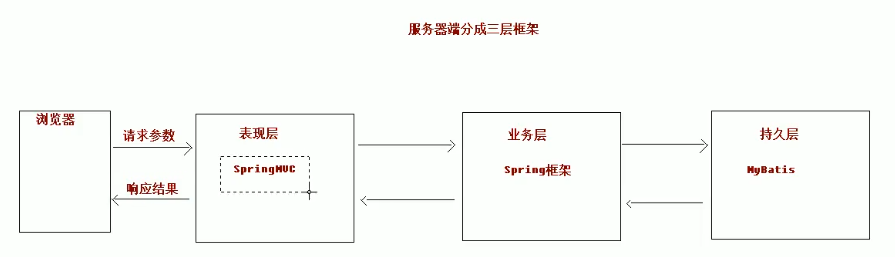
# 第一天

## 一： 三层架构以及mvc设计模式

1. ：javaee项目的三层架构：分别是表现层，业务层和持久层。对应了ssm框架技术中的不同框架。

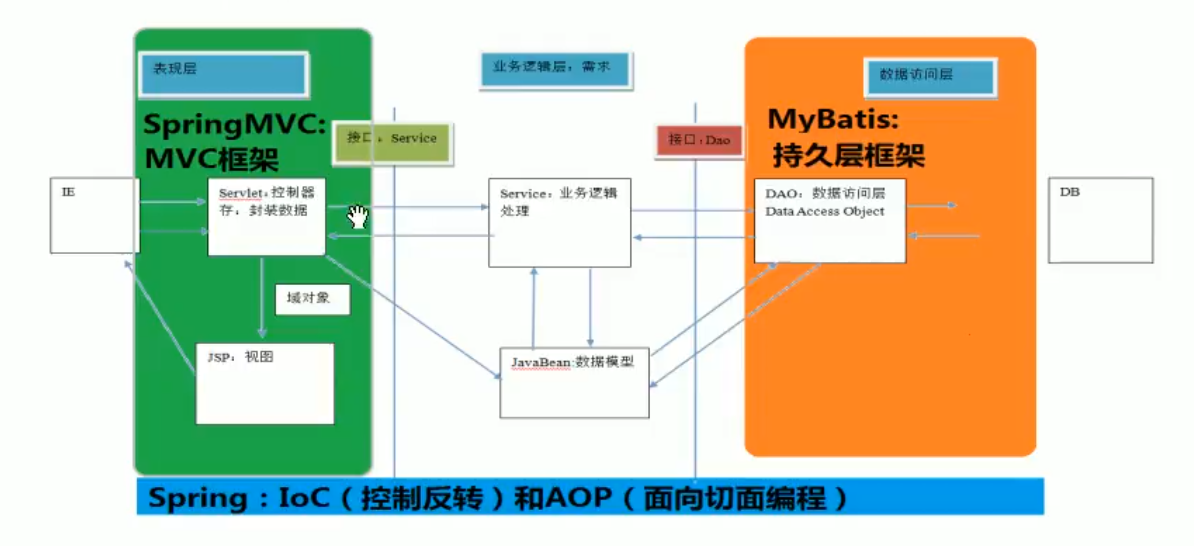


1. ：mvc(model(java bean对象,用来封装结果)，view(jsp等，用来页面显示)，controller(servlet，用来接受请求完成数据查询，bean封装并返回数据的))设计模型，表现层一般都会采用mcv设计模式。

