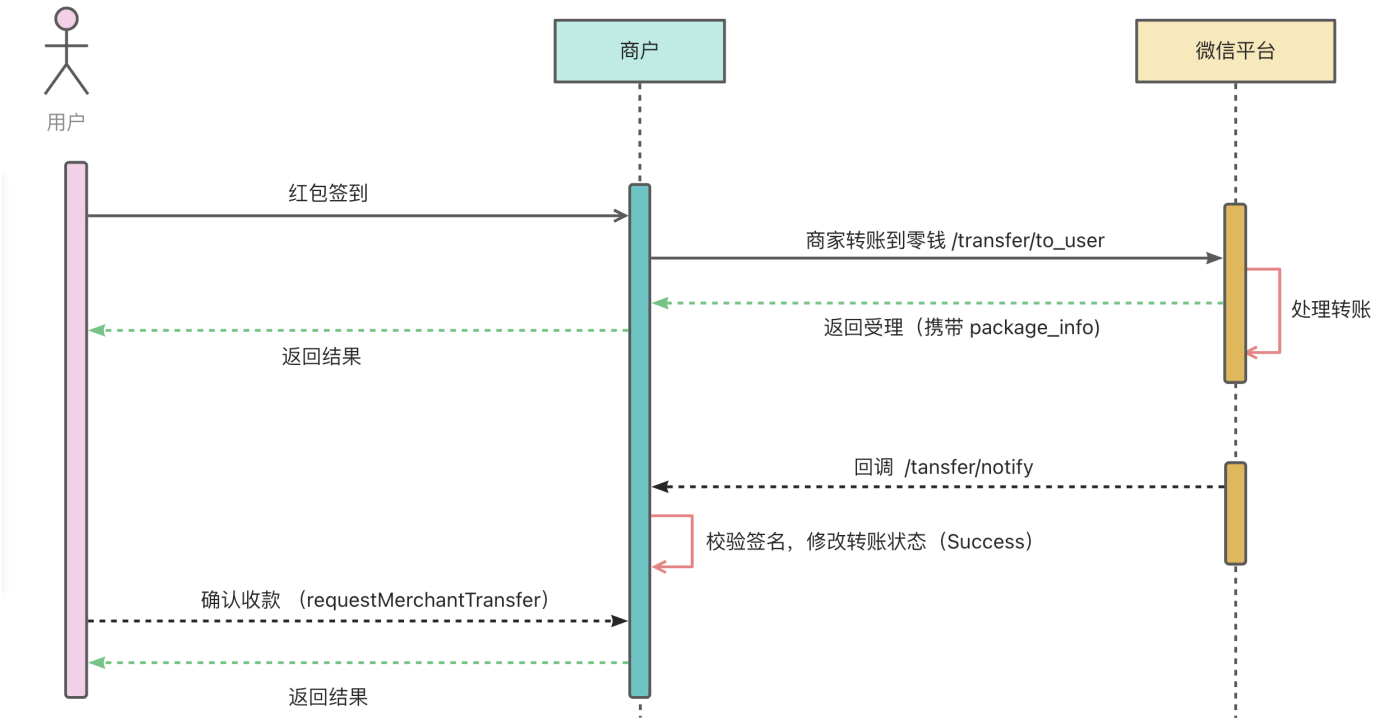
背景

1. 阅读链接的文档，理解产品形态 微信支付V3：商家转账到零钱 API 文档
2. 你模拟一个商户，接入商家转账产品，完成功能：打开微信小程序之后，用户可以领取奖励，收到钱。（类似于淘宝签到领红包）
3. 交付件：设计文档、核心代码，以及对产品文档的改进建议

如何启动

- 数据库：`docker compose up`
- 后端：`go run main.go`
- 小程序：微信开发者平台
- 内网穿透：`./natapp --authtoken=5xxx`

