环境搭建

## 01-数据库设计

### 一、数据库设计

#### 1、数据库

guli\_edu

#### 2、数据表

guli\_edu.sql



### 二、数据库设计规约

以下规约只针对本模块，更全面的文档参考《阿里巴巴Java开发手册》：五、MySQL数据库

1、库名与应用名称尽量一致

2、表名、字段名必须使用小写字母或数字，禁止出现数字开头，

3、表名不使用复数名词

4、表的命名最好是加上“业务名称\_表的作用”。如，edu\_teacher

5、表必备三字段：id, gmt\_create, gmt\_modified

说明：

其中 id 必为主键，类型为 bigint unsigned、单表时自增、步长为 1。

（如果使用分库分表集群部署，则id类型为verchar，非自增，业务中使用分布式id生成器）

gmt\_create, gmt\_modified 的类型均为 datetime 类型，前者现在时表示主动创建，后者过去分词表示被 动更新。

6、单表行数超过 500 万行或者单表容量超过 2GB，才推荐进行分库分表。 说明：如果预计三年后的数据量根本达不到这个级别，请不要在创建表时就分库分表。

7、表达是与否概念的字段，必须使用 is\_xxx 的方式命名，数据类型是 unsigned tinyint （1 表示是，0 表示否）。

说明：任何字段如果为非负数，必须是 unsigned。

注意：POJO 类中的任何布尔类型的变量，都不要加 is 前缀。数据库表示是与否的值，使用 tinyint 类型，坚持 is\_xxx 的 命名方式是为了明确其取值含义与取值范围。

正例：表达逻辑删除的字段名 is\_deleted，1 表示删除，0 表示未删除。

8、小数类型为 decimal，禁止使用 float 和 double。 说明：float 和 double 在存储的时候，存在精度损失的问题，很可能在值的比较时，得到不 正确的结果。如果存储的数据范围超过 decimal 的范围，建议将数据拆成整数和小数分开存储。

9、如果存储的字符串长度几乎相等，使用 char 定长字符串类型。

10、varchar 是可变长字符串，不预先分配存储空间，长度不要超过 5000，如果存储长度大于此值，定义字段类型为 text，独立出来一张表，用主键来对应，避免影响其它字段索 引效率。

11、唯一索引名为 uk\_字段名；普通索引名则为 idx\_字段名。

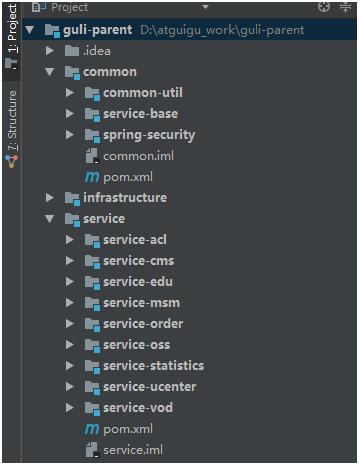
说明：uk\_ 即 unique key；idx\_ 即 index 的简称

12、不得使用外键与级联，一切外键概念必须在应用层解决。外键与级联更新适用于单机低并发，不适合分布式、高并发集群；级联更新是强阻塞，存在数据库更新风暴的风险；外键影响数据库的插入速度。

## 02-搭建项目工程（父工程）

### 一、工程结构介绍

#### 1、工程结构



#### 2、模块说明

**guli-parent：在线教学根目录（父工程），管理四个子模块：**

**canal-client：canal数据库表同步模块（统计同步数据）**

**common：公共模块父节点**

        common-util：工具类模块，所有模块都可以依赖于它

        service-base：service服务的base包，包含service服务的公共配置类，所有service模块依赖于它

        spring-security：认证与授权模块，需要认证授权的service服务依赖于它

**infrastructure：基础服务模块父节点**

        api-gateway：api网关服务

**service：api接口服务父节点**

service-acl：用户权限管理api接口服务（用户管理、角色管理和权限管理等）

service-cms：cms api接口服务

service-edu：教学相关api接口服务

service-msm：短信api接口服务

service-order：订单相关api接口服务

service-oss：阿里云oss api接口服务

service-statistics：统计报表api接口服务

service-ucenter：会员api接口服务

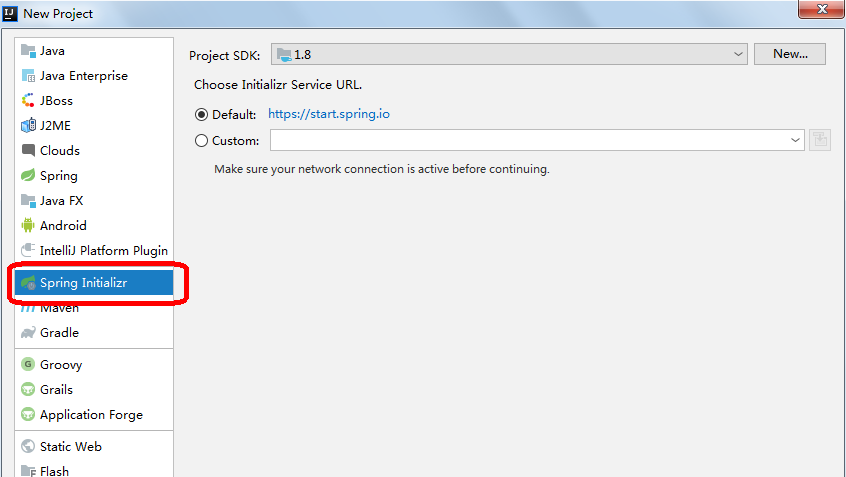
service-vod：视频点播api接口服务

### 二、创建父工程

0、创建新project 选择Spring Initializr创建

**1、创建sprigboot工程guli-parent**

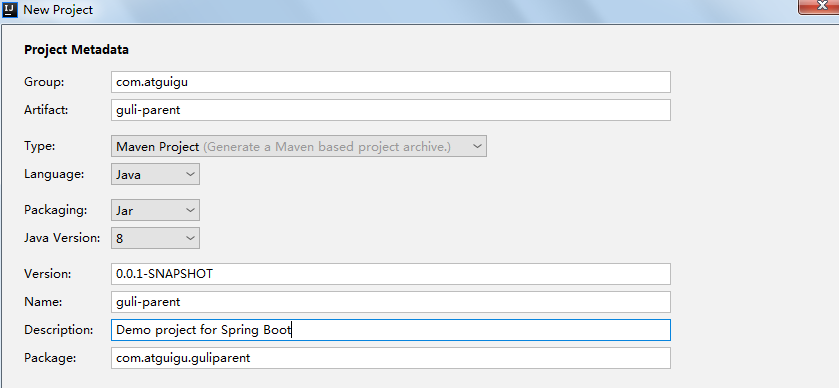
在idea开发工具中，使用 Spring Initializr 快速初始化一个 Spring Boot 模块，版本使用：2.2.1.RELEASE



**配置：**

groupId：com.atguigu  
artifactId：guli-parent

一直下一步到完成

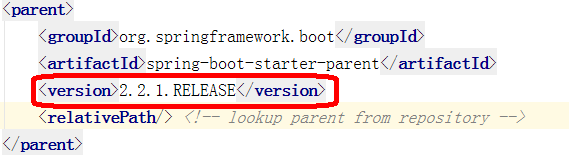


#### 2、删除 src 目录



#### 3、配置 pom.xml

修改版本为 ：2.2.1.RELEASE



<artifactId> 节点后面添加 pom类型

*<*artifactId*>*guli\_parent*</*artifactId*>  
<*packaging*>*pom*</*packaging*>*

**4、在pom.xml中添加依赖的版本**

**删除pom.xml中的**

**<dependencies>内容**

<!-- 以下内容删除 -->

*<*dependencies*>*

*<*dependency*>  
 <*groupId*>*org.springframework.boot*</*groupId*>  
 <*artifactId*>*spring-boot-starter*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*org.springframework.boot*</*groupId*>  
 <*artifactId*>*spring-boot-starter-test*</*artifactId*>  
 <*scope*>*test*</*scope*>  
 </*dependency*>  
</*dependencies*>*

**添加 <properties>确定依赖的版本**

*<*properties*>*

*<*java.version*>*1.8*</*java.version*>  
 <*guli.version*>*0.0.1-SNAPSHOT*</*guli.version*>  
 <*mybatis-plus.version*>*3.0.5*</*mybatis-plus.version*>  
 <*velocity.version*>*2.0*</*velocity.version*>  
 <*swagger.version*>*2.7.0*</*swagger.version*>  
 <*aliyun.oss.version*>*2.8.3*</*aliyun.oss.version*>  
 <*jodatime.version*>*2.10.1*</*jodatime.version*>  
 <*poi.version*>*3.17*</*poi.version*>  
 <*commons-fileupload.version*>*1.3.1*</*commons-fileupload.version*>  
 <*commons-io.version*>*2.6*</*commons-io.version*>  
 <*httpclient.version*>*4.5.1*</*httpclient.version*>  
 <*jwt.version*>*0.7.0*</*jwt.version*>  
 <*aliyun-java-sdk-core.version*>*4.3.3*</*aliyun-java-sdk-core.version*>  
 <*aliyun-sdk-oss.version*>*3.1.0*</*aliyun-sdk-oss.version*>  
 <*aliyun-java-sdk-vod.version*>*2.15.2*</*aliyun-java-sdk-vod.version*>  
 <*aliyun-java-vod-upload.version*>*1.4.11*</*aliyun-java-vod-upload.version*>  
 <*aliyun-sdk-vod-upload.version*>*1.4.11*</*aliyun-sdk-vod-upload.version*>  
 <*fastjson.version*>*1.2.28*</*fastjson.version*>  
 <*gson.version*>*2.8.2*</*gson.version*>  
 <*json.version*>*20170516*</*json.version*>  
 <*commons-dbutils.version*>*1.7*</*commons-dbutils.version*>  
 <*canal.client.version*>*1.1.0*</*canal.client.version*>  
 <*docker.image.prefix*>*zx*</*docker.image.prefix*>  
 <*cloud-alibaba.version*>*0.2.2.RELEASE*</*cloud-alibaba.version*>  
</*properties*>*

