

移动开发平台

腾讯计费（米大师）

产品文档



腾讯云

【版权声明】

©2013-2019 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

腾讯计费（米大师）

产品介绍

功能介绍

控制台使用手册

使用说明

Android

Android 快速入门

Android 编程手册

集成 QQ 支付

iOS

iOS 快速入门

iOS 编程手册

Web 和小程序

Web 和小程序编程手册

开发者手册

支付渠道申请指引

发货服务器配置

服务器端 API

签名计算

常见问题

腾讯计费（米大师） 产品介绍

最近更新时间：2018-11-21 14:28:41

腾讯计费（下文中也叫米大师）孵化于支撑腾讯内部业务千亿级营收的互联网计费平台，具备十余年计费经验，现面向各行业伙伴全面开放。米大师提供专业的一站式计费解决方案，汇集国内外主流支付渠道，提供账户管理、精准营销、安全风险、稽核分账、计费分析等多维度服务，覆盖多类型、多场景、多终端，支持多级商户管理与分润，全方位支持各行业交易场景，打造计费生态服务圈。

优势

优势	简介
多平台支持	支持 Android/iOS 两大主流智能手机平台，同时支持微信支付和手 Q 支付，免去接入不同支付方式烦恼。
数据展示	多维数据展示，潜在付费分析，轻松拥有。
简化运维	日志流水详情，计费信息监控，简化日常运维工作。
风险控制	500+ 风控策略模型，提高业务实收防范羊毛党恶意刷单，第三方低价代充。

功能

方便的交易查询

通过指定支付时间、支付状态以及支付渠道，让您快速查询到相应的交易订单。

清晰的账单报表

清晰的账单报表让您迅速了解到应用的收入情况。

丰富的数据分析

数据分析功能帮您对应用的支付笔数、支付人数以及人均支付支付金额。

客户案例

王者荣耀、火影忍者。

功能介绍

最近更新时间：2018-11-14 15:22:24

Payment 概述

简介

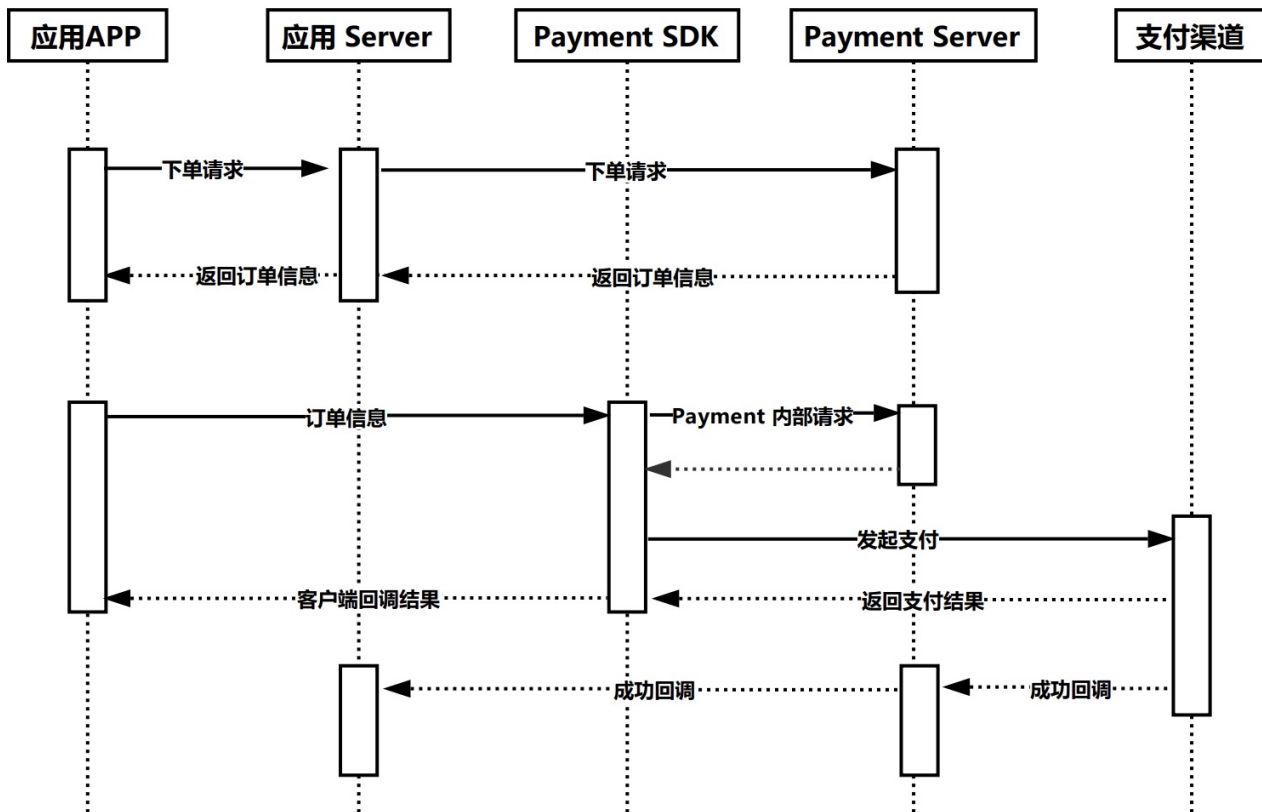
Payment 是 MobileLine 的移动计费服务，给用户提供微信支付与手 Q 支付两种支付渠道，支持 Android 和 iOS 两大主流智能手机平台，开发者可以在移动应用中调起微信和手 Q 来完成支付。

注意：

这里的微信支付和手 Q 支付都是指的其 App 支付模式。

交易流程

应用 App 在支付时，首先需要通过应用 Server 调用 Payment Server 统一下单接口，然后用获取的订单信息来初始化 Payment 支付请求并唤起支付，后续流程均由 Payment SDK 完成，支付完成后会返回支付的结果信息。最终支付成功后，应用后台会收到 Payment 后台服务的通知。整个支付过程如下：



Payment Server 回调地址即为控制台上配置的回调地址，应用 Server 需要根据实际情况来按格式进行返回。

名词解释

名词	解释
应用 App	您自己的 App，用于集成我们的 MobileLine Payment 服务。
应用 Server	您自己的 Server，需要在该 Server 上配置下单、通知回调等服务。
Payment SDK	即 MobileLine Payment SDK，您可以通过 Gradle，或者本地方式来集成。
Payment Server	即 MobileLine Payment 服务端。
支付渠道	这里指的是微信支付和 QQ 支付，需要您自己去申请和配置相关渠道信息。

功能介绍

渠道聚合

渠道聚合功能是指您在接入 Payment 服务后，可以直接给商品指定支付金额，然后调用【商品下单】接口进行下单，下单成功后，即可通过我们的 SDK 拉起支付界面进行支付。注意，在这种支付方式下，下单地址中 \$appid\$ 参数对应的是 MobileLine 应用移动支付模块的 offerid，这也意味着所有的支付数据也均留存在该 offerid 下，您可以在控制台的【交易查询】和【数据分析】中查看支付信息。

即下单地址为 `https://api.openmidas.com/v1/r/$offerid$/unified_order`

这种支付模式适用于没有虚拟货币，直接由业务方发放虚拟道具和服务的场景。

接入流程

下面将按步骤详细介绍如何接入我们的 Payment 服务，主要分为如下几步：

1. 创建 MobileLine 应用

如果您还没有自己的 MobileLine 应用，那么您首先需要在控制台上 [创建项目](#)，创建好项目后，请在该项目下 [创建 MobileLine 应用](#)，创建应用时，[请按照控制台指引配置您自己的应用 App](#)。

2. 配置应用 Server

应用 Server 指的是您自己业务的服务端，使用 Payment 服务时，您必须首先 [部署发货服务器](#)，然后根据您自己的业务情况配置 [服务端 API](#)。

3. 配置支付渠道

支付渠道包括微信支付和 QQ 支付两个渠道，您需要自己在微信开放平台或者腾讯开放平台上申请渠道信息，然后在控制台上进行配置，支付渠道配置请参见 [Payment 支付渠道配置指引](#)。

4. 接入 Payment SDK

配置好支付渠道后，您需要将 Payment SDK 集成到应用中，具体步骤请参见 [Payment Android 集成指南](#) 和 [Payment IOS 集成指南](#)。

5. 使用 Payment SDK 发起支付

至此，您已经完成了整个 Payment 服务的配置，现在您可以使用 Payment SDK 来发起支付了，详情请参见 [Payment Android 编程手册](#) 和 [Payment IOS 编程手册](#)。

6. 沙箱联调和发布上线

Payment 支持沙箱和线上两种环境，在您未发布到线上环境前，应用默认的是沙箱环境，这两种环境分布在不同的 Payment 服务器上，请根据您自己应用的实际情况，合理的使用相应环境。

沙箱环境

Payment 的默认环境，主要用于调试，在这个环境下，支付成功后，Payment Server 会回调控制台上设置的沙箱回调地址。

线上环境

当您的应用调试成功后，请您将应用发布上线，上线后，Payment Server 会在您支付成功后回调控制台上设置的主回调地址或者备回调地址，如何发布上线请参考 Payment 控制台使用手册的 [配置参数](#) 部分。

控制台使用手册

使用说明

最近更新时间：2019-01-11 21:49:18

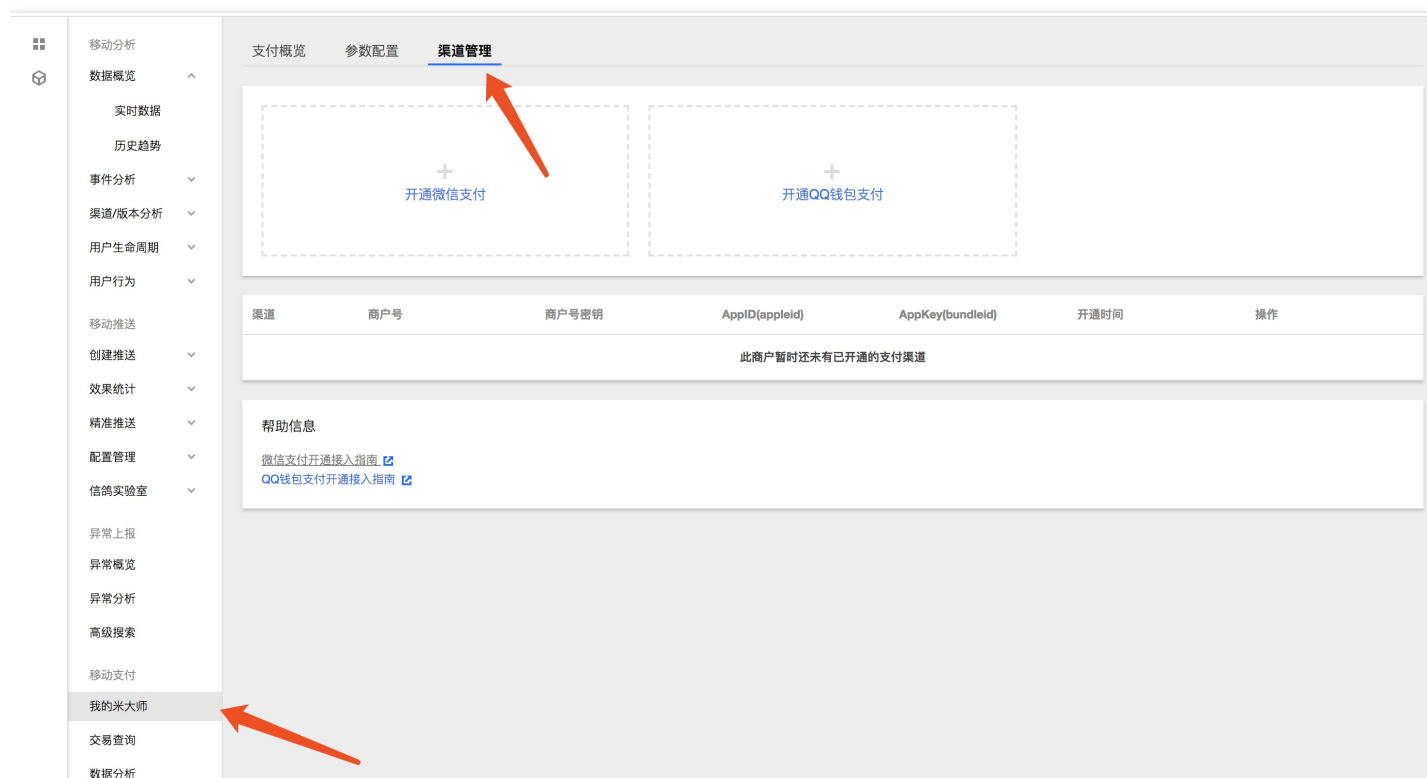
配置

在使用 Payment 支付前，您必须先开通支付渠道，然后配置好相关参数，才能成功发起支付请求。

开通支付渠道

1. 进入渠道管理页

登录 [控制台](#)，单击【项目】，选择【我的米大师】>【渠道管理】，进入渠道管理界面。



2. 填写渠道信息

进入渠道管理页后，如果您创建的是 Android 应用，那么会看到【开通微信支付】和【开通 QQ 钱包支付】两个按钮（如下图所示），如果您创建的是 IOS 应用，那么您也会看到【开通 IAP 支付】按钮。



