记账APP——流水

智能终端课程设计文档

3150104650 王　淼

3150102098 沈　锴

3150103695 吴亦全

目录

[1. 背景与功能简介 3](#_Toc501570185)

[1.1 背景 3](#_Toc501570186)

[1.2 功能 3](#_Toc501570187)

[2. 项目架构 4](#_Toc501570188)

[2.1 工程文件组织 4](#_Toc501570189)

[2.2 Java文件组织 6](#_Toc501570190)

[2.3 Res文件结构 8](#_Toc501570191)

[3. 项目关键技术及实现 10](#_Toc501570192)

[3.1 网络连接 10](#_Toc501570193)

[3.2 登录模块 11](#_Toc501570194)

[3.2.1 注册 11](#_Toc501570195)

[3.2.2 找回密码的实现 16](#_Toc501570196)

[3.2.3 登录的实现 19](#_Toc501570197)

[3.2.4 人脸识别登录的实现 22](#_Toc501570198)

[3.3 导出账单 33](#_Toc501570199)

[3.4 统计图表的生成 35](#_Toc501570200)

[3.4.1 选择数据 36](#_Toc501570201)

[3.4.2 数据转换 40](#_Toc501570202)

[3.4.3 绘制 41](#_Toc501570203)

[3.5 同步支付宝、银行卡 43](#_Toc501570204)

[3.5.1 银行卡同步实现 43](#_Toc501570205)

[3.5.2 支付宝同步实现 45](#_Toc501570206)

[3.6 用户头像的获取与更新 47](#_Toc501570207)

[3.6.1 用户图片接口 48](#_Toc501570208)

[3.6.2 Download实现 49](#_Toc501570209)

[3.7 数据库的逻辑与实现 52](#_Toc501570210)

[3.7.1 表的建立 52](#_Toc501570211)

[3.7.2 数据库基础逻辑 55](#_Toc501570212)

[3.7.3 数据库同步 68](#_Toc501570213)

[3.8 UI部分 75](#_Toc501570214)

[3.8.1 实现 75](#_Toc501570215)

[3.8.2 展示 83](#_Toc501570216)

[4. 项目分工 85](#_Toc501570217)

[5. 总结和展望 86](#_Toc501570218)

[5.1 总结： 86](#_Toc501570219)

[5.2 展望： 86](#_Toc501570220)

# 背景与功能简介

## 背景

随着支付手段及购物场景的日趋多样化，人们对于自己的开支消费未能有着很好的整体把握，而单纯的采用记事本记账，条目繁多而杂乱。不得不说，当下人们对于收支记账有着较大的需求，专业的记账软件，应该能通过良好的归类和便捷的记账功能，让用户方便的记下自己的每一笔开支，并且通过图表形式，清晰的了解自己的收支情况和变化趋势，若能做到自动同步用户的电子支付账户（诸如支付宝、银行卡等），用户即可拥有一款软件、掌握整个财务状况。但目前市场上的诸多记账软件都各有不足，有些分类不清晰，有些繁琐复杂，都没有完全满足用户的心理需求。

## 功能

基于这个背景，我们意欲开发一款条目清晰、可自动同步电子支付账户、支持导出账单、方便快捷且安全的Android记账APP，来满足用户的需求。

根据这个需求，我们将这一APP的功能细分为以下六类：

* + 1. 用户记账

用户可以以添加账单的形式记录日常消费与借贷情况，记账时需选择消费/收入类型与金额，并可对账单进行备注。APP会自动获取添加账单的时间作为这一笔账的时间戳。同时，用户可以在APP主界面对每月的账务进行浏览，单击可查看详细信息，也可以修改或删除。

* + 1. 报表生成

APP会按照年/月/日的形式自动生成用户的消费/收入报表，以折线图与饼状图的形式展现，使用户更直观清晰地掌握阶段时间的财务状况。

* + 1. 账单导出

为了使用户对自己的账务有一个更清晰的认识，我们提供了账单导出至Excel格式的功能，用户只需点击“账单导出”，即可在本地得到一份账单表格。

* + 1. 数据备份

为保证数据的安全性，APP会在服务端对用户的账单进行备份，以免数据丢失。

* + 1. 自动同步

APP实现了建设银行、农业银行的收入、支出账单自动同步，支付宝的支出自动同步，尽可能地减少了用户的记账负担。同时用户可以在APP中查看每个账户的余额统计信息。

* + 1. 人脸识别

进入APP时用户可以选用密码登录或人脸识别登录，保障了用户的财务信息安全，也使用户可以放心地使用该产品。

# 项目架构

整个项目可以分为三大模块，前端，后端与网络，以及这三个模块之间的通信。如下图所示：

界面更新

用户操作

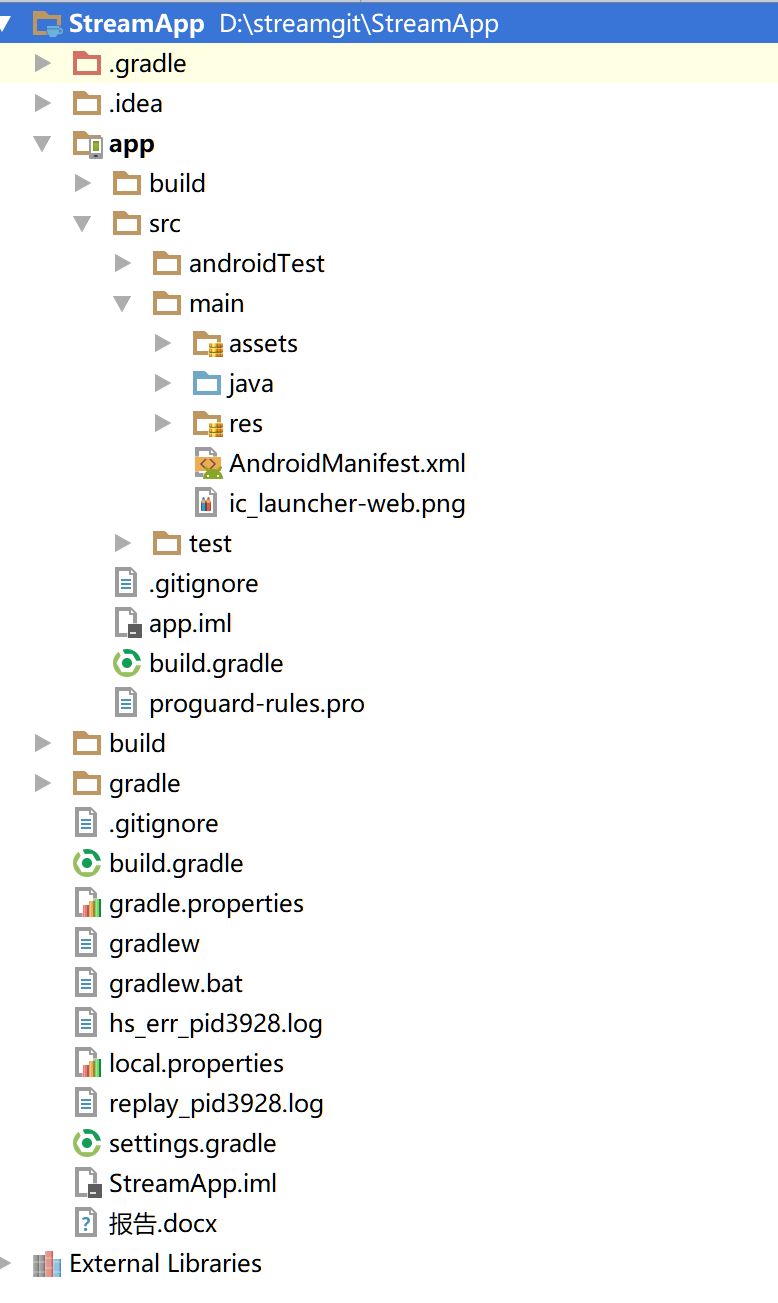
W网络

后端

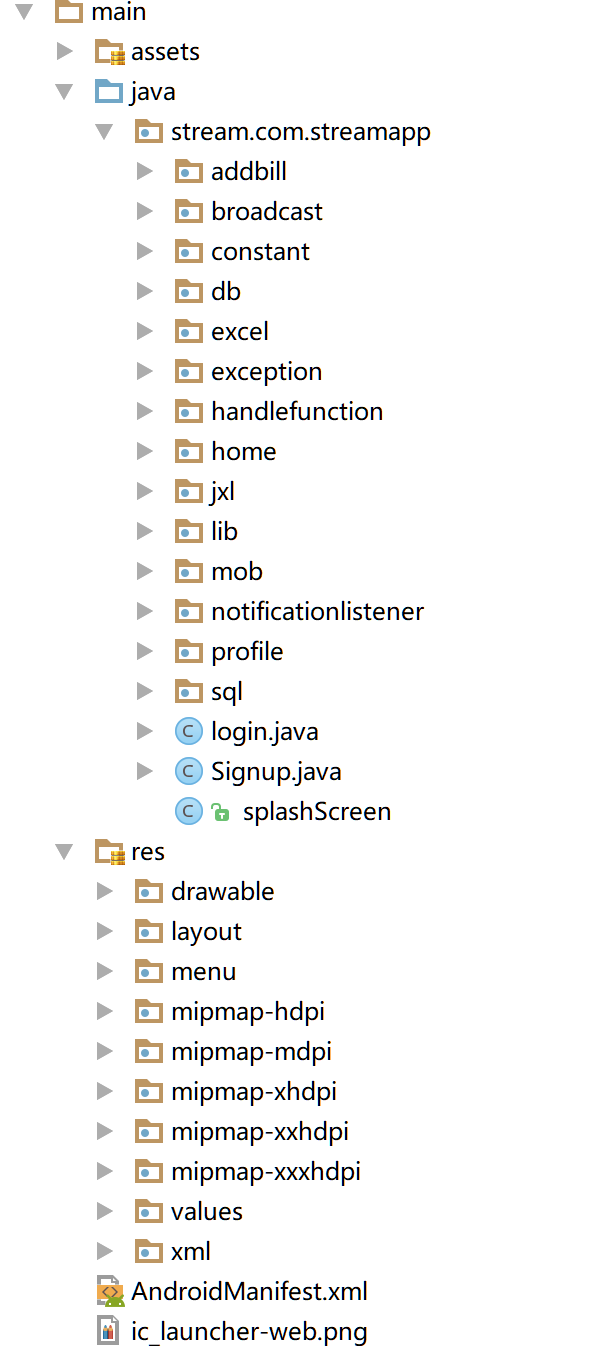
前端

## 工程文件组织

整个工程的文件结构如下图：



主要的代码在app目录下的src/main中，其他文件都是一些配置文件。需要注意的是两个build.gradle，在app文件内的是需要我们修改的，里面声明了一些编译时要用到的外部库。我们主要看main文件夹：

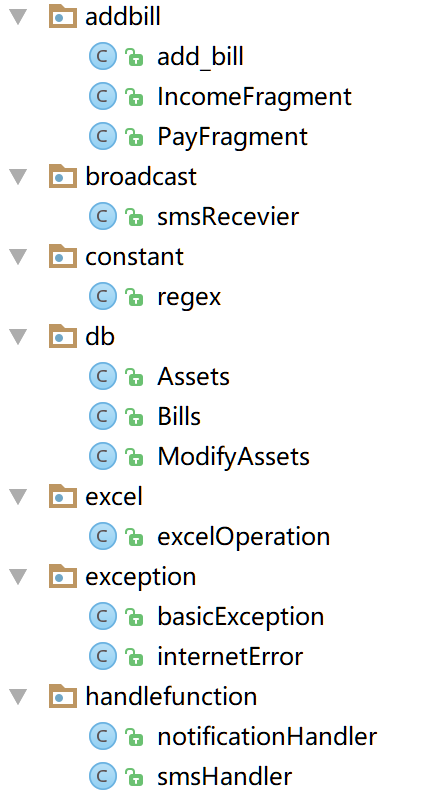


Assets目录下是本地数据库要用到的配置文件，java文件里面则是java程序，res文件夹里面是UI界面需要用到的资源，包括页面布局、文字、图片等。AndriodManifest.xml中完成对各个activity的注册以及app所需权限的定义。

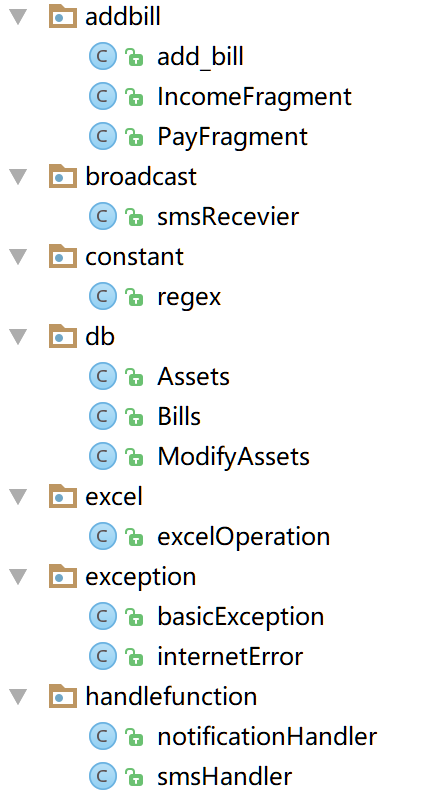
## Java文件组织

下面对java目录展开介绍。

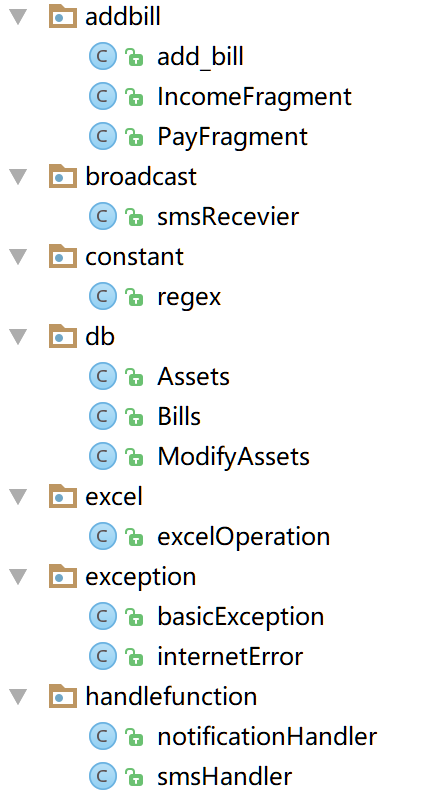
Addbill目录里面的程序是添加账单时用到的，一旦有新帐单添加，会调用add\_bill（是一个activity），这个activity会托管两个Fragment（IncomeFragment和PayFragment），随后根据用户操作调用不同的类完成对本地数据库的添加。



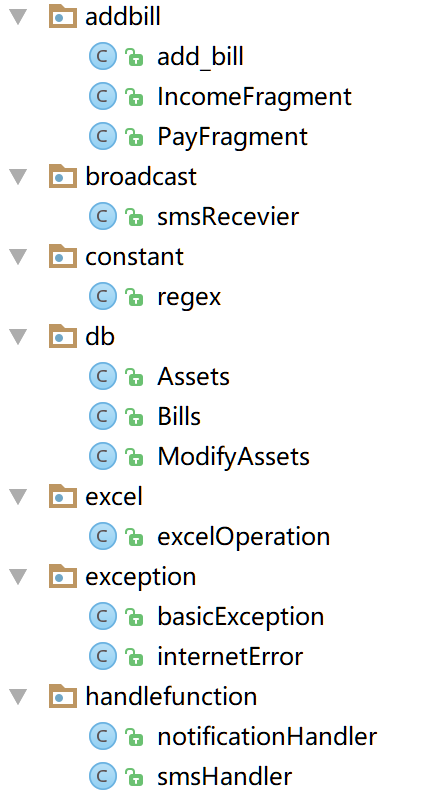
Broadcast目录负责处理读取到的短信。



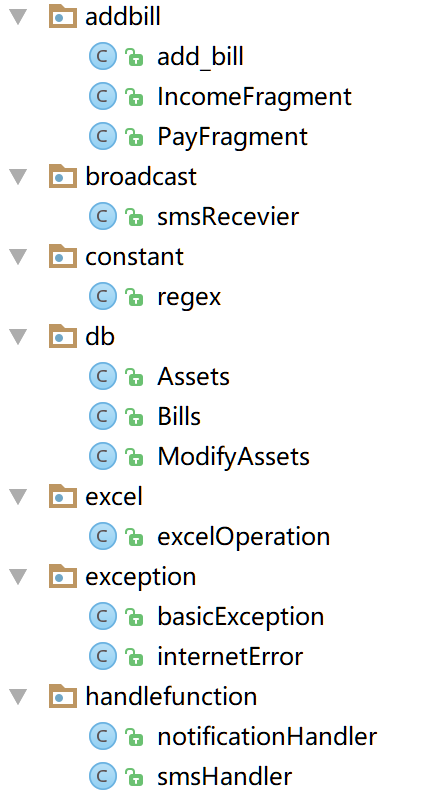
Constant目录下的regex类定义了我们会用到的正则表达式。



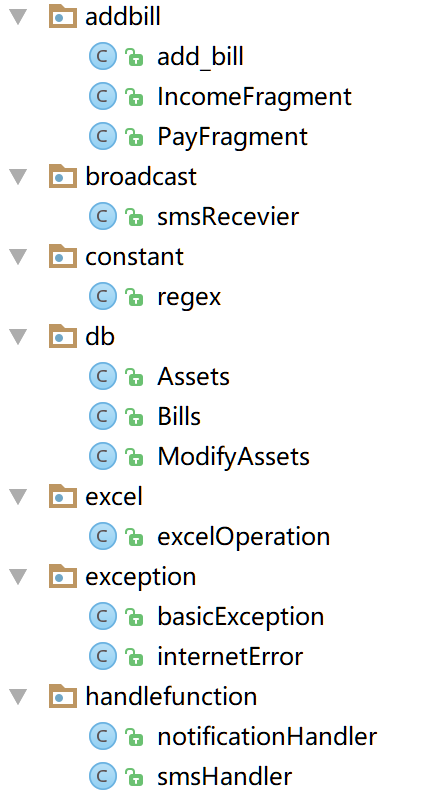
Db目录下的各个类定义了数据库所要用到的表，包括表中的字段以及接口函数。



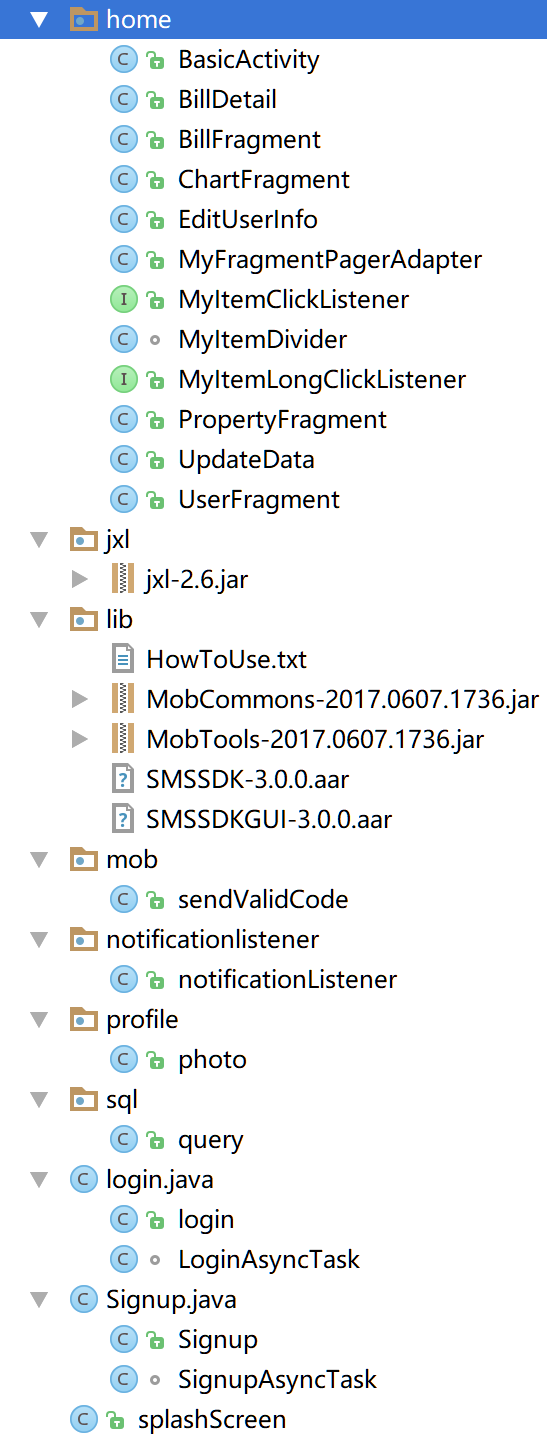
Excel目录负责账单导出生成excel。



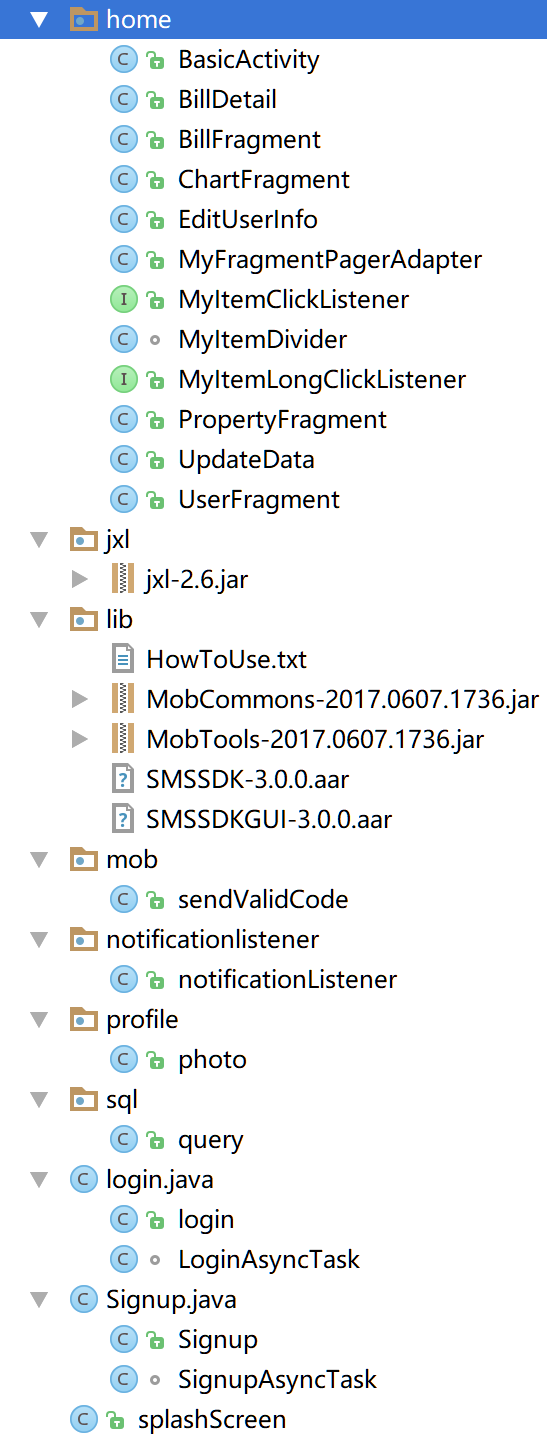
Exception和handlefunction目录下是一些处理异常或者信号的类的定义与声明，并没有实际的逻辑代码。



