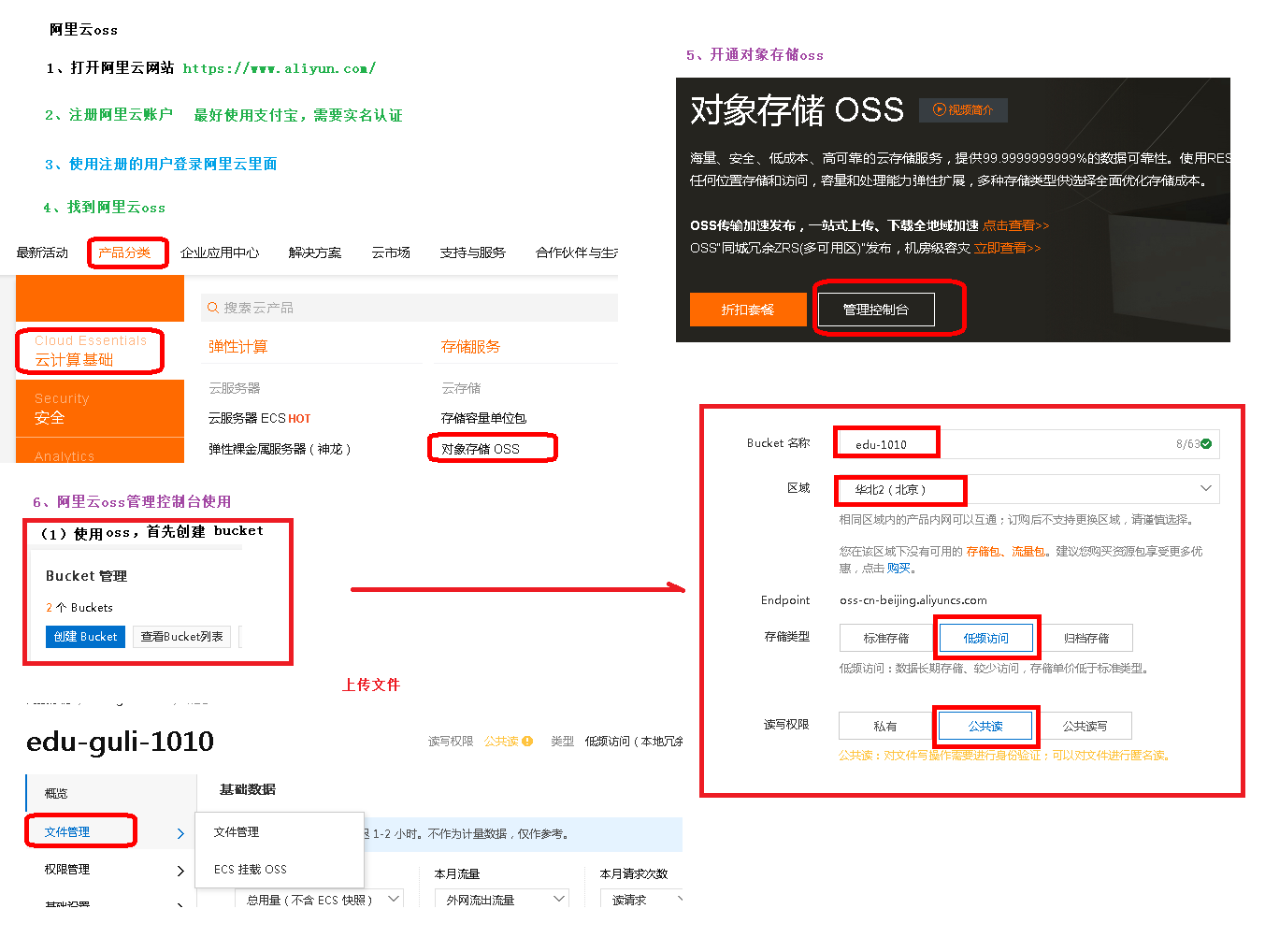
-- 整合阿里云OSS

# 阿里云存储OSS



## 一、对象存储OSS

为了解决海量数据存储与弹性扩容，项目中我们采用云存储的解决方案- 阿里云OSS。

### 1、开通“对象存储OSS”服务

（1）申请阿里云账号

（2）实名认证

（3）开通“对象存储OSS”服务

（4）进入管理控制台

### 2、创建Bucket

选择：标准存储、公共读、不开通



### 3、上传默认头像

创建文件夹avatar，上传默认的用户头像



### 4、创建RAM子用户



## 二、使用SDK





https://help.aliyun.com/document\_detail/32008.html?spm=5176.208357.1107607.21.198d390fbRROsG

### 1、创建Mavaen项目

com.atguigu

aliyun-oss

### 2、pom

<dependencies>

<!--aliyunOSS-->

<dependency>

<groupId>com.aliyun.oss</groupId>

<artifactId>aliyun-sdk-oss</artifactId>

<version>2.8.3</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

</dependencies>

### 3、找到编码时需要用到的常量值

（1）endpoint

（2）bucketName

（3）accessKeyId

（4）accessKeySecret

### 4、测试创建Bucket的连接



package com.atguigu.oss;

public class OSSTest {

// Endpoint以杭州为例，其它Region请按实际情况填写。

String endpoint = "your endpoint";

// 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。

String accessKeyId = "your accessKeyId";

String accessKeySecret = "your accessKeySecret";

String bucketName = "guli-file";

@Test

public void testCreateBucket() {

// 创建OSSClient实例。

OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 创建存储空间。

ossClient.createBucket(bucketName);

// 关闭OSSClient。

ossClient.shutdown();

}

}

### 5、判断存储空间是否存在



@Test

public void testExist() {

// 创建OSSClient实例。

OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

boolean exists = ossClient.doesBucketExist(bucketName);

System.out.println(exists);

// 关闭OSSClient。

ossClient.shutdown();

}

### 6、设置存储空间的访问权限

@Test

public void testAccessControl() {

// 创建OSSClient实例。

OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret);

// 设置存储空间的访问权限为：公共读。

ossClient.setBucketAcl(bucketName, CannedAccessControlList.PublicRead);