接下来，您需要点击【开通微信支付】或者【开通 QQ 钱包支付】，填写渠道配置信息：

开通微信支付 ×

商户号

商户号密钥

AppID(应用ID)

AppKey(应用appkey)

填写好渠道信息后，点击【确定】。

相关信息说明

商户号

商户号需要在根据您开通的支付渠道在 [微信商户平台](#) 或者 [QQ 钱包商户平台](#) 上申请，申请的方式请参见 [这里](#)。

商户号密钥

申请商户号成功后，商户平台会给您生成一个商户号密钥，这是您的机密信息，请不要泄露。

AppID

在开通支付前，您需要根据开通的渠道在 [微信开放平台](#) 或者 [腾讯开放平台](#) 上注册，并创建一个应用，创建好应用后请将其 AppId 拷贝过来，同样可以参见 [这里](#)。

AppKey

请您将创建好的应用 appKey 一并拷贝过来，这也是您的机密信息，请不要泄露。

配置参数

1. 进入参数配置页

配置好渠道信息并点击【确认】后，会提示您进入参数配置页，单击【我的米大师】>【配置参数】>【修改】来进入参数配置页。

2. 填写配置信息

配置项如下：

← 支付配置

1 参数配置 > 2 发布预览

渠道配置 本配置对应用户看到的支付渠道顺序，折扣比例及展示状态。

配置详情

顺序	支付渠道	折扣比例	展示状态	顺序调整
1	微信支付	<input type="text" value="100"/> %	<input checked="" type="radio"/> 展示 <input type="radio"/> 隐藏	↑ ↓

回调地址

主回调地址

备回调地址

沙箱回调地址

平台参数配置

应用签名

应用包名

填好配置信息后，点击【保存 & 预览】进入下一步。

3. 发布上线

如果您所有的参数验证完成，并且准备发布上线，您可以点击【发布上线】按钮发布到线上。

移动分析

数据概览 ^

 实时数据

 历史趋势

事件分析 v

渠道/版本分析 v

用户生命周期 v

用户行为 v

移动推送

创建推送 v

效果统计 v

精准推送 v

配置管理 v

信鸽实验室 v

异常上报

异常概览

异常分析

高级搜索

移动支付

我的米大师

交易查询

数据分析

← 支付配置

1 参数配置 > 2 发布预览

支付OfferID: [redacted]

沙箱OfferKey: [redacted]

渠道配置 本配置对应用户看到的支付渠道顺序、折扣比例及展示状态。

配置详情

顺序	支付渠道	折扣比例	展示状态
1	微信支付	100 %	展示

回调地址

主回调地址

备回调地址

沙箱回调地址

平台参数

应用签名

应用包名

我已完成上述验证，确认上线后所需功能与沙箱环境所示一致

相关信息说明

主回调地址

发布上线后，每次支付成功后，Payment 后台会回调这个地址，通知您的服务器已经支付成功。

备回调地址

备用的线上回调地址。

沙箱回调地址

在您未发布上线前，Payment 后台的回调地址。

应用签名

如果您创建的是 Android 应用，那么需要填入您应用的签名值，这里必须和开放平台上注册应用时的签名值保持一致。您可以用如下示例命令生成应用签名，这里假设您的签名文件 alias 为 androiddebugkey，签名文件的示例路径：`~/android/debug.keystore`。

签名文件路径：

```
keytool -exportcert -list -v -alias androiddebugkey -keystore ~/android/debug.keystore
```

命令生成的 SHA1 值即为您的应用签名。

应用包名

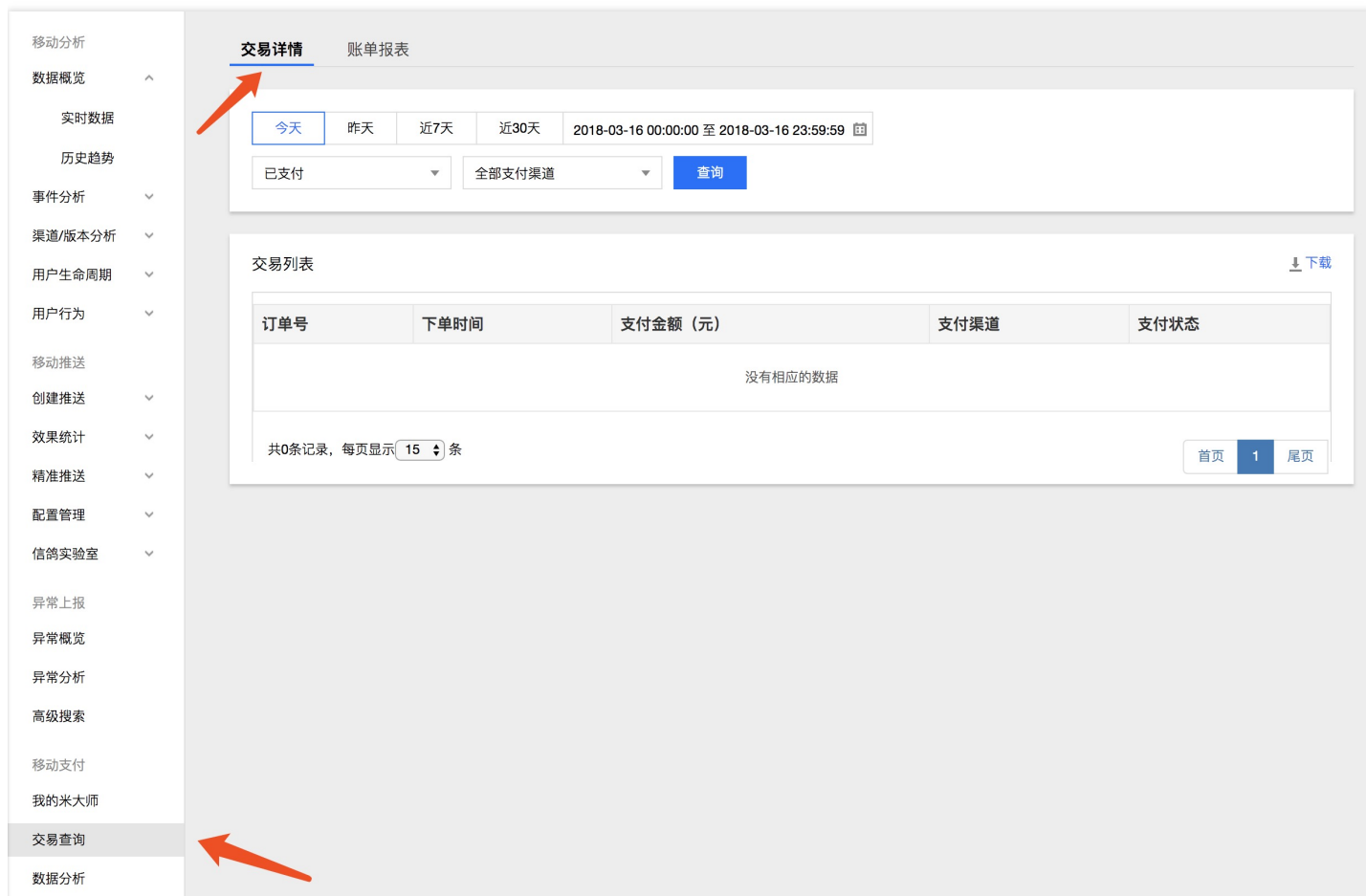
如果您创建的是 Android 应用，那么需要填入您的包名，这里也必须和开放平台上的包名保持一致。

数据查询与分析

交易查询

交易详情

点击右侧导航栏中的【交易查询】，然后选择上方导航栏中的【交易详情】，您可以通过支付时间段、支付状态、支付渠道来查询对应的交易信息。



移动分析

数据概览

实时数据

历史趋势

事件分析

渠道/版本分析

用户生命周期

用户行为

移动推送

创建推送

效果统计

精准推送

配置管理

信鸽实验室

异常上报

异常概览

异常分析

高级搜索

移动支付

我的米大师

交易查询

数据分析

交易详情 账单报表

今天 昨天 近7天 近30天 2018-03-16 00:00:00 至 2018-03-16 23:59:59

已支付 全部支付渠道 查询

交易列表 [下载](#)

订单号	下单时间	支付金额 (元)	支付渠道	支付状态
没有相应的数据				

共0条记录, 每页显示 15 条

首页 1 尾页

账单报表

点击右侧导航栏中的【交易查询】，然后选择上方导航栏中的【账单报表】，您可以通过支付时间段、支付状态、支付渠道来查询对应的账单报表。

- 移动分析
- 数据概览 ^
- 实时数据
- 历史趋势
- 事件分析 v
- 渠道/版本分析 v
- 用户生命周期 v
- 用户行为 v
- 移动推送
- 创建推送 v
- 效果统计 v
- 精准推送 v
- 配置管理 v
- 信鸽实验室 v
- 异常上报
- 异常概览
- 异常分析
- 高级搜索
- 移动支付
- 我的米大师
- 交易查询
- 数据分析