**配置 <dependencyManagement> 锁定依赖的版本**

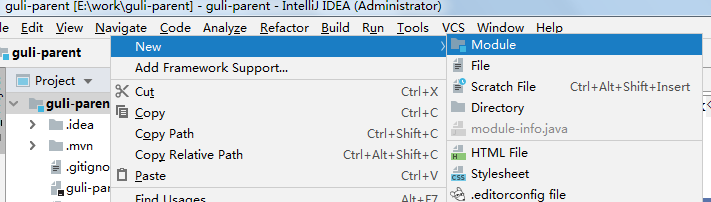
*<*dependencyManagement*>*

*<*dependencies*>* <!--Spring Cloud-->  
 *<*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-dependencies*</*artifactId*>  
 <*version*>*Hoxton.RELEASE*</*version*>  
 <*type*>*pom*</*type*>  
 <*scope*>*import*</*scope*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-alibaba-dependencies*</*artifactId*>  
 <*version*>*${cloud-alibaba.version}*</*version*>  
 <*type*>*pom*</*type*>  
 <*scope*>*import*</*scope*>  
 </*dependency*>* <!--mybatis-plus 持久层-->  
 *<*dependency*>  
 <*groupId*>*com.baomidou*</*groupId*>  
 <*artifactId*>*mybatis-plus-boot-starter*</*artifactId*>  
 <*version*>*${mybatis-plus.version}*</*version*>  
 </*dependency*>* <!-- velocity 模板引擎, Mybatis Plus 代码生成器需要 -->  
 *<*dependency*>  
 <*groupId*>*org.apache.velocity*</*groupId*>  
 <*artifactId*>*velocity-engine-core*</*artifactId*>  
 <*version*>*${velocity.version}*</*version*>  
 </*dependency*>* <!--swagger-->  
 *<*dependency*>  
 <*groupId*>*io.springfox*</*groupId*>  
 <*artifactId*>*springfox-swagger2*</*artifactId*>  
 <*version*>*${swagger.version}*</*version*>  
 </*dependency*>* <!--swagger ui-->  
 *<*dependency*>  
 <*groupId*>*io.springfox*</*groupId*>  
 <*artifactId*>*springfox-swagger-ui*</*artifactId*>  
 <*version*>*${swagger.version}*</*version*>  
 </*dependency*>* <!--aliyunOSS-->  
 *<*dependency*>  
 <*groupId*>*com.aliyun.oss*</*groupId*>  
 <*artifactId*>*aliyun-sdk-oss*</*artifactId*>  
 <*version*>*${aliyun.oss.version}*</*version*>  
 </*dependency*>* <!--日期时间工具-->  
 *<*dependency*>  
 <*groupId*>*joda-time*</*groupId*>  
 <*artifactId*>*joda-time*</*artifactId*>  
 <*version*>*${jodatime.version}*</*version*>  
 </*dependency*>* <!--xls-->  
 *<*dependency*>  
 <*groupId*>*org.apache.poi*</*groupId*>  
 <*artifactId*>*poi*</*artifactId*>  
 <*version*>*${poi.version}*</*version*>  
 </*dependency*>* <!--xlsx-->  
 *<*dependency*>  
 <*groupId*>*org.apache.poi*</*groupId*>  
 <*artifactId*>*poi-ooxml*</*artifactId*>  
 <*version*>*${poi.version}*</*version*>  
 </*dependency*>* <!--文件上传-->  
 *<*dependency*>  
 <*groupId*>*commons-fileupload*</*groupId*>  
 <*artifactId*>*commons-fileupload*</*artifactId*>  
 <*version*>*${commons-fileupload.version}*</*version*>  
 </*dependency*>* <!--commons-io-->  
 *<*dependency*>  
 <*groupId*>*commons-io*</*groupId*>  
 <*artifactId*>*commons-io*</*artifactId*>  
 <*version*>*${commons-io.version}*</*version*>  
 </*dependency*>* <!--httpclient-->  
 *<*dependency*>  
 <*groupId*>*org.apache.httpcomponents*</*groupId*>  
 <*artifactId*>*httpclient*</*artifactId*>  
 <*version*>*${httpclient.version}*</*version*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.google.code.gson*</*groupId*>  
 <*artifactId*>*gson*</*artifactId*>  
 <*version*>*${gson.version}*</*version*>  
 </*dependency*>* <!-- JWT -->  
 *<*dependency*>  
 <*groupId*>*io.jsonwebtoken*</*groupId*>  
 <*artifactId*>*jjwt*</*artifactId*>  
 <*version*>*${jwt.version}*</*version*>  
 </*dependency*>* <!--aliyun-->  
 *<*dependency*>  
 <*groupId*>*com.aliyun*</*groupId*>  
 <*artifactId*>*aliyun-java-sdk-core*</*artifactId*>  
 <*version*>*${aliyun-java-sdk-core.version}*</*version*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.aliyun.oss*</*groupId*>  
 <*artifactId*>*aliyun-sdk-oss*</*artifactId*>  
 <*version*>*${aliyun-sdk-oss.version}*</*version*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.aliyun*</*groupId*>  
 <*artifactId*>*aliyun-java-sdk-vod*</*artifactId*>  
 <*version*>*${aliyun-java-sdk-vod.version}*</*version*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.aliyun*</*groupId*>  
 <*artifactId*>*aliyun-java-vod-upload*</*artifactId*>  
 <*version*>*${aliyun-java-vod-upload.version}*</*version*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.aliyun*</*groupId*>  
 <*artifactId*>*aliyun-sdk-vod-upload*</*artifactId*>  
 <*version*>*${aliyun-sdk-vod-upload.version}*</*version*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.alibaba*</*groupId*>  
 <*artifactId*>*fastjson*</*artifactId*>  
 <*version*>*${fastjson.version}*</*version*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*org.json*</*groupId*>  
 <*artifactId*>*json*</*artifactId*>  
 <*version*>*${json.version}*</*version*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*commons-dbutils*</*groupId*>  
 <*artifactId*>*commons-dbutils*</*artifactId*>  
 <*version*>*${commons-dbutils.version}*</*version*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.alibaba.otter*</*groupId*>  
 <*artifactId*>*canal.client*</*artifactId*>  
 <*version*>*${canal.client.version}*</*version*>  
 </*dependency*>  
 </*dependencies*>  
</*dependencyManagement*>*

