银行业务管理系统

系统设计与实现报告

姓名：高楚晴
学号：PB18111688

计算机科学与技术学院
中国科学技术大学
2021年7月

0. 目录

1. 系统

1.1系统目标

本系统的主要开发目标是实现一个可用安全的银行管理系统，采用B/S结构，实现系统的前端页面、后端服务和数据库设计构建。

1.2需求说明

1.2.1 数据需求

银行有多个支行。各个支行位于某个城市，每个支行有唯一的名字。银行要监控每个支行的 资产。 银行的客户通过其身份证号来标识。银行存储每个客户的姓名、联系电话以及家庭 住址。为了安全起见，银行还要求客户提供一位联系人的信息，包括联系人姓名、手机号、 Email 以及与客户的关系。客户可以有帐户，并且可以贷款。客户可能和某个银行员工发生 联系，该员工是此客户的贷款负责人或银行帐户负责人。银行员工也通过身份证号来标识。 员工分为部门经理和普通员工，每个部门经理都负责领导其所在部门的员工，并且每个员工 只允许在一个部门内工作。每个支行的管理机构存储每个员工的姓名、电话号码、家庭地 址、所在的部门号、部门名称、部门类型及部门经理的身份证号。银行还需知道每个员工开 始工作的日期，由此日期可以推知员工的雇佣期。银行提供两类帐户——储蓄帐户和支票帐 户。帐户可以由多个客户所共有，一个客户也可开设多个账户，但在一个支行内最多只能开 设一个储蓄账户和一个支票账户。每个帐户被赋以唯一的帐户号。银行记录每个帐户的余 额、开户日期、开户的支行名以及每个帐户所有者访问该帐户的最近日期。另外，每个储蓄 帐户有利率和货币类型，且每个支票帐户有透支额。每笔贷款由某个分支机构发放，能被一 个或多个客户所共有。每笔贷款用唯一的贷款号标识。银行需要知道每笔贷款所贷金额以及 逐次支付的情况(银行将贷款分几次付给客户)。虽然贷款号不能唯一标识银行所有为贷款 所付的款项，但可以唯一标识为某贷款所付的款项。对每次的付款需要记录日期和金额。

1.2.2主要功能需求

- 客户管理：提供客户所有信息的增、删、改、查功能;如果客户存在着关联账户或者贷 款记录，则不允许删除;
- 账户管理： 提供账户开户、销户、修改、查询功能，包括储蓄账户和支票账户;账户号 不允许修改;
- 贷款管理： 提供贷款信息的增、删、查功能，提供贷款发放功能;贷款信息一旦添加成 功后不允许修改;要求能查询每笔贷款的当前状态(未开始发放、发放中、已全部发 放);处于发放中状态的贷款记录不允许删除;
- 业务统计： 按业务分类(储蓄、贷款)和时间(月、季、年)统计各个支行的业务总金 额和用户数，要求对统计结果同时提供表格和曲线图两种可视化展示方式。

1.3 本报告的主要贡献

- 提供了详细的数据库设计方案
- 提供了满足以上银行系统的解决方案
- 给出了前后端功能结构设计和模块关系
- 给出了展示结果

1.4 Get Start

在目录文件中`config.py`中更改配置，用来连接数据库

```
class DevelopmentConfig(Config):
    DEBUG = True
    SQLALCHEMY_DATABASE_URI =
'mysql+pymysql://root:passwd@localhost:3306/dbname'
    SQLALCHEMY_TRACK_MODIFICATIONS = True
```

在项目目录下配置环境变量`set FLASK_APP=bank.py`

启动服务器`python -m flask run`并在浏览器中打开

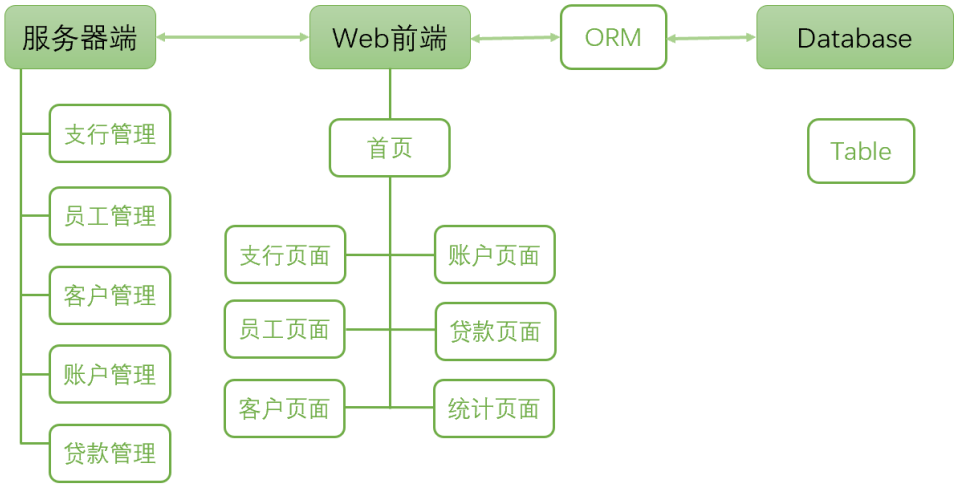
2.总体设计

2.1 项目结构

本项目采用Flask作为后端结构

```
./
├── README.md                                  # 项目文档
├── app                                        # 主体代码文件夹
|   ├── __init__.py
|   ├── main
|   |   ├── __init__.py
|   |   ├── errors.py                          # 错误页面处理
|   |   ├── forms.py                           # 表单类
|   |   └── views.py                           # 视图函数
|   ├── models.py                              # 数据库模型
|   ├── static                                 # css静态文件
|   |   └── styles.css
|   └── templates/                             # html文件
├── bank.py                                    # main函数
├── mybank.sql                                 # mysql文件，用于构建数据库
├── config.py                                  # 配置文件
└── venv                                       # 虚拟环境
```

