**平台使用说明文档**

**2025-05-01**

**目录**

[1 平台整体介绍 1](#_Toc9900)

[1.1 后端软件包 1](#_Toc309)

[1.2 前端软件包 1](#_Toc29744)

[1.3 运维部署环境 1](#_Toc23530)

[2 后台说明 1](#_Toc18797)

[2.1 模块说明 1](#_Toc7871)

[2.2 业务子模块拆分 2](#_Toc24939)

[2.3 主要配置 4](#_Toc15595)

[2.4 Controller 4](#_Toc15760)

[2.5 Service 5](#_Toc18723)

[2.5.1 实现思路 5](#_Toc26901)

[2.5.2 事务规则 6](#_Toc16297)

[2.5.3 常用方法 7](#_Toc23905)

[2.5.3.1 查询列表对象 7](#_Toc4558)

[2.5.3.2 查询单个对象 8](#_Toc13817)

[2.5.3.3 保存 8](#_Toc1685)

[2.5.3.4 更新 9](#_Toc6431)

[2.5.3.5 删除 10](#_Toc24917)

[2.5.3.6 SQL语句工具类方法 10](#_Toc8804)

[2.6 Dao 10](#_Toc1682)

[2.6.1 一个数据库只有一个Dao 11](#_Toc18093)

[2.6.2 数据库dialect 11](#_Toc22620)

[2.6.3 使用多个数据库 12](#_Toc9598)

[2.7 Entity 12](#_Toc3835)

[2.7.1 使用的注解 13](#_Toc6538)

[2.7.2 使用别名 13](#_Toc14262)

[2.8 Finder 14](#_Toc10969)

[2.8.1 直接写SQL语句 14](#_Toc4840)

[2.8.2 获取具体操作的Finder对象 14](#_Toc22275)

[2.8.3 like写法 14](#_Toc5070)

[2.8.4 使用范例 15](#_Toc2123)

[2.9 Lucene 15](#_Toc6061)

[2.10 业务权限 16](#_Toc6823)

[2.10.1 URL请求拦截 16](#_Toc22341)

[2.10.2 业务数据权限 16](#_Toc1529)

[2.10.2.1 权限实现的功能 16](#_Toc27812)

[2.10.2.2 表结构设计 17](#_Toc16124)

[2.10.2.3 不同菜单下,数据权限不同 17](#_Toc22748)

[2.10.2.4 调用示例 18](#_Toc11445)

[2.11 安全 18](#_Toc4141)

[2.11.1 SQL注入 18](#_Toc15984)

[2.11.2 XSS防护 18](#_Toc2197)

[2.11.3 CSRF 19](#_Toc20106)

[2.12 缓存 19](#_Toc26098)

[2.13 数据库读写分离 21](#_Toc8338)

[2.13.1 事务粘性 21](#_Toc30741)

[2.13.2 读写强制切换 21](#_Toc32148)

[2.14 代码生成器(springrain-gencode) 22](#_Toc8816)

[2.14.1.1 原理 22](#_Toc18306)

[2.14.1.2 使用 22](#_Toc16973)

[2.14.1.3 维护数据库注释 23](#_Toc25877)

[2.15 微服务 24](#_Toc4019)

[2.15.1 说明 24](#_Toc12710)

[2.15.2 实现思路 25](#_Toc5721)

[2.15.3 限制 25](#_Toc14534)

[2.15.4 实现代码 25](#_Toc3936)

[2.15.4.1 项目启动加载SpringBean 25](#_Toc927)

[2.15.4.2 事务处理 26](#_Toc27048)

[2.15.4.3 GRPC调用 27](#_Toc11883)

[2.15.4.4 POM依赖例子 27](#_Toc32249)

[2.15.4.5 serviceimpl独立运行 27](#_Toc28284)

[2.15.4.6 效果演示 28](#_Toc5470)

[2.16 规范约定 28](#_Toc13738)

[2.16.1 不允许改动frame包下的文件 28](#_Toc23389)

[2.16.2 不允许SQL拼接前台参数,必须使用占位符 28](#_Toc17965)

[2.16.3 不允许手写表名，只能使用Finder工具类获取表名 28](#_Toc25334)

[2.16.4 Controller编写规范 29](#_Toc25117)

[2.16.5 Service编写规范及常用方法 30](#_Toc23694)

[2.16.6 不允许复写Dao，一个数据库只能有一个Dao 31](#_Toc23610)

[2.16.7 不允许手动编写分页函数和其他特定函数 31](#_Toc19500)

[3 前端说明 32](#_Toc15270)

[3.1 目录结构 32](#_Toc19127)

[3.1.1 目录规范说明 32](#_Toc29365)

[3.2 目录重点文件说明 32](#_Toc29210)

[3.2.1 目录位置：src/commons 32](#_Toc26883)

[3.2.2 目录位置：src/router 33](#_Toc21358)

[3.3 菜单配置 34](#_Toc23405)

[3.3.1 目录位置 34](#_Toc4001)

[3.3.2 字段说明 35](#_Toc26786)

[3.4 页面配置 35](#_Toc11830)

[3.4.1 页面模板 35](#_Toc5988)

[3.4.2 页面配置 35](#_Toc27402)

[3.4.3 页面显示和隐藏 37](#_Toc19382)

[3.4.4 页面容器PageContent 38](#_Toc15176)

[3.4.5 弹窗页面 39](#_Toc2691)

[3.5 AJAX配置 40](#_Toc2797)

[3.5.1 Ajax请求 40](#_Toc10122)

[3.5.2 Ajax调用方式 40](#_Toc32659)

[3.5.3 Ajax配置参数 41](#_Toc6029)

[3.5.4 Ajax请求结果提示 42](#_Toc12882)

[3.5.5 Loading处理 42](#_Toc12565)

[3.6 Models(redux)封装 42](#_Toc8705)

[3.6.1 Redux 42](#_Toc14418)

[3.6.2 Redux models配置 43](#_Toc26647)

[3.6.3 组件和redux进行连接 43](#_Toc14032)

[3.7 权限控制，权限组件 45](#_Toc19964)

[3.7.1 菜单权限 45](#_Toc26149)

[3.7.2 功能权限 45](#_Toc3474)

[3.8 其他重点规范 46](#_Toc27844)

[3.8.1 组件类命名 46](#_Toc7985)

[3.8.2 import引入规范 47](#_Toc26255)

[3.8.3 打包规范 47](#_Toc18550)

[3.8.4 路由前缀命名和请求地址命名规范 48](#_Toc16103)

[3.8.5 额外功能 48](#_Toc31379)

[4 界面预览 49](#_Toc20549)

[4.1 登录 49](#_Toc17403)

[4.2 首页 49](#_Toc7592)

[4.3 用户管理 50](#_Toc30883)

[4.3.1 添加用户 50](#_Toc23640)

[4.3.2 修改用户 51](#_Toc3385)

[4.3.3 删除用户 52](#_Toc30921)

[4.3.4 树形部门查询 52](#_Toc17647)

[4.4 角色管理 53](#_Toc22444)

[4.4.1 添加角色 53](#_Toc3797)

[4.4.2 修改角色 54](#_Toc23661)

[4.4.3 删除角色 54](#_Toc5849)

[4.4.4 分配角色权限 55](#_Toc18603)

[4.5 部门管理 55](#_Toc28640)

[4.5.1 添加部门 56](#_Toc25714)

[4.5.2 修改部门 56](#_Toc10138)

[4.5.3 删除部门 57](#_Toc7482)

[4.5.4 添加子部门 57](#_Toc23036)

[4.6 菜单管理 58](#_Toc31232)

[4.6.1 添加菜单 58](#_Toc18823)

[4.6.2 修改菜单 59](#_Toc29932)

[4.6.3 删除菜单 59](#_Toc27329)

[4.6.4 添加子菜单 60](#_Toc3938)

[4.6.5 批量添加菜单 60](#_Toc1170)

[4.6.6 添加功能 61](#_Toc23414)

[4.7 字典管理 61](#_Toc21599)

[4.7.1 添加字典 62](#_Toc19989)

[4.7.2 修改字典 63](#_Toc23571)

[4.7.3 删除字典 64](#_Toc1892)

[4.7.4 添加子项 64](#_Toc29060)

# 平台整体介绍

框架基于JDK21平台,Spring/SpringBoot技术栈,JWT认证,ORM基于Spring JDBC封装实现.前后端分离,前端基于React框架实现.使用JSON格式交换数据.

## 后端软件包

|  |  |  |  |
| --- | --- | --- | --- |
| **软件包名称** | **版本** | **用途** | **备注** |
| OpenJDK | 21 | Java运行平台 |  |
| SpringBoot | 3.5.0 | 应用服务器 |  |
| MySQL | 8.0.25 | 数据库 |  |
| Redis | 8.0.0 | 缓存,锁和消息服务器 | 非必选 |
| Maven | 3.5.0 | 项目构建 |  |
| Git |  | 版本控制 |  |
| ElasticSearch | 9.0.1 | 全文检索 | 非必选 |

## 前端软件包

|  |  |  |  |
| --- | --- | --- | --- |
| **软件包名称** | **版本** | **用途** | **备注** |
| Node | 14.15.4+ | 前端服务 |  |
| Yarn | 1.22.5+ | 包管理工具 |  |
| React | 16.13.1 | 基础框架 |  |
| Ant-design | 4.9.2+ | UI框架 |  |
| 其他参见package.json |  |  |  |

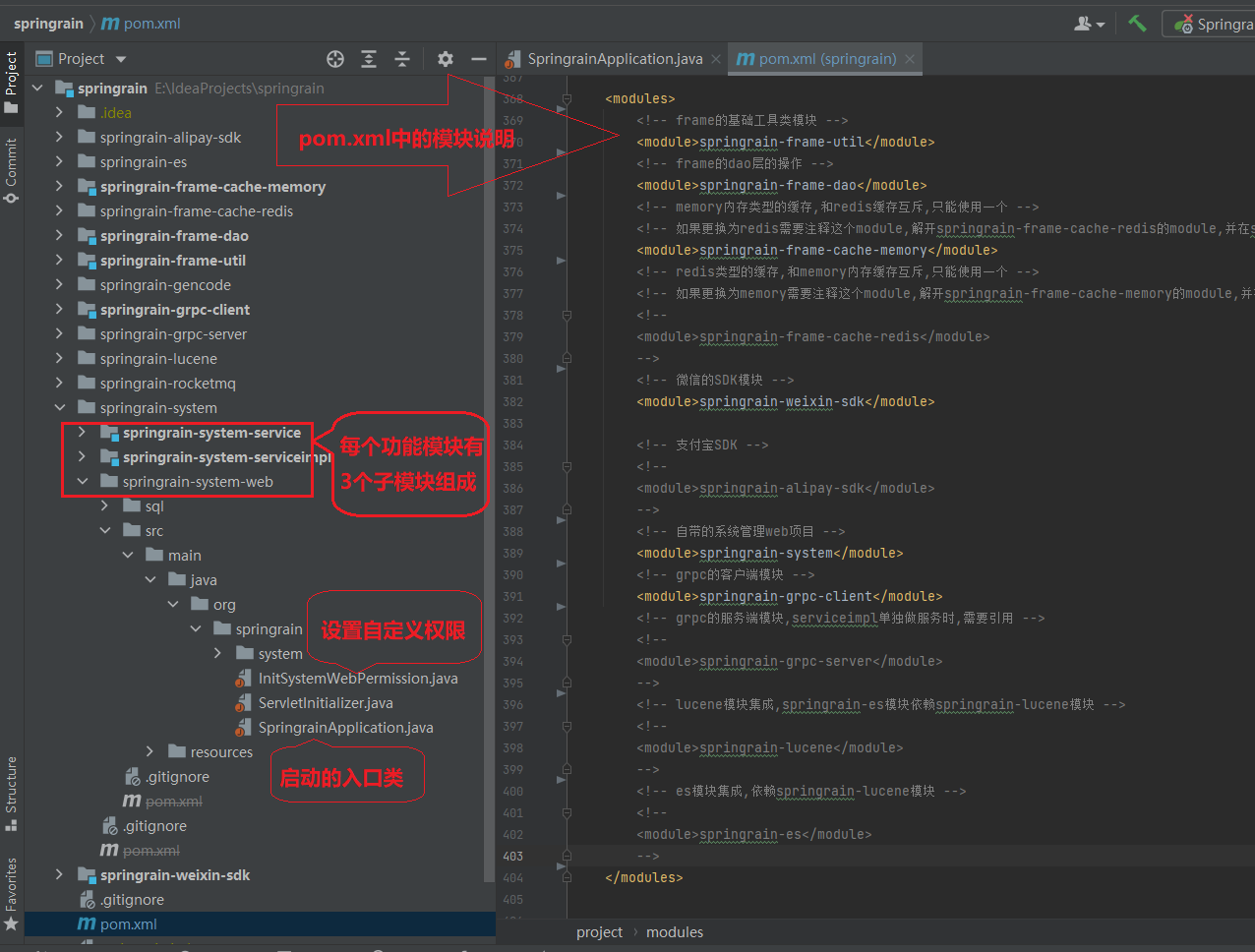
## 运维部署环境

# 后台说明

## 模块说明

平台是标准的Maven项目,直接导入到开发工具即可,例如IDEA.本文档依system模块为例拆解说明.

平台基于maven多模块拆分,模块说明如下图:



导入到IDEA中,运行org.springrain.SpringrainApplication启动类,访问 http://IP:8080/system/api/checkHealth,返回正常响应页面.

22222

注意:

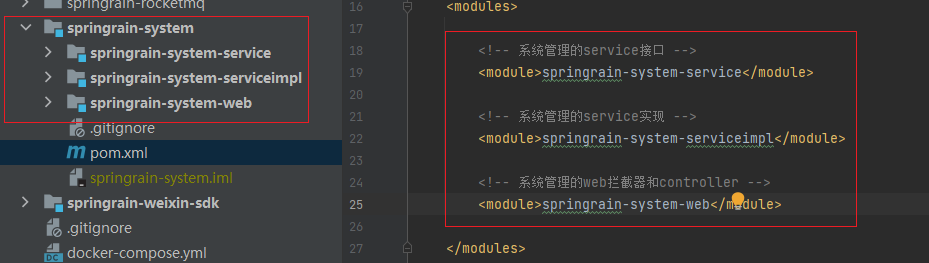
1. memory内存类型的缓存,和redis缓存互斥,只能使用一个

默认使用springrain-frame-cache-memory如果更换为redis需要注释这个module,解开springrain-frame-cache-redis的module,并在springrain-system-web的pom中依赖修改为 springrain-frame-cache-redis

1. serviceimpl单独做服务时,需要引用springrain-grpc-server模块
2. 默认使用springrain-frame-cache-redis 模块中的redis stream实现MQ
3. es模块集成,依赖springrain-lucene模块,所以两个模块依赖的Lucene版本要保持一致.

## 业务子模块拆分

依系统管理的springrain-system为例,实现用户,角色,菜单的权限管理.其他的业务模块开发,可以参照springrain-system拆分子模块.有springrain-system-servcie,springrain-system-servcieimpl,springrain-system-web三个子模块.



Service的接口放到springrain-system-servcie模块,实现放到,springrain-system-servcieimpl模块.

这样做的目的是为了保持接口和实现独立,没有其他耦合依赖,在微服务场景,第三方服务只需要依赖接口模块,实现模块可以独立运行,具体参考 2.15微服务 的说明.