整体架构



1. 用户发起红包签到

- 用户在微信小程序内点击“红包签到”，发起签到请求。
- 小程序向 **商户**（后端）发起 `/transfer/to_user` 接口请求。

2．商户服务器处理转账

- 商户服务器接收到请求，生成 `转账单号`，保存请求，调用微信商户转账API，转账到用户零钱。
- 微信平台处理转账，立即返回一个受理响应（同步），响应中携带 `package_info`（用于后续收款拉起）。

3．返回结果给用户

- 商户服务器将转账请求的受理结果（包含package_info）返回给小程序端，前端可以用它拉起收款页面，提示用户“签到成功”。

4．微信平台异步回调通知（Notify）

- 微信平台**转账处理完成后**，会异步回调商户的 `/transfer/notify` 接口，推送转账单据终态（如转账成功、失败、撤销）。
- 商户服务器接收到回调，首先**验签**（保证消息来源可靠），然后根据通知内容**修改本地转账记录状态**（置为“Success”）。

5．用户确认收款

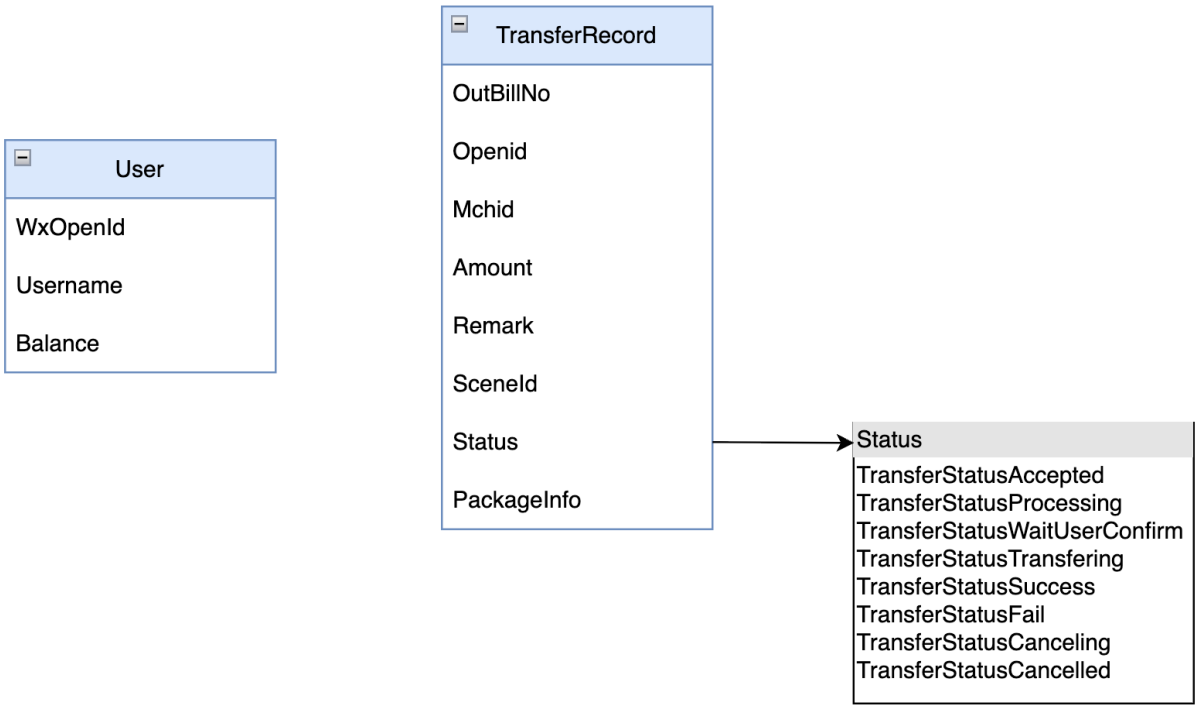
- 用户在微信收款页面点击**确认收款**，小程序端可以调用 requestMerchantTransfer 拉起收款界面。（模拟调用的是商户的 `/transfer/confirm` 接口）
- 收款完成后，返回结果给用户，用户的余额增加。