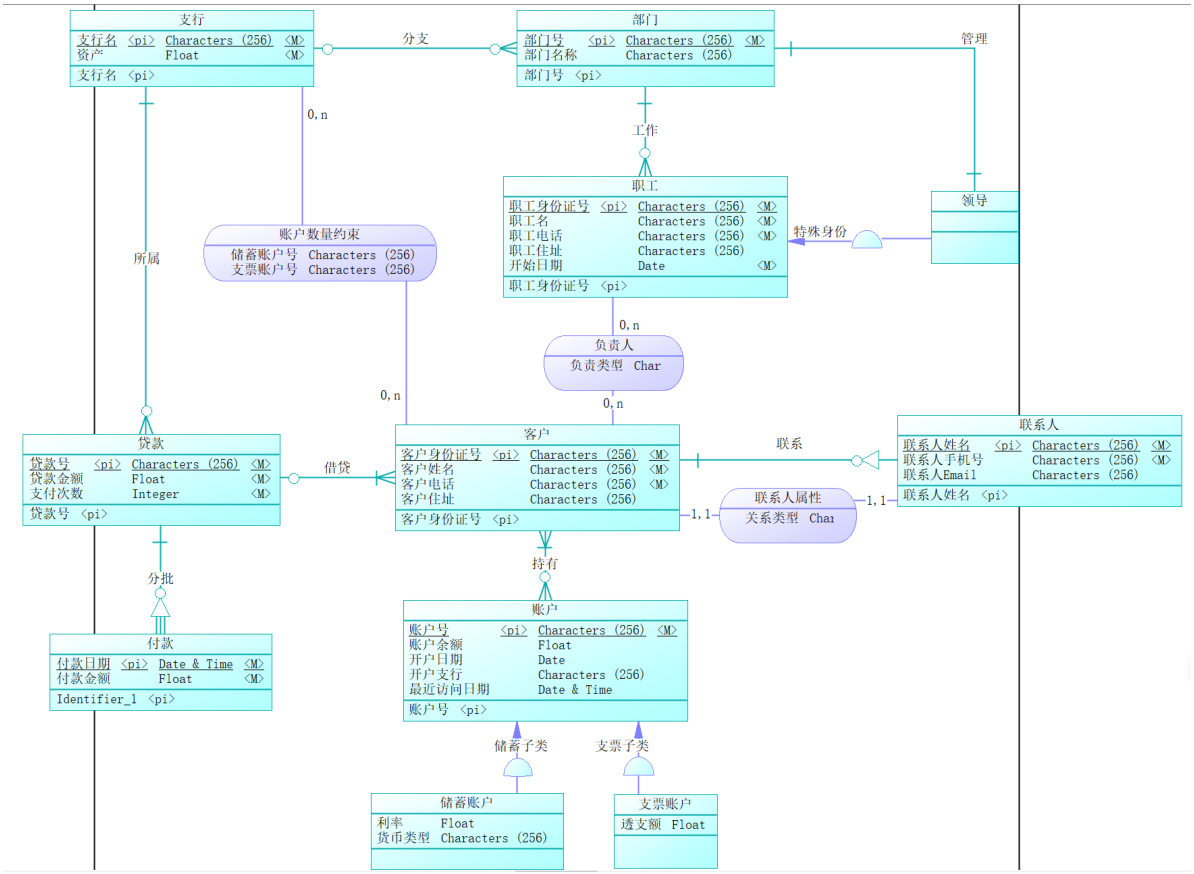
2.2 系统模块结构



2.3 系统工作流程

2.4 数据库设计

2.4.1ER图



2.4.2逻辑数据库结构

主要介绍涉及到的表结构

2.4.2.1 支行表

```
create  table branches (
    name VARCHAR(100) NOT NULL,  --支行名
    city VARCHAR(20),            --所在城市
    asset INTEGER,               --总资产
    constraint branches_pk PRIMARY KEY (name)
);
```

2.4.2.2 客户表

```
create table clients(
    id VARCHAR(20) NOT NULL,         --客户ID
    name VARCHAR(20),                --客户姓名
    phone VARCHAR(15),               --客户电话
    address VARCHAR(256),            --客户地址
    contact_name VARCHAR(20),        --联系人姓名
    contact_phone VARCHAR(15),       --联系人电话
    contact_email VARCHAR(30),       --联系人邮箱
    contact_relation VARCHAR(40),    --联系人和客户关系
    constraint clients_pk PRIMARY KEY(id)
);
```

2.4.2.3 员工表

存储银行员工的信息

```
create table employees (
    id VARCHAR(20) NOT NULL,     --员工id，主键
    branch_name VARCHAR(100),    --支行名称
    name VARCHAR(20),            --员工姓名
    phone VARCHAR(15),           --员工电话
    address VARCHAR(256),        --员工地址
    enroll_date DATETIME,        --入职日期
    constraint employees_pk PRIMARY KEY (id),
    constraint employees_fk FOREIGN KEY(branch_name) REFERENCES branches (name)
);
```

2.4.2.4 贷款表

存储贷款账户信息

```
create table loans(
    id VARCHAR(20) NOT NULL,          --贷款号，主键
    branch_name VARCHAR(100),         --支行名
    employee_id VARCHAR(20),          --负责员工id
    amount FLOAT,                     --贷款总金额
    status VARCHAR(20),               --贷款状态
    constraint loans_pk PRIMARY KEY (id),
    constraint loans_fk_name FOREIGN KEY(branch_name) REFERENCES branches (name),
    constraint loans_fk_id FOREIGN KEY(employee_id) REFERENCES employees (id)
);
```

2.4.2.5 储蓄账户

存储储蓄账户信息

```
create table savings(
    id VARCHAR(20) NOT NULL,          --账户ID，主键
    branch_name VARCHAR(100),         --支行名
    employee_id VARCHAR(20),          --负责员工ID
    balance FLOAT,                    --账户余额
    open_date DATETIME,               --开户日期
    interest_rate FLOAT,              --利率
    currency_type VARCHAR(5),         --货币类型
    last_access_date DATETIME,        --最后一次使用时间
    constraint savings_pk PRIMARY KEY (id),
    constraint savings_fk_name FOREIGN KEY(branch_name) REFERENCES branches (name),
    constraint savings_fk_id FOREIGN KEY(employee_id) REFERENCES employees (id)
);
```

2.4.2.6 支票账户

存储支票账户信息

```
create table checks(
    id VARCHAR(20) NOT NULL,          --支票号，主键
    branch_name VARCHAR(100),         --发行支行名
    employee_id VARCHAR(20),          --负责员工id
    balance FLOAT,                    --余额
    open_date DATETIME,               --开户日期
    over_draft FLOAT,                 --透支额度
    last_access_date DATETIME,        --最后一次使用时间
    constraint checks_pk PRIMARY KEY (id),
    constraint checks_fk_name FOREIGN KEY(branch_name) REFERENCES branches (name),
    constraint checks_fk_id FOREIGN KEY(employee_id) REFERENCES employees (id)
);
```

2.4.2.7 客户-贷款表

存储客户拥有的贷款，用来限制。

```
create table hasloans(
    client_id VARCHAR(20) NOT NULL,      --客户id
    loan_id VARCHAR(20) NOT NULL,        --贷款id
    PRIMARY KEY (client_id, loan_id),
    constraint hasloans_pk FOREIGN KEY(client_id) REFERENCES clients (id),
    constraint hasloans_fk FOREIGN KEY(loan_id) REFERENCES loans (id)
);
```

2.4.2.8 客户-储蓄账户表

存储客户拥有的储蓄账户，用来进行限制。

```
create table clientsavings (
    client_id VARCHAR(20) NOT NULL,      --客户id
    saving_id VARCHAR(20) NOT NULL,      --储蓄账户id
    constraint  clients_pk PRIMARY KEY (client_id, saving_id),
    constraint  clients_fk_id FOREIGN KEY(client_id) REFERENCES clients (id),
    constraint  clients_fk_sid FOREIGN KEY(saving_id) REFERENCES savings (id)
);
```

2.4.2.9 客户-支票账户

存储客户拥有的支票账户，用来限制。

```
CREATE TABLE clientchecks (
    client_id VARCHAR(20) NOT NULL,      --客户id
    check_id VARCHAR(20) NOT NULL,       --支票id
    constraint  clientcheck_pk PRIMARY KEY (client_id, check_id),
    constraint  clientcheck_fk_id1 FOREIGN KEY(client_id) REFERENCES clients (id),
    constraint  clientcheck_fk_id2 FOREIGN KEY(check_id) REFERENCES checks (id)
);
```

2.4.2.10 贷款记录表

存储发放贷款的记录信息

```
create table loanlogs (
    id VARCHAR(30) NOT NULL,     --放贷号
    loan_id VARCHAR(20),         --贷款号
    date DATETIME,               --放贷日期
    amount FLOAT,                --放贷金额
    constraint loanlogs_pk PRIMARY KEY (id),
    constraint loanlogs_fk FOREIGN KEY(loan_id) REFERENCES loans (id)
);
```

