PUP_CRM_2020 开发笔记

> 偶然在 B 站上看到了 Helen 老师主讲的谷粒在线教育项目课程，惊喜地发现前端居然也用了 vue-element-admin 框架。决定跟随视频敲一遍代码，同步实现 PUP CRM 2020 项目。该课程总计 22 天。涉及内容广泛，除 PUP CRM 2020 所必需的内容外，其他部分仅完成课程代码。
>
> 课程第一天说了一些项目相关的业务背景、技术应用基础知识，重点介绍了 MybatisPlus 框架。建立项目从第 2 天开始。

Day 02 - 后端项目创建（2020-9-7）

一、项目环境搭建

（一）工程结构（TODO）

> 参考视频项目，将 PUP CRM 2020 项目结构搭建好。先这么抄过来，总体架构是这样，随着项目推进，逐步修改。
>
> 蓝色的表示已经建立的。

- **crm：根目录（父工程），管理 ?? 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_sys：系统管理 api 接口服务（字典，字典分类，行政区划，高校，产品，丛书，经销商等基础数据）**
    - service_academy：二级院校管理 api 接口服务
    - service_teacher：教师管理 api 接口服务
    - service_apply：样书管理 api 接口服务
    - service_activity：活动管理 api 接口服务
    - service_flow：教材流向管理 api 接口服务
    - service_report：阿里云oss api 接口服务
    - 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_statistics：统计报表api接口服务
    - service_cms：cms api接口服务
    - service_sms：短信api接口服务
    - service_trade：订单和支付相关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 crm

1、创建 Spring Boot 项目

使用 Spring Initializr 快速初始化一个 Spring Boot 项目，其中，JDK 选择 8。

Group：cn.pup

Artifact：crm

2、删除 src 目录

这个项目是用来做依赖管理的，而且只管理版本号，以便子模块可以按需加载依赖并统一版本，所以可以删除 crm 下的 src 文件夹。

3、配置 SpringBoot 版本

考虑到版本的兼容性，计划所有依赖的版本都严格按照视频项目。其中 Spring Boot 的版本选择为 **2.2.1.RELEASE**。这个需要先用 Spring Initializr 创建后，到 POM 文件中修改。

4、配置 pom 依赖版本号

删除所有的 dependencies 和 properties，将下面的版本参数加入到 POM 文件中。

```
<properties>
  <java.version>1.8</java.version>
  <mybatis-plus.version>3.3.1</mybatis-plus.version>
  <velocity.version>2.0</velocity.version>
  <swagger.version>2.7.0</swagger.version>
  <aliyun.oss.version>3.1.0</aliyun.oss.version>
  <jodatime.version>2.10.1</jodatime.version>
  <commons-fileupload.version>1.3.1</commons-fileupload.version>
  <commons-io.version>2.6</commons-io.version>
  <commons-lang.version>3.9</commons-lang.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-java-sdk-vod.version>2.15.2</aliyun-java-sdk-vod.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>
  <alibaba.easyexcel.version>2.1.1</alibaba.easyexcel.version>
  <apache.xmlbeans.version>3.1.0</apache.xmlbeans.version>
</properties>
```

5、配置 pom 依赖

使用 dependencyManagement 标签加上所需要的依赖，这里只是管理了版本，依赖并未被实际引入。

其中，注释掉的 aliyun-sdk-vod-upload 是不开源项目，后面做特殊处理。

> 这里发现个小问题，就是本地没有引入的版本时会显示为红色，这是因为标签里面的依赖只是管理版本，并未真正安装。在 GuLi 项目中去掉 dependencyManagement 标签后，会真正安装依赖（下载到本地），可以看见 crm 项目中的 POM 就没又红色了，再把 GuLi 的标签复原。

```
```

```xml
<dependencyManagement>
  <dependencies>
    <!--Spring Cloud-->
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>Hoxton.SR1</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>

    <dependency>
      <groupId>com.alibaba.cloud</groupId>
      <artifactId>spring-cloud-alibaba-dependencies</artifactId>
      <version>2.1.0.RELEASE</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>${mybatis-plus.version}</version>
    </dependency>
    <dependency>
      <groupId>com.baomidou</groupId>
      <artifactId>mybatis-plus-generator</artifactId>
      <version>${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>

        <!--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>

        <!--commons-lang3-->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>${commons-lang.version}</version>
        </dependency>

        <!--httpclient-->
        <dependency>
            <groupId>org.apache.httpcomponents</groupId>
            <artifactId>httpclient</artifactId>
            <version>${httpclient.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>${aliyun-java-sdk-core.version}</version>
        </dependency>
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>aliyun-java-sdk-vod</artifactId>
            <version>${aliyun-java-sdk-vod.version}</version>
        </dependency>
        <!--<dependency>-->
        <!--<groupId>com.aliyun</groupId>-->
        <!--<artifactId>aliyun-sdk-vod-upload</artifactId>-->
        <!--<version>${aliyun-sdk-vod-upload.version}</version>-->
        <!--</dependency>-->
```

```
117
118         <!--json-->
119         <dependency>
120           <groupId>com.alibaba</groupId>
121           <artifactId>fastjson</artifactId>
122           <version>${fastjson.version}</version>
123         </dependency>
124         <dependency>
125           <groupId>org.json</groupId>
126           <artifactId>json</artifactId>
127           <version>${json.version}</version>
128         </dependency>
129         <dependency>
130           <groupId>com.google.code.gson</groupId>
131           <artifactId>gson</artifactId>
132           <version>${gson.version}</version>
133         </dependency>
134
135         <dependency>
136           <groupId>commons-dbutils</groupId>
137           <artifactId>commons-dbutils</artifactId>
138           <version>${commons-dbutils.version}</version>
139         </dependency>
140
141         <dependency>
142           <groupId>com.alibaba.otter</groupId>
143           <artifactId>canal.client</artifactId>
144           <version>${canal.client.version}</version>
145         </dependency>
146
147         <dependency>
148           <groupId>com.alibaba</groupId>
149           <artifactId>easyexcel</artifactId>
150           <version>${alibaba.easyexcel.version}</version>
151         </dependency>
152
153         <dependency>
154           <groupId>org.apache.xmlbeans</groupId>
155           <artifactId>xmlbeans</artifactId>
156           <version>${apache.xmlbeans.version}</version>
157         </dependency>
158
159       </dependencies>
160     </dependencyManagement>
```