技术栈

- 采用 **DDD（领域驱动设计） 架构**，项目分为 Domain、Repository、Service、Web 层，**实现业务逻辑与基础设施的解耦**，支持模块化扩展和后期重构。 设计原则：单一职责，依赖注入（DI），接口封装
- 后端：采用 `Gin` 作为 Web 框架，具有路由和中间件支持，开发效率高
- 前端：`微信小程序` ，利用微信的原生API 支持
- 数据库：`MySQL`，存储转账记录、用户等信息
- 单元测试：`Mock`
- 版本管理：使用 `Github` 进行版本管理

主要功能

数据模型设计



用户（User）

- WxOpenId（微信 openid，唯一索引）
- Balance ：用户余额

转账记录（TransferRecord）

- OutBillNo：商户订单号
- PackageInfo：跳转领取页面的package信息
- Status：转账的状态

商户信息配置（MchConfig）

```
type MchConfig struct {
    mchId                      string           // 商户号
    certificateSerialNo        string           // 商户API证书序列号
    privateKeyFilePath         string           // 商户API证书对应的私钥文
件路径
    wechatPayPublicKeyId       string           // 微信支付公钥ID
    wechatPayPublicKeyFilePath string           // 微信支付公钥文件路径
    privateKey                 *rsa.PrivateKey  // 商户API证书对应的私钥
    wechatPayPublicKey         *rsa.PublicKey   // 微信支付公钥
}
```

通过 CreateMchConfig 方法进行初始化，读取本地密钥文件，加载为结构体属性，后续直接调用。

- 私钥签名：`func SignSHA256WithRSA(source string, privateKey *rsa.PrivateKey) (signature string, err error)`
- 公钥验证：`func VerifySHA256WithRSA(source string, signature string, publicKey *rsa.PublicKey) error`
- 构建请求头中的 Authorization：`BuildAuthorization`

```
authorization, err := BuildAuthorization(
   mchid,
   certificateSerialNo,
   privateKey,
   "POST",
   "/v3/pay/transactions/jsapi",
   body,
)
req.Header.Set("Authorization", authorization)
```

- 提取 Response 中的 Body：`func ExtractResponseBody(response *http.Response) ([]byte, error)`
- 微信支付回调校验(验证签名信息)：`ValidateResponse`

```
func ValidateResponse(
    wechatpayPublicKeyId string,
    wechatpayPublicKey *rsa.PublicKey,
    headers *http.Header,
    body []byte,
) error
```

- 返回错误：`ApiException`

```
func NewApiException(statusCode int, header http.Header, body
[]byte) error
```

关键算法设计

发起转账

发起转账 ： `func (*t* *TransferHandler) InitiateTransfer(*ctx* *gin.Context)`

- 输入：Openid，Amount（转账金额），Remark（备注），Time（时间戳）
- 步骤
  1. 校验参数完整性，完善其他参数设置
  2. 生成 `OutBillNo（商户单号）`
  3. 生成 `packageInfo` （原本应该是调用微信商户转账API 的 Response 中的，这里模拟，直接商户端生成了）
  4. 保存下转账记录（落库）
  5. 构造微信商户转账需要的 TransferToUserRequest 对象，调用 API 接口
  6. **立即（同步）**返回一个组装好的 TransferToUserResponse 对象，此处的转账状态为 `TransferStatusProcessing`
  7. **异步修改转账状态** 为 `TransferStatusTransfering`，表示微信平台正在处理转账

核心代码：

```
if err := ctx.ShouldBind(&req); err != nil {
        ctx.JSON(http.StatusBadRequest, gin.H{"error": "参数不合法: " +
err.Error()})
```

```
        return
    }

// 生成唯一outbillno, packageInfo并保存转账请求
    outbillno := t.svc.GenerateOutBillNo(req.Openid, req.Amount)
    packageInfo := generatePackageInfo(req.Openid, req.Time)

// 保存转账记录
err := t.svc.AddTransferRequest(ctx, requestRecord)

// 调用微信 API
_, err = t.svc.TransferToUser(t.client.MchConfig, request)

// 异步处理状态
ctx.JSON(http.StatusOK, response)

go func() {
    time.Sleep(30 * time.Second)
    t.svc.UpdateTransferStatus(ctx, outbillno,
domain.TransferStatusTransfering)
}()
```

