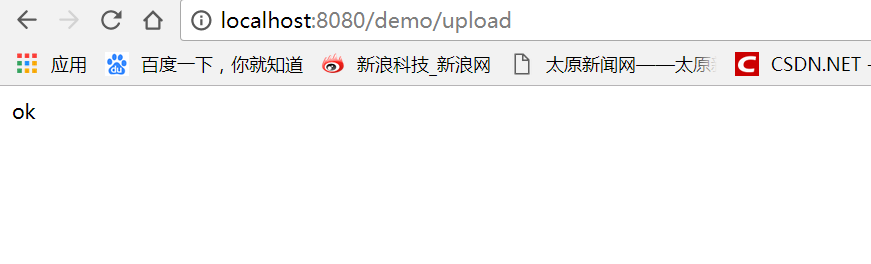
单击选择文件按钮：



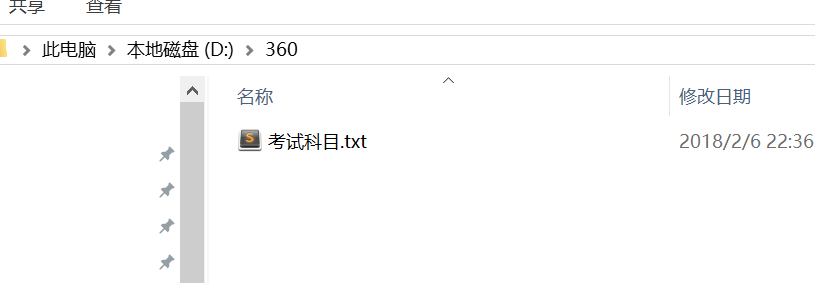
选择一个文件进行添加，然后添加提交。



上传成功：



可以在d:\360文件夹下面看到上传的文件。

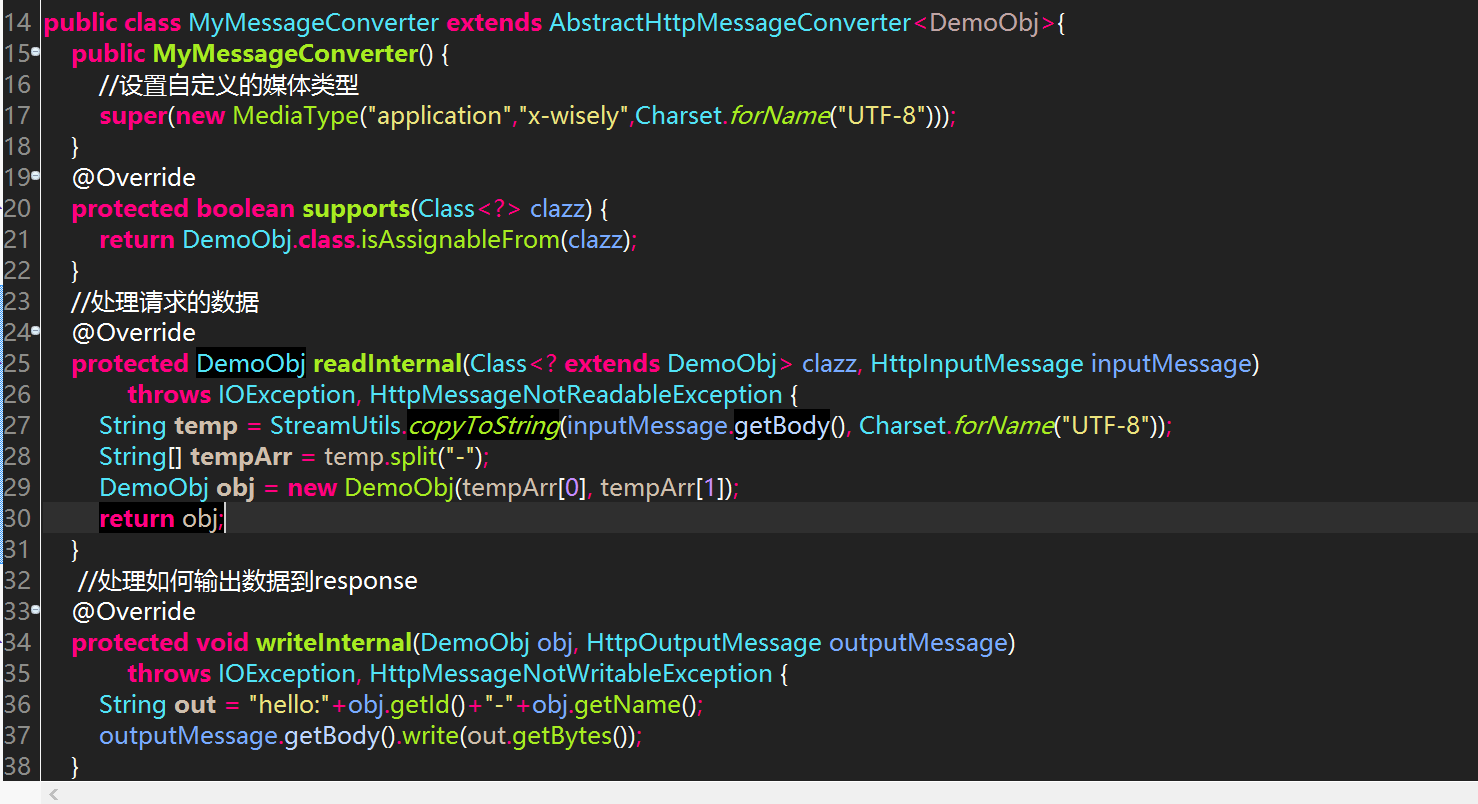


### 自定义消息转换器

HttpMessageConverter是用来处理request和response里的数据，Spring为我们内置了大量的HttpMessageConverter。