交易详情
账单报表

昨天
近7天
近30天
2018-03-16 00:00:00 至 2018-03-16 23:59:59

▾
全部支付渠道

查询

账单报表

支付渠道	收入 (元)	退款 (元)	账单明细
没有相应的数据			

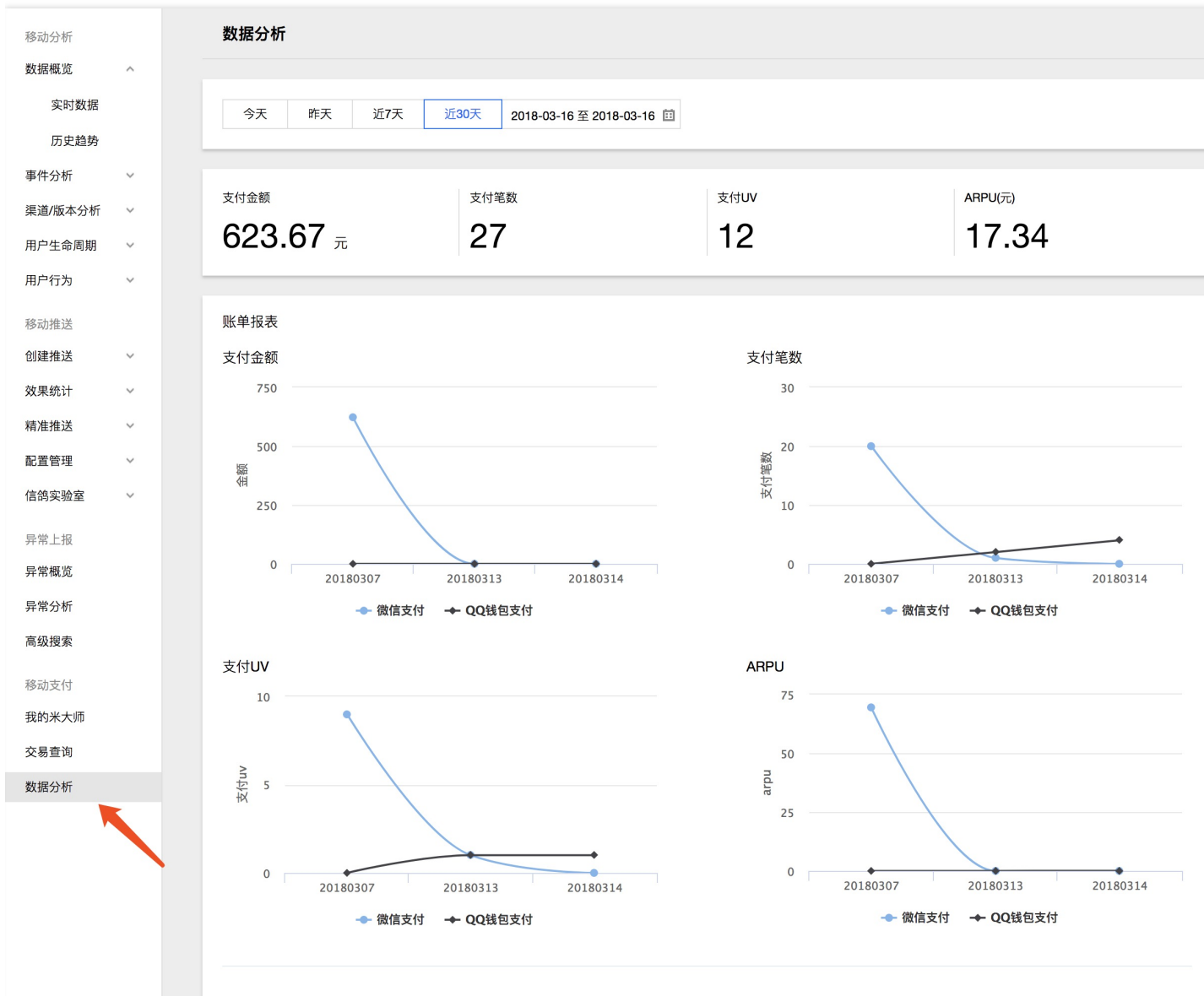
共0条记录, 每页显示 15 条

首页
1
尾页

数据分析

查看分析数据

点击右侧导航栏中的【数据分析】，您可以查看所有用户支付数据的分析情况。



相关信息说明

名称	意义
支付金额	所有用户支付的总金额数
支付笔数	所有用户支付的笔数
支付UV	支付的用户总数
ARPU	平均每个用户支付的金额数

Android

Android 快速入门

最近更新时间：2018-08-16 16:30:10

准备工作

- 您首先需要有一个 Android 工程，这个工程可以是您现有的工程，也可以是您新建的一个空的工程。
- 其次您需要 [配置后台服务器](#)。
- 最后您需要申请到相关支付渠道（[如何自行申请渠道](#)）。

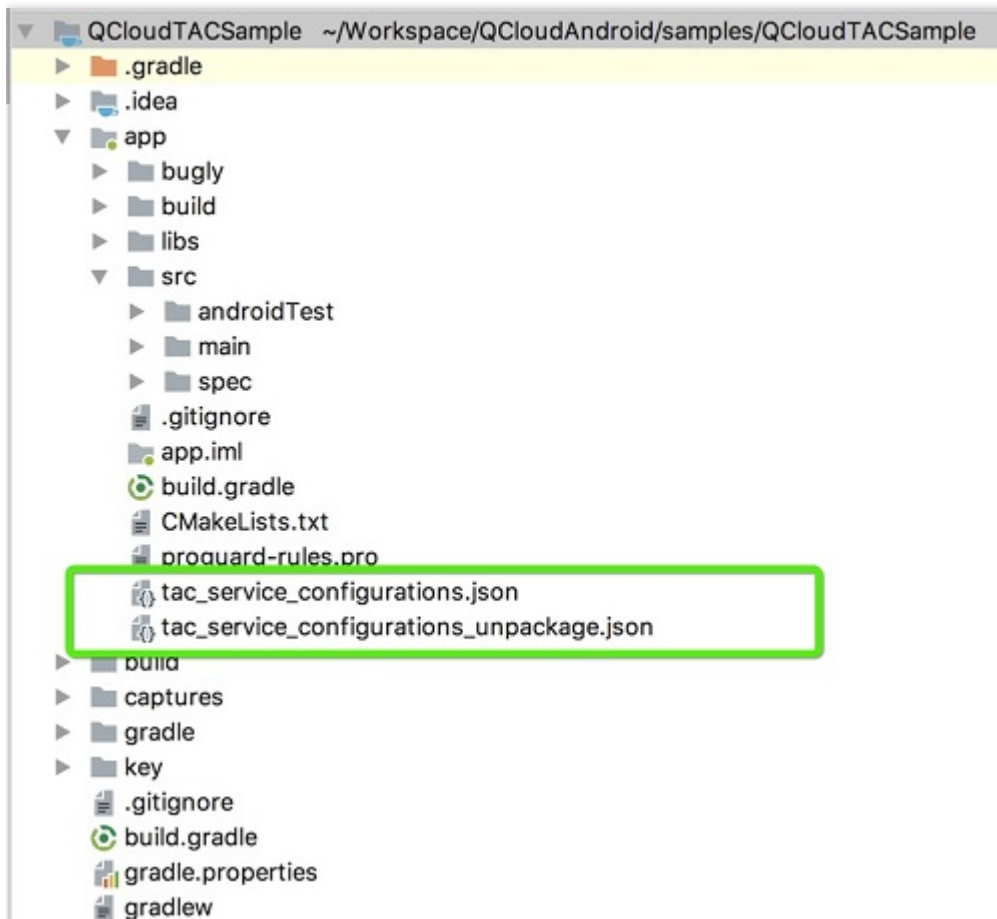
第一步：创建项目和应用（已完成请跳过）

在使用我们的服务前，您必须先先在 MobileLine 控制台上 [创建项目和应用](#)。

第二步：添加配置文件（已完成请跳过）

在您创建好的应用上单击【下载配置】按钮来下载该应用的配置文件的压缩包：

解压该压缩包，您会得到 `tac_service_configurations.json` 和 `tac_service_configurations_unpackage.json` 两个文件，请您如图所示添加到您自己的工程中去。



注意：

请您按照图示来添加配置文件，`tac_service_configurations_unpackage.json` 文件中包含了敏感信息，请不要打包到 APK 文件中，MobileLine SDK 也会对此进行检查，防止由于您误打包造成的敏感信息泄露。

第三步：集成 SDK

您需要在工程级 `build.gradle` 文件中添加 SDK 插件的依赖：

```
buildscript {
...
dependencies {
classpath 'com.android.tools.build:gradle:3.0.1'
// 添加这行
classpath 'com.tencent.tac:tac-services-plugin:1.3.+
}
}
```

在您应用级 build.gradle 文件（通常是 app/build.gradle）中添加 payment 服务依赖，并使用插件：

```
dependencies {  
    // 增加这两行  
    compile 'com.tencent.tac:tac-core:1.3.'  
    compile 'com.tencent.tac:tac-payment:1.3.'  
}  
...  
  
// 在文件最后使用插件  
apply plugin: 'com.tencent.tac.services'
```

到此您已经成功接入了 MobileLine 移动付费服务。

Proguard 配置

如果您的代码开启了混淆，为了 SDK 可以正常工作，请在 `proguard-rules.pro` 文件中添加如下配置：

```
# MobileLine Core  
  
-keep class com.tencent.qcloud.core.** { *;}  
-keep class bolts.** { *;}  
-keep class com.tencent.tac.** { *;}  
-keep class com.tencent.stat.*{*;}  
-keep class com.tencent.mid.*{*;}  
-dontwarn okhttp3.**  
-dontwarn okio.**  
-dontwarn javax.annotation.**  
-dontwarn org.conscrypt.**  
  
# MobileLine Payment  
  
-keep class com.tencent.midas.** { *;}  
-keep class com.tencent.openmidas.** { *;}
```

Android 编程手册

最近更新时间：2019-04-17 16:06:32

商品下单

在您发起支付前，需要先调用米大师的 [下单接口](#) 进行下单，下单成功后，米大师会给您返回本次下单的具体信息，下单服务器返回的数据示例如下：

```
{
  "ret" : 0,
  "transaction_id" : "E-180202180100144947",
  "out_trade_no" : "open_1517568769743",
  "pay_info" : "data={\"appid\":\"TC100008\",\"user_id\":\"rickenwang\",\"out_trade_no\":\"open_1517568769743\",\"product_id\":\"product_test\",\"pay_method\":\"wechat\"}&sign=PplSFOrimAfU1dobsFwa09limmtk+lr9D5dxFwwV+Edjq9dROhB6fwx9hwhf1H27FMT83qQdlSgHtLo52Rv97MoL7nR5xNJFph9G7Gd2KRmgJFQ2llGfHVE+eekjPhRQCElt5MMbDuSEOOGJN4agMiCs9yOXJbusCYAa68bcZTOnGgfDOsbpNvpsQt9JA+Q/AVDyymXv0f6e+ibpXITy3Fu3lQZKzPUiioj197Kpi4l0J6CGCWsxRp4XqWSF7k90o1NMOcbUnzJ87MSCXq5NA1iynYxrD5Cc5KusJxpy84udTtD9XzdzXpO+QJBoO2v0RzGGgT2OJQfgRLqsNNgzw==\""}
}
```

我们强烈建议您在自己的后台服务器上下单，然后由服务器给终端返回下单信息，直接在终端进行下单操作可能会泄漏您的密钥信息。

初始化支付请求

每次发起支付前，您必须先初始化一个 `PaymentRequest` 支付请求对象。

支付请求 `PaymentRequest`

参数名称	参数描述	类型	备注
userId	发起支付的 userId	String	最好和下单时的 user_id 保持一致
payInfo	支付信息	String	下单接口返回的 pay_info 字段

每次初始化支付请求都必须先进行下单获取支付信息，同一个支付信息不可以多次使用，初始化示例代码如下：

```
// 1、向您自己的服务器请求下单信息，也即下单服务器返回的 pay_info 字段。
// 请您自己根据实际情况自己设计和实现 getPayInfoFromServer() 方法
String payInfo = getPayInfoFromServer();
```

```
// 2、初始化 PaymentRequest 实例
String userId = "tencent";
PaymentRequest paymentRequest = new PaymentRequest(userId, payInfo);
```

发起支付请求

初始化好支付请求 `paymentRequest` 后，您可以通过 `TACPaymentService` 实例来发起支付请求了，具体示例代码如下：

```
// 1、获取 TACPaymentService 实例
TACPaymentService paymentService = TACPaymentService.getInstance();

// 2、通过 TACPaymentService 实例发起支付
paymentService.launchPayment(context, paymentRequest, new TACPaymentCallback() {
    @Override
    public void onResult(int resultCode, PaymentResult paymentResult) {

        // 支付码 resultCode
        // 支付结果 paymentResult
    }
});
```

返回支付结果

在您发起支付后，SDK 会最终给您返回流程错误码 `resultCode` 和支付结果 `PaymentResult`。

流程错误码 `resultCode`

错误码名称	值	描述
PAYRESULT_SUCC	0	支付流程成功
PAYRESULT_ERROR	-1	支付流程失败
PAYRESULT_CANCEL	-2	用户取消
PAYRESULT_PARAMERROR	-3	参数错误
PAYRESULT_UNKNOWN	-4	支付流程结果未知

支付结果 PaymentResult

参数名称	参数描述	类型
code	内部错误码	String
message	错误信息	String
metaData	自定义拓展信息	Map

添加自定义信息

在发起支付时，您可以添加自定义信息。

```
String key = "name";  
String value = "xiaoming";  
paymentRequest.addMetaData(key, value);
```

```
// other code  
...
```

该次支付结束后，回调中 PaymentResult 对象会携带您添加的自定义信息。

```
public void onResult(int resultCode, PaymentResult result) {  
  
    Map<String, String> metaData = result.getMetaData();  
}
```

集成 QQ 支付

最近更新时间：2018-07-10 13:19:07

无论您是采用 gradle 远程依赖还是手动集成的方式来接入 Payment 服务，如果您需要接入 QQ 支付，那么您都必须手动添加 `mqqopenpay.jar` 到您工程的 `libs` 目录。