（三）创建父模块 common

1、创建模块

在 crm 下创建普通 maven 模块

Group： cn.pup

Artifact： common

2、删除src目录

3、配置pom

这次可没有 dependencyManagement 标签了，依赖真实导入到 External Libraries 中。

```
<dependencies>
  <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>

  <!--lombok用来简化实体类：需要安装lombok插件-->
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
  </dependency>

  <!--swagger-->
  <dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
  </dependency>
  <dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
  </dependency>

</dependencies>
```

（四）创建模块 common_util

在common下创建普通 maven 模块

Group： cn.pup

Artifact： common_util

注意：项目路径

（五）创建模块service_base

1、创建模块

在common下创建普通 maven 模块

Group： cn.pup

Artifact： service_base

注意：项目路径

2、配置pom

现在 service_base 模块依赖于 common_util 了。

```xml
<dependencies>
  <dependency>
    <groupId>cn.pup</groupId>
    <artifactId>common_util</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </dependency>
</dependencies>
```

（六）创建父模块 service

1、创建模块

在 crm 下创建普通 maven 模块

Group： cn.pup

Artifact： service

2、删除src目录

3、配置pom

```xml
<dependencies>
  <dependency>
    <groupId>cn.pup</groupId>
    <artifactId>service_base</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </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>
  <dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-generator</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>
```

```
38          <artifactId>springfox-swagger2</artifactId>
39      </dependency>
40      <dependency>
41          <groupId>io.springfox</groupId>
42          <artifactId>springfox-swagger-ui</artifactId>
43      </dependency>
44
45      <!--日期时间工具-->
46      <dependency>
47          <groupId>joda-time</groupId>
48          <artifactId>joda-time</artifactId>
49      </dependency>
50
51      <!--lombok用来简化实体类：需要安装lombok插件-->
52      <dependency>
53          <groupId>org.projectlombok</groupId>
54          <artifactId>lombok</artifactId>
55      </dependency>
56
57      <dependency>
58          <groupId>commons-fileupload</groupId>
59          <artifactId>commons-fileupload</artifactId>
60      </dependency>
61
62      <!--httpclient-->
63      <dependency>
64          <groupId>org.apache.httpcomponents</groupId>
65          <artifactId>httpclient</artifactId>
66      </dependency>
67      <!--commons-io-->
68      <dependency>
69          <groupId>commons-io</groupId>
70          <artifactId>commons-io</artifactId>
71      </dependency>
72
73      <!--json-->
74      <dependency>
75          <groupId>com.alibaba</groupId>
76          <artifactId>fastjson</artifactId>
77      </dependency>
78      <dependency>
79          <groupId>org.json</groupId>
80          <artifactId>json</artifactId>
81      </dependency>
82      <dependency>
83          <groupId>com.google.code.gson</groupId>
84          <artifactId>gson</artifactId>
85      </dependency>
86
87      <dependency>
88          <groupId>org.springframework.boot</groupId>
89          <artifactId>spring-boot-starter-test</artifactId>
90          <scope>test</scope>
91          <exclusions>
92              <exclusion>
93                  <groupId>org.junit.vintage</groupId>
94                  <artifactId>junit-vintage-engine</artifactId>
95              </exclusion>
```

```
        </exclusions>
    </dependency>
</dependencies>
```

（七）创建模块 service_sys

在 service 下创建普通 maven 模块

Group： cn.pup

Artifact： service_sys

注意：项目路径

> 这个模块就是一个可以独立运行的 Spring Boot 项目了

到此为止，项目的基本结构创建完成!

二、创建系统管理微服务

（一）数据库设计（TODO）

1、数据库

创建数据库：pup_crm_2020

2、数据表

- 系统管理模块
- [x] 字典分类表：sys_dictionary_category
- [ ] 字典表
- [ ] ……
- 信息管理模块
- [ ] 二级院校表
- [ ] 教师表
- [ ] 教材流向表
- 样书管理模块
- [ ] 样书表
- [ ] 样书明细表
- 活动管理模块
- [ ] 巡展表
- [ ] 巡展院校表
- [ ] 会议表
- [ ] 会议通讯录表
- 业务管理模块
- [ ] 日报表
- [ ] 周报表
- 查询统计模块
- [ ] ……各种视图?

- 权限管理模块

- [ ] 用户表
- [ ] 角色表
- [ ] 权限表
- [ ] ……

（二）数据库设计规约

> **注意：**数据库设计规约并不是数据库设计的严格规范，根据不同团队的不同要求设计。

本项目参考《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

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(unique key)；普通索引名则为 idx* 字段名(index)。

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。

> 还可以参照《58到家数据库30条军规》。作者：沈剑

（三）Mybatis Plus 代码生成器

> 视频中是先创建，逐步实践后优化，这里则先做准备，最后创建代码生成器，只需要运行一次（但是下次增加了表，还是得运行，回头研究一下，只针对一个表生成的办法）

1、创建 BaseEntity （Super Class）

在 service_base 中创建 BaseEntity 类，里面包含了所有表都有的属性（id，gmtCreate，gmtModified）。

```
package cn.pup.crm.service.base.model;

/**
 * Super Class
 * 为所有的实体类提供公共属性的父类
 * @author Howard.Ge
 * @since 2020-09-07
 */

@Data
@EqualsAndHashCode(callSuper = false)
@Accessors(chain = true)
public class BaseEntity implements Serializable {

    private static final long serialVersionUID=1L;

    @ApiModelProperty(value = "ID")
    @TableId(value = "id", type = IdType.ASSIGN_ID)
    private String id;

    @ApiModelProperty(value = "创建时间")
    @TableField(fill = FieldFill.INSERT)
    private Date gmtCreate;

    @ApiModelProperty(value = "更新时间")
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private Date gmtModified;
}
```

相关注解的说明

- `@Data`

相当于@Getter @Setter @RequiredArgsConstructor @ToString @EqualsAndHashCode这5个注解的合集。

