移动推送

SDK 文档

腾讯云

文档目录

SDK 文档
Harmony 接入指南
简介
SDK 合规使用指南
SDK 集成
接口文档
鸿蒙参数申请指南
鸿蒙通道高级功能
鸿蒙消息分类功能使用说明
鸿蒙通道抵达回执获取指南
错误码
Android 接入指南
简介
SDK 合规使用指南
SDK 集成
接口文档
通知分类说明
厂商通道接入指南
华为通道 V5 接入
荣耀通道接入
小米通道接入
魅族通道接入
FCM 通道接入
vivo 通道接入
OPPO 通道接入
厂商通道高级功能
角标适配指南
厂商通道测试方法
厂商通道注册失败排查指南
厂商消息分类功能使用说明
厂商通道抵达回执获取指南
厂商通道限额说明
厂商通道 QPS 限制说明
错误码
iOS 接入指南
简介
SDK 合规使用指南
SDK 集成
接口文档
推送证书获取指引
推送环境选择说明
错误码
拓展功能
通知服务扩展的使用说明
客户端集成插件
macOS 接入指南
简介
SDK 集成
接口文档
推送证书说明

SDK 文档
Harmony 接入指南
简介

最近更新时间：2024-12-18 18:34:03

移动推送（Tencent Push Notification Service）是一款专业的移动 App 推送平台，支持百亿级的通知和消息推送，秒级触达移动用户，现已全面支持 Harmony、Android 和 iOS 三大主流平台，开发者可以方便地通过嵌入 SDK，通过 API 调用或 Web 端可视化操作，实现对特定用户推送，大幅提升用户活跃度，有效唤醒沉睡用户，并实时查看推送效果。

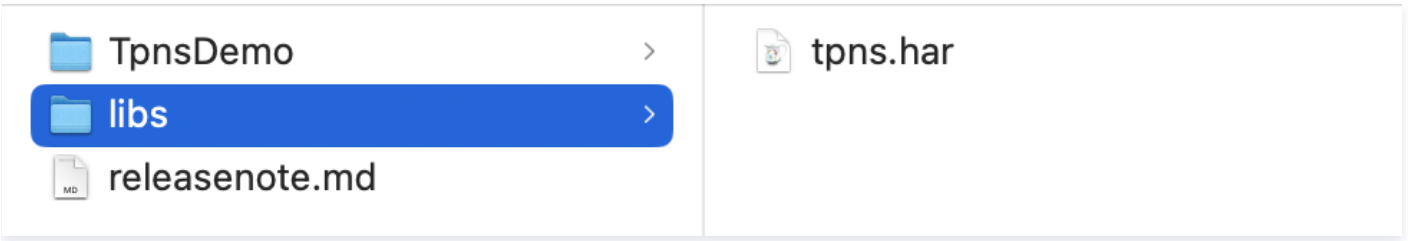
功能说明

Harmony SDK 是移动推送服务为客户端实现消息推送而提供给开发者的接口，主要负责完成以下功能：

- 提供通知和消息两种推送形式，方便用户使用。
- 账号、标签与设备的绑定接口，以便开发者实现特定群组的消息推送，丰富推送方式。
- 点击量上报，统计消息被用户点击的次数。
- 提供鸿蒙厂商通道集成功能，方便用户使用鸿蒙厂商推送。

SDK 说明

获取到移动推送 Harmony SDK 的包，解压后内容如下：。



详细集成步骤可以参考 SDK 集成文档 。

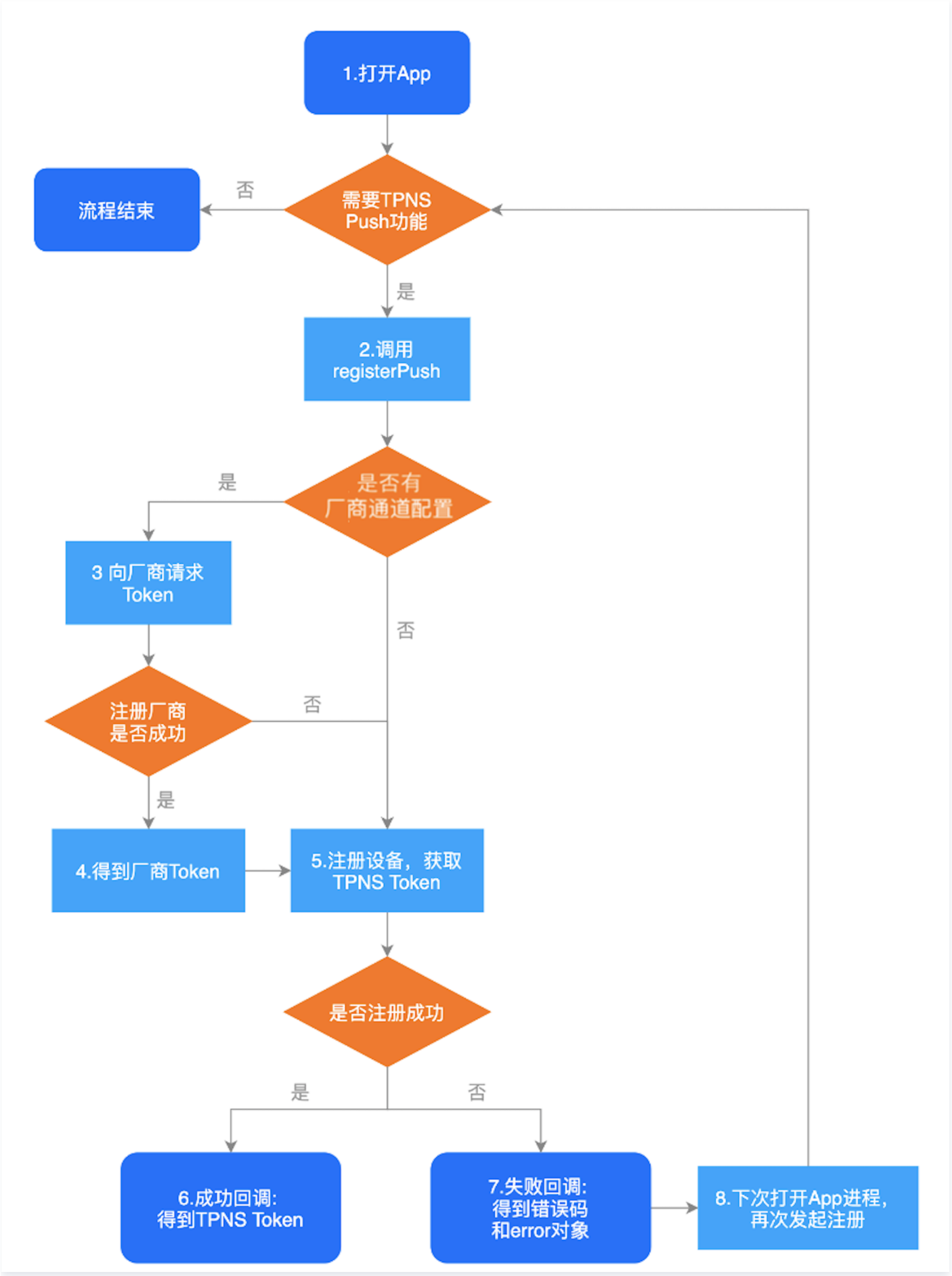
解压后根目录两个文件夹内容：

- TpnsDemo 文件夹：移动推送官方 Demo 程序，用户可以参考相关配置。
- libs 文件夹：包含移动推送的 sdk 文件

常用场景流程说明

设备注册流程

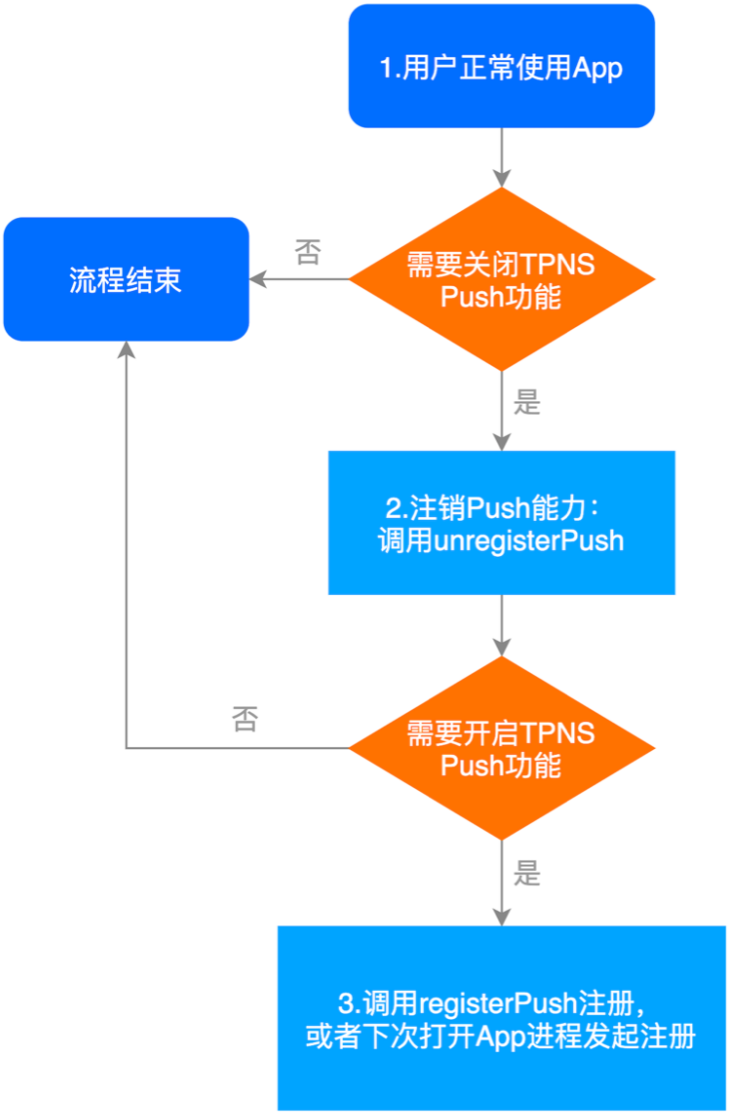
下图为设备注册相关流程，具体接口方法请查看 启动与注册 。


1.打开App
需要TPNS Push功能
否
流程结束
是
2.调用 registerPush
是否有 厂商通道配置
是
3 向厂商请求 Token
否
注册厂商 是否成功
否
是
4.得到厂商Token
5.注册设备，获取 TPNS Token
是否注册成功
是
否
6.成功回调: 得到TPNS Token
7.失败回调: 得到错误码 和error对象
8.下次打开App进程，再次发起注册

> **注意：**
> 如何配置厂商通道 Client ID 参考：厂商 APPID 配置指引

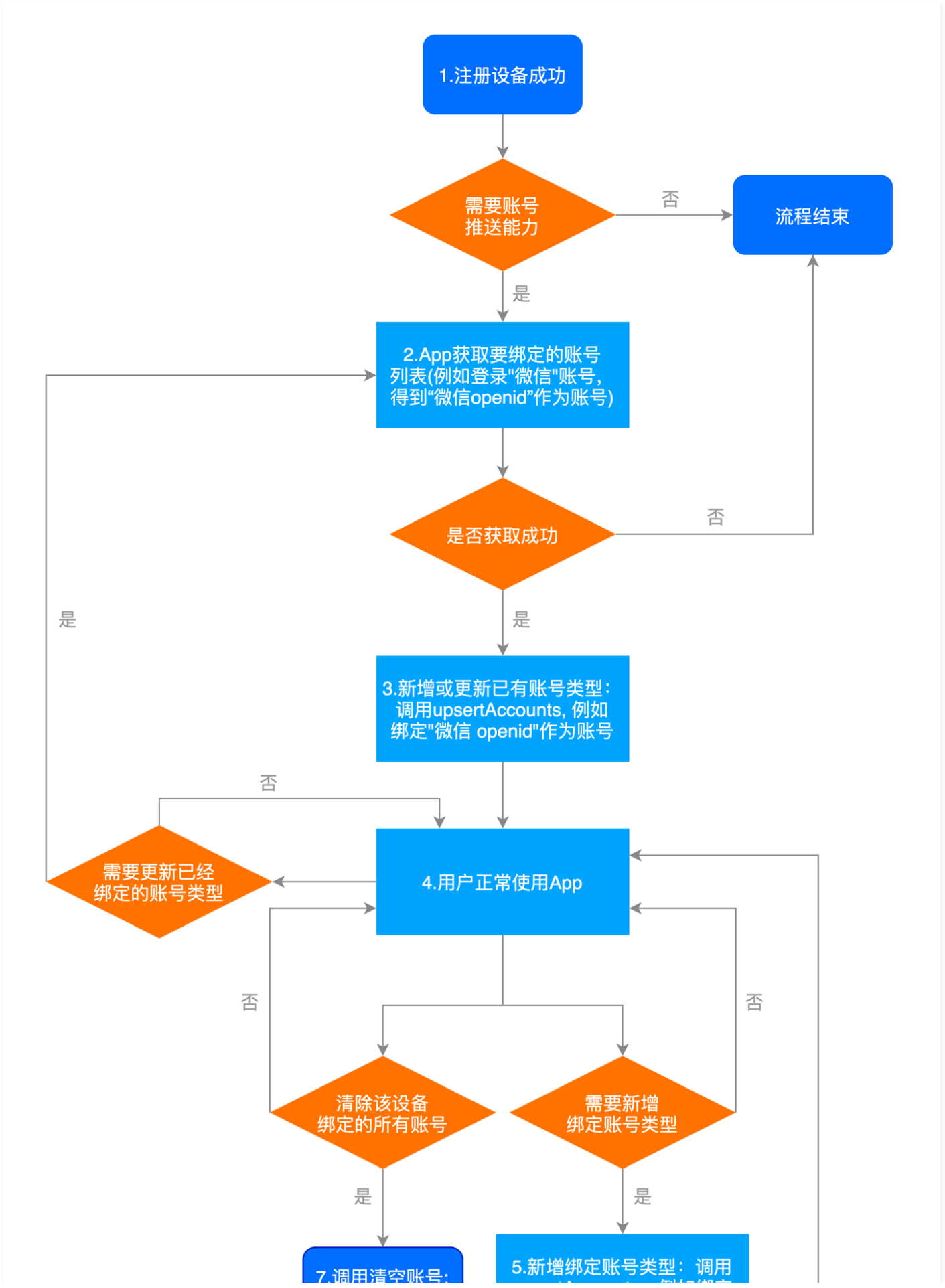
设备反注册流程

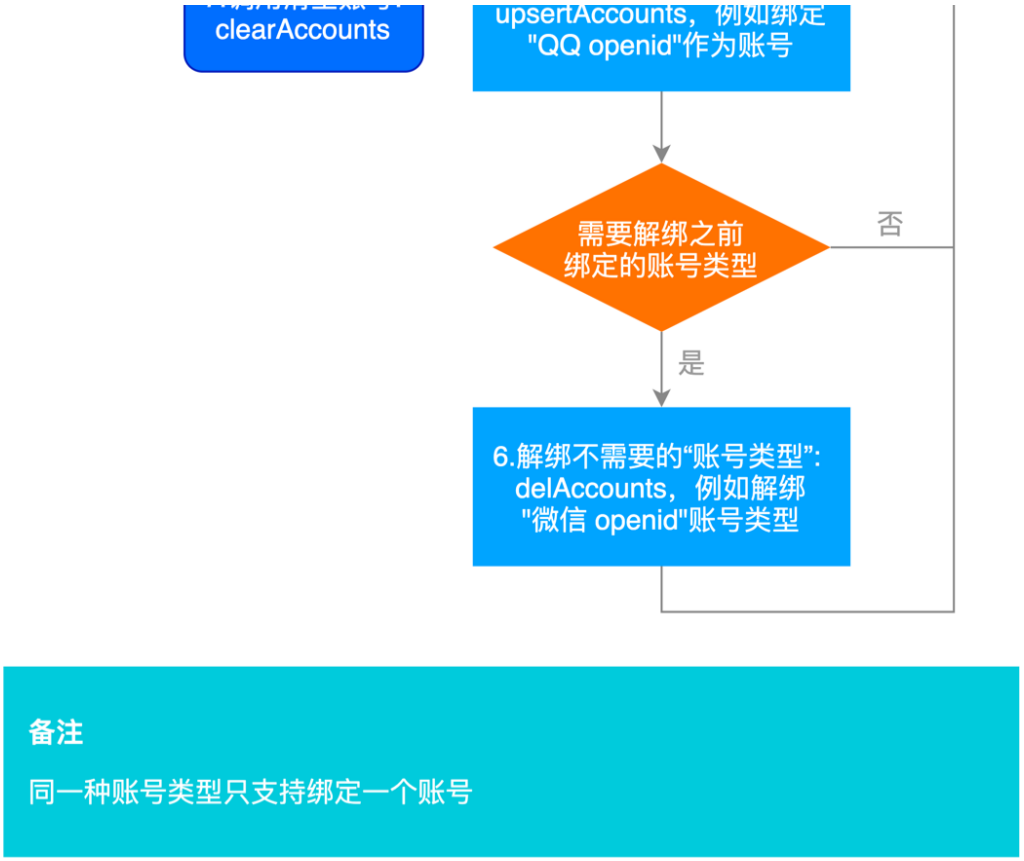
下图为设备反注册相关流程，具体接口方法请查看 反注册 。



账号相关流程

下图为账号相关流程，具体接口方法请查看 账号管理 。


1.注册设备成功
需要账号
推送能力
否
流程结束
是
2.App获取要绑定的账号
列表(例如登录"微信"账号,
得到"微信openid"作为账号)
是否获取成功
否
是
是
3.新增或更新已有账号类型:
调用upsertAccounts, 例如
绑定"微信 openid"作为账号
否
需要更新已经
绑定的账号类型
4.用户正常使用App
否
否
清除该设备
绑定的所有账号
需要新增
绑定账号类型
是
是
5.新增绑定账号类型：调用