// 关闭OSSClient。

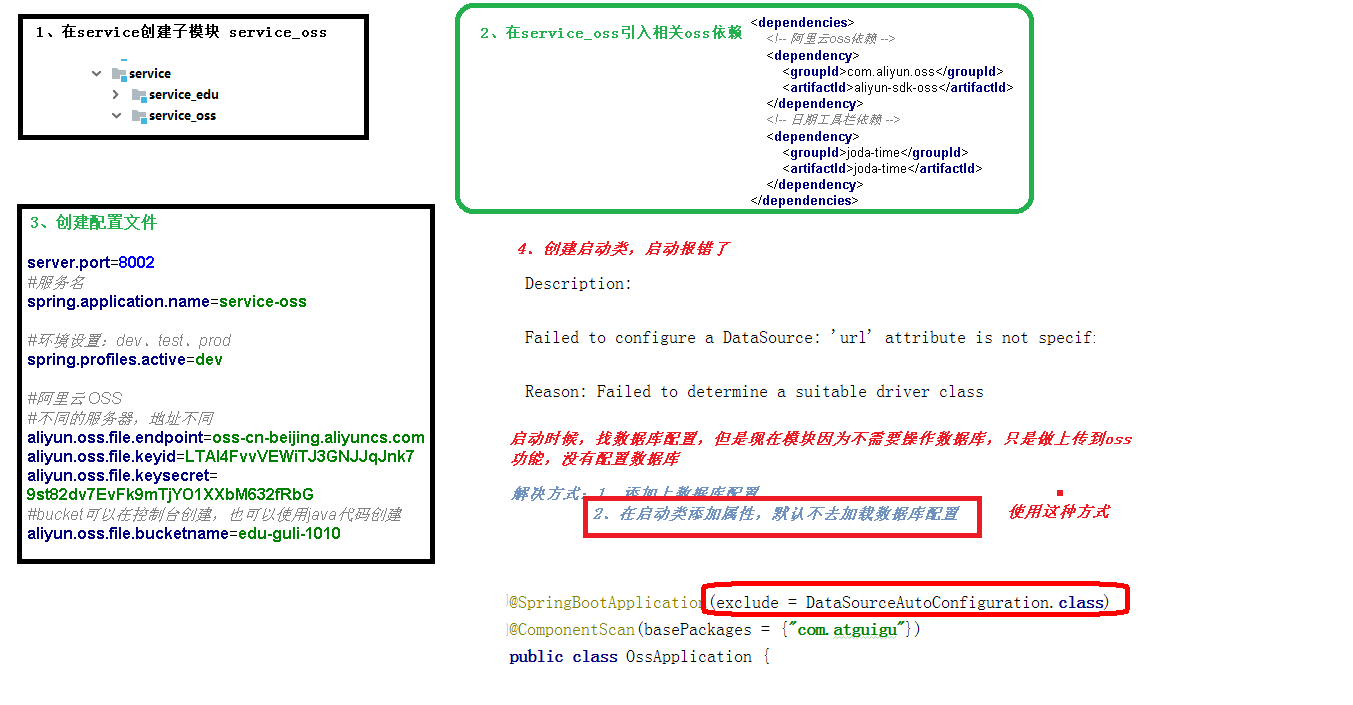
ossClient.shutdown();

}

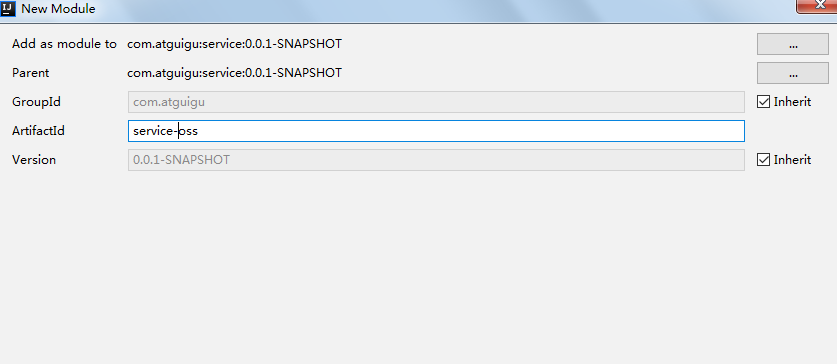


# 后端集成OSS

## 新建云存储微服务



### 1、在service模块下创建子模块service-oss



### 2、配置pom.xml

service-oss上级模块service已经引入service的公共依赖，所以service-oss模块只需引入阿里云oss相关依赖即可，

service父模块已经引入了service-base模块，所以Swagger相关默认已经引入

*<*dependencies*>*

<!-- 阿里云oss依赖 -->  
 *<*dependency*>  
 <*groupId*>*com.aliyun.oss*</*groupId*>  
 <*artifactId*>*aliyun-sdk-oss*</*artifactId*>  
 </*dependency*>* <!-- 日期工具栏依赖 -->  
 *<*dependency*>  
 <*groupId*>*joda-time*</*groupId*>  
 <*artifactId*>*joda-time*</*artifactId*>  
 </*dependency*>  
</*dependencies*>*

### 3、配置application.properties

#服务端口

server.port=8002  
#服务名  
spring.application.name=service-oss  
#环境设置：dev、test、prod  
spring.profiles.active=dev  
#阿里云 OSS  
#不同的服务器，地址不同  
aliyun.oss.file.endpoint=your endpoint  
aliyun.oss.file.keyid=your accessKeyId  
aliyun.oss.file.keysecret=your accessKeySecret  
#bucket可以在控制台创建，也可以使用java代码创建  
aliyun.oss.file.bucketname=guli-file

### 4、logback-spring.xml

### 5、创建启动类

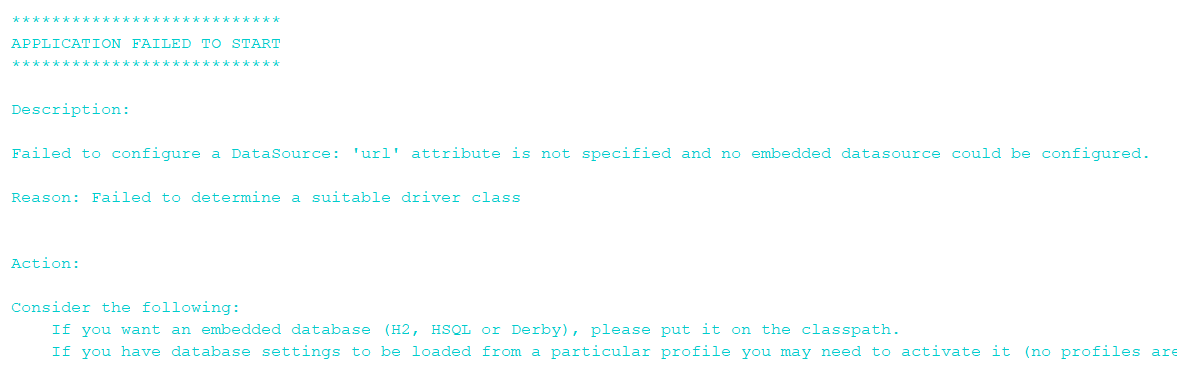
创建OssApplication.java

package com.atguigu.oss;

@SpringBootApplication  
@ComponentScan*(*basePackages = *{*"com.atguigu"*})*public class OssApplication *{* public static void main*(*String*[]* args*) {* SpringApplication.*run(*OssApplication.class, args*)*;  
 *}  
}*