SpringMVC就是以上的思路开发出来的框架技术。

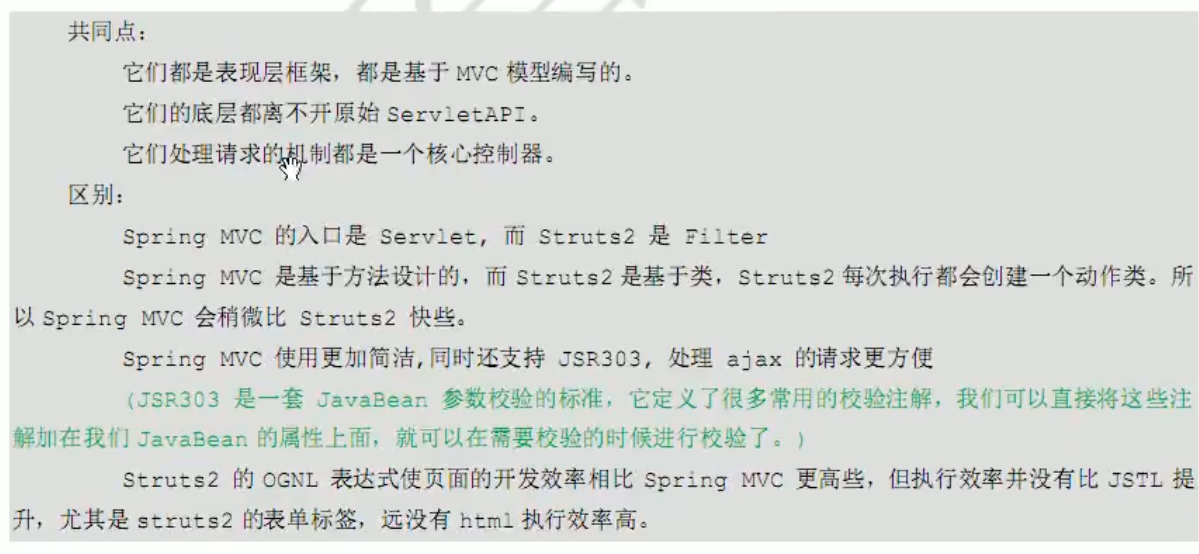
## 二： springmvc框架介绍

Springmvc在三层架构中的位置：



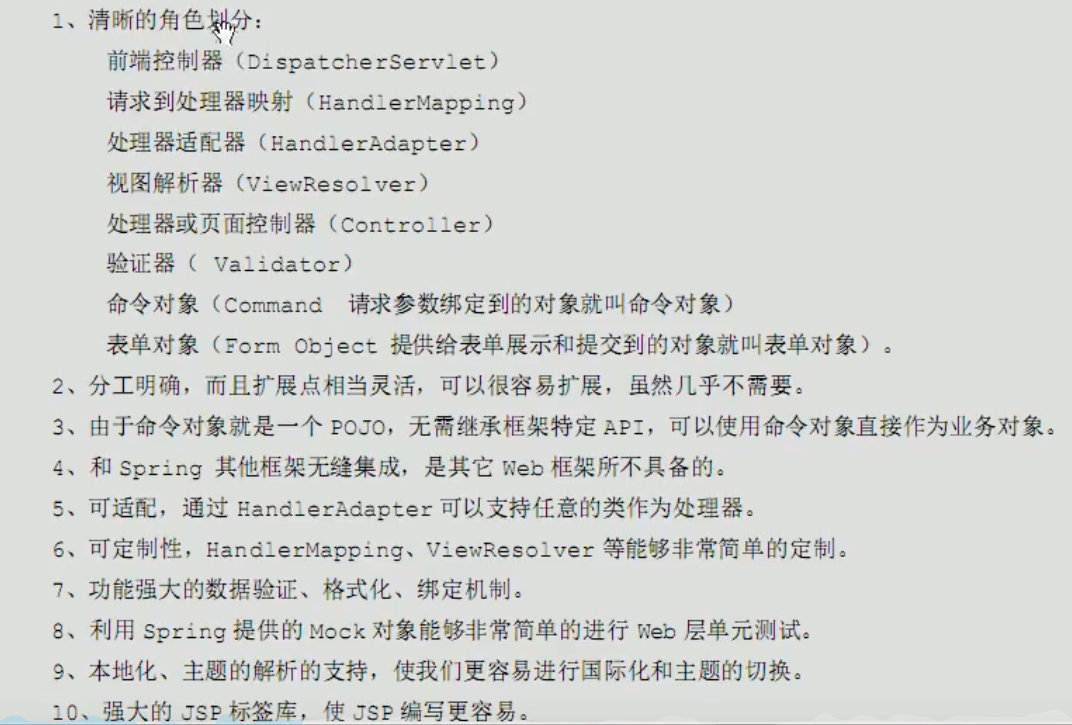
Springmvc和struts2框架的异同点

其实二者本质上依然依赖于servlet那一套，只要接受请求，一定是request，发送响应，一定是response。Mvc是单例的，struts2是多例的。

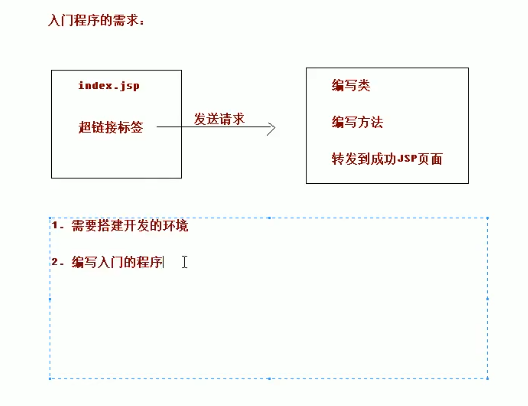


Springmvc优势：

各种器(组件)来配合工作完成任务，实现了灵活度。可自由配置，另外他还支持restful风格。

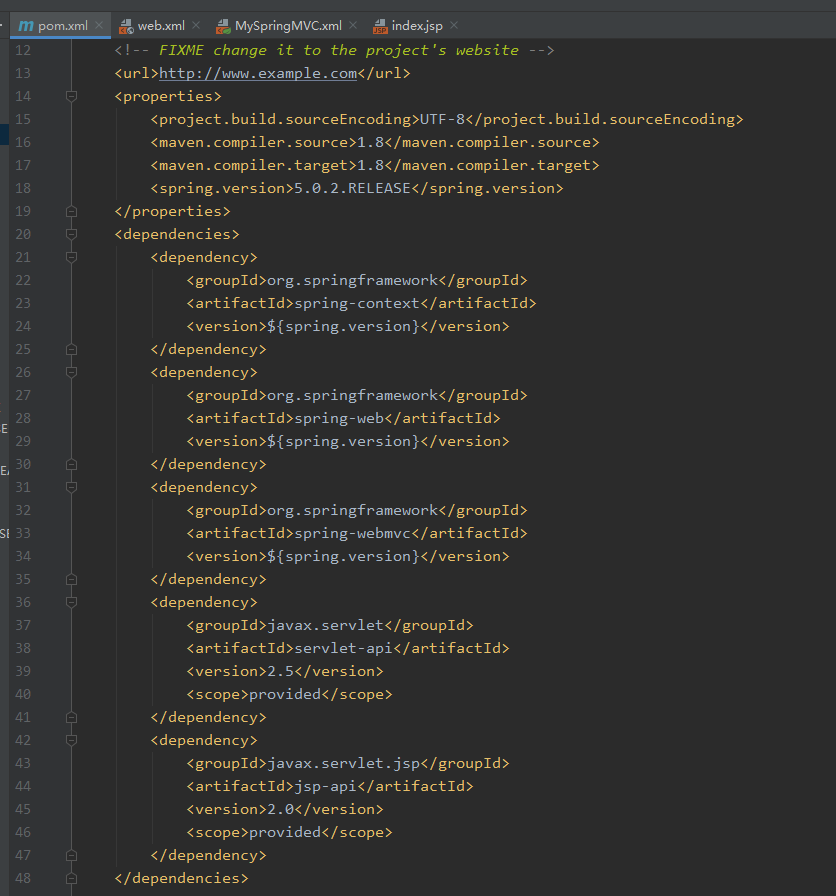


## 三： 入门程序需求

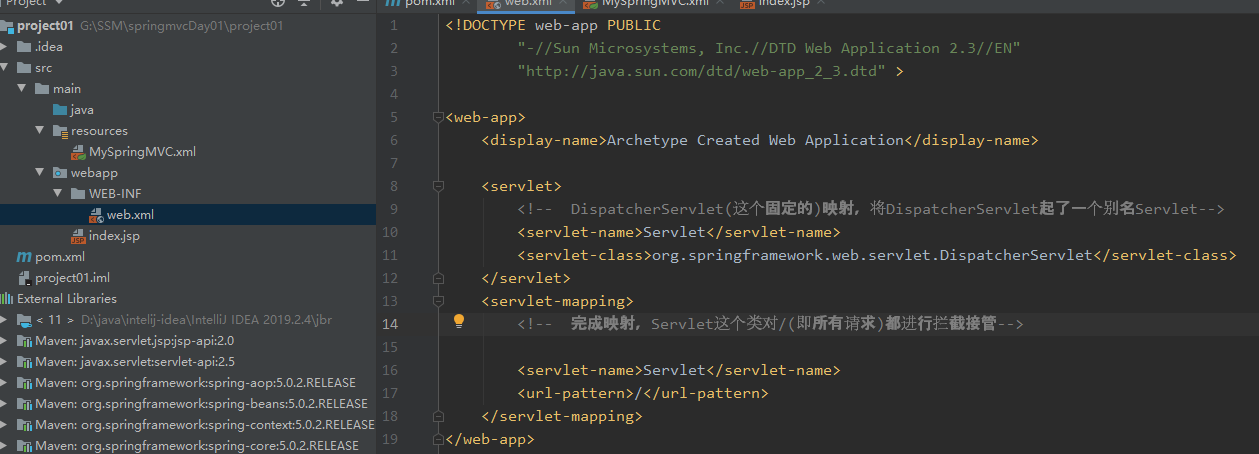


## 四： 环境搭建

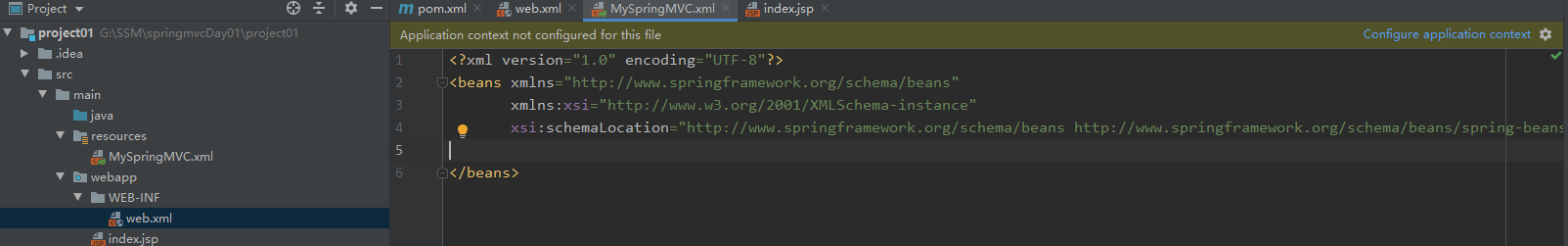
首先创建一个maven项目文件，选顶wepapp结构，进入项目，补全项目结构(添加java目录和resources目录)，导入maven坐标：



在web-inf文件夹下的web.xml文件中进行springmvc配置：（意在让所有的请求都被mvc接管）如下将dispatcherservlet类映射到/，实现目的。

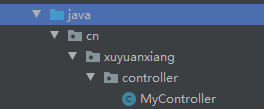
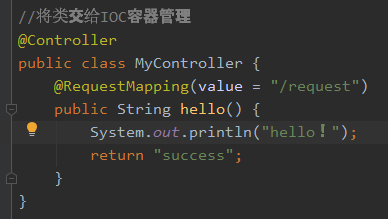


Resource目录下创建springmvc的配置文件：专门用来以后配置springmvc（类和页面请求进行映射。）

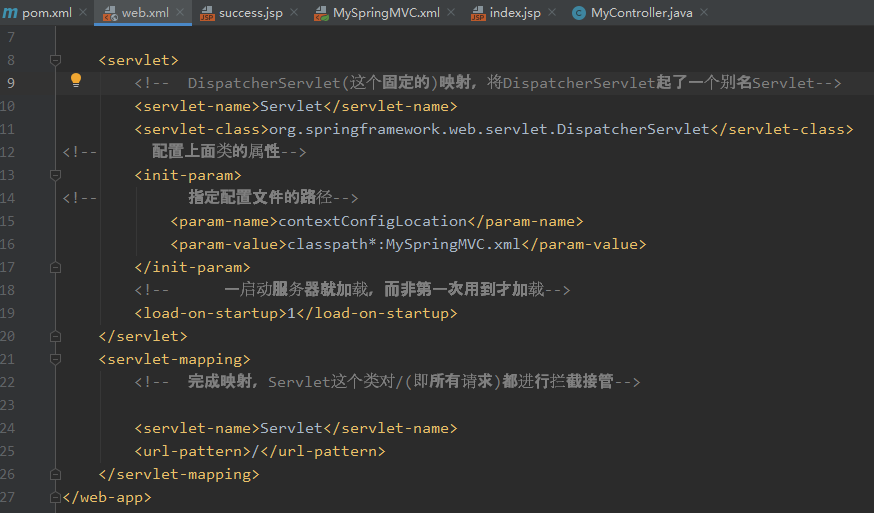


## 五： 入门程序编写步骤

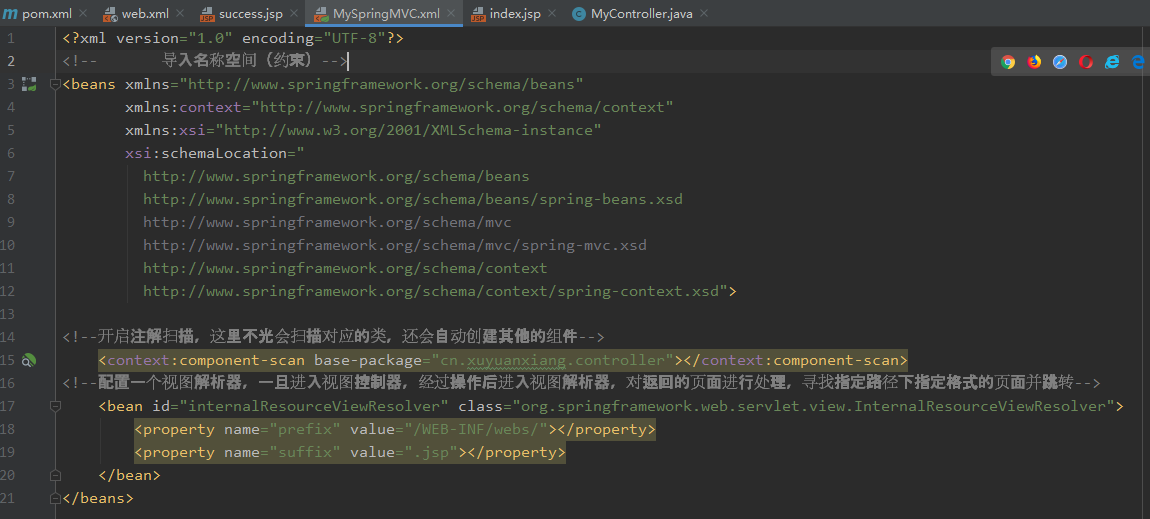
1. 于java目录下创建一个controller类，用来实现请求接受和页面跳转：

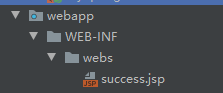
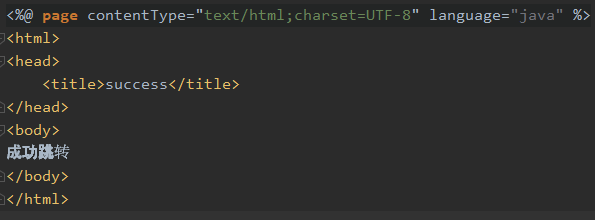
1. 在web.xml文件中指定springmvc的配置文件：



1. 在springmvc配置文件中配置视图映射器，将指定的目录映射为返回跳转页面。另外这里也需要开启注解扫描，这样才能扫描指定类的注解，其中的注解才能生效。



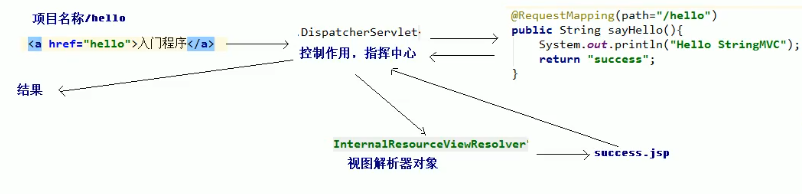
1. 在指定的映射目录下创建对应的jsp文件实现跳转到本页面。

## 六： springmvc流程略述

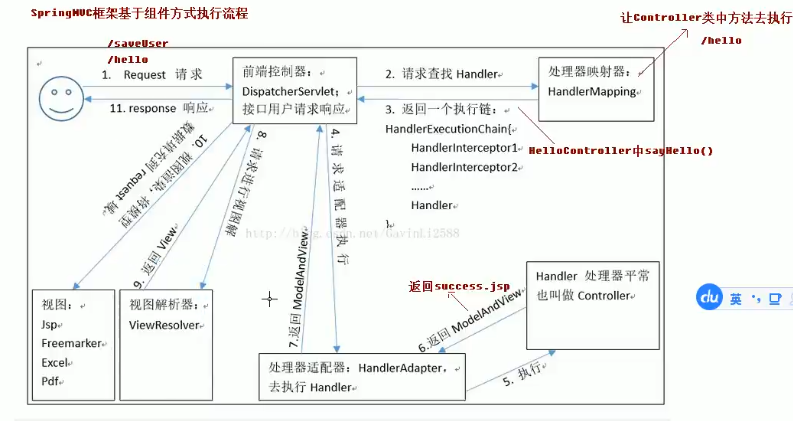
启动服务器===>加载web.xml配置文件===>dispatcherServlet对象被创建===>解析springmvc.xml文件===>开始扫描controller类中的注解，创建了对象加入到容器中===>创建一个视图解析器

发送请求===>被dispatcherServlet(核心控制中心：前端控制器)拦截===>执行类中的方法/request===>返回的success经过视图解析器转为/web-inf/webs/success.jsp===>跳转到此页面显示



## 七： springmvc流程详解

这里详细讲解了springmvc在处理过程中各个组件的协调作用



其中前端控制器和视图解析器我们已经配置了，其他没有配置的其实在这里已经自动配置了

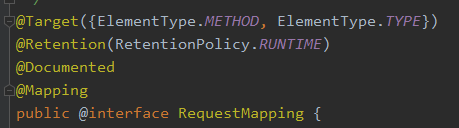


## 八： springmvc中requestMapping注解

用来建立请求和方法的关联：



它的原型：

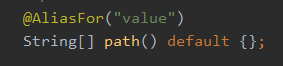


它可以放在类上面也可以放在方法上。如果一个请求有多级路径（user/success），就可以如此拆分：例如下，在类上添加了@requestMapping就必须修改href了，否则无法实现正确跳转。

## 八： springmvc中requestMapping注解属性详解

1. Path和value互为对方的别名，所以他俩都可以指定映射的资源路径。

1. Method属性用来指定请求方式，只有请求方式与指定的请求方式对上，才可以进去方法执行体。通过枚举类型指定。





1. Parms属性用来指定请求参数的限定

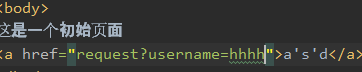


请求参数里必须有一个username，没有就不会执行

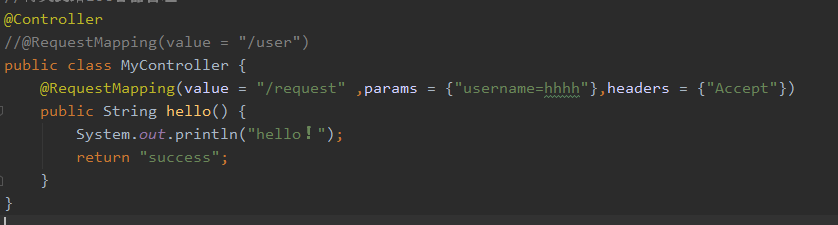


请求参数里必须有一个username，且参数对应的值为hhhh，否则就不会执行

例如：



1. headers属性用来指定请求头的限定，请求头里必须有accept否则不执行



## 九： springmvc中请求参数的绑定

支持的数据类型：

1基本数据类型

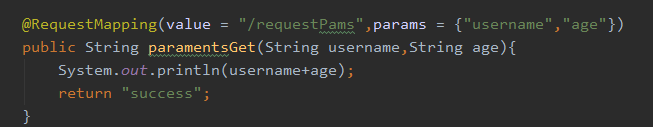
2 javabean

3复杂类型，集合，map等



如图，如果请求参数中的名字和mvc@controller类中的参数名字正好对应，就可以实现自动封装，方便调用

例如：

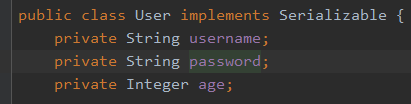




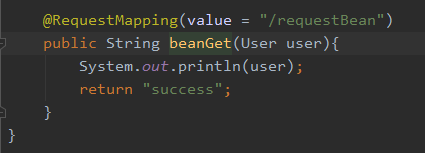
这里就会把参数名字相同的自动进行封装，但是有时后参数实在太多，我们需要一个实体的javabean类来进行参数的封装，请看下节课！

## 十： springmvc中请求参数的绑定到javabean实体类中

首先写一个javabean封装类：



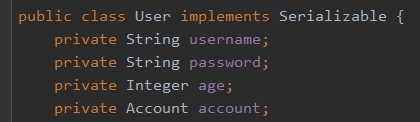
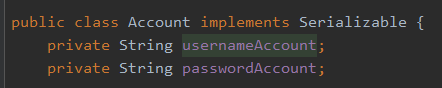
在request求情中方法参数直接写javabean类