2.4.2.11 储蓄账户限制表

为了限制： 客户在一个支行内最多只能开 设一个储蓄账户和一个支票账户

```
CREATE TABLE savingconstraints (
    client_id VARCHAR(20) NOT NULL,          --客户id
    branch_name VARCHAR(100) NOT NULL,       --支行名
    saving_id VARCHAR(20) NOT NULL,          --储蓄账号id
    constraint savingconstraints_pk PRIMARY KEY (client_id, branch_name, saving_id),
    CONSTRAINT con1 UNIQUE (client_id, branch_name),
    constraint savingconstraints_fk_id1 FOREIGN KEY(client_id) REFERENCES clients (id),
    constraint savingconstraints_fk_name FOREIGN KEY(branch_name) REFERENCES branches (name),
    constraint savingconstraints_fk_id2 FOREIGN KEY(saving_id) REFERENCES savings (id)
);
```

2.4.2.12 支票账户限制表

为了限制： 客户在一个支行内最多只能开 设一个储蓄账户和一个支票账户

```
CREATE TABLE checkconstraint (
    client_id VARCHAR(20) NOT NULL,          --客户id
    branch_name VARCHAR(100) NOT NULL,       --支行名
    check_id VARCHAR(20) NOT NULL,           --支票id
    constraint checkconstraint_pk PRIMARY KEY (client_id, branch_name, check_id),
    CONSTRAINT con1 UNIQUE (client_id, branch_name),
    constraint checkconstraint_fk_id1 FOREIGN KEY(client_id) REFERENCES clients (id),
    constraint checkconstraint_fk_name FOREIGN KEY(branch_name) REFERENCES branches (name),
    constraint checkconstraint_fk_id2 FOREIGN KEY(check_id) REFERENCES checks (id)
);
```

2.4.15 支行操作记录表

用来记录支行的各个操作，便于统计支行信息

```
CREATE TABLE branchrecords (
    id INTEGER NOT NULL,             --支行id
    branch_name VARCHAR(100),        --支行名
    OpType VARCHAR(4),               --操作类型
    OpTime DATETIME,                 --操作时间
    OpMoney FLOAT,                   --操作金额
    constraint branchrecords_pk PRIMARY KEY (id),
    constraint branchrecords_fk FOREIGN KEY(branch_name) REFERENCES branches (name)
);
```

3. 详细设计

3.1 后端

后端采用Flask架构， 采用`flask-sqlalchemy`库来进行对数据库进行操作。

3.1.1支行管理模块

设计框架

本模块一共四个视图函数：

- ```
  @main.route('/branch', methods=['GET', 'POST'])
  def branch()  # 查询
  ```

- ```
  @main.route('/branch_all')
  def branch_show_all()  #展示所有支行
  ```

- ```
  @main.route('/branch_edit/<string:branch_name>', methods=['GET', 'POST'])
  def branch_edit(branch_name)   #增加或者对某个支行进行编辑
  ```

- ```
  @main.route('/branch_delete/<string:id>')
  def branch_delete(id)  # 删去某个支行
  ```

两个表单类：

- 支行搜查表单：

  ```
  class BranchSearchForm(FlaskForm):
      name = StringField("支行名")
      city = StringField("所在城市")
      submit = SubmitField("搜索")
  ```

- 支行编辑表单：

  ```
  class BranchEditForm(FlaskForm):
      name = StringField("支行名", validators=[InputRequired()])
      city = StringField("所在城市", validators=[InputRequired()])
      asset = FloatField("资产", validators=[InputRequired(),
                                             NumberRange(min=0., message='资产必须大于0')])
      submit = SubmitField("提交")
  ```

支行搜索

默认会展示所有的分行。当要查询某个具体的分行时（提供模糊查询），利用表单：

- 输入：分行的名字`form.name.data`或者所在城市`form.city.data`的关键词
- 输出：
  - `form`： 表单信息
  - `branches`：搜索到的支行的列表
  - `pagination`：分页的一些信息
- 错误情况处理
  - 没有该支行的信息

添加支行

- 输入：路径为`/branch_edit/create`，输入上列编辑表单的基本信息
- 输出：根据创建的支行是否符合标准来返回成功或者失败的信息。
  - 成功：“已经添加新支行”+`branch.name`
- 错误情况处理：
  - 支行已经存在
  - 支行资产不能为负数

编辑支行

- 输入：路径为`/branch_edit/<string:branch_name>`，输入上列编辑表单的基本信息
- 输出： 根据信息的符合标准与否返回相应的信息：
  - 成功：“该支行信息已经更新‘+`branch.name`
- 错误情况处理
  - 支行资产不能为负数

删除支行

- 输入：支行的名称
- 输出：根据信息的符合标准与否返回相应的信息：
  - 成功：“删除成功”
  - 失败： ”该支行仍有贷款，无法删除“ / “该支行仍有储蓄账户，无法删除” / “该支行仍有支票账户，无法删除” / “该支行仍有员工，无法删除”+ `form`
- 错误情况处理：
  - 支行仍有贷款
  - 支行仍有储蓄账户
  - 支行仍有支票账户
  - 支行仍有员工

3.1.2 员工模块

设计框架

本模块一共四个视图函数：

- ```
  @main.route('/employee', methods=['GET', 'POST'])
  def employee():  # 查询
  ```
- ```
  @main.route('/employee_all')
  def employee_show_all():  #展示所有客户
  ```
- ```
  @main.route('/employee_edit/<string:employee_id>', methods=['GET', 'POST'])
  def employee_edit(employee_id):   #增加或者对某个客户进行编辑
  ```
- ```
  @main.route('/employee_delete/<string:id>')
  def employee_delete(id):  # 删去某个客户
  ```

两个表单类：

- 员工信息编辑表单：

```
class EmployeeEditForm(FlaskForm):
    id = StringField("身份证号", validators=[
        InputRequired(),
        Regexp(r'^([1-9]\d{5}[12]\d{3}(0[1-9]|1[012])(0[1-9]|[12][0-
9]|3[01])\d{3}[0-9xX])$',
               0, "身份证号不合法")])
    name = StringField("姓名", validators=[InputRequired()])
    branch_name = SelectField('所在支行')
    phone = StringField("手机号")
    address = StringField("联系地址")
    enroll_date = DateField("入职日期", format="%Y-%m-%d", validators=
[DataRequired()])
    submit = SubmitField("提交")
```

- 员工搜索表单类似，在此不再重复

搜索员工

默认会展示所有的员工。当要查询某个具体的员工时（提供模糊查询），利用表单：

- 输入：路径为`/employee`,输入上面编辑表单的基本信息，可以输入部分。
- 输出：
  - `form`： 表单信息
  - `employees`：搜索到的员工的列表
  - `pagination`：分页的一些信息
- 错误情况处理
  - 找不到该员工

添加员工

- 输入：路径为`/employee_edit/create`，输入上列编辑表单的基本信息
- 输出：根据创建的员工是否符合标准来返回成功或者失败的信息。
  - 成功：“已经添加员工”+ `employee.id`
- 错误情况处理
  - 该员工已任职（身份证已存在）

编辑员工

- 输入：路径为`/employee_edit/<string:employee_id>`，输入上列编辑表单的基本信息
- 输出： 根据员工信息的符合标准与否返回相应的信息：
  - 成功：“该员工信息已经更新‘ + `employee.id`

删除员工

- 输入：员工的ID
- 输出：根据员工信息的符合标准与否返回相应的信息：
  - 成功："删除成功"
- 错误情况处理
  - 该员工仍有负责贷款
  - 该员工仍有负责储蓄业务
  - 该员工仍有负责支票业务