### 6、启动项目

**报错**



spring boot 会默认加载org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration这个类，

而DataSourceAutoConfiguration类使用了@Configuration注解向spring注入了dataSource bean，又因为项目（oss模块）中并没有关于dataSource相关的配置信息，所以当spring创建dataSource bean时因缺少相关的信息就会报错。

**解决办法：**

方法1、在@SpringBootApplication注解上加上exclude，解除自动加载DataSourceAutoConfiguration

@SpringBootApplication*(*exclude = *{*DataSourceAutoConfiguration.class*})*

## 实现文件上传





### 1、从配置文件读取常量

创建常量读取工具类：ConstantPropertiesUtil.java

使用@Value读取application.properties里的配置内容

用spring的 InitializingBean 的 afterPropertiesSet 来初始化配置信息，这个方法将在所有的属性被初始化后调用。

*/\*\**

*\* 常量类，读取配置文件application.properties中的配置  
 \*/*@Component  
//@PropertySource("classpath:application.properties")  
public class ConstantPropertiesUtil implements InitializingBean *{* @Value*(*"${aliyun.oss.file.endpoint}"*)* private String endpoint;  
 @Value*(*"${aliyun.oss.file.keyid}"*)* private String keyId;  
 @Value*(*"${aliyun.oss.file.keysecret}"*)* private String keySecret;  
 @Value*(*"${aliyun.oss.file.bucketname}"*)* private String bucketName;

public static String *END\_POINT*;  
 public static String *ACCESS\_KEY\_ID*;  
 public static String *ACCESS\_KEY\_SECRET*;  
 public static String *BUCKET\_NAME*;  
  
  
 @Override  
 public void afterPropertiesSet*()* throws Exception *{  
 END\_POINT* = endpoint;  
 *ACCESS\_KEY\_ID* = keyId;  
 *ACCESS\_KEY\_SECRET* = keySecret;  
 *BUCKET\_NAME* = bucketName;  
  
 *}  
}*

### 2、文件上传

创建Service接口：FileService.java

public interface FileService {

/\*\*

\* 文件上传至阿里云

\* @param file

\* @return

\*/

String upload(MultipartFile file);

}

实现：FileServiceImpl.java

参考SDK中的：Java->上传文件->简单上传->流式上传->上传文件流



public class FileServiceImpl implements FileService *{*

@Override

public String upload*(*MultipartFile file*) {* //获取阿里云存储相关常量  
 String endPoint = ConstantPropertiesUtil.END\_POINT;  
 String accessKeyId = ConstantPropertiesUtil.ACCESS\_KEY\_ID;  
 String accessKeySecret = ConstantPropertiesUtil.ACCESS\_KEY\_SECRET;  
 String bucketName = ConstantPropertiesUtil.BUCKET\_NAME;  
 String fileHost = ConstantPropertiesUtil.FILE\_HOST;  
 String uploadUrl = null;  
 try *{* //判断oss实例是否存在：如果不存在则创建，如果存在则获取  
 OSSClient ossClient = new OSSClient*(*endPoint, accessKeyId, accessKeySecret*)*;  
 if *(*!ossClient.doesBucketExist*(*bucketName*)) {* //创建bucket  
 ossClient.createBucket*(*bucketName*)*;  
 //设置oss实例的访问权限：公共读  
 ossClient.setBucketAcl*(*bucketName, CannedAccessControlList.PublicRead*)*;  
 *}* //获取上传文件流  
 InputStream inputStream = file.getInputStream*()*;  
 //构建日期路径：avatar/2019/02/26/文件名  
 String filePath = new DateTime*()*.toString*(*"yyyy/MM/dd"*)*;  
 //文件名：uuid.扩展名  
 String original = file.getOriginalFilename*()*;  
 String fileName = UUID.*randomUUID()*.toString*()*;  
 String fileType = original.substring*(*original.lastIndexOf*(*"."*))*;  
 String newName = fileName + fileType;  
 String fileUrl = fileHost + "/" + filePath + "/" + newName;  
 //文件上传至阿里云  
 ossClient.putObject*(*bucketName, fileUrl, inputStream*)*;  
 // 关闭OSSClient。  
 ossClient.shutdown*()*;  
 //获取url地址  
 uploadUrl = "http://" + bucketName + "." + endPoint + "/" + fileUrl;  
 *}* catch *(*IOException e*) {* throw new GuliException*(*ResultCodeEnum.FILE\_UPLOAD\_ERROR*)*;  
 *}* return uploadUrl;  
 *}  
}*