注意：

因为 `mqqopenpay.jar` 没有上传到 jcenter 仓库下，因此在 gradle 远程依赖的方式下我们暂时无法自动帮您添加。

iOS

iOS 快速入门

最近更新时间：2019-03-04 16:25:59

MobileLine iOS 移动支付快速入门

移动开发平台 (MobileLine) 使用起来非常容易，只需要简单的 4 步，您便可快速接入移动崩溃监测。接入后，您即可获得我们提供的各项能力，减少您在开发应用时的重复工作，提升开发效率。

准备工作

为了使用移动开发平台 (MobileLine) iOS 版本的 SDK，您首先需要有一个 iOS 工程，这个工程可以是您现有的工程，也可以是您新建的一个空的工程。

第一步：创建项目和应用

在使用我们的服务前，您必须先要在 MobileLine 控制台上 [创建项目和应用](#)。

如果您已经在 MobileLine 控制台上创建过了项目和应用，请跳过此步。

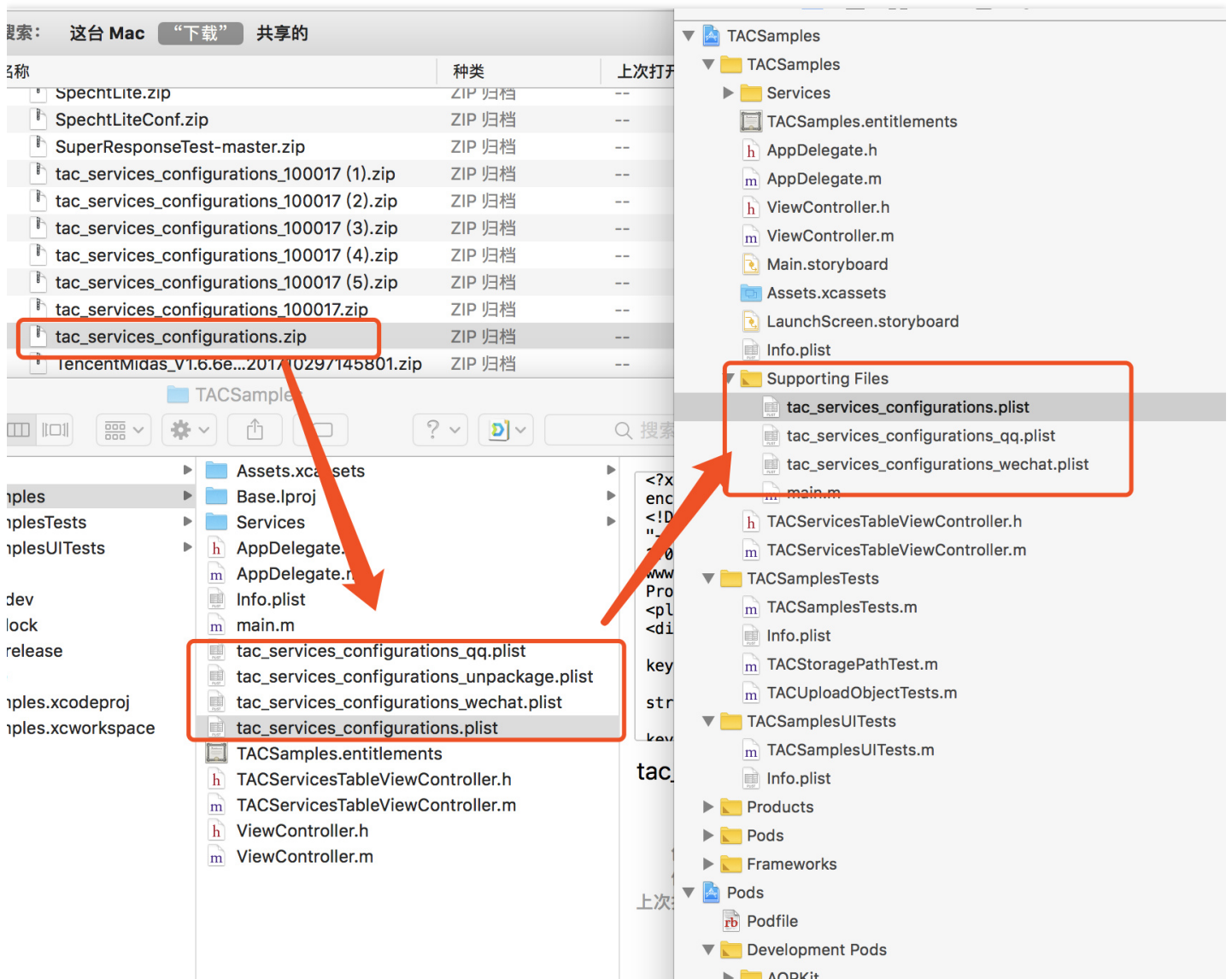
第二步：添加配置文件

如果您已经添加过配置文件，请跳过此步。

创建好应用后，您可以点击红框中的【下载配置】来下载该应用的配置文件的压缩包：



解压后将 tac_services_configurations.plist 文件集成进项目中。其中有一个 tac_services_configurations_unpackage.plist 文件，请将该文件放到您工程的根目录下(切记不要将改文件添加进工程中)。添加好配置文件后，继续点击【下一步】。



注意：

请您按照图示来添加配置文件，`tac_service_configurations_unpackage.plist` 文件中包含了敏感信息，请不要打包到 apk 文件中，MobileLine SDK 也会对此进行检查，防止由于您误打包造成的敏感信息泄露。

第三步：集成 SDK

如果还没有 Podfile，请创建一个。

```
$ cd your-project directory
$ pod init
```

并在您的 Podfile 文件中添加移动开发平台（MobileLine）的私有源：

```
source "https://git.cloud.tencent.com/qcloud_u/cocopoads-repo"
source "https://github.com/CocoaPods/Specs"
```

在 Podfile 中添加依赖：

```
pod 'TACPayment'
```

目前我们支持两个渠道的支付能力：

模块名	渠道名称
TACPaymentPluginQQ	QQ支付
TACPaymentPluginWechat	微信支付

如果您需要两个渠道的能力则需要添加依赖

```
pod 'TACPaymentPluginQQ'
pod 'TACPaymentPluginWechat'
```

如果您使用了QQ支付，则您将引入 TACSocialQQ 模块，请参考该模块的配置，并添加配置文件 `tac_services_configurations_qq.plist`，并将文件添加进XCode工程中。文件内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
```

```
<dict>
<key>services</key>
<dict>
<key>social</key>
<dict>
<key>qq</key>
<dict>
<key>appld</key>
<string>请填写您的 Appld</string>
<key>appKey</key>
<string>请填写您的 APPKey</string>
<key>permissions</key>
<array>
<string>get_user_info</string>
<string>get_simple_userinfo</string>
<string>add_t</string>
</array>
</dict>
</dict>
</dict>
</dict>
</plist>
```

如果您使用微信支付，则您将引入 TACSocialWechat模块，请参考该模块的配置，并添加配置文件 `tac_services_configurations_wechat.plist`，并将文件添加进XCode工程中。文件内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>services</key>
<dict>
<key>social</key>
<dict>
<key>wechat</key>
<dict>
<key>appld</key>
<string>wx256642f480c15e3e</string>
</dict>
</dict>
</dict>
</dict>
</plist>
```

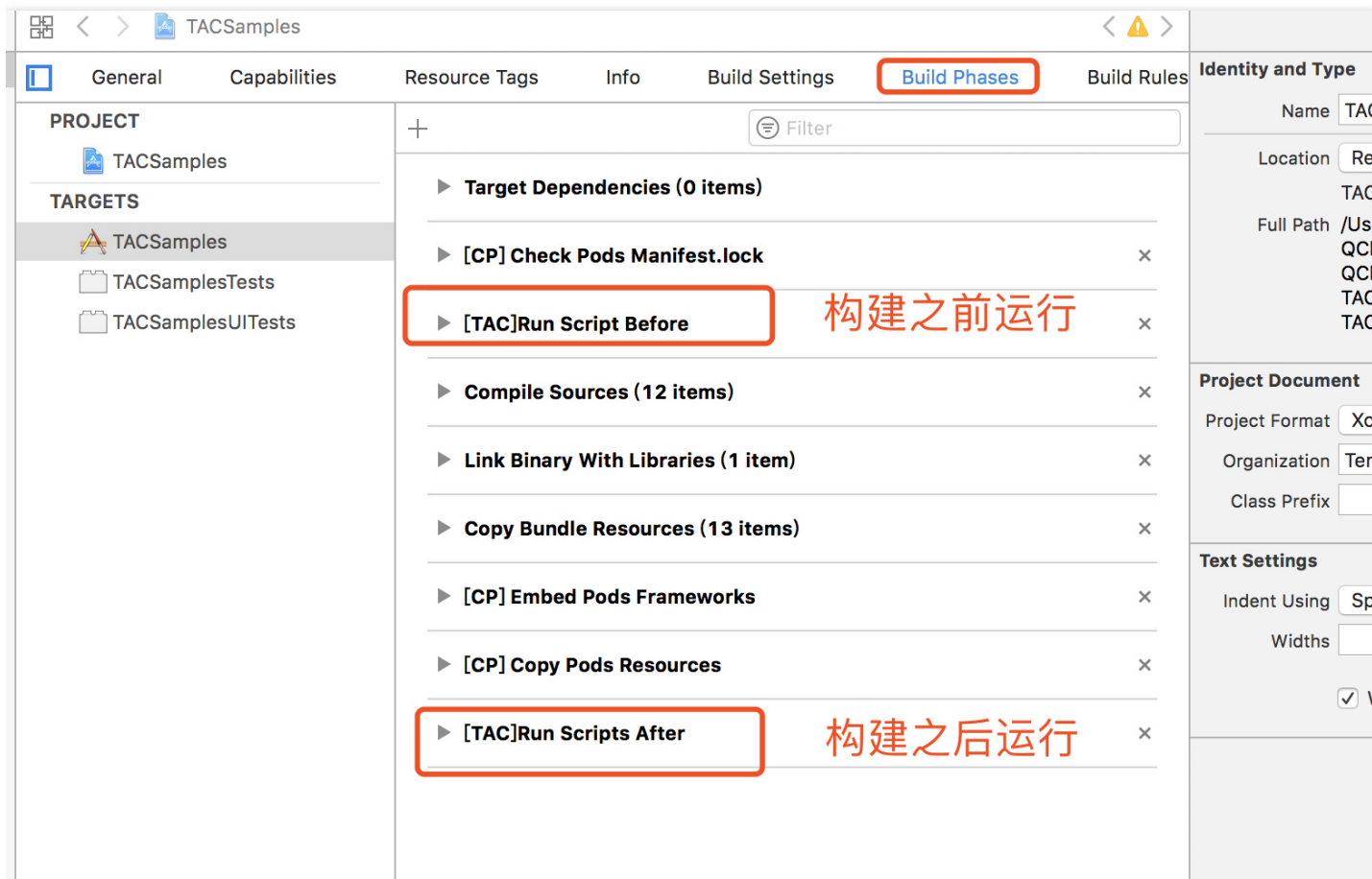
配置程序需要脚本

如果您在其他模块中完成了此步骤，请不要重复执行。

为了简化 SDK 的接入流程，我们使用 shell 脚本，帮助您自动化的去执行一些繁琐的操作，比如 crash 自动上报，在 Info.plist 里面注册各种第三方 SDK 的回调 scheme。因而，需要您添加以下脚本来使用我们自动化的加入流程。

脚本主要包括两个：

1. 在构建之前运行的脚本，该类型的脚本会修改一些程序的配置信息，比如在 Info.plist 里面增加 qqwallet 的 scheme 回调。
2. 在构建之后运行的脚本，该类型的脚本在执行结束后做一些动作，比如 Crash 符号表上报。



请按照以下步骤来添加脚本：

添加构建之前运行的脚本

1. 在导航栏中打开您的工程。

2. 打开 Tab Build Phases。
3. 点击 Add a new build phase，并选择 New Run Script Phase，您可以将改脚本命名 TAC Run Before

