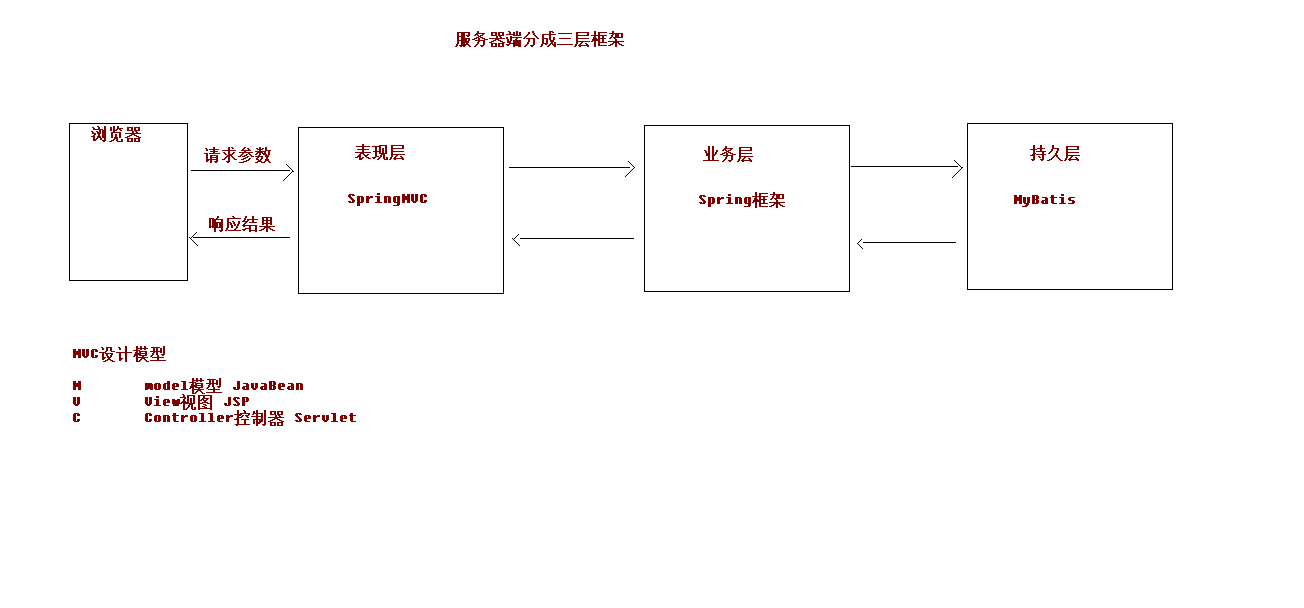
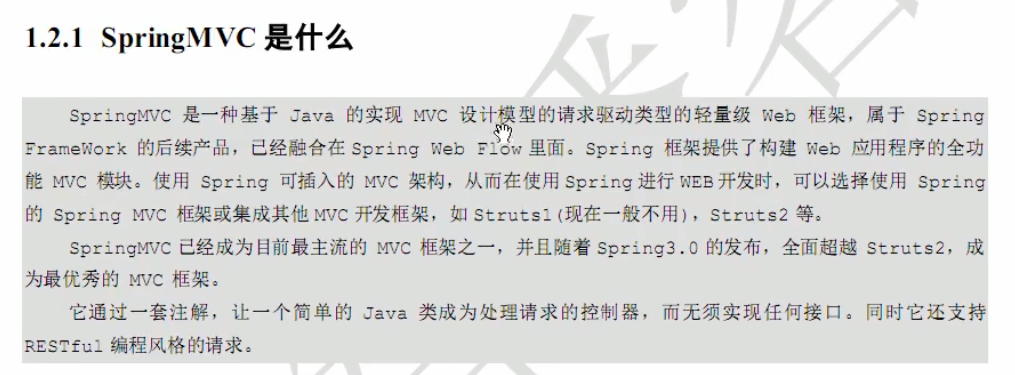
老样子，先来看看大体架构：

