zgrobot

发行版本 *2.0.4*

pylover

2023 年 03 月 05 日

Contents

1 入门 **3**
- 1.1 Hello World . . . 3
- 1.2 消息处理 . . . 4
- 1.3 使用 Session 记录用户状态 . . . 4
- 1.4 创建自定义菜单 . . . 5

2 例子 **7**
- 2.1 HelloWorld . . . 7
- 2.2 关注回复 . . . 8
- 2.3 快速回复图片 . . . 8
- 2.4 关键词回复 . . . 8
- 2.5 多次回复 . . . 8
- 2.6 导入其他文件 . . . 9

3 消息加解密 **11**

4 部署 **13**
- 4.1 在独立服务器上部署 . . . 13

5 Handler **17**
- 5.1 类型过滤 . . . 18
- 5.2 robot.key_click() . . . 20
- 5.3 robot.filter() . . . 21

6 Session **23**
- 6.1 开启/关闭 Session . . . 23
- 6.2 修改 Handler 以使用 Session . . . 24

7 Client **25**

7.1 开始开发 . . . 25
7.2 自定义菜单 . . . 26
7.3 消息管理 . . . 30
7.4 用户管理 . . . 36
7.5 账户管理 . . . 38
7.6 素材管理 . . . 39
7.7 用户标签管理 . . . 44
7.8 模板消息 . . . 46
7.9 返回码都是什么意思? . . . 47
7.10 48001 -- API Unauthorized . . . 47

8 Message **49**
8.1 Message 公共属性 . . . 49
8.2 TextMessage . . . 49
8.3 ImageMessage . . . 50
8.4 LinkMessage . . . 50
8.5 LocationMessage . . . 50
8.6 VoiceMessage . . . 50
8.7 VideoMessage . . . 51
8.8 UnknownMessage . . . 51

9 Event **53**
9.1 Event 公共属性 . . . 53
9.2 SubscribeEvent . . . 53
9.3 UnSubscribeEvent . . . 54
9.4 ScanEvent . . . 54
9.5 ScanCodePushEvent . . . 54
9.6 ScanCodeWaitMsgEvent . . . 54
9.7 PicSysphotoEvent . . . 55
9.8 PicPhotoOrAlbumEvent . . . 55
9.9 PicWeixinEvent . . . 55
9.10 LocationSelectEvent . . . 56
9.11 ClickEvent . . . 56
9.12 ViewEvent . . . 56
9.13 LocationEvent . . . 56
9.14 TemplateSendJobFinishEvent . . . 57
9.15 UserScanProductEvent . . . 57
9.16 UserScanProductEnterSessionEvent . . . 57
9.17 UserScanProductAsyncEvent . . . 58
9.18 UserScanProductVerifyActionEvent . . . 58
9.19 CardPassCheckEvent . . . 58
9.20 CardNotPassCheckEvent . . . 59

9.21 UserGetCardEvent . . . 59
9.22 UserGiftingCardEvent . . . 59
9.23 UserDelCardEvent . . . 60
9.24 UserConsumeCardEvent . . . 60
9.25 UserPayFromPayCellEvent . . . 60
9.26 UserViewCardEvent . . . 61
9.27 UserEnterSessionFromCardEvent . . . 61
9.28 UpdateMemberCardEvent . . . 61
9.29 CardSkuRemindEvent . . . 62
9.30 CardPayOrderEvent . . . 62
9.31 SubmitMembercardUserInfoEvent . . . 63
9.32 UnknownEvent . . . 63

10 回复 **65**
10.1 TextReply . . . 65
10.2 ImageReply . . . 66
10.3 VoiceReply . . . 66
10.4 VideoReply . . . 66
10.5 ArticlesReply . . . 67
10.6 MusicReply . . . 68
10.7 TransferCustomerServiceReply . . . 69
10.8 SuccessReply . . . 69

11 Config **71**
11.1 默认配置 . . . 72

12 与其他 Web 框架集成 **73**
12.1 FastApi . . . 73
12.2 Django . . . 74
12.3 Flask . . . 75
12.4 Bottle . . . 75
12.5 Tornado . . . 77

13 错误页面 **79**
13.1 定制错误页面 . . . 79

14 小工具 **81**
14.1 Token 生成器 . . . 81

15 贡献指南 **83**
15.1 贡献代码 . . . 83

16 API **87**
16.1 应用对象 . . . 87

16.2 配置对象 92
16.3 Session 对象 93
16.4 log 96

17 Q&A **97**
17.1 重写 get_access_token 函数来自定义 token 的存储 97

Python 模块索引 **99**

索引 **101**

ZgRobot 是一个基于 WeRoBot 而开发的微信公众号开发框架。

建议在使用前先阅读【微信开发平台官方文档】

CHAPTER 1

入门

1.1 Hello World

最简单的 Hello World 例子，会给收到的每一条信息回复 Hello World

```
import zgrobot

robot = zgrobot.ZgRoBot(token='tokenhere')

@robot.handler
def hello(message):
    return 'Hello World!'

# 让服务器监听在 0.0.0.0:80
robot.config['HOST'] = '0.0.0.0'
robot.config['PORT'] = 80
robot.run()
```

1.2 消息处理

ZgRoBot 会解析微信服务器发来的消息，并将消息转换成成*Message* 或者是*Event* 。*Message* 表示用户发来的消息，如文本消息、图片消息；*Event* 则表示用户触发的事件，如关注事件、扫描二维码事件。在消息解析、转换完成后，**ZgRoBot** 会将消息转交给*Handler* 进行处理，并将*Handler* 的返回值返回给微信服务器。

在刚才的 Hello World 中，我们编写的：

```
@robot.handler
def hello(message):
    return 'Hello World!'
```

就是一个简单的*Handler* ，`@robot.handler` 意味着 `robot` 会将所有接收到的消息（包括*Message* 和*Event* ）都转交给这个*Handler* 来处理。当然，你也可以编写一些只能处理特定消息的*Handler*：

```
# @robot.text 修饰的 Handler 只处理文本消息
@robot.text
def echo(message):
    return message.content

# @robot.image 修饰的 Handler 只处理图片消息
@robot.image
def img(message):
    return message.img
```

1.3 使用 Session 记录用户状态

ZgRoBot 提供了*Session* 功能，可以让你方便的记录用户状态。比如，这个*Handler* 可以判断发消息的用户之前有没有发送过消息

```
@robot.text
def first(message, session):
    if 'first' in session:
        return '你之前给我发过消息'
    session['first'] = True
    return '你之前没给我发过消息'
```

`Session` 功能默认开启，并使用 `SQLite` 存储 `Session` 数据。详情请参考*Session* 文档

1.4 创建自定义菜单

自定义菜单能够帮助公众号丰富界面，让用户更好更快地理解公众号的功能。Client 封装了微信的部分 API 接口，我们可以使用*create_menu()* 来创建自定义菜单。在使用 Client 之前，我们需要先提供微信公众平台内的 AppID 和 AppSecret

```
from zgrobot import ZgRoBot
robot = ZgRoBot()
robot.config["APP_ID"] = "你的 AppID"
robot.config["APP_SECRET"] = "你的 AppSecret"

client = robot.client
```

然后，我们就可以创建自定义菜单了

```
client.create_menu({
    "button":[{
          "type": "click",
          "name": "今日歌曲",
          "key": "music"
    }]
})
```

注意以上代码只需要运行一次就可以了。在创建完自定义菜单之后，我们还需要写一个*Handler* 来响应菜单的点击操作

```
@robot.key_click("music")
def music(message):
    return '你点击了“今日歌曲”按钮'
```

CHAPTER 2

例子

2.1 HelloWorld

Hello World

```
# -*- coding: utf-8 -*-

import zgrobot

robot = zgrobot.ZgRoBot(token='tokenhere')

@robot.filter("帮助")
def show_help(message):
return """
帮助
XXXXX
"""

@robot.text
    def hello_world(message):
    return 'Hello World!'

    robot.run()
```

2.2 关注回复

关注回复

```
# 关注回复
@robot.subscribe
def subscribe_reply(message):
    return "感谢这位小可爱的关注~"
```

2.3 快速回复图片

回复图片

```
# 图片回复，回复原图片
@robot.image
def img_reply(message):
    return ImageReply(message=message, media_id=message.media_id)
```

2.4 关键词回复

关键词回复

```
# 关键词回复
@robot.filter("你好呀")
def key_reply(message):
    return "你也好呀"
```

2.5 多次回复

使用客服多次回复消息

```
# 多次回复
@robot.filter("多次回复")
def reply_again(message):
    kf_account = my_client.get_custom_service_account_list().get("kf_list")
    my_client.send_text_message(user_id=message.source, content="这是客服发的消息",␣
↪kf_account=kf_account)
    return "这是第二次回复哦"
```

2.6 导入其他文件

如果你为了保证你的 bot.py （或者其他）文件的干净整洁，或者不想将所有的功能都写到一个文件里面，那么就可以将其他的功能写到其他的文件内，只在执行文件内进行机器人的创建、配置和功能的导入，你可以向下面一样：bot.py

```
# 导入其他文件，创建菜单
import menu

menu.create_menu(robot, my_client)
```

menu.py

```
import zgrobot
from zgrobot.client import Client


def create_menu(robot: zgrobot.ZgRoBot, client: Client):
    client.create_menu({
        "button": [
            {
                "type": "miniprogram",
                "name": "打卡",
                "url": "https://www.dengtadaka.com/website/index.html",
                "appid": "wxf3ca7ea27608450c",
                "pagepath": "/pages/supervise/groupDetail?id=10025"
            },
            {
                "type": "view",
                "name": "主页",
                "url": "https://mp.weixin.qq.com/s?__biz=Mzg3NzU1MjgxOA==&
↪mid=100002639&idx=1&sn"
                       "=3d7e9959038e0a3d01fa3f76ba715200&chksm"

↪"=4f207e927857f78482f85af737e719c3eef35afcba6e58510bc65a6296284c2999fda51babf0#rd "
            },
            {
                "type": "click",
                "name": "帮助",
                "key": "help"
            }]
    })

    @robot.key_click("help")
```

(续下页)

(接上页)

```
    def music():
        return "现在还没有什么帮助哦~"
```

CHAPTER 3

消息加解密

ZgRoBot 支持对消息的加解密，即微信公众号的安全模式。在开启消息加解密功能之前，请先阅读微信官方的 消息加解密说明

为 **ZgRoBot** 开启消息加密功能，首先需要安装 `cryptography`

```
pip install cryptography
```

之后，你只需要将开发者 ID(`AppID`) 和微信公众平台后台设置的 `EncodingAESKey` 加到 **ZgRoBot** 的*Config* 里面就可以了

```
from zgrobot import ZgRoBot
robot = ZgRoBot()
robot.config["APP_ID"] = "Your AppID"
robot.config['ENCODING_AES_KEY'] = 'Your Encoding AES Key'
```

ZgRoBot 之后会自动进行消息的加解密工作。

CHAPTER 4

部署

备注： 本节所讨论的是将 **ZgRoBot** 作为独立服务运行情况下的部署操作。如果你希望将 **ZgRoBot** 集成到其他 **Web** 框架内，请阅读*与其他 Web 框架集成*

4.1 在独立服务器上部署

4.1.1 使用 zgrobot.run 来启动 WSGI 服务器

你可以在*Config()* 中配置好 ZgRoBot 需要监听的地址和端口号，然后使用你生成的 robot 调用*run()* 方法来启动服务器

```
import zgrobot

robot = zgrobot.ZgRoBot(token='tokenhere')

@robot.handler
def echo(message):
    return 'Hello World!'

robot.config['HOST'] = '0.0.0.0'
robot.config['PORT'] = 80
```

(续下页)

(接上页)

```
robot.run()
```

备注： 你需要 **root** 或管理员权限才能监听 1024 以下的端口。

你可以通过传递 `server` 参数来手动指定使用的服务器

```
import zgrobot

robot = zgrobot.ZgRoBot(token='tokenhere')

@robot.handler
def echo(message):
    return 'Hello World!'

robot.config['HOST'] = '0.0.0.0'
robot.config['PORT'] = 80

robot.run(server='gevent')
```

`server` 支持以下几种:

- cgi
- flup
- wsgiref
- waitress
- cherrypy
- paste
- fapws3
- tornado
- gae
- twisted
- diesel
- meinheld
- gunicorn
- eventlet