新建模块的时候,也是按照这样的模式拆分,例如 springrain-kaoqin 模块,又拆分三个子项目:

springrain-kaoqin-service,springrain-kaoqin-serviceimpl,springrain-kaoqin-web

springrain-kaoqin-service依赖springrain-system-service

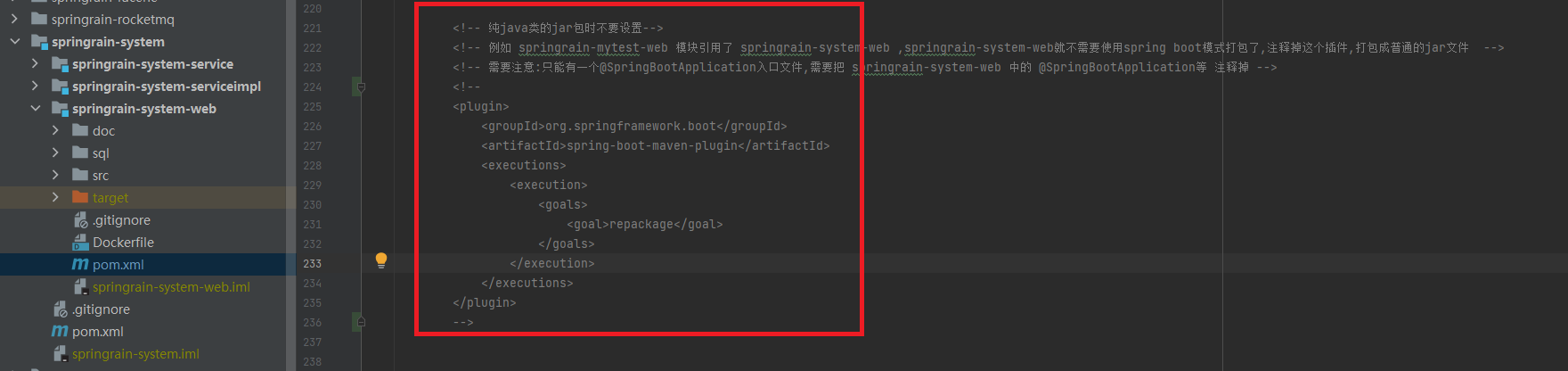
springrain-kaoqin-serviceimpl依赖springrain-system-serviceimpl

springrain-kaoqin-web依赖springrain-system-web

注意:

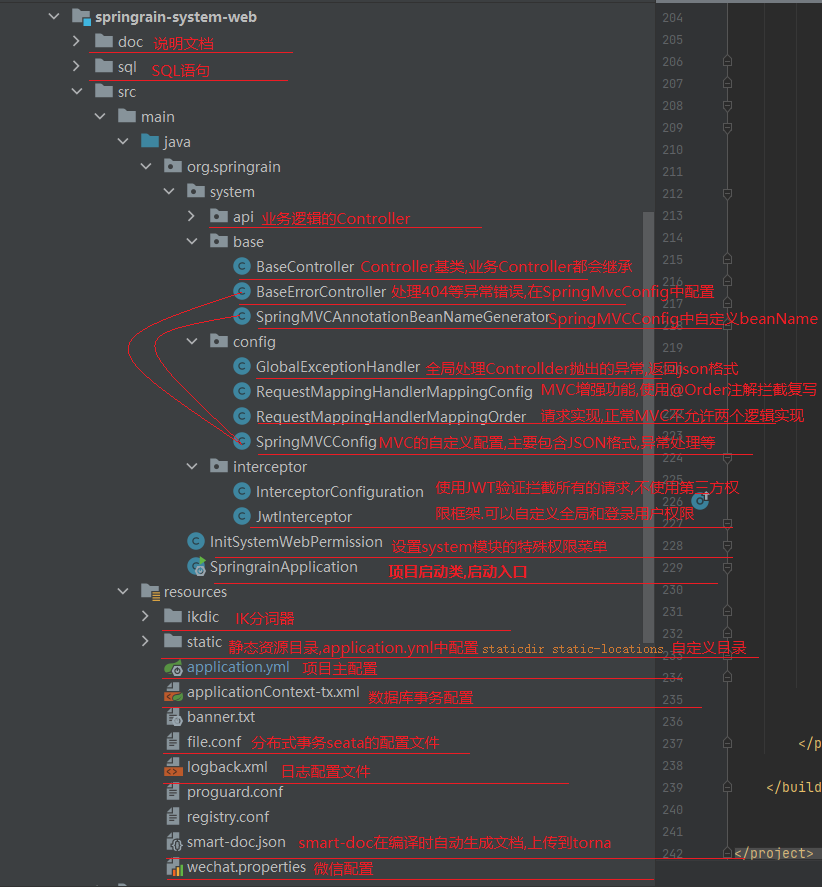
初始化权限的类名要是唯一的,一般带上模块名即可,例如org.springrain.InitKaoqinWebPermission.java

springrain-kaoqin-web中的入口类名称和路径要和springrain-system-web中保持一致org.springrain.SpringrainApplication,就是要覆盖掉springrain-system-web中的入口类.此时springrain-system-web pom.xml中,要注释掉打包插件



## 主要配置

springrain-system-web是权限管理控制的基础模块,其他业务也需要依赖这个模块,所以,项目的核心配置基本都在这个模块里.



## Controller

所有的Controller都返回JSON格式数据,因为前后端分离之后,rest已经没有意义,POST更方便传递JSON,所以不再使用DELETE和PUT方法,默认使用JSON.

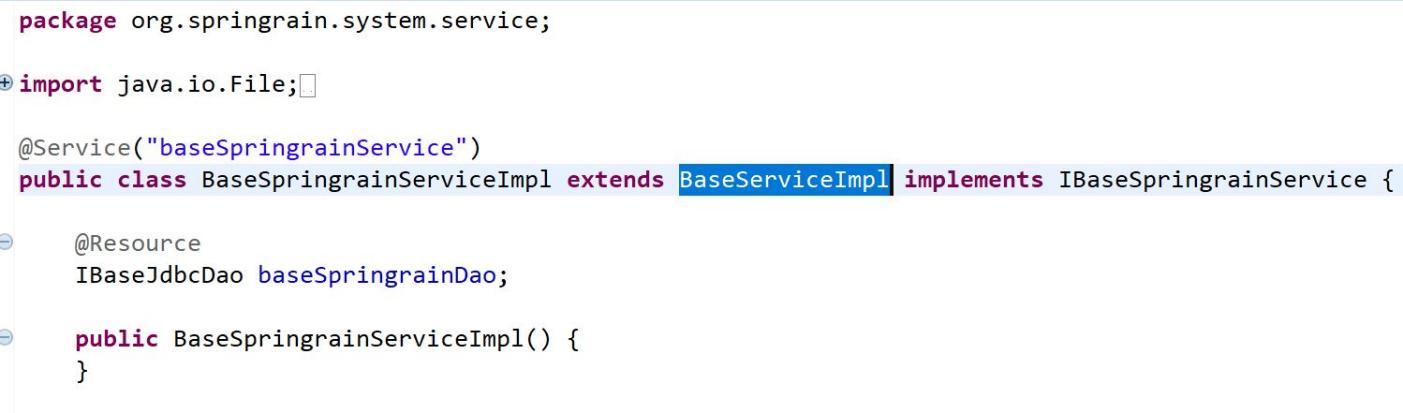
返回值统一使用org.springrain.frame.util.ReturnDatas对象,因为要根据Controller注释和参数生成文档,所以Controller方法的注释要详细,入参和返回值不允许使用Map等不确定类型,要使用Dto对象.例如:

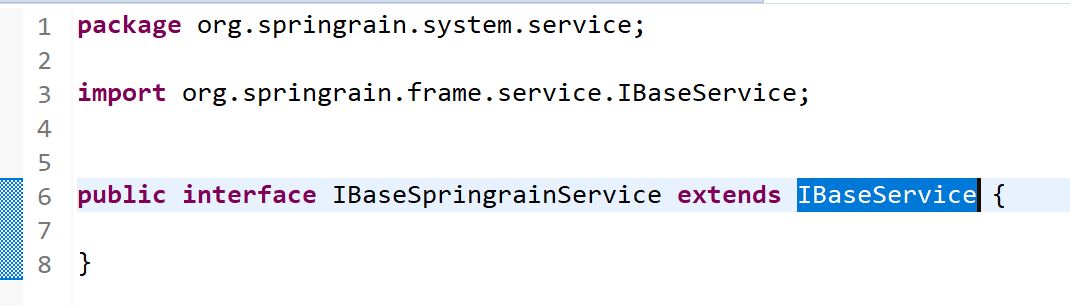


## Service

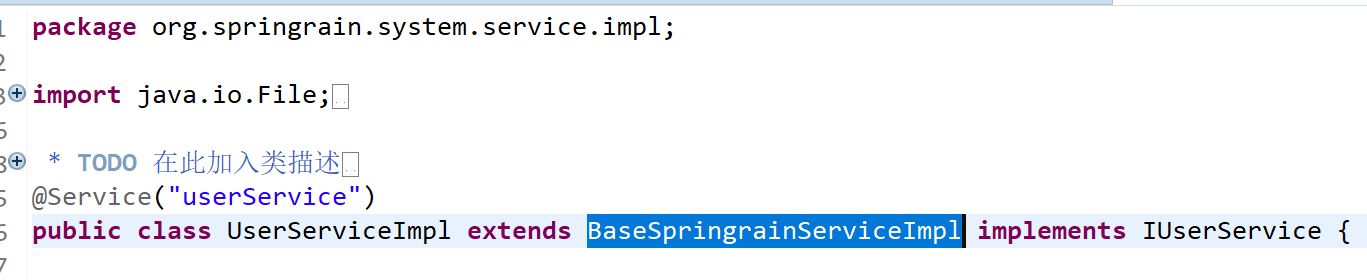
### 实现思路

每个数据库都会基本的Service,例如下图中的baseSpringrainService,基本的Service需要继承BaseServiceImpl这个Service基类,实现IBaseService这个基本接口.





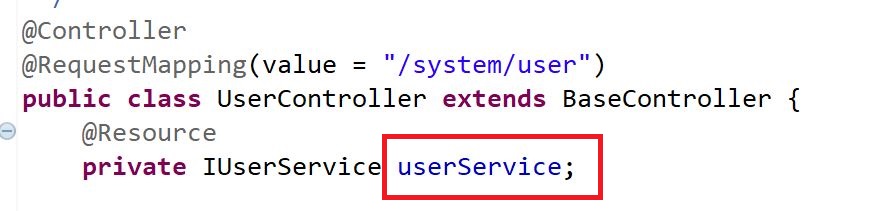
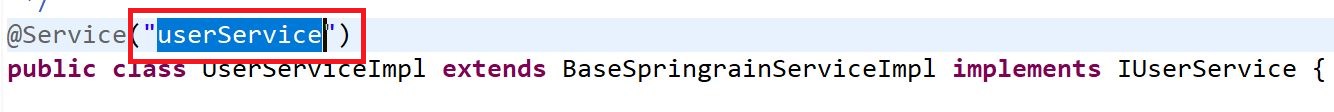
继承基本Service(baseSpringrainService)派生业务service,例如 userService



IBaseService已经提供了基本的操作方法,包含增删改查,可以看下接口和实现.

Service在注入使用时,注入的属性名必须和Service注解名字相同.

例如UserController使用了userService,需要保持一致



### 事务规则

Servcie方法命名必须严格按照事务拦截的规则,如果一个方法内有事务操作,主方法必须符合事务命名规则.事务拦截配置在:applicationContext-tx.xml



方法名 save,update,delete 开头的会被事务拦截,如果更新操作内没有事务,操作会抛出异常.只要业务操作内有事务操作,执行入口的方法名就必须符合拦截的规则.

例如:

public void saveLog(){

finda();

findb();

save();

}

因为有一个save操作,所以方法名是saveLog,依save开头.如果不符合规范,会执行抛错



### 常用方法

#### 查询列表对象

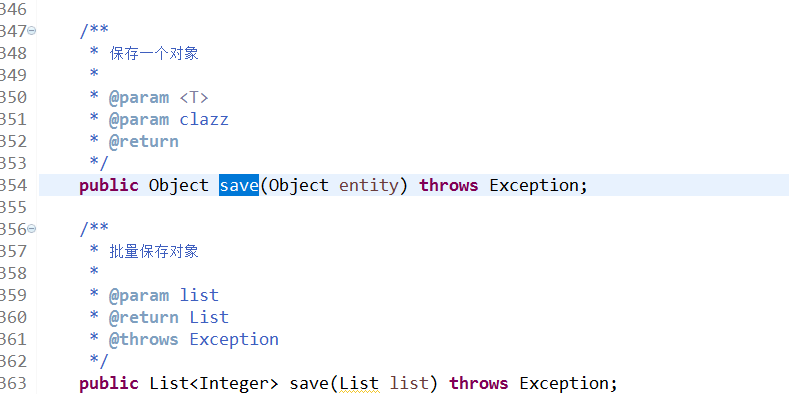
这里只列出部分常用API方法,具体还请查看接口方法注释doc:



#### 查询单个对象



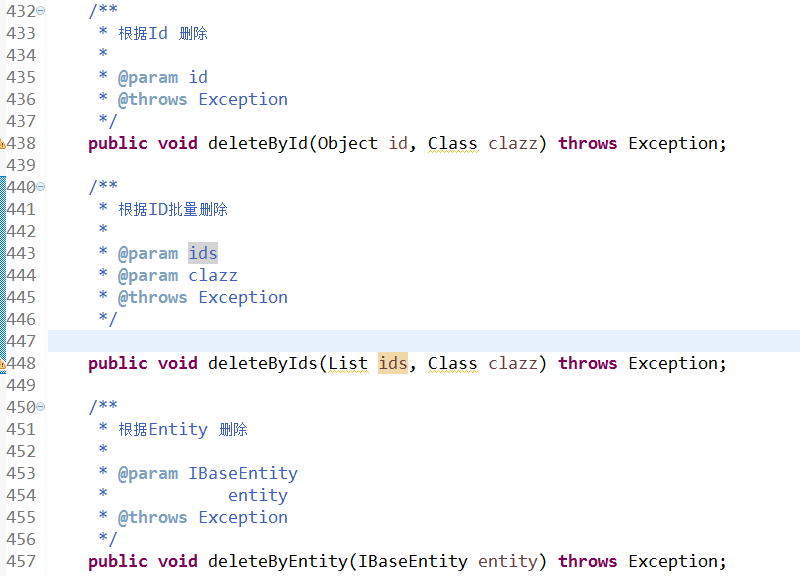
#### 保存



#### 更新



#### 删除



#### SQL语句工具类方法



## Dao

IBaseService的实现只是封装了Dao层方法,IBaseJdbcDao的数据库操作接口和IBaseService接口保持一致.

### 一个数据库只有一个Dao

一个数据库只有一个Dao,业务通过Service实现,例如userService

每个数据库的Dao需要继承抽象类BaseJdbcDaoImpl,实现IBaseJdbcDao接口.