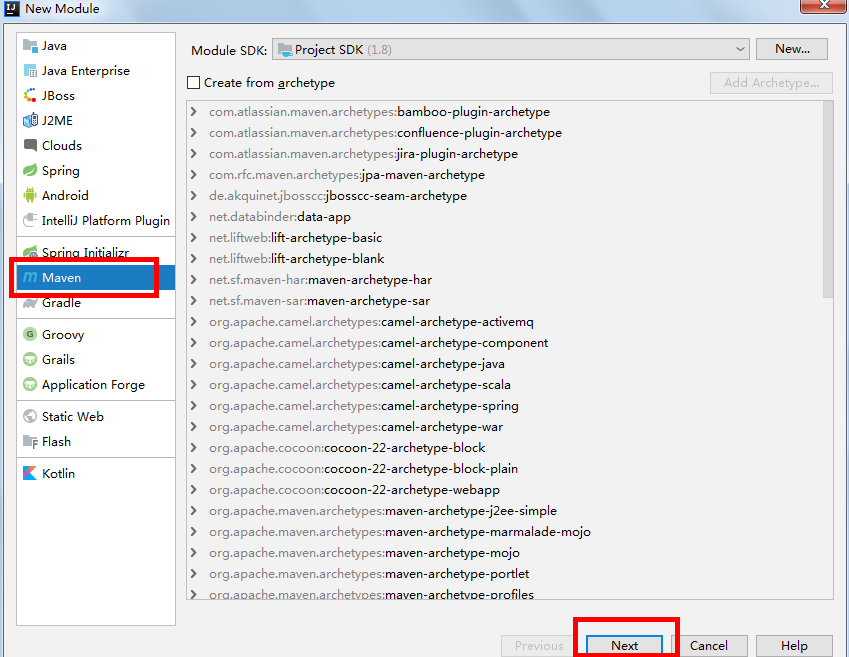
## 03-搭建项目工程（service模块）

### 一、搭建service模块

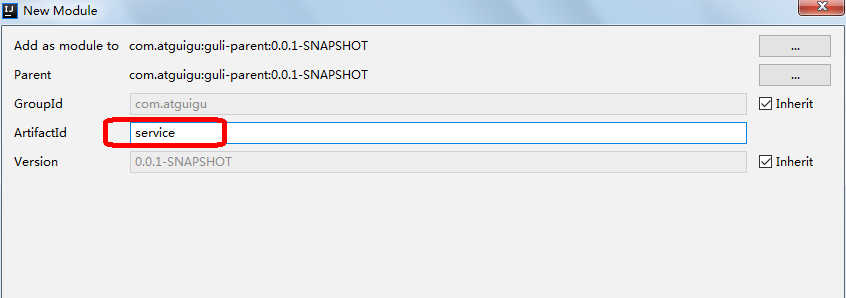
#### 1、在父工程guli-parent下面创建模块service



**选择 maven类型，点击下一步**



**输入模块名称 service，下一步完成创建**



#### 2、添加模块类型是pom

<artifactId> 节点后面添加 pom类型

*<*artifactId*>*service*</*artifactId*>  
<*packaging*>*pom*</*packaging*>*

#### 3、添加项目需要的依赖

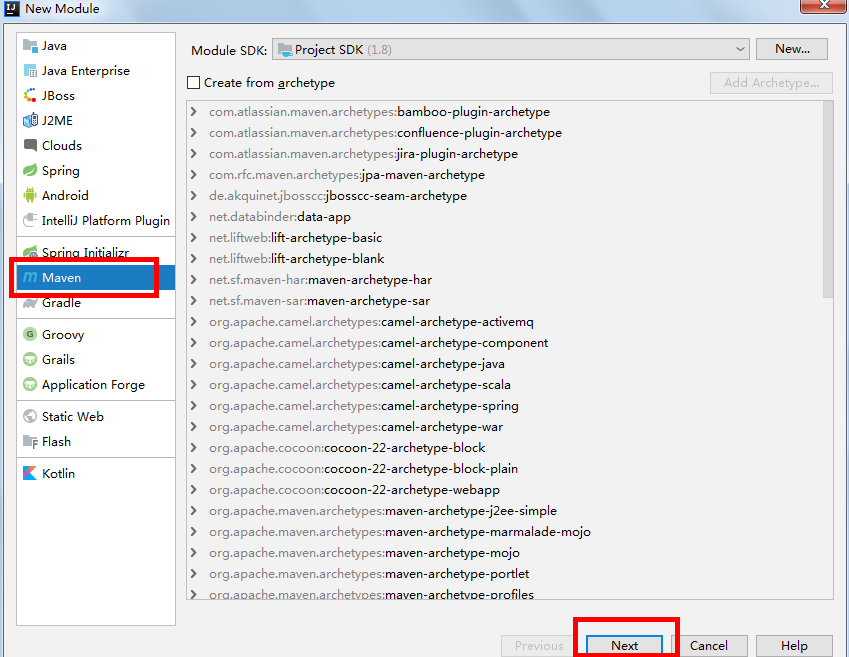
*<*dependencies*>*

*<*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-netflix-ribbon*</*artifactId*>  
 </*dependency*>* <!--hystrix依赖，主要是用 @HystrixCommand -->  
 *<*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-netflix-hystrix*</*artifactId*>  
 </*dependency*>* <!--服务注册-->  
 *<*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-alibaba-nacos-discovery*</*artifactId*>  
 </*dependency*>* <!--服务调用-->  
 *<*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-openfeign*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*org.springframework.boot*</*groupId*>  
 <*artifactId*>*spring-boot-starter-web*</*artifactId*>  
 </*dependency*>* <!--mybatis-plus-->  
 *<*dependency*>  
 <*groupId*>*com.baomidou*</*groupId*>  
 <*artifactId*>*mybatis-plus-boot-starter*</*artifactId*>  
 </*dependency*>* <!--mysql-->  
 *<*dependency*>  
 <*groupId*>*mysql*</*groupId*>  
 <*artifactId*>*mysql-connector-java*</*artifactId*>  
 </*dependency*>* <!-- velocity 模板引擎, Mybatis Plus 代码生成器需要 -->  
 *<*dependency*>  
 <*groupId*>*org.apache.velocity*</*groupId*>  
 <*artifactId*>*velocity-engine-core*</*artifactId*>  
 </*dependency*>* <!--swagger-->  
 *<*dependency*>  
 <*groupId*>*io.springfox*</*groupId*>  
 <*artifactId*>*springfox-swagger2*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*io.springfox*</*groupId*>  
 <*artifactId*>*springfox-swagger-ui*</*artifactId*>  
 </*dependency*>* <!--lombok用来简化实体类：需要安装lombok插件-->  
 *<*dependency*>  
 <*groupId*>*org.projectlombok*</*groupId*>  
 <*artifactId*>*lombok*</*artifactId*>  
 </*dependency*>* <!--xls-->  
 *<*dependency*>  
 <*groupId*>*org.apache.poi*</*groupId*>  
 <*artifactId*>*poi*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*org.apache.poi*</*groupId*>  
 <*artifactId*>*poi-ooxml*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*commons-fileupload*</*groupId*>  
 <*artifactId*>*commons-fileupload*</*artifactId*>  
 </*dependency*>* <!--httpclient-->  
 *<*dependency*>  
 <*groupId*>*org.apache.httpcomponents*</*groupId*>  
 <*artifactId*>*httpclient*</*artifactId*>  
 </*dependency*>* <!--commons-io-->  
 *<*dependency*>  
 <*groupId*>*commons-io*</*groupId*>  
 <*artifactId*>*commons-io*</*artifactId*>  
 </*dependency*>* <!--gson-->  
 *<*dependency*>  
 <*groupId*>*com.google.code.gson*</*groupId*>  
 <*artifactId*>*gson*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*junit*</*groupId*>  
 <*artifactId*>*junit*</*artifactId*>  
 <*version*>*4.12*</*version*>  
 </*dependency*>  
</*dependencies*>*

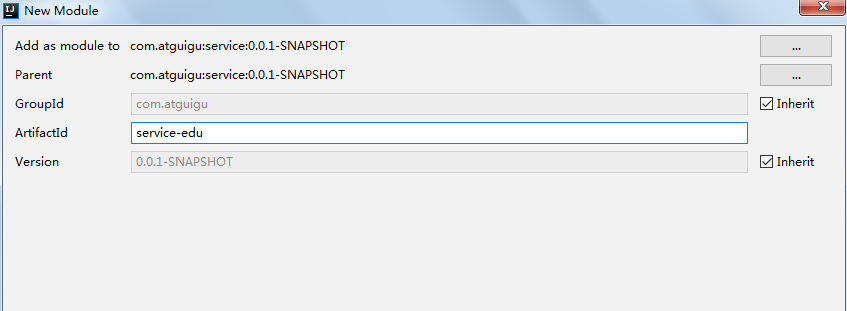
Service 模块不需要编码删除src目录

### 二、搭建service-edu模块

#### 1、在父工程service模块下面创建子模块service-edu



**输入模块名称 service-edu，下一步完成创建**

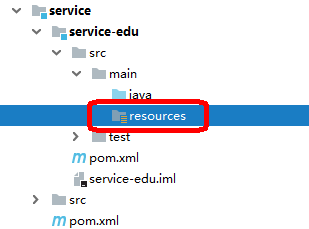


讲师管理接口开发

## 讲师管理模块配置和生成代码

### 一、讲师管理模块配置

**1、在service下面service-edu模块中创建配置文件**



**resources目录下创建文件 application.properties**

# 服务端口

server.port=8001  
# 服务名  
spring.application.name=service-edu  
# 环境设置：dev、test、prod  
spring.profiles.active=dev  
# mysql数据库连接  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/guli?serverTimezone=GMT%2B8  
spring.datasource.username=root  
spring.datasource.password=root  
#mybatis日志  
mybatis-plus.configuration.log-impl=org.apache.ibatis.logging.stdout.StdOutImpl

#mapperxml配置地址

mybatis-plus.mapper-locations=classpath:/com/atguigu/eduservice/mapper/xml/\*\*.xml

**或者在resources目录下创建文件 application.yml**

#### application.yml

spring:

application:

name: service-edu

profiles:

active: dev

#### application-dev.yml

server:

port: 8001

mybatis-plus:

configuration:

log-impl: org.apache.ibatis.logging.stdout.StdOutImpl

mapper-locations: classpath:com/atguigu/service/\*/mapper/\*.xml

global-config:

db-config:

logic-delete-value: 1

logic-not-delete-value: 0

spring:

datasource:

type: com.zaxxer.hikari.HikariDataSource

driver-class-name: com.mysql.cj.jdbc.Driver

url: jdbc:mysql://localhost:3306/guli?serverTimezone=GMT%2B8

username: root

password: root

hikari:

connection-test-query: SELECT 1

connection-timeout: 60000

idle-timeout: 500000

max-lifetime: 540000

maximum-pool-size: 12

minimum-idle: 10

pool-name: GuliHikariPool

jackson:

date-format: yyyy-MM-dd HH:mm:ss

time-zone: GMT+8

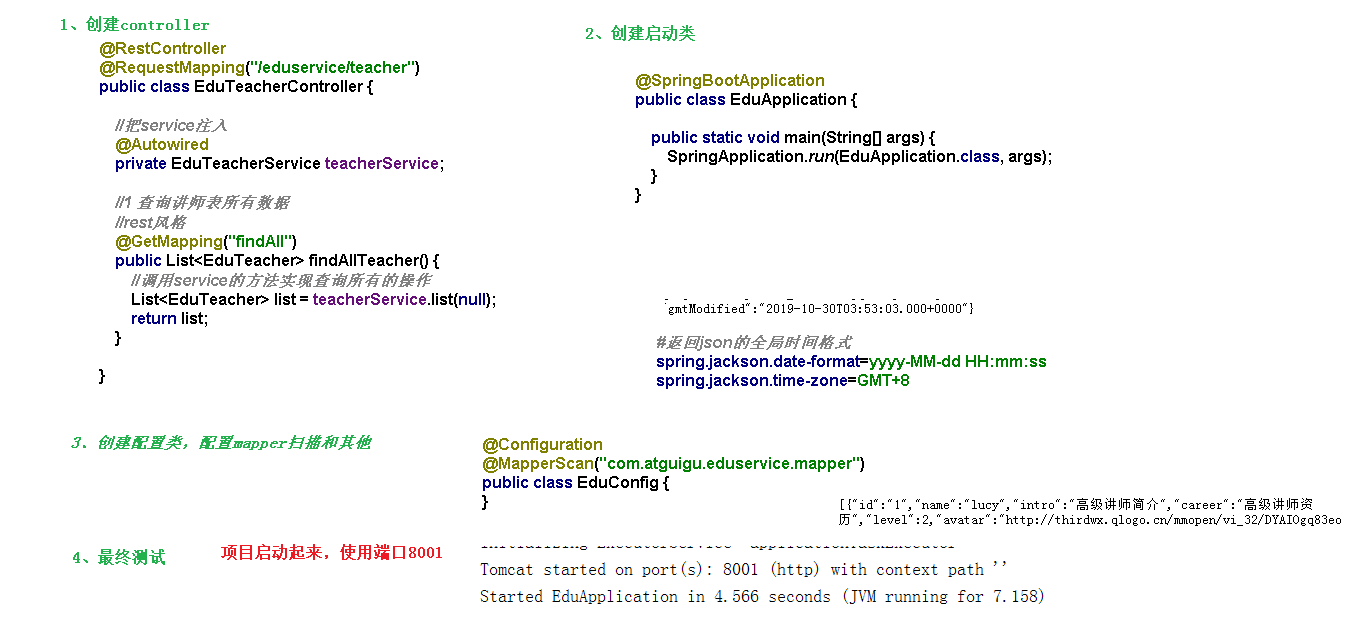
#### 2、创建MP代码生成器

在test/java目录下创建包com.atguigu.eduservice，创建代码生成器：CodeGenerator.java

public class CodeGenerator *{*

@Test  
 public void run*() {* // 1、创建代码生成器  
 AutoGenerator mpg = new AutoGenerator*()*;  
  
 // 2、全局配置  
 GlobalConfig gc = new GlobalConfig*()*;  
 String projectPath = System.*getProperty(*"user.dir"*)*;  
 //===修改为绝对路径===  
 gc.setOutputDir*(*"D:\\code\\guli\_school\\guli\_parent\\service\\service\_edu" + "/src/main/java"*)*;  
 gc.setAuthor*(*"gouwenyong"*)*;  
 gc.setOpen*(*false*)*; //生成后是否打开资源管理器  
 gc.setFileOverride*(*false*)*; //重新生成时文件是否覆盖  
 gc.setServiceName*(*"%sService"*)*; //去掉Service接口的首字母I  
 gc.setIdType*(*IdType.*ID\_WORKER)*; //主键策略 ID\_WORKER\_STR  
 gc.setDateType*(*DateType.*ONLY\_DATE)*;//定义生成的实体类中日期类型  
 gc.setSwagger2*(*true*)*;//开启Swagger2模式  
  
 mpg.setGlobalConfig*(*gc*)*;  
  
 // 3、数据源配置  
 DataSourceConfig dsc = new DataSourceConfig*()*;  
 dsc.setUrl*(*"jdbc:mysql://localhost:3306/guli?serverTimezone=GMT%2B8"*)*;  
 dsc.setDriverName*(*"com.mysql.cj.jdbc.Driver"*)*;  
 dsc.setUsername*(*"root"*)*;  
 dsc.setPassword*(*"root"*)*;  
 dsc.setDbType*(*DbType.*MYSQL)*;  
 mpg.setDataSource*(*dsc*)*;  
  
 // 4、包配置  
 PackageConfig pc = new PackageConfig*()*;  
 pc.setModuleName*(*"eduservice"*)*; //模块名  
  
 pc.setParent*(*"com.atguigu"*)*;//包名  
 pc.setController*(*"controller"*)*;  
 pc.setEntity*(*"entity"*)*;  
 pc.setService*(*"service"*)*;  
 pc.setMapper*(*"mapper"*)*;  
 mpg.setPackageInfo*(*pc*)*;  
  
