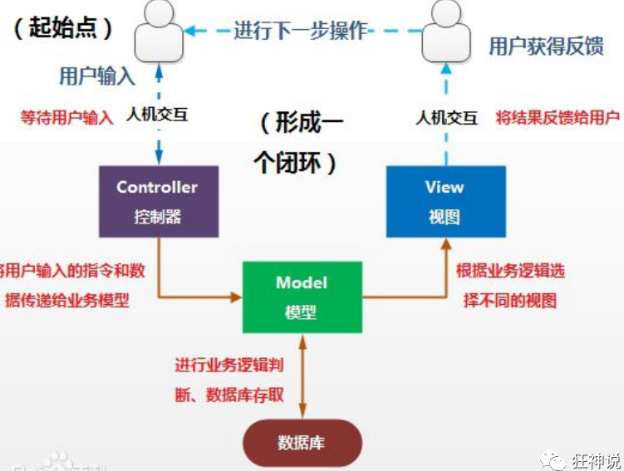
# SpringMVC

## 一、SpringMVC基本概念

### 1.MVC回顾

开发架构一般有两种形式：一种是C/S结构，即客户端/服务器；一种是B/S架构，即浏览器/服务器。

MVC三层架构是一种架构模式，**主要作用**是降低视图和业务逻辑间的双向耦合关系。其中Model是数据模型层，包含了model(实体类)、service(业务逻辑层)、dao(持久层)；View是视图层，主要进行页面数据的展示，如html、jsp等。Controller层，一般为servlet,主要用来接收客户请求，然后根据请求调用数据模型层中的service处理不同的业务，处理完之后将数据返回给视图。



常用的服务器端MVC框架有：Structs2、SpringMVC等。常见的前端MVC框架有vue等。

### 2.SpringMVC

SpringMVC是一种基于java实现MVC的轻量级Web框架。它可以通过一套注解，让一个简单的java类成为处理请求的控制器，而无需实现任何接口。

SpringMVC框架是MVC框架，处于表现层；Mybatis框架是持久层框架，处于持久层(或者称数据访问层)。

#### ⑴SpringMVC的优势：

**①**角色划分清晰：前端控制器(DispatcherServlet)、请求到处理器映射(HandlerMapping)、处理器适配器(HandlerAdapter)、视图解析器(ViewResolver)、控制器(Controller)、验证器(Validator)、命令对象(Command **请求参数绑定到的对象就叫命令对象**)、表单对象(Form Object **提供给表单展示和提交到的对象就叫表单对象**)

**②**分工明确，扩展灵活，高效，基于请求相应的MVC框架。

**③**能和Spring框架无缝结合。

**④**命令对象就是一个POJO，可以使用命令对象直接作为业务对象。

**⑤**可适配，通过HandlerAdapter可以支持任意的类作为处理器。

**⑥**有强大的数据验证、格式化、绑定机制、RESTful。

**⑦**强大的JSP标签库，使jsp编写更容易。

#### ⑵SpringMVC和Structs2的优略分析

**共同点**：

**①**它们都是表现层框架，都是基于MVC模型编写的。

**②**它们的底层都离不开原始ServletAPI。

**③**它们处理请求的机制都是一个核心控制器。SpringMVC的核心控制器是一个Servlet，Structs2的核心控制器则是一个过滤器。都是基于前端控制器的开发模式。

**区别**：

**①**SpringMVC的入口是Servlet，而Structs2的则是Filter。

**②**SpringMVC是基于方法设计的，而Structs2则是基于类。Structs2框架可以理解为多例，每发送一次请求都会创建一个Structs2的框架，然后执行方法处理请求。而SpringMVC的对象则是单例的，只会创建一个实例对象，每接收一个请求都会调用方法处理请求。

**③**SpringMVC使用更加简洁，同时还支持JSR303。

**④**SpringMVC集成了Ajax，处理ajax的请求非常方便。

**⑤**Structs2的OGNL表达式使页面的开发效率相比SpringMVC要高些，但执行效率并没有比JSTL提升。

**⑥**SpringMVC的开发效率和性能高于Structs2。

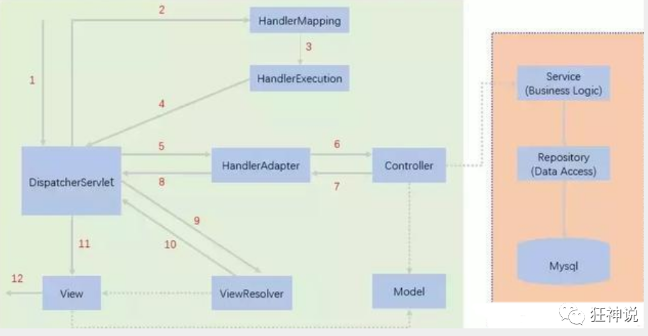
**⑦**在设计思想上，Stucts更符合OOP的编程思想，而SpringMVC就较谨慎，是在Serlvet上扩展。

**JSR303**：是一套JavaBean参数校验的标准，它定义了很多常用的校验注解，我们可以直接将这些注解加在我们JavaBean的属性上面，这样就可以在需要校验的时候进行校验。

## 二、SpringMVC的原理

Spring的web框架围绕DispatcherServlet设计。DispatcherServlet实际上就是一个Servlet，它继承了HttpServlet。**DispatcherServlet(前端控制器)的作用是将请求分发到不同的处理器**。

### 1.SpringMVC的执行原理(重要)



图为SpringMVC一个较完整的流程图，实线表示SpringMVC提供的技术，不需要开发者实现，虚线表示需要开发者实现。

**简要分析执行流程**：

1.DispatcherServlet表示前端控制器：用户发送请求，DispatcherServlet接收并拦截请求。

如请求的url为：http:localhost:8080/SpringMVC/hello，这个url拆分为三部分，<http://localhost:8088>为服务器域名，SpringMVC则是部署在服务器上的web站点，hello表示控制器。这个url表示请求位于服务器localhost:8080上的SpringMVC站点的hello控制器。

2.HandlerMapping表示处理器映射：DispatcherServlet调用HandlerMapping，HandlerMapping根据请求的url查找相应的Handler(页面控制器，也称处理器)，然后返回一个执行链(HandlerExecutionChain)。

3.HandlerExecutionChain：它包含一个Handler(页面控制器)和多个HandlerInterceptor拦截器。Handler根据url查找控制器，如url中被查找的控制器为：hello。前端控制器接收这个请求执行链。

5.HandlerAdapter表示处理器适配器。它按照特定的规则去执行Handler。

6. Handler让具体的Controller执行。

7. Controller将具体的执行信息返回给HandlerAdapter,如ModelAndView。

8. HandlerAdapter将ModelAndView（模型数据和逻辑视图名）传递给DispatcherServlet。

9. DispatcherServlet调用视图解析器(ViewResolver)来解析HandlerAdapter传递的ModelAndView

10. 视图解析器将解析的逻辑视图名传给DispatcherServlet。

11. DispatcherServlet根据视图解析器解析的视图结果，调用具体的视图。

12. 最终视图呈现给用户。

**原理总结(非常重要)**：**用户发送请求，该请求被前端控制器(DispatcherServlet)接收到，前端控制器根据请求中的信息(如url)选择对应的处理映射器(HandlerMapping)，处理映射器根据请求的url查找对应的Handler(页面控制器)，然后给前端控制器返回一个处理执行链(HandlerExecutionChain)，这个处理执行链中就包含了Handler。前端控制器选择对应的处理适配器(HandlerAdapter)，处理适配器去执行Handler(也称为Controller)，执行之后返回一个ModelAndView(模型数据和视图)，然后处理适配器将ModelAndView返回给前端控制器。前端控制器再调用视图解析器(ViewResolver)去解析ModelAndView，解析之后返回一个视图名称给前端控制器，前端控制器根据视图名称查找对应的视图，然后将模型数据传入视图中进行渲染。视图渲染完毕，前端控制器再次收回控制权，并将响应返回给用户。至此整个过程结束**。

**SpringMVC的核心三要素**：处理映射器(HandlerMapping)、处理适配器(HandlerAdapter)、视图解析器(ViewResolver)。

不管是采用配置xml方式，还是采用注解方式，SpringMVC都必须配置上面的三大件：**处理映射器、处理适配器、视图解析器**。只不过采用纯xml方式时，这三个都要手动配置到Spring容器中。而采用注解方式，处理映射器和处理适配器则被**<mvc:annotation-driven>**帮我们注入到Spring容器中了，只需手动配置视图解析器即可。

## 二、SpringMVC使用配置的方式(过时了)

**步骤一**：导入相应的依赖，需要使用spring-webmvc、servlet、jsp的依赖。要使用SpringMVC需要导入spring-webmvc这个依赖，导入了这个依赖之后，项目中会自动添加spring-context、spring-webmvc、spring-web等jar包。

**步骤二**：在web.xml中配置一个前端控制器(DispatcherServlet)，并关联一个springMVC的配置文件。这个前端控制器是一个Servlet，就和以前在xml中配置servlet一样。 在关联springMVC的配置文件时，<param-name>标签中的内容是固定的，<param-value>则用来指定springMVC的配置文件的位置，可以使用classpath表示类路径下，也可以将springMVC的配置文件放在WEB-INF目录下。



