**“银行业务管理系统”**

系统设计报告

**学号：\*\*\*\*\***

**姓名：\*\*\*\*\*\***

**计算机科学与技术学院**

**中国科学技术大学**

2019年3月

**目 录**

1 概述 1

1.1 系统目标 1

1.2 需求说明 1

1.3 本报告的主要贡献 1

2 总体设计 1

2.1 系统模块结构 1

2.2 系统工作流程 1

2.3 数据库设计 1

3 详细设计 2

3.1 \*\*\*\* 模块 2

3.2 \*\*\*\* 模块 2

3.3 \*\*\*\* 模块 2

4 实现与测试 2

4.1 实现结果 2

4.2 测试结果 2

5 总结与讨论 2

附录 团队介绍与工作分工 3

# 

# 1 概述

## 1.1 系统目标

// 概述本系统的主要开发目标

## 1.2 需求说明

// 说明本系统的主要功能需求

## 1.3 本报告的主要贡献

// 概述本报告的主要内容

# 2 总体设计

## 2.1 系统模块结构

// 给出本系统的模块结构图，包括各级子模块，以及模块之间的接口关系。定义每个模块的基本功能。

## 2.2 系统工作流程

// 给出系统工作流程图

## 2.3 数据库设计

// 给出数据库设计ER图，以及最终的数据库表结构

# 3 详细设计

## 3.1 修改/插入模块

3.1.1 输入

本工程基于*python/flask+peewee*框架，因此尽管数据库中各类基本表内容、格式差距较大，前端页面输入数据的类型也不尽相同，但依旧可以统一到一个框架之中，对于数据库的修改/插入模块是通过views.py中的***common\_edit\_%***（%表示具体模块名，具体见流程图小节）函数来实现的。

***common\_edit函数***的输入参数总共有三个：

1. *DynamicModel*:

表示此时数据库的动态模型，用于和基本表建立对应关系，从而可以通过该模型读取数据库中的内容，并进行。

1. *form*：前端静态网页传回的数据格式，具体在*form.py*中定义，表示前端网页传送到后端的数据格式，可以用*form\_to\_model*函数将前端的*form*转化为与数据库基本表格式相符的*model*。
2. *view*：在前端显示的网页，即*app/template*中定义的各类静态网页，是用户可见的html界面。

3.1.2 输出

在数据库中插入或修改相关条目，并返回到用户界面，提示用户修改/插入成功还是失败。

1. 数据库操作：通过peewee的相关API执行SQL语句，实现对数据库的插入和修改。
2. *flash*：通过*flash()*函数，我们可以把提示信息动态的呈现在用户界面上。
3. 函数的整体返回的是一个动态页面，通过flask框架中的*render\_template*实现。

3.1.3 详细描述

本部分主要由以下三个部分实现：

1. 前端html文件：通过读取前端用户输入数据，组织为form的格式输出到views.py中。
2. views.py文件：该部分对应于B/S架构中的应用服务器功能，其中针对于每一个不同的模块设计了不同的方法，修改插入和函数***common\_edit\_***相关，***common\_edit\_***中调用了*peewee*中的*mysql*数据库操作方法，实现了对数据库的操作，其中三种类型的操作如下：
   1. 统计表中与输入form主键相同的元素个数，用于判断当前执行的是修改操作（统计结果为1）还是插入操作（统计结果为0）：



* 1. 对表中记录进行修改：



* 1. 向表中插入新的记录：



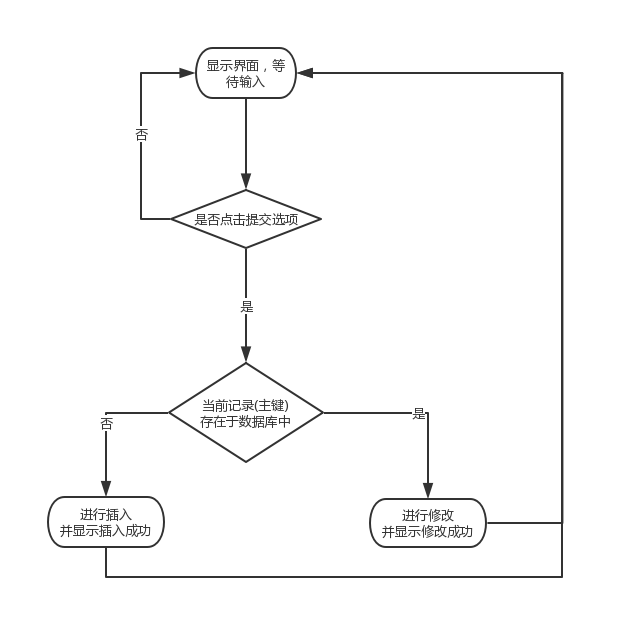
其中，从前端传回的form定义在form.py中，而peewee定义的数据库model则包含在models.py中，因此在views.py中，需要将这两个文件导入。