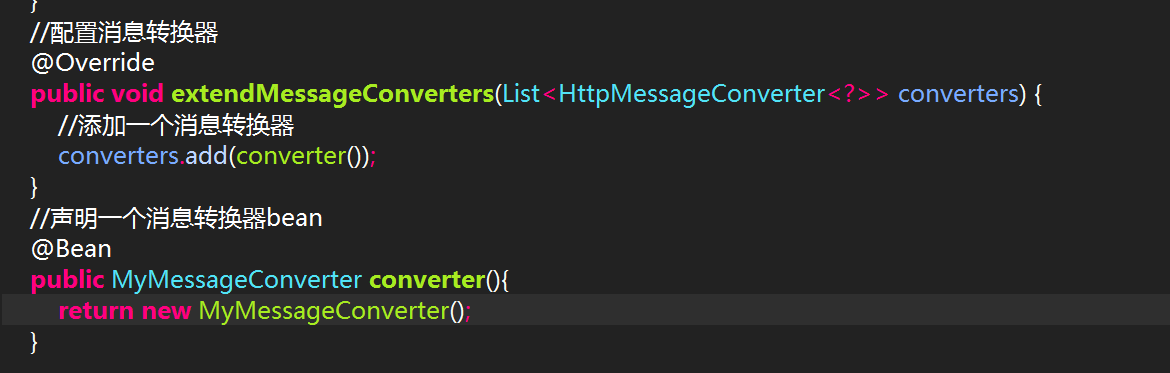
#### 代码示例

（1）定义一个类，继承AbstractHttpMessageConverter类



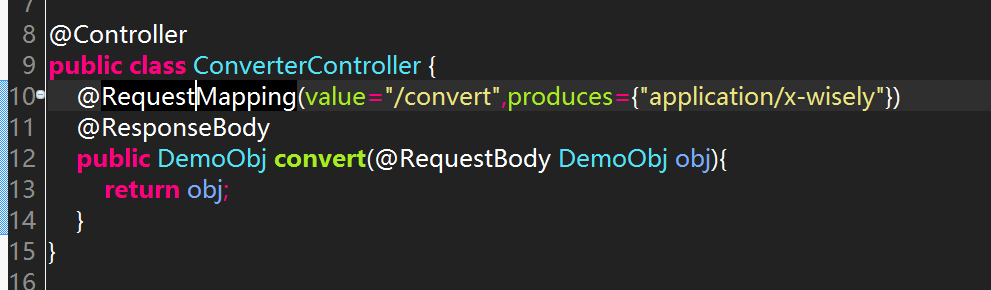
说明：自定义消息转换器HttpMessageConverter

（2）在Spring MVC配置类中注册消息转换器

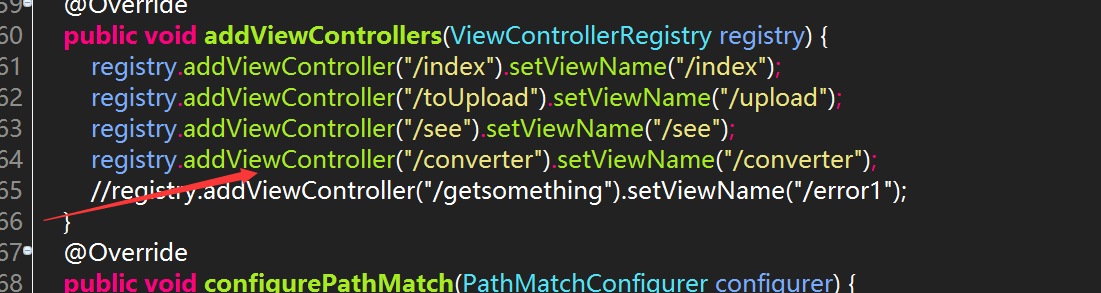


说明：重写extendMessageConverters方法，不会覆盖默认注册的HttpMessageConverter。

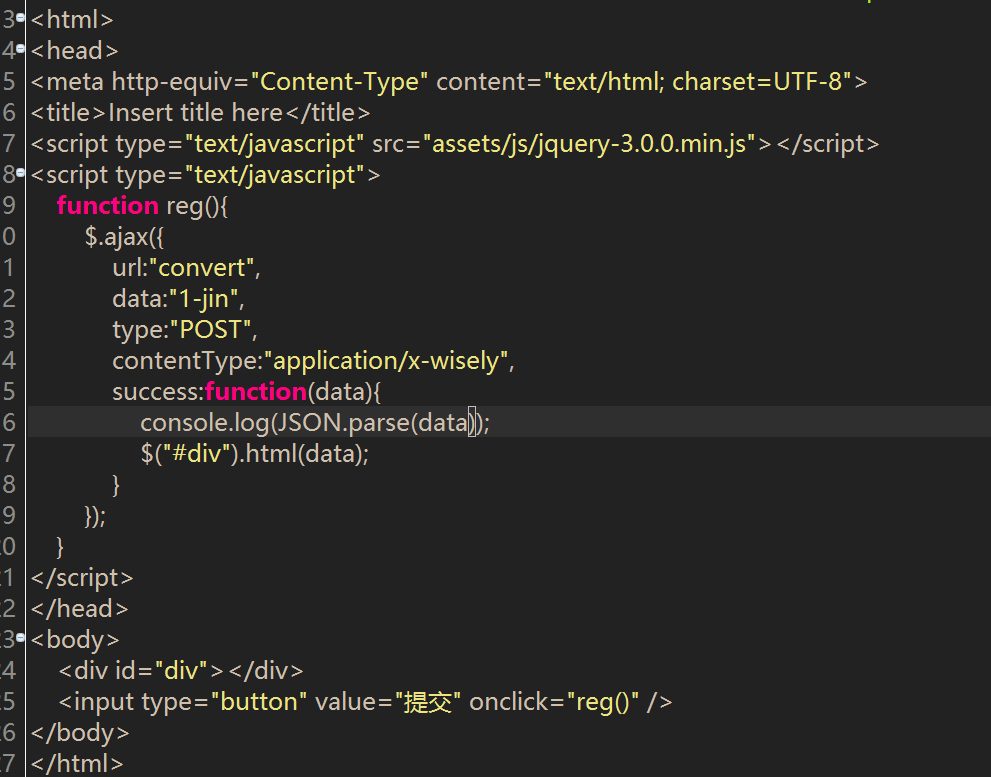
（3）编写控制器类



（4）在Spring MVC配置类中定义ViewController



（5）编写converter.jsp页面

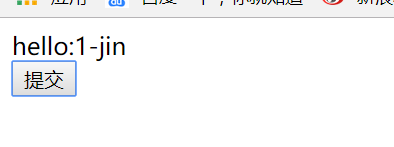


（6）运行

输入：http://localhost:8080/demo/converter



点击提交按钮



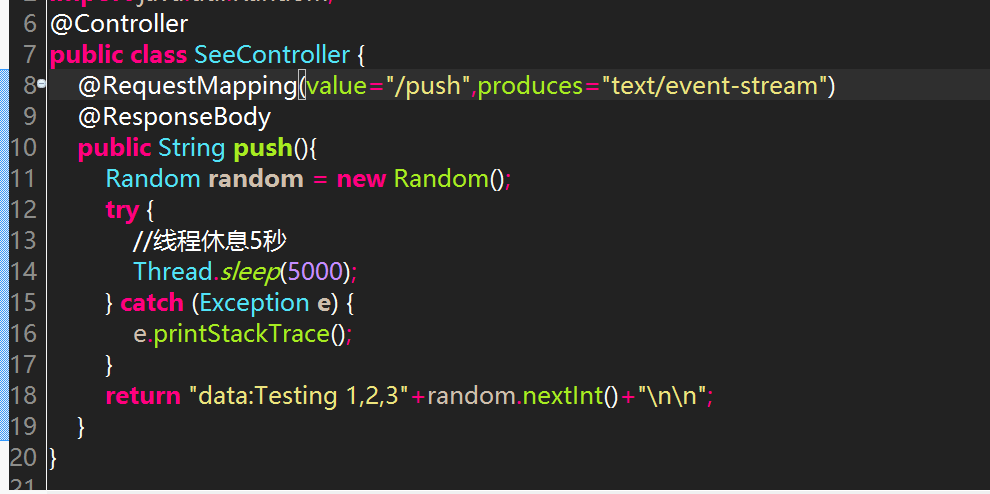
### 服务器端推送技术

服务器端推送方案基于：当客户端向服务器发送请求时，服务端会抓住这个请求不放，等有数据更新的时候才返回给客户端，当客户端接收到消息后，再向服务端发送请求，周而复始，这种方式的好处是减少了服务器的请求数量，大大减少了服务器的压力。我们通过基于SSE（Server Send Event 服务端发送事件）的服务器端推送和基于Servlet3.0+的异步方法特性，其中第一种方式需要新式浏览器的支持，第二种方式是跨浏览器的。

#### SSE

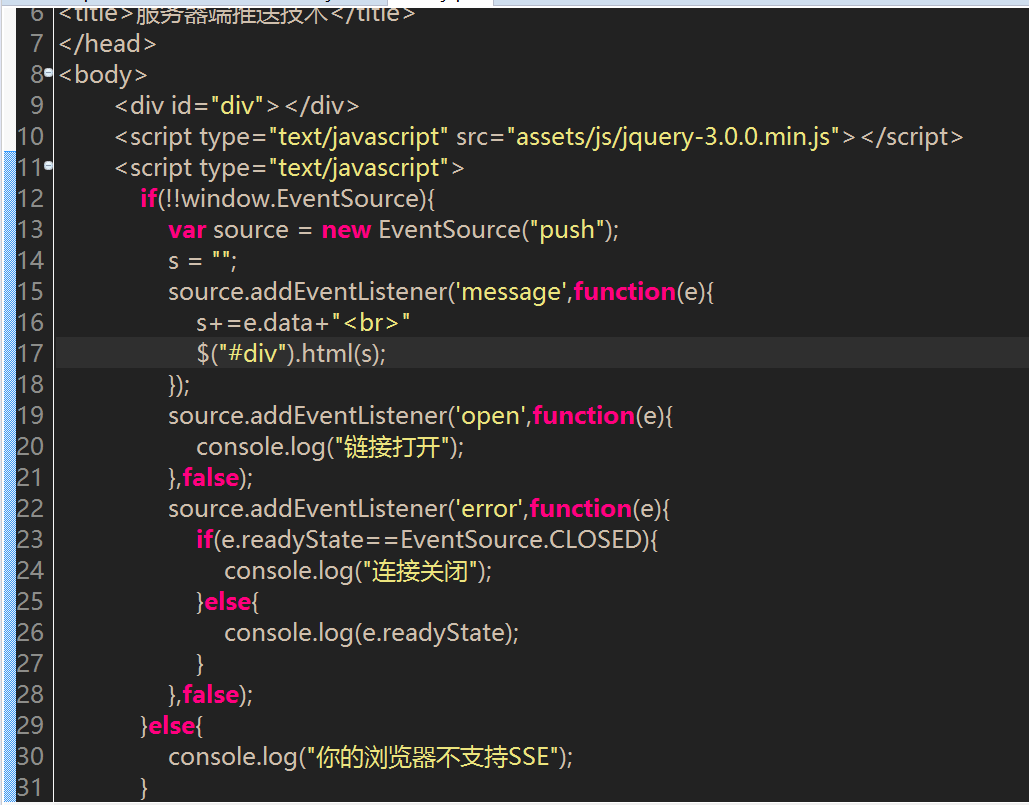
##### 代码示例

（1）定义控制器类



说明：这里使用输出的媒体类型是text/event-stream，这是服务器端SSE的支持，本例演示每五秒向浏览器推送随机消息。

（2）定义页面see.jsp



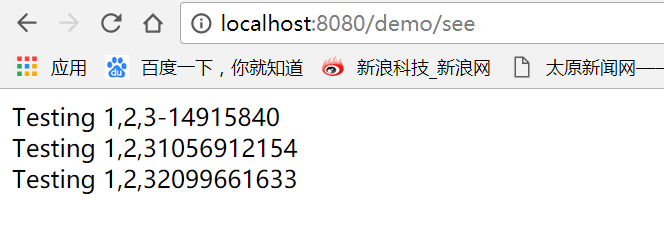
说明：EventSource对象只有新式的浏览器才有（Chrome Firefox）等，EventSource是SSE的客户端。添加SSE客户端监听，在此获取服务器端推送的消息。

（3）在Spring MVC配置类中添加ViewController，跳转到see.jsp页面



（4）运行

输入：http://localhost:8080/demo/see



#### Servlet3.0+异步方法处理

## Spring MVC测试

# SpringMVC框架的优点

（1）springMVC与spring的无缝整合，结合spring可以一站式开发。

（2）springMVC本身比struts2灵活易用，使用注解大量简化配置。

（3）springMVC是方法级别的拦截，一个方法对应一个request上下文，而方法同时又跟一个url对应。

（4）springMVC的方法之间基本上独立，独享request、response数据，请求数据通过参数获取，处理结果通过ModelMap交回给框架，方法之间不能共享变量。

