1. 网络连接部分

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。

2. 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

2.1 注册

2.1.1 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，给用户回答。

2.1.2 注册页面设计

//TODO:页面的设计部分

2.1.3 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。

//TODO ：server的配置过程

2.1.4 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();  }   } }; |

2.1.5 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**);  } }); |

2.1.6 登录部分的实现

2.1.6.1 使用用户名和密码

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***);  }   }  }; |

2.1.7 人脸识别登录

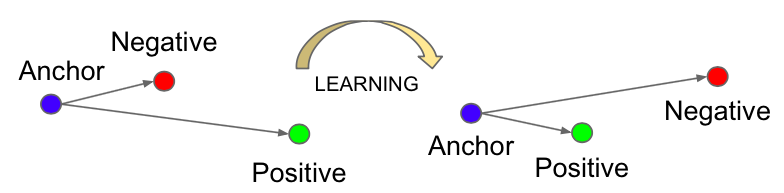
我们组想做人脸识别是因为近年来机器学习非常火热，因此有必要与时俱进。传统基于规则的图像识别成功率不高而基于机器学习的人脸识别准确率可以达到99.9%以上，可以说是非常成熟了。我们认为这种登陆方式的安全性是比传统的账号密码登陆的方式要高很多的，因此采用了它。同时我们也觉得这种安全的验证方式在后续APP的维护和更新中可以得到更为广泛的应用，因此是很有必要的。

2.1.7.1 简述

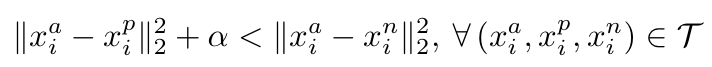
人脸识别基于深度学习，我们使用了facenet这个模型。Facenet是谷歌开发的，拥有0.992+-0.003这个惊人的准确率。该项目开源在：<https://github.com/davidsandberg/facenet.git>。

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。

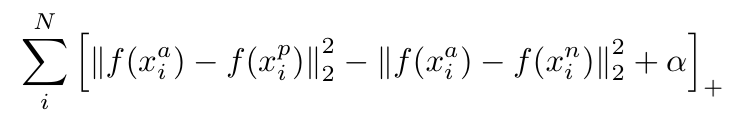
Facenet采用的是经过改良的CNN。他们替换掉了softmax，而是训练三元组之间的距离关系：



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。

2.1.7.2 服务端的搭建

服务器的搭建。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。

实现：

1. 写人脸识别部分的代码：

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** |

2. API搭建

我使用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> |

然后写一下逻辑代码：

这里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), |

3. APP端的实现

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); } |

打开相机部分：

|  |
| --- |
| **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0.88。

|  |
| --- |
| **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<0.88)  {  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**; } |

3. 导出账单部分

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();  } } |

4. 统计图的生成

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。

4.1 选择数据

// TODO

|  |
| --- |
| **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]);  }  }  } } |

4.2 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* }    } |

4.3 绘制部分

4.3.1 画饼状图

|  |
| --- |
| **private void** drawPieChart() {  **mPieChartView**.setPieChartData(**mPieChartData**);  **mPieChartView**.setCircleFillRatio(0.9f);*//设置放大缩小范围* } |

4.3.2 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);  } |

5. 同步支付宝、银行卡

这一部分的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。

5.1 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);  *// TO DO* **try** {  UpdateData.*addBills*(Double.*parseDouble*(amount),(isIn?**"in"**:**"out"**),**"Jianhang"**);  } **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();  }  }  } } |

5.2 支付宝同步的实现

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。

5.2.1 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**> |

5.2.2 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()  {   } } |

5.2 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。

5.2.1 用户图片的API

用户头像的API位于阿里云服务器上，使用Django搭建。

Html文件：

|  |
| --- |
| <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文件：

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

这里做的是绑定工作：

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

5.2.2 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;  } } |