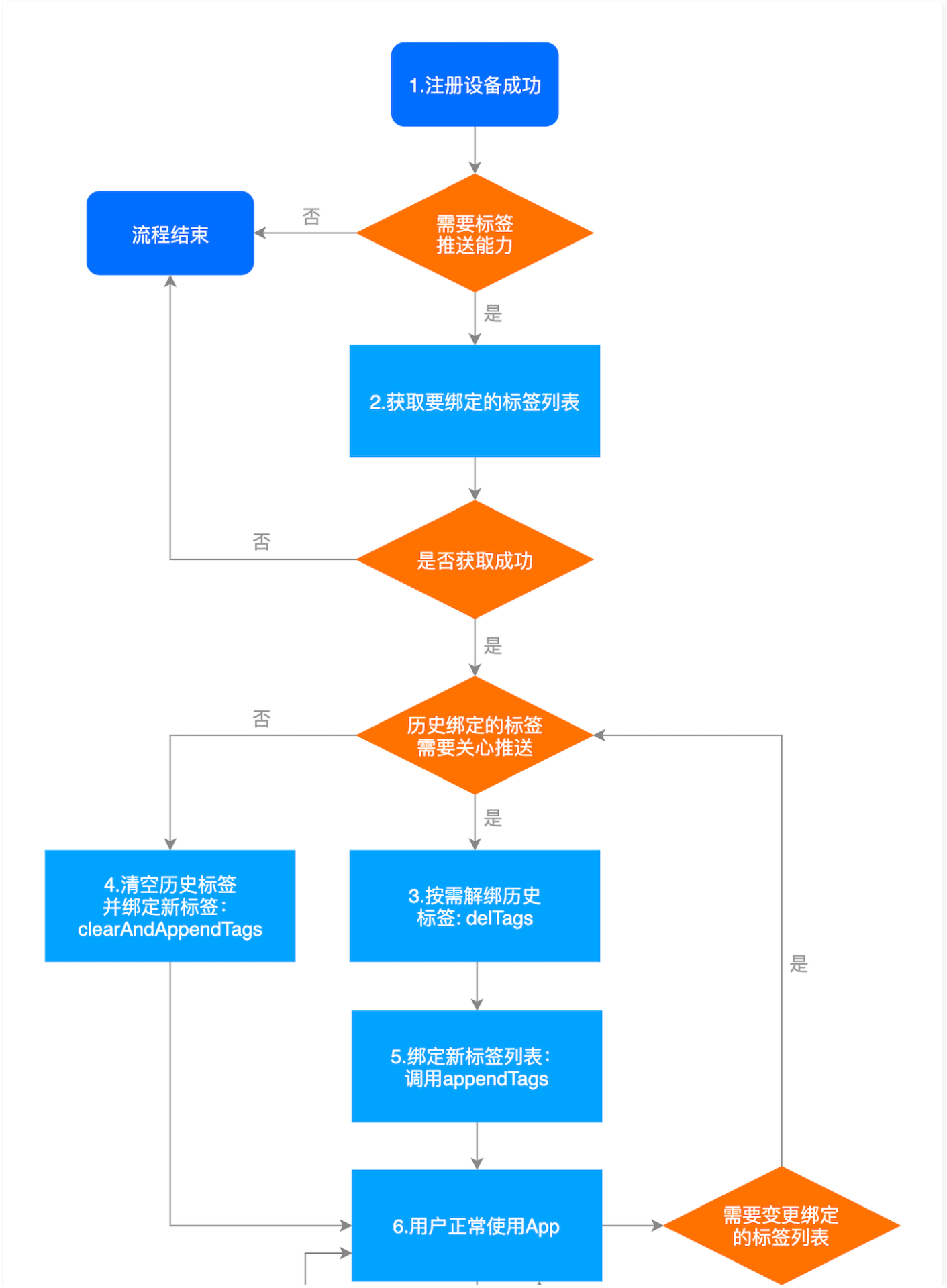


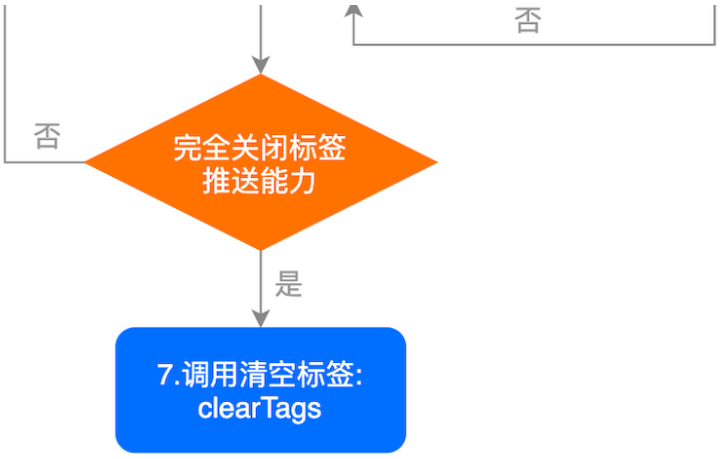
备注

同一种账号类型只支持绑定一个账号

标签相关流程

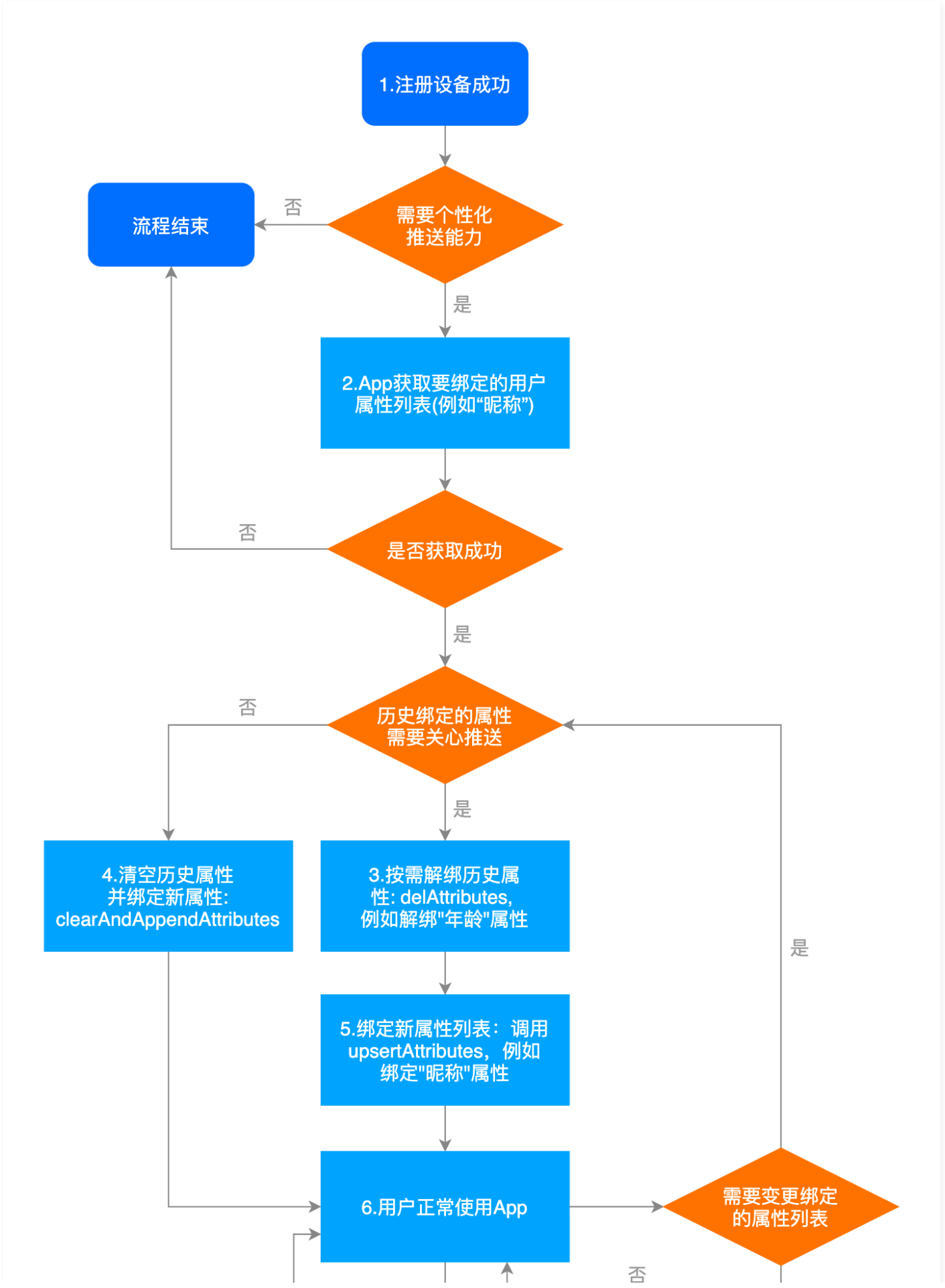
下图为标签相关流程，具体接口方法请查看 标签管理 。

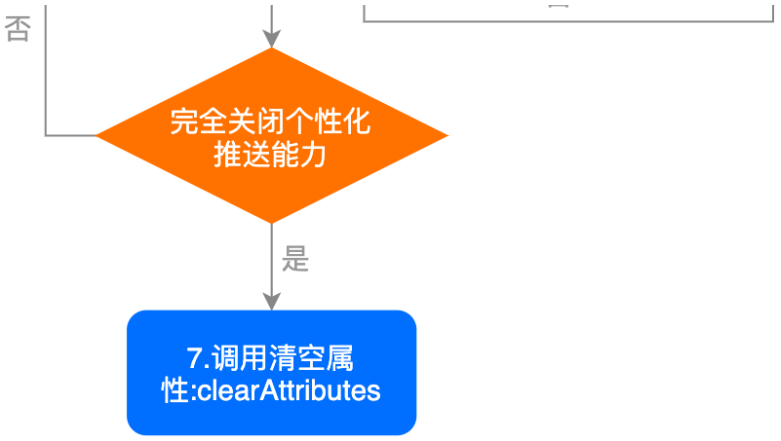

1.注册设备成功
需要标签
推送能力
否
流程结束
是
2.获取要绑定的标签列表
是否获取成功
否
是
历史绑定的标签
需要关心推送
否
是
4.清空历史标签
并绑定新标签：
clearAndAppendTags
3.按需解绑历史
标签: delTags
5.绑定新标签列表：
调用appendTags
6.用户正常使用App
需要变更绑定
的标签列表
是



用户属性相关流程

下图为用户属性相关流程，具体接口方法请查看 用户属性管理 。


1.注册设备成功
需要个性化推送能力
否
流程结束
是
2.App获取要绑定的用户属性列表(例如“昵称”)
是否获取成功
否
是
历史绑定的属性需要关心推送
否
是
4.清空历史属性并绑定新属性: clearAndAppendAttributes
3.按需解绑历史属性: delAttributes, 例如解绑"年龄"属性
5.绑定新属性列表：调用upsertAttributes，例如绑定"昵称"属性
6.用户正常使用App
需要变更绑定的属性列表
是
否



备注

支持的用户属性需要到 Web 管理台或者 RestAPI 先进行配置

SDK 合规使用指南

最近更新时间：2024-06-26 09:51:51

> **注意：**
> 根据《个人信息保护法》、《数据安全法》、《网络安全法》等法律法规和监管部门规章要求，App 开发运营者（以下简称为“开发者”）在提供网络产品服务时应尊重和保护最终用户的个人信息，不得违法违规收集使用个人信息，保证和承诺就个人信息处理行为获得最终用户的授权同意，遵循最小必要原则，且应当采取有效的技术措施和组织措施确保个人信息安全。为帮助开发者在使用移动推送 SDK 的过程中更好地落实用户个人信息保护相关要求，避免出现侵害最终用户个人信息权益的情形，特制定本合规使用说明，开发者使用 移动推送 SDK 时必须在《隐私政策》中告知终端用户 SDK 使用用途，并且在终端用户未同意《隐私政策》前不得初始化任何 SDK。请您务必按照以下步骤做好合规自查，避免被监管部门通报或下架您的应用。

1. 升级至最新版本移动推送 SDK

请务必确保您已经将移动推送 SDK 升级至满足监管新规的最新版本。请前往 Harmony SDK 发布动态 查看和下载最新版本 SDK。

2. 隐私政策中添加移动推送相关说明

请您确保您开发或运营的应用有符合监管要求的《隐私政策》文本。同时请您务必明确告知终端用户您的应用使用了移动推送服务。建议您在《隐私政策》对应的章节、列表中添加关于移动推送的说明，推荐条款如下：

SDK名称：移动推送 SDK
第三方名称：深圳市腾讯计算机系统有限公司
SDK 用途：在移动终端设备进行消息推送
收集个人信息类型：
(1)、 设备信息(手机型号，系统类型、系统版本等)用于标签化推送以及识别是否是真机、网络信息(网络类型)支持根据不同网络类型进行不同类型推送、应用数据（AAID（应用匿名标识符）推送流程中产生的送达、点击、曝光等数据）用于推送业务数据统计。
(2)、（可选信息，依赖开发者设置）账号绑定信息(根据您所选用的不同推送渠道，QQ号、微信Union ID、 手机号、邮箱等)用于根据账号信息进行推送
数据处理方式：通过去标识化、加密传输及其他安全方式
官网链接：https://cloud.tencent.com/product/tpns
隐私政策链接：https://privacy.qq.com/document/preview/8565a4a2d26e480187ed86b0cc81d727

3.《隐私政策》弹出条件

您需要确保 App 有《隐私政策》，且在用户首次启动 App 时弹出《隐私政策》取得用户同意。

4. 隐私接口确认与移动推送 SDK 初始化及业务功能调用时机

要求内容：《SDK 合规使用说明》应详细说明 SDK 初始化及各项业务功能接口合规调用时机。App 应当在 App 登录注册页面及 App 首次运行时，通过弹窗、文本链接及附件等简洁明显且易于访问的方式，向最终用户告知涵盖个人信息处理主体、处理目的、处理方式、处理类型、保存期限等内容的个人信息处理规则，并且获得最终用户授权同意后才能处置最终用户数据。
接入说明：请务必确保终端用户有效同意您 App 中的隐私政策后，再进行移动推送 SDK 初始化。用户同意隐私政策之前，避免动态申请涉及用户个人信息的敏感设备权限；用户同意隐私政策前，您应避免私自采集和上报个人信息。当您的 App 未向用户提供服务时，例如 App 在后台运行时，请勿请求移动推送 SDK 的相关服务接口。
强烈建议您参见 Harmony SDK 发布动态 升级使用最新版本 SDK。
同时，SDK 提供以下接口配置，建议用户按实际需求配置使用。

3.1 初始化相关说明

初始化时机配置

移动推送 SDK 在后续 App 启动时移动推送 Harmony SDK 提供以下接口初始化移动推送 SDK，在终端用户有效同意您 App 中的隐私政策调用如下接口进行初始化，详细参考初始化接口：初始化接口 。

> **注意：**
> App 只有在完成移动推送的初始化才可以使用移动推送 SDK 提供的 Push 服务， 移动推送 SDK 初始化需要在自定义的 AbilityStage 的 onCreate 方法中初始化！

示例代码

```
//1. 自定义AbilityState
export default class MyAbilityStage extends AbilityStage {
  onCreate(): void {
    try {
      let tpnsClient = TPush.createTPushClient(context, {
        accessId: 150000xxxx, //您应用对应移动推送控制台生成的accessId
        accessKey: "您的accessKey",//您应用对应移动推送控制台生成的accessKey
        host: "tpns.tencent.com", //接入点：默认广州集群
        debugMode: true //上线后记得设置为false
      });
      this.tpnsClient.setMessageReceiver(new MessageReiver());
    } catch (err) {
      //TODO:: TPush.createTPushClient 传递的必选参数为空或者非法会抛出异常此处要注意处理
    }
  }

  onAcceptWant(want: Want): string {
    // 仅specified模式下触发
    return 'MyAbilityStage';
  }

  onNewProcessRequest(want: Want): string{
    return "";
  }


  onMemoryLevel(level: AbilityConstant.MemoryLevel): void{
  }
}

//2. 在module.json5中配置MyAbilityStage
{
"module": {
  "name": "entry",
  "type": "entry",
  "srcEntry": "./ets/abilitystage/MyAbilitystage.ets", //此处配置MyAbilitystage
  "description": "$string:module_desc",
  "mainElement": "EntryAbility",
  }
}
```