- gevent
- rocket
- bjoern
- auto

当 `server` 为 `auto` 时，**ZgRoBot** 会自动依次尝试以下几种服务器：

- Waitress
- Paste
- Twisted
- CherryPy
- WSGIRef

所以，只要你安装了相应的服务器软件，就可以使用 `zgrobot.run` 直接跑在生产环境下。

备注： server 的默认值为 `auto` 。

> **注意：** WSGIRef 的性能非常差，仅能用于开发环境。如果你要在生产环境下部署 **ZgRoBot** ，请确保你在使用其他 server 。

4.1.2 通过 WSGI HTTP Server 运行 ZgRoBot

`zgrobot.wsgi` 暴露了一个 WSGI Application , 你可以使用任何你喜欢的 WSGI HTTP Server 来部署 ZgRoBot。比如，如果你想用 Gunicorn 来部署

```
# FileName: robot.py
from zgrobot import ZgRoBot
robot = ZgRoBot()
```

那么你只需要在 Shell 下运行

```
gunicorn robot:robot.wsgi
```

就可以了。

4.1.3 使用 Supervisor 管理守护进程

请注意，`zgrobot.run` 是跑在 **非守护进程模式下**的——也就是说，一旦你关闭终端，进程就会自动退出。

我们建议您使用 Supervisor 来管理 **ZgRoBot** 的进程。

配置文件样例：

```
[program:wechat_robot]
command = python /home/<username>/robot.py
user = <username>
redirect_stderr = true
stdout_logfile = /home/<username>/logs/robot.log
stdout_errfile = /home/<username>/logs/robot_error.log
```

4.1.4 使用 Nginx 进行反向代理

微信服务器只支持 80 端口的机器人——显然，你的服务器上不会只跑着一个微信机器人。对于这种情况，我们建议您使用 Nginx 来进行反向代理。

备注： 建议新建一个子配置文件对机器人进行配置，并在主配置文件的 `server` 中添加 `include xxx/*.conf;`（配置文件的绝对路径），然后重新加载 `nginx` 服务。

配置文件样例：

```
server {
    server_name example.com;
    listen 80;

    location / {
        proxy_pass_header Server;
        proxy_redirect off;
        proxy_pass http://127.0.0.1:12233;
    }
}
```

备注： 在这个例子中，ZgRoBot 的端口号为 12233。你应该在微信管理后台中将服务器地址设为 `http://example.com` 。

更多的 `Nginx` 配置请参看 万字长文看 Nginx 配置详解!

CHAPTER 5

Handler

ZgRoBot 会将合法的请求发送给 `Handlers` 依次执行。

如果某一个 `Handler` 返回了非空值，**ZgRoBot** 就会根据这个值创建回复，后面的 `handlers` 将不会被执行。

你可以通过修饰符或*`add_handler()`* 添加 `handler`

```
import zgrobot

robot = zgrobot.ZgRoBot(token='tokenhere')

# 通过修饰符添加handler
@robot.handler
def echo(message):
    return 'Hello World!'

# 通过`add_handler`添加handler
def echo(message):
    return 'Hello World!'
robot.add_handler(echo)
```

5.1 类型过滤

在大多数情况下，一个 Handler 并不能处理所有类型的消息。幸运的是，**ZgRoBot** 可以帮你过滤收到的消息。

只想处理被新用户关注的消息?

```
import zgrobot

robot = zgrobot.ZgRoBot(token='tokenhere')

@robot.subscribe
def subscribe(message):
    return 'Hello My Friend!'

robot.run()
```

或者，你的 Handler 只能处理文本?

```
import zgrobot

robot = zgrobot.ZgRoBot(token='tokenhere')

@robot.text
def echo(message):
    return message.content

robot.run()
```

在 **ZgRobot** 中我们把请求分成了*Message* 和*Event* 两种类型, 针对两种类型的请求分别有不同的 Handler。

修饰符	类型
robot.text	文本 (Message)
robot.image	图像 (Message)
robot.location	位置 (Message)
robot.link	链接 (Message)
robot.voice	语音 (Message)
robot.unknown	未知类型 (Message)
robot.subscribe	被关注 (Event)
robot.unsubscribe	被取消关注 (Event)
robot.click	自定义菜单事件 (Event)
robot.view	链接 (Event)
robot.scancode_push	扫描推送 (Event)

续下页

表 1 – 接上页

修饰符	类型
robot.scancode_waitmsg	扫描弹消息 (Event)
robot.pic_sysphoto	弹出系统拍照发图（Event）
robot.pic_photo_or_album	弹出拍照或者相册发图（Event）
robot.pic_weixin	弹出微信相册发图器（Event）
robot.location_select	弹出地理位置选择器（Event）
robot.scan	已关注扫描二维码（Event）
robot.user_scan_product	打开商品主页事件推送（Event）
robot.user_scan_product_enter_session	进入公众号事件推送（Event）
robot.user_scan_product_async	地理位置信息异步推送（Event）
robot.user_scan_product_verify_action	商品审核结果推送（Event）
robot.card_pass_check	卡券通过审核 (Event)
robot.card_not_pass_check	卡券未通过审核 (Event)
robot.user_get_card	用户领取卡券 (Event)
robot.user_gifting_card	用户转赠卡券 (Event)
robot.user_del_card	用户删除卡券 (Event)
robot.user_consume_card	卡券被核销 (Event)
robot.user_pay_from_pay_cell	微信买单完成 (Event)
robot.user_view_card	用户进入会员卡 (Event)
robot.user_enter_session_from_card	用户卡券里点击查看公众号进入会话 (Event)
robot.update_member_card	会员卡积分余额发生变动 (Event)
robot.card_sku_remind	库存警告 (Event)
robot.card_pay_order	券点发生变动 (Event)
robot.templatesendjobfinish_event	模板信息推送事件 (Event)
robot.submit_membercard_user_info	激活卡券 (Event)
robot.location_event	上报位置 (Event)
robot.unknown_event	未知类型 (Event)

额，这个 `handler` 想处理文本信息和地理位置信息？

```
import zgrobot

robot = zgrobot.ZgRoBot(token='tokenhere')

@robot.text
@robot.location
def handler(message):
    # Do what you love to do
    pass

robot.run()
```

当然，你也可以用*add_handler()* 函数添加 handler，就像这样:

```
import zgrobot

robot = zgrobot.ZgRoBot(token='tokenhere')

def handler(message):
    # Do what you love to do
    pass

robot.add_handler(handler, type='text')
robot.add_handler(handler, type='location')

robot.run()
```

备注: 通过*add_handler()* 添加的 handler 将收到所有信息，并且只有在其它 handler 没有给出返回值的情况下，通过*add_handler()* 添加的 handler 才会被调用。

5.2 robot.key_click()

你可以使用*key_click()* 回应自定义菜单。

key_click() 是对*click()* 修饰符的改进。

如果你在自定义菜单中定义了一个 Key 为 `abort` 的菜单，响应这个菜单的 `handler` 可以写成这样

```
@robot.key_click("abort")
def abort():
    return "I'm a robot"
```

当然，如果你不喜欢用*key_click()* ，也可以写成这样

```
@robot.click
def abort(message):
    if message.key == "abort":
        return "I'm a robot"
```

两者是等价的。

5.3 robot.filter()

你可以使用*filter()* 回应有指定文本的消息

filter() 是对*text()* 修饰符的改进。

现在你可以写这样的代码

```
@robot.filter("a")
def a():
    return "正文为 a "

import re


@robot.filter(re.compile(".*?bb.*?"))
def b():
    return "正文中含有 bb "

@robot.filter(re.compile(".*?c.*?"), "d")
def c():
    return "正文中含有 c 或正文为 d"

@robot.filter(re.compile("(.*)?e(.*)?"), "f")
def d(message, session, match):
    if match:
        return "正文为 " + match.group(1) + "e" + match.group(2)
    return "正文为 f"
```

这段代码等价于

```
@robot.text
def a(message):
    if message.content == "a":
        return "正文为 a "
import re


@robot.text
def b(message):
    if re.compile(".*?bb.*?").match(message.content):
        return "正文中含有 b "

@robot.text
def c(message):
```

(续下页)

(接上页)

```
    if re.compile(".*?c.*?").match(message.content) or message.content == "d":
        return "正文中含有 c 或正文为 d"

@robot.text
def d(message):
    match = re.compile("(.*)?e(.*)?").match(message.content)
    if match:
        return "正文为 " + match.group(1) + "e" + match.group(2)
    if  message.content == "f":
        return "正文为 f"
```

如果你想通过修饰符以外的方法添加 filter，可以使用*add_filter()* 方法

```
def say_hello():
    return "hello!"

robot.add_filter(func=say_hello, rules=["hello", "hi", re.compile(".*?hello.*?")])
```

更多内容详见*BaseRoBot()*

CHAPTER 6

Session

你可以通过 Session 实现用户状态的记录。

一个简单的使用 Session 的 Demo

```
from zgrobot import ZgRoBot
robot = ZgRoBot(token=zgrobot.utils.generate_token())

@robot.text
def first(message, session):
    if 'last' in session:
        return
    session['last'] = message.content
    return message.content

robot.run()
```

6.1 开启/关闭 Session

Session 在 **ZgRoBot** 中默认开启，并使用*SQLiteStorage()* 作为存储后端。如果想要更换存储后端，可以修改*Config* 中的 SESSION_STORAGE 值：

```
from zgrobot import ZgRoBot
from zgrobot.session.filestorage import FileStorage
```

(续下页)

(接上页)

```
robot = ZgRoBot(token="token")
robot.config['SESSION_STORAGE'] = FileStorage()
```

如果想要关闭 Session 功能，只需把 SESSION_STORAGE 设为 False 即可

```
from zgrobot import ZgRoBot
robot = ZgRoBot(token="token")
robot.config['SESSION_STORAGE'] = False
```

6.2 修改 Handler 以使用 Session

没有打开 Session 的时候，一个标准的 **ZgRoBot** Handler 应该是这样的

```
@robot.text
def hello(message):
    return "Hello!"
```

而在打开 Session 之后，如果你的 handler 不需要使用 Session，可以保持不变；如果需要使用 Session，则这个 Handler 需要修改为接受第二个参数：session

```
@robot.subscribe_event
def intro(message):
    return "Hello!"

@robot.text
def hello(message, session):
    count = session.get("count", 0) + 1
    session["count"] = count
    return "Hello! You have sent %s messages to me" % count
```

传入的 session 参数是一个标准的 Python 字典。

更多可用的 Session Storage 详见*Session 对象*。

CHAPTER 7

Client

微信 API 操作类，有部分接口暂未实现，可自行调用微信接口。

7.1 开始开发

7.1.1 获取 access token

详细请参考 微信开放文档

`Client.`**`grant_token`**`()`

获取 Access Token。

返回

返回的 JSON 数据包

正常时返回 `{"access_token":"ACCESS_TOKEN", "expires_in":7200}`

错误时返回 `{"errcode":40013,"errmsg":"invalid appid"}`

`Client.`**`get_access_token`**`()` → str

判断现有的 token 是否过期。用户需要多进程或者多机部署可以手动重写这个函数来自定义 token 的存储，刷新策略。

返回

返回 `token`

备注： Client 的操作都会自动进行 *access token* 的获取和过期刷新操作，如果有特殊需求（如多进程部署）可重写 `get_access_token`。

7.1.2 获取微信服务器 IP 地址

详细请参考 微信开放文档

`Client.get_ip_list()`

获取微信服务器 IP 地址。

返回

返回的 JSON 数据包

正常时返回 `{"ip_list": ["127.0.0.1", "127.0.0.2", "101.226.103.0/25"]}`

错误时返回 `{"errcode":40013,"errmsg":"invalid appid"}`

7.2 自定义菜单

7.2.1 自定义菜单创建接口

详细请参考 微信开放文档

`Client.create_menu(menu_data)`

创建自定义菜单:

```
client.create_menu({
    "button":[
        {
            "type":"click",
            "name":"今日歌曲",
            "key":"V1001_TODAY_MUSIC"
        },
        {
            "type":"click",
            "name":"歌手简介",
            "key":"V1001_TODAY_SINGER"
        },
        {
            "name":"菜单",
```

(续下页)