启用 lombok，可以只写属性，其他的 setter 和 getter 等方法都自动生成。

- `@EqualsAndHashCode(callSuper = false)`

1. 此注解会生成equals(Object other) 和 hashCode()方法。
2. 它默认使用非静态，非瞬态的属性
3. 可通过参数exclude排除一些属性
4. 可通过参数of指定仅使用哪些属性
5. 它默认仅使用该类中定义的属性且不调用父类的方法
6. 可通过callSuper=true解决上一点问题。让其生成的方法中调用父类的方法。

- `@Accessors(chain = true)`

1. 使用fluent属性，getter和setter方法的方法名都是属性名，且setter方法返回当前对象
2. 使用chain属性，setter方法返回当前对象

3. 使用prefix属性，getter和setter方法会忽视属性名的指定前缀（遵守驼峰命名）

2、创建代码生成器

在 service_sys 的 test 目录中创建代码生成器 CodeGenerator 类。放在 test 文件夹可以在构建项目时排除掉。

```
package cn.pup.crm.service.sys;

/**
 * 代码生成器
 * @author Howard.Ge
 * @since 2020-09-07
 */

public class CodeGenerator {

    @Test
    public void genCode() {

        String prefix = ""; // 数据库前缀 检查是否需要修改
        String moduleName = "sys"; // 模块名称 检查是否需要修改

        // 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");
        gc.setOutputDir(projectPath + "/src/main/java");
        gc.setAuthor("Howard.Ge");
        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.ASSIGN_ID); //主键策略
        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/pup_crm_2020?serverTimezone=GMT%2B8"); //检查是否需要修改
        dsc.setDriverName("com.mysql.cj.jdbc.Driver");
        dsc.setUsername("root");
        dsc.setPassword("Dashu123");  //检查是否需要修改
        dsc.setDbType(DbType.MYSQL);
        mpg.setDataSource(dsc);

        // 4、包配置
        PackageConfig pc = new PackageConfig();
        pc.setModuleName(moduleName);  //模块名  检查是否需要修改
        pc.setParent("cn.pup.crm.service"); // 父包名 检查是否需要修改
        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.setNaming(NamingStrategy.underline_to_camel); //数据库表映射到实体的命名策略
        strategy.setTablePrefix(moduleName + "_"); //设置表前缀不生成

        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.setLogicDeleteFieldName("is_deleted"); //逻辑删除字段名
        strategy.setEntityBooleanColumnRemoveIsPrefix(true); //去掉布尔值的is_前缀

        //自动填充
        TableFill gmtCreate = new TableFill("gmt_create", FieldFill.INSERT);
        TableFill gmtModified = new TableFill("gmt_modified", FieldFill.INSERT_UPDATE);
        ArrayList<TableFill> tableFills = new ArrayList<>();
        tableFills.add(gmtCreate);
        tableFills.add(gmtModified);
        strategy.setTableFillList(tableFills);

        strategy.setRestControllerStyle(true); //restful api风格控制器
        strategy.setControllerMappingHyphenStyle(true);  //url中驼峰转连字符
        mpg.setStrategy(strategy);

        //设置BaseEntity
        strategy.setSuperEntityClass("cn.pup.crm.service.base.model.BaseEntity");  //提供公共属性的父类，检查是否需要修改
        // 填写BaseEntity中的公共字段
        strategy.setSuperEntityColumns("id", "gmt_create", "gmt_modified");//公共属性，检查是否需要修改

        // 6、执行
        mpg.execute();
    }
}

```

3、执行代码生成器

所有实体类都继承了BaseEntity，XxxServiceImpl 继承了 ServiceImpl 类，并且 Mybatis Plus 为我们在 ServiceImpl 中注入了 baseMapper。其他的参照生成器中的策略检查是否正确。

4、修改 entity 包中的（部分）实体

引入缺少的 @TableField 的包。

> 到目前为止，项目只针对 sys_dictionary_category 一张表进行了生成，计划 Day 02 全部完成后，将系统管理的相关表的 api 接口补充完整。

（四）启动应用程序

1、创建 application.yml 文件

在 service_sys 中的 src\main\resources 下创建 application.yml。

```
server:
  port: 8010 # 服务端口

spring:
  profiles:
```

```yaml
  active: dev # 环境设置
 application:
  name: service-edu # 服务名
 datasource: # mysql数据库连接，注意修改数据库名和密码
  driver-class-name: com.mysql.cj.jdbc.Driver
  url: jdbc:mysql://localhost:3306/pup_crm_2020?
serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=UTF-8&useSSL=false
  username: root
  password: Dashu123

#mybatis日志
mybatis-plus:
 configuration:
  log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
```

2、创建 SpringBoot 配置文件

在 service_base 中创建 cn.pup.crm.service.base.config 包并在其中创建 MybatisPlusConfig 类。

```java
package cn.pup.crm.service.base.config;

@EnableTransactionManagement
@Configuration
@MapperScan("cn.pup.crm.service.*.mapper") // 扫描包，* 就可以包括所有未来新的微服务的 mapper
public class MybatisPlusConfig {

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

3、创建 SpringBoot 启动类

在 cn.pup.crm.service.sys 中创建启动类 ServiceSysApplication，扫描 cn.pup.crm

```java
package cn.pup.crm.service.sys;

@SpringBootApplication
@ComponentScan({"cn.pup.crm"})
public class ServiceSysApplication {

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

4、运行启动类

查看控制台 8110 端口是否成功启动

（五）字典分类列表 API

1、编写字典分类管理接口

因为这个管理接口是面向后端管理员而非前端用户的，所以在 cn.pup.crm.service.edu.controller 包中创建 admin 包，专门供后台管理使用。然后将字典管理的 Controller 移到 admin 包中。

修改 DictionaryCategoryController 的 @RequestMapping 注解中的路径，在前面添加 "/admin"。

```
package cn.pup.crm.service.sys.controller.admin;

@RestController
@RequestMapping("/admin/sys/dictionary-category")
public class DictionaryCategoryController {
    @Autowired
    private DictionaryCategoryService dictionaryCategoryService;

    @GetMapping("list")
    public List<DictionaryCategory> listAll() {
        return dictionaryCategoryService.list();
    }
}

```

2、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.yml中设置

```
#spring:
  jackson: #返回json的全局时间格式
    date-format: yyyy-MM-dd HH:mm:ss
    time-zone: GMT+8
```