3.2 业务接口调用说明

请务必确保终端用户有效同意《隐私政策》后再调用移动推送 SDK 业务功能。SDK 功能业务接口链接：注册接口 。

注册业务代码示例

```
/**
 * 在用户隐私政策协议同意后，再做调用移动推送TPNS SDK接口
 * 建议在
 */
function callTPNSPushSDKRegister() {
        /*
         * 当用户同意隐私政策协议时，再调用移动推送 SDK业务功能；
         * 当用户未同意隐私政策协议时，需等用户同意后，再通过callTPNSPushSDKRegister(...)方法进行注册TPNS
         * 应用只有在完成移动推送TPNS的注册后才可以使用TPNS提供Push服务，在这之前请确保有初始化配置AccessId和AccessKey
         */
        let agreed = true;
        if (agreed) {
            //建议在线程中执行调用
```

```
            this.tpnsClient.registerPush().then((data) => {
                console.error("TPush-Demo", "registerPush result:" + data);
            }).catch((err: BusinessError<string>) => {
                console.error("TPush-Demo", "registerPush fail err:" + JSON.stringify(err));
            });
        }
}
```

4. SDK可选信息配置开关

要求内容：《SDK 合规使用说明》应详细说明 SDK 各项可选个人信息使用目的、场景及对应关闭的配置方式、示例。
接入说明：移动推送 SDK 向您提供了可选个人信息及权限的控制开关，您可以根据 App 所需的 SDK 功能服务自行配置打开或关闭隐私信息请求开关，对于移动推送 SDK 可选收集的个人信息的控制，开发者可以参考如下配置指引进行配置操作。因相关信息的不收集将会对其对应的功能造成影响，请开发者结合业务实际需要进行合理配置。

4.1. 配置可选个人信息

基于账号推送功能，第三方开发者需要通过预埋账号类型绑定共享给移动推送 SDK，**根据第三方开发者选择的推送通道的不同所共享给移动推送 SDK 的信息也会有所不同。如果第三方开发者未使用账号推送功能则不需要共享此类信息**。
如需清除已收集账号数据，可以参考 SDK 账号接口进行清除，SDK 业务功能接口链接：清除账号接口 。

可选个人信息类型及字段		使用目的	使用场景
设备信息（可选）	OAID（仅鸿蒙与 Android）	用 OAID 作为移动推送的账号，账号推送时可以使用 OAID 进行账号推送。	根据 OAID 进行账号推送时使用
	微信 UnionID	使用微信 UnionID 作为移动推送的账号，账号推送时可以使用微信 UnionID 进行账号推送。	根据 UnionID 进行账号推送时使用
	QQ openID	使用 QQ openID 作为移动推送的账号，账号推送时可以使用 QQ openID 进行账号推送。	根据 QQ openID 进行账号推送时使用
账号信息（可选）	邮箱账号、新浪微博账号、支付宝账号、淘宝账号、豆瓣账号、FaceBook 账号、Twitter 账号、Google 账号、百度账号、京东账号、linkedin 账号	使用设置的账号类型和账号信息作为移动推送的账号，账号推送时可以使用该账号类型和账号信息进行账号推送。	根据设置账号类型进行账号推送
用户基本信息（可选）	***手机号***	使用设置手机号作为推送的账号，账号推送时可以使用该手机号进行推送	根据设置手机号进行账号推送

4.2. SDK申请系统权限说明

要求内容：《SDK 合规使用说明》应详细说明SDK所需的系统权限与各业务功能间的关系，并说明权限申请时机。
接入说明：对于移动推送 SDK 申请的系统权限，您可以参考相关如下表格的内容，详细了解相关权限与各业务功能的关系及其申请时机，因相关权限的不申请将会对其对应的功能造成影响，您可以结合业务实际需要进行合理配置。配置文档链接：Harmony接入指南 - SDK集成 。

操作系统	权限名称	使用目的	功能场景（申请时机）	是否可选
Harmony	ohos.permission.INTERNET	进行与推送服务进行网络访问，进行业务数据上报	使用推送服务 SDK 时	必选
Harmony	ohos.permission.GET_BUNDLE_INFO	获取当前使用移动推送SDK的应用信息进行应用信息bundle，版本号等信息的校验	使用推送服务 SDK 时	必选
Harmony	ohos.permission.GET_WIFI_INFO	获取网络类型，用于SDK网络访问	使用推送服务 SDK 时	必选

Harmony	ohos.permission.GET_NETWORK_INFO	获取网络信息(网络是否连接， 网络变化)	使用推送服务 SDK 时	必选

5. SDK扩展业务功能的配置说明

要求内容：《SDK 合规使用指南》应详细说明SDK各项扩展业务功能介绍及对应关闭的配置方式、示例。
接入说明：移动推送鸿蒙 SDK 无扩展业务功能。开发者需遵守相关法律法规的要求，在 App 内为最终用户提供退出个性化推送的功能，保证在最终用户点击退出功能后调用移动推送 SDK 的能力接口。

6. SDK 可按照不同频次、精度收集个人信息的配置说明

要求内容：如果 SDK 可按照不同频次、精度收集个人信息的，《SDK 合规使用说明》应说明不同频次、精度的使用目的、场景及对应选择的配置方式、示例。
接入说明：收集频次方面，移动推送 SDK 的数据采集仅在 App 调用/最终用户触发相关功能时触发，不涉及定时逻辑等频次控制选项。收集精度方面不涉及精度相关控制，相关业务接口务必确保终端用户有效同意《隐私政策》后调用，可参考 SDK 接口文档链接：Harmony SDK接口文档 。

7. 最终用户同意方式的示例

要求内容：《SDK 合规使用说明》应详细说明 App 获取最终用户授权同意的建议方式，其中需要取得最终用户单独同意的，应显著提示并给出示例。
接入说明：App 首次运行时应当有隐私弹窗，隐私弹窗中应公示简版隐私政策内容并附完整版隐私政策链接，并明确提示最终用户阅读并选择是否同意隐私政策；隐私弹窗应提供同意按钮和拒绝同意的按钮，并由最终用户主动选择。
披露示例：

8. 撤回同意机制说明

请您告知终端用户其享有选择关闭相应权限或行使退出（opt-out）权利，一旦终端用户行使前述权利，其个人信息将不再会被处理。我们建议您在终端用户撤销同意处理其个人信息的授权时，参见腾讯推送服务 移动推送Harmony接入指南 - 接口文档 “反注册接口说明”部分进行处理，以便用户更便捷行使退出的选择权。

9. 隐私保护机制

如果您对 SDK 权限有任何疑问、意见和建议，或者需要腾讯协助关闭某项权限采集能力，可通过以下联系方式与我们联系：

- 电子邮件：tpns_team@tencent.com
- 联系地址：深圳市南山区粤海街道麻岭社区科技中一路腾讯大厦

您还可以随时通过访问移动推送官网在线客服系统与我们联系，我们将及时为您提供咨询和服务，确保各项隐私机制的落实和执行。

SDK 集成

最近更新时间：2024-12-11 16:29:32

简介

本文档提供关于 SDK 接入以及开启推送服务。

> **注意**
> 为了避免您的 App 被监管部门通报或下架，请您在接入 SDK 之前务必按照 Harmony 合规指南 在《隐私政策》中增加移动推送相关说明，并且在用户同意《隐私政策》后再调用移动推送 SDK 接口。

SDK 集成

操作步骤

> **注意**
> 在配置 SDK 前，确保已创建 Android 平台的应用。

1. 登录 移动推送控制台，在**产品管理>配置管理**页面获取应用的 AccessID、AccessKey。
2. 在 SDK 下载 页面，获取当前最新版本。
3. 在 App 模块下的 oh-package.json5 文件中添加移动推送鸿蒙 SDK 依赖。

```
{
  "dependencies": {
    "@ohos/tpns":"file:./libs/tpns.har"
  }
}
```

鸿蒙离线通道配置

想要推送功能，需要配置鸿蒙平台信息。主要步骤为：

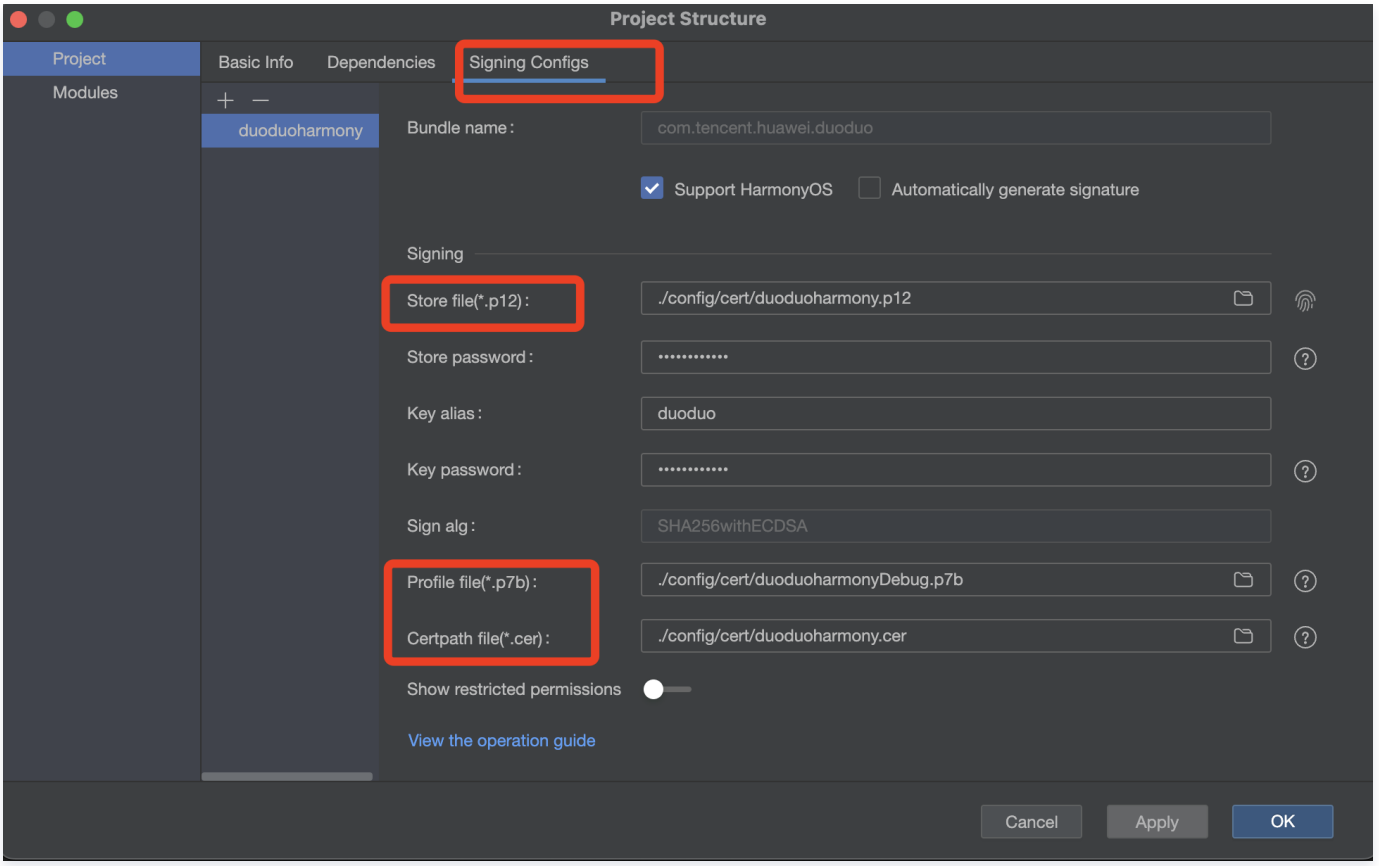
1. 在华为开放平台 开通推送服务。
2. 在华为开放平台 获取到 client_id。
3. 本地工程配置 client_id。

   App 模块的 module.json5文件中配置鸿蒙离线推送的 client_id， 获取 client_id 参考： 鸿蒙参数申请指南 。

```
  {
  "module": {
    "metadata": [
      {// 配置为华为鸿蒙离线推送Client ID
        "name": "client_id",
        "value": "您应用的华为鸿蒙离线推送ClientID" //直接替换双引号中的内容
      },
    ]
  }
}
```

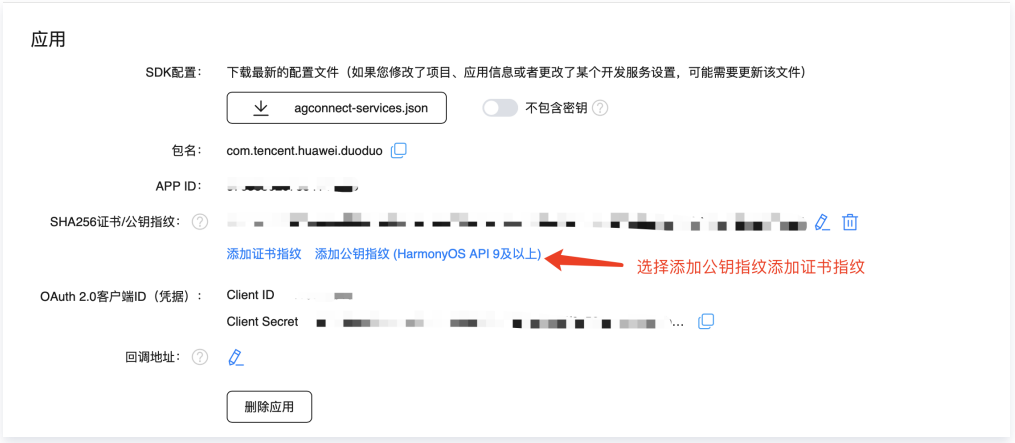
4. 配置签署

   配置签署可以参考鸿蒙文档： 为应用/服务进行签名 。



5. 设置公钥指纹

参考 调试应用（HarmonyOS）。创建好证书后，进入 华为开放平台，在鸿蒙推送应用配置下设置公钥指纹。



调试及设备注册

移动推送 SDK 初始化

创建自定义 AbilityStage，并在应用模块的 module.json5 中配置自定义的 AbilityStage，如果您已有自定义 AbilityStage 可以使用已有的自定义 AbilityStage，
在自定义 AbilityStage 的 onCreate 方法中初始化移动推送 SDK。

```
//1. 自定义MyAbilityStage
export default class MyAbilityStage extends AbilityStage {
   process:string = "1233";
  onCreate(): void {
    console.debug("TPush-Demo","MyAbilityStage onCreate");
    // 应用的HAP在首次加载的时，为该Module初始化操作
    this.init(this.context.getApplicationContext());
```

```
  }

  onAcceptWant(want: Want): string {
    // 仅specified模式下触发
    return 'MyAbilityStage';
  }

  onNewProcessRequest(want: Want): string{
    return "";
  }


  onMemoryLevel(level: AbilityConstant.MemoryLevel): void{
  }

   init(context: Context) {
      try {
        const tpnsClient = TPush.createTPushClient(context, {
                accessId: 150000xxxx, //您的accessId
                accessKey: "xxxxxx", //您的accessKey
                host: "tpns.tencent.com", //创建产品时选择的接入点
                debugMode: true //上线后需要设置为false
                });
  } catch (err) {
    //TODO:: TPush.createTPushClient 传递的必选参数为空或者非法会抛出异常此处要注意处理
    console.error("TPush-Demo", "init fail errCode:" + err.code + " data:" + err.data);
  }
}
}

//2. 在module.json5文件中配置自定义AbilityStage
{
  "module": {
    ......
    "srcEntry": "./ets/MyAbilitystage.ets",
    ......

  }
}
```

> **注意：**
> debugMode 上线时请设置为 false。

Token 注册

初始化成功移动推送 SDK 后，在需要推送服务的地方调用推送服务注册接口：
调用时机： 请务必确保终端用户有效同意《隐私政策》后再调用移动推送 SDK 业务功能接口。 移动推送相关功能接口调用时机您可在确保合规的前提下视您业务需要选择调用。

```
this.tpnsClient.registerPush().then((data) => {
  console.error("TPush-Demo", "registerPush result:" + data);
}).catch((err: BusinessError<string>) => {
  console.error("TPush-Demo", "registerPush fail err:" + JSON.stringify(err));
})
```

过滤 "TPush" 注册成功的日志如下

```
TPush-Demo registerPush success token:011aa78bed7bbc21c648f77b7c612dfd7032
```

配置通知跳转页

通知支持跳转到指定页面。

主要步骤为：要配置 uris，目前主要通过 uri 来匹配页面，推送时 `url=${您应用的scheme}://${您应用的host}/${您应用的path}`

`actions` 一定要配置，没有可以配置空字符串。

```
{
  "name": "EntryAbility2",
  "srcEntry": "./ets/entryability/EntryAbility.ets",
  "exported": false,
  "skills": [ //uris 和actions单独配置skills
    {
      "actions": [ //使用uri时 actions必须要配置,否则点击通知打开uri对应ability失败
        "",
      ],
      "uris": [
        {
          "scheme": "您应用的scheme",
          "host": "您应用的host",
          "path": "您应用的path"
        }
      ]
    },
    {
      "actions": [ //actions 一定要配置，没有可以配置为空字符串
        "您应用的action",
      ],}
  ]
}
//示例 下面配置，推送时 uri的值填为: tpns://com.tpns.push/notify_detail
{
  "name": "EntryAbility2",
  "srcEntry": "./ets/entryability/EntryAbility.ets",
  "exported": false,
  "skills": [
    {
      "actions": [ //使用uri时 actions必须要配置,否则点击通知打开uri对应ability失败
        "",
      ],
      "uris": [
        {
          "scheme": "tpns", //scheme可以替换成业务自己的
          "host": "com.tpns.push", //host可以替换成业务自己的
          "path": "notify_detail" //path可以替换成业务自己的
        }
      ]
    },
    {
      "actions": [ //actions
        "actions.tpns.push", //actions可以替换成业务自己的
      ],
    }
  ]
}
```