注意：

请确保该脚本在 Build Phases 中排序为第二。

4. 根据自己集成的模块和集成方式将代码粘贴入 Type a script... 文本框。

需要黏贴的代码

```
#export TAC_SCRIPTS_BASE_PATH=[自定义执行脚本查找路径，我们会在该路径下寻找所有以“tac.run.all.before.sh”命名的脚本，并执行，如果您不需要自定义不用动这里]
${TAC_CORE_FRAMEWORK_PATH}/Scripts/tac.run.all.before.sh
```

其中 THIRD_FRAMEWORK_PATH 变量的取值根据您的安装方式而不同：

- 如果您使用 Cocoapods 来集成的则为 \${PODS_ROOT}/TACCore，您需要黏贴的代码实例如下：

```
${SRCROOT}/Pods/TACCore/Scripts/tac.run.all.before.sh
```

- 如果您使用手工集成的方式则为 您存储 TACCore 库的地址，即您 TACCore framework 的引入路径，您需要黏贴的代码实例如下：

```
export TAC_SCRIPTS_BASE_PATH=[自定义执行脚本查找路径，我们会在该路径下寻找所有以“tac.run.all.after.sh”命名的脚本，并执行，如果您不需要自定义不用动这里]
[您存储 TACCore 库的地址]/TACCore.framework/Scripts/tac.run.all.before.sh
```

添加构建之后运行的脚本

1. 在导航栏中打开您的工程。
2. 打开 Tab Build Phases。
3. 点击 Add a new build phase，并选择 New Run Script Phase，您可以将改脚本命名 TAC Run Before。

注意：

请确保该脚本在 Build Phases 中排序需要放到最后。

4. 根据自己集成的模块和集成方式将代码粘贴入 Type a script... 文本框。

需要黏贴的代码

```
#export TAC_SCRIPTS_BASE_PATH=[自定义执行脚本查找路径,我们会在该路径下寻找所有以“tac.run.all.after.sh”命名的脚本,并执行,如果您不需要自定义不用动这里]
${TAC_CORE_FRAMEWORK_PATH}/Scripts/tac.run.all.after.sh
```

其中 `THIRD_FRAMEWORK_PATH` 变量的取值根据您的安装方式而不同：

- 如果您使用 Cocoapods 来集成的则为 `${PODS_ROOT}/TACCore`，您需要黏贴的代码实例如下：

```
${SRCROOT}/Pods/TACCore/Scripts/tac.run.all.after.sh
```

- 如果您使用手工集成的方式则为 `[您存储 TACCore 库的地址]`，即您 TACCore framework 的引入路径，您需要黏贴的代码实例如下：

```
#export TAC_SCRIPTS_BASE_PATH=[自定义执行脚本查找路径,我们会在该路径下寻找所有以“tac.run.all.after.sh”命名的脚本,并执行,如果您不需要自定义不用动这里]
[您存储 TACCore 库的地址]/TACCore.framework/Scripts/tac.run.all.after.sh
```

第四步：初始化

集成好我们提供的 SDK 后，您需要在您自己的工程中添加初始化代码，从而让 MobileLine 服务在您的应用中进行自动配置。整个初始化的过程很简单。

步骤 1 在 UIApplicationDelegate 子类中导入移动开发平台（MobileLine）模块。

Objective-C 代码示例：

```
#import <TACCore/TACCore.h>
```

Swift 代码示例：

```
import TACCore
```

步骤 2 配置一个 TACApplication 共享实例，通常是在应用的 `application:didFinishLaunchingWithOptions:` 方法中配置。

使用默认配置

通常对于移动开发平台 (MobileLine) 的项目他的配置信息都是通过读取 tac_services_configuration.plist 文件来获取的。

Objective-C 代码示例：

```
[TACApplication configure];
```

Swift 代码示例：

```
TACApplication.configure();
```

启动服务

Payment 服务无需启动，到此您已经成功接入了 MobileLine 移动付费服务。

后续步骤

了解 MobileLine：

- 查看 [MobileLine 应用示例](#)

向您的应用添加 MobileLine 功能：

- 借助 [Analytics](#) 深入分析用户行为。
- 借助 [messaging](#) 向用户发送通知。
- 借助 [crash](#) 确定应用崩溃的时间和原因。
- 借助 [storage](#) 存储和访问用户生成的内容（如照片或视频）。
- 借助 [authorization](#) 来进行用户身份验证。
- 借助 [payment](#) 获取微信和手 Q 支付能力

iOS 编程手册

最近更新时间：2019-04-17 16:26:27

支付流程

整个支付的流程分为：商品下单>客户端>调起支付>获回支付结果

下单

在您发起支付前，需要先在后台调用我们的 [下单接口](#) 进行下单，下单成功后，服务器会给您返回本次下单的具体信息，下单服务器返回的数据示例如下：

```
{
  "ret": 0,
  "transaction_id": "E-180202180100144947",
  "out_trade_no": "open_1517568769743",
  "pay_info": "data={\"appid\":\"TC100008\",\"user_id\":\"rickenwang\",\"out_trade_no\":\"open_1517568769743\",\"product_id\":\"product_test\",\"pay_method\":\"wechat\"}&sign=PplSFOrimAfU1dobsFvva09limmtk+lr9D5dxFwwV+Edjq9dROhB6fwx9hwhf1H27FMT83qQdlSgHtLo52Rv97MoL7nR5xNJFph9G7Gd2KRmgJFQ2llGfHVE+eekjPhRQCElt5MMbDuSEOOGJN4agMiCs9yOXJbusCYAa68bcZTOnGgfDOsbpNvpsQt9JA+Q/AVDyymXv0f6e+ibpXITy3Fu3lQZKzPUiioj197Kpi4lOJ6CGCWsxRp4XqWSF7k90o1NMOcbUnzJ87MSCXq5NA1iynYxrD5Cc5KusJxpy84udTtD9XzdzNxpO+QJBoO2v0RzGGgT2OJQfgRLqsNNgzw==\""}
}
```

我们强烈建议您在自己的后台服务器上下单，然后由服务器给终端返回下单信息，直接在终端进行下单操作可能会泄漏您的密钥信息。

调起支付

对商品完成下单后，会得到一个 `pay_info`，对应前面 JSON 中 `pay_info` 字段的值，`pay_info` 实质上就是一个字符串，我们可以拿着这个 `pay_info` 调用 SDK 的接口去直接拉起相应的支付：

Objective-C 代码示例：

```
[[TACPaymentService defaultService] pay:payInfo appMeataData:nil completion:^(TACPaymentResult * result) {
    TACLogDebug(@"支付结果 %d %@", result.resultCode, result.resultMsg);
}];
```


Swift 代码示例：

```
TACPaymentService.default().pay(payInfo, appMeataData: nil) { (result:TACPaymentResult?)->Void in
    print("支付结果 ",result?.resultCode as Any,result?.resultMsg as Any)
}
```

返回支付结果

支付完成以后，结果会包装在 TACPaymentResult 对象中，在完成回调 block 里以参数传入的形式返回。它包含了如下信息：

属性	类型	描述
resultCode	TACPaymentErrorCode	支付结果的返回码
payMethod	NSString*	本次支付的方式
innerCode	NSString*	系统内部 Code, 如果出错，则是系统的内部错误码
resultMsg	NSString*	结果信息
appMetadata	NSString*	额外的一些元数据

resultCode 是本次支付返回的错误码，具体有：

返回码名称	值	描述
TACPaymentCodeSuccess	0	支付流程成功
TACPaymentCodeError	-1	支付流程失败
TACPaymentCodeCancel	-2	用户取消
TACPaymentCodeParamterError	-3	参数错误
TACPaymentCodeUnkown	-4	支付流程结果未知

常见问题

- 无法拉起微信支付，请按照以下步骤进行排查：

-
- 微信商户号是否已经成功开通。
 - 微信开放平台上，是否已经成功开通支付能力。
 - [控制台](#) 中检查，商户号、商户号密钥是否与微信商户号的一致。AppID，AppKey 是否与微信开放平台上的一致。
 - 检查下单是否成功，pay_info 是否是有效的字符串而不是空。
 - 检查控制台日志中是否有初始化错误，或者错误码等信息。
-
- 支付完成后无法跳转回应用内
 - 检查 URL Scheme 是否已经设置为微信开放平台上的 APPID（以WX开头）。
 - 若是 QQ 支付，检查 URL Scheme 是否已经设置为“qqwallet”+ id 的形式，例如 qqwallet1234567。

Web 和小程序

Web 和小程序编程手册

最近更新時間：2018-07-24 14:58:07

集成說明

引入 JsApi：`<script src="https://midas.gtimg.cn/openmidas/jsapi/openMidas.js"></script>`

為了支持多域名情況，除了 `openMidas.js`，還有一個額外的 `openMidasConfig.js` 路徑為

`https://midas.gtimg.cn/openmidas/jsapi/config/openMidasConfig.xxx.js` 用於域名配置，在引入之前需要引入這個配置文件，默認情況下 `openMidas` 會走公有雲的環境，可以不引入配置。

支付初始化

說明：初始化接口，使用 `OpenMidas` 其它接口之前必須調用本接口。

接口：`OpenMidas.init(env)`

參數說明如下：

參數名	參數類型	必填	參數說明
env	String	是	環境，release 表示正式環境，test 表示測試環境

支付接口

Web 支付接口

說明：本接口同時支持 H5 支付、公眾號支付。

接口 1：彈框形式調用

```
OpenMidas.pay(payInfo, callback, appMetadata);
```

參數說明如下：

參數名	參數類型	必填	參數說明
payInfo	String	是	支付參數，詳見 服務器端 API 商品下单接口返回值里的 pay_info

参数名	参数类型	必填	参数说明
callBack	Function	是	支付完成回调函数，回调参数说明看下文“回调 url 示例”
appMetadata	String	否	扩展字段，key=value 形式，最大长度 255。客户端回调时回传给调用方。

调用方式示例：

```
someAsyncRequest(function(payInfo){
    OpenMidas.pay(payInfo, function(resultCode, innerCode, resultMsg, appMetadata){
        //业务处理代码
    }, appMetadata); //方式1调用
});
```

接口 2：页面跳转形式调用支付完成后会回调业务在下单时传入的 Web 回调地址。

```
OpenMidas.pay(payInfo, appMetadata);
```

参数说明如下：

参数名	参数类型	必填	参数说明
payInfo	String	是	支付参数，详见 服务器端 API 商品下单接口返回值里的 pay_info
appMetadata	String	否	扩展字段，key=value 形式，最大长度 255。客户端回调时回传给调用方。

调用方式示例：

```
someAsyncRequest(function(payInfo){
    OpenMidas.pay(payInfo, appMetadata); //方式2调用
});
```

回调 url 示例：

```
[callbackurl]?resultCode=0&innerCode=100-xxx&resultMsg=encode(支付成功)&appMetadata=xxxxx
```

小程序支付接口

说明：将 openMidas.js 文件放入工程目录，并通过如下方式引入

```
var OpenMidas = require("openMidas");
```

为了支持多域名情况，除了 openMidas.js 还有一个额外的 openMidasConfig.js 用于域名配置，在 require jsapi 之前需要引入这个配置文件，默认情况下 openMidas 会走公有云的环境。

说明：初始化接口，使用 openMidas 其它接口之前必须调用本接口。

接口： OpenMidas.init(env)

参数说明如下：

参数名	参数类型	必填	参数说明
env	String	是	环境，release 表示正式环境，test 表示测试环境

接口：

OpenMidas.pay(String payInfo, Function callback, String appMetadata)

参数说明如下：

参数名	参数类型	必填	参数说明
payInfo	String	是	支付参数，详见 服务器端 API 商品下单接口返回值里的 pay_info
callBack	Function	是	支付完成回调函数，回调参数说明详见“小程序支付接口”
appMetadata	String	否	扩展信息回传，透传支付时传入的参数。同支付时传入的 appMetadata

调用方式示例：

```
var OpenMidas=require("openMidas");
OpenMidas.init(env);
OpenMidas.pay(payInfo, function(resultCode, innerCode, resultMsg, appMetadata){
    //业务处理代码
}, appMetadata);
```

支付回调参数

属性	类型	取值
resultCode	Int	0(PAYRESULT_SUCC 支付流程成功),-1(PAYRESULT_ERROR 支付流程失败),-2(PAYRESULT_CANCEL 用户取消),-3(PAYRESULT_PARAMERROR 参数错误),-4(PAYRESULT_UNKNOWN 支付流程结果未知，如第三方渠道未回调米大师)

