今日任务完成16

SpringBoot框架中的DAO层、Entity层、Service层、Controller层

一般的项目模块中都有DAO、Entity、Service、Controller层

Entity层：实体层数据库在项目中的类

Entity层是实体层，也就是所谓的model,也称pojo层,是数据库在项目中的类，该文件包含实体类的属性和对应属性的set和get方法；

DAO层：持久层主要与数据库进行交互

DAO层=mapper层。现在用Mybatis逆向工程生成的mapper层、其实就是dao层。DAO层会调用Entity层，DAO中会定义实际使用到的方法，比如增删改查。DAO层的数据源和数据库的连接的参数都是在配置文件中进行配置的，配置文件一般在同层的XML文件夹中。数据持久化操作就是指：把数据放到持久化的介质中，同时提供增删改查操作。

Service层：业务层 控制业务

Service层主要负责业务模块的逻辑应用设计。先设计放接口的类，再创建实现的类，然后在配置文件中进行配置其实现的关联。service层调用dao 层接口，接收dao层返回的数据，完成项目的基本功能设计。

Controller层：控制层 控制业务逻辑

Controller层负责具体的业务模块流程的控制，controller层负责前后端交互，接受前端请求，调用service层返回的数据，最后返回具体的页面和数据到客户端。

Controller层像是一个服务员，他把客人(前端)点的菜(数据、请求的类型等)进行汇总什么口味、咸淡、量的多少、交给厨师(Service层),厨师长则告诉粘板厨师（DAO 1）、汤料房(Dao 2)、配菜厨师(Dao3)等(统称Dao层)我需要什么样的半成品，副厨师(Dao层)就负责完成厨师长(Service)交代的任务。

在具体的项目中，其流程为：

Controller-- >service接口-- >serviceImp-- >dao接口-- >daoImp-- >mapper-- >db

Controller层调用Service层的方法、Service层的调用Dao层中方法，其中调用的参数是使用Entity层进行传递的。

VO:view Object,视图层，其作用是将指定页面的展示数据封装起来

DTO:Data Transfer Object,数据传输对象

DO:Domain Objcet ,领域对象

PO:Persistent Object,持久化对象

用户发出请求( 填写表单)，表单的数据被展示层匹配为VO

展示层把VO转化为服务层对应方法所要求的DTO，提交给服务层

服务层先将DTO的数据构造(或重建)一个DO，调用DO的业务方法完成具体业务

服务层再将DO转换为持久层对应的PO，调用持久层的持续化方法，把PO传递持久化方法，完成持久化操作

PO：持久对象(Persistent Object)就是在Object/Relation Mapping框架中的Entity,po的每个属性基本上都对应数据库表里面的某个字段。完全是一个符合Java Bean规范的纯Java对象，没增加别的属性和方法。持久对象是由insert数据库创建，由数据库delete删除的。基本上持久对象生命周期和数据库密切相关

VO：表现层对象(View Object)，主要对应展示界面显示的数据对象，用一个OV对象来封装整个界面展示所需要的对象数据，数据脱敏(对某些敏感信息通过脱敏规则进行数据的变形，实现敏感隐私数据的可靠保护),去掉用户隐私数据。

BO：业务对象层(Business Object)，封装业务逻辑的java对象，通过调用DAO方法，结合PO、VO进行业务操作。例如：

比如一个简历，有教育经历、工作经历、社会关系等等。

我们可以把教育经历对应一个PO，工作经历对应一个PO，社会关系对应一个PO。

建立一个对应简历的BO对象处理简历，每个BO包含这些PO。

这样处理业务逻辑时，我们就可以针对BO处理。

DTO：数据传输对象(Data Transfer Object),是一种设计模式之间传输数据的软件应用系统。数据传输目标往往是数据访问对象从数据库中检索数据。数据传输对象与数据交互对象或数据访问对象之间的差异是一个以不具有任何行为除了存储和检索的数据(访问和存取器)，当我们需要一个对象10个字段的内容，但这个对象总共有20个字段，我们不需要把整个PO对象全部字段传输到客户端，而是可以用DTO重新封装，传递到客户端。此时，如果这个对象用来对应界面的表现，就叫VO

项目文件：

处理前台发送请求

controller.java

service.java

定义接口（业务逻辑）

对接口函数进行实现

serviceImp.java

Mapper接口，方法名与Mapper.xml(UserMapper.xml)中定义的statement的id相同

mapper.java

mapper.xml

写sql语句查询数据库

mall-admin

1. @Api: 用在请求的类上，表示对类的说明

tags="说明该类的作用，可以在前台界面上看到的注解"

value="该参数无意义，在UI界面上看不到，不需要配置"

