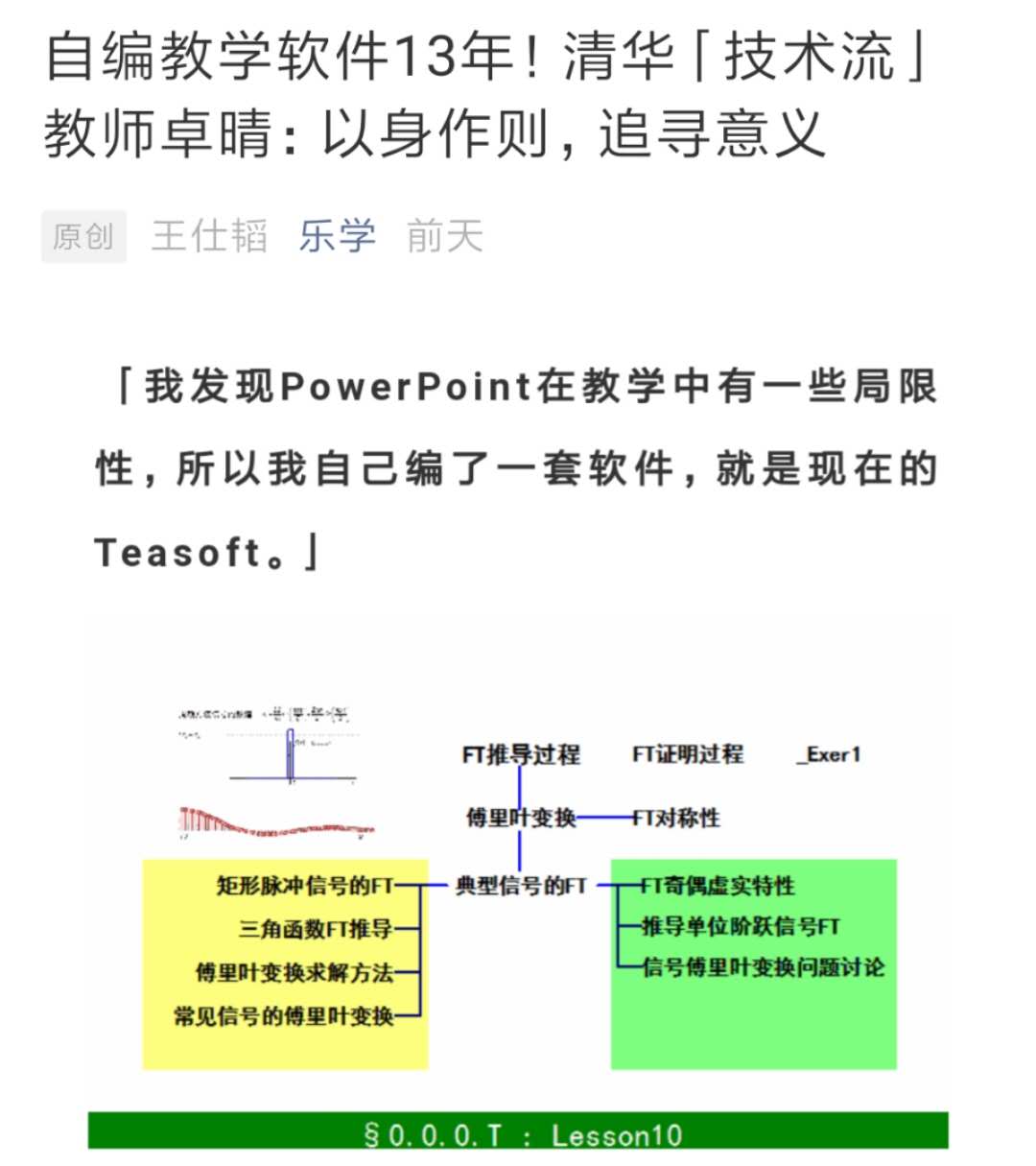
已经开学五个星期了，想必大家对各个课程的老师都逐渐熟悉了起来，每个老师都有不同的教学风格。但想必有一个技术流老师让大家印象最为深刻：