3.1.3 客户模块

设计框架

本模块一共四个视图函数：

- ```
  @main.route('/client', methods=['GET', 'POST'])
  def client():  # 查询
  ```
- ```
  @main.route('/client_all')
  def client_show_all(): #展示所有客户
  ```
- ```
  @main.route('/client_edit/<string:client_id>', methods=['GET', 'POST'])
  def client_edit(client_id):   #增加或者对某个客户进行编辑
  ```
- ```
  @main.route('/client_delete/<string:id>')
  def client_delete(id):  # 删去某个客户
  ```

两个表单类：

- 客户信息编辑表单：

```
class ClientEditForm(FlaskForm):
  id = StringField("身份证号", validators=[
        InputRequired(),
        Regexp(r'^([1-9]\d{5}[12]\d{3}(0[1-9]|1[012])(0[1-9]|[12][0-
9]|3[01])\d{3}[0-9xX])$',
               0,"身份证号不合法")])
    name = StringField("姓名", validators=[InputRequired()])
    phone = StringField("手机号", validators=[InputRequired()])
    address = StringField("联系地址", validators=[InputRequired()])
    contact_name = StringField("联系人姓名", validators=[InputRequired()])
    contact_phone = StringField("联系人手机号")
    contact_email = StringField("联系人邮箱", validators=[Optional(),
Email()])
    contact_relation = StringField("联系人与客户关系")
    submit = SubmitField("提交")
```

- 客户搜索表单类似，不再重复

搜索客户

默认会展示所有的客户。当要查询某个具体的客户时（提供模糊查询），利用表单：

- 输入：路径为`/client`,输入上面编辑表单的基本信息，可以输入部分。
- 输出：
  - `form`： 表单信息
  - `client`：搜索到的员工的列表
  - `pagination`：分页的一些信息
- 错误情况处理
  - 找不到该客户

添加客户

- 输入：路径为`/client_edit/create`，输入上列编辑表单的基本信息
- 输出：根据创建的员工是否符合标准来返回成功或者失败的信息。
  - 成功：“已经添加客户”+ `client.id`
- 错误情况处理
  - 该客户已存在（身份证已存在）

编辑客户

- 输入：路径为`/client_edit/<string:client_id>`，输入上列编辑表单的基本信息
- 输出： 根据客户信息的符合标准与否返回相应的信息：
  - 成功：“该客户信息已经更新‘ + `client.id`

删除客户

- 输入：客户的ID
- 输出：根据客户信息的符合标准与否返回相应的信息：
  - 成功：“删除成功”
  - 失败： ”该客户仍有贷款，无法删除“ / “该客户仍有某储蓄账户，无法删除” / “该客户仍有某支票账户，无法删除” + `form`
- 错误情况处理
  - 该客户仍有贷款
  - 该客户仍有储蓄账户
  - 该客户仍有支票账户

3.1.4 储蓄账户模块

设计框架

本模块一共四个视图函数：

- ```
  @main.route('/saving_account', methods=['GET', 'POST'])
  def saving_account():  # 查询
  ```
- ```
  @main.route('/saving_account_all')
  def saving_account_show_all():   #展示所有储蓄账户
  ```
- ```
  @main.route('/saving_account_edit/<string:account_id>', methods=['GET',
  'POST'])
  def saving_account_edit(account_id): #增加或者对某个储蓄账户进行编辑
  ```
- ```
  @main.route('/saving_account_delete/<string:account_id>')
  def saving_account_delete(account_id):# 删去某个储蓄账户
  ```

两个表单类：

- 储蓄账户信息编辑表单：

  ```
  class SavingAccountEditForm(FlaskForm):
      id = StringField("账号编号", validators=[InputRequired()])
      branch_name = SelectField('开户支行', validators=[InputRequired()])
      employee_id = SelectField('负责员工', validators=[InputRequired()])
      balance = FloatField("余额", validators=[InputRequired(),
                                               NumberRange(min=0., message="余额
  必须大于0")])
      interest_rate = StringField("利率", validators=[InputRequired()])
      currency_type = StringField("货币类型", validators=[InputRequired()])
      clients = SelectMultipleField("所属客户", validators=[InputRequired()])
      submit = SubmitField("提交")
  ```

- 储蓄账户查询表单类似

搜索账户

默认会展示所有的储蓄账户。当要查询某个具体的储蓄账户时（提供模糊查询），利用表单：

- 输入：路径为`/saving`,输入上面编辑表单的基本信息，可以输入部分。
- 输出：
  - `form`：表单信息
  - `saving_account`：搜索到的储蓄账户的列表
  - `pagination`：分页的一些信息

添加账户

- 输入：路径为`/saving_edit/create`，输入上列编辑表单的基本信息
- 输出：根据创建的储蓄账户是否符合标准来返回成功或者失败的信息。
  - 成功：“已经添加储蓄账户”+ `saving_account.id`
  - 失败：提示信息并刷新表单
- 错误情况处理
  - 该账户编号已经存在
  - 不允许余额为负数
  - 每个客户在一个支行只能有一个储蓄账户

编辑账户

- 输入：路径为`/saving_edit/<string:saving_id>`，输入上列编辑表单的基本信息
- 输出： 根据储蓄账户信息的符合标准与否返回相应的信息：
  - 成功：“该储蓄账户信息已经更新‘ + `saving_account.id`
  - 失败：提示信息并刷新表单
- 错误情况处理
  - 不允许余额为负数
  - 每个客户在一个支行只能有一个储蓄账户

删除账户

- 输入：储蓄账户的ID
- 输出：根据储蓄账户信息的符合标准与否返回相应的信息：
  - 成功：“删除成功”
  - 失败：提示信息并刷新表单
- 错误情况处理
  - 该账户仍有余额

3.1.5 支票账户模块

设计框架

本模块一共四个视图函数：

- ```
  @main.route('/check_account', methods=['GET', 'POST'])
  def check_account(): # 查询
  ```
- ```
  @main.route('/check_account_all')
  def check_account_show_all():  #展示所有支票账户
  ```

- ```
  @main.route('/check_account_edit/<string:account_id>', methods=['GET',
  'POST'])
  def check_account_edit(account_id):    #增加或者对某个支票账户进行编辑
  ```

- ```
  @main.route('/check_account_delete/<string:account_id>')
  def check_account_delete(account_id):# 删去某个支票账户
  ```

两个表单类：

- 支票账户信息编辑表单：

  ```
  class CheckAccountEditForm(FlaskForm):
      id = StringField("账号编号", validators=[InputRequired()])
      branch_name = SelectField('开户支行', validators=[InputRequired()])
      employee_id = SelectField('负责员工', validators=[InputRequired()])
      balance = FloatField("余额", validators=[InputRequired(),
                                               NumberRange(min=0., message="余额
  必须大于0")])
      over_draft = FloatField("透支额", validators=[InputRequired(),
                                                   NumberRange(min=0.,
  message="透支额必须大于0")])
      clients = SelectMultipleField("所属客户", validators=[InputRequired()])
      submit = SubmitField("提交")
  ```

- 支票账户搜索表单类似

搜索账户

默认会展示所有的支票账户。当要查询某个具体的支票账户时（提供模糊查询），利用表单：

- 输入：路径为`/saving`,输入上面编辑表单的基本信息，可以输入部分。
- 输出：
  - `form`： 表单信息
  - `saving_account`：搜索到的储蓄账户的列表
  - `pagination`：分页的一些信息