所有实体类中的所有的 Date 属性添加数据类型转换，可以覆盖全局配置（包括 BaseEntity 类）

```
@JsonFormat(timezone = "GMT+8", pattern = "yyyy-MM-dd")
```

3、重启程序

访问： http://localhost:8010/admin/sys/dictionary-category/list 查看结果 json 数据，可以看到显示格式和时区都正常了。

> 字典分类还需要有其他的 API，改变一下视频笔记的顺序，先弄 Swagger，然后建立 API 并使用 Swagger 进行测试。

（六）配置 Swagger

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。官网：https://swagger.io/

前后端分离开发模式中，api文档是最好的沟通方式，具有以下优势。

- **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)
- **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)
- **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)
- **可测性** (直接在接口文档上进行测试，以方便理解业务)

前后端分离开发的一般步骤如下：

- 前端工程师编写接口文档（使用swagger2编辑器或其他接口生成工具）
- 交给后端工程师

- 根据swagger文档编写后端接口
- 最终根据生成的swagger文件进行接口联调

1. 添加依赖

在 common 中添加依赖。

```
<!--swagger-->
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
</dependency>
```

2. 创建 Swagger 配置类

在 service_base 中创建 Swagger2Config 类。

```
package cn.pup.crm.service.base.config;

@Configuration
@EnableSwagger2
public class Swagger2Config {

  @Bean
  public Docket webApiConfig(){
    return new Docket(DocumentationType.SWAGGER_2)
        .groupName("webApi")
        .apiInfo(webApiInfo())
        .select()
        //只显示api路径下的页面
        .paths(Predicates.and(PathSelectors.regex("/api/.*")))
        .build();
  }

  @Bean
  public Docket adminApiConfig(){
    return new Docket(DocumentationType.SWAGGER_2)
        .groupName("adminApi")
        .apiInfo(adminApiInfo())
        .select()
        //只显示admin路径下的页面
        .paths(Predicates.and(PathSelectors.regex("/admin/.*")))
        .build();
  }

  private ApiInfo webApiInfo(){
    return new ApiInfoBuilder()
        .title("网站-API文档")
        .description("本文档描述了网站微服务接口定义")
        .version("1.0")
        .contact(new Contact("Howard.Ge", "http://pup.cn", "ghh@pup.cn"))
        .build();
  }

```

```
    private ApiInfo adminApiInfo(){
        return new ApiInfoBuilder()
                .title("后台管理系统-API文档")
                .description("本文档描述了后台管理系统微服务接口定义")
                .version("1.0")
                .contact(new Contact("Howard.Ge", "http://pup.cn", "ghh@pup.cn"))
                .build();
    }
}
```

3. 重启服务查看接口

http://localhost:8010/swagger-ui.html

3. 常见注解

（1）API 模型

entity的实体类中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 = "2010-01-01")
@TableField(fill = FieldFill.INSERT_UPDATE)
private Date gmtModified;
```

（2）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

```
package cn.pup.crm.service.sys.controller.admin;

@Api(description = "字典分类管理")
@RestController
@RequestMapping("/admin/sys/dictionary-category")
public class DictionaryCategoryController {

    @Autowired
    private DictionaryCategoryService dictionaryCategoryService;

    @ApiOperation("所有字典分类列表")
    @GetMapping("list")
    public List<DictionaryCategory> listAll() {
        return dictionaryCategoryService.list();
    }

    @ApiOperation(value = "根据 ID 删除字典分类", notes = "逻辑删除")
    @DeleteMapping("remove/{id}")
    public boolean removeById(@ApiParam(value = "字典分类ID", required = true) @PathVariable String id) {
        return dictionaryCategoryService.removeById(id);
    }

}
```

（七）逻辑删除 API

1、添加删除方法

在 TeacherController 添加 removeById 方法

```
@DeleteMapping("remove/{id}")
public boolean removeById(@PathVariable String id){
  return teacherService.removeById(id);
}
```

2、在 Swagger 页面中测试删除

三、非业务功能统一处理

（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)对数据的操作更加一致、轻松。

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。一般情况下，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：

1. 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"
   }
  ]
 }
}
```

2. 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"
   }
  ]
```

```
    }
  }
```

3. 没有返回数据

```
{
  "success": true,
  "code": 20000,
  "message": "成功",
  "data": {}
}
```

4. 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）定义统一返回结果

1、创建返回码定义枚举类

在 common_util 中创建包：cn.pup.crm.common.base.result，并在其中创建两个类：

- 枚举类（定义返回码）： ResultCodeEnum.java

（代码后补）

- 结果类：R.java

```
package cn.pup.crm.common.base.result;

@Data
@ApiModel(value = "全局统一返回结果")
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>();

    public R(){}

    public static R ok(){
      R r = new R();
      r.setSuccess(ResultCodeEnum.SUCCESS.getSuccess());
      r.setCode(ResultCodeEnum.SUCCESS.getCode());
      r.setMessage(ResultCodeEnum.SUCCESS.getMessage());
      return r;
    }

    public static R error(){
      R r = new R();
      r.setSuccess(ResultCodeEnum.UNKNOWN_REASON.getSuccess());
      r.setCode(ResultCodeEnum.UNKNOWN_REASON.getCode());
      r.setMessage(ResultCodeEnum.UNKNOWN_REASON.getMessage());
      return r;
    }

    public static R setResult(ResultCodeEnum resultCodeEnum){
      R r = new R();
      r.setSuccess(resultCodeEnum.getSuccess());
      r.setCode(resultCodeEnum.getCode());
      r.setMessage(resultCodeEnum.getMessage());
      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;
    }
  }

```

2、修改 Controller 中的返回结果

修改 service_sys 中的 DictionaryCategoryController 的接口方法

```
@ApiOperation("所有字典分类列表")
@GetMapping("list")
public R listAll() {
List<DictionaryCategory> list = dictionaryCategoryService.list();
return R.ok().data("items", list).message("获取字典分类列表成功。");
}