不同的数据库需要注入不同的数据库方言(IDialect接口)实现,默认使用MySQL,所以注入了mysqlDialect

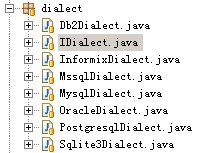
例如baseSpringrainDao:



### 数据库dialect

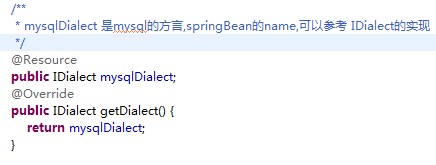
只有在查询分页时需要考虑数据库差异,

实现org.springrain.frame.dao.dialect.IDialect接口即可.



|  |  |  |
| --- | --- | --- |
| 数据库名称 | 实现分页函数 | 说明 |
| SQLServer | OFFSET ROWS FETCH NEXT | 支持sql2005+的版本,不支持sql2000 |
| Oracle | OFFSET ROWS FETCH NEXT |  |
| Informix | SKIP FIRST |  |
| PostgreSql | limit |  |
| SQLite3 | limit |  |

在dao中注入实现的spring bean 即可.



### 使用多个数据库

1. 在 application.yml 中添加数据库的连接字符串和账号密码
2. 拷贝改(或者写到同一个配置文件)

org.springrain.config.DataSourceConfig和 applicationContext-tx.xml的配置文件,例如 新增 org.springrain.config.DataSourceConfig2 和 applicationContext-tx2.xml

1. 创建基本的Dao和Service

在dao中注入配置文件声明的 NamedParameterJdbcTemplate和SimpleJdbcCall,名称和spirng Bean一致.

## Entity

Entity默认包是org.springrain.frame.entity.BaseEntity为基础父类,所有的实体Entity必须继承BaseEntity

### 使用的注解

@Table为映射的表名

@TableSuffix分表后缀.值为获取分表后缀的字段,在save 或者update 对象操作时,可以根据对象的属性值确定分表的后缀.参见 org.springrain.demo.entity.AuditLog

@NotLog 不记录日志  
@Id为主键ID,放在字段的get方法上,可以支持UUID和自增,默认为UUID  
@Transient放在字段的get方法上,标示数据库不存在的属性  
@WhereSQL,拼装sql的where条件,对于简单查询,enity可以直接作为querybean作为查询条件.

最后通过org.springrain.frame.dao.BaseJdbcDaoImpl.getFinderWhereByQueryBean(Finder, Object)

拼装where条件,Object 形参就是QueryBean ,默认为Entity.

通过 org.springrain.frame.dao.BaseJdbcDaoImpl.getFinderOrderBy(Finder, Page)可以拼接前台界面拼接的order by

~FN32H_RW~YENXBU(98N$JEA~{8GNFSYJD}`R8(9OH~SHE  
Entity的属性名需要和数据库完全一致,也可以在拼写sql语句时起别名.

@PKSequence,处理数据库sequence的主键自增,这个注解必须和@Id配合使用,当Number类型的主键值为null时,会取值@PKSequence 下面的是oracle的序列

![L$RK9X2{]O{Z]%Z}](]9IGP](data:image/jpeg;base64,)

最终执行的语句 类似如下

insert into testtable(id ,name) values(test.nextvalue,"testName");

### 使用别名

当使用Finder拼装语句时,可以使用setFrameTableAlias设置Entity作为QueryBean的SQL别名.

## Finder

Finder 是封装SQL查询的实体对象,框架中所有的业务SQL语句最终都是通过Finder执行的.Finder的方法返回值基本都是Finder本身,方便链式调用.

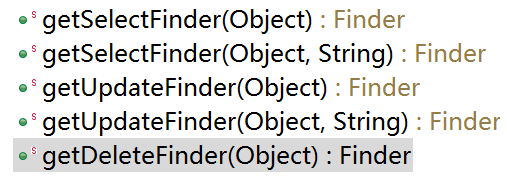
### 直接写SQL语句

最灵活的方式,可以实现你的任何sql语句编写,不允许显示写表名,必须使用Finder.getTableName(Object)方法获取表名!!!!

ZSE$0]_A5Z{S}86VBK@([8S

### 获取具体操作的Finder对象

快捷方法,能够根据实体类获取Finder对象



例如：

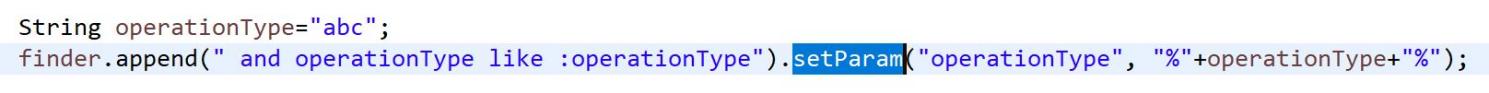
Finder.getSelectFinder(User.class)

Finder.getSelectFinder(User.class，“id，name”)

### like写法

数据库的like查询在框架有两种实现:

* 通过SQL语句实现like



* 通过@WhereSQL注解实现like功能

A~{8GNFSYJD}`R8(9OH~SHE

这种方法底层也是转成了finder的SQL语句进行执行的,只是使用QueryBean时会方便点.

### 使用范例

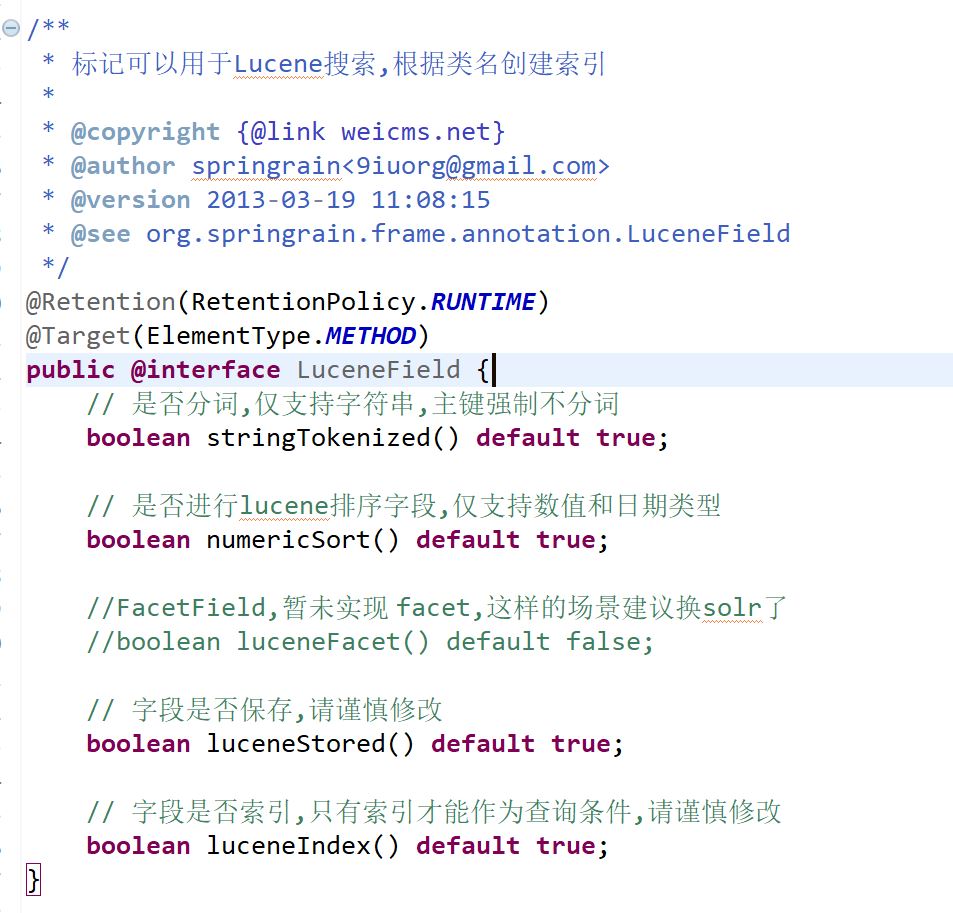


## Lucene

在Entity上使用@LuceneSearch注解标识这个实体类会使用Lucene进行全文检索.底层在对这个类对象进行更新操作时会自动更新Lucene索引文件,注意只能监控到针对对象的更新操作,通过Finder的语句更新无法监控,所以不建议使用@LuceneSearch自动处理索引.

实体类上不使用@LuceneSearch注解,可以通过LuceneUtils手动控制索引的创建和更新.建议这种方式,比较灵活.

字段的get方法上使用@LuceneField标识字段进行索引



通过LuceneFinder控制查询条件.具体参见test.LuceneTest

## 业务权限

### URL请求拦截

通过org.springrain.system.interceptor.JwtInterceptor拦截所有的请求,使用GlobalStatic.excludePathPatterns设置全局不拦截的请求,使用GlobalStatic.userDefaultUrl设置登录用户默认的请求.参照org.springrain.InitSystemWebPermission设置的方法,每个模块都可以定义自己的例外权限.

### 业务数据权限

依据orgId实现业务数据权限,所有的业务表都要有orgId字段,标识本条数据产生时的归属部门.

#### 权限实现的功能

1. 实现上级部门查看下级部门的数据级联
2. 实现用户自定义部门权限,例如张三可以查看部门A,B,C三个部门的数据,虽然张三不是部门主管
3. 实现不通菜单下不通的部门权限,例如张三在用户管理可以查看8个部门,在审批管理可以查看9个部门
4. 实现人员查看任意人员的权限功能,例如张三要查看李四的数据.

#### 表结构设计

如果是部门主管,是通过人员部门中间表(t\_user\_org)managerType=2字段作为判断依据,根据t\_org的pid形成树形结构,实现部门层级权限.

依据角色(t\_role)为中心,角色关联菜单(t\_menu),分配角色可以查看的菜单.角色关联人员(t\_user),分配人员的角色,这样实现人员到菜单的权限.

角色(t\_role)关联部门(t\_org),实现角色关联的部门列表(t\_role\_org),角色(t\_role)中roleOrgType字段标识角色关联部门的类型,值可以是:0自己的数据,1所在部门,2所在部门及子部门数据,3.自定义部门数据,4全部权限

根据角色就可以获取到对应的部门列表了,多个角色取合并集合,就能获取当前用户能够看到的所有部门数据,至此实现自定义用户的部门级权限

如果要看指定人员的数据,给当前用户创建一个虚拟部门,把当前用户设置成部门主管,然后把被查看人拉到虚拟部门.

#### 不同菜单下,数据权限不同

每个菜单的url地址必须是唯一的,因为前后端分离,所以前端的菜单的显示不受后台控制,如果菜单地址需要重复出现,让前端做下处理,后端数据库里,菜单地址是唯一的.

角色(t\_role)指定 privateOrg 角色的部门是否私有,0否,1是,默认0.当角色私有时,菜单只使用此角色的部门权限,不再扩散到全局角色权限,用于处理不同菜单不同的数据权限.公共权限时部门主管有所管理部门的数据全权限,无论角色是否分配.私有部门权限时,严格按照配置的数据执行,部门主管可能没有部门权限.私有权限和公共权限分别处理,不能交叉.处理公共权限时会跳过私有权限

后台根据请求地址就可以反向查询到菜单ID,根据菜单ID,查找到菜单ID所属的用户角色,优先处理私有角色,根据获取的到的角色ID查询到关联的部门数据.就此实现不同菜单下,不同的部门权限.

#### 调用示例



使用IUserRoleOrgService.wrapOrgIdFinderByFinder(Finder finder,String orgIdColumn,String createUserIdColumn)方法,包装当前登录人的数据集权限Finder对象.需要获取当前登陆人信息,用户必须是登录状态,业务数据表 必须要有 orgId 和 createUserId 这两个字段,用于标识归属部门和创建人,用于权限过滤!!!!

## 安全

### SQL注入

后台主要的安全问题是SQL注入,处理方式是禁止sql语句中包含单引号(‘),避免SQL中拼接字符串参数,规避SQL注入风险,具体实现org.springrain.frame.util.Finder.getSql()方法,因为所有的SQL都是通过Finder执行,可以全局判断SQL语句,发现即中断执行.

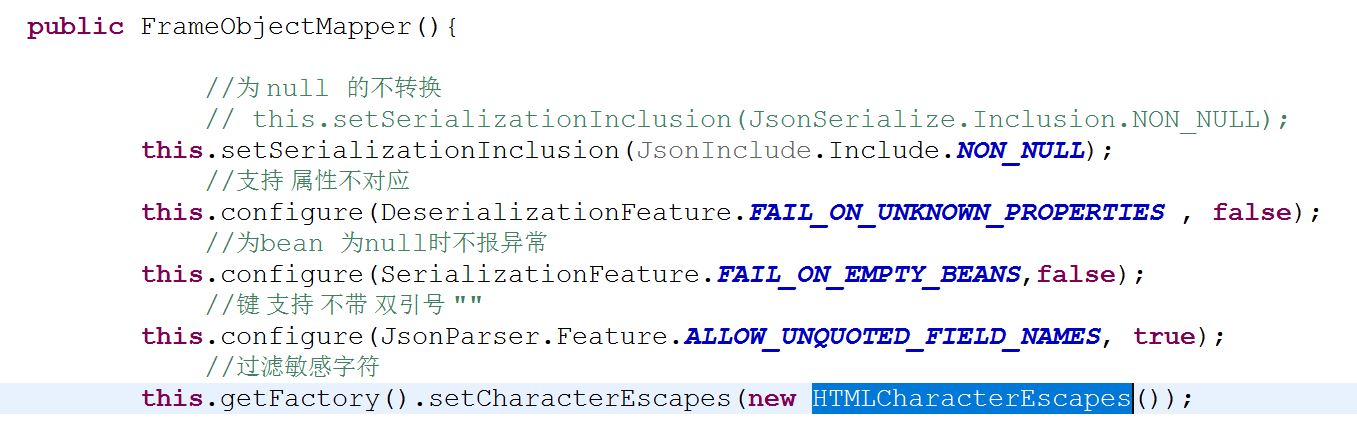
Finder中预留了特殊情况的处理开关,通过查看方法引用,就可以查看到特殊情况,若无必要,会责令修改.

### XSS防护

XSS是最常见的js攻击方式,具体实现百度.....

针对XSS防护,最基本的是特殊字符转义和过滤.一般有,转义后保存和渲染时转义,我们使用了渲染时转义.

参见:org.springrain.frame.util.FrameObjectMapper,转义特殊字符



富文本编辑是注入高发区,需要过滤用户输入的html代码标签,我们使用jsoup,具体实现:

org.springrain.frame.util.InputSafeUtils.filterTextContent(String)



### CSRF

整体使用JWT验证方式,天然避免CSRF攻击.

## 缓存

缓存是系统性能的生死线......

一般缓存系统的结构是Map<String,Map<String,Object>>

平台默认使用基于内存的实现springrain-frame-cache-memory,和redis的实现互斥,只能在pom.xml的module中引入其中一个.如果使用redis需要注释这个module,解开springrain-frame-cache-redis的module,并在springrain-system-web的pom中依赖修改为 springrain-frame-cache-redis.

使用手动控制,不使用Spring的@Cacheable 和@CacheEvict注解注解控制缓存,手动控制更灵活.