2.@RequestMapping注解的作用是什么？

@RequestMapping注解是web应用程序中最常被用到的注解之一，作用就是映射URL路径，将http的请求地址映射到控制器(controller)类的处理方法上。

@RequestMapping注解可以定义在控制器类上，也可以定义在类里面的方法上。

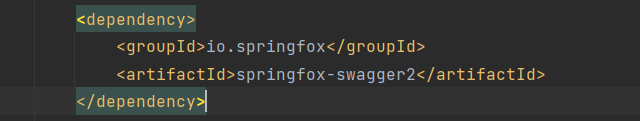
定义类上：将http请求映射到该控制器上，规定初步的请求映射地址，相当于请求地址的父路径。

定义类方法上：进一步指定到控制器处理方法的映射关系，如果在控制器类上没有定义@RequestMapping注解，则直接将请求地址映射到处理方法上。

[3.@ApiOperation](mailto:3.@ApiOperation)作用：用来构建Api文档

1. 引入依赖

在pom文件中引入依赖



1. 用法

①使用与在方法上，表示一个http请求操作

②@ApiOperation(value = “接口说明”，httpMethod = “请求接口方式”,response =“接口返回参数类型”,note = “接口发布说明”,其他参数可参考源码)

③具体例子





3.@Bean的作用

Spring的@Bean注解用于告诉方法，产生一个Bean对象，然后这个Bean对象交给Spring管理。 产生这个Bean对象的方法Spring只会调用一次，随后这个Spring将会将这个Bean对象放在自己的IOC容器中。@Bean明确地指示了一种方法，什么方法呢？产生一个bean的方法，并且交给Spring容器管理；从这我们就明白了为啥@Bean是放在方法的注释上了，因为它很明确地告诉被注释的方法，你给我产生一个Bean，然后交给Spring容器，剩下的你就别管了。记住，@Bean就放在方法上，就是让方法去产生一个Bean，然后交给Spring容器。

@Configuration

在配置类上打个一个 @Configuration 注解，表示声明该类为 Spring 的配置类。在创建一个方法，方法返回对象即我们要创建的对象 Person ， 返回该对象的实例。在方法上打上注解 @Bean即表示声明该方法返回的实例是受 Spring 管理的 Bean





@ResponseBody注解作用

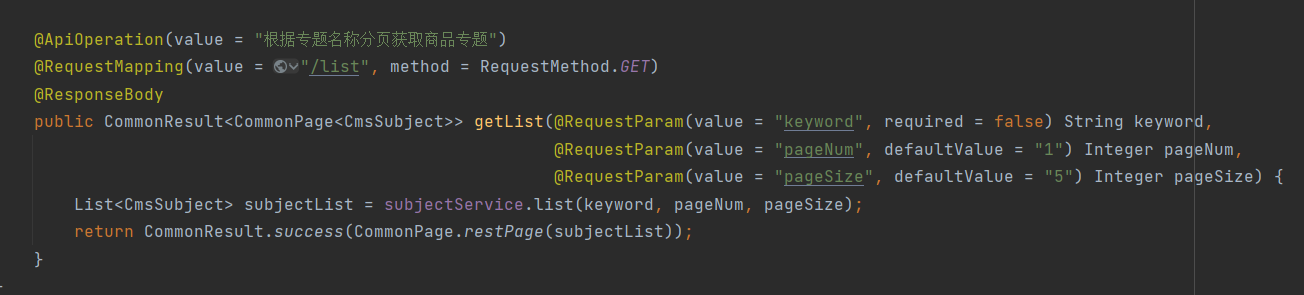
将controller的方法返回对象，通过适当的转换器，转换为指定的格式之后，写入到response对象的body区(响应体中),通用来返回JSON数据或者是XML

数据需要注意，在使用注解之后不会再走视图处理器，而是直接将数据写入到输入流中，它的效果等同于response对象输出指定格式数据

注意:要通过@ResponseBody注解将返回的json字符串放入响应体中，然后在前台js才能拿到json字符串进行解析，如果不加，响应体中就没有放入json字符串，前台拿不到数据的。

@RequestParam

用于传递参数，将请求参数区数据映射到功能处理方法的参数上





@Autowired作用

@Autowired是一种注解，可以对成员变量、方法和构造函数进行标注，来完成自动装配的工作，@Autowired标注可以放在成员变量上，也可以放在成员变量的set方法上，也可以放在任意方法上表示，自动执行当前方法，如果方法有参数，会在IOC容器中自动寻找同类型参数为其传值。