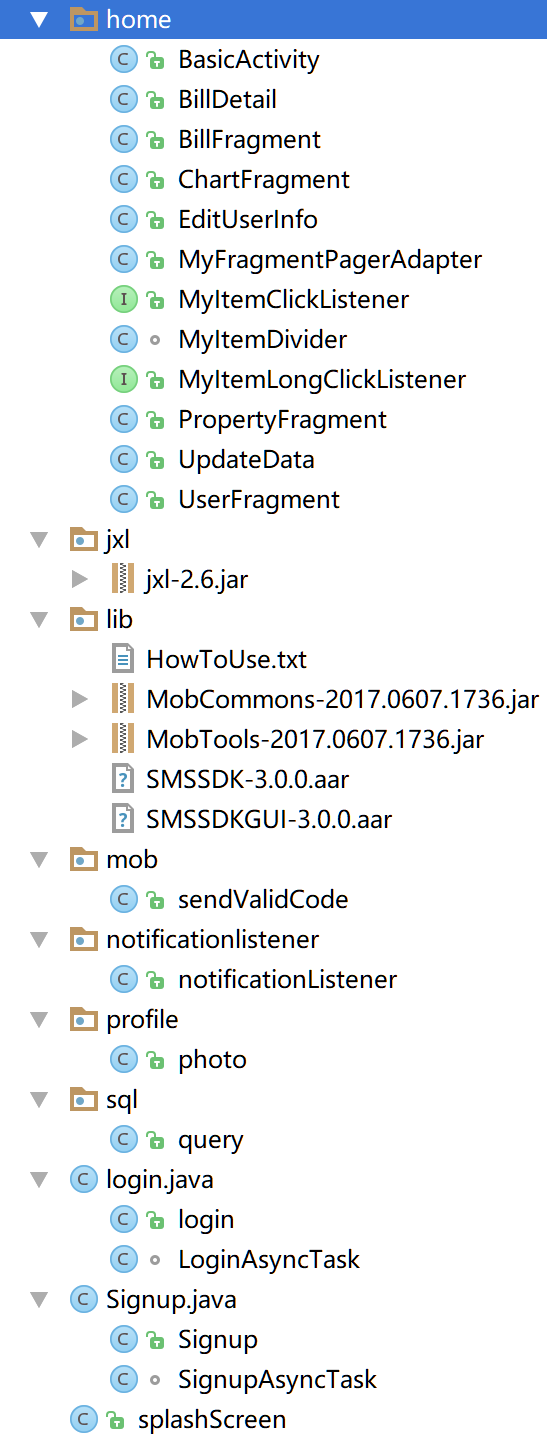
Home目录是本工程的核心部分，BasicActivity是进入界面的绘制以及监听的设置，同时托管了四个UI Fragment。BillDetail是一个activity，负责账单详情的界面绘制与数据库操作。BillFragment负责Bill列表的界面绘制与数据库操作。ChartFragment负责图表界面的绘制与数据库操作。EditUserInfo负责用户信息编辑的界面绘制与相关操作。MyFragmentPagerAdapter是使用ViewPager实现锁必需的adapter实现。MyItemClickListener和MtItemLongClickListener两个类是RecyclerView中item点击或长按监听接口。MyItemDivider类负责账单页与资产页分割线的实现，PropertyFragment类负责绘制用户资产的界面与数据库操作。UpdateData负责与远程数据库同步。UserFragment负责用户界面的绘制与相关事件监听。



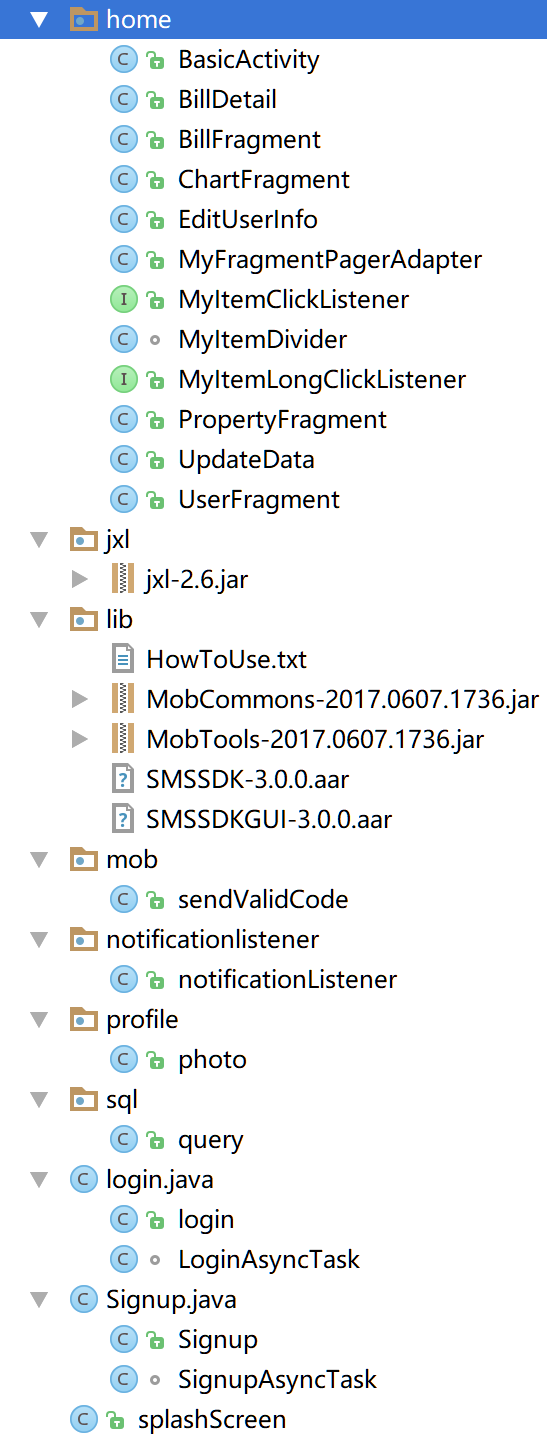
Jxl与lib目录下都是一些配置与说明文件



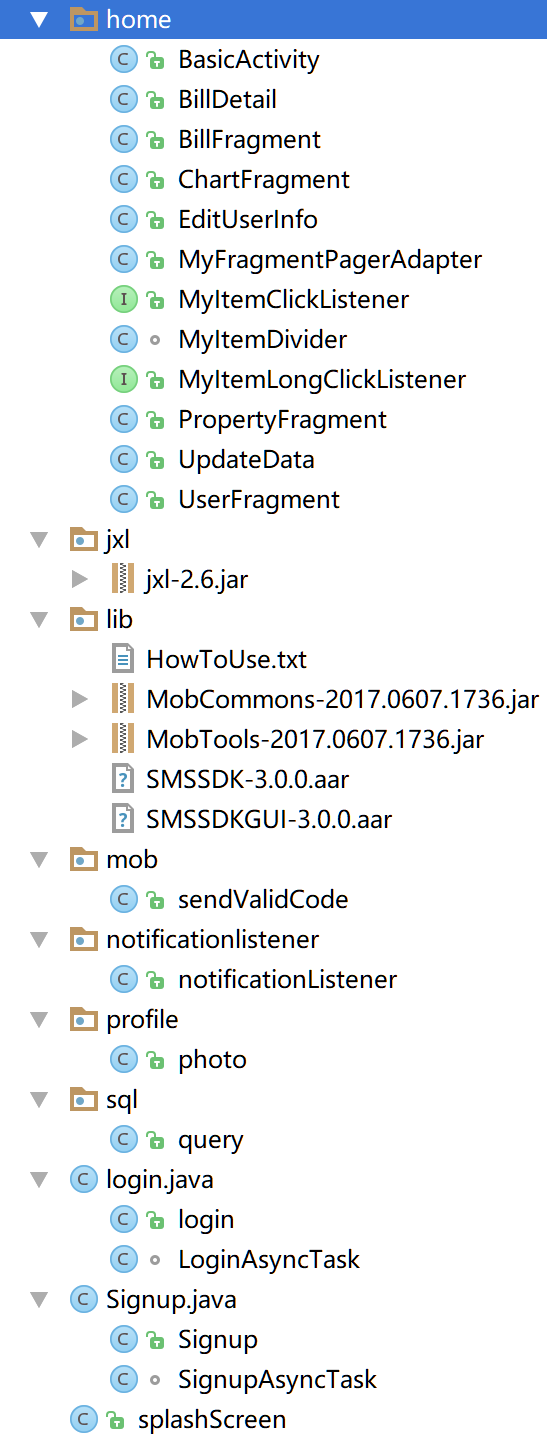
Mob目录是我们使用的发送短信的API，负责发送验证码。



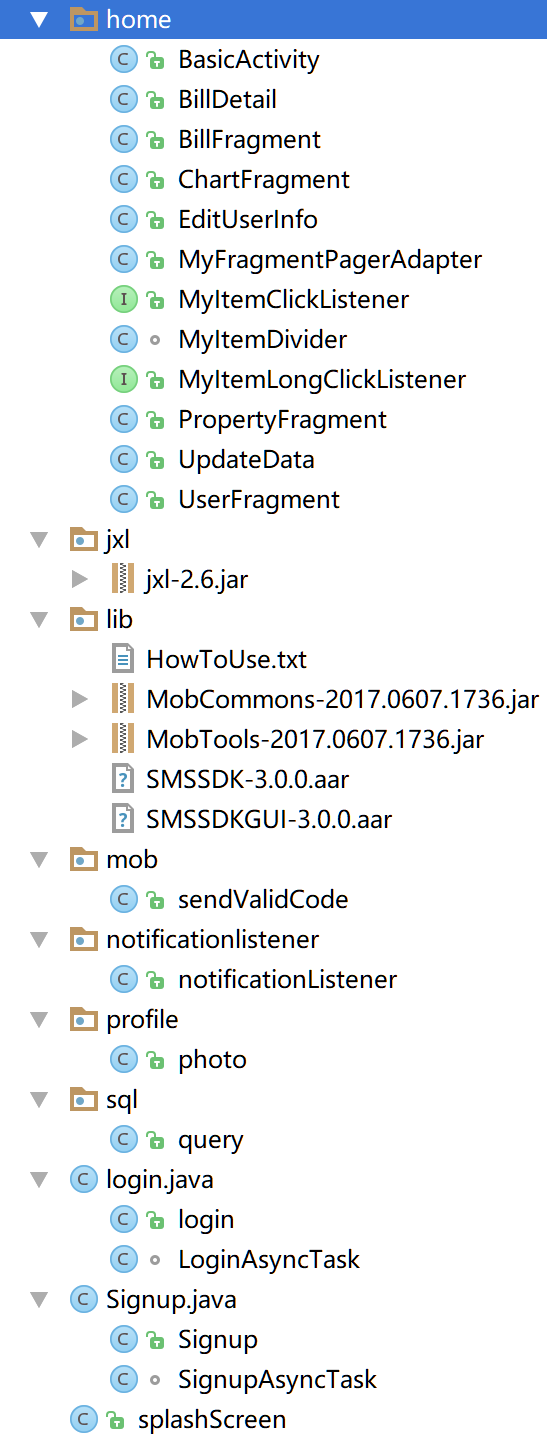
Notificationlistener目录负责监听通知栏的通知，以便完成支付宝同步。



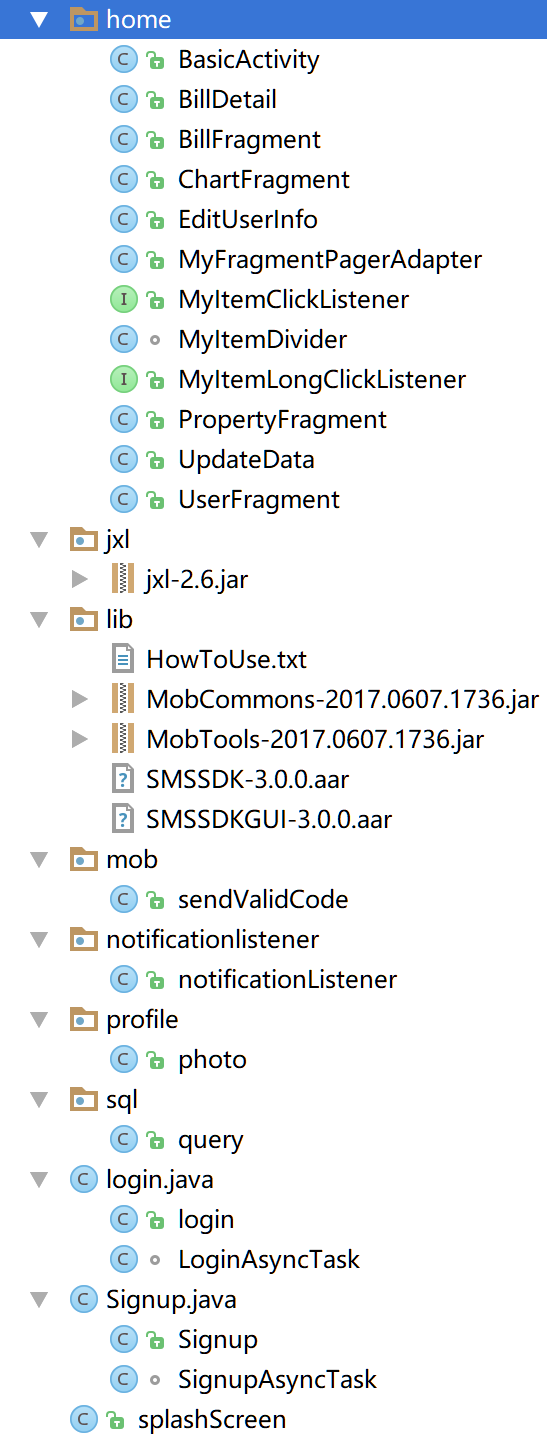
Profile目录负责处理用户头像。



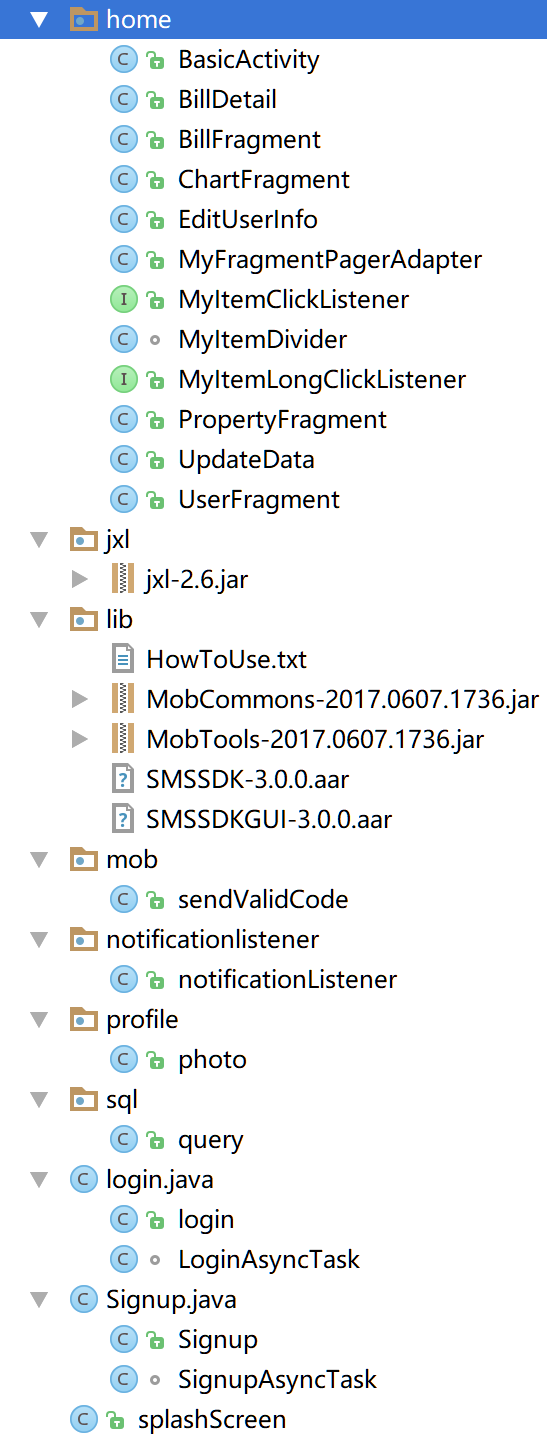
Sql目录负责封装远程数据库的query语句。



Login.java和Signup.java负责用户登录与注册界面的监听以及远程连接实现。

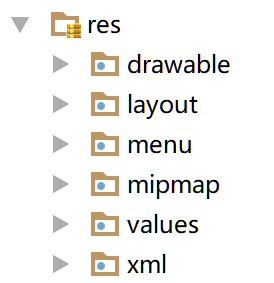


SplashScreen类负责欢迎界面的绘制。

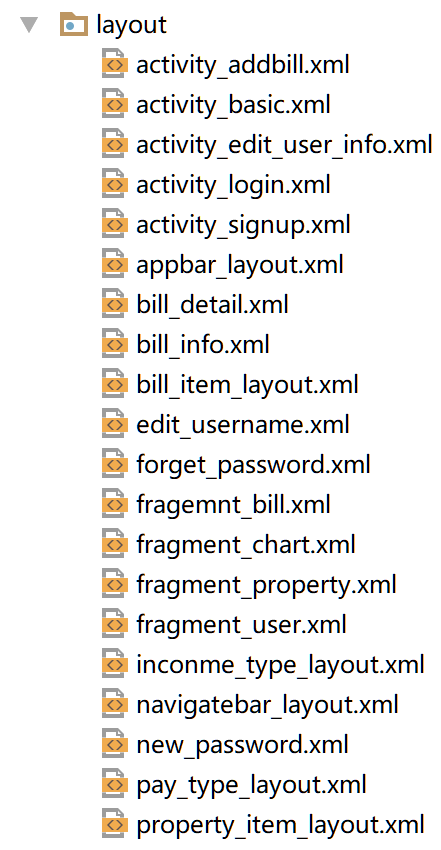


## Res文件结构

Res目录下都是工程中用到的资源文件，下设六个子目录。drawable目录中存储了图片及一些形状，layout中存储了布局文件，menu中存储了用到的菜单，mipmap中只有app的icon，values中包含对配色、字符串、UI主题的定义，xml中定义了文件存储地址。