@ApiOperation(value = "根据 ID 删除字典分类", notes = "逻辑删除")
@DeleteMapping("remove/{id}")
public R removeById(@ApiParam(value = "字典分类ID", required = true) @PathVariable String id) {
boolean result = dictionaryCategoryService.removeById(id);
return result ? R.ok().message("删除成功") : R.ok().message("数据不存在");
}
```

3、重启测试

（三）分页

1、分页 Controller 方法

在 DictionaryCategoryController 中添加分页方法

```
@ApiOperation("分页字典分类列表")
@GetMapping("list/{page}/{limit}")
public R listPage(@ApiParam(value = "当前页码", required = true) @PathVariable Long page,
@ApiParam(value = "每页记录数", required = true) @PathVariable Long limit){

Page<DictionaryCategory> pageParam = new Page<>(page, limit);
IPage<DictionaryCategory> pageModel = dictionaryCategoryService.page(pageParam);
List<DictionaryCategory> records = pageModel.getRecords();
long total = pageModel.getTotal();
return  R.ok().data("total", total).data("rows", records);
}
```

2、Swagger 中测试

（四）条件查询

实际上就是每个实体类列表页的筛选表单包含的字段作为条件。采用查询对象的方式，便于前台数据包装。在 service_sys 中创建包 cn.pup.crm.service.sys.entity.vo，里面的查询类命名格式为 `实体名+QueryVo`

根据字典分类名称 name，字典分类代码 code、创建时间 gmtCreate、最后修改时间 gmtModified 查询。

1、创建查询对象

（DictionaryCategoryQueryVo）

```java
package cn.pup.crm.service.sys.entity.vo;

@Data
public class DictionaryCategoryQueryVo implements Serializable {
    private static final long serialVersionUID = 1L;

    private String name;
    private String code;
    private Date gmtCreate;
    private Date gmtModified;
}
```

2、修改 service

接口

```java
package cn.pup.crm.service.sys.service;

public interface DictionaryCategoryService extends IService<DictionaryCategory> {
    IPage<DictionaryCategory> selectPage(Long page, Long limit, DictionaryCategoryQueryVo dictionaryCategoryQueryVo);
}
```

实现

```java
package cn.pup.crm.service.sys.service.impl;

/**
 * 字典分类 服务实现类
 *
 * @author Howard.Ge
 * @since 2020-09-07
 */
@Service
public class DictionaryCategoryServiceImpl extends ServiceImpl<DictionaryCategoryMapper, DictionaryCategory> implements DictionaryCategoryService {

    /**
     * 分页条件查询
     * @param page 当前页码
     * @param limit 每页记录数
     * @param dictionaryCategoryQueryVo 字典分类VO（属性为查询条件）
     * @return
     */
    @Override
    public IPage<DictionaryCategory> selectPage(Long page, Long limit, DictionaryCategoryQueryVo dictionaryCategoryQueryVo) {

        Page<DictionaryCategory> pageParam = new Page<>(page, limit);

        QueryWrapper<DictionaryCategory> queryWrapper = new QueryWrapper<>();
        queryWrapper.orderByAsc("sort");

        if (dictionaryCategoryQueryVo == null) {
            return baseMapper.selectPage(pageParam, queryWrapper);
        }

        String name = dictionaryCategoryQueryVo.getName();
```

```
        String code = dictionaryCategoryQueryVo.getCode();
        String gmtCreateBegin = dictionaryCategoryQueryVo.getGmtCreateBegin();
        String gmtCreateEnd = dictionaryCategoryQueryVo.getGmtCreateEnd();
        String gmtModifiedBegin = dictionaryCategoryQueryVo.getGmtModifiedBegin();
        String gmtModifiedEnd = dictionaryCategoryQueryVo.getGmtModifiedEnd();

        //如果查询的字段为数字，判断是否为空

        // 逐个条件注入
        if (!StringUtils.isEmpty(name)) {
            // 左%会使索引失效
            queryWrapper.like("name", name);
        }
        if (!StringUtils.isEmpty(code)) {
            queryWrapper.like("code", code);
        }
        if (!StringUtils.isEmpty(gmtCreateBegin)) {
            queryWrapper.ge("gmtCreate", gmtCreateBegin);
        }
        if (!StringUtils.isEmpty(gmtCreateEnd)) {
            queryWrapper.le("gmtCreate", gmtCreateEnd);
        }
        if (!StringUtils.isEmpty(gmtModifiedBegin)) {
            queryWrapper.ge("gmtModified", gmtModifiedBegin);
        }
        if (!StringUtils.isEmpty(gmtModifiedEnd)) {
            queryWrapper.le("gmtModified", gmtModifiedEnd);
        }
        return baseMapper.selectPage(pageParam, queryWrapper);

    }
}

```

3、修改 controller

修改 DictionaryCategoryController 中的 listPage 方法：增加参数 DictionaryCategoryQueryVo dictionaryCategoryQueryVo，非必选

dictionaryCategoryService.page 修改成 dictionaryCategoryService.selectPage，并传递teacherQueryVo参数。

```
    @ApiOperation("分页字典分类列表")
    @GetMapping("list/{page}/{limit}")
    public R listPage(@ApiParam(value = "当前页码", required = true) @PathVariable Long page,
                      @ApiParam(value = "每页记录数", required = true) @PathVariable Long limit,
                      @ApiParam(value = "字典分类查询对象") DictionaryCategoryQueryVo dictionaryCategoryQueryVo){

        IPage<DictionaryCategory> pageModel = dictionaryCategoryService.selectPage(page, limit,
dictionaryCategoryQueryVo);
        List<DictionaryCategory> records = pageModel.getRecords();
        long total = pageModel.getTotal();

        return  R.ok().data("total", total).data("rows", records);
    }
```

4、Swagger 中测试

发现只能查英文，不能查中文！需要依次检查下面几个地方：

- application.yml

```
url: jdbc:mysql://localhost:3306/pup_crm_2020?
serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=UTF-8&useSSL=false
```

其中的 `useUnicode=true` 和 `characterEncoding=UTF-8` 不能少，注意大小写。

- 数据库属性

```
CREATE DATABASE pup_crm_2020
CHARACTER SET utf8mb4
COLLATE utf8mb4_general_ci;
```

- idea 的设置

在 settings 中搜索 encoding，统统改成 UTF-8。

（五）自动填充

在 service_base 中创建包：cn.pup.crm.service.base.handler 并在其中创建自动填充处理类：CommonMetaObjectHandler

