目　录

前言

微信SDK For Golang

WeChat SDK 是一个Golang版本微信SDK，简单、易用。

SDK源码：https://github.com/silenceper/wechat

> 此文档使用sdk>=2.0版本，1.x系列版本文档在：这里

安装

推荐使用go module进行依赖管理

```
go get github.com/silenceper/wechat/v2
```

使用

根据功能不同，进入对应的模块文档进行查看

例子

仓库地址：https://github.com/gowechat/example

关注公众号

公众号

开发前必读：微信公众官方文档

获取微信公众号操作对象

```
wc := wechat.NewWechat()
//设置全局cache，也可以单独为每个操作实例设置
redisOpts := &cache.RedisOpts{
    Host:        "127.0.0.1:6379",
}
redisCache := cache.NewRedis(redisOpts)
wc.SetCache(redisCache)
cfg := &offConfig.Config{
    AppID:     "xxx",
    AppSecret: "xxx",
    Token:     "xxx",
    //EncodingAESKey: "xxxx",
    //Cache: redisCache, //也可以单独设置
}
officialAccount := wc.GetOfficialAccount(cfg)
//TODO 使用 `officialAccount` 操作公众号相关接口
```

快速入门

以下例子就演示了一个启动一个server，接收到用户发往公众号的消息然后做处理。

> - 测试公众号可以使用微信公众平台接口测试平台
> - 本地环境开发的话，可以使用 ngrok工具映射出来的公网地址，方便调试。

通过 `go mod` 初始化一个项目，并将 `wechat sdk` 下载下来：

```
go mod init github.com/silenceper/wechat-example
go get -v github.com/silenceper/wechat/v2
```

包含一个文件 `main.go`

> 代码中的配置参数请更改为自己的!

```
package main

import (
    "fmt"
    "net/http"

    wechat "github.com/silenceper/wechat/v2"
    "github.com/silenceper/wechat/v2/cache"
    offConfig "github.com/silenceper/wechat/v2/officialaccount/config"
    "github.com/silenceper/wechat/v2/officialaccount/message"
)

func serveWechat(rw http.ResponseWriter, req *http.Request) {
    wc := wechat.NewWechat()
    //这里本地内存保存access_token，也可选择redis，memcache或者自定cache
    memory := cache.NewMemory()
    cfg := &offConfig.Config{
        AppID:     "xxx",
        AppSecret: "xxx",
        Token:     "xxx",
        //EncodingAESKey: "xxxx",
        Cache: memory,
    }
    officialAccount := wc.GetOfficialAccount(cfg)

    // 传入request和responseWriter
```

```
    server := officialAccount.GetServer(req, rw)
    //设置接收消息的处理方法
    server.SetMessageHandler(func(msg message.MixMessage) *message.Reply {
        //TODO
        //回复消息：演示回复用户发送的消息
        text := message.NewText(msg.Content)
        return &message.Reply{MsgType: message.MsgTypeText, MsgData: text}
    })

    //处理消息接收以及回复
    err := server.Serve()
    if err != nil {
        fmt.Println(err)
        return
    }
    //发送回复的消息
    server.Send()
}

func main() {
    http.HandleFunc("/", serveWechat)
    fmt.Println("wechat server listener at", ":8001")
    err := http.ListenAndServe(":8001", nil)
    if err != nil {
        fmt.Printf("start server error , err=%v", err)
    }
}
```

运行

```
# go run main.go
wechat server listen at :8001
```

配置

通常通过如下配置就可以获取到一个 `officialAccount` 的操作实例了。

```
//使用memcache保存access_token，也可选择redis或自定义cache
wc := wechat.NewWechat()
redisOpts := &cache.RedisOpts{
    Host:        "127.0.0.1:6379",
    Database:    0,
    MaxActive:   10,
    MaxIdle:     10,
    IdleTimeout: 60, //second
}
redisCache := cache.NewRedis(redisOpts)
cfg := &offConfig.Config{
    AppID:     "xxx",
    AppSecret: "xxx",
    Token:     "xxx",
    //EncodingAESKey: "xxxx",
    Cache: redisCache,
}
officialAccount := wc.GetOfficialAccount(cfg)
```

微信公众号支持的配置，如下：

```
//Config config for 微信公众号
type Config struct {
    AppID          string `json:"app_id"`          //appid
    AppSecret      string `json:"app_secret"`      //appsecret
    Token          string `json:"token"`           //token
    EncodingAESKey string `json:"encoding_aes_key"` //EncodingAESKey
    Cache          cache.Cache
}
```