3. form.py wen

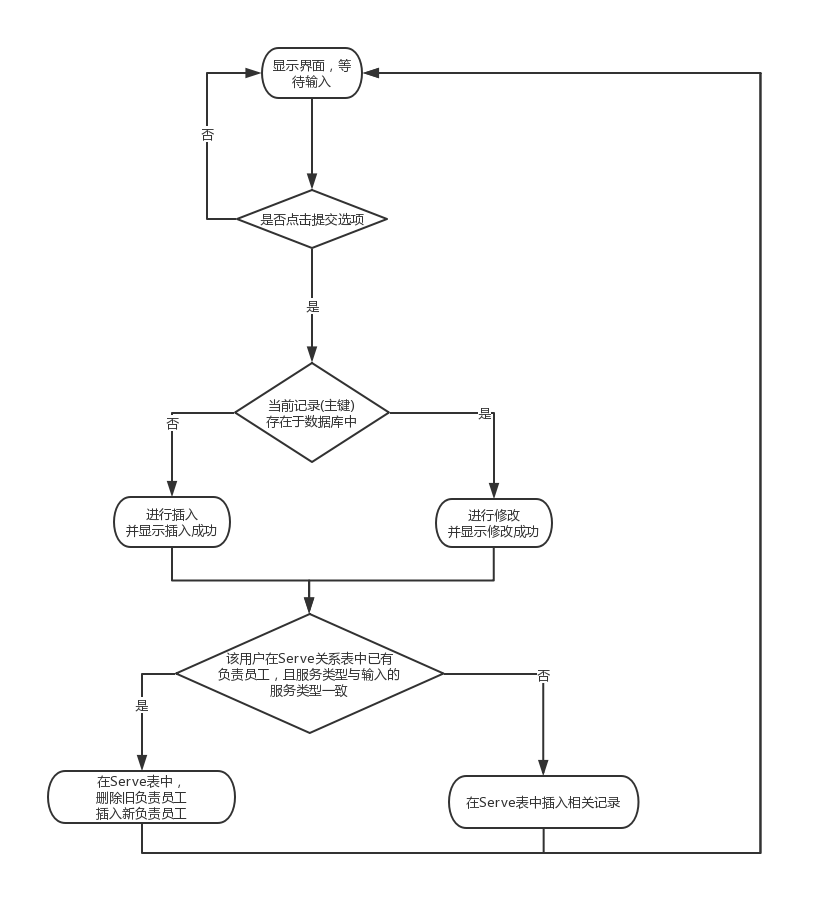
3.1.4 程序流程图

*common\_edit\_*型方法包括如下几个函数，由于对于各个关系实体有着不同的需求和约束，因此各函数的具体流程有所不同，各自的程序流程图如下：

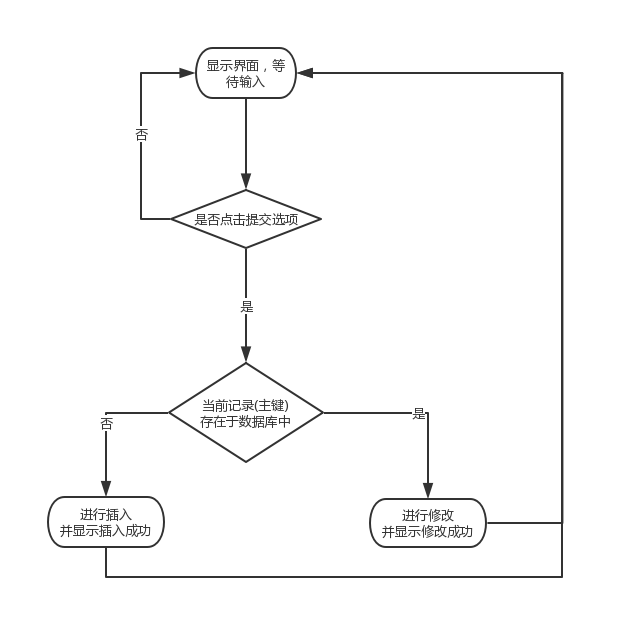
1. ***Common\_edit\_bank*：插入/修改支行信息**



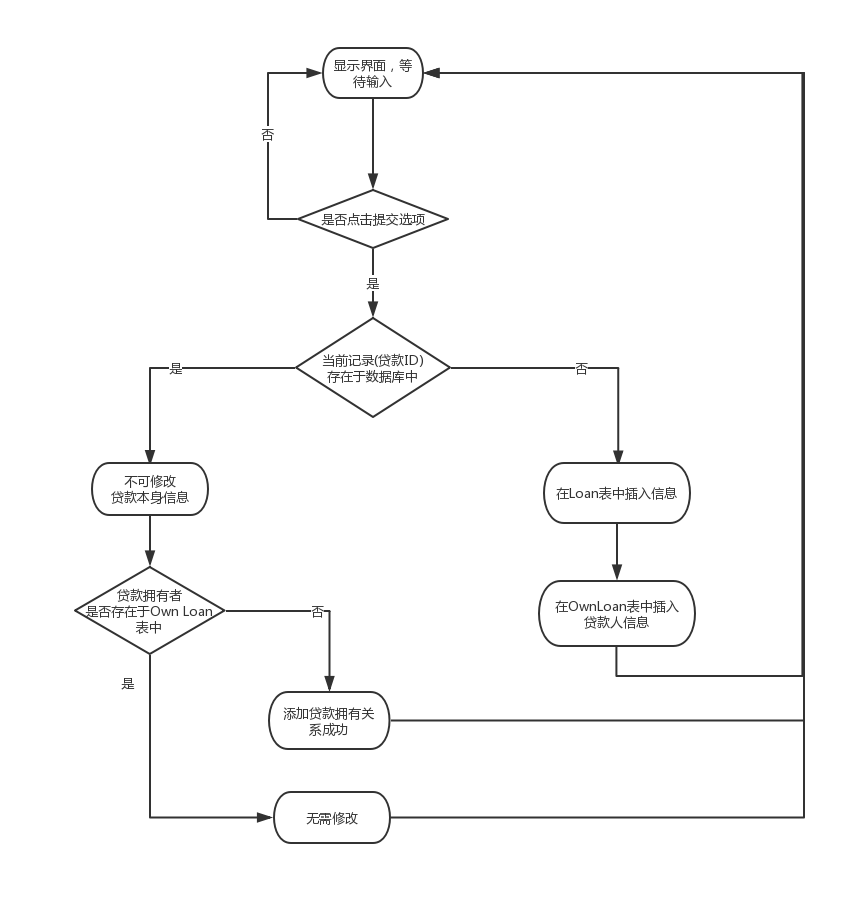
1. ***Common\_edit\_client*：插入/修改客户信息**

****

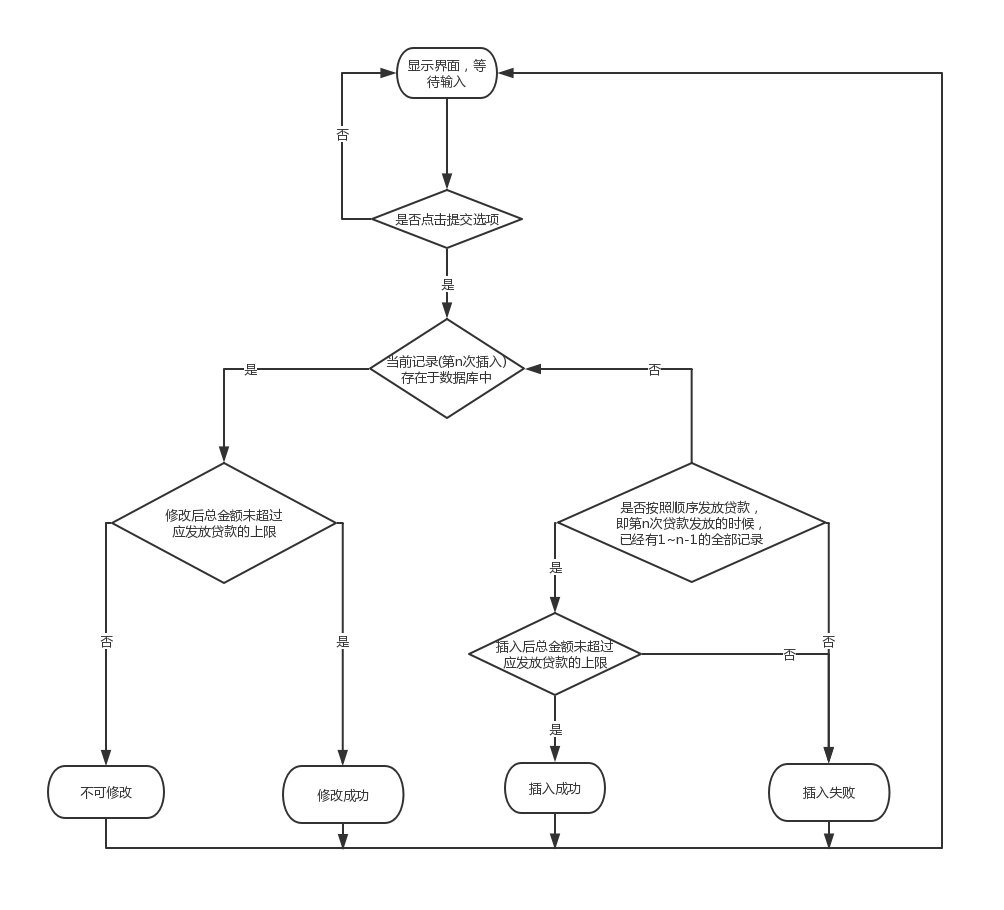
1. ***Common\_edit\_staff*：插入/修改员工信息**



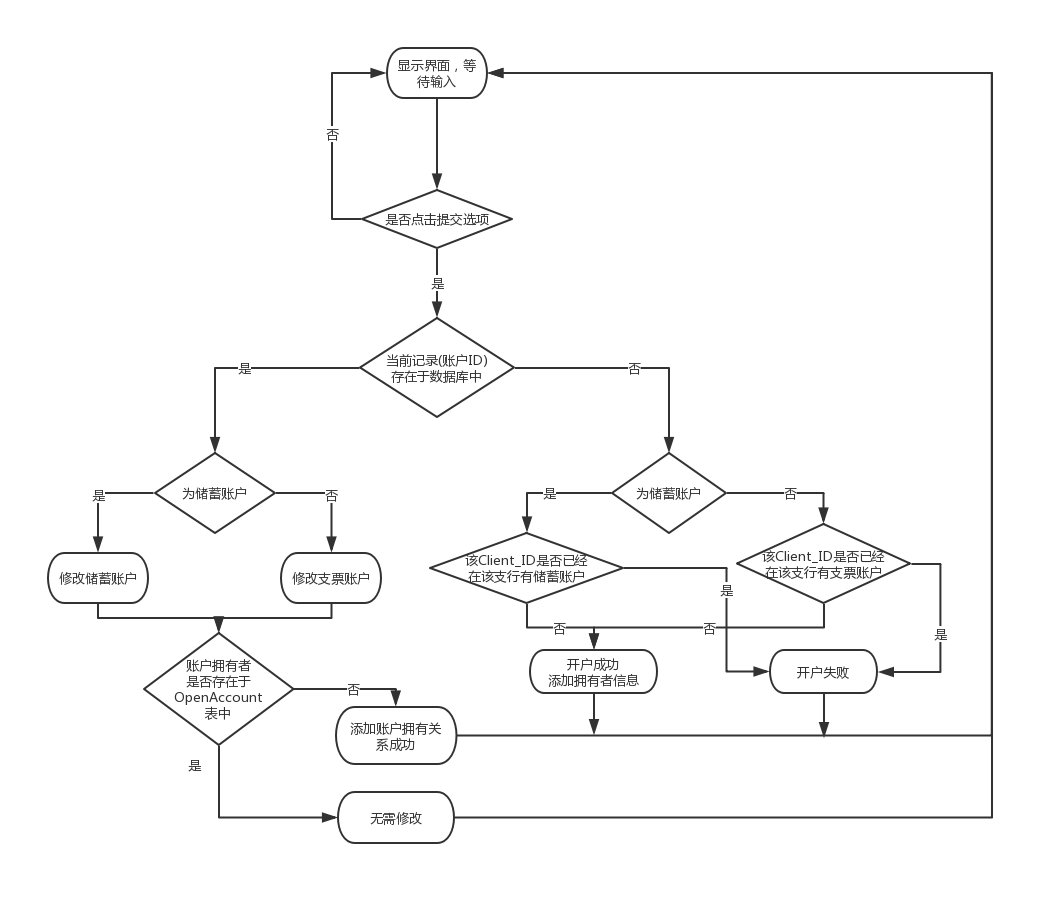
1. ***Common\_edit\_loan*：插入/修改贷款信息**

****

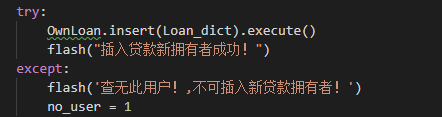
1. ***Common\_edit\_grant*：插入/修改贷款发放信息**



1. ***Common\_edit\_account*：插入/修改账户信息，包括储蓄账户和支票账户**



以上关系流程图中，均默认了三类完整性约束均满足，如不满足参照完整性，则程序会抛出异常，该异常被捕获到之后，会在用户界面上显示出该关系涉及到的用户/员工/账户不存在，从而保证了参照完整性约束。如以下代码，即实现了捕捉参照完整性异常并向用户提示错误信息：