属性	类型	取值
innerCode	String	系统内部错误码，不直接展示给用户
resultMsg	String	返回信息，不直接展示给用户
appMetadata	String	扩展信息回传，透传支付时传入的参数。同支付时传入的 appMetadata

签约接口

Web 签约接口

引入 JsApi : `<script src="https://midas.gtimg.cn/midas/open/openMidas.js"></script>`

说明：初始化接口，使用 OpenMidas 其它接口之前必须调用本接口。

接口：OpenMidas.init(env)

参数说明如下：

参数名	参数类型	必填	参数说明
env	String	是	环境，release 表示正式环境，test 表示测试环境

说明：签约接口

接口：

OpenMidas.signContract(Object params,Function errorCallback)

参数说明如下：

参数名	参数类型	必填	参数说明
params.appld	String	是	米大师的应用 ID
params.userId	String	是	用户 ID
params.channel	String	是	签约渠道，可选值：wechat（使用微信签约）
params.redirectUrl	String	是	签约完成之后的回调 url，当用户从签约 url 返回时，会跳转到这个 url 上，url 参数会带上 appld、openId、channel、fromSign=1。

参数名	参数类型	必填	参数说明
errorCallback	Function	否	当内部参数校验不通过或者后台返回错误时，会执行回调，回调参数参见下表“错误回调参数说明”。

错误回调参数说明：

属性	类型	取值
resultCode	Int	-1(PAYRESULT_ERROR 签约流程失败流程失败),-3(PAYRESULT_PARAMERROR 参数错误)
innerCode	String	系统内部错误码，不直接展示给用户
resultMsg	String	返回信息，不直接展示给用户

说明：签约接口事件

signContractDone 已经获取到相关签约参数，准备跳转到签约页面之前触发，调用方可以在这个事件内把自定义的 loading 取消。注意需要在调用了 signContract 接口之后才能注册这个事件。

示例：

```
OpenMidas.once('signContractDone',function(){
    //调用方代码
})
```

小程序签约接口

说明：将 openMidas.js 文件放入工程目录，并通过如下方式引入

```
var OpenMidas=require("openMidas");
```

为了支持多域名情况，除了 openMidas.js 还有一个额外的 openMidasConfig.js 用于域名配置，在 require jsapi 之前需要引入这个配置文件，默认情况下 openMidas 会走公有云的环境。

接口：

```
OpenMidas.init(env)
```

参数说明如下：

参数名	参数类型	必填	参数说明
-----	------	----	------

参数名	参数类型	必填	参数说明
env	String	是	环境，release 表示正式环境，test 表示测试环境

说明：签约接口

接口：

```
OpenMidas.signContract(Object params,Function errorCallback)
```

参数说明如下：

参数名	参数类型	必填	参数说明
params.appld	String	是	米大师的应用 ID
params.userId	String	是	用户 ID
errorCallback	Function	否	当内部参数校验不通过或者后台返回错误时，会执行回调，回调参数参见下表“错误回调参数说明”。

错误回调参数说明：

属性	类型	取值
resultCode	Int	-1 签约流程失败流程失败,-3 参数错误,-101 重复签约
innerCode	String	系统内部错误码，不直接展示给用户，以下特殊 innerCode 需要调用方特殊处理： 402-1-2-1 小程序签约接口不存在，原因是微信客户端版本未满足 Android：6.5.10，IOS：6.5.9。402-1-2-2 未成功跳转到小程序签约，此时 resultMsg 透传微信侧返回的错误。
resultMsg	String	返回信息，不直接展示给用户

说明：签约接口事件

signContractDone 已经获取到相关签约参数，准备跳转到签约页面之前触发，调用方可以在这个事件内把自定义的 loading 取消。注意需要在调用了 signContract 接口之后才能注册这个事件。

```
var OpenMidas=require("openMidas");
OpenMidas.once('signContractDone',function(){
//调用方代码
})
```


签约回调

小程序签约回调是由微信侧回调到App层的，详见微信文档 [小程序纯签约](#)。

用户签约完成之后，会跳转回商户小程序，可通过onShow(OBJECT)所携带的参数判断判断用户由签约小程序返回商户小程序，OBJECT返回参数请查看[小程序开发文档onShow参数说明](#)

referrerInfo.extraData字段说明

返回码	变量名	必填	示例值	类型	描述
返回码	return_code	是	SUCCESS	String	客户端小程序收到的签约结果 SUCCESS:签约成功 FAIL:签约失败
错误信息	return_msg	是		String	签约失败的错误信息
委托代扣协议id	contract_id	是	201710180325670965	String	签约成功后微信返回的委托代扣协议id

注意：如果用户正常点击微信签约页的确定按钮返回商户小程序，那么会返回extraData；如果用户点击浏览器左上角的返回，则不返回extraData

示例：

```
App({
  onShow(res) {
    if (res.scene === 1038) { // 场景值1038: 从被打开的小程序返回
      const { appId, extraData } = res.referrerInfo
      if (appId === 'wxbd687630cd02ce1d') { // appId为wxbd687630cd02ce1d: 从签约小程序跳转回来
        if (typeof extraData === 'undefined'){
          // TODO
          // 客户端小程序不确定签约结果，需要向商户侧后台请求确定签约结果
          return;
        }
        if(extraData.return_code === 'SUCCESS'){
          // TODO
          // 客户端小程序签约成功，需要向商户侧后台请求确认签约结果
          var contract_id = extraData.contract_id
          return;
        } else {
          // TODO
          // 签约失败
          return;
        }
      }
    }
  }
})
```

开发者手册

支付渠道申请指引

最近更新时间：2019-04-17 16:12:23

准备工作

Payment 支持微信支付和手 Q 支付的 [APP 支付](#) 模式，您可以通过在移动端 APP 中集成 Payment SDK 来调起微信支付和手 Q 支付。

配置微信支付

您需要先在 [微信开放平台](#) 和 [微信商户平台](#) 上申请好相关信息，然后才能给应用配置微信支付功能。下面将详细说明如何在 MobileLine 平台上配置 Payment 的微信支付渠道。

第一步：注册微信开放平台账号并进行认证

登录 [微信开放平台](#)，单击首页右上角的【注册】按钮，注册成为微信开放平台开发者。

第二步：开发者资质认证

开放平台需进行开发者资质认证后才可申请微信支付，可通过单击【账号中心】>【开发者资质认证】来进行认证，认证费：300 元/次。

第三步：创建 App

单击【管理中心】>【创建移动应用】，在开放平台上创建 App，审核通过后即可获得该应用的 AppID 以及 AppSecret。

第四步：提交资料申请微信支付

单击【管理中心】，选择需要申请支付功能对应的 App，在微信支付一栏中单击【申请开通】按钮，然后开始填写资料等待审核，审核时间为 1 - 5 个工作日内。

第五步：开户成功，登录商户平台进行验证

资料审核通过后，请登录联系人邮箱查收商户号和密码，并登录 [商户平台](#) 填写财付通备付金打的小额资金数额，完成账户验证（[查看验证方法](#)）。

第六步：在线签署协议

本协议为线上电子协议（[点此提前预览协议内容](#)），签署后方可进行交易及资金结算，签署完立即生效。

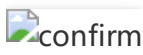
第七步：配置渠道信息

打开之前在 [MobileLine 平台](#) 上创建的应用，单击【我的米大师】>【渠道管理】，然后单击【开通微信支付】：

填入之前申请好微信商户平台上的商户号和商户号密钥，以及微信开放平台上审核通过的应用 AppID 和 AppKey，然后单击【确认】。

第八步：设置支付参数

单击【我的米大师】>【参数配置】，单击【修改】：



配置好对应的渠道信息、回调地址和平台参数后，单击【保存&预览】：

到此，您即已经完成了微信支付的配置。

注意：

其中回调地址是支付完成后服务器回调的地址，而平台参数则是用于鉴定来源的 App 信息，安卓平台下为应用签名和应用包名，iOS 平台下为 Bundle Identifier。

配置手 Q 支付

您需要先在 [QQ 钱包商户平台](#) 和 [腾讯开放平台](#) 上申请好相关信息，然后才能给应用配置 QQ 支付功能。下面将详细说明如何在 MobileLine 平台上配置 Payment 的手 Q 支付。

第一步：在 QQ 钱包上填写资料

登录 [QQ 钱包商户平台](#)，点击【立即接入】，然后按照指引填写相关资料，提交的资料将会在 3 ~ 5 个工作日内审核完成。资料提交后，QQ 钱包将会向商户银行账户汇一笔金额随机的验证款，以确认账户真实性。

第二步：查收开户邮件

在所有资料审核通过后，将会收到 QQ 钱包的开户邮件，包含了商户平台的登录账户和密码等重要信息。

第三步：查收款项

您提交资料后，将会收到一笔来自“财付通支付科技有限公司客户备付金”账户的随机金额款项，后续需要通过金额来对账户进行验证。

第四步：验证账户

您可以使用开户邮件中提供的登录账号和登录密码登录商户平台，然后单击【去验证】，在商户平台验证账户页面回填收到的金额，金额正确将会通过验证。

第五步：在线签署协议

账户验证成功后，即可在线签署协议，确认相关信息无误后，单击【确定签署】完成协议签署。

第六步：登录腾讯开放平台

首先登录 [腾讯开放平台](#)，单击首页右上角的【登录】按钮进行登录。

第七步：创建 App

登录腾讯开放平台后，单击【应用接入】>【应用接入】，

接着单击【创建应用】按钮来创建新的应用。

第八步：在 QQ 钱包商户平台上添加应用

登录 [QQ 钱包商户平台](#)，单击【账户管理】>【开发配置】，然后单击【上传材料】按钮将之前在腾讯开放平台上创建的 App 添加到商户平台上，添加成功后，会在两个工作日内生效。

第九步：配置渠道信息

打开之前在 [MobileLine 平台](#) 上创建的应用，单击【我的米大师】>【渠道管理】，然后单击【开通 QQ 钱包支付】：

填入之前申请的 QQ 商户号上的商户号和商户号密钥，腾讯开放平台上审核通过的应用 AppID 和 AppKey 然后点击【确认】。

第十步：设置支付参数

点击【我的米大师】>【参数配置】，单击【修改】：

配置好对应的渠道信息、回调地址和平台参数后，单击【保存&预览】：

到此，您即已经完成了 QQ 支付的配置。

注意：

其中回调地址是支付完成后服务器回调的地址，而平台参数则是用于鉴定来源的 App 信息，安卓平台下为应用签名和应用包名，iOS 平台下为 Bundle Identifier。

发货服务器配置

最近更新时间：2019-04-17 16:09:17

发货回调通知（必选）

注意：

支付成功后，Payment 后台会将支付结果通知应用服务器，应用需要接收处理，并返回应答。如果没有收到业务的成功应答，Payment 后台会通过一定的策略定期重新发起通知，尽可能提高通知的成功率，但不保证通知最终能成功。（通知间隔频率为15/15/30/180/1800/1800/1800/1800/3600，单位：秒）

请求方式：POST

参数格式：JSON

参数说明如下：

字段名	类型	含义
appid	string[50]	Payment 的 offerId
sub_appid	string[50]	子应用 ID
user_id	string[255]	用户 ID，长度不小于 5 位，仅支持字母和数字的组合。
out_trade_no	string[32]	应用支付订单号。
product_id	string[128]	商品 id，仅支持数字、字母、下划线（_）、横杠字符（-）、点（.）的组合。
currency_type	string[3] ISO	货币代码，CNY
amount	int	支付金额，单位：分
pay_channel	string[10]	"实际支付渠道：wechat：微信支付；qqwallet：QQ 钱包；bank：网银"
pay_scene	int	"支付场景：1：AndroidApp 支付；2：IOSApp 支付；3：PCWeb 支付；4：H5/公众号支付；5：小程序支付；6：扫码支付"
pay_channel_orderid	string[255]	支付渠道侧订单号
metadata	string[255]	透传字段，支付成功回调透传给应用，用于业务透传自定义内容。

