1. 系统设计
2. 开发模式

采用前后端分离的开发模式，前端与后端并行开发。

1. 技术选型

后端：springboot +springcloud + mybatis plus + mysql

前端：nodejs + NUXT + elementUI + vue

1. 开发模型

传统项目中使用的都是瀑布模型，这是一套完整的体系，从项目需求分析，到概要设计，到详细设计，到开发，测试，上线。在出现问题时，又要从头走一遍流程，非常地耗时。

敏捷开发，客户与开发者即时交流，有问题就沟通，直接修改开发，测试，再上线。

1. 微服务模块划分

根据功能划分。

1. 数据库表结构分析

采用分库分表设计，每个微服务模块为一个独立的数据库。分表主要是为了解决性能问题。

1. API文档

在开发时，前后端时同步进行的，前端和后端的开发都是根据API文档来开发的。前后端开发接口文档是采用Swagger语言进行编写的，并与Nginx进行了整合。Nginx启动后，在地址栏输入<http://localhost:801>即可访问API文档

API文档中定义了各种方法的参数，返回值，满足我们开发的各种需求。

1. RESTful
2. 概述

RESTful是目前最流行的一种互联网软件架构，是所有Web应用都应该遵守的架构设计指导原则，一共有7个HTTP方法：GET，POST，PUT，DELETE，PATCH，HEAD，OPTIONS。

1. 接口规范

项目使用GET，POST，PUT，DELET四种方法。

幂等性：不论你请求多少次，资源的状态是一样的。

1. GET

安全且幂等性，所谓安全指的是不会对数据产生影响。

1. POST

不安全且不幂等性。所谓的不幂等性指的是每次请求得到的都是不同的结果。

1. PUT

不安全但幂等性。所谓幂等性指的是一次请求和多次请求结果是一样的。

1. DELETE

不安全但幂等性。

1. 如何设计接口的幂等性
2. insert
3. 设置一个全局的id
4. 先查询一下，如果存在则更新，如果不存在则插入
5. 在Redis中创建一个去重表，如果有重复的则更新，否则插入
6. update
7. 多版本控制，收到更新，就将版本更新一次，如果版本不同，则不能修改。
8. 数据库乐观锁/悲观锁，使用的还是多版本控制思想
9. 项目开发准备
10. 开发环境

虚拟机Centos7

容器docker

JDK1.8

数据库mysql5.7

开发工具IDEA

项目构建工具maven

虚拟机账户root

虚拟机密码itcast

1. 创建docker容器
2. 创建mysql容器
3. 安装Navicat Premium
4. 创建article数据库
5. 安装过postman

用于测试接口的实现，能够发送任何类型的HTTP请求，附带任意数量的参数。

1. 项目搭建
2. 结构

父工程是所有服务的parent，common工程是公共的服务，其他微服务需要依赖common工程。

1. 搭建parent工程
2. 配置pom.xml

parent工程是所有工程的父工程，可以在该工程的pom.xml中配置最基本的依赖。

添加SpringBoot parent 启动依赖，web启动依赖，test启动依赖。设置所需要使用到的一些依赖的版本。

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.tensquare</groupId>  
 <artifactId>tensquare\_parent</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <parent>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <groupId>org.springframework.boot</groupId>  
 <version>2.1.4.RELEASE</version>  
 </parent>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
 <java.version>1.8</java.version>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 <mybatisplus-spring-boot-starter.version>1.0.5</mybatisplus-spring-boot-starter.version>  
 <mybatisplus.version>2.2.0</mybatisplus.version>  
 <fastjson.version>1.2.39</fastjson.version>  
 <gson.version>2.8.0</gson.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 </dependency>  
 </dependencies>  
  
</project>

1. 搭建公共子模块
2. 返回对象Result类

该模块为其他服务提供公共类，同样要继承parent工程。可以从API文档中看出，其他服务返回的内容大都是code, flag, message, data，因此可以在公共工程common中设置一个POJO，用于其他服务的返回对象之中。flag表示是否成功，code表示返回码,message表示返回信息，data表示返回数据。