## 3.2 查询/删除模块

3.2.1 输入

该部分的输入和3.1中描述的一致，函数的参数分为Dynamic Model、form和View三个部分。但是除了以form格式输出到后端的信息，由于涉及到删除操作，因此在点击了“删除”选项后，系统还以变量的格式，将待删除记录的主键传输到后端。

在前端html页面中，在删除操作的选项上附加上如下信息：

C:\Users\lwh\AppData\Roaming\Tencent\Users\2231805638\TIM\WinTemp\RichOle\%ZVE7HFO[3MK}D4ZD5E}HIC.png

在后端views.py上，通过如下方式，可以获得变量id的值：

C:\Users\lwh\AppData\Roaming\Tencent\Users\2231805638\TIM\WinTemp\RichOle\]FN)119R060)E}76YG$G]5V.png

该方法属于flask框架中的“GET”方法，而利用form传参的方法为“POST”方法，在设置路由的时候，两者均被列为有效的传参方式：

C:\Users\lwh\AppData\Roaming\Tencent\Users\2231805638\TIM\WinTemp\RichOle\4UM$UKVZ0C$XO4XC8B3((UW.png

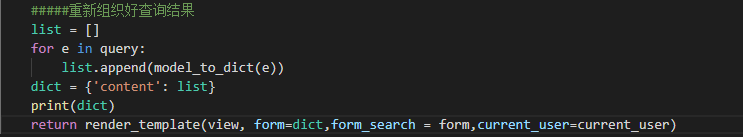