添加账户

- 输入：路径为`/check_edit/create`，输入上列编辑表单的基本信息
- 输出：根据创建的支票账户是否符合标准来返回成功或者失败的信息。
  - 成功：“已经添加支票账户”+ `check_account.id`
  - 失败：错误提示并刷新表单
- 错误情况处理
  - 该账户编号已经存在
  - 不允许余额少于透支额
  - 每个客户在一个支行只能有一个支票账户

编辑账户

- 输入：路径为`/check_edit/<string:check_id>`，输入上列编辑表单的基本信息
- 输出： 根据支票账户信息的符合标准与否返回相应的信息：
  - 成功：“该支票账户信息已经更新’ + `check_account.id`
  - 失败：提示信息并刷新表单
- 错误情况处理
  - 不允许余额少于透支额
  - 每个客户在一个支行只能有一个支票账户

删除账户

- 输入：支票账户的ID
- 输出：根据支票账户信息的符合标准与否返回相应的信息：
  - 成功：“删除成功”
  - 失败：提示信息并刷新表单
- 错误情况处理
  - 该账户仍有余额

3.1.6 贷款模块

设计框架

本模块一共六个视图函数：

- ```
  @main.route('/loan', methods=['GET', 'POST'])
  def loan() # 查询
  ```
- ```
  @main.route('/loan_all')
  def loan_show_all():   #展示所有贷款
  ```
- ```
  @main.route('/loan_log_edit/<string:loan_log_id>', methods=['GET', 'POST'])
  def loan_log_edit(loan_log_id):   #增加或者对某个贷款记录进行编辑
  ```
- ```
  @main.route('/loan_delete/<string:loan_id>')
  def loan_delete(loan_id):   # 删去某个贷款
  ```
- ```
  @main.route('/loan_log/<string:loan_id>')
  def loan_log(loan_id):      #贷款记录查询
  ```
- ```
  @main.route('/check_account_delete/<string:account_id>')
  def check_account_delete(account_id):  #  增加或者对某个贷款进行编辑
  ```

三个表单类：

- 贷款信息编辑表单：

```
class LoanEditForm(FlaskForm):
    id = StringField("贷款编号", validators=[InputRequired()])
    branch_name = SelectField('贷款支行', validators=[InputRequired()])
    employee_id = SelectField('负责员工', validators=[InputRequired()])
    amount = FloatField("贷款金额", validators=[InputRequired(),
                                            NumberRange(min=0., message="贷款
金额必须大于0")])
    clients = SelectMultipleField("所属客户", validators=[InputRequired()])
    submit = SubmitField("提交")
```

- 贷款搜索表单和放贷记录表格类似，约束会少一点

贷款

搜索贷款

默认会展示所有的贷款。当要查询某个具体的贷款记录时（提供模糊查询），利用表单：

- 输入：路径为`/loan`,输入上面编辑表单的基本信息，可以输入部分。
- 输出：
  - `form`： 表单信息
  - `loan`：搜索到的储蓄账户的列表
  - `pagination`：分页的一些信息

添加贷款

- 输入：路径为`/loan/create`，输入上列编辑表单的基本信息
- 输出：根据创建的支票账户是否符合标准来返回成功或者失败的信息。
  - 成功：“已添加贷款”+ `loan.id`
- 错误情况处理
  - 账户信息已存在

编辑贷款

- 输入：路径为`/loan_edit/<string:loan_id>`，输入上列编辑表单的基本信息
- 输出： 根据贷款的的符合标准与否返回相应的信息：
  - 成功：“该贷款信息已更新‘ + `loan.id`
- 错误情况处理
  - 贷款编号不存在

删除贷款

- 输入：贷款的ID
- 输出：根据支票账户信息的符合标准与否返回相应的信息：
  - 成功："删除成功"
  - 失败： 提示信息并刷新表单
- 错误情况处理
  - 贷款正在发放中

发放贷款

发放记录查询

- 输入：贷款的id
- 输出：
  - `loan_logs`：该贷款的记录
  - `loan_id`：该贷款的id
  - `pagination`：分页的一些信息

发放贷款

- 输入：路径为`/loan_log/create`，输入上列编辑表单的基本信息
- 输出：根据创建的支票账户是否符合标准来返回成功或者失败的信息。
  - 成功："已添加贷款"+ `loan.id`
  - 失败：提示信息并刷新表单
- 错误情况处理
  - 贷款已发放完毕

发放记录编辑

- 输入：路径为`/loan_log_edit/<string:loan_log_id>`，输入上列编辑表单的基本信息
- 输出：根据贷款的的符合标准与否返回相应的信息：
  - 成功："该贷款信息已更新' + `loan.id`
  - 失败：提示信息并刷新表单
- 错误情况处理
  - 贷款号已存在

3.1.7 统计模块

这个模块对于后端比较简单，就是按业务分类(储蓄、贷款)和时间(月、季、年)统计各个支行的业务总金额和用户数。

我们采取的统计逻辑是将新建账户的余额以及编辑账户前后的差值作为存款、取款业务金额进行统计。

输入输出

- 输入： 路径 `/census/<string:id>`
  - 若 `id == 0`，则表示查看整体信息，输出：
    - `bs` ：支行名列表
    - `userNums` ： 用户数
    - `totalMoneyIn1` ： 存款业务金额
    - `totalMoneyIn2` ： 取款业务金额
    - `totalMoneyOut1` ： 放贷业务金额
    - `totaoMoneyOut2` ： 还款业务金额
    - `year` ： 年份

3.2 前端

前段只要采用`flaks-boostrap`，因为重复性过高，这里只列举一些代表性的模块：

3.2.1 展示模块

这个模块一般是展示查询结果，我们这里拿支行作为代表：

```
<table class="table table-hover">
    <tr>
        <th>支行名</th>
        <th>城市</th>
        <th>资产</th>
        <th>操作</th>
    </tr>
    {% for branch in branches %}
    <tr>
        <td>{{ branch.name }}</td>
        <td>{{ branch.city }}</td>
        <td>{{ branch.asset }}</td>
        <td>
                <a class="btn btn-success btn-sm" type="buttoon" href="{{
url_for('main.branch_edit', branch_name=branch.name) }}">编辑</a>
                <a class="btn btn-danger btn-sm" type="button" href="{{
url_for('main.branch_delete', id=branch.name) }}">删除</a>
        </td>
    </tr>
    {% endfor %}
</table>
```

主要是列出相应的属性名称，以及相应的操作按钮，然后加上一个循环`{% for branch in branches %}`进行渲染。

3.2.2 整体模块

这个模块包含了上述的模块，我们这里仍拿支行模块作为代表

```
{% extends "base.html" %}
{% import "bootstrap/wtf.html" as wtf %}
{% import "_macros.html" as macros %}

{% block title %}支行管理{% endblock %}

{% block page_content %}
<div class="page-header">
    {{ wtf.quick_form(form, extra_classes='form-inline') }}
</div>
{% include '_branch.html' %}

{% if pagination %}
<div class="pagination">
    {{ macros.pagination_widget(pagination, '.branch') }}
</div>
{% endif %}
{% endblock %}
```

- `base.html`是基础模块，即首页
- `wtf.quick_form()`是flask中的一个插件，用于构造表单，十分方便，我们只需要将相应的类写好就可以简单的调用
- `macros.pagination_widget()`用于分页。