在SpringMVC中，<url-pattern>**/**</url-pattern>**表示只匹配所有的请求，不包括匹配jsp页面，一般直接用这个就可以。**<url-pattern>**/\***</url-pattern>**则表示会匹配所有的请求，包括jsp页面**，这种情况会出现返回 jsp视图 时再次进入spring的DispatcherServlet 类，导致找不到对应的controller所以报404错。<url-pattern>**\*.do**</url-pattern>则表示当请求是xx.do时，这个请求会被DispatcherServlet接收。

**步骤三**：编写springMVC的配置文件。使用spring-beans的约束，将SpringMVC中的处理映射器(HandlerMapping)、处理适配器(HandlerAdapter)、视图解析器(ViewResolver)注入到Spring的容器中。

**步骤四**：编写一个XxxController类，实现Controller接口，重写相应的方法。

**步骤四**：将XxxController注入到Spring的容器中。

**完整例子：**

**导入的依赖**：



**web.xml中配置前端控制器**



**spring-servlet.xml配置文件**：

**写法一**：使用SimpleUrlHandlerMapping处理映射器，然后配置访问Controller的路径。



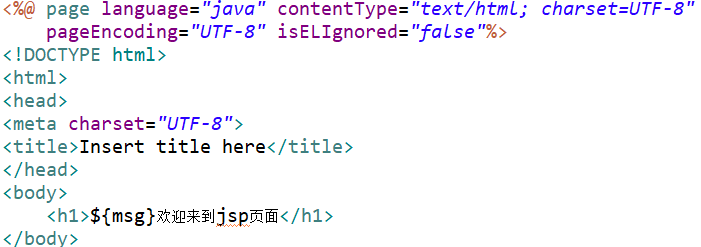
**写法二**：使用BeanNameUrlHandlerMapping，然后直接将Controller注入到Spring容器中的id属性作为访问路径。



**编写Controller类：采用实现Controller接口的方式**

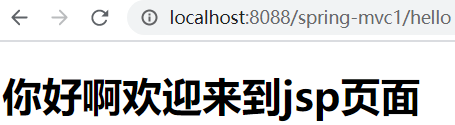


**index.jsp页面**：



**运行这个SpringMVC项目**：

先将项目清理(mvn clean)，然后将项目打包(mvn package)，注意一定要打包，不然tomcat可能报错，然后刷新这个项目，若这个项目中target目录下有打包好的war包，则可以直接右键项目🡪Run as Server—>选择tomcat运行，然后会跳转到<http://localhost:8088/spring-mvc1/>，之后直接输入访问Controller的urlhello，即<http://localhost:8088/spring-mvc1/hello>，则访问了该Controller之后就可以响应对应的index.jsp的内容。



**注意：若jsp页面的EL表达式没有效果，则要在<%@Page%>中添加isELIgnored=false才行，因为springMVC中默认这个是true**。

## 三、SpringMVC中使用注解方式

### 1.常用的注解和标签

#### ⑴@Controller注解：

这个是Spring中的注解，用于将Controller类的注入到Spring容器中。同时也表明这个Controller类会经过视图解析器。

#### ⑵@RestController注解：

也可用来注解Controller类，被这个注解修饰的Controller类下的所有方法都不会经过视图解析器。若方法的返回值若为String类型，则直接返回一个字符串。相当于@Controller注解和@ResponseBody注解的结合。

**只使用@Controller注解修饰的Controller类，会经过视图解析器解析。使用@RestController注解修饰的Controller类，不会经过视图解析器解析。@ResponseBody注解配合@Controller注解使用时，被@ResponseBody修饰的方法不会经过视图解析器，而没被@ResponseBody修饰的方法则会经过视图解析器解析**。

#### ⑶@RequestMapping注解：

该注解是用来映射请求路径，可用在类上，也可用在方法上。该注解有**value属性、path属性**：用来指定映射的url，若只有url则value可以省略，**value属性是注解的默认值**。**method属性**：用来指定请求的类型，如RequestMethod.***POSt、***RequestMethod.***GET***等。**name属性**：为该映射起一个名字，而不表示该映射的具体路径。**consumes属性**：指定处理请求的提交内容类型(Content-Type)，如：application/json，text/html。**produces属性**：指定返回的内容类型，仅当request请求头中的(Accept)类型中包含该指定类型才返回。**params属性**： 指定request中必须包含某些参数值是，才让该方法处理。**headers属性**： 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。

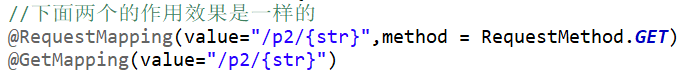


要想要使该注解生效，必须要向Spring容器中注入DefaultAnnotationHandlerMapping和AnnotationMethodHandlerAdapter的实例。这两个实例分别是在类级别和方法级别处理。**<mvc:annotation-driven>**标签会将这个两个实例自动注入到spring容器中。

若类上使用的这个注解同时方法上也使用了这个注解则请求路径为：localhost:8080/类上的url/方法上的url。若只有方法上使用了这个注解，则请求路径为localhost:8080/方法上的url。

#### ⑷@GetMapping注解：

该注解的作用是指定请求方式为Get类型，其value属性也可以用来指定映射路径，当只有value属性时value可以省略。同类型的注解有**@PostMapping、@DeleteMapping、@PutMapping**等。



**注意**：所有地址栏请求默认都是get的方式。

#### ⑸<mvc:default-servlet-handler>标签：

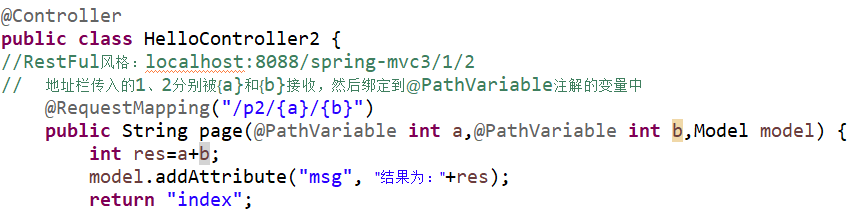
使用SpringMVC时，在web.xml文件中配置DispatcherServlet时，在<url-pattern>标签中配置的内容为/，这表示DispatcherServlet将接收所有除了jsp的请求，它把所有的请求都看成是一个普通的请求，包括静态资源的请求。对于静态资源的请求被看成一个普通的后台控制器请求，这样会导致请求找不到报404错误。因此对于静态资源的请求，SpringMVC提供了一个<mvc:default-servlet-handler>标签。该标签的作用是在Web容器启动时在上下文中定义一个DefaultServletHttpRequestHandler，它会对DispatcherServlet的请求进行筛选，如果是静态请求，则该请求会交由Web应用服务器默认的Servlet处理，若不是静态资源的请求则交由DispatcherServlet处理。这样就可以解决上面报的404问题。

#### ⑹<mvc:annotation-driven>标签：

该标签的作用是配置支持注解驱动。这个标签会将DefaultAnnotationHandlerMapping和AnnotationMethodHandlerAdapter的实例注入到Spring容器中。

#### ⑺@PathVariable注解：

该注解用在方法的参数上，可以将@RequestMapping注解中的URL中占位符{xxx}参数绑定到方法中被@PathVariable注解的变量中。这个注解和@RequestParam注解的区别是：**@RequestParam用来获得静态的URL请求参数；@PathVariable用来获得动态的URL请求参数**。





#### ⑻@RequestParam注解：

该注解用在方法参数上，当前端向后端发送数据时，数据名称和方法中参数名称不一致时，使用该注解修饰方法中的参数时，则方法中参数可以和指定的数据名匹配，从而获得数据。**该注解的作用类似于request.getParameter()方法，因此可以接收Get方式拼接在路径后面的数据，也可以接收Post方式中在请求体中的数据**。该注解有常用的属性为，**defaultValue属性**：表示给参数设置默认值；**required属性**：值为true/false表示是否必须传入参数。**name、value属性**：这两个属性的效果是一样的，都是绑定请求参数中的名称，默认是这个属性。

@RequestParam注解的值为路径中拼接的数据名称。



**注意**：如果方法的参数前写了@RequestParam(xxx)，那么前端在路径上拼接的数据必须有对应的xxx名字才行(不管其是否有值，当然可以通过设置该注解的required属性来调节是否必须传)，如果没有xxx名的话，那么请求会出错，报400。若方法的参数前没有使用@RequestParam注解，则前端在路径上拼接的数据必须和方法中的参数名一致才能进行数据的映射，否则后端方法中的参数将获取不到值，为null，路径上不拼接数据或者拼接的数据名和方法中的参数名不一致时，请求也能正常发送，只不过方法中的参数将获取不到值。

该注解可用来处理请求中Content-Type类型为：application/x-www-form-urlencoded编码的内容，提交方式为GET、POST的数据。