注意，构造方法有三种，一个是无参的，一个是3个参数，没有data参数的，因为有的返回对象中没有data，一个是4个参数的。

package entity;  
  
public class Result {  
  
 private Boolean flag;  
 private Integer code;  
 private String message;  
 private Object data;  
  
 public Result() {  
 }  
  
 public Result(Boolean flag, Integer code, String message) {  
 this.flag = flag;  
 this.code = code;  
 this.message = message;  
 }  
  
 public Result(Boolean flag, Integer code, String message, Object data) {  
 this.flag = flag;  
 this.code = code;  
 this.message = message;  
 this.data = data;  
 }  
  
 public Boolean getFlag() {  
 return flag;  
 }  
  
 public void setFlag(Boolean flag) {  
 this.flag = flag;  
 }  
  
 public Integer getCode() {  
 return code;  
 }  
  
 public void setCode(Integer code) {  
 this.code = code;  
 }  
  
 public String getMessage() {  
 return message;  
 }  
  
 public void setMessage(String message) {  
 this.message = message;  
 }  
  
 public Object getData() {  
 return data;  
 }  
  
 public void setData(Object data) {  
 this.data = data;  
 }  
  
 @Override  
 public String toString() {  
 return "Result{" +  
 "flag=" + flag +  
 ", code=" + code +  
 ", message='" + message + '\'' +  
 ", data=" + data +  
 '}';  
 }  
}

1. 返回分页类

返回的data数据也需要用一个类来封装，在分页时，有total总页数，rows当前页的数据。rows中的数据是我无法预知的，可以用List来存储，其中的类型用泛型。

package entity;  
  
import java.util.List;  
  
public class PageResult<T> {  
  
 private Long total;  
  
 private List<T> rows;  
  
 public PageResult() {  
 }  
  
 public PageResult(Long total, List<T> rows) {  
 this.total = total;  
 this.rows = rows;  
 }  
  
 public Long getTotal() {  
 return total;  
 }  
  
 public void setTotal(Long total) {  
 this.total = total;  
 }  
  
 public List<T> getRows() {  
 return rows;  
 }  
  
 public void setRows(List<T> rows) {  
 this.rows = rows;  
 }  
  
 @Override  
 public String toString() {  
 return "PageResult{" +  
 "total=" + total +  
 ", rows=" + rows +  
 '}';  
 }  
}

1. 维护状态码表的类

由于API文档中规定了前后端约定好的的返回码列表，因此我们可以设置一个专门用于维护前后端之间返回状态码的类，当状态码需要修改时，直接在类中设置即可。每个状态码作为一个类中的属性存在。注意，这些返回码并不是HTTP的返回值状态码，而是这个项目中自己设置的前后端之间的返回码约定。

1. 分布式ID生成器

不能使用数据库本身的自增功能来产生主键值，因为生产环境是分片部署的，如果使用自增的主键，就会出现ID重复的情况，要使用snowflake（雪花）算法来生成唯一的主键值。

雪花算法就是一个64位的二进制，前41位为时间戳，中间10位为工作机器id，后面12位为序列号，就是这三个数字序列组成了一个主键值。10位的工作机器id可以支持1024台机器，12位的序列号支持1毫秒生成4096个自增序列id。

直接将文件夹 \工具类\IdWorker.java导入 tensquare\_common工程的util包下。

1. 搭建文章子模块article
2. 添加依赖

添加mysql驱动，以及common工程依赖。

<dependencies>  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.tensquare</groupId>  
 <artifactId>tensquare\_common</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
</dependencies>

1. 添加配置文件

文章项目的默认配置文件application.yml中配置mysql连接的数据池，服务名称，服务端口号。mysql的配置是源于之前在docker中生成的mysql容器，服务端口号在API文档已经定义好了9004.

spring:  
 application:  
 name: tensquare-article  
 datasource:  
 driver-class-name: com.mysql.jdbc.Driver  
 url: jdbc:mysql：//192.168.112.128:3306/tensquare\_article?characterEncoding=utf-8  
 username: root  
 password: root  
server:  
 port: 9004

1. 创建引导类

添加SpringBoot的引导类，查看工程是否能够正常启动。

1. 整合article工程和mybatis-plus
2. mybatis plus概述

对mybatis的二次封装和扩展，仅仅依赖Mybatis以及Mybatis-Spring，自动热加载，减少重启Web服务器时间，避免SQL注入。

1. 在article工程的pom.xml中引入相关依赖

引入mybatisplus的启动依赖，以及mybatisplus。

<!-- mybatis-pllus-begin-->  
 <dependency>  
 <groupId>com.baomidou</groupId>  
 <artifactId>mybatisplus-spring-boot-starter</artifactId>  
 <version>${mybatisplus-spring-boot-starter.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>com.baomidou</groupId>  
 <artifactId>mybatis-plus</artifactId>  
 <version>${mybatisplus.version}</version>  
 </dependency>  
<!-- mybatis-plus-end-->