# spring上下文和springMVC上下文的关系

springMVC上下文继承了spring上下文

# Spring与SpringMVC集成（重点）

（1）拷贝准确的jar文件

（2）在web.xml文件中配置spring的监听器和Spring的配置文件路径，如下：

<listener>

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

</listener>

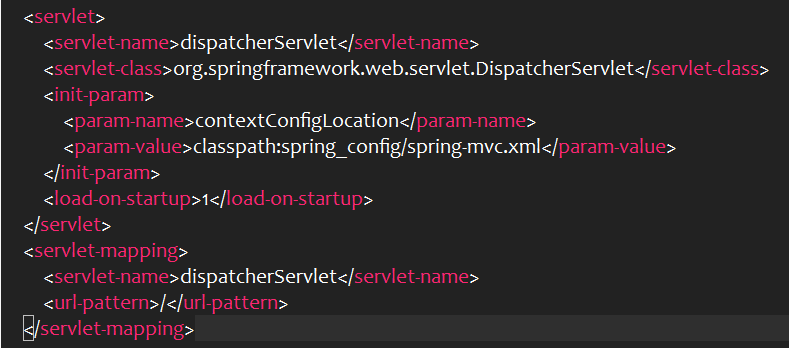
<context-param>

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

<param-value>classpath:spring\_config/applicationContext.xml,classpath:spring\_config/scheduler-config.xml</param-value>

</context-param>

（3）在web.xml文件中配置springMVC的servler以及路径



（4）编写spring配置文件

（5）编写springMVC配置文件

（6）定义接口及实现类

（7）定义控制器类，在类中编写属性以便调用实现类中的方法。

# Spring MVC工作流程描述（重点）

（1）用户向服务器发送请求，请求被DispatcherServlet捕获。

（2）DispatcherServlet对请求的URL进行解析，调用HandlerMapping获得该Handler配置的所有相关的对象（包括Handler对象以及Handler对象对应的拦截器），最后以HandlerExecutionChain对象的形式返回。

（3）DispatcherServlet根据获得的Handler，选择一个合适的HandlerAdapter。(附注：如果成功获得HandlerAdapter后，此时将开始执行拦截器的preHandler()方法)

HandlerAdapter作用是让固定servlet处理方法调用灵活的Handler来进行处理请求。

（4）提取request中的数据，填充Controller中的参数，开始执行Handler(Controller)， 在填充Handler的参数过程中，根据你的配置，Spring将帮你做一些额外的工作。

数据类型转换：对请求消息进行数据转换，如String转换成Integer、Double等。

数据格式化：对请求消息进行数据格式化，如字符串格式化成数字或日期。

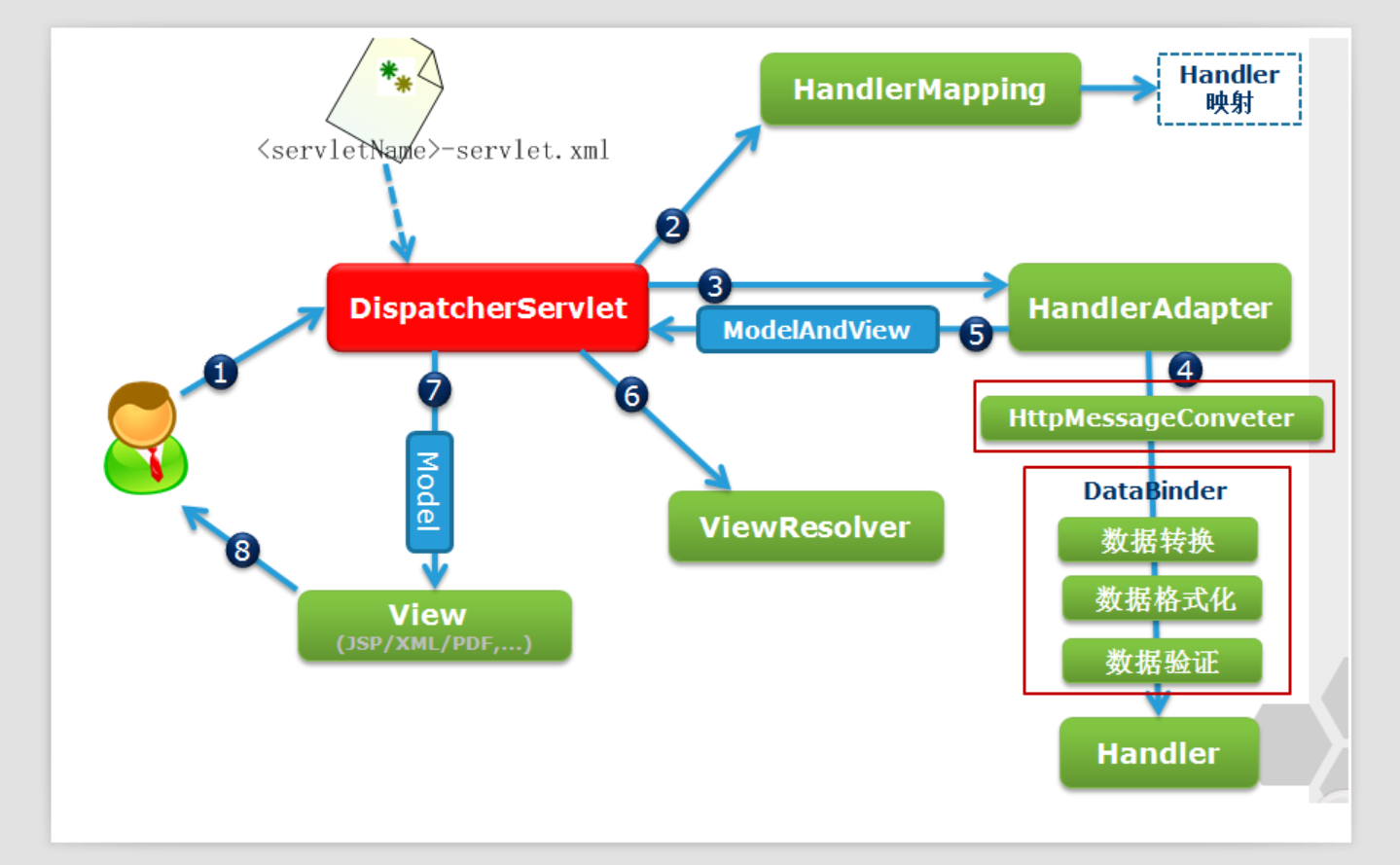
数据验证：验证数据的有效性（长度、格式等），验证结果存储到BindingResult或Error中。

（5）Handler执行完成后，向DispatcherServlet返回一个ModelAndView对象。

（6）根据返回的ModelAndView对象，选择一个合适的ViewResolver(必须是已经注册到Spring容器中的ViewResolver)查询视图，返回给DispatcherServlet。

（7）ViewResolver结合Model和View来渲染视图。

（8）将渲染结果返回给客户端浏览器。



# SpringMVC为什么要结合使用HandlerMapping以及HandlerAdapter来处理Handler

符合面向对象中的单一职责原则，代码架构清晰、便于维护，最重要的是代码可复用性高。如HandlerAdapter可能会被用于处理多种Handler。SpringMVC框架是高度可配置的，而且包含多种视图技术，如：JSP、Velocity、Tiles，Spring MVC 框架并不知道使用的视图，所以不会强迫您只使用 JSP 技术。

# DispatcherServlet类

DispatcherServlet是Spring MVC最核心的类，DispatcherServlet执行处理的入口方法是doService方法，不过doService并没有直接进行处理，而是交给了doDispatch方法进行具体的处理，在doDispatch处理前doService做了一些事情，首先判断是不是include请求，如果是则对request的Attribute做个快照备份，等doDispatch处理完之后进行还原，在做完快照后又对request设置了一些属性。

DispatcherServlet继承了FrameworkServlet，FrameworkServlet继承了HttpServletBean，而HttpServletBean类继承HttpServlet类，所以DispatcherServlet本质是HttpServlet。

onRefresh方法是DispatcherServlet类的入口方法，onRefresh方法调用了initStrategies方法。initStrategies方法初始化了九个组件，分别是：MultipartResolver

LocaleResolver

ThemeResolver

HandlerMappings

HandlerAdapters

HandlerExceptionResolvers

RequestToViewNameTranslator

ViewResolvers

FlashMapManager

# doDispatch方法设计了整个请求处理的过程

（1）根据request找到Handler（处理器）

（2）根据Handler找到对应的HandlerAdapter

（3）用HandlerAdapter处理Handler

（4）调用processDispatchResult方法处理上面处理后的结果（包含找到view并渲染输出给用户）

# Handler（处理器）

处理器，直接对应着MVC中的Controller，它的具体表现形式很多，可以是类，也可以是方法，如果你能想到别的表现形式也可以使用，它的类型是Object，只要可以处理请求就可以是Handler。Spring MVC中每个控制器可以定义多个请求处理方法，我们把这种请求处理方法简称为Action，每个请求处理方法可以有多个不同的参数，以及一个多种类型的返回结果。

# HandlerMapping

用来查找Handler的，在springMVC中会处理很多请求，每个请求都需要一个Handler来处理，具体接收到一个请求后会由HandlerMapping查找对应的Hander进行处理请求。

# HandlerAdapter（处理器适配器）

处理器适配器，作用是让固定servlet处理方法调用灵活的Handler来进行处理请求。

# 理解Handler、HandlerMapping、HandlerAdapter

通俗理解：Handler是用来干活的工具，HandlerMapping是根据需要干的活找到对应的工具，HandlerAdapter是使用工具干活的人，不同的Handler需要不同的HandlerAdapter去使用。View是用于展示数据的，而ViewResolver用来查找View的。

# 如何修改SpringMVC配置文件的路径

SpringMVC的配置文件的路径可以改变，在web.xml文件中进行配置。

<init-param>

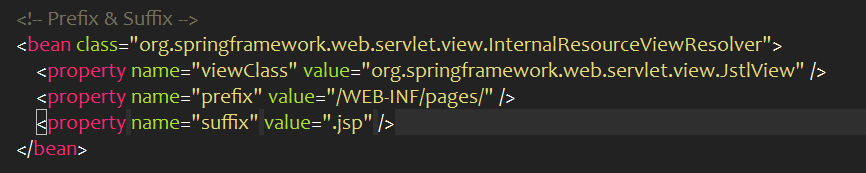
<param-name>contextConfigLocation</param-name>