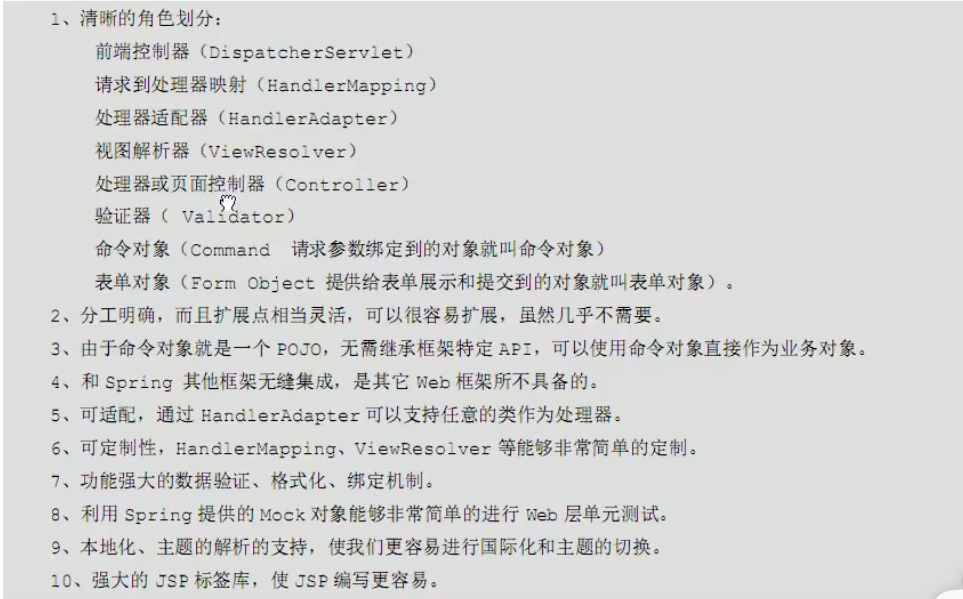


SpringMVC框架本身：



也就是说前端不用mvc框架也可以，spring都会帮你整合（比如vue）。

SpringMVC的优势在于：



当然了，springMVC也支持rest风格！

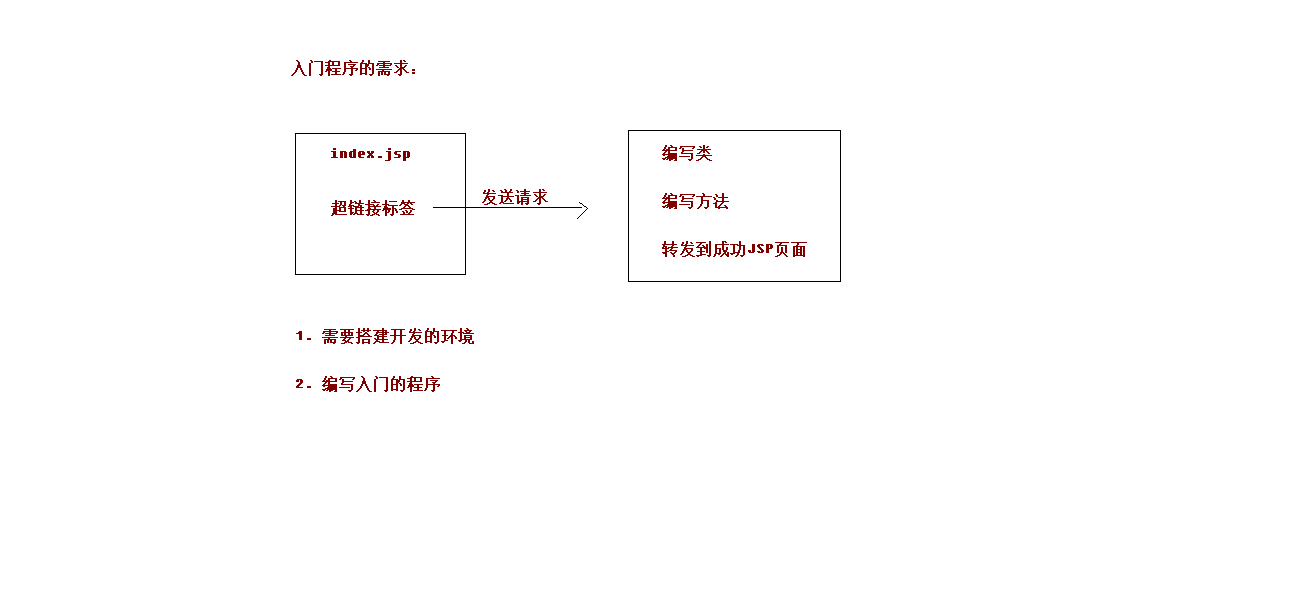
#第一个示例程序

接下来我们编写第一个springMVC程序。

该程序的功能是：

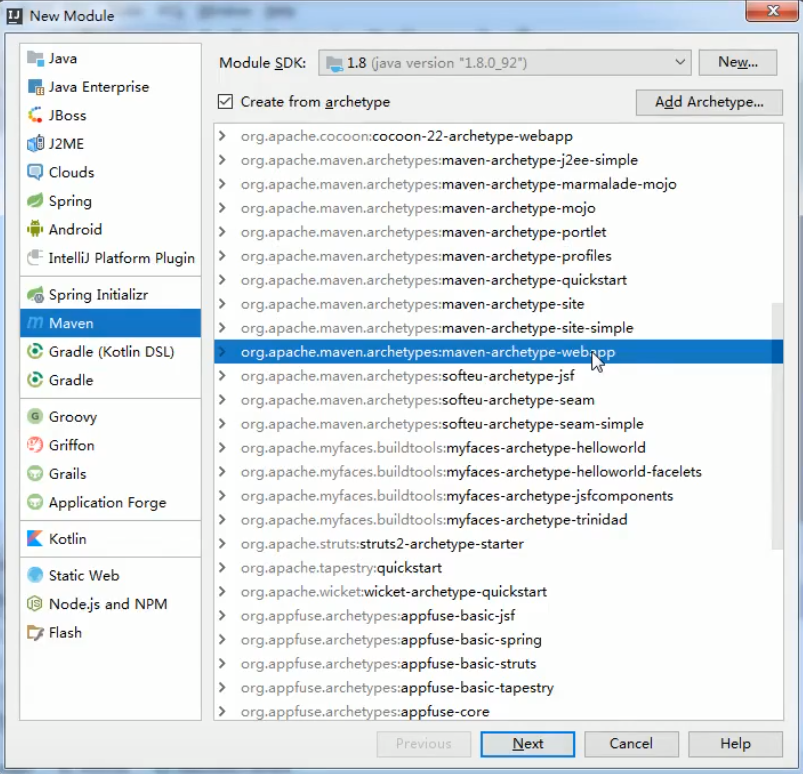
有一个页面，上面有一个超链接可以点击；

前端点击了这个链接，就会调用后端一个特定类的特定函数；该函数的功能是：跳转到一个特定jsp页面上！

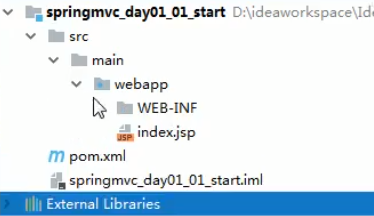


搭建环境：

新建项目选maven项目，然后选webapp开发！

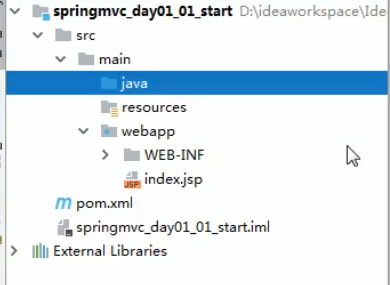


项目初始文件结构：



多了webapp包；多了.iml配置文件。

目录结构补全：



坑：没法新建配置文件or类or包：

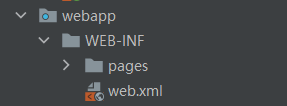
右键java，设置为root文件夹；

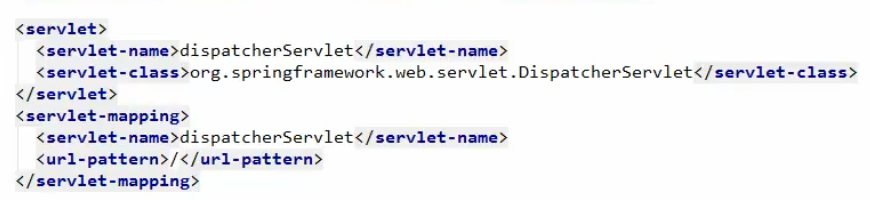
右键resources，设置为sources文件夹。

然后是常规的pom中配置工作，此处省略；

接下来，想要使用springMVC第一步：加入【前端控制器】！（其实就是一个servlet）

在webapp.WEB-INF.web.xml中配置





都是固定的！

第二步：在resources中新建.xml项目配置文件。

第三步：配置tomcat服务器！

如何安装：

<https://www.runoob.com/servlet/servlet-environment-setup.html>

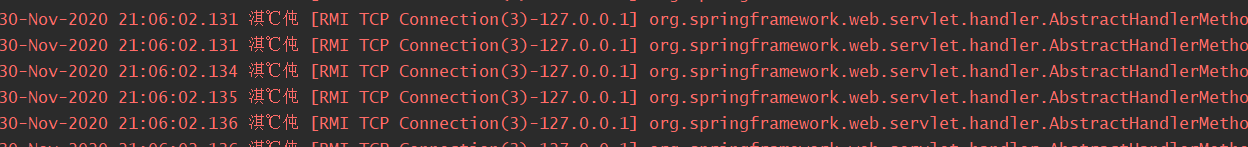
如何在idea中配置：

<https://www.jianshu.com/p/91a2d0a1e45a>

坑：运行报错Address localhost:8080 is already in use.更改tomcat的默认端口就好了。

<https://blog.csdn.net/Jay_1989/article/details/52870760>

坑：运行乱码：



解决：<https://blog.csdn.net/B2345012/article/details/88365722>

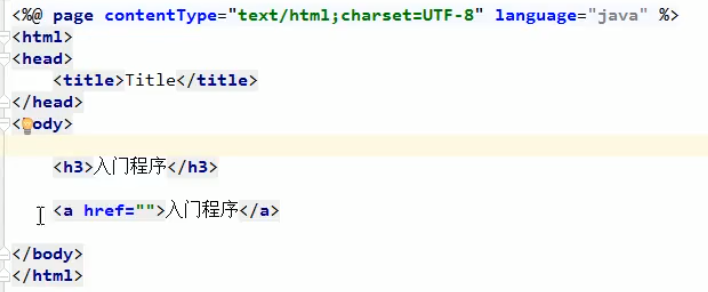
环境搭建搞定！

接下来我们开写入门程序！

第一步：进入index.jsp。默认的index.jsp没写头部编码2，咱给他加上一行：

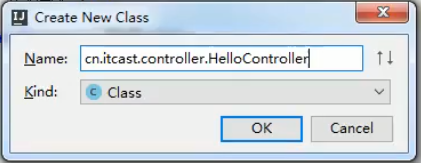


然后写一个简单的超链接：



Href里写啥先留个悬念。

第二步：到java里编写对应的类来处理点击超链接的请求