配置说明：

参数	是否必须	说明
AppID	是	微信公众号APP ID
AppSecret	是	微信公众号App Secret
EncodingAESKey	否	如果指定则表示开启AES加密， 消息和结果都会进行解密和加密

参数	是否必须	说明
Cache	否	单独指定微信公众号用到的AccessToken保存的位置，会覆盖全局通过 `wechat.SetCache` 的设置

> 参数配置请前往微信公众号后台获取

缓存

作用： 缓存 `access_token` ，保存在独立服务上可以保证 `access_token` 在有效期内， `access_token` 是公众号的全局唯一接口调用凭据，公众号调用各接口时都需使用 `access_token` 。

> 推荐使用 `redis` 或者 `memcache`

Redis

实例化如下：

```
redisOpts := &cache.RedisOpts{
    Host:        "127.0.0.1:6379", // redis host
    Password:    "",//redis password
    Database:    0, // redis db
    MaxActive:   10, // 连接池最大活跃连接数
    MaxIdle:     10,  //连接池最大空闲连接数
    IdleTimeout: 60, //空闲连接超时时间，单位：second
}
redisCache := cache.NewRedis(redisOpts)
```

Memcache

实例化如下：

```
memCache:=cache.NewMemcache("127.0.0.1:11211")
```

Memory

进程内内存

> 不推荐使用该模式用于生产

实例化如下：

```
memoryCache:=cache.NewMemory()
```

自定义Cache

接口如下：

```
//Cache interface
type Cache interface {
    Get(key string) interface{}
    Set(key string, val interface{}, timeout time.Duration) error
    IsExist(key string) bool
    Delete(key string) error
}
```

文件位置： `/cache/cache.go`

日志

sdk中使用 `github.com/sirupsen/logrus` 来进行日志的输出。
默认的日志参数为：

```
    // 日志输出类型，Text文本
    log.SetFormatter(&log.TextFormatter{})

    // 将日志直接输出到stdout
    log.SetOutput(os.Stdout)

    // 输出日志为Debug 模式
    log.SetLevel(log.DebugLevel)
```

你可以在自己的项目中自定义logrus的输出配置来覆盖sdk中默认的行为
参考: logrus配置文档

跳过接口验证

微信公众号后台在填写接口配置信息时会进行接口的校验以确保接口是否可以正常相应。
如果想要在sdk中关闭该设置，可以通过如下方法：

```
officialAccount := wc.GetOfficialAccount(cfg)
// 传入request和responseWriter
server := officialAccount.GetServer(req, rw)

//关闭接口验证，则validate结果则一直返回true
server.SkipValidate(true)
```

基础接口

微信公众号操作基础接口

获取`access_token`

> 在每个操作实例对象中也有`GetAccessToken`方法
>
> ```
> ak,err:=officialAccount.GetAccessToken()
> ```

替换获取`access_token`的方法

默认在sdk内部是ak获取之后是存放在cache中，如果你有其他系统以及存放了ak的话，只需要实现如下interface就可以了：

```
//AccessTokenHandle AccessToken 接口
type AccessTokenHandle interface {
    GetAccessToken() (accessToken string, err error)
}
```

然后通过wechat对象，进行设置:

```
officialAccount.SetAccessTokenHandle=customAccessTokenHandle
```

获取微信服务器 IP (或IP段)

如果公众号基于安全等考虑，需要获知微信服务器的IP地址列表，以便进行相关限制，可以通过该接口获得微信服务器IP地址列表或者IP网段信息。

获取微信callback IP地址

```
ipList, err:=officialAccount.GetBasic().GetCallbackIP()
```

获取微信API接口 IP地址

```
ipList, err:=officialAccount.GetBasic().GetAPIDomainIP()
```

清理接口调用频次

> 此接口官方有每月调用限制，不可随意调用

```
err:=officialAccount.GetBasic().ClearQuota()
```

消息

> 当普通微信用户向公众账号发消息时，微信服务器将POST消息的XML数据包到开发者填写的URL上。

在快速入门一节中就已经演示了如果收到消息以及对消息进行回复
在SDK中通过 `SetMessageHandler` 方法对消息进行接收以及处理

```
server.SetMessageHandler(func(msg message.MixMessage) *message.Reply {
    //TODO 对接收到的消息以及处理
})
```