<param-value>classpath\*:config/SpringMVC-servlet.xml</param-value>

</init-param>

# 配置视图解析器（重点）

## 配置示例



说明：在springmvc配置文件中进行配置。

## ViewResolver接口

由可以按名称解析视图的对象实现的接口，在运行应用程序期间，视图状态不会更改，因此实现可以自由地缓存视图。鼓励实现支持国际化，即：本地化视图分辨率。

## InternalResourceViewResolver类

支持InternalResourceView(即servlet和jsp)和子类(如JstlView)的UrlBasedViewResolver的方便子类，这个解析器生成的所有视图的视图类可以通过setViewClass()指定。默认值是InternalResourceView类，如果JSTL API存在，则为JstlView。顺便说一下，在WEB-INF下只作为视图使用的JSP文件是一个很好的实践，这样可以避免直接访问(例如通过手动输入URL)。届时只有控制器才能访问它们。当链接ViewResolver时，一个InternalResourceViewResolver总是需要排在最后，因为它将尝试解析任何视图名，无论底层资源是否实际存在。

InternalResourceViewResolver：它是URLBasedViewResolver的子类，所以URLBasedViewResolver支持的特性它都支持。在实际应用中InternalResourceViewResolver也是使用的最广泛的一个视图解析器。InternalResourceViewResolver会把返回的视图名称都解析为InternalResourceView对象，InternalResourceView会把Controller处理器方法返回的模型属性都存放到对应的request属性中，然后通过RequestDispatcher在服务器端把请求forword到目标URL。比如在InternalResourceViewResolver中定义了prefix=/WEB-INF/，suffix=.jsp，然后请求的Controller处理器方法返回的视图名称为test，那么这个时候InternalResourceViewResolver就会把test解析为一个InternalResourceView对象，先把返回的模型属性都存放到对应的HttpServletRequest属性中，然后利用RequestDispatcher在服务器端把请求forword到/WEB-INF/test.jsp。

## 视图解析器的作用

只对Controller类中每个方法返回值那里的String类型前后加路径，而不对@RequestMapping处加路径，对@RequestMapping处无影响。

# 编写Controller的方法

## （1）编写类并实现Controller接口

编写类并实现Controller接口，重写handleRequest方法，返回值是ModelAndView对象，然后在springMVC配置文件中进行配置，这种方法只能处理一个请求。说明：用name进行请求

示例：

public class DataController implements Controller{

//没有使用注解

public ModelAndView handleRequest(HttpServletRequest request,

HttpServletResponse response) throws Exception {

return new ModelAndView("MyJsp");

}

}

<bean id="dataController" name="/data" class="controller.annotation.action.DataController" />

## （2）编写类并继承MultiActionController类

编写类并继承MultiActionController类，然后自定义方法，方法中必须带HttpServletRequest request,HttpServletResponse response参数，返回值是字符串，然后在springMVC配置文件中进行配置，这种方法可以处理多个请求。

## （3）通过注解进行配置（重点）

通过注解进行配置，直接在类中定义方法就行，不需要在配置文件中进行配置，方法的返回值可以是任何类型。（推荐使用这种方法）

说明：Controller类中方法返回值的处理，可以返回ModelMap、ModelAndView、Map、List、Set、Object、无返回值。 一般建议返回字符串。

# springMVC如何从一个Controller方法跳转到另一个Controller方法

## （1）使用重定向方法

redirect:方法的路径（说明：也可以返回外网的地址）

redirect:http://www.baidu.com

## （2）转发

forward:方法的路径

# ModelAndView类

web MVC框架中模型和视图的持有者。注意这些是完全不同的。这个类只持有这两个值，使控制器能够以单个返回值返回模型和视图。表示处理程序返回的模型和视图，由DispatcherServlet解析。视图可以采用字符串视图名称的形式，该名称需要由ViewResolver对象解析，或者可以直接指定视图对象。模型是一个Map，允许使用多个按名称键入的对象。

ModelAndView类中的Model代表模型，View代表视图，这个类把要显示的数据存储到了model属性中，要跳转的视图信息存储到了view属性中。

## 构造函数

### public ModelAndView()

### public ModelAndView(String viewName)

### public ModelAndView(View view)

### public ModelAndView(String viewName, Map<String, ?> model)

## 方法

### public void setViewName(String viewName)

### public ModelAndView addObject(String attributeName, Object attributeValue)

## 代码示例

@Controller

@RequestMapping("/user.do")

public class UserController extends MultiActionController {

@RequestMapping(params="method=reg")

public ModelAndView reg(String uname){

ModelAndView mv = new ModelAndView();

mv.setViewName("index");

// mv.setView(new RedirectView("index"));

User user = new User();

user.setUname("高淇");

mv.addObject(user); //查看源代码，得知，直接放入对象。属性名为”首字母小写的类名”。 一般建议手动增加属性名称。

mv.addObject("a", "aaaa");

return mv;

}

}

<h1>${requestScope.a}</h1>

<h1>${requestScope.user.uname}</h1>

# ModelMap类

当构建模型数据用于UI工具时使用的Map的实现，支持链式调用和模型属性名的生成，这个类充当Servlet和Portlet MVC的通用模型持有者。

这个类是LinkedHashMap<String, Object>的子类，可以在其中存放属性，作用域同request，我们可以在modelMap中放入数据，然后在forward的页面上显示这些数据，通过el、jstl表达式、java代码显示这些数据。

## 构造函数

### public ModelMap()

创建一个新的空的ModelMap对象

### public ModelMap(String attributeName, Object attributeValue)

构造一个新的ModelMap，其中包含提供名称下的提供属性。

## 方法

### public ModelMap addAttribute(String attributeName, Object attributeValue)

在提供的名称下添加提供的属性

### public ModelMap addAttribute(Object attributeValue)

将提供的属性添加到此Map中。

### public ModelMap addAllAttributes(Collection<?> attributeValues)

使用每个元素的属性名称生成，将提供的集合中的所有属性复制到此映射中。

## 代码示例

@Controller

@RequestMapping("/user.do")

public class UserController extends MultiActionController {

@RequestMapping(params="method=reg")

public String reg(String uname,ModelMap map){

map.put("a", "aaa");

return "index";

}

}

<h1>${requestScope.a}</h1>

<c:out value="${requestScope.a}"></c:out>

# 实现静态文件的访问

可以访问图片、js文件、css文件。需要在springMVC配置文件中配置：

<mvc:resources location="/image/" mapping="/image/\*\*"/>//访问图片

<mvc:resources location="/js/" mapping="/js/\*\*"/>//访问js文件

# 注解的类型（重点）

## @Controller

使用Controller注解用于指示Spring类的实例是一个控制器，Spring使用扫描机制来找到应用程序中所有基于注解的控制器类。@Controller负责注册一个bean到spring上下文中，bean的ID默认为类名称开头字母小写，ID也可以不写。springMVC的控制器是单例的，这就意味着会被多个请求线程共享，因此我们将控制器设计成无状态类。

属性：value，用于指定组件的名称。

## @Service

指示一个类是服务类，在业务处理层，一般在serviceImpl中编写，@Service负责注册一个bean到spring上下文中，bean的ID默认为类名称开头字母小写。

属性：value，用于指定组件的名称。

## @Repository

指示类是一个持久层类，一般在daoImpl中编写，负责向spring上下文中注册bean，bean的ID默认为类名称开头字母小写。

属性：value，用于指定组件的名称。

## @RequestMapping

作用：映射一个请求和一种方法。可以使用@RequestMapping注释一种方法或类，一个采用@RequestMapping注释的方法将成为一个请求处理的方法，并由调度程序在接收到对应的URL请求时调用，使用RequestMapping注解的value属性将URL映射到方法，由于value属性是RequestMapping注解的默认属性，因此若只有唯一的属性，则可以省略属性名称，但是如果超过一个属性时，就必须写value属性名称。method属性用来指示该方法处理哪些HTTP方法，如果method属性只有一个http方法值，则无需花括号，如果没有指定method属性，可以处理任何HTTP方法。

属性：

（1）String[] value()

（2）RequestMethod[] method()

（3）String[] params()

（4）String[] headers()

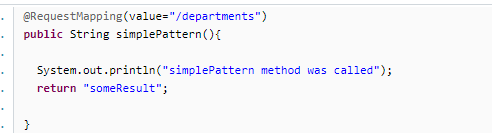
（5）String[] consumes()

（6）String[] produces()

### 使用方法

#### 最基本的方法级别上的应用

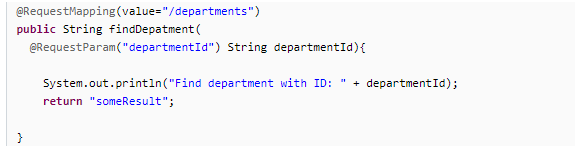
代码：



访问http://localhost/xxxx/departments的时候，会调用 simplePattern方法。

#### 参数绑定

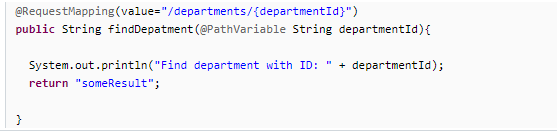
代码：



访问形式 /departments?departmentId=23就可以触发访问findDepatment方法了 。

#### REST风格的参数

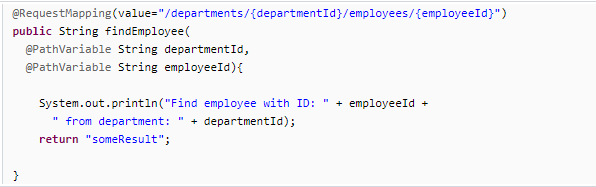
代码：



访问形式：/departments/23，把23作为传入的departmetnId,，但是在实际的方法findDepatment中，使用 @PathVariable("departmentId") String departmentId将其绑定为 departmentId,所以这里departmentId为23 。