直接在路径上拼接数据的方式是Get方式的请求，a标签、window.href默认是get方式的请求。

#### ⑼@ResponseBody注解：

该注解可使用在控制器的方法上，作用是将方法的返回值以特定格式写入到response的body中，进而将数据返回给客户端。**注意**使用了此注解后，该方法不会再走视图解析器，而是直接将数据写入到输入流中，作用等同于直接使用response对象输出指定格式的数据。如果方法的返回值是字符串，则直接将字符串返回给客户端；若方法的返回值是一个对象，则会将该对象转化成json格式的字符串，然后再返回给客户端(这个的前提是导入了Jackson的依赖或者fastjson的依赖等，同时对象的属性必须有对应的get方法)。

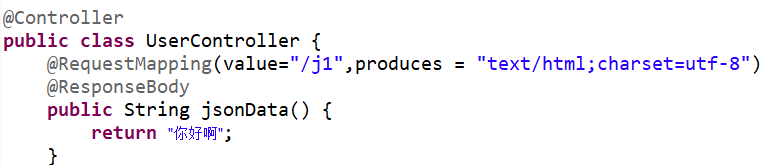
这里**将返回值为对象转换成json字符串的原理**：其实是通过**HttpMessageConverter**中的方法实现的，它本是一个接口，在其实现类完成转换，如Jackson包中的实现类，若没有导入相关的json转换的依赖，则会报没有converter错误。如果是bean对象，会调用对象的getXXX()方法获取属性值并且以键值对的形式进行封装，进而转化为json串。如果是map集合，采用get(key)方式获取value值，然后进行封装。

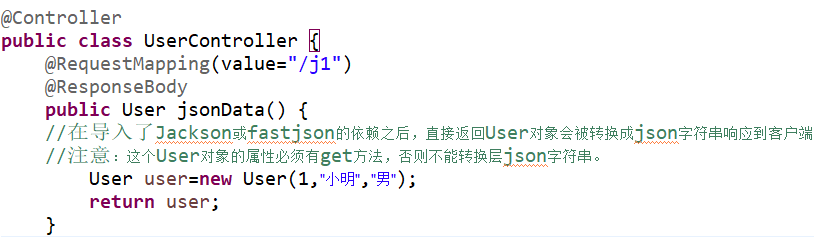
使用时机：返回的数据不是html标签的页面，而是其他某种格式的数据时(如json、xml等)使用。

**注意**：如果方法返回的是对象，客户端接收的是json格式的字符串，默认是按utf-8编码；若返回的是String类型的字符串，则默认是按iso8859-1编码，因此页面可能乱码；因此可在注解中指定编码格式，如在**@RequestMapping(value="/j1",produces = "text/html;charset=utf-8")**，produces中的内容用来指定编码格式。

**注意**：在使用 @RequestMapping后，返回值通常解析为跳转路径，但是加上 @ResponseBody 后返回结果不会被解析为跳转路径，而是直接写入 HTTP response body 中。

**例子**：



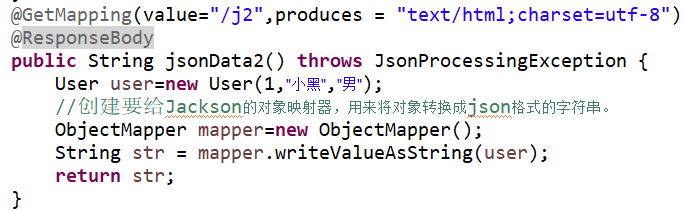
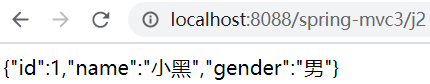




**例子2**：



实体类：

#### ⑽@RequestBody注解：

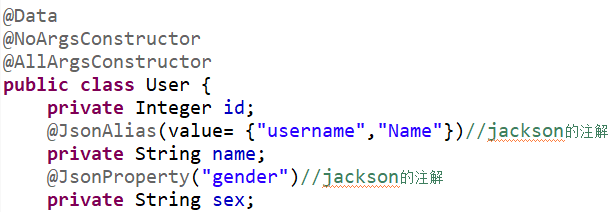
该注解是使用在控制器的方法的参数上，常用于将前端发送过来的固定格式的数据(如xml，json)封装到方法的参数为javaBean类型的对象中。**用于获取请求体中的内容，直接使用得到的是key-value结构的数据，get方式不适用，适用post方式**。

**注意**：这个从前端中发送给后端的数据必须是在请求体中的数据，即必须是POST方式提交的数据。因为GET方式无请求体，所以使用@RequestBody接收数据时，前端不能使用GET方式提交数据，而是用POST方式进行提交。

**注意**：在后端的同一个接收方法里，@RequestBody与@RequestParam()可以同时使用，@RequestBody最多只能有一个，而@RequestParam()可以有多个。即：一个请求中，只有一个RequestBody(请求体)；一个请求中，可以有多个RequestParam(请求参数)。**@RequestBody接收的是请求体中的数据，@RequestParam接收的是路径后面的key-value中的参数**。

若后端控制器的方法的参数是一个对象，且被@RequestBody注解修饰，则前端发送的json格式的数据能否映射到这个对象中有如下的情况：

**①**若json格式的字符串的key和实体类的属性名一致，则会调用实体类的对应属性的setter方法将json中相匹配的key对应的value值赋值给属性。若不一致，或者对应实体类没有相应的setter方法，则具体情况如下：



@JsonAlias注解和@JsonProperty注解的具体用法见下面的Jackson工具：[使用Jackson工具](#_1.使用Jackson工具)

若没有使用上面两种注解，且json字符串的key和实体类的属性名称不一致，则key不能和属性映射，属性的值为null。

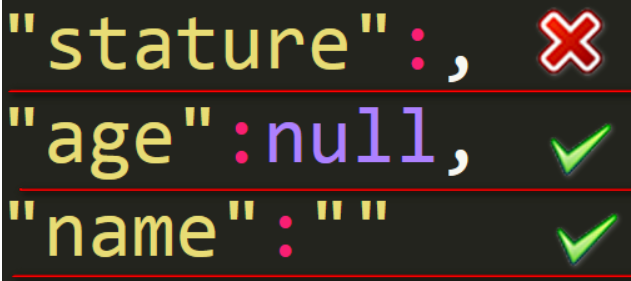
在不考虑上述两个注解的一般情况下，key与属性匹配时,默认大小写敏感。

有多个相同的key的json字符串中，转换为模型时，会以相同的几个key中，排在最后的那个key的值给模 型属性复制，因为setter会覆盖原来的值。

**②**json字符串中，若value的值为""，若实体类的属性为String类型，则该属性值为""，若为Integer、Double等，则值为null。

**③**json字符串中，若value的值为null，则封装到实体类的属性中的值也为null。

**④**在传json字符串给后端时，如果某个key对应的value值没有，则不要把该key写到json字符串中，若要写key的话则对应的value值为null或""都行，而不能写成下面打错的形式。



对于Request headers中的Content-Type的类型为application/x-www-form-urlencoded，可以使用@RequestParam或者@RequestBody处理，使用@RequestBody处理的较少。而**对于Content-Type类型为application/xml、application/json，则只能采用@RequestBody处理**。

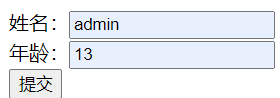
前端请求传Json对象则后端使用@RequestParam；

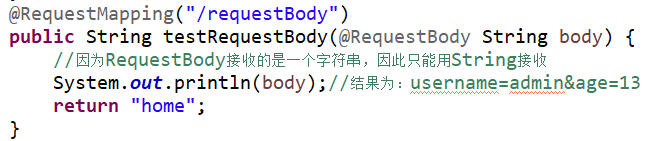
前端请求传Json对象的字符串则后端使用@RequestBody。**注意@RequestBody接收的是一个json格式的字符串，一定是一个字符串。**

总结一个@RequestParam、@RequestBody、@ResponseBody的用法，对比。

例子：





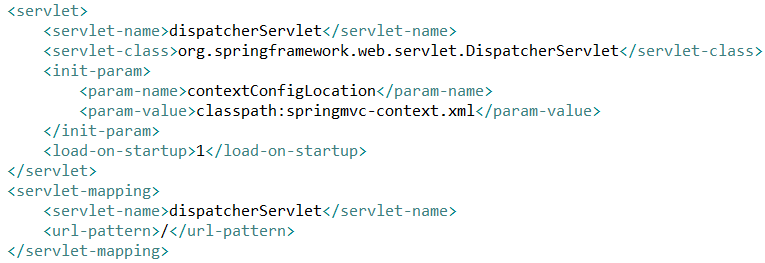


### 2.使用注解的方式

**步骤一**：导入相应的依赖。注意为了防止maven中的资源过滤问题，最好添加如下内容：



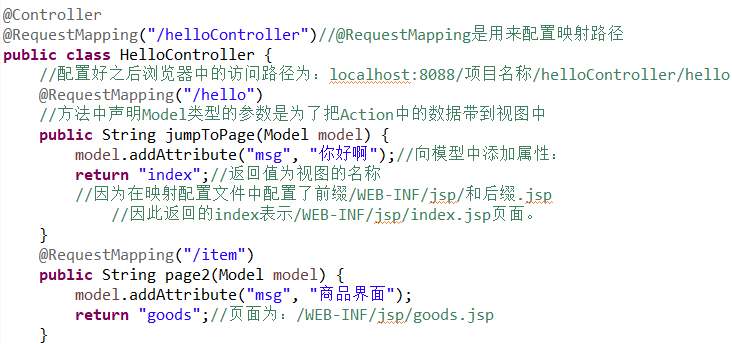
**步骤二**：在web.xml文件中配置前端控制器(DispatcherServlete)。