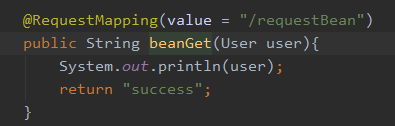
链接路径中指定参数，参数名字需要和javabean中的属性名一直，否则无法封装。



如果javabean中又包含了其他的javabean：（这里的javabean要实现序列化接口）

在参数封装的时候，参数可以以xxxx.xxxxx来进行封装。



当然这里的参数比较多的时候可以采用表单提交！



## 十二： springmvc中请求参数的绑定的中文乱码问题

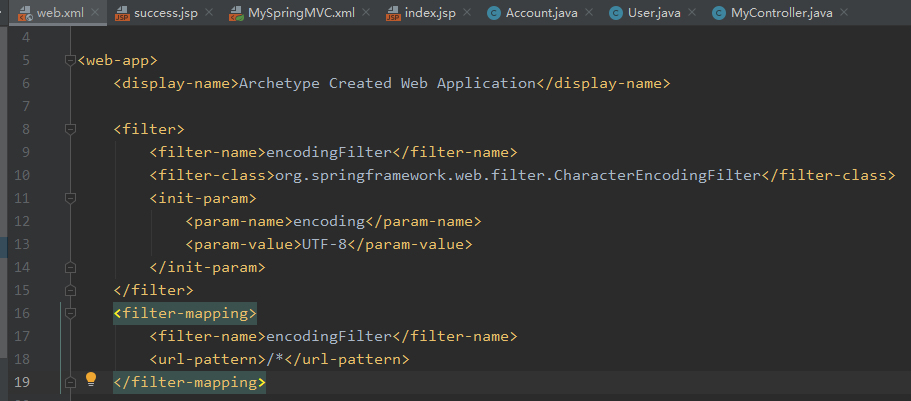
Get方式已经解决，而post方式没有解决呢如下：





解决方法：配置一个全路径的过滤器，在转到指定路径之前经过过滤器，添加属性更改编码方式就可以了！

在web.xml文件中新加一个过滤器，这个方式的顺序受到约束的影响，需要放到servlet前面

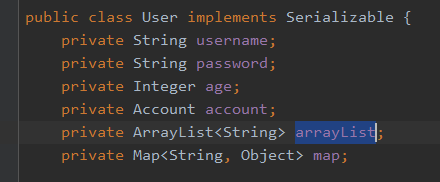


再次运行，正常编码了！



## 十三： springmvc中请求参数的绑定之复杂参数

先修改user类：



修改提交表单的地方：

对于list采用list[index]的方式指定具体索引

对于map采用map[键]的方式指定键的名称



最后参数绑定结果：

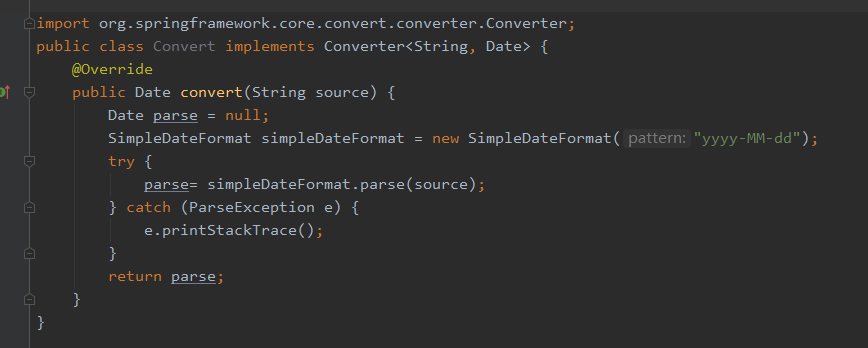


## 十四： springmvc中请求参数的自定义类型转换器

Springmvc后台会自动转型参数类型，例如上面所有href参数都是string型的，我们得到的十string，填入的age会自动变成interge型的。但是对于日期格式，默认的xxxx/xx/xx格式的，如果写入xxxx-xx-xx就会报错。这里就需要我们自定义类型转化器。

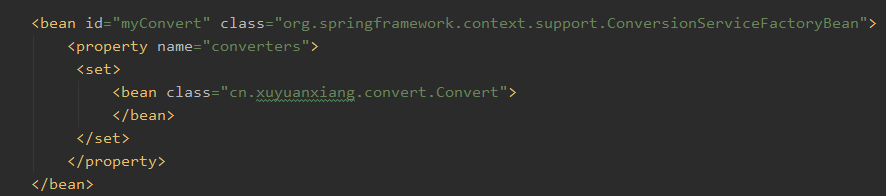
类型转换器使用步骤：

1. 首先创建一个converter的实现类，两个泛型，一个用来指定要被转换的类型，一个指定转换后的类型，实现其中的convert方法，返回转换后的变量，其中source就是传入的变量



1. 将上面的自定义类型转换器配置到容器中：

在springmvc.xml中实现如下配置，意思是在转换格式工厂中添加了一个转换类型（实际的操作是将我们自己写的转换类添加到了ConversionServletFactoryBean容器的一个set集合属性中）



1. 在开启注解框架支持的配置中添加我们的convert组件，时期生效。

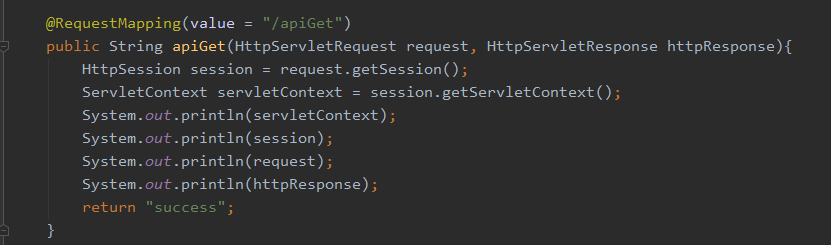


需要注意的是这里以前的格式就失效了！

## 十六： springmvc中获取servlet原生的api

如果想在控制器的方法中获取request，response等

获取方法非常简单：直接在参数中指定想要获取的类（可能这里是单例的？所以可以如此获取！）





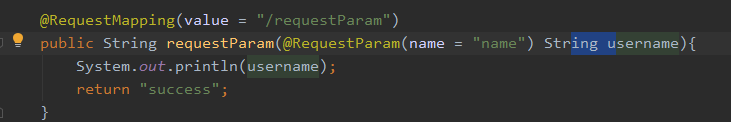
## 十七： springmvc中的requestParam注解

上面的请求参数要求页面提交的参数和我们方法中的参数名字一样的，否则无法自动进行封装，而我们想强行把名字不对应额请求参数进行封装就需要用到这个注解！

首先编写一个跳转页面标签，提交求情参数name



在@Contrlloer类下的方法中接受并封装请求参数，由于名字一个是name，一个是username，所以需要采用@RequestParma注解进行指定。需要注意的是一旦指定，页面的求情参数必须有name这个参数，没有不行，是username也不行！

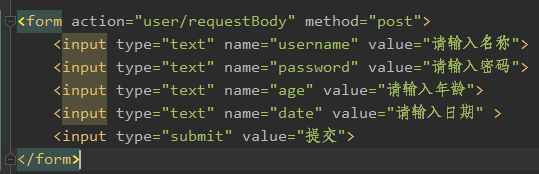


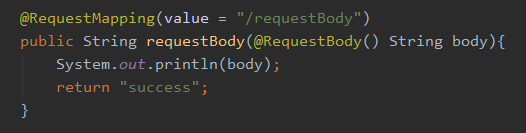
## 十八： springmvc中的requestBody注解

@requestBody注解用来获取求情体的所有内容，适用于post求情方式，因为get方式没有请求体。

一般用于异步界面展示，json数据的传递。

实例如下：





## 十九： springmvc中的pathvariable注解

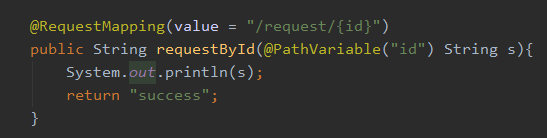
这里需要结合restful风格，非常重要！！！结构清晰，利于缓存！

以前的访问方式是通过/user/save；/user/find等等。每一个方法都有自己的请求路径。

而restful每个方法的请求路径都是一样的，我们根据请求方式来区分，或者根据请求路径后面的路径名来区分。例如：get方式的/user；post方式的/user；put方式的/user，会执行不同的方法体。也有/user/{id}，这里的{id}表示占位符，可以用后面的id值来进行区分，请求时输入/user/10就是把id=10传入。

实例如下：





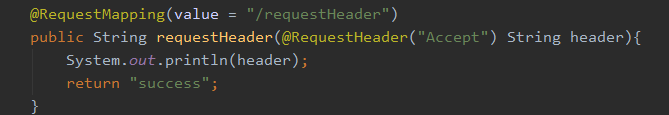
## 二十： springmvc中的HiddenHttpMethodFilter注解

由于我们表单只能提交get和post方式请求，如果想提交put等方式，就需要他，但是非常麻烦，我们一般不用，现在前端有各种插件来帮助我们更改请求方式例如thymeleaf

## 二十一： springmvc中的RequestHeader注解

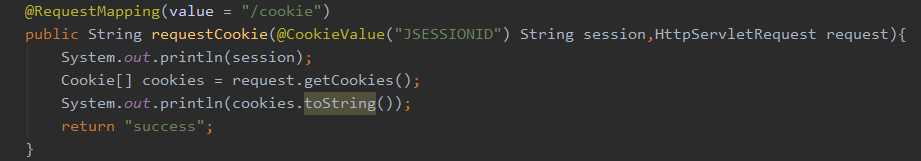
专门用来获取请求头的注解：





## 二十二： springmvc中的Cookievalue注解





由于session是通过cookie存贮的，所以这里通过cookie的键名JSESSIONID来获取session。

## 二十三： springmvc中的ModelAttribute注解

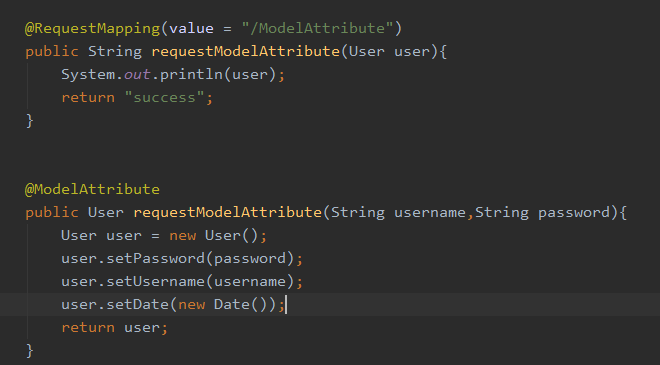
@ModelAttribute放在方法上面，所有其他方法执行之前都会有限执行此方法。

作用场景：当我们表单只提交了部分值，而剩下的值需要从数据库中查找完之后补齐，就需要用到这个注解来进行查询数据库，之后再传入后面的方法中。

第一种实现方法：（有返回值）

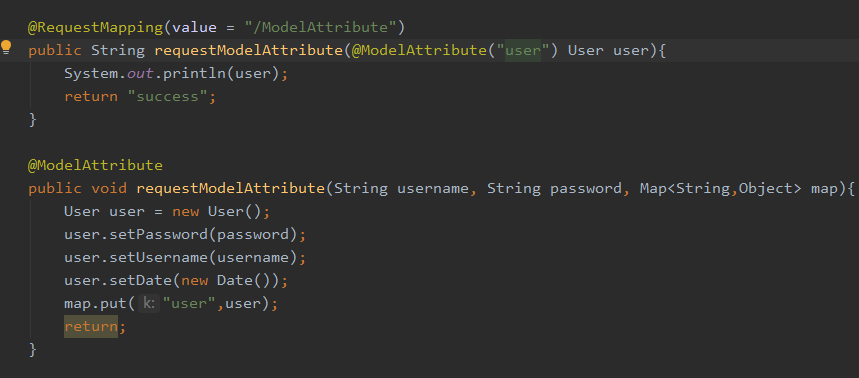
例如：这里参数值传递了name和age，但是由于注解@MethodAttribute的作用，我们为user添加了date属性，并且将user返回，再啊接下来的方法中获取到的user便有了data属性。





第二种实现方法：（没有返回值）

由于没有返回值，又想把user对象传递过去，所以只能传入一个map参数，再获取map集合中的键值对。



## 二十四： springmvc中的SessionAttribute注解

为了解耦，springmvc摒弃了httpservletrequest，改用model(底层是一个hashmap)来代替他充当request与对象(二者本质上是一样的！)