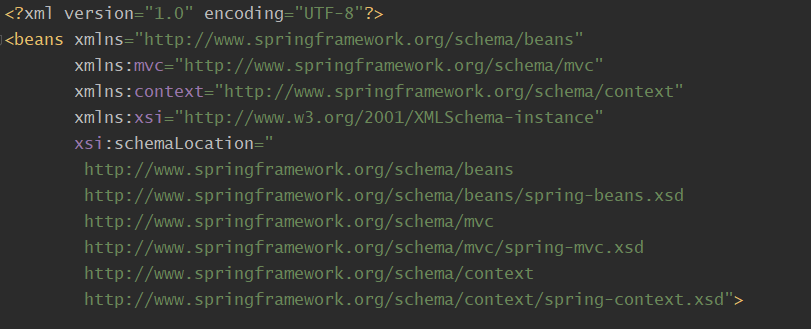
文件结构如上。写一个简单的hello：



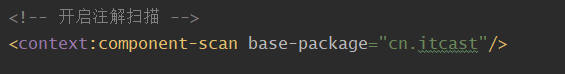
接下来，如何才能让点击那个超链接和这个函数关联起来呢？

用spring中讲到过的ioc！（使用注解方法）

在项目配置文件springmvc.xml中，咱们先导入必要的坐标：



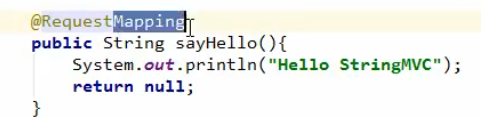
然后开始注解扫描：



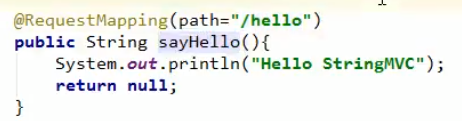
这样我们就可以给hellocontroller类加注解了：



然后再给想要调用的方法加注解：



Request为请求，mapping为映射，正如其名！该注解还需要一个参数，确定请求该方法的路径：



不过这个路径并不是指url中的路径，而是指前端发出请求的路径。

所以前端页面的href我们应该填上：

第三步：让前端控制器servlet导入我们的项目配置文件springmvc.xml

在web.xml中：

Servlet有固定的接受配置文件参数，名为contextConfigLocation。如下：



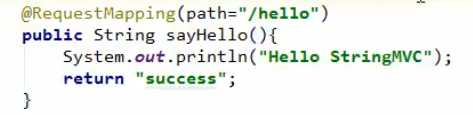
到此为止，我们就实现了：点击超链接，对应的方法sayhello（）会执行！

接下来，我们要实现页面跳转！

这里有一个规则：

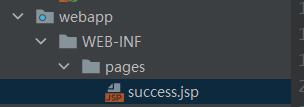
如果我们让sayhello（）返回一个字符串，这个字符串就【默认】是新页面的名字！（需要配置一个视图解析器）

例如：



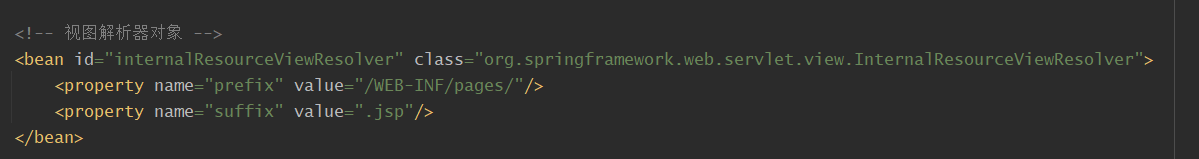
新的跳转页面就是success.jsp！

于是我们写一个名为success的页面：

（常用文件结构）

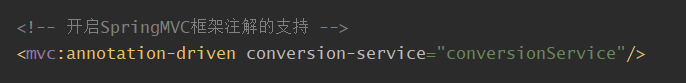
接下来我们配置一下【视图解析器】，有了它，程序就会根据返回的页面名字“success”寻找名为success的页面。

在项目配置文件springmvc.xml中：



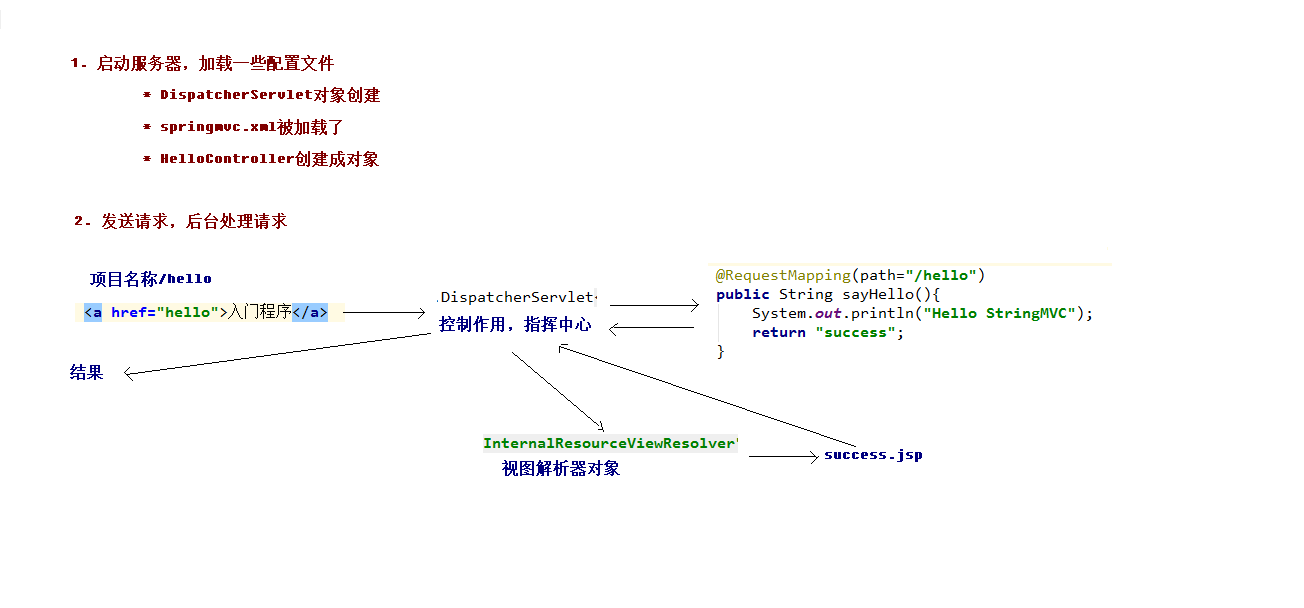
Prefix是跳转页面所在的目录；suffix是跳转页面的后缀名。

最后一步：打开springmvc框架对注解的支持！

就一句：

该看p164了。

解析来我们对上述例子做一个简单的过程回顾：



Servlet到底是干嘛的：

前端发出请求后，由后端的哪个函数来接呢？这个过程就是servlet处理的！

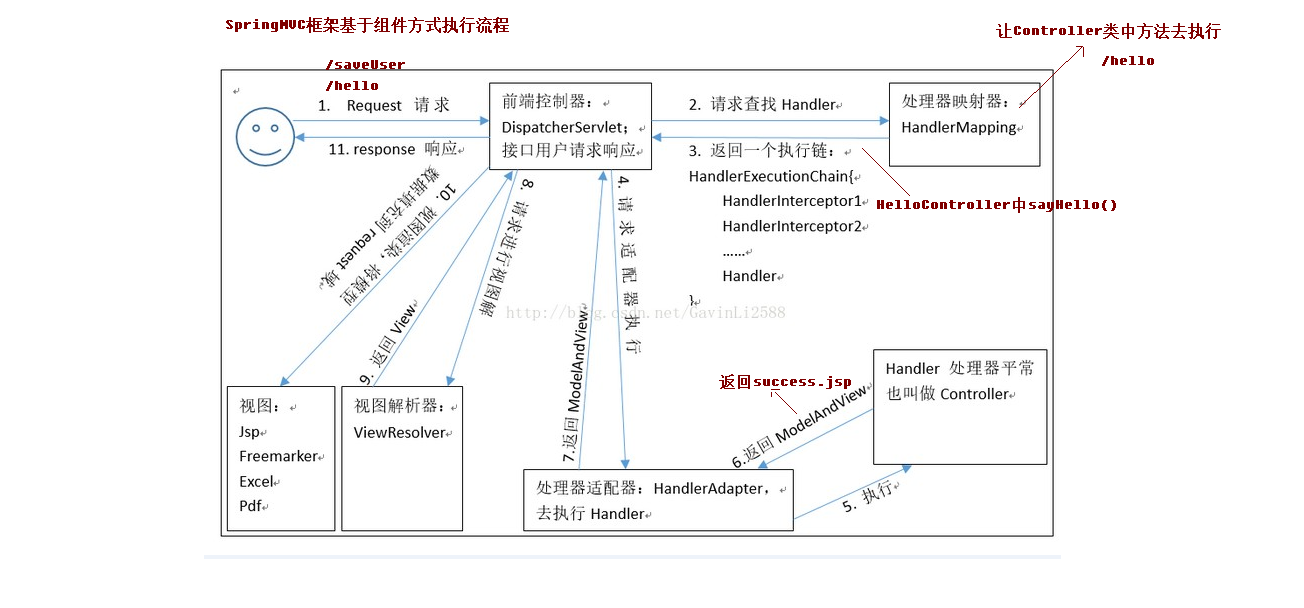
它是一个【指挥中心】：

首先它收到前端的请求，然后根据注解配置、找到对应的后端函数，将其调用；

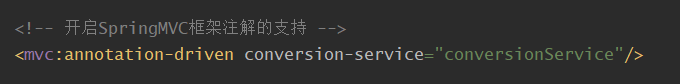
后端函数返回一个值给它，它在根据视图解析器的配置（也是注解）、找到要返回的页面hello.jsp；