(接上页)

```
            "sub_button":[
                {
                    "type":"view",
                    "name":"搜索",
                    "url":"http://www.soso.com/"
                },
                {
                    "type":"view",
                    "name":"视频",
                    "url":"http://v.qq.com/"
                },
                {
                    "type":"click",
                    "name":"赞一下我们",
                    "key":"V1001_GOOD"
                }
            ]
        }
    ]})
```

参数

menu_data -- Python 字典

返回

返回的 JSON 数据包

正确时返回 `{"errcode":0,"errmsg":"ok"}`

错误时返回 `{"errcode":40018,"errmsg":"invalid button name size"}`

7.2.2 自定义菜单获取自定义菜单配置

详细请参考 微信开放文档

`Client.get_menu()`

查询自定义菜单。

返回

返回的 JSON 数据包

7.2.3 自定义菜单删除接口

详细请参考 微信开放文档

Client.**delete_menu**()

删除自定义菜单。

返回

返回的 JSON 数据包

正确时返回 {"errcode":0,"errmsg":"ok"}

7.2.4 自定义菜单个性化菜单接口

详细请参考 微信开放文档

Client.**create_custom_menu**(*menu_data*, *matchrule*)

创建个性化菜单:

```
button = [
    {
        "type":"click",
        "name":"今日歌曲",
        "key":"V1001_TODAY_MUSIC"
    },
    {
        "name":"菜单",
        "sub_button":[
        {
            "type":"view",
            "name":"搜索",
            "url":"http://www.soso.com/"
        },
        {
            "type":"view",
            "name":"视频",
            "url":"http://v.qq.com/"
        },
        {
            "type":"click",
            "name":"赞一下我们",
            "key":"V1001_GOOD"
        }]
 }]
 matchrule = {
```

(续下页)

(接上页)

```
    "group_id":"2",
    "sex":"1",
    "country":"中国",
    "province":"广东",
    "city":"广州",
    "client_platform_type":"2",
    "language":"zh_CN"
}
client.create_custom_menu(button, matchrule)
```

参数

- **menu_data** -- 如上所示的 Python 字典
- **matchrule** -- 如上所示的匹配规则

返回

返回的 JSON 数据包

正确时返回 `{"menuid":"208379533"}`

错误时的返回码请见接口返回码说明.

`Client.`**`delete_custom_menu`**`(`*menu_id*`)`

删除个性化菜单。

参数

menu_id -- 菜单的 ID

返回

返回的 JSON 数据包

正确时返回 `{"errcode":0,"errmsg":"ok"}`

错误时的返回码请见接口返回码说明。

`Client.`**`match_custom_menu`**`(`*user_id*`)`

测试个性化菜单匹配结果。

参数

user_id -- 要测试匹配的用户 ID，也可以是用户的微信号

返回

返回的 JSON 数据包

7.2.5 获取自定义菜单配置接口

详细请参考 微信开放文档

Client.**get_custom_menu_config**()

获取自定义菜单配置接口。

返回

返回的 JSON 数据包

7.3 消息管理

7.3.1 客服接口

详细请参考 微信开放文档

发送卡券接口暂时未支持。可自行实现。

Client.**add_custom_service_account**(*account*, *nickname*)

添加客服帐号。

参数

- **account** -- 客服账号的用户名
- **nickname** -- 客服账号的昵称

返回

返回的 JSON 数据包

Client.**update_custom_service_account**(*account*, *nickname*, *password*)

修改客服帐号。

参数

- **account** -- 客服账号的用户名
- **nickname** -- 客服账号的昵称
- **password** -- 客服账号的密码

返回

返回的 JSON 数据包

Client.**delete_custom_service_account**(*account*, *nickname*, *password*)

删除客服帐号。

参数

- **account** -- 客服账号的用户名

- **nickname** -- 客服账号的昵称
- **password** -- 客服账号的密码

返回

返回的 JSON 数据包

Client.**upload_custom_service_account_avatar**(*account*, *avatar*)

设置客服帐号的头像。

参数

- **account** -- 客服账号的用户名
- **avatar** -- 头像文件，必须是 jpg 格式

返回

返回的 JSON 数据包

Client.**get_custom_service_account_list**()

获取所有客服账号。

返回

返回的 JSON 数据包

Client.**get_online_custom_service_account_list**()

获取状态为" 在线" 的客服账号列表。

返回

返回的 JSON 数据包

Client.**send_text_message**(*user_id*, *content*, *kf_account=None*)

发送文本消息。

参数

- **user_id** -- 用户 ID 。就是你收到的 *Message* 的 source
- **content** -- 消息正文
- **kf_account** -- 发送消息的客服账户，默认值为 None，None 为不指定

返回

返回的 JSON 数据包

Client.**send_image_message**(*user_id*, *media_id*, *kf_account=None*)

发送图片消息。

参数

- **user_id** -- 用户 ID 。就是你收到的 *Message* 的 source
- **media_id** -- 图片的媒体 ID。可以通过*upload_media()* 上传。

- **kf_account** -- 发送消息的客服账户，默认值为 None，None 为不指定

返回

返回的 JSON 数据包

Client.**send_voice_message**(*user_id*, *media_id*, *kf_account=None*)

发送语音消息。

参数

- **user_id** -- 用户 ID 。就是你收到的 *Message* 的 source
- **media_id** -- 发送的语音的媒体 ID。可以通过*upload_media()* 上传。
- **kf_account** -- 发送消息的客服账户，默认值为 None，None 为不指定

返回

返回的 JSON 数据包

Client.**send_video_message**(*user_id*, *media_id*, *title=None*, *description=None*, *kf_account=None*)

发送视频消息。

参数

- **user_id** -- 用户 ID 。就是你收到的 *Message* 的 source
- **media_id** -- 发送的视频的媒体 ID。可以通过*upload_media()* 上传。
- **title** -- 视频消息的标题
- **description** -- 视频消息的描述
- **kf_account** -- 发送消息的客服账户，默认值为 None，None 为不指定

返回

返回的 JSON 数据包

Client.**send_music_message**(*user_id*, *url*, *hq_url*, *thumb_media_id*, *title=None*, *description=None*, *kf_account=None*)

发送音乐消息。注意如果你遇到了缩略图不能正常显示的问题，不要慌张；目前来看是微信服务器端的问题。对此我们也无能为力 (#197)

参数

- **user_id** -- 用户 ID 。就是你收到的 *Message* 的 source
- **url** -- 音乐链接
- **hq_url** -- 高品质音乐链接，wifi 环境优先使用该链接播放音乐
- **thumb_media_id** -- 缩略图的媒体 ID。可以通过*upload_media()* 上传。
- **title** -- 音乐标题
- **description** -- 音乐描述

- **kf_account** -- 发送消息的客服账户，默认值为 None，None 为不指定

返回

返回的 JSON 数据包

Client.**send_article_message**(*user_id*, *articles*, *kf_account=None*)

发送图文消息:

```
articles = [
    {
        "title":"Happy Day",
        "description":"Is Really A Happy Day",
        "url":"URL",
        "picurl":"PIC_URL"
    },
    {
        "title":"Happy Day",
        "description":"Is Really A Happy Day",
        "url":"URL",
        "picurl":"PIC_URL"
    }
]
client.send_acticle_message("user_id", acticles)
```

参数

- **user_id** -- 用户 ID 。就是你收到的 *Message* 的 source
- **articles** -- 一个包含至多 8 个 article 字典或 Article 对象的数组
- **kf_account** -- 发送消息的客服账户，默认值为 None，None 为不指定

返回

返回的 JSON 数据包

Client.**send_news_message**(*user_id*, *media_id*, *kf_account=None*)

发送永久素材中的图文消息。

参数

- **user_id** -- 用户 ID 。就是你收到的 *Message* 的 source
- **media_id** -- 媒体文件 ID
- **kf_account** -- 发送消息的客服账户，默认值为 None，None 为不指定

返回

返回的 JSON 数据包

Client.**send_miniprogrampage_message**(*user_id*, *title*, *appid*, *pagepath*, *thumb_media_id*, *kf_account=None*)

发送小程序卡片（要求小程序与公众号已关联）

参数

- **user_id** -- 用户 ID 。就是你收到的 *Message* 的 source
- **title** -- 小程序卡片的标题
- **appid** -- 小程序的 appid，要求小程序的 appid 需要与公众号有关联关系
- **pagepath** -- 小程序的页面路径，跟 app.json 对齐，支持参数，比如 pages/index/index?foo=bar
- **thumb_media_id** -- 小程序卡片图片的媒体 ID，小程序卡片图片建议大小为 520*416
- **kf_account** -- 需要以某个客服帐号来发消息时指定的客服账户

返回

返回的 JSON 数据包

7.3.2 群发接口

Client.**send_mass_msg**(*msg_type*, *content*, *user_list=None*, *send_ignore_reprint=False*, *client_msg_id=None*)

向指定对象群发信息。

参数

- **msg_type** -- 群发类型，图文消息为 mpnews，文本消息为 text，语音为 voice，音乐为 music，图片为 image，视频为 video，卡券为 wxcard。
- **content** -- 群发内容。
- **user_list** -- 发送对象，整型代表用户组，列表代表指定用户，如果为 None 则代表全部发送。
- **send_ignore_reprint** -- 图文消息被判定为转载时，是否继续群发。True 为继续群发（转载），False 为停止群发。该参数默认为 False。
- **client_msg_id** -- 群发时，微信后台将对 24 小时内的群发记录进行检查，如果该 clientmsgid 已经存在一条群发记录，则会拒绝本次群发请求，返回已存在的群发 msgid, 控制再 64 个字符内。

返回

返回的 JSON 数据包。

Client.**delete_mass_msg**(*msg_id*, *article_idx=0*)

群发之后，随时可以通过该接口删除群发。

参数

- **msg_id** -- 发送出去的消息 ID。
- **article_idx** -- 要删除的文章在图文消息中的位置，第一篇编号为 1，该字段不填或填 0 会删除全部文章。

返回

微信返回的 json 数据。

Client.**send_mass_preview_to_user**(*msg_type*, *content*, *user*, *user_type='openid'*)

开发者可通过该接口发送消息给指定用户，在手机端查看消息的样式和排版。为了满足第三方平台开发者的需求，在保留对 openID 预览能力的同时，增加了对指定微信号发送预览的能力，但该能力每日调用次数有限制（100 次），请勿滥用。

参数

- **user_type** -- 预览对象，*openid* 代表以 openid 发送，*wxname* 代表以微信号发送。
- **msg_type** -- 发送类型，图文消息为 mpnews，文本消息为 text，语音为 voice，音乐为 music，图片为 image，视频为 video，卡券为 wxcard。
- **content** -- 预览内容。
- **user** -- 预览用户。

返回

返回的 json。

Client.**get_mass_msg_status**(*msg_id*)

查询群发消息发送状态。

参数

msg_id -- 群发消息后返回的消息 id。

返回

返回的 json。

Client.**get_mass_msg_speed**()

获取群发速度。

返回

返回的 json。

7.4 用户管理

7.4.1 用户分组管理

详细请参考 微信开放文档

Client.**create_group**(*name*)

创建分组。

参数

name -- 分组名字（30 个字符以内）

返回

返回的 JSON 数据包

Client.**get_groups**()

查询所有分组。

返回

返回的 JSON 数据包

Client.**get_group_by_id**(*openid*)

查询用户所在分组。

参数

openid -- 用户的 OpenID

返回

返回的 JSON 数据包

Client.**update_group**(*group_id*, *name*)

修改分组名。

参数

- **group_id** -- 分组 ID，由微信分配
- **name** -- 分组名字（30 个字符以内）

返回

返回的 JSON 数据包

Client.**move_user**(*user_id*, *group_id*)

移动用户分组。

参数

- **user_id** -- 用户 ID，即收到的 *Message* 的 source
- **group_id** -- 分组 ID

返回

返回的 JSON 数据包

Client.**move_users**(*user_id_list*, *group_id*)

批量移动用户分组。

参数

- **user_id_list** -- 用户 ID 的列表（长度不能超过 50）
- **group_id** -- 分组 ID

返回

返回的 JSON 数据包

Client.**delete_group**(*group_id*)

删除分组。

参数

group_id -- 要删除的分组的 ID

返回

返回的 JSON 数据包

7.4.2 设置备注名

详细请参考 微信开放文档

Client.**remark_user**(*user_id*, *remark*)