### 3、控制层

创建controller：FileUploadController.java

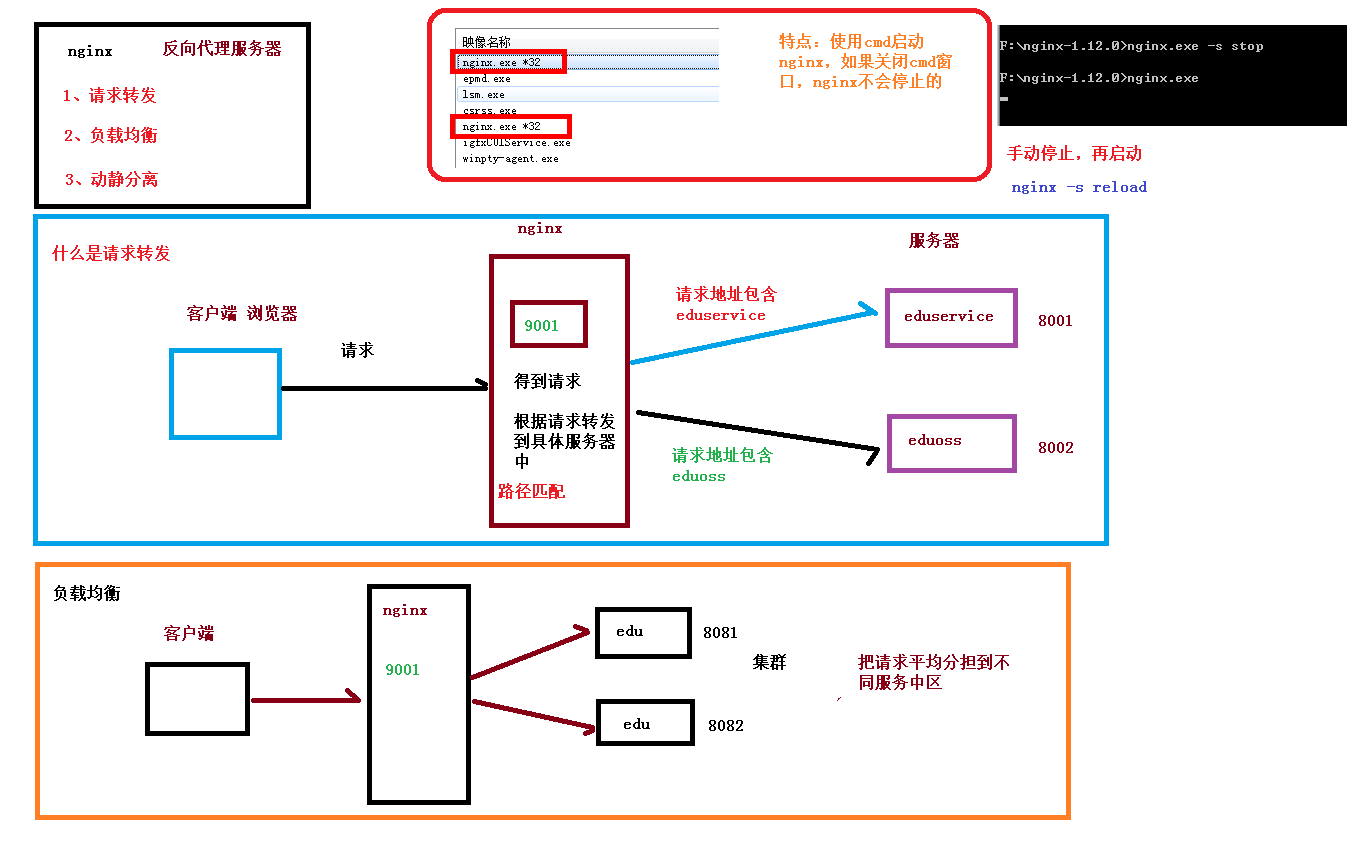
@Api*(*description = "阿里云文件管理"*)*

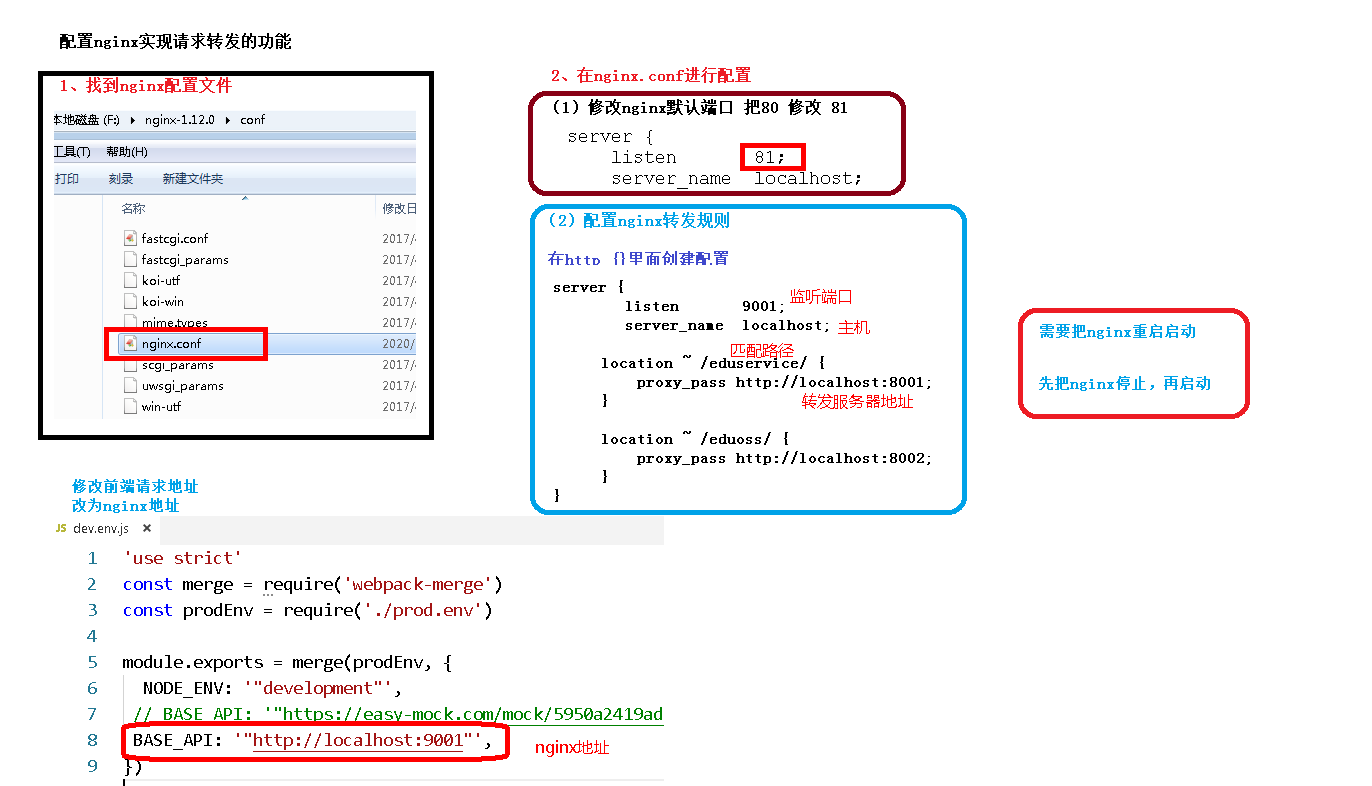
@CrossOrigin //跨域  
@RestController  
@RequestMapping*(*"/admin/oss/file")  
public class FileController {  
 @Autowired  
 private FileService fileService;  
  
 */\*\*  
 \* 文件上传  
 \*  
 \** ***@param*** *file  
 \*/* @ApiOperation*(*value = "文件上传"*)* @PostMapping*(*"upload")  
 public R upload(  
 @ApiParam(name = "file", value = "文件", required = true)  
 @RequestParam("file") MultipartFile file) {  
 String uploadUrl = fileService.upload(file);  
 //返回r对象  
 return R.ok().message("文件上传成功").data("url", uploadUrl);  
 }  
}

### 4、重启oss服务

### 5、Swagger中测试文件上传

### 6、配置nginx反向代理





将接口地址加入nginx配置

# 谷粒学院

server {  
 listen 9001;  
 server\_name localhost;  
 # 路径中包含eduservice转发到8001  
 # ~ 表示正则匹配  
 location ~ /eduservice/ {   
 proxy\_pass http://localhost:8001;  
 }  
 location ~ /eduOss/ {   
 proxy\_pass http://localhost:8002;  
 }  
}

