# 1、数据架构

## 1.1 数据库ER模型

## 1.2 数据库逻辑模型

## 1.2 数据库物理模型

# 2、接口

## 2.1 权限管理

### 2.1.1 根权限列表查询

（1）接口所属的类

1. 接口定义

GET /api/v1.0/auth/priority/roots/

（3）接口描述

查询所有的根权限

（4）输入

无

（5）输出

[

{

"id": 1,

"code": "TEST\_PRIORITY\_01",

"url": "/api/v1.0/test",

"priorityComment": "测试权限",

"priorityType": 1,

"parentId": 2,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

},{

"id": 2,

"code": "TEST\_PRIORITY\_02",

"url": "/api/v1.0/test02",

"priorityComment": "测试权限02",

"priorityType": 1,

"parentId": 2,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

}

]

### 2.1.2 子权限列表查询

（1）接口所属的类

1. 接口定义

GET /api/v1.0/auth/priority/children/{id}

（3）接口描述

查询某个权限的子权限

（4）输入

权限id

（5）输出

[

{

"id": 1,

"code": "TEST\_PRIORITY\_01",

"url": "/api/v1.0/test",

"priorityComment": "测试权限",

"priorityType": 1,

"parentId": 2,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

},{

"id": 2,

"code": "TEST\_PRIORITY\_02",

"url": "/api/v1.0/test02",

"priorityComment": "测试权限02",

"priorityType": 1,

"parentId": 2,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

}

]

### 2.1.3 添加权限

（1）接口所属的类

1. 接口定义

POST /api/v1.0/auth/priority/

（3）接口描述

添加权限

（4）输入

{

"code": "TEST\_01",

"url": "/api/v1.0/test01",

"priorityComment": "测试权限01",

"priorityType": 1,

"parentId": 1

}

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.1.4 查询权限

（1）接口所属的类

1. 接口定义

GET /api/v1.0/auth/priority/{id}

（3）接口描述

查询某个权限

（4）输入

权限id

（5）输出

{

"id": 1,

"code": "TEST\_01",

"url": "/api/v1.0/test01",

"priorityComment": "测试权限01",

"priorityType": 1,

"parentId": 1,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:00:05"

}

### 2.1.5 更新权限

（1）接口所属的类

1. 接口定义

PUT /api/v1.0/auth/priority/{id}

（3）接口描述

更新权限

（4）输入

权限id

{

"id": 1,

"code": "TEST\_01",

"url": "/api/v1.0/test01",

"priorityComment": "测试权限01",

"priorityType": 1

}

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.1.6 删除权限

（1）接口所属的类

1. 接口定义

DELETE /api/v1.0/auth/priority/{id}

（3）接口描述

删除权限

（4）输入

权限id

（5）输出

{

"status": "success",

"message": "成功"

}

## 2.2 角色管理

### 2.2.1 查询角色列表

（1）接口所属的类

1. 接口定义

GET /api/v1.0/auth/role/

（3）接口描述

查询角色列表

（4）输入

{

"offset": 0,

"size": 10,

"code": "test\_01",

"roleName": "测试角色",

"isEnabled": 1 // 0是未启用，1是启用

}

（5）输出

[

{

"id": 1,

"code": "ROLE\_01",

"roleName": "测试角色",

"roleComment": "测试角色备注",

"isEnabled": 1

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

},{

"id": 1,

"code": "ROLE\_01",

"roleName": "测试角色",

"roleComment": "测试角色备注",

"isEnabled": 1

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

]

### 2.2.2 添加角色

（1）接口所属的类

1. 接口定义

POST /api/v1.0/auth/role/

（3）接口描述

添加角色

（4）输入

{

"code": "TEST\_01",

"roleName": "测试角色01",

"roleComment": "测试角色01",

"isEnabled": 1,

"priorityIds": [1, 3, 9, 10, 20]

}

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.3 查询角色

（1）接口所属的类

1. 接口定义

GET /api/v1.0/auth/role/{id}

（3）接口描述

添加角色

（4）输入

角色id

（5）输出

{

"id": 1,

"code": "TEST\_01",

"roleName": "测试角色01",

"roleComment": "测试角色01",

"isEnabled": 1,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:00:00",

"priorityIds": [1, 3, 9, 10, 20]