 // 5、策略配置  
 StrategyConfig strategy = new StrategyConfig*()*;  
 strategy.setInclude*(*"edu\_teacher"*)*;//对应的表名  
 strategy.setNaming*(*NamingStrategy.*underline\_to\_camel)*;//数据库表映射到实体的命名策略  
 strategy.setTablePrefix*(*pc.getModuleName*()* + "\_"*)*; //生成实体时去掉表前缀  
  
 strategy.setColumnNaming*(*NamingStrategy.*underline\_to\_camel)*;//数据库表字段映射到实体的命名策略  
 strategy.setEntityLombokModel*(*true*)*; // lombok 模型 @Accessors(chain = true) setter链式操作  
  
 strategy.setRestControllerStyle*(*true*)*; //restful api风格控制器  
 strategy.setControllerMappingHyphenStyle*(*true*)*; //url中驼峰转连字符  
  
 mpg.setStrategy*(*strategy*)*;  
  
  
 // 6、执行  
 mpg.execute*()*;  
 *}  
}*

### 二、编写后台管理api接口



**1、编写controller代码**

@Autowired

private EduTeacherService eduTeacherService;  
  
//查询讲师表所有数据  
@GetMapping*(*"/findAll"*)*public List*<*EduTeacher*>* getAllTeacher*(){* List*<*EduTeacher*>* list = eduTeacherService.list*(*null*)*;  
 return list;  
*}*

#### 2、创建SpringBoot配置类

在edu包下创建config包，创建MyBatisPlusConfig.java

package com.atguigu.eduservice.config;

import org.mybatis.spring.annotation.MapperScan;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.transaction.annotation.EnableTransactionManagement;  
  
@Configuration  
@EnableTransactionManagement  
@MapperScan*(*"com.atguigu.eduservice.mapper"*)*public class MyBatisPlusConfig *{  
  
}*

**3、**MyBatisPlusConfig**配置SQL执行性能分析插件**

*/\*\**

*\* SQL 执行性能分析插件  
 \* 开发环境使用，线上不推荐。 maxTime 指的是 sql 最大执行时长  
 \*/*@Bean  
@Profile*({*"dev", "test"*})*// 设置 dev test 环境开启  
public PerformanceInterceptor performanceInterceptor*() {* PerformanceInterceptor performanceInterceptor = new PerformanceInterceptor*()*;  
 performanceInterceptor.setMaxTime*(*1000*)*;//ms，超过此处设置的ms则sql不执行  
 performanceInterceptor.setFormat*(*true*)*;  
 return performanceInterceptor;  
*}*

#### 4、创建SpringBoot启动类

创建启动类 EduApplication.java，注意启动类的创建位置

@SpringBootApplication

public class EduApplication *{* public static void main*(*String*[]* args*) {* SpringApplication.*run(*EduApplication.class, args*)*;  
 *}  
}*

#### 5、运行启动类

访问GET http://localhost:8001/eduservice/teacher/findAll

得到json数据

#### 6、统一返回的json时间格式

默认情况下json时间格式带有时区，并且是世界标准时间，和我们的时间差了八个小时

在application.properties中设置

#返回json的全局时间格式

spring.jackson.date-format=yyyy-MM-dd HH:mm:ss  
spring.jackson.time-zone=GMT+8

### 三、讲师逻辑删除功能



#### 1、EduTeacherController添加删除方法

@DeleteMapping*(*"/{id}"*)*

public boolean removeTeacher*(*@PathVariable Integer id*){* boolean remove = eduTeacherService.removeById*(*id*)*;  
 return remove;  
*}*

#### 2、配置逻辑删除插件

MyBatisPlusConfig中配置

*/\*\**

*\* 逻辑删除插件  
 \*/*@Bean  
public ISqlInjector sqlInjector*() {* return new LogicSqlInjector*()*;  
*}*

#### 3、逻辑标识位添加注解@TableLogic

@TableLogic

private Integer isDeleted;

#### 4、使用postman测试删除



测试结果：数据库中的is\_deleted字段被修改为1

### 四、跨域配置

#### 1、什么是跨域

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 。前后端分离开发中，需要考虑ajax跨域的问题。

这里我们可以从服务端解决这个问题

#### 2、配置

在Controller类上添加注解

@CrossOrigin //跨域

## 配置Swagger2生成API接口文档

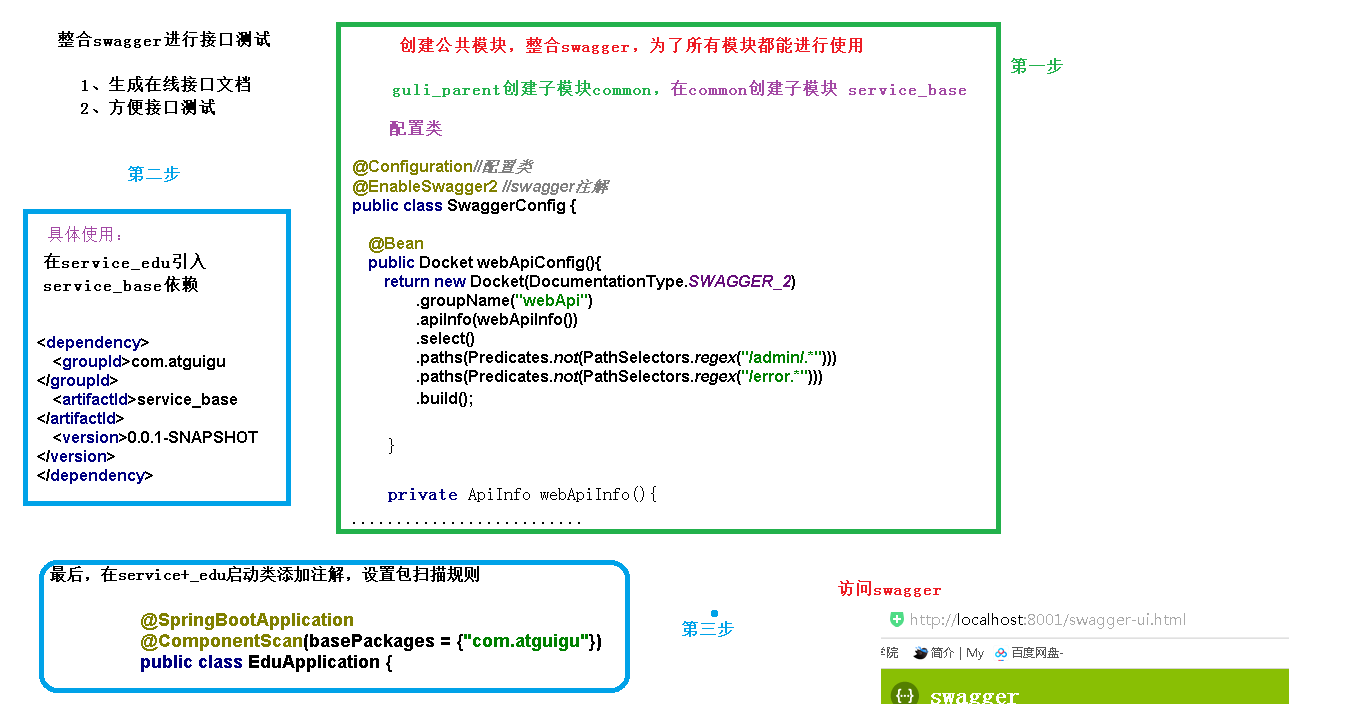
### 一、Swagger2介绍

前后端分离开发模式中，api文档是最好的沟通方式。

Swagger 是一个规范和完整的框架，用于生成、描述、调用和可视化 RESTful 风格的 Web 服务。

1. 及时性 (接口变更后，能够及时准确地通知相关前后端开发人员)
2. 规范性 (并且保证接口的规范性，如接口的地址，请求方式，参数及响应格式和错误信息)
3. 一致性 (接口信息一致，不会出现因开发人员拿到的文档版本不一致，而出现分歧)
4. 可测性 (直接在接口文档上进行测试，以方便理解业务)

### 二、配置Swagger2



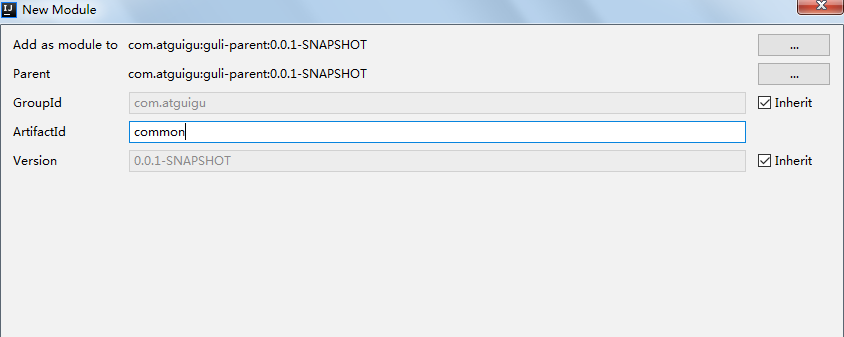
**1、创建common模块**

**在guli-parent下创建模块common**

配置：

groupId：com.atguigu

artifactId：common



修改打包方式：

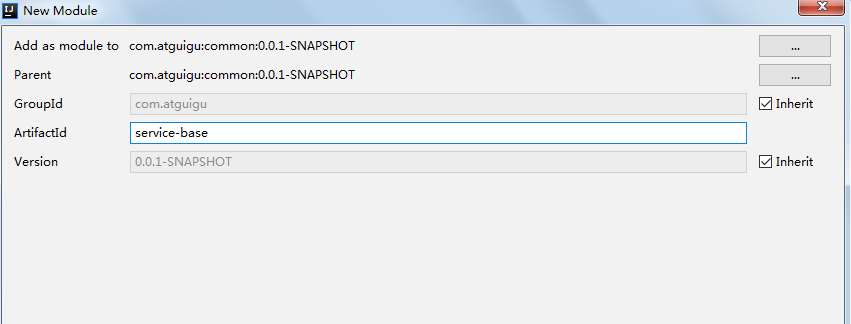
*<*artifactId*>*common*</*artifactId*>  
<*packaging*>*pom*</*packaging*>*

**2、在common中引入相关依赖**

*<*dependencies*>*

*<*dependency*>  
 <*groupId*>*org.springframework.boot*</*groupId*>  
 <*artifactId*>*spring-boot-starter-web*</*artifactId*>  
 <*scope*>*provided*</*scope*>  
 </*dependency*>* <!--mybatis-plus-->  
 *<*dependency*>  
 <*groupId*>*com.baomidou*</*groupId*>  
 <*artifactId*>*mybatis-plus-boot-starter*</*artifactId*>  
 <*scope*>*provided*</*scope*>  
 </*dependency*>* <!--lombok用来简化实体类：需要安装lombok插件-->  
 *<*dependency*>  
 <*groupId*>*org.projectlombok*</*groupId*>  
 <*artifactId*>*lombok*</*artifactId*>  
 <*scope*>*provided*</*scope*>  
 </*dependency*>* <!--swagger-->  
 *<*dependency*>  
 <*groupId*>*io.springfox*</*groupId*>  
 <*artifactId*>*springfox-swagger2*</*artifactId*>  
 <*scope*>*provided*</*scope*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*io.springfox*</*groupId*>  
 <*artifactId*>*springfox-swagger-ui*</*artifactId*>  
 <*scope*>*provided*</*scope*>  
 </*dependency*>* <!-- redis -->  
 *<*dependency*>  
 <*groupId*>*org.springframework.boot*</*groupId*>  
 <*artifactId*>*spring-boot-starter-data-redis*</*artifactId*>  
 </*dependency*>* <!-- spring2.X集成redis所需common-pool2  
 <dependency>  
 <groupId>org.apache.commons</groupId>  
 <artifactId>commons-pool2</artifactId>  
 <version>2.6.0</version>  
 </dependency>-->  
*</*dependencies*>*