演示

微信回调

微信回调：`func (t *TransferHandler) TransferNotify(ctx *gin.Context)`

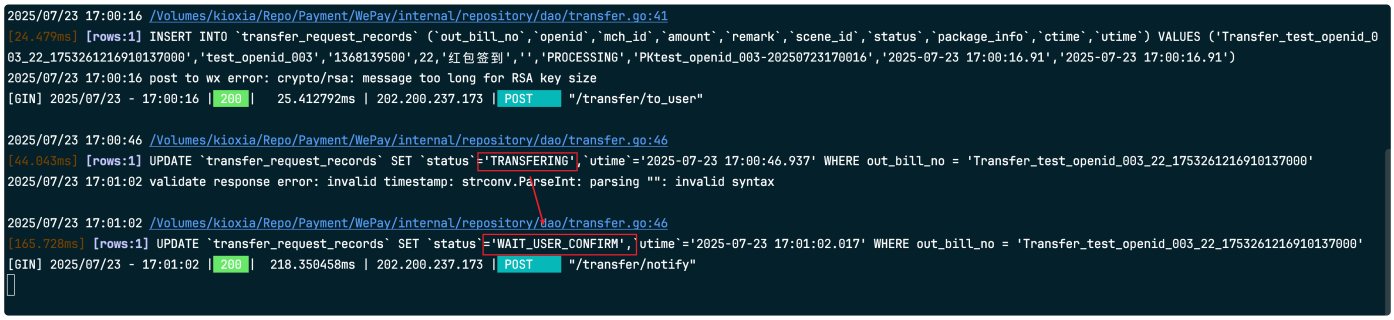
- 输入：OutBillNo
- 步骤
  1. 接收和校验参数
  2. 调用 `wxpay_utility.ValidateResponse` 对微信回调进行验签，防止伪造
  3. 更新转账状态为 `WaitUserConfirm`， 等待用户确认收款
  4. 返回 `Http.StatusOK` ,告诉微信已成功接收通知。

核心代码：

```
var req struct {
    OutBillNo string `json:"out_bill_no"  binding:"required"`
}
if err := ctx.ShouldBindJSON(&req); err ≠ nil {
    ctx.JSON(400, gin.H{"code": "FAIL", "message": "invalid body"})
    return
}
// 2. 校验回调请求
headers := ctx.Request.Header
body, err := io.ReadAll(ctx.Request.Body)
if err ≠ nil {
    ctx.JSON(http.StatusInternalServerError, err.Error())
    return
}
err =
wxpay_utility.ValidateResponse(t.client.MchConfig.WechatPayPublicKeyI
d(), t.client.MchConfig.WechatPayPublicKey(), &headers, body)

// 更新   requestRecord 状态
err = t.svc.UpdateTransferStatus(ctx, req.OutBillNo,
domain.TransferStatusWaitUserConfirm)
```

演示



确认转账

确认转账：`func (t *TransferHandler) ConfirmTransfer(ctx *gin.Context)`

- 输入：Mchid，Appid，PackageInfo
- 步骤

1. 接收参数并校验
2. 根据小程序传过来的 `PackageInfo` 调用 GetTransferRecordByPackageInfo 拿到对应的转账记录
3. 转账记录上的状态为 `TransferStatusWaitUserConfirm` ，则更新余额，更新转账记录的状态为 `TransferStatusSuccess`
4. 否则，返回失败。