已经在底层org.springrain.frame.service.BaseServiceImpl实现,每个继承的service都可以直接使用



针对分页的Page对象,定义了专门的API方法,用于同时缓存Page对象



## 数据库读写分离

### 事务粘性

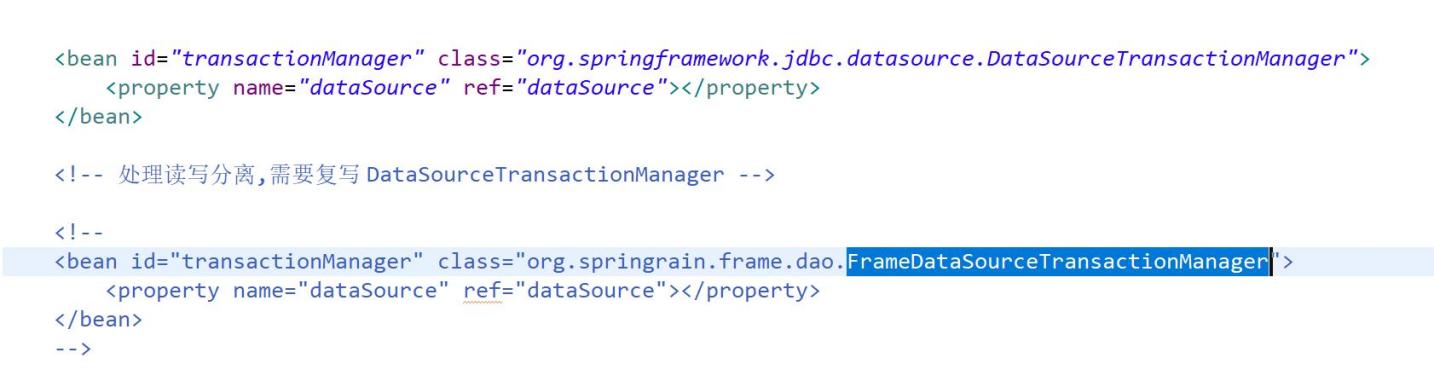
一个事务内的操作在同一个数据库,避免读写分离的延迟问题.推荐!!!

如果有事务就使用写库操作,没有事务就使用读库操作.

修改DataSource的实现配置,使用TransactionDataSourceRouter做为DateSource,注入dataSourceWrite和dataSourceRead两个标准的DateSource,核心是基于Spring的AbstractRoutingDataSource实现数据源的动态切换.



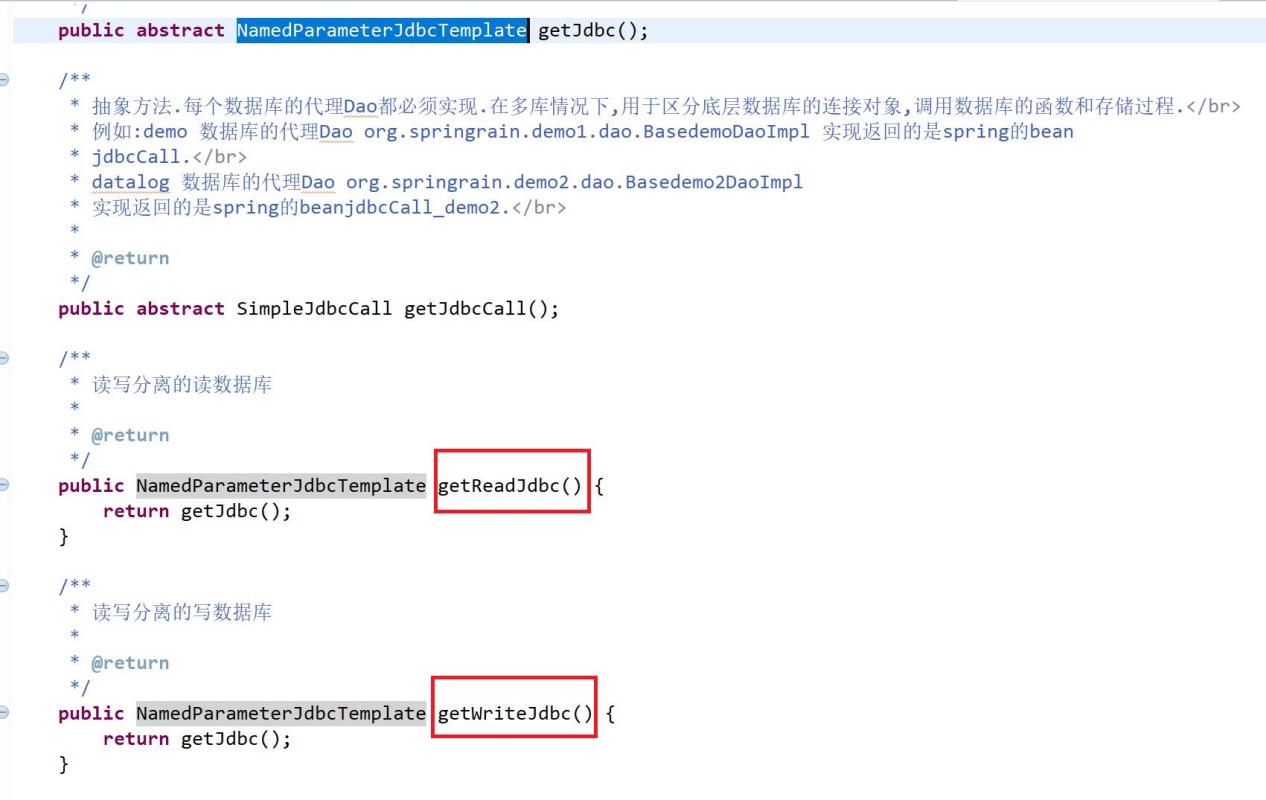
使用FrameDataSourceTransactionManager替换默认的事务管理器.



### 读写强制切换

不考虑事务,强制读写切换,理论性能较好,风险较高,不推荐.

是基于底层Dao,注入读写的NamedParameterJdbcTemplate实现.



## 代码生成器(springrain-gencode)

在平台下有springrain-gencode模块文件夹,generator.xml是配置文件,rapid-gen.bat是执行生成代码的命令文件

#### 原理

.java .html .css .js 本质上都是文本文件,可以通过freemarker模板文件,生成对应的java,html,css和js文件.

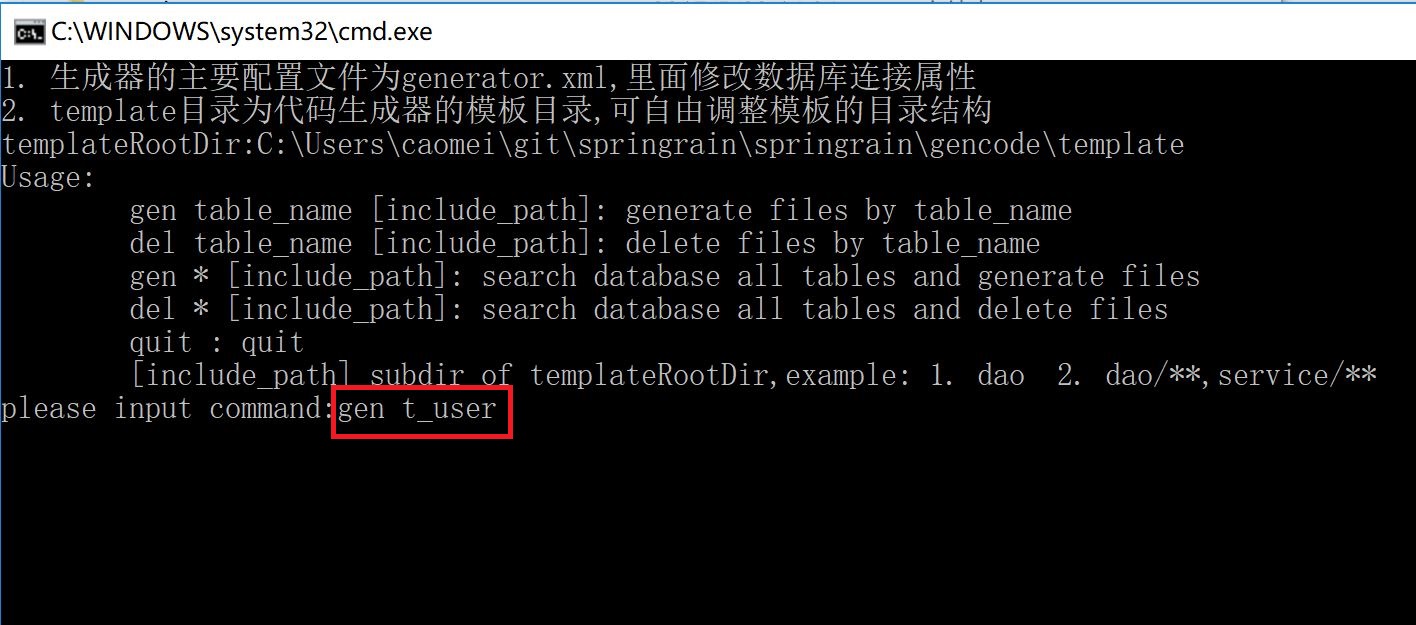
通过数据库连接,获取数据库表的元数据,例如:表的字段名称,中文注释等,依此生成对应的增删改查功能.

#### 使用

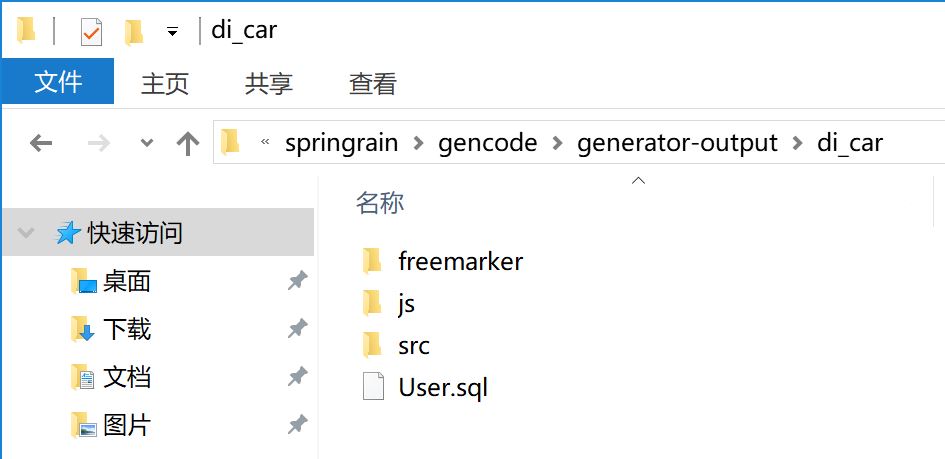
生成器的这

在文件夹中点击运行springrain-gencode/rapid-gen.bat ,输入 gen +表名 回车运行

例如gen t\_user 生成t\_user表的增删改查



回车生成代码,代码在springrain-gencode/generator-output目录下,会自动弹出目录



执行代码生成命令(例如: gen t\_user)后,生成的代码对应拷贝如下:

di\_car/src 对应拷贝到 src,包含Entity,Service和Controller

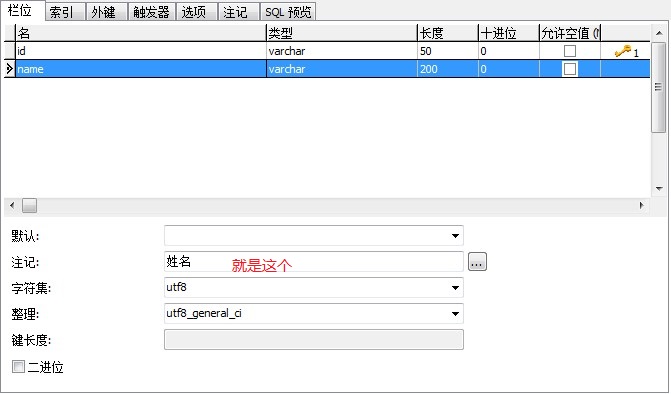
生成的.sql文件是菜单添加的sql,一般根据业务修改后在数据库执行.

重点说明:这些代码只是按照默认规则和样式生成的增删改查代码,需要根据实际业务进行修改.

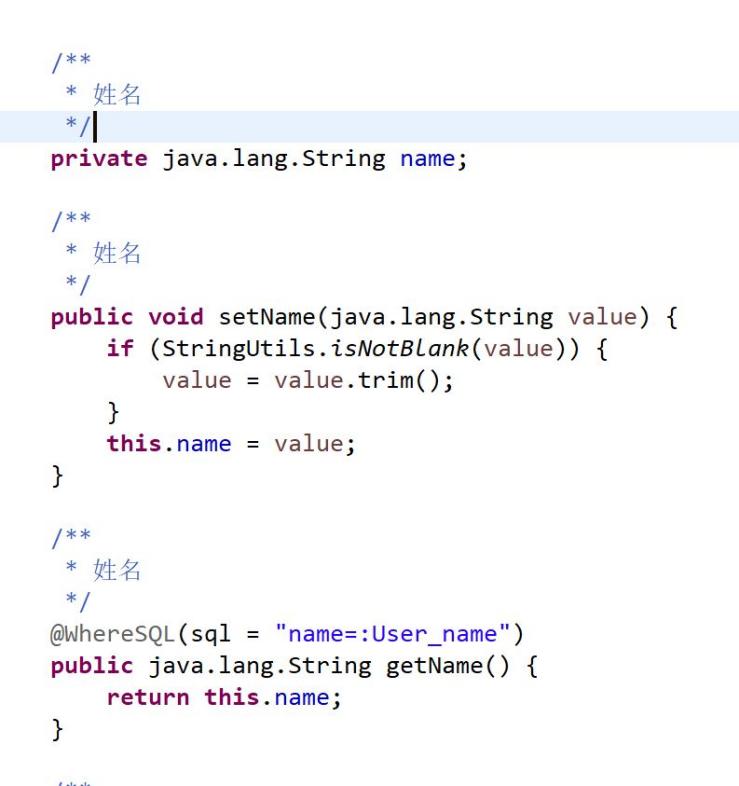
#### 维护数据库注释

代码生成器从数据库取值字段的说明,例如:t\_user表中,字段 name的注记(备注)是 "姓名",这样就会生成Entity

维护好数据库中字段的备注说明,这样生成的代码会友好很多.



生成的Entity代码



## 微服务

参考: <https://jiagou.com/post/58-cloud-native-service-mesh/>

### 说明

springrain实现功能模块拆分,根据业务需要,选择不同的依赖,例如springrain-frame-cache-memory 和 springrain-frame-cache-redis 缓存组件.

根据场景修改Maven打包POM依赖,隔离业务代码的影响  
每个业务模块(例如[springrain-system](https://gitee.com/chunanyong/springrain/tree/master/springrain-system))里有service接口,service实现,web三个子模块项目,隔离相互的关联性.service接口和service实现对类的命名和路径由严格的规范要求. 通过Maven的POM配置,按需引用依赖的模块.  
例如在微服务方式下,服务B依赖服务A,服务B的POM只需要依赖服务A的service接口项目,不依赖服务A的service实现.如果是单体项目,依赖服务A的service实现即可.

### 实现思路

启动加载springbean时,先检查本地是否有实现,如果没有就启动GRPC远程调用,如果开启了GRPC,就会调用Seata的配置,同时开启分布式事务.(开发人员无感知)

基于seata分布式事务实现.支持有注解和无注解(底层记录日志)混合使用.(开发人员无感知)

基于K8S的Service实现服务注册和发现,ConfigMap实现配置中心.(开发人员无感知)

基于Istio实现微服务的发现,监控,熔断,限流.(开发人员无感知)

### 限制

接口和实现的命名强制规范.

一个RPC接口只能有一个实现.

分布式事务,一定要避免A服务update表t,RPC调用B服务,B服务也update表t.这样A等待B结果,B等待A释放锁,造成死锁.

Service层不可以使用Servlet API,例如 HttpRequest

### 实现代码

在service mesh,业务系统只需要处理rpc和分布式事务,其他的微服务功能由istio完成.所以重点说下对springrain的设计实现.