设置备注名。

参数

- **user_id** -- 设置备注名的用户 ID
- **remark** -- 新的备注名，长度必须小于 30 字符

返回

返回的 JSON 数据包

7.4.3 获取用户基本信息

详细请参考 微信开放文档

Client.**get_user_info**(*user_id*, *lang='zh_CN'*)

获取用户基本信息。

参数

- **user_id** -- 用户 ID 。就是你收到的 *Message* 的 source

- **lang** -- 返回国家地区语言版本，zh_CN 简体，zh_TW 繁体，en 英语

返回

返回的 JSON 数据包

Client.**get_users_info**(*user_id_list*, *lang='zh_CN'*)

批量获取用户基本信息。

参数

- **user_id_list** -- 用户 ID 的列表
- **lang** -- 返回国家地区语言版本，zh_CN 简体，zh_TW 繁体，en 英语

返回

返回的 JSON 数据包

7.5 账户管理

长链接转短链接接口和微信认证事件推送暂未添加，可自行实现。

7.5.1 生成带参数的二维码

详细请参考 微信开放文档

Client.**create_qrcode**(*data*)

创建二维码。

参数

data -- 你要发送的参数 dict

返回

返回的 JSON 数据包

Client.**show_qrcode**(*ticket*)

通过 ticket 换取二维码。

参数

ticket -- 二维码 ticket 。可以通过*create_qrcode()* 获取到

返回

返回的 Request 对象

7.5.2 获取用户列表

详细请参考 微信开放文档

Client.**get_followers**(*first_user_id=None*)

获取关注者列表详情请参考 https://developers.weixin.qq.com/doc/offiaccount/User_Management/Getting_a_User_List.html

参数

first_user_id -- 可选。第一个拉取的 OPENID，不填默认从头开始拉取

返回

返回的 JSON 数据包

7.6 素材管理

7.6.1 新增临时素材

详细请参考 微信开放文档

Client.**upload_media**(*media_type*, *media_file*)

上传临时多媒体文件。

参数

- **media_type** -- 媒体文件类型，分别有图片（image）、语音（voice）、视频（video）和缩略图（thumb）
- **media_file** -- 要上传的文件，一个 File-object: open('xxx', 'rb')

返回

返回的 JSON 数据包

7.6.2 获取临时素材

详细请参考 微信开放文档

Client.**download_media**(*media_id*)

下载临时多媒体文件。

参数

media_id -- 媒体文件 ID

返回

requests 的 Response 实例

7.6.3 新增永久素材

详细请参考 微信开放文档

Client.**add_news**(*articles*)

新增永久图文素材:

```
articles = [{
   "title": TITLE,
   "thumb_media_id": THUMB_MEDIA_ID,
   "author": AUTHOR,
   "digest": DIGEST,
   "show_cover_pic": SHOW_COVER_PIC(0 / 1),
   "content": CONTENT,
   "content_source_url": CONTENT_SOURCE_URL
}
# 若新增的是多图文素材，则此处应有几段articles结构，最多8段
]
client.add_news(articles)
```

参数

articles -- 如示例中的数组

返回

返回的 JSON 数据包

Client.**upload_news_picture**(*file*)

上传图文消息内的图片。

参数

file -- 要上传的图片文件，类型仅支持 jpg/png 格式，大小限制为 <10Mb，如：open('xxx', 'rb')

返回

返回的 JSON 数据包

Client.**upload_permanent_media**(*media_type*, *media_file*)

上传其他类型永久素材。

参数

- **media_type** -- 媒体文件类型，分别有图片（image）、语音（voice）和缩略图（thumb）
- **media_file** -- 要上传的文件，一个 File-object: open('xxx', 'rb')

返回

返回的 JSON 数据包

Client.**upload_permanent_video**(*title*, *introduction*, *video*)

上传永久视频。

参数

- **title** -- 视频素材的标题
- **introduction** -- 视频素材的描述
- **video** -- 要上传的视频，一个 File-object: open('xxx', 'rb')，大小限制 <10Mb

返回

requests 的 Response 实例

7.6.4 获取永久素材

详细请参考 微信开放文档

Client.**download_permanent_media**(*media_id*)

获取永久素材。

参数

media_id -- 媒体文件 ID

返回

requests 的 Response 实例

7.6.5 删除永久素材

详细请参考 微信开放文档

Client.**delete_permanent_media**(*media_id*)

删除永久素材。

参数

media_id -- 媒体文件 ID

返回

返回的 JSON 数据包

7.6.6 上传图文消息素材

Client.**upload_news**(*articles*)

上传图文消息素材

```
articles = [{
    "thumb_media_id":"qI6_Ze_6PtV7svjolgs-rN6stStuHIjs9_DidOHaj0Q-
↪mwvBelOXCFZiq2OsIU-p",
    "author":"xxx",
    "title":"Happy Day",
    "content_source_url":"www.qq.com",
    "content":"content",
    "digest":"digest",
    "show_cover_pic":1,
    "need_open_comment":1,
    "only_fans_can_comment":1
}]
```

具体请参考: https://developers.weixin.qq.com/doc/offiaccount/Message_Management/Batch_Sends_and_Originality_Checks.html#1

参数

articles -- 上传的图文消息数据。

返回

返回的 JSON 数据包。

7.6.7 修改永久图文素材

详细请参考 微信开放文档

Client.**update_news**(*update_data*)

修改永久图文素材:

```
update_data = {
    "media_id":MEDIA_ID,
    "index":INDEX,
    "articles": {
        "title": TITLE,
        "thumb_media_id": THUMB_MEDIA_ID,
        "author": AUTHOR,
        "digest": DIGEST,
        "show_cover_pic": SHOW_COVER_PIC(0 / 1),
        "content": CONTENT,
        "content_source_url": CONTENT_SOURCE_URL
```

(续下页)

(接上页)

```
    }
}
client.update_news(update_data)
```

参数

update_data -- 更新的数据，要包含 media_id（图文素材的 ID），index（要更新的文章在图文消息中的位置），articles（新的图文素材数据）

返回

返回的 JSON 数据包

7.6.8 获取素材总数

详细请参考 微信开放文档

Client.**get_media_count**()

获取素材总数。

返回

返回的 JSON 数据包

7.6.9 获取素材列表

详细请参考 微信开放文档

Client.**get_media_list**(*media_type*, *offset*, *count*)

获取素材列表。

参数

- **media_type** -- 素材的类型，图片（image）、视频（video）、语音（voice）、图文（news）
- **offset** -- 从全部素材的该偏移位置开始返回，0 表示从第一个素材返回
- **count** -- 返回素材的数量，取值在 1 到 20 之间

返回

返回的 JSON 数据包

7.7 用户标签管理

详细请参考 微信开放文档

7.7.1 创建标签

Client.**create_tag**(*tag_name*)

创建一个新标签

参数

tag_name -- 标签名

返回

返回的 JSON 数据包

7.7.2 获取公众号已创建的标签

Client.**get_tags**()

获取已经存在的标签

返回

返回的 JSON 数据包

7.7.3 编辑标签

Client.**update_tag**(*tag_id*, *tag_name*)

修改标签

参数

- **tag_id** -- 标签 ID
- **tag_name** -- 新的标签名

返回

返回的 JSON 数据包

7.7.4 删除标签

Client.**delete_tag**(*tag_id*)

删除标签

参数

tag_id -- 标签 ID

返回

返回的 JSON 数据包

7.7.5 获取标签下粉丝列表

Client.**get_users_by_tag**(*tag_id*, *next_open_id=''*)

获取标签下粉丝列表

参数

- **tag_id** -- 标签 ID
- **next_open_id** -- 第一个拉取用户的 OPENID，默认从头开始拉取

返回

返回的 JSON 数据包

7.7.6 批量为用户打标签

Client.**tag_users**(*tag_id*, *open_id_list*)

批量为用户打标签

参数

- **tag_id** -- 标签 ID
- **open_id_list** -- 包含一个或多个用户的 OPENID 的列表

返回

返回的 JSON 数据包

7.7.7 批量为用户取消标签

Client.**untag_users**(*tag_id*, *open_id_list*)

批量为用户取消标签

参数

- **tag_id** -- 标签 ID
- **open_id_list** -- 包含一个或多个用户的 OPENID 的列表

返回

返回的 JSON 数据包

7.7.8 获取用户身上的标签列表

Client.**get_tags_by_user**(*open_id*)

获取用户身上的标签列表

参数

open_id -- 用户的 OPENID

返回

返回的 JSON 数据包

7.8 模板消息

Client.**send_template_message**(*user_id*, *template_id*, *data*, *url=''*, *miniprogram=None*)

发送模板消息详情请参考 https://developers.weixin.qq.com/doc/offiaccount/Message_Management/Template_Message_Interface.html

参数

- **user_id** -- 用户 ID 。就是你收到的 *Message* 的 source
- **template_id** -- 模板 ID。
- **data** -- 用于渲染模板的数据。
- **url** -- 模板消息的可选链接。
- **miniprogram** -- 跳小程序所需数据的可选数据。

返回

返回的 JSON 数据包

7.9 返回码都是什么意思？

详细请参考 微信开放文档

7.10 48001 -- API Unauthorized

如果你遇到了这个错误，请检查你的微信公众号是否有调用该接口的权限。详细请参考 微信开放文档

CHAPTER 8

Message

8.1 Message 公共属性

除了 UnknownMessage，每一种 Message 都包括以下属性：

name	value
message_id	消息 id，64 位整型
target	开发者账号（OpenID ）
source	发送方账号（OpenID ）
time	信息发送的时间，一个 UNIX 时间戳。
raw	信息的原始 XML 格式

8.2 TextMessage

TextMessage 的属性：

name	value
type	'text'
content	信息的内容

8.3 ImageMessage

ImageMessage 的属性：

name	value
type	'image'
img	图片网址。你可以从这个网址下到图片

8.4 LinkMessage

name	value
type	'link'
title	消息标题
description	消息描述
url	消息链接

8.5 LocationMessage

LocationMessage 的属性：

name	value
type	'location'
location_x	纬度
location_y	经度
scale	地图缩放大小
label	地理位置信息

8.6 VoiceMessage

VoiceMessage 的属性：

name	value
type	'voice'
media_id	消息媒体 ID
format	声音格式
recognition	语音识别结果

8.7 VideoMessage

VideoMessage 的属性：

name	value
type	'video'
media_id	消息媒体 ID
thumb_media_id	视频缩略图媒体 ID

8.8 UnknownMessage

UnknownMessage 的属性：

name	value
type	'unknown'
raw	请求的正文部分。标准的 XML 格式。

备注： 如果你不为 **ZgRoBot** 贡献代码，你完全可以无视掉 `UnknownMessage` 。在正常的使用中，**ZgRoBot** 应该不会收到 `UnknownMessage` ——除非 **ZgRoBot** 停止开发。

CHAPTER 9

Event

9.1 Event 公共属性

除了 UnknownEvent, 每一种 Event 都包括以下属性:

name	value
message_id	消息 id
target	开发者账号（OpenID ）
source	发送方账号（OpenID ）
time	信息发送的时间，一个 UNIX 时间戳
raw	信息的原始 XML 格式

9.2 SubscribeEvent

SubscribeEvent 的属性:

name	value
type	'subscribe_event'
key	事件 key 值。当且仅当未关注公众号扫描二维码时存在
ticket	二维码的 ticket。当且仅当未关注公众号扫描二维码时存在

9.3 UnSubscribeEvent

UnSubscribeEvent 的属性:

name	value
type	'unsubscribe_event'

9.4 ScanEvent

ScanEvent 的属性:

name	value
type	'scan_event'
key	事件 KEY 值，是一个 32 位无符号整数，即创建二维码时的二维码 scene_id
ticket	二维码的 ticket，可用来换取二维码图片

9.5 ScanCodePushEvent

ScanCodePushEvent 的属性:

name	value
type	'scancode_push_event'
scan_type	扫描类型，一般是 qrcode
scan_result	扫描结果，即二维码对应的字符串信息

9.6 ScanCodeWaitMsgEvent

ScanCodeWaitMsgEvent 的属性:

name	value
type	'scancode_waitmsg_event'
scan_type	扫描类型，一般是 qrcode
scan_result	扫描结果，即二维码对应的字符串信息

9.7 PicSysphotoEvent

弹出系统拍照发图的事件推送的 Event。属性:

name	value
type	'pic_sysphoto_event'
key	事件 KEY 值，由开发者在创建菜单时设定
count	发送的图片数量
pic_list	图片列表，例如 [{'pic_md5_sum': '123'}]