下面对layout目录进行展开介绍。



我们从每个文件的名字即可猜测出该文件的大致用途。activity\_addbill.xml是添加账单界面的布局，activity\_basic.xml是主界面的布局，activity\_edit\_user\_info.xml负责编辑个人资料时的布局，activity\_login.xml是登陆界面的布局文件，activity\_signup.xml负责注册页面的布局，appbar\_layout.xml是顶部标题栏的布局，bill\_detail.xml是账单详情页的布局，bill\_info.xml是添加账单是的选择界面布局，bill\_item\_layout.xml和property\_item\_layout.xml分别是在账单页面和资产页吗中每个条目的布局，交由RecylclerView托管，edit\_username.xml是修改用户名的对话框布局，forget\_password.xml和new\_password是忘记密码重置密码的对话框布局（这两个布局实际没有用到，实际用到的是发送验证码的api提供的布局），fragment\_bill.xml、fragment\_chart.xml、fragment\_property.xml、fragment\_user.xml分别是账单页、图表页、资产页、用户页fragment的布局，income\_type\_layout.xml和pay\_type\_layout.xml分别是添加账单时收入与支出的种类布局，事实上是两个GridView，navigatebar\_layout.xml是底部导航栏的布局。

# 项目关键技术及实现

## 网络连接

Stream APP使用网络服务器同步本地数据库数据，因此需要通过网络访问传输必要的数据。

网络连接需要用到的工具有HttpClient、HttpUrlConnection、Okhttp等，HttpClient已经被废弃，相比之下，Okhttp封装的更好，对于资源利用率也更高，因此我们采用Okhttp。

这一部分离散分布在工程内部，因此只能给出大概的框架。

首先，创建一个client。

private final OkHttpClient client = *new*

OkHttpClient.Builder().connectTimeout(60, TimeUnit.SECONDS).build();

然后，建立request，添加需要post的数据。

RequestBody requestBody = *new* MultipartBody.Builder()

.setType(MultipartBody.FORM)

.addFormDataPart("file", "1", fileBody)

.addFormDataPart("id", ""+getUser\_id())

.addFormDataPart("isInsert", "0")

.build();

Request request = *new* Request.Builder()

.url("http://10.214.129.171:9999/upload")

.post(requestBody)

.build();

建立连接，得到response，解析response内容。

Response response;

*try* {

response = client.newCall(request).execute();

String jsonString = response.body().string();

} *catch* (IOException e) {

Log.d("ff","upload IOException ",e);

}

网络访问服务是在子线程中完成的，子线程是以回调函数的方式与主线程交换数据，由于因实际情况而异，因此用到的时候会详细说明。

## 登录模块

如果用户是新用户，那么可以通过注册的方式得到自己的账号。

用户登录APP可以选择两种方式，一是最传统的使用用户名和密码登录，第二种是使用人脸识别登录。

如果用户忘记了自己的密码，可以通过手机号找回。

### 注册

1. 注册流程

注册需要用户输入用户名、密码、手机号，然后判断输入合法性。如果合法，就插入网络端数据库。最后判断插入是否成功，给用户反馈。

1. 注册页面设计

采用LinearLayout布局，背景为色彩渐变，效果如下：



1. 服务器配置

我们使用的服务器是阿里云的学生版，配置为：CPU：1核，内存：2G，网络带宽：1Mbps。系统环境为：Ubuntu 16.04 32位。数据库为mysql。

1. 注册实现

首先根据输入，简单判断合法性。我们固定密码必须是8位以上，手机号必须为中国11位手机号。

手机号的判断基于规则，大陆手机号为11位，第一位一定是1，第二位是3、4、5、8的一位，然后九位数字，因此我们通过正则表达式判断。

public static final String phonePattern="^[1][3,4,5,8][0-9]{9}$";

public boolean isMobile(String str) {

Pattern p = null;

Matcher m = null;

boolean b = false;

p = Pattern.compile(regex.phonePattern); *// 验证手机号*

m = p.matcher(str);

b = m.matches();

*return* b;

}

如果能够通过简单的规则测试，那么进入下一步网络端的验证。

网络的访问我使用的是AsyncTask这个异步的类。其将子线程封装成三步，也就是执行前，执行中，执行后。

class SignupAsyncTask extends AsyncTask<Object,Object,Object>

{

String userName,passwd,phoneNum;

private final WeakReference<Activity> mActivity;

int result = 1;

private int errorType=-1;

SignupAsyncTask(Signup activity, String userName,String passwd,String phoneNum)

{

*this*.userName=userName;

*this*.passwd=passwd;

*this*.phoneNum=phoneNum;

mActivity=*new* WeakReference<Activity>(activity);

}

@Override

protected void onPreExecute()

{

}

@Override

protected Object doInBackground(Object... objects)

{

}

@Override

protected void onPostExecute(Object result)

{

}

}

首先判断用户名是否存在，如果不存在，就去判断手机号是否存在。经过这两步验证，就将用户的数据插入数据库。最后进行验证数据库中是否存在用户。

query是我们封装的访问网络数据库的类。

query tQuery = *new* query();

String sqlQuery1="insert into user values(null,\""+userName+"\",\""+passwd+"\",\""+phoneNum+"\");";

Log.e("fff","sqlins:"+sqlQuery1);

String sqlQuery2="select passwd from user where name = \"" + userName + "\";";

String sqlQuery3="select passwd from user where phone = \"" + phoneNum + "\";";

*try* {

String rePd=tQuery.select("user",sqlQuery2);

Pattern r = Pattern.compile(regex.passwdPattern);

Matcher m = r.matcher(rePd);

*if*(m.find())

{

String replaceBefore=m.group(0);

String res=m.group(0).replaceAll("(passwd|<br>)","");

*if*(!res.equals(""))

{

result = 0;

errorType=0;

*return* result;

}

}

String rephone=tQuery.select("user",sqlQuery3);

r = Pattern.compile(regex.passwdPattern);

m = r.matcher(rephone);

*if*(m.find())

{

String replaceBefore=m.group(0);

String res=m.group(0).replaceAll("(passwd|<br>)","");

*if*(!res.equals(""))

{

result = 3;

*return* result;

}

}

tQuery.select("user",sqlQuery1);

rePd=tQuery.select("user",sqlQuery2);

r = Pattern.compile(regex.passwdPattern);

m = r.matcher(rePd);

*if*(m.find())

{

String res=m.group(0).replaceAll("(passwd|<br>)","");

*if*(res.equals(passwd))

{

result = 1;

}

*else*{

result = 0;

errorType=1;

}

}

*else*{

result = 0;

errorType=1;

}

} *catch* (InterruptedException e1) {

e1.printStackTrace();

} *catch* (internetError e)

{

result=2;

}

然后根据result判断注册的情况。

Result为0为用户名已经存在，1表示注册成功，2表示网络连接错误，3表示手机号重复。

*if*((int)result==1)

{

Bundle re=*new* Bundle();

re.putString("Return","0");

Message msg=*new* Message();

msg.setData(re);

((Signup)mActivity.get()).mHandler.sendMessage(msg);

}

*else* *if*((int)result==0)*//error*

{

Bundle re=*new* Bundle();

re.putString("Return","1");

Message msg=*new* Message();

msg.setData(re);

((Signup)mActivity.get()).mHandler.sendMessage(msg);

}

*else* *if*((int)result==2)*//internet error*

{

Bundle re=*new* Bundle();

re.putString("Return","2");

Message msg=*new* Message();

msg.setData(re);

((Signup)mActivity.get()).mHandler.sendMessage(msg);

}

*else* *if*((int)result==3)*//phone repeat*

{

Bundle re=*new* Bundle();

re.putString("Return","4");

Message msg=*new* Message();

msg.setData(re);

((Signup)mActivity.get()).mHandler.sendMessage(msg);

}

我们和主线程交流的方式是通过回调函数。回调函数定义如下：

public Handler mHandler=*new* Handler()

{

@Override

public void handleMessage(Message msg)

{

*super*.handleMessage(msg);

Bundle bd=msg.getData();

String content=bd.getString("Return");

*if*(content.equals("0"))*//success*

{

progressView.setVisibility(View.GONE);

progressView.stopAnimation();

signup.setText(R.string.signupSuccess);

signup.setVisibility(View.VISIBLE);

Intent intent = *new* Intent(Signup.this,login.class);

startActivity(intent);

finish();

}

*else* *if*(content.equals("1"))*//username taken*

{

progressView.setVisibility(View.GONE);

progressView.stopAnimation();

signup.setVisibility(View.VISIBLE);

Toast.makeText(Signup.this, R.string.usernameTaken, Toast.LENGTH\_LONG).show();

}

*else* *if*(content.equals("2"))*//password error*

{

progressView.setVisibility(View.GONE);

progressView.stopAnimation();

signup.setVisibility(View.VISIBLE);

Toast.makeText(Signup.this, R.string.internetError, Toast.LENGTH\_LONG).show();

}

*else* *if*(content.equals("3"))*//*

{

progressView.setVisibility(View.GONE);

progressView.stopAnimation();

signup.setVisibility(View.VISIBLE);

Toast.makeText(Signup.this, R.string.internetError, Toast.LENGTH\_LONG).show();

}

*else* *if*(content.equals("4"))*//*

{

progressView.setVisibility(View.GONE);

progressView.stopAnimation();

signup.setVisibility(View.VISIBLE);

Toast.makeText(Signup.this, R.string.phoneRepeat, Toast.LENGTH\_LONG).show();

}

}

};

### 找回密码的实现

找回密码需要用户提供手机号，然后发送验证码给用户手机，如果能够验证通过，那么允许用户修改密码。

手机验证功能的实现。

这一部分牵涉到了服务商，因此我们这样的学生团体是做不了的，只能寻找可用的廉价API。经过仔细的查找和对比，我们确定了mob这一家的服务，原因是因为每天免费20条。Mob提供了API，我们只需重写回调函数部分即可。

首先下载sdk，配好环境。

compile name*:* 'SMSSDK-3.0.0', ext*:* 'aar'

compile name*:* 'SMSSDKGUI-3.0.0', ext*:* 'aar'

compile files('src/main/java/stream/com/streamapp/lib/MobCommons-2017.0607.1736.jar')

compile files('src/main/java/stream/com/streamapp/lib/MobTools-2017.0607.1736.jar')

然后调用库函数，并重写回调函数。

首先获得用户输入的新密码，然后判断密码的合法性，最后修改数据库，并做确认。

RegisterPage registerPage = *new* RegisterPage();

registerPage.setRegisterCallback(*new* EventHandler() {

public void afterEvent(int event, int result, Object data) {

*// 解析注册结果*

*if* (result == SMSSDK.RESULT\_COMPLETE) {

@SuppressWarnings("unchecked")

HashMap<String,Object> phoneMap = (HashMap<String, Object>) data;

String country = (String) phoneMap.get("country");

final String cphone = (String) phoneMap.get("phone");

*if*(event==SMSSDK.EVENT\_SUBMIT\_VERIFICATION\_CODE)

{

View newView = LayoutInflater.from(login.this).inflate(R.layout.new\_password,null);

final EditText newpassword = (EditText)newView.findViewById(R.id.newpassword);

final AlertDialog.Builder newbuilder = *new* AlertDialog.Builder(login.this);

newbuilder.setView(newView);

newbuilder.setNegativeButton("取消",null);

newbuilder.setPositiveButton("确定", *new* DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

String newPassword = newpassword.getText().toString().trim();

*if*(newPassword.length()<8)

{

Toast.makeText(login.this,R.string.passwordTooShort,Toast.LENGTH\_LONG).show();

newbuilder.setCancelable(false);

}

*else*{

String updateSql="update user set passwd = \""+newPassword+"\" where phone = \""+cphone+"\";";

query tQuery = *new* query();

*try* {

tQuery.select("user",updateSql);

String sqlQuery="select passwd from user where phone = \""+cphone+"\";";

String html=tQuery.select("user",sqlQuery);

Pattern r = Pattern.compile(regex.passwdPattern);

Matcher m = r.matcher(html);

*if*(m.find())

{

String res=m.group(0).replaceAll("(passwd|<br>)","");

*if*(!res.equals(newPassword))

{

Toast.makeText(login.this,R.string.internetError,Toast.LENGTH\_LONG).show();

newbuilder.setCancelable(false);

}

*else*{

Toast.makeText(login.this,R.string.changeSuccess,Toast.LENGTH\_LONG).show();

}

}

} *catch* (InterruptedException e) {

e.printStackTrace();

}*catch*( internetError e)

{

loginPromptET.setText(R.string.internetError);

}

}

}

});

newbuilder.create().show();

}

}

}

});

registerPage.show(login.this);

### 登录的实现

用户名和密码的登录方式是获得用户输入的用户名和密码之后，和服务器上的用户名和密码进行对比，如果完全一样就是正确的，允许其登陆，否则就拒绝用户的请求。

Mytask是一个继承了AsyncTask类的类，负责的是网络部分。

loginTXT.setVisibility(View.GONE);

progressView.startAnimation();

progressView.setVisibility(View.VISIBLE);

loginPromptET.setVisibility(View.GONE);

String username = usernameET.getText().toString().trim();

String password = passwordET.getText().toString().trim();

setUserName(username);

*if* (username.equals("")) {

loginPromptET.setText(R.string.usernameError);

loginPromptET.setVisibility(View.VISIBLE);

progressView.setVisibility(View.GONE);

progressView.stopAnimation();

loginTXT.setVisibility(View.VISIBLE);

*return*;

}

LoginAsyncTask myTask = *new* LoginAsyncTask(login.this, username, password);

myTask.execute();

Mytask中我们需要找到用户真实的密码，然后进行对比。最后以回调函数的方式进行传递消息。

query tQuery = *new* query();

String sqlQuery="select passwd from user where name = \"" + userName + "\";";

*try* {

String rePd=tQuery.select("user",sqlQuery);

Pattern r = Pattern.compile(regex.passwdPattern);

Matcher m = r.matcher(rePd);

*if*(m.find())

{

String res=m.group(0).replaceAll("(passwd|<br>)","");

*if*(res.equals(passwd))

{

sqlQuery="select id from user where name = \"" + userName + "\";";

rePd=tQuery.select("user",sqlQuery);

r=Pattern.compile(regex.idPattern);

m=r.matcher(rePd);

*if*(m.find())

{

res=m.group(0).replaceAll("(id|<br>)","");

user\_id=Integer.parseInt(res);

result=1;

}

*else*{

result=0;

}

}

*else*{

result = 0;

}

}

*else*{

result = 0;

}

} *catch* (InterruptedException e1) {

e1.printStackTrace();

}

*catch*(internetError e)

{

result=2;

}

然后根据result判断登陆是否成功。

*if*((int)result==1)

{

Bundle re=*new* Bundle();

re.putString("Return","0");

re.putInt("ID",user\_id);

Message msg=*new* Message();

msg.setData(re);

((login)mActivity.get()).mHandler.sendMessage(msg);

}

*else* *if* ((int)result==0){

Bundle re=*new* Bundle();

re.putString("Return","2");

Message msg=*new* Message();

msg.setData(re);

((login)mActivity.get()).mHandler.sendMessage(msg);

}

*else* *if*((int)result==2)

{

Bundle re=*new* Bundle();

re.putString("Return","3");

Message msg=*new* Message();

msg.setData(re);

((login)mActivity.get()).mHandler.sendMessage(msg);

}

在主类中需要创建回调函数，根据子线程传递的消息判断是否登录成功。

public Handler mHandler=*new* Handler()

{

@Override

public void handleMessage(Message msg)

{

*super*.handleMessage(msg);

Bundle bd=msg.getData();

String content=bd.getString("Return");

*if*(content.equals("0"))

{

int uid=bd.getInt("ID");

setUser\_id(uid);

*try* {

UpdateData.downloadBill();

*// UpdateData.downloadAssets();*

} *catch* (InterruptedException e) {

e.printStackTrace();

} *catch* (internetError e)

{

loginPromptET.setText(R.string.internetError);

loginPromptET.setVisibility(View.VISIBLE);

progressView.setVisibility(View.GONE);

progressView.stopAnimation();

loginTXT.setVisibility(View.VISIBLE);

*return*;

}

File storageDir = getExternalFilesDir(Environment.DIRECTORY\_PICTURES);

photo.setPath(storageDir.getAbsolutePath());

photo.download(getUser\_id());

loginPromptET.setVisibility(View.GONE);

progressView.setVisibility(View.GONE);

progressView.stopAnimation();

loginTXT.setText(R.string.loginSuccess);

loginTXT.setVisibility(View.VISIBLE);

Intent intent = *new* Intent(login.this,BasicActivity.class) ; *//切换Login Activity至User Activity*

startActivity(intent);

finish();

}

*else* *if*(content.equals("1"))*//face error*

{

loginPromptET.setText(R.string.faceError);

loginPromptET.setVisibility(View.VISIBLE);

progressView.setVisibility(View.GONE);

progressView.stopAnimation();

loginTXT.setVisibility(View.VISIBLE);

}

*else* *if*(content.equals("2"))*//password error*

{

loginPromptET.setText(R.string.passwordError);

loginPromptET.setVisibility(View.VISIBLE);

progressView.setVisibility(View.GONE);

progressView.stopAnimation();

loginTXT.setVisibility(View.VISIBLE);

}

*else* *if*(content.equals("3"))

{

loginPromptET.setText(R.string.internetError);

loginPromptET.setVisibility(View.VISIBLE);

progressView.setVisibility(View.GONE);

progressView.stopAnimation();

loginTXT.setVisibility(View.VISIBLE);

}

}

};

### 人脸识别登录的实现

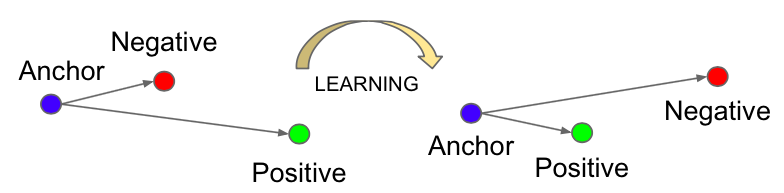
我们组想做人脸识别是因为近年来机器学习非常火热，因此有必要与时俱进。传统基于规则的图像识别成功率不高而基于机器学习的人脸识别准确率可以达到99.9%以上，可以说是非常成熟了。我们认为这种登陆方式的安全性是比传统的账号密码登陆的方式要高很多的，因此采用了它。同时我们也觉得这种安全的验证方式在后续APP的维护和更新中可以得到更为广泛的应用，因此是很有必要的。

1. 简述

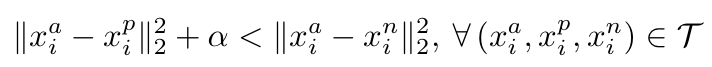
人脸识别基于深度学习，我们使用了facenet这个模型。Facenet是谷歌开发的，拥有0.992±0.003 这惊人的准确率。该项目开源在：<https://github.com/davidsandberg/facenet.git>。

facenet采用不同于传统机器学习的softmax的方法进行学习，而是直接训练一个图像到欧氏空间的一个向量的方式，根据向量之间的distance进行判断。

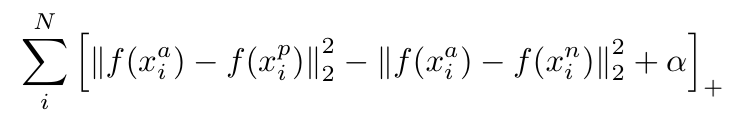
Facenet采用的是经过改良的CNN。他们替换掉了softmax，而是训练三元组之间的距离关系：



对于每一个三元组(anchor,negative,positive)，取他们的二范数，他们需要做到对于同一类的xp不同类的xn，有尽可能多的三元组满足：



目标函数为：



最难的是三元组的选择，好的选择可以让收敛速度很快，否则收敛就会变的非常慢。这里面的实现细节论文中写的很清楚，由于和项目相差的很大，因此不再赘述。

1. 服务端的搭建

有了pre-train model，下一部要做的就是搭建服务器。

由于模型需要非常强大的GPU进行计算，因此不可能像数据库一样通过租借网络服务器来搭建（成本太大），因此我借用了我实验室的服务器进行搭建。**因此不能一直开着，如果老师在看到报告的时候发现服务器已经关闭，希望老师能够理解，这部分的测试我们在视频中演示**。

环境：CPU:i7-7200，内存：16G，GPU：GTX-TITAN X。

系统环境：Ubuntu 16.04.