隐私协议声明建议

您可在申请 App 权限使用时，使用以下内容声明授权的用途：

我们使用 腾讯云移动推送 TPNS 用于实现产品信息的推送，在您授权我们“访问网络连接”和“访问网络状态”权限后，表示您同意 腾讯 SDK 隐私协议 。您可以通过关闭终端设备中的通知选项来拒绝接受此 SDK 推送服务。

其中上述声明授权的两个链接如下：

- 腾讯云移动推送
- 腾讯 SDK 隐私协议

接口文档

最近更新时间：2024-12-18 18:34:03

说明

本文档中提供移动推送相关的基础能力，包含注册、账号、标签、属性，此文档适用于 SDK 1.0.0 beta 或更高版本。
TPNS主要接口类如下：

类名	说明
TPush	创建移动推送服务推送对象
TPNS.ConfigOptions	移动推送服务相关配置
TPNS.TPushClient	移动推送服务推送接口类

引用SDK

```
import { TPNS, TPush } from '@ohos/tpns';
```

初始化

TPush.createTPushClient

接口说明

App 只有在完成移动推送的初始化才可以使用移动推送 SDK 提供的 Push 服务，在这里，请确保配置信息中设置的 AccessId 和 AccessKey、接入点（广州集群可选）正确。初始化成功会返回 TPush 服务推送接口对象，当参数传入错误时会抛出异常，需要做好异常处理。

```
TPush.createTPushClient(applicationContext:Context,configOptions:TPNS.ConfigOptions):
TPNS.TPushClient|never
```

示例代码

```
try {
  let tpnsClient = TPush.createTPushClient(context, {
    accessId: 150000xxxx, //您应用对应移动推送控制台生成的accessId
    accessKey: "您的accessKey",//您应用对应移动推送控制台生成的accessKey
    host: "tpns.tencent.com", //接入点：默认广州集群
    debugMode: true //上线后记得设置为false
  });
  this.tpnsClient.setMessageReceiver(new MessageReiver());
} catch (err) {
  //TODO:: TPush.createTPushClient 传递的必选参数为空或者非法会抛出异常此处要注意处理
}
```

参数说明

参数名	类型	必填	说明
applicationContext	Context	是	应用的applicationContext
configOptions	TPNS.ConfigOptions	是	推送sdk相关的配置参数

返回值

类型	说明
TPNS.TPushClient	返回移动推送SDK 推送服务操作接口

设置接收消息回调

初始化完移动推送服务后，需要立即设置接收消息回调方法，避免遗漏下发的推送消息。

接口说明

```
setMessageReceiver(messageReciver:MessageReceiver);
```

示例代码

```
class MessageReiver implements TPNS.MessageReceiver {
  public onNotificationDeleted(err:Error,notificationMessage: TPNS.NotificationMessage): void {
    if(err){
      console.error(TAG, "onNotificationDeleted fail, err:" + JSON.stringify(err as BusinessError<string>)
+ " , notificationMessage:"+JSON.stringify(notificationMessage));
    }else {
      console.debug(TAG, "onNotificationDeleted notificationMessage:" +
JSON.stringify(notificationMessage));
    }
  }

  public onNotificationShow(err:Error,notificationMessage: TPNS.NotificationMessage): void {
    if(err){
      console.error("TPush-Demo", "onNotificationShowed fail, err:" + JSON.stringify(err as
BusinessError<string>) + " , notificationMessage:"+JSON.stringify(notificationMessage));
    }else {
      console.debug("TPush-Demo", "onNotificationShowed notificationMessage:" +
JSON.stringify(notificationMessage));
    }
  }

  public onTextMessage(err:Error,pushMessage: TPNS.PushMessage): void {
    if(err) {
      console.error("TPush-Demo", "onTextMessage fail:" + JSON.stringify(err as BusinessError<string>));
    }else{
      console.debug("TPush-Demo", "onTextMessage pushMessage:" + JSON.stringify(pushMessage));
    }
  }
}

//初始化移动推送服务后，设置接收推送消息回调
this.tpnsClient.setMessageReceiver(new MessageReiver());
```

参数说明

参数名	类型	必填	说明
messageReceiver	TPNS.MessageReceiver	是	开发者自行实现的消息回调接口类的对象

注册与反注册

- App 只有在完成移动推送的初始化与注册后才可以移动推送 SDK 提供 Push 服务。
- 注册成功后，会返回设备 Token，Token 用于标识设备唯一性，同时也是移动推送维持与后台连接的唯一身份标识。

注册接口通常提供使用 Promise 异步回调和使用 callback 异步回调的接口，请根据业务需要决定选择接口。

设备注册

以下为设备注册相关接口方法，若需了解调用时机及调用原理，可查看 设备注册流程 。

普通注册只注册当前设备，后台能够针对不同的设备 Token 发送推送消息，以下有4个版本的 API 接口方法：

接口说明

```
registerPush():Promise<string> //使用Promise异步回调
```

示例代码

```
this.tpnsClient.registerPush().then((data) => {
  console.error("TPush-Demo", "registerPush result:" + data);
}).catch((err: BusinessError<string>) => {
  console.error("TPush-Demo", "registerPush fail err:" + JSON.stringify(err));
})
```

返回值

类型	说明
Promise<string>	注册返回的移动推送 token，发生错误时会抛出异常

接口说明

```
registerPush(callback: Callback<string>)//使用callback异步回调
```

示例代码

```
this.tpnsClient.registerPush((err,result)=>{
  if(err){
    console.error("TPush-Demo", "registerPush fail err:" + JSON.stringify(err));
  }else{
    console.error("TPush-Demo", "registerPush result:" + result);
  }
});
```

参数说明

参数名	类型	必填	说明
callback	Callback<string>	是	回调函数，不能为空, err 不为空则表示失败

接口说明

```
registerPushWithAccount(account:TPNS.Account):Promise<string> //使用Promise异步回调
```

示例代码

```
let account: TPNS.Account = {
  account:"testaccount",
  accountType:0
};

this.tpnsClient.registerPushWithAccount(account).then((data) => {
   console.error("TPush-Demo", "registerPushWithAccount result:" + data);
 }).catch((err: BusinessError<string>) => {
```

```
    console.error("TPush-Demo", "registerPushWithAccount fail err:" + JSON.stringify(err));
  })
```

参数说明

参数名	类型	必填	说明
account	TPNS.Account	是	不能为空与 undefined，注册时携带的账号信息，绑定成功后可以通过该账号下发推送

返回值

类型	说明
Promise<string>	注册返回的移动推送 token，发生错误时会抛出异常

接口说明

```
registerPushWithAccount(account:TPNS.Account, callback: Callback<string>) //使用callback异步回调
```

示例代码

```
let account: TPNS.Account = {
  account:"testaccount",
  accountType:0
};
this.tpnsClient.registerPushWithAccount(account,(err,result)=>{
  if(err){
    console.error("TPush-Demo", "registerPushWithAccount fail err:" + JSON.stringify(err));
  }else{
    console.error("TPush-Demo", "registerPushWithAccount result:" + result);
  }
});
```

参数说明

参数名	类型	必填	说明
account	TPNS.Account	是	不能为空与undefined，注册时携带的账号信息，绑定成功后可以通过该账号下发推送
callback	Callback<string>	是	回调函数，不能为空, err 不为空则表示失败

反注册

以下为反注册接口方法，若需了解调用时机及调用原理，可查看 设备反注册流程 。

> **注意：**
> 调用反注册接口后，需要重新调用注册接口才可接收到推送。

当用户已退出或 App 被关闭，不再需要接收推送时，可以取消注册 App，即反注册（一旦设备反注册，直到这个设备重新注册成功期间内，下发的消息该设备都无法收到）。

接口说明

```
unRegisterPush():Promise<void>
```

示例代码

```
this.tpnsClient.unRegisterPush().then(() => {
  console.error("TPush-Demo", "Promise unregister success:");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "unregister fail err:" + JSON.stringify(err));
})
```

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
unRegisterPush(callback: Callback<void>)//使用callback异步回调
```

示例代码

```
this.tpnsClient.unRegisterPush((err)=>{
  if(err){
    console.error("TPush-Demo", "unregister faile:" + JSON.stringify(err));
  }else{
    console.debug("TPush-Demo", "unregister success");
  }
});
```

参数说明

参数名	类型	必填	说明
callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

推送通知（展现在通知栏）

指的是在设备通知栏展示的内容，由移动推送 SDK 完成所有的操作，App 可以监听通知被打开的行为，即在前台下发的通知，无需 App 做任何处理，默认会展示在通知栏。

> **说明：**
> - 成功注册移动推送服务后，通常不需要任何设置便可下发通知。
> - 通常来说，常规的通知，能够满足大部分业务需求，如果需要更灵活的方式，请考虑使用透传消息。

获取通知

接口说明

移动推送 SDK 提供回调接口供开发者获取抵达的通知内容，可以通过 setMessageReceiver 接口设置接收推送消息回调，设置推送消息中对象的 onNotificationShowed(err:Error,notificationMessage: TPNS.NotificationMessage): void 方法实现，notificationMessage 对象提供读取通知内容的接口。

> **注意：**
> 因鸿蒙厂商通道未提供通知抵达回调方法，且当 App 进程未运行时，厂商通道的抵达回调方法无法触发。因此 SDK 内提供的回调接口 onNotificationShowed 仅支持移动推送自建通道下发通知抵达的监听，不支持鸿蒙通道消息抵达的监听。

```
public onNotificationShow(err:Error,notificationMessage: TPNS.NotificationMessage): void
```

示例代码

```
public onNotificationShow(err:Error,notificationMessage: TPNS.NotificationMessage): void {
  if(err){
    console.error("TPush-Demo", "onNotificationShow fail, err:" + JSON.stringify(err as
BusinessError<string>) + " , notificationMessage:"+JSON.stringify(notificationMessage));
  }else {
    console.debug("TPush-Demo", "onNotificationShow notificationMessage:" +
JSON.stringify(notificationMessage));
  }
}
```

参数说明

参数名	类型	必填	说明
err	BusinessError	是	当回调方法被调用时，err 不为 null 或者 undefined，则代表通知展示失败
notificationMessage	TPNS.NotificationMessage	是	当回调方法被调用时，err 为 null 或者 undefined，则代表通知展示成功，此时可以获取通知相关信息

获取通知点击结果

通知点击

使用移动推送 SDK 默认已经统计通知/消息的抵达量、通知的点击。点击通知会打开对应的 UIAbility，在 UIAbility 的 onCreate(want:Want, launchParam: AbilityConstant.LaunchParam) 或 onNewWant(want: Want, launchParam: AbilityConstant.LaunchParam): void 中调用移动推送通过移动推送 SDK 提供parseNotificationMessage(want: Want): TPNS.NotificationMessage ;方法来判断是否是移动推送的通知点击。

接口说明

```
parseNotificationMessage(want: Want):TPNS.NotificationMessage;
```

示例代码

```
 onCreate(want:Want, launchParam: AbilityConstant.LaunchParam) {
  super.onCreate(want,launchParam);
  const notificationMessage:TPNS.NotificationMessage = this.tpnsClient.parseNotificationMessage(want);
  if(notificationMessage){
    //移动推送通知点击
    console.debug("TPush-Demo", "onNotificationClick notificationMessage:" +
JSON.stringify(notificationMessage));
  }else{
    //非移动推送通知点击，进行其他逻辑处理
  }
}
onNewWant(want: Want, launchParam: AbilityConstant.LaunchParam): void {
  const notificationMessage:TPNS.NotificationMessage = this.tpnsClient.parseNotificationMessage(want);
  if(notificationMessage){
    //移动推送通知点击
    console.debug("TPush-Demo", "onNotificationClick notificationMessage:" +
JSON.stringify(notificationMessage));
  }else{
    //非移动推送通知点击，进行其他逻辑处理
```

```
  }
}
```

参数说明

参数名	类型	必填	说明
want	Want	是	不能为空与 undefined，UIAbility 的 onCreate 或onNewWant 方法中的 want 参数，用于移动推送SDK 判断是否是移动推送点击移动推送通知触发的。

返回值

类型	说明
TPNS.NotificationMessage	判断 Ability 打开时是不是点击通知触发，是则返回不为空的通知消息对象

通知移除监听

接口说明

移动推送 SDK 提供回调接口供开发者获取移除的通知动作，可以通过setMessageReceiver接口设置推送消息接收器，设置推送消息接收器中对象的onNotificationDeleted(err:Error,notificationMessage: TPNS.NotificationMessage): void 方法实现.

> **注意：**
> 因鸿蒙厂商通道未提供通知抵达移除回调方法，厂商通道的通知移除方法无法触发，并且应用进程被杀掉后通过移动推送通道或者本地通知触发展示的通知也收不到此回调。因此 SDK 内提供的移除回调接口 onNotificationDeleted 仅支持移动推送自建通道下发通知以及本地通知触发的通知。

```
public onNotificationDeleted(err:Error,notificationMessage: TPNS.NotificationMessage): void
```

示例代码

```
public onNotificationDeleted(err:Error,notificationMessage: TPNS.NotificationMessage): void {
  if(err){
    console.error(TAG, "onNotificationDeleted fail, err:" + JSON.stringify(err as BusinessError<string>) +
" , notificationMessage:"+JSON.stringify(notificationMessage));
  }else {
    console.debug(TAG, "onNotificationDeleted notificationMessage:" +
JSON.stringify(notificationMessage));
  }
}
```

参数说明

参数名	类型	必填	说明
err	BusinessError	是	当回调方法被调用时，err 不为 null 或者 undefined，则代表通知展示失败
notificationMessage	TPNS.NotificationMessage	是	当回调方法被调用时，err 为 null 或者 undefined，则代表通知展示成功，此时可以获取通知相关信息

清除所有通知

接口说明

清除本 App 在通知栏上的所有通知。

```
cancelAllNotification():Promise<void>//使用Promise异步回调
```

示例代码

```
this.tpnsClient.cancelAllNotification().then(()=>{
  console.debug("TPush-Demo", "cancelNotification success");
}).catch((err:BusinessError)=>{
  console.debug("TPush-Demo", "cancelNotification fail err:" + JSON.stringify(err));
})
```

返回值

类型	说明
Promise<void>	失败时抛出异常

接口说明

```
anelAllNotification(callback:Callback<void>);
```

示例代码

```
this.tpnsClient.cancelAllNotification((err)=>{
  if(err){
    console.debug("TPush-Demo", "cancelNotification fail err:" + JSON.stringify(err));
  }else{
    console.debug("TPush-Demo", "cancelNotification success");
  }
})
```

参数说明

参数名	类型	必填	说明
callback	Callback<void>	是	回调函数。不能为空, err 不为空则表示失败

推送透传消息（消息不展示到通知栏）

指的是由移动推送下发给 App 的内容，需要 App 通过 setMessageReceiver 接口设置推送消息接收方法，实现并自主处理所有操作过程，也就是说，下发的消息默认是不会展示在通知栏的，移动推送只负责将消息从移动推送服务器下发到 App 这个过程，不负责消息的处理逻辑，需要 App 自己实现。

- 消息指的是由开发者通过前台或后台脚本下发的文本消息，移动推送只负责将消息传递给 App，App 完全自主负责消息体的处理。
- 消息具有灵活性强和高度定制性的特点，更适合 App 自主处理个性化业务需求，例如下发 App 配置信息、自定义处理消息的存储和展示等。

获取透传消息

开发者在前台下发消息，需要 App 通过 setMessageReceiver 接口设置接收推送消息接口，并实现 onTextMessage 方法接收，成功接收后，再根据特有业务场景进行处理。

接口说明

```
public onTextMessage(err:Error,pushMessage: TPNS.PushMessage): void
```

示例代码

```
public onTextMessage(err:Error,pushMessage: TPNS.PushMessage): void {
  if(err) {
    console.error("TPush-Demo", "onTextMessage fail:" + JSON.stringify(err as BusinessError<string>));
  }else{
    console.debug("TPush-Demo", "onTextMessage pushMessage:" + JSON.stringify(pushMessage));
  }
}
```

参数说明

参数名	类型	必填	说明
err	BusinessError	是	当回调方法被调用时，err 不为 null 或者 undefined，则代表通知展示失败
pushMessage	TPNS.PushMessage	是	当回调方法被调用时，err 不为 null 或者 undefined，则代表透传消息抵达成功，此时可以获取推送相关信息

本地通知

增加本地通知

本地通知由用户自定义设置，保存在本地。当应用打开，移动推送 SDK 服务 会根据网络心跳，判断当前是否有通知（5分钟一次），本地通知需要 移动推送服务 开启才能弹出，可能存在5分钟左右延时。（当设置的时间小于当前设备时间通知弹出）

接口说明

```
addLocalNotification(localMessage:TPNS.LocalMessage):Promise<void>; //使用Promise异步回调
```

示例代码