```
package cn.pup.crm.service.base.handler;

@Component
public class CommonMetaObjectHandler 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);
    }
}
```

（六）定义新增和修改 API

在 service_sys 中新增 controller 方法。

1、新增

新增的时候，id 如果不为空则直接赋值（有重复的会报错），id为空值 null 或者空字符串的时候才会自动生成。自动填充的时间字段不会受传入对象的影响，会依据规则自动填入。

```
    @ApiOperation("新增字典分类")
    @PostMapping("save")
    public R save(@ApiParam(value = "字典分类对象", required = true) @RequestBody DictionaryCategory
dictionaryCategory) {
        boolean result = dictionaryCategoryService.save(dictionaryCategory);
        return result ? R.ok().message("保存成功") : R.error().message("保存失败");
    }
```

2、根据 id 修改

```
    @ApiOperation("更新字典分类")
    @PutMapping("update")
    public R updateById(@ApiParam(value = "字典分类对象", required = true) @RequestBody DictionaryCategory
dictionaryCategory) {
        boolean result = dictionaryCategoryService.updateById(dictionaryCategory);
        if (result) {
            return R.ok().message("修改成功");
        } else {
            return R.error().message("数据不存在");
        }
    }
```

3、根据 id 获取字典分类信息

```
    @ApiOperation("根据id获取字典分类信息")
    @GetMapping("get/{id}")
    public R getById(@ApiParam(value = "字典分类ID", required = true) @PathVariable String id) {
        DictionaryCategory dictionaryCategory = dictionaryCategoryService.getById(id);
        if (dictionaryCategory != null) {
            return R.ok().data("item", dictionaryCategory);
        } else {
            return R.error().message("数据不存在");
        }
    }
```

（七）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

1、创建统一异常处理器

在 service-base 中的 handler 包中，创建统一异常处理类：GlobalExceptionHandler.java：

```
package com.atguigu.guli.service.base.handler;

@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(Exception.class)
    @ResponseBody
    public R error(Exception e){
        e.printStackTrace();
        return R.error();
    }
}
```

2、添加异常处理方法

在 GlobalExceptionHandler.java 中添加下面的异常处理方法。

（1）处理 SQL 语法错误产生的 BadSqlGrammarException 异常。

```
@ExceptionHandler(BadSqlGrammarException.class)
@ResponseBody
public R error(BadSqlGrammarException e){
  e.printStackTrace();
  return R.setResult(ResultCodeEnum.BAD_SQL_GRAMMAR);
}
```

（2）处理由于输入非法的 json 参数产生的 HttpMessageNotReadableException 异常

```
@ExceptionHandler(HttpMessageNotReadableException.class)
@ResponseBody
public R error(HttpMessageNotReadableException e){
  e.printStackTrace();
  return R.setResult(ResultCodeEnum.JSON_PARSE_ERROR);
}
```

3、自定义异常

TODO

（八）统一日志处理

1、什么是日志

通过日志查看程序的运行过程，运行信息，异常信息等

2、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：**FATAL**、**ERROR**、**WARN**、**INFO**、**DEBUG**

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: ERROR
```

这种方式能将ERROR级别以及以上级别的日志打印在控制台上

3、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。

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.yml中的日志配置

安装idea彩色日志插件：grep console

resources 中创建 logback-spring.xml （默认日志的名字，必须是这个名字）

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration  scan="true" scanPeriod="10 seconds">

  <contextName>logback</contextName>

  <property name="log.path" value="D:/Project2020/CRM2020_log/sys" />
```

```
7
8       <!--控制台日志格式：彩色日志-->
9       <!-- magenta:洋红 -->
10      <!-- boldMagenta:粗红-->
11      <!-- cyan:青色 -->
12      <!-- white:白色 -->
13      <!-- magenta:洋红 -->
14      <property name="CONSOLE_LOG_PATTERN"
15              value="%yellow(%date{yyyy-MM-dd HH:mm:ss}) |%highlight(%-5level) |%blue(%thread)
    |%blue(%file:%line) |%green(%logger) |%cyan(%msg%n)"/>
16
17      <!--文件日志格式-->
18      <property name="FILE_LOG_PATTERN"
19              value="%date{yyyy-MM-dd HH:mm:ss} |%-5level |%thread |%file:%line |%logger |%msg%n" />
20
21      <!--编码-->
22      <property name="ENCODING"
23              value="UTF-8" />
24
25      <!--输出到控制台-->
26      <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
27        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
28          <!--日志级别-->
29          <level>DEBUG</level>
30        </filter>
31        <encoder>
32          <!--日志格式-->
33          <Pattern>${CONSOLE_LOG_PATTERN}</Pattern>
34          <!--日志字符集-->
35          <charset>${ENCODING}</charset>
36        </encoder>
37      </appender>
38
39      <!--输出到文件-->
40      <appender name="INFO_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
41        <!--日志过滤器：此日志文件只记录INFO级别的-->
42        <filter class="ch.qos.logback.classic.filter.LevelFilter">
43          <level>INFO</level>
44          <onMatch>ACCEPT</onMatch>
45          <onMismatch>DENY</onMismatch>
46        </filter>
47        <!-- 正在记录的日志文件的路径及文件名 -->
48        <file>${log.path}/log_info.log</file>
49        <encoder>
50          <pattern>${FILE_LOG_PATTERN}</pattern>
51          <charset>${ENCODING}</charset>
52        </encoder>
53        <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
54        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
55          <!-- 每天日志归档路径以及格式 -->
56          <fileNamePattern>${log.path}/info/log-info-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
57          <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
58            <maxFileSize>500MB</maxFileSize>
59          </timeBasedFileNamingAndTriggeringPolicy>
60          <!--日志文件保留天数-->
61          <maxHistory>15</maxHistory>
62        </rollingPolicy>
63      </appender>
```