其他环境：使用Django框架搭建前端。使用tensorflow跑神经网络。

1. 人脸识别部分

识别部分是完成模型导入等预处理后，将模型导入网络，然后将用户的照片传入神经网络，计算得到对应到欧氏空间的一组embedding向量（如果有多个人，那么会得到所有人的embedding结果，这是我们判断最相似的满不满足即可）。如果我们这是做的是保存结果，就直接写入文件。如果是识别工作，那么读一下这个人预存的embedding结果，计算两个向量的二范数，也就是欧氏空间的距离，返回这个值。

def main(args):

images = load\_and\_align\_data(args.image\_files, args.image\_size, args.margin, args.gpu\_memory\_fraction)

*with* tf.Graph().as\_default():

*with* tf.Session() *as* sess:

*# Load the model*

facenet.load\_model(args.model)

*# Get input and output tensors*

images\_placeholder = tf.get\_default\_graph().get\_tensor\_by\_name("input:0")

embeddings = tf.get\_default\_graph().get\_tensor\_by\_name("embeddings:0")

phase\_train\_placeholder = tf.get\_default\_graph().get\_tensor\_by\_name("phase\_train:0")

*# Run forward pass to calculate embeddings*

feed\_dict = {images\_placeholder: images, phase\_train\_placeholder: False}

emb = sess.run(embeddings, feed\_dict=feed\_dict)

nrof\_images = len(args.image\_files)

print('Images:')

*for* i in range(nrof\_images):

print('%1d: %s' % (i, args.image\_files[i]))

print('')

print("emdeddings:")

*for* i in emb:

print(i)

print('')

*# Print distance matrix*

print('Distance matrix')

print(' ', end='')

*for* i in range(nrof\_images):

print(' %1d ' % i, end='')

print('')

*for* i in range(nrof\_images):

print('%1d ' % i, end='')

*for* j in range(nrof\_images):

dist = np.sqrt(np.sum(np.square(np.subtract(emb[i, :], emb[j, :]))))

print(' %1.4f ' % dist, end='')

print('')

如果我们不考虑效率的话，这段代码已经可以work了。但是这样还不够，因为模型的导入非常慢，而这属于重复性质的工作，需要优化。

我开始尝试用静态变量保存模型的结果，但是经过很久的尝试，发现在tensorflow下实现这个还很不容易。因此我转变思路，由于计算资源是有限的，一次只能让一个人进行计算，也就是说访问其实是串行的。我认为如果将网络请求串行化，就可以变成生产者消费者问题，可以通过加互斥锁解决。于是上面的代码需要做一些更改。

def do(self,cond,ffcond):

*with* tf.Graph().as\_default():

gpu\_options = tf.GPUOptions(per\_process\_gpu\_memory\_fraction=0.5)

*with* tf.Session(config=tf.ConfigProto(gpu\_options=gpu\_options, log\_device\_placement=False)) *as* sess:

mywork.Rsaver=facenet.load\_model("/home/sk/shina/facenet/model")

*# Get input and output tensors*

images\_placeholder = tf.get\_default\_graph().get\_tensor\_by\_name("input:0")

embeddings = tf.get\_default\_graph().get\_tensor\_by\_name("embeddings:0")

phase\_train\_placeholder = tf.get\_default\_graph().get\_tensor\_by\_name("phase\_train:0")

cond.acquire()

*while* True:

print("start.......")

cond.wait()

imagePath = [*self*.filePath]

*try*:

images = load\_and\_align\_data(imagePath, 160, 44, 0.5)

*# Run forward pass to calculate embeddings*

feed\_dict = {images\_placeholder: images, phase\_train\_placeholder: False}

emb = sess.run(embeddings, feed\_dict=feed\_dict)

*except*:

*self*.result=100

ffcond.acquire()

ffcond.notifyAll()

ffcond.release()

*continue*

nrof\_images = len(imagePath)

emb\_old = None

*if* (*self*.isInsert == "1"):

*# write database*

*if* len(emb)!=1:

*self*.result=-2

*else*:

*with* open(outputpath + str(*self*.id) + ".txt", 'w') *as* f:

print("fuck:" + outputpath + str(*self*.id) + ".txt")

pickle.dump(emb, f, pickle.HIGHEST\_PROTOCOL)

*with* tf.Graph().as\_default():

gpu\_options = tf.GPUOptions(per\_process\_gpu\_memory\_fraction=0.5)

*with* tf.Session(config=tf.ConfigProto(gpu\_options=gpu\_options,

log\_device\_placement=False)) *as* sess:

mywork.Rsaver = facenet.load\_model("/home/sk/shina/facenet/model")

*# Get input and output tensors*

images\_placeholder = tf.get\_default\_graph().get\_tensor\_by\_name("input:0")

embeddings = tf.get\_default\_graph().get\_tensor\_by\_name("embeddings:0")

phase\_train\_placeholder = tf.get\_default\_graph().get\_tensor\_by\_name(

"phase\_train:0")

cond.acquire()

*while* True:

print("start.......")

cond.wait()

imagePath = [*self*.filePath]

*try*:

images = load\_and\_align\_data(imagePath, 160, 44, 0.5)

*# Run forward pass to calculate embeddings*

feed\_dict = {images\_placeholder: images, phase\_train\_placeholder: False}

emb = sess.run(embeddings, feed\_dict=feed\_dict)

*except*:

*self*.result = 100

ffcond.acquire()

ffcond.notifyAll()

ffcond.release()

*continue*

nrof\_images = len(imagePath)

emb\_old = None

*if* (*self*.isInsert == "1"):

*# write database*

*if* len(emb) != 1:

*self*.result = -2

*else*:

*with* open(outputpath + str(*self*.id) + ".txt", 'w') *as* f:

print("fuck:" + outputpath + str(*self*.id) + ".txt")

pickle.dump(emb, f, pickle.HIGHEST\_PROTOCOL)

*# ---- ok -----*

*self*.result = 10

*else*:

*if* os.path.exists(outputpath + str(*self*.id) + ".txt"):

fr = open(outputpath + str(*self*.id) + ".txt")

inf = pickle.load(fr)

fr.close()

dist = 101

*for* i in range(0, len(emb)):

dist = min(dist, np.sqrt(

np.sum(np.square(np.subtract(emb[i, :], inf[0, :])))))

*# dist = np.sqrt(np.sum(np.square(np.subtract(emb[0, :], inf[0, :]))))*

print(dist)

*self*.result = dist

*else*:

*self*.result = -1

*pass*

ffcond.acquire()

ffcond.notifyAll()

ffcond.release()

*# ---- ok -----*

*self*.result=10

*else*:

*if* os.path.exists(outputpath + str(*self*.id) + ".txt"):

fr = open(outputpath + str(*self*.id) + ".txt")

inf = pickle.load(fr)

fr.close()

dist=101

*for* i in range(0,len(emb)):

dist=min(dist,np.sqrt(np.sum(np.square(np.subtract(emb[i, :], inf[0, :])))))

*#dist = np.sqrt(np.sum(np.square(np.subtract(emb[0, :], inf[0, :]))))*

print(dist)

*self*.result=dist

*else*:

*self*.result=-1

*pass*

ffcond.acquire()

ffcond.notifyAll()

ffcond.release()

我们通过一个函数去激发条件变量，并等待结果。

def ff(self,id,filePath,isInsert,cond,ffcond,mutex):

mutex.acquire()

ffcond.acquire()

*self*.id=id

*self*.filePath=filePath

*self*.isInsert=isInsert

cond.acquire()

cond.notifyAll()

cond.release()

ffcond.wait()

ffcond.release()

mutex.release()

*return* *self*.result

这部分全部封装到一个类中。类初始化的时候做的工作是创建子线程，子线程执行do函数。

def \_\_init\_\_(self,cond,ffcond,ffmutex):

*self*.id=1

*self*.filePath=""

*self*.isInsert=0

*self*.result=-1

*self*.t=threading.Thread(target=*self*.do,args=(cond,ffcond,))

*self*.t.start()

*pass*

1. 接口搭建

我们使用Django框架搭建API。

首先创建工程，然后写一下html代码。由于我们的API只是用来给APP进行交互的，因此不用在意美观问题。

<form *method*="post" *enctype*="multipart/form-data">

<input *type*="file" *name*="file">

<input *type*="submit" *value*="upload">

<input *type*='text' *name*='id'>

<input *type*='text' *name*='isInsert'>

</form>

<p>{{result}}</p>

逻辑部分：我们需要实例化第一部写的类，创建好相关的互斥锁，条件变量等。然后得到request后，得到相关的数据，调用识别的函数，并将结果返回网页端。

*from* django.shortcuts *import* render,render\_to\_response

*from* django.http *import* HttpResponse

*from* facer *import* \*

*import* threading

ffcond=threading.Condition()

cond=threading.Condition()

mutex=threading.Lock()

uploadmutex=threading.Lock()

flow=mywork(cond,ffcond,mutex)

def upload(request):

uploadmutex.acquire()

ff={}

*if* request.method=="POST":

upload\_dir="/home/sk/shina/facenet/server/images"

file\_obj=request.FILES['file']

*with* open(upload\_dir+"/1.jpg",'wb') *as* f:

*for* i in file\_obj.chunks():

f.write(i)

pid=request.POST['id']

isInsert=request.POST['isInsert']

result=flow.ff(pid,upload\_dir+"/1.jpg",isInsert,cond,ffcond,mutex)

ff['result']=result

print(result)

uploadmutex.release()

*return* render(request,'upload.html',ff)

在urls.py中进行网址绑定：

url(r'^upload$',view.upload)

1. App端实现

APP端收到人脸识别的请求之后，会首先确认用户名已经输入，然后获取用户的ID，如果用户的id是存在的，那么调用系统前置相机，让用户拍摄人脸照片：

id = getid(username);

*if*(id==-1)

{

Bundle re=*new* Bundle();

re.putString("Return","1");

Message msg=*new* Message();

msg.setData(re);

mHandler.sendMessage(msg);

}

*else*{

openCamera(id);

}

打开相机：

private void openCamera(int uid)

{

Intent id=*new* Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);

*if* (id.resolveActivity(getPackageManager()) != null) {

File photoFile = null;

*try* {

photoFile = createImageFile();

} *catch* (IOException ex) {

Log.e("wrong file", ex.getMessage(), ex);

}

*if* (photoFile != null) {

Uri photoURI = FileProvider.getUriForFile(*this*, "stream.com.streamapp.fileprovider", photoFile);

id.putExtra(MediaStore.EXTRA\_OUTPUT, photoURI);

id.putExtra("android.intent.extras.CAMERA\_FACING\_FRONT", 1);

id.putExtra("android.intent.extras.CAMERA\_FACING", 1);

setUser\_id(uid);

startActivityForResult(id, 1);

}

}

}

拍摄完成后会调用回调函数，然后我们会把用户id、用户的照片一起输入到服务器，等待服务器的计算结果。

@Override

protected void onActivityResult(int requestCode,int resultCode,Intent data)

{

*super*.onActivityResult(requestCode,resultCode,data);

*if* (resultCode==RESULT\_OK)

{

*if*(requestCode==1)

{

FileInputStream fis = null;

*try* {

final File ff=*new* File(mFilePath);

Thread tThread=*new* Thread(*new* Runnable() {

@Override

public void run() {

checkFace(ff);

}

});

tThread.start();

}

}

}

}

Checkface做的是上传数据，解析返回的distance结果。通常我们认为<0.8就是同一个人，1.1以上基本不会是一个人。由于双胞胎的存在比较危险，因此我们经过很多的尝试和调参，将参数定在0.66。

private void checkFace(File file)

{

RequestBody fileBody = RequestBody.create(MediaType.parse("image/jpeg"), file);

RequestBody requestBody = *new* MultipartBody.Builder()

.setType(MultipartBody.FORM)

.addFormDataPart("file", "1", fileBody)

.addFormDataPart("id", ""+getUser\_id())//Integer.toString(getUser\_id()))

.addFormDataPart("isInsert", "0")

.build();

Request request = *new* Request.Builder()

.url("http://10.214.129.171:9999/upload")

.post(requestBody)

.build();

Response response;

*try* {

response = client.newCall(request).execute();

String jsonString = response.body().string();

Pattern r = Pattern.compile("<p>.\*?</p>");

Matcher m = r.matcher(jsonString);

*if*(m.find())

{

double emb=Double.valueOf(m.group(0).replace("<p>","").replace("</p>",""));

*if* (emb<0.66)

{

Bundle re=*new* Bundle();

re.putString("Return","0");

re.putInt("ID",getUser\_id());

Message msg=*new* Message();

msg.setData(re);

mHandler.sendMessage(msg);

}

*else*{

Bundle re=*new* Bundle();

re.putString("Return","1");

Message msg=*new* Message();

msg.setData(re);

mHandler.sendMessage(msg);

}

}

} *catch* (IOException e) {

Log.d("ff","upload IOException ",e);

}

*return*;

}

## 导出账单

导出账单的部分主要是如何生成excel。我们查阅了相关资料，决定使用开源的jxl。这是安卓端excel操作最为成熟的开源库

我们自己封装了一个静态类，能在用户sd卡的根目录下生成一个excel文件。

首先检查用户sd卡读写的权限，如果没有就动态申请：

public static void verifyStoragePermissions(Activity activity) {

*try* {

*//检测是否有写的权限*

int permission = ActivityCompat.checkSelfPermission(activity,

"android.permission.WRITE\_EXTERNAL\_STORAGE");

*if* (permission != PackageManager.PERMISSION\_GRANTED) {

*// 没有写的权限，去申请写的权限，会弹出对话框*

ActivityCompat.requestPermissions(activity, PERMISSIONS\_STORAGE,REQUEST\_EXTERNAL\_STORAGE);

}

Log.e("fff","IIIII");

} *catch* (Exception e) {

e.printStackTrace();

}

}

通过setData函数，设置需要导出到excel的内容。

这是一个二维的ArrayList，第一维为账单，第二维为账单的集合。

private static ArrayList<ArrayList<String>> excelData=*new*

ArrayList<>();

public static void setData()

{

List<Bills> bills;

*/\* excelData是一个二维的ArrayList 第一维为bill*

*"收入或支出",//in or out String:"收入"*

*"类型", // Type String:"买书"*

*"时间", //date String:"2017-1-1"*

*"地点", //place String:玉泉*

*"金额", //amount String:"100RMB"*

*"备注" //Note String:"无"*

*\*/*

excelData.clear();

bills = DataSupport.where("user\_id = ? ", String.valueOf(login.getUser\_id())).order("date asc").find(Bills.class);

*for*(int i=0;i<bills.size();i++)

{

ArrayList<String> bill= *new* ArrayList<>();

bill.add(bills.get(i).getInOrOut());

bill.add(bills.get(i).getType());

bill.add(bills.get(i).getDate());

bill.add(bills.get(i).getPlace());

bill.add(""+bills.get(i).getAmount());

bill.add(bills.get(i).getNote());

excelData.add(bill);

}

}

然后通过createExcel函数生成相应的excel文件。具体的流程为创建工作表，然后一个个Cell添加进去。

public static void createExcel(Activity activity)

{

*try* {

verifyStoragePermissions(activity);

File saveFile= *new* File("mnt/sdcard/text.xls");

*if*(saveFile.isFile()&&saveFile.exists())

{

saveFile.delete();

}

WritableWorkbook book = Workbook.createWorkbook(saveFile);

WritableSheet sheet = book.createSheet("账单",0);

String[] labelNames= { "收入或支出",*//in or out*

"类型",*// Type*

"时间",*//date*

"地点",*//place*

"金额",*//amount*

"备注" *//Note*

};

*for* (int i=0;i<labelNames.length;i++)

{

Label tmp=*new* Label(i,0,labelNames[i]);

sheet.addCell(tmp);

}

*for*(int i=0;i<excelData.size();i++)

{

ArrayList<String> bill= excelData.get(i);

*for*(int j=0;j<bill.size();j++)

{

sheet.addCell(*new* Label(j,i+1,bill.get(j)));

}

}

book.write();

book.close();

final WeakReference<Activity> mActivity=*new* WeakReference<Activity> (activity);

((BasicActivity)mActivity.get()).handle(1);

*return*;

} *catch* (IOException e) {

solveFail(activity);

e.printStackTrace();

} *catch* (RowsExceededException e) {

solveFail(activity);

e.printStackTrace();

} *catch* (WriteException e) {

solveFail(activity);

e.printStackTrace();

}

}

## 统计图表的生成

这一部分我们用来将用户的账单数据可视化成折线图和饼状图的形式展示给用户。

统计图的生成我们查找了相关的控件，最后采用了hellocharts这个非常受开发者欢迎的开源控件。同样的我们封装了一下相关的操作。

我们使用折线图反应用户年月日的支出收入情况，通过饼状图反应用户在一定时间内的各类支出占比。

通过用户不同的选择，展示不同的图给用户。

### 选择数据

绘制图表前，我们需要获得绘制所需要的数据。这一操作较为简单，主要分为三步。

对于第一张折线图，首先根据用户的选择，判断读取数据的类型，是支出还是收入，时间尺度是年还是月还是日。然后去数据库中读出需要的数据，并进行统计。最后把统计出来的数据按坐标点(x,y)的方式存放进List以供调用。

对于第二张饼图，首先还是根据用户的选择，设置日期范围以及收入或支出，然后调用数据库读取数据，再统计各个种类的金额，把金额以及对应的名称放入List，以供调用。

private void setData(String InOrOut, int scale)

{

dateIn.clear();

pointInX.clear();

pointInY.clear();

Calendar calendar = Calendar.getInstance();

Date d=*new* Date();

calendar.setTime(d);

*for*(int i=0;i<5;i++)

{

String text;

SimpleDateFormat sf = *new* SimpleDateFormat("yyyy-MM-dd");

*switch* (scale){

*case* 0*://日*

text = sf.format(calendar.getTime());

calendar.add(Calendar.DATE, -1);

dateIn.add(i,text);

*break*;

*case* 1*://月*

text = sf.format(calendar.getTime());

calendar.add(Calendar.MONTH, -1);

dateIn.add(i,text.substring(0,7));

*break*;

*case* 2*://年*

text = sf.format(calendar.getTime());

calendar.add(Calendar.YEAR, -1);

dateIn.add(i,text.substring(0,4));

*break*;

*default:break*;

}

}

d=*new* Date();

calendar.setTime(d);

*for*(int i=0;i<5;i++)

{

pointInX.add(i,(float)i);

double sum = 0;

SimpleDateFormat sf = *new* SimpleDateFormat("yyyy-MM-dd");

*if* (scale == 0) {

calendar.add(Calendar.DATE, 1);*//后一天日期*

String day1 = sf.format(calendar.getTime());

calendar.add(Calendar.DATE, -1);*//前一天日期*

String day2 = sf.format(calendar.getTime());

sum = DataSupport.where("user\_id = ? and date > ? and date < ? and inOrOut = ? and state <> 3 ", String.valueOf(login.getUser\_id()), day2, day1, InOrOut).sum(Bills.class, "amount", double.class);

calendar.add(Calendar.DATE, -1);

}

*if* (scale == 1) {

calendar.add(Calendar.MONTH, 1);*//后一个月日期*

String month1 = sf.format(calendar.getTime());

calendar.add(Calendar.MONTH, -1);*//前一个月日期*

String month2 = sf.format(calendar.getTime());

sum = DataSupport.where("user\_id = ? and date > ? and date < ? and inOrOut = ? and state <> 3", String.valueOf(login.getUser\_id()), month2.substring(0, 7), month1.substring(0, 7), InOrOut).sum(Bills.class, "amount", double.class);

calendar.add(Calendar.MONTH, -1);

}

*if* (scale == 2 ) {

calendar.add(Calendar.YEAR, 1);*//后一年日期*

String year1 = sf.format(calendar.getTime());

calendar.add(Calendar.YEAR, -1);*//前一年日期*

String year2 = sf.format(calendar.getTime());

sum = DataSupport.where("user\_id = ? and date > ? and date < ? and inOrOut = ? and state <> 3", String.valueOf(login.getUser\_id()), year2.substring(0, 4), year1.substring(0, 4), InOrOut).sum(Bills.class, "amount", double.class);

calendar.add(Calendar.YEAR, -1);

}

pointInY.add(i, (float)sum);

}

Collections.reverse(dateIn);

Collections.reverse(pointInY);

mPieData.clear();

mlabels.clear();

String[] labelsOut = {"meal","transportation","shopping",

"daily","clothes","vegetables","fruit","snack",

"book","study", "house", "investment", "social",

"amusement", "makeup", "call", "sport", "travel",

"medicine", "office", "digit", "gift", "repair",

"wine", "redpacket", "other"};

String[] labelsIn = {"salary", "redpacket", "parttime", "other"};

Log.d("labelsIn", String.valueOf(labelsIn.length));

Log.d("scale",String.valueOf(scale));

Log.d("inorout",InOrOut);

*if* (InOrOut.equals("in")) {

*for* (int i = 0; i < labelsIn.length; i++) {

double sum = 0;

String day1, day2;

SimpleDateFormat sf = *new* SimpleDateFormat("yyyy-MM-dd");

*if* (scale == 0) {*//近七天以来*

d = *new* Date();

calendar.setTime(d);

calendar.add(calendar.DATE, 1);

day1 = sf.format(calendar.getTime());

calendar.add(calendar.DATE, -7);

day2 = sf.format(calendar.getTime());

sum = DataSupport.where("user\_id = ? and date > ? and date < ? and inOrOut = ? and type = ? and state <> 3 ", String.valueOf(login.getUser\_id()), day2, day1, InOrOut, labelsIn[i]).sum(Bills.class, "amount", double.class);

Log.d("date1", day1);

Log.d("date2", day2);

Log.d("sum", String.valueOf(sum));

}

*if* (scale == 1) {*//近三月以来*

d = *new* Date();

calendar.setTime(d);

calendar.add(calendar.MONTH, 1);

day1 = sf.format(calendar.getTime());

calendar.add(calendar.MONTH, -3);

day2 = sf.format(calendar.getTime());

sum = DataSupport.where("user\_id = ? and date > ? and date < ? and inOrOut = ? and type = ? and state <> 3 ", String.valueOf(login.getUser\_id()), day2, day1, InOrOut, labelsIn[i]).sum(Bills.class, "amount", double.class);