# 前端整合上传组件



## 一、前端整合图片上传组件

### 1、复制头像上传组件

**从vue-element-admin复制组件：**

vue-element-admin/src/components/ImageCropper

vue-element-admin/src/components/PanThumb

### 2、前端参考实现

src/views/components-demo/avatarUpload.vue

### 3、前端添加文件上传组件

**src/views/edu/teacher/form.vue**

template：

<!-- 讲师头像 -->

<el-form-item label="讲师头像">

    <!-- 头衔缩略图 -->

    <pan-thumb :image="teacher.avatar"/>

    <!-- 文件上传按钮 -->

    <el-button type="primary" icon="el-icon-upload" @click="imagecropperShow=true">更换头像

    </el-button>

    <!--

      v-show：是否显示上传组件

      :key：类似于id，如果一个页面多个图片上传控件，可以做区分

      :url：后台上传的url地址

      @close：关闭上传组件

      @crop-upload-success：上传成功后的回调 -->

    <image-cropper

                  v-show="imagecropperShow"

                  :width="300"

                  :height="300"

                  :key="imagecropperKey"

                  :url="BASE\_API+'/admin/oss/file/upload'"

                  field="file"

                  @close="close"

                  @crop-upload-success="cropSuccess"/>

</el-form-item>

引入组件模块

import ImageCropper from '@/components/ImageCropper'

import PanThumb from '@/components/PanThumb'

### 4、设置默认头像

config/dev.env.js中添加阿里云oss bucket地址

OSS\_PATH: '"https://guli-file.oss-cn-beijing.aliyuncs.com"'

组件中初始化头像默认地址

const defaultForm = {

......,

avatar: process.env.OSS\_PATH + '/avatar/default.jpg'

}

### 5、js脚本实现上传和图片回显

export default {

components: { ImageCropper, PanThumb },//声明组件

data() {

return {

//其它数据模型

......,

BASE\_API: process.env.BASE\_API, // 接口API地址

imagecropperShow: false, // 是否显示上传组件

imagecropperKey: 0 // 上传组件id

}

},

......,

methods: {

//其他函数

......,

// 上传成功后的回调函数

cropSuccess(data) {

console.log(data)

this.imagecropperShow = false

this.teacher.avatar = data.url

// 上传成功后，重新打开上传组件时初始化组件，否则显示上一次的上传结果

this.imagecropperKey = this.imagecropperKey + 1

},

// 关闭上传组件

close() {

this.imagecropperShow = false

// 上传失败后，重新打开上传组件时初始化组件，否则显示上一次的上传结果

this.imagecropperKey = this.imagecropperKey + 1

}

}

}

## 二、测试文件上传

前后端联调

* - EasyExcel导入课程分类

# 01-EasyExcel读写Excel的基本使用

## 一、Excel导入导出的应用场景

1、数据导入：减轻录入工作量

2、数据导出：统计信息归档

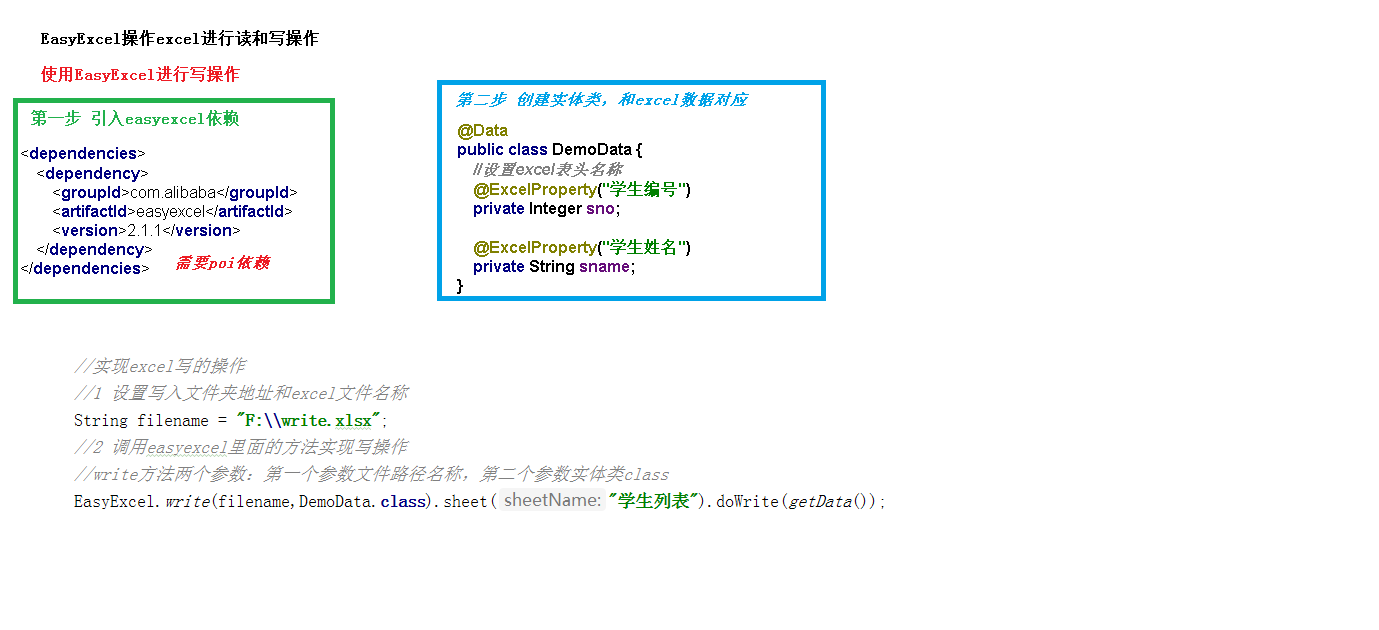
3、数据传输：异构系统之间数据传输

## 二、EasyExcel简介

### 1、EasyExcel特点

* Java领域解析、生成Excel比较有名的框架有Apache poi、jxl等。但他们都存在一个严重的问题就是非常的耗内存。如果你的系统并发量不大的话可能还行，但是一旦并发上来后一定会OOM或者JVM频繁的full gc。
* EasyExcel是阿里巴巴开源的一个excel处理框架，**以使用简单、节省内存著称**。EasyExcel能大大减少占用内存的主要原因是在解析Excel时没有将文件数据一次性全部加载到内存中，而是从磁盘上一行行读取数据，逐个解析。
* EasyExcel采用一行一行的解析模式，并将一行的解析结果以观察者的模式通知处理（AnalysisEventListener）。

# 02-Excel写



## 一、创建项目，实现EasyExcel对Excel写操作

### 1、创建一个普通的maven项目

项目名：excel-easydemo

### 2、pom中引入xml相关依赖

*<*dependencies*>*

<!-- https://mvnrepository.com/artifact/com.alibaba/easyexcel -->  
 *<*dependency*>  
 <*groupId*>*com.alibaba*</*groupId*>  
 <*artifactId*>*easyexcel*</*artifactId*>  
 <*version*>*2.1.1*</*version*>  
 </*dependency*>  
</*dependencies*>*