#### url中同时绑定多个id

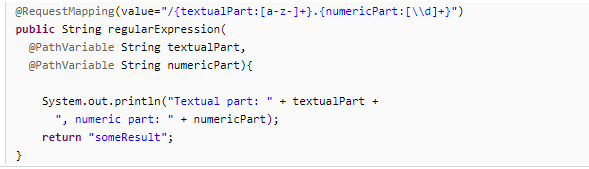
代码



访问形式：/departments/23/employees/21

#### 支持正则表达式

代码



访问形式：/sometext.123

输出：Textual part: sometext, numeric part: 123

## @GetMapping

@GetMapping是一个组合注解，是@RequestMapping(method = RequestMethod.GET)的缩写。

说明：这个注解是在Spring4.3版本以后出来的。

## @PostMapping

@PostMapping是一个组合注解，是@RequestMapping(method = RequestMethod.POST)的缩写。

说明：这个注解是在Spring4.3版本以后出来的。

## @Autowired

对成员变量、方法和构造函数进行标注来完成自动注入，默认按类型注入。

## @Component（不推荐使用）

作用同@Controller一样，也是把一个Spring类的实例看做一个控制器。@Component是所有受Spring管理组件的通用形式，Spring还提供了更加细化的注解形式： @Repository、@Service、@Controller，当你的类不清楚是哪一层的时候使用该注解。

## @Resource（一般不用这个）

@Resource

private DataSource dataSource; // inject bean named 'dataSource'

　　或者：

　　　　@Resource(name="dataSource")

　　　　@Resource(type=DataSource.class)

@Resource默认按bean的name进行查找，如果没有找到会按type进行查找，此时与@Autowired类似。

## @RequestParam

一般用于将指定的请求参数赋给方法中的形参。示例代码如下：

@RequestMapping(params="method=reg5")

public String reg5(@RequestParam("name")String uname,ModelMap map) {

System.out.println("HelloController.handleRequest()");

System.out.println(uname);

return "index";

}

这样就会将name参数的值赋给uname，当然如果请求参数名称和形参名称保持一致，则不需要这种写法。

属性：

（1）String value()

（2）boolean required()

（3）String defaultValue()

## @Scope

在使用XML定义Bean时，可以通过bean的scope 属性来定义一个Bean的作用范围，同样可以通过@Scope注解来完成。@Scope中可以指定如下值：

singleton:定义bean的范围为每个spring容器一个实例（默认值）

prototype:定义bean可以被多次实例化（使用一次就创建一次）

request:定义bean的范围是http请求（springMVC中有效）

session:定义bean的范围是http会话（springMVC中有效）

global-session:定义bean的范围是全局http会话（portlet中有效）

示例：

@Scope("session")

@Repository

public class UserSessionBean implement Serializable{

}

## @SessionAttributes（一般不用）

将ModelMap中指定的属性放到session中，示例：

@Controller

@RequestMapping("/user.do")

@SessionAttributes({"u","a"}) //将ModelMap中属性名字为u、a的值放入session中。这样，request和session中都有了。

public class UserController {

@RequestMapping(params="method=reg4")

public String reg4(ModelMap map) { System.out.println("HelloController.handleRequest()");

map.addAttribute("u","uname"); //将u放入request作用域中，这样转发页面也可以取到这个数据。

return "index";

}

}

<body>

<h1>\*\*\*\*\*\*\*\*\*\*${requestScope.u.uname}</h1>

<h1>\*\*\*\*\*\*\*\*\*\*${sessionScope.u.uname}</h1>

</body>

说明：名字为”user”的属性再结合使用注解@SessionAttributes可能会报错。

## @ModelAttributes（一般不用）

这个注解可以跟@SessionAttributes配合在一起用。可以将ModelMap中属性的值通过该注解自动赋给指定变量。示例：

@Controller

@RequestMapping("/user.do")

@SessionAttributes({"u","a"})

public class UserController {

@RequestMapping(params="method=reg4")

public String reg4(ModelMap map) {

System.out.println("HelloController.handleRequest()");

map.addAttribute("u","尚学堂高淇");

return "index";

}

@RequestMapping(params="method=reg5")

public String reg5(@ModelAttribute("u")String uname,ModelMap map) {

System.out.println("HelloController.handleRequest()");

System.out.println(uname);

return "index";

}

}

先调用reg4方法，再调用reg5方法。我们发现控制台打印出来：尚学堂高淇。

## @RequestBody

@RequestBody一般用在方法的参数中。这个注解没有属性

### 作用

（1）这个注解用于读取request请求的body数据，使用系统默认配置的HttpMessageConverter进行解析，然后把响应的数据绑定到要返回的对象上。

（2）再把HttpMessageConverter返回的对象数据绑定到 controller中方法的参数上。

### 使用时机

#### （1）GET、POST方式

根据request header Content-Type的值来判断。

application/x-www-form-urlencoded，可选（即非必须，因为这种情况的数据@RequestParam, @ModelAttribute也可以处理，当然@RequestBody也能处理）

multipart/form-data, 不能处理（即使用@RequestBody不能处理这种格式的数据）

其他格式， 必须（其他格式包括application/json, application/xml等，这些格式的数据必须使用@RequestBody来处理）

#### （2）PUT方式

根据request header Content-Type的值来判断:

application/x-www-form-urlencoded， 必须；

multipart/form-data, 不能处理；

其他格式， 必须；

说明：request的body部分的数据编码格式由header部分的Content-Type指定。@RequestBody需要把所有请求参数作为json解析，因此不能包含key=value这样的写法在请求url中，所有的请求参数都是一个json，直接通过浏览器输入url时，@RequestBody获取不到json对象，需要用java编程或者基于ajax的方法请求，将Content-Type设置为application/json。

## @ResponseBody

这个注解没有属性，这个注解用于方法上。

### 作用

  该注解用于将Controller类的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区。

### 使用时机

  返回的数据不是html标签的页面，而是其他某种格式的数据时（如json、xml等）使用。HttpMessageConverter这个接口定义了四个方法：分别是读取数据时的 canRead(), read() 和 写入数据时的canWrite(), write()方法。

在使用 <mvc:annotation-driven />标签配置时，默认配置了RequestMappingHandlerAdapter（注意是RequestMappingHandlerAdapter不是AnnotationMethodHandlerAdapter。）

当使用@RequestBody和@ResponseBody注解时，RequestMappingHandlerAdapter就使用它们来进行读取或者写入相应格式的数据。

## @PathVariable

通过@PathVariable 可以将 URL 中占位符参数绑定到控制器处理方法的入参中。

示例：



## @ RestController

使用@Controller和@ResponseBody进行注释的便利注释，携带此注释的类型被视为控制器，其中@RequestMapping方法默认为@ResponseBody语义。

属性：

value：该值可能指示逻辑组件名称的建议，以便在自动检测组件的情况下转换为Spring bean。

# spring配置文件中加入注解配置

为了保证spring能找到你用注解配置的控制器，需要如下配置：

（1）在spring配置文件中加入spring-context的dtd格式。

（2）加入<context:component-scan />元素，在元素中指定注解控制器类的基本包。

# 注解优化

## （1）启用注解驱动

<mvc:annotation-driven />

## （2）添加@RequestMapping

给类中的方法添加请求映射@RequestMapping注解

# springMVC解决乱码的方法

## （1）通过服务器的配置文件去处理

修改服务器中的配置文件server.xml，添加URIEncoding=UTF-8

## （2）在web.xml中配置过滤器去处理

<filter>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

<init-param>

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

# springMVC文件上传

## （1）编写前台页面

注意点：

文本类型是：type="file"，示例：

<input type="file" name="file">

在form表单中加上这个属性：enctype="multipart/form-data"

示例代码：

<form name="uploadForm" action="uploadFile" method="post" enctype="multipart/form-data">

<input type="file" name="file" />

<input type="submit" value="上传"/>

</form>

说明：需要写在单独的form表单中。

## （2）配置文件上传的解析器

在springMVC配置文件中配置文件上传的解析器

<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

<property name="defaultEncoding" value="UTF-8" />//默认编码

<property name="maxUploadSize" value="10485760000" />//最大上传的文件大小，-1是无限制

<property name="maxInMemorySize" value="40960" />//最大内存大小

</bean>

说明：id的值必须写对，不能随意写。

## （3）编写Controller类

方法中带有参数：CommonsMultipartFile file,HttpServletRequest request

public String upload(@RequestParam("file") CommonsMultipartFile file,HttpServletRequest request) throws Exception{

System.out.println("FileName------>"+file.getOriginalFilename());

try{

if(!file.isEmpty()){

FileOutputStream os = new FileOutputStream("D:/"+file.getOriginalFilename());

InputStream in = file.getInputStream();

int b = 0;

while((b = in.read())!=-1){

os.write(b);

}

os.flush();

os.close();

in.close();

}

}catch (Exception e) {

e.printStackTrace();

}

return "fileuploadsuccess";

}

## （4）编写返回成功的页面

编写filesuccess.jsp，表示上传成功。

# springMVC文件上传优化（支持上传多个文件）

Spring通过对servlet API的HttpServletRequest接口进行扩展，使其能够很好的处理文件的上传，扩展的接口名是MultipartHttpServletRequest，这种方法可以上传多个文件。