根据项目情况,最小只需要springrain做为单体项目裸机运行,不使用k8s和istio及相关组件,把运维成本降到最低;复杂点可以使用Nginx负载GRPC协议,更复杂的就是K8S+Istio了……

#### 项目启动加载SpringBean

在[springrain-grpc-client](https://gitee.com/chunanyong/springrain/tree/master/springrain-grpc-client)模块中[org.springrain.rpc.springbind.GrpcBeanFactoryPostProcessor](https://gitee.com/chunanyong/springrain/blob/master/springrain-grpc-client/src/main/java/org/springrain/rpc/springbind/GrpcBeanFactoryPostProcessor.java) 在springbean容器初始化前,会检查业务service的接口和对应实现,如果service有接口也有实现,可以认为是本地模式,不做处理.  
如果只有接口没有实现,认为是RCP远程调用,会获取service接口上的[RpcServiceAnnotation](https://gitee.com/chunanyong/springrain/blob/master/springrain-grpc-client/src/main/java/org/springrain/rpc/annotation/RpcServiceAnnotation.java)注解,获取到远程的RCP地址和端口,然后为这个service做一个代理,让spring正常加载代理bean,代理bean再RPC调用远程,返回结果.这样就对业务逻辑代码是透明的,业务逻辑代码只有在运行时,才知道是本地运行还是RPC调用.

#### 事务处理

如果是本地运行,使用本地普通事务,不做任何处理.如果是RPC调用,根据配置,设置开启Seata事务,使用Seata的AT模式,自动回滚,不写补偿事务.



设置开启事务之后,就会在[springrain-frame-dao](https://gitee.com/chunanyong/springrain/tree/master/springrain-frame-dao)模块的[org.springrain.frame.config.DataSourceConfig](https://gitee.com/chunanyong/springrain/blob/master/springrain-frame-dao/src/main/java/org/springrain/frame/config/DataSourceConfig.java)类中设置dataSource



使用[springrain-frame-dao](https://gitee.com/chunanyong/springrain/tree/master/springrain-frame-dao)模块的[org.springrain.frame.dao.SeataDataSourceTransactionManager](https://gitee.com/chunanyong/springrain/blob/master/springrain-frame-dao/src/main/java/org/springrain/frame/dao/SeataDataSourceTransactionManager.java),实现本地事务和Seata的XID绑定,隔离本地事务和Seata事务的切换过程.

使用了Seata代理的DataSource,开启了分布式事务.需要把Seata服务端启动,具体参考[Seata的官方文档](http://seata.io/zh-cn/).

#### GRPC调用

没有找到本地service实现的时候,就会开启GRPC调用,通过注册SpringBean的代理类,实现业务代码隔离.  
在[springrain-grpc-client](https://gitee.com/chunanyong/springrain/tree/master/springrain-grpc-client)模块中使用通用的[grpcCommonService.proto](https://gitee.com/chunanyong/springrain/blob/master/springrain-grpc-client/src/main/proto/grpcCommonService.proto)文件,把请求参数封装成[org.springrain.rpc.grpcimpl.GrpcCommonRequest](https://gitee.com/chunanyong/springrain/blob/master/springrain-grpc-client/src/main/java/org/springrain/rpc/grpcimpl/GrpcCommonRequest.java)对象,作为序列化参数,放到grpcCommonService通用模板.  
在[springrain-grpc-server](https://gitee.com/chunanyong/springrain/tree/master/springrain-grpc-server)中使用[org.springrain.rpc.grpcimpl.CommonGrpcService](https://gitee.com/chunanyong/springrain/blob/master/springrain-grpc-server/src/main/java/org/springrain/rpc/grpcimpl/CommonGrpcService.java)响应GRPC调用,处理grpc事务,同时把SessionUser这个载体处理好.

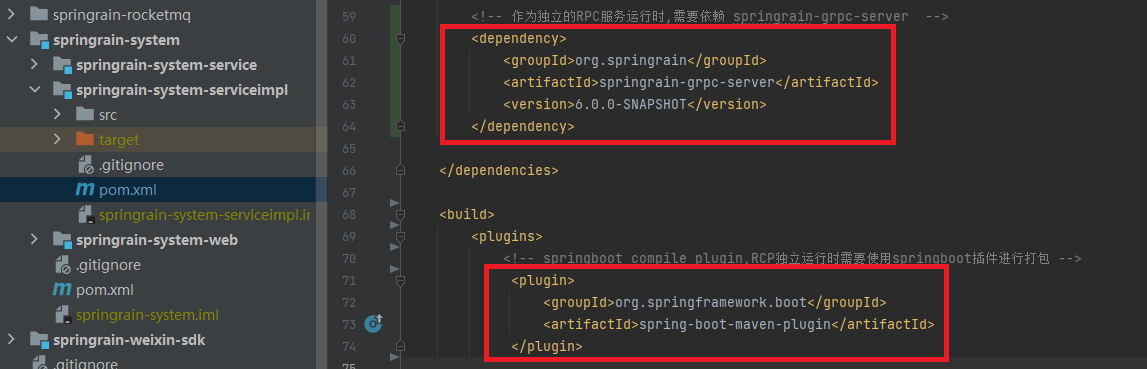
一般是业务系统只调用第三方系统,只依赖[springrain-grpc-client](https://gitee.com/chunanyong/springrain/tree/master/springrain-grpc-client)就可以了,如果也提供RPC服务也需要依赖[springrain-grpc-server](https://gitee.com/chunanyong/springrain/tree/master/springrain-grpc-server)模块.

#### POM依赖例子



#### serviceimpl独立运行

微服务模式下,service实现需要独立运行,例如springrain-system-serviceimpl模块的pom.xml配置如下图,需要依赖springrain-grpc-server模块,使用springboot的maven打包插件



#### 效果演示

从业务代码已经无法感知是本地调用还是GRPC了.



## 规范约定

### 不允许改动frame包下的文件

### 不允许SQL拼接前台参数,必须使用占位符

**错误写法!SQL注入啊,大哥!!!!前台数据都是骗人的啊!!!**

}CM]LO$EPE09K9{$SIT@DHK

**正确写法:**

A@51C(Y275ZQ5{OX[07X4~P

### 不允许手写表名，只能使用Finder工具类获取表名

手动拼写容易写错,而且还有数据库大小写的问题,如果数据库表名做了修改,后果是灾难性的.

Finder工具类更方便书写,IDE也会有提示校验,安全稳定效率好.

### Controller编写规范

Controller中不能编写SQL语句,SQL语句应在Service中进行编写.

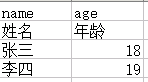
原则上不允许新增新的逻辑方法，以代码生成为准。

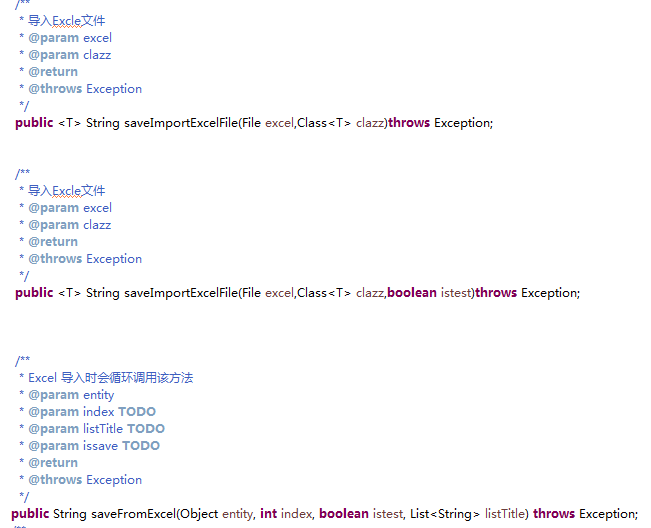
只能返回String，void，ReturnDatas。

所有的数据库语句写在Service，Controller 不允许写SQL语句

/json,/pre,/more 会被截取然后在判断权限,/update/pre 的权限等同 /update

/\*\*/ajax/\*\*=user,URL中包含/ajax/的路径只要是登陆用户都可以访问,特殊的URL配置权限过滤即可.

导入Excle模板 ,第一行是实体类的属性名,第二行是用户能看懂的中文名,第三行是实际数据.调用Service的方法接口



导出是直接使用了查询的列表模板,通过HTML注释区别特殊情况.实现了查询和导出的数据格式统一.

### Service编写规范及常用方法

业务查询方法 增:save开头,删:delete开头,改:update开头,查:find开头.

尽量避免使用数据的特殊函数,例如日期操作的函数.

queryForList

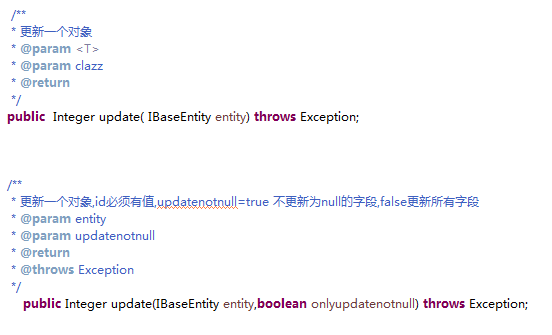
queryForObject

update(Finder finder)

update(List list)

save(Object object)

save(List list)



所有的Serivce方法中不允许出现Servlet API的参数例如 HttpServletRequest等.

### 不允许复写Dao，一个数据库只能有一个Dao

### 不允许手动编写分页函数和其他特定函数

关于分页,如果确定查询一个对象,可以使用queryForObject.

如果是从list中想取出第一条,不要手动编写limit 0,1 可以使用Page对象分页查询,page.setPageIndex(1);page.setPageSize(1);就是查询列表的第一页第一条.这样兼容性好,可以切换到不同的数据库,如果是limit 写死代码,只能使用mysql了!

除了常用的聚合函数:sum,count,max,min,avg之外,原则上不再允许使用其他数据库函数.特别是日期相关的函数,绝大部分情况都可以使用java代码配合实现.

因为数据库资源是最宝贵的,而函数都是很耗数据库资源的,能java代码实现的就不要麻烦数据库了!