### 3、创建实体类

**设置表头和添加的数据字段**

import com.alibaba.excel.annotation.ExcelProperty;

import lombok.Data;  
  
//设置表头和添加的数据字段  
@Data  
public class DemoData *{* //设置表头名称  
 @ExcelProperty*(*"学生编号"*)* private int sno;  
  
 //设置表头名称  
 @ExcelProperty*(*"学生姓名"*)* private String sname;  
   
*}*

### 4 、实现写操作

**（1）创建方法循环设置要添加到Excel的数据**

 //循环设置要添加的数据，最终封装到list集合中

private static List*<*DemoData*>* data*() {* List*<*DemoData*>* list = new ArrayList*<*DemoData*>()*;  
 for *(*int i = 0; i < 10; i++*) {* DemoData data = new DemoData*()*;  
 data.setSno*(*i*)*;  
 data.setSname*(*"张三" + i*)*;  
 list.add*(*data*)*;  
 *}* return list;  
*}*

**（2）实现最终的添加操作（写法一）**

public static void main*(*String*[]* args*)* throws Exception *{*

// 写法1  
 String fileName = "F:\\11.xlsx";  
 // 这里 需要指定写用哪个class去写，然后写到第一个sheet，名字为模板 然后文件流会自动关闭  
 // 如果这里想使用03 则 传入excelType参数即可  
 EasyExcel.*write(*fileName, DemoData.class*)*.sheet*(*"写入方法一"*)*.doWrite*(data())*;  
*}*

**（3）实现最终的添加操作（写法二）**

public static void main*(*String*[]* args*)* throws Exception *{*

// 写法2，方法二需要手动关闭流  
 String fileName = "F:\\112.xlsx";  
 // 这里 需要指定写用哪个class去写  
 ExcelWriter excelWriter = EasyExcel.*write(*fileName, DemoData.class*)*.build*()*;  
 WriteSheet writeSheet = EasyExcel.*writerSheet(*"写入方法二"*)*.build*()*;  
 excelWriter.write*(data()*, writeSheet*)*;  
 /// 千万别忘记finish 会帮忙关闭流  
 excelWriter.finish*()*;  
*}*

# Excel读



## 一、实现EasyExcel对Excel读操作

### 1、创建实体类

@Data

public class ReadData *{* //设置列对应的属性  
 @ExcelProperty*(*index = 0*)* private int sid;  
  
 //设置列对应的属性  
 @ExcelProperty*(*index = 1*)* private String sname;  
*}*

### 2、创建读取操作的监听器

public class ExcelListener extends AnalysisEventListener*<*ReadData*> {*

//创建list集合封装最终的数据  
 List*<*ReadData*>* list = new ArrayList*<*ReadData*>()*;  
  
 //一行一行去读取excle内容  
 @Override  
 public void invoke*(*ReadData user, AnalysisContext analysisContext*) {* System.*out*.println*(*"\*\*\*" + user*)*;  
 list.add*(*user*)*;  
 *}* //读取excel表头信息  
 @Override  
 public void invokeHeadMap*(*Map*<*Integer, String*>* headMap, AnalysisContext context*) {* System.*out*.println*(*"表头信息：" + headMap*)*;  
 *}* //读取完成后执行  
 @Override  
 public void doAfterAllAnalysed*(*AnalysisContext analysisContext*) {*

System.*out*.println*(*"read over..."*)*; *}  
  
}*

### 3、调用实现最终的读取

public static void main*(*String*[]* args*)* throws Exception *{*

// 写法1：  
 String fileName = "F:\\01.xlsx";  
 // 这里 需要指定读用哪个class去读，然后读取第一个sheet 文件流会自动关闭  
 EasyExcel.*read(*fileName, ReadData.class, new ExcelListener*())*.sheet*()*.doRead*()*;

// 写法2：  
 InputStream in = new BufferedInputStream*(*new FileInputStream*(*"F:\\01.xlsx"*))*;  
 ExcelReader excelReader = EasyExcel.read*(*in, ReadData.class, new ExcelListener()).build*()*;  
 ReadSheet readSheet = EasyExcel.*readSheet(*0*)*.build*()*;  
 excelReader.read*(*readSheet*)*;  
 // 这里千万别忘记关闭，读的时候会创建临时文件，到时磁盘会崩的  
 excelReader.finish*()*;  
*}*

# 课程分类管理接口

## 一、添加依赖

**1、service-edu模块配置依赖**

*<*dependencies*>*

<!-- https://mvnrepository.com/artifact/com.alibaba/easyexcel -->  
 *<*dependency*>  
 <*groupId*>*com.alibaba*</*groupId*>  
 <*artifactId*>*easyexcel*</*artifactId*>  
 <*version*>*2.1.1*</*version*>  
 </*dependency*>  
</*dependencies*>*

## 二、业务处理

### 1、SubjectAdminController

@Api*(*description = "课程分类管理"*)*

@CrossOrigin //跨域  
@RestController  
@RequestMapping*(*"/eduservice/subject"*)*public class SubjectAdminController *{* @Autowired  
 private SubjectService subjectService;  
  
 //添加课程分类  
 @ApiOperation*(*value = "Excel批量导入"*)* @PostMapping*(*"addSubject"*)* public R addSubject*(*MultipartFile file*) {* //1 获取上传的excel文件 MultipartFile  
 //返回错误提示信息  
 subjectService.importSubjectData*(*file, subjectService*)*;  
 //判断返回集合是否为空  
 return R.ok*()*;  
 *}  
}*

**2、创建和Excel对应的实体类**

@Data

public class ExcelSubjectData *{* @ExcelProperty*(*index = 0*)* private int oneSubjectName;  
 @ExcelProperty*(*index = 1*)* private String twoSubjectName;  
*}*

**3、SubjectService**

**（1）接口**

void batchImport(MultipartFile file);

**（2）实现类**

//添加课程分类

//poi读取excel内容  
 @Override  
 public void importSubjectData*(*MultipartFile file, EduSubjectService subjectService*) {* try *{* //1 获取文件输入流  
 InputStream inputStream = file.getInputStream*()*;  
 // 这里 需要指定读用哪个class去读，然后读取第一个sheet 文件流会自动关闭  
 EasyExcel.read*(*inputStream, ExcelSubjectData.class, new SubjectExcelListener*(*subjectService*))*.sheet*()*.doRead*()*;  
 *}* catch *(*Exception e*) {* e.printStackTrace*()*;  
 throw new GuliException*(*20002, "添加课程分类失败"*)*;  
 *}  
 }*