}

### 2.2.4 更新角色

（1）接口所属的类

1. 接口定义

PUT /api/v1.0/auth/role/{id}

（3）接口描述

更新角色

（4）输入

{

"id": 1,

"code": "TEST\_01",

"roleName": "测试角色01",

"roleComment": "测试角色01",

"isEnabled": 1,

"priorityIds": [1, 3, 9, 10, 20]

}

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.5 删除角色

（1）接口所属的类

1. 接口定义

DELETE /api/v1.0/auth/role/{id}

（3）接口描述

删除角色

（4）输入

角色ID

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.6 启用/停用角色

（1）接口所属的类

1. 接口定义

PUT /api/v1.0/auth/role/enabled/{id}

（3）接口描述

更新角色的启用状态

（4）输入

{

"id": 1,

"isEnabled": 1

}

（5）输出

{

"status": "success",

"message": "成功"

}

## 2.3 账号管理

### 2.3.1 查询账号列表

（1）接口所属的类

1. 接口定义

GET /api/v1.0/auth/account/

（3）接口描述

查询账号列表

（4）输入

{

"offset": 0,

"size": 10,

"username": "test\_01",

"employeeName": "测试角色",

"roleId": 1,

"isEnabled": 1 // 1是启用，0是未启用

}

（5）输出

[

{

"id": 1,

"username": "testuser",

"employeeName": "张三",

"accountComment": "测试账号备注",

"isEnabled": 1

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

},{

"id": 1,

"username": "testuser",

"employeeName": "张三",

"accountComment": "测试账号备注",

"isEnabled": 1

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

}

]

### 2.2.2 添加账号

（1）接口所属的类

1. 接口定义

POST /api/v1.0/auth/account/

（3）接口描述

添加账号

（4）输入

{

"username": "testuser01",

"password": "12345678",

"employeeName": "张三",

"accountComment": "张三员工",

"isEnabled": 1,

"roleIds": [1, 3, 9, 10, 20],

"priorityIds": [1, 3, 9, 10, 20]

}

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.3 查询账号

（1）接口所属的类

1. 接口定义

GET /api/v1.0/auth/account/{id}

（3）接口描述

查询账号

（4）输入

账号id

（5）输出

{

"id": 1,

"username": "testuser01",

"employeeName": "张三",

"accountComment": "张三员工",

"isEnabled": 1,

"roleIds": [1, 3, 9, 10, 20],

"priorityIds": [1, 3, 9, 10, 20]

}

### 2.2.4 更新账号

（1）接口所属的类

1. 接口定义

PUT /api/v1.0/auth/account/{id}

（3）接口描述

更新账号

（4）输入

{

"id": 1,s

"username": "testuser01",

"password": "12345678",

"employeeName": "张三",

"accountComment": "张三员工",

"isEnabled": 1,

"roleIds": [1, 3, 9, 10, 20],

"priorityIds": [1, 3, 9, 10, 20]

}

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.5 删除账号

（1）接口所属的类

1. 接口定义

DELETE /api/v1.0/auth/account/{id}

（3）接口描述

删除账号

（4）输入

账号ID

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.6 启用/停用账号

（1）接口所属的类

1. 接口定义

PUT /api/v1.0/auth/account/enabled/{id}

（3）接口描述

更新账号的启用状态

（4）输入

{

"id": 1,

"isEnabled": 1

}

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.7 修改密码

（1）接口所属的类

1. 接口定义

PUT /api/v1.0/auth/account/password/{id}

（3）接口描述

更新账号的密码

（4）输入

{

"id": 1,

"password": 1

}

（5）输出

{

"status": "success",

"message": "成功"

}

## 2.4 权限控制

### 2.4.1 查询账号有权限的菜单树

上来先查询账号有权限的根菜单权限，显示在页面左侧的菜单栏里，如果用户点击某个菜单，异步加载那个菜单下面的子菜单即可

（1）接口所属的类

1. 接口定义

GET /api/v1.0/auth/priority/tree/authorized/?accountId=1

（3）接口描述

查询某个账号被授权的菜单树

（4）输入

账号ID

（5）输出

[

{

"id": 1,

"code": "TEST\_PRIORITY\_01",

"url": "/api/v1.0/test",

"priorityComment": "测试权限",

"priorityType": 1,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

},{

"id": 3,

"code": "TEST\_PRIORITY\_02",

"url": "/api/v1.0/test02",

"priorityComment": "测试权限02",

"priorityType": 1,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

}

]