其中 `MixMessage` 中包含了一个 `MsgType` 字段，主要会有以下几种类型：

接收普通消息

```
const (
    //MsgTypeText 表示文本消息
    MsgTypeText MsgType = "text"
    //MsgTypeImage 表示图片消息
    MsgTypeImage = "image"
    //MsgTypeVoice 表示语音消息
    MsgTypeVoice = "voice"
    //MsgTypeVideo 表示视频消息
    MsgTypeVideo = "video"
    //MsgTypeShortVideo 表示短视频消息[限接收]
    MsgTypeShortVideo = "shortvideo"
    //MsgTypeLocation 表示坐标消息[限接收]
    MsgTypeLocation = "location"
    //MsgTypeLink 表示链接消息[限接收]
    MsgTypeLink = "link"
    //MsgTypeMusic 表示音乐消息[限回复]
    MsgTypeMusic = "music"
    //MsgTypeNews 表示图文消息[限回复]
    MsgTypeNews = "news"
    //MsgTypeTransfer 表示消息消息转发到客服
    MsgTypeTransfer = "transfer_customer_service"
    //MsgTypeEvent 表示事件推送消息
    MsgTypeEvent = "event"
)
```

接收事件推送

如果 `msg.MsgType` 值为 `MsgTypeEvent` 则表示接收到的是一个事件推送，通过 `msg.EventType` 可以判断事件类型，可以为以下几种：

```
const (
    //EventSubscribe 订阅
    EventSubscribe EventType = "subscribe"
    //EventUnsubscribe 取消订阅
    EventUnsubscribe = "unsubscribe"
    //EventScan 用户已经关注公众号，则微信会将带场景值扫描事件推送给开发者
    EventScan = "SCAN"
    //EventLocation 上报地理位置事件
    EventLocation = "LOCATION"
    //EventClick 点击菜单拉取消息时的事件推送
    EventClick = "CLICK"
    //EventView 点击菜单跳转链接时的事件推送
    EventView = "VIEW"
    //EventScancodePush 扫码推事件的事件推送
    EventScancodePush = "scancode_push"
    //EventScancodeWaitmsg 扫码推事件且弹出“消息接收中”提示框的事件推送
    EventScancodeWaitmsg = "scancode_waitmsg"
    //EventPicSysphoto 弹出系统拍照发图的事件推送
    EventPicSysphoto = "pic_sysphoto"
    //EventPicPhotoOrAlbum 弹出拍照或者相册发图的事件推送
    EventPicPhotoOrAlbum = "pic_photo_or_album"
    //EventPicWeixin 弹出微信相册发图器的事件推送
    EventPicWeixin = "pic_weixin"
    //EventLocationSelect 弹出地理位置选择器的事件推送
    EventLocationSelect = "location_select"
    //EventTemplateSendJobFinish 发送模板消息推送通知
    EventTemplateSendJobFinish = "TEMPLATESENDJOBFINISH"
    //EventWxaMediaCheck 异步校验图片/音频是否含有违法违规内容推送事件
    EventWxaMediaCheck = "wxa_media_check"
)
```

被动回复用户消息

当收到用户向公众号发送的消息后可以对发过来的进行一次回复：

现支持一下几种回复： `文本` 、 `图片` 、 `图文` 、 `语音` 、 `视频` 、 `音乐`

回复文本

```
server.SetMessageHandler(func(msg message.MixMessage) *message.Reply {
    //TODO 演示回复文本
        //回复消息：演示回复用户发送的消息
        text := message.NewText(msg.Content)
        return &message.Reply{MsgType: message.MsgTypeText, MsgData: text}
})
```

回复图片

```
image := message.NewImage(mediaID)
return &message.Reply{MsgType: message.MsgTypeImage, MsgData: image}
```

其中 `mediaID` 为媒体资源 ID ,可以通过素材管理(`Material`)中的接口进行上传

回复图文

```
article1 := message.NewArticle("测试图文1", "图文描述", "http://图片链接地址",
"http://图文链接地址")
articles := []*message.Article{article1}
news := message.NewNews(articles)
return &message.Reply{MsgType: message.MsgTypeNews, MsgData: news}
```

回复语音

```
voice := message.NewVoice(mediaID)
return &message.Reply{MsgType: message.MsgTypeVoice, MsgData: voice}
```

