

技术研发部开发手册

*接收方*

**2020-06-30**

**版本 1.1**

*编写人员*

**梁超**

**[52375138@qq.com](mailto:52375138@qq.com)**

*其它贡献者*

**[文档贡献者列表]**

**文档变更记录**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **时间** | **版本** | **修改人** | **修改章节** | **修改原因** | **修改内容** |
| 2020-05-13 | V1.0 | 梁超 |  |  |  |
|  |  |  |  |  |  |

**目 录**

[技术研发部开发手册 0](#_Toc64626782)

[接收方 0](#_Toc64626783)

[版本 1.1 0](#_Toc64626784)

[编写人员 0](#_Toc64626785)

[梁超 0](#_Toc64626786)

[1 前言 1](#_Toc64626787)

[1.1 文档目的 1](#_Toc64626788)

[1.2 文档读者 1](#_Toc64626789)

[1.3 规范的意义和作用 1](#_Toc64626790)

[2 编码规范 1](#_Toc64626791)

[2.1 注释 1](#_Toc64626793)

[2.1.1 文件头注释 1](#_Toc64626794)

[2.1.2 类注释 2](#_Toc64626795)

[2.1.3 方法注释 2](#_Toc64626796)

[2.1.4 代码间注释 2](#_Toc64626797)

[2.2 命名规范 3](#_Toc64626798)

[2.1.5 一般原则 3](#_Toc64626799)

[2.1.6 命名规则 5](#_Toc64626800)

[通用命名规则 5](#_Toc64626801)

[控件命名规则 5](#_Toc64626802)

[命名示例 6](#_Toc64626803)

[书写规范 10](#_Toc64626804)

[3 架构规范 11](#_Toc64626805)

[2.1 简述 11](#_Toc64626808)

[2.2 项目命名 13](#_Toc64626809)

[2.3 开发环境 13](#_Toc64626810)

[2.4 Domain规范 13](#_Toc64626811)

[3.1.1 目录规范 13](#_Toc64626812)

[3.1.2 书写规范 14](#_Toc64626813)

[2.5 Infrastructure规范 16](#_Toc64626814)

[3.1.3 目录规范 17](#_Toc64626815)

[3.1.1 映射范例 17](#_Toc64626816)

[2.6 Repository规范 17](#_Toc64626817)

[3.1.2 书写规范 17](#_Toc64626818)

[IOC注入 17](#_Toc64626819)

[3.1.2.1 仓储接口声明 18](#_Toc64626820)

[3.1.2.1 仓储接口实现 18](#_Toc64626821)

[3.1.2.2 查询 19](#_Toc64626822)

[3.1.2.3 新增 20](#_Toc64626823)

[3.1.2.4 修改 21](#_Toc64626824)

[3.1.2.5 删除 22](#_Toc64626825)

[2.7 Application规范 23](#_Toc64626826)

[3.1.3 目录规范 23](#_Toc64626827)

[3.1.4 书写规范 24](#_Toc64626828)

[IOC注入 24](#_Toc64626829)

[3.1.4.1 接口声明 24](#_Toc64626830)

[3.1.4.2 接口实现 25](#_Toc64626831)

[2.8 Constant规则 27](#_Toc64626832)

[3.1.5 目录规范 27](#_Toc64626833)

[3.1.6 书写规范 28](#_Toc64626834)

[2.9 UI规则 28](#_Toc64626835)

[3.1.7 目录规范 28](#_Toc64626836)

[3.1.8 书写规范 29](#_Toc64626837)

[Controller规范 29](#_Toc64626838)

[IOC注入 30](#_Toc64626839)

[3.1.8.1 调用接口方法 31](#_Toc64626840)

[3.1.8.1 Dto规范 31](#_Toc64626841)

[2.10 配置文件规范 33](#_Toc64626842)

[3.1.9 数据库配置 33](#_Toc64626843)

[3.1.10 参数配置 33](#_Toc64626844)

[访问地址配置 33](#_Toc64626845)

[参数配置 33](#_Toc64626846)

[配置类 34](#_Toc64626847)

[获取配置信息 36](#_Toc64626848)

[2.11 日志记录规范 36](#_Toc64626849)

[3.1.11 记录文本日志 36](#_Toc64626850)

[3.1.1 记录操作日志 36](#_Toc64626851)

[4 项目结构 37](#_Toc64626852)

[4.1 简述 37](#_Toc64626855)

[4.2 zh-etp-core类库 37](#_Toc64626856)

[4.3 zh-etp-generator代码生成器 38](#_Toc64626857)

[4.4 zh-etp-oap运营平台 38](#_Toc64626858)

[4.5 zh-etp-eap企业管控平台 39](#_Toc64626859)

[4.6 zh-etp-idcode扫码入口 40](#_Toc64626860)

[5 部署教程 42](#_Toc64626861)

[6.1 命令说明 42](#_Toc64626864)

[6.1.1 常用命令 42](#_Toc64626867)

[6.1.2 上传码包 43](#_Toc64626868)

[6.1.3 删除码包 43](#_Toc64626869)

[6.1.4 查看容器 43](#_Toc64626870)

[6.1.5 查看docker镜像 43](#_Toc64626871)

[6.1.6 删除docker容器 43](#_Toc64626872)

[6.1.7 删除docker镜像 43](#_Toc64626873)

[6.1.8 构建容器 43](#_Toc64626874)

[6.1.9 启动镜像 44](#_Toc64626875)

# 前言

## 文档目的

保证项目中程序代码的强一致性，及可维护性．

## 文档读者

软件研发工程师

## 规范的意义和作用

1. 编码规范可以最大限度的提高团队开发的合作效率。

2.减少软件的维护成本 , 并且几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护。

3. 编码规范可以改善软件的可读性，可以让开发人员尽快而彻底地理解新的代码。

4. 规范性编码还可以让开发人员养成好的编码习惯，甚至锻炼出更加严谨的思维。

# 编码规范



## 注释

注释有助于理解代码，有效的注释是指在代码的功能、意图层次上进行注释，提供有用、额外的信息，而不是代码表面意义的简单重复。

注释的原则是有助于对程序的阅读理解，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。有效的注释是指在代码的功能、意图层次上进行注释，提供有用、额外的信息。注释使用中文，保证代码和注释的一致性。修改代码同时修改相应的注释，不再有用的注释要删除。

### 文件头注释

在每个文件头必须包含以下注释说明



文件创建时,要注明创建的*模块名称,功能描述,创建人,创建时间*

每次需求产生变更,需要修改文件时,要注明修改时对应的*修改时间*,*修改人*,简单*概述信息*.