核心代码

```
var req struct {
    MchId       string `json:"mch_id" binding:"required"`
    Appid       string `json:"appid" binding:"required"`
    PackageInfo string `json:"package_info" binding:"required"`
}
if err := ctx.ShouldBind(&req); err ≠ nil {
    ctx.JSON(http.StatusBadRequest, gin.H{"error": "参数不合法: " +
err.Error()})
    return
}
// 获取转账记录
record, err := t.svc.GetTransferRecordByPackageInfo(ctx,
req.PackageInfo)

if record.Status == domain.TransferStatusWaitUserConfirm {
    // 如果状态为 TransferStatusWaitUserConfirm，则更新用户余额
    err := t.userSvc.UpdateBalance(ctx, record.Openid, record.Amount)
    if err ≠ nil {
        ctx.JSON(http.StatusInternalServerError, "")
        log.Printf("更新用户余额失败: %v", err)
        return
    }
    err = t.svc.UpdateTransferStatus(ctx, record.OutBillNo,
domain.TransferStatusSuccess)
    if err ≠ nil {
        ctx.JSON(http.StatusInternalServerError, "")
        log.Printf("更新转账状态失败: %v", err)
        return
    }
    ctx.JSON(http.StatusOK, gin.H{"message": "转账确认成功"})
```

```
} else {
    ctx.JSON(http.StatusInternalServerError, "")
}
```

演示

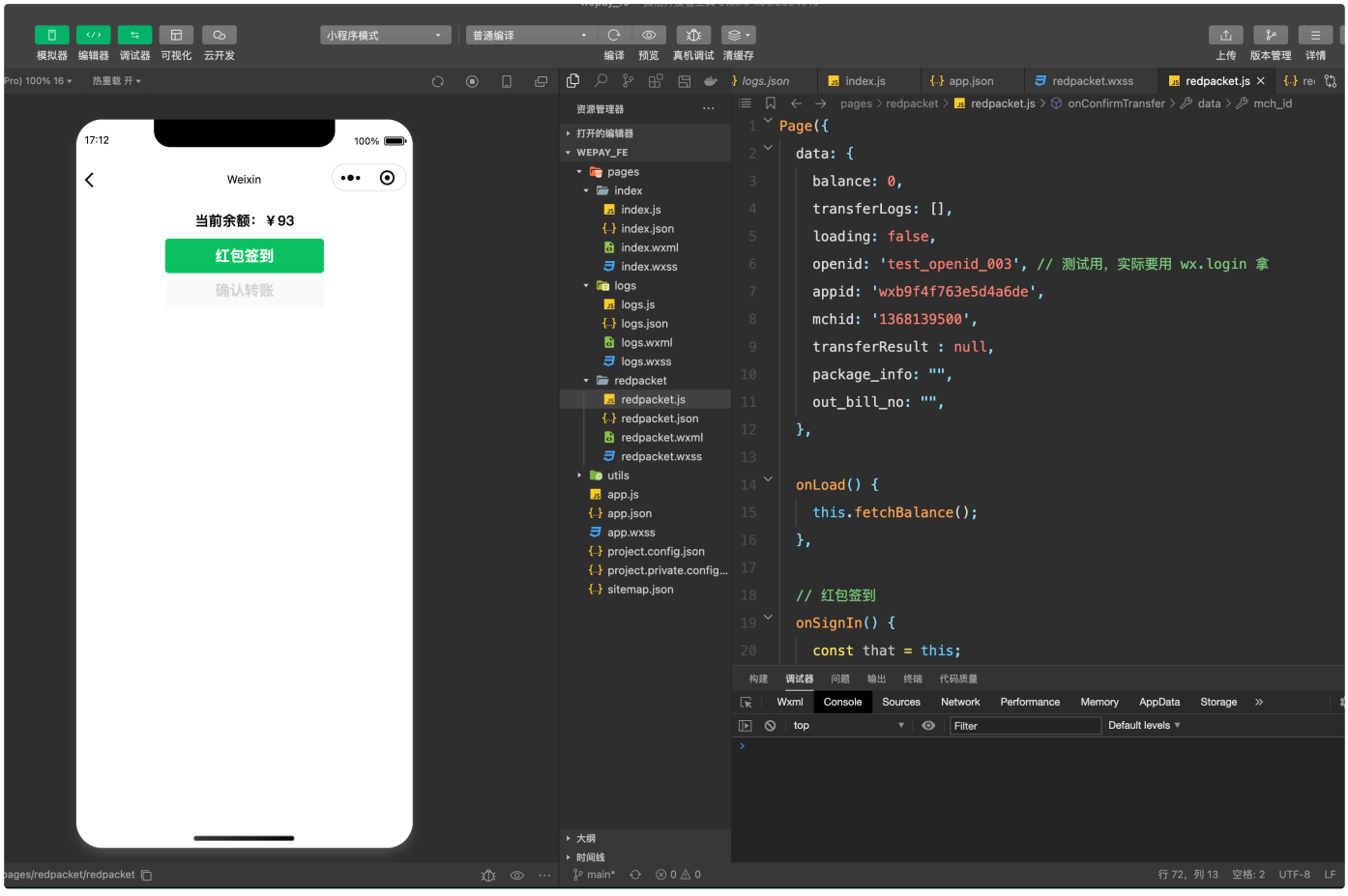
```
2025/07/23 17:01:02 /Volumes/kioxia/Repo/Payment/WePay/internal/repository/dao/transfer.go:46