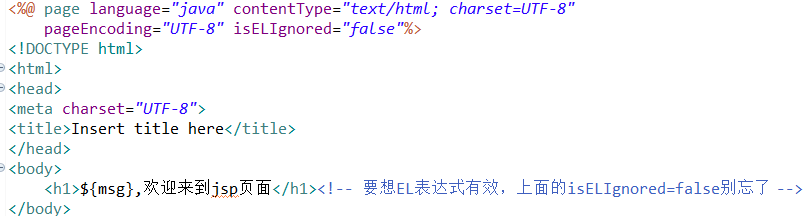
**步骤三**：创建springmvc的配置文件，并进行如下的配置。



**步骤四**：创建需要的Controller类。



**步骤五**：在WEB-INF/jsp目录下创建需要使用的jsp页面。



**步骤六**：将项目清理，打包，run as server，然后在浏览器中输入映射的路径，如下：



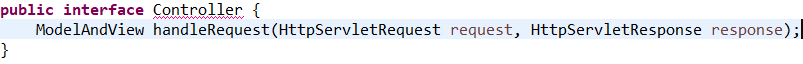
## 四、Controller和RestFul风格

### 1.Controller

**控制器通常有两种实现方式**：**①**通过实现Controller接口，重写对应方法的方法；**②**使用@Controller注解。

控制器负责解析用户的请求，并将其转换成为一个模型。一个控制器类中可以包含多个方法，每个方法对应不同的页面。

实现了Controller接口的类就获得了控制器的功能。Controller接口的handleRequest()方法是用来处理请求并返回一个ModelAndView对象。



实现Controller接口是一种较老的方法，不推荐使用，因为这种方式的控制器中只有一个方法用来返回ModelAndView对象，若需要接收不同的请求返回不同的ModelAndView对象，则要有多个控制器。

使用@Controller注解的好处：每个控制器中都可以有多个方法，且每个方法都可以返回不同的ModelAndView对象。

**注意**：被@Controller注解的类中的所有方法，若返回值是String，经过视图解析器解析后，若有页面可以跳转则跳转到相应的页面，没有则不跳。

**例子**：





分析上面的例子，可以发现针对不同的请求可以指向同一个视图，且视图显示的信息也不一样，从而使视图得到复用，也说明了控制器和视图之间是弱耦合关系。

### 2.RestFul风格

RestFul是一种软件架构模式，只是一种设计风格，而不是标准，它提供了一组设计原则和约束条件。基于这个风格设计的软件可以更简洁、更有层次、更易于实现缓存等机制。

资源：互联网所有的事物都可以被抽象为资源 。

资源操作：使用POST、DELETE、PUT、GET，使用不同方法对资源进行操作。分别对应新增、 删除、修改、查询。

**传统方式操作资源：通过传入不同的参数实现不同的效果**。如：

localhost:8080/项目名称/资源/queryUser：对应查询User操作。

localhost:8080/项目名称/资源/updateUser?id=1：对应更新操作。

localhost:8080/项目名称/资源/operation=1：当传入的值为1时在后台做判断对应查询操作，传入的值为2时对应新增操作。

传统方式在做查询操作时传入一个queryUser参数，更新操作时传入了一个updateUser参数，其实完全没必要这样。可以使用RestFul风格，通过不同的请求方式来实现不同的效果。当使用了get请求时就对应查询，使用post请求就对应新增，使用delete请求就对应删除，使用put请求就对应修改。

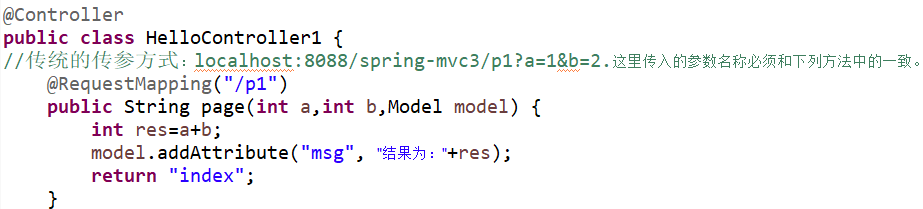
如：请求地址一样，但通过不同的请求来实现不同的效果。

localhost:8080/项目名称/资源/operation：GET请求，对应查询操作。

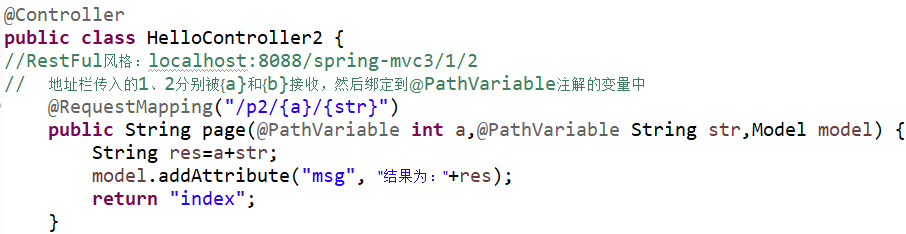
localhost:8080/项目名称/资源/operation：POST请求，对应新增操作。

localhost:8080/项目名称/资源/operation/1：delete请求，对应删除操作。

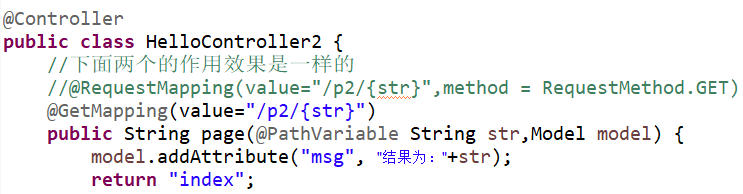
**①传统传参方式**：



**②RestFul风格**：



**指定请求方式**：



使用RestFul风格的好处：可以使路径更加简洁、高效、安全(暴露在路径中的参数不会让客户知道具体代表什么含义)。

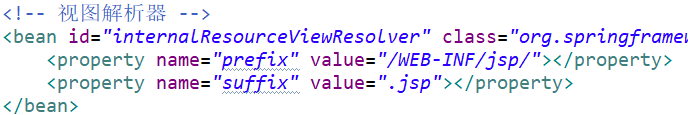
## 五、结果跳转方式

### 1.采用视图解析器(推荐这种)

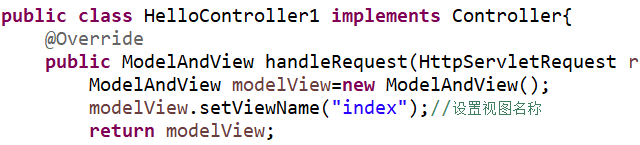
创建一个ModelAndView或者Model对象，然后设置视图名称viewName，根据视图名称和视图解析器然后跳转到指定的页面。

页面：{视图解析器前缀}+viewName+{视图解析器后缀}。

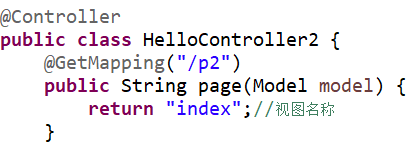
(**个人理解**)视图解析器解析ModelAndView或者对象，然后根据视图名称和视图解析器中的前后缀解析成指定的页面，然后跳转到该页面。



**法一**：



**法二**：



### 2.采用Servlet的API进行跳转(不推荐)

使用HttpServletRequest进行请求转发，使用HttpServletResponse进行重定向，使用HttpServletResponse将内容输出到页面。



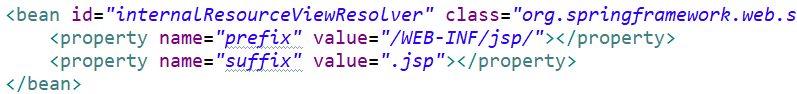
### 3.其他方式(推荐)

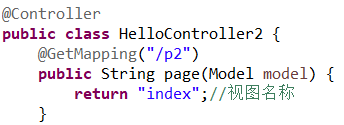
**法一**：**不使用视图解析器**。



**法二**：**使用视图解析器**。

在SpringMVC中，若使用forward、redirect等前缀，则不会经过视图解析器，因此要写视图的完整路径(带后缀名)。在response.sendRedirect()方法中重定向时，绝对路径要带上项目名称。而SpringMVC中重定向时会自动为我们拼接上项目名称，因此只需要写资源名称即可。