4. 实现与测试

4.1 首页

> 欢迎使用银行业务管理系统

4.2支行管理

添加支行

已存在的支行名不能重复添加

该支行名已经存在

编辑支行信息

支行名

南京第一支行

所在城市

南京

资产

1000000

提交

由于我们添加了约束，资产不能为负数

编辑支行信息

支行名

上海第一支行

所在城市

上海

资产

-10

资产必须大于0

提交

由于我们添加了约束，资产不能为负数

修改后可以成功添加

已添加新支行

编辑支行信息

支行名

上海第一支行

所在城市

上海

资产

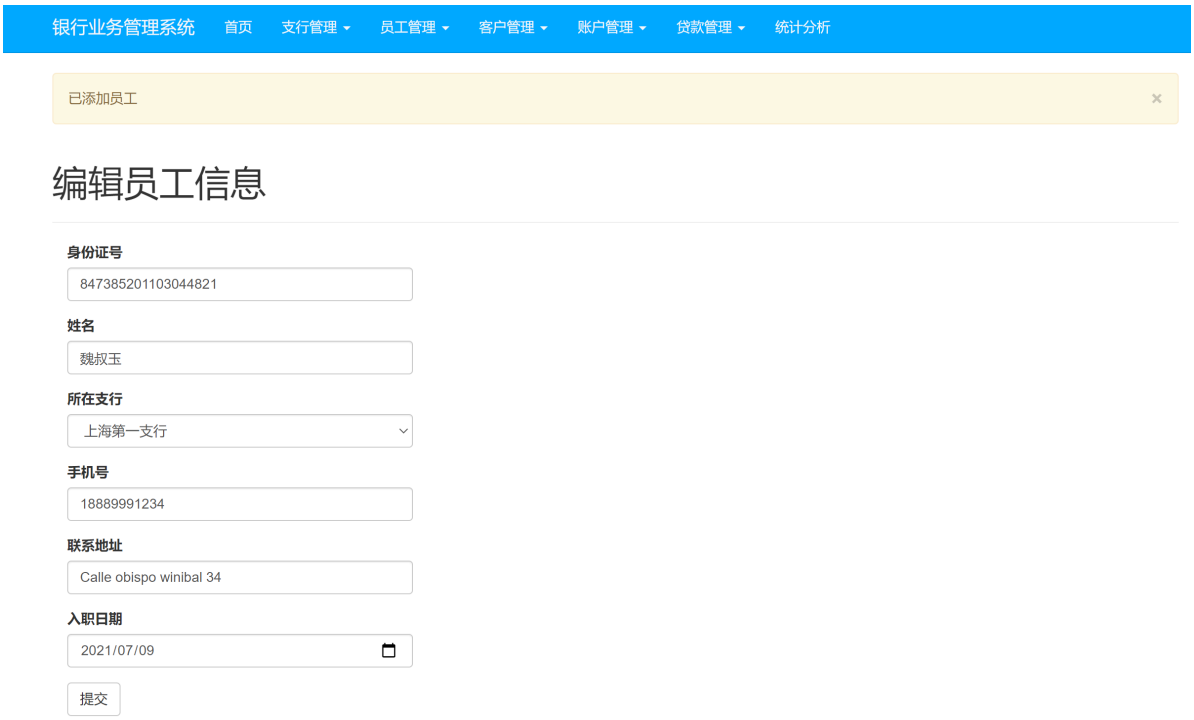
100000

提交

查询支行

4.3员工管理

添加员工



查询员工

和其他搜索一样，不指定条件时会输出所有员工

姓名

所在支行

手机号

联系地址

搜索

身份证号	姓名	所在支行	手机号	联系地址	入职日期	操作
019013	李四	合肥第二支行	222333	槽郢路	2018-09-12	编辑 删除
117027	赵六	合肥第一支行	777766	滨湖路	2021-06-04	编辑 删除
138942	张三	合肥第一支行	111222	黄山路	2020-03-06	编辑 删除
216458	王五	合肥第三支行	999222	官亭路	2019-08-19	编辑 删除
812732	无名	南京第一支行	123456	合肥路	2021-06-24	编辑 删除
847385201103044821	魏叔玉	上海第一支行	18889991234	Calle obispo winibal 34	2021-07-09	编辑 删除

« 1 »

可以指定搜索条件

所在支行

合肥第一支行

手机号

联系地址

搜索

身份证号	姓名	所在支行	手机号	联系地址	入职日期	操作
117027	赵六	合肥第一支行	777766	滨湖路	2021-06-04	编辑 删除
138942	张三	合肥第一支行	111222	黄山路	2020-03-06	编辑 删除

删除员工

在删除的时候我们添加了一些限制，例如如果一个员工仍有没办理完的贷款，则不能删除

银行业务管理系统 首页 支行管理 员工管理 客户管理 账户管理 贷款管理 统计分析

该员工仍管理贷款，无法删除 ×

身份证号

4.3客户模块

添加客户

如果身份证号不符合约束，则会报错

编辑客户信息

身份证号

12345678

身份证号不合法

正确添加后，则会显示添加成功

此时在界面进行查询，新添加的用户已经可以显示

身份证号	姓名	手机号	联系地址	联系人姓名	联系人手机号	联系人邮箱	联系人与客户关系	操作
123456197701010101	唐朔飞	11133334444	奥体中心	姚永雷	333444		同事	编辑 删除