#### 3、在common下面创建子模块service-base



#### 3、在模块service-base中，创建swagger的配置类

创建包com.atguigu.servicebase.config，创建类SwaggerConfig

import com.google.common.base.Predicates;

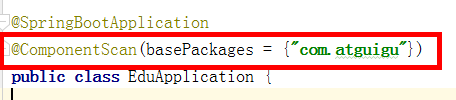
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import springfox.documentation.builders.ApiInfoBuilder;  
import springfox.documentation.builders.PathSelectors;  
import springfox.documentation.service.ApiInfo;  
import springfox.documentation.service.Contact;  
import springfox.documentation.spi.DocumentationType;  
import springfox.documentation.spring.web.plugins.Docket;  
import springfox.documentation.swagger2.annotations.EnableSwagger2;  
  
@Configuration  
@EnableSwagger2  
public class SwaggerConfig *{* @Bean  
 public Docket webApiConfig*() {* return new Docket*(*DocumentationType.*SWAGGER\_2)* .groupName*(*"webApi"*)* .apiInfo*(*webApiInfo*())* .select*()* .paths*(*Predicates.*not(*PathSelectors.*regex(*"/admin/.\*"*)))*//不生产测试的路径  
 .paths*(*Predicates.*not(*PathSelectors.*regex(*"/error.\*"*)))* .build*()*;  
 *}* private ApiInfo webApiInfo*() {* return new ApiInfoBuilder*()* .title*(*"网站-课程中心API文档"*)* .description*(*"本文档描述了课程中心微服务接口定义"*)* .version*(*"1.0"*)* .contact*(*new Contact*(*"Helen", "http://atguigu.com", "55317332@qq.com"*))* .build*()*;  
 *}  
}*

#### 4、在模块service模块中引入service-base

<!--swagger 配置-->

*<*dependency*>  
 <*groupId*>*com.atguigu*</*groupId*>  
 <*artifactId*>*service-base*</*artifactId*>  
 <*version*>*0.0.1-SNAPSHOT*</*version*>  
 </*dependency*>*

**5、在service-edu启动类上添加注解，进行测试**



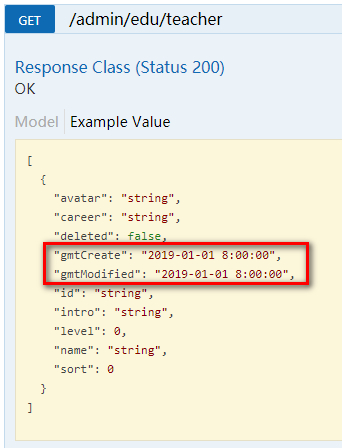
#### 6、API模型

可以添加一些自定义设置，例如：

定义样例数据

@ApiModelProperty*(*value = "创建时间", example = "2019-01-01 8:00:00"*)*

@TableField*(*fill = FieldFill.*INSERT)*private Date gmtCreate;  
@ApiModelProperty*(*value = "更新时间", example = "2019-01-01 8:00:00"*)*@TableField*(*fill = FieldFill.*INSERT\_UPDATE)*private Date gmtModified;



http://localhost:8001/swagger-ui.html

#### 5、定义接口说明和参数说明

定义在类上：@Api

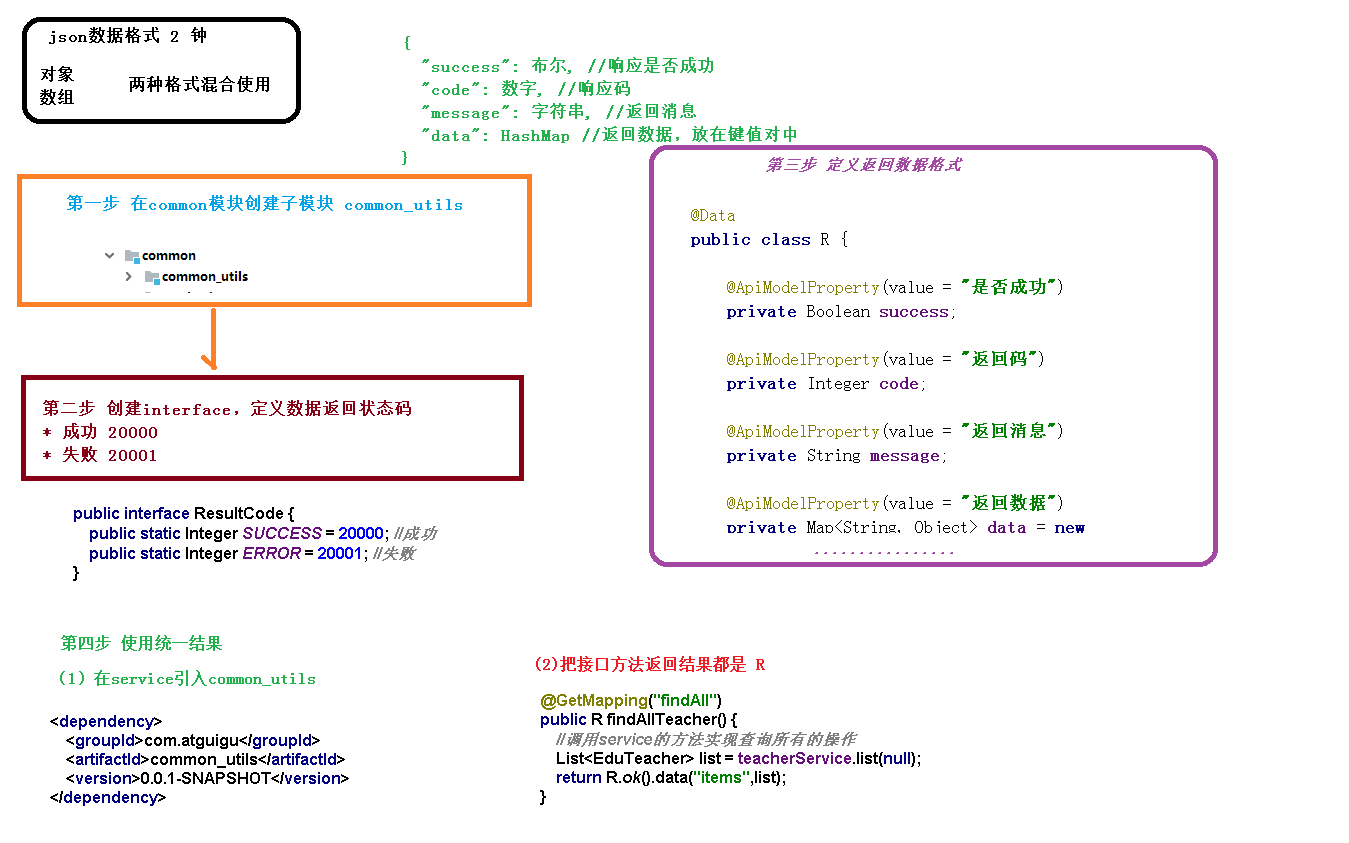
定义在方法上：@ApiOperation

定义在参数上：@ApiParam

@Api*(*description = "讲师管理"*)*

@RestController  
@RequestMapping*(*"/admin/edu/teacher"*)*public class TeacherAdminController *{* @Autowired  
 private TeacherService teacherService;  
  
 @ApiOperation*(*value = "所有讲师列表"*)* @GetMapping  
 public List*<*Teacher*>* list*() {* return teacherService.list*(*null*)*;  
 *}* @ApiOperation*(*value = "根据ID删除讲师"*)* @DeleteMapping*(*"{id}"*)* public boolean removeById*(* @ApiParam*(*name = "id", value = "讲师ID", required = true*)* @PathVariable String id*) {* return teacherService.removeById*(*id*)*;  
 *}  
}*

## 03-统一返回结果对象



### 一、统一返回数据格式

项目中我们会将响应封装成json返回，一般我们会将所有接口的数据格式统一， 使前端(iOS Android, Web)对数据的操作更一致、轻松。

一般情况下，统一返回数据格式没有固定的格式，只要能描述清楚返回的数据状态以及要返回的具体数据就可以。但是一般会包含状态码、返回消息、数据这几部分内容

例如，我们的系统要求返回的基本数据格式如下：

**列表：**

{

"success": true,

"code": 20000,

"message": "成功",

"data": {

"items": [

{

"id": "1",

"name": "刘德华",

"intro": "毕业于师范大学数学系，热爱教育事业，执教数学思维6年有余"

}

]

}

}

**分页：**

{

"success": true,

"code": 20000,

"message": "成功",

"data": {

"total": 17,

"rows": [

{

"id": "1",

"name": "刘德华",

"intro": "毕业于师范大学数学系，热爱教育事业，执教数学思维6年有余"

}

]

}

}

**没有返回数据：**

{

"success": true,

"code": 20000,

"message": "成功",

"data": {}

}

**失败：**

{

"success": false,

"code": 20001,

"message": "失败",

"data": {}

}

因此，我们定义统一结果

{

"success": 布尔, //响应是否成功

"code": 数字, //响应码

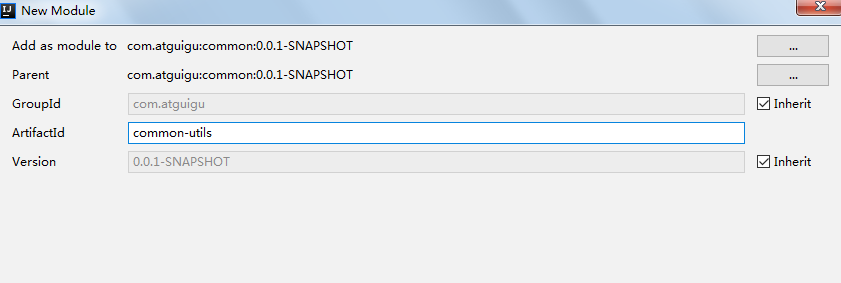
"message": 字符串, //返回消息

"data": HashMap //返回数据，放在键值对中

}

### 二、创建统一结果返回类

#### 1、在common模块下创建子模块common-utils



**2、创建接口定义返回码**

**创建包com.atguigu.commonutils，创建接口 ResultCode.java**

package com.atguigu.commonutils;

public interface ResultCode *{* public static Integer *SUCCESS* = 20000;  
 public static Integer *ERROR* = 20001;  
*}*