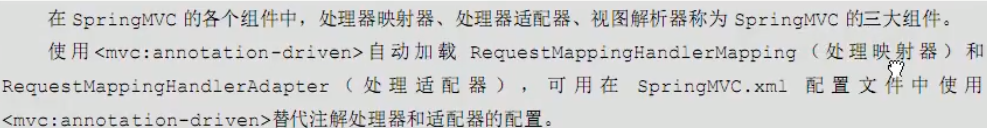
然后把要返回的页面返回给前端进行跳转！

具体而言：



这时候你可能会问：这么多“器”我都没配置过啊，怎么能使用呢？

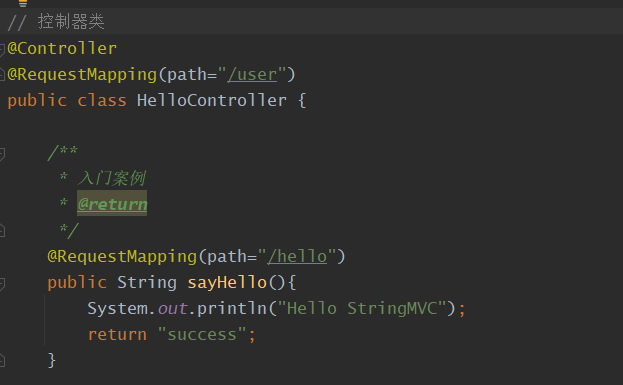
事实上我们配置了：

就相当于配置了处理器映射器和处理器适配器！

#RequestMapping注解

基础的用法就是加在方法前面（如上例）；

但其实也可以加在类前面，like：

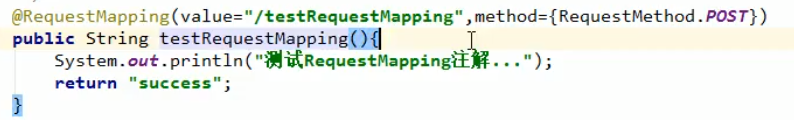


这样的话前端请求时href应该是：user/hello！

RequestMapping中的【属性】：

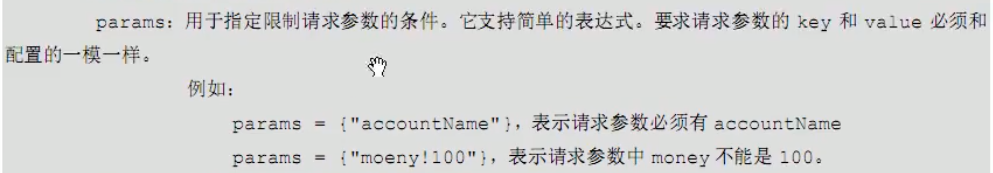
首先是【path】（也可以写成【value】 完全相同！）：如果你的requestMapping只有这一个属性，可以直接省略写为

【method】属性：用于规定调用该方法的请求类型！

例如：

加上这个属性，以后只有POST请求可以调用该函数！（上例的超链接就不能调用该函数了，因为超链接的请求固定是GET类型！）

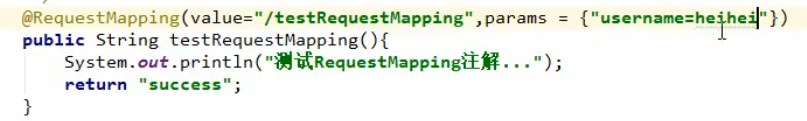
【params】属性：



比如上例的超链接请求肯定是不行了，因为那个请求没有传任何参数！

请求如何传参数：就可以了！

当然如果你把params属性改成：



上述超链接也不行了，虽然传了username属性、但是值不对！

【headers】属性：

举例：

请求中必须包含Accept请求头才能访问！

请求头相关知识后面会讲。

如何看一个请求的请求头？

浏览器内按f12，选择“网络”选项，就可以看请求的各种信息！（其实postman也可以）

该看168了！

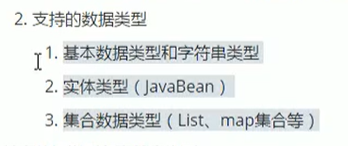
# 请求参数的绑定（对应：day01下webapp中params.jsp和controller中的PramaController）

概念：客户端传请求给后端，请求中一定有参数；后端拿到参数的过程，就是请求参数的绑定！

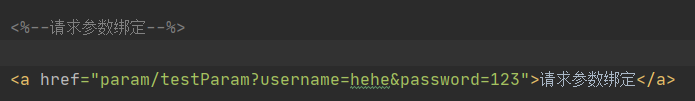
表达提交的数据都是键值对：

方法1：直接在controller的方法中使用同名参数即可：



该方法支持的参数类型：

1. 字符串类型（基本数据同理）

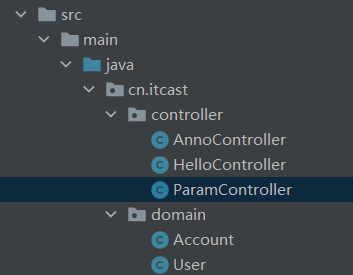
前端页面：

后端controller：

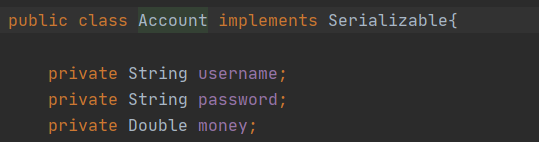


注意后端是两层RequestMapping注解，所以前端要两层路由！

1. Javabean类型参数

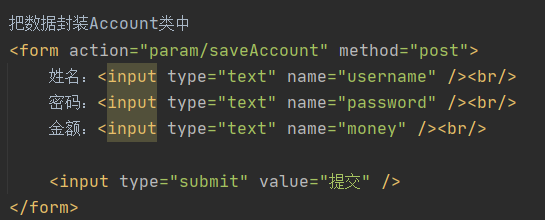
先来一个account类：

类代码：



（可序列化）

前端页面：



表单内的属性和实体类属性同名！便于封装

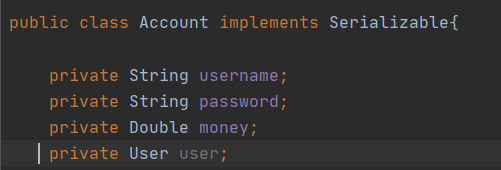
后端ParamController中：



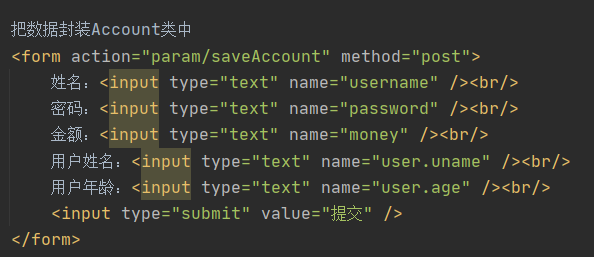
也就是说，前端并没有指定参数是实体类；但当后端方法的参数是实体类时，前端传来的数据就会自动封装到参数实体类中！（当然了，你传来的数据的命名必须和实体类属性名对应的上）

