目录

[一、 通用模块-common 1](#_Toc483298353)

[1.0 通用部分 1](#_Toc483298354)

[1.0.1返回值类型设计 1](#_Toc483298355)

[1.0.2 DAO设计 2](#_Toc483298356)

[1.0.3 业务层设计 2](#_Toc483298357)

[1.0.4 单元测试设计 2](#_Toc483298358)

[1.0.5 接口设计 3](#_Toc483298359)

[1.1 注册账户 5](#_Toc483298360)

[1.1.1 DAO设计 5](#_Toc483298361)

[1.1.2 业务层设计 5](#_Toc483298362)

[1.1.3 单元测试设计 7](#_Toc483298363)

[1.1.4 接口设计 7](#_Toc483298364)

[1.2 注销账户 8](#_Toc483298365)

[1.2.1 DAO设计 8](#_Toc483298366)

[1.2.2 业务层设计 8](#_Toc483298367)

[1.2.3 单元测试设计 9](#_Toc483298368)

[1.2.4 接口设计 10](#_Toc483298369)

[1.3 修改账户信息 11](#_Toc483298370)

[1.3.1 DAO设计 11](#_Toc483298371)

[1.3.2 业务层设计 11](#_Toc483298372)

[1.3.3 单元测试设计 12](#_Toc483298373)

[1.3.4 接口设计 12](#_Toc483298374)

[1.5 登录验证 14](#_Toc483298375)

[1.5.1 DAO设计 14](#_Toc483298376)

[1.5.2 业务层设计 14](#_Toc483298377)

[1.5.3 单元测试设计 16](#_Toc483298378)

[1.5.4 接口设计 16](#_Toc483298379)

[1.9 验证码 18](#_Toc483298380)

[1.9.1 请求发送手机验证码 18](#_Toc483298381)

[1.9.2 校验验手机证码 19](#_Toc483298382)

[1.9.3 请求发送邮箱验证连接 19](#_Toc483298383)

[二、其它设计 21](#_Toc483298384)

[2.1 common包结构设计 21](#_Toc483298385)

[2.2 PO(持久化)类设计 21](#_Toc483298386)

[2.2.1 RBAC模块 21](#_Toc483298387)

[2.2.2 信息记录模块 22](#_Toc483298388)

[2.3 DAO基本接口设计 23](#_Toc483298389)

[2.4 自定义异常 24](#_Toc483298390)

[<1> 非设计预期异常 24](#_Toc483298391)

[<2> 账号不存在异常 24](#_Toc483298392)

[<3> 手机号已注册异常 24](#_Toc483298393)

[<4> 邮箱已注册异常 24](#_Toc483298394)

[<5> 密码错误异常 24](#_Toc483298395)

# 通用模块-common

## 1.0 通用部分

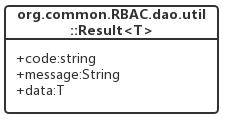
该部分包含的是其他几个模块所使用的一些通用部分，其主要包含：

* 检查账号的用户名是否已被注册
* 检查手机是否已被注册
* 检查邮箱是否已被注册
* 检查账号的登录状态

### 1.0.1返回值类型设计

#### <1> 通用返回类型

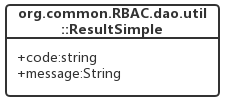
针对整个项目的返回值进行泛型类设计，其全路径为：org.common.RBAC.dao.util.Result<T>



类图 Result

#### <2> 简单返回类型

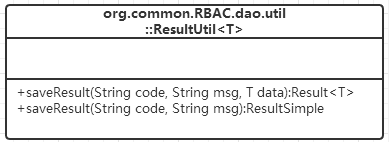
只是针对上面的类型进行简化，有的地方只需要返回标识就可以，无须返回数据，其全路径为：org.common.RBAC.dao.util.ResultSimple



类图 ResultSimple

#### <3> 返回值工具类设计

该工具类可以快速的写入信息并返回对应的返回值对象，由于是工具类，所以设计成静态的。其全路径为：org.common.RBAC.dao.util.ResultUtil<T>