**4、创建读取Excel监听器**

public class SubjectExcelListener extends

AnalysisEventListener<ExcelSubjectData> *{*

public EduSubjectService subjectService;  
  
 public SubjectExcelListener*() {  
 }* //创建有参数构造，传递subjectService用于操作数据库  
 public SubjectExcelListener*(*EduSubjectService subjectService*) {* this.subjectService = subjectService;  
 *}* //一行一行去读取excle内容  
 @Override  
 public void invoke*(*ExcelSubjectData user, AnalysisContext analysisContext*) {* if *(*user == null*) {* throw new GuliException*(*20001, "添加失败"*)*;  
 *}* //添加一级分类  
 EduSubject existOneSubject = this.existOneSubject*(*subjectService, user.getOneSubjectName*())*;  
 if *(*existOneSubject == null*) {*//没有相同的  
 existOneSubject = new EduSubject*()*;  
 existOneSubject.setTitle*(*user.getOneSubjectName*())*;  
 existOneSubject.setParentId*(*"0"*)*;  
 subjectService.save*(*existOneSubject*)*;  
 *}* //获取一级分类id值  
 String pid = existOneSubject.getId*()*;  
  
 //添加二级分类  
 EduSubject existTwoSubject = this.existTwoSubject*(*subjectService, user.getTwoSubjectName*()*, pid*)*;  
 if *(*existTwoSubject == null*) {* existTwoSubject = new EduSubject*()*;  
 existTwoSubject.setTitle*(*user.getTwoSubjectName*())*;  
 existTwoSubject.setParentId*(*pid*)*;  
 subjectService.save*(*existTwoSubject*)*;  
 *}  
 }* //读取excel表头信息  
 @Override  
 public void invokeHeadMap*(*Map*<*Integer, String*>* headMap, AnalysisContext context*) {* System.*out*.println*(*"表头信息：" + headMap*)*;  
 *}* //读取完成后执行  
 @Override  
 public void doAfterAllAnalysed*(*AnalysisContext analysisContext*) {  
 }* //判断一级分类是否重复  
 private EduSubject existTwoSubject*(*EduSubjectService subjectService, String name, String pid*) {* QueryWrapper*<*EduSubject*>* wrapper = new QueryWrapper*<>()*;  
 wrapper.eq*(*"title", name*)*;  
 wrapper.eq*(*"parent\_id", pid*)*;  
 EduSubject eduSubject = subjectService.getOne*(*wrapper*)*;  
 return eduSubject;  
 *}* //判断一级分类是否重复  
 private EduSubject existOneSubject*(*EduSubjectService subjectService, String name*) {* QueryWrapper*<*EduSubject*>* wrapper = new QueryWrapper*<>()*;  
 wrapper.eq*(*"title", name*)*;  
 wrapper.eq*(*"parent\_id", "0"*)*;  
 EduSubject eduSubject = subjectService.getOne*(*wrapper*)*;  
 return eduSubject;  
 *}  
}*

# 前端页面的实现

## 一、Excel模板

### 1、编辑Excel模板

### 2、将文件上传至阿里云OSS



## 二、配置路由

### 1、添加路由

// 课程分类管理

{

path: '/edu/subject',

component: Layout,

redirect: '/edu/subject/list',

name: 'Subject',

meta: { title: '课程分类管理', icon: 'nested' },

children: [

{

path: 'list',

name: 'EduSubjectList',

component: () => import('@/views/edu/subject/list'),

meta: { title: '课程分类列表' }

},

{

path: 'import',

name: 'EduSubjectImport',

component: () => import('@/views/edu/subject/import'),

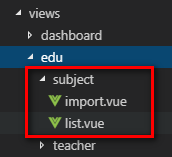
meta: { title: '导入课程分类' }

}

]

},

### 2、添加vue组件



## 三、表单组件import.vue

### 1、js定义数据

<script>

export default {

data() {

return {

BASE\_API: process.env.BASE\_API, // 接口API地址

OSS\_PATH: process.env.OSS\_PATH, // 阿里云OSS地址

fileUploadBtnText: '上传到服务器', // 按钮文字

importBtnDisabled: false, // 按钮是否禁用,

loading: false

}

}

}

</script>

### 2、template

<template>

<div class="app-container">

<el-form label-width="120px">

<el-form-item label="信息描述">

<el-tag type="info">excel模版说明</el-tag>

<el-tag>

<i class="el-icon-download"/>

<a :href="OSS\_PATH + '/excel/%E8%AF%BE%E7%A8%8B%E5%88%86%E7%B1%BB%E5%88%97%E8%A1%A8%E6%A8%A1%E6%9D%BF.xls'">点击下载模版</a>

</el-tag>

</el-form-item>

<el-form-item label="选择Excel">

<el-upload

ref="upload"

:auto-upload="false"

:on-success="fileUploadSuccess"

:on-error="fileUploadError"

:disabled="importBtnDisabled"

:limit="1"

:action="BASE\_API+'/admin/edu/subject/import'"

name="file"

accept="application/vnd.ms-excel">

<el-button slot="trigger" size="small" type="primary">选取文件</el-button>

<el-button

:loading="loading"

style="margin-left: 10px;"

size="small"

type="success"

@click="submitUpload">{{ fileUploadBtnText }}</el-button>

</el-upload>

</el-form-item>

</el-form>

</div>

</template>

### 3、js上传方法

methods: {

    //点击按钮上传文件

    submitUpload() {

      this.fileUploadBtnText = "正在上传";

      this.importBtnDisabled = true;

      this.loading = true;

      this.$refs.upload.submit();//提交表单  固定写法

    },

    //上传成功

    fileUploadSuccess(*response*) {    },

    //上传失败

    fileUploadError(*response*) { },

  },

### 4、回调函数

    //上传成功

    fileUploadSuccess(*response*) {

      if (*response*.success === true) {

        this.fileUploadBtnText = "导入成功";

        this.loading = false;

        this.$message({

          type: "success",

          message: *response*.message,

        });

        //调整到列表页面

        this.$router.push({ path: "/subject/list" });

      }

    },

    //上传失败

    fileUploadError(*response*) {

      this.fileUploadBtnText = "导入失败";

      this.loading = false;

      this.$message({

        type: "error",

        message: "导入失败",

      });

    },

# 分类列表展示



## 一、前端实现

### 1、参考 views/tree/index.vue

### 2、创建api

api/edu/subject.js

import request from '@/utils/request'

const api\_name = '/admin/edu/subject'

export default {

getNestedTreeList() {

return request({

url: `${api\_name}`,

method: 'get'

})

}

}

### 3、list.vue

 <template>

  <div class="app-container">

    <el-input v-model="filterText" placeholder="Filter keyword" style="margin-bottom:30px;" />

    <el-tree

      ref="subjectTree"

      :data="subjectList"

      :props="defaultProps"