#### 3、创建结果类

**创建类 R.java**

@Data

public class R *{* @ApiModelProperty*(*value = "是否成功"*)* private Boolean success;  
 @ApiModelProperty*(*value = "返回码"*)* private Integer code;  
 @ApiModelProperty*(*value = "返回消息"*)* private String message;  
 @ApiModelProperty*(*value = "返回数据"*)* private Map*<*String, Object*>* data = new HashMap*<*String, Object*>()*;  
  
 //私有化构造器  
 private R*() {  
 }* public static R ok*() {* R r = new R*()*;  
 r.setSuccess*(*true*)*;  
 r.setCode*(*ResultCode.*SUCCESS)*;  
 r.setMessage*(*"成功"*)*;  
 return r;  
 *}* public static R error*() {* R r = new R*()*;  
 r.setSuccess*(*false*)*;  
 r.setCode*(*ResultCode.*ERROR)*;  
 r.setMessage*(*"失败"*)*;  
 return r;  
 *}* public R success*(*Boolean success*) {* this.setSuccess*(*success*)*;  
 return this;  
 *}* public R message*(*String message*) {* this.setMessage*(*message*)*;  
 return this;  
 *}* public R code*(*Integer code*) {* this.setCode*(*code*)*;  
 return this;  
 *}* public R data*(*String key, Object value*) {* this.data.put*(*key, value*)*;  
 return this;  
 *}* public R data*(*Map*<*String, Object*>* map*) {* this.setData*(*map*)*;  
 return this;  
 *}  
}*

### 三、统一返回结果使用

#### 1、在service模块中添加依赖

*<*dependency*>*

*<*groupId*>*com.atguigu*</*groupId*>  
 <*artifactId*>*common-utils*</*artifactId*>  
 <*version*>*0.0.1-SNAPSHOT*</*version*>  
</*dependency*>*

#### 2、修改Controller中的返回结果

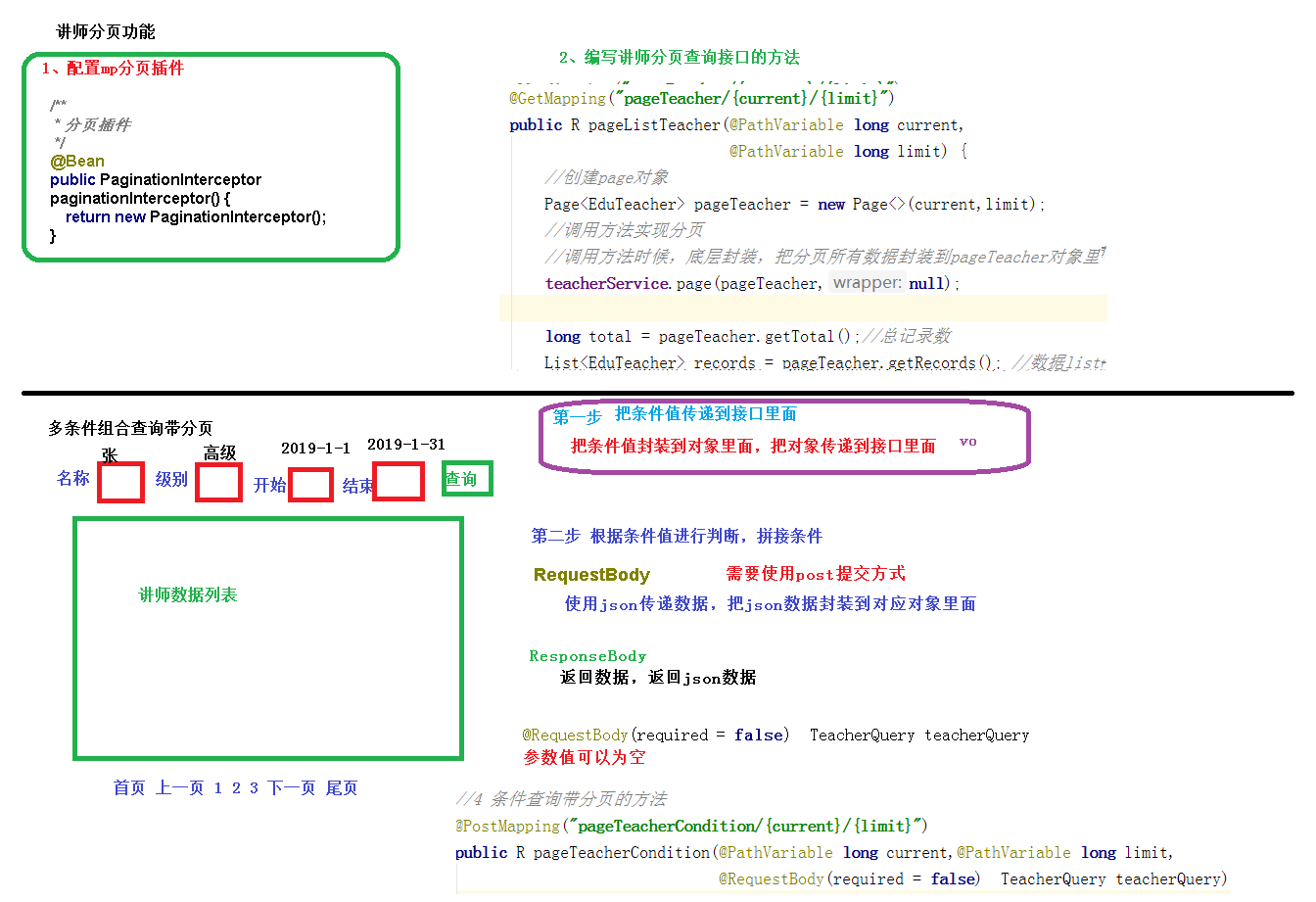
列表

@ApiOperation*(*value = "所有讲师列表"*)*@GetMapping*(*"/findAll"*)*public R getAllTeacher*(){* List*<*EduTeacher*>* list = eduTeacherService.list*(*null*)*;  
 return R.*ok()*.data*(*"items",list*)*;  
*}*

删除

@ApiOperation*(*value = "根据ID删除讲师"*)*@DeleteMapping*(*"/{id}"*)*public R removeTeacher*(* @ApiParam*(*name = "id", value = "讲师ID", required = true*)* @PathVariable String id*){* boolean remove = eduTeacherService.removeById*(*id*)*;  
 return R.*ok()*;  
*}*

## 分页和条件查询接口开发



### 一、分页

#### 1、MyBatisPlusConfig中配置分页插件

*/\*\**

*\* 分页插件  
 \*/*@Bean  
public PaginationInterceptor paginationInterceptor*() {* return new PaginationInterceptor*()*;  
*}*

#### 2、分页Controller方法

TeacherAdminController中添加分页方法

@ApiOperation*(*value = "分页讲师列表"*)*

@GetMapping*(*"/pageTeacher/{current}/{limit}"*)*public R pageList*(* @ApiParam*(*name = "current", value = "当前页码", required = true*)* @PathVariable Long current,  
 @ApiParam*(*name = "limit", value = "每页记录数", required = true*)* @PathVariable Long limit*) {* //创建page对象  
 Page*<*EduTeacher*>* pageTeacher = new Page*<*EduTeacher*>(*current, limit*)*;  
 //分页查询 底层插件封装  
 IPage*<*EduTeacher*>* page = eduTeacherService.page*(*pageTeacher, null*)*;  
  
 return R.*ok()* .data*(*"total", page.getTotal*())* .data*(*"items", page.getRecords*())*;  
*}*

#### 3、Swagger中测试

### 二、条件查询

根据讲师名称name，讲师头衔level、讲师入驻时间gmt\_create（时间段）查询

#### 1、创建查询对象

创建com.atguigu.eduservice.entity.vo;包，创建TeacherQuery.java查询对象

@ApiModel*(*value = "Teacher查询对象", description = "讲师查询对象封装"*)*

@Data  
public class TeacherQuery implements Serializable *{* private static final long *serialVersionUID* = 8498579144988180444L;  
 @ApiModelProperty*(*value = "教师名称,模糊查询"*)* private String name;  
 @ApiModelProperty*(*value = "头衔 1高级讲师 2首席讲师"*)* private Integer level;  
  
 @ApiModelProperty*(*value = "查询开始时间", example = "2019-01-01 10:10:10"*)* private String begin;//注意，这里使用的是String类型，前端传过来的数据无需进行类型转换  
 @ApiModelProperty*(*value = "查询结束时间", example = "2019-12-01 10:10:10"*)* private String end;  
*}*

@ApiOperation*(*value = "分页讲师列表"*)*@GetMapping*(*"pageTeacherCondition/{current}/{limit}"*)*public R pageTeacherCondition*(* @ApiParam*(*name = "current", value = "当前页码", required = true*)* @PathVariable Long current,  
 @ApiParam*(*name = "limit", value = "每页记录数", required = true*)* @PathVariable Long limit,  
 @ApiParam*(*name = "teacherQuery", value = "查询对象", required = false*)* @RequestBody*(*required = false*)* TeacherQuery teacherQuery*) {*//@RequestBody(required = false) 必须post提交 填充数据可以为空  
  
 //动态构建查询条件  
 QueryWrapper*<*EduTeacher*>* queryWrapper = new QueryWrapper*<>()*;  
  
  
 String name = teacherQuery.getName*()*;  
 String leave = String.*valueOf(*teacherQuery.getLevel*())*;  
 String begin = teacherQuery.getBegin*()*;  
 String end = teacherQuery.getEnd*()*;  
  
 if *(*Strings.*isNotEmpty(*name*)) {* queryWrapper.like*(*"name", name*)*;//模糊匹配  
 *}* if *(*Strings.*isNotEmpty(*leave*)) {* queryWrapper.eq*(*"level", leave*)*;  
 *}* if *(*Strings.*isNotEmpty(*begin*)) {* queryWrapper.ge*(*"gmt\_create", begin*)*;  
 *}* if *(*Strings.*isNotEmpty(*end*)) {* queryWrapper.le*(*"gmt\_create", end*)*;  
 *}* //创建page对象  
 Page*<*EduTeacher*>* pageTeacher = new Page*<*EduTeacher*>(*current, limit*)*;  
  
 //分页查询 底层插件封装  
 IPage*<*EduTeacher*>* page = eduTeacherService.page*(*pageTeacher, queryWrapper*)*;  
  
 return R.*ok()* .data*(*"total", page.getTotal*())* .data*(*"rows", page.getRecords*())*;  
*}*

#### 2、service

接口

package com.atguigu.eduservice.service;

public interface EduTeacherService extends IService*<*EduTeacher*> {* void pageQuery*(*Page*<*EduTeacher*>* pageParam, TeacherQuery teacherQuery*)*;  
*}*

实现

@Service

public class EduTeacherServiceImpl extends ServiceImpl*<*EduTeacherMapper, EduTeacher*>* implements EduTeacherService *{* @Override  
 public void pageQuery*(*Page*<*EduTeacher*>* pageParam, TeacherQuery teacherQuery*) {* //动态构建查询条件  
 QueryWrapper*<*EduTeacher*>* queryWrapper = new QueryWrapper*<>()*;  
  
