注意：

这里只是设计的思路，是否具体要做到框架中去，需要在进行考虑。

1、该功能总体实现的想法

2、已完成的框架如何修改才能实现的思路。

# 架构搭建功能设计

## 帐套管理

1、每个帐套都是单独的数据库。新建帐套时，根据维护的sql文件，对帐套数据库进行创建，然后进行数据的初始化。

2、可以考虑单独的管理系统对于帐套进行管理。这里创建的框架都可以直接作为管理端。管理端可以对用户、权限、角色、岗位等信息进行管理。不同帐套之间共享公共数据库的数据。对于帐套数据库可能用到的用户表等信息时，可以考虑使用视图映射的形式实现。在一种实现方式就是在系统建立系统公共信息的缓存池。但是对于内存要求就高了。

3、帐套数据库只是针对业务数据的信息，且帐套间不会产生数据的交互。

4、帐套信息可以通过配置文件或者公共的数据库表进行管理，创建帐套最好是说明该帐套是数据哪个组织下的，并可以针对不同的帐套进行组织人员的选择控制等。

5、人员登录时，可以给用户设定只能操作哪个帐套的权限，同时也可以考虑采用帐套加组织架构的形式进行验证，即该用户在该组织架构下，则可以访问该组织架构的帐套信息。

## 控制单用户登录

控制系统用户，同一时间只可以登录一次。实现方式考虑如下：

1. 用户登录后，在Application或者静态变量中存储用户登录的信息。

格式：

SessionId→用户登录信息

1. 在登录时，检测公共变量中，相同的sessionId下是否存在该信息。

如果存在该sessionId数据，则查看用户信息是否一致。不一致则切换为最新的用户信息。一致则直接登录。

如果不存在该sessionId数据，则认为未登录过。则检测是否存在相同用户的session信息。如果存在，则提示已经登录了，是否强制登录。

1. 如果强制登录时，将该用户所对应的sessionId对应的值置为强制下线状态。
2. 在所有的请求中，检测该用户的状态，如果检测到已经强制下线，则直接注销登录系统。
3. Session结束时，检测缓存中该sessionId的缓存是否存在，如果存在，则查看下用户信息是否有，如果有，则标记该用户为下线状态。如果为强制下线状态，直接删除掉即可。
4. 注销系统时，将sessionId对应的缓存信息进行删除。

如果兼容多用户登录判断时：

1. 在进行登录时，直接记录该session的信息，不进行判断即可。
2. 在进行注销时，检测该session下用户是否还存在相应的session。

如果存在，则不进行操作，如果不存在了，登录状态修改为下线。

1. session end的时候，检测是否存在该用户的信息，如果存在，则检测是否为强制下线状态，如果是则直接删除掉。如过是用户信息，则检测缓存中是否还存在该用户信息，则不管，如果不存在，则标记用户为下线状态，删除该缓存。

## 权限控制

**功能权限**

根据用户分配的菜单权限信息。在用户登陆时获取用户信息。在登录到主界面后，获取前台缓存时，将用户的菜单权限一并获取到前台，其中获取用户权限时要去重处理。

可以获取到的菜单都是该用户所具有的功能权限。用户没有的权限根本获取不到。

然后将获取到的模块，菜单等信息直接渲染到界面进行展示相应功能即可。

功能权限的区分，对于功能需要考虑是否设置“虚拟功能”，即该类功能也需要单独的权限控制，但是，在系统界面中不需要进行列表的展示。

**操作权限**

在用户登录后，对用户的按钮权限进行获取，直接存放在系统缓存中。

然后在进行相应的功能时，对于用户的权限内容进行判断。权限判断可以考虑在权限需要配置的按钮上设置上一个固定的属性，然后判断在进入界面后，直接根据这个特定的属性，检测该按钮是否具有权限。对于有的则展示，对于没有的按钮直接删除掉即可。

特殊属性设置两个，一个正常属性：

|  |  |
| --- | --- |
| btnauth | 值为true和false， 用于判断该按钮是否属于权限控制的按钮。 |
| authcode | 值为数据库设置的按钮权限的编号。用于对该按钮进行权限判定用。 |
| id | 在authcode未设置时，作为授权验证使用的编号。 |

**数据权限**

这部分待考虑，目前考虑在岗位方面来进行限制。

## 授权认证实现方式

**第一种思路：**

1. 创建授权生成器。实现功能： 根据用户录入的机器的Mac地址、Ip地址、运行使用日期、允许用户量等信息，对于lisence文件进行生成。
2. Lisence文件放入项目指定位置，以供系统进行读取，并判断mac与ip的对应关系，系统使用的权限进行判断。
3. 生成时操作： 根据用户录入的使用最后期限、用户量、mac、Ip信息，使用RSA生成非对称加密的公钥和私钥。将公钥也写入到lisence文件中。并对内容采用私钥进行加密。内容包括：mac与Ip结合采用AES加密然后base64转码后的字符串与时间和用户量结合产生的内容。
4. 在进行登录时，对于mac地址与Ip地址，根据当前机器情况进行加密校验。对于日期和用户量信息，根据公钥解密后，即可得到对应的数据。
5. 具体的加密方式可以在做的时候进行考虑加入，对于mac与ip可以考虑md5加密。