### 类注释

为 每一个重要的***类***、***接口***、***抽象类***、***结构体*** 作注释



### 方法注释



### 代码间注释



## 命名规范

名称应该说明“什么”而不是“如何”。通过避免使用公开基础实现（它们会发生改变）的名称，可以保留简化复杂性的抽象层。例如，可以使用 *GetNextStudent()*，而不是 *GetNextArrayElement()*。

命名原则是：

选择正确名称时的困难可能表明需要进一步分析或定义项的目的。使名称足够长以便有一定的意义，并且足够短以避免冗长。唯一名称在编程上仅用于将各项区分开。表现力强的名称是为了帮助人们阅读；因此，提供人们可以理解的名称是有意义的。不过，请确保选择的名称符合适用语言的规则和标准。

### 一般原则

1. 避免容易被主观解释的难懂的名称，如方面名 *AnalyzeThis()*，或者属性名 *xxK8*。这样的名称会导致多义性。
2. 在类属性的名称中包含类名是多余的，如 *Book.BookTitle*。而是应该使用 *Book.Title*。
3. 只要合适，在变量名的末尾或开头加计算限定符（*Avg*、*Sum*、*Min*、*Max*、*Index*）。
4. 标识符的命名应当符合“**min-length** && **max-information**”原则。较短的单词可通过去掉“元音”形成缩写，较长的单词可取单词的头几个字母形成缩写，一些单词有大家公认的缩写，常用单词的缩写必须统一。协议中的单词的缩写与协议保持一致。对于某个系统使用的专用缩写应该在某处做统一说明。

**正例**：如下单词的缩写能够被大家认可：

***temp*** 可缩写为 ***tmp***；

***flag*** 可缩写为 ***flg***；

***statistic*** 可缩写为 ***stat***；

***increment*** 可缩写为 ***inc***；

***message*** 可缩写为 ***msg***；

…

1. 在变量名中使用互补对，如 :

***add/remove*** ； ***begin/end*** ； ***create/destroy*** ； ***insert/delete ；***

***first/last*** ；  ***get/release*** ； ***increment/decrement*** ； ***put/get ；***

***add/delete*** ； ***lock/unlock*** ； ***open/close*** ； ***min/max ；***

***old/new*** ； ***start/stop*** ； ***next/previous*** ； ***source/target ；***

***show/hide*** ； ***send/receive*** ； ***source/destination*** ； ***cut/paste ;***

***up/down ;***

1. 布尔变量名应该包含 Is，这意味着 Yes/No 或 True/False 值，如 *fileIsFound*。
2. 在命名状态变量时，避免使用诸如 Flag 的术语。状态变量不同于布尔变量的地方是它可以具有两个以上的可能值。不是使用 *documentFlag*，而是使用更具描述性的名称，如 *documentFormatType*。 （此项只供参考）
3. 即使对于可能仅出现在几个代码行中的生存期很短的变量，仍然使用有意义的名称。仅对于短循环索引使用单字母变量名，如 i 或 j。 可能的情况下，尽量不要使用原义数字或原义字符串，如

For i = 1 To 7。而是使用命名常数，如 *For i = 1 To NUM\_DAYS\_IN\_WEEK* 以便于维护和理解。命名常数可定义在类的字段定义区域；

* Good

1. public class VariableExample
2. {
3. //定义工作薄最大行数
4. private const int SHEET\_MAX\_ROWS = 125;
5. public void GetCellText()
6. {
7. //code here...
8. for (int i = 0; i < SHEET\_MAX\_ROWS; i++)
9. {
10. //code here...
11. }
12. //code here...
13. }
14. }
15. 建议常量名都要使用大写字母, 用下划线 ‘\_’ 分割单词;
16. private const int SHEET\_MAX\_ROWS = 125;
17. private const string ALLOW\_FILE\_TYPE = "JPEG;GIF";
18. private const int MAX\_EMAIL\_ADDRESS\_LENGTH = 150;
19. 建议成员变量的命名采用Pascal命名法，前面加上下划线(\_)；
20. 属性的命名采用Pascal命名法；
21. 一行只声明一个成员变量；在行尾用”//”开始的注释解释该成员变量的含义和用途；
22. private decimal \_OrderTotal; //订单总额
23. /// <summary>
24. /// 订单总额
25. /// </summary>
26. public decimal OrderTotal { get; set; }

### 命名规则

#### 通用命名规则

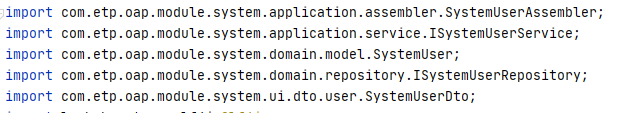
|  |  |  |
| --- | --- | --- |
| 标识符 | 大小写 | 示例 |
| 类 | Pascal | AppDomain |
| 枚举类型 | Pascal | ErrorLevel |
| 枚举值 | Pascal | FatalError |
| 事件 | Pascal | ValueChange |
| 异常类 | Pascal | WebException  注意 总是以 Exception 后缀结尾。 |
| 只读的静态字段 | Pascal | RedValue |
| 接口 | Pascal | IDisposable  注意 总是以 I 前缀开始。 |
| 方法 | Pascal | ToString |
| 命名空间 | Pascal | System.Drawing |
| 属性 | Pascal | BackColor |
| 公共实例字段 | Pascal | RedValue  注意 很少使用。属性优于使用公共实例字段。 |
| 受保护的实例字段 | Camel | redValue  注意 很少使用。属性优于使用受保护的实例字段。 |
| 私有的实例字段 | Camel | redValue |
| 参数 | Camel | typeName |
| 方法内的变量 | Camel | backColor |

#### 控件命名规则

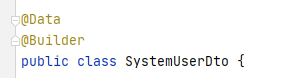
|  |  |  |
| --- | --- | --- |
| **控件类型** | **控件句柄命名格式** | **MFC 控件命名格式** |
| Animation Control | hXxxAnimate | aniXxx |
| Button | hXxxButton | btnXxx |
| Check Box | hXxxCheck | chkXxx |
| ComboBox | hXxxCombo | cmbXxx |
| Date Time Picker | hXxxDatePicker | dtpXxx |
| Edit Control | hXxxEdit | tbXxx |
| Group Box | hXxxGroup | grpXxx |
| Horizontal Scroll Bar | hXxxScroll | hsbXxx |
| IP Address Control | hXxxIpAddr | ipXxx |
| List Box | hXxxList | lstXxx |
| List（View） Control | hXxxList | lvwXxx |
| Menu | hXxxMenu | N/A |
| Month Calendar Control | hXxxCalendar | mclXxx |
| Picture Box | hXxxPicture | pctXxx |
| Progress Control | hXxxProgress | prgXxx |
| Radio Box | hXxxRadio | radXxx |
| Rich Edit Control | hXxxRichEdit | rtfXxx |
| Slider Control | hXxxSlider | sldXxx |
| Spin Control | hXxxSpin | spnXxx |
| Static Text | hXxxLabel | lbXxx |
| SysLink Control | hXxxLink | lnkXxx |
| Tab Control | hXxxTab | tabXxx |
| Tree（View） Control | hXxxTree | tvwXxx |
| Vertical Scroll Bar | hXxxScroll | vsbXxx |