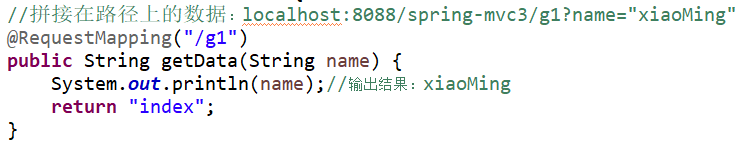




## 六、数据处理

### 1.数据自动映射

**①拼接在路径上的数据的名称和处理方法的参数名称一致**。



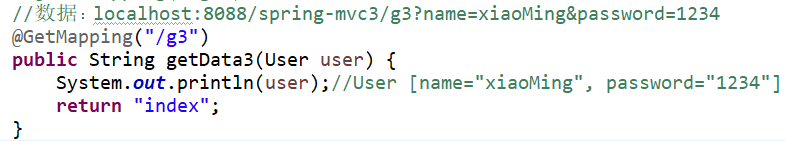
**②拼接在路径上的数据的名称和处理方法的参数名称不一致**

不一致时，在方法参数上使用**@RequestParam**注解修饰方法中的参数。

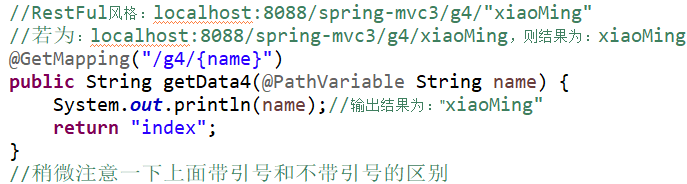


**③提交的数据是一个实体类**。

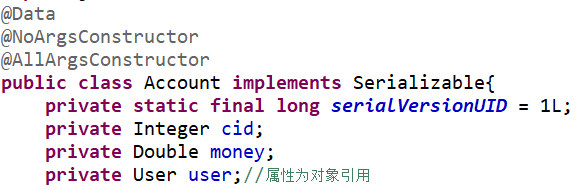
提交的数据的名称和对象的属性名一致，则数据可以和对象的属性自动映射，若不一致时，则对象的属性值为null。



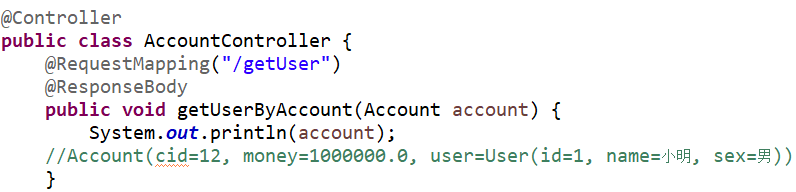
**④采用RestFul风格**。



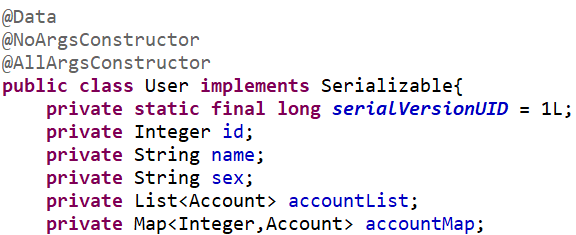
**⑤实体类中包含一个对象的引用**。



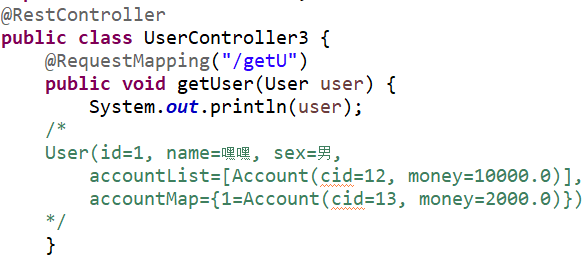




**⑥实体类中的属性是集合类型(如List、Map)**。



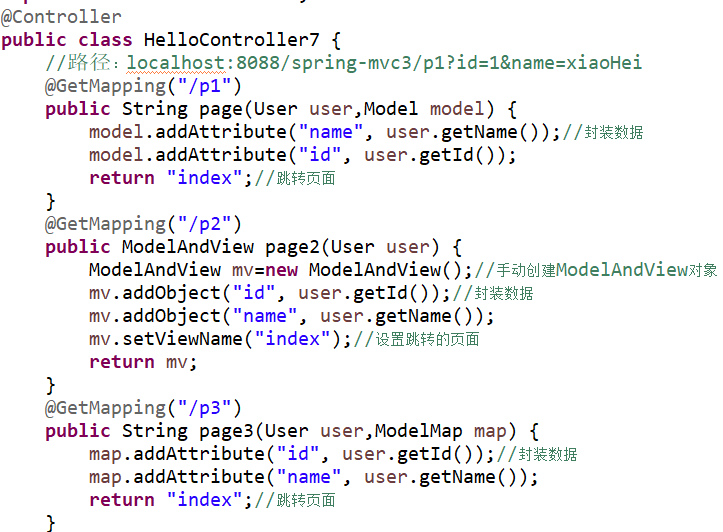




### 2.将数据显示到前端

将数据显示到前端有三种方式：采用ModelAndView对象、Model对象、ModelMap对象将数据封装然后在前端接收。

**例子**：



**ModelAndView、Model、ModelMap的区别**：

**ModelAndView**是一个类，它包含模型和视图，在开发中需要自己手动创建该对象。该对象主要有**两个作用**：①指定返回页面：比如采用该对象的setViewName()方法设置要返回的页面②封装数据并传递到页面：调用该对象的**addObject()**方法进行数据封装。

**Model**是一个接口，可用来封装要传递到页面的数据，常用方法为**addAttribute()**。Model本身不能用来指定跳转的页面，但可通过控制器的方法的String类型的返回值来设置要跳转的页面。不需要手动创建该Model的对象，可以直接作为方法的参数，然后使用该参数。

**ModelMap**是一个类，它继承了LinkedHashMap，它的功能和Model类似。它也可用来封装要传递到页面的数据，常用方法也为**addAttribute()**。ModelMap的实例是spirng mvc框架自动创建并作为控制器方法参数传入，无需自己创建。要跳转的页面也是通过控制器的方法的String类型的返回值指定。

Model最常用。

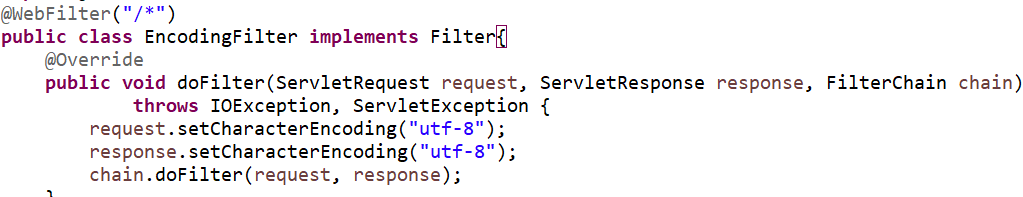
ModelAndView的addObject()方法和Model的addAttribute()方法和ModelMap的addAttribute()方法同request.setAttribute()的作用是一样的。

## 七、解决乱码问题

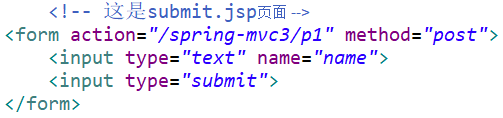
### 1.设置编码一

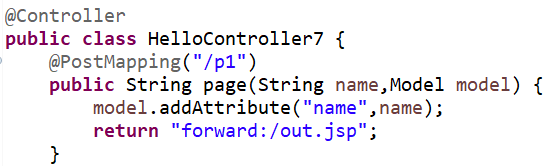
对于get请求需要在tomcat的server.xml文件中设置编码格式为utf-8，具体看以前Servlet中关于编码的笔记。

**法一**：针对post请求和响应到页面内容乱码解决办法是编写一个Filter进行过滤编码。



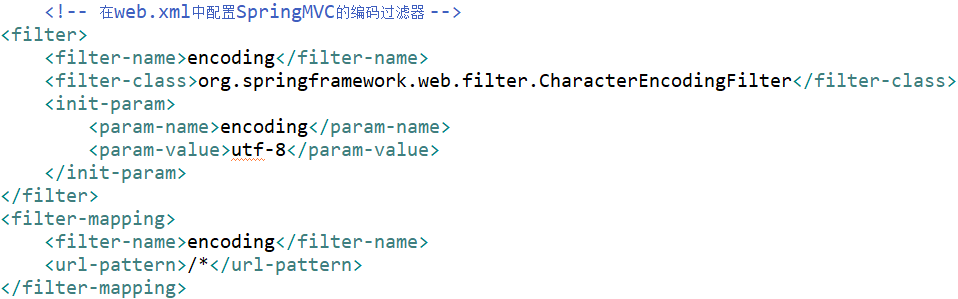
**例子**：







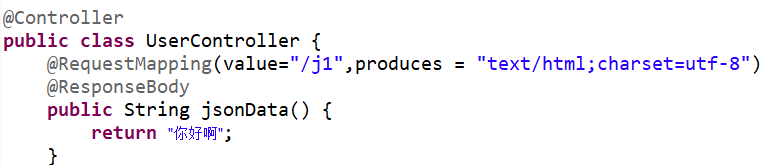
**法二**：使用SpringMVC提供的Filter过滤器。在web.xml中进行配置。