查询操作和3.1中的插入/修改操作类似，从前端输入关键信息后，通过form传递参数到后端，后端依据参数信息进行查询。

3.2.2 输出

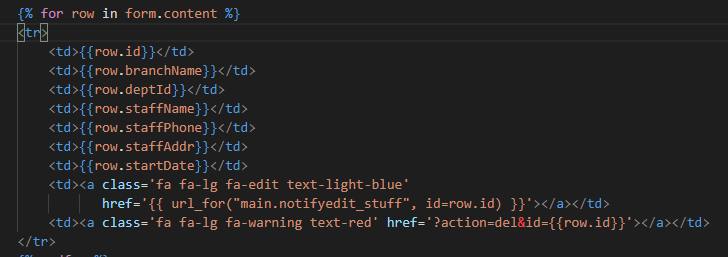
对于查询操作，通过一个字典将所有的查询结果返回到前端，并在html代码中嵌入一个循环来输出到前端：

1. 将输出结果组织为字典：

query表示查询的结果，为一个迭代器，将每一个model类型的数据转化为dict，最后组织成一个列表，列表由一个字典的索引项指示：



1. 在html代码中嵌入循环，将结果以表格的形式输出：



上述过程表现了如何将查询结果输出到用户界面。

对于删除操作，删除操作需要对数据库执行DELETE操作，并通过存储过程检查相关约束，若满足方可执行删除语句，否则不可执行删除语句。删除成功或者失败的相关信息同样通过flash语句将结果显示在用户界面上，提示用户删除失败的原因。

在peewee中，删除语句可以通过以下方法实现：

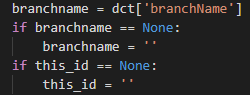
C:\Users\lwh\AppData\Roaming\Tencent\Users\2231805638\TIM\WinTemp\RichOle\BLBL$J3Z[FJT`6(@TRA5%OT.png

3.2.3 详细涉及

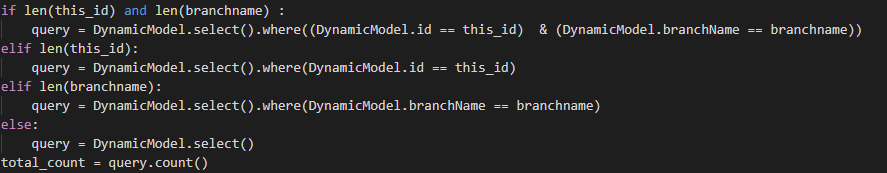
插入/删除模块的函数名统一为***common\_list\_%***（%表示任意字符串），由于每一个对象的约束不同，查询需求也不同，因此需要针对不同的模块涉及不同的查询方式和删除约束：