1. 在application.yml中添加配置

添加mybatis-plus的配置。

# Mybatis-Plus 配置

mybatis-plus:

# mapper-locations: classpath:/mapper/\*Mapper.xml

#实体扫描，多个package用逗号或者分号分隔

typeAliasesPackage: com.tensquare.article.pojo

global-config:

id-type: 1 #0:数据库ID自增 1:用户输入id

db-column-underline: false

refresh-mapper: true

configuration:

map-underscore-to-camel-case: true

cache-enabled: true #配置的缓存的全局开关

lazyLoadingEnabled: true #延时加载的开关

multipleResultSetsEnabled: true #开启延时加载，否则按需加载属性

log-impl: org.apache.ibatis.logging.stdout.StdOutImpl #打印sql语句,调试用

1. 修改引导类，添加Mapper包扫描

@MapperScan("com.tensquare.article.dao")

1. 实现查询所有文章
2. 创建Article实体类

首先需要创建一个Article实体类，包含文章的所有信息，在com.tensquare.article包下面创建pojo包，并创建Article实体类。

注解@TableName表示该类对应的数据库中表的名称，注解@TableId表示表的id的类型是从输入中得到的。如下所示：

package com.tensquare.article.pojo;  
  
import com.baomidou.mybatisplus.annotations.TableId;  
import com.baomidou.mybatisplus.annotations.TableName;  
import com.baomidou.mybatisplus.enums.IdType;  
  
import java.util.Date;  
  
@TableName("tb\_article")  
public class Article implements Serializable{  
  
 @TableId(type= IdType.*INPUT*)  
 private String id; //ID  
  
 private String columnid; //专栏ID  
 private String userid; //用户ID  
 private String title; //标题  
 private String content; //文章正文  
 private String image; //文章封面  
 private Date createtime; //发表日期  
 private Date updatetime; //修改日期  
 private String ispublic; //是否公开  
 private String istop; //是否置顶  
 private Integer visits; //浏览量  
 private Integer thumbup; //点赞数  
 private Integer comment; //评论数  
 private String state; //审核状态  
 private String channelid; //所属频道  
 private String url; //URL  
 private String type; //类型  
  
 public String getId() {  
 return id;  
 }  
  
 public void setId(String id) {  
 this.id = id;  
 }  
  
 public String getColumnid() {  
 return columnid;  
 }  
  
 public void setColumnid(String columnid) {  
 this.columnid = columnid;  
 }  
  
 public String getUserid() {  
 return userid;  
 }  
  
 public void setUserid(String userid) {  
 this.userid = userid;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public void setTitle(String title) {  
 this.title = title;  
 }  
  
 public String getContent() {  
 return content;  
 }  
  
 public void setContent(String content) {  
 this.content = content;  
 }  
  
 public String getImage() {  
 return image;  
 }  
  
 public void setImage(String image) {  
 this.image = image;  
 }  
  
 public Date getCreatetime() {  
 return createtime;  
 }  
  
 public void setCreatetime(Date createtime) {  
 this.createtime = createtime;  
 }  
  
 public Date getUpdatetime() {  
 return updatetime;  
 }  
  
 public void setUpdatetime(Date updatetime) {  
 this.updatetime = updatetime;  
 }  
  