#### 命名示例

##### 命名空间



##### 类

 [Visual Basic]

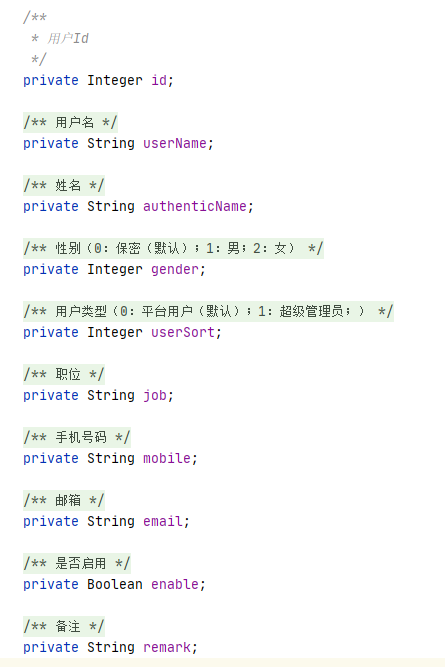
Public Class **FileStream**

Public Class **Button**

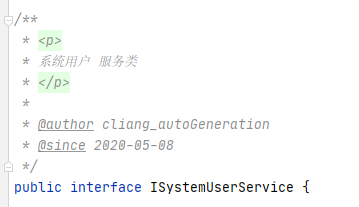
Public Class **String**

[C#]

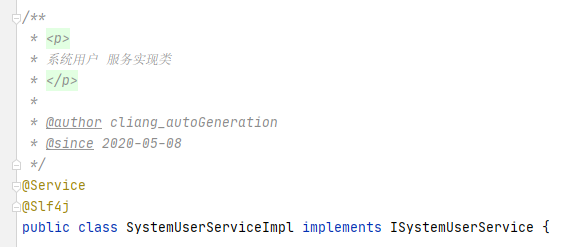
##### 类成员变量



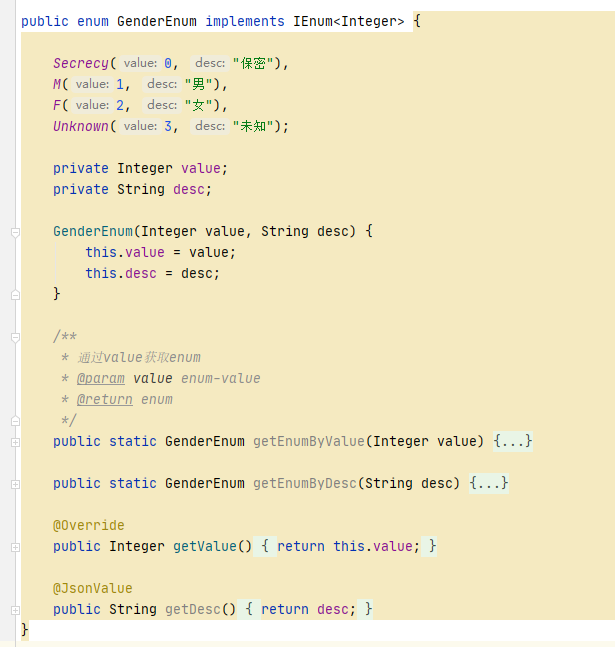
##### 接口定义



##### 接口



##### 枚举



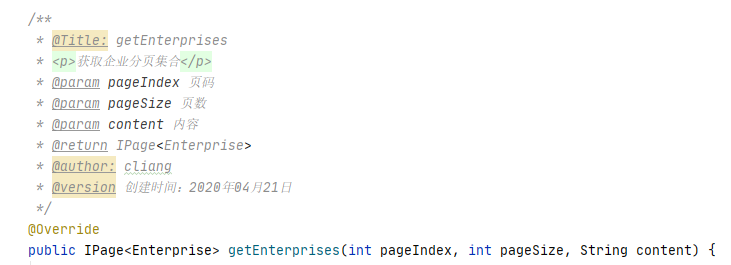
##### 静态字段



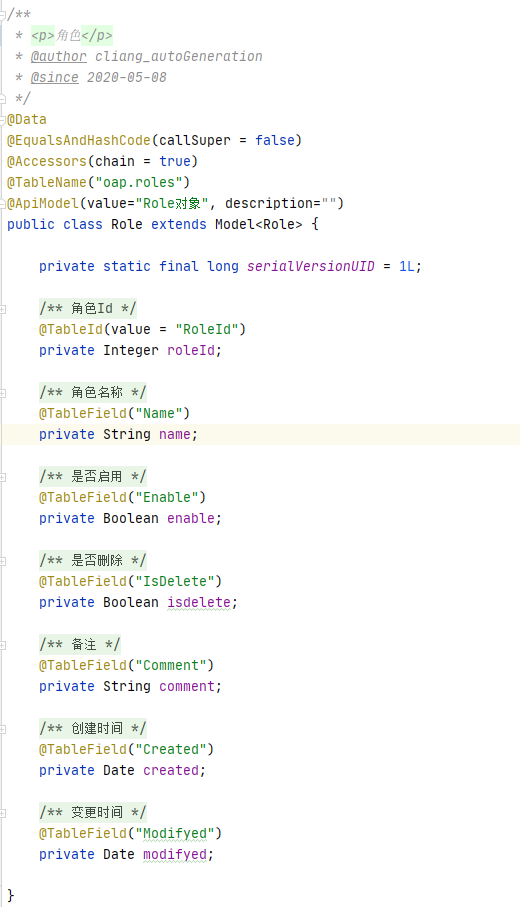
##### 参数



##### 方法



##### 对象属性



#### 书写规范

1. 功能相关的方法放在一起。
2. （建议）一个源文件原则上不能超过500行代码；
3. （建议）代码的每行长度不能超过80个字符；
4. 代码必须正确地按逻辑层次缩进，缩进使用制表符；一般缩进为4个空格，可以对IDE统一设置；不统一的情况下，避免使用TAB进行缩进；
5. if、else、else if、for、while、do等语句自占一行，执行语句不得紧跟其后。不论执行语句有多少都要加 { }。
6. 空行：方法之间以及代码块之间用空行隔开；
7. 空格：(建议)IDE默认的格式
8. 程序的分界符‘{’和‘}’应独占一行并且位于同一列，同时与引用它们的语句左对齐。{ }之内的代码块使用缩进规则对齐。
9. （建议）长表达式（超过80列）要在低优先级操作符处拆分成新行，操作符放在新行之首（以便突出操作符）。拆分出的新行要进行适当的缩进，使排版整齐。
10. 当一行被分为几行时，通过将串联运算符放在每一行的末尾而不是开头，清楚地表示没有后面的行是不完整的。
11. 每一行上放置的语句避免超过一条。
12. 在大多数运算符之前和之后使用空格，这样做时不会改变代码的意图却可以使代码。
13. 单一职责原则：