类图 ResultUtil

### 1.0.2 DAO设计

该部分功能位于org.common.RBAC.dao.AccountDao接口中，主要方法为：

* Account checkAccountIsExist(String name) throws AccountNotExistException;
* Boolean checkAccountIsExistNoException(String name);
* Account checkPhonIsExist(String phone) throws PhonIsAlreadyRegisteException;
* Boolean checkPhonIsExistNoException(String phone);
* Account checkEmailIsExist(String eMail) throws EmailIsAlreadyRegisteException;
* Boolean checkEmailIsExistNoException(String eMail);

注意：每种情况分两种形式，一种是返回账户信息，此时可能会抛异常；另一种就是返回Boolean值，这种情况下true表示已存在，反之为不存在。

### 1.0.3 业务层设计

该部分功能位于org.common.RBAC.service.AccountManager接口中，主要方法为：

* ResultSimple verifyAccountNameIsExist(String name);
* boolean verifyAccountNameIsExist\_Flag(String name);
* ResultSimple verifyAccountPhoneIsExist(String phone);
* boolean verifyAccountPhoneIsExist\_Flag(String phone);
* ResultSimple verifyAccountEmailIsExist(String email);
* boolean verifyAccountEmailIsExist\_Flag(String email);

该功能所在的接口需要注入一下两个组件才可以实现其功能：

* AccountDao accountDao;
* ResultUtil<AccountSimple> resultUtil;

### 1.0.4 单元测试设计

|  |  |  |
| --- | --- | --- |
| **被测试方法** | **测试情况** | **预期结果** |
| ResultSimple verifyAccountNameIsExist  (String name) | 英文账号已注册 | 000003 |
| 英文账号未被注册 | 000004 |
| 中文账号已注册 | 000003 |
| 中文账号未被注册 | 000004 |
| 账号为手机号 | 000015 |
| 账号为邮箱 | 000015 |
| boolean verifyAccountNameIsExist\_Flag  (String name) | 英文账号已注册 | true |
| 英文账号未被注册 | false |
| 中文账号已注册 | true |
| 中文账号未被注册 | false |
| ResultSimple verifyAccountPhoneIsExist  (String phone) | 手机号已注册 | 000001 |
| 手机号未注册 | 000000 |
| 手机号格式异常 | 000002 |
| boolean verifyAccountPhoneIsExist\_Flag  (String phone) | 手机号已注册 | true |
| 手机号未注册 | false |
| 手机号格式异常 | false |
| ResultSimple verifyAccountEmailIsExist  (String email) | 邮箱已被注册 | 000005 |
| 邮箱未被注册 | 000006 |
| 验证的邮箱格式异常 | 00007 |
| boolean verifyAccountEmailIsExist\_Flag  (String email) | 邮箱已被注册 | true |
| 邮箱未被注册 | false |
| 验证的邮箱格式异常 | false |

### 1.0.5 接口设计

#### <1> Action设计

该Action的全路径名为：dftsms.web.fxb.action.VerifyAccount，其主要方法为：

* verifyOnlyAccountNameIsExist()：验证账户名是否已被注册
* verifyOnlyAccountPhoneIsExist()：验证手机号是否已被注册
* verifyOnlyAccountEmailIsExist()：验证邮箱是否已被注册
* verifyAccountLoginStatus()：验证账号是否登录

这三个方法属于验证方法，只返回状态，而无需返回数据，所以返回值类型为ResultSimple类型。

#### <2>具体实现

**账号验证**

接口：/verify/AccountNameIsRegistered

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| accountName | String |  | 验证的用户名 |

**手机号验证**

接口：/verify/PhoneIsRegistered

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| accountPhone | String |  | 验证的手机号 |

**邮箱验证**

接口：/verify/EmailIsRegistered

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| accountEmail | String |  | 验证的邮箱 |

**登录状态验证**

接口：/verify/accountLoginStatus

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| accountID | String |  | 账号ID |

## 1.1 注册账户

### 1.1.1 DAO设计

该部分功能位于org.common.RBAC.dao.AccountDao接口中，主要方法为：

* Account registByPhone(String name, String pass, String phone);
* Account registByEmail(String name,String pass,String e\_mail,String em\_Acticode,String em\_token\_exptime)