**没错，他就是——信号与系统老师 卓晴！**



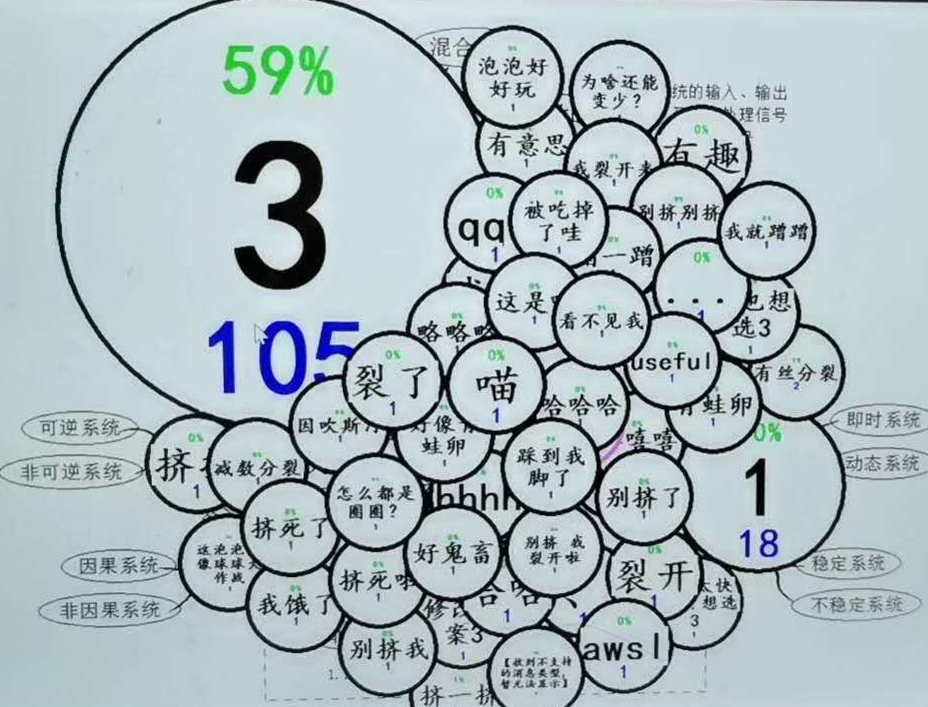
来源：乐学公众号

卓晴老师让人印象最深刻的地方

是~~作业好多好难…~~

是~~令人头晕目眩的PPT…~~

咳咳，当然是TsinghuaJoking公众号强大的互动功能！！



某次“激烈”的答题场景

那么，它到底是如何实现的呢？如何获得发向公众号后台的消息呢？这就是我们今天要讨论的主题了。

其实，在当今万物皆可Python的时代，大家想必也都猜到了答案。

没错，我们这次就来学习Web编程框架-Flask的基本使用！

注：其实别的编程语言也可以，不过Flask可以大大简化我们的编程工作。



当然如果觉得技术内容太长，你也可以直接拉到最后，点击原文链接可以直接“调戏”公众号，并且参与一个爆肝度的调查活动，只要输入信息，就可以与**所有填过的人进行对比**。看看**你能打败多少人**。



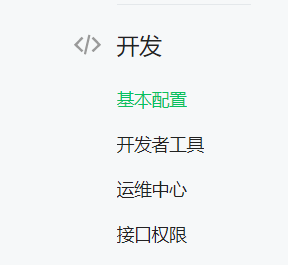
# 准备工作

首先，为了让你的公众号数据能够被实时收集，你需要一个能在公网运行的服务器。服务器有很多来源，有阿里巴巴的阿里云，腾讯的腾讯云。这一环节，还是要做出一点点金钱上的牺牲的。不过其实也不贵，我们可以租学生机。