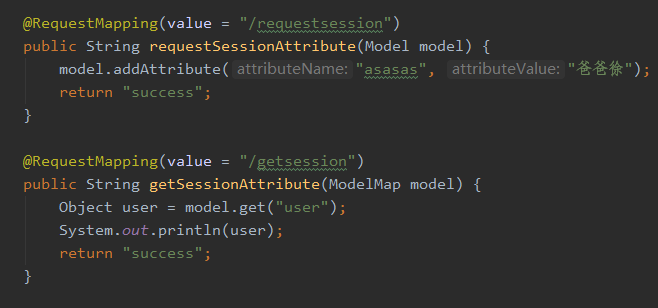
首先编写超链接实现跳转页面！



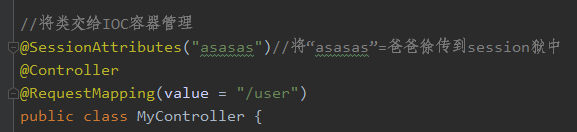
在controller注解类中添加如下方法，

Model就相当于request域对象，我们往model中添加键值对。

在第二个方法里，采用modelMap来获取session中user键对应的值



@SessionAttributes只能用来修饰类，将asasas键和对应的值添加到session对象中去。

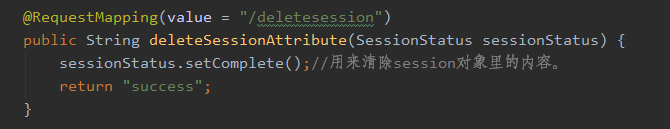


在success.jsp中通过el表达式获取域对象。





此方法用来清除session中的内容。

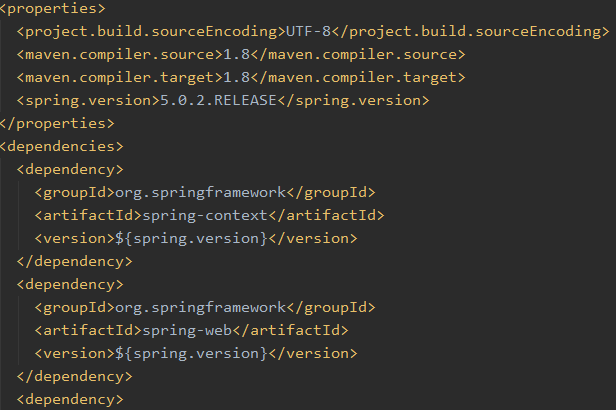


以上：G:\SSM\springmvcDay01\project01

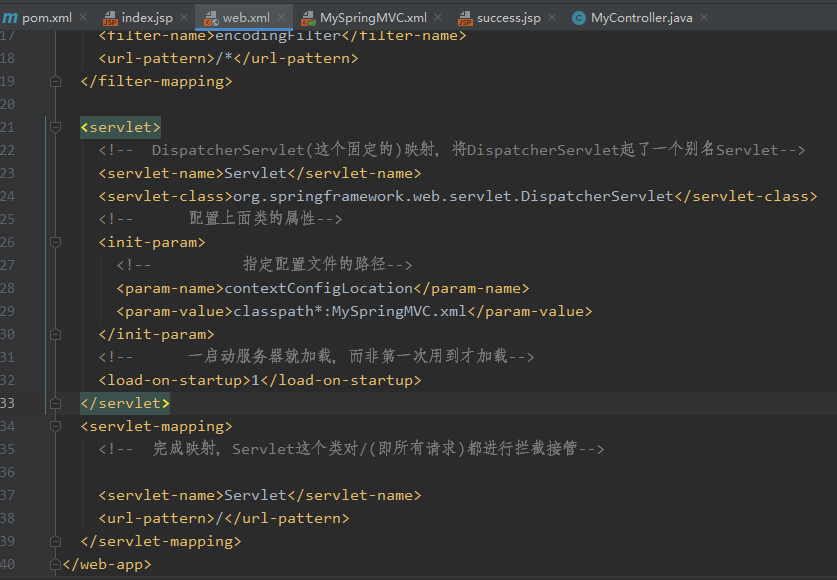
# 第二天

## 一： 环境搭建

固定版本以及导入maven坐标



Web.xml文件创建前端控制器，指定springmvc的配置文件，创建过滤器解决中文乱码问题



在springmvc.xml文件中开启注解扫描并且配置前端控制器：



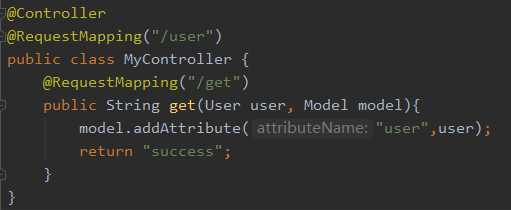
## 二： 响应之返回值类型为string

如果配置了视图解析器，返回值就会被封装成指定路径下的指定格式文件，然后跳转到指定页面，通常用于显示数据：例如

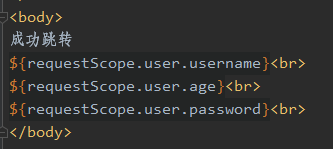
创建一个超链接，实现指定资源路径的跳转



这里的user封装了我们的请求参数，这里的model相当于以前的request域对象，专门用来进行数据共享。返回的success被封装成了web-inf/webs/success.jsp路径。



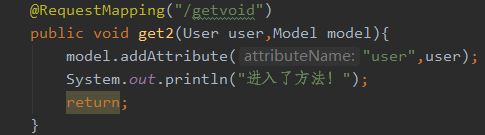
跳转后通过el表达式获取域对象中的值



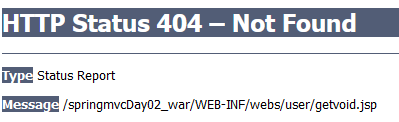
## 三： 响应之返回值类型为void

如果返回值是void：





这里提示请求路径不存在，说明我们返回页面有一个默认路径就请求路径！



如果我们依然想要实现页面按照我们的意愿跳转，需要实现类似以前servlet中的请求转发和重定向的方式。这里由于是采用的原生转发和重定向，视图解析器失效，所以要写全路径名。

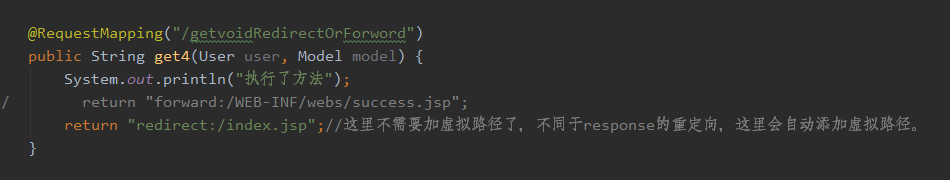
这里发现路径前面加“/”是不是就是从当前类路径的根目录开始，而不加“/”就表示从当前目录为相对路径开始呢？





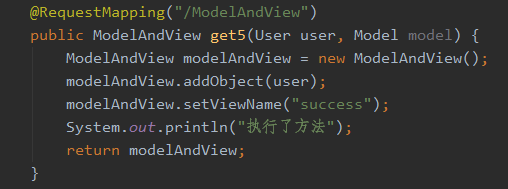
## 四： 响应之使用forword和redirect实现页面的转发

这种方式也不支持视图解析器。(貌似在springboot中是支持视图解析器的？)



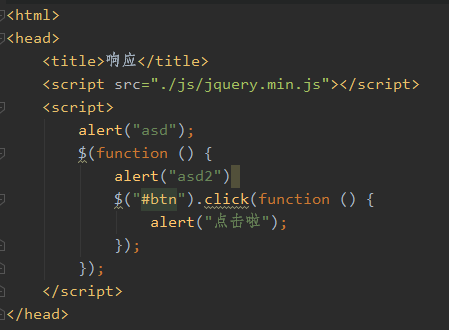
## 四： 响应之返回值是ModelAndView(对象和视图)对象

创建一个modelanview对象，设定要传输的对象，并且设定要跳转的路径，将此对象返回，就会自动跳转我们添加的路径，并且携带了我们要传递的对象。其实我们以前写的返回String类型的底层原理就是这个！



## 五： 响应json数据之过滤静态资源

如下引入jquery文件，但是由于前端控制器对所有的路径都进行了拦截，这里引入的jquery也就自然而然地被拦截了，所以实际上并没有引入，当然下面的事件也没有成功添加。



想要释放拦截资源需要在springmvc.xml文件中添加代码：

其中的location是实际路径，mapping是我们映射的虚拟路径。



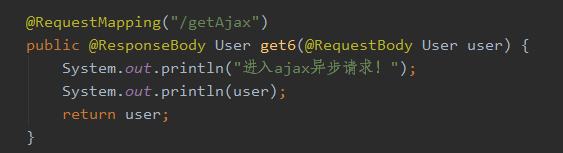
## 六： 响应json数据ajax请求

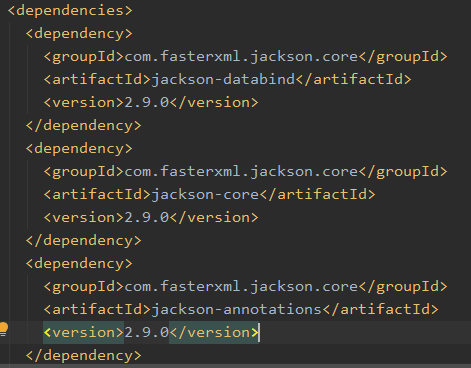
首先前端编写ajax请求，采用的原生js



之后跳转到页面，页面中需要指定返回值是json数据格式，所以在User前面添加了@RequestBody注解，否则返回user了。

值得注意的是：ajax请求发送来的参数也是json格式的，单纯的利用以前的User user 已经不能够将数据封装进去了，我们在参数user前面添加了@RequestBody注解，就可以实现将json格式转入user对象中。这里底层用到一些api，需要我们导入maven(jackson包)：





## 十二： 文件上传以及文件上传原理

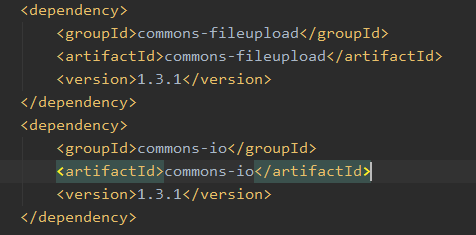
首先对表单有一定要求：

表单的enctpye必须是如下属性值，且必须是post求情方式，因为文件太大，get对参数大小有要求，需要注意的是一旦设置了enctpye为如下属性，request中的getparameter就是没有作用了，并且请求参数的格式已经发生了变化！



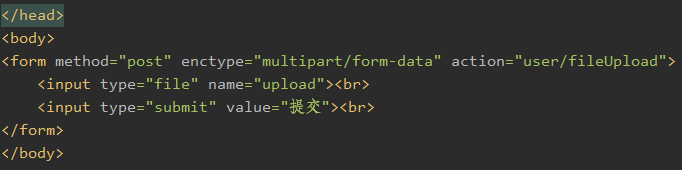


上传的文件想要被解析到，就必须用到指定的api，需要我们引入如下jar包！



## 十三： 文件上传之传统方法

首先前端页面完成表单的提交：



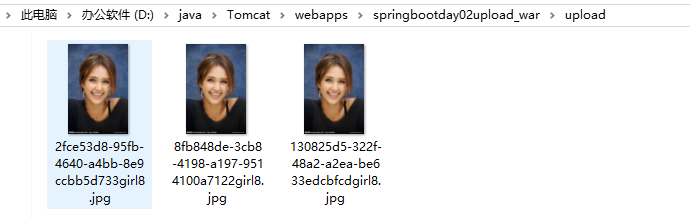
接下来跳转到了controller注解类下面，通过request方法获取真实路径(存不存在无所谓，不存咋就创建就完事了)，然后创建diskFileItemFactory工厂类，创建ServletFileUpload类传入参数diskFileItemFactory，得到的对象就可以用来解析传入的request域对象，返回所有请求参数的list集合，我们遍历集合，判断如果是普通类型的参数，跳过，如果不是，获取这个请求参数文件的名字，然后将文件写入指定的路径。



此为打印输出的路径：



多次上传，我们在文件路径就可以发现文件！



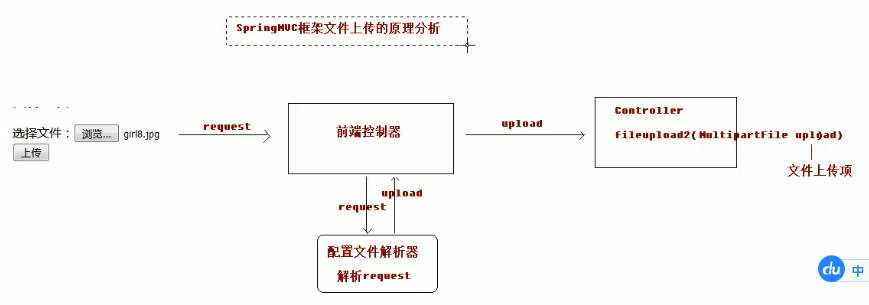
## 十四： springmvc文件上传原理

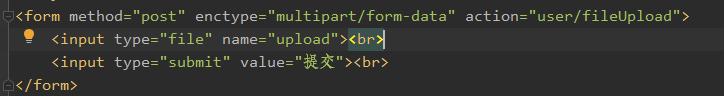
可以配置一个文件解析器，发送请求会经过前端控制器，前端控制器便调用文件解析器，解读request中的文件部分并返回，在由前端控制器传入controller中进行上传。