注意：注册方法可能会抛出NotExpectedException(非设计预期异常)。正常情况下会得到注册后的账号的ID，但是如果无法获取到，就是出现了异常情况，这时候就会抛出该异常。

### 1.1.2 业务层设计

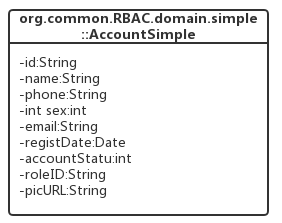
该部分功能位于org.common.RBAC.service.AccountManager接口中，主要方法为：

* Result<AccountSimple> registeByPhone(String name, String pass, String phone);
* Result<AccountSimple> registByEmail(String name, String pass, String e\_mail, String em\_Acticode,String em\_token\_exptime);

该功能所在的接口需要注入一下两个组件才可以实现其功能：

* AccountDao accountDao;
* ResultUtil<AccountSimple> resultUtil;

返回数据中的data为AccountSimple类型，其中包含了账号的必要信息，其类图为：



#### 业务流程图

使用手机号注册业务方法的具体实现流程图为：



使用邮箱注册业务方法的具体实现流程图为：



### 1.1.3 单元测试设计

|  |  |  |
| --- | --- | --- |
| **被测试方法** | **测试情况** | **预期结果** |
| Result<AccountSimple> registeByPhone(  String name,  String pass,  String phone) | 手机号为null或长度为0 | 状态码为：000105 |
| 账号为null或长度为0 | 状态码为：000105 |
| 密码为null或长度为0 | 状态码为：000105 |
| 手机号已注册 | 状态码为：000107 |
| 手机号格式不正确 | 状态码为：000106 |
| 账号已注册 | 状态码为：000101 |
| 账号为手机号 | 状态码为：000102 |
| 账号为邮箱 | 状态码为：000103 |
| 注册信息正常 | 状态码为：000100 |
| Result<AccountSimple> registByEmail(  String name,  String pass,  String e\_mail,  String em\_Acticode,  String em\_token\_exptime) | 邮箱为null或长度为0 | 状态码为：000105 |
| 账号为null或长度为0 | 状态码为：000105 |
| 密码为null或长度为0 | 状态码为：000105 |
| 激活码为null或长度为0 | 状态码为：000105 |
| 时间戳为null或长度为0 | 状态码为：000105 |
| 邮箱已注册 | 状态码为：000108 |
| 邮箱格式不正确 | 状态码为：000106 |
| 账号已注册 | 状态码为：000101 |
| 账号为手机号 | 状态码为：000102 |
| 账号为邮箱 | 状态码为：000103 |
| 注册信息正常 | 状态码为：000100 |

### 1.1.4 接口设计

#### <1> Action设计

该Action的全路径名为：dftsms.web.fxb.action.RegistAction，其主要方法为：

* registbyPhoneAndPass\_SysCreateName ();
* registbyPhoneAndPass();
* registByEmailAndPass();
* registByEmailAndPass\_SysCreateName();

**注意：**

**通过registbyPhoneAndPass\_SysCreateName ()和registbyPhoneAndPass()方法是针对通过手机验证码进行注册的方式进行设计的，也就是说你前面必须执行：****请求发送验证码🡪校验验证码，这两个接口然后才可以调用此方法对应的接口。**

**通过registByEmailAndPass()和registByEmailAndPass\_SysCreateName()方法注册会在注册信息写入到数据库后自动发送邮箱验证链接。**

#### <2> 具体实现

**通过手机验证码注册账户(验证码校验通过,账户名由系统生成)**

接口：/dftsms-web/regist/RegistByPhoneAndPass\_NoAccountName

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| OneTimeInputPass | String |  | 第一次输入密码 |
| SecondTimeInputPass | String |  | 第二次输入密码 |
| nameGenerationStrategy | int | 可选 | 账户名生成策略 |
| nameMixLenth | int | 可选 | 账户名最短长度 |
| nameMaxLenth | int | 可选 | 账户名最长长度 |
| nameIsGirl | boolean | 可选 | 是否为女性名称 |