其中 `mediaID` 为媒体资源 ID ,可以通过素材管理(`Material`)中的接口进行上传

回复视频

```
video := message.NewVideo(mediaID, "标题", "描述")
return &message.Reply{MsgType: message.MsgTypeVideo, MsgData: video}
```

其中 `mediaID` 为媒体资源 ID ,可以通过素材管理(`Material`)中的接口进行上传

回复音乐

```
music := message.NewMusic("标题", "描述", "音乐链接", "高质量音乐链接", "缩略图的媒体id")
return &message.Reply{MsgType: message.MsgTypeMusic, MsgData: music}
```

`NewMusic` 参数说明：

参数	是否必须	描述
Title	否	音乐标题
Description	否	音乐描述
MusicURL	否	音乐链接
HQMusicUrl	否	高质量音乐链接，WIFI环境优先使用该链接播放音乐
ThumbMediaId	否	缩略图的媒体id， 通过素材管理中的接口上传多媒体文件，得到的id

其中不必须的参数可以填写空字符串 `""`

多客服消息转发

请参考 多客服消息转发

多客服消息转发

```
transferCustomer := message.NewTransferCustomer("")
return &message.Reply{MsgType: message.MsgTypeTransfer, MsgData: transferCustomer}
```

其中 `NewTransferCustomer` **如果参数可选，表示指定客服账号**

群发消息

获取群发操作实例

```
oa := wc.GetOfficialAccount(cfg)
bd:=oa.GetBroadcast()
//TODO bd.SendText
```

发送对象

- 发送方法的第一个参数为`broadcast.User`对象，为nil则表示发送给所有人
- `broadcast.User{TagID:1}`：根据tagID发送
- `broadcast.User{OpenID:[]string{"openid-1","openid-2"}}`：根据openid发送

群发消息类型

发送文本消息

```
bd.SendText(user *User, content string)
```

发送图文

```
bd.SendNews(user *User, mediaID string,ignoreReprint bool)
```

发送图片

```
bd.SendImage(user *User, images *Image)
```

发送语音

```
bd.SendVoice(user *User, mediaID string)
```

发送视频

```
bd.SendVideo(user *User, mediaID string,title,description string)
```

素材管理

获取素材操作实例

```
oa := wc.GetOfficialAccount(cfg)
m:=oa.GetMaterial()
```

新增永久图文素材

```
AddNews(articles []*Article)
```

新增其他类型永久素材(除视频)

```
AddMaterial(mediaType MediaType, filename string)
```

新增永久视频素材

```
AddVideo(filename, title, introduction string)
```

删除永久素材

```
DeleteMaterial(mediaID string)
```

批量获取永久素材

```
BatchGetMaterial(permanentMaterialType PermanentMaterialType, offset, count int64)
```

获取永久图文素材

```
GetNews(id string) ([]*Article, error)
```

JSSDK

获取js-sdk操作实例

```
oa := wc.GetOfficialAccount(cfg)
j:=oa.GetJs()
```

获取js配置

```
GetConfig(uri string) (config *Config, err error)
```

其中 `Config` 结果为:

```
// Config 返回给用户jssdk配置信息
type Config struct {
    AppID     string `json:"app_id"`
    Timestamp int64  `json:"timestamp"`
    NonceStr  string `json:"nonce_str"`
    Signature string `json:"signature"`
}
```

替换 `js-ticket` 取值方式

默认js-ticket是存放在sdk设置的cache，如果需要自定义取值，可以实现 `credential.JsTicketHandle` 接口:

```
//JsTicketHandle js ticket获取
type JsTicketHandle interface {
    //GetTicket 获取ticket
    GetTicket(accessToken string) (ticket string, err error)
}
```

然后通过 `js.SetJsTicketHandle(ticketHandle credential.JsTicketHandle)` 进行设置

菜单管理

获取菜单操作实例

```
oa := wc.GetOfficialAccount(cfg)
m:=oa.GetMenu()
```

获取当前菜单设置

```
GetMenu() (resMenu ResMenu, err error)
```

其中 `ResMenu` 结果为:

```
//ResMenu 查询菜单的返回数据
type ResMenu struct {
    util.CommonError

    Menu struct {
        Button []Button `json:"button"`
        MenuID int64    `json:"menuid"`
    } `json:"menu"`
    Conditionalmenu []resConditionalMenu `json:"conditionalmenu"`
}
```