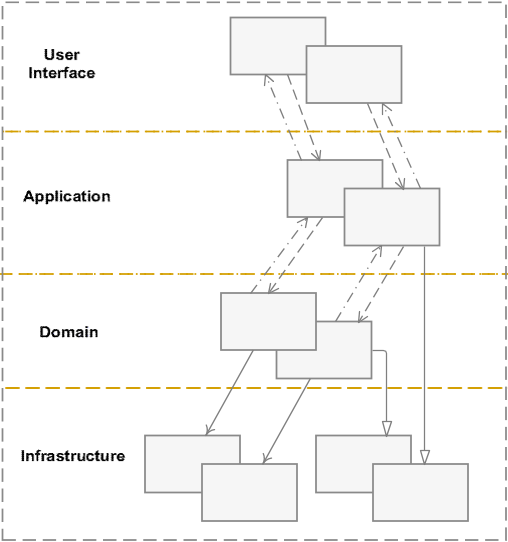
可以表现在类和方法层面上，从方法的层面上，一个方法只能做一件事情；从类的层面上，一个类只能有一个职责。否则，就要对类和方法进行拆分重构。对于方法的拆分重构，目标是清晰度，能提升代码的可读性，但是不能提升代码的自适应能力。要提升代码的自适应能力，就要做抽象，将每个职责划分到不同的类中。

# 架构规范

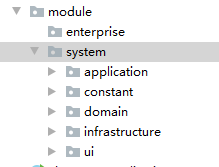


## 简述

架构基于领域驱动设计- 应对软件的复杂性：



项目模块接口如下：



user interface：

        该层包含与其他系统/客户进行交互的接口与通信设施，在多数应用里，该层可能提供包括WebAPI、Web Services、或Rest等在内的一种或多种通信接口。

application：

        application层中负责业务逻辑的实现和协调委派业务逻辑给领域对象进行处理。

domain：

domain层是整个系统的核心层，该层维护一个使用面向对象技术实现的领域模型。domain层包含entity（实体）、valueobject(值对象)、domain event（领域事件）等多种重要的领域组件。

领域模型具备自己的属性行为和状态，领域对象元素之间通过聚合配合解决实际业务应用。可复用，可维护，易扩展，可以采用合适的设计模型进行详细设计，相对复杂，要求设计人员有良好的抽象能力。

infrastructure：

        infrastructure（基础设施层）为interfaces、application和domain三层提供支撑。避免三层特别是domain层掺杂进这些实现，从而“污染”领域模型。infrastructure中最常见的一类设施是对象持久化的具体实现。

## 项目命名

要求：

* 名称：com.{项目名称} 名称简洁体现职责
* 名字作为组件的名称

## 开发环境

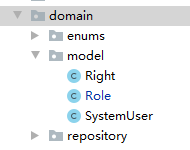
* 开发环境：Springboot+MyBatis-plus+postgresSQL
* 开发工具：idea 2020
* 代码管理工具：Git /TortoiseGit