Advanced：如果参数类中又有其他的实体类作为成员该怎么传递值呢？

Like Account类中：（User类本身有啥不重要就不写出了）



如何把从前端传的数据封装进User中呢？：



就在属性名前加上类名限定就可以了！（uname和age是user类中的属性名）

Advanced：中文乱码

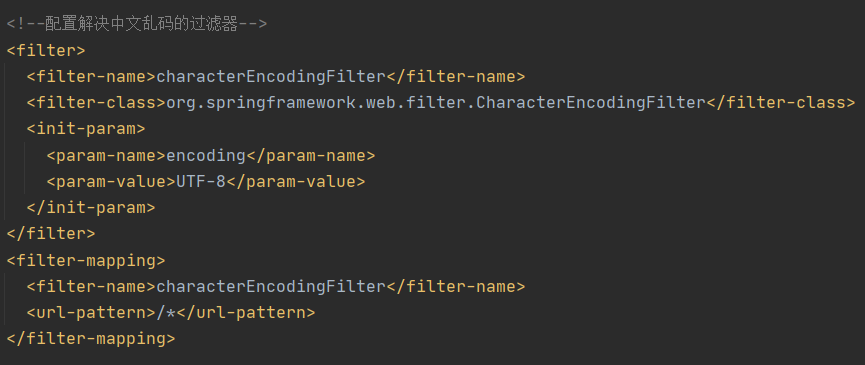
如果你在前端页面的表单中写入中文，运行，控制台：



不会输出中文。

解决方法：配置过滤器！

第一步：在web.xml中配置：



Advanced：参数对象中有集合：

Like在Account类中：



前端页面：



执行，控制台：



实际开发中，list不可能写死为0；map也不会写死一个键one。这里是为了演示。

到这里，我们注意到一个问题：事实上在前端页面提交的数据，都是字符串类型（你啥都可能往里输嘛，都是用字符串类型来接收）；而后端实体类User中的age等是Int类型。我们没有转换，但程序也照样运行了没有出错；实际上是SpringMVC这个框架为我们完成了类型转换的工作！

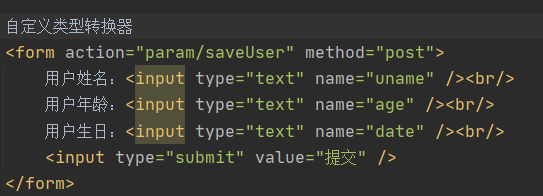
当然了，框架帮你转的代价就是有固定的格式。例如你想把一个字符串转换成Date日期类型：上面那个正常运行，下面则会报错！

不过我们也可以自己定义格式，那就是【自定义类型转换器】。

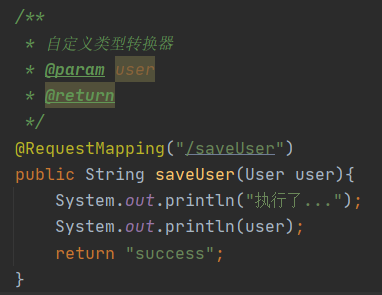
# 自定义类型转换器

在User实体类中，我们加入属性：

前端页面先来一个表单：



后端来一个对应的saveUser方法：



这个时候，如果表单输入正确的日期格式：



完全没问题运行。控制台：

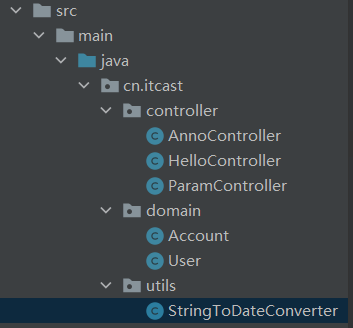


但如果在表单输入2000-11-11，就不正确了：

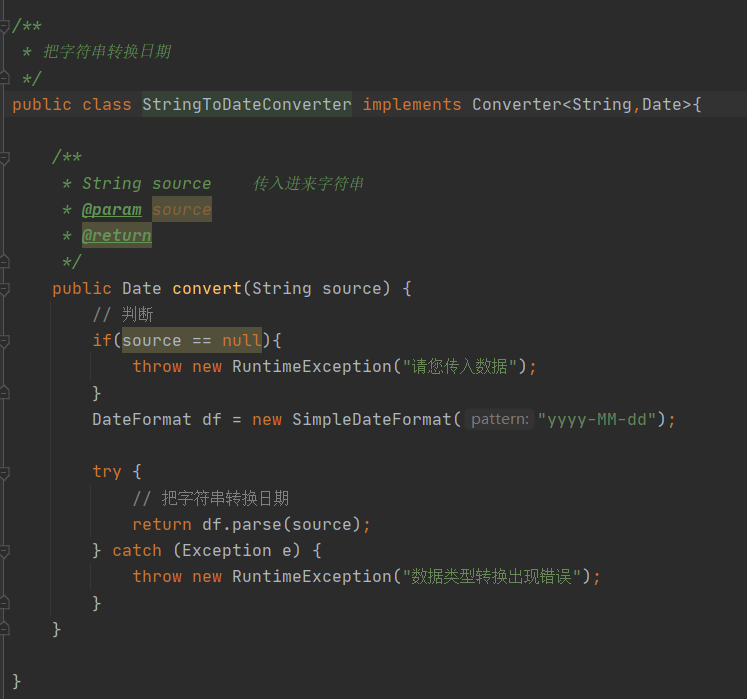


使用【类型转换器】：

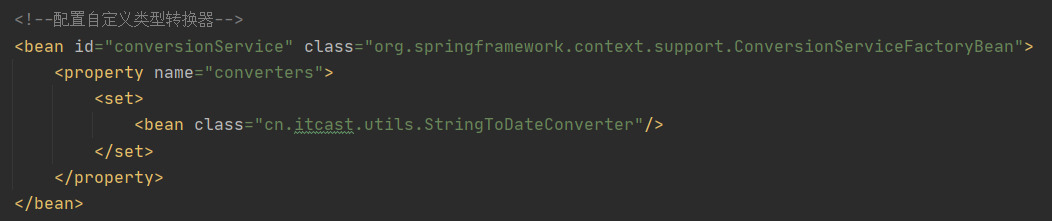
第一步：在后端的实体类，实现【Converter接口】。



直接来一个cn.itcast.utils.StringToDateConverter工具类（转换前后的类型，Converter泛型就已规定好了！）：

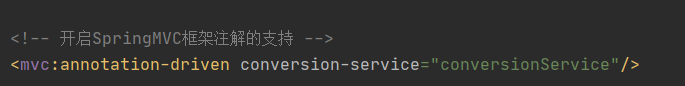


第二步：在项目主配置文件springmvc.xml中配置转换器：



converters是人家本身自己写的一些转换函数（Converter接口有很多实现类），比如有n个，都存在一个set集合里；现在把你自己写的方法也注册在set集合里，converters就有n+1个转换方法了。

然后在annotationdriver（默认会把基本的处理器适配器啥的注册，转换器还是需要手动注册一下的）里把类型转换器本身注册了：

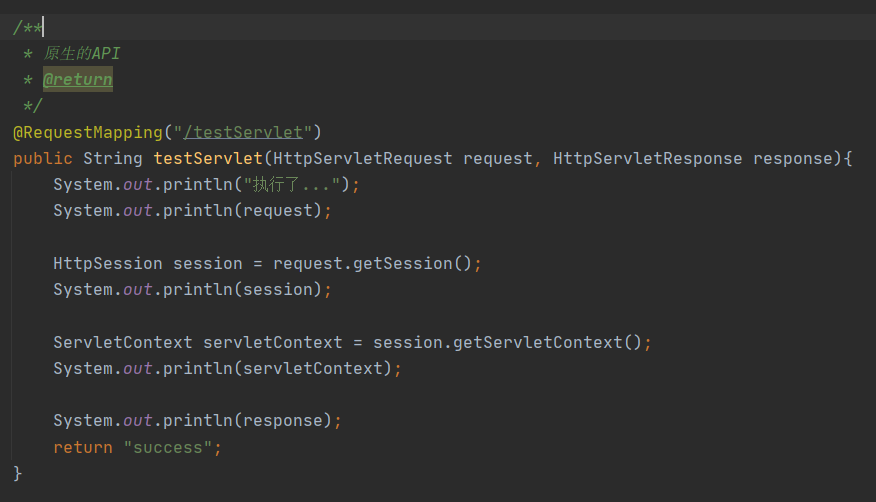


#servlet原生API

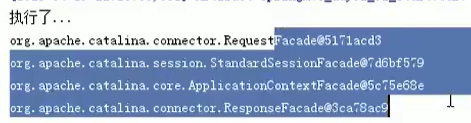
在页面中：



后端实现一下testServlet方法：



可以看出，servlet原生的对象，直接写在参数列表就可以获得！



该看175了

目前为止，我们学习了两个最最常用的注解RequestMapping和Controller。解析来我们学一些其它【常用注解】：（对应anno.jsp和AnnoController）