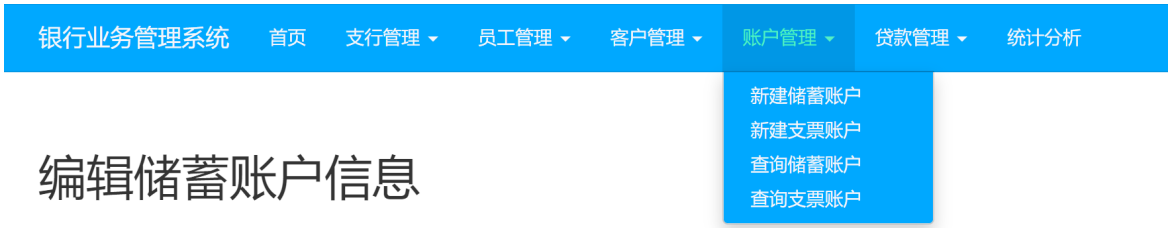
同时我们也可以点击编辑对用户信息进行编辑

查询客户

在查询时我们支持模糊查询，示例如下

4.1.4 账户管理

可以分别对储蓄账户和支票账户进行新建和查询

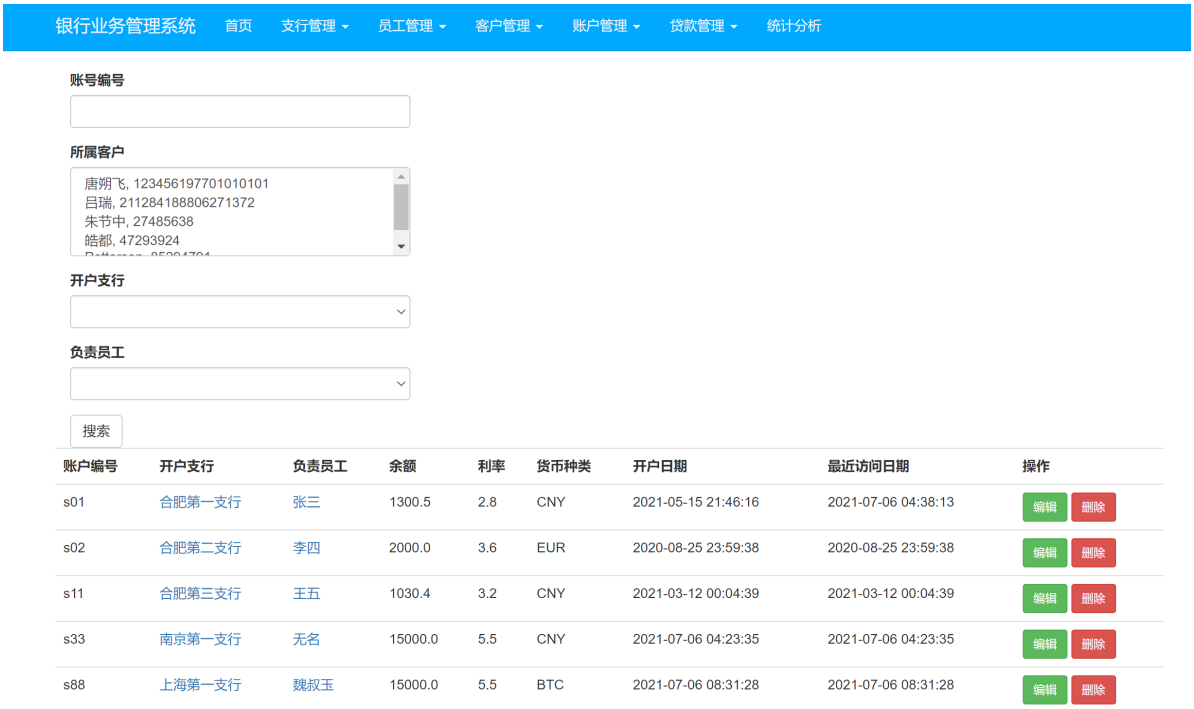


开户

开户的示例如下

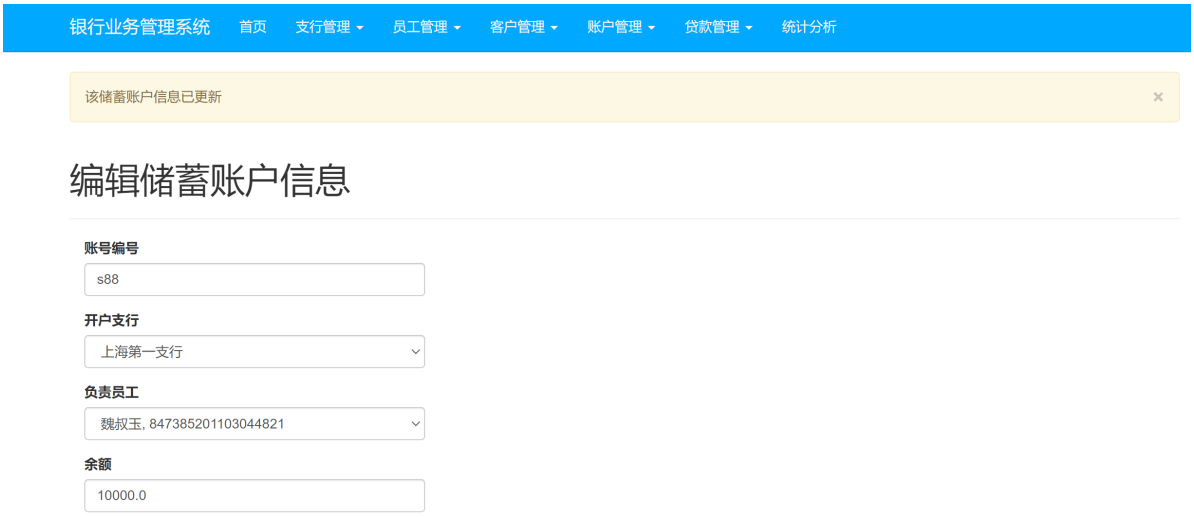
查询

查询界面如下



不输入信息的时候默认输出所有账户

我们可以点击编辑对账户信息进行修改，例如，这里我们修改账户的余额



提示显示账户信息已更新，

然后在查询界面我们看到账户的最近访问时间已经被修改

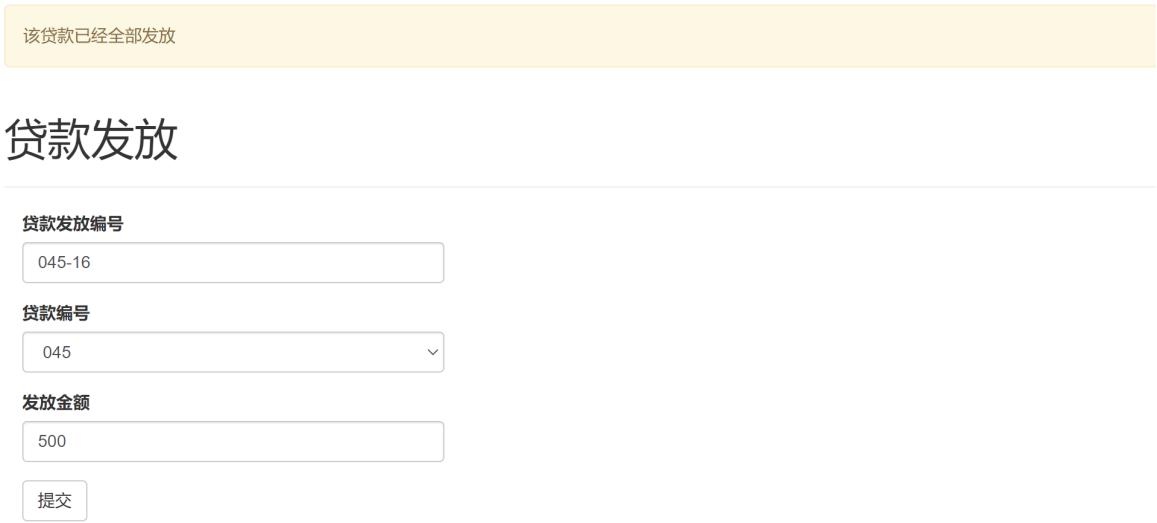
4.5 贷款模块

新建贷款

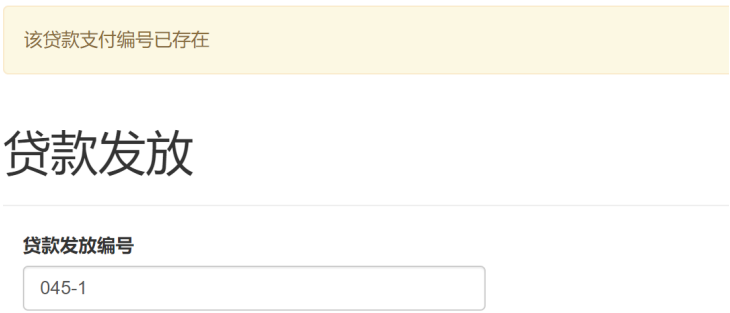
显示添加贷款成功，并且下面表单中客户唐朔飞名下多了一笔贷款

我们还可以对其发放贷款

如果对一个已经发完了的贷款进行发放会报错



如果贷款号已经存在会报错



修改为077-1后发放成功，界面自动刷新可以看到贷款状况已经修改为正在发放中

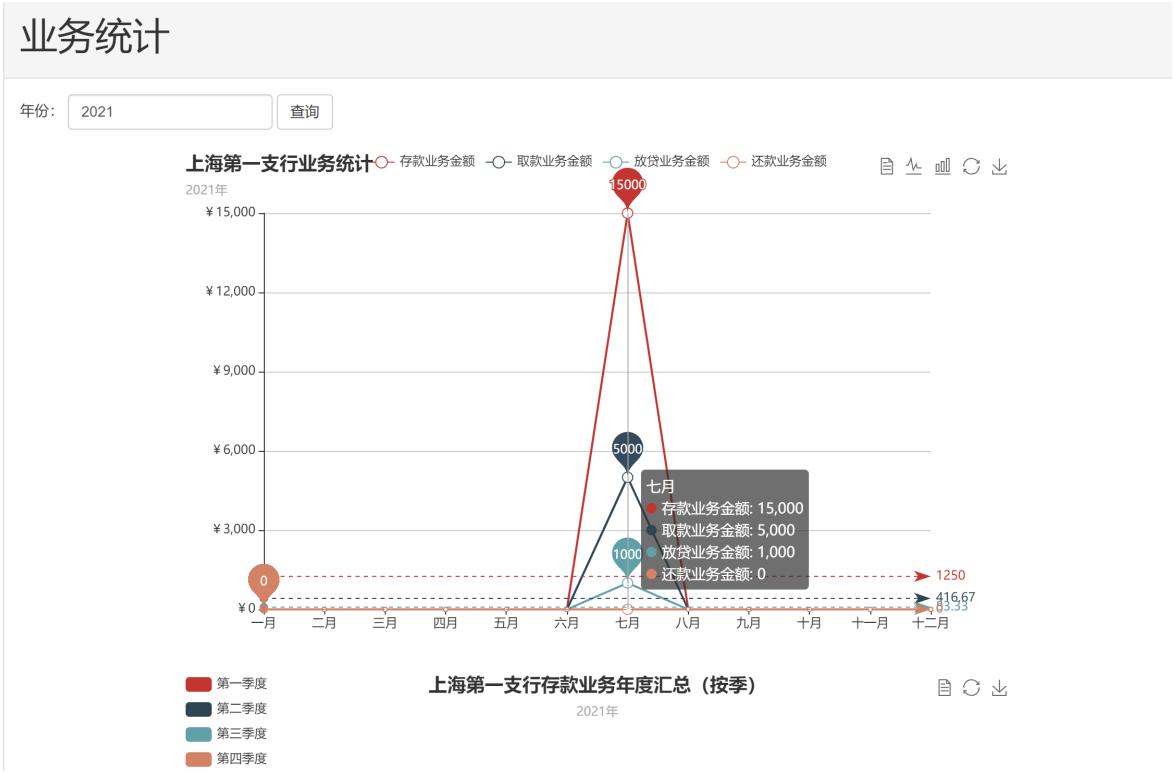
可以点击发放记录进行查询