## Domain规范

### 目录规范

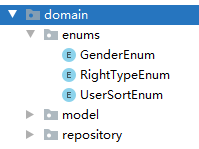
* model目录：领域模型（包含与数据库相关表映射的实体类）

注：不同业务模块实体需包含在各自的目录下



* enums目录：枚举类

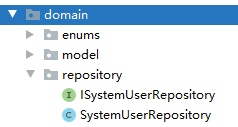
命名：接口名称+Enum



* repository目录：仓储的接口实现

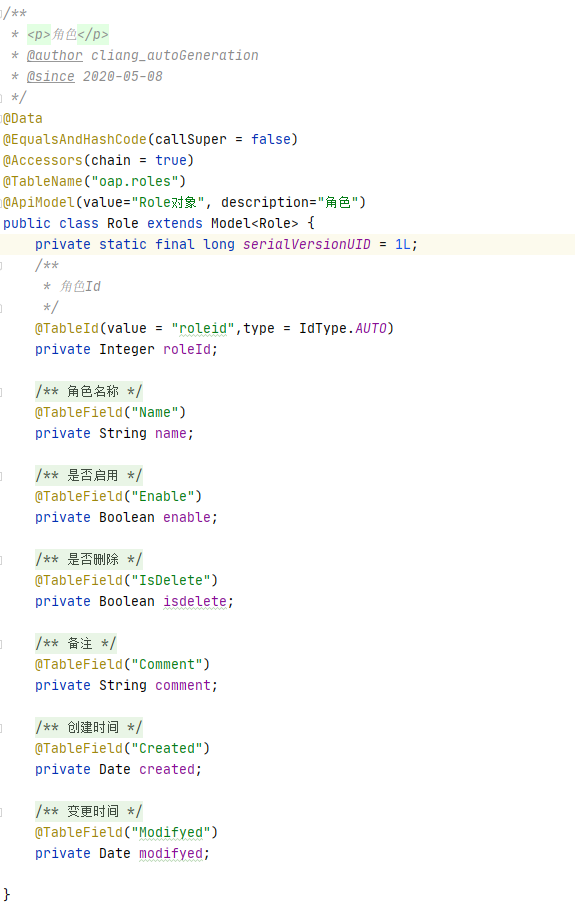
命名：接口名称+Repository

注：不同业务模块实体需包含在各自的目录下

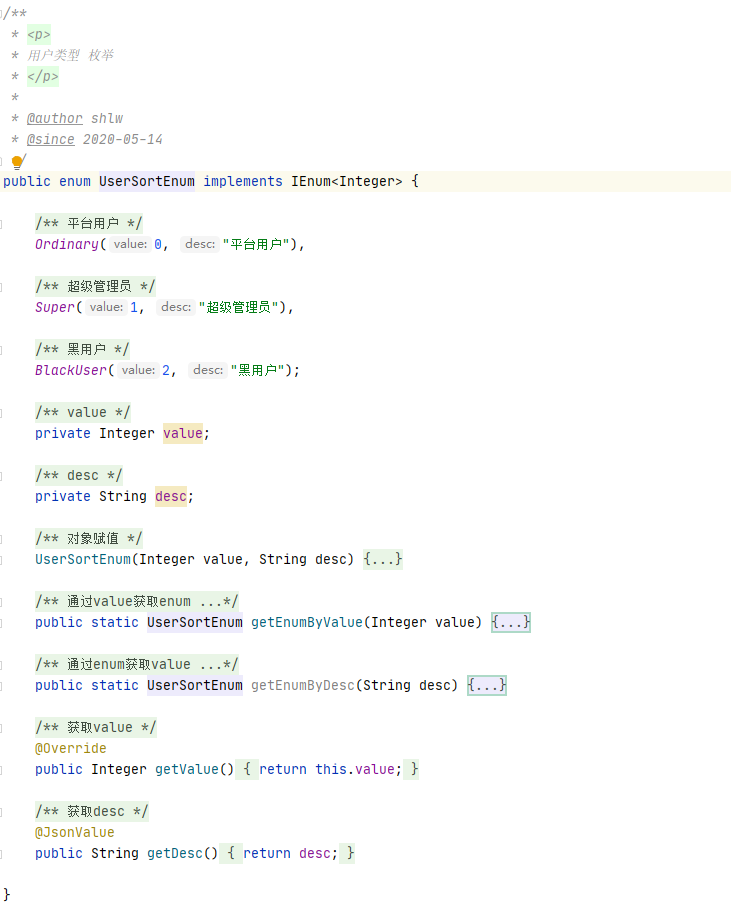


### 书写规范

领域实体规范如下：



枚举规范如下：

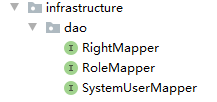


## Infrastructure规范

### 目录规范

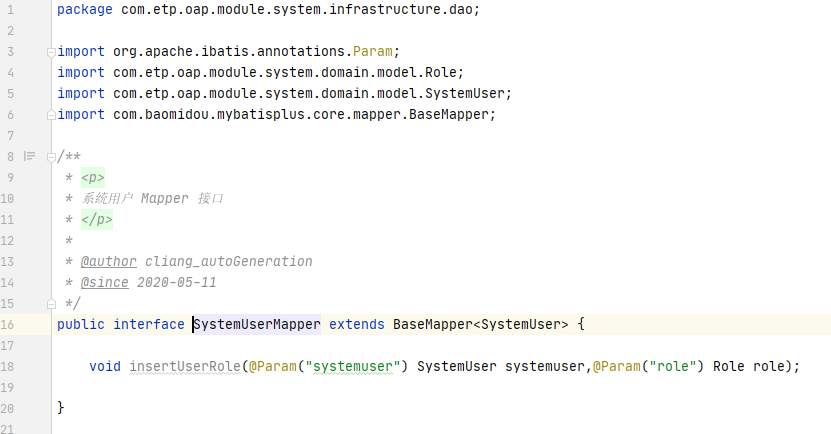
* dao目录：领域实体映射，

命名：实体名称+Mapper

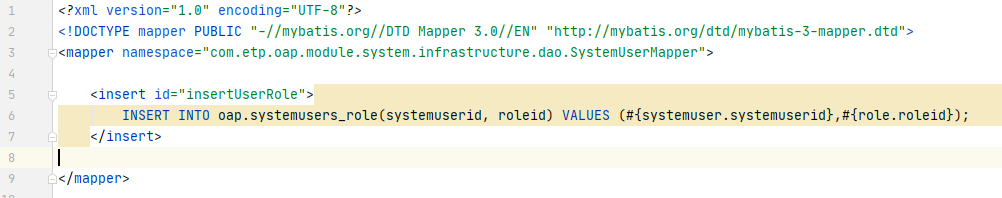


### 映射范例

Mapper格式如下：



Xml格式如下：



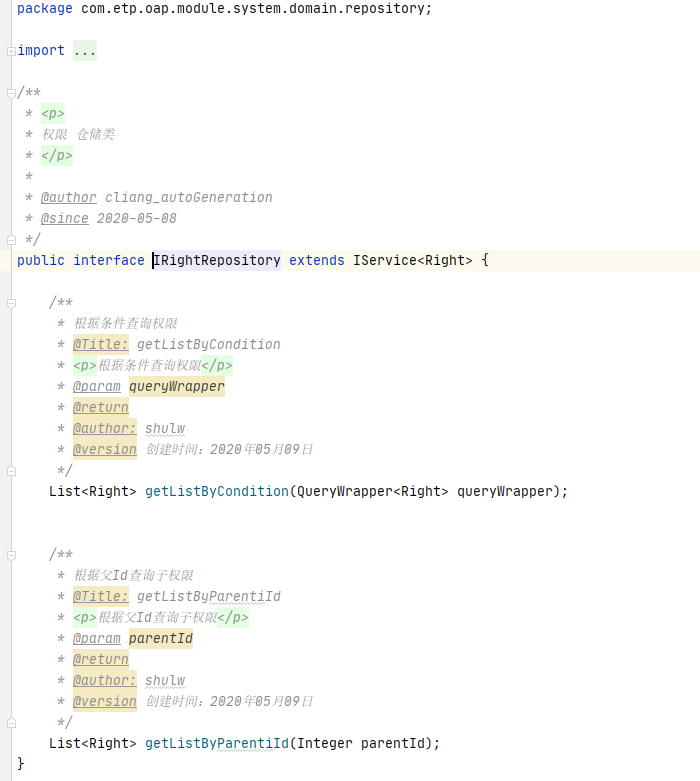
## Repository规范

### 书写规范

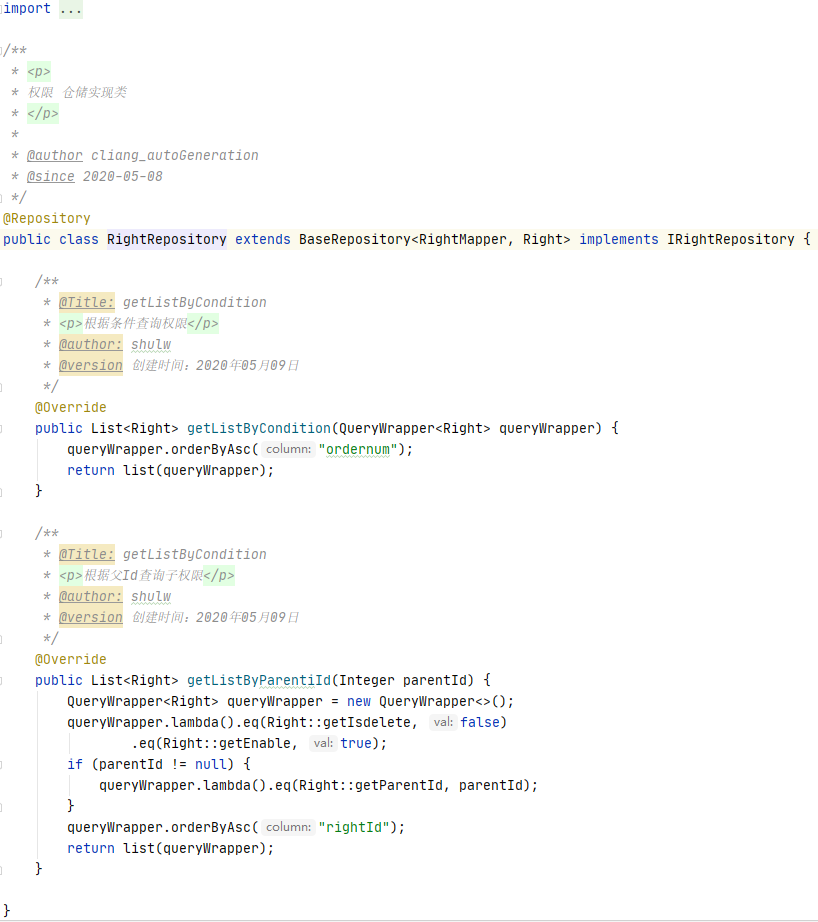
#### IOC注入

在构造函数做依赖注入

#### 仓储接口声明



#### 仓储接口实现



#### 查询



#### 新增



#### 修改



#### 删除

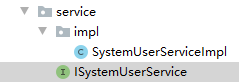


## Application规范

### 目录规范

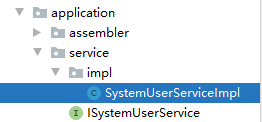
* service目录：接口声明，

命名：I+接口名称+ Service

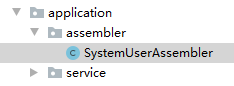


* impl目录：接口实现

命名：接口名称+ ServiceImpl



* application目录：模块的组装器，完成domain model对象到dto的转换



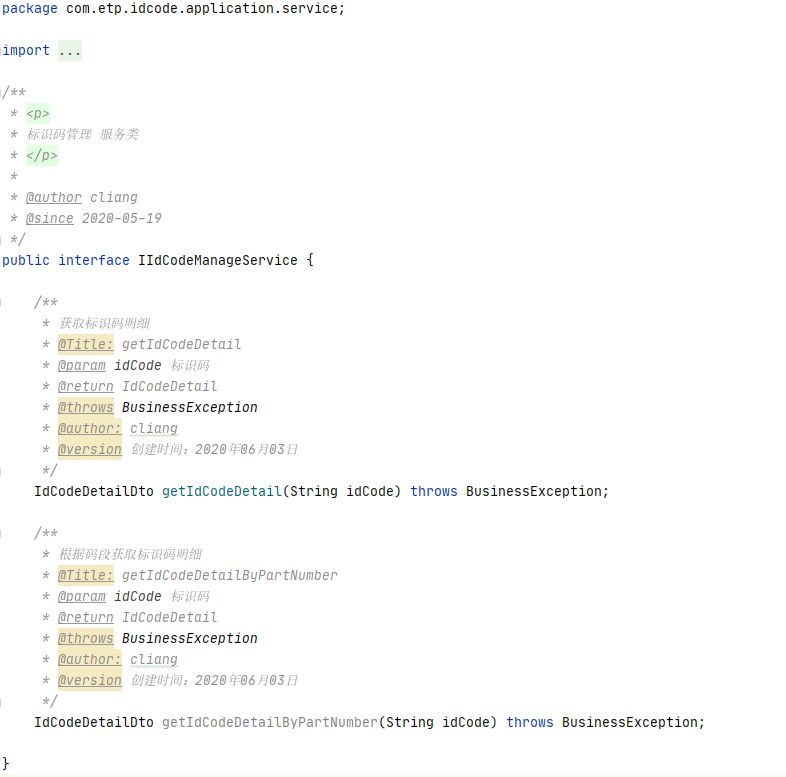