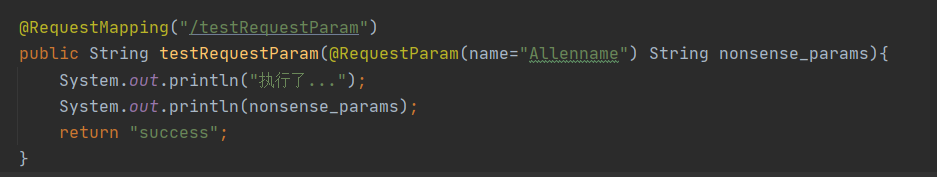
【@RequestParam】注解：

之前我们前端表单发请求的参数，名字必须和后端方法名参数/实体类参数相同才封装的进去；

在前端anno.jsp页面中，我们加一个超链接：

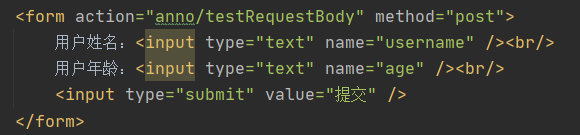


在后端对应的方法中，只要使用RequestParam注解，方法的形参名字就可以随便起啦：

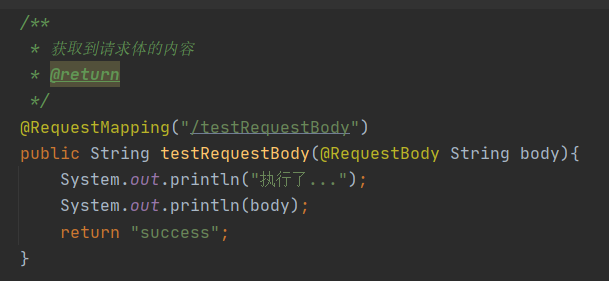


【@RequestBody】注解：

前端先加一个表单：



后端实现对应方法：

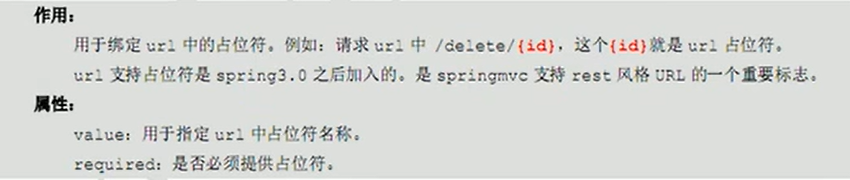


加上RequestBody注解后，前端传的多个参数会合成一个整体的body作为一个String被后端接收到！控制台输出：

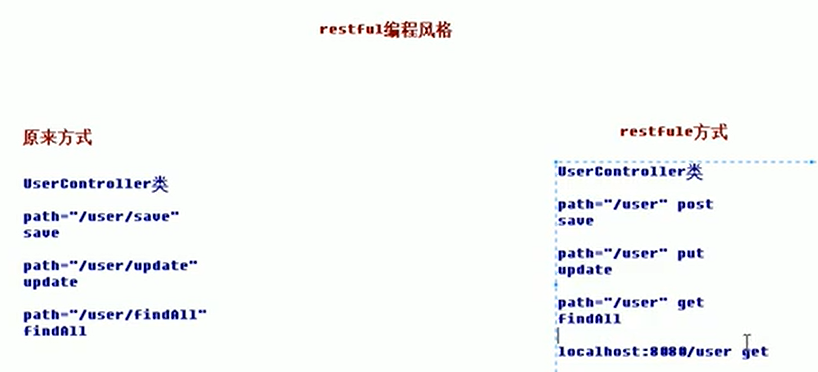


（后面前端会穿json文件，就用这个方法）

【@PathVariable】注解：

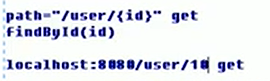


RESTful风格：



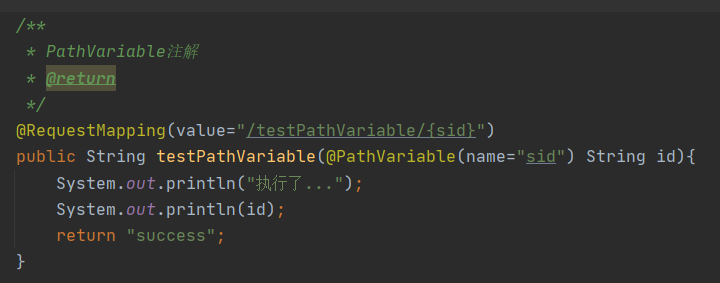
那三个方法可以用同一路径来访问！

若方法有参数：



findById和findAll可以根据url区分！

后端我们在url中加一个占位符：



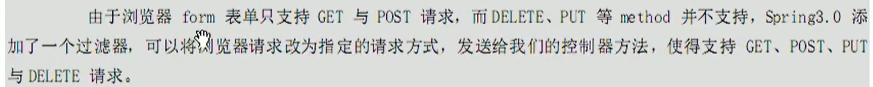
前端我们对应要写为：



（直接/参数值！）

# 使用【HiddenHttpMethondFilter】更改浏览器请求方法的类型（GET POST PUT DELETE）

背景：



但是用这个Filter很复杂，需要配置一堆东西。

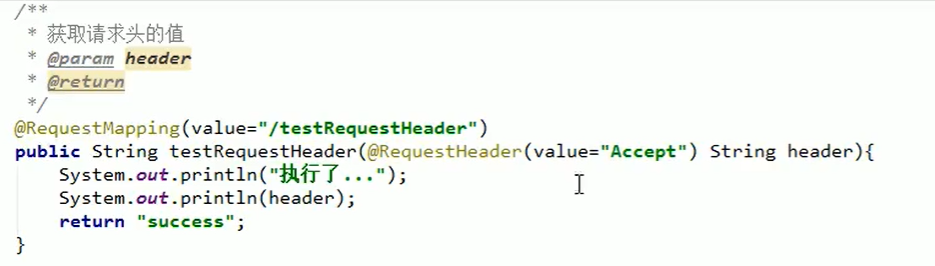
事实上，后期我们会学习WebClient类，里面有很多静态方法可以直接实现请求方法的类型控制

。啥都不用配置，方便的一批。后面我主要用那个，这个方法就不多介绍了。

【@RequestHeader】注解：

可以获取请求头信息！

后端中：



（头部信息很多，Accept是其中一个）

前端中对应加一个超链接就可

【@CookieValue】注解：

背景：Cookie是什么？

**Cookie 是什么**

1. Cookie 是浏览器访问服务器后，服务器传给浏览器的一段数据。

2. 浏览器需要保存这段数据，不得轻易删除。

3. 此后每次浏览器访问该服务器，都必须带上这段数据。

Cookie 就是这么简单，这就是 Web 开发里 Cookie 的含义。

**如何使用 Cookie**

Cookie 一般有两个作用。

第一个作用是识别用户身份。

比如用户 A 用浏览器访问了 [http://a.com](https://link.zhihu.com/?target=http%3A//a.com)，那么 [http://a.com](https://link.zhihu.com/?target=http%3A//a.com) 的服务器就会立刻给 A 返回一段数据「uid=1」（这就是 Cookie）。当 A 再次访问 [http://a.com](https://link.zhihu.com/?target=http%3A//a.com) 的其他页面时，就会附带上「uid=1」这段数据。

同理，用户 B 用浏览器访问 [http://a.com](https://link.zhihu.com/?target=http%3A//a.com) 时，[http://a.com](https://link.zhihu.com/?target=http%3A//a.com) 发现 B 没有附带 uid 数据，就给 B 分配了一个新的 uid，为2，然后返回给 B 一段数据「uid=2」。B 之后访问 [http://a.com](https://link.zhihu.com/?target=http%3A//a.com) 的时候，就会一直带上「uid=2」这段数据。

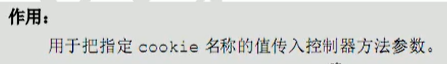
借此，[http://a.com](https://link.zhihu.com/?target=http%3A//a.com) 的服务器就能区分 A 和 B 两个用户了。