4.6 统计模块

统计模块界面如下，

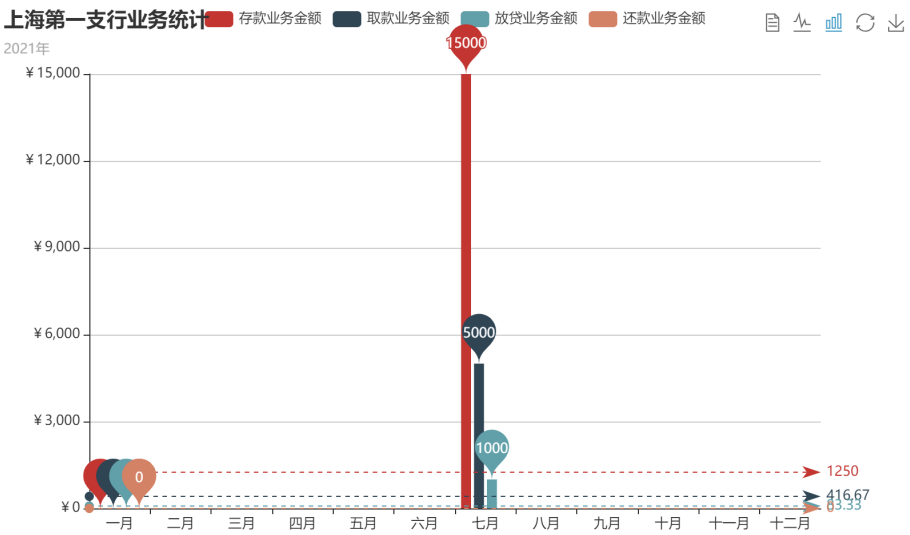
这里的存款来自新建账户以及编辑账户时余额的增加值，取款则来自于余额的减少

可以点击统计详情进行查看，可以将鼠标光标放在数据上查看详情，示例如下



还有其他数据呈现形式，如柱状图

年份： 2021 查询



以及数据视图

业务统计

年份： 2021 查询

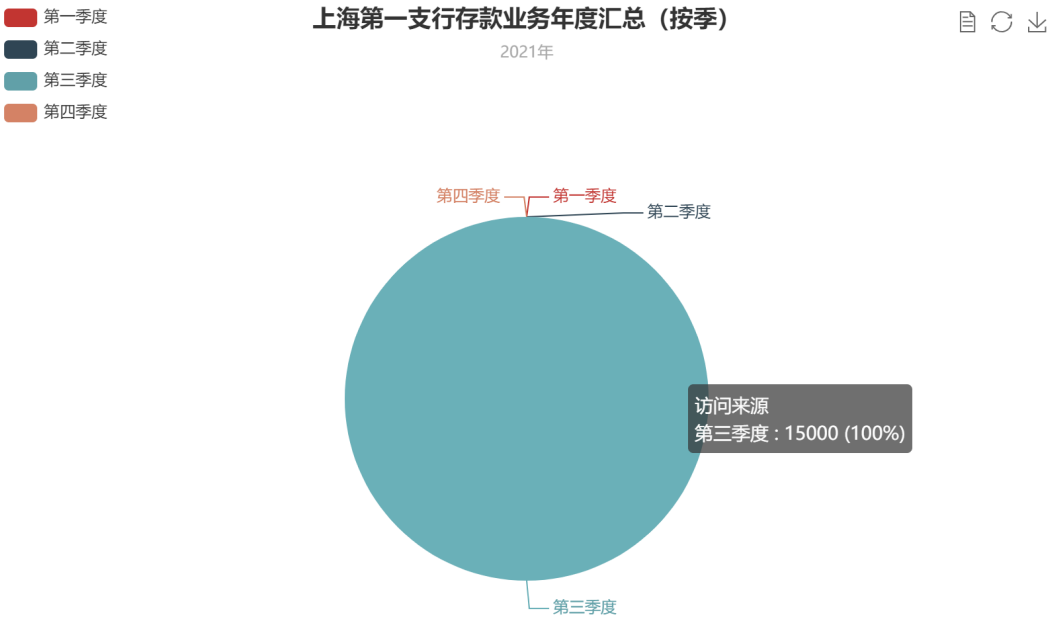
数据视图

	存款业务金额	取款业务金额	放贷业务金额	还款业务金额
一月	0	0	0	0
二月	0	0	0	0
三月	0	0	0	0
四月	0	0	0	0
五月	0	0	0	0
六月	0	0	0	0
七月	15000	5000	1000	0
八月	0	0	0	0
九月	0	0	0	0
十月	0	0	0	0
十一月	0	0	0	0
十二月	0	0	0	0

关闭 刷新

还有扇形图进行统计，不过由于我们并没有插入什么前几个月的数据，因此没什么能看的

分别对单项业务按季度统计，这里以存款为例



5 总结与讨论

5.1总结

- 学习了很多东西，从前端到后端全栈式开发，
- 了解了数据库的应用技术
- 提高了软件开发能力

5.2 建议

- 感觉数据库设计完成后，数据库方面的任务就完成了，剩下的与课程所学内容差的很多，像是软工