1. 查询方式：

本实验支持多种查询方式，允许输入项为空，对于每一个输入项，首先检测其是否为空，如果为空，将查询用字符串置为空字符串；否则按照form中的数据生成字符串：



在实际查询的时候，根据查询字段是否为空，可以采用不同的SQL查询语句：



1. 删除约束：

由于本实验中，除了实体基本表以外，还附加了许多关系类基本表，因此我们需要一一检查这些约束是否满足，只有当所有条件均满足的时候，我们才能执行删除操作。

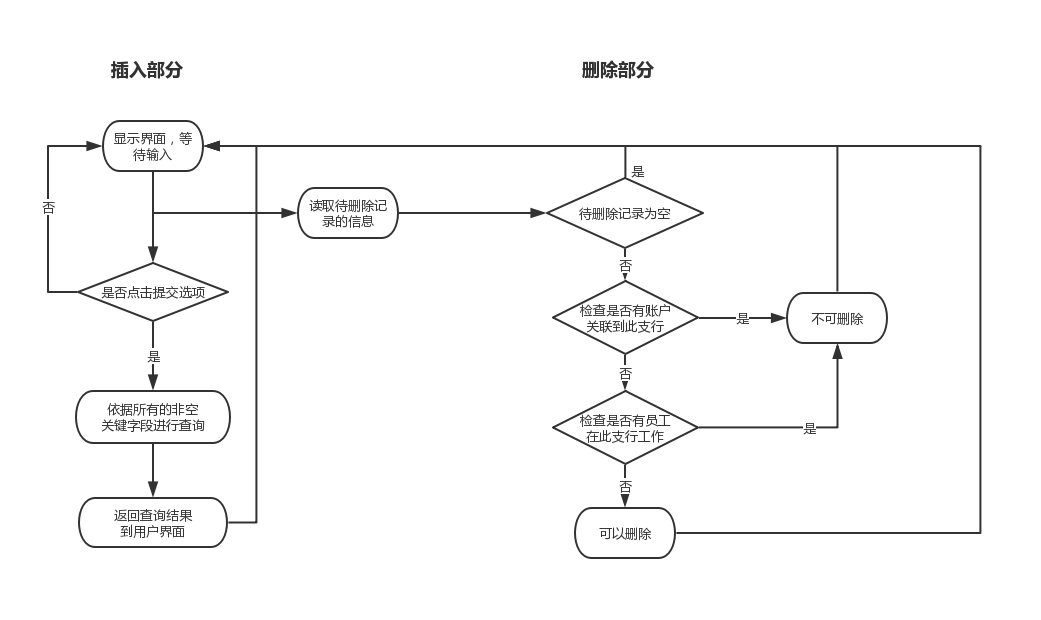
在存储过程执行时，我们首先检查输入的待删除记录的主键是否为*None*，如果主键为None，证明此时用户只是在进行查询操作，没有点击删除选项。所以删除操作的大前提如下：

![C:\Users\lwh\AppData\Roaming\Tencent\Users\2231805638\TIM\WinTemp\RichOle\1](QPYIIJGD%~[]G3~((_@T.png](data:image/png;base64,)