### 书写规范

#### IOC注入

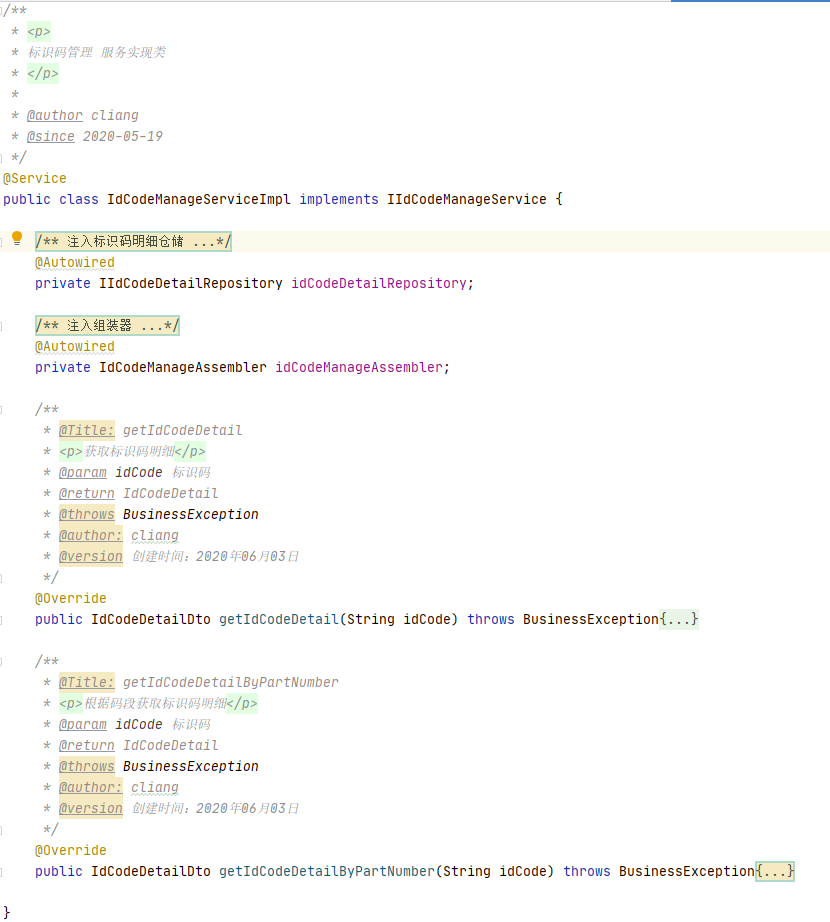
做仓储的依赖注入

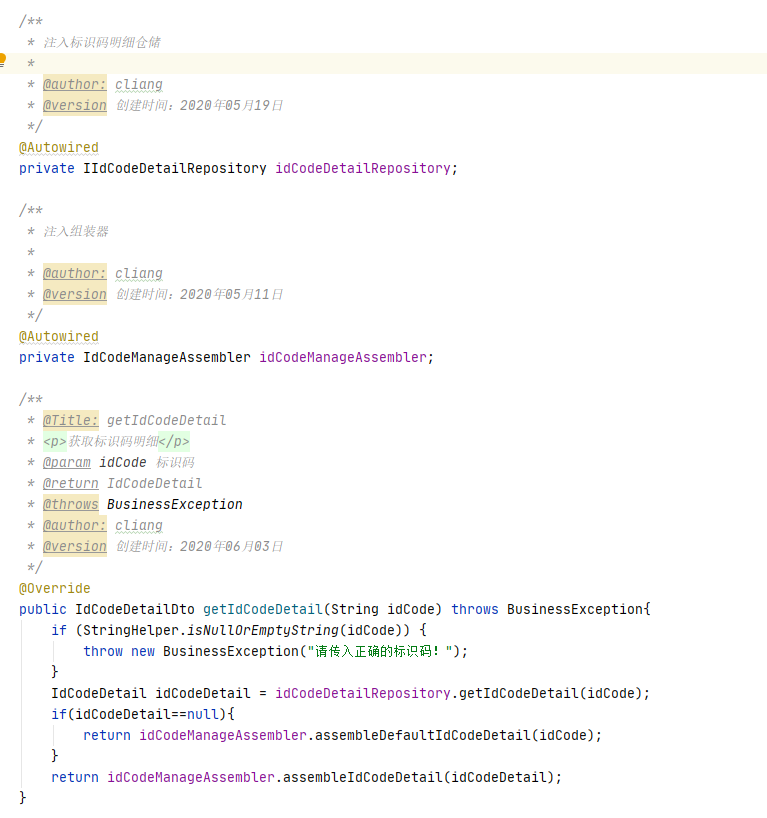


#### 接口声明



#### 接口实现

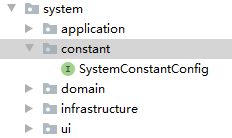




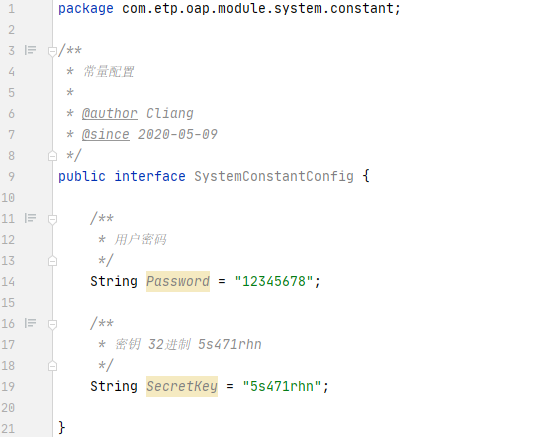
## Constant规则

### 目录规范

* constant目录：该目录存放常量配置。



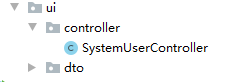
### 书写规范



## UI规则

### 目录规范

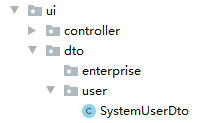
* Controllers目录：控制器方法类，编写接口加载逻辑



* Dtos：数据模型实体（包含数据传输实体，使用场景是接口数据返回），

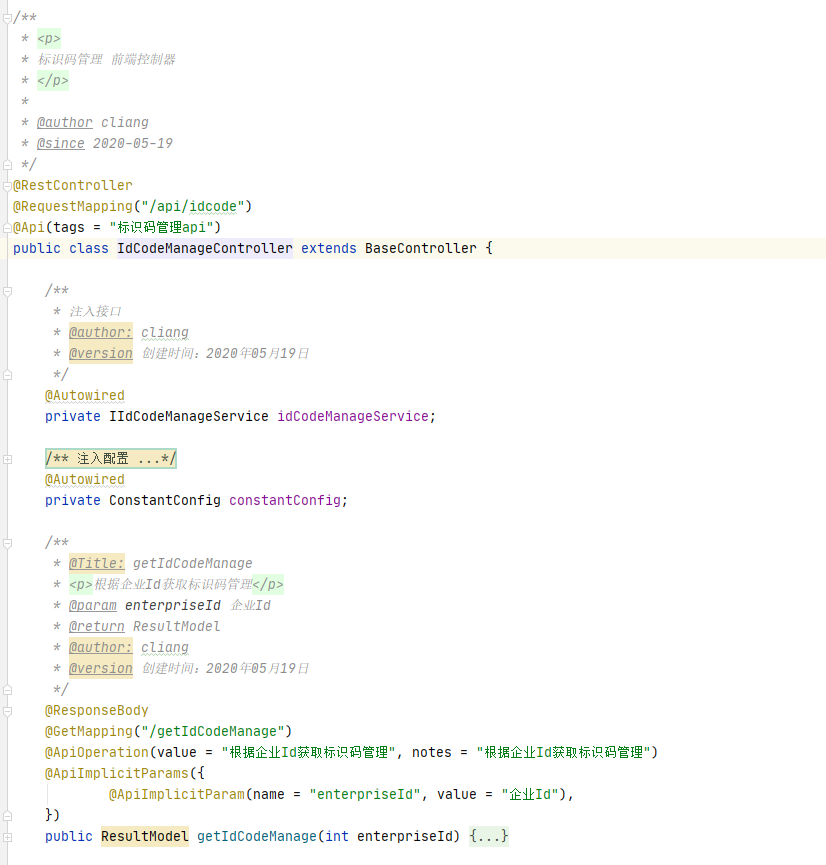
命名：实体名称+Dto

注：不同业务模块实体需包含在各自的目录下。



### 书写规范

#### Controller规范



#### IOC注入

做接口的依赖注入



#### 调用接口方法



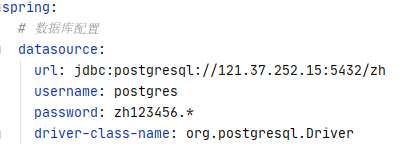
#### Dto规范



## 配置文件规范

### 数据库配置

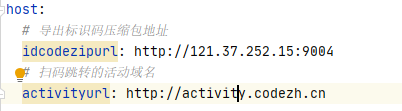
* 在application-config.yml文件里配置数据库参数



### 参数配置

#### 访问地址配置

在application-config.yml文件里配置访问地址，每次变更可在配置中做对应的调整



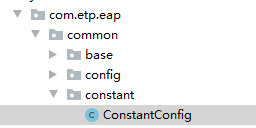
#### 参数配置

在application.yml文件里配置参数，每次变更可在配置中做对应的调整



#### 配置类

配置类在项目中的位置

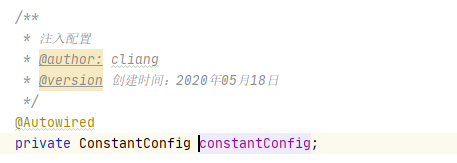


绑定对应的配置信息



#### 获取配置信息

先注入配置类



代码里直接获取



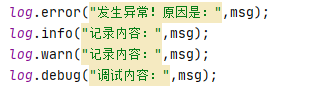
## 日志记录规范

### 记录文本日志

* 引入@Slf4j

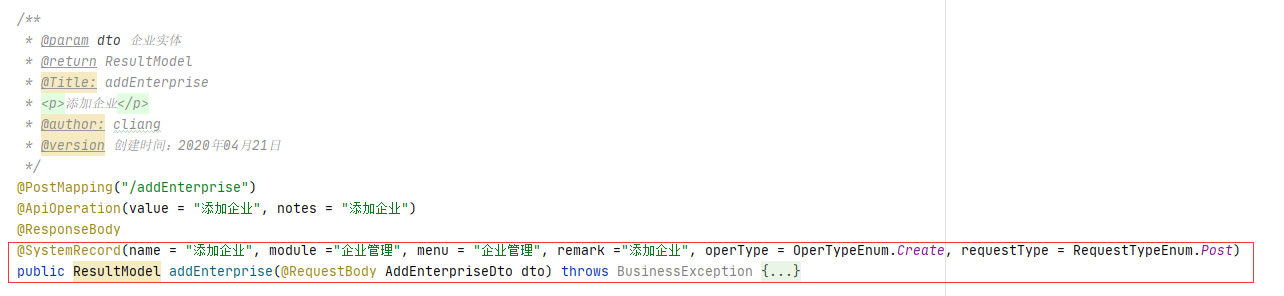