### 2.4.2 查询账号对某个权限是否授权

在页面上，显示了某个按钮的时候，对那个按钮是否显示需要使用特殊的标签，在标签里给一个按钮对应的权限的编号，然后后台会去查询账号对这个编号的权限，是否有授权

（1）接口所属的类

1. 接口定义

GET /api/v1.0/auth/priority/authorized/{id}?accountId=1

（3）接口描述

查询某个账号对某个权限是否授权

（4）输入

权限ID、账号ID

（5）输出

{

"authorized": true // true是授权，false是未授权

}

### 2.4.3 请求URL权限控制

这一块接口，不是对外提供的web http接口，不是给其前端调用的，是给我们后面要写一个全局的过滤器，对所有的请求都要过滤，然后在过滤器中调用这里的接口，查看当前登录的这个用户是否有权限执行某个请求

（1）接口类

PriorityService

（2）接口定义

Boolean checkPriorityAuthorized(Long accountId, String url)

（3）接口描述

查询某个账号对某个权限是否授权

（4）输入

权限ID、账号ID

（5）输出

是否授权，true是授权，false是未授权

# 3、开发架构

## 3.1 实现类图

[权限中心实现类图](https://www.processon.com/diagraming/5a796a90e4b024b99bf22f66)

将账号、角色、权限，相关的数据，通过享元模式，在内存里维护一份

## 3.2 包设计

com.zhss.eshop.auth.web

com.zhss.eshop.auth.service

com.zhss.eshop.auth.service.impl

com.zhss.eshop.auth.dao

com.zhss.eshop.auth.dao.impls

com.zhss.eshop.auth.mapper

com.zhss.eshop.auth.vo

com.zhss.eshop.auth.dto

com.zhss.eshop.auth.do

com.zhss.eshop.auth.query

# 运行流程设计

## 4.1 权限管理

### 4.1.1 根权限列表查询

直接查询出来所有的根权限即可，就是没有parentId就是根权限。

### 4.1.2 子权限列表查询

根据parentId去查询某个权限的子权限即可

### 4.1.3 添加权限

将权限信息保存到数据库即可

### 4.1.4 查询权限

根据权限ID来查询即可

### 4.1.5 更新权限

根据权限ID来跟新即可

### 4.1.6 删除权限

这块必须要进行递归删除，也就是说，你要删除一个权限的话：

第一，你必须先删除最底层的子权限，然后一级一级往上删除，删除到当前的权限

第二，你除了删除权限之外，你还得care一下角色、账号跟权限之间的关联关系，我们要做一个check，也就是说，如果这个权限还跟有一些角色和账号跟之进行关联，就不能直接删除

第三，此处采用设计模式，组合模式+访问者模式，采用组合模式，在这里将权限组装成一颗树，或者是不用提前组装，就是不断的去递归动态组装出来一颗树也可以。然后对这颗树注入一个删除的visistor访问者，这样，以后我们还可以动态注入别的visitor来实现对这颗权限树的不同的操作。

第四，在对这棵树递归删除的过程中，如果发现某个权限还被别人给引用，此时要返回一个特殊的状态码，这个状态码要标明某一级权限还被某个角色或者是账号在引用，就不能直接这样删除，需要回滚整个事务。

我们在这里因为要采取设计模式，所以需要为了这块东西，单独设计一套类出来

这个场景，应该是完美的可以让我们后面去实践组合模式和访问者模式的一个场景，就是我们需要开发两个visitor，第一个visitor需要遍历所有子权限，判断是否有某个权限还被账号或者角色在关联着；第二个visitor，是如果全包所有的子权限都没有被别人在关联，那么就递归删除