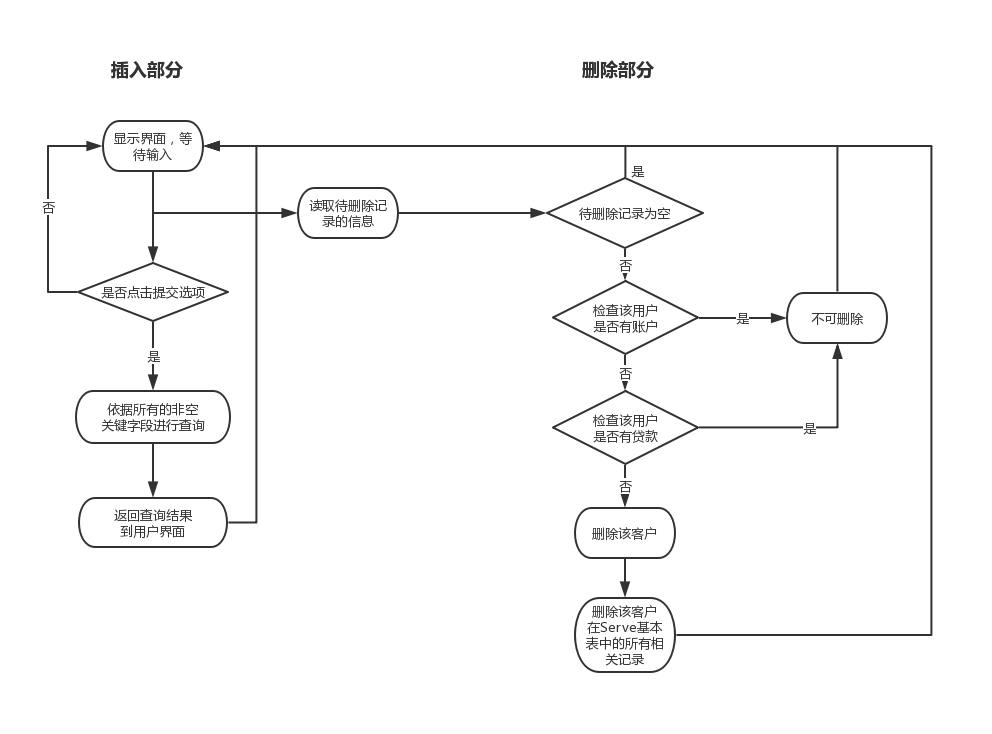
3.3.4程序流程图

本工程中涉及到查询/删除操作的函数及其流程图如下：

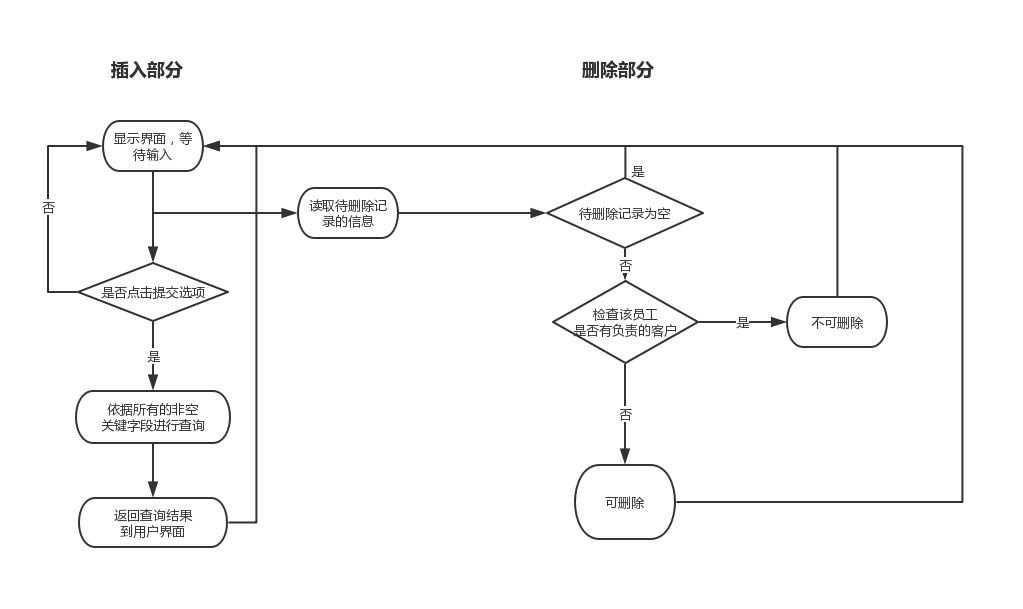
1. ***common\_list\_bank:***对支行的查询/删除操作



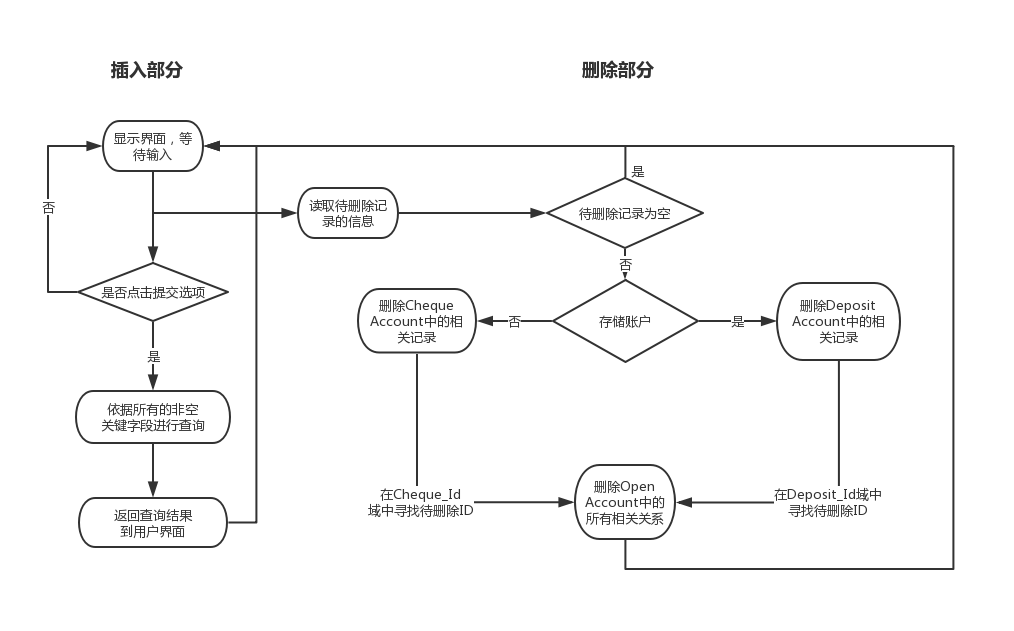
1. ***common\_list\_client:***对客户的查询/删除操作



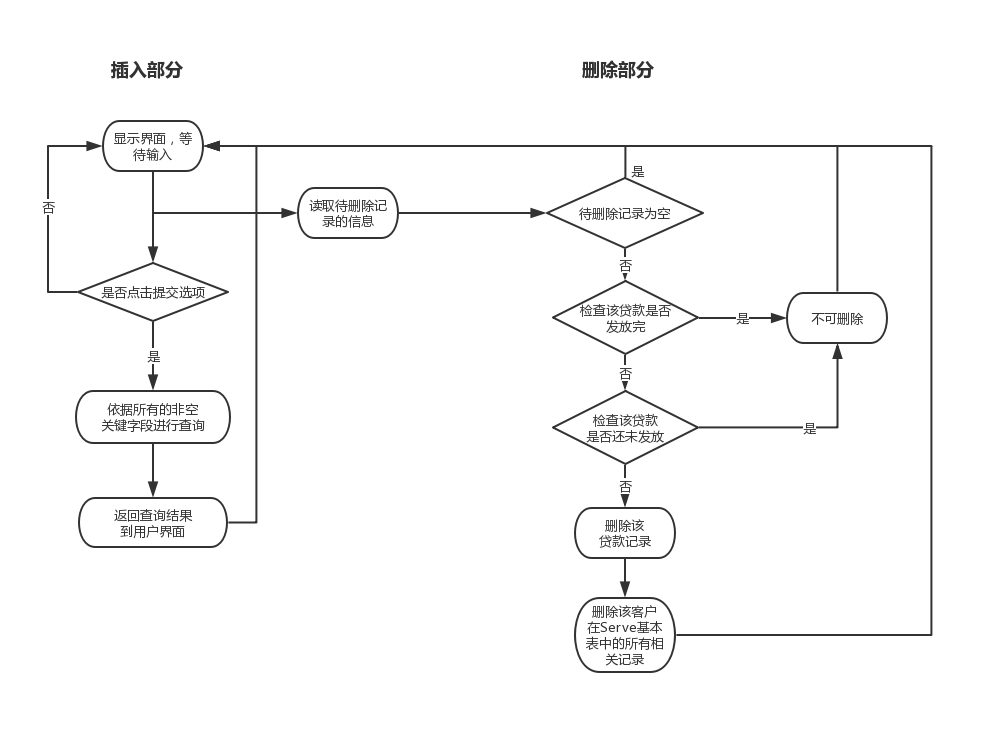
1. ***common\_list\_staff:*** 对员工的查询/删除操作