对于返回的文件有要求：

方法参数名字和前端name属性的值必须一样，这样才能实现自动封装；

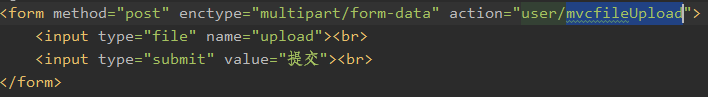
方法参数中的类对象应该是Multipartfile，必须是！





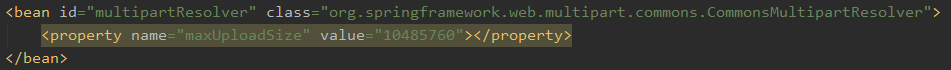
## 十五： springmvc文件上传代码实现

首先创建前端标签页面：

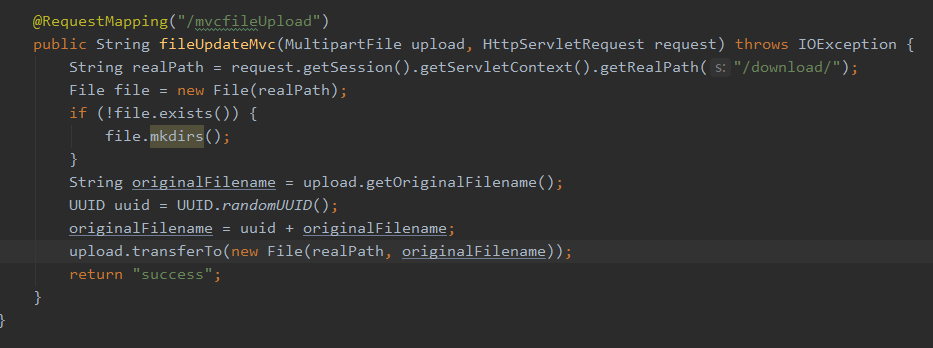


在springmvc.xml文件中配置好文件解析器：参数可以指定，这里仅仅指定了文件允许大小

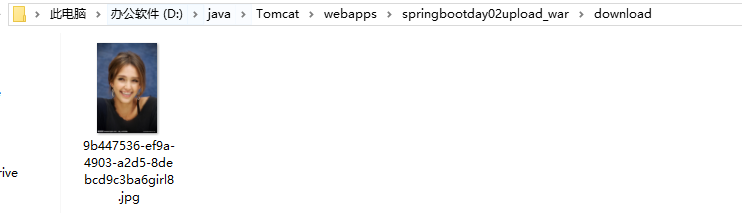
不同以往的解析器，这里的id属性值必须如此，这是规定！



在controller中实现文件的，只要我们的传入参数名字和前端form表单文件的请求name属性值一致，就可以实现文件的自动封装。注意这里的参数类型必须是MultipartFile。

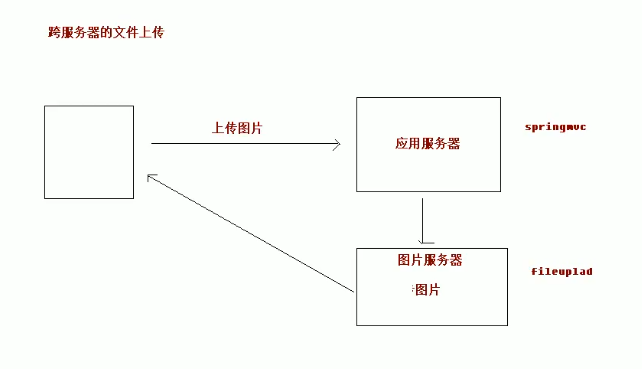


最后完成了文件的上传，在指定的路径便可以查到我们的上传文件的路径和文件。



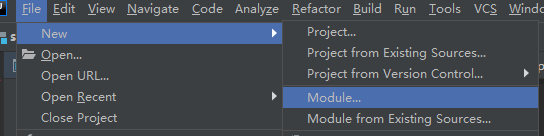
## 十六： springmvc跨服务器文件上传分析

实际情况中会专门有一台服务器用来保存用户的上传文件，实现功能的分离。

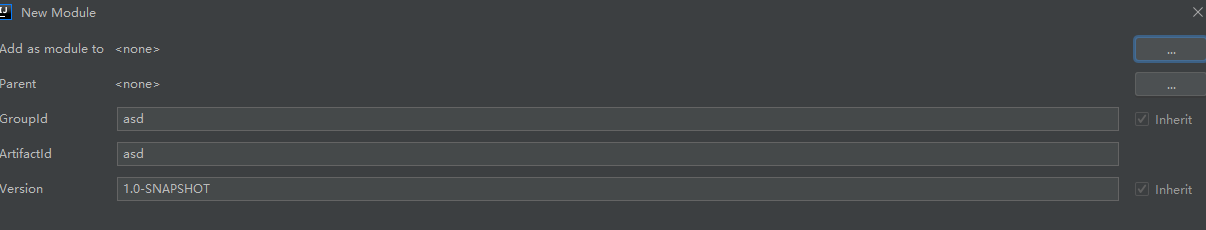


这里采用两个tomcat来实现模拟：

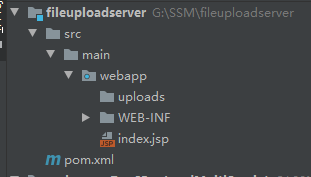
在上一节的基础上在新建一个webapp的maven项目



在这里将前两个都设置为none，表示此项目不是任何项目的子项目。项目相互独立。

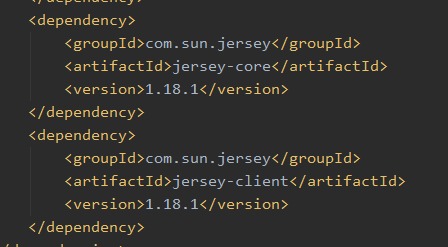


之后完成配置就好了（创建uploads），主要是为了创建一个目录专门来存用户上传的内容！

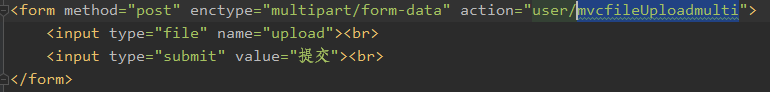


## 十七： springmvc跨服务器文件上传代码实现

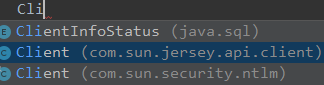
由于需要使用到指定跨服务器上传的api，所以这里导入两个重要的maven坐标



前端页面完成表单的提交！



@Controller注解类下的方法实现功能的时候需要导入类，千万不要导错了！



核心部分的内容：

创建client服务器，通过Client静态方法!

获取链接，通过resource方法，参数写要连接的地址。

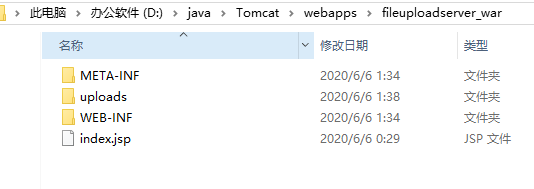
上传，通过upload方法，参数必须是字节流！！！



这里上传过程中会遇到403报错，是因为tomcat服务器默认文件只读，不可以保存和删除，需要在tomcat里的config文件中的web.xml文件添加如下属性值。当然也可以添加在我们项目中的web.xml中！



上传后又出现409错误，原因是没有这个uploads的文件夹，手动补全

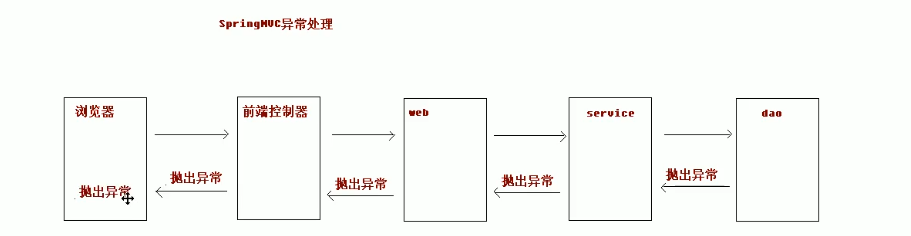


再次上传！成功！

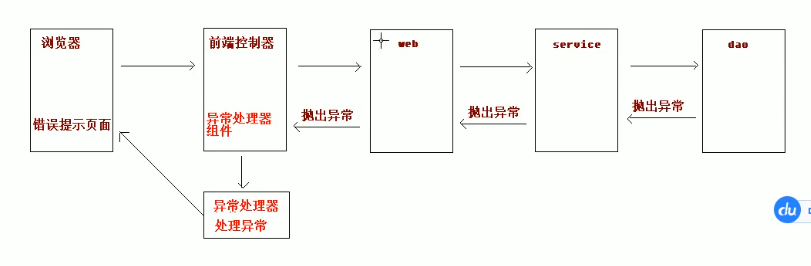


## 十七： springmvc异常处理原理和环境搭建

默认情况下，我们的项目在任意一个环节出现异常，他都会向上抛出，直到浏览器出现错误页面！



Springmvc帮我们实现了异常处理的模式，需要我们来配置一个异常处理器，在前端控制器就会默认返回到异常处理器中，在将我们的异常页面返回给浏览器，而不是直接返回默认的异常页面！



## 十八： springmvc异常处理代码实现

、首先编写一个异常处理器的实现类：

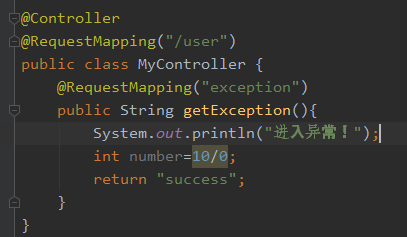
继承HanderExceptionResolver接口，实现其中的抽象方法resolveException，方法的参数有一个exception这就是我们接收到的异常，可以进行获取信息。方法的返回值必须是modelandview，视图解析器会帮我们跳转到指定的错误显示页面！



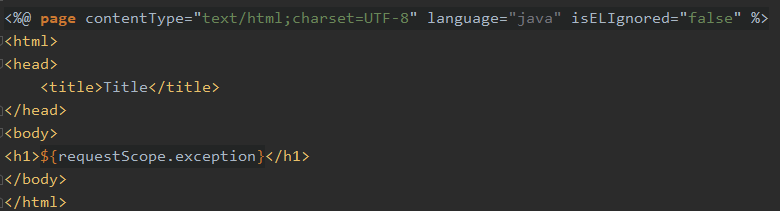
将我们自己写的异常处理器添加到ioc容器中。



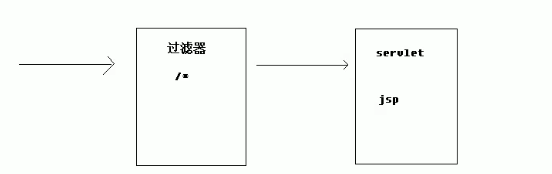
在@Controller注解标注的类中实现一个异常！

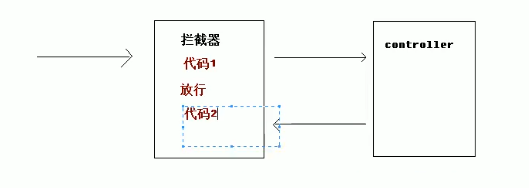


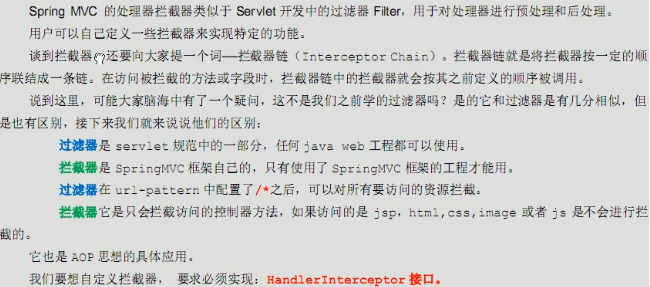
编写错误页面，获取我们异常信息！



## 十九： springmvc拦截器的原理的环境搭建







总的来说就是过滤器是属于javaweb的，所以只要是web项目九可以生效而且拦截所有的请求，拦截器是springmvc框架提出的，所有必须依赖于mvc框架，并且拦截器只会拦截@controller控制器下面的请求，不会拦截例如jsp，image，is文件等。