账户名生成策略：

* 1：纯英文字母不包含数字（长度默认）
* 2：纯英文字母不包含数字（指定长度范围）
* 3：纯英文字母包含数字（长度默认）
* 4：纯英文字母包含数字（指定长度范围）
* 5：生成中文名称
* 6：根据性别生成中文名称

注意事项：

* 默认生成姓名策略为1
* 当指定账户名生成策略为2或4的时候，必须指定账户名的范围（nameMixLenth和nameMaxLenth）
* 当指定账户生成策略为6的时候，必须指定生成账户的性别（nameIsGirl）；使用生成策略6，可以针对性的生成偏男性或女性的中文名称
* 当指定账户名长度时，账户名长度的最小值不得小于3，账户名长度的最大值不得大于20

**通过手机验证码注册账户(验证码校验通过)**

接口：/dftsms-web/regist/RegistByPhoneAndPass

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| accountName | String |  | 账户名称 |
| oneTimeInputPass | String |  | 第一次输入密码 |
| SecondTimeInputPass | String |  | 第二次输入密码 |

**通过邮箱验证链接注册账户(账户名由系统生成)**

接口：/dftsms-web/regist/RegistByEmailAndPass\_NoAccountName

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| email | String |  | 验证邮箱 |
| OneTimeInputPass | String |  | 第一次输入密码 |
| SecondTimeInputPass | String |  | 第二次输入密码 |
| nameGenerationStrategy | int | 可选 | 账户名生成策略 |
| nameMixLenth | int | 可选 | 账户名最短长度 |
| nameMaxLenth | int | 可选 | 账户名最长长度 |
| nameIsGirl | boolean | 可选 | 是否为女性名称 |

**通过邮箱验证链接注册账户**

接口：/dftsms-web/regist/RegistByEmailAndPass

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| accountName | String |  | 账户名称 |
| email | String |  | 验证邮箱 |
| oneTimeInputPass | String |  | 第一次输入密码 |
| SecondTimeInputPass | String |  | 第二次输入密码 |

## 1.2 注销账户

### 1.2.1 DAO设计

该部分功能位于org.common.RBAC.dao.AccountDao接口中，由于直接继承BaseDao所以其在AccountDaoImpl中没有具体实现，而是直接在业务层直接调用。

### 1.2.2 业务层设计

该部分功能位于org.common.RBAC.service.AccountManager接口中，主要方法为：

* ResultSimple deleteAccountByID(Serializable ID);
* ResultSimple deleteAccountByName(String name);

该功能所在的接口需要注入一下两个组件才可以实现其功能：

* AccountDao accountDao;
* ResultUtil<AccountSimple> resultUtil;

#### 业务流程图

**<1> 通过ID删除账户**



**<2> 通过Name删除账户**



### 1.2.3 单元测试设计

### 1.2.4 接口设计

#### <1> Action设计

该Action的全路径名为：，其主要方法为：

* deleteAccountByID();
* deleteAccountByNamec();

#### <2> 具体实现

**通过ID注销账户**

接口：/dftsms-web/deleteAccount/byID

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| deleteAccountId | String |  | 注销账户ID |

**通过Name注销账户**

接口：/dftsms-web/deleteAccount/ byName

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| deleteAccountName | String |  | 注销账户名 |

## 1.3 修改账户信息

### 1.3.1 DAO设计

该部分功能位于org.common.RBAC.dao.AccountDao接口中，主要方法为：

* void updateAccount\_pass(Account account, String newPass);
* void updateAccount\_sex(Account account, Sex sex);
* void updateAccount\_phon(Account account, String newPhon);
* void updateAccount\_email(Account account, String newEmail);
* void updateAccount\_pic(Account account, Pictures pic);
* void updateAccount\_status(Account account, AccountStatus accountStatus);
* void updateAccount\_role(Account account, Role role);

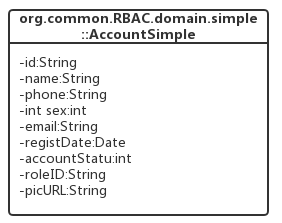
设计思路：设计采用每个修改独立的原则，这样在业务 就可以进行随意组拼，而不是在数据层将其写死。