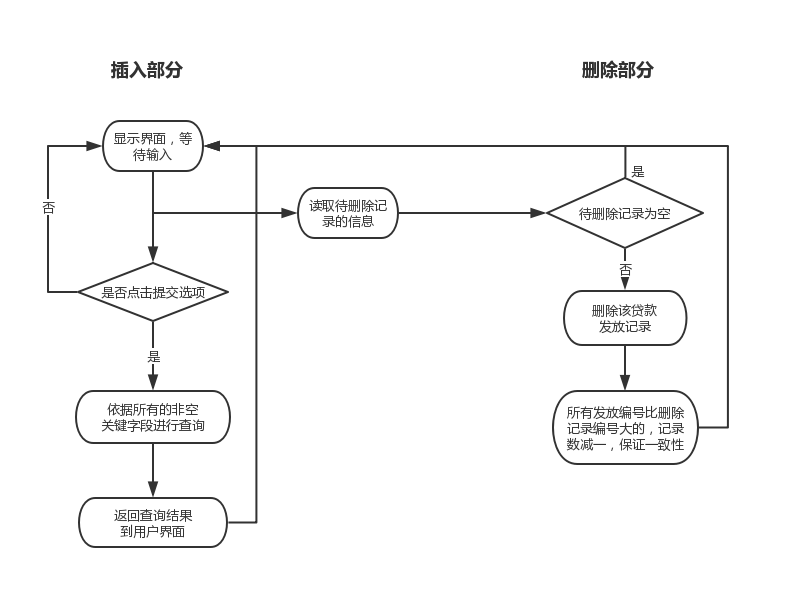
1. ***common\_list\_account:*** 对账户的查询/删除操作



1. ***common\_list\_loan:***对贷款的查询/删除操作

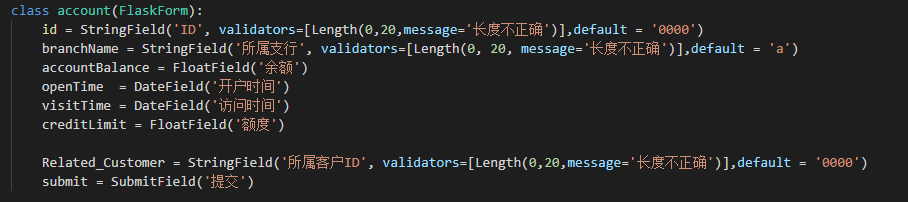


1. ***common\_list\_grant:*** 对贷款支付情况的查询/删除操作。



## 3.3 form模块

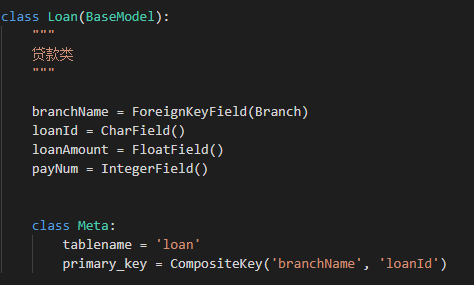
本模块没有输入输出的内容，更类似于一个头文件，里面定义了诸多基于flask框架、用于在前端html页面和后端的python源程序中互相传送数据的form数据类型。form的数据类型定义在form.py中，通常按照如下格式定义：



form类型可以方便的与model类型（与数据库中基本表绑定的类）、字典类型之间进行转换，是flask框架中获得前端数据的一种方便而快捷的方式，本工程中的大部分数据传递都是通过该方法实现。

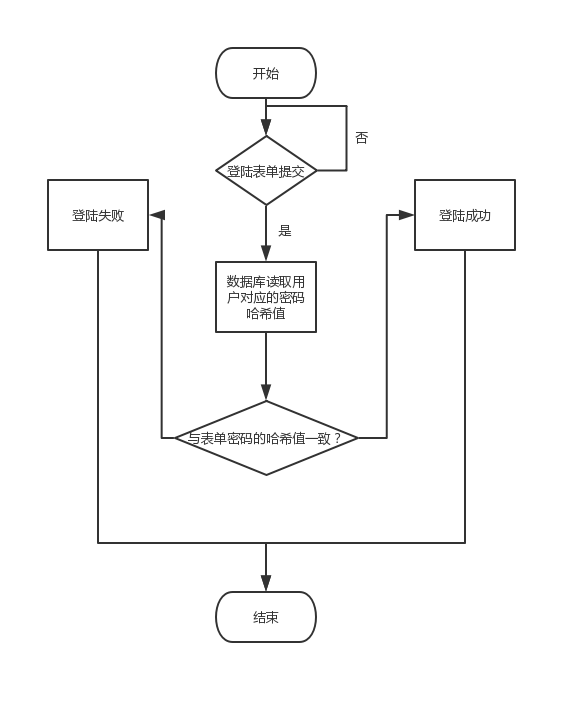
## 3.4 数据库模块

和form类似，models.py文件也没有特定的输入输出，他的作用是将数据库中的基本表和一个类进行绑定，如果程序要对数据库中的基本表操作，只需要对这些类中的元素进行操作即可，本工程中的models通常如下定义：