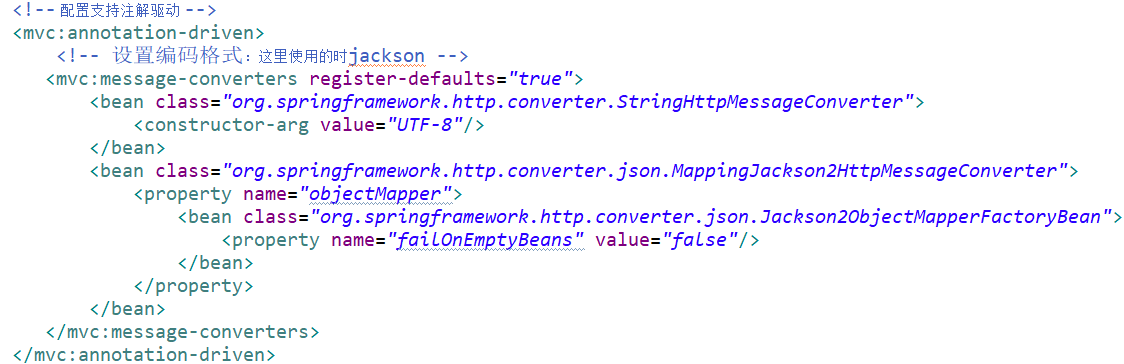
### 2.设置编码二

在控制器的方法上使用了@ResponseBody注解，或者在类上使用的@RestController注解，若方法的返回的是String类型的字符串，则默认是按iso8859-1编码，因此页面可能乱码。解决办法有如下两种：

**法一**：在配置请求路径的映射注解中添加produces属性，并指定编码格式。如@RequestMapping、@GetMapping、@PostMapping注解中。每一个返回字符串的方法都要配置编码格式，有点繁琐。



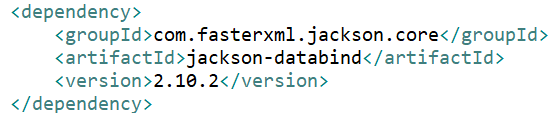
**法二**：在springMVC的配置文件中进行配置。当使用了jackson时，若页面出现乱码，也可以在注解驱动中添加如下内容。这样就可以不用在注解中配置编码格式。若使用的是fastjson，则替换成相应的fastjson的对象即可。



## 八、解析JSON数据

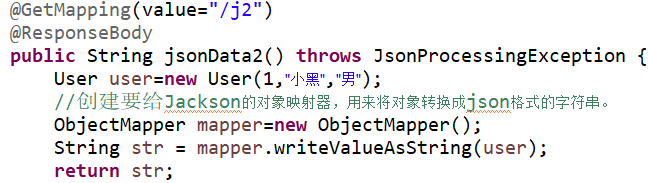
### 1.使用Jackson工具

**导入Jackson的依赖**：

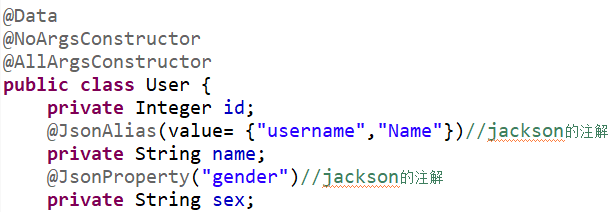


之后就可以使用Jackson了。

**ObjectMapper**对象：对象映射器，这个是jackson包中的，可用来将对象转换成json格式的字符串，方法是**writeValueAsString()**。



**两个常用注解**：

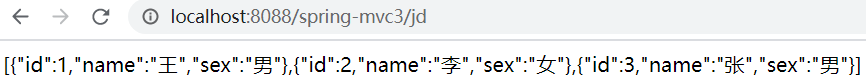


**@JsonAlias**注解：该注解的作用是，当json字符串转对象时，使用json中特定的key能转化成实体类的属性。如上面当json字符串的key为username(或者Name、 name)时，也能对应上User类的name属性。但是，当对象转化为json字符串时，实体类的属性转换成json字符串的key，使用的仍是实体类的属性名。如User对象转换成json字符串，则json中的key仍是name，而不是username或者Name。使用@JsonAlias注解需要依赖于getter、setter方法。

**@JsonProperty**注解：该注解的作用是，当json字符串转对象时，使用json中特定的key能转化成实体类的属性。若上面当json字符串的key为gender时，能对应上实体类User的sex属性，但key为sex时，不能对应上User的sex。当对象转化为json字符串时，实体类的属性转换json字符串的key时，使用的是json中指定的key。如User转换成json字符串时，json中的key仍是gender,而不是sex。@JsonProperty注解不需要依赖getter、setter方法。

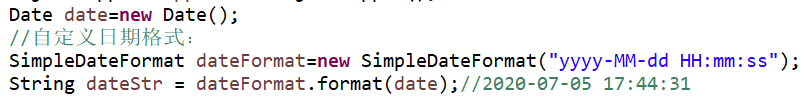
**将集合转化成json格式的字符串**：





**将时间转换成json字符串**：

**①纯java形式**：(这种方式更简单)

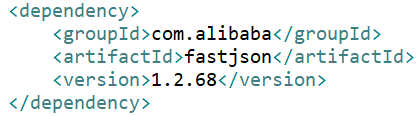


**②使用ObjectMapper对象**：



### 2.使用fastjson工具

导入fastjson的依赖。具体fastjson使用见笔记：[11.AJAX和JSON.docx](../4.javaEE基础/11.AJAX和JSON.docx)

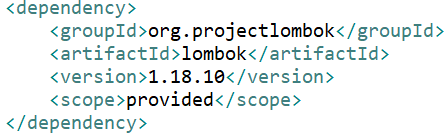


## 九、lombok

### 1.配置 lombok

对于一个JavaBean类或者POJO类，对于其getter、setter、构造器、toString方法，可以使用注解的方式实现。需要在Eclipse中添加lombok的插件，第一次使用时，操作如下：

步骤一：在maven项目的pom.xml添加lombok的依赖，更新项目，然后找到本地仓库中下载的该依赖的jar包。



步骤二：进入本地仓库中，找到该lombok的jar包，然后在该文件夹路径下启动cmd，使用java -jar lombok-1.16.20.jar命令对jar包进行解压。lombok的jar包解压事会自动扫描eclipse安装位置，若扫描失败可以点specify location自己进行路径选择 。路径选择好之后install/update就行了，安装完之后quit installer退出。

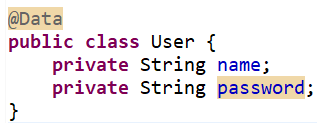
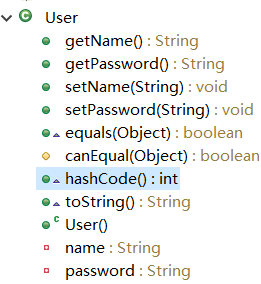
步骤三：验证lombok是否安装成功。打开eclipse软件安装路径下的eclipse.ini，若有-javaagent:E:\eclipse\eclipse\lombok.jar这样的配置句则表示插件配置成功，接下来只需要重启eclipse即可在项目中使用lombok插件。

步骤四：之后对于javaBean类或者POJO类的setter、getter等方法只需要使用相应的注解即可。

### 2.lombok的注解

#### ⑴注解

**①@Data注解**：@Data注解在类上，会为类的所有属性自动生成setter、getter、equals、canEqual、hashCode、toString方法，以及默认的无参构造器。如为final属性，则不会为该属性生成setter方法。

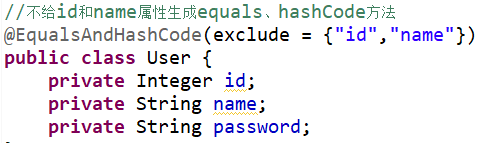
 

**②@Setter注解、@Getter注解**：若觉得使用@Data注解太暴力了，则可以在对应的属性上使用@Setter、@Getter注解自动生成相应的getter、setter方法。也可以注解在类上，则表示类中所有的属性都生成对应的setter、getter方法。

**③@NonNull注解**：该注解用在属性或构造器的参数上，Lombok会生成一个非空的声明，可用于校验参数，能帮助避免空指针。

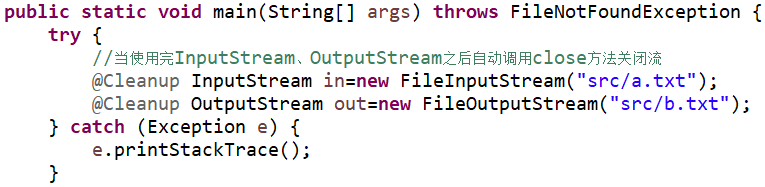
**④@ToString注解**：生成toString方法。默认情况下，会输出类名、所有属性（会按照属性定义顺序），用逗号来分割。可以通过exclude、includeFieldNames等指定要使用哪些属性生成toString方法。