### 1.3.2 业务层设计

该部分功能位于org.common.RBAC.service.AccountManager接口中，主要方法为：

* Result<AccountSimple> updatePassByID(Serializable id, String newPass);
* Result<AccountSimple> updatePassByID(Serializable id, String newPass,String oldPass);
* Result<AccountSimple> updateSexByID(Serializable id, Sex sex);
* Result<AccountSimple> updatePhonByID(Serializable id, String ewPhoneNum);
* Result<AccountSimple> updateRoleByID(Serializable id, Serializable RoleId);
* Result<AccountSimple> updateStatuByID(Serializable id, AccountStatus ccountStatus);
* Result<AccountSimple> updatePicByID(Serializable id, String PicURL);
* Result<AccountSimple> updateEmailByID(Serializable id, String eMail);
* Result<AccountSimple> updateAccoount(AccountSimple account, Boolean fUpdateSex);

返回数据中的data为AccountSimple类型，其中包含了账号的必要信息，其类图为：



注意：其中最后一个方法，其实就是对修改信息的一次封装，也就是说修改什么信息就添加进去，当然禁止修改的信息除外，根据用户的ID以及账户名称进行双重锁定后，修改需要修改的信息，当然其中还包含一个boolean类型的参数，其原因为传递的性别参数为int类型，因为其默认有初始值，如果不进行辨识，很可能在没有想改变性别的情况下将性别进行了修改，参数的含义为：true表示性别也需要修改。

该业务方法的具体实现流程图为：

### 1.3.3 单元测试设计

### 1.3.4 接口设计

#### <1> Action设计

该Action的全路径名为：dftsms.web.fxb.action.UpdateAccountInfoAction，其主要方法为：

* updatePassByOldPass ()
* updatePassByIdentifyCode()

#### <2> 具体实现

**通过原有密码修改当前账户密码**

接口：/update/passwordByOldPass

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| accountID | String |  | 修改账户ID |
| oldPassword | String |  | 账户原密码 |
| newPassword | String |  | 新密码 |
| newPasswordSecondInput | String |  | 第二次输入新密码 |

**通过手机验证码修改当前账户密码**

接口：/update/passwordByIdentifyCode

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| accountID | String |  | 手机号 |
| identifyCode | String |  | 需要验证的验证码 |
| newPassword | String |  | 新密码 |
| newPasswordSecondInput | String |  | 第二次输入新密码 |

## 1.5 登录验证

### 1.5.1 DAO设计

该部分功能位于org.common.RBAC.dao.AccountDao接口中，主要方法为：

* Account verifyAccountByNameAndPass(String name, String pass);
* Account verifyAccountByPhoneAndPass(String phone, String pass);
* Account verifyAccountByEmailAndPass(String email, String pass);
* Account verifyAccountByPhoneAndIdentifyCode(String phone);

### 1.5.2 业务层设计

该部分功能位于org.common.RBAC.service.AccountManager接口中，主要方法为：

* Result<AccountSimple> verifyAccount(String name, String pass);
* Result<AccountSimple> loginByPhoneAndIdentifyCode(String phone);

该功能所在的接口需要注入一下两个组件才可以实现其功能：

* AccountDao accountDao;
* ResultUtil<AccountSimple> resultUtil;

#### 业务流程图

**<1> 通过密码登录**



注意：登录成功后写入登录标识(fxb\_AccountID和fxb\_roleID)。

**<2>通过验证码登录**



注意：登录成功后首先清除验证码验证标识，并写入登录标识(fxb\_AccountID和fxb\_roleID)。

### 1.5.3 单元测试设计

|  |  |  |
| --- | --- | --- |
| **被测试方法** | **测试情况** | **预期结果** |

### 1.5.4 接口设计

#### <1> Action设计

Action的全路径名为：dftsms.web.fxb.action.VerifyAccount，其主要方法为：

* loginByPhoneAndIdentifyCode();
* verifyAccountByPass();
* LogOut();

注意：loginByPhoneAndIdentifyCode ()方法是针对手机号验证码登录进行设计的，也就是说你前面必须执行：请求发送验证码，然后才可以调用此方法对应的接口。

#### <2> 具体实现

**根据账号密码登录**

接口：/login/ByPass

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| account | String |  | 账号名(包含手机邮箱) |
| password | String |  | 密码 |