      :filter-node-method="filterNode"

      class="filter-tree"

      default-expand-all

    />

  </div>

</template>

<script>

import subject from '@/api/edu/subject'

export default {

  data() {

    return {

      filterText: "",

      data2: [], //返回分类数据

      defaultProps: {

        children: "children", //属性分类

        label: "label", // 显示标题

      },

  },

  watch: {

    filterText(*val*) {

      this.$refs.subjectTree.filter(*val*)

    }

  },

  created() {

    this.fetchNodeList()

  },

  methods: {

    fetchNodeList() {

      subject.getNestedTreeList().then(*response* *=>* {

        if (*response*.success === true) {

          this.subjectList = *response*.data.items

        }

      })

    },

    filterNode(*value*, *data*) {

      if (!*value*) return true

      return *data*.title.indexOf(*value*) !== -1

    }

  }

}

</script>

## 二、后端实现

### 1、创建vo

@Data

public class SubjectVo *{* private String id;

@JsonProperty*(*"label"*)*//属性转换成json时名称  
 private String title;  
*}*

@Data

@ApiModel*(*"一级分类"*)*public class SubjectNestedVo *{* private String id;

@JsonProperty*(*"label"*)*//属性转换成json时名称  
 private String title;  
 private List*<*SubjectVo*>* children = new ArrayList*<>()*;  
*}*

### 2、创建controller

@ApiOperation*(*value = "嵌套数据列表"*)*

@GetMapping*(*""*)*public R nestedList*() {* List*<*SubjectNestedVo*>* subjectNestedVoList = subjectService.nestedList*()*;  
 return R.*ok()*.data*(*"items", subjectNestedVoList);  
}

### 3、创建service

接口

List<SubjectNestedVo> nestedList();

实现Final

@Override

public List*<*SubjectNestedVo*>* nestedList*() {* //最终要的到的数据列表  
 ArrayList*<*SubjectNestedVo*>* subjectNestedVoArrayList = new ArrayList*<>()*;

//获取一级分类数据记录  
 QueryWrapper*<*Subject*>* queryWrapper = new QueryWrapper*<>()*;  
 queryWrapper.eq*(*"parent\_id", 0*)*;  
 queryWrapper.orderByAsc*(*"sort", "id"*)*;  
 List*<*Subject*>* subjects = baseMapper.selectList*(*queryWrapper);

//获取二级分类数据记录  
 QueryWrapper*<*Subject*>* queryWrapper2 = new QueryWrapper*<>()*;  
 queryWrapper2.ne*(*"parent\_id", 0*)*;  
 queryWrapper2.orderByAsc*(*"sort", "id"*)*;  
 List*<*Subject*>* subSubjects = baseMapper.selectList*(*queryWrapper2);

//填充一级分类vo数据  
 int count = subjects.size*()*;  
 for *(*int i = 0; i < count; i++*) {* Subject subject = subjects.get*(*i*)*;  
 //创建一级类别vo对象  
 SubjectNestedVo subjectNestedVo = new SubjectNestedVo*()*;  
 BeanUtils.*copyProperties(*subject, subjectNestedVo*)*;//对象属性拷贝  
 subjectNestedVoArrayList.add*(*subjectNestedVo*)*;

//填充二级分类vo数据  
 ArrayList*<*SubjectVo*>* subjectVoArrayList = new ArrayList*<>()*;  
 int count2 = subSubjects.size*()*;  
 for *(*int j = 0; j < count2; j++*) {* Subject subSubject = subSubjects.get*(*j*)*;  
 if *(*subject.getId*()*.equals*(*subSubject.getParentId*())) {* //创建二级类别vo对象  
 SubjectVo subjectVo = new SubjectVo*()*;  
 BeanUtils.copyProperties*(*subSubject, subjectVo*)*;  
 subjectVoArrayList.add*(*subjectVo*)*;  
 *}  
 }* subjectNestedVo.setChildren*(*subjectVoArrayList*)*;  
 *}* return subjectNestedVoArrayList;  
*}*

### 拓展：后端resultMap实现封装数据

VO

@Data

public class SubjectTree *{* private String id;  
 private String label;  
  
 */\*\*  
 \* 属性 为null时不输出  
 \*/* @JsonInclude*(*JsonInclude.Include.*NON\_NULL)* private List*<*SubjectTree*>* children;  
*}*

eduSubjectMapper.xml

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

*<!DOCTYPE* mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>  
<*mapper namespace="com.atguigu.eduservice.mapper.EduSubjectMapper"*>  
 <*resultMap type="com.atguigu.eduservice.entity.vo.SubjectTree" id="subjectTree"*>* <!-- 一级菜单映射 -->  
 *<*id property="id" column="pid"*/>* <!--属性映射 -->  
 *<*result property="label" column="ptitle"*/>* <!-- 二级菜单映射 -->  
 *<*collection property="children" ofType="com.atguigu.eduservice.entity.vo.SubjectTree"*>* <!-- ofType每一个元素的类型 -->  
 *<*id property="id" column="id"*/>  
 <*result property="label" column="title"*/>  
 </*collection*>  
 </*resultMap*>  
  
  
 <*select id="getTreeList" resultMap="subjectTree" *>* SELECT p.id AS pid, s.id id, s.parent\_id, p.title AS ptitle, s.title AS title  
 FROM edu\_subject p  
 INNER JOIN edu\_subject s ON p.id = s.parent\_id AND p.parent\_id = 0  
  
 *</*select*>  
</*mapper*>*

Dao

@Mapper

public interface EduSubjectMapper extends BaseMapper*<*EduSubject*> {* List*<*SubjectTree*>* getTreeList*()*;  
*}*

application.properties

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

pom.xml打包xml异常时添加

*<*build*>*

<!-- 如果不添加此节点mybatis的mapper.xml文件都会被漏掉。 -->  
 *<*resources*>  
 <*resource*>  
 <*directory*>*src/main/java*</*directory*>  
 <*includes*>  
 <*include*>*\*\*/\*.properties*</*include*>  
 <*include*>*\*\*/\*.xml*</*include*>  
 </*includes*>  
 <*filtering*>*false*</*filtering*>  
 </*resource*>  
 <*resource*>  
 <*directory*>*src/main/resources*</*directory*>  
 <*includes*>  
 <*include*>*\*\*/\*.properties*</*include*>  
 <*include*>*\*\*/\*.xml*</*include*>  
 </*includes*>  
 <*filtering*>*false*</*filtering*>  
 </*resource*>  
 </*resources*>  
</*build*>*

## 三、优化前端过滤功能【不区分大小写】

filterNode(*value*, *data*) {

      if (!value) return true;

      return data.title.toLowerCase().indexOf(value.toLowerCase()) !== -1;

    },