  
 String name = teacherQuery.getName*()*;  
 String leave = String.*valueOf(*teacherQuery.getLevel*())*;  
 String begin = teacherQuery.getBegin*()*;  
 String end = teacherQuery.getEnd*()*;  
  
 if *(*Strings.*isNotEmpty(*name*)) {* queryWrapper.like*(*"name", name*)*;//模糊匹配  
 *}* if *(*Strings.*isNotEmpty(*leave*)) {* queryWrapper.eq*(*"level", leave*)*;  
 *}* if *(*Strings.*isNotEmpty(*begin*)) {* queryWrapper.ge*(*"gmt\_create", begin*)*;  
 *}* if *(*Strings.*isNotEmpty(*end*)) {* queryWrapper.le*(*"gmt\_create", end*)*;  
 *}* //分页查询 底层插件封装  
 IPage*<*EduTeacher*>* page = this.page*(*pageParam, queryWrapper*)*;  
 *}  
}*

#### 3、controller

TeacherAdminController中修改 pageList方法：

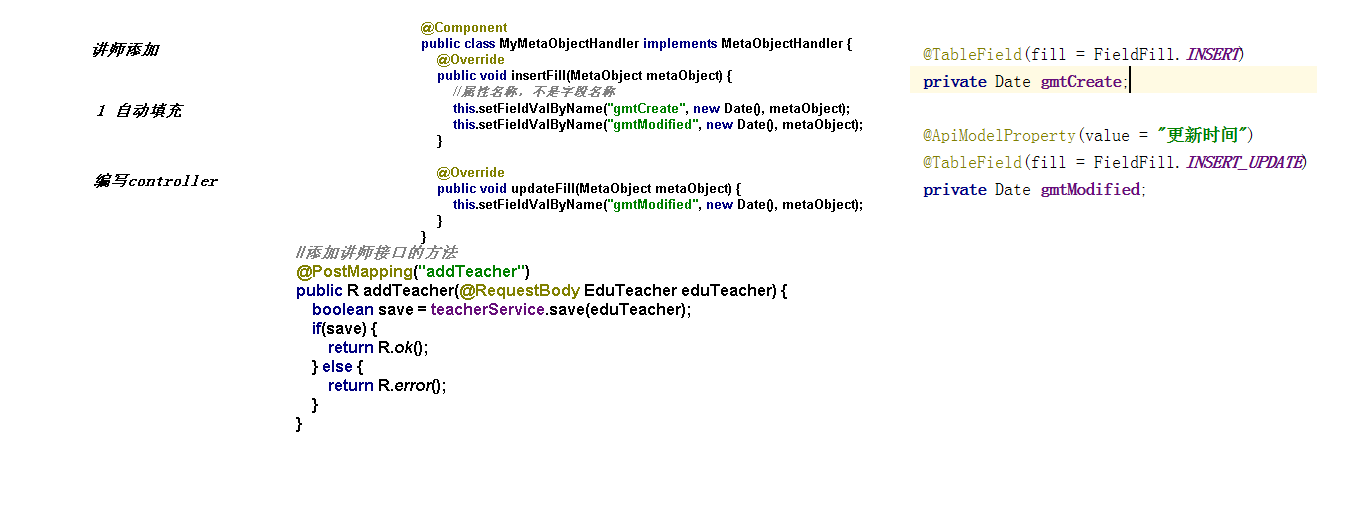
增加参数TeacherQuery teacherQuery，非必选

@ApiOperation*(*value = "条件分页讲师列表"*)*

@PostMapping*(*"pageTeacherCondition/{current}/{limit}"*)*public R pageTeacherCondition*(* @ApiParam*(*name = "current", value = "当前页码", required = true*)* @PathVariable Long current,  
 @ApiParam*(*name = "limit", value = "每页记录数", required = true*)* @PathVariable Long limit,  
 @ApiParam*(*name = "teacherQuery", value = "查询对象", required = false *)* @RequestBody*(*required = false*)* TeacherQuery teacherQuery*) {*//@RequestBody(required = false) 必须post提交 填充数据可以为空  
  
 //创建page对象  
 Page*<*EduTeacher*>* page = new Page*<*EduTeacher*>(*current, limit*)*;  
  
 eduTeacherService.pageQuery*(*page, teacherQuery*)*;  
 return R.*ok()* .data*(*"total", page.getTotal*())* .data*(*"rows", page.getRecords*())*;  
*}*

#### 4、Swagger中测试

## 新增和修改讲师接口开发



### 一、自动填充封装

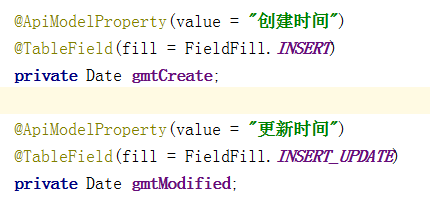
#### 1、在service-base模块中添加

**创建包handler，创建自动填充类 MyMetaObjectHandler**

@Component

public class MyMetaObjectHandler implements MetaObjectHandler *{* @Override  
 public void insertFill*(*MetaObject metaObject*) {* this.setFieldValByName*(*"gmtCreate", new Date*()*, metaObject*)*;  
 this.setFieldValByName*(*"gmtModified", new Date*()*, metaObject*)*;  
 *}* @Override  
 public void updateFill*(*MetaObject metaObject*) {* this.setFieldValByName*(*"gmtModified", new Date*()*, metaObject*)*;  
 *}  
}*

#### 2、在实体类添加自动填充注解



### 二、controller方法定义

#### 1、新增

@ApiOperation*(*value = "新增讲师"*)*

@PostMapping*(*"addTeacher"*)*public R save*(* @ApiParam*(*name = "teacher", value = "讲师对象", required = true*)* @RequestBody EduTeacher eduTeacher*) {* boolean save = eduTeacherService.save*(*eduTeacher*)*;  
 if *(*save*) {* return R.*ok()*;  
 *}* else *{* return R.*error()*;  
 *}  
}*



#### 2、根据id查询

@ApiOperation*(*value = "根据ID查询讲师"*)、*

@GetMapping*(*"getTeacher/{id}"*)*public R queryById*(* @ApiParam*(*name = "id", value = "讲师ID", required = true*)* @PathVariable String id*) {* EduTeacher teacher = eduTeacherService.getById*(*id*)*;  
  
 return R.*ok()*.data*(*"teacher", teacher*)*;  
*}*

#### 3、根据id修改

@ApiOperation*(*value = "根据ID修改讲师"*)*

@PostMapping*(*"updateTeacher/{id}"*)*public R updateById*(* @ApiParam*(*name = "teacher", value = "讲师对象包含id", required = true*)* @PathVariable EduTeacher eduTeacher*) {* boolean b = eduTeacherService.updateById*(*eduTeacher*)*;  
  
 if *(*b*) {* return R.*ok()*;  
 *}* else *{* return R.*error()*;  
 *}  
}*

Put提交需要注意id的处理：

@ApiOperation*(*value = "根据ID修改讲师"*)*@PutMapping*(*"{id}"*)*public R updateById*(* @ApiParam*(*name = "id", value = "讲师ID", required = true*)* @PathVariable String id,  
 @ApiParam*(*name = "teacher", value = "讲师对象", required = true*)* @RequestBody Teacher teacher*) {* teacher.setId*(*id*)*;  
 teacherService.updateById*(*teacher*)*;  
 return R.*ok()*;  
*}*

## 统一异常处理

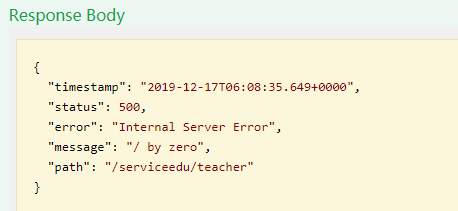


### 一、什么是统一异常处理

#### 1、制造异常

除以0

int a = 10/0;



#### 2、什么是统一异常处理

我们想让异常结果也显示为统一的返回结果对象，并且统一处理系统的异常信息，那么需要统一异常处理

### 二、统一异常处理

#### 1、创建统一异常处理器

在service-base中创建统一异常处理类GlobalExceptionHandler.java：

*/\*\**

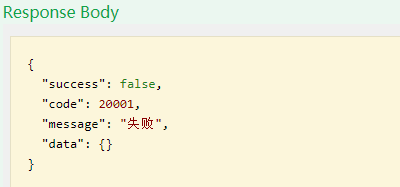
*\* 统一异常处理类  
 \*/*@ControllerAdvice  
public class GlobalExceptionHandler *{* @ExceptionHandler*(*Exception.class*)* @ResponseBody  
 public R error*(*Exception e*) {* e.printStackTrace*()*;  
 return R.*error()*;  
 *}  
}*

将service中以下依赖移动到service-base中【防止依赖冲突】

*<*dependency*>  
 <*groupId*>*com.atguigu*</*groupId*>  
 <*artifactId*>*common-utils*</*artifactId*>  
 <*version*>*0.0.1-SNAPSHOT*</*version*>  
</*dependency*>*

#### 2、测试

返回统一错误结果



### 三、处理特定异常

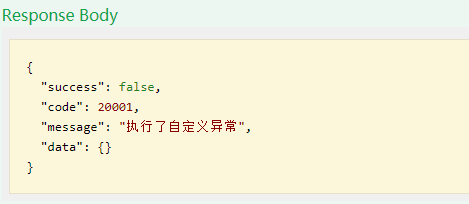
#### 1、添加异常处理方法

GlobalExceptionHandler.java中添加

@ExceptionHandler*(*ArithmeticException.class*)*

@ResponseBody  
public R error*(*ArithmeticException e*) {* e.printStackTrace*()*;  
 return R.*error()*.message*(*"执行了自定义异常"*)*;  
*}*

#### 2、测试



### 四、自定义异常

#### 1、创建自定义异常类

@Data

@AllArgsConstructor  
@NoArgsConstructor  
public class GuliException extends RuntimeException *{* @ApiModelProperty*(*value = "状态码"*)* private Integer code;  
 private String msg;  
  
*}*

#### 2、业务中需要的位置抛出GuliException

try {  
 int a = 10 / 0;  
} catch (Exception e) {  
 throw new GuliException(20001, "出现自定义异常");  
}

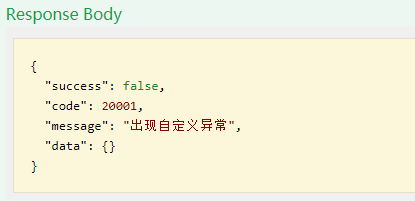
#### 3、添加异常处理方法

GlobalExceptionHandler.java中添加

@ExceptionHandler*(*GuliException.class*)*

@ResponseBody  
public R error*(*GuliException e*) {* e.printStackTrace*()*;  
 return R.*error()*.message*(*e.getMsg*())*.code*(*e.getCode*())*;  
*}*