添加菜单（struct方式）

```
SetMenu(buttons []*Button) error
```

添加菜单（JSON方式）

直接传入json

```
SetMenuByJSON(jsonInfo string) error
```

删除菜单

```
DeleteMenu() error
```

添加个性化菜单（struct方式）

```
AddConditional(buttons []*Button, matchRule *MatchRule) error
```

添加个性化菜单（JSON方式）

直接传入json

```
AddConditionalByJSON(jsonInfo string) error
```

测试个性化菜单匹配结果

```
MenuTryMatch(userID string) (buttons []Button, err error)
```

删除个性化菜单

```
DeleteConditional(menuID int64) error
```

获取自定义菜单配置接口

```
GetCurrentSelfMenuInfo() (resSelfMenuInfo ResSelfMenuInfo, err error)
```

其中 `ResSelfMenuInfo` 结果为：

```
type ResSelfMenuInfo struct {
    util.CommonError

    IsMenuOpen   int32 `json:"is_menu_open"`
    SelfMenuInfo struct {
        Button []SelfMenuButton `json:"button"`
    } `json:"selfmenu_info"`
}
```

用户管理

获取用户管理操作实例

```
oa := wc.GetOfficialAccount(cfg)
u:=oa.GetUser()
```

获取用户基本信息

```
GetUserInfo(openID string) (userInfo *Info, err error)
```

设置用户备注名

```
UpdateRemark(openID, remark string) (err error)
```

返回用户列表

```
ListUserOpenIDs(nextOpenid ...string) (*OpenidList, error)
```

其中返回结果为：

```
/ OpenidList 用户列表
type OpenidList struct {
    Total int `json:"total"`
    Count int `json:"count"`
    Data  struct {
        OpenIDs []string `json:"openid"`
    } `json:"data"`
    NextOpenID string `json:"next_openid"`
}
```

新版客服消息

TODO

模板消息

获取模板消息实例

```
oa := wc.GetOfficialAccount(cfg)
m:=oa.GetTemplate()
```

发送模板消息

```
Send(msg *TemplateMessage) (msgID int64, err error)
```

其中 `TemplateMessage` 结构为：

```
type TemplateMessage struct {
    ToUser      string                          `json:"touser"`          // 必须, 接受者OpenID
    TemplateID string                          `json:"template_id"`     // 必须, 模版ID
    URL         string                          `json:"url,omitempty"`   // 可选, 用户点击后跳转的URL, 该URL必须处于开发者在公众平台网站中设置的域中
    Color       string                          `json:"color,omitempty"` // 可选, 整个消息的颜色, 可以不设置
    Data        map[string]*TemplateDataItem `json:"data"`            // 必须, 模板数据

    MiniProgram struct {
        AppID    string `json:"appid"`    //所需跳转到的小程序appid（该小程序appid必须与发模板消息的公众号是绑定关联关系）
        PagePath string `json:"pagepath"` //所需跳转到小程序的具体页面路径, 支持带参数,（示例index?foo=bar）
    } `json:"miniprogram"` //可选,跳转至小程序地址
}
```

小程序

获取小程序操作对象

```
wc := wechat.NewWechat()
memory := cache.NewMemory()
cfg := &miniConfig.Config{
    AppID:     "xxx",
    AppSecret: "xxx",
    Cache: memory,
}
miniprogram := wc.GetMiniProgram(cfg)
//TODO 调用对应接口
miniprogram.GetAnalysis().GetAnalysisDailyRetain()
```

微信登录

获取操作实例

```
mini := wc.GetMiniProgram(cfg)
a:=mini.GetAuth()
```

根据 jscode 获取用户 session 信息

```
Code2Session(jsCode string) (result ResCode2Session, err error)
```

消息解密

获取操作实例

```
mini := wc.GetMiniProgram(cfg)
a:=mini.GetAuth()
```

解密数据

```
Decrypt(sessionKey, encryptedData, iv string) (*PlainData, error)
```

其中结果为：

```
type PlainData struct {
    OpenID    string `json:"openId"`
    UnionID   string `json:"unionId"`
    NickName  string `json:"nickName"`
    Gender    int    `json:"gender"`
    City      string `json:"city"`
    Province  string `json:"province"`
    Country   string `json:"country"`
    AvatarURL string `json:"avatarUrl"`
    Language  string `json:"language"`
    PhoneNumber     string `json:"phoneNumber"`
    PurePhoneNumber string `json:"purePhoneNumber"`
    CountryCode     string `json:"countryCode"`
    Watermark struct {
        Timestamp int64  `json:"timestamp"`
        AppID     string `json:"appid"`
    } `json:"watermark"`
}
```