开发思路如下：

## （1）实例化一个文件解析器

CommonsMultipartResolver multipartResolver =new CommonsMultipartResolver (

request.getSession().getServletContext());

## （2）HttpServletRequest转换为MultipartHttpServletRequest

MultipartHttpServletRequest multirequest = (MultipartHttpServletRequest)request;//转换数据类型

## （3）创建iterator接口，对上传的文件进行迭代

## （4）创建文件名

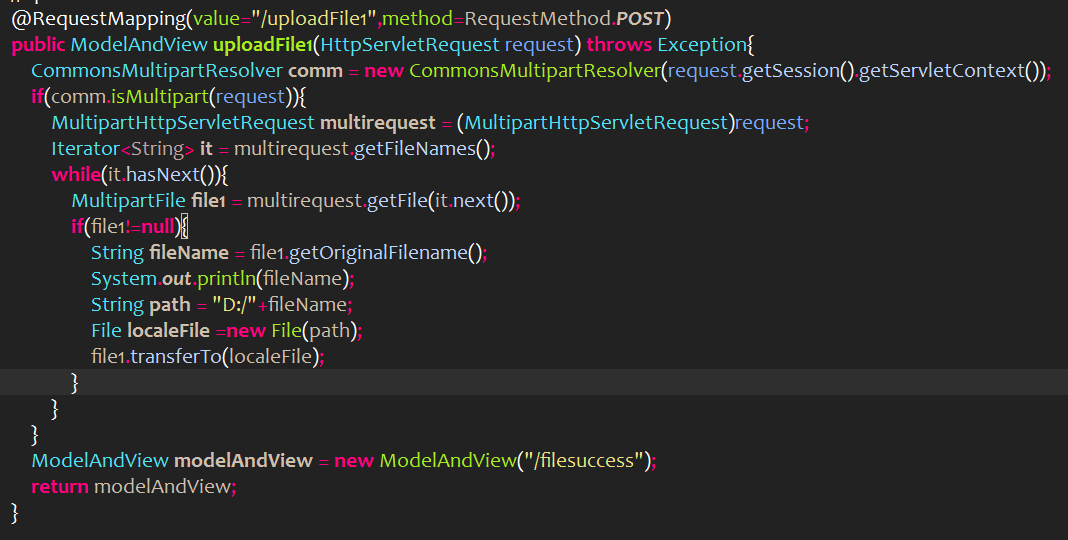
## （5）创建文件径路

## （6）创建文件对象File

## （7）将上传的文件转移到目的地

## 示例

具体代码：



# MultipartFile接口

这个接口的作用是实现文件上传的功能。多部分请求中接收到的上传文件的表示形式，文件内容要么存储在内存中，要么临时存储在磁盘上，在这两种情况下，用户都要根据需要将文件内容复制到会话级别或持久存储中。在请求处理结束时，将清除临时存储。

## 方法

### String getName()

在multipart表单中返回参数的名称

### String getOriginalFilename();

返回客户端文件系统中的原始文件名，这可能包含路径信息，这取决于所使用的浏览器，但它通常不会与Opera以外的任何浏览器一起使用。

返回原始的文件名称，如果在multipart表单中没有选择文件，则返回空字符串。

### boolean isEmpty();

返回上传的文件是否为空，即在多部分表单中没有选择文件，或者选择的文件没有内容。

### String getContentType();

返回文件的内容类型。

### long getSize();

以字节为单位返回文件的大小。

### byte[] getBytes()

以字节数组的形式返回文件的内容。

### InputStream getInputStream()

返回一个InputStream来读取文件的内容。用户负责关闭流。

### void transferTo(File dest)

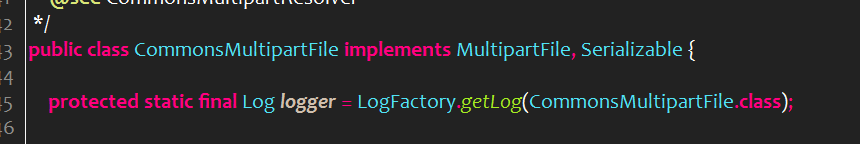
将接收到的文件传输到给定的目标文件。参数：目标文件

### public final FileItem getFileItem()

返回底层FileItem实例。几乎没有必要访问它。

# CommonsMultipartFile类

这个类用于实现文件上传，这个类实现了MultipartFile接口。具体的方法看MultipartFile接口。源码如下：



# MultipartHttpServletRequest接口

这个接口继承了HttpServletRequest, MultipartRequest这两个接口，提供其他方法来处理servlet请求中的多部分内容，允许访问上传的文件，实现还需要覆盖标准的ServletRequest的方法对于访问参数，使多部分参数可利用。

## 方法

### HttpMethod getRequestMethod()

将此请求的方法作为一个方便的HttpMethod实例返回

### HttpHeaders getRequestHeaders();

将此请求的标头作为一个方便的HttpHeaders实例返回。

### HttpHeaders getMultipartHeaders(String paramOrFileName)

返回与多部分请求的指定部分相关联的标头

### Iterator<String> getFileNames();

返回包含此请求中包含的多部分文件的参数名称的字符串对象的迭代器。这些是表单的字段名(与普通参数类似)，而不是原始文件名。

### MultipartFile getFile(String name);

参数：指定多部分文件的参数名的字符串

返回以MultipartFile对象的形式上传的内容

在此请求中返回上传文件的内容和描述，如果它不存在则返回null。

# MultipartResolver接口

一种多部分文件上传解析的策略接口，通常实现在应用程序上下文中和独立时都可用，在Spring中只有一个具体的实现，即CommonsMultipartResolver类。要定义实现，请创建一个id为“multipartResolver”的bean在DispatcherServlet应用程序上下文，这样的解析器应用于由DispatcherServlet处理的所有请求。

## 方法

### boolean isMultipart(HttpServletRequest request);

确定给定的请求是否包含多部分内容。

### MultipartHttpServletRequest resolveMultipart(HttpServletRequest request)

将给定的HTTP请求解析为多部分文件和参数，将请求包装在MultipartHttpServletRequest对象中，该对象提供对文件描述符的访问，并通过标准的ServletRequest方法访问包含的参数。返回包装好的servlet请求。

### void cleanupMultipart(MultipartHttpServletRequest request)

清理用于多部分处理的任何资源，比如用于上传文件的存储。

# CommonsMultipartResolver类

这个类实现了MultipartResolver接口，提供了maxUploadSize、maxInMemorySize、defaultEncoding设置作为bean的属性（继承自CommonsFileUploadSupport类），将临时文件保存到servlet容器的临时目录中，需要由应用程序上下文或通过接受servlet上下文(独立使用)的构造函数进行初始化。

# HttpEntity

表示HTTP请求或响应实体，由头和主体组成，通常与RestTemplate组合使用，也可以在Spring MVC中使用，作为Controller方法的返回值。

## 构造函数

### protected HttpEntity()

创建一个新的空的HttpEntity类型

### public HttpEntity(T body)

创建一个新的并且有Body没有header的HttpEntity类型

## 方法

### public HttpHeaders getHeaders()

返回实体的header

### public T getBody()

返回实体的body

# ResponseEntity

HttpEntity类型的扩展，并且添加了一个状态码。也可以在Spring MVC中使用，作为Controller方法的返回值。封装成ResponseEntity是目前能够返回多种数据类型的方案，想返回String的话可以直接用@ResponseBody注解。

## 构造函数

### public ResponseEntity(HttpStatus statusCode)

用给定的状态码创建一个新的ResponseEntity对象，没有body和header。

### public ResponseEntity(T body, HttpStatus statusCode)

用给定的body和状态码创建一个新的ResponseEntity对象，没有header。

### public ResponseEntity(MultiValueMap<String, String> headers, HttpStatus statusCode)

创建一个新的HttpEntity对象，包含给定的头和状态代码，没有body

### public ResponseEntity(T body, MultiValueMap<String, String> headers, HttpStatus statusCode)

用给定的body、header、状态码去创建一个新的HttpEntity对象

# SpringMVC实现文件下载

我们通过两种方法去实现。

## 基于ResponseEntity实现

### （1）编写前台页面

<div>

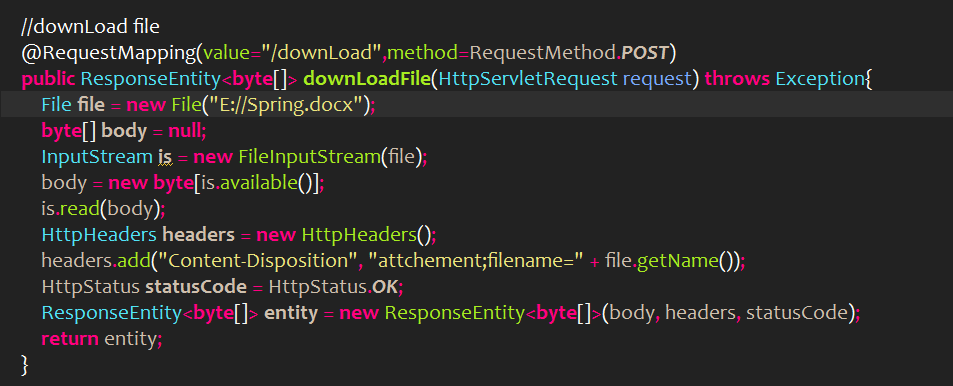
<form name="downloadForm" action="downLoad" method="post">