#### 4、测试



## 07-统一日志处理

### 一、日志

#### 1、配置日志级别

日志记录器（Logger）的行为是分等级的。如下表所示：

分为：OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL【从小到大】

默认情况下，spring boot从控制台打印出来的日志级别只有INFO及以上级别，可以配置日志级别

# 设置日志级别

logging.level.root=WARN

这种方式只能将日志打印在控制台上

### 二、Logback日志

spring boot内部使用Logback作为日志实现的框架。

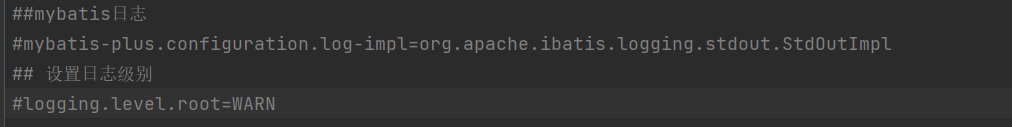
Logback和log4j非常相似，如果你对log4j很熟悉，那对logback很快就会得心应手。

logback相对于log4j的一些优点：<https://blog.csdn.net/caisini_vc/article/details/48551287>



#### 1、配置logback日志

删除application.properties中的日志配置



安装idea彩色日志插件：grep-console

resources 中创建 logback-spring.xml

*<?*xml version="1.0" encoding="UTF-8"*?>*

*<*configuration scan="true" scanPeriod="10 seconds"*>* <!-- 日志级别从低到高分为TRACE < DEBUG < INFO < WARN < ERROR < FATAL，如果设置为WARN，则低于WARN的信息都不会输出 -->  
 <!-- scan:当此属性设置为true时，配置文件如果发生改变，将会被重新加载，默认值为true -->  
 <!-- scanPeriod:设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒。当scan为true时，此属性生效。默认的时间间隔为1分钟。 -->  
 <!-- debug:当此属性设置为true时，将打印出logback内部日志信息，实时查看logback运行状态。默认值为false。 -->  
 *<*contextName*>*logback*</*contextName*>* <!-- name的值是变量的名称，value的值时变量定义的值。通过定义的值会被插入到logger上下文中。定义变量后，可以使“${}”来使用变量。 -->  
 *<*property name="log.path" value="D:/guli\_log/edu"*/>* <!-- 彩色日志 -->  
 <!-- 配置格式变量：CONSOLE\_LOG\_PATTERN 彩色日志格式 -->  
 <!-- magenta:洋红 -->  
 <!-- boldMagenta:粗红-->  
 <!-- cyan:青色 -->  
 <!-- white:白色 -->  
 <!-- magenta:洋红 -->  
 *<*property name="CONSOLE\_LOG\_PATTERN"  
 value="%yellow(%date{yyyy-MM-dd HH:mm:ss}) |%highlight(%-5level) |%blue(%thread) |%blue(%file:%line) |%green(%logger) |%cyan(%msg%n)"*/>* <!--输出到控制台-->  
 *<*appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender"*>* <!--此日志appender是为开发使用，只配置最底级别，控制台输出的日志级别是大于或等于此级别的日志信息-->  
 <!-- 例如：如果此处配置了INFO级别，则后面其他位置即使配置了DEBUG级别的日志，也不会被输出 -->  
 *<*filter class="ch.qos.logback.classic.filter.ThresholdFilter"*>  
 <*level*>*INFO*</*level*>  
 </*filter*>  
 <*encoder*>  
 <*Pattern*>*${CONSOLE\_LOG\_PATTERN}*</*Pattern*>* <!-- 设置字符集 -->  
 *<*charset*>*UTF-8*</*charset*>  
 </*encoder*>  
 </*appender*>* <!--输出到文件-->  
 <!-- 时间滚动输出 level为 INFO 日志 -->  
 *<*appender name="INFO\_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender"*>* <!-- 正在记录的日志文件的路径及文件名 -->  
 *<*file*>*${log.path}/log\_info.log*</*file*>* <!--日志文件输出格式-->  
 *<*encoder*>  
 <*pattern*>*%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n*</*pattern*>  
 <*charset*>*UTF-8*</*charset*>  
 </*encoder*>* <!-- 日志记录器的滚动策略，按日期，按大小记录 -->  
 *<*rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy"*>* <!-- 每天日志归档路径以及格式 -->  
 *<*fileNamePattern*>*${log.path}/info/log-info-%d{yyyy-MM-dd}.%i.log*</*fileNamePattern*>  
 <*timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP"*>  
 <*maxFileSize*>*100MB*</*maxFileSize*>  
 </*timeBasedFileNamingAndTriggeringPolicy*>* <!--日志文件保留天数-->  
 *<*maxHistory*>*15*</*maxHistory*>  
 </*rollingPolicy*>* <!-- 此日志文件只记录info级别的 -->  
 *<*filter class="ch.qos.logback.classic.filter.LevelFilter"*>  
 <*level*>*INFO*</*level*>  
 <*onMatch*>*ACCEPT*</*onMatch*>  
 <*onMismatch*>*DENY*</*onMismatch*>  
 </*filter*>  
 </*appender*>* <!-- 时间滚动输出 level为 WARN 日志 -->  
 *<*appender name="WARN\_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender"*>* <!-- 正在记录的日志文件的路径及文件名 -->  
 *<*file*>*${log.path}/log\_warn.log*</*file*>* <!--日志文件输出格式-->  
 *<*encoder*>  
 <*pattern*>*%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n*</*pattern*>  
 <*charset*>*UTF-8*</*charset*>* <!-- 此处设置字符集 -->  
 *</*encoder*>* <!-- 日志记录器的滚动策略，按日期，按大小记录 -->  
 *<*rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy"*>  
 <*fileNamePattern*>*${log.path}/warn/log-warn-%d{yyyy-MM-dd}.%i.log*</*fileNamePattern*>  
 <*timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP"*>  
 <*maxFileSize*>*100MB*</*maxFileSize*>  
 </*timeBasedFileNamingAndTriggeringPolicy*>* <!--日志文件保留天数-->  
 *<*maxHistory*>*15*</*maxHistory*>  
 </*rollingPolicy*>* <!-- 此日志文件只记录warn级别的 -->  
 *<*filter class="ch.qos.logback.classic.filter.LevelFilter"*>  
 <*level*>*warn*</*level*>  
 <*onMatch*>*ACCEPT*</*onMatch*>  
 <*onMismatch*>*DENY*</*onMismatch*>  
 </*filter*>  
 </*appender*>* <!-- 时间滚动输出 level为 ERROR 日志 -->  
 *<*appender name="ERROR\_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender"*>* <!-- 正在记录的日志文件的路径及文件名 -->  
 *<*file*>*${log.path}/log\_error.log*</*file*>* <!--日志文件输出格式-->  
 *<*encoder*>  
 <*pattern*>*%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n*</*pattern*>  
 <*charset*>*UTF-8*</*charset*>* <!-- 此处设置字符集 -->  
 *</*encoder*>* <!-- 日志记录器的滚动策略，按日期，按大小记录 -->  
 *<*rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy"*>  
 <*fileNamePattern*>*${log.path}/error/log-error-%d{yyyy-MM-dd}.%i.log*</*fileNamePattern*>  
 <*timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP"*>  
 <*maxFileSize*>*100MB*</*maxFileSize*>  
 </*timeBasedFileNamingAndTriggeringPolicy*>* <!--日志文件保留天数-->  
 *<*maxHistory*>*15*</*maxHistory*>  
 </*rollingPolicy*>* <!-- 此日志文件只记录ERROR级别的 -->  
 *<*filter class="ch.qos.logback.classic.filter.LevelFilter"*>  
 <*level*>*ERROR*</*level*>  
 <*onMatch*>*ACCEPT*</*onMatch*>  
 <*onMismatch*>*DENY*</*onMismatch*>  
 </*filter*>  
 </*appender*>* <!--  
 <logger>用来设置某一个包或者具体的某一个类的日志打印级别、以及指定<appender>。  
 <logger>仅有一个name属性，  
 一个可选的level和一个可选的addtivity属性。  
 name:用来指定受此logger约束的某一个包或者具体的某一个类。  
 level:用来设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL 和 OFF，  
 如果未设置此属性，那么当前logger将会继承上级的级别。  
 -->  
 <!--  
 使用mybatis的时候，sql语句是debug下才会打印，而这里我们只配置了info，所以想要查看sql语句的话，有以下两种操作：  
 第一种把<root level="INFO">改成<root level="DEBUG">这样就会打印sql，不过这样日志那边会出现很多其他消息  
 第二种就是单独给mapper下目录配置DEBUG模式，代码如下，这样配置sql语句会打印，其他还是正常DEBUG级别：  
 -->  
 <!--开发环境:打印控制台-->  
 *<*springProfile name="dev"*>* <!--可以输出项目中的debug日志，包括mybatis的sql日志-->  
 *<*logger name="com.guli" level="INFO"*/>* <!--  
 root节点是必选节点，用来指定最基础的日志输出级别，只有一个level属性  
 level:用来设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL 和 OFF，默认是DEBUG  
 可以包含零个或多个appender元素。  
 -->  
 *<*root level="INFO"*>  
 <*appender-ref ref="CONSOLE"*/>  
 <*appender-ref ref="INFO\_FILE"*/>  
 <*appender-ref ref="WARN\_FILE"*/>  
 <*appender-ref ref="ERROR\_FILE"*/>  
 </*root*>  
 </*springProfile*>* <!--生产环境:输出到文件-->  
 *<*springProfile name="pro"*>  
 <*root level="INFO"*>  
 <*appender-ref ref="CONSOLE"*/>  
 <*appender-ref ref="DEBUG\_FILE"*/>  
 <*appender-ref ref="INFO\_FILE"*/>  
 <*appender-ref ref="ERROR\_FILE"*/>  
 <*appender-ref ref="WARN\_FILE"*/>  
 </*root*>  
 </*springProfile*>  
</*configuration*>*

#### 2、将错误日志输出到文件

GlobalExceptionHandler.java 中

类上添加注解

@Slf4j

异常输出语句

*log*.error*(*e.getMessage*())*;

#### 3、将日志堆栈信息输出到文件

定义工具类

guli-framework-common下创建util包，创建ExceptionUtil.java工具类

package com.guli.common.util;

public class ExceptionUtil *{*

public static String getMessage*(*Exception e*) {* StringWriter sw = null;  
 PrintWriter pw = null;  
 try *{* sw = new StringWriter*()*;  
 pw = new PrintWriter*(*sw*)*;  
 // 将出错的栈信息输出到printWriter中  
 e.printStackTrace*(*pw*)*;  
 pw.flush*()*;  
 sw.flush*()*;  
 *}* finally *{* if *(*sw != null*) {* try *{* sw.close*()*;  
 *}* catch *(*IOException e1*) {* e1.printStackTrace*()*;  
 *}  
 }* if *(*pw != null*) {* pw.close*()*;  
 *}  
 }* return sw.toString*()*;  
 *}  
}*

调用

*log*.error*(*ExceptionUtil.*getMessage(*e*))*;

GuliException中创建toString方法

@Override

public String toString*() {* return "GuliException{" +  
 "message=" + this.getMessage*()* +  
 ", code=" + code +  
 '}';  
*}*