 public String getIspublic() {  
 return ispublic;  
 }  
  
 public void setIspublic(String ispublic) {  
 this.ispublic = ispublic;  
 }  
  
 public String getIstop() {  
 return istop;  
 }  
  
 public void setIstop(String istop) {  
 this.istop = istop;  
 }  
  
 public Integer getVisits() {  
 return visits;  
 }  
  
 public void setVisits(Integer visits) {  
 this.visits = visits;  
 }  
  
 public Integer getThumbup() {  
 return thumbup;  
 }  
  
 public void setThumbup(Integer thumbup) {  
 this.thumbup = thumbup;  
 }  
  
 public Integer getComment() {  
 return comment;  
 }  
  
 public void setComment(Integer comment) {  
 this.comment = comment;  
 }  
  
 public String getState() {  
 return state;  
 }  
  
 public void setState(String state) {  
 this.state = state;  
 }  
  
 public String getChannelid() {  
 return channelid;  
 }  
  
 public void setChannelid(String channelid) {  
 this.channelid = channelid;  
 }  
  
 public String getUrl() {  
 return url;  
 }  
  
 public void setUrl(String url) {  
 this.url = url;  
 }  
  
 public String getType() {  
 return type;  
 }  
  
 public void setType(String type) {  
 this.type = type;  
 }  
  
 @Override  
 public String toString() {  
 return "Article{" +  
 "id='" + id + '\'' +  
 ", columnid='" + columnid + '\'' +  
 ", userid='" + userid + '\'' +  
 ", title='" + title + '\'' +  
 ", content='" + content + '\'' +  
 ", image='" + image + '\'' +  
 ", createtime=" + createtime +  
 ", updatetime=" + updatetime +  
 ", ispublic='" + ispublic + '\'' +  
 ", istop='" + istop + '\'' +  
 ", visits=" + visits +  
 ", thumbup=" + thumbup +  
 ", comment=" + comment +  
 ", state='" + state + '\'' +  
 ", channelid='" + channelid + '\'' +  
 ", url='" + url + '\'' +  
 ", type='" + type + '\'' +  
 '}';  
 }  
}

1. 创建文章的Dao层接口

使用MabatisPlus的内置操作，需要继承BaseMapper类，同时需要指定操作的是哪个类的对象，如下所示：

package com.tensquare.article.dao;  
  
import com.baomidou.mybatisplus.mapper.BaseMapper;  
import com.tensquare.article.pojo.Article;  
  
public interface ArticleDao extends BaseMapper<Article> {  
}

1. 创建Service层的类

Service层的类需要调用Dao层的接口，并且添加注解@Service，如下所示：

package com.tensquare.article.service;  
  
import com.tensquare.article.dao.ArticleDao;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
@Service  
public class ArticleService {  
  
 @Autowired  
 private ArticleDao articleDao;  
   
}

1. 创建Controller层的类

注解@RestController表示注解@Controller和注解@ResponseBody，再通过注解@RequestMapping添加访问地址，如下所示：

package com.tensquare.article.controller;  
  
import com.tensquare.article.service.ArticleService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
@RequestMapping("article")  
public class ArticleController {  
  
 @Autowired  
 private ArticleService articleService;  
}

1. 查询文章信息

后端的开发需要根据API文档进行，查看文章模块，可以看到查询的接口有两个，一个是查询所有文章，一个是根据ID查询文章。根据API文档规定的访问地址，返回的是我们在公共模块中定义的Result类。这里调用的是ArticleService中的查询方法，在后面进行编写。

package com.tensquare.article.controller;  
  
import com.tensquare.article.pojo.Article;  
import com.tensquare.article.service.ArticleService;  
import entity.Result;  
import entity.StatusCode;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/article")  
public class ArticleController {  
  
 @Autowired  
 private ArticleService articleService;  
  
 //GET /article 文章全部列表  
  
 public Result findAll(){  
 List<Article> list = articleService.findAll();  
 return new Result(true, StatusCode.*OK*, "查询成功", list);  
 }  
}

1. 实现ArticleService的findAll方法

ArticleService类的findAll方法调用ArticleDao接口的方法，ArticleDao接口继承了BaseMapper的内置方法，可以直接调用,如selectList()方法，需要使用什么方法通过返回值判断。

package com.tensquare.article.service;  
  
import com.tensquare.article.dao.ArticleDao;  
import com.tensquare.article.pojo.Article;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class ArticleService {  
  
 @Autowired  
 private ArticleDao articleDao;  
  
 public List<Article> findAll(){  
 return articleDao.selectList(null);  
  