## 4.2 角色管理

### 4.2.1 查询角色列表

根据筛选条件+分页查询角色即可

### 2.2.2 添加角色

（1）先添加角色

（2）添加角色和权限之间的关系

### 2.2.3 查询角色

（1）根据ID查询角色

（2）查询角色被授予的权限ID列表

### 2.2.4 更新角色

（1）更新角色

（2）将角色与权限的关联关系都删除

（3）重新插入角色与权限的关联关系

### 2.2.5 删除角色

（1）是否有账号关联了这个角色，如果有是不允许删除角色的

（1）删除角色跟权限的关联关系

（2）删除角色本身

### ~~2.2.6 启用/停用角色~~

~~（1）是否有账号关联了这个角色，如果有是不允许停用角色的~~

~~（2）更新角色的启用状态即可~~

## 4.3 账号管理

### 4.3.1 查询账号列表

根据筛选条件分页查询账号列表即可

### 4.3.2 添加账号

（1）添加账号

（2）添加账号和角色的关联关系

（3）添加账号和权限的关联关系

### 4.3.3 查询账号

（1）查询账号

（2）查询账号关联的角色id

（3）查询账号关联的权限id

### 4.3.4 更新账号

（1）更新账号

（2）删除账号与角色的关联关系，再插入新的账号与角色的关联关系

（3）删除账号与权限的关联关系，再插入新的账号与权限的关联关系

### 4.3.5 删除账号

（1）删除账号与角色的关联关系

（2）删除账号与权限的关联关系

（3）删除账号本身

### ~~4.3.6 启用/停用账号~~

~~直接更新账号的状态即可~~

### 4.3.7 修改密码

直接更新账号的密码即可

## 4.4 权限控制

### 4.4.1 查询账号有权限的菜单树

我们这个是个后台系统，然后我们其实是会通过前面做好的权限管理模块，去录入我们系统的权限树，每个权限对应着一个菜单或者是一个按钮。因为一个菜单对应着一个模块，比如说品牌管理 - > 菜单，模块，但是点击品牌管理之后，进入了一个页面，页面里，可能会包含多个功能，比如说新增品牌，编辑品牌，删除品牌，每个功能都是一个按钮。

然后这个按钮，也是一个权限，如果你有这个权限就可以去看到和点击这个按钮。

菜单，也是一个权限，如果你对菜单有权限，你就可以看到这个菜单。

左侧，都有一个菜单树，菜单树的权限控制，根据当前登录的这个用户，查询到这个用户有权限的这个菜单树。

基于我们之前做好的那个组合模式+访问者模式的思路，在里面加入一个visitor就可以了，访问者就可以了。

（1）查询这个账号有权限的根菜单

（2）然后对每个根菜单，使用visitor去递归每个根菜单，对每个根菜单都去查询这个账号有权限的子菜单，填充为根菜单的子菜单

（3）以此类推，就可通过visitor得到一颗账号有权限的菜单树

### 4.4.2 查询账号对某个权限是否授权

在页面上，对每个按钮，都有一个标签，对这个按钮对应的权限的编号，去查询这个用户对这个编号的权限，有没有授权。

显示每个按钮之前，都发送一个请求到后端，来判断一下，用户对这个按钮是否有权限，如果没有权限的话，直接就不要显示这个按钮了。前端应该是知道每个按钮对应的这个权限的编号的

新建品牌 -> SAVE\_BRAND -> 我们要看一下用户对这个编号的权限是否有授权，显示这个按钮，如果没有授权，这个按钮就不要显示了

1. 查询这个编号对应的权限
2. 查询这个账号对这个权限是否有授权，然后一条SQL，关联账号表、角色表、权限表，去查询

### 4.4.3 请求URL权限控制

如果有个坏蛋，不管什么菜单或者是按钮，直接自己在浏览器里构造访问某个功能的URL请求，直接就请求过来了

这一块接口，不是对外提供的web http接口，不是给其前端调用的，是给我们后面要写一个全局的过滤器，对所有的请求都要过滤，然后在过滤器中调用这里的接口，查看当前登录的这个用户是否有权限执行某个请求

1. 查询这个URL对应的权限
2. 查询这个账号对这个权限是否有授权，然后一条SQL，关联账号表、角色表、权限表，去查询

### 4.4.4 对权限进行缓存

基于享元模式，在纯jvm中进行缓存，权限数据，几乎是很少变化的。99%的情况下，直接访问缓存就可以了。数据量一般很小。权限数据访问频率很高的，因为用户随意出入系统，每次进来都要查询菜单，查询按钮，还要对请求进行认证。但是修改频率很低，一年也不会修改几次。

# 5、测试用例设计

## 5.1 单元测试用例设计

要求，就是对每个controller、service、dao组件的每个方法，都进行单元测试

## 5.2 冒烟测试用例设计

要求，是系统在本地运行起来，直接依赖了数据库之后，对controller层所有的接口访问都可以正常工作

# 6、日志设计

要求的是每一层，每一个类都捕获自己的异常，然后对异常要求必须打印详细的日志