}

*if* (scale == 2) {*//近一年来*

d = *new* Date();

calendar.setTime(d);

calendar.add(calendar.YEAR, 1);

day1 = sf.format(calendar.getTime());

calendar.add(calendar.YEAR, -2);

day2 = sf.format(calendar.getTime());

sum = DataSupport.where("user\_id = ? and date > ? and date < ? and inOrOut = ? and type = ? and state <> 3", String.valueOf(login.getUser\_id()), day2, day1, InOrOut, labelsIn[i]).sum(Bills.class, "amount", double.class);

}

Log.d("pi", labelsIn[i] + String.valueOf(sum));

*if*(sum!=0) {

mPieData.add((float) sum);

mlabels.add(labelsIn[i]);

}

}

}

*if* (InOrOut.equals("out")) {

*for* (int i = 0; i < labelsOut.length; i++) {

double sum = 0;

String day1, day2;

SimpleDateFormat sf = *new* SimpleDateFormat("yyyy-MM-dd");

*if* (scale == 0) {*//近七天以来*

d = *new* Date();

calendar.setTime(d);

calendar.add(calendar.DATE, 1);

day1 = sf.format(calendar.getTime());

calendar.add(calendar.DATE, -7);

day2 = sf.format(calendar.getTime());

sum = DataSupport.where("user\_id = ? and date > ? and date < ? and inOrOut = ? and type = ? and state <> 3 ", String.valueOf(login.getUser\_id()), day2, day1, InOrOut, labelsOut[i]).sum(Bills.class, "amount", double.class);

}

*if* (scale == 1) {*//近三月以来*

d = *new* Date();

calendar.setTime(d);

calendar.add(calendar.MONTH, 1);

day1 = sf.format(calendar.getTime());

calendar.add(calendar.MONTH, -3);

day2 = sf.format(calendar.getTime());

sum = DataSupport.where("user\_id = ? and date > ? and date < ? and inOrOut = ? and type = ? and state <> 3 ", String.valueOf(login.getUser\_id()), day2, day1, InOrOut, labelsOut[i]).sum(Bills.class, "amount", double.class);

}

*if* (scale == 2) {*//近一年来*

d = *new* Date();

calendar.setTime(d);

calendar.add(calendar.YEAR, 1);

day1 = sf.format(calendar.getTime());

calendar.add(calendar.YEAR, -2);

day2 = sf.format(calendar.getTime());

sum = DataSupport.where("user\_id = ? and date > ? and date < ? and inOrOut = ? and type = ? and state <> 3", String.valueOf(login.getUser\_id()), day2, day1, InOrOut, labelsOut[i]).sum(Bills.class, "amount", double.class);

}

*if*(sum!=0) {

mPieData.add((float) sum);

mlabels.add(labelsOut[i]);

}

}

}

}

### 数据转换

将账单数据转换成hellocharts能够接受的形式。

private void getData()

{

mAxisXValues.clear();

*for* (int i=0;i<dateIn.size();i++)

{

mAxisXValues.add(*new* AxisValue(i).setLabel(dateIn.get(i)));

}

mPointValues.clear();

*for* (int i=0;i<pointInX.size();i++)

{

mPointValues.add(*new* PointValue(pointInX.get(i),pointInY.get(i)));

}

mValues.clear();

*for* (int i=0;i<mlabels.size();i++)

{

SliceValue tmp=*new* SliceValue(mPieData.get(i),ChartUtils.pickColor());

tmp.setLabel(mlabels.get(i));

mValues.add(tmp);

}

mPieChartData = *new* PieChartData(mValues);

mPieChartData.setHasLabels(hasLabels);

mPieChartData.setHasLabelsOnlyForSelected(false);

mPieChartData.setHasLabelsOutside(hasLabelsOutside);

mPieChartData.setHasCenterCircle(hasCenterCircle);

*if* (isExploded)

{

mPieChartData.setSlicesSpacing(24);*//设置分离距离*

}

}

### 绘制

1. 画饼状图

private void drawPieChart()

{

mPieChartView.setPieChartData(mPieChartData);

mPieChartView.setCircleFillRatio(0.9f);*//设置放大缩小范围*

}

1. 画折线图

private void drawLineChart()

{

Line line = *new* Line(mPointValues).setColor(Color.parseColor("#FFCD41")); *//折线的颜色*

ArrayList<Line> lines = *new* ArrayList<Line>();

line.setShape(ValueShape.CIRCLE);*//折线图上每个数据点的形状 这里是圆形 （有三种 ：ValueShape.SQUARE ValueShape.CIRCLE ValueShape.SQUARE）*

line.setCubic(false);*//曲线是否平滑*

*// line.setStrokeWidth(3);//线条的粗细，默认是3*

line.setFilled(false);*//是否填充曲线的面积*

line.setHasLabels(true);*//曲线的数据坐标是否加上备注*

*// line.setHasLabelsOnlyForSelected(true);//点击数据坐标提示数据（设置了这个line.setHasLabels(true);就无效）*

line.setHasLines(true);*//是否用直线显示。如果为false 则没有曲线只有点显示*

line.setHasPoints(true);*//是否显示圆点 如果为false 则没有原点只有点显示*

lines.add(line);

LineChartData data = *new* LineChartData();

data.setLines(lines);

*//坐标轴*

Axis axisX = *new* Axis(); *//X轴*

axisX.setHasTiltedLabels(true); *//X轴下面坐标轴字体是斜的显示还是直的，true是斜的显示*

*// axisX.setTextColor(Color.WHITE); //设置字体颜色*

axisX.setTextColor(Color.parseColor("#D6D6D9"));*//灰色*

*// axisX.setName("未来几天的天气"); //表格名称*

axisX.setTextSize(11);*//设置字体大小*

axisX.setMaxLabelChars(7); *//最多几个X轴坐标，意思就是你的缩放让X轴上数据的个数7<=x<=mAxisValues.length*

axisX.setValues(mAxisXValues); *//填充X轴的坐标名称*

data.setAxisXBottom(axisX); *//x 轴在底部*

*// data.setAxisXTop(axisX); //x 轴在顶部*

axisX.setHasLines(true); *//x 轴分割线*

Axis axisY = *new* Axis(); *//Y轴*

axisY.setName("");*//y轴标注*

axisY.setTextSize(11);*//设置字体大小*

data.setAxisYLeft(axisY); *//Y轴设置在左边*

mLineChartView.setInteractive(true);

mLineChartView.setZoomType(ZoomType.HORIZONTAL); *//缩放类型，水平*

mLineChartView.setMaxZoom((float) 3);*//缩放比例*

mLineChartView.setLineChartData(data);

mLineChartView.setVisibility(View.VISIBLE);

mLineChartView.setContainerScrollEnabled(true, ContainerScrollType.HORIZONTAL);

mLineChartView.setContainerScrollEnabled(true, ContainerScrollType.VERTICAL);

Viewport v = *new* Viewport(mLineChartView.getMaximumViewport());

v.bottom = 0;

v.top = 5;

v.left = 10 - 7;

v.right = 10 - 1;

mLineChartView.setCurrentViewport(v);

}

## 同步支付宝、银行卡

关于这一部分的想法我们小组经历了很大的转折。首先我们尝试寻找免费的API。支付宝自己肯定不会提供，但我们查找了很多第三方的，基本都是需要收费的。由于资金限制，因此我们肯定不能轻易通过购买的方式解决。

然后我们尝试通过爬虫的方式将账单爬去下来。但这条路挣扎了很久，我们发现支付宝的反爬虫措施做的强的可怕，用我们能做的哪怕是cookies都是不可行的。因此虽然很不甘心，爬虫的道路基本宣告死亡。

最后我们小组在会议中讨论决定，可以转变思维用监听短信和通知的方式，获取相关的账单信息。具体的思路是：当银行卡有收入支出的时候，会给用户的手机发送短信，这是一个广播请求，所以app可以在获得权限的情况下监听到短信的内容，通过过滤，就可以得到相关的账单信息。支付宝的话，只要去监听通知内容即可。

其实监听通知也会在收到短信的时候触发，但是我们决定如果能够获得短信，那么以获得短信为主，因为短信内容比通知内容更为详细。

### 银行卡同步实现

银行卡需要监听SMS短信。我们需要开一个receiver，然后继承BroadcastReceiver类，重写Onreceive函数。当短信到的时候，会先触发这个函数，然后我们获取发信人，如果是银行，就分析内容，如果是账单信息，那么插入数据库即可。.

Manifest文件：

<receiver

android*:*name=".broadcast.smsRecevier"

android*:*permission="android.permission.BROADCAST\_SMS">

<intent-filter android*:*priority="2147483647">

<action android*:*name="android.provider.Telephony.SMS\_DELIVER"/>

<action android*:*name="android.provider.Telephony.SMS\_RECEIVED"/>

</intent-filter>

</receiver>

Java文件：

public class smsRecevier extends BroadcastReceiver{

public smsHandler mSmsHandler;

private Handler mHandler;

private void sendHandlerMessage(String sender,String content)

{

mSmsHandler=*new* smsHandler();

mHandler=mSmsHandler.getHandler();

String account="";

String amount="";

boolean isIn=false;

boolean isValid=false;

*if*(sender.equals("95533"))*//建设银行*

{

Pattern r = Pattern.compile("您尾号[0-9]\*");

Matcher m = r.matcher(content);

*if* (m.find())

{

account=m.group(0).substring(3);

*if* (account.length()!=4)

{

*return*;

}

*else*{

*if*(content.contains("支出"))

{

isIn=false;

}

*else* *if*(content.contains("收入")||content.contains("存入"))

{

isIn=true;

}

*else*{

*return*;

}

r=Pattern.compile("人民币\\d\*\\.\\d\*");

m=r.matcher(content);

*if*(m.find())

{

amount=m.group(0).substring(3);

}

*else*{

*return*;

}

Log.e("sms", "acount:"+account+",amount:"+amount+",sender:"+sender);

*try* {

UpdateData.addBills(Double.parseDouble(amount),(isIn*?*"in"*:*"out"),"jianshe");

} *catch* (InterruptedException e) {

*//e.printStackTrace();*

}

}

}

}

}

@Override

public void onReceive(Context context, Intent intent) {

*if*(intent.getAction().equals(SMS\_RECEIVED\_ACTION))

{

Object[] pdus = (Object[]) intent.getExtras().get("pdus");

*for* (Object pdu *:* pdus) {

SmsMessage smsMessage = SmsMessage.createFromPdu((byte[]) pdu);

String sender = smsMessage.getDisplayOriginatingAddress();

*if*(sender.equals("95533"))

{

*//短信内容*

String content = smsMessage.getDisplayMessageBody();

long date = smsMessage.getTimestampMillis();

Date tiemDate = *new* Date(date);

SimpleDateFormat simpleDateFormat = *new* SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String time = simpleDateFormat.format(tiemDate);

sendHandlerMessage(sender,content);

}

abortBroadcast();

}

}

}

}

### 支付宝同步实现

支付宝同步是监听通知，因为支付宝不会有短信的通知。监听通知需要开一个service，然后继承NotificationListenerService类，并重写相应的函数。当获得通知的时候，筛选出支付宝的，然后匹配出相应的信息，如果是合法的账单信息，就入库。

Manifest文件：

<service

android*:*name=".notificationlistener.notificationListener"

android*:*label="@string/service\_name"

android*:*permission="android.permission.BIND\_NOTIFICATION\_LISTENER\_SERVICE">

<intent-filter>

<action android*:*name="android.service.notification.NotificationListenerService"/>

</intent-filter>

</service>

Java文件：

@RequiresApi(api = Build.VERSION\_CODES.JELLY\_BEAN\_MR2)

public class notificationListener extends NotificationListenerService {

public notificationHandler mHandleShare;

public Handler mHandler;

private void sendHandlerMessage(String title, String content)

{

mHandleShare= *new* notificationHandler();

mHandler=mHandleShare.getHandler();

String amount="";

boolean isIn=false;

Pattern r = Pattern.compile("你有一笔[0-9|\\.]\*");

Matcher m = r.matcher(content);

*if*(m.find())

{

amount=m.group(0).substring(4);

*try* {

UpdateData.addBills(Double.parseDouble(amount),"out","Alipay");

} *catch* (InterruptedException e) {

*//e.printStackTrace();*

}

}

*else*{

*return*;

}

}

@Override

public void onNotificationPosted(StatusBarNotification sbn) {

Log.e("SevenNLS","Notification posted"+sbn.getPackageName());

*if* (! "com.eg.android.AlipayGphone".equals(sbn.getPackageName()))

{

*return*;

}

Notification notification = sbn.getNotification();

*if* (notification == null) {

*return*;

}

PendingIntent pendingIntent = null;

*if* (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.KITKAT) {

Bundle extras = notification.extras;

*if* (extras != null) {

String title = extras.getString(Notification.EXTRA\_TITLE, "");

String content = extras.getString(Notification.EXTRA\_TEXT, "");

sendHandlerMessage(title,content);

}

}

}

@Override

public void onNotificationRemoved(StatusBarNotification sbn) {

}

@Override

public void onListenerConnected()

{

}

}

## 用户头像的获取与更新

用户头像的下载我专门封装了类，由于其有可能会被多个子线程执行，而数据是static的，因此会有线程安全问题，需要加上锁。

### 用户图片接口

用户头像的API位于阿里云服务器上，使用Django搭建。

HTML:

<form *method*="post" *enctype*="multipart/form-data">

    <input *type*="file" *name*="file">

    <input *type*="submit" *value*="upload">

    <input *type*="text" *name*="id">

</form>

<p>{{result}}</p>

savephoto.py: 这个文件的作用是保存用户post的图片，并且使用PIL进行图片压缩和1:1裁剪。

*from* django.shortcuts *import* render

*from* django.http *import* HttpResponse

*from* PIL *import* Image

def savePhoto(request):

ff={}

*if* request.method=="POST":

save\_dir="/home/shina/shina/Stream/server/server/images"

file\_obj=request.FILES['file']

uid=request.POST['id']

*with* open(save\_dir+"/"+str(uid)+".jpg",'wb') *as* f:

*for* i in file\_obj.chunks():

f.write(i)

img=Image.open(save\_dir+"/"+str(uid)+".jpg")

print(img.size,img.info)

img=img.convert('RGB')

minsize=min(img.size[0],img.size[1])

region=img.crop((0,0,minsize,minsize))

print(region.info)

*if* minsize>=500:

minsize=500

region.thumbnail((minsize,minsize))

region.save(save\_dir+"/"+str(uid)+".jpg")

ff['result']=1

*return* render(request,'editphoto.html',ff)

ff['result']=0

*return* render(request,'editphoto.html',ff)

getphoto.py：这个文件的作用是返回用户请求的图片。以byteStream的形式返回一个httpresponse。

*from* django.shortcuts *import* render,render\_to\_response

*from* django.http *import* HttpResponse

*from* PIL *import* Image

def getphoto(request,new\_id):

*#parent\_path = path.dirname(d)*

*#uid=request.POST['id']*

imagepath = "/home/shina/shina/Stream/server/server/images/"+str(new\_id)+".jpg"

print("imagepath="+str(imagepath))

*try*:

image\_data = open(imagepath,"rb").read()

*except*:

*return* HttpResponse('<h>key not found</h>')

*return* HttpResponse(image\_data,content\_type="image/png")

Urls.py:：绑定url

url(r'^editphoto$',savephoto.savePhoto),

url(r'^getphoto/(?P<new\_id>.+)/$',getphoto.getphoto,name="image"),

### Download实现

download函数以用户id为请求post到服务器上，然后得到服务器返回的byteStream，并且保存在本地的目录中。

本身不是很难，主要是文件路径比较难处理。我使用的是getExtraFilesDir的方式获取一个file，然后得到他的绝对路径。

当获取图片成功的时候，标记一下用户是有头像的。否则就标记成用默认头像。

File storageDir = getExternalFilesDir(Environment.DIRECTORY\_PICTURES);

photo.setPath(storageDir.getAbsolutePath());

public class photo {

private static String path="";

private static String savePath="";

final private static Integer photoProtect= Integer.valueOf(1);

private static Boolean hasPhoto = Boolean.valueOf(false);

private static final OkHttpClient client = *new* OkHttpClient.Builder().connectTimeout(10, TimeUnit.SECONDS).build();

*/\**

*\* if you want to know whether the user has photo, call this*

*\* @return the state*

*\*/*

public static boolean getPhotoState()

{

synchronized (photoProtect)

{

*return* hasPhoto;

}

}

*/\**

*\* if you want to know the absolute path of the photo, call this*

*\* @return the absolute path of the photo*

*\*/*

public static String getPath()

{

*return* savePath;

}

*/\**

*\* if you want to flash photo, just call this*

*\* @param uid is the user's id*

*\*/*

public static void download(int uid)

{

synchronized (photoProtect)

{

hasPhoto=false;

Thread t = *new* Thread(*new* Runnable() {

private final String serverUrl="http://47.95.245.4:9999/getphoto/[%d]/";

@Override

public void run() {

String url=serverUrl.replace("[%d]",""+ login.getUser\_id());

Request request = *new* Request.Builder()

.url(url)

.build();

Response response;

*try* {

response = client.newCall(request).execute();

InputStream inputStream = response.body().byteStream();*//得到图片的流*

Bitmap bitmap = BitmapFactory.decodeStream(inputStream);

File file = *new* File(path, "/"+login.getUser\_id()+".jpg");

FileOutputStream fos = null;

*try* {

file.createNewFile();

fos = *new* FileOutputStream(file.getAbsolutePath());

*if* (fos != null) {

bitmap.compress(Bitmap.CompressFormat.JPEG, 100, fos);

hasPhoto=true;

savePath=file.getAbsolutePath();

Log.e("aaa",file.getAbsolutePath());

fos.close();

}

} *catch* (IOException e) {

e.printStackTrace();

}

} *catch* (IOException e) {

Bundle re=*new* Bundle();

re.putString("Return","1");

Message msg=*new* Message();

msg.setData(re);

*//mHandler.sendMessage(msg);*

Log.d("ff", "upload IOException ", e);

}*catch*(NullPointerException e)

{

hasPhoto=false;

}

}

});

t.start();

}

}

*/\**

*\* don't call this*

*\*/*

public static void setPath(String tpath)

{

path=tpath;

}

}

## 数据库的逻辑与实现

由于这是一个记账APP，数据的存放以及读取是它的核心部分，所以这部分我们花了很多的力气，修改了很多次。整个APP用到数据库的地方有两处，一个是登录部分，这部分逻辑比较简单，单纯的匹配即可，在前文已经有所提到，而且被后一部分所涵盖，因此略去不讲。第二部分是账单的添加、修改、删除时，本模块主要对其进行展开，主要包括业务逻辑、本地实现、云端同步等。

在本地数据库我们使用了Litepal，这是github上开源的一个项目，它把数据库的各种基本操作如建表、添加、修改、删除、查找、聚合等都封装地很好，而且也可以用原始的sql语句来进行操作，非常的方便。

在服务器端我们使用了MySQL，这是我们在数据库课上使用过的，所以对其比较熟悉。

### 表的建立

首先说一下整个表的结构。对于账单，本来对于消费与资产，我们用了两张表，一张用来记录消费，一张用来记录资产。但经过仔细分析，我们发现这两张表的功能有所重叠，所以最后只用了一张表，其他功能全部可以通过对这一张表的操作来实现。表的定义在Bills.java中：

public class Bills extends DataSupport{

@Column(unique = true, defaultValue = "unknown")

private int id;

private int user\_id;

private double amount;

private String date;

private String place;

private String inOrOut;*//收入或支出*

private String type;

private String note;

private String methods;

private int state;

private String timeStamp;

public String getMethods() {

*return* methods;

}

public void setMethods(String methods) {

*this*.methods = methods;

}

public int getState() {

*return* state;

}

public void setState(int state) {

*this*.state = state;

}

public String getTimeStamp() {

*return* timeStamp;

}

public void setTimeStamp(String timeStamp) {

*this*.timeStamp = timeStamp;

}

public String getInOrOut() {

*return* inOrOut;

}

public void setInOrOut(String inOrOut) {

*this*.inOrOut = inOrOut;

}

public int getId() {

*return* id;

}

public void setId(int id) {

*this*.id = id;

}

public int getUser\_id() {

*return* user\_id;

}

public void setUser\_id(int user\_id) {

*this*.user\_id = user\_id;

}

public double getAmount() {

*return* amount;

}

public void setAmount(double amount) {

*this*.amount = amount;

}

public String getDate() {

*return* date;

}

public void setDate(String date) {

*this*.date = date;

}

public String getPlace() {

*return* place;

}

public void setPlace(String place) {

*this*.place = place;

}

public String getType() {

*return* type;

}

public void setType(String type) {

*this*.type = type;

}

public String getNote() {

*return* note;

}

public void setNote(String note) {

*this*.note = note;