第二个作用是记录历史。

假设 [http://a.com](https://link.zhihu.com/?target=http%3A//a.com) 是一个购物网站，当 A 在上面将商品 A1 、A2 加入购物车时，JS 可以改写 Cookie，改为「uid=1; cart=A1,A2」，表示购物车里有 A1 和 A2 两样商品了。

这样一来，当用户关闭网页，过三天再打开网页的时候，依然可以看到 A1、A2 躺在购物车里，因为浏览器并不会无缘无故地删除这个 Cookie。

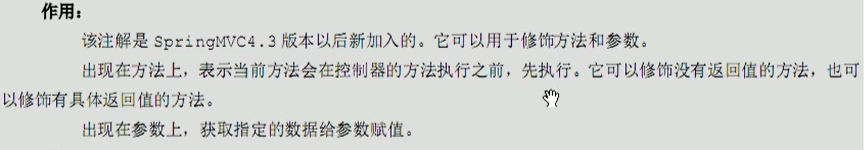
借此，就达到里记录用户操作历史的目的了。

回到CookieValue注解。它的作用是：

Cookie有固定的格式，里面有很多属性。这里我们以JESSIONID为例：

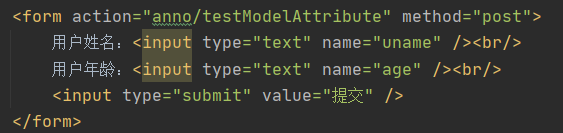


【@ModelAttribute】注解：

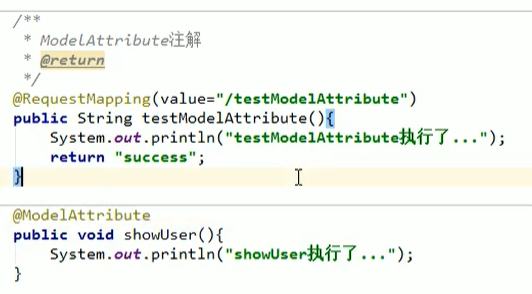


【@ModelAttribute加在方法上】

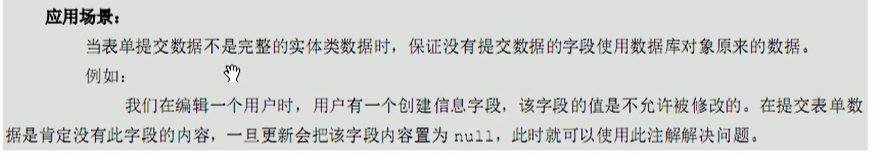
在前端我们加一个表单：



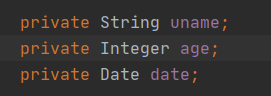
在后端实现对应的方法，并写一个用ModelAttribute注解的方法：



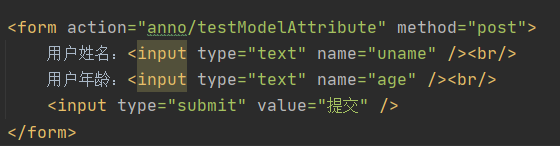
哪个先执行，8用我多说了吧。



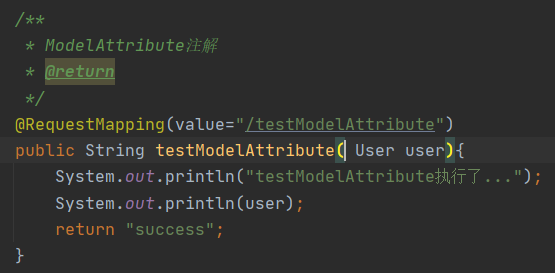
举例：

我们的User类中，有三个属性。

如果我们在前端表单中只提交两个数据：



并在后端中输出：



这样直接输出的话，由于缺一个属性，它是这个样子的：



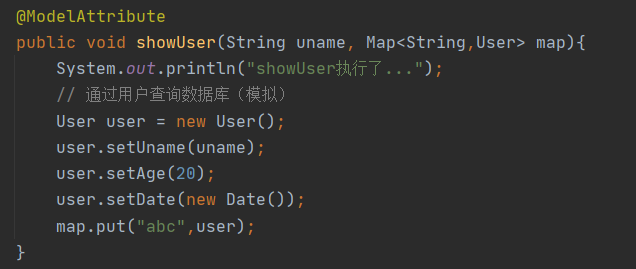
为了避免这个问题，我们应当在controller方法（@Controller标注的那个类里的方法都叫Controller方法）先执行一个方法，让user把自己的date载入了：

前端发请求导航到这个类来，先执行showUser方法（它的形参是从前端那拿到的），该方法返回一个User类；紧接着执行testModel Attribute方法，之前方法返回的那个user类就作为该方法的参数！（有点管道机制那味儿）

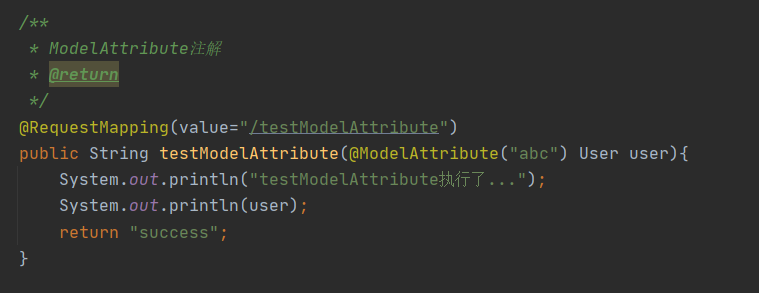
这个不太好理解，SpringMVC框架也提供了一个showuser无返回值的做法：

Showuser参数列表中必须提供一个Map，返回的user就存在这个map中！

Showuser方法写成：



对应的testModelAttribute方法需要写为：



将showuser存入的名为abc的用户打印出来！

【@SessionAttribute】注解

背景知识：Session是什么：

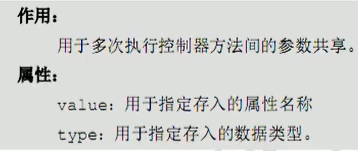
**什么是Session？Session什么时候产生？**

Session:在计算机中，尤其是在网络应用中，称为“会话控制”。Session 对象存储特定用户会话所需的属性及配置信息。这样，当用户在应用程序的 【Web 页之间跳转】时，存储在 Session 对象中的变量将不会丢失，而是在整个用户会话中一直存在下去。当用户请求来自应用程序的 Web 页时，如果该用户还没有会话，则 Web 服务器将自动创建一个 Session 对象。当会话过期或被放弃后，服务器将终止该会话。

**为什么要使用Session？**

因为很多第三方可以获取到这个Cookie，服务器无法判断Cookie是不是真实用户发送的，所以Cookie可以伪造，伪造Cookie实现登录进行一些HTTP请求。如果从安全性上来讲，Session比Cookie安全性稍微高一些，我们先要知道一个概念--SessionID。SessionID是什么？客户端第一次请求服务器的时候，服务器会为客户端创建一个Session，并将通过特殊算法算出一个session的ID，下次请求资源时（Session未过期），浏览器会将sessionID(实质是Cookie)放置到请求头中，服务器接收到请求后就得到该请求的SessionID，服务器找到该id的session返还给请求者使用。