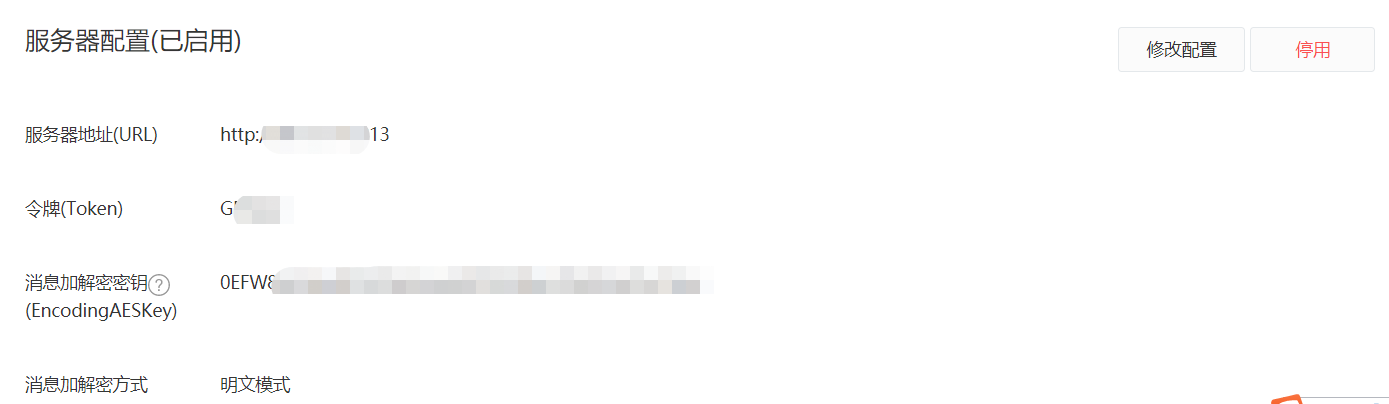
阿里云学生机：总之比B站大会员便宜

有了服务器之后，我们就得到了这个服务器的地址。搞到服务器之后，我们还需要做的一件事是将服务器与自己的公众号连接配对，这时我们需要登录微信公众平台，登录自己的公众号，左边菜单下滑到底，选择基本配置：



看到右边有 公众号开发信息、服务器配置、已绑定的微信开放平台帐号 三个栏目。选择 服务器配置 - 修改配置：

其中服务器地址就填之前服务器的地址，Token可以起一个自己喜欢的名字，秘钥随机生成即可。别忘记改完这些配置后一定要点击右上角的“启用”（如果无法启用，那么等到编写好Flask程序并运行之后点击启用）。启用成功后应该是这个样子：



以上，准备工作结束。

# Flask连接基本配置

接下来我们就正式进入了利用Flask进行编程的阶段。Flask可以按照python正常的pip/conda安装模式进行安装。

首先我们进行Flask应用的基本初始化。

//代码块开始（希望排版时要作出代码块的效果，秀米上应该有，后面的代码也希望用代码块的方式，因为截图比较丑而且不清晰，这里不是图片，是保留了原格式，代码可复制）

from flask import Flask, request, abort, render\_template  
import hashlib  
import xmltodict  
import time # 导入一些必要的包  
  
from wechatpy.replies import ImageReply  
from wechatpy import parse\_message # 微信提供的工具  
  
from myapp import response # 自己编写的应用，将另一个脚本中编写的应用导入  
  
# 微信的token令牌，注意修改成自己的  
WECHAT\_TOKEN = 'MyToken'  
app = Flask(\_\_name\_\_)  
@app.route("/", methods=["POST","GET"])

接下来我们就正式开始编写一个用于微信公众号后台信息处理的程序。这部分可能会涉及一些计算机通信协议的知识，大家不理解也没关系，因为以下基本就是一套程序模板，可以直接用。（但如果想要更全面的开发，还是需要这方面知识的）有关微信公众号的开发，有官方文档，在公众号管理页面下面“开发”一栏内的“开发者工具”之内。