9.8 PicPhotoOrAlbumEvent

弹出拍照或者相册发图的事件推送的 Event。属性:

name	value
type	'pic_photo_or_album_event'
key	事件 KEY 值，由开发者在创建菜单时设定
count	发送的图片数量
pic_list	图片列表，例如 [{'pic_md5_sum': '123'}]

9.9 PicWeixinEvent

弹出微信相册发图器的事件推送的 Event。属性:

name	value
type	'pic_weixin_event'
key	事件 KEY 值，由开发者在创建菜单时设定
count	发送的图片数量
pic_list	图片列表，例如 [{'pic_md5_sum': '123'}]

9.10 LocationSelectEvent

弹出地理位置选择器的事件推送的 Event。属性:

name	value
type	'location_select_event'
key	事件 KEY 值，由开发者在创建菜单时设定
location_x	X 坐标信息
location_y	Y 坐标信息
scale	精度，可理解为精度或者比例尺、越精细的话 scale 越高
label	地理位置的字符串信息
poi_name	朋友圈 POI 的名字，可能为 None

9.11 ClickEvent

ClickEvent 的属性:

name	value
type	'click_event'
key	事件 key 值。

9.12 ViewEvent

ViewEvent 的属性:

name	value
type	'view_event'
key	事件 key 值。

9.13 LocationEvent

LocationEvent 的属性:

name	value
type	'location_event'
latitude	地理位置纬度
longitude	地理位置经度
precision	地理位置精度

9.14 TemplateSendJobFinishEvent

模版消息发送任务完成后的 Event 通知。属性:

name	value
status	发送是否成功。为'success' 或失败原因

9.15 UserScanProductEvent

打开商品主页事件推送的 Event。属性:

name	value
type	'user_scan_product_event'
key_standard	商品编码标准。
key_str	商品编码内容。
country	用户在微信内设置的国家。
province	用户在微信内设置的省份。
city	用户在微信内设置的城市。
sex	用户的性别，1 为男性，2 为女性，0 代表未知。
scene	打开商品主页的场景，1 为扫码，2 为其他打开场景（如会话、收藏或朋友圈）。
ext_info	调用“获取商品二维码接口”时传入的 extinfo，为标识参数。

9.16 UserScanProductEnterSessionEvent

当用户从商品主页进入公众号会话时推送的 Event。属性:

name	value
type	'user_scan_product_enter_session_event'
key_standard	商品编码标准。
key_str	商品编码内容。
ext_info	调用“获取商品二维码接口”时传入的 extinfo，为标识参数。

9.17 UserScanProductAsyncEvent

当用户打开商品主页，微信会将该用户实时的地理位置信息以异步事件的形式推送的 Event。属性:

name	value
type	'user_scan_product_async_event'
key_standard	商品编码标准。
key_str	商品编码内容。
ext_info	调用“获取商品二维码接口”时传入的 extinfo，为标识参数。
region_code	用户的实时地理位置信息（目前只精确到省一级），可在国家统计局网站查到对应明细：http://www.stats.gov.cn/tjsj/tjbz/xzqhdm/201504/t20150415_712722.html

9.18 UserScanProductVerifyActionEvent

提交审核的商品，完成审核后，微信会将审核结果以事件的形式推送的 Event。属性:

name	value
type	'user_scan_product_verify_action_event'
key_standard	商品编码标准。
key_str	商品编码内容。
result	审核结果。verify_ok 表示审核通过，verify_not_pass 表示审核未通过。
reason_msg	审核未通过的原因。

9.19 CardPassCheckEvent

生成的卡券通过审核时，微信推送的 Event。属性:

name	value
type	'card_pass_check_event'
card_id	卡券 ID。
refuse_reason	审核不通过原因。

9.20 CardNotPassCheckEvent

生成的卡券未通过审核时，微信推送的 Event。属性:

name	value
type	'card_not_pass_check_event'
card_id	卡券 ID。
refuse_reason	审核不通过原因。

9.21 UserGetCardEvent

用户在领取卡券时，微信推送的 Event。属性:

name	value
type	'user_get_card_event'
card_id	卡券 ID。
user_card_code	code 序列号。
is_give_by_friend	是否为转赠领取，1 代表是，0 代表否。
friend_user_name	当 is_give_by_friend 为 1 时填入的字段，表示发起转赠用户的 openid。
outer_id	未知。
old_user_card_code	为保证安全，微信会在转赠发生后变更该卡券的 code 号，该字段表示转赠前的 code。
outer_str	领取场景值，用于领取渠道数据统计。可在生成二维码接口及添加 Addcard 接口中自定义该字段的字符串值。
is_restore_member_card	用户删除会员卡后可重新找回，当用户本次操作为找回时，该值为 1，否则为 0。
is_recommend_by_friend	未知。

9.22 UserGiftingCardEvent

用户在转赠卡券时，微信推送的 Event。属性:

name	value
type	'user_gifting_card_event'
card_id	卡券 ID。
user_card_code	code 序列号。
friend_user_name	接收卡券用户的 openid。
is_return_back	是否转赠退回，0 代表不是，1 代表是。
is_chat_room	是否是群转赠。

9.23 UserDelCardEvent

用户在删除卡券时，微信推送的 Event。属性:

name	value
type	'user_del_card_event'
card_id	卡券 ID。
user_card_code	code 序列号。自定义 code 及非自定义 code 的卡券被领取后都支持事件推送。

9.24 UserConsumeCardEvent

卡券被核销时，微信推送的 Event。属性:

name	value
type	'user_consume_card_event'
card_id	卡券 ID。
user_card_code	code 序列号。
consume_source	核销来源。
location_name	门店名称，当前卡券核销的门店名称（只有通过自助核销和买单核销时才会出现该字段）。
staff_open_id	核销该卡券核销员的 openid（只有通过卡券商户助手核销时才会出现）。
verify_code	自助核销时，用户输入的验证码。
remark_amount	自助核销时，用户输入的备注金额。
outer_str	开发者发起核销时传入的自定义参数，用于进行核销渠道统计。

9.25 UserPayFromPayCellEvent

用户微信买单完成时，微信推送的 Event。属性:

name	value
type	'user_pay_from_pay_cell_event'
card_id	卡券 ID。
user_card_code	code 序列号。
trans_id	微信支付交易订单号（只有使用买单功能核销的卡券才会出现）。
location_id	门店 ID，当前卡券核销的门店 ID（只有通过卡券商户助手和买单核销时才会出现）。
fee	实付金额，单位为分。
original_fee	应付金额，单位为分。

9.26 UserViewCardEvent

用户在进入会员卡时，微信推送的 Event。属性:

name	value
type	'user_view_card_event'
card_id	卡券 ID。
user_card_code	code 序列号。
outer_str	商户自定义二维码渠道参数，用于标识本次扫码打开会员卡来源来自于某个渠道值的二维码。

9.27 UserEnterSessionFromCardEvent

用户在卡券里点击查看公众号进入会话时（需要用户已经关注公众号），微信推送的 Event。属性:

name	value
type	'user_enter_session_from_card_event'
card_id	卡券 ID。
user_card_code	code 序列号。

9.28 UpdateMemberCardEvent

用户的会员卡积分余额发生变动时，微信推送的 Event。属性:

name	value
type	'update_member_card_event'
card_id	卡券 ID。
user_card_code	code 序列号。
modify_bonus	变动的积分值。
modify_balance	变动的余额值。

9.29 CardSkuRemindEvent

当某个 card_id 的初始库存数大于 200 且当前库存小于等于 100 时，用户尝试领券，微信推送的 Event。属性:

name	value
type	'card_sku_remind_event'
card_id	卡券 ID。
detail	报警详细信息。

9.30 CardPayOrderEvent

用户的会员卡积分余额发生变动时，微信推送的 Event。属性:

name	value
type	'card_pay_order_event'
card_id	卡券 ID。
user_card_code	code 序列号。
order_id	本次推送对应的订单号。
status	本次订单号的状态。
create_order_time	购买券点时，支付二维码的生成时间。
pay_finish_time	购买券点时，实际支付成功的时间。
desc	支付方式，一般为微信支付充值。
free_coin_count	剩余免费券点数量。
pay_coin_count	剩余付费券点数量。
refund_free_coin_count	本次变动的免费券点数量。
refund_pay_coin_count	本次变动的付费券点数量
order_type	所要拉取的订单类型。
memo	系统备注，说明此次变动的缘由，如开通账户奖励、门店奖励、核销奖励以及充值、扣减。
receipt_info	所开发票的详情。

9.31 SubmitMembercardUserInfoEvent

用户通过一键激活的方式提交信息并点击激活或者用户修改会员卡信息时，微信推送的 Event。属性:

name	value
type	'submit_membercard_user_info_event'
card_id	卡券 ID。
user_card_code	code 序列号。

9.32 UnknownEvent

UnknownEvent 的属性:

name	value
type	'unknown_event'
raw	请求的正文部分。标准的 XML 格式。

备注: 如果你不为 **ZgRoBot** 贡献代码，你完全可以无视掉 `UnknownEvent` 。在正常的使用中，**ZgRoBot** 应该不会收到 `UnknownEvent` ——除非 **ZgRoBot** 停止开发。

CHAPTER 10

回复

你可以在构建 “Reply“时传入一个合法的 `Message` 对象来自动生成 `source` 和 `target`

```
reply = TextReply(message=message, content='Hello!')
```

10.1 TextReply

`TextReply` 是简单的文本消息，构造函数的参数如下：

name	value
content	信息正文。
target	信息的目标用户。通常是机器人用户。
source	信息的来源用户。通常是发送信息的用户。
time	信息发送的时间，一个 UNIX 时间戳。默认情况下会使用当前时间。

备注： 如果你的 `handler` 返回了一个字符串，**ZgRoBot** 会自动将其转化为一个文本消息。

10.2 ImageReply

`ImageReply` 为回复图片消息，构造函数的参数如下：

name	value
media_id	通过素材管理接口上传多媒体文件，得到的 id。
target	信息的目标用户。通常是机器人用户。
source	信息的来源用户。通常是发送信息的用户。
time	信息发送的时间，一个 UNIX 时间戳。默认情况下会使用当前时间。

10.3 VoiceReply

`VoiceReply` 为回复语音消息，构造函数的参数如下：

name	value
media_id	通过素材管理接口上传多媒体文件，得到的 id。
target	信息的目标用户。通常是机器人用户。
source	信息的来源用户。通常是发送信息的用户。
time	信息发送的时间，一个 UNIX 时间戳。默认情况下会使用当前时间。

10.4 VideoReply

`VideoReply` 为回复视频消息，构造函数的参数如下：

name	value
media_id	通过素材管理接口上传多媒体文件，得到的 id。
title	视频消息的标题。可为空。
description	视频消息的描述。可为空。
target	信息的目标用户。通常是机器人用户。
source	信息的来源用户。通常是发送信息的用户。
time	信息发送的时间，一个 UNIX 时间戳。默认情况下会使用当前时间。

10.5 ArticlesReply

`ArticlesReply` 是图文消息，构造函数的参数如下：

name	value
content	信息正文。**可为空**。
target	信息的目标用户。通常是机器人用户。
source	信息的来源用户。通常是发送信息的用户。
time	信息发送的时间，一个 UNIX 时间戳。默认情况下会使用当前时间。

你需要给 `ArticlesReply` 添加 `Article` 来增加图文。`Article` 类位于 `zgrobot.reply.Article` 。

`Article` 的构造函数的参数如下：

name	value
title	标题
description	描述
img	图片链接
url	点击图片后跳转链接

备注： 微信公众平台对图片链接有特殊的要求，详情可以在 消息接口使用指南 里看到。

在构造完一个 `Article` 后，你需要通过 `ArticlesReply` 的 `add_article` 参数把它添加进去。就像这样：

```
from zgrobot.replies import ArticlesReply, Article
reply = ArticlesReply(message=message)
article = Article(
    title="ZgRoBot",
    description="ZgRoBot是一个微信机器人框架",
    img="https://github.com/apple-touch-icon-144.png",
    url="https://github.com/whtsky/ZgRoBot"
)
reply.add_article(article)
```