 }  
  
}

1. 测试查询所有文章接口

测试时出现BUG，发现是Mysql的连接URL出现错误，URL格式需要改正。通过POSTMAN进行GET请求的测试，页面中显示了所有文章信息。

1. 总结

注意！！！Rest风格需要在每个Controller方法的注解@RequestMapping中添加HTTP的请求类型。

6. 实现根据Id号查询文章

（1） 编写Controller方法

在Controller类中编写查询方法findById，通过从访问地址中的id进行查询。

//GET /article/{aticleId} 根据ID查询文章  
@RequestMapping(value="{articleId}", method=RequestMethod.*GET*)  
public Result findById(@PathVariable String articleId){  
 Article article = articleService.findById(articleId);  
 return new Result(true, StatusCode.*OK*, "查询成功", article);  
}

（2）编写Service方法findById

public Article findById(String articleId) {  
 return articleDao.selectById(articleId);  
}

1. 测试

使用PostMan发送GET请求。

1. 增加文章功能

增加文章是使用Post请求的，将文章数据传入、

1. 编写Controller方法save
2. //POST /article 增加文章  
   @RequestMapping(method=RequestMethod.*POST*)  
   public Result save(@RequestBody Article article){  
    articleService.save(article);  
    return new Result(true, StatusCode.*OK*, "新增成功");  
   }

注解@RequestBody用于将Json格式的数据转换为Article对象。

（2）编写Service方法

要添加文章，就需要为文章添加ID，ID是使用雪花算法生成的，需要在引导类中添加雪花算法的实例。为什么在引导类中添加该实例，它就能够注入到Spring中呢？因为引导类的注解@SpringBootApplication中已经包含了注解@SpringBootConfiguration了，该注解中又包含了注解@Configuration，因此相当于配置类。

Id生成器如下所示：

//Id生成器  
@Bean  
public IdWorker createIdWorker(){  
 return new IdWorker(1, 1);  
}

接下来就可以在Service中使用该ID生成器了，首先注入该ID生成器：

@Autowired  
private IdWorker idWorker;

然后编写方法save，在该方法中设置id，同时有些并非为后台设置的数据，如浏览量等这些自动自增的数据需要初始化，这个初始化也可以在建表语句中进行初始化。最后将设置好的article对象传入ArticleDao层中，增加到数据库中。

public void save(Article article) {  
 //使用Id生成器  
 String id = idWorker.nextId()+"";  
 article.setId(id);  
  
 //初始化非设置的数据，而是自增的数据  
 article.setVisits(0);//浏览量  
 article.setThumbup(0);//点赞数  
 article.setComment(0);//评论数  
  
 //新增  
 articleDao.insert(article);  
}

1. 测试

使用POSTMAN发送POST请求，并且在Body中设置article对象的参数，注意格式是JSON格式，发送成功后，数据库中多了一行数据。

1. 修改文章功能

修改文章通过PUT请求发送，同样是传入新的文章的Json数据，然后执行修改。

1. 编写Controller方法updateById

该方法需要传入 articleId，和article对象，设置Id，执行修改，返回Result。

//PUT /article/{articleId} 修改文章  
@RequestMapping(value="{articleId}", method=RequestMethod.*PUT*)  
public Result updateById(@PathVariable String articleId, @RequestBody Article article){  
 //设置ID  
 article.setId(articleId);  
 //修改  
 articleService.updateById(article);  
 return new Result(true, StatusCode.*OK*, "修改成功");  
}

1. 编写Serivce方法updateById

可以根据主键id查询，也可以自定义条件进行查询。自定义条件查询需要定义一个条件对象wrapper，这是一个抽象类，在该类中通过CTRL+H查询子类，发现有子类EntityWrapper，因此可以使用该类定义条件。

public void updateById(Article article) {  
 //根据主键ID修改  
 articleDao.updateById(article);  
  
 //根据自定义条件修改  
 //创建条件对象  
 //EntityWrapper<Article> wrapper = new EntityWrapper<>();  
 //设置条件  
 //wrapper.eq("id", article.getId());  
 //articleDao.update(article ,wrapper);  
}

1. 测试

通过POSTMAN进行测试，发送PUT请求，修改某个文章数据，修改成功。

9． 根据ID删除文章功能

（1）编写Controller方法deleteById

//DELETE /article/{articleId} 根据ID删除文章  
@RequestMapping(value="{article}", method=RequestMethod.*DELETE*)  
public Result deletById(@PathVariable String articleId){  
 articleService.deleteById(articleId);  
 return new Result(true, StatusCode.*OK*, "删除成功");  
}

（2） 编写Service方法deletById

public void deleteById(String articleId) {  
 articleDao.deleteById(articleId);  
}