```
addLocalMessage(inboxNotification:boolean){
    let inlines:string[] = ["第一行","第二行","第三行","第四行"];
    let localMessage:TPNS.LocalMessage;
    if(inboxNotification){
      localMessage =  {
        type:TPNS.LocalMessageType.NOTIFICATION,
        title:"我是本地多行文本通知标题",
        content:"我是本地通知内容",
        action_type:TPNS.LocalMessageActionType.OPEN_APP,
        inlines:inlines
      } as TPNS.LocalMessage;
    }else{
      localMessage = {
         type:TPNS.LocalMessageType.NOTIFICATION,
         title:"我是本地通知标题",
         content:"我是本地通知内容",
         action_type:TPNS.LocalMessageActionType.OPEN_APP,
         action:"action.tpns.push"
      } as TPNS.LocalMessage;
    }
    this.tpnsClient.addLocalNotification(localMessage).then(()=>{
       console.debug("TPush-Demo", "addLocalMessage success");
    }).catch((err:BusinessError)=>{
       console.error("TPush-Demo", "addLocalMessage fail err:" + JSON.stringify(err));
    })
```

```
}
```

参数说明

参数名	类型	必填	说明
localMessage	TPNS.LocalMessage	是	要展示的本地通知对象，不能为空或 undefined

返回值

类型	说明
Promise<void>	失败时抛出异常，参考 err 信息

接口说明

```
addLocalNotification(localMessage:TPNS.LocalMessage, callback:Callback<void>); //使用callback异步回调
```

示例代码

```
addLocalMessage(inboxNotification:boolean){
    let inlines:string[] = ["第一行","第二行","第三行","第四行"];
    let localMessage:TPNS.LocalMessage;
    if(inboxNotification){
      localMessage =  {
        type:TPNS.LocalMessageType.NOTIFICATION,
        title:"我是本地多行文本通知标题",
        content:"我是本地通知内容",
        action_type:TPNS.LocalMessageActionType.OPEN_APP,
        inlines:inlines
      } as TPNS.LocalMessage;
    }else{
      localMessage = {
         type:TPNS.LocalMessageType.NOTIFICATION,
         title:"我是本地通知标题",
         content:"我是本地通知内容",
         action_type:TPNS.LocalMessageActionType.OPEN_APP,
         action:"action.tpns.push"
      } as TPNS.LocalMessage;
    }
    this.tpnsClient.addLocalNotification(localMessage, (err) => {
        if(err){
          console.error("TPush-Demo", "addLocalMessage fail err:" + JSON.stringify(err));
        }else{
          console.debug("TPush-Demo", "addLocalMessage success");
        }
    });
}
```

参数说明

参数名	类型	必填	说明
localMessage	TPNS.LocalMessage	是	要展示的本地通知对象，不能为空与 undefined
callback	Callback<void>	是	回调函数。不能为空, err 不为空则表示失败

清除本地通知

接口说明

清除本 App 已经创建但未弹出的本地通知。

```
clearLocalNotification():void;
```

示例代码

```
this.tpnsClient.clearLocalNotification();
```

账号管理

以下为账号管理相关接口方法，若需了解调用时机及调用原理，可查看 账号相关流程 。

添加账号

接口说明

添加或更新账号。若原来没有该类型账号，则添加；若原来有，则覆盖。可以同时添加多个账号，一个账号对应一个账号类型。

> **说明：**
> - 每个账号最多支持绑定100个 token。
> - 账号可以是邮箱、QQ 号、手机号、用户名等任意类别的业务账号，账号类型取值可参考 账号类型取值表 。
> - 同一个账号绑定多个设备时，后台将默认推送消息到最后绑定的设备，如需推送所有绑定的设备可查看 Rest API 文档中 account_push_type 参数设置。

接口说明

```
upsetAccount(accounts:ArrayList<TPNS.Account>):Promise<void>; //使用Promise异步回调
```

示例代码

```
let accounts: ArrayList<TPNS.Account> = new ArrayList<TPNS.Account>();
const account: TPNS.Account = {
  account:"testaccount",
  accountType:0
};
accounts.add(account);
const account1: TPNS.Account = {
  account:"testaccount2",
  accountType:1
};
accounts.add(account1);
console.debug("TPush-Demo", "accounts:" + JSON.stringify(accounts.convertToArray()));
this.tpnsClient.upsetAccount(accounts).then(()=>{
  console.debug("TPush-Demo", "upsetAccount success");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "upsetAccount fail err:" + JSON.stringify(err));
})
```

参数说明

参数名	类型	必填	说明
accounts	ArrayList<TPNS.Account>	是	要添加的账号列表，不能为空与undefined

说明：
- 每个账号最多支持绑定100个 token。
- 账号可以是邮箱、QQ 号、手机号、用户名等任意类别的业务账号，账号类型取值可参考 账号类型取值表 。
- 同一个账号绑定多个设备时，后台将默认推送消息到最后绑定的设备，如需推送所有绑定的设备可查看 Rest API 文档中 account_push_type 参数设置。

返回值

类型	说明
Promise<void>	失败时会抛出异常

接口说明

```
upsetAccount(accounts:ArrayList<TPNS.Account>, callback:Callback<void>); //使用callback异步回调
```

示例代码

```
let accounts: ArrayList<TPNS.Account> = new ArrayList<TPNS.Account>();
const account: TPNS.Account = {
  account:"testaccount",
  accountType:0
};
accounts.add(account);
const account1: TPNS.Account = {
  account:"testaccount2",
  accountType:1
};
accounts.add(account1);
console.debug("TPush-Demo", "accounts:" + JSON.stringify(accounts.convertToArray()));
this.tpnsClient.upsetAccount(accounts, (err) => {
  if(err){
    console.error("TPush-Demo", "upsetAccount fail err:" + JSON.stringify(err));
  }else{
    console.debug("TPush-Demo", "upsetAccount success");
  }
})
```

参数说明

参数名	类型	必填	说明
accounts	ArrayList<TPNS.Account>	是	要添加的账号列表，不能为空与undefined
callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

说明：
- 每个账号最多支持绑定100个 token。
- 账号可以是邮箱、QQ 号、手机号、用户名等任意类别的业务账号，账号类型取值可参考 账号类型取值表 。
- 同一个账号绑定多个设备时，后台将默认推送消息到最后绑定的设备，如需推送所有绑定的设备可查看 Rest API 文档中 account_push_type 参数设置。

添加手机号

接口说明

添加或更新手机号码。若原来没有绑定手机号码，则绑定；

> **说明：**
> 手机号格式为 `+[国家或地区码][手机号]`，例如+8613711112222（其中前面有一个+号，86为国家或地区码，13711112222为手机号）。若绑定的手机号不带**国家或地区码**，则移动推送下发短信时自动增加+86的前缀；若带上**国家或地区码**，则按照指定的号码绑定。如需删除绑定的手机号，则需调用 `delAccounts` 接口并设置 `accountType` 为 `1002`。

```
upsertPhoneNumber(phoneNumber:string):Promise<void>; //使用Promise异步回调
```

示例代码

```
const phoneNumber = "18682358888";
this.tpnsClient.upsertPhoneNumber(phoneNumber).then(()=>{
    console.debug("TPush-Demo", "upsertPhoneNumber success");
}).catch((err:BusinessError)=>{
    console.error("TPush-Demo", "upsertPhoneNumber fail err:" + JSON.stringify(err));
})
```

参数说明

参数名	类型	必填	说明
phoneNumber	string	是	E.164标准，格式为+[国家或地区码][手机号]，例如+8613711112222。SDK 内部加密传输。

> **说明：**
> 手机号格式为 `+[国家或地区码][手机号]`，例如+8613711112222（其中前面有一个+号，86为国家或地区码，13711112222为手机号）。若绑定的手机号不带**国家或地区码**，则移动推送下发短信时自动增加+86的前缀；若带上**国家或地区码**，则按照指定的号码绑定。如需删除绑定的手机号，则需调用 `delAccounts` 接口并设置 `accountType` 为 `1002`。

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
upsertPhoneNumber(phoneNumber:string, callback:Callback<boolean>); //使用callback异步回调
```

示例代码

```
const phoneNumber = "18682358888";
this.tpnsClient.upsertPhoneNumber(phoneNumber).then(()=>{
  console.debug("TPush-Demo", "upsertPhoneNumber success");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "upsertPhoneNumber fail err:" + JSON.stringify(err));
})
this.tpnsClient.upsertPhoneNumber(phoneNumber, (err,) => {
  if(err){
    console.error("TPush-Demo", "upsertPhoneNumber fail err:" + JSON.stringify(err));
  }else {
```

```
    console.debug("TPush-Demo", "upsertPhoneNumber success");
  }
});
```

参数说明

参数名	类型	必填	说明
phoneNumber	string	是	E.164标准，格式为+[国家或地区码][手机号]，例如+8613711112222。SDK 内部加密传输。
callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

说明：
手机号格式为 `+[国家或地区码][手机号]`，例如+8613711112222（其中前面有一个+号，86为国家或地区码，13711112222为手机号）。若绑定的手机号不带国家或地区码，则移动推送下发短信时自动增加+86的前缀；若带上国家或地区码，则按照指定的号码绑定。如需删除绑定的手机号，则需调用 `delAccounts` 接口并设置 `accountType` 为 `1002`。

账号类型解绑

接口说明

对一个或多个账号类型的账号进行解绑。

```
delAccounts(accountTypeSet:HashSet<number>):Promise<void>; //使用Promise异步回调
```

示例代码

```
let accountTypeHashSet: HashSet<number> = new HashSet<number>();
accountTypeHashSet.add(0);
this.tpnsClient.delAccounts(accountTypeHashSet).then(()=>{
  console.debug("TPush-Demo", "delaccount success");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "delaccount fail,err:" + JSON.stringify(err));
});
```

参数说明

参数名	类型	必填	说明
accountTypeSet	HashSet<number>	是	需解绑账号的账号类型

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
delAccounts(accountTypeSet:HashSet<number>, callback:Callback<void>); //使用callback异步回调
```

示例代码

```
let accountTypeHashSet: HashSet<number> = new HashSet<number>();
accountTypeHashSet.add(0);
```

```
this.tpnsClient.delAccounts(accountTypeHashSet, (err) => {
  if(err){
    console.error("TPush-Demo", "delaccount fail,err:" + JSON.stringify(err));
  }else{
    console.debug("TPush-Demo", "delaccount success");
  }
});
```

参数说明

参数名	类型	必填	说明
accountTypeSet	HashSet<number>	是	需解绑账号的账号类型。
callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

清空所有账号

接口说明

对所有已绑定账号进行解绑。

```
clearAccounts():Promise<void> //使用Promise异步回调
```

示例代码

```
this.tpnsClient.clearAccounts().then(() =>{
  console.debug("TPush-Demo", "clearAccount success");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "delaccount fail err:" + JSON.stringify(err));
})
```

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
clearAccounts(callback:Callback<void>) //使用callback异步回调
```

示例代码

```
this.tpnsClient.clearAccounts((err) => {
  if(err){
    console.debug("TPush-Demo", "delaccount fail err:" + JSON.stringify(err));
  }else {
    console.debug("TPush-Demo", "clearAccount success");
  }
});
```

参数说明

参数名	类型	必填	说明
callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

标签管理

以下为标签管理相关接口方法，若需了解调用时机及调用原理，可查看 标签相关流程 。

预设标签

目前移动推送平台提供的预设标签包括：App 版本，系统版本，省份，活跃信息，系统语言，SDK 版本，国家&地区，手机品牌，手机机型。预设标签会在 SDK 内部自动上报。

覆盖多个标签

接口说明

一次设置多个标签，会覆盖这个设备之前设置的标签。
开发者可以针对不同的用户设置标签，然后根据标签名群发通知。一个应用最多有10000个 tag， 每个 Token 在一个应用下最多100个 tag，如需提高该限制，请联系 在线客服 。每个自定义 tag 可绑定的设备 Token 数量无限制，tag 中不准包含空格。

```
clearAndAppendTags( tags:HashSet<string>):Promise<void>; //使用Promise异步回调
```

示例代码

```
let tags = new HashSet<string>();
tags.add("testTag");
tags.add("testTag2");
this.tpnsClient.clearAndAppendTags(tags).then(()=>{
  console.debug("TPush-Demo", "clearAndAppendTags success");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "clearAndAppendTags fail err:" + JSON.stringify(err));
});
```

参数说明

参数名	类型	必填	说明
tags	HashSet<string>	是	标签名集合，每个标签是一个 String。限制：每个 tag 不能超过50字节（超过会抛弃），不能包含空格（含有空格会删除空格）。最多设置100个 tag，超过部分会抛弃。

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
clearAndAppendTags(tags:HashSet<string>,callback:Callback<void>); //使用callback异步回调
```

示例代码

```
this.tpnsClient.clearAndAppendTags(tags, (err) => {
  if(err){
    console.error("TPush-Demo", "clearAndAppendTags fail err:" + JSON.stringify(err));
  }else {
    console.debug("TPush-Demo", "clearAndAppendTags success");
  }
});
```

参数说明

参数名	类型	必填	说明
tags	HashSet<string>	是	标签名集合，每个标签是一个 String。限制：每个 tag 不能超过50字节（超过会抛弃），不能包含空格（含有空格会删除空格）。最多设置100个 tag，超过部分会抛弃。
callback	Callback<void>	是	回调函数，不能为空

新增多个标签

接口说明

- 如果新覆盖的标签都带有 `:` 号，例如 `test:2, level:2`，则会删除这个设备已绑定的所有 `test:*` 和 `level:*` 标签，再新增 `test:2` 和 `level:2`。
- 如果新增的标签有部分不带 `:` 号，例如 `test:2  level`，则会删除这个设备的全部历史标签，再新增 `test:2` 和 `level` 标签。

> **说明：**
> 新增的 tags 中，`:` 号为后台关键字，请根据具体的业务场景使用。

- 此接口调用的时候需要间隔一段时间（建议大于5s），否则可能造成更新失败。

```
appendTags(tags:HashSet<string>):Promise<void>; //使用Promise异步回调
```

示例代码

```
let tags = new HashSet<string>();
tags.add("testTag");
this.tpnsClient.appendTags(tags).then(()=>{
  console.debug("TPush-Demo", "appendTags success");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "appendTags fail err:" + JSON.stringify(err));
});
```

参数说明

参数名	类型	必填	说明
tags	HashSet<string>	是	标签名集合，每个标签是一个 String。限制：每个 tag 不能超过50字节（超过会抛弃），不能包含空格（含有空格会删除空格）。最多设置100个 tag，超过部分会抛弃。

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
appendTags(tags:HashSet<string>,callback:Callback<void>); //使用callback异步回调
```

示例代码

```
let tags = new HashSet<string>();
tags.add("testTag");
this.tpnsClient.appendTags( tags, (err) => {
  if(err){
```

```
    console.error("TPush-Demo", "appendTags fail,err:" + JSON.stringify(err));
  }else {
    console.debug("TPush-Demo", "appendTags success");
  }
});
```

参数说明

参数名	类型	必填	说明
tags	HashSet<string>	是	标签名集合，每个标签是一个 String。限制：每个 tag 不能超过50字节（超过会抛弃），不能包含空格（含有空格会删除空格）。最多设置100个 tag，超过部分会抛弃。
callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

删除多个标签

接口说明

一次删除多个标签。

```
delTags(tags:HashSet<string>):Promise<void>; //使用Promise异步回调
```

示例代码

```
let delTag = new HashSet<string>();
delTag.add("testTag");
this.tpnsClient.delTags(delTag).then(()=>{
  console.debug("TPush-Demo", "delTags success");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "delTags fail err:" + JSON.stringify(err));
});
```

参数说明

参数名	类型	必填	说明
tags	HashSet<string>	是	标签名集合，每个标签是一个 String。限制：每个 tag 不能超过50字节（超过会抛弃），不能包含空格（含有空格会删除空格）。最多设置100个 tag，超过部分会抛弃。

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
delTags(tags:HashSet<string>, callback:Callback<void>); //使用callback异步回调
```

示例代码

```
let delTag = new HashSet<string>();
delTag.add("testTag");
this.tpnsClient.delTags(delTag, (err) => {
```

```
  if(err){
    console.error("TPush-Demo", "delTags fail err:" + JSON.stringify(err));
  }else {
    console.debug("TPush-Demo", "delTags success");
  }
});
```

参数说明

参数名	类型	必填	说明
tags	HashSet<string>	是	标签名集合，每个标签是一个 String。限制：每个 tag 不能超过50字节（超过会抛弃），不能包含空格（含有空格会删除空格）。最多设置100个 tag，超过部分会抛弃。
callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

清除所有标签

接口说明

清除这个设备的所有标签。

```
clearTags():Promise<void>;//使用Promise异步回调
```

示例代码

```
this.tpnsClient.clearTags().then(()=>{
  console.debug("TPush-Demo", "clearTags success");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "clearTags fail err:" + JSON.stringify(err));
})
```

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
clearTags(callback:Callback<void>); //使用callback异步回调
```

示例代码

```
this.tpnsClient.clearTags((err) => {
  if(err){
    console.error("TPush-Demo", "clearTags fail err:" + JSON.stringify(err));
  }else {
    console.debug("TPush-Demo", "clearTags success");
  }
});
```

参数说明

参数名	类型	必填	说明
callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

查询标签

接口说明

获取这个设备的标签。

```
queryTags(offset:number, limit:number):Promise<string>; //使用Promise异步回调
```

示例代码

```
this.tpnsClient.queryTags( 0, 100).then((result)=>{
  console.debug("TPush-Demo", "queryTags result:" + JSON.stringify(result));
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "queryTags fail err:" + JSON.stringify(err));
})
```