* 记录方式



### 记录操作日志

* 在方法上引入@SystemRecord



填入对应的内容：name = "添加企业", module ="企业管理", menu = "企业管理", remark ="添加企业", operType = OperTypeEnum.*Create*, requestType = RequestTypeEnum.*Post*

# 项目结构



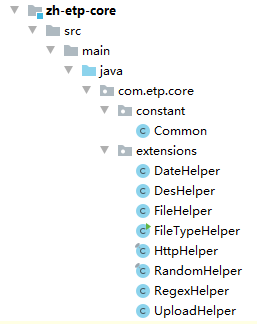
## 简述

项目整合了多个平台和类库：



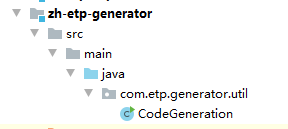
## zh-etp-core类库

所有平台的公共处理类都集中在这里处理实现：



## zh-etp-generator代码生成器

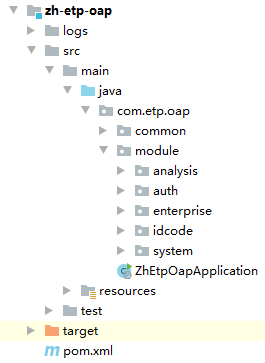
根据数据库的表结构自动生成代码文件：



运行CodeGeneration自动生成代码文件

## zh-etp-oap运营平台

运营平台的源码：



平台测试地址：

<http://zhoap.codezh.cn/login>

账号：admin

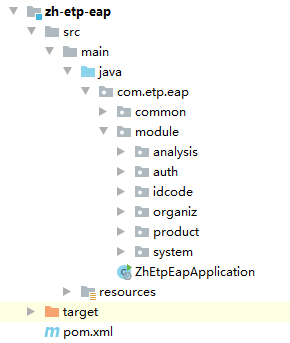
密码：123456789

界面如下：



## zh-etp-eap企业管控平台

企业管控平台的源码：



平台测试地址：

<http://zheap.codezh.cn/login>

账号：admin@shkj

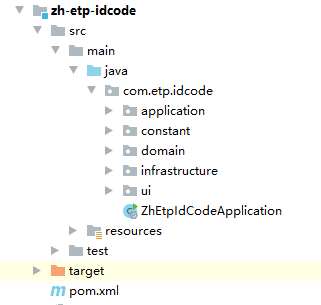
密码：123456789

界面如下：



## zh-etp-idcode扫码入口

公共的企业标识码扫码入口



通过微信扫码标识码的时候，首先会先访问该扫码入口，

里面的业务逻辑是：先根据标识码查询该标识码对应的活动地址，并跳转到该活动地址。

例如：

访问该地址：<http://121.37.252.15:9005/0012006300000000194961730>

会自动跳转到对应该标识码的活动地址：

<http://activity.codezh.cn/scan/shkj/sys001/index.html?idCode=0012006300000000194961730>

页面会显示如下信息：



# 部署教程



## 命令说明



### 常用命令

cd 回退到根目录

cd .. 回退上级目录

进入oap

cd /mnt/sdc/application/update/zh-etp/zh-etp-eap/