在应用的开始，首先，我们需要获取三个信息，因为这涉及我们的Web程序和微信公众平台之间的传输协议。

def wechat():  
 *"""验证服务器地址的有效性"""* # 开发者提交信息后，微信服务器将发送GET请求到填写的服务器地址URL上，GET请求携带四个参数:  
 # signature:微信加密, signature结合了开发者填写的token参数和请求中的timestamp参数 nonce参数  
 # timestamp:时间戳(chuo这是拼音)  
 # nonce: 随机数  
 # echostr: 随机字符串  
 # 接收微信服务器发送参数  
 signature = request.args.get("signature")  
 timestamp = request.args.get("timestamp")  
 nonce = request.args.get("nonce")

将 token、timestamp、nonce 三个参数进行字典序排序并加密，之后进行对比，标识该请求来源于微信。

# 将token、timestamp、nonce三个参数进行字典序排序  
# 将三个参数字符串拼接成一个字符串进行sha1加密  
# 开发者获得加密后的字符串可与signature对比，标识该请求来源于微信  
if not all([signature, timestamp, nonce]):  
 # 抛出400错误  
 abort(400)  
  
# 按照微信的流程计算签名  
li = [WECHAT\_TOKEN, timestamp, nonce]  
# 排序  
li.sort()  
# 拼接字符串  
tmp\_str = "".join(li)  
tmp\_str = tmp\_str.encode('utf-8')  
  
# 进行sha1加密, 得到正确的签名值  
sign = hashlib.sha1(tmp\_str).hexdigest()  
  
# 将自己计算的签名值, 与请求的签名参数进行对比, 如果相同, 则证明请求来自微信  
if signature != sign:  
 # 代表请求不是来自微信  
 # 弹出报错信息, 身份有问题  
 abort(403)  
else:  
 # 表示是微信发送的请求

确认了是微信公众号的消息请求之后，我们就可以进行信息处理了：

微信第一次接入时会发出一个GET请求，我们需要按照流程正确回应：

if request.method == "GET":  
 # 表示第一次接入微信服务器的验证  
 echostr = request.args.get("echostr")  
 # 校验echostr  
 if not echostr:  
 abort(400)  
 return echostr

普通用户向公众号发消息是一种POST请求。这里其实很有意思，我们所发的文本、图片、语音，在这种Web应用中都不仅仅是一个文件的形式。而是都经过一定的规则编码成了xml格式。所以我们需要对这种请求进行解码：

elif request.method == "POST":  
 # 表示微信服务器转发消息过来  
 # 拿去xml的请求数据  
 xml\_str = request.data  
  
 # 当xml\_str为空时  
 if not xml\_str:  
 abort(400)  
  
 # 对xml字符串进行解析成字典  
 xml\_dict = xmltodict.parse(xml\_str)  
  
 xml\_dict = xml\_dict.get("xml")  
 # print(xml\_dict)  
 # MsgType是消息类型 这里是提取消息类型  
 msg\_type = xml\_dict.get("MsgType")

解码出xml文件和消息类型后，我们就可以有针对性的进行回复：

如果消息是文本类型：

if msg\_type == "text":  
 # 表示发送文本消息  
 # 够造返回值, 经由微信服务器回复给用户的消息内容  
 # 回复消息  
 # ToUsername: (必须传) 接收方账号(收到的OpenID)  
 # FromUserName: (必须传) 开发者微信号  
 # CreateTime: (必须传) 消息创建时间(整形)  
 # MsgType: (必须传) 消息类型  
 # Content: (必须传) 回复消息的内容(换行:在Content中能够换行, 微信客户端就支持换行显示)  
  
 # 我们自己的消息处理逻辑  
 user\_name = xml\_dict.get("FromUserName")  
 text = xml\_dict.get("Content")  
 print("text:", text)  
  
 reply = response(user\_name, text) # 我们自己的消息处理逻辑  
  
 resp\_dict = { # 用字典形式表示xml文件，之后可以利用工具包进行转换  
 "xml":{  
 "ToUserName":xml\_dict.get("FromUserName"),  
 "FromUserName":xml\_dict.get("ToUserName"),  
 "CreateTime":int(time.time()),  
 "MsgType":"text",  
 "Content":reply  
 }  
 }