```
64
65      <appender name="WARN_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
66        <!-- 日志过滤器：此日志文件只记录WARN级别的 -->
67        <filter class="ch.qos.logback.classic.filter.LevelFilter">
68          <level>WARN</level>
69          <onMatch>ACCEPT</onMatch>
70          <onMismatch>DENY</onMismatch>
71        </filter>
72        <!-- 正在记录的日志文件的路径及文件名 -->
73        <file>${log.path}/log_warn.log</file>
74        <encoder>
75          <pattern>${FILE_LOG_PATTERN}</pattern>
76          <charset>${ENCODING}</charset> <!-- 此处设置字符集 -->
77        </encoder>
78        <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
79        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
80          <fileNamePattern>${log.path}/warn/log-warn-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
81          <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
82            <maxFileSize>100MB</maxFileSize>
83          </timeBasedFileNamingAndTriggeringPolicy>
84          <!--日志文件保留天数-->
85          <maxHistory>15</maxHistory>
86        </rollingPolicy>
87      </appender>
88
89      <appender name="ERROR_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
90        <!-- 日志过滤器：此日志文件只记录ERROR级别的 -->
91        <filter class="ch.qos.logback.classic.filter.LevelFilter">
92          <level>ERROR</level>
93          <onMatch>ACCEPT</onMatch>
94          <onMismatch>DENY</onMismatch>
95        </filter>
96        <!-- 正在记录的日志文件的路径及文件名 -->
97        <file>${log.path}/log_error.log</file>
98        <encoder>
99          <pattern>${FILE_LOG_PATTERN}</pattern>
100         <charset>${ENCODING}</charset> <!-- 此处设置字符集 -->
101       </encoder>
102       <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
103       <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
104         <fileNamePattern>${log.path}/error/log-error-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
105         <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
106           <maxFileSize>100MB</maxFileSize>
107         </timeBasedFileNamingAndTriggeringPolicy>
108         <!--日志文件保留天数-->
109         <maxHistory>15</maxHistory>
110       </rollingPolicy>
111     </appender>
112
113     <!--开发环境-->
114     <springProfile name="dev">
115       <!--可以灵活设置此处，从而控制日志的输出-->
116       <root level="INFO">
117         <appender-ref ref="CONSOLE" />
118         <appender-ref ref="INFO_FILE" />
119         <appender-ref ref="WARN_FILE" />
120         <appender-ref ref="ERROR_FILE" />
121       </root>
```

```
    </springProfile>

    <!--生产环境-->
    <springProfile name="pro">
      <root level="ERROR">
        <appender-ref ref="ERROR_FILE" />
      </root>
    </springProfile>

  </configuration>
```

（2）节点说明

- `<property>` ：定义变量
- `<appender>` ：定义日志记录器
- - `<filter>` ：定义日志过滤器
  - `<rollingPolicy>` ：定义滚动策略
- `<springProfile>` ：定义日志适配的环境
- - `<root>` ：根日志记录器

（3）控制日志级别

通过在开发环境设置以下 `<root>` 节点的 level 属性的值，调节日志的级别

```
<!--开发环境-->
<springProfile name="dev">
  <!--可以灵活设置此处，从而控制日志的输出-->
  <root level="DEBUG">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="INFO_FILE" />
    <appender-ref ref="WARN_FILE" />
    <appender-ref ref="ERROR_FILE" />
  </root>
</springProfile>
```

在 controller 的 listAll 方法中输出如下日志，调节日志级别查看日志开启和关闭的效果（需要在 Controller 上加上 `@Slf4j` 注解）

```
log.info("所有讲师列表.....");
```

4、错误日志处理

（1）用日志记录器记录错误日志

在 GlobalExceptionHandler 类中添加注解

```
@Slf4j
```

修改异常输出语句

```
//e.printStackTrace();
log.error(e.getMessage());
```

运行后可以看见定义好的异常信息，但是没有打印异常堆栈信息。

（2）输出日志堆栈信息

为了保证日志的堆栈信息能够被输出，我们需要定义工具类。在 common_util 中创建 ExceptionUtil.java 工具类。

```
package cn.pup.crm.common.base.util;

public class ExceptionUtils {

    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();
    }

}

```

修改 GlobalExceptionHandler 中的异常输出语句

```
//e.printStackTrace();
//log.error(e.getMessage());
log.error(ExceptionUtils.getMessage(e));
```

（九） Knife4j 引入，用于导出 离线 API 文档

前身 swagger-bootstrap-ui 是springfox-swagger的增强UI实现，为Java开发者在使用 Swagger 的时候，能拥有一份简洁、强大的接口文档体验。一开始项目初衷是为了写一个增强版本的Swagger 前端UI，但是随着项目的发展,面对越来越多的个性化需求，项目正式更名为**knife4j**，取名knife4j是希望她能像一把匕首一样小巧、轻量，并且功能强悍，更名也是希望把她做成一个为 Swagger 接口文档服务的通用性解决方案，而不仅仅只是专注于前端UI。

1. 添加依赖

在 Common 项目的 POM 文件中添加依赖

```
<!-- Knife4j -->
<dependency>
  <groupId>com.github.xiaoymin</groupId>
  <artifactId>knife4j-spring-boot-starter</artifactId>
  <version>2.0.4</version>
</dependency>
```

2. 添加注解

在 Swagger 的配置文件 Swagger2Config 中为类添加注解 `@EnableKnife4j`

3. 运行

在浏览器中输入地址： http://localhost:8010/doc.html ，可以实现 Swagger-ui 的全部功能，并且可以下载离线文档。

Day 03/04 - 前端开发准备（2020-9-9）

> 这部分视频用了两天的时间对前端知识进行了梳理。这里仅记录一下核心内容。

一、开发工具

选择 Visual Studio Code 作为开发工具（以前都用 WebStorm）。

（一）必装扩展

- Chinese (Simplified) Language Pack
- ESLint
- Live Server
- Node.js Modules Intellisense
- Vetur
- VueHelper

（二）推荐扩展

- Material Theme
- Debugger for Chrome
- Auto Rename Tag
- Bracket Pair Colorizer 2
- indent-rainbow
- Prettier
- vscode-icons

（三）调整字体

管理→设置→搜索 “font”，根据需要设置字体。

（四）开启完整的 Emmet 语法支持

管理→设置→搜索 “emmet”，启用 `Emmet: Trigger Expansion On Tab`

（五）设置默认终端为 powershell

这样命令保留字会高亮。

二、Node.js

貌似没啥好记的，

三、ECMAScript 6 的基本语法

ES6相对之前的版本语法更严格，新增了面向对象的很多特性以及一些高级特性。本部分只学习项目开发中涉及到ES6的最少必要知识，方便项目开发中对代码的理解。

创建文件夹es6_pro

1、let声明变量

创建 01-let-01.js