[165.728ms] [rows:1] UPDATE `transfer_request_records` SET `status`='WAIT_USER_CONFIRM',`utime`='2025-07-23 17:01:02.017' WHERE out_bill_no = 'Transfer_test_openid_003_22_1753261216910137000'
[GIN] 2025/07/23 - 17:01:02 | 200 |  218.350458ms | 202.200.237.173 | POST     "/transfer/notify"

2025/07/23 17:02:30 /Volumes/kioxia/Repo/Payment/WePay/internal/repository/dao/transfer.go:68
[41.765ms] [rows:1] SELECT * FROM `transfer_request_records` WHERE package_info = 'PKtest_openid_003-20250723170016' ORDER BY `transfer_request_records`.`id` LIMIT 1

2025/07/23 17:02:30 /Volumes/kioxia/Repo/Payment/WePay/internal/repository/dao/user.go:49
[24.715ms] [rows:2] INSERT INTO `users` (`wx_open_id`,`username`,`balance`) VALUES ('test_openid_003','test_openid_003',22) ON DUPLICATE KEY UPDATE `balance`=balance + 22

2025/07/23 17:02:30 /Volumes/kioxia/Repo/Payment/WePay/internal/repository/dao/transfer.go:46
[5.321ms] [rows:1] UPDATE `transfer_request_records` SET `status`='SUCCESS',`utime`='2025-07-23 17:02:30.153' WHERE out_bill_no = 'Transfer_test_openid_003_22_1753261216910137000'
[GIN] 2025/07/23 - 17:02:30 | 200 |   92.406292ms | 202.200.237.173 | POST     "/transfer/confirm"
2025/07/23 17:02:30 openid test_openid_003
```