# 前端说明

## 目录结构



### 目录规范说明

所有文件夹目录必须使用小写，子项目顶级目录Project除外。

所有组件文件夹目录命名为**components**

所有页面，组件入口文件命名为**index.jsx**

所有页面，组件样式文件命名为**style.less**

组件文件夹命名采用小写+横线的方式：**home-header**

组件文件命名采用驼峰方式命名：**HomeHeader**

## 目录重点文件说明

### 目录位置：src/commons

**Ajax.js:**ajax工具类封装

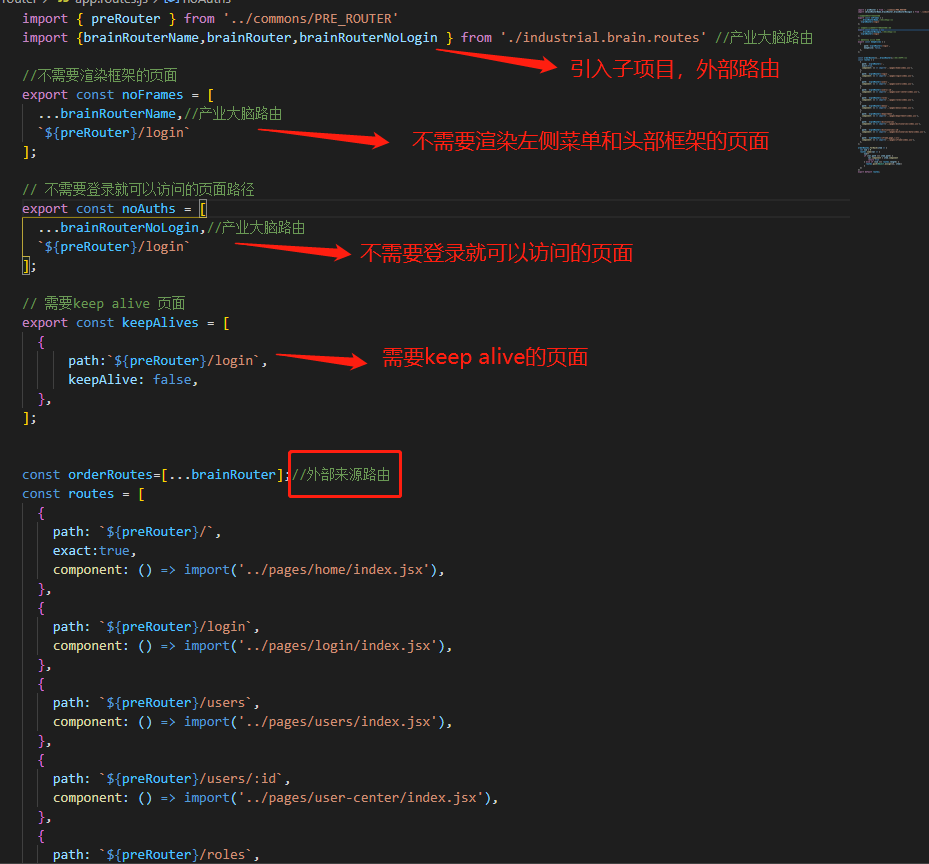
**Config-hoc.js:**高阶组件集合，类装饰器封装函数

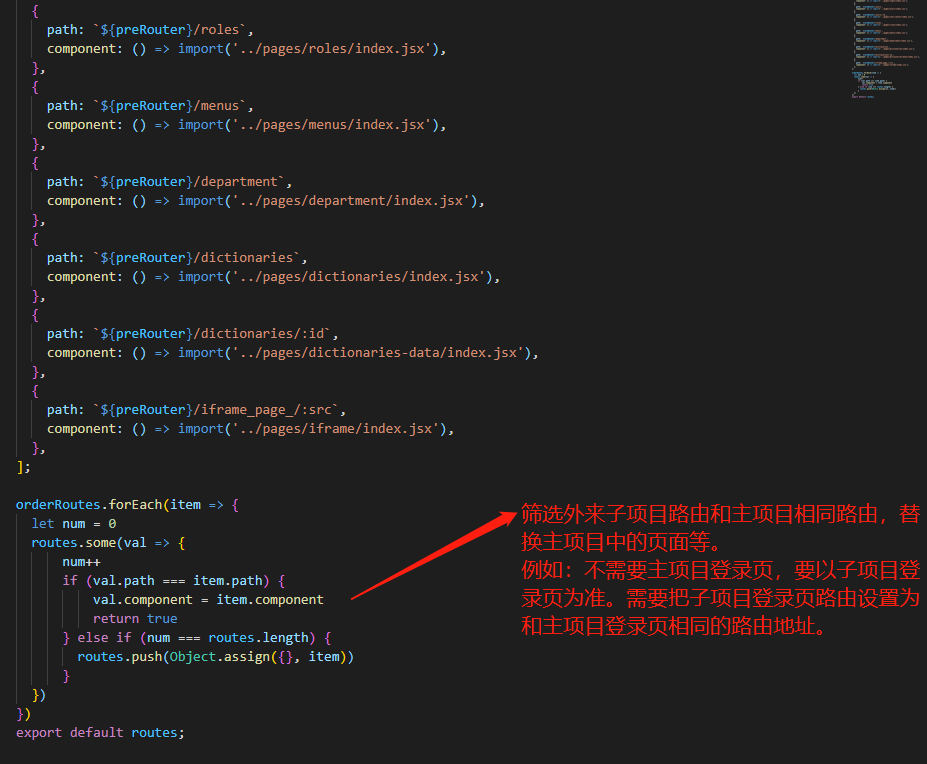
**Index.js:**公共工具类函数封装。一些公共的操作方法。

**PER\_ROUTER.js(特别重要！！)：项目参数配置文件.路由前缀配置，外链前缀配置，请求地址前缀配置，默认登录成功后跳转首页配置。特别重要，子项目部署配置都需要修改该文件。**

### 目录位置：src/router

**App.router.js:**项目路由主配置文件。





**AppRouter.jsx、AuthRoute.jsx、routes.js：**路由处理配置文件。

**industrial.brain.routes.js：**子项目路由文件，命名方式以.分隔，需注意其中要带上子项目文件夹名字。例如：产业大脑子项目文件夹名字为：brain。

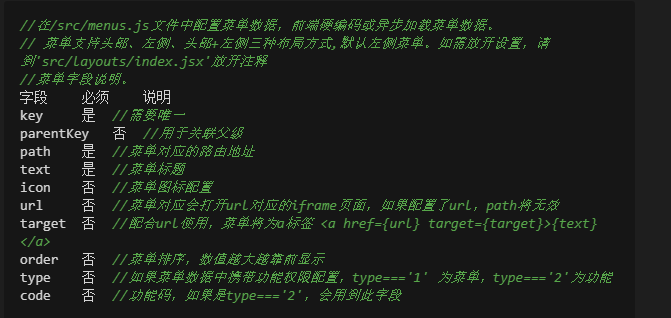
**其他文件，详情见目录说明。**

## 菜单配置

### 目录位置

Src/menus.js,菜单支持三种布局方式，头部+左侧，左侧，头部。默认左侧菜单。具体配置文件在src/layouts/index.jsx。

### 字段说明



## 页面配置

### 页面模板



### 页面配置

装饰器config配置参数：

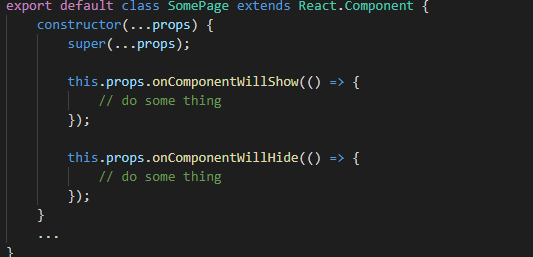
|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **默认值** | **说明** |
| noFrame | boolean | false | 标记当前页面不需要导航框架页面 |
| noAuth | Boolean | False | 标记当前页面不需要登录就可以访问的页面 |
| keepAlive | Boolean | - | 标记当前页面内容在切换页面后是否要保持 |
| title | Boolean,string,  reactNode,  object,function | True | true：当前页面显示通过菜单结构自动生成的title；false：当前页面不显示title；string：自定义title；object：{text，icon} text为显示的名称，icon为图标；function(props): 返回值作为title |
| breadcrumbs | Boolean ,array，function | true | true：当前页面显示通过菜单结构自动生成的面包屑；false：当前页面不显示面包屑；object：[{icon, text, ...}]；function(props): 返回值作为面包屑 |
| appendBreadcrumbs | array 或 function(props) | [] | 在当前面包屑基础上添加；function(props): 返回值作为新添加的面包屑 |
| pageHead | boolean | - | 页面头部是否显示 |
| side | Boolean | - | 页面左侧是否显示 |
| sideCollapsed | boolean | - | 左侧是否收起 |
| ajax | boolean | false | 是否添加ajax高阶组件，内部可以通过this.props.ajax使用ajax API，组件卸载时，会自动打断未完成的请求 |
| router | boolean | false | 是否添加withRouter装饰器，组件内部可以使用this.props.history等API |
| query | boolean | false | 是否添加地址查询字符串转换高阶组件，内部可以通过this.props.query访问查询字符串 |
| connect | Boolean,function | false | 是否与redux进行连接，true：只注入了this.props.action相关方法；false：不与redux进行连接；(state) => ({title: state.page.title})：将函数返回的数据注入this.props |
| event | Boolean | false | 是否添加event高阶组件，可以使用this.props.addEventListener添加dom事件，并在组件卸载时会自动清理；通过this.props.removeEventListener移出dom事件 |
| pubSub | boolean | false | 是否添加发布订阅高阶组件，可以使用this.props.subscribe(topic, (msg, data) => {...})订阅事件，并在组件卸载时，会自动取消订阅; 通过this.props.publish(topic, data)发布事件 |
| modal | string 或 object | false | 当前组件是否是modal。string: 弹框标题；object：弹框配置 |

**特别注意：**

noFrame、noAuth、keepAlive 只有配置了path才有效！config装饰器可以用于任何组件，但是title、breadcrumbs、appendBreadcrumbs、pageHead、side、sideCollapsed最好在路由对应的页面组件中使用。

### 页面显示和隐藏

Config装饰器注入了两个事件，如果页面使用了 keepalive功能，切换显示/隐藏时候才会触发。



### 页面容器PageContent

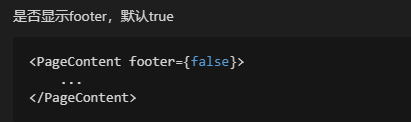
系统提供了页面的跟节点PageContent，有如下特性：

- 添加了margin padding 样式；

- 添加了footer；

- 支持页面loading；

- 自动判定是否有底部工具条FixBottom组件，为底部工具条腾出空间；





### 弹窗页面

添加、修改等场景，往往会用到弹框，antd Modal组件使用不当会产生脏数据问题（两次弹框渲染数据互相干扰）系统提供了基于modal封装的高阶组件，每次弹框关闭，都会销毁弹框内容，避免互相干扰。



Model所有参数说明如下：

如果是string，作为modal的title

如果是函数，函数返回值作为 Modal参数

如果是对象，为Modal相关配置，具体参考 antd Modal组件

options.fullScreen boolean 默认false，是否全屏显示弹框

ModelContent组件，具体参数如下：



## AJAX配置

### Ajax请求

系统的ajax请求基于axios封装。

基于restful规范，提供了5个方法：

- get 获取服务端数据，参数拼接在url上，以 query string 方式发送给后端

- post 新增数据，参数以body形式发送给后端

- put 修改数据，参数以body形式发送给后端

- del 删除数据，参数拼接在url上，以 query string 方式发送给后端

- patch 修改部分数据，参数以body形式发送给后端

### Ajax调用方式

一共三种调用方式：

//第一种 config装饰器ajax属性（推荐）

//第二种 ajax装饰器

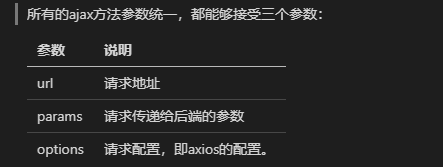
//第三种 直接引入ajax对象



### Ajax配置参数

注：config、ajaxHoc方式做了封装，页面被卸载之后会自动打断未完成的请求

所有的ajax方法参数统一，都能够接受三个参数：



Options配置：

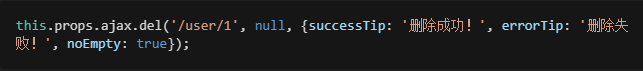


### Ajax请求结果提示

系统对ajax失败做了自动提示，开发人员可通过src/commons/handle-error.js进行配置；

成功提示默认不显示，如果需要成功提示，可以配置successTip参数，或者.then()中自行处理；

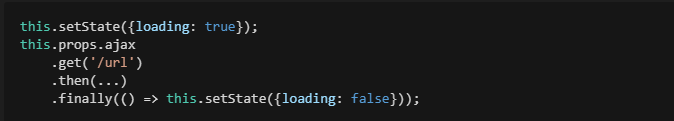
成功提示在src/commons/handle-success.js中配置；



也可以选择ant-design中的message组件（目前项目中最多使用）

### Loading处理

系统扩展了promise，提供了finally方法，用于无论成功还是失败，都要进行的处理。一般用于关闭loading



## Models(redux)封装

### Redux

基于redux进行封装，不改变redux源码，可以结合使用redux社区中其他解决方案。

注：一般情况下，用不到redux~

models用于管理数据，解决的问题：

命名空间（防止数据、方法命名冲突）：数据与方法，都归属于具体model，比如：state.userCenter.xxx，this.props.action.userCenter.xxx();

如何方便的获取数据：connect与组件连接；@connect(state => ({name: state.user.name}));

如何方便的修改数据：this.props.action中方法；

客户端数据持久化（保存到LocalStorage中）：syncStorage配置；

异步数据处理：基于promise异步封装；

请求错误提示：error处理封装，errorTip配置，自动提示；

请求成功提示：successTip配置，自动提示；

简化写法：types actions reducers 可以在一个文件中编写，较少冲突，方便多人协作，参见models/page.js中的写法;

业务代码可集中归类：在models目录中统一编写，或者在具体业务目录中，模块化方式。

### Redux models配置

所有的model直接在models或pages下定义：

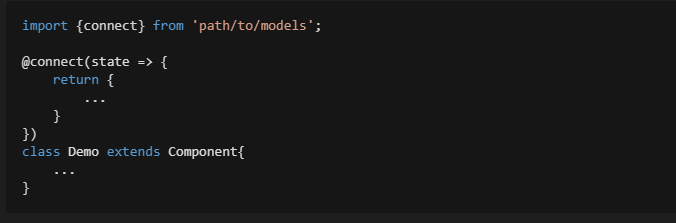
model模块名规则：



### 组件和redux进行连接

提供了多种种方式，装饰器方式、函数调用、hooks、js文件直接使用；

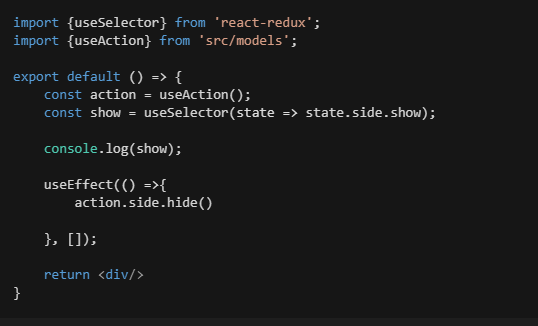
推荐使用装饰器方式：



函数方式：

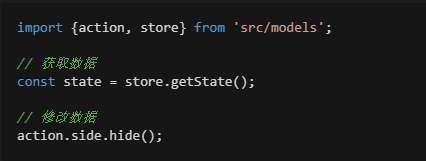


Hooks方式：



js文件中使用：

没有特殊需求，一般不会在普通js文件中使用



redux更多配置详见项目目录README.md

## 权限控制，权限组件

系统菜单、具体功能点都可以进行权限控制。

### 菜单权限

菜单由后端提供（一般系统都是后端提供），后台通过登录用户返回用户的菜单权限；页面只显示获取到的菜单；

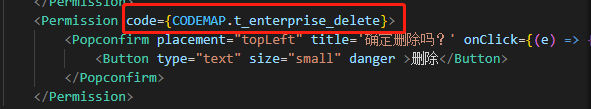
### 功能权限

可以通过`src/components/permission`组件对功能的权限进行控制

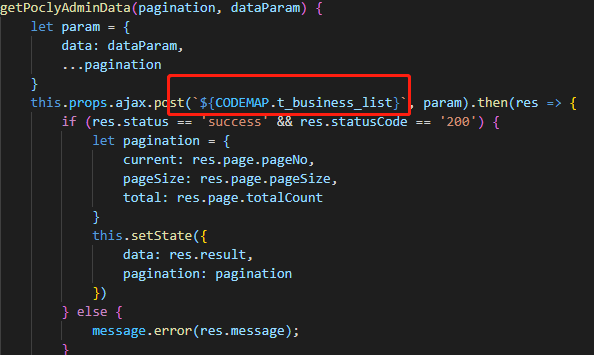


注意：权限的code前端使用时，注意语义化、唯一性。

实现方式：用户登录成功后，会获取其拥有的权限，存储到session storage中。codeMap是拥有的权限code，我们在前端配置功能权限组件时候，需要传值其中的key。



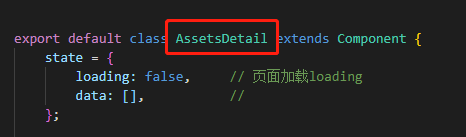
特别注意：所有请求接口，均为动态接口，变量承接。（特殊接口除外）。



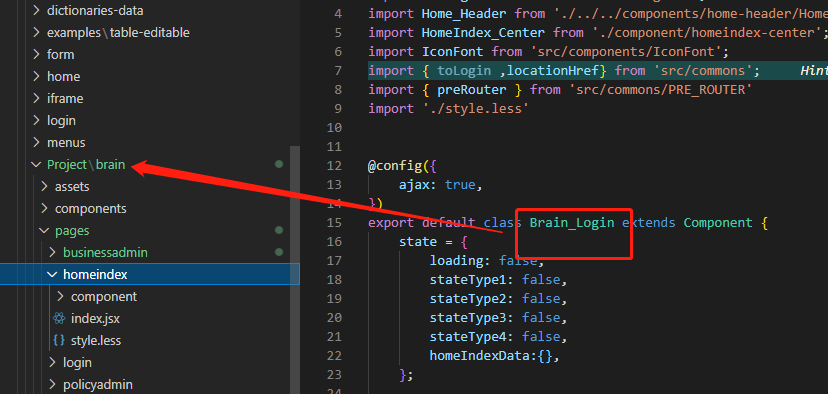
## 其他重点规范

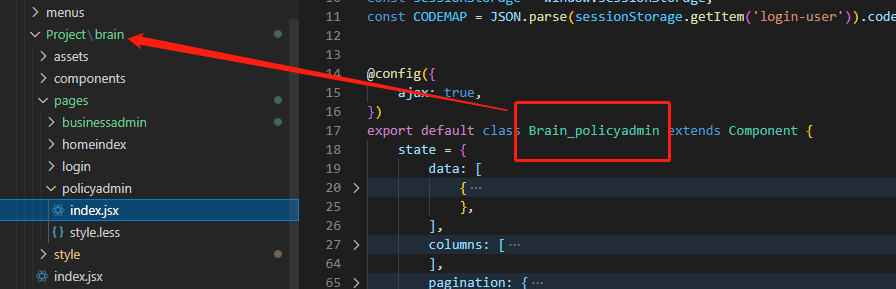
### 组件类命名

主项目system中所有组件命名均采用驼峰方式命名。



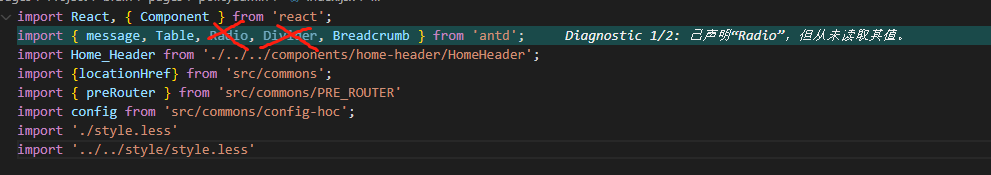
子项目中，所有页面组件和功能组件命名的时候，均要带上子项目目录文件名加下划线的方式，且单词开头大写：例如 Brain\_Homeheader。具体如下图：