cd /mnt/sdc/application/frontend/zh-etp

sudo unzip eap.zip 输入命令a+回车

### 上传码包

sudo rz –E

### 删除码包

sudo rm -rf 码包

### 查看容器

sudo docker ps -a

### 查看docker镜像

sudo docker images

### 删除docker容器

sudo docker rm -f 容器id（如：a4d260ad345c）

### 删除docker镜像

sudo docker rmi 镜像id（如：f4f1482606ac）

### 构建容器

构建idcode项目的容器：

sudo docker build -t etp-idcode -f dockerfile-etp-idcode .

构建oap项目的容器：

sudo docker build -t etp-oap -f dockerfile-etp-oap .

构建eap项目的容器：

sudo docker build -t etp-eap -f dockerfile-etp-eap .

### 启动镜像

启动idcode镜像：

sudo docker run -it -d -p 9005:8080 -v /mnt/sdc/application/data/etp-idcode/log:/app/log --restart always etp-idcode

启动oap镜像：

sudo docker run -it -d -p 9001:8080 -v /mnt/sdc/application/data/etp-oap/log:/app/logs -v /mnt/sdc/application/data/etp-oap/static:/app/static -v /mnt/sdc/application/frontend/zh-etp/activity/:/app/activity --restart always etp-oap

启动eap镜像：

sudo docker run -it -d -p 9004:8080 -v /mnt/sdc/application/data/etp-eap/log:/app/logs -v /mnt/sdc/application/frontend/zh-etp/activity/:/app/activity -v /mnt/sdc/application/data/etp-eap/static:/app/static --restart always etp-eap