参数说明

参数名	类型	必填	说明
offset	number	是	开始的位置
limit	number	是	获取标签的数量，最多为100个

返回值

类型	说明
Promise<string>	查询结果，返回结果不为空时结果为JSON字符串，为空时为空字符串。

接口说明

```
queryTags(offset:number, limit:number, callback:Callback<string>); //使用callback异步回调
```

示例代码

```
this.tpnsClient.queryTags( 0, 100, (err,result) => {
  if(err){
    console.error("TPush-Demo", "queryTags fail err:" + JSON.stringify(err));
  }else {
    console.debug("TPush-Demo", "queryTags result:" + JSON.stringify(result));
  }
});
```

参数说明

参数名	类型	必填	说明
offset	number	是	开始的位置
limit	number	是	获取标签的数量，最多为100个
callback	Callback<boolean>	是	查询结果，返回结果不为空时结果为JSON字符串，为空时为空字符串, err 不为空则表示失败

用户属性管理

开发者可以针对不同的用户设置属性，然后在管理平台推送的时候进行个性化推送。以下为用户属性相关接口方法，若需了解调用时机及调用原理，可查看 用户属性相关流程 。

新增用户属性

接口说明

添加属性：有则覆盖，无则添加。

```
upsertAttributes(attributes:HashMap<string,string>):Promise<void>; //使用Promise异步回调
```

示例代码

```
let attributes: HashMap<string, string> = new HashMap<string, string>();
attributes.set("nickname", "hello");
this.tpnsClient.upsertAttributes(attributes).then(()=>{
  console.debug("TPush-Demo", "upsertAttributes success");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "queryTags fail err:" + JSON.stringify(err));
});
```

参数说明

参数名	类型	必填	说明
attributes	HashMap<string,string>	是	属性集合，每个属性通过 key-value 标识

注意：

- 属性使用键值对传输，都只接受 string 字符串类型，非空串。
- 属性个数限制50个。
- 属性 key，value 长度都限制50个字符以内。

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
upsertAttributes(attributes:HashMap<string,string>, callback:Callback<void>); //使用callback异步回调
```

示例代码

```
let attributes: HashMap<string, string> = new HashMap<string, string>();
attributes.set("nickname", "hello");
this.tpnsClient.upsertAttributes(attributes, (err) => {
  if(err){
    console.error("TPush-Demo", "queryTags fail err:" + JSON.stringify(err));
  }else {
    console.debug("TPush-Demo", "upsertAttributes success");
  }
});
```

参数说明

参数名	类型	必填	说明
attributes	HashMap<string,string>	是	属性集合，每个属性通过 key-value 标识

callback	Callback<void>	是	回调函数，不能为空,err 不为空则表示失败

注意：
- 属性使用键值对传输，都只接受 string 字符串类型，非空串。
- 属性个数限制50个。
- 属性 key，value 长度都限制50个字符以内。

删除用户属性

接口说明

删除指定的属性

```
delAttributes(attributesKey:HashSet<string>):Promise<void>; //使用Promise异步回调
```

示例代码

```
let attributes = new HashSet<string>();
attributes.add("nickname");
this.tpnsClient.delAttributes(attributes).then(()=>{
  console.debug("TPush-Demo", "delAttributes success");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "delAttributes fail err:" + JSON.stringify(err));
});
```

参数说明

参数名	类型	必填	说明
attributesKey	HashSet<string>	是	属性集合，每个属性通过 key-value 标识

注意：
- 属性使用键值对传输，都只接受 string 字符串类型，非空串。
- 属性个数限制50个。
- 属性 key，value 长度都限制50个字符以内。

返回值

类型	说明
Promise<void>	发生错误时抛出异常

接口说明

```
delAttributes(attributesKey:HashSet<string>, callback:Callback<void>); //使用callback异步回调
```

示例代码

```
let attributes = new HashSet<string>();
attributes.add("nickname");
this.tpnsClient.delAttributes(attributes, (err) => {
  if(err){
    console.error("TPush-Demo", "delAttributes fail err:" + JSON.stringify(err));
  }else{
```

```
    console.debug("TPush-Demo", "delAttributes success");
  }
});
```

参数说明

参数名	类型	必填	说明
attributesKey	HashSet<string>	是	属性集合，每个属性通过 key-value 标识
callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

注意：

- 属性使用键值对传输，都只接受 string 字符串类型，非空串。
- 属性个数限制50个。
- 属性 key，value 长度都限制50个字符以内。

清空已有用户属性

接口说明

删除已设置的所有属性。

```
clearAttributes():Promise<void>;//使用Promise异步回调
```

示例代码

```
this.tpnsClient.clearAttributes().then(()=>{
  console.debug("TPush-Demo", "clearAttributes success");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "clearAttributes fail err:" + JSON.stringify(err));
})
```

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
clearAttributes(callback:Callback<void>); //使用callback异步回调
```

示例代码

```
this.tpnsClient.clearAttributes((err) => {
  if(err){
    console.error("TPush-Demo", "clearAttributes fail err:" + JSON.stringify(err));
  }else{
    console.debug("TPush-Demo", "clearAttributes success");
  }
});
```

参数说明

参数名	类型	必填	说明

callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

更新用户属性

接口说明

设置属性（带回调），会覆盖这个设备之前设置的所有属性（即清理并设置）。

```
clearAndAppendAttributes(attributes:HashMap<string,string>):Promise<void>; //使用Promise异步回调
```

示例代码

```
let attributes: HashMap<string, string> = new HashMap<string, string>();
attributes.set("nickname", "haha");
this.tpnsClient.clearAndAppendAttributes(attributes).then(()=>{
  console.debug("TPush-Demo", "clearAndAppendAttributes success");
}).catch((err:BusinessError)=>{
  console.error("TPush-Demo", "clearAndAppendAttributes fail err:" + JSON.stringify(err));
})
```

参数说明

参数名	类型	必填	说明
attributes	HashMap<string,string>	是	属性集合，每个属性通过 key-value 标识

> **注意：**
> - 属性使用键值对传输，都只接受 string 字符串类型，非空串。
> - 属性个数限制50个。
> - 属性 key，value 长度都限制50个字符以内。

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
clearAndAppendAttributes(attributes:HashMap<string,string>, callback:Callback<void>); //使用callback异步回调
```

示例代码

```
let attributes: HashMap<string, string> = new HashMap<string, string>();
attributes.set("nickname", "haha");
this.tpnsClient.clearAndAppendAttributes(attributes, (err) => {
  if(err){
    console.error("TPush-Demo", "clearAndAppendAttributes fail err:" + JSON.stringify(err));
  }else{
    console.debug("TPush-Demo", "clearAndAppendAttributes success");
  }
});
```

参数说明

参数名	类型	必填	说明

attributes	HashMap<string,string>	是	属性集合，每个属性通过 key-value 标识
callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

注意：

- 属性使用键值对传输，都只接受 string 字符串类型，非空串。
- 属性个数限制50个。
- 属性 key，value 长度都限制50个字符以内。

角标管理

设置角标

通过该接口可以设置应用角标数

```
setBadgeNumber(badgeNumber:number):Promise<void>; //使用Promise异步回调
```

示例代码

```
const bageNumbber = 1;
this.tpnsClient.setBadgeNumber(bageNumbber).then(() => {
  console.debug("TPush-Demo", "setBadge success");
}).catch((err: Error) => {
  console.debug(TAG, "setBadge fail err:" + JSON.stringify(err));
});
```

参数说明

参数名	类型	必填	说明
badgeNumber	number	是	角标数

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
setBadgeNumber(badgeNumber:number, callback:Callback<void>);//使用callback异步回调
```

示例代码

```
const bageNumbber = 1;
this.tpnsClient.setBadgeNumber(bageNumbber,(err)=>{
  if(err){
    console.debug(TAG, "setBadge fail err:" + JSON.stringify(err));
  }else{
    console.debug("TPush-Demo", "setBadge success");
  }
});
```

参数说明

参数名	类型	必填	说明

badgeNumber	number	是	角标数
callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

清除角标

接口说明

清除应用角标。

```
clearBadge():Promise<void>;//使用Promise异步回调
```

示例代码

```
this.tpnsClient.clearBadge().then(() => {
   console.debug("TPush-Demo", "clearBadge success");
}).catch((err: Error) => {
  console.debug(TAG, "clearBadge fail err:" + JSON.stringify(err));
});
```

返回值

类型	说明
Promise<void>	发生错误时会抛出异常

接口说明

```
clearBadge(callback:Callback<void>); //使用callback异步回调
```

示例代码

```
this.tpnsClient.clearBadge((err)=>{
  if(err){
    console.debug(TAG, "clearBadge fail err:" + JSON.stringify(err));
  }else{
    console.debug("TPush-Demo", "clearBadge success");
  }
});
```

参数说明

参数名	类型	必填	说明
callback	Callback<void>	是	回调函数，不能为空, err 不为空则表示失败

接口关联的类

以下为移动推送对外提供接口中关联的类的属性和方法的说明。

TPNS.ConfigOption

参数名	类型	必填	说明
accessKey	string	是	在移动推送TPNS控制台创建应用时生成的AccessKey
debugMode	boolean	否	debug模式，默认关闭

debugLevel	TPNS.LogLevel	否	移动推送SDK日志输出等级，默认等级TPNS.LogLevel.DEBUG
keepAliveInterval	number	否	移动推送SDK长链接心跳间隔，单位秒，默认270秒
host	string	否	移动推送SDK接入点，默认接入点广州集群，非广州集群需要配置
backGroundAutoDisConnectMqtt	boolean	否	应用退后台自动断开mqtt功能，默认断开 **说明：** 该配置为 TPNS Harmony SDK 1.0.5版本开始支持，老版本默认应用切到后台不断开 SDK 内的推送长链接。

TPNS.LogLevel

参数名	说明
INFO	debugMode开启时，logLevel设置为INFO，输出移动推送SDK INFO、WARN及ERROR级别日志
WARN	debugMode开启时，logLevel设置为WARN，输出移动推送SDK WARN及ERROR级别日志
ERROR	debugMode开启时，logLevel设置为ERROR，输出移动推送SDK ERROR级别日志

TPNS.TPushClient

接口描述	说明
registerPush():Promise<string>	移动推送 SDK 注册接口，通过此接口可以获取到移动推送的设备token，用于后台触发推送
registerPush(callback: Callback<string>)	
registerPushWithAccount(account: TPNS.Account):Promise<string>	移动推送 SDK 注册接口，此接口在注册时一起绑定一个账号，此接口可以获取到移动推送的设备token
registerPushWithAccount(account:TPNS.Account, callback: Callback<string>)	
unRegisterPush():Promise<void>	不再需要接收推送时，可以取消注册 ，即反注册（一旦设备反注册，直到这个设备重新注册成功期间内，下发的消息该设备都无法收到）。
unRegisterPush(callback: Callback<void>)	
upsetAccount(accounts:ArrayList<TPNS.Account>):Promise<void>	添加或更新账号。若原来没有该类型账号，则添加；若原来有，则覆盖。可以同时添加多个账号，一个账号对应一个账号类型
upsetAccount(accounts:ArrayList<TPNS.Account>, callback:Callback<void>)	
upsertPhoneNumber(phoneNumber:string):Promise<void>	添加或更新手机号码。若原来没有绑定手机号码，则绑定；若原来有，则覆盖
upsertPhoneNumber(phoneNumber:string, callback:Callback<void>)	
delAccounts(accountTypeSet:HashSet<number>):Promise<void>	对一个或多个账号类型的账号进行解绑。
delAccounts(accountTypeSet:HashSet<number>, callback:Callback<void>)	
clearAccounts():Promise<void>	对所有已绑定账号进行解绑
clearAccounts(callback:Callback<void>)	
appendTags(tags:HashSet<string>):Promise<void>	新增多个标签
appendTags(tags:HashSet<string>,callback:Callback<void>)	
clearAndAppendTags(tags:HashSet<string>):Promise<void>	一次设置多个标签，会覆盖这个设备之前设置的标签。
clearAndAppendTags(tags:HashSet<string>,callback:Callback<void>)	

delTags(tags:HashSet<string>):Promise<void> delTags(tags:HashSet<string>, callback:Callback<void>)	一次删除多个标签
clearTags():Promise<void> clearTags(callback:Callback<void>)	清除这个设备的所有标签
queryTags(offset:number, limit:number):Promise<string> queryTags(offset:number, limit:number, callback:Callback<string>)	获取这个设备的标签
upsertAttributes(attributes:HashMap<string,string>):Promise<void> upsertAttributes(attributes:HashMap<string,string>, callback:Callback<void>)	该设备添加属性，有则覆盖，无则添加
delAttributes(attributesKey:HashSet<string>):Promise<void> delAttributes(attributesKey:HashSet<string>, callback:Callback<void>)	删除该设备指定的属性
clearAttributes():Promise<void> clearAttributes(callback:Callback<void>)	删除该设备已设置的所有属性。
clearAndAppendAttributes(attributes:HashMap<string,string>):Promise<void> clearAndAppendAttributes(attributes:HashMap<string,string>, callback:Callback<void>)	设置属性，会覆盖这个设备之前设置的所有属性（即清理并设置）。
setBadgeNumber(badgeNumber:number):Promise<void> setBadgeNumber(badgeNumber:number, callback:Callback<void>)	设置应用角标数，会覆盖当前应用已有的角标数
clearBadge():Promise<void> clearBadge(callback:Callback<void>)	清除应用角标数
addLocalNotification(localMessage: TPNS.LocalMessage):Promise<void> addLocalNotification(localMessage: TPNS.LocalMessage, callback:Callback<void>)	增加本地通知
clearLocalNotification():void	清除本地通知
cancelNotification(notificationId:number):Promise<void> cancelNotification(notificationId:number, callback:Callback<void>)	根据通知id 取消当前应用通知栏中某个通知
cancelAllNotification():Promise<void> cancelAllNotification(callback:Callback<void>)	取消当前应用所有通知栏所有的通知
parseNotificationMessage(want: Want): TPNS.NotificationMessage;	进入页面时oncreate，onNewWant ， 根据want 获取是否是通过通知点击，是则返回通知消息，否则返回空
setMessageReceiver(messageReciver: TPNS.MessageReceiver)	设置消息接收器，用于接收透传消息、自建通道通知消息抵达、展示回调

TPNS.Account

参数名	类型	必填	说明
account	string	是	账号
accountType	number	否	账号类型，账号可以是邮箱、QQ 号、手机号、用户名等任意类别的业务账号，账号类型取值可参考：账号类型取值表

TPNS.NotificationMessage

参数名	类型	必填	说明
msgId	string	是	通知消息的
title	string	否	通知标题，当为鸿蒙厂商通道的通知标题为空
content	string	否	通知内容，当为鸿蒙厂商通道的通知内容为空
customContent	string	否	附加参数
pushChannel	TPNS.MessagePushChannel	是	通知触发渠道
templatedId	string	否	通知模板id
traceId	string	否	通知traceId

TPNS.MessagePushChannel

参数名	账号类型之值	说明
LOCAL	99	本地通知
TPNS	100	移动推送自建通道
HARMONY	108	鸿蒙系统通道

TPNS.PushMessage

移动推送透传消息，透传消息仅当消息通过移动推送通道下发。

参数名	类型	必填	说明
msgId	string	是	消息ID
title	string	否	消息标题，点击鸿蒙厂商通道的标题为空
customContent	string	否	附加参数
templatedId	string	否	模板id
traceId	string	否	trace ID

TPNS.LocalMessage

用于添加本地消息时，需要使用到的本地消息体。

参数名	类型	必填	说明
notifyId	string	否	通知id，设置SDK内部会根据时间生成
threadId	string	否	通知分组，不传则使用默认分组
traceId	string	否	附加参数
type	TPNS.LocalMessageType	是	模板id
notificationSlotType	notificationManager.SlotType	否	通知slot 类型，参考 notificationManager.SlotType 已有如下值 0 – UNKNOWN_TYPE 未知类型。 1 – SOCIAL_COMMUNICATION 社交通信 2 – SERVICE_INFORMATION 服务提醒。

			3 – CONTENT_INFORMATION 内容资讯。 4 – LIVE_VIEW 实况窗。（预留能力，暂未支持）。 5 – CUSTOMER_SERVICE 客服消息。该类型用于用户与商家之间的客服消息，需由用户主动发起。
isAlertOnce	boolean	否	设置是否仅有一次此通知提醒。
largeIcon	string	否	通知大图标。可选字段，图像像素的总字节数不超过100KB。实际显示效果依赖于设备能力和通知中心UI样式。
largeIconType	number	否	大图标图片类型： 0 本地图片资源文件名，图片应位于media目录下 1 图片url
title	string	是	本地通知标题
content	string	是	本地通知内容
additionalText	string	否	通知概要内容，是对通知内容对补充,不传则使用content代替
inlines	string[]	否	多行类型通知，最多支持3行内容,并且每行内容不能为空，如果为空则弹出普通通知。
badgeNumber	number	否	本地通知为通知消息时生效，要增加的角标数，大于等于0， 不传角标数不变
date	number	否	本地通知触发的日期
hour	number	否	在触发本地通知日期前提下，当天允许触发本地通知开启的小时， 24小时制
min	number	否	在触发本地通知日期前提下，当天允许触发本地通知的分钟，与hour字段配合使用
actionType	TPNS.LocalMessageActionType	是	点击通知后触发的行为 打开应用首页或打开应用自定义页面
action	string	否	当actionType为打开应用自定义页面时，字段uri和action至少填写一个，优先使用action字段
uri	string	否	应用内置页面ability对应的uri，ur对象内部结构请参见 skills标签。当actionType为打开自定义页面时，字段uri和action 至少填写一个。当存在多个Ability时，分别填写不同Ability的action和uri，优先使用action查找对应的应用内置页面
customContent	string	否	附加参数
ttl	number	否	消息过期时间，默认72h
showMode	TPNS.LocalMessageShowMode	否	通知展示模式，默认都展示 或只有后台才展示