**⑤@EqualsAndHashCode注解**：生成equals和hashCode方法。默认情况下，会使用所有非静态（non-static）和非瞬态（non-transient）属性来生成equals和hasCode，也能通过exclude注解来排除一些属性。



**⑥@AllArgsConstructor、@NoArgsConstructor、@RequiredArgsConstructor注解**分别生成全参、无参、部分参数的构造器。

**⑦@Cleanup注解**：该注解能帮助我们自动调用close()方法。



#### ⑵使用lombok的优缺点

**优点**：

能通过注解的形式自动生成构造器、getter/setter、equals、hashcode、toString等方法，提高了一定的开发效率。

让代码变得简洁，不用过多的去关注相应的方法。

属性做修改时，也简化了维护为这些属性所生成的getter/setter方法等。

**缺点**：

不支持多种参数构造器的重载。

虽然省去了手动创建getter/setter方法的麻烦，但大大降低了源代码的可读性和完整性，降低了阅读源代码的舒适度。

## 十、SpringMVC中的拦截器

请求转发，可以不用将数据放到request作用域中，然后在页面上采用EL的<param... >或<paramValues....>方式可以获得数据。但是若采用重定项的方式则获取不到。

**注意**：尽量把一些客户端不能直接访问的视图放在WEB-INF目录下，可以保证视图的安全。登录页面一般不能直接访问到，一般是通过点击登录才跳转到登录界面，因此可将登录页面也放到WEB-INF目录下，而首页则直接放在webapp目录下。WEB-INF目录下的页面只有服务器才能访问到，即Controller、Servlet可以访问。

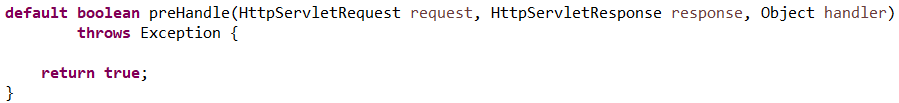
### 1.过滤器和拦截器的区别：

SpringMVC中的拦截器Interceptor是一种用于对控制器Controller进行预处理和后处理的技术，主要利用的是AOP思想，自定义拦截器需要实现HandlerInterceptor接口或者继承实现了该接口的类。拦截器是SpringMVC框架自己的，且拦截器只会拦截访问控制器方法的请求， 若访问的是jsp、html、image、js、css等是不会进行拦截的。

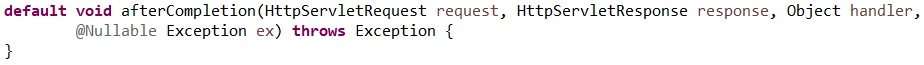
过滤器是Servlet中的，任何JavaWeb工程都能使用，过滤器设置/\*可以拦截任何资源，包括jsp、html等资源，自定义过滤器一般实现Filter接口。

### 2. HandlerInterceptor

HandlerInterceptor接口，有三个方法，如下所示：





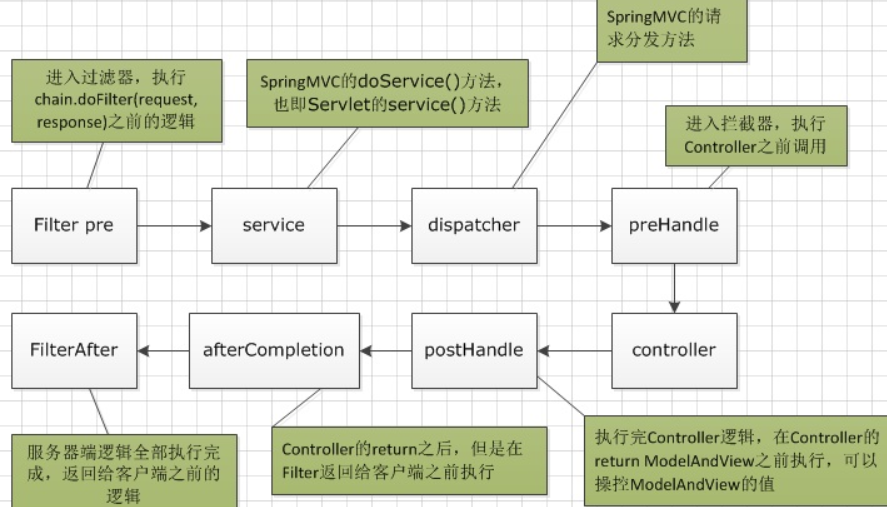


**preHandle()方法**：这个方法是在调用Controller的方法之前执行，即在请求处理之前调用。一个请求中可以同时存在多个拦截器，每个拦截器的调用会根据它声明的顺序依次执行，且最先执行的是拦截器中的preHandle()方法。因此，可在preHandle()方法中对当前请求执行一个预处理操作，或者在该方法中进行一些判断用来决定请求是否继续执行下去。**该方法返回值为boolean类型，若返回false，表示请求结束，不会继续往下执行。若返回true，则表示继续往下执行，若配置了多个拦截器，则继续执行下一个拦截器的preHandle()方法，若后面没有拦截器了，则调用当前请求的Controller中的方法**。在这个方法中可以通过request、response指定跳转页面。

**postHandle()方法**：这个方法只有在preHandle()方法返回值为true时才能被执行。这个方法是在当前请求被处理之后，即调用完Controller的方法之后才能被执行。但是这个方法会在前端控制器(DispatcherServlet)执行视图的渲染之前执行，也就是说可在这个方法中对ModelAndView对象进行操作。**注意，先声明的拦截器的这个方法会被后调用**。在这个方法中也可以指定跳转页面。

**afterCompletion()方法**：该方法也是只有在preHandle()方法返回值为true时才能被执行。这个方法会在整个请求结束之后，即前端控制器执行完视图的渲染之后执行。因此，**主要用于进行资源的清理工作**。**注意，先声明的拦截器的这个方法会被后调用**。在这个方法中不能用request、response指定跳转界面。

**总结**：preHandle()方法会在执行Controller的方法之前执行，且只有preHandle()方法的返回值为true，才会接着往下执行。postHandle()方法会在Controller的方法执行之后执行，且在前端控制器进行视图渲染之前执行。afterCompletion()方法会在当前请求执行完成之后执行，即前端控制器渲染完视图之后执行。





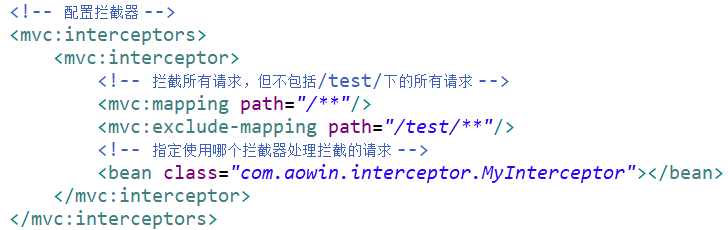
### 3.拦截器的使用步骤

**步骤一**：自定义拦截器要实现HandlerInterceptor接口，然后重写相应的方法。



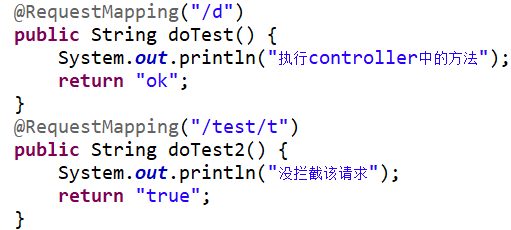
**步骤二**：在springmvc的配置文件中配置拦截器。

其中**<mvc:mapping>**标签用于配置要拦截的请求，**<mvc:exclude-mapping>**标签用于指定不需要被拦截的特定的请求。

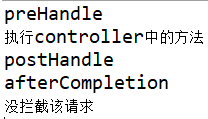


在配置拦截请求的路径时， /\*\*表示拦截该请求下的所有请求，如/admin/\*\*，表示/admin下的所有请求，/admin/user、/admin/user/test下的请求都会被拦截。

步骤三：编写Controller类进行测试。

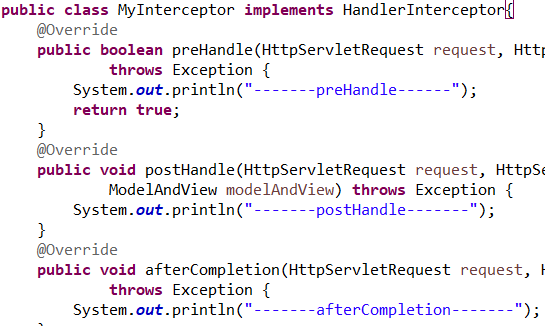


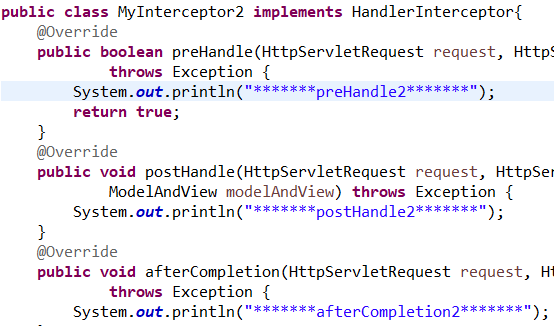
当路径为[localhost:8088/spring-mvc-ajax/d](http://localhost:8088/spring-mvc-ajax/d) 时，输出内容如下：