<input type="submit" value="下载文件"/>

</form>

</div>

### （2）编写后台代码实现



## Java通用下载实现

### （1）编写前台页面

<div>

<form name="downloadForm1" action="downLoad1" method="post">

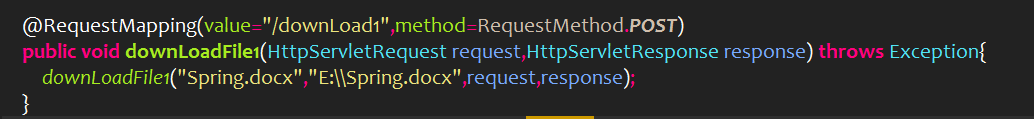
<input type="submit" value="下载文件"/>

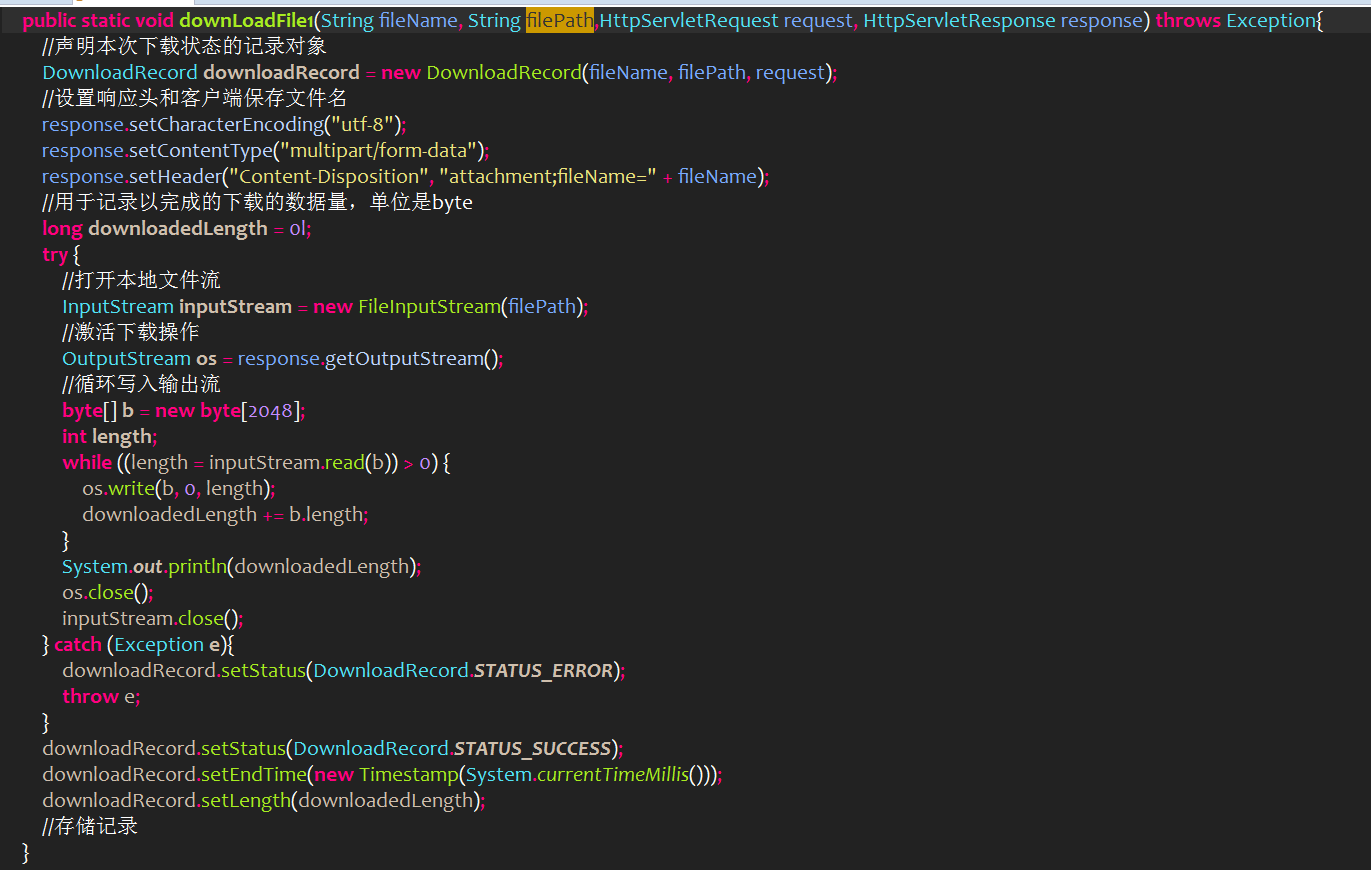
</form>

</div>

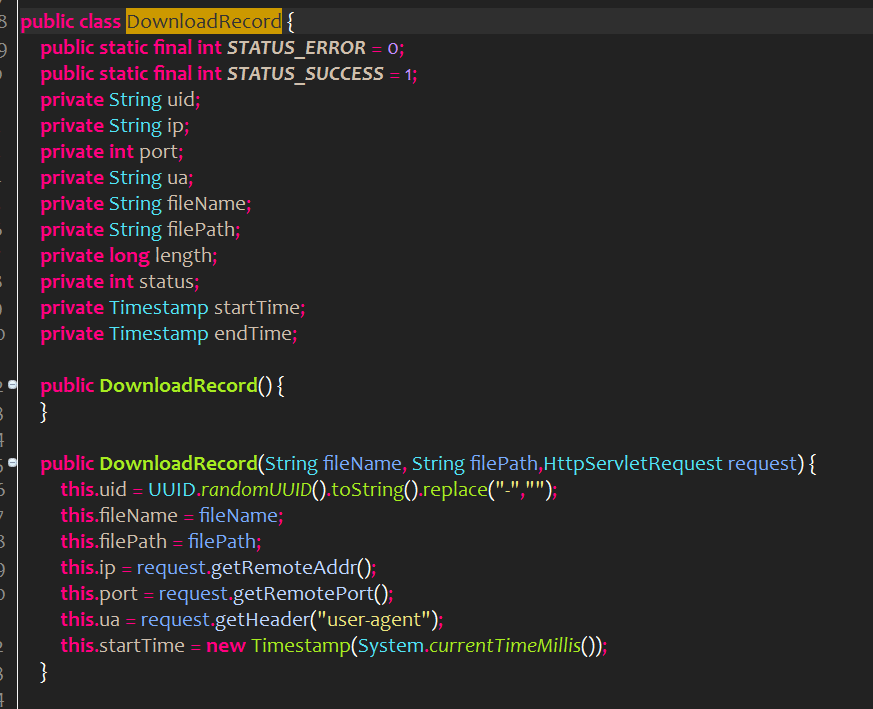
### （2）编写后台代码

下载过程的实现，在java中调用reponse.getOutputStream()方法会自动激活下载





记录信息实体类



## 两种方法的比较

基于ResponseEntity的实现的局限性还是很大，从代码中可以看出这种下载方式是一种一次性读取的下载方式，在文件较大的时候会直接抛出内存溢出（我自己亲测一个1.8G的文件在执行下载操作的时候直接抛出了内存溢出）。还有就是这种方式在进行下载统计的时候也存在局限性，无法统计在下载失败的情况已完成下载量，因此限制了对下载的功能扩展。虽然这种实现方式有局限性，但是也有着优点——简洁。在很多时候我们并不需要那么复杂的下载功能时，这种实现就应该是首选了。

下载java通用实现在功能上比第一种实现更加丰富，对下载的文件大小无限制（循环读取一定量的字节写入到输出流中，因此不会造成内存溢出，但是在下载人数过多的时候应该还是出现一些异常，不过下载量较大的文件一般都会使用ftp服务器来做吧），另外因为是这种实现方式是基于循环写入的方式进行下载，在每次将字节块写入到输出流中的时都会进行输出流的合法性检测，在因为用户取消或者网络原因造成socket断开的时候，系统会抛出SocketWriteException，系统可以捕捉这个过程中抛出的异常，当捕捉到异常的时候我们可以记录当前已经传输的数据量，这样就可以完成下载状态和对应状态下载量和速度之类的数据记录。另外这种方式还可以实现一种断点续载的功能。

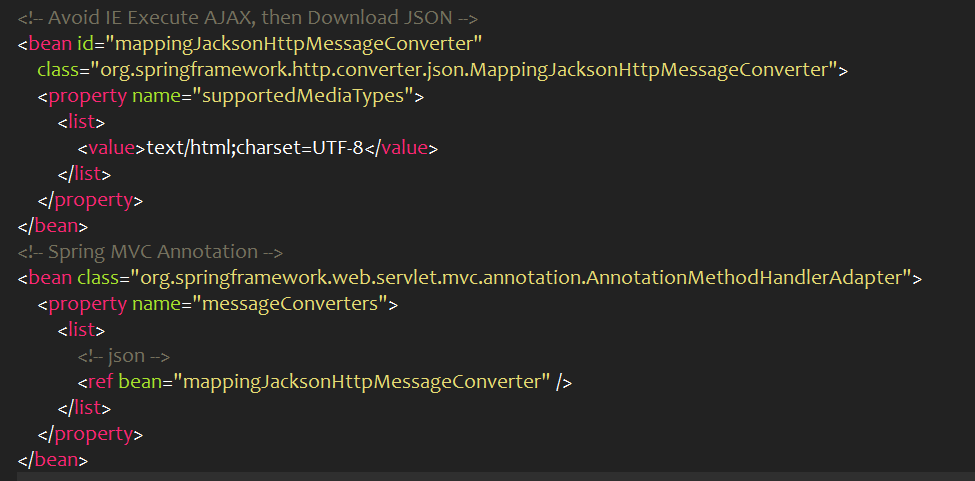
# AnnotationMethodHandlerAdapter类

## 基本概念

这个类是HandlerAdapter接口的实现，该接口映射基于HTTP路径、HTTP方法和通过RequestMapping注解表示的请求参数的处理程序方法。通过RequestParam注释支持请求参数绑定。还支持用于向视图公开模型属性值的ModelAttribute注释，以及用于绑定器初始化方法的InitBinder注解和用于特定属性的自动会话管理的sessionattributes注解。可以通过各种bean属性定制这个适配器。一个常见的用例是通过自定义的WebBindingInitializer应用共享绑定器初始化逻辑。