这里唯一可以保密的就是采用非对称加密，防止根据源码进行解密出来数据。从一定程度上可以缓解被破解的可能。因为破解至多也就可以获取到公钥，无法得到私钥。除非自己创建一套同样的授权文件生成器。这个难免。

这里可以考虑授权生成器中加入一个小型的数据库，sqlite或者access之类的，对于已经生成的授权进行存储。以防止后续生成时，可以留住已经用过的公钥。对于后续扩展网络认证也有好处。

**第二种思路：**

1. 在软件方面，生成相关的机器码。及对机器的mac与ip进行相应的加密操作后，生成的机器码。然后将机器码发送给授权生成器。
2. 在授权生成器设置时间、用户量、授权类型（mac、ip、mac and ip）等信息，然后同思路一，对授权文件进行生成。对于解码操作一致。
3. 这里同样但是可以避免的是对于授权生成器部分，可以不需要再写一遍mac与ip的加密。

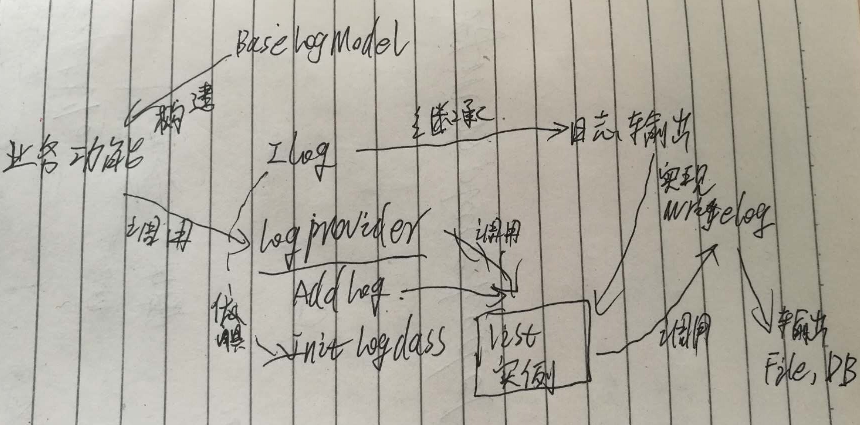
**第三种思路：**

1. 网络授权认证方式，在公网固定一个认证接口。客户端将相应的机器码发送到公网，公网进行认证后，下发系统的授权信息。包括到期日期、用户量、下次检测时间等。
2. 在网络上存储这对应的权限的信息。
3. 需要具有一个固定的公司平台进行支持，并且系统必须是外网系统才可以使用。

*这里考虑采用第二种思路实现。*

## 日志输出

日志输出的思路：



框架提供基础的日志操作类的基类，后续日志输出的类需要集成该基类，或者直接使用该基类。基类中提供转换方法Copy,可以将该基类内容转化到子类中。

框架提供ILog接口，定义日志输出方式以及输出方法两部分内容。输出端需要集成该接口。并制定实现类日志输出的方式，这里的输出方式与系统框架的日志输出配置对应。对于实现该接口的类，在日志提供转换方法中会通过反射进行实例化，进而进行调用。

框架提供主要的调度类：LogProvider类。该类采用单例模式实现，保证日志输出的口只有一个；该类起到了一个呈上启下的作用。用于初始化所有该程序中集成了ILog接口的实现类，认定是进行日志输出的正式类。此外该类提供日志输出调用的公共方法，供需要日志输出的位置进行日志的输出。

对于后续如果需要优化日志输出的性能问题时，可以针对LogProvider类进行优化，主要是考虑多线程输出处理。

对与后续需要接入的日志输出等内容，可以直接调用logprovider的公共方法实现即可。需要添加输出的位置，直接集成Ilog接口，直接生成一个实现类即可。

## 手机端授权

这里考虑采用令牌的方式对手机端的授权进行控制。

1. 后台提供获取用户授权令牌的接口，该接口负责为有权限的用户发放授权令牌。同时对于令牌相关的信息进行存储。包括令牌的授权类型以及授权码的有效时长等。
2. 后台提供刷新令牌接口，该App端可以通过调用该接口，对两个令牌进行重新获取。
3. 在手机端登录时，App通过用户名、密码、机器相关信息，调用获取令牌的接口进行令牌获取。手机端获取令牌为两个：刷新令牌和授权令牌。刷新令牌为长时间不过期的令牌，授权令牌为近期有效令牌。App可以通过刷新令牌调用刷新接口，对刷新令牌和授权令牌进行重新获取。
4. App端通过获取的授权令牌作为参数，对用户的登录信息进行获取。此后可以通过用户的信息进行后续操作。（这里可以考虑后续App操作都只是允许使用授权令牌进行，具体待定）
5. App端可以对两个令牌进行存储。在下次登录时，直接通过授权令牌和刷新令牌进行用户登录信息的获取。
6. 用户进行授权令牌的登录获取时，服务端会根据新传入的令牌、用户信息、IP地址信息、设备信息等判断用户是否满足登录的需求。以及判断用户授权码是否已经超时。