回到注解。SessionAttribute注解的功能：



首先，我们用框架自带的Model类：



参数是model类，调用addAttributes，就会把自定义键值对加入到【request域】对象中！

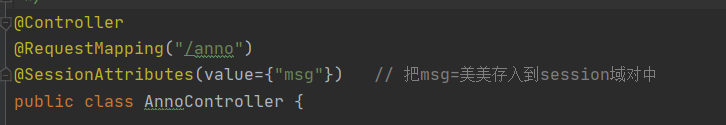
既然域中已经有了该对象，我们就可以在其他页面中调用它了！

比如我们在返回页面success.jsp中：

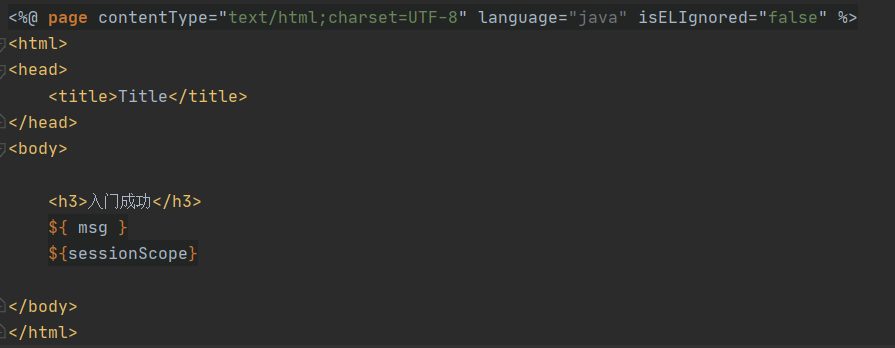


就会看到所有的request域对象，包括msg=美美！

要加入Session域对象，需要在类上加注解：



在返回页面中：



（打印request域中的msg这个键对应的值对象；打印session域中的全部对象）

返回页面就长这个样子：



上面的方法是存域对象值（并打印在页面上）。还可以用Model的其他实现类来实现【取】和【删】。



（前端加上对应的超链接，点击就可以实现对应的取值或删值功能）

至此，我们已经实现了不同方法间共享参数的功能！

【从此处开始第二天项目】

目前为止，我们的controller方法都会返回一个string”success”，便于我们跳转到success.jsp返回页面；如果controller方法没有返回值会怎么样：

点击前端对应超链接，直接报错404.且报错信息：



也就是说，它会在你配置的返回页面包（pages）中寻找名为请求路径（调用方法的RequestMapping中配的路径）的页面！

如果又想给返回页面自己起名字、又想调用方法返回void，怎么办？用servletAPI实现。有【转发】和【重定向】两种方法。

背景知识：转发和重定向的区别：

**forward（转发）**：

是服务器请求资源,服务器直接访问目标地址的URL,把那个URL的响应内容读取过来,然后把这些内容再发给浏览器.浏览器根本不知道服务器发送的内容从哪里来的,因为这个跳转过程实在服务器实现的，并不是在客户端实现的所以客户端并不知道这个跳转动作，所以它的地址栏还是原来的地址.

**redirect（重定向）**：

是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显示的是新的URL.

转发是服务器行为，重定向是客户端行为。

**区别**：

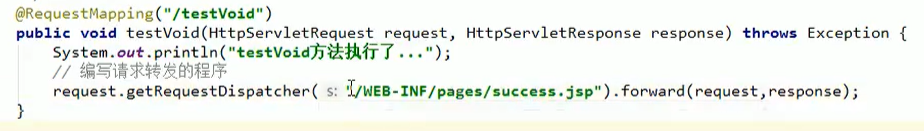
1. 从地址栏显示来说  
forward是服务器请求资源,服务器直接访问目标地址的URL,把那个URL的响应内容读取过来,然后把这些内容再发给浏览器.浏览器根本不知道服务器发送的内容从哪里来的,所以它的地址栏还是原来的地址.

redirect是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显示的是新的URL.

2. 从数据共享来说  
forward:转发页面和转发到的页面可以共享request里面的数据.  
redirect:不能共享数据.

3. 从运用地方来说  
forward:一般用于用户登陆的时候,根据角色转发到相应的模块.  
redirect:一般用于用户注销登陆时返回主页面和跳转到其它的网站等

1.【转发】方法



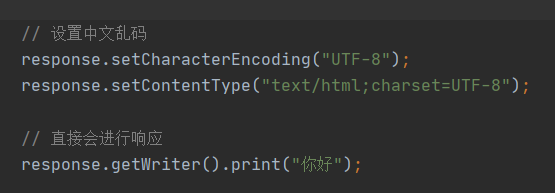
Dispatcher的参数就是转发路径，需要写全路径名！

2.【重定向】方法



重定向请求不到pages里的路径？

还有一种方法，就是在后端直接调用输出流把文本写出来：（不跳页面了，直接在原页面上相应）



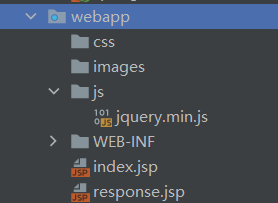
这里还有一个小小的语法糖：用字符串关键字进行转发/重定向：



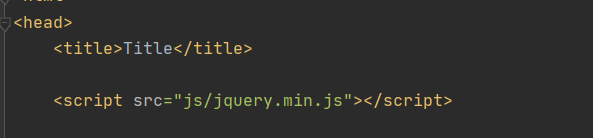
# ResponseBody相应【json数据】

首先我们需要为项目搭建一个异步环境：

Webapp包下新建js文件夹，导入jquery：



第二天的实验我们在response.jsp页面中完成。首先要【引入jQuery库】（它是一个JavaScript的API库，里面装了一堆好用的js函数）：



在head中用script标签引入！

接下来我们试验一下异步请求（ajax请求）。首先在页面中加一个按钮：

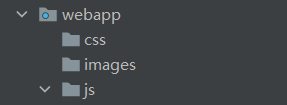


然后要在本页面中【**编写js代码绑定事件**】！

先写一个最简单的弹出提示：



也是写在script标签里。

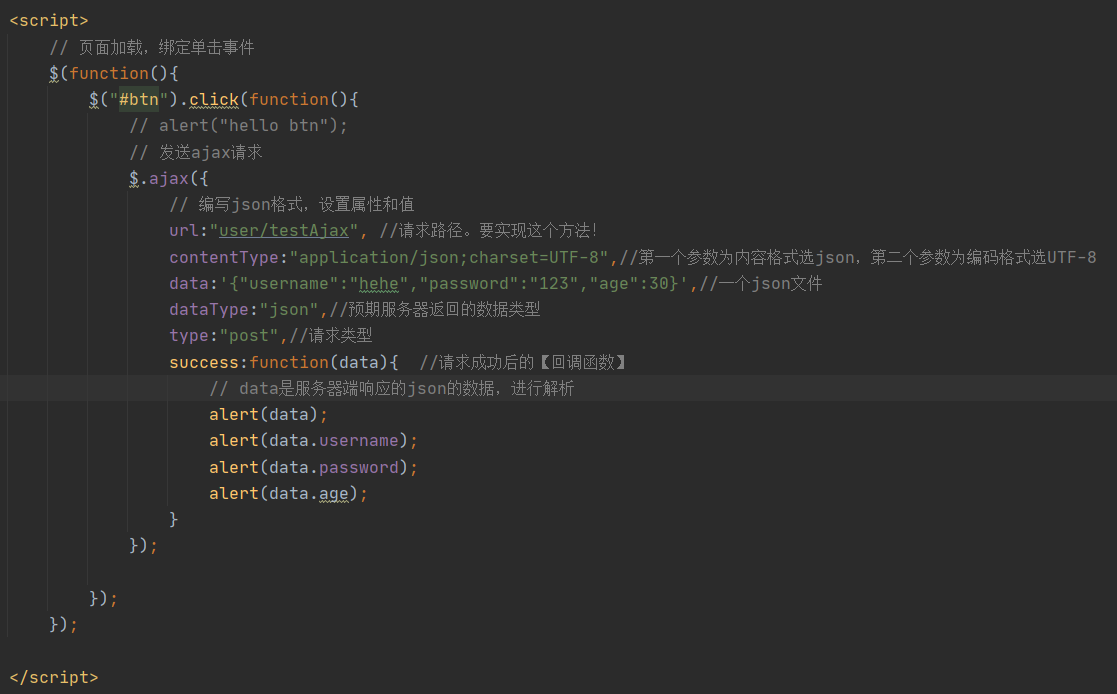
要使用js中的资源（我们引入的jQuery库），需要重新配置一下，不能让前端控制器拦截必要的服务器静态资源（一般而言就是这仨：）。

在springmvc.xml中：



现在我们点击页面按钮可以弹提示了！

解析来我们试试发一个ajax请求。先看一下最终的代码：



Ajax请求内，那些紫色的都是ajax请求固定的属性（实际开发中可以查jQuery文档，常用的基本就这几个）！打一个字母就能补全的那种。