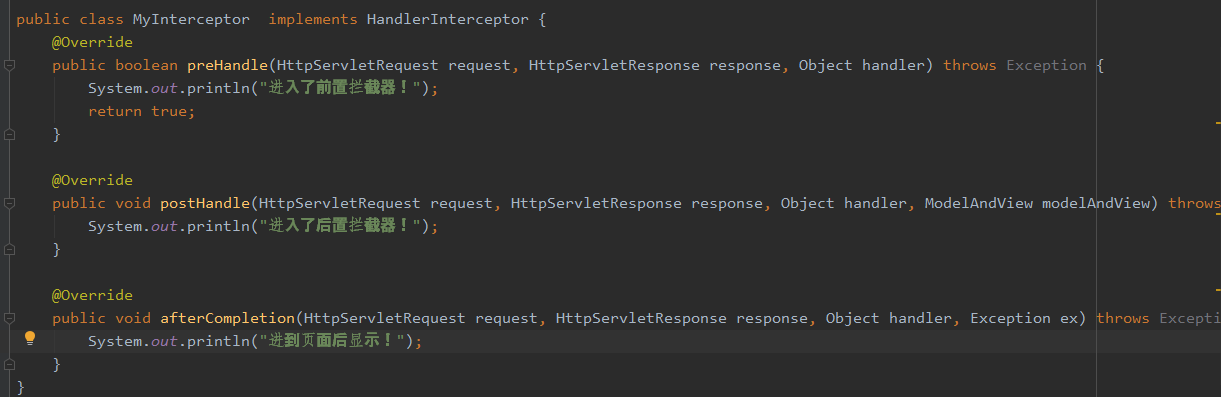
拦截器能做的，过滤器都能。

过滤器能的，拦截器不一定能。

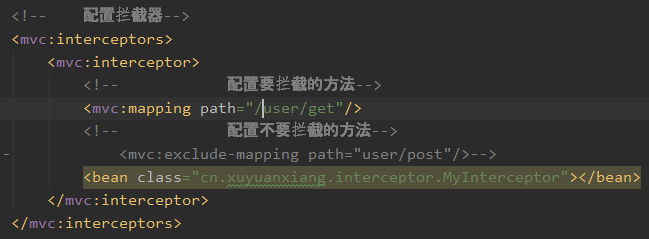
## 二十： springmvc拦截器的代码入门

首先编写一个控制器的类。此类必须继承HanderInterceptor接口，重写其中的三个方法（分别对应的是拦截前执行(返回值为true就会继续执行，否则直接可以request或者response跳转指定页面！)。拦截后执行。到页面后执行。）

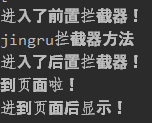
这里发现不重写接口的方法竟然不报错了，这个是jdk1.8之后的特性，可以在接口中写default方法了！



之后再springmvc.xml文件中配置我们编写的拦截器！

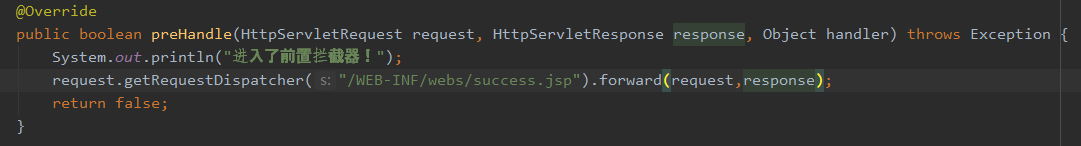


执行后结果显示:



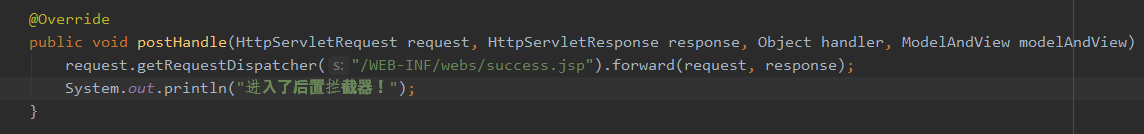
## 二十： springmvc拦截器各种使用方式

前置方法返回false，直接转发到指定页面（这个以后很常用！）

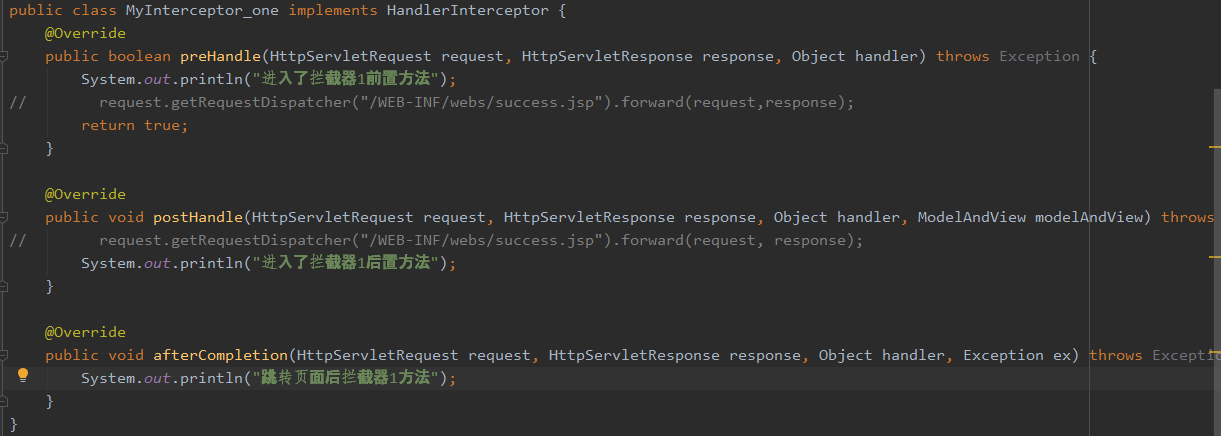


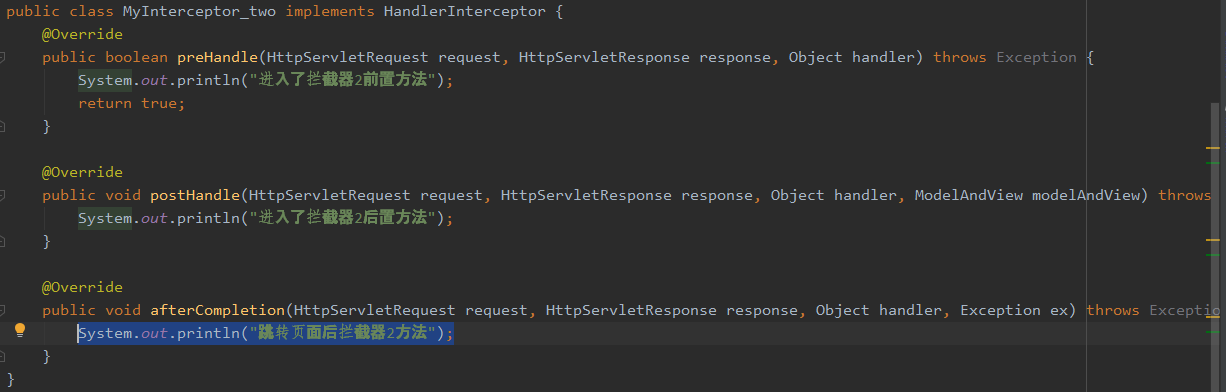


后置拦截也可以实现页面的跳转。（值得注意的是所有的web-inf下的jsp页面是不能通过链接直接访问的，这是tomcat的内部配置！想要访问就只能通过action或者@Controller）



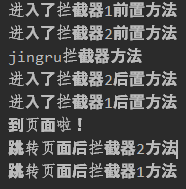
多个拦截器：按照配置顺序进行拦截！







执行结果如下：先进后出！

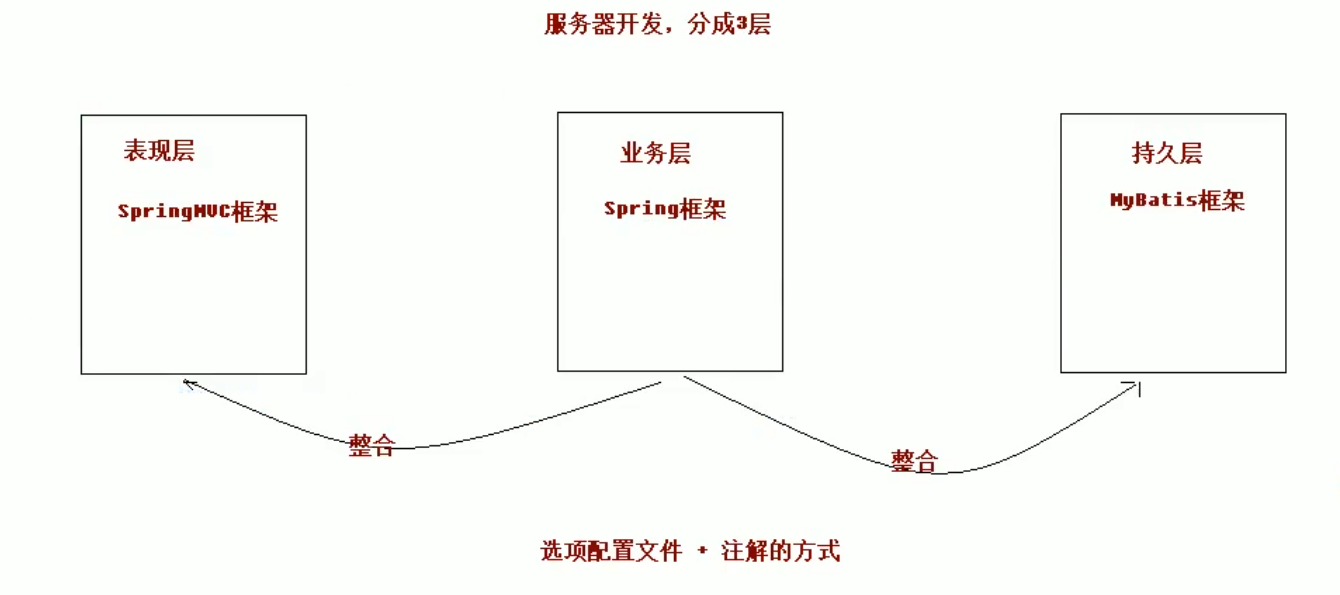


# 第三天

## 一： ssm整合概述

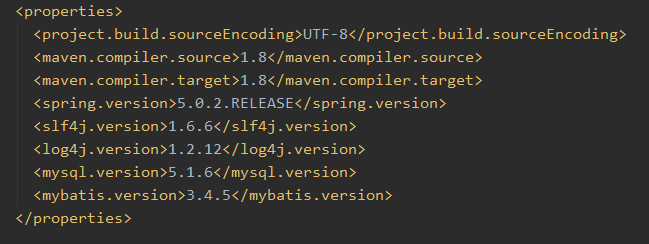
我们采用的是xml和注解结合的方式进行整合，实际开发中看情况使用，那个简单用哪个

整合的思路是用spring来整合另外两个框架，先单独把springmvc和mybatis配置好且测试运行通过，再用spring将其整合进来，否则容易懵逼。

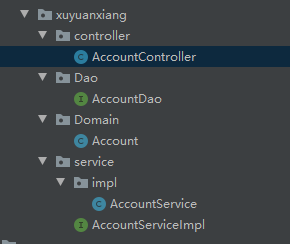


## 二： ssm整合之环境搭建

创建好项目后我们需要导入maven坐标，这里先控制好版本！然后导入一大堆maven坐标。太多了不一一列举大体上就是：log4j，junit，test，spring，webmvc，mybatis等等



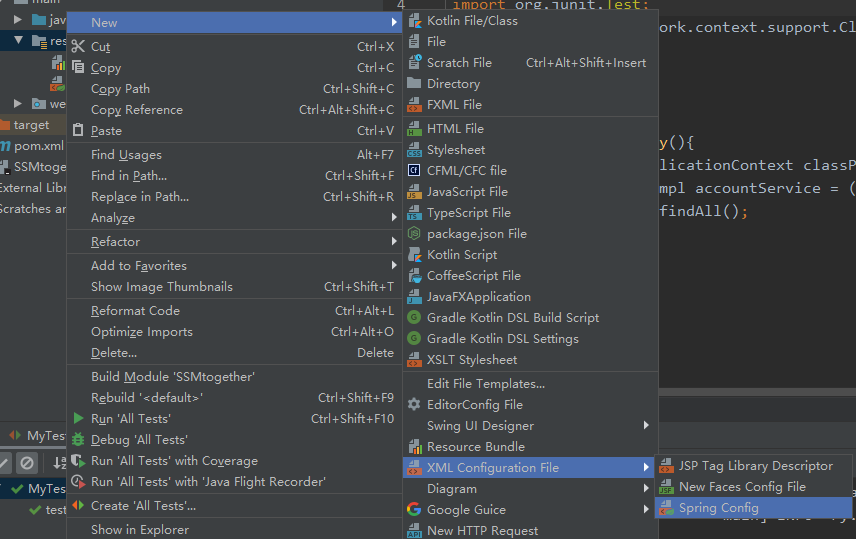
整合首先要是项目结构完整：创建三层架构该有包以及类，再dao层我们只需要接口就可以了，因为后期mybatis会帮我们实现，service层就必须我们自己写实现类了！