字段名	类型	含义
sub_out_trade_no_list	array	调用下单接口传进来的 sub_out_trade_no_list
ts	string[10]	UNIX 时间戳 (格林威治时间) ,精确到秒。
sign	string[32]	请求签名 。

返回 JSON 结果：

字段名	类型	含义
ret	int	结果码 0 : 成功 ; -1 : 失败
msg	string[255]	成功固定为小写的“ok”，失败则说明错误原因，不能返回空 (使用 utf-8 编码) 。

服务器端 API

最近更新时间：2018-08-27 18:03:30

通用服务端接口

商品下单

说明：聚合支付模式和账户托管模式在您的应用发起支付前，都必须先调用商品下单接口来进行下单，并将应答的 pay_info 透传给米大师 SDK，拉起支付。

接口： unified_order

地址： `https://api.openmidas.com/v1/r/$appid$/unified_order`

注意：聚合支付模式下 appid 为您 MobileLine 控制台上移动支付的 offerid。
 托管账户模式下 appid 为该托管账户的 id。

请求方式： POST

参数格式： k=v

参数说明如下：

字段名	类型	必填	含义
user_id	string[255]	是	用户 ID，长度不小于 5 位，仅支持字母和数字的组合。
out_trade_no	string[32]	是	应用支付订单号，米大师要求应用订单号保持唯一性（建议根据当前系统时间加随机序列来生成订单号）。重新发起一笔支付要使用原订单号，避免重复支付；已支付过或已调用关单的订单号不能重新发起支付。仅支持数字、字母、下划线（_）、横杠字符（-）、点（.）的组合。
product_id	string[128]	是	商品 id，仅支持数字、字母、下划线（_）、横杠字符（-）、点（.）的组合。
currency_type	string[3]	是	ISO 货币代码，CNY
amount	int	是	支付金额，单位：分
product_name	string[128]	是	商品名称

字段名	类型	必填	含义
product_detail	string[255]	是	商品详情
ts	string[10]	是	UNIX 时间戳（格林威治时间），精确到秒。
sign	string[32]	是	请求签名
sub_appid	string[50]	否	子应用 ID
channel	string[10]	否	指定支付渠道：wechat：微信支付；qqwallet：QQ 钱包
type	string[10]	否	如果是账户托管充值，值必须为 save
metadata	string[255]	否	透传字段，支付成功回调透传给应用，用于业务透传自定义内容。
num	int	否	购买数量，托管账户充值下有效。
callback_url	string[255]	否	Web 端回调地址
wx_openid	string[50]	否	微信公众号支付时，需要传微信下的openid
platform_income	int	否	平台应收金额，单位：分
buss_settle_amount	int	否	结算应收金额，单位：分
platform_income_detail	array[100]	否	平台应收金额详情。格式：json，如：{"A":50,"B":50}
sub_out_trade_no_list	array[512]	否	子订单信息列表

子订单信息列表格式：子订单号、子应用 ID、金额

```
[
  {
    "sub_out_trade_no": "2017112700001",
    "sub_appid": "1450000766",
    "amount": 999,
    "product_name": "子订单物品名称",
    "product_detail": "子订单物品描述",
    "platform_income": 100,
    "buss_settle_amount": 899,
    "platform_income_detail": {"A":50,"B":50},
    "metadata": "test1"
  },
  {
```

```

"sub_out_trade_no" : "2017112700002",
"sub_appid" : "900000490",
"amount" : 888,
"product_name" : "子订单物品名称",
"product_detail" : "子订单物品描述",
"platform_income" : 100,
"buss_settle_amount" : 788,
"platform_income_detail" : {"A":50,"B":50},
"metadata" : "test2"
}
]
    
```

返回 JSON 结果

字段名	类型	含义
ret	int	结果码 0 : 成功 ; 其他 : 失败
msg	string[512]	失败的错误信息。
transaction_id	string[32]	米大师的交易订单。
out_trade_no	string[32]	应用支付订单号
pay_info	string[512]	支付参数, 透传给米大师 SDK

示例请求

```

POST https://api.openmidas.com/v1/r/TC100008/unified_order?amount=1&channel=wechat&currency_type=CNY&original_amount=1&out_trade_no=open_1519638186904&product_detail=你懂得&product_id=product_test&product_name=金元宝&sign=ZyF0PZ0lzuyZtND6QPD78Cb59qEHd/VCugJ90ID8S+p9umy6M/6wbLDbhIEV+xwQEqGjMomHa/L+AgNtfT5xLQ==&ts=1519609386&user_id=rickenwang HTTP/1.1
User-Agent: Fiddler
Host: api.openmidas.com
Content-Length: 0
    
```

示例响应

```

HTTP/1.1 200 OK
Server: nginx/1.12.2
Date: Mon, 26 Feb 2018 10:22:05 GMT
Content-Type: text/html;charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
    
```

UUID: 5a93dfcd-bb51-41be-96bb-a49438514eae

METRIC: 1

```
{
  "ret": 0,
  "transaction_id": "E-180226180100230001",
  "out_trade_no": "open_1519640480282",
  "pay_info": {
    "data": {
      "appid": "TC100008",
      "user_id": "rickenwang",
      "out_trade_no": "open_1519640480282",
      "product_id": "product_test",
      "pay_method": "wechat"
    }
  },
  "sign": "cQHQYAdrn4ECpiAGiWqZEcynw+eNhC2AdXqtYoQqZvXn2+CupvB4fcH+cUQEPZfikgjwxM2oGylmPTtFIMp1HXUJz0sNH5C1ZEUR00/Q/kdEQyZg32MwC3Pom0d6IbM3L+BLcX8Nbf3QD7mrneL9ahng+Z8eo50jXhRNQv3UGPHiqSoR7JVjplQEq45vj8eVjzi5SSDFF6OGmZgn8WiYXdGXd7bOTpTakOnDukLOInzwUHKvKk70k5PjaRneJlbEu0xG0pQCpk0253fg8RhJola2Ej8iGFAyPbnjPH7oScDGSPir0PYTtD3zKoll5dFV8FzKi/VufiB3yrO4z0knQ=="
}
```

聚合支付服务端接口

查询订单

说明：根据订单号，或者用户 ID，查询支付订单状态

接口：query_order

地址： [https://api.openmidas.com/v1/r/\\$appid\\$/query_order](https://api.openmidas.com/v1/r/$appid$/query_order)

物品直购下 `appid` 为您 MobileLine 控制台上移动支付的 `offerid`。
 账户充值下 `appid` 为该账户的 id。

请求方式：POST

参数格式：k=v

参数说明如下：

字段名	类型	必填	含义
user_id	string[255]	是	用户ID，长度不小于 5 位，仅支持字母和数字的组合。
type	string[10]	是	type=by_order，根据用户 id 查订单；type=by_user，根据订单号查订单
ts	string[10]	是	unix时间戳（格林威治时间），精确到秒。
sign	string[32]	是	请求签名
type=by_order，以下两个字段，必须传一个，都传的话，优先使用out_trade_no			

out_trade_no	string[32]	否	应用支付订单号
transaction_id	string[32]	否	调用下单接口获取的米大师交易订单。
type=by_user			
starttime	string[64]	是	查询的起始时间，格式：unix 时间戳（格林威治时间），精确到秒。
endtime	string[64]	是	查询的结束时间，格式：unix 时间戳（格林威治时间），精确到秒。
num	int	是	每页返回的记录数。根据用户号码查询订单列表时需要传，用于分页展示。
off_set	int	是	记录数偏移量，默认从 0 开始。根据用户号码查询订单列表时需要传，用于分页展示。

返回 JSON 结果

字段名	类型	含义
ret	int	结果码 0：成功；其他：失败
msg	string[512]	失败的错误信息。
appid	string[50]	米大师的应用 ID
sub_appid	string[50]	子应用 ID
out_trade_no	string[255]	应用支付订单号
sub_out_trade_no_list	array	调用下单接口传进来的sub_out_trade_no_list
transaction_id	string[32]	调用下单接口获取的米大师交易订单。
user_id	string[32]	用户 ID
pay_channel	string[12]	支付渠道：wechat：微信支付；qqwallet：QQ 钱包；bank：网银
product_id	string[255]	物品 id
metadata	string[255]	发货标识，由业务在调用米大师下单接口的时候下发。
currency_type	string[3]	ISO 货币代码，CNY
amount	int	支付金额，单位：分。

字段名	类型	含义
order_state	string[2]	当前订单的订单状态 0：初始状态，获取米大师交易订单成功；1：拉起米大师支付页面成功，用户未支付；2：用户支付成功，正在发货；3：用户支付成功，发货失败；4：用户支付成功，发货成功；5：米大师支付页面正在失效中；6：米大师支付页面已经失效；"
order_time	string[24]	下单时间，格式：YYYY-MM-DD hh:mm:ss
pay_time	string[24]	支付时间，格式：YYYY-MM-DD hh:mm:ss
callback_time	string[24]	支付回调时间，格式：YYYY-MM-DD hh:mm:ss
refund_time	string[24]	退款时间，格式：YYYY-MM-DD hh:mm:ss

关闭订单

说明：根据订单号关闭订单

接口：close_order

地址： [https://api.openmidas.com/v1/r/\\$appid\\$/close_order](https://api.openmidas.com/v1/r/$appid$/close_order)

物品直购下 `appid` 为您 MobileLine 控制台上移动支付的 `offerid`。
 账户充值下 `appid` 为该账户的 id。

请求方式：POST

参数格式：k=v

参数说明如下：

字段名	类型	必填	含义
user_id	string[255]	是	用户ID，长度不小于 5 位，仅支持字母和数字的组合。
ts	string[10]	是	unix时间戳（格林威治时间），精确到秒。
sign	string[32]	是	请求签名
以下两个字段，必须传一个，都传的话，优先使用out_trade_no			
out_trade_no	string[32]	否	应用支付订单号。优先使用应用交易订单号。
transaction_id	string[32]	否	调用下单接口获取的米大师交易订单。

返回 JSON 结果

字段名	类型	含义
ret	int	结果码 0：成功；其他：失败
msg	string[512]	失败的错误信息。

退款

说明：如交易订单需退款，可以通过本接口将支付款全部或部分退还给付款方，米大师将在收到退款请求并且验证成功之后，按照退款规则将支付款按原路退回到支付帐号。最长支持 1 年的订单退款。

接口：refund

地址：[https://api.openmidas.com/v1/r/\\$appid\\$/refund](https://api.openmidas.com/v1/r/$appid$/refund)

物品直购下 `appid` 为您 MobileLine 控制台上移动支付的 `offerid`。
 账户充值下 `appid` 为该账户的 id。

请求方式：POST

参数格式：k=v

参数说明如下：

字段名	类型	必填	含义
user_id	string[255]	是	用户ID，长度不小于 5 位，仅支持字母和数字的组合。
refund_id	string[32]	是	退款订单号，仅支持数字、字母、下划线 (_)、横杠字符 (-)、点 (.) 的组合。
amount	int	是	退款金额，单位：分
platform_income	int	否	平台应收金额，单位：分
buss_settle_amount	int	否	结算应收金额，单位：分
platform_income_detail	array[100]	否	平台应收金额详情。格式：json，如：{"A":50,"B":50}
ts	string[10]	是	unix时间戳（格林威治时间），精确到秒。
sign	string[32]	是	请求签名

以下两个字段，必须传一个，都传的话，优先使用 out_trade_no			
out_trade_no	string[32]	否	应用支付订单号。
transaction_id	string[32]	否	调用下单接口获取的米大师交易订单。
如果需要子订单进行退款，需要传以下字段			
sub_out_trade_no	string[32]	否	子订单号，一次只支持一个子订单号。

返回 JSON 结果

字段名	类型	含义
ret	int	结果码 0：成功；其他：失败
msg	string[512]	失败的错误信息。

查询退款结果

说明：提交退款申请后，通过调用该接口查询退款状态。退款可能有一定延时，用微信零钱支付的退款约 20 分钟内到账，银行卡支付的退款约 3 个工作日后到账。

接口：query_refund

地址：[https://api.openmidas.com/v1/r/\\$appid\\$/query_refund](https://api.openmidas.com/v1/r/$appid$/query_refund)