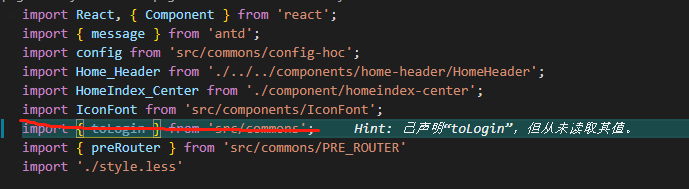




### import引入规范

页面未使用到的，严禁引入！去除页面未使用到的import，未使用到的模块严禁导入。



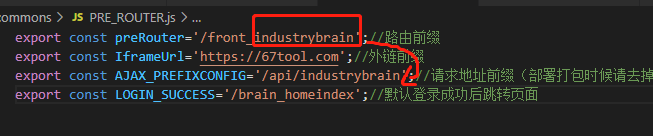


### 打包规范

根据实际使用项目打包，请在路由引入文件中处理，未使用到的子项目请不要在路由文件中引入。严禁打包不需要的子项目！！具体根据实际部署需求。

### 路由前缀命名和请求地址命名规范

前端所有项目前缀路由以front开头+下划线+名称，后半部分名称要和请求地址前缀保持一致。



注意！！请求地址前缀中的api为本地开发环境配置，打包部署时请删除。

### 额外功能

**打印配置：**页面打印，通过给元素添加相应的class，控制打印内容：

- `.just-print` 只在打印时显示

- `.no-print` 在打印时不显示

**支持判断运算符：**

const name = res?.data?.user?.name || '匿名';

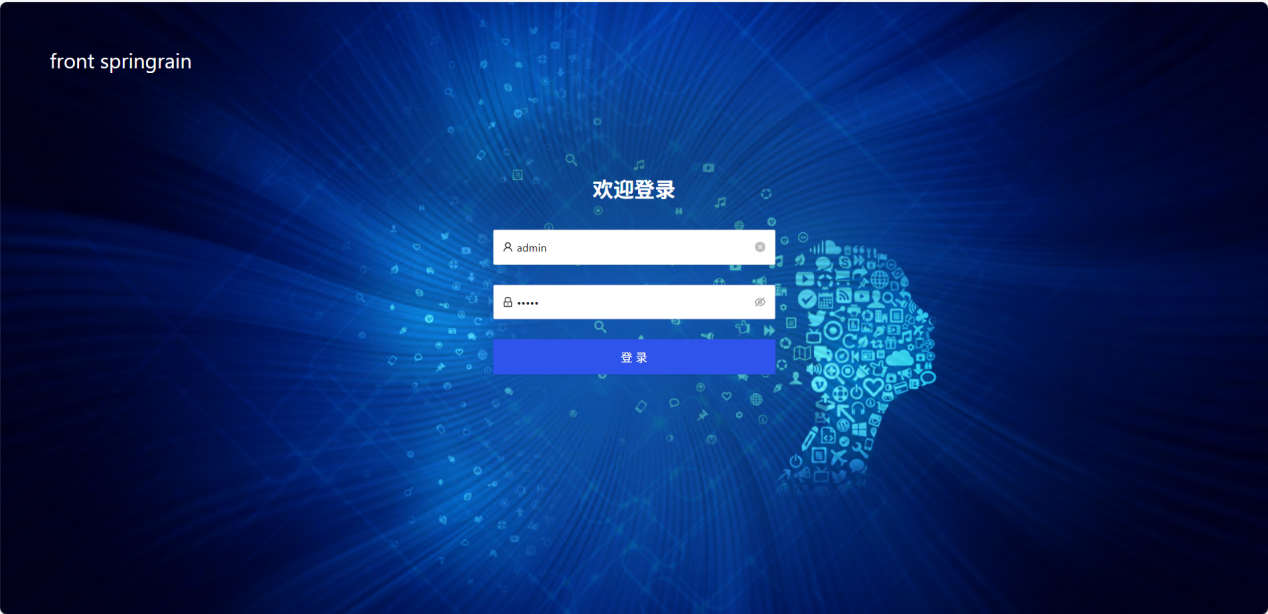
**ra-lib.js,中封装的form，FormElement：**

FormElement:类型有：

'input', 'hidden', 'number', 'textarea', 'password', 'mobile', 'email', 'select', 'select-tree', 'checkbox', 'checkbox-group', 'radio', 'radio-button', 'radio-group', 'switch', 'date', 'time', 'date-time', 'date-range', 'cascader', 'transfer', 'icon-picker'