**根据手机号及验证码登录**

接口：/login/ByPhoneAndIdentifyCode

需要参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **约束** | **说明** |
| phone | String |  | 手机号 |
| identifyCode | String |  | 需要验证的验证码 |

**注销账号登录**

接口：/login/logOut

需要参数：无

## 1.9 验证码

手机验证码请求正常流程为：

* 请求服务器向手机发送验证码，然后请求服务器并验证验证码的正确性。

邮箱验证请求正常流程为：

* 请求服务器向邮箱发送验证连接，然后点击验证连接进行验证

注意：手机验证码登录的时候的流程是，请求发送验证码后，然后将手机号及验证码提交验证，无须再调用验证码验证接口了。

### 1.9.1 请求发送手机验证码

流程图：



注意：当验证短信发送成功后，向Session中保存当前的验证码、时间戳以及发送到的手机号，其对应的变量名分别为：identifyCode、icTimestamp和icPhone。并且保存一个验证码验证标识IdentifyCodeFlag其值为false，表示当前验证码还未通过。

### 1.9.2 校验验手机证码

主要上面请求发送验证码的时候已经保存了其手机号，所以验证验证码的时候以及后续都无需传递手机号参数。

该接口主要是使用的是《秒嘀科技》提供的免费提供的试用接口(有次数限制)，其调用的秒嘀的官方接口主要位于com.miaodiyun包下，对其进行封装并进行调用的工具方法在org.common.miaodiyun包下。

流程图：



注意：验证码验证成功后，将Session中保存的验证码及其时间戳删除，并保留了手机号(icPhone)，以便进行后续操作，同时保存一个码通过标识IdentifyCodeFlag，其值为true。

### 1.9.3 请求发送邮箱验证连接

请求发送邮箱验证链接，主要用于绑定邮箱，使用邮箱注册的时候会自动发送邮箱验证链接，所以无须进行请求此接口。

该部分的验证信息是保存在数据库中的，但是有效时间是一小时（可调）。

流程图：



### 1.9.3 校验邮箱验证链接

流程图：



# 二、其它设计

## 2.1 common包结构设计

org.common.RBAC：该包下是关于RBAC（基于角色的访问控制）的所有操作

org.common.RBAC.domain：该包下是所有数据库表对应的PO(持久化类)类

org.common.RBAC.dao：该包下保存的是所有DAO组件的接口

org.common.RBAC.dao.impl：该包下保存的是所有DAO组件的实现

org.common.RBAC.service：该包下保存的是所有业务逻辑组件的接口

org.crazyit.hrsystem.service.impl：该包下保存的是所有业务逻辑组件的接口

org.common.FDBK：该包下是关于反馈信息的所有操作(整体结构与上面类同)

## 2.2 PO(持久化)类设计

### 2.2.1 RBAC模块

从数据库建模角度来看账户、角色、权限三者中，账户与角色是N-N关系，而角色与权限是N-N关系。

从面相对象建模角度来看，账户管理与角色的关联关系，角色管理与权限的关联关系，因此账户与角色是单向N-N关系，角色和权限是单向N-N关系。

* Account：账户基本信息，对应底层数据库的t\_customer表
* Role：角色信息，对应底层的t\_role表
* Power：权限信息，对应底层的t\_power表

账户、角色、权限三者是基于中间表进行关联的，而使用Account管理与Role的关联，而Role管理与Power的关联。

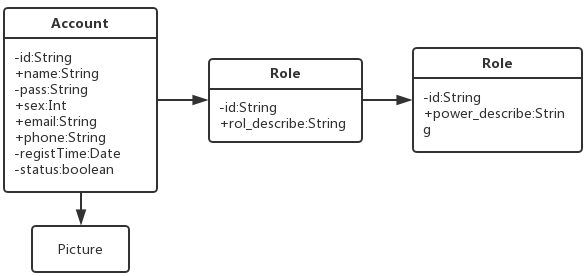


图 1

Account账户还单向关联图片表实体，用来保存用户头像。

### 2.2.2 信息记录模块

#### <1>登录信息记录

从数据库建模角度来看，这部分不与其它表进行关联。而不与账户关联的主要目的是记录那些不存在的账户以及已经删除的账户所产生的登录历史。

从面相对象建模角度来看，LoginInfo与LoginInfoStatus是单向N对1的关系。

* LoginInfo：登录信息记录，对应底层数据库的t\_logininfo表
* LoginInfoStatus：登录信息中记录登录状态，对应底层数据库的t\_logininfo\_ status表