}

}

表中有11个attributes，介绍如下：

Id 为一个消费的流水号，一笔支出或收入有一个唯一的id，且按时间递增。

User\_id 为用户的身份标记，是代表这笔消费属于哪个用户。

Amount 为这笔消费的金额，可以是小数。

Date 为这笔消费的日期，由系统自动获取。

InOrOut 标记了这笔消费是支出还是收入，”in”代表收入，”out”代表支出。

Type 代表了消费的类型，其中支出有饮食、交通、购物、日用、服饰、蔬菜、水果、零食、书籍、学习、住房、投资、社交、娱乐、美容、通讯、运动、红包、旅行、医疗等，而收入主要分为工资、红包、兼职等。

Note 为注记，作为对这笔消费的备注。

Methods 为消费的方式，主要分为支付宝、各大银行或者钱包。

State 用来标记这笔消费的状态，这个状态是指与云端数据相比较的状态，主要分为“添加”、“修改”以及“删除”，这个会在后面展开。

TimeStamp 指这笔消费的时间戳，即最近一次修改的时间，主要用来与云端数据同步用。

### 数据库基础逻辑

这里的基础逻辑值的就是item的添加、删除、修改。这里只讲本地数据库，云端数据会放到后面讲。

1. 添加

本地数据库的添加会有两处，一处是手动添加的，即用户自己输入金额种类，一种是自动添加的，即系统通过监听手机的通知与短信，自动把item生成，用户在修改处修改种类即可。本地数据库添加后，需要把item的state属性标记为”添加”，这里用数字1表示，这是用作同步的。然后再调用同步的接口，接口会单独开线程去同步，不会影响主线程。

收入的手动添加:

public void onClick(View v){

String type = *new* String();

*switch*(v.getId())

{

*case* R.id.salary*:*

type = "salary";

*break*;

*case* R.id.redpacket*:*

type = "redpacket";

*break*;

*case* R.id.parttime*:*

type = "parttime";

*break*;

*case* R.id.other*:*

type = "other";

*break*;

}

final String typeFinal = *new* String(type) ;

final AlertDialog.Builder builder = *new* AlertDialog.Builder(getContext());

View dialogView = LayoutInflater.from(getContext()).inflate(R.layout.bill\_info,null);

final EditText amountET = dialogView.findViewById(R.id.amount);

final EditText noteET = dialogView.findViewById(R.id.note);

final RadioGroup methodRG = dialogView.findViewById(R.id.chooseMethod);

final RadioButton walletBTN = dialogView.findViewById(R.id.walletBTN);

methodRG.setOnCheckedChangeListener(*new* RadioGroup.OnCheckedChangeListener() {

@Override

public void onCheckedChanged(RadioGroup group, @IdRes int checkedId) {

*switch*(checkedId){

*case* R.id.alipayBTN*:*

method = "alipay";

*break*;

*case* R.id.nongyeBTN*:*

method = "nongye";

*break*;

*case* R.id.jiansheBTN*:*

method = "jianshe";

*break*;

*case* R.id.walletBTN*:*

method = "other";

*break*;

}

Log.e("choose",method);

}

});

builder.setView(dialogView);

builder.setNegativeButton("取消",null);

builder.setPositiveButton("确定", *new* DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

Bills bill = *new* Bills();

String amount=amountET.getText().toString().trim();

String note = noteET.getText().toString().trim();

bill.setType(typeFinal);

bill.setAmount(Double.valueOf(amount));

bill.setNote(note);

bill.setPlace("somewhere");

bill.setInOrOut("in");

SimpleDateFormat sdf = *new* SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String time = sdf.format(*new* Date());

bill.setDate(time);

bill.setUser\_id(login.getUser\_id());

bill.setTimeStamp(time);

bill.setState(1);

bill.setMethods(method);

bill.save();

*try* {

UpdateData.UploadBill();

} *catch* (InterruptedException e) {

e.printStackTrace();

}

Toast.makeText(getActivity(),"账单添加成功",Toast.LENGTH\_SHORT).show();

}

});

AlertDialog dialog = builder.create();

dialog.setCanceledOnTouchOutside(true);

dialog.show();

}

}

支出的手动添加：

public void onClick(View v){

String type = *new* String();

*switch*(v.getId())

{

*case* R.id.meal*:*

type = "meal";

*break*;

*case* R.id.transportation*:*

type = "transportation";

*break*;

*case* R.id.shopping*:*

type = "shopping";

*break*;

*case* R.id.daily*:*

type = "daily";

*break*;

*case* R.id.clothes*:*

type = "clothes";

*break*;

*case* R.id.vegetables*:*

type = "vegetables";

*break*;

*case* R.id.fruit*:*

type = "fruit";

*break*;

*case* R.id.snack*:*

type = "snack";

*break*;

*case* R.id.book*:*

type = "book";

*break*;

*case* R.id.study*:*

type = "study";

*break*;

*case* R.id.house*:*

type = "house";

*break*;

*case* R.id.investment*:*

type = "investment";

*break*;

*case* R.id.social*:*

type = "social";

*break*;

*case* R.id.amusement*:*

type = "amusement";

*break*;

*case* R.id.makeup*:*

type = "makeup";

*break*;

*case* R.id.call*:*

type = "call";

*break*;

*case* R.id.sport*:*

type = "sport";

*break*;

*case* R.id.travel*:*

type = "travel";

*break*;

*case* R.id.medicine*:*

type = "medicine";

*break*;

*case* R.id.office*:*

type = "office";

*break*;

*case* R.id.digit*:*

type = "digit";

*break*;

*case* R.id.gift*:*

type = "gift";

*break*;

*case* R.id.repair*:*

type = "repair";

*break*;

*case* R.id.wine*:*

type = "wine";

*break*;

*case* R.id.redpacket*:*

type = "redpacket";

*break*;

*case* R.id.other*:*

type = "other";

*break*;

}

final String typeFinal = *new* String(type) ;

final AlertDialog.Builder builder = *new* AlertDialog.Builder(getContext());

View dialogView = LayoutInflater.from(getContext()).inflate(R.layout.bill\_info,null);

final EditText amountET = dialogView.findViewById(R.id.amount);

final EditText noteET = dialogView.findViewById(R.id.note);

final RadioGroup methodRG = dialogView.findViewById(R.id.chooseMethod);

final RadioButton walletBTN = dialogView.findViewById(R.id.walletBTN);

methodRG.setOnCheckedChangeListener(*new* RadioGroup.OnCheckedChangeListener() {

@Override

public void onCheckedChanged(RadioGroup group, @IdRes int checkedId) {

*switch*(checkedId){

*case* R.id.alipayBTN*:*

method = "alipay";

*break*;

*case* R.id.nongyeBTN*:*

method = "nongye";

*break*;

*case* R.id.jiansheBTN*:*

method = "jianshe";

*break*;

*case* R.id.walletBTN*:*

method = "other";

*break*;

}

Log.e("choose",method);

}

});

builder.setView(dialogView);

builder.setNegativeButton("取消",null);

builder.setPositiveButton("确定", *new* DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

Bills bill = *new* Bills();

String amount=amountET.getText().toString().trim();

String note = noteET.getText().toString().trim();

bill.setType(typeFinal);

bill.setAmount(Double.valueOf(amount));

bill.setNote(note);

bill.setPlace("somewhere");

bill.setInOrOut("out");

SimpleDateFormat sdf = *new* SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String time = sdf.format(*new* Date());

bill.setDate(time);

bill.setUser\_id(login.getUser\_id());

bill.setState(1);

bill.setTimeStamp(time);

bill.setMethods(method);

bill.save();

*try* {

UpdateData.UploadBill();

} *catch* (InterruptedException e) {

e.printStackTrace();

}

Toast.makeText(getActivity(),"账单添加成功",Toast.LENGTH\_SHORT).show();

*//Log.d("insetttttttttt", DataSupport.findAll(Bills.class).toString());*

}

});

AlertDialog dialog = builder.create();

dialog.setCanceledOnTouchOutside(true);

dialog.show();

}

}

账单自动添加的部分是调用一个已经写好的接口，其实手动添加也可以调用接口，但是由于接口是之后添加的，就没有调用。这也算是设计上的一个小纰漏，但好在并不会影响程序性能。

public static void addBills(double amount, String inOrOut, String

methods) throws InterruptedException {

Bills bill = *new* Bills();

bill.setType("other");

bill.setAmount(Double.valueOf(amount));

bill.setNote("");

bill.setPlace("somewhere");

bill.setInOrOut(inOrOut);

SimpleDateFormat sdf = *new* SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String time = sdf.format(*new* Date());

bill.setDate(time);

bill.setUser\_id(login.getUser\_id());

bill.setTimeStamp(time);

bill.setState(1);

bill.setMethods(methods);

bill.save();

UploadBill();

*return*;

}

}

1. 修改

修改时，根据点击事件传来的Billid确定需要更新的item，以及需要更新的值，调用方法即可，Litepal都已经封装好了。修改完后需要把state值设为”修改”，这里用2表示。同样地，本地修改完之后调用同步函数UploadBill();

private void save(){

String newInOrOut, newTime, newPlace, newNote, newMethod;

double newAmount;

newInOrOut = InOrOut.getText().toString();

newMethod = method.getText().toString();

newTime = time.getText().toString();

newNote = note.getText().toString();

newPlace = place.getText().toString();

newAmount = Double.valueOf(amount.getText().toString());

Log.e("inorout",newInOrOut);

Bills newBill = *new* Bills();

newBill.setInOrOut(newInOrOut.equals("收入")*?*"in"*:*"out");

newBill.setDate(newTime);

newBill.setNote(newNote);

newBill.setPlace(newPlace);

newBill.setAmount(newAmount);

newBill.setState(2);

*switch* (newMethod){

*case* "支付宝"*:*

newBill.setMethods("alipay");

*break*;

*case* "建设银行"*:*

newBill.setMethods("jianshe");

*break*;

*case* "农业银行"*:*

newBill.setMethods("nongye");

*break*;

*case* "钱包"*:*

newBill.setMethods("other");

*break*;

}

SimpleDateFormat sdf = *new* SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String time = sdf.format(*new* Date());

newBill.setTimeStamp(time);

newBill.update(BillId);

*try* {

UpdateData.UploadBill();

} *catch* (InterruptedException e) {

e.printStackTrace();

}

*//DataSupport.update(Bills.class, values, BillId);*

changed=true;

Toast.makeText(*this*,"已保存",Toast.LENGTH\_SHORT).show();

}

1. 删除

本地的删除分为两个阶段。第一个阶段是软删除，即把state标记为”删除“即可，这里用3表示。真正的删除在UploadBill()中，这是为了同步的方便，真删除会增加同步时查找item的耗时，软删除则可以方便的知晓被删除的item。同步完后本地的数据也会删除。由于有了state，即使本地数据还没被真正删除，在操作数据库时我们也能方便地避开它。

public void onClick(View v) {

changed=true;

ContentValues values = *new* ContentValues();

values.put("state", 3);

DataSupport.update(Bills.class, values, BillId);

Toast.makeText(BillDetail.this,"已删除",Toast.LENGTH\_SHORT).show();

*try* {

UpdateData.UploadBill();

} *catch* (InterruptedException e) {

e.printStackTrace();

}

Intent intent = *new* Intent();

intent.putExtra("result", changed);

setResult(RESULT\_CODE, intent);

finish();

}

1. 查找

需要用到查找的地方有很多，在显示消费时要按时间查找，再生成统计图时要按种类查找，在导出账单时候也要查找。我们还是利用Litepal强大的DataSuuport包，它可以方便地让你指定删选条件，再返回指定的值，可以是item对象，也可以是诸如sum，max这样的聚合值。

举几个例子 在显示账单时，我们根据user\_id确定哪些item是属于该用户的，再按照收入或支出选择”in”还是“out”，然后再根据用户选择的日期来找该日期符合条件的item。但有一点需要注意，由于之前用了软删除，我们需要额外判断item的state是否是删除状态，我们只收集未被删除的item。

bills = DataSupport.where(

"user\_id = ? and date >= ? and date < ? and state <> 3 ",

String.valueOf(login.getUser\_id()),mYear+"-"+mMonth+"-01", nextYear+"-"+nextMonth+"-01")

.order("date desc").limit(5).find(Bills.class);

计算当日总支出时，则可以把最后的.find替换成.sum，这样就会返回总和了。

expense= DataSupport.where(

"user\_id = ? and date >= ? and date < ? and inOrOut = ? and state <> 3 ",

String.valueOf(login.getUser\_id()), mYear+"-"+mMonth+"-01", nextYear+"-"+nextMonth+"-01" ,"out" )

.sum(Bills.class, "amount", double.class);

显示账单时填入数据的整个函数如下：

private void initData(){

String nextYear = mMonth.equals("12")*?* String.valueOf(Integer.valueOf(mYear)+1)*:*mYear;

*//Log.d("nextyear", nextYear);*

String nextMonth = "";

*switch* (mMonth){

*case* "01"*:* nextMonth="02";*break*;

*case* "02"*:* nextMonth="03";*break*;

*case* "03"*:* nextMonth="04";*break*;

*case* "04"*:* nextMonth="05";*break*;

*case* "05"*:* nextMonth="06";*break*;

*case* "06"*:* nextMonth="07";*break*;

*case* "07"*:* nextMonth="08";*break*;

*case* "08"*:* nextMonth="09";*break*;

*case* "09"*:* nextMonth="10";*break*;

*case* "10"*:* nextMonth="11";*break*;

*case* "11"*:* nextMonth="12";*break*;

*case* "12"*:* nextMonth="01";*break*;

*default:break*;

}

*//Log.d("nextmonth", nextMonth);*

income = DataSupport.where("user\_id = ? and date >= ? and date < ? and inOrOut = ? and state <> 3 ", String.valueOf(login.getUser\_id()), mYear+"-"+mMonth+"-01", nextYear+"-"+nextMonth+"-01" ,"in" ).sum(Bills.class, "amount", double.class);

expense= DataSupport.where("user\_id = ? and date >= ? and date < ? and inOrOut = ? and state <> 3 ", String.valueOf(login.getUser\_id()), mYear+"-"+mMonth+"-01", nextYear+"-"+nextMonth+"-01" ,"out" ).sum(Bills.class, "amount", double.class);

dataList =*new* ArrayList<String>();

iconList = *new* ArrayList<Integer>();

categoryList = *new* ArrayList<Integer>();

*//Log.d("date", mYear+"-"+mMonth+"-01");*

bills = DataSupport.where("user\_id = ? and date >= ? and date < ? and state <> 3 ", String.valueOf(login.getUser\_id()), mYear+"-"+mMonth+"-01", nextYear+"-"+nextMonth+"-01").order("date desc").limit(5).find(Bills.class);

*for* (int i = 0; i < */\*((bills.size()>5)?5:\*/*bills.size(); i++) {

dataList.add( (bills.get(i).getInOrOut().equals("in") *?* "+"*:*"-") + String.valueOf(bills.get(i).getAmount()));

*switch*(bills.get(i).getType())

{

*case* "meal"*:*

categoryList.add(R.string.meal);

iconList.add(R.drawable.meal);

*break*;

*case* "transportation"*:*

categoryList.add(R.string.transportation);

iconList.add(R.drawable.transportation);

*break*;

*case* "shopping"*:*

categoryList.add(R.string.shopping);

iconList.add(R.drawable.shopping);

*break*;

*case* "daily"*:*

categoryList.add(R.string.daily);

iconList.add(R.drawable.daily);

*break*;

*case* "clothes"*:*

categoryList.add(R.string.clothes);

iconList.add(R.drawable.clothes);

*break*;

*case* "vegetables"*:*

categoryList.add(R.string.vegetable);

iconList.add(R.drawable.vegetable);

*break*;

*case* "fruit"*:*

categoryList.add(R.string.fruit);

iconList.add(R.drawable.fruit);

*break*;

*case* "snack"*:*

categoryList.add(R.string.snack);

iconList.add(R.drawable.snack);

*break*;

*case* "book"*:*

categoryList.add(R.string.book);

iconList.add(R.drawable.book);

*break*;

*case* "study"*:*

categoryList.add(R.string.study);

iconList.add(R.drawable.study);

*break*;

*case* "house"*:*

categoryList.add(R.string.house);

iconList.add(R.drawable.house);

*break*;

*case* "investment"*:*

categoryList.add(R.string.investment);

iconList.add(R.drawable.investment);

*break*;

*case* "social"*:*

categoryList.add(R.string.social);

iconList.add(R.drawable.social);

*break*;

*case* "amusement"*:*

categoryList.add(R.string.amusement);

iconList.add(R.drawable.amusement);

*break*;

*case* "makeup"*:*

categoryList.add(R.string.makeup);

iconList.add(R.drawable.makeup);

*break*;

*case* "call"*:*

categoryList.add(R.string.call);

iconList.add(R.drawable.call);

*break*;

*case* "sport"*:*

categoryList.add(R.string.sport);

iconList.add(R.drawable.sport);

*break*;

*case* "travel"*:*

categoryList.add(R.string.travel);

iconList.add(R.drawable.travel);

*break*;

*case* "medicine"*:*

categoryList.add(R.string.medicine);

iconList.add(R.drawable.medicine);

*break*;

*case* "office"*:*

categoryList.add(R.string.office);

iconList.add(R.drawable.office);

*break*;

*case* "digit"*:*

categoryList.add(R.string.digit);

iconList.add(R.drawable.digit);

*break*;

*case* "gift"*:*

categoryList.add(R.string.gift);

iconList.add(R.drawable.gift);

*break*;

*case* "repair"*:*

categoryList.add(R.string.repair);

iconList.add(R.drawable.repair);

*break*;

*case* "wine"*:*

categoryList.add(R.string.wine);

iconList.add(R.drawable.wine);

*break*;

*case* "redpacket"*:*

categoryList.add(R.string.redpacket);

iconList.add(R.drawable.redpacket);

*break*;

*case* "other"*:*

categoryList.add(R.string.other);

iconList.add(R.drawable.other);

*break*;

*case* "parttime"*:*

categoryList.add(R.string.parttime);

iconList.add(R.drawable.parttime);

*break*;

*case* "salary"*:*

categoryList.add(R.string.salary);

iconList.add(R.drawable.salary);

*break*;

}

}

比如生成统计图时，根据用户给定的时间尺度，分别给出近几日、近几月、近几年的收支情况。再把值传入绘图所需的dataList中。

*for*(int i=0;i<5;i++)

{

pointInX.add(i,(float)i);

double sum = 0;

SimpleDateFormat sf = *new* SimpleDateFormat("yyyy-MM-dd");

*if* (scale == 0) {

calendar.add(Calendar.DATE, 1);*//后一天日期*

String day1 = sf.format(calendar.getTime());

calendar.add(Calendar.DATE, -1);*//前一天日期*

String day2 = sf.format(calendar.getTime());

sum = DataSupport.where("user\_id = ? and date > ? and date < ? and inOrOut = ? and state <> 3 ", String.valueOf(login.getUser\_id()), day2, day1, InOrOut).sum(Bills.class, "amount", double.class);

calendar.add(Calendar.DATE, -1);

}

*if* (scale == 1) {

calendar.add(Calendar.MONTH, 1);*//后一个月日期*

String month1 = sf.format(calendar.getTime());

calendar.add(Calendar.MONTH, -1);*//前一个月日期*

String month2 = sf.format(calendar.getTime());

sum = DataSupport.where("user\_id = ? and date > ? and date < ? and inOrOut = ? and state <> 3", String.valueOf(login.getUser\_id()), month2.substring(0, 7), month1.substring(0, 7), InOrOut).sum(Bills.class, "amount", double.class);

calendar.add(Calendar.MONTH, -1);

}

*if* (scale == 2 ) {

calendar.add(Calendar.YEAR, 1);*//后一年日期*

String year1 = sf.format(calendar.getTime());

calendar.add(Calendar.YEAR, -1);*//前一年日期*

String year2 = sf.format(calendar.getTime());

sum = DataSupport.where("user\_id = ? and date > ? and date < ? and inOrOut = ? and state <> 3", String.valueOf(login.getUser\_id()), year2.substring(0, 4), year1.substring(0, 4), InOrOut).sum(Bills.class, "amount", double.class);

calendar.add(Calendar.YEAR, -1);

}

pointInY.add(i, (float)sum);

}

其他的地方也差不多，都是根据条件去本地数据库查找数据，再将返回结果进行处理。就是时间的处理比较麻烦，比如月份，如果要得到12月的数据，就要从后一年1月1日的00:00:00来往前统计，我们用了一个switch-case语句来处理这个。

### 数据库同步

同步分为upload以及download，即上载数据以及下载数据。这里用到了一种比较巧妙的算法，介绍如下:

首先是与远端连接的操作，我们把它写成了一个接口，只要接受sql语句就能返回结果的html文本值。同时也负责把同步后的本地item的state设为9或者删除。

private static class myThread extends Thread {

final private String url;

final private String query;

final private int BillId;

final private int op;

myThread(String url, String query, int BillId, int op) {

*this*.url = url;

*this*.query = query;

*this*.BillId = BillId;

*this*.op = op;

}

@Override

public void run() {

*try* {

OkHttpClient mClient = *new* OkHttpClient();

*/\**

*\*\* Request body*

*\*/*

RequestBody body = *new* FormBody.Builder()

.add("content", query)

.build();

Log.e("fff", "query:" + query);

*// Log.e("fff",body.toString());*

*/\**

*\*\* headers, url, body should be here*

*\*/*

Request tRequest = *new* Request.Builder()