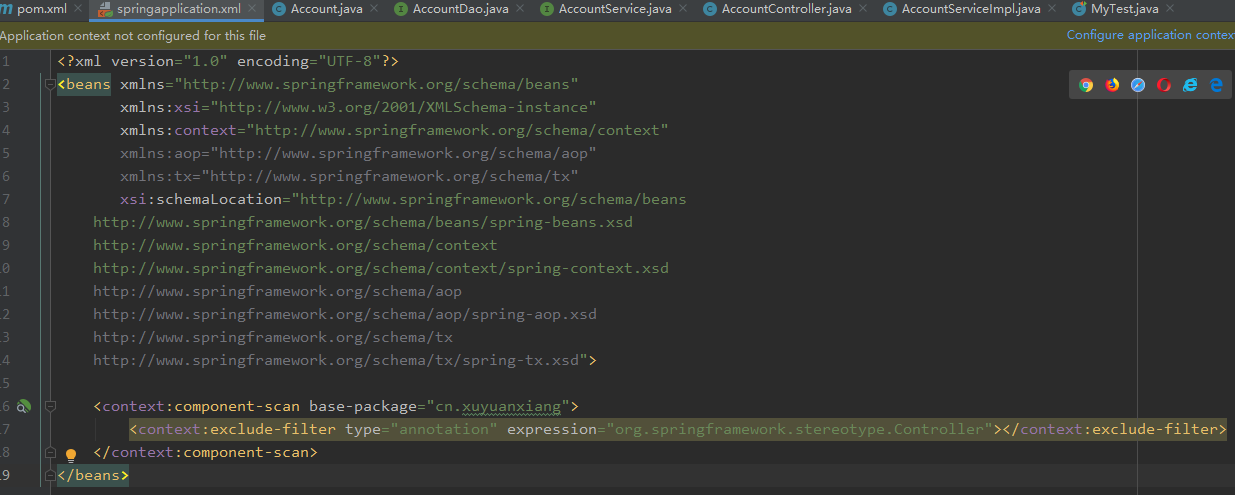
## 三： ssm整合之编写spring框架

编写spring的配置文件：

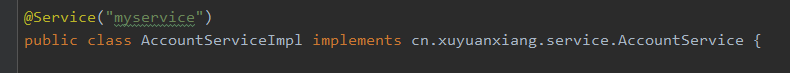
在resources文件目录下面创建spring config的xml文件



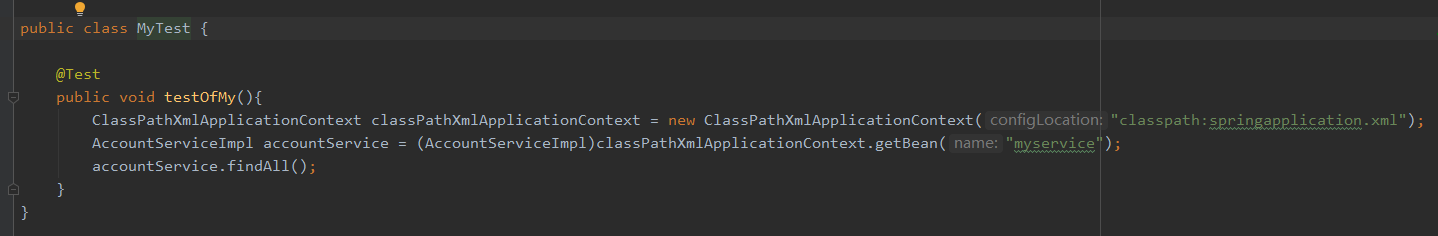
导入约束文件，然后开启注解扫描，在其中配置要扫描的类包路径，由于我们只需要他扫描Dao层和service层，所以这里需要配置排除Controller的注解配置！



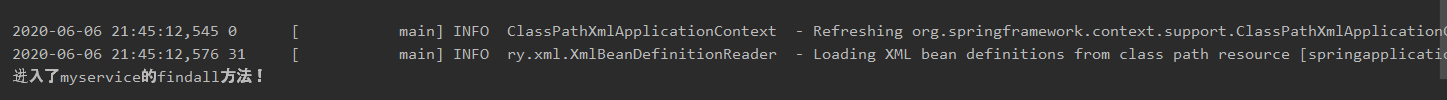
Service层的实现类头上需要加上@Service注解，表示将这个类交给ioc进行处理。（就是说会自动配置他，将他加入到容器中，而且键名就是我们写的“myservice“）。



然后进行spring的测试，看看是否配置正确。这里一定要测试，否者后面出了问题就很麻烦！其中的方法就是主动去解析我们spring的配置文件（后面我们会配置让tomcat运行的时候解析这个文件（其本质其实是监听器实现的），这里手动解析是因为我们仅仅是测试，并没有启动tomcat）。解析完城后，ioc容器中就会添加上我们的@service标注的类。通过id值进行获取，便可以得到对应的实现类，完成其中的方法调用!



这里打印的输出产生变化是因为我们采用log4j日志，重写了输出的格式！

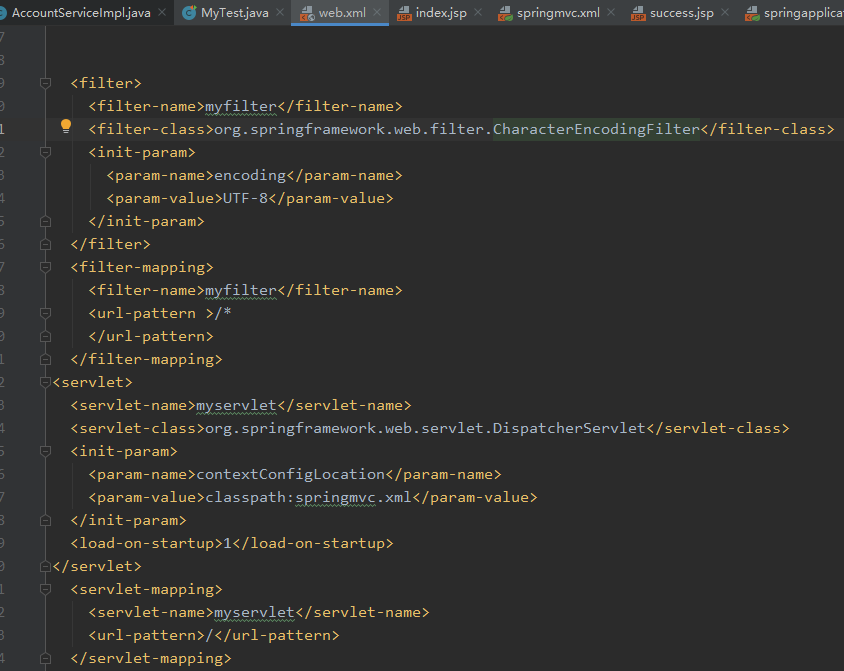


## 四： ssm整合之编写springmvc框架

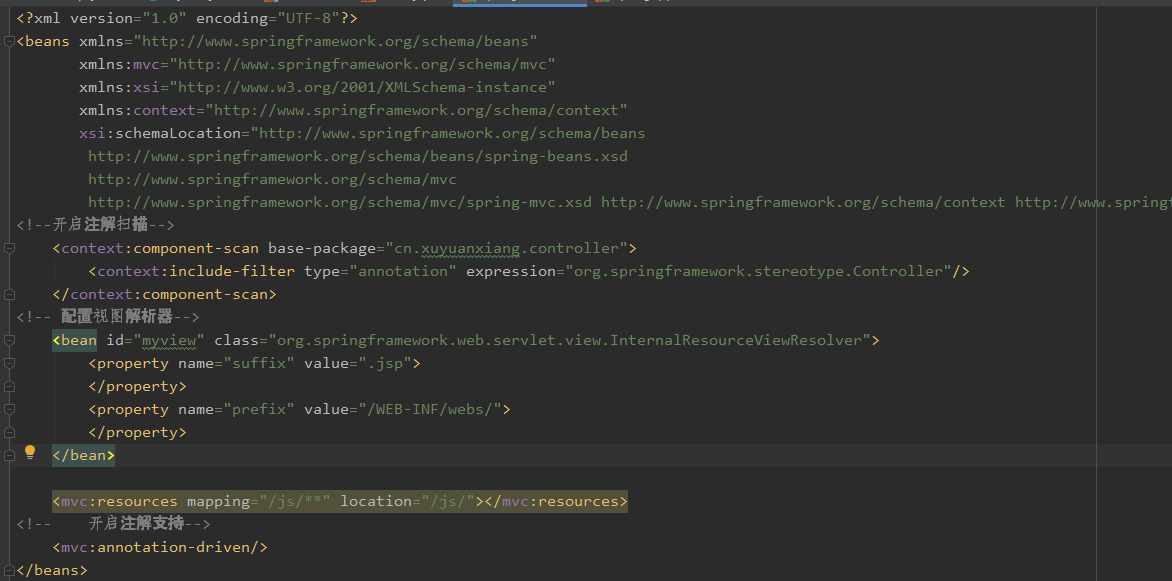
要想整合springmvc框架，首先要编写一个能成功运行的mvc结构！

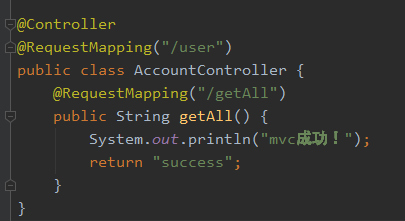
这里的写法完全和以前相同！

首先在web.xml文件里配置好前端控制器（指定我们的springmvc配置文件）和过滤器（解决编码问题）



然后再在我们自己写的springmvc配置文件中进行视图解析器配置，开启注解扫描(spring的配置文件中也有这一步！只不过这里恰恰相反，只允许扫描controller注解)，最后开启注解支持。



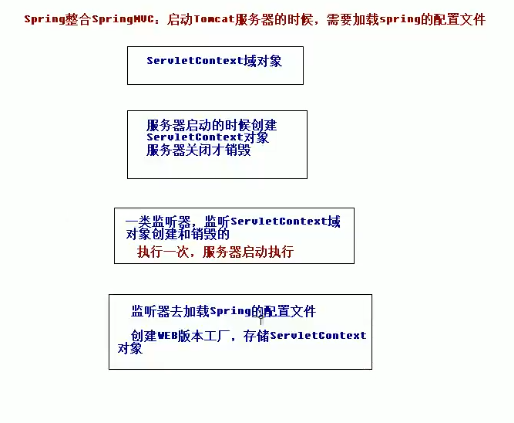


## 五： ssm整合之spring整合springmvc框架

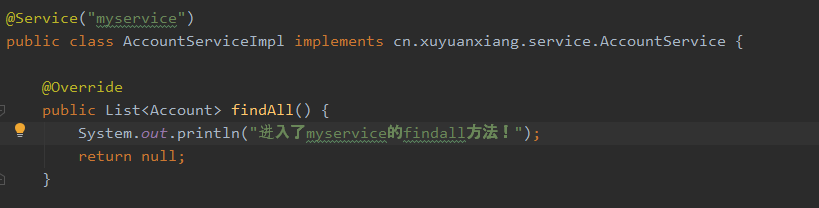
其实只要我们的springmvc层（@controller）能调用spring层（@service）的方法，就是整合成功了。那既然想要获取其中的方法，必然需要service注解类的实例对象，然而我们并没有加载spring的配置文件，自然不会去扫描@service注解，也就不会将其类加入到ioc容器中，也就不能够调用这个类，更不能使用其中的方法！

所以我们需要实现spring.xml的解析，来扫描@service注解，将其添加到组件中！

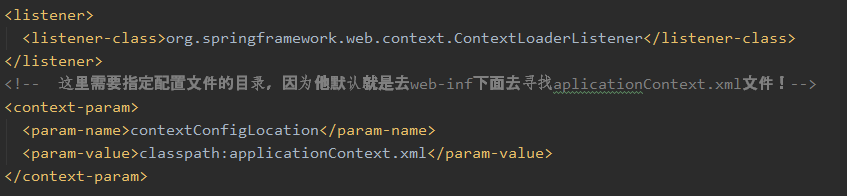
而这个事情其实tomcat已经帮我们完成了，我们需要做的就是在配置的时候添加一个监听器（实际上一旦tomcat一旦监听到Servletcontext（这个是我们项目的最大域对象，一旦项目启动就会自动创建！）对象的创建，就会自己去解析applicationContext.xml的配置文件）。