物品直购下 `appid` 为您 MobileLine 控制台上移动支付的 `offerid`。
 账户充值下 `appid` 为该账户的 id。

请求方式：POST

参数格式：k=v

参数说明如下：

字段名	类型	必填	含义
user_id	string[255]	是	用户 ID，长度不小于 5 位，仅支持字母和数字的组合。
refund_id	string[32]	是	退款订单号，应用需要保持唯一性。仅支持数字、字母、下划线（_）、横杠字符（-）、点（.）的组合。"
ts	string[10]	是	unix 时间戳（格林威治时间），精确到秒。

字段名	类型	必填	含义
sign	string[32]	是	请求签名

返回 JSON 结果

字段名	类型	含义
ret	int	结果码 0 : 成功 ; 其他 : 失败
msg	string[512]	失败的错误信息。
refund_id	string[32]	退款订单号
state	int	退款状态码 1 : 退款中 ; 2 : 退款成功 ; 3 : 退款失败 ;

托管账户服务端接口

查询余额

说明 : 查询查询账户余额

接口 : get_balance

地址 : [https://api.openmidas.com/v1/r/\\$accoutid\\$/get_balance](https://api.openmidas.com/v1/r/$accoutid$/get_balance)

请求方式 : POST

参数格式 : JSON

请求参数说明 :

字段名	类型	必填	含义
user_id	string[255]	是	用户ID, 长度不小于5位, 仅支持字母和数字的组合。
ts	string[10]	是	unix时间戳 (格林威治时间), 精确到秒。
sign	string[32]	是	签名

返回的 JSON 结果

字段名	类型	含义
ret	int	结果码 0 : 成功, 其他 : 失败

字段名	类型	含义
balance	int	当前余额
gen_balance	int	当前赠送部分余额
save_amt	int	累计充值金额

扣款

说明：扣款

接口：pay

地址：https://api.openmidas.com/v1/r/\$accoutid\$/pay

请求参数：POST

参数格式：JSON

请求参数说明：

字段名	类型	必填	含义
user_id	string[255]	是	用户ID，长度不小于5位，仅支持字母和数字的组合。
amt	int	是	扣款金额
billno	string[32]	是	订单号，仅支持数字、字母、下划线（_）、横杠字符（-）、点（.）的组合。
ts	string[10]	是	unix时间戳（格林威治时间），精确到秒。
sign	string[32]	是	签名

返回的 JSON 结果

字段名	类型	含义
ret	int	结果码 0：成功，其他：失败
billno	string[32]	订单号
balance	int	当前余额
gen_balance	int	当前赠送部分余额
used_gen_amt	int	本次支付使用的赠送部分金额

取消扣款

说明：取消扣款

接口：cancel_pay

地址： [https://api.openmidas.com/v1/r/\\$accoutid\\$/cancel_pay](https://api.openmidas.com/v1/r/$accoutid$/cancel_pay)

请求方式：POST

参数格式：JSON

请求参数说明：

字段名	类型	必填	含义
user_id	string[255]	是	用户ID，长度不小于5位，仅支持字母和数字的组合。
billno	string[32]	是	订单号，仅支持数字、字母、下划线（_）、横杠字符（-）、点（.）的组合。
ts	string[10]	是	unix时间戳（格林威治时间），精确到秒。
sign	string[32]	是	签名

返回的 JSON 结果

字段名	类型	含义
ret	int	结果码 0：成功，其他：失败
billno	string[32]	订单号
balance	int	当前余额
gen_balance	int	当前赠送部分余额

赠送

说明：赠送接口

接口：present

地址： [https://api.openmidas.com/v1/r/\\$accoutid\\$/present](https://api.openmidas.com/v1/r/$accoutid$/present)

请求方式：POST

参数格式：JSON

请求参数说明：

字段名	类型	必填	含义
user_id	string[255]	是	用户ID，长度不小于5位，仅支持字母和数字的组合。
amt	int	是	赠送金额
billno	string[32]	是	订单号，仅支持数字、字母、下划线（_）、横杠字符（-）、点（.）的组合。
ts	string[10]	是	unix时间戳（格林威治时间），精确到秒。
sign	string[32]	是	签名

返回的 JSON 结果

字段名	类型	含义
ret	int	结果码 0：成功，其他：失败
billno	string[32]	订单号
balance	int	当前余额
gen_balance	int	当前赠送部分余额

签名计算

最近更新时间：2018-04-24 19:47:12

签名说明

在调用 Payment 后台接口时，所有的请求均需要携带 sign 参数用于核实请求发起者的身份。对于签名的计算过程，可简单分为如下三个步骤：获取签名所需信息，拼接明文字符串，将明文字符串转化为签名。

获取签名所需信息

生成签名所需信息包括应用的 offerId、offerKey 和 privateKey

拼接明文字符串

在计算请求签名时，去掉 sign 参数，将其他参数 key 值，按照字母表顺序升序排列，再把所有参数以"&"字符连接起来，最后加上业务的 appkey，即得到明文字符串 original。

注意：

appkey 的获取：在接入 Payment 时，Payment 为每个应用分配 appkey，应用负责人可以自行在管理系统中查询到，该 key 值为保密内容，负责人不要对外泄露。

将明文字符串转化为签名

```
sign = Base64.encode(RSA-SHA256(original, Base64.decode(privateKey)))
```

签名示例

获取签名所需信息

```
offerKey : bBJ2la1zfmssX28fhe39dv9OcFe6JFvY
privateKey : MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBCkwggSjAgEAAoIBAQMmGQ2rElxx00P+wtGf5
o7HCa6EAd6CG4JRDalyvxkJGcnVhkutD3KBzftyHvsC3SCFnGuLBDvz3+nIF2HF+DnrL88gX9zslG/I4GxFEs
TLZM4hAyVIGdmQg2m1CBXCqvnDAEG276ijk8nrRdXalLzzivMCs+fbUM31LjHvCzyqhACPFemoisQYw0
gm9ZscQNVbnnTV2VzzWa+EtjHaEC6zFhbyXDEj2VzdpDtQZYldj8tG8IIBCiGVOqxJ4d0l3WR56gWemGU
GKNwEqXnYeyeEZGvPwa9f3fewNmGmhu6uWWCdAUQbUyH3Y8E+srbz+u1KW2QW0KRqqdAW9uPI+
RD1AgMBAAECggEAJod/cGvsquqzcg9xRpM2yUFnxZcCiUrN8YCU7mZtbMays1x/g4UYqX+yYROq+Hn
KnlTo9ShKwBxe0bCloLvGBwnEZylmNuvfjAV4W8Cez2gb1A9QyNf7RGWc58Fo02YJ2ltMdbN/y5RNtjwb
aTSKx0U3F/bEFkdbIXicx+hbyjlq+quCNCDK2kwx6ikgsaa8XY5naEVOZVkdKvDiO+d3dlcInRHCmYfHNIF
aL734vzSZIOfe7+xMW8xSlTGr0Uo3KsoCzC4EveUWmW9kJtxwvHcXHF3WB8gH4TQsodxDeI73uUXCMj
6bq+iDKzQUVBllDnT9wQGCaHLTCrupkAQKBgQD9iEhsvo3UhamPe0pRTnNh4McWMWdLKI4pt0Q6
GtEujx4MVhQprFeBIL9oewsorjhh7OgLtP8Lgs28mX3QtdE8t5VQtrLS3GAVweMBm5gD667XsGL687s9En0
```

```
mq+wX4TdR6LS7djTExPRiW6c4ar5Red++5auUI6k4sTjRi7RgQKBgQDoVmEwBC9w7O9aXv1KBJg5klPes
43syJ1yWo7HWLa0Xn+rQK4Jv8WUyIV5j909oTGwhKlcWiZ2IPBS+frVWSfMRS/bUudV/Lzx/aW3vZnVO9
Q1c79ulCWeGsC0WfxkHWRD2vfw7Iheq1gVffV2uhHU7G7CjTp8eYs5czT0nwnRdQKBgEdqxWQEhXVRN
YNaaopE1k4EFzW79+afdxmqf2IM3zKb936v/72wRxRCWcxieWHVcJl/V2UO694EplOTDBALvxIIDAMecv2
DQfSyD0ktYpgissjLF2Avy7lJ0y9ZB5Z6QbYHgOFD6KkEnbePrB37H6taOqy1eCugO55M1sNiuLuBAoGBA
OV3Qr+gspZOofEt+G516EjpyKgF+xwKAVPh3NiLunGxiFKe31uvsCqFcgLTyAKl4xdG4BGkg5HljwYPCPxt1
5+pCY4NQEuttZMvV/ez+/YdBAbszdPR36FafM7jPSJH4Rofbtf1VO5jPjjj0YY0WZ448V+XCpebl4Tjlv8jAB
htAoGAVY4drZjLv8SNcwjAsE1BGBnYVRtfheODuos2LGBIEU2zKRfd0j0abgv1WxlaT4GH80uKbJFadVXep
lQ4yX3BTunjD24odA2dW3ceXtmg/WU/mAPtA+HkOui+k/hC4Y0syTOVSnHHIJ0J3tYR3Pfo35KniIUM8
HYu1i2UUK6pses=
```

拼接明文字符串

这里以统一下单接口为例，假设除了 `sign` 外，所有的请求参数如下：

字段名	值
user_id	rickenwang
out_trade_no	open_1519652529956
product_id	product_test
currency_type	CNY
amount	1
original_amount	1
product_name	金元宝
product_detail	你懂得
ts	1519623729
channel	wechat

将参数按照字母表排序，并用 `&` 链接后，可以得到明文字符串 `original` 为：

```
amount=1&channel=wechat&currency_type=CNY&original_amount=1&out_trade_no=open_151969
8041025&product_detail=你懂得&product_id=product_test&product_name=金元宝&ts=1519669241&
user_id=rickenwangbBJ2la1zfmssX28fhe39dv9OcfE6JFvY
```

将明文字符串转化为签名

示例代码（JAVA）：

```
public static String encode(String original, String privateKey) {  
  
    // 私钥需要进行Base64解密  
    byte[] b1 = Base64.getDecoder().decode(privateKey);  
  
    try {  
        // 将字节数组转换成PrivateKey对象  
        PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(b1);  
        KeyFactory kf = KeyFactory.getInstance("RSA");  
        PrivateKey pk = kf.generatePrivate(spec);  
  
        Signature privateSignature = Signature.getInstance("SHA256withRSA");  
        privateSignature.initSign(pk);  
        // 输入需要签名的内容  
        privateSignature.update(original.getBytes("UTF-8"));  
        // 拿到签名后的字节数组  
        byte[] s = privateSignature.sign();  
  
        // 将签名后拿到的字节数组做一个Base64编码，以便以字符串的形式保存  
        return Base64.getEncoder().encodeToString(s);  
    } catch (NoSuchAlgorithmException e) {  
        e.printStackTrace();  
    } catch (InvalidKeyException e) {  
        e.printStackTrace();  
    } catch (InvalidKeySpecException e) {  
        e.printStackTrace();  
    } catch (SignatureException e) {  
        e.printStackTrace();  
    } catch (UnsupportedEncodingException e) {  
        e.printStackTrace();  
    }  
  
    return "";  
}
```

计算得到最终的签名 sign 为：

```
PfxjspbME7SRtIWj+QPRvjndLtQUupausGJV2DfPHXGGcyPErB5SK96MBOWCK3clewDe3VVb0g/epirP3k  
HFN/nXlv43zBrqfU1vUMvqFRX1IMWM/A1JD3k8IZ/VZi+wZLcvtvhMuVcfQuFXHlnLp5lOa+jp22vuVoC  
RyDG6HPjx9zDELzUUObwSaN9zlaeL9llcx+NKaLHbMxDMHRRWhkuQiFABvkoJe1NiW6JudhSTjNjcBM  
0luEVyz/d9sxBNMKtKvc4+yfv16HJBQLHhYaQB/FBJ/QbVJPyt8tajkQp3bF52zMXTqmUhRs3YoQ2PBzkN  
aKktsdmq5wA5Zsjxg==
```

常见问题

最近更新时间：2018-03-28 16:50:40

支付成功后，收支日报中没有数据

请查看 Payment 支付成功回调时，是否返回了发货成功的应答，只有发货成功的支付数据才会显示到收支日报中。