备注： 根据微信公众平台的 最新公告 ，每个 ArticlesReply 中 **最多添加 1 个 Article** 。

你也可以让你的 `handler` 返回一个列表，里面每一个元素都是一个长度为四的列表，
ZgRoBot 会将其自动转为 `ArticlesReply` 。就像这样：

```
import zgrobot

robot = zgrobot.ZgRoBot(token='tokenhere')

@robot.text
def articles(message):
    return [
        [
            "title",
            "description",
            "img",
            "url"
        ],
        [
            "whtsky",
            "I wrote ZgRoBot",
            "https://secure.gravatar.com/avatar/0024710771815ef9b74881ab21ba4173?
↪s=420",
            "http://whouz.com/"
        ]
    ]

robot.run()
```

10.6 MusicReply

`MusicReply` 是音乐消息，构造函数的参数如下：

name	value
target	信息的目标用户。通常是机器人用户。
source	信息的来源用户。通常是发送信息的用户。
time	信息发送的时间，一个 UNIX 时间戳。默认情况下会使用当前时间。
title	标题
description	描述
url	音乐链接
hq_url	高质量音乐链接，WIFI 环境优先使用该链接播放音乐。可为空[3]

你也可以让你的 `handler` 返回一个长度为三或四的列表，[3] **ZgRoBot** 会将其自动转为 `MusicReply`，就像这样：

[3] 如果你省略了高质量音乐链接的地址，**ZgRoBot** 会自动将音乐链接的地址用于高质量音乐链接。

```
import zgrobot

robot = zgrobot.ZgRoBot(token='tokenhere')

@robot.text
def music(message):
    return [
        "title",
        "description",
        "music_url",
        "hq_music_url"
        ]

@robot.text
def music2(message):
    return [
        "微信你不懂爱",
        "龚琳娜最新力作",
        "http://weixin.com/budongai.mp3",
        ]

robot.run()
```

10.7 TransferCustomerServiceReply

将消息转发到多客服, 构造函数的参数如下:

name	value
target	信息的目标用户。通常是机器人用户。
source	信息的来源用户。通常是发送信息的用户。
time	信息发送的时间，一个 UNIX 时间戳。默认情况下会使用当前时间。
account	指定会话接入的客服账号，可以没有此参数，没有时会自动分配给可用客服。

10.8 SuccessReply

给微信服务器回复 `success`

假如服务器无法保证在五秒内处理并回复，需要回复 `SuccessReply` ，这样微信服务器才不会对此作任何处理，并且不会发起重试。

CHAPTER 11

Config

ZgRoBot 使用 ZgRoBot.Config 类来存储配置信息。ZgRoBot 类实例的 config 属性是一个*Config()* 实例。

Config() 继承自 dict 。因此，你可以像使用普通 dict 一样使用它

```
from zgrobot import ZgRoBot
robot = ZgRoBot(token='2333')

robot.config.update(
    HOST='0.0.0.0',
    PORT=80
)
```

当然，你也可以先创建一个 Config ，然后在初始化 ZgRobot 的时候传入自己的 Config

```
from zgrobot.config import Config
config = Config(
    TOKEN="token from config!"
)
robot = ZgRoBot(config=config, token="token from init")
assert robot.token == "token from config!"
```

备注： 如果你在初始化 ZgRoBot 时传入了 config 参数，ZgRoBot 会忽略除 logger 外其他所有的初始化参数。如果你需要对 ZgRoBot 进行一些配置操作，请修改 Config 。

与普通 dict 不同的是，你可以先把配置文件保存在一个对象或是文件中，然后在*Config()* 中导入配置

```
from zgrobot import ZgRoBot
robot = ZgRoBot(token='2333')

class MyConfig(object):
    HOST = '0.0.0.0'
    PORT = 80

robot.config.from_object(MyConfig)
robot.config.from_pyfile("config.py")
```

11.1 默认配置

```
dict(
    TOKEN=None,
    SERVER="auto",
    HOST="127.0.0.1",
    PORT="8888",
    SESSION_STORAGE=None,
    APP_ID=None,
    APP_SECRET=None,
    ENCODING_AES_KEY=None
)
```

CHAPTER 12

与其他 Web 框架集成

ZgRoBot 可以作为独立服务运行，也可以集成在其他 **Web** 框架中一同运行。

12.1 FastApi

ZgRoBot 支持 **fastapi 0.78+**

先写好你的机器人

```
# Filename: robot.py

from zgrobot import ZgRoBot

myrobot = ZgRoBot(token='token')

@myrobot.handler
def hello():
    return "Hello World"
```

然后创建最简单的 **FastApi** 项目 `main.py`

```
# Filename: main.py

from zgrobot.contrib.fastapi import make_view
from fastapi import FastAPI
```

(续下页)

(接上页)

```
from robot import myrobot

app = FastAPI()

app.add_route("/", make_view(robot=robot), methods=["GET", "POST"])
```

或者使用 **FastApi** 的中间件

```
# Filename: main.py

from fastapi import FastAPI
from fastapi.middleware.wsgi import WSGIMiddleware

from robot import myrobot

app = FastAPI()

app.mount("/", WSGIMiddleware(myrobot.wsgi))
```

12.2 Django

ZgRoBot 支持 **Django 2.2+** 。

首先，在一个文件中写好你的微信机器人

```
# Filename: robot.py

from zgrobot import ZgRoBot

myrobot = ZgRoBot(token='token')


@myrobot.handler
def hello(message):
    return 'Hello World!'
```

然后，在你 **Django** 项目中的 `urls.py` 中调用 `zgrobot.contrib.django.make_view()` ，将 **ZgRoBot** 集成进 **Django**

```
from django.conf.urls import patterns, include, url
from zgrobot.contrib.django import make_view
```

(续下页)

(接上页)

```
from robot import myrobot

urlpatterns = patterns('',
    url(r'^robot/', make_view(myrobot)),
)
```

12.3 Flask

首先, 同样在文件中写好你的微信机器人

```
# Filename: robot.py

from zgrobot import ZgRoBot

myrobot = ZgRoBot(token='token')


@myrobot.handler
def hello(message):
    return 'Hello World!'
```

然后, 在 **Flask** 项目中为 **Flask** 实例集成 **ZgRoBot**

```
from flask import Flask
from robot import myrobot
from zgrobot.contrib.flask import make_view

app = Flask(__name__)
app.add_url_rule(rule='/robot', # ZgRoBot 挂载地址
                 endpoint='zgrobot', # Flask 的 endpoint
                 view_func=make_view(myrobot),
                 methods=['GET', 'POST'])
```

12.4 Bottle

在你的 Bottle App 中集成 ZgRoBot

```
from zgrobot import ZgRoBot

myrobot = ZgRoBot(token='token')
```

(续下页)

(接上页)

```
@myrobot.handler
def hello(message):
    return 'Hello World!'

from bottle import Bottle
from zgrobot.contrib.bottle import make_view

app = Bottle()
app.route('/robot',  # ZgRoBot 挂载地址
          ['GET', 'POST'],
          make_view(myrobot))
```

zgrobot.contrib.bottle.**make_view**(*robot*)

为一个 BaseRoBot 生成 Bottle view。

Usage

```
from zgrobot import ZgRoBot

robot = ZgRoBot(token='token')


@robot.handler
def hello(message):
    return 'Hello World!'

from bottle import Bottle
from zgrobot.contrib.bottle import make_view

app = Bottle()
app.route(
    '/robot',  # ZgRoBot 挂载地址
    ['GET', 'POST'],
    make_view(robot)
)
```

参数

robot -- 一个 BaseRoBot 实例

返回

一个标准的 Bottle view

12.5 Tornado

最简单的 Hello World

```
import tornado.ioloop
import tornado.web
from zgrobot import ZgRoBot
from zgrobot.contrib.tornado import make_handler

myrobot = ZgRoBot(token='token')


@myrobot.handler
def hello(message):
    return 'Hello World!'

application = tornado.web.Application([
    (r"/robot/", make_handler(myrobot)),
])

if __name__ == "__main__":
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

CHAPTER 13

错误页面

ZgRoBot 自带了一个错误页面，它将会在 Signature 验证不通过的时候返回错误页面。

13.1 定制错误页面

如果你想为 ZgRoBot 指定 Signature 验证不通过时显示的错误页面，可以这么做:

```
@robot.error_page
def make_error_page(url):
    return "<h1>喵喵喵 %s 不是给麻瓜访问的快走开</h1>" % url
```

CHAPTER 14

小工具

14.1 Token 生成器

ZgRoBot 帮你准备了一个 `Token` 生成器

```
import zgrobot.utils

print(zgrobot.utils.generate_token())
```

CHAPTER 15

贡献指南

有许多种为 ZgRoBot 做贡献的方式，包括但并不仅限于

- 上报 bug
- 提交 Feature Request
- 贡献代码
- 把 ZgRoBot 安利给你周围的人:)

15.1 贡献代码

如果你希望为 **ZgRoBot** 贡献代码，请现在 **GitHub** 上 Fork **ZgRobot** 仓库，然后在 `develop` 分支上开一个新的分支。

备注： 请注意，是在 `develop` 分支上新建分支，`master` 分支仅允许来自 `develop` 分支的合并。

如果你的贡献的代码是修复 **Bug** ，请确认这个 **Bug** 已经有了对应的 **Issue** （如果没有，请先创建一个）；然后在 **Pull Request** 的描述里面引用这个 **Bug** 的 **Issue ID** ，就像这样 (假设 **Issue ID** 为 153)

```
Fix #153
```

15.1.1 环境搭建

建议使用 `virtualenv` 创建虚拟环境进行开发, 然后安装开发环境需要的 `packages`。关于 Python 版本, 推荐使用 **Python 3.6** 进行开发。

如果使用的是 3.x 版本

```
# Python 3.5
python -m venv venv
```

如果是其他版本

```
# virtualenv is highly recommended.
virtualenv venv
# Activate virtualenv.
source venv/bin/activate
# Install dev packages.
pip install -r dev-requirements.txt
```

15.1.2 代码风格

我们使用 yapf 进行代码格式化。在提交代码之前，请格式化一下你的代码

```
# Install yapf
pip install yapf
# format code
yapf -i -p -r zgrobot/ tests/ *.py
```

你也可以 安装 yapf Pre-Commit Hook 来自动进行代码格式化工作。

15.1.3 测试

在代码提交之前, 请先运行本地的测试。每次提交之后会有在线的 CI 运行更多版本兼容性的测试, 请密切关注测试结果。

```
# Run tests locally.
python setup.py test
```

当然也可以使用 tox 在本地运行多版本的兼容性测试。

```
# Run multi-version tests locally.
tox
```

如果你的 Pull Request 添加了新的模块或者功能，请为这些代码添加必要的测试。所有的测试文件都在 tests 文件夹下。

当一切开发完成之后, 可以发 Pull Request 到 `develop` 分支, 我们会为你的代码做 Review。同时 CI 也会自动运行测试。

备注： 我们只会 Merge 通过了测试的代码。

CHAPTER 16

API

16.1 应用对象

class zgrobot.robot.**BaseRoBot**(*token=None*, *logger=None*, *enable_session=None*, *session_storage=None*, *app_id=None*, *app_secret=None*, *encoding_aes_key=None*, *config=None*, ***kwargs*)

BaseRoBot 是整个应用的核心对象，负责提供 handler 的维护，消息和事件的处理等核心功能。

参数

- **logger** -- 用来输出 log 的 logger，如果是 None，将使用 zgrobot.logger
- **config** -- 用来设置的*zgrobot.config.Config* 对象

备注： 对于下面的参数推荐使用*Config* 进行设置，并且以下参数均已 **deprecated**。

参数

- **token** -- 微信公众号设置的 token **(deprecated)**
- **enable_session** -- 是否开启 session **(deprecated)**
- **session_storage** -- 用来储存 session 的对象，如果为 None，将使用 zgrobot.session.sqlitestorage.SQLiteStorage **(deprecated)**
- **app_id** -- 微信公众号设置的 app id **(deprecated)**

- **app_secret** -- 微信公众号设置的 app secret **(deprecated)**
- **encoding_aes_key** -- 用来加解密消息的 aes key **(deprecated)**

add_filter(*func*, *rules*)

为 BaseRoBot 添加一个 `filter handler`。

参数

- **func** -- 如果 rules 通过，则处理该消息的 handler。
- **rules** -- 一个 list，包含要匹配的字符串或者正则表达式。

返回

None

add_handler(*func*, *type='all'*)

为 BaseRoBot 实例添加一个 handler。

参数

- **func** -- 要作为 handler 的方法。
- **type** -- handler 的种类。

返回

None

card_not_pass_check(*f*)

为生成的卡券未通过审核 (`card_not_pass_check_event`) 事件添加一个 handler 方法的装饰器。

card_pass_check(*f*)

为生成的卡券通过审核 (`card_pass_check_event`) 事件添加一个 handler 方法的装饰器。

card_pay_order(*f*)

为券点发生变动 (`card_pay_order_event`) 事件添加一个 handler 方法的装饰器。

card_sku_remind(*f*)

为某个 card_id 的初始库存数大于 200 且当前库存小于等于 100 (`card_sku_remind_event`) 事件添加一个 handler 方法的装饰器。

check_signature(*timestamp*, *nonce*, *signature*)

根据时间戳和生成签名的字符串 (nonce) 检查签名。

参数

- **timestamp** -- 时间戳
- **nonce** -- 生成签名的随机字符串
- **signature** -- 要检查的签名

返回

如果签名合法将返回 `True`，不合法将返回 `False`

click(*f*)

为自定义菜单事件（`click`）事件添加一个 handler 方法的装饰器。

error_page(*f*)

为 robot 指定 Signature 验证不通过时显示的错误页面。

Usage:

```
@robot.error_page
def make_error_page(url):
    return "<h1>喵喵喵 %s 不是给麻瓜访问的快走开</h1>" % url
```

filter(**args*)

为文本（`text`）消息添加 handler 的简便方法。

使用 `@filter("xxx")`, `@filter(re.compile("xxx"))` 或 `@filter("xxx", "xxx2")` 的形式为特定内容添加 handler。

get_encrypted_reply(*message*)

对一个指定的 ZgRoBot Message ， 获取 handlers 处理后得到的 Reply。如果可能，对该 Reply 进行加密。返回 Reply Render 后的文本。

参数

message -- 一个 ZgRoBot Message 实例。

返回

reply （纯文本）

handler(*f*)

为每一条消息或事件添加一个 handler 方法的装饰器。

image(*f*)

为图像（`image`）消息添加一个 handler 方法的装饰器。

key_click(*key*)

为自定义菜单（`click`）事件添加 handler 的简便方法。

@key_click('KEYNAME') 用来为特定 key 的点击事件添加 handler 方法。

link(*f*)

为链接（`link`）消息添加一个 handler 方法的装饰器。

location(*f*)

为位置（`location`）消息添加一个 handler 方法的装饰器。

location_event(*f*)

为上报位置（location_event）事件添加一个 handler 方法的装饰器。

location_select(*f*)

为弹出地理位置选择器的事件推送（location_select_event）事件添加一个 handler 方法的装饰器。

parse_message(*body*, *timestamp=None*, *nonce=None*, *msg_signature=None*)

解析获取到的 Raw XML，如果需要的话进行解密，返回 ZgRoBot Message。:param body: 微信服务器发来的请求中的 Body。:return: ZgRoBot Message

pic_photo_or_album(*f*)

为弹出拍照或者相册发图的事件推送（pic_photo_or_album_event）事件添加一个 handler 方法的装饰器。

pic_sysphoto(*f*)

为弹出系统拍照发图的事件推送（pic_sysphoto_event）事件添加一个 handler 方法的装饰器。

pic_weixin(*f*)

为弹出微信相册发图器的事件推送（pic_weixin_event）事件添加一个 handler 方法的装饰器。

scan(*f*)

为扫描推送（scan）事件添加一个 handler 方法的装饰器。

scancode_push(*f*)

为扫描推送（scancode_push）事件添加一个 handler 方法的装饰器。

scancode_waitmsg(*f*)

为扫描弹消息（scancode_waitmsg）事件添加一个 handler 方法的装饰器。

shortvideo(*f*)

为小视频（shortvideo）消息添加一个 handler 方法的装饰器。

submit_membercard_user_info(*f*)

为用户通过一键激活的方式提交信息并点击激活或者用户修改会员卡信息（submit_membercard_user_info_event）事件添加一个 handler 方法的装饰器。

subscribe(*f*)

为被关注（subscribe）事件添加一个 handler 方法的装饰器。

templatesendjobfinish_event(*f*)

在模版消息发送任务完成后，微信服务器会将是否送达成功作为通知，发送到开发者中心中填写的服务器配置地址中

text(*f*)

为文本（text）消息添加一个 handler 方法的装饰器。

unknown(*f*)

为未知类型（unknown）消息添加一个 handler 方法的装饰器。

unknown_event(*f*)

为未知类型（unknown_event）事件添加一个 handler 方法的装饰器。

unsubscribe(*f*)

为被取消关注（unsubscribe）事件添加一个 handler 方法的装饰器。

update_member_card(*f*)

为用户的会员卡积分余额发生变动（update_member_card_event）事件添加一个 handler 方法的装饰器。

user_consume_card(*f*)

为卡券被核销（user_consume_card_event）事件添加一个 handler 方法的装饰器。

user_del_card(*f*)

为用户删除卡券（user_del_card_event）事件添加一个 handler 方法的装饰器。

user_enter_session_from_card(*f*)

为用户卡券里点击查看公众号进入会话（user_enter_session_from_card_event）事件添加一个 handler 方法的装饰器。

user_get_card(*f*)

为用户领取卡券（user_get_card_event）事件添加一个 handler 方法的装饰器。

user_gifting_card(*f*)

为用户转赠卡券（user_gifting_card_event）事件添加一个 handler 方法的装饰器。

user_pay_from_pay_cell(*f*)

为微信买单完成（user_pay_from_pay_cell_event）事件添加一个 handler 方法的装饰器。

user_scan_product(*f*)

为打开商品主页事件推送（user_scan_product_event）事件添加一个 handler 方法的装饰器。

user_scan_product_async(*f*)

为地理位置信息异步推送（user_scan_product_async_event）事件添加一个 handler 方法的装饰器。

user_scan_product_enter_session(*f*)

为进入公众号事件推送（user_scan_product_enter_session_event）事件添加一个 handler 方法的装饰器。

user_scan_product_verify_action(*f*)

为商品审核结果推送（user_scan_product_verify_action_event）事件添加一个 handler 方法的装饰器。

user_view_card(*f*)

为用户进入会员卡（user_view_card_event）事件添加一个 handler 方法的装饰器。

video(*f*)

为视频（video）消息添加一个 handler 方法的装饰器。

view(*f*)

为链接（view）事件添加一个 handler 方法的装饰器。

voice(*f*)

为语音（voice）消息添加一个 handler 方法的装饰器。

class zgrobot.robot.**ZgRoBot**(*token=None*, *logger=None*, *enable_session=None*, *session_storage=None*, *app_id=None*, *app_secret=None*, *encoding_aes_key=None*, *config=None*, ***kwargs*)

ZgRoBot 是一个继承自 BaseRoBot 的对象，在 BaseRoBot 的基础上使用了 bottle 框架，提供接收微信服务器发来的请求的功能。

run(*server=None*, *host=None*, *port=None*, *enable_pretty_logging=True*)

运行 ZgRoBot。

参数

- **server** -- 传递给 Bottle 框架 run 方法的参数，详情见 bottle 文档
- **host** -- 运行时绑定的主机地址
- **port** -- 运行时绑定的主机端口
- **enable_pretty_logging** -- 是否开启 log 的输出格式优化

16.2 配置对象

class zgrobot.config.**Config**

from_object(*obj*)

在给定的 Python 对象中读取配置。

参数

obj -- 一个 Python 对象

from_pyfile(*filename*)

在一个 Python 文件中读取配置。

参数

filename -- 配置文件的文件名

返回

如果读取成功，返回 `True`，如果失败，会抛出错误异常

16.3 Session 对象

class zgrobot.session.sqlitestorage.**SQLiteStorage**(*filename='zgrobot_session.sqlite3'*)

SQLiteStorge 会把 Session 数据储存在一个 SQLite 数据库文件中

```
import zgrobot
from zgrobot.session.sqlitestorage import SQLiteStorage

session_storage = SQLiteStorage
robot = zgrobot.ZgRoBot(token="token", enable_session=True,
                        session_storage=session_storage)
```

参数

filename -- SQLite 数据库的文件名, 默认是 `zgrobot_session.sqlite3`

class zgrobot.session.filestorage.**FileStorage**(*filename: str = 'zgrobot_session'*)

FileStorage 会把你的 Session 数据以 dbm 形式储存在文件中。

参数

filename -- 文件名，默认为 `zgrobot_session`

class zgrobot.session.mongodbstorage.**MongoDBStorage**(*collection*)

MongoDBStorage 会把你的 Session 数据储存在一个 MongoDB Collection 中

```
import pymongo
import zgrobot
from zgrobot.session.mongodbstorage import MongoDBStorage

collection = pymongo.MongoClient()["wechat"]["session"]
session_storage = MongoDBStorage(collection)
robot = zgrobot.ZgRoBot(token="token", enable_session=True,
                        session_storage=session_storage)
```

你需要安装 `pymongo` 才能使用 MongoDBStorage 。

参数

collection -- 一个 MongoDB Collection。

class zgrobot.session.redisstorage.**RedisStorage**(*redis*, *prefix='ws_'*)

RedisStorage 会把你的 Session 数据储存在 Redis 中

```
import redis
import zgrobot
from zgrobot.session.redisstorage import RedisStorage

db = redis.Redis()
session_storage = RedisStorage(db, prefix="my_prefix_")
robot = zgrobot.ZgRoBot(token="token", enable_session=True,
                        session_storage=session_storage)
```

你需要安装 `redis` 才能使用 RedisStorage 。

参数

- **redis** -- 一个 Redis Client。
- **prefix** -- Reids 中 Session 数据 key 的 prefix 。默认为 `ws_`

class zgrobot.session.saekvstorage.**SaeKVDBStorage**(*prefix='ws_'*)

SaeKVDBStorage 使用 SAE 的 KVDB 来保存你的 session

```
import zgrobot
from zgrobot.session.saekvstorage import SaeKVDBStorage

session_storage = SaeKVDBStorage()
robot = zgrobot.ZgRoBot(token="token", enable_session=True,
                        session_storage=session_storage)
```

需要先在后台开启 KVDB 支持

参数

prefix -- KVDB 中 Session 数据 key 的 prefix 。默认为 `ws_`

class zgrobot.session.mysqlstorage.**MySQLStorage**(*conn*)

MySQLStorage 会把你的 Session 数据储存在 MySQL 中

```
import MySQLdb # 使用 mysqlclient
import zgrobot
from zgrobot.session.mysqlstorage import MySQLStorage

conn = MySQLdb.connect(user='', db='', passwd='', host='')
session_storage = MySQLStorage(conn)
robot = zgrobot.ZgRoBot(token="token", enable_session=True,
                        session_storage=session_storage)
```

或者

```
import pymysql # 使用 pymysql
import zgrobot
from zgrobot.session.mysqlstorage import MySQLStorage

session_storage = MySQLStorage(
conn=pymysql.connect(
    user='喵',
    password='喵喵',
    db='zgrobot',
    host='127.0.0.1',
    charset='utf8'
))
robot = zgrobot.ZgRoBot(token="token", enable_session=True,
                        session_storage=session_storage)
```

你需要安装一个 MySQL Client 才能使用 MySQLStorage，比如 `pymysql`，`mysqlclient` 。

理论上符合 PEP-249 的库都可以使用，测试时使用的是 `pymysql`。

参数

conn -- PEP-249 定义的 Connection 对象

class zgrobot.session.postgresqlstorage.**PostgreSQLStorage**(*conn*)

PostgreSQLStorage 会把你的 Session 数据储存在 PostgreSQL 中

```
import psycopg2  # pip install psycopg2-binary
import zgrobot
from zgrobot.session.postgresqlstorage import PostgreSQLStorage

conn = psycopg2.connect(host='127.0.0.1', port='5432', dbname='zgrobot', user='nya
↪', password='nyanya')
session_storage = PostgreSQLStorage(conn)
robot = zgrobot.ZgRoBot(token="token", enable_session=True,
                        session_storage=session_storage)
```

你需要安装一个 `PostgreSQL Client` 才能使用 PostgreSQLStorage，比如 `psycopg2`。

理论上符合 PEP-249 的库都可以使用，测试时使用的是 `psycopg2`。

参数

conn -- PEP-249 定义的 Connection 对象

16.4 log

zgrobot.logger.**enable_pretty_logging**(*logger*, *level='info'*)

按照配置开启 log 的格式化优化。

参数

- **logger** -- 配置的 logger 对象
- **level** -- 要为 logger 设置的等级

CHAPTER 17

Q&A

17.1 重写 get_access_token 函数来自定义 token 的存储

首先我们看 get_access_token() 的源码:

```
def get_access_token(self) -> str:
    """
    判断现有的token是否过期。
    用户需要多进程或者多机部署可以手动重写这个函数
    来自定义token的存储，刷新策略。

    :return: 返回 ``token``
    """
    if self._token:
        now = time.time()
        if self.token_expires_at - now > 60:
            return self._token
    with LOCK:
        json_str = self.grant_token()
        self._token = json_str["access_token"]
        self.token_expires_at = int(time.time()) + json_str["expires_in"]
    return self._token
```

这其中包含了两个功能:

- 通过时间戳对现有 token 判断是否过期；若未过期，则直接返回现有的 token

- 对现有 token 判断是否过期；若过期了，对进程进行上锁，并重新获取 token

因此，在重写 get_access_token() 时，请务必保证这两个功能的实现，并在重新获取 token 和 token_expires_at 后，即可对其进行自定义操作，例如:

```
def get_access_token(self) -> str:
    if self._token:
        now = time.time()
        if self.token_expires_at - now > 60:
            return self._token
    with LOCK:
        json_str = self.grant_token()
        self._token = json_str["access_token"]
        self.token_expires_at = int(time.time()) + json_str["expires_in"]

    # 从这里开始即可进行自定义 ``token`` 的操作
    return self._token
```

Python 模块索引

z

zgrobot, 87
zgrobot.client, 25
zgrobot.config, 92
zgrobot.contrib.bottle, 76
zgrobot.contrib.django, 75
zgrobot.contrib.fastapi, 74
zgrobot.contrib.flask, 75
zgrobot.contrib.tornado, 77
zgrobot.logger, 96
zgrobot.robot, 87
zgrobot.session.filestorage, 93
zgrobot.session.mongodbstorage, 93
zgrobot.session.mysqlstorage, 94
zgrobot.session.postgresqlstorage, 95
zgrobot.session.redisstorage, 93
zgrobot.session.saekvstorage, 94
zgrobot.session.sqlitestorage, 93

索引

A

add_custom_service_account() (zgrobot.client.Client 方法), 30
add_filter() (zgrobot.robot.BaseRoBot 方法), 88
add_handler() (zgrobot.robot.BaseRoBot 方法), 88
add_news() (zgrobot.client.Client 方法), 40

B

BaseRoBot (zgrobot.robot 中的类), 87

C

card_not_pass_check() (zgrobot.robot.BaseRoBot 方法), 88
card_pass_check() (zgrobot.robot.BaseRoBot 方法), 88
card_pay_order() (zgrobot.robot.BaseRoBot 方法), 88
card_sku_remind() (zgrobot.robot.BaseRoBot 方法), 88
check_signature() (zgrobot.robot.BaseRoBot 方法), 88
click() (zgrobot.robot.BaseRoBot 方法), 89
Config (zgrobot.config 中的类), 92
create_custom_menu() (zgrobot.client.Client 方法), 28
create_group() (zgrobot.client.Client 方法), 36
create_menu() (zgrobot.client.Client 方法), 26
create_qrcode() (zgrobot.client.Client 方法), 38
create_tag() (zgrobot.client.Client 方法), 44

D

delete_custom_menu() (zgrobot.client.Client 方法), 29
delete_custom_service_account() (zgrobot.client.Client 方法), 30
delete_group() (zgrobot.client.Client 方法), 37
delete_mass_msg() (zgrobot.client.Client 方法), 34
delete_menu() (zgrobot.client.Client 方法), 28
delete_permanent_media() (zgrobot.client.Client 方法),

41
delete_tag() (zgrobot.client.Client 方法), 45
download_media() (zgrobot.client.Client 方法), 39
download_permanent_media() (zgrobot.client.Client 方法), 41

E

enable_pretty_logging() (在 zgrobot.logger 模块中), 96
error_page() (zgrobot.robot.BaseRoBot 方法), 89

F

FileStorage(zgrobot.session.filestorage 中的类), 93
filter() (zgrobot.robot.BaseRoBot 方法), 89
from_object() (zgrobot.config.Config 方法), 92
from_pyfile() (zgrobot.config.Config 方法), 92

G

get_access_token() (zgrobot.client.Client 方法), 25
get_custom_menu_config() (zgrobot.client.Client 方法), 30
get_custom_service_account_list() (zgrobot.client.Client 方法), 31
get_encrypted_reply() (zgrobot.robot.BaseRoBot 方法), 89
get_followers() (zgrobot.client.Client 方法), 39
get_group_by_id() (zgrobot.client.Client 方法), 36
get_groups() (zgrobot.client.Client 方法), 36
get_ip_list() (zgrobot.client.Client 方法), 26
get_mass_msg_speed() (zgrobot.client.Client 方法), 35
get_mass_msg_status() (zgrobot.client.Client 方法), 35
get_media_count() (zgrobot.client.Client 方法), 43
get_media_list() (zgrobot.client.Client 方法), 43
get_menu() (zgrobot.client.Client 方法), 27
get_online_custom_service_account_list() (zgrobot.client.Client 方法), 31
get_tags() (zgrobot.client.Client 方法), 44
get_tags_by_user() (zgrobot.client.Client 方法), 46
get_user_info() (zgrobot.client.Client 方法), 37
get_users_by_tag() (zgrobot.client.Client 方法), 45
get_users_info() (zgrobot.client.Client 方法), 38
grant_token() (zgrobot.client.Client 方法), 25

H

handler() (zgrobot.robot.BaseRoBot 方法), 89

I

image() (zgrobot.robot.BaseRoBot 方法), 89

K

key_click() (zgrobot.robot.BaseRoBot 方法), 89

L

link() (zgrobot.robot.BaseRoBot 方法), 89
location() (zgrobot.robot.BaseRoBot 方法), 89
location_event() (zgrobot.robot.BaseRoBot 方法), 89
location_select() (zgrobot.robot.BaseRoBot 方法), 90

M

make_view() (在 zgrobot.contrib.bottle 模块中), 76
match_custom_menu() (zgrobot.client.Client 方法), 29
MongoDBStorage(zgrobot.session.mongodbstorage 中的类), 93
move_user() (zgrobot.client.Client 方法), 36
move_users() (zgrobot.client.Client 方法), 37
MySQLStorage(zgrobot.session.mysqlstorage 中的类), 94

P

parse_message() (zgrobot.robot.BaseRoBot 方法), 90
pic_photo_or_album() (zgrobot.robot.BaseRoBot 方法), 90
pic_sysphoto() (zgrobot.robot.BaseRoBot 方法), 90
pic_weixin() (zgrobot.robot.BaseRoBot 方法), 90
PostgreSQLStorage (zgrobot.session.postgresqlstorage 中的类), 95

R

RedisStorage(zgrobot.session.redisstorage 中的类), 93
remark_user() (zgrobot.client.Client 方法), 37
run() (zgrobot.robot.ZgRoBot 方法), 92

S

SaeKVDBStorage(zgrobot.session.saekvstorage 中的类), 94
scan() (zgrobot.robot.BaseRoBot 方法), 90
scancode_push() (zgrobot.robot.BaseRoBot 方法), 90
scancode_waitmsg() (zgrobot.robot.BaseRoBot 方法), 90
send_article_message() (zgrobot.client.Client 方法), 33
send_image_message() (zgrobot.client.Client 方法), 31
send_mass_msg() (zgrobot.client.Client 方法), 34
send_mass_preview_to_user() (zgrobot.client.Client 方法), 35
send_miniprogrampage_message() (zgrobot.client.Client 方法), 33
send_music_message() (zgrobot.client.Client 方法), 32
send_news_message() (zgrobot.client.Client 方法), 33
send_template_message() (zgrobot.client.Client 方法), 46

send_text_message() (zgrobot.client.Client 方法), 31
send_video_message() (zgrobot.client.Client 方法), 32
send_voice_message() (zgrobot.client.Client 方法), 32
shortvideo() (zgrobot.robot.BaseRoBot 方法), 90
show_qrcode() (zgrobot.client.Client 方法), 38
SQLiteStorage(zgrobot.session.sqlitestorage 中的类), 93
submit_membercard_user_info() (zgrobot.robot.BaseRoBot 方法), 90
subscribe() (zgrobot.robot.BaseRoBot 方法), 90

T

tag_users() (zgrobot.client.Client 方法), 45
templatesendjobfinish_event() (zgrobot.robot.BaseRoBot 方法), 90
text() (zgrobot.robot.BaseRoBot 方法), 90

U

unknown() (zgrobot.robot.BaseRoBot 方法), 90
unknown_event() (zgrobot.robot.BaseRoBot 方法), 91
unsubscribe() (zgrobot.robot.BaseRoBot 方法), 91
untag_users() (zgrobot.client.Client 方法), 46
update_custom_service_account() (zgrobot.client.Client 方法), 30
update_group() (zgrobot.client.Client 方法), 36
update_member_card() (zgrobot.robot.BaseRoBot 方法), 91
update_news() (zgrobot.client.Client 方法), 42
update_tag() (zgrobot.client.Client 方法), 44
upload_custom_service_account_avatar() (zgrobot.client.Client 方法), 31
upload_media() (zgrobot.client.Client 方法), 39
upload_news() (zgrobot.client.Client 方法), 42
upload_news_picture() (zgrobot.client.Client 方法), 40
upload_permanent_media() (zgrobot.client.Client 方法), 40
upload_permanent_video() (zgrobot.client.Client 方法), 40
user_consume_card() (zgrobot.robot.BaseRoBot 方法), 91
user_del_card() (zgrobot.robot.BaseRoBot 方法), 91
user_enter_session_from_card() (zgrobot.robot.BaseRoBot 方法), 91
user_get_card() (zgrobot.robot.BaseRoBot 方法), 91
user_gifting_card() (zgrobot.robot.BaseRoBot 方法), 91
user_pay_from_pay_cell() (zgrobot.robot.BaseRoBot 方法), 91
user_scan_product() (zgrobot.robot.BaseRoBot 方法)

, 91
user_scan_product_async() (zgrobot.robot.BaseRoBot 方法), 91
user_scan_product_enter_session() (zgrobot.robot.BaseRoBot 方法), 91
user_scan_product_verify_action() (zgrobot.robot.BaseRoBot 方法), 91
user_view_card() (zgrobot.robot.BaseRoBot 方法), 91

V

video() (zgrobot.robot.BaseRoBot 方法), 92
view() (zgrobot.robot.BaseRoBot 方法), 92
voice() (zgrobot.robot.BaseRoBot 方法), 92

Z

zgrobot
    模块, 87
zgrobot.client
    模块, 25
zgrobot.config
    模块, 92
zgrobot.contrib.bottle
    模块, 76
zgrobot.contrib.django
    模块, 75
zgrobot.contrib.fastapi
    模块, 74
zgrobot.contrib.flask
    模块, 75
zgrobot.contrib.tornado
    模块, 77
zgrobot.logger
    模块, 96
zgrobot.robot
    模块, 87
zgrobot.session.filestorage
    模块, 93
zgrobot.session.mongodbstorage
    模块, 93
zgrobot.session.mysqlstorage
    模块, 94
zgrobot.session.postgresqlstorage
    模块, 95
zgrobot.session.redisstorage
    模块, 93
zgrobot.session.saekvstorage
    模块, 94
zgrobot.session.sqlitestorage
    模块, 93
ZgRoBot (zgrobot.robot 中的类), 92

模块
    zgrobot, 87
    zgrobot.client, 25
    zgrobot.config, 92
    zgrobot.contrib.bottle, 76
    zgrobot.contrib.django, 75
    zgrobot.contrib.fastapi, 74
    zgrobot.contrib.flask, 75
    zgrobot.contrib.tornado, 77
    zgrobot.logger, 96
    zgrobot.robot, 87
    zgrobot.session.filestorage, 93
    zgrobot.session.mongodbstorage, 93
    zgrobot.session.mysqlstorage, 94
    zgrobot.session.postgresqlstorage, 95
    zgrobot.session.redisstorage, 93
    zgrobot.session.saekvstorage, 94
    zgrobot.session.sqlitestorage, 93