这个类是用于配置Spring MVC的注解配置，用于配置在Spring MVC的配置文件中。

## 代码示例



# HttpMessageConverter接口（重点）

策略接口，指定一个可以从HTTP请求和响应进行转换的转换器。

## 方法

### boolean canRead(Class<?> clazz, MediaType mediaType)

指示此转换器是否可以读取给定的类

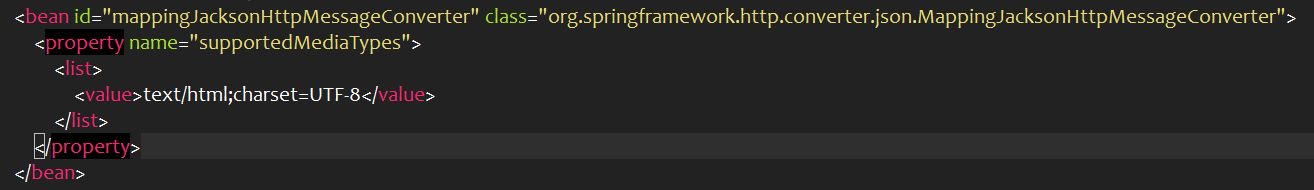
### boolean canWrite(Class<?> clazz, MediaType mediaType);

指示该转换器是否可以编写给定的类。

# MappingJacksonHttpMessageConverter类

这个类实现了HttpMessageConverter接口，可以使用Jackson读写Json，此转换器可用于绑定到类型化bean或非类型化HashMap实例，默认情况下，这个转换器支持application/json，这可以通过设置supportedmediatypes属性来重写。

## 代码示例



# DefaultServletHttpRequestHandler类

# SpringMVC中的拦截器（重点）

定义springmvc拦截器的两种方法

## （1）实现HandlerInterceptor接口

定义类并实现HandlerInterceptor接口，接口中有如下方法需要重写，注意：参数中的Object handler是下一个拦截器。

public boolean preHandle

(HttpServletRequest request,HttpServletResponse response,

Object handler) throws Exception

该方法在action执行前执行，可以实现对数据的预处理，比如：编码、安全控制等。

如果方法返回true，则继续执行action。

public void postHandle

(HttpServletRequest request,HttpServletResponse response,

Object handler,ModelAndView modelAndView) throws Exception

该方法在action执行后，生成视图前执行。在这里我们有机会修改视图层数据。

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

最后执行，通常用于释放资源，处理异常。我们可以根据ex是否为空，来进行相关的异常处理。因为我们在平时处理异常时，都是从底层向上抛出异常，最后到了spring框架从而到了这个方法中。

### 代码示例

public class MyInterceptor implements HandlerInterceptor {

@Override

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

System.out.println("最后执行！！！一般用于释放资源！！");

}

@Override

public void postHandle(HttpServletRequest request,HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {

System.out.println("Action执行之后，生成视图之前执行！！");

}

@Override

public boolean preHandle(HttpServletRequest request,HttpServletResponse response, Object handler) throws Exception {

System.out.println("action之前执行！！！");

return true; //继续执行action

}

}

## （2）继承HandlerInterceptorAdapter类

继承org.springframework.web.servlet.handler.HandlerInterceptorAdapter，这个适配器实现了HandlerInterceptor接口，提供了这个接口中所有方法的空实现。

### 代码示例

package com.sxt.interceptor;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

public class MyInterceptor2 extends HandlerInterceptorAdapter {

@Override

public boolean preHandle(HttpServletRequest request,HttpServletResponse response, Object handler) throws Exception {

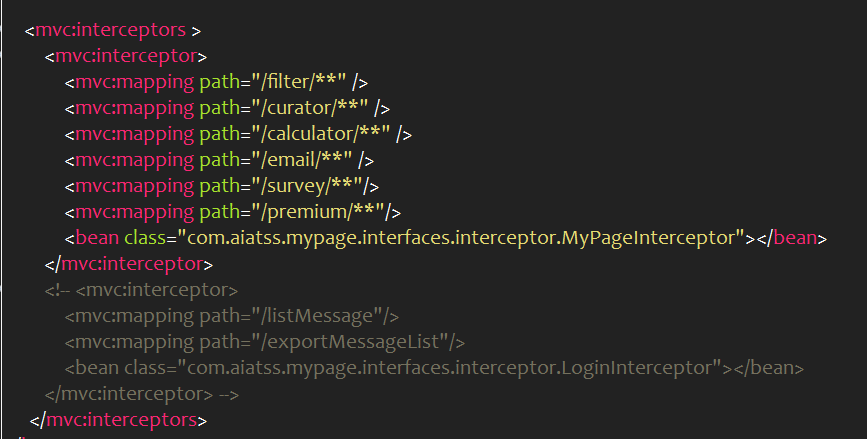
System.out.println("MyInterceptor2.preHandle()");

return true; //继续执行action

}

}

## springMVC配置文件中配置拦截器



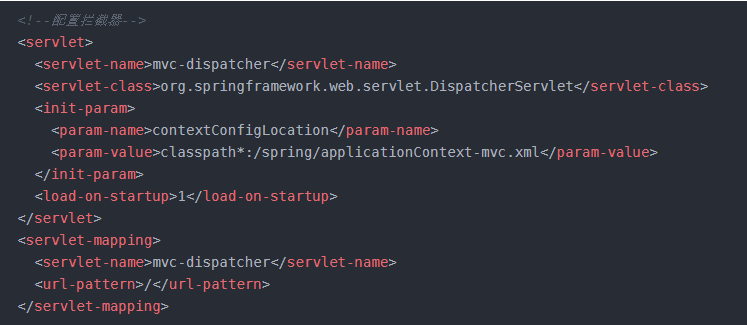
说明：在一个<mvc:interceptor>元素中只能配置一个bean元素，不能配置两个bena元素，否则启动服务器就报错。

# 遇到的问题

## 1、No mapping found for HTTP request with URI

问题描述：项目正常启动，可以访问页面，但是无法找到静态资源文件，如css，js等文件资源。

我之前的拦截器配置



从配置上分析，如此配置会把所有的请求都会进行拦截，交给spring去处理。而spring所有请求的URL都是在controller中使用注解@RequestMapping标明，所以这样的情况下访问静态资源是访问不到的。

以下给出三种解决方案，自行选择使用：

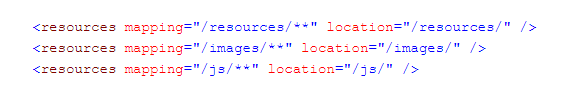
（1）简单粗暴地告诉服务器，这些资源不需要拦截

直接在web.xml文件中进行配置，如下：



（2）直接告诉spring，这个你就得这么做

在MVC配置文件中添加资源文件映射



说明：mapping：映射，location：本地资源路径，注意必须是webapp根目录下的路径。两个\*，它表示映射resources/下所有的URL，包括子路径（即接多个/）配置的location一定要是web-app根目录下才行，如果你将资源目录，放置到web-app/WEB-INF下面的话，恭喜你，访问失败！404 NOT FOUND

（3）全部使用默认的，也别操啥心了



比较懒得童鞋可以直接在MVC配置文件加上这么一句，这个确实方便，但是弊端就是他会把你发出的请求全部按照默认的处理了。

## 2、No mapping found for HTTP request with URI [/Springmvc/test] in DispatcherServlet with dispatcher

突然出现了这个问题，这个就是说明你的httpurl没有映射出来，也就是说的你controller没有注入。通常出现这种问题有2种情况：

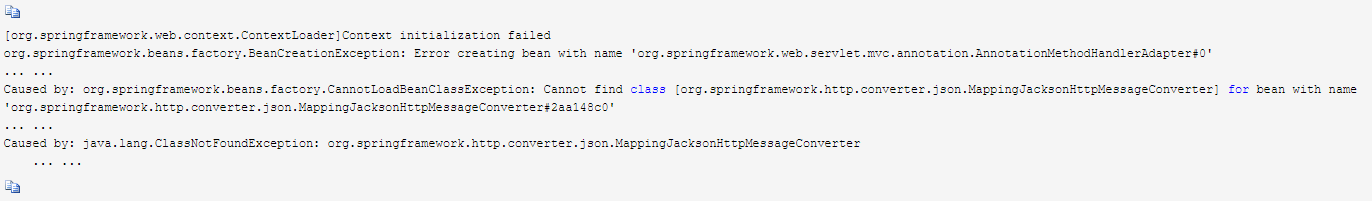
（1）在springmvc-config.xml即spring配置文件中 加上这样一句话



（2）spring配置文件在src目录下，但是我的springmvc配置文件丢在了web-inf下，虽然没有报错，但是还是不能注入我的controller，所以，当我放在一起的时候，问题就解决了。

## 3、[Spring3升级到Spring4时， 运行时出现找不到MappingJacksonHttpMessageConverter的情况](https://www.cnblogs.com/shihaiming/p/6702572.html)

先看报错截图：



原因是Spring 3.x 和4.X处理JSON的一个类不一样，而这个东西又配置在xml文件中，所以编译时又无法发现。

spring3.x是org.springframework.http.converter.json.MappingJacksonHttpMessageConverter

spring4.x是org.springframework.http.converter.json.MappingJackson2HttpMessageConverter

解决方法：

查找替换所有XML中名字

同时加入FastJackson依赖