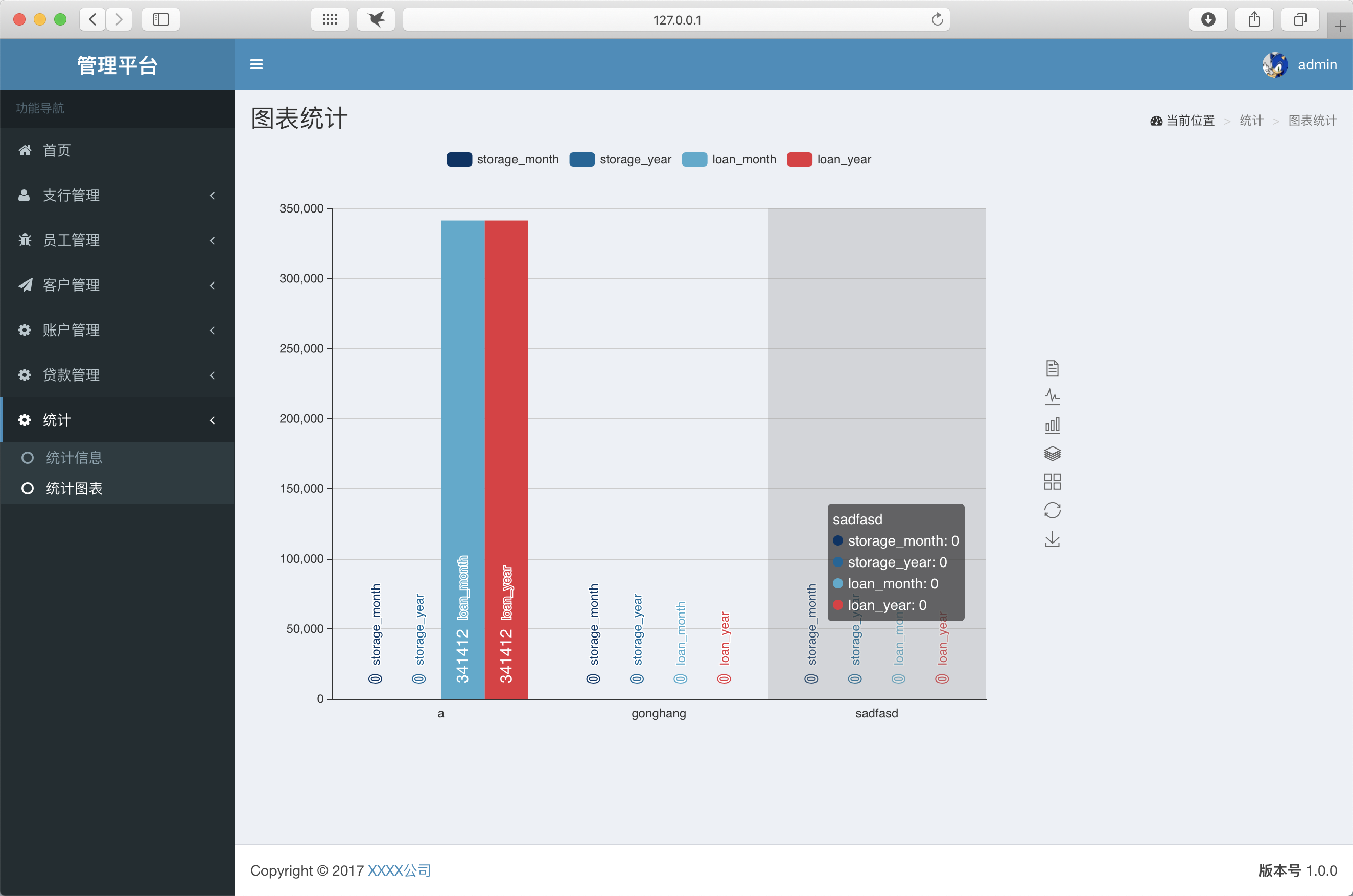
3.4 登陆模块

银行管理系统需要管理员使用管理员的账户密码才能登陆才能进行一定的操作。我们这里直接调用falsk框架的redirect和flask\_login函数进行登录。大致过程是，后端的login函数监听form的submit，一旦有login\_form提交，就去数据库的user表中查找用户名和对应密码的哈希值，然后直接调用flask\_login的verify函数验证登陆。流程图如下：



3.5 统计图表模块

这一模块使用上述的统计模块，将从统计模块中得到的数字数据可视化味图表。图表使用百度开源的echarts来实现。由于echarts的数据无法通过类似flask和bootstrap之间函数传递的方式实现，所以我使用jquery模块，然后用过js的ajax方式来请求后端的数据，后端将所有的数据使用jsonify函数转换为json的格式，最后将数据传入到option的data之中，最后由echart来显示，最终的效果如下：



3.6 路由说明

由于flask的优良的路由管理，我们不用太操心路由。在bootstrap网页的超链接中，直接使用{{ url\_for(你想调用的函数)}}即可，然后在view.py中，会被超链接链接的函数只要在函数前加上类似@main.route('/notifyedit\_bank', methods=['GET', 'POST'])这样的格式即可。

# 4 实现与测试

## 4.1 实现结果

// 给出各个功能需求的实现界面和运行结果。

## 4.2 测试结果

// 给出各个功能需求的测试用例和测试结果。

# 5 总结与讨论

// 总结本系统开发过程中的主要收获、教训。

# 附录 团队介绍与工作分工

// 团队成员介绍。详细陈述各自的任务分工情况。