```
// var 声明的变量没有局部作用域
// let 声明的变量 有局部作用域
{
  var a = 0
  let b = 1
}
console.log(a) // 0
console.log(b) // ReferenceError: b is not defined
```

创建 01-let-02.js

```
// var 可以声明多次
// let 只能声明一次
var m = 1
var m = 2
let n = 3
let n = 4 // Identifier 'n' has already been declared
console.log(m) // 2
console.log(n)
```

创建 01-let-03.js

```
// var 会变量提升
// let 不存在变量提升
console.log(x) //undefined
var x = 'apple'

console.log(y) //ReferenceError: y is not defined
let y = 'banana'
```

2、const声明常量（只读变量）

创建 02-const.js

```
// 1、声明之后不允许改变
const PI = '3.1415926'
PI = 3 // TypeError: Assignment to constant variable.
// 2、一但声明必须初始化，否则会报错
const MY_AGE // SyntaxError: Missing initializer in const declaration
```

3、解构赋值

创建 03-解构赋值-数组解构.js

解构赋值是对赋值运算符的扩展。是一种针对数组或者对象进行模式匹配，然后对其中的变量进行赋值。在代码书写上简洁且易读，语义更加清晰明了；也方便了复杂对象中数据字段获取。

```
//1、数组解构
let arr = [1, 2, 3]

// 传统
let a = arr[0]
let b = arr[1]
let c = arr[2]
console.log(a, b, c)

// ES6
let [x, y, z] = arr
console.log(x, y, z)
```

创建 03-解构赋值-对象解构.js

```
//2、对象解构
let user = {name: 'Helen', age: 18}
// 传统
let name1 = user.name
let age1 = user.age
console.log(name1, age1)
// ES6
let { name, age } = user//注意：解构的变量必须和user中的属性同名
console.log(name, age)
```

4、模板字符串

创建 04-模板字符串.js

模板字符串相当于加强版的字符串，用反引号 `，除了作为普通字符串，还可以用来定义多行字符串，还可以在字符串中加入变量和表达式。

```
// 字符串插入变量和表达式。变量名写在 ${} 中，${} 中可以放入 JavaScript 表达式。
let name = 'Mike'
let age = 27
let info = `My Name is ${name},I am ${age+1} years old next year.`
console.log(info)
// My Name is Mike,I am 28 years old next year.

//原样输出
let fun = `function(){
 console.log('hello')
}`
console.log(fun)
```

5、声明对象简写

创建 05-声明对象简写.js

```
let age = 12
let name = 'Amy'

// 传统
let person1 = {
   age: age,
```

```
    name: name
}
console.log(person1)

// ES6
let person2 = {
    age,
    name
}
console.log(person2) //{age: 12, name: 'Amy'}
```

6、定义方法简写

创建 06-定义方法简写.js

```
// 传统
let person1 = {
    sayHi:function(){
        console.log('Hi')
    }
}
person1.sayHi();//'Hi'


// ES6
let person2 = {
    sayHi(){
        console.log('Hi')
    }
}
person2.sayHi()  //'Hi'
```

7、对象拓展运算符

创建 07-对象拓展运算符.js

拓展运算符（...）用于取出参数对象所有可遍历属性然后拷贝到当前对象。

```
let person = {name: 'Amy', age: 15}
// let someone = person //引用赋值
let someone = { ...person } //对拷拷贝
someone.name = 'Helen'
console.log(person)  //{name: 'Amy', age: 15}
console.log(someone)  //{name: 'Helen', age: 15}
```

8、函数的默认参数

函数在JavaScript中也是一种数据类型，JavaScript中没有方法的重载

创建 08-函数的默认参数.js

```
function showInfo(name, age = 17) {
    console.log(name + "," + age)
}

// 只有在未传递参数，或者参数为 undefined 时，才会使用默认参数
// null 值被认为是有效的值传递。
showInfo("Amy", 18)  // Amy,18
showInfo("Amy")    // Amy,17
showInfo("Amy", undefined) // Amy,17
showInfo("Amy", null)  // Amy, null
```

9、箭头函数

创建 09-箭头函数.js

箭头函数提供了一种更加简洁的函数书写方式。基本语法是：`参数 => 函数体``箭头函数多用于匿名函数的定义`

```
let arr = ["10", "5", "40", "25", "1000"]
arr.sort()
console.log(arr)

//上面的代码没有按照数值的大小对数字进行排序，
//要实现这一点，就必须使用一个排序函数
//若 a 小于 b，在排序后的数组中 a 应该出现在 b 之前，则返回一个小于 0 的值。
//若 a 等于 b，则返回 0。
//若 a 大于 b，则返回一个大于 0 的值。
arr.sort(function(a,b){
    return a - b
})
console.log(arr)
```

使用箭头函数

```
//使用箭头函数
arr2 = arr.sort((a,b) => {
    return a - b
})
```

特例

```
// 当只有一行语句，并且需要返回结果时，可以省略 {}，结果会自动返回。
//当只有一个参数时，可以不使用圆括号
arr2 = arr.sort((a,b) => a - b)
```

四、Vue 的基本语法

Vue.js 是一款流行的 JavaScript 前端框架，目的是简化 Web 开发。Vue 所关注的核心是 MVC 模式中的视图层，同时，它也能方便地获取数据更新，实现视图与模型的交互。

声明式渲染：Vue.js 的核心是一个允许采用简洁的模板语法来声明式地将数据渲染进 DOM 的系统

这里的核心思想就是没有繁琐的DOM操作，例如jQuery中，我们需要先找到div节点，获取到DOM对象，然后进行一系列的节点操作

Vue 的内容很多，这里也暂时总结与项目相关的基本语法。

创建文件夹vue_pro

创建文件夹vuejs，将vue.min.js引入文件夹

五、axios

六、element-ui

七、NPM 包管理器

八、模块化

九、Webpack

十、vue-element-admin

十一、ESLint 语法规范检查

十二、前端项目架构

十三、临时登录接口

十四、Vue 路由

十五、前端路由配置