可以看出，因为我们需要返回一个xml文件，我们先用dict字典形式表示我们要返回的东西，比较清楚。我们自己的消息处理逻辑中，我们可以利用用户名与用户发来的文本信息进行处理，假如这个函数直接返回发来的text，那么整体就应该是一个“复读机”程序。



可是，框架到目前为止还不能结束，因为我们还有可能接收到图片、语音等信息。对于其他信息我们不做处理，返回一个文本信息，如代码所示：

else:  
 if msg\_type == 'image':  
 msg = parse\_message(xml\_str)  
 media\_id = '6QMxv1WgvAmt\_9YJMA9zgmG2QUnr-8M2xPErDHCllWrbvmM\_YASURaPS0rTDewta'  
 reply = ImageReply(media\_id=media\_id, message=msg)  
 xml = reply.render()  
 return xml  
  
 resp\_dict = {  
 "xml": {  
 "ToUserName": xml\_dict.get("FromUserName"),  
 "FromUserName": xml\_dict.get("ToUserName"),  
 "CreateTime": int(time.time()),  
 "MsgType": "text",  
 "Content": "对不起，不能识别您发的内容！"  
 }  
 }

对于图片信息我们返回一个已经存在的图片。这个图片是我发给公众号的，在发的过程中，打印输出一下其包含的xml信息（**前提是你写完了所有的代码框架并运行**，否则你没法观察到用户发来的信息，所以这个图片处理逻辑是后加的），中间有一个项目是‘media\_id’，这个id唯一标识了微信发来的图片，将其复制过来，利用微信提供的ImageReply函数实现回复图片的功能，解析出xml文件。

所以处理图片的逻辑是，返回一张我提前设置好的图片。大家想必都宅得快疯了，我在个人公众号里设置的是一张大海的图片，让大家云赏海景：



大连市金州区金石滩附近沿岸 拍摄于2019.01.26 by李炜（ps.欢迎大家来玩）

# 实现更复杂的消息逻辑

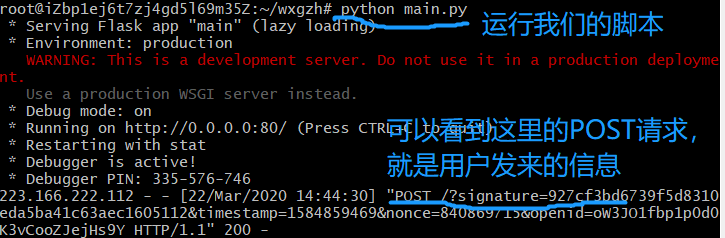
之前我们的程序只会复读。那么如何实现更复杂的消息逻辑呢？其实这一部分和Flask就没有什么太大的关系了。之前用于文本处理的代码有这么一段。事实上，这一段就是我们进行文本处理的全部。接收一个用户名，和用户发来的文本信息作为参数，输出一个返回文本信息reply。这就是一种编程的思维方式，只要控制好response函数内的代码即可。

user\_name = xml\_dict.get("FromUserName") # 得到用户名  
text = xml\_dict.get("Content") # 得到用户发来的信息  
  
reply = response(user\_name, text)

再往下就是普通的Python编程了。最后我做了一个**爆肝度调查**的程序，形式类似于**卓晴老师的小测。（只不过我们不算分）**

这里代码比较复杂就不再贴出了。

另外如果想要运行时Debug观察输出，每一次相应的输出会跟在程序收到的POST请求后面。



编写自己的程序逻辑中，比较重要的一点是异常处理。因为用户可能会发过来一些不符合格式的东西，或者不符合顺序的东西。除此之外，如果你想要的不是一个即时系统，还需要对用户发来的信息进行存储。