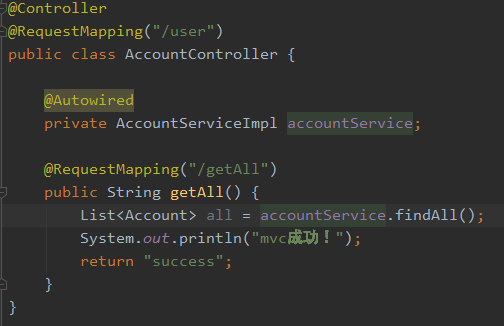
这里首先修改@service方法添加一行打印方便观察！



在web.xml文件中配置监听器（至此，servlet（也就是服务区）的三大组件都已经齐全），单数监听器配置好，会自动就解析指定路径的指定文件，所以这里需要修改他的指定路径：



然后再我们的controller层可以采用@autowired注解进行注入了。也就可以调用其中的方法。

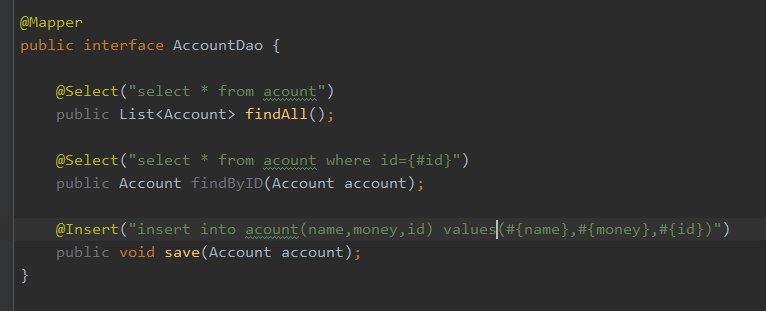


结果显示，springmvc和spring已经整合在一起了。

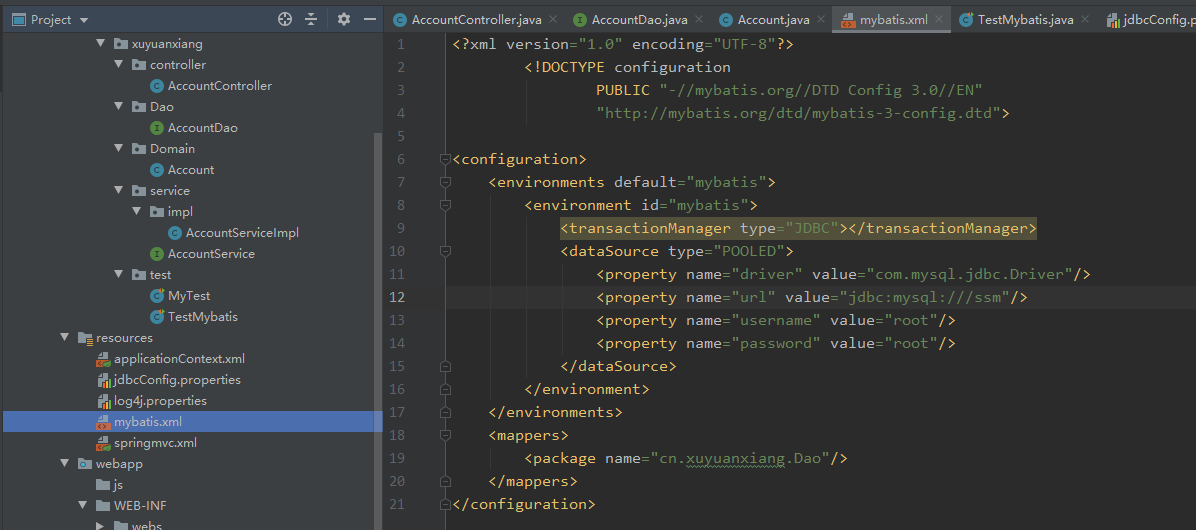


## 六： ssm整合之mabatis框架编写

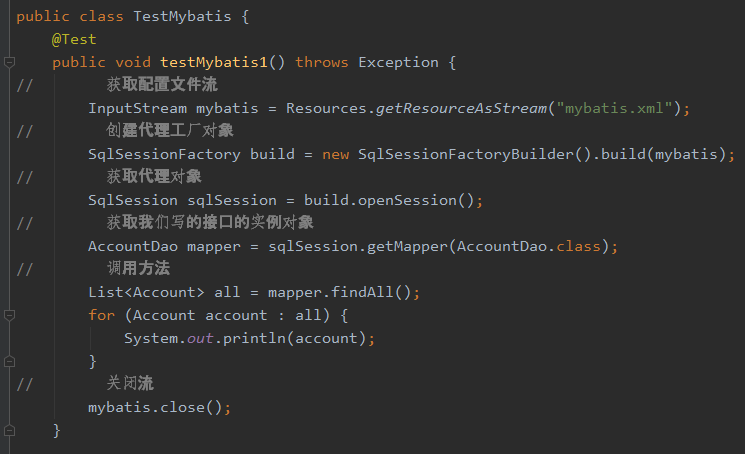
首先配置好我们的Dao层，在方法头上加好注解！



完成mybatis.xml文件的配置！当然在<mapper>中可以采用class属性指定注解方式配置，也可以resources属性指定xml配置方式。这里采用了name属性，会自动识别配置方式！



编写一个测试类看一下我们的mybatis是否能执行成功！



返回数据正确!

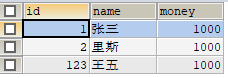


测试上传参数功能：

需要注意的是我们的getMapper获取对象的实例时，会对方法进行增强，除了查询语句，所有的其他语句都会增加事务类型，所以要想实现添加成功，必须先提交事务才可以！



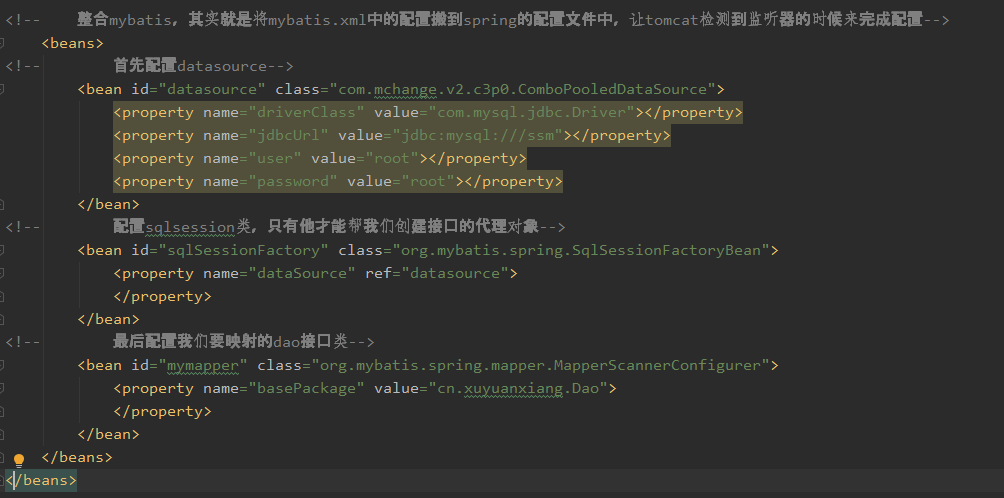
结果显示也正确~！



## 八： ssm整合之mabatis框架

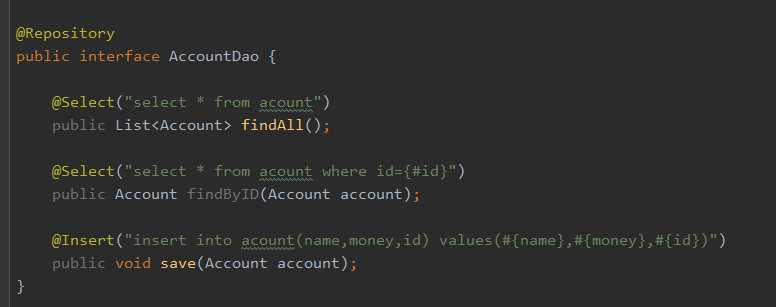
整合的意思也是只要能够调用成功就可以的

我们在spring的配置文件中把mybatis的配置添加进来就可以了，因为前面配置的监听器，一旦监听到ServletContext的创建就会自动加载该配置文件！

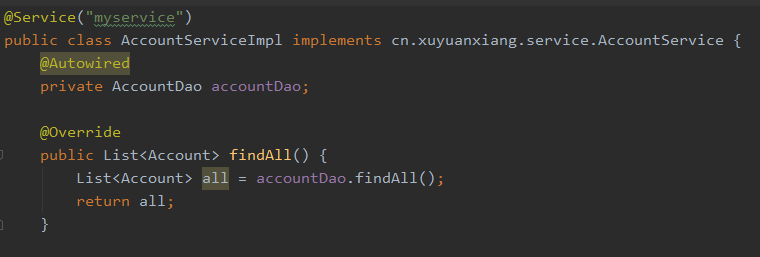


然后在dao接口的头顶添加@Repository注解，当然也可以采用版本更新的@mapper注解(mapper注解更方便，她不用配置这么多bean！)

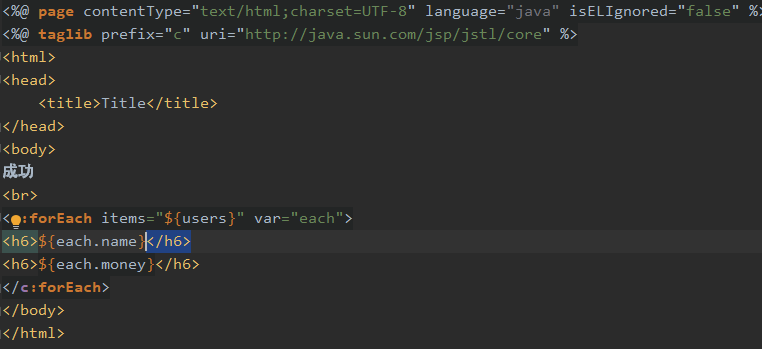




最后在service层采用自动注入的方式将我们的dao对象注入！然后调用

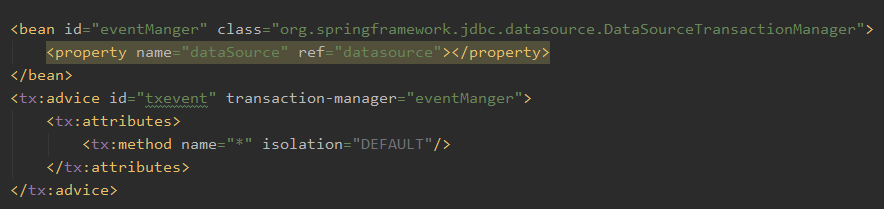


更改前端页面，观察返回数据是否正确！这里又采用了jsp的技术，做简单回顾！

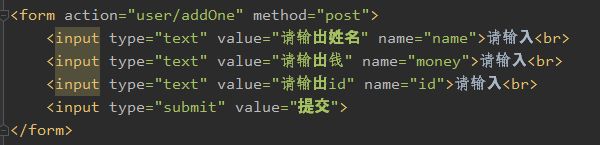


## 九： ssm整合之mabatis框架事务管理

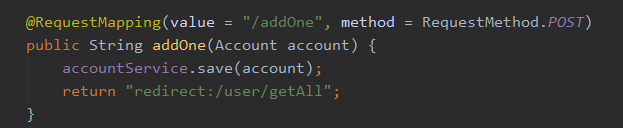
首先在spring配置文件中添加事务管理组件（本质上就是类javabean），然后指定事务的隔离等级，这里指定所有都是默认等级。



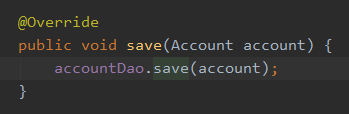
修改前端页面添加一个提交的表单



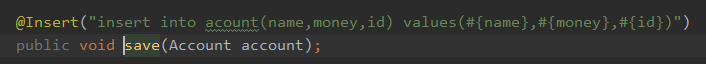
表单提交调用controller层



表现层调用了业务层



业务层又进入了持久层

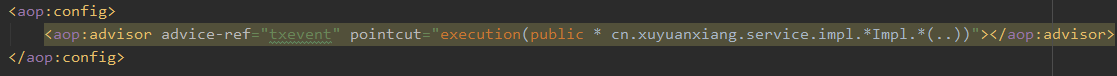


最后将数据返回给我们的页面显示！



最后也可以为我们的方法做增强代理：案例中没有使用到，模板如下：

这里表示对所有的service层的impl结尾的类的所有方法进行了事务代理增强！



更详细的增强如下：