LoginInfoStatus存在的意义就是用来区别登录信息所产生的状态，例如：账号不存在，密码错误等。

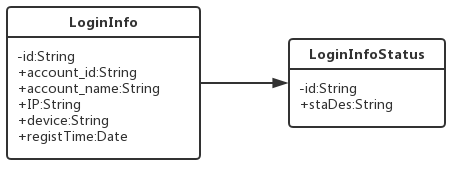
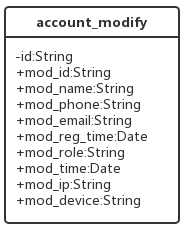


图 2

#### <2>账户信息修改记录

从数据库建模来看，这部分不与其它表进行关联。其不与账户关联的原因，与上面一样。只不过该部分针对的角度是账户信息修改。

* account\_modify



## 2.3 DAO基本接口设计

采用面向接口的方式进行定义，并将其中将所有通用的操作提取出来。下面是BaseDao接口的详细定义：

* T getByID(Class<T> entityClazz, Serializable id)：根据ID加载实体
* Serializable save(T entity)：保存实体
* void update(T entity)：更新实体
* void delete(T entity)：删除实体
* void deleteByID(Class<T> entityClazz, Serializable id)：根据ID删除实体
* List<T> findAll(Class<T> entityClazz)： 获取所有实体
* long findCount(Class<T> entityClazz) 获取实体总数

#### BaseDao测试用例设计

|  |  |  |
| --- | --- | --- |
| **被测试方法** | **测试情况** | **预期结果** |
| getByID (  Class<T> entityClazz, Serializable id) | ID存在，实体类型匹配 | 返回对应的实体对象 |
| ID存在，实体类型不存在 | 非实体对象异常 |
| ID不存在 | ID不存在异常 |
| 参数包含null | NULL异常 |
| save(T entity) | 保存完整正确实体对象 | 返回Serializable (待定) |
| 保存已存在的实体对象 | 实体对象已存在异常 |
| 保存已存在的实体对象  但部分修改 | 实体对象已存在异常(修改) |
| 保存缺少必要信息的实体对象 | 信息不全异常 |
| 参数为null | NULL异常 |
| 保存非实体类型对象 | 参数异常 |
| delete(T entity) | 删除已存在实体对象 | 数据库中清除该对象 |
| **删除实体对象不存在** | **NULL异常** |
| 参数为null | NULL异常 |
| 删除非实体类型对象 | 非实体对象异常 |
| deleteByID (  Class<T> entityClazz,  Serializable id) | ID存在，实体类型匹配 | 数据库中清除该对象 |
| ID存在，实体类型不存在 | 非实体对象异常 |
| ID不存在 | ID不存在异常 |
| 参数包含null | NULL异常 |
| findAll(Class<T> entityClazz) | 正确的实体类型 | 返回所有该类型实体对象 |
| 获取非实体类型 | 参数异常 |
| 参数为null | NULL异常 |
| findCount(  Class<T> entityClazz) | 正确的实体类型 | 返回正确统计数字 |
| 获取非实体类型 | 参数异常 |
| 参数为null | NULL异常 |

## 2.4 自定义异常

### <1> 非设计预期异常

NotExpectedException：并非设计中预期的异常，主要是针对后期开发中所未发现的异常。

### <2> 账号不存在异常

AccountNotExistException：账号不存在异常，主要针对登录验证操作

### <3> 手机号已注册异常

PhonIsAlreadyRegisteException：手机号已注册异常，主要针对注册操作

### <4> 邮箱已注册异常

EmailIsAlreadyRegisteException：邮箱已注册异常，主要针对注册操作

### <5> 密码错误异常

PasswordWrong：密码错误异常，主要针对登录操作