10 . 有条件的分页查询

要实现分页查询，使用MyBatis Plus提供的Page对象，每一个分页就是一个Page对象。并且要通过一个配置类MybatisPlusConfig来向MyBatis Plus中注入PaginationInterceptor插件。条件查询操作通过Mybatis Plus提供的EntityWrapper对象封装where查询条件。

1. MybatisPlus配置类

package com.tensquare.article.config;  
  
import com.baomidou.mybatisplus.plugins.PaginationInterceptor;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class MyBatisPlusConfig {  
   
 @Bean  
 public PaginationInterceptor createPaginationInterceptor(){  
 return new PaginationInterceptor();  
 }  
}

1. 编写Controller类的findByPage方法

通过API文档我们可以看到，需要传入的参数有页码page，页大小size以及查询条件。之前使用一个POJO来接受传入的@RequestBody参数，这里如果将条件转为一个POJO时，由于需要判断传过来的POJO中到底有几个属性，需要对POJO的所有属性进行遍历，而遍历POJO的所有属性需要使用反射的方式，性能较低。这里可以使用集合的方式进行遍历，将POJO改为接受一个Map。Map的键值为POJO的属性名，值为属性的值，这样我们直接从Map中判断传过来的属性值，非常地便捷。

这里的需求是通过条件查询分页，每个分页是一个Page对象，每个Page对象包含着多个文章Article对象。同时，由于分页返回对象的data与其他的返回对象的data不一样，因此我们还特定地封装了分页返回对象PageResult类，将Page对象的total和records都放进去。最后返回Result即可。

//POST /article/search/{page}/{size} 文章分页  
public Result findByPage(@PathVariable Integer page,  
 @PathVariable Integer size,  
 @RequestBody Map<String,Object> map){  
 //根据条件分页查询  
 Page<Article> pageData = articleService.findByPage(page, size, map);  
   
 //封装分页返回对象  
 PageResult<Article> pageResult = new PageResult<>(pageData.getTotal(), pageData.getRecords());  
   
 //返回数据  
 return new Result(true, StatusCode.*OK*, "查询成功")  
}

1. 编写Service的findByPage方法

Service的findByPage方法就是实现分页查询，首先设置查询条件，遍历map，取出非空的键值设置到查询条件中，然后使用DAO层的selectPage分页函数进行分页查询，最后封装返回Page对象。

public Page<Article> findByPage(Integer page, Integer size, Map<String, Object> map) {  
 //设置查询条件  
 EntityWrapper<Article> wrapper = new EntityWrapper<>();  
 Set<String> keyset = map.keySet();  
 for(String key : keyset){  
 //动态SQL的写法  
 wrapper.eq(map.get(key)!=null, key, map.get(key));  
 }  
 //设置分页参数  
 Page<Article> pageData = new Page<>(page, size);  
  
 //执行查询  
 List<Article> list = articleDao.selectPage(pageData, wrapper);  
  
 //封装page对象  
 pageData.setRecords(list);  
 return pageData;  
}

11. 设置公共异常处理类

Controller和Service代码中一定会出现异常，如果要一个个地去处理会非常地麻烦，这里我们可以构建一个公共异常处理类，集中处理异常。

在Controller包下创建一个公共异常处理类BaseExceptionHandler，如下所示：

package com.tensquare.article.controller;  
  
import entity.Result;  
import entity.StatusCode;  
import org.springframework.web.bind.annotation.ControllerAdvice;  
import org.springframework.web.bind.annotation.ExceptionHandler;  
import org.springframework.web.bind.annotation.ResponseBody;  
  
@ControllerAdvice  
public class BaseExceptionHandler {  
 @ExceptionHandler(Exception.class)  
 @ResponseBody  
 public Result handler(Exception e){  
 System.*out*.println("处理异常");  
 return new Result(false, StatusCode.*ERROR*,e.getMessage());  
 }  
}

注解@ControllerService表示这是一个异常处理类，注解@ExceptionHandler表示处理哪个异常。

可以在Controlller中设置一个异常进行测试。

12. 跨域处理

所谓的跨域是指浏览器从一个域名的网页去请求另一个域名的资源时，域名，端口，协议任一不同都是跨域，要解决跨域问题，可以在Controller类上添加注解@CrossOrigin。