根据需要取用户信息还是手机号信息

小程序码

获取操作实例

```
mini := wc.GetMiniProgram(cfg)
qr:=mini.GetQrcode()
```

获取小程序二维码，适用于需要的码数量较少的业务场景

```
CreateWXAQRCode(coderParams QRCoder) (response []byte, err error)
```

获取小程序码，适用于需要的码数量较少的业务场景

```
GetWXACode(coderParams QRCoder) (response []byte, err error)
```

获取小程序码，适用于需要的码数量极多的业务场景

```
GetWXACodeUnlimit(coderParams QRCoder) (response []byte, err error)
```

云开发

Tencent Cloud Base 文档

使用说明

初始化配置

```
wc := wechat.NewWechat()
//使用memcache保存access_token，也可选择redis或自定义cache
memCache:=cache.NewMemcache("127.0.0.1:11211")

//配置小程序参数
config := &wechat.Config{
    AppID:     "your app id",
    AppSecret: "your app secret",
    Cache:     memCache,
}
miniprogram := wc.GetMiniProgram(cfg)
wcTcb := miniprogram.GetTcb()
```

举例

触发云函数

```
res, err := wcTcb.InvokeCloudFunction("test-xxxx", "add", `{"a":1,"b":2}`)
if err != nil {
    panic(err)
}
```

更多使用方法参考GODOC

开放平台

状态：beta

官方文档

快速入门

```
wc := wechat.NewWechat()
memory := cache.NewMemory()
cfg := &openplatform.Config{
    AppID:     "xxx",
    AppSecret: "xxx",
    Token:     "xxx",
    EncodingAESKey: "xxx",
    Cache: memory,
}

//授权的第三方公众号的appID
appID := "xxx"
// 下面文档中提到的openPlatform都是这个变量
openPlatform := wc.GetOpenPlatform(cfg)
officialAccount := openPlatform.GetOfficialAccount(appID)

// 传入request和responseWriter
server := officialAccount.GetServer(req, rw)
//设置接收消息的处理方法
server.SetMessageHandler(func(msg message.MixMessage) *message.Reply {
    if msg.InfoType == message.InfoTypeVerifyTicket {
        componentVerifyTicket, err := openPlatform.SetComponentAccessToken(msg.C
omponentVerifyTicket)
        if err != nil {
            log.Println(err)
            return nil
        }
        //debug
        fmt.Println(componentVerifyTicket)
        rw.Write([]byte("success"))
        return nil
    }
    //handle other message
    //
```

```
    return nil
})

//处理消息接收以及回复
err := server.Serve()
if err != nil {
    fmt.Println(err)
    return
}
//发送回复的消息
server.Send()
```

代第三方公众号 - 发起网页授权

```
//第三方公众号appid
appid := ""
officialAccount := openPlatform.GetOfficialAccount(appid)
oauth := officialAccount.PlatformOauth()
//重定向到微信oauth授权登录
oauth.Redirect(rw, req, callback, "snsapi_userinfo", "", appid)
```

代第三方公众号 - 通过网页授权的code 换取access_token

```
officialAccount := openPlatform.GetOfficialAccount(appid)
componentAccessToken, err := openPlatform.GetComponentAccessToken()
if err != nil {
    fmt.Println(err)
}
accessToken, err := officialAccount.PlatformOauth().GetUserAccessToken(code, app
id, componentAccessToken)
if err != nil {
    fmt.Println(err)
}
fmt.Println(accessToken)
// 通过accessToken获取用户信息请参考微信公众号的业务
```

维护AuthrToken

AuthrToken是平台代第三方公众号调用微信接口的凭据，通过第三方公众号授权给平台时得到的refreshToken来获取

```
func CheckAuthrToken(appid, refreshToken string) {
    // 获取authrToken
    token, err := openPlatform.GetAuthrAccessToken(appid)
    if err != nil {
        fmt.Println(err)
    }
    if token == "" {
        openPlatform.RefreshAuthrToken(appid, refreshToken)
    }
}
```