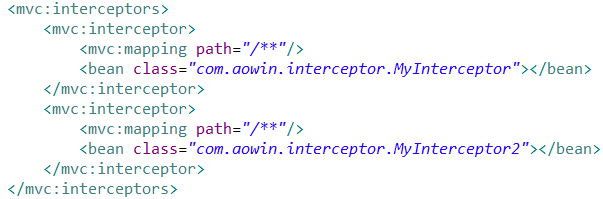


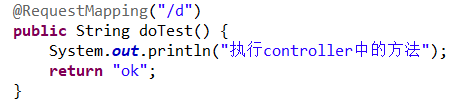
当路径为[localhost:8088/spring-mvc-ajax/test/t时，只输出了：没拦截该请求。](http://localhost:8088/spring-mvc-ajax/test/t时，输出如下：)

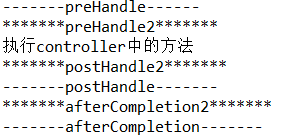
**配置多个拦截器的例子**：







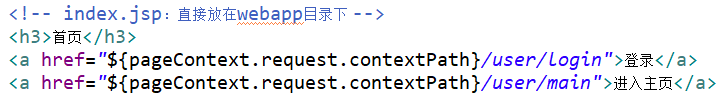




通过输出结果可以发现多个拦截器中的preHandle()、postHandle2()、afterCompletion()方法的执行顺序同以前配置多个Filter中的执行过程类似。

### 4.使用拦截器实现登录过滤

**index.jsp页面**：



**login.jsp页面**：



**main.jsp页面**：



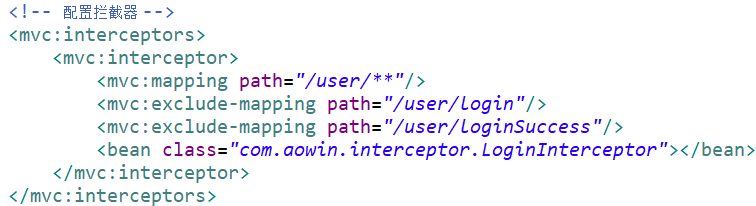
**LoginController**：



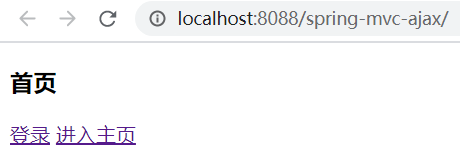
**LoginInterceptor**:

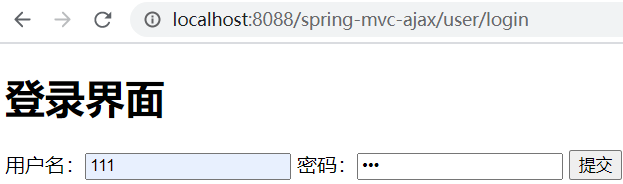


**springmvc的配置文件中配置拦截器**：



**测试**：







## 十一、文件上传和下载

为了能在前端上传文件，<form>表单有如下要求：表单的method值必须为post，并将enctype设置为multipart/form-data。只有在这样的情况下，浏览器才会把用户选择的文件以二进制数据发送给服务器。

### 1.<form>表单中的enctype属性

enctype就是encodingType的简写。**enctype的值有如下三种**：

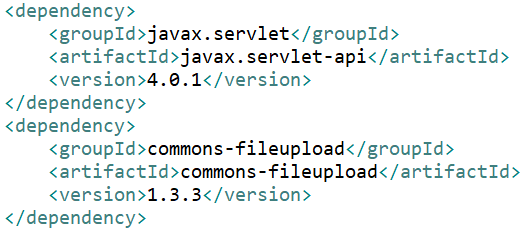
**application/x-www=form-urlencoded**：这是**默认方式**。这种编码类型会把表单中发送的数据编码为键值对的方式，且一般采用request.getParameter()的方式接收数据。若要上传文件只能上传文本格式的文件，若文件较大则上传效率低。**在发送前编码所有字符(默认)**。

**multipart/form-data**：这种编码方式会以二进制流的方式来处理表单数据。**不对字符编码，但在使用包含文件上传控件的表单，必须使用此值**。一般用这个编码格式传输文本、图片、音频等。

**text/plain**：除了把空格转换为 "+" 号外，其他字符都不做编码处理。这种方式**适用直接通过表单发送邮件**。

### 2.文件上传

**步骤一**：SpringMVC中实现文件上传需要导入commons-fileupload和Servlet的依赖。



**步骤二**：在springmvc的配置文件中配置multipartResolver对象。



采用CommonsMultipartFile来获取上传的文件，该类有如下几个常用的方法：

String getOriginalFilename()：获取上传文件的原名。

InputStream getInputStream()：获取输入流。

void tranferTo(File destination)：将上传的文件保存到一个目录中。

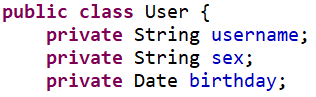
还没弄明白。

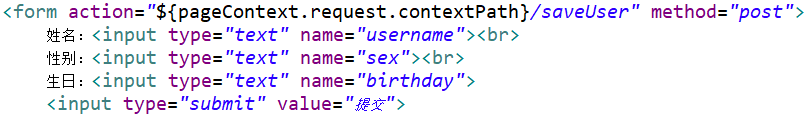
3.文件下载

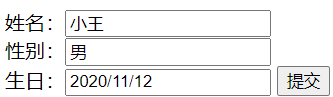
有空再补充。

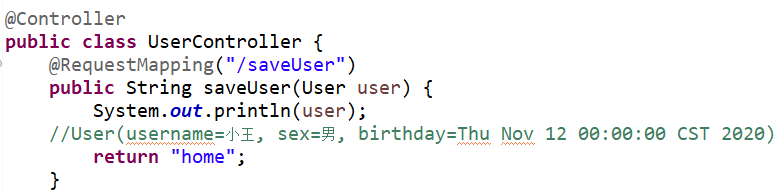
## 十二、自定义类型转换器

**例子**：







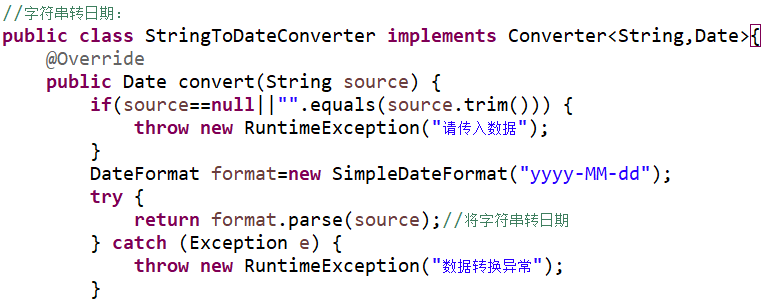


当前端传入的日期格式为2020/11/23的形式，可以自动封装到Date对象中，而若传入的格式为2020-11-23，则不能自动封装到Date对象中，会报400错误。

**使用自定义类型转换器可以解决上面的问题，步骤如下**：

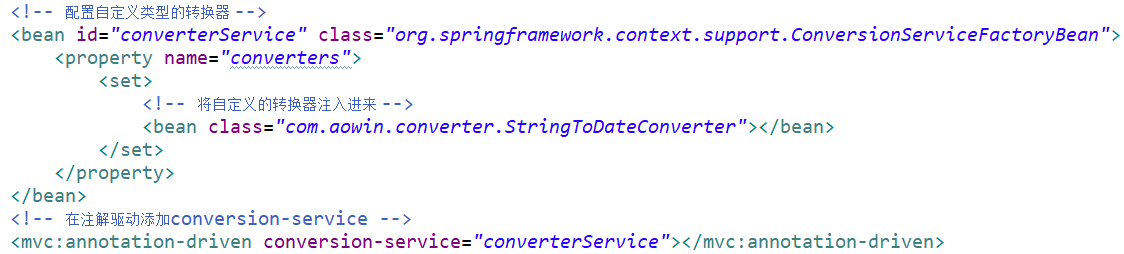
**步骤一**：定义一个类，实现Converter接口，该接口有两个泛型，S表示接收的类型，T表示目标类型。这个Converter是org.springframework.core下的包中的。





**步骤二**：在SpringMVC的配置中配置自定义类型的转换器，然后在注解驱动中添加conversion-service属性让类型转换器生效。

<mvc:annotaion-driven>会默认使映射处理器和处理适配器生效，但不会使类型转换器生效，因此要使类型转换器生效需要在这配置。



操作完上面的步骤之后，则可以让2020-11-23这种形式的字符串也能封装到Date中。因为在步骤一中指定了字符串的格式为yyyy-MM-dd形式，所以其他形式的字符串就就不能被封装到Date对象中了，如2020/11/23转换就会报400错误。

## 十三、异常处理器

需要学一下。

@ControllerAdvice注解：

@ExceptionHandler注解：