.url(url)

.post(body)

.build();

Call call = mClient.newCall(tRequest);

Response tResponse = call.execute();

*// Log.e("eee","\*\*\*"+tResponse.toString());*

String ans = tResponse.body().string();

result = ans;

*if* (op == 0)

{

ContentValues values = *new* ContentValues();

values.put("state", 9);

DataSupport.update(Bills.class, values, BillId);

}

*else* *if* (op == 1){

ContentValues values = *new* ContentValues();

values.put("state", 9);

DataSupport.update(Bills.class, values, BillId);

}

*else* *if* (op == 2){

DataSupport.delete(Bills.class, BillId);

}

*// Log.e("eee",ans);*

} *catch* (IOException e) {

Log.e("fff", "1111");

e.printStackTrace();

} *catch* (Exception e) {

Log.e("fff", "\*\*\*\*\*");

e.printStackTrace();

}

}

}

本地upload时，筛选出所有state是1的item,这是新在本地添加还未同步到云端的数据，需要在远端insert，然后把state置为9；筛选出所有state是2的item，这是在本地修改但还为同步到云端的数据，需要在远端upload，然后把state置为9；筛选出所有state为3的item，这是本地已经软删除的数据，需要在远端delete，然后本地也delete。

public static void UploadBill() throws InterruptedException {

List<Bills> bills;

String query = "";

*//添加同步*

bills = DataSupport.where("user\_id = ? and state = 1 ", String.valueOf(login.getUser\_id())).find(Bills.class);

*for* (int i = 0; i < bills.size(); i++) {

query = "insert into Bills (id, user\_id, amount, date, place, inOrOut, type, note, timeStamp, methods ) values( " + bills.get(i).getId() + "," +

bills.get(i).getUser\_id() + "," + bills.get(i).getAmount() + ", \"" + bills.get(i).getDate() + "\" ,\"" + bills.get(i).getPlace() + "\", \"" +

bills.get(i).getInOrOut() + "\",\"" + bills.get(i).getType() + "\", \"" + bills.get(i).getNote() + "\", \"" + bills.get(i).getTimeStamp() + "\", \""+bills.get(i).getMethods()+ "\" );";

Log.d("query", query);

myThread tThread = new myThread(mUrl, query, bills.get(i).getId(), 0);

tThread.start();

}

*//修改同步*

bills = DataSupport.where("user\_id = ? and state = 2 ", String.valueOf(login.getUser\_id())).find(Bills.class);

*for* (int i = 0; i < bills.size(); i++) {

query = "UPDATE Bills set amount = " + bills.get(i).getAmount() + " ,date = '" + bills.get(i).getDate() + "' ,place = '" +

bills.get(i).getPlace() + "' ,inOrOut = '" + bills.get(i).getInOrOut() + "' ,type = '" + bills.get(i).getType() + "' ,note = '"

+ bills.get(i).getNote() + "' ,timeStamp = '" + bills.get(i).getTimeStamp() + "' ,methods = '" + bills.get(i).getMethods() + "' WHERE id = " + bills.get(i).getId() + ";";

Log.d("query1", query);

myThread tThread = new myThread(mUrl, query, bills.get(i).getId(), 1);

tThread.start();

}

*//删除同步*

bills = DataSupport.where("user\_id = ? and state = 3 ", String.valueOf(login.getUser\_id())).find(Bills.class);

*for* (int i = 0; i < bills.size(); i++) {

query = "DELETE FROM Bills where id = " + bills.get(i).getId() + " and user\_id = " + bills.get(i).getUser\_id() +";" + "INSERT into DeletedBills (id, user\_id, timeStamp)" +

"values( " + bills.get(i).getId() +"," + bills.get(i).getUser\_id() + ",\"" + bills.get(i).getTimeStamp() + "\");";

Log.d("query2", query);

myThread tThread = new myThread(mUrl, query, bills.get(i).getId(), 2);

tThread.start();

}

*return*;

}

而本地download则比较简单。每次登录成功后，会进行一次download。如果本地还没有数据，则把远端所有数据都复制下来即可，如果本地有数据，则根据时间戳（timeStamp），找到远端所有时间戳大于本地时间戳最大值的item，同步到本地即可。这里有个问题，就是远端删除的item,本地不能知道，所以只能在远端加一个DeletedBills表，记录已经被删除的item的id,user\_id以及timeStamp，然后也根据时间戳来获知哪些是要在本地删除的。

public static void downloadBill() throws InterruptedException,

internetError {

result = "";

int count = DataSupport.where("user\_id = ?", String.valueOf(login.getUser\_id())).count(Bills.class);

Log.d("userid", "" + login.getUser\_id());

String latestTimeStamp;

String query = "";

String query1 = "";

Log.d("count", "" + count);

*if* (count == 0)

query = "select id, user\_id, amount, date, place, inOrOut, note, type, timeStamp, methods from Bills where user\_id = " + login.getUser\_id() + ";";

*else* {

latestTimeStamp = DataSupport.max(Bills.class, "timeStamp", String.class);

query = "select id, user\_id, amount, date, place, inOrOut, note, type, timeStamp, methods from Bills where user\_id = " + login.getUser\_id() + " and timeStamp > '" + latestTimeStamp + "';";

query1 = "select id from DeletedBills where user\_id = " + login.getUser\_id() + "and timeStamp > '" + latestTimeStamp + "';";

}

myThread tThread = new myThread(mUrl, query , 0 , -1);

tThread.start();

tThread.join();

*if* (result.equals("")) {

*throw* *new* internetError("网络连接错误!");

}

Log.d("result:", result);

Pattern r = Pattern.compile(regex.resultPattern);

Matcher m = r.matcher(result);

*if* (m.find()) {

String res = m.group(0).replaceAll("<br>", "&");

res = res.replaceAll("\t", "#");

Log.d("res:", res);

String[] lines = res.split("&");

Log.d("length:", "" + lines.length);

*for* (int i = 2; i < lines.length; i++) {

String line = lines[i];

String[] bill = line.split("#");

count = DataSupport.where("id = ?", bill[0]).count(Bills.class);

*if* (count == 0) {

Bills bills = *new* Bills();

bills.setId(Integer.valueOf(bill[0]));

bills.setUser\_id(Integer.valueOf(bill[1]));

bills.setAmount(Double.valueOf(bill[2]));

bills.setDate(bill[3]);

bills.setPlace(bill[4]);

bills.setInOrOut(bill[5]);

bills.setNote(bill[6]);

bills.setType(bill[7]);

bills.setTimeStamp(bill[8]);

bills.setState(9);

bills.setMethods(bill[9]);

bills.save();

}

*else*

{

Bills bills = *new* Bills();

bills.setUser\_id(Integer.valueOf(bill[1]));

bills.setAmount(Double.valueOf(bill[2]));

bills.setDate(bill[3]);

bills.setPlace(bill[4]);

bills.setInOrOut(bill[5]);

bills.setNote(bill[6]);

bills.setType(bill[7]);

bills.setTimeStamp(bill[8]);

bills.setState(9);

bills.setMethods(bill[9]);

bills.update(Integer.valueOf(bill[0]));

}

}

}

tThread = *new* myThread(mUrl, query1 , 0 , -1);

tThread.start();

tThread.join();

*if* (result.equals("")) {

*throw* *new* internetError("网络连接错误!");

}

Log.d("result1:", result);

r = Pattern.compile(regex.resultPattern);

m = r.matcher(result);

*if* (m.find()) {

String res = m.group(0).replaceAll("<br>", "&");

res = res.replaceAll("\t", "#");

Log.d("res:", res);

String[] lines = res.split("&");

Log.d("length:", "" + lines.length);

*for* (int i = 2; i < lines.length; i++) {

String line = lines[i];

String[] bill = line.split("#");

DataSupport.delete(Bills.class, Integer.valueOf(bill[0]));

}

}

*return*;

}

下面是一些同步效果演示，包括一条item的创建、修改与删除的upload，download操作无法用截图演示，但通过卸载后重装，能够发现数据还是与远端一样的，证明了download的成功。

我们添加一条item：

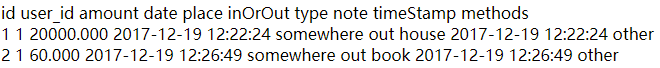


在未联网时，通过adb shell查看本地数据库。



我们可以看见有一条状态为1，类型为book，金额为60的新数据。

然后联网，查看远端数据库



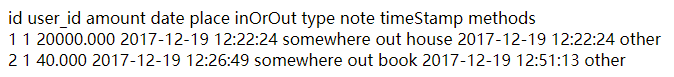
本地的状态变成了9：



修改这条数据，本地state变成了2：



联网后，查看远端数据：



本地状态又变成了9：



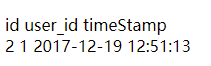
删除时，本地state变成了3：



联网后，查看远端数据：



远端DeletedBills中数据：



本地数据被删除：



## UI部分

### 实现

在UI部分，我们的整体风格借鉴了微信的设计风格，使用Andriod原生的各种控件以及事件监听函数实现。UI部分事实上是一个庞大且繁杂的部分，在这里我们只列出以下四个关键的实现，至于事件监听、布局加载、与后台交互等较为容易的部分便略去不谈。

1. 主页滑动切换或点击下方导航栏切换

采用UI Fragment+ViewPager技术，将主页划分为四个Fragment碎片并交由Fragment Manager托管，避免了activity频繁切换造成过度占用内存的问题。

这里需要实现ViewPager的adapter：

public class MyFragmentPagerAdapter extends FragmentPagerAdapter {

private FragmentManager mFragmentManager;

private List<Fragment> mlist;

public MyFragmentPagerAdapter(FragmentManager fm, List<Fragment> list)

{

*super*(fm);

mFragmentManager = fm;

mlist=list;

}

@Override

public Fragment getItem(int index)

{

*return* mlist.get(index);

}

@Override

public int getCount()

{

*return* mlist.size();

}

}

在BasicActivity类中初始化一个adapter并将ViewPager组件交由其托管。

mViewPager = (ViewPager) findViewById(R.id.myViewPager);

mlist = *new* ArrayList<>();

mlist.add(*new* BillFragment());

mlist.add(*new* PropertyFragment());

mlist.add(*new* ChartFragment());

mlist.add(*new* UserFragment());

madapter = *new* MyFragmentPagerAdapter(getSupportFragmentManager(),mlist);

mViewPager.setAdapter(madapter);

mViewPager.addOnPageChangeListener(*new* OnPageChangeListener() {

@Override

public void onPageScrolled(int position, float positionOffset, int positionOffsetPixels) {

}

@Override

public void onPageSelected(int position) {

}

@Override

public void onPageScrollStateChanged(int state) {

*if*(state == 2){

selected(mViewPager.getCurrentItem());

}

}

});

mViewPager.setCurrentItem(0);

其中selected函数表示某个页面已选中，其实现如下：

private void selected(int index)

{

*switch*(index)

{

*case* 0*:*

billBTN.setCompoundDrawables(null,\_drawable0,null,null);

propertyBTN.setCompoundDrawables(null,drawable1,null,null);

chartBTN.setCompoundDrawables(null,drawable2,null,null);

userBTN.setCompoundDrawables(null,drawable3,null,null);

*break*;

*case* 1*:*

billBTN.setCompoundDrawables(null,drawable0,null,null);

propertyBTN.setCompoundDrawables(null,\_drawable1,null,null);

chartBTN.setCompoundDrawables(null,drawable2,null,null);

userBTN.setCompoundDrawables(null,drawable3,null,null);

*break*;

*case* 2*:*

billBTN.setCompoundDrawables(null,drawable0,null,null);

propertyBTN.setCompoundDrawables(null,drawable1,null,null);

chartBTN.setCompoundDrawables(null,\_drawable2,null,null);

userBTN.setCompoundDrawables(null,drawable3,null,null);

*break*;

*case* 3*:*

billBTN.setCompoundDrawables(null,drawable0,null,null);

propertyBTN.setCompoundDrawables(null,drawable1,null,null);

chartBTN.setCompoundDrawables(null,drawable2,null,null);

userBTN.setCompoundDrawables(null,\_drawable3,null,null);

*break*;

}

}

1. 账单页与资产页的列表显示

采用RecyclerView控件而非ListView控件，将数据库中数据进行处理并以item的形式展现，尽可能地回收资源，避免了使用中可能出现的卡顿问题。

使用RecyclerView前需要实现自己的适配器myAdapter()与ViewHolder()：

class myAdapter extends RecyclerView.Adapter<myAdapter.myViewHolder> {

private MyItemClickListener mItemClickListener;

private MyItemLongClickListener myItemLongClickListener;

public void setOnItemClickListener(MyItemClickListener listener){

*this*.mItemClickListener=listener;

}

public void setOnItemLongClickListener(MyItemLongClickListener listener){

*this*.myItemLongClickListener = listener;

}

@Override

public myAdapter.myViewHolder onCreateViewHolder(ViewGroup parent, int viewType)

{

myAdapter.myViewHolder holder = *new* myAdapter.myViewHolder(LayoutInflater.from(

getActivity()).inflate(R.layout.bill\_item\_layout, parent,

false),mItemClickListener,myItemLongClickListener);

*return* holder;

}

@Override

public void onBindViewHolder(myAdapter.myViewHolder holder, int position)

{

holder.data.setText(dataList.get(position));

holder.category.setText(categoryList.get(position));

holder.icon.setImageResource(iconList.get(position));

}

@Override

public int getItemCount()

{

*return* dataList.size();

}

class myViewHolder extends RecyclerView.ViewHolder {

TextView category;

TextView data;

ImageView icon;

private MyItemClickListener mListener;

private MyItemLongClickListener myItemLongClickListener;

public myViewHolder(View view,MyItemClickListener listener,MyItemLongClickListener longClickListener){

*super* (view);

category = (TextView)view.findViewById(R.id.category);

data = (TextView)view.findViewById(R.id.data);

icon = (ImageView)view.findViewById(R.id.icon);

*this*.mListener = listener;

*this*.myItemLongClickListener=longClickListener;

view.setOnClickListener(*new* View.OnClickListener() {

@Override

public void onClick(View v) {

*if*(mListener != null){

mListener.onItemClick(v,getAdapterPosition());

}

}

});

view.setOnLongClickListener(*new* View.OnLongClickListener() {

@Override

public boolean onLongClick(View v) {

*if*(myItemLongClickListener!=null){

myItemLongClickListener.onLongItemClick(v,getAdapterPosition());

}

*return* false;

}

});

}

}

}

接着添加item之间的分割线类，通过android的官方文档我们发现可以对recyclerView设置item decoration。按照文档，我们首先新建一个类，为每个item设置一个margin：

class MyItemDivider extends RecyclerView.ItemDecoration {

@Override

public void getItemOffsets(Rect outRect, View view, RecyclerView parent, RecyclerView.State state) {

*//设定底部边距为1px*

outRect.set(0, 0, 0, 1);

}

}

使用时，首先new一个LayoutManager，将recyclerView交由其托管，再设置适配器、监听函数与分割线装饰等：

recyclerView=view.findViewById(R.id.bill\_recycler);

mLayoutManager = *new* LinearLayoutManager(view.getContext());

recyclerView.setLayoutManager(mLayoutManager);

recyclerView.setItemAnimator(*new* DefaultItemAnimator());

recyclerView.setAdapter(mAdapter = *new* myAdapter());

recyclerView.addItemDecoration(*new* MyItemDivider());

mAdapter.setOnItemClickListener(*new* MyItemClickListener() {

@Override

public void onItemClick(View view, int postion) {

Intent intent = *new* Intent(getContext(), BillDetail.class);

int BillId = bills.get(postion).getId();

intent.putExtra("BillId",BillId);

startActivityForResult(intent, REQUEST\_CODE);

}

});

mAdapter.setOnItemLongClickListener(*new* MyItemLongClickListener(){

@Override

public void onLongItemClick(View v, int position){

final int pos = position;

PopupMenu popupMenu = *new* PopupMenu(getContext(),v);

popupMenu.getMenuInflater().inflate(R.menu.deletemenu,popupMenu.getMenu());

popupMenu.setOnMenuItemClickListener(*new* PopupMenu.OnMenuItemClickListener() {

@Override

public boolean onMenuItemClick(MenuItem item) {

int BillId = bills.get(pos).getId();

ContentValues values = *new* ContentValues();

values.put("state", 3);

DataSupport.update(Bills.class, values, BillId);

*try* {

UpdateData.UploadBill();

} *catch* (InterruptedException e) {

e.printStackTrace();

}

*new* MyTask().execute();

Toast.makeText(getActivity(),"已删除",Toast.LENGTH\_SHORT).show();

*return* true;

}

});

popupMenu.show();

}

});

列表的刷新通过一个线程来实现，即再次调用initData()函数从数据库中获取数据，执行完后同时适配器数据改变：

public class MyTask extends AsyncTask{

@Override

protected Object doInBackground(Object[] objects){

initData();

*try* {

UpdateData.UploadBill();

} *catch* (InterruptedException e) {

e.printStackTrace();

}

*return* null;

}

@Override

protected void onPostExecute(Object o){

expenseSum.setText(String.valueOf(expense)+" 元");

incomeSum.setText(String.valueOf(income)+" 元");

*super*.onPostExecute(o);

mAdapter.notifyDataSetChanged();

}

}

这里的代码是以BillFragment为例的，PropertyFragment中操作类似，在此不再赘述了。

1. 时间选取的实现

这里我们使用了github上的一个开源项目——TimePickerDialog。

具体使用如下，通过点击将用户的选择传回mYear和mMonth，然后交给数据库进行筛选。

timePickerDialog = *new* TimePickerDialog.Builder()

.setType(Type.YEAR\_MONTH)

.setThemeColor(R.color.colorPrimary)

.setCallBack(*new* OnDateSetListener(){

@Override

public void onDateSet(TimePickerDialog timePickerDialog, long millseconds) {

SimpleDateFormat sf = *new* SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String text = sf.format(*new* Date(millseconds));

mYear = text.substring(0,4);

mMonth = text.substring(5,7);

*//Log.d("year", mYear);*

yearTV.setText(mYear+"年");

monthTV.setText(mMonth+"月");

initData();

*new* MyTask().execute();

}

})

.build();

pickTime = view.findViewById(R.id.pickTime);

pickTime.setOnClickListener(*new* View.OnClickListener() {

@Override

public void onClick(View view){

timePickerDialog.show(getChildFragmentManager(),"选择日期");

}

});

1. 欢迎界面的实现

初对于欢迎界面的想法是设置一个activity，让其显示欢迎图片即可，但实践后发现会有几秒等待时间，并不是我们想要实现的效果。因为setContentView需要时间完成。

后来查询资料以后发现可以通过修改style的方式实现，将欢迎界面的图片放入资源文件，并在style中讲其设为背景即可。

<style name="startTheme" parent="Theme.AppCompat.NoActionBar">

<item name="android:windowBackground">@drawable/splashscreen</item>

<item name="android:windowNoTitle">true</item>

</style>

修改manifest配置文件：

<activity

android*:*name=".splashScreen"

android*:*screenOrientation="portrait"

android*:*theme="@style/startTheme">

<intent-filter>

<action android*:*name="android.intent.action.MAIN"/>

<category android*:*name = "android.intent.category.LAUNCHER"/>

</intent-filter>

</activity>

最终splashScreen的实现如下：

public class splashScreen extends AppCompatActivity {

private static int SPLASH\_TIME\_OUT = 3000;

@Override

protected void onCreate(Bundle savedInstanceState) {

*super*.onCreate(savedInstanceState);

*//to avoid the time spent on setContentView*

*//setContentView(R.layout.splashscreen);*

*if*((getIntent().getFlags()&Intent.FLAG\_ACTIVITY\_BROUGHT\_TO\_FRONT)!=0) {

finish();

*return*;

}

*new* Handler().postDelayed(*new* Runnable(){

public void run(){

Intent i = *new* Intent(splashScreen.this, login.class);

startActivity(i);

finish();

}

}, SPLASH\_TIME\_OUT);