代第三方公众号 - 调用微信接口（以发送微信模板消息为例）

平台代第三方公众号调用微信接口，需要在调用前确保AuthrToken有效，其余操作与公众号一致。

```
import "github.com/silenceper/wechat/v2/officialaccount/message"

// 在代第三方公众号调用微信接口的时候，需要确保AuthrToken有效
// 这里的appid是第三方公众号的appid
CheckAuthrToken(appid, refreshToken)
msg := &message.TemplateMessage{
    ToUser:      openid,
    TemplateID: templateID,
    URL:         url,
    Data:        data,
}
officialAccount := openPlatform.GetOfficialAccount(appid)
template := message.NewTemplate(officialAccount.GetContext())
msgID, err := template.Send(msg)
if err != nil {
    fmt.Println(err)
}
fmt.Println(msgID)
```

> 微信的部分接口（如：获取jsconfig信息）区分了第三方平台调用和公众号直接调用的地址，在文档下方单独进行说明。

代第三方公众号 - 获取jsconfig信息

```
CheckAuthrToken(appid, refreshToken)
jsConfig, err := openPlatform.GetOfficialAccount(appid).PlatformJs().GetConfig(u
ri, appid)
if err != nil {
    fmt.Println(err)
}
fmt.Println(jsConfig)
```

全网发布校验

微信第三方平台进行全网发布的时候，会有一个全网发布接入检测的过程。
官方文档

```
wc := wechat.NewWechat()
memory := cache.NewMemory()
cfg := &openplatform.Config{
    AppID:     "xxx",
    AppSecret: "xxx",
    Token:     "xxx",
    EncodingAESKey: "xxx",
    Cache: memory,
}

//授权的第三方公众号的appID
appID := "xxx"
// 下面文档中提到的openPlatform都是这个变量
openPlatform := wc.GetOpenPlatform(cfg)
officialAccount := openPlatform.GetOfficialAccount(appID)

// 传入request和responseWriter
server := officialAccount.GetServer(req, rw)
//设置接收消息的处理方法
server.SetMessageHandler(func(msg message.MixMessage) *message.Reply {
    switch msg.InfoType {
    case message.InfoTypeVerifyTicket:
        // 在这里处理推送的VerifyTicket
        // 测试验证票据推送流程
        rw.Write([]byte("success"))
    case message.InfoTypeAuthorized:
        // 微信会推送测试号的query_auth_code过来，需要在这里获取到测试号的AuthrT
oken
```

```
        // 参照开放平台的`维护AuthrToken`小节
    }
    switch msg.MsgType {
    case message.MsgTypeText:
        if msg.Content == "TESTCOMPONENT_MSG_TYPE_TEXT" {
            // 测试公众号处理用户消息
            return &message.Reply{
                MsgType: message.MsgTypeText,
                MsgData: message.NewText("TESTCOMPONENT_MSG_TYPE_TEXT_callback"
),
            }
        }
        // 测试公众号使用客服消息接口处理用户消息
        if strings.HasPrefix(msg.Content, "QUERY_AUTH_CODE") {
            // 立即回复空串
            rw.Write([]byte(""))
            var data = strings.Split(msg.Content, ":")
            if len(data) == 2 {
                // 调用客服接口回复消息
                customerMsg := message.NewCustomerTextMessage(string(msg.FromUse
rName), fmt.Sprintf("%s_from_api", data[1]))
                CheckAuthrToken(appid, refreshToken)
                officialAccount := openPlatform.GetOfficialAccount(appid)
                msgManager := message.NewMessageManager(officialAccount.GetConte
xt())
                msgManager.Send(msg)
            }
        }
    }
    return nil
})

//处理消息接收以及回复
err := server.Serve()
if err != nil {
    fmt.Println(err)
    return
}
//发送回复的消息
server.Send()
```

公众号授权流程

小程序或者公众号授权给第三方平台的流程
官方文档

扫码授权

```
// 获取公众号扫码授权页面链接
loginPageURL, err := openPlatform.GetComponentLoginPage(redirectURI, authType, "")
// ...引导用户扫码授权
// 注意: 这里微信会校验跳转到授权页的referer,必须与第三方平台后台设置的`登录授权的发起页域名`一致
// --------
// 通过授权回调获取到的authCode换取公众号或小程序的接口调用凭据和授权信息
authInfo, err := openPlatform.QueryAuthCode(authCode)
// ...处理公众号授权后的逻辑, 存储refreshToken
```