前端设计



1. 用户在小程序端点“红包签到”，触发该请求。
2. 后端 `/transfer/to_user` 接口收到参数，发起一笔转账业务，并返回唯一的 `package_info` 标识。
3. 前端拿到 package_info，后续可用于 **确认转账**

```
wx.request({
  url: 'http://wepay.selfknow.cn/transfer/to_user',
  method: 'POST',
  header: { 'content-type': 'application/json' },
  data: {
    openid: that.data.openid,
    amount:  Math.floor(Math.random() * 49) + 1,  // 分
    remark: '红包签到',
    time: time,
  },
  success: (res) => {
    console.log(res);
    if (res.data && res.data.package_info) {
      wx.showToast({ title: res.data.msg || '签到成功', icon: 'success'
});
      this.setData({package_info: res.data.package_info})
      console.log(res.data.out_bill_no);
    } else {
      wx.showToast({ title: res.data.msg || '签到失败', icon: 'none'
});
    }
  },
  fail: (err) => {
    wx.showToast({ title: '网络错误', icon: 'none' });
  },
  complete: () => {
    this.setData({ loading: false });
  }
});
```

总结

收获

- 首先通过查阅资料、代码，学习别人是如何写微信支付的
- 结合自己的习惯，整理思维逻辑，实现一个大体的框架
- 逐步实现每一个功能

- 修补bug，完善细节，梳理文档
- 分析，优化，提升。

学到了很多东西，在这个过程中不断地发现问题，解决问题，又发现问题，又解决问题.... 能力慢慢得到了提高，对业务的理解又提高了一个认识，同时也有一些感想。

完成优先于完美 ： 先简单的跑起来，然后再去做细节上的调整

不谋全局者，不足以谋一域：刚开始做的时候，想着先实现一个功能，后面的整体框架以后再说。做着做着就在想这个应该放在哪，那个应该放在哪，逻辑慢慢的就搞混了。 后面就重新梳理思路，把整个框架梳理清楚之后，再去逐步实现代码。

Commits on Jul 23, 2025

docs: update doc
LzcGeorge committed 10 minutes ago

fix: refactor client initialization
LzcGeorge committed 7 hours ago

fix: correct transfer status update
LzcGeorge committed 17 hours ago

fix: add package_info field and improve the entire transfer flow
LzcGeorge committed 17 hours ago

Commits on Jul 22, 2025

feat: add mocks for testing, refactor to interface-based service
LzcGeorge committed 20 hours ago

Commits on Jul 20, 2025

feat: implement user balance management
LzcGeorge committed 3 days ago

feat: implement confirm transfer
LzcGeorge committed 3 days ago

feat: implement transfer notify and add WePay miniProgram
LzcGeorge committed 3 days ago

feat: new client configuration
LzcGeorge committed 4 days ago

Commits on Jul 19, 2025

feat: add transfer function
LzcGeorge committed 4 days ago

Commits on Jul 16, 2025

init
LzcGeorge committed last week

Typo

resource中ciphertext解密后字段

out_bill_no 必填 string(32)

【商户单号】商户系统内部的商家单号，在商户系统内部唯一

transfer_bill_no 必填 string(64)

【商家转账订单号】微信单号，微信商家转账系统返回的唯一标识

state 必填 string(32)

【单据状态】商家转账订单状态
ACCEPTED：单据已受理
PROCESSING：单据处理中，转账结果尚未明确，如一直处于此状态，建议检查账户余额是否足够
WAIT_USER_CONFIRM：待收款用户确认，可拉起微信收款确认页面进行收款确认
TRANSFERING：转账中，转账结果尚未明确，可拉起微信收款确认页面再次重试确认收款
SUCCESS：转账成功
FAIL：转账失败
CANCELING：撤销中
CANCELLED：已撤销

对resource中ciphertext进行解密后，得到的资源对象示例

```
{
    "out_bill_no": "plfk2020042013",
    "transfer_bill_no":"1330000071100999991182020050700019480001",
    "state": "SUCCESS",
    "mch_id": "1900001109",
    "transfer_amount": 2000,
    "openid": "o-MYE421800elYMDE34nYD456Xoy",
    "fail_reason:" "PAYEE_ACCOUNT_ABNORMAL",
    "create_time": "2015-05-20T13:29:35+08:00",
    "update_time": "2023-08-15T20:33:22+08:00"
}
```

- 应该是 `"fail_reason":`
- 地址：https://pay.weixin.qq.com/doc/v3/merchant/4012712115

后续

3、对回调通知内容进行解密

为了保证业务信息的安全性，微信支付将业务信息进行了AES-256-GCM加密，并通过参数resource将加密信息回调给商户，商户需要进行解密后才能获取到业务信息。

解密步骤如下：

1. 获取商户平台上设置的APIv3密钥，设置APIv3密钥可参考文档：APIv3密钥设置方法；
2. 通过回调通知参数resource.algorithm确认加密算法（目前仅支持AEAD_AES_256_GCM，算法的接口细节，请参考：rfc5116）。
3. 使用APIv3密钥与回调通知参数resource.nonce和resource.associated_data，对数据密文resource.ciphertext进行解密，最终可得到JSON格式的业务信息。

解密示例代码可参考文档：如何解密回调报文

注意

一些操作应该采用事务的操作，比如对两个表的修改

在小程序与商户进行交互的时候，实现对数据的加密（目前是明文传输）

在对回调通知内容上不够细致，没有对其进行解密（缺少 APIv3）