}

}

### 展示

这里节选几个主要的页面进行展示。

欢迎界面：



数秒以后进入登陆界面：



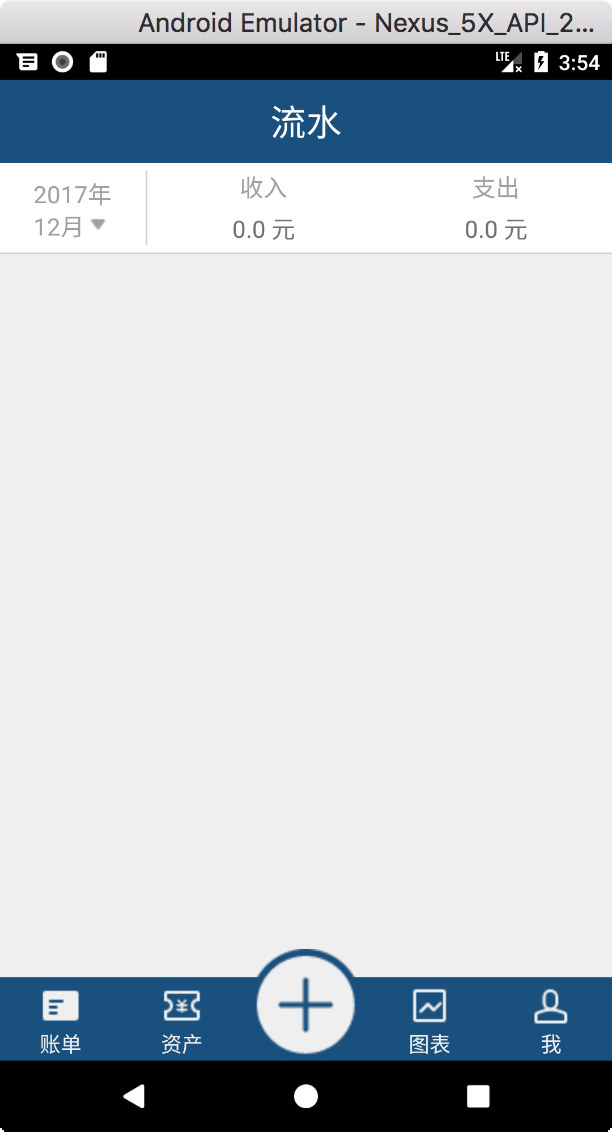
我们首先注册一个账号：



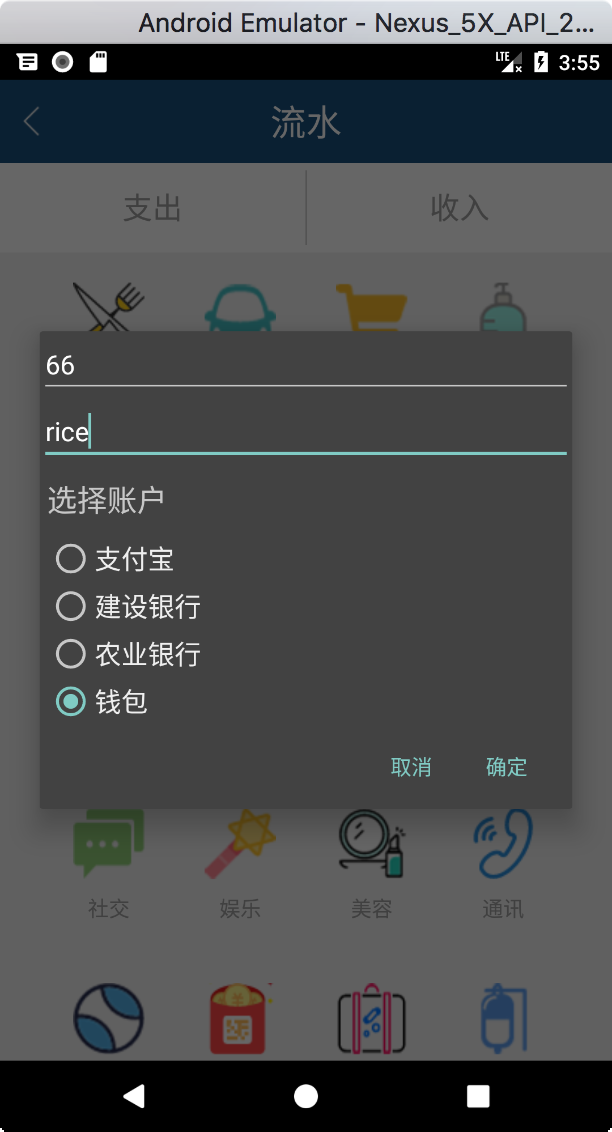
接着使用刚刚的用户名与密码登录：



登陆成功进入主界面：



添加一笔支出账单：



再添加一笔收入账单：



同理再添加两笔支出账单后我们看到主页一共有四条账单了：



资产页也进行了统计：



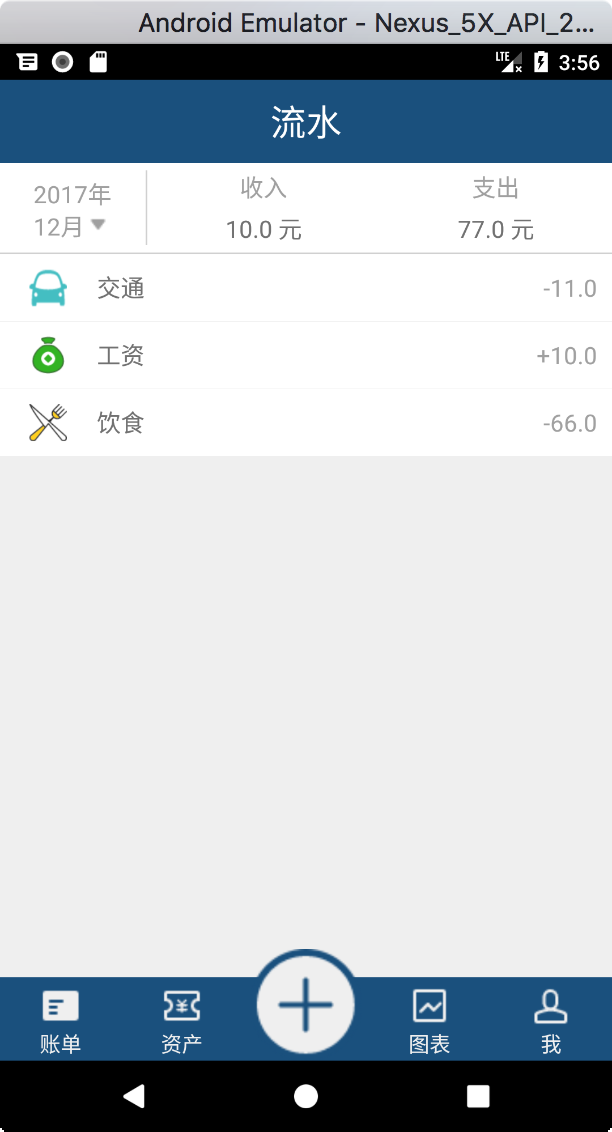
回到账单页，我们点击一条进入详情页：



编辑该账单，对时间进行修改，修改为11月：

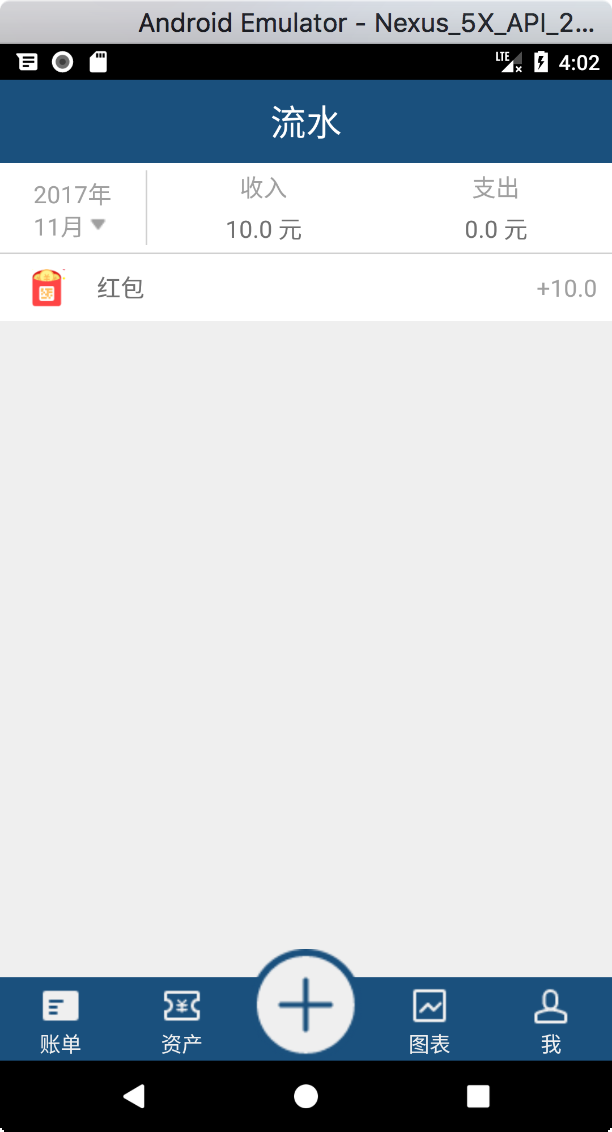


再回到主页，我们看到12月的账单已经变成了三条，此时选择时间为11月

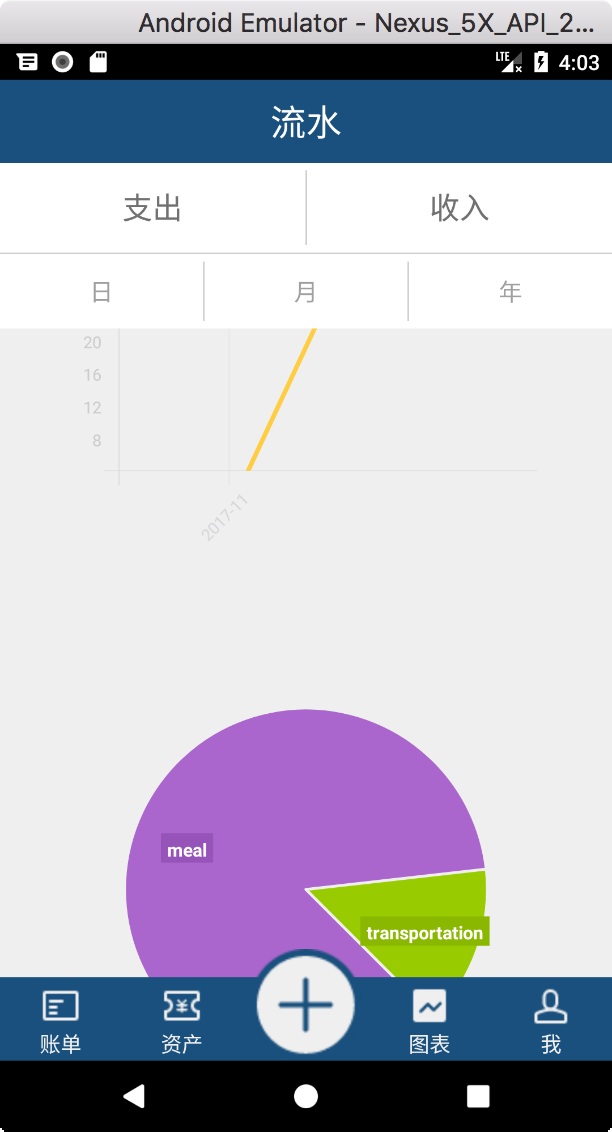




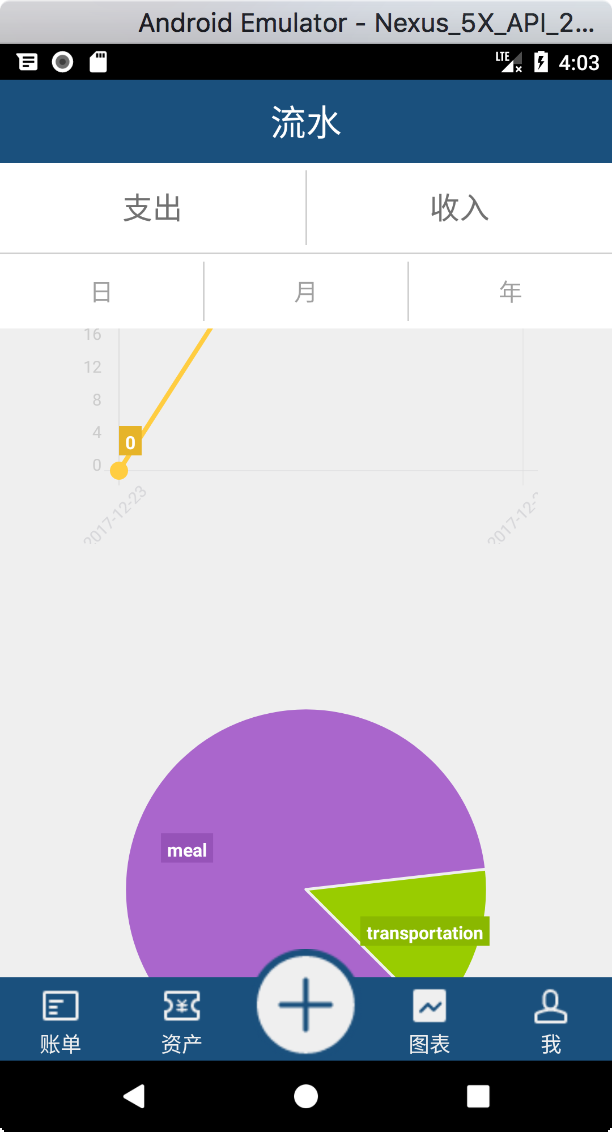
确定后账单页显示刚刚修改的那条：



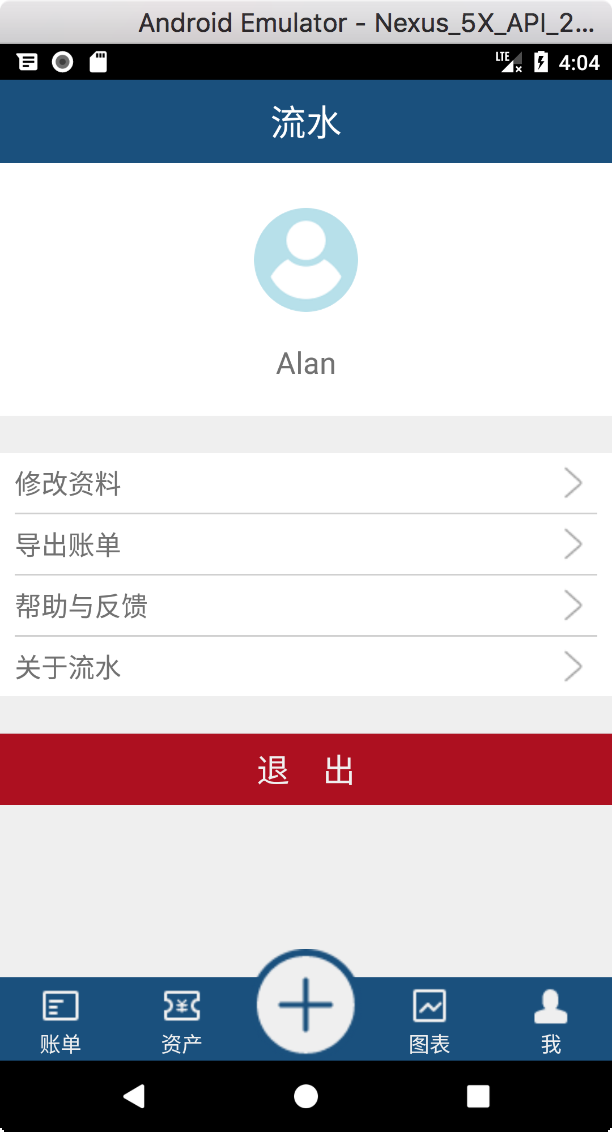
此时我们可以看看图表页的情况，对支出进行按月查看：



再对收入进行按日查看：



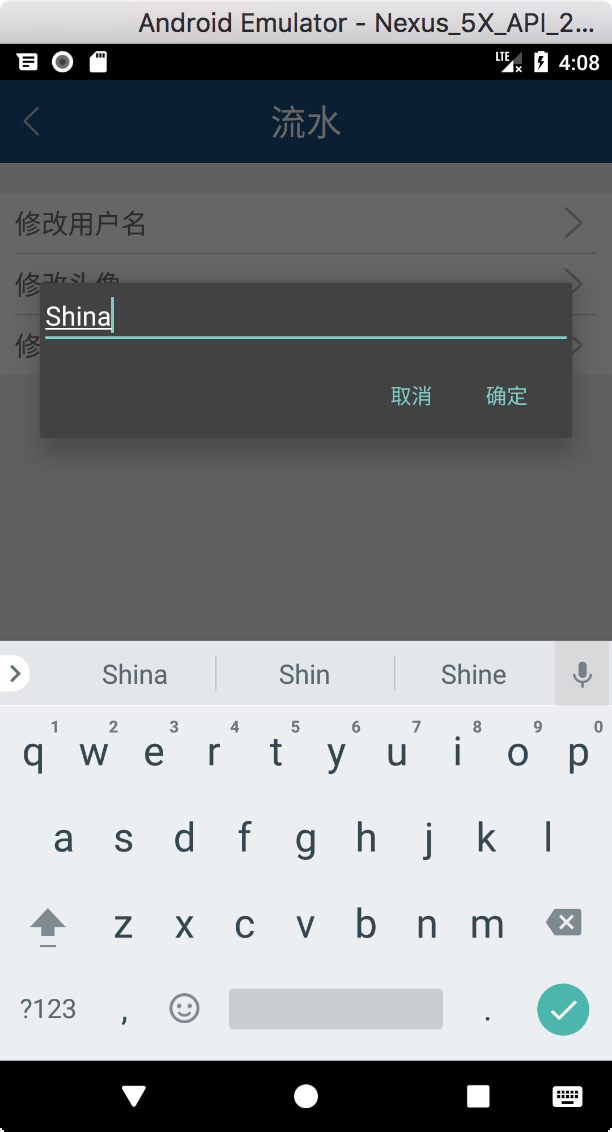
接着我们到用户页，看到自己的头像与用户名都显示在上方：



点击进入编辑页：



点击修改自己的用户名与头像：





回到用户页，可以看到已经修改成功：



这时退出，使用新的用户名进行登录：



登陆成功：

有关支付宝银行卡自动同步、账单导出与人脸识别等功能由于需要摄像头、文件管理、短信等服务，在模拟器上难以很好地展示，我们将在演示视频中进行展示。

# 项目分工

|  |  |  |  |
| --- | --- | --- | --- |
| 学号 | 姓名 | 职责 | 项目参与比 |
| 3150102098 | 沈锴 | 人脸识别实现  登陆与注册实现  图像下载接口实现  账单导出接口实现  支付宝、银行卡监听接口实现 | 33% |
| 3150103695 | 吴亦全 | 逻辑主要实现  数据库搭建与操作实现  测试 | 33% |
| 3150104650 | 王淼 | UI设计与实现  逻辑部分实现  服务器前端接口实现  报告撰写  PPT、视频制作 | 33% |

关于参与比，经过我们的讨论我们一致认为，三人的工作对于流水这款APP的诞生都是极为重要而不可缺少的，在整个开发过程中三人都是负责认真的，没有一人有打酱油的心态。开发中遇到的问题也是三人一起讨论解决，整个开发过程和谐愉快也能按照原本计划进行。因此我们将参与比平均分，各为33%。

# 总结和展望

## 总结：

经过小组三人两个多月的不懈努力，我们小组成功终于完成了所有的预期任务，使得流水这一款理财记账类APP顺利诞生。

如今，这款APP支持普通的记账，数据上传与下载，管理支付宝、银行资金的流动，消费数据可视化，消费账单生成。使得每一个用户都有独立的数据库，支持异地登录，并可以上传自己喜欢的头像，编辑个人信息等等。整个系统完整性很好。

我们完成的APP拥有简易而精美的页面设计，用户无需任何知识即可快速入手。注册过程也很方便，因此使用起来毫无障碍。

在保证方便和功能性的同时，我们的APP也有完备的安全保障系统。用户在丢失密码的时候，可以通过手机接收验证码的方式重置密码，使得用户不用担心自己的账号丢失。我们通过实现人脸识别登录，可以免除用户输入密码的烦恼并提供其更好的安全保障。

我们的每一笔记账操作都很简单，总体操作不会超过四步。以牺牲逻辑代码上的建议性换取更好的用户体验。记账会被自动备份到云端，不用担心自己的信息丢失。

我们实现了初步的理财功能。银行账户、支付宝账户的每一笔收入支出都会被我们的APP统计下来。这样的操作是自动完成的，即使APP不在前台工作。

对于记录的每一笔账，我们提供了初步的可视化，按照用户的需求生成图表，而不必尝试阅读原始数据。

当然用户如果需要导出自己的消费账单，我们可以生成excel表格，满足用户的需求。

## 展望：

我们的Stream APP如今处于测试阶段，虽然目前总体完成了，但后续还需要在以下这些部分进行完善：

我们的服务器比较简陋。这是因为我们资金不足，数据库的服务器只能购买学生版本的，人脸识别借用的是实验室的。如果后续有机会我们会让其支持更多的用户使用。

功能上目前在同步支付宝、银行卡方面有些缺陷，这是因为我们没有三证。后续如果有机会，就可以拿到相应的API，这一部分就可以做的更好。

后续我们还会对同步进行优化，尽可能减小用户的手机资源的消耗。

我们目前因为时间原因没有使用GPS模块，未来会记录用户每一笔账单的地点，并针对地理数据做一些应用。

此外我们还会做一下并发测试。目前只能支持少量用户的并发访问，后续会在这一方面进行改进，在计算资源有限的前提下尽可能提高并发度。

我们目前只是完成了软件本身的模块，没有考虑盈利情况。未来会通过会员制度、广告推送、理财推送等方式实现盈利，当然不会过多的影响免费用户的体验。