TPNS.LocalMessageType

本地通知类型

参数名	说明
NOTIFICATION	通知

PUSHMESSAGE	透传消息

TPNS.LocalMessageActionType

本地通知为通知时，actionType的定义。

参数名	说明
OPEN_APP	打开应用首页
OPEN_CUSTOM_ABILITY	打开自定义页面

TPNS.LocalMessageShowMode

本地通知为通知时，通知的展示模式。

参数名	说明
DEFAULT	默认，应用在前台和后台时本地通知都可以展示
BACKGROUND	应用在后台时展示通知。 ⚠ **注意：** 本地通知依赖app进程存活，app进程被杀掉时本地通知不会被触发。

TPNS.MessageReceiver

设置推送消息回调接口的对象的接口类，开发者自行实现该接口类才能收到推送消息的回调。

回调方法名	说明
onNotificationDeleted(err:Error,notificationMessage:TPNS.NotificationMessage)	当通知被移除时回触发此回调方法 ⚠ **注意：** 因为系统限制，当前只有本地通知和移动推送通道展示的通知被清除时才会触发此回调方法，并且 要求应用进程没有被系统杀掉。
onNotificationShow(err:Error notificationMessage:TPNS.NotificationMessage)	本地推送通知和移动推送通道展示的通知会触发此回调，err为空则表示展示成功
onTextMessage(err:Error,pushMessage:TPNS.PushMessage)	透传消息收到时会触发此回调

鸿蒙参数申请指南

最近更新时间：2024-11-05 14:44:21

简介

本文内容引导用户获取鸿蒙通道的推送参数，并完成移动推送管控制台上配置 JSON 文件。

鸿蒙参数获取步骤

1. 进入 华为开放平台 。
2. 注册和登录开发者账号，详情参见 账号注册认证（如果您是新注册账号，需进行实名认证）。
3. 在华为推送平台中新建鸿蒙项目和应用，详情参见 创建应用（应用包名需跟您在移动推送平台填写的一致）。
4. 进入**我的项目** > **项目设置** > **常规**中获取 Client ID 配置客户端本地工程，另外获取**服务密钥 JSON 文件**，填入 移动推送控制台 > **App 推送管理** > **基础配置** > **鸿蒙官方推送通道**栏目中。

获取客户端 client_id 信息

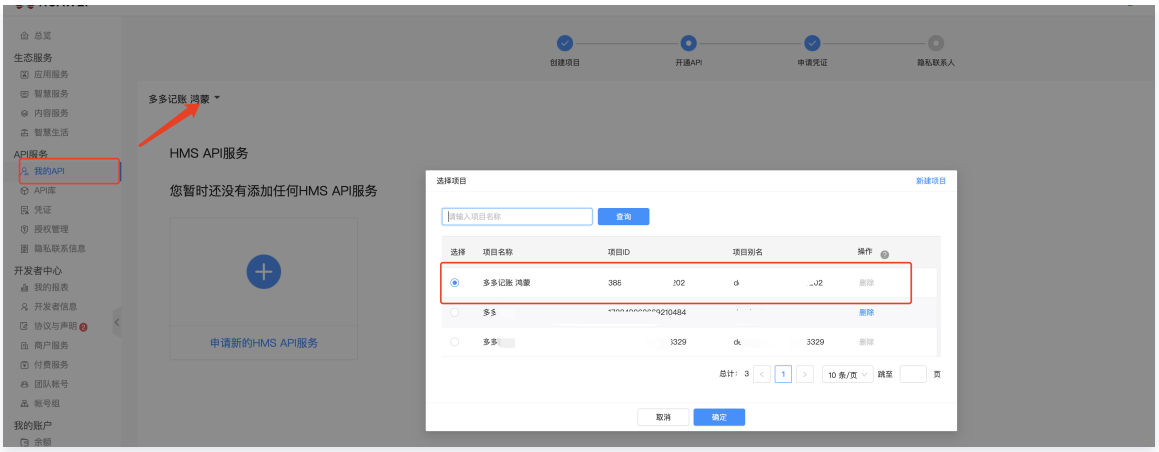
> **说明：**
> Client ID 配置详情请参见 SDK集成文档 进行配置。

> **注意：**
> 获取的是“应用”中 Client ID。

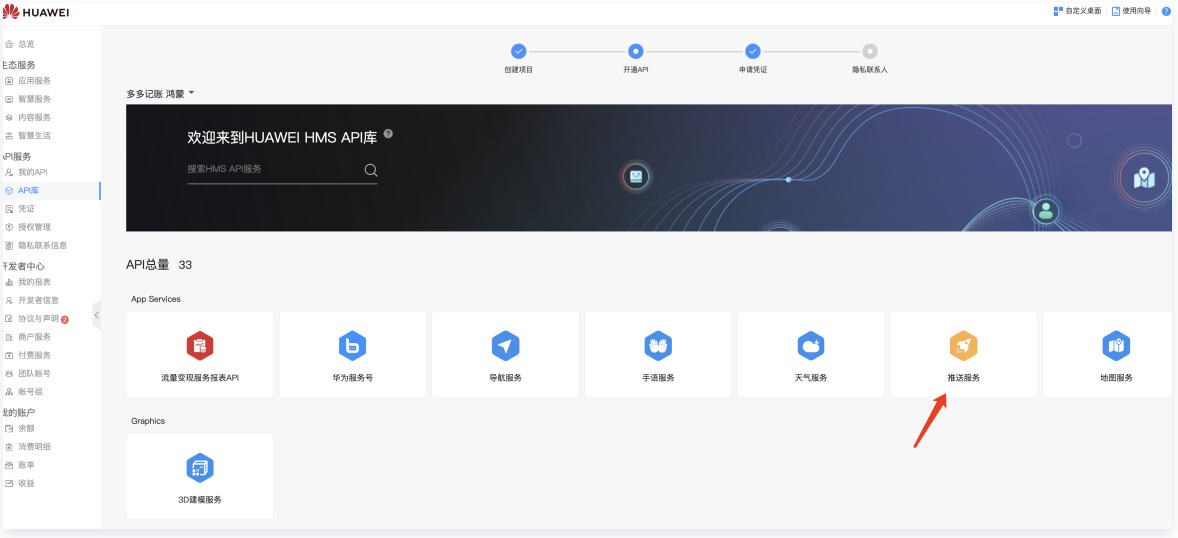
获取服务账号密钥文件

您需要在华为开发者联盟的API Console上创建并下载推送服务 API 的服务账号密钥文件，相关创建步骤请参见 API Console 操作指南-服务账号密钥 。

1. 点击“管理中心”>“API 服务”>“ 我的 API”>“选择鸿蒙项目”



2. 点击“申请新的 HMS API 服务”，在 API 库里面找到推送服务的 API。



3. 点击“启用”，完成 API 添加。

4. 点击“ 凭证”>“服务账号密钥”>“创建凭证”

5. 点击“生成公私钥”，公钥上传，下载 JSON 文件上传至控制台。

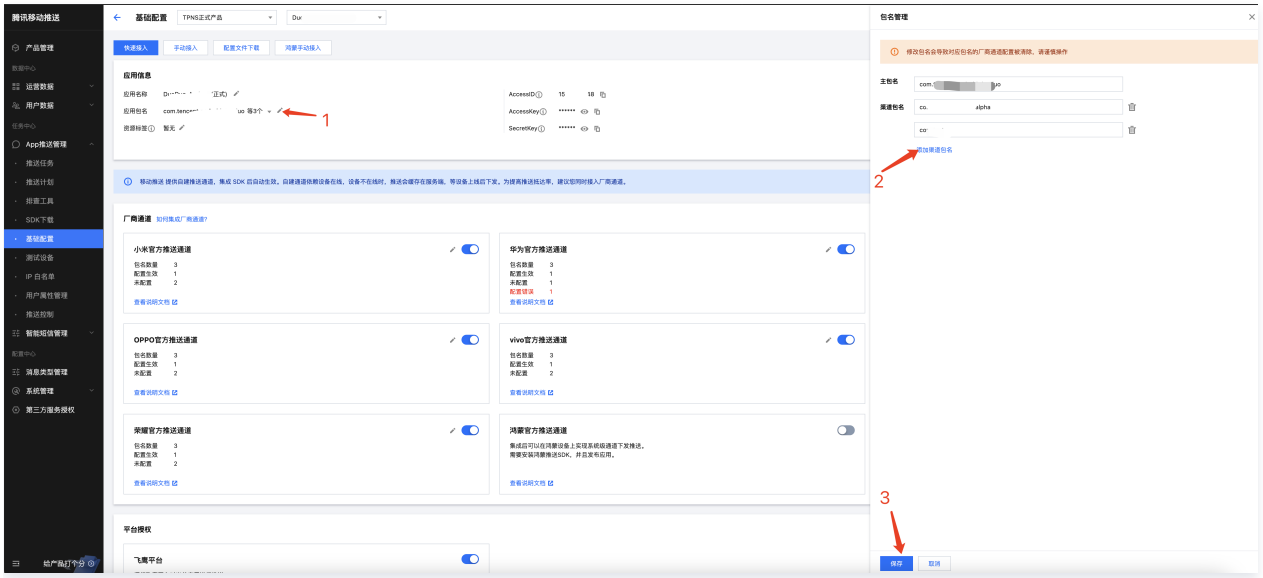
> **注意：一定要生成公私钥匙，否则下载的j son 里面没有私钥信息。**

移动推送平台配置

> **注意：**
> 请确保包名与华为开发平台保持一致，以及 json 文件中的 project ID 与项目中的项目 ID 保持一致。

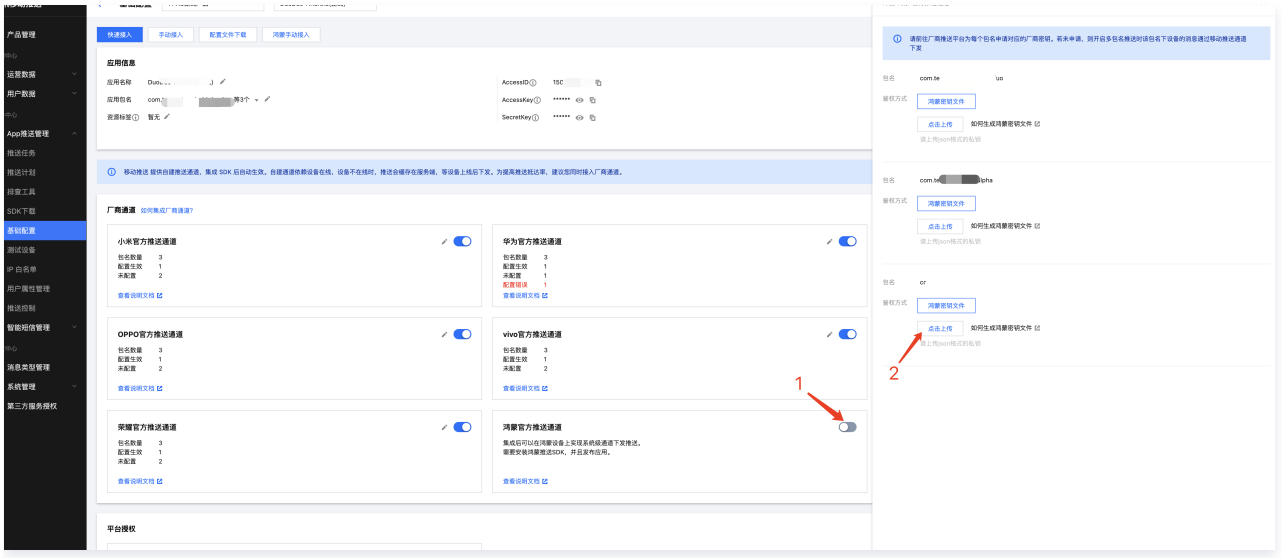
新增鸿蒙包名

登录 移动推送控制台 > App 推送管理 > 基础配置 >添加渠道包名>保存。



配置鸿蒙通道

登录 移动推送控制台 > App 推送管理 > 基础配置 > 开启鸿蒙官方推送通道>上传鸿蒙平台生成的 JSON 文件>保存。



鸿蒙通道已完成后台配置。

鸿蒙通道高级功能
鸿蒙消息分类功能使用说明

最近更新时间：2024-11-27 16:28:23

鸿蒙消息分类介绍

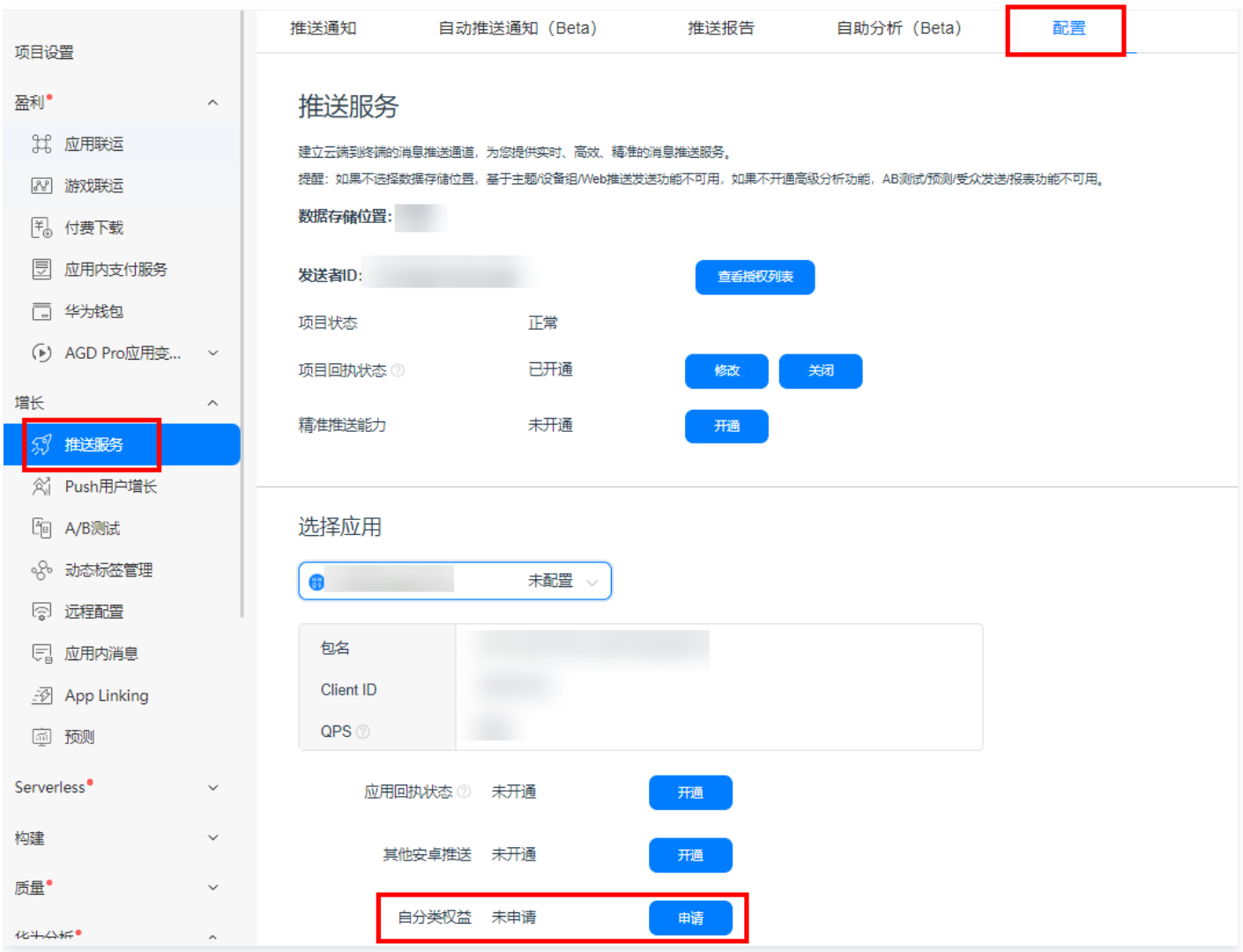
鸿蒙对 App 开发者的通知消息根据分类进行限额限频，以此保证终端用户不被过度骚扰，将通知消息智能分成两个级别：服务与通讯消息和资讯营销消息。
资讯营销类消息的每日推送数量自2023年01月05日起根据应用类型对推送数量进行上限管理，服务与通讯类消息每日推送数量不受限。
不同消息级别呈现样式对比：