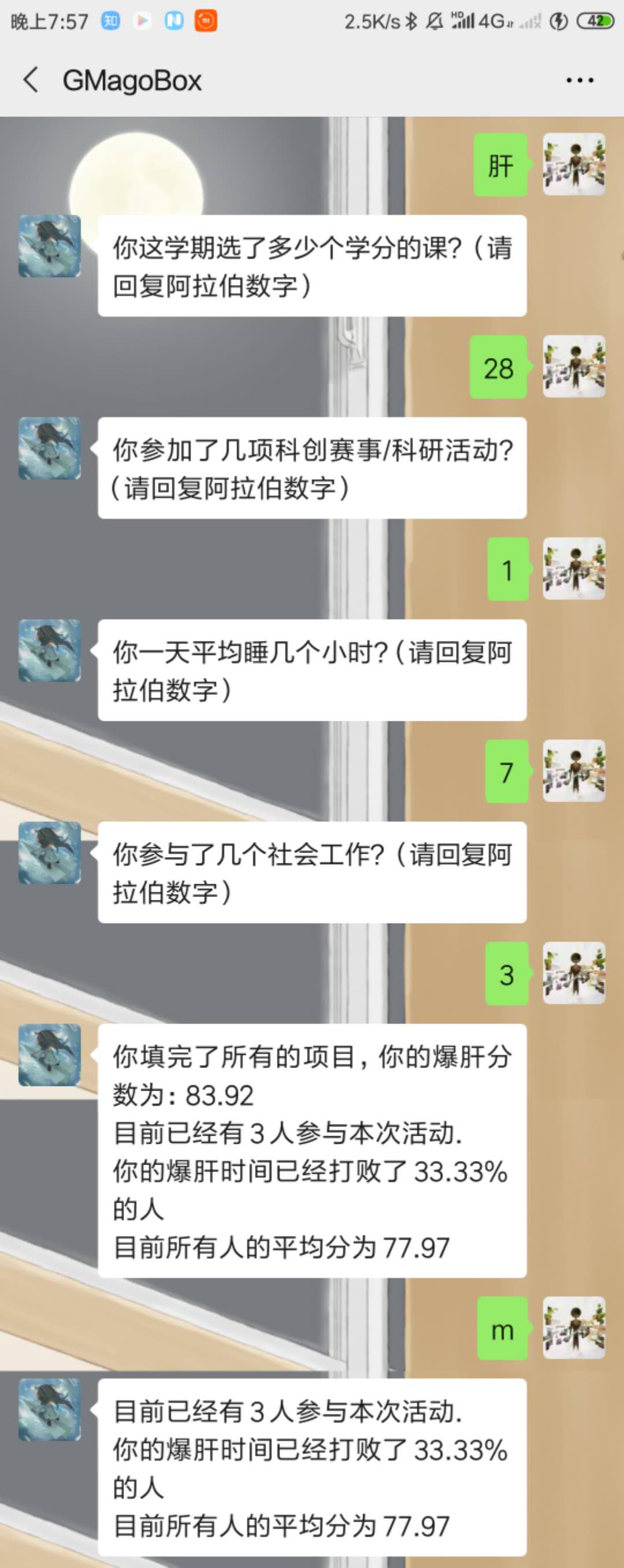
在“答题”程序中，需要引入各种异常状态去进行检验。

以下是“答题”程序中的错误流程的处理范例，大家只需要感受一下就好，因为这仅仅只是一部分逻辑。

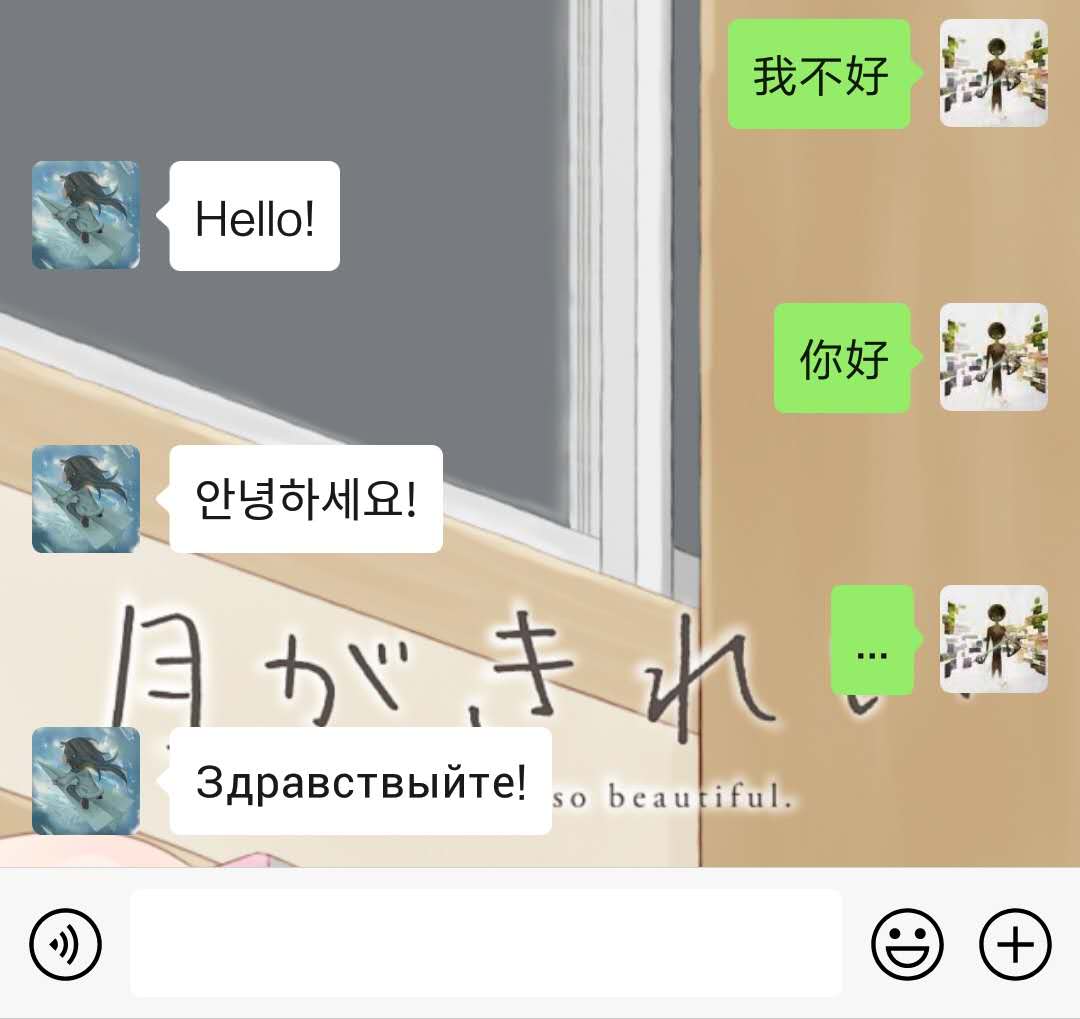
if text == enterStr: # 想进入问题  
 csvpath = 'data/' + self.user + '/gan.csv'  
 if os.path.exists(csvpath):  
 return -1 # 返回“已经参加过本项活动，不要试图重新开始”  
 else:  
 mkdir('data/' + self.user)  
 return -5 # 进入问题  
else: # 已进行到中间阶段 或 没进行到中间阶段在这里乱涂乱画的  
 csvpath = 'data/' + self.user + '/gan.csv'  
 if os.path.exists(csvpath): # 判断文件是否存在，判断进行到了第几阶段  
 df = pd.read\_csv(csvpath)  
 item\_n = 0 # 从第0列开始检测  
 for item in df.columns:  
 if df.loc[0, item] == -1:  
 break  
 else:  
 item\_n += 1  
 if item\_n == FeatureNum: # 已经填完表，就不要再捣乱了  
 return -2 # 已填完表，返回“捣乱”错误  
 return item\_n # 返回到填写的位置（即第几项）  
 else: # 没填过表，就是来捣乱的  
 return -2 # 返回“捣乱”错误

最终效果：

答题并观看结果：



输入其他信息时，随机用六种语言回复‘你好’：



以上就是关于Flask的Web编程介绍。总结为以下这样几个方面：

* 运营公众号后台处理的基本准备
* 处理公众号后台信息的框架
* 回复逻辑及其格式

点击阅读原文即可参与到以上对公众号的“调戏”中。

文案：李炜