# 界面预览

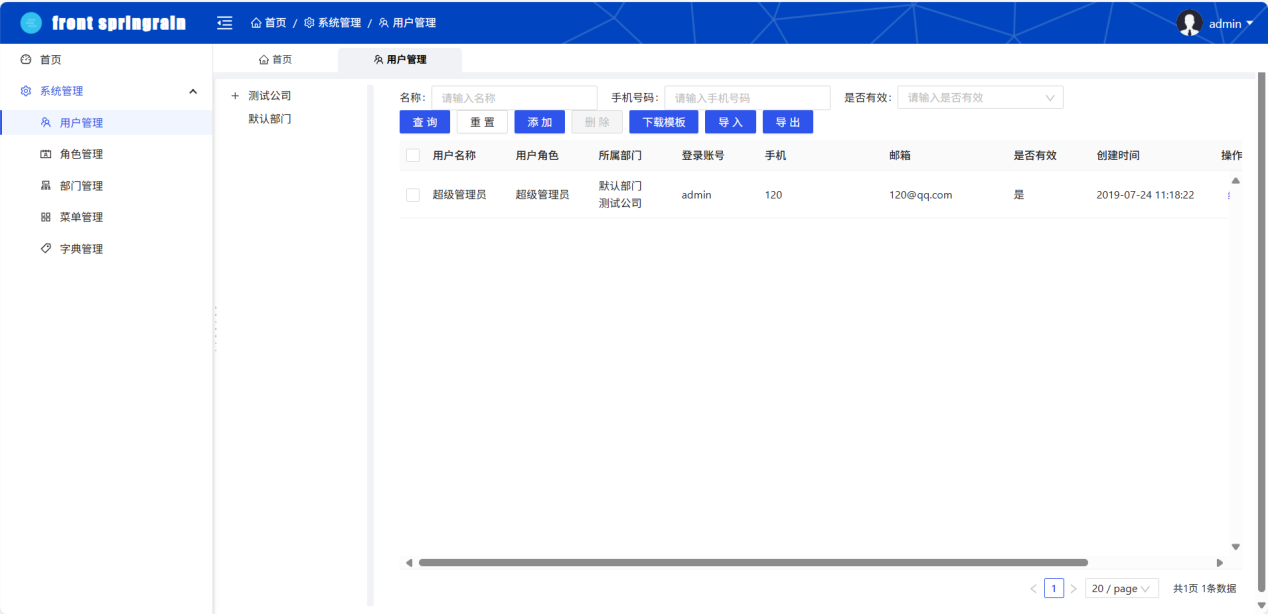
## 登录



## 首页



## 用户管理



### 添加用户



### 修改用户



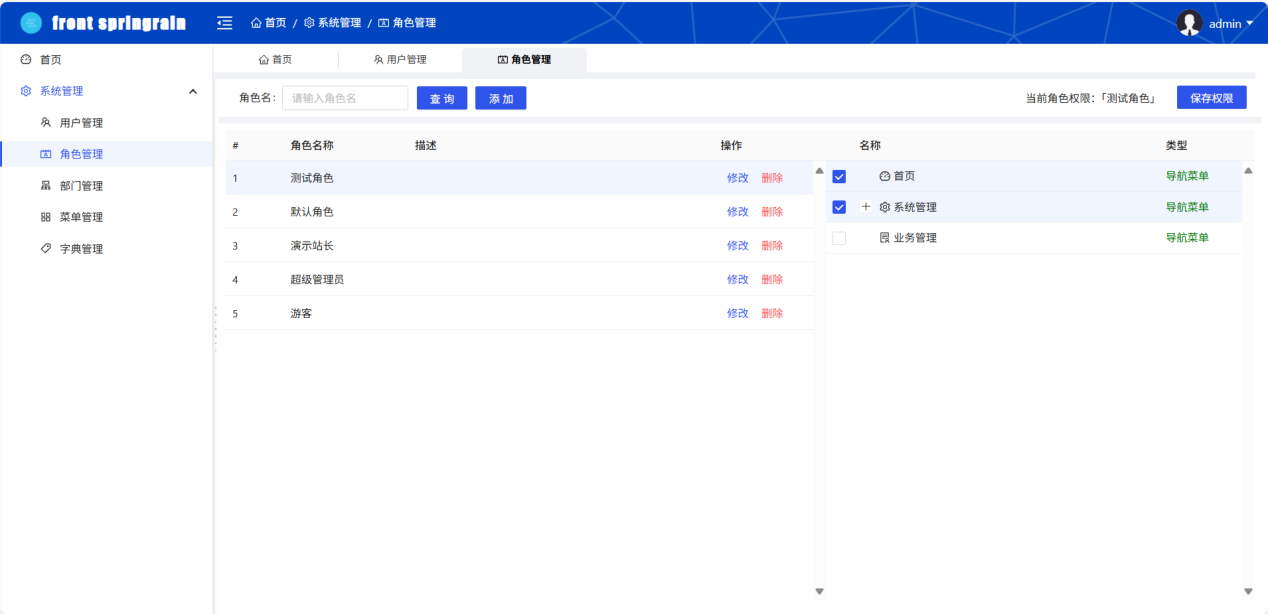
### 删除用户



### 树形部门查询



## 角色管理



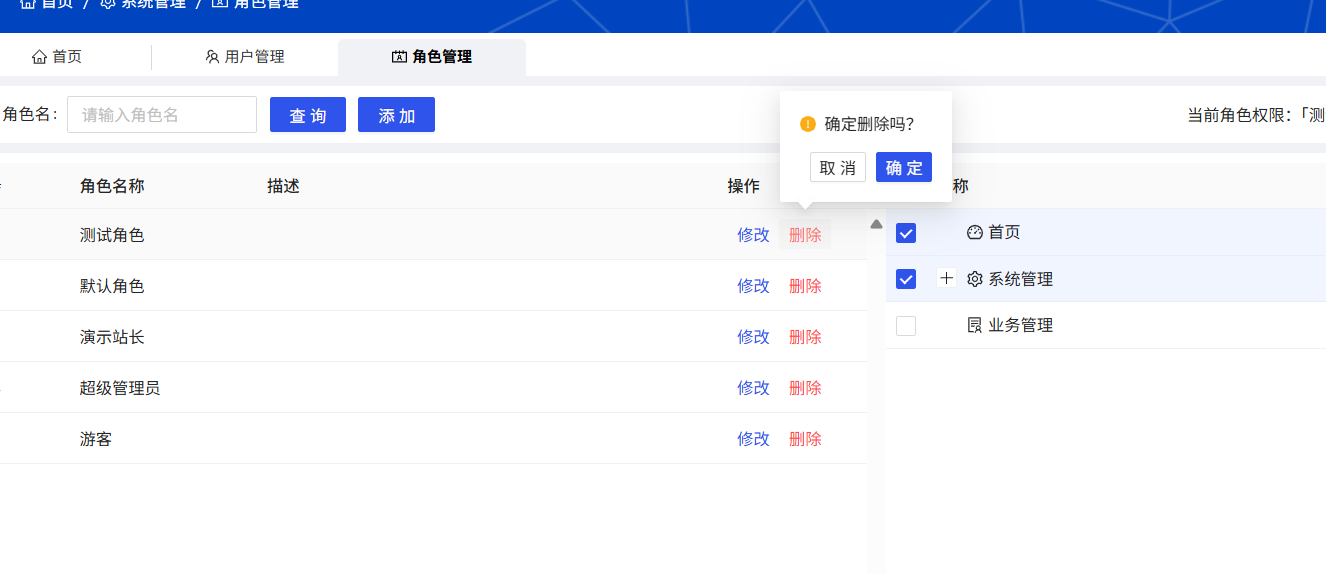
### 添加角色



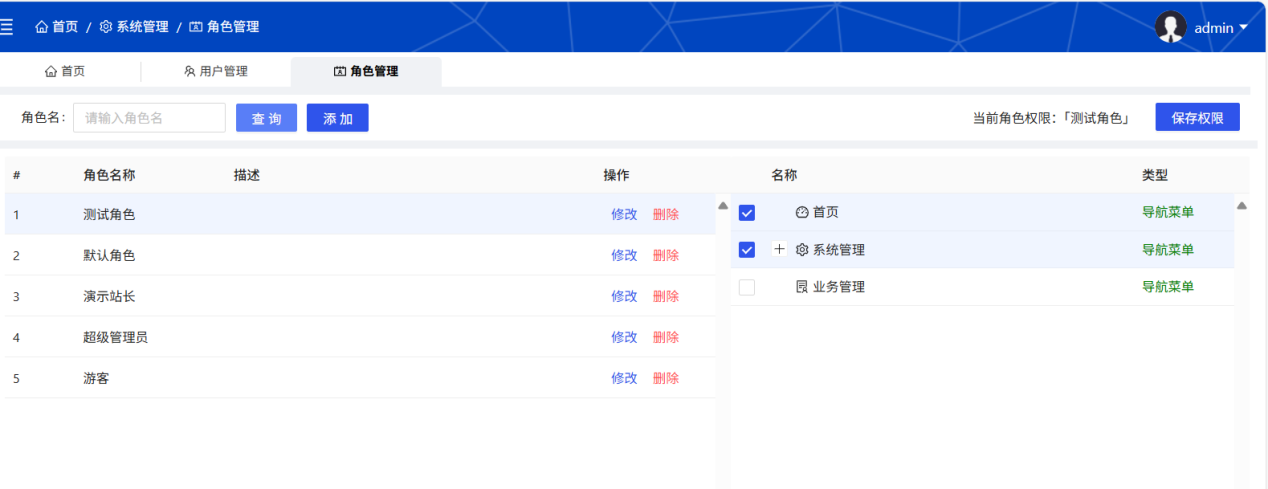
### 修改角色



### 删除角色



### 分配角色权限



## 部门管理



### 添加部门



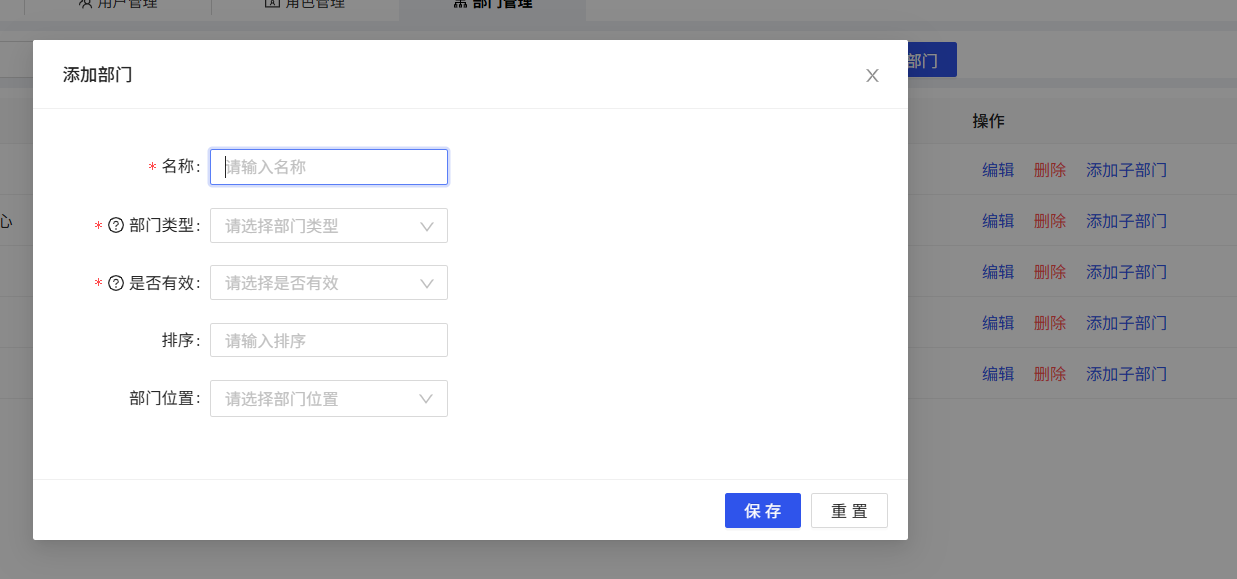
### 修改部门



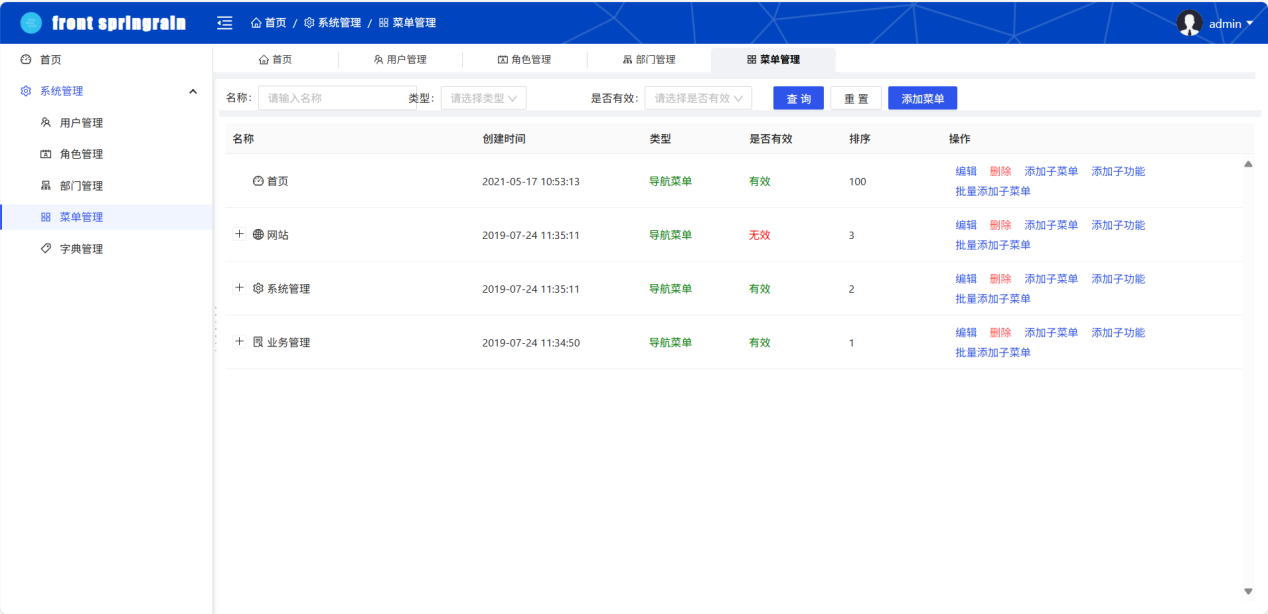
### 删除部门



### 添加子部门



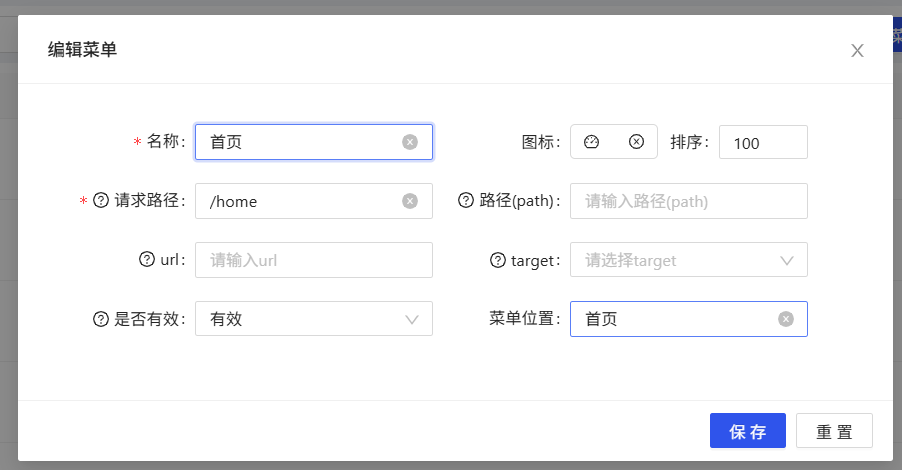
## 菜单管理



### 添加菜单



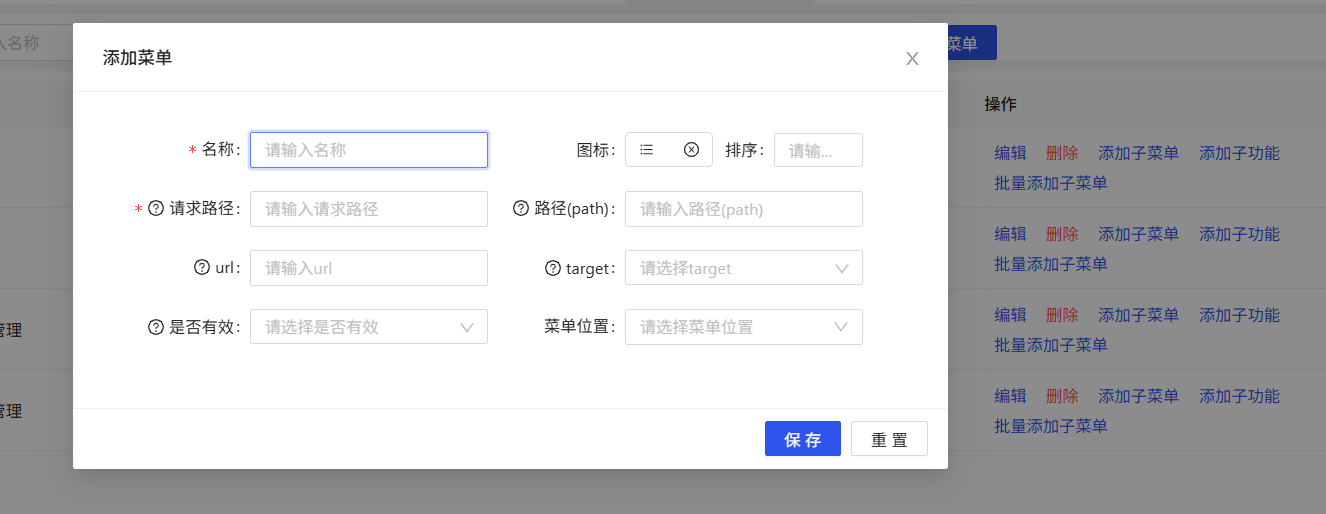
### 修改菜单



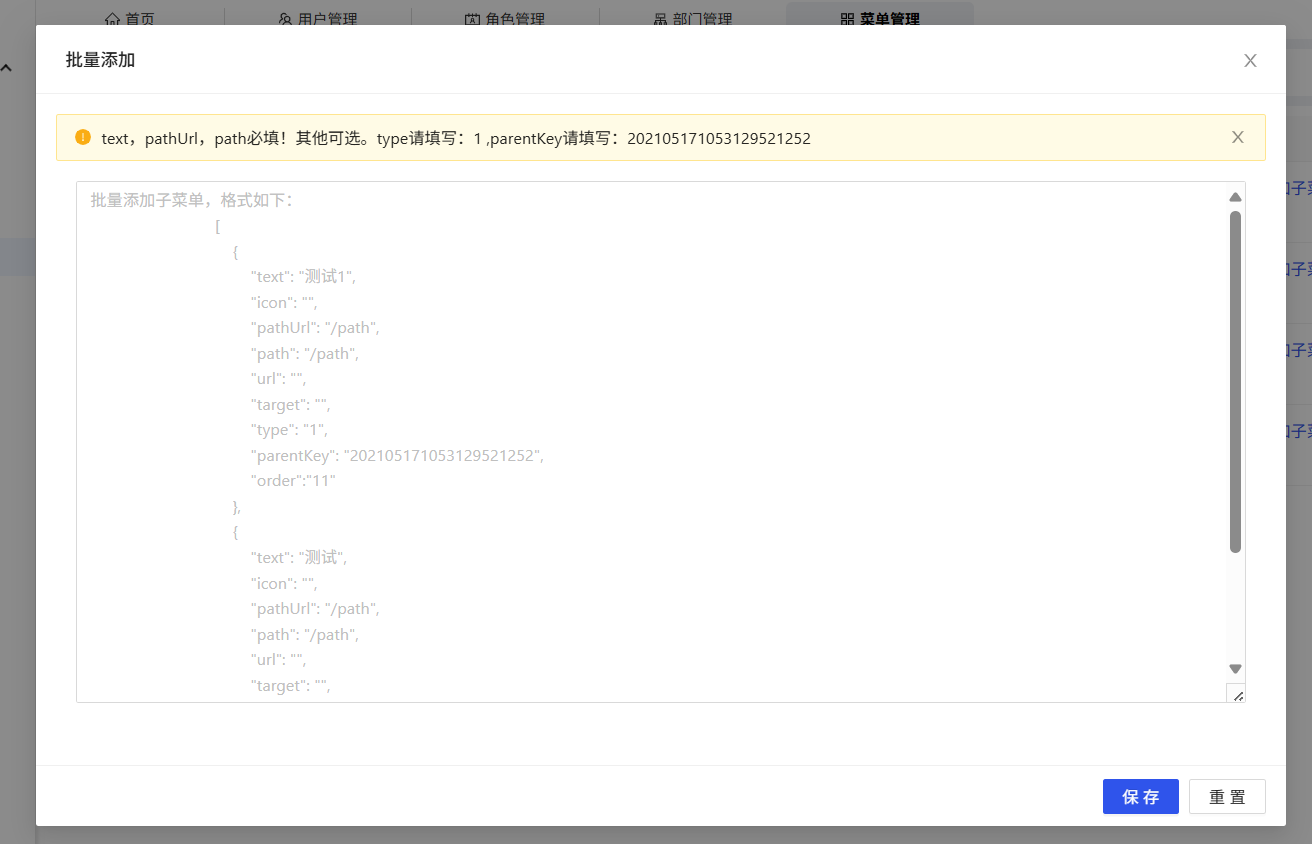
### 删除菜单



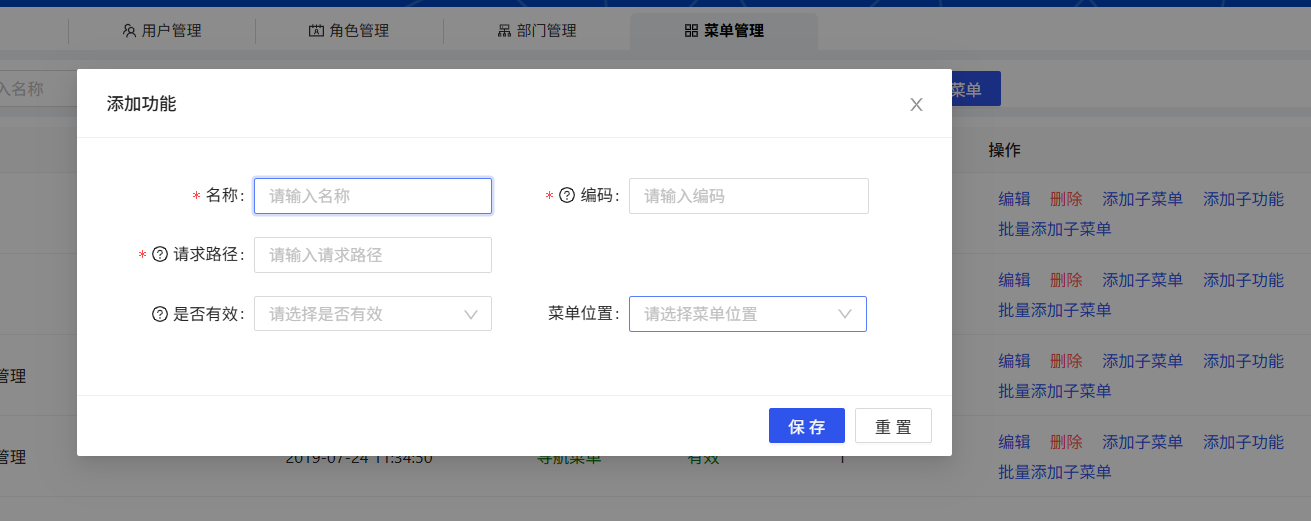
### 添加子菜单



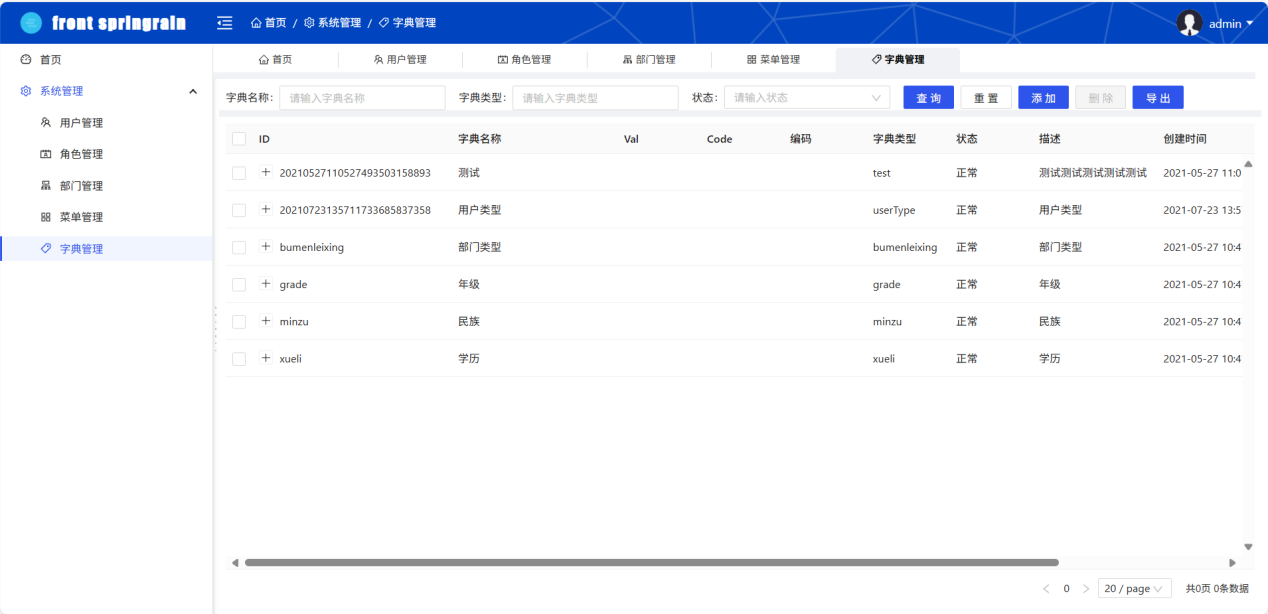
### 批量添加菜单



### 添加功能



## 字典管理



### 添加字典



### 修改字典



### 删除字典



### 添加子项