消息分类	提醒方式	类型说明	推送数量限制	
服务与通讯消息	锁屏、铃声、振动	社交通讯：即时聊天，音频、视频通话。 服务提醒：订阅，出行，健康，工作事项提醒，账号动态，订单&物流，财务，设备提醒，系统提示，邮件，闹钟/计时器，秒表，进度，位置共享。 具体类别请参见 服务与通讯类消息场景说明 。	无限制。	
资讯营销消息	静默通知，仅在通知栏展示消息	内容资讯：内容推荐，新闻，财经动态，生活资讯，社交动态，调研，其他。 营销活动：产品促销，功能推荐，运营活动。 具体类别请参见 资讯营销类消息场景说明 。	新闻类（需具备《互联网新闻信息服务许可证》）	5条
			其他类	2条 根据应用类别限制每日推送数量，具体要求参见 不同应用类别的推送数量上限要求 。

自分类消息权益申请

鸿蒙通知消息自分类权限需要申请并激活后才能生效，申请流程详情请参见 鸿蒙自分类权益申请：

> **注意：**
> - 若应用没有自分类权益，则应用的推送消息将通过智能分类进行自动归类。
> - 若应用有自分类权益，将信任开发者提供的分类信息，消息不经过智能分类。



自分类消息使用

支持 Rest API 推送，暂不支持控制台消息分类推送。
在 Rest API 请求参数 `Android` 结构体中设置 `category` 参数，可实现自分类消息下发，详情请参见 PushAPI 参数说明。
推送示例如下：

```
{
    "audience_type": "account",
    "account_list": ["123"],
    "message_type": "notify",
    "message": {
        "title": "帐号下线：",
        "content": "您的帐号出现异地登录，已经下线。",
        "android": {
            "ring": 1,
            "badge_type":1,
            "harmony_message":{
                "push_type": 0,
                "category": "WORK",
                "slot_type": 1
            }
        }
    }
}
```

鸿蒙通道抵达回执获取指南

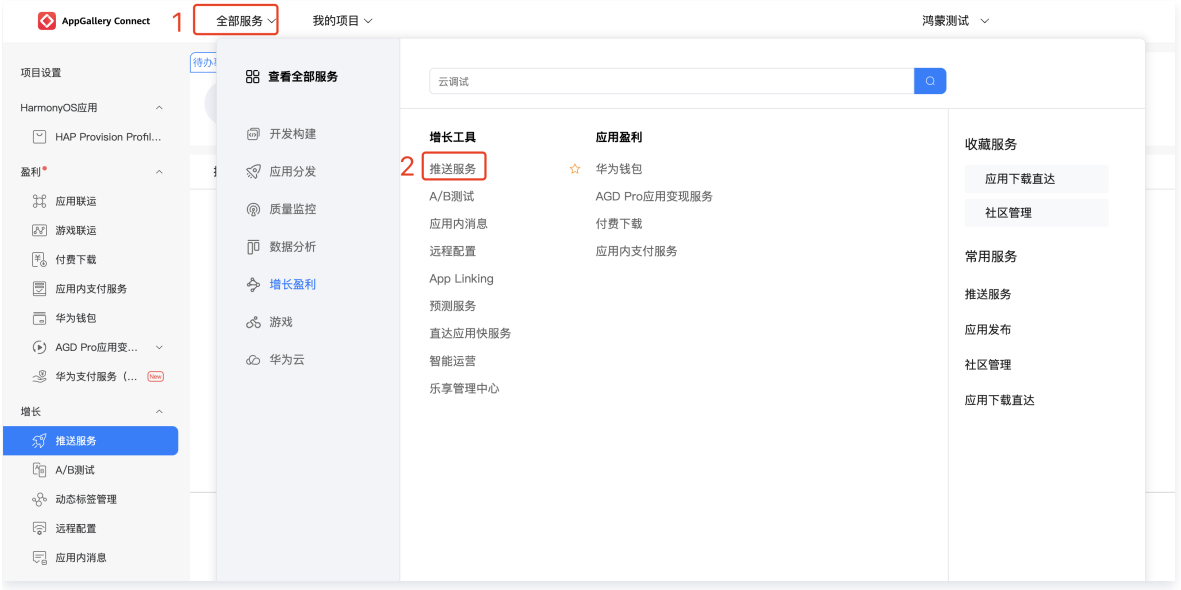
最近更新时间：2025-02-24 10:41:32

完成鸿蒙通道 SDK 集成后，需要开发者在华为开放平台开通并配置v2版本的消息回执，才能获取到鸿蒙通道抵达数据。具体配置方法可参见华为开放平台 消息回执，配置流程如下：

开通回执权益

1. 登录 AppGallery Connect 网站，选择**我的应用**。

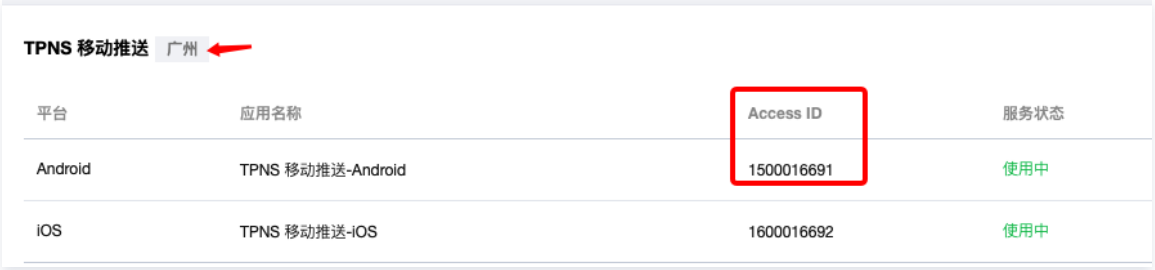
2. 选择需要开通服务的 HarmonyOS 应用所属产品的名称，进入应用信息页面。
3. 在“应用信息”页面，选择**全部服务 > 推送服务**。



4. 在“推送服务”页面，找到**应用回执状态**，单击**开通**。

回执参数配置

1. 配置消息回执地址。请在 移动推送控制台 查看您的应用的服务接入点，并复制该应用的 AccessID，选择对应服务接入点的回执地址进行配置：



服务接入点	回执地址
广州服务接入点	https://stat.tpns.tencent.com/log/statistics/harmony/AccessiD
中国香港服务接入点	https://stat.tpns.hk.tencent.com/log/statistics/harmony/AccessID
新加坡服务接入点	https://stat.tpns.sgp.tencent.com/log/statistics/harmony/AccessID
上海服务接入点	https://stat.tpns.sh.tencent.com/log/statistics/harmony/AccessID

> **注意：**
> AccessID 务必要替换为您应用的 AccessID，否则配置失败。例如应用为广州服务接入点，AccessID为 `1500016691` ，则回执地址配置为：`https://stat.tpns.tencent.com/log/statistics/` harmony `/1500016691`。

2. 配置用户名和密钥（非必填）进行身份验证。
3. 单击**测试回执**，可以对回执地址进行功能测试。

> **注意**
> - 目前单击**测试回执**，会提示“测试回调地址失败”，请忽略并直接单击**提交**。
> - 请配置鸿蒙的 v2 版本的抵达回执。

4. 单击**提交**，即可完成服务的开通。

错误码

最近更新时间：2025-03-25 15:34:42

客户端返回码

错误码	原因以及解决办法
-5	获取 Guid 出错，请检查网络，以及 AccessId 和 AccessKey 是否正确，资源是否已经购买
-7	发送请求包出错，请检查网络，或保存日志 联系我们
-101	SDK 出现某些内容 JSON 格式错误，请保存日志 联系我们
-503	获取 Guid 出错，请检查网络，以及服务接入点域名配置是否正确，详情参见 SDK 集成文档
-701	发送请求包时出现网络异常
-702	发送请求包超时
10000	起始错误
10100	当前网络不可用，网络恢复重试
10109	未知异常，切换网络或者重启设备
10110	创建链路的 handler 为 null
其他	如出现其他未知错误，请记录错误日志并 联系我们
10110	认证过程错误
10115	短时间内重复注册
10116	无效的账号 参考 接口文档添加账号
10117	无效的标签 参考 接口文档标签管理参数说明
10118	无效的用户属性 参考 接口文档属性管理参数说明
10119	无效查询标签参数 参考 接口文档标签查询参数说明
1600001	系统内部错误
1600002	序列化或反序列化错误
1600003	连接通知服务失败
1600007	通知不存在
1600012	内存空间不够，确认系统内存是否足够
10030006	资源销毁了，服务停止，请及时续费恢复
抛出异常	accessId 或着 accessKey 不对，（初始化接口传入的配置信息不对）参考： SDK集成

服务端返回码

错误码	含义
1010001	资源未部署，请确认应用是否已购买推送资源
1008001	参数解析错误
1008002	必填参数缺失
1008003	认证失败

1008004	调用服务失败
1008006	Token 无效，请检查设备 Token 是否注册成功
1008007	参数校验失败
1008011	文件上传失败
1008012	上传文件为空
1008013	证书解析失败
1008015	推送任务 ID 不存在
1008016	日期时间参数格式不对
1008019	被内容安全服务判定不和谐
1008020	证书包名校验失败
1008021	p12 证书格式内容校验失败
1008022	p12 证书密码不对
1008025	创建应用失败，产品下已存在该平台的应用
1008026	批量操作，部分失败
1008027	批量操作，全部失败
1008028	超出限频
1008029	Token 校验非法
1008030	App 未付费
1008031	App 资源已销毁
10110008	查询的 Token ， 账号不存在
10010005	推送目标不存在
10010012	非法的推送时间，请更改推送时间。 定时推送时 send_time 传入的值如果是过去的时间，具体规则如下： • 如果 | send_time − 当前时间 | <= 10min， 推送任务会创建，接收任务后会立即调度 • 如果 | send_time − 当前时间 | > 10min， 推送任务会被拒绝， 接口返回失败
10010018	重复推送
10030002	AccessID 和 AccessKey 不匹配

Android 接入指南
简介

最近更新时间：2025-02-24 10:41:32

移动推送（Tencent Push Notification Service）是一款专业的移动 App 推送平台，支持百亿级的通知和消息推送，秒级触达移动用户，现已全面支持 Android 、iOS和 macOS 三大主流平台，开发者可以方便地通过嵌入 SDK，通过 API 调用或 Web 端可视化操作，实现对特定用户推送，大幅提升用户活跃度，有效唤醒沉睡用户，并实时查看推送效果。

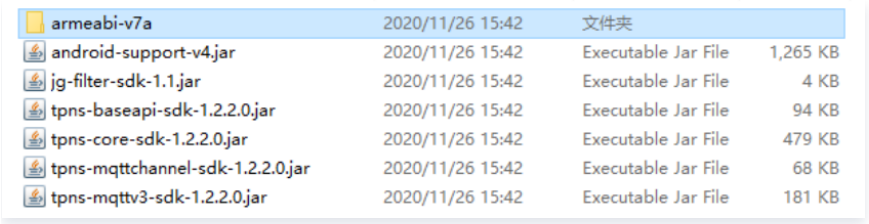
功能说明

Android SDK 是移动推送服务为客户端实现消息推送而提供给开发者的接口，主要负责完成以下功能：

- 提供通知和消息两种推送形式，方便用户使用。
- 账号、标签与设备的绑定接口，以便开发者实现特定群组的消息推送，丰富推送方式。
- 点击量上报，统计消息被用户点击的次数。
- 提供多厂商通道集成功能，方便用户集成多厂商推送。

SDK 说明

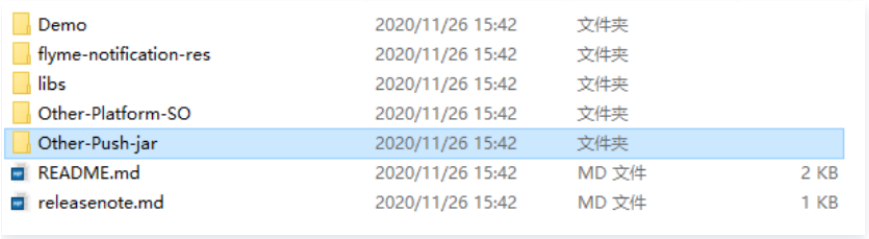
从官网上下载下来的包，解压后内容如下：



解压后根目录五个文件夹内容：

- Demo 文件夹：移动推送官方 Demo 程序，用户可以参考相关配置。
- flyme-notification-res 文件夹：魅族推送通道的资源文件，用于低版本魅族手机兼容，Flyme 6.0 及以下版本的魅族手机用户需要将对应的文件复制到 App 的 res 目录下。
- libs 文件夹：包含移动推送的 jar 和 so 文件。
- Other-Platform-SO 文件夹：包含其它不常用 CPU 架构的 so 文件。
- Other-Push-jar 文件夹：移动推送封装的华为、魅族、小米、OPPO、VIVO、FCM 的 jar 包。

libs 目录详细介绍

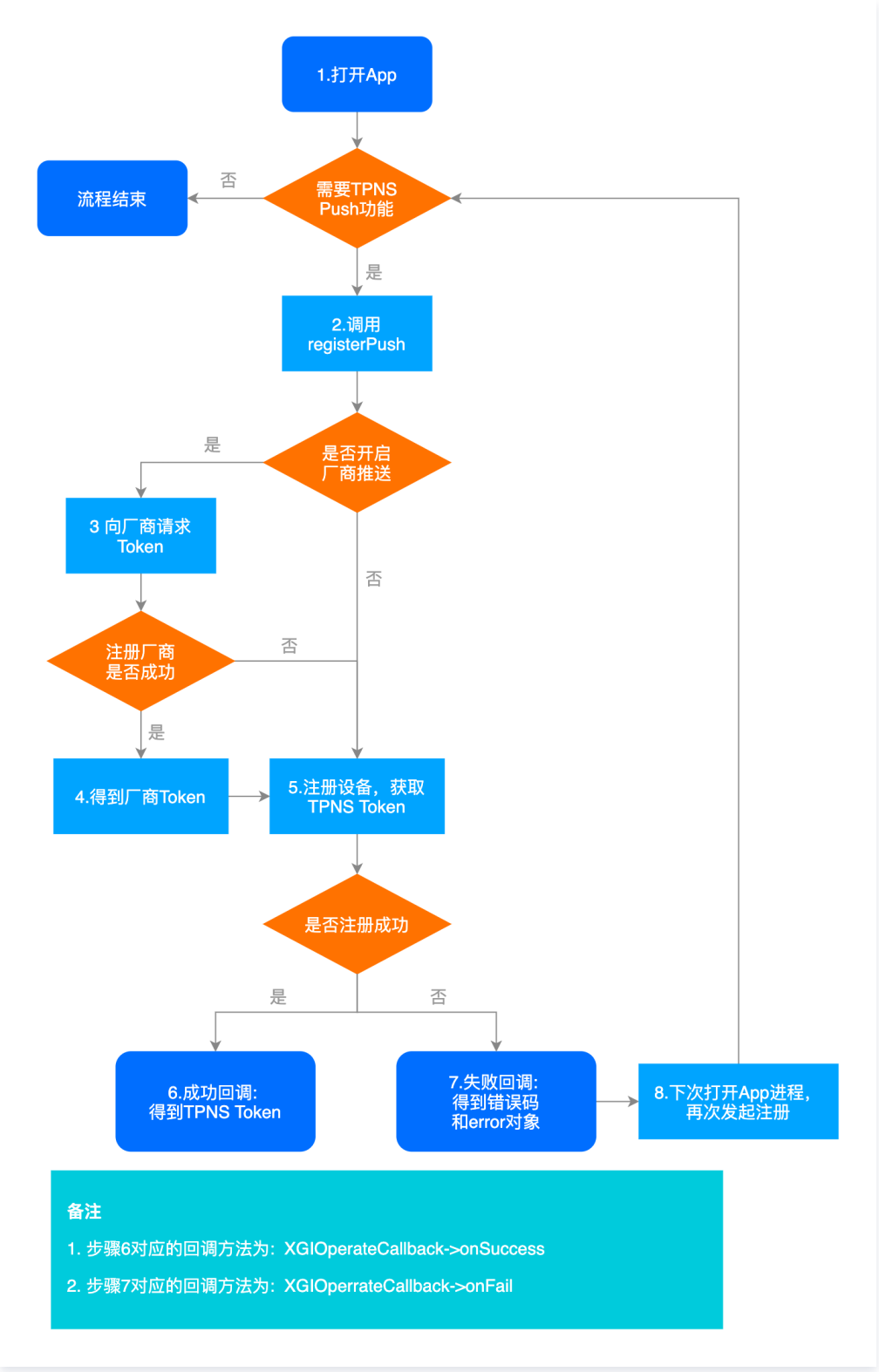


- android-support-v4.jar： 谷歌推出的兼容包，兼容 Android1.6 以上的系统。
- jg-filter-sdk-1.1.jar：金刚扫描的 jar 包，使用腾讯 SDK 的产品必须带上。
- tpns-baseapi-sdk-x.x.x.x.jar：移动推送提供的部分底层公共 API。
- tpns-core-sdk-x.x.x.x.jar：移动推送 SDK 核心模块代码，存放所有对外的类、API 和组件。
- tpns-mqttchannel-sdk-x.x.x.x.jar：移动推送上层实现基于 MQTT 协议的通信功能，长连接独立在一个进程中。
- tpns-mqttv3-sdk-x.x.x.x.jar：移动推送改造后的 MQTT 协议包，提供多厂商通道集成功能，方便用户集成多厂商推送。

常用场景流程说明

设备注册流程

下图为设备注册相关流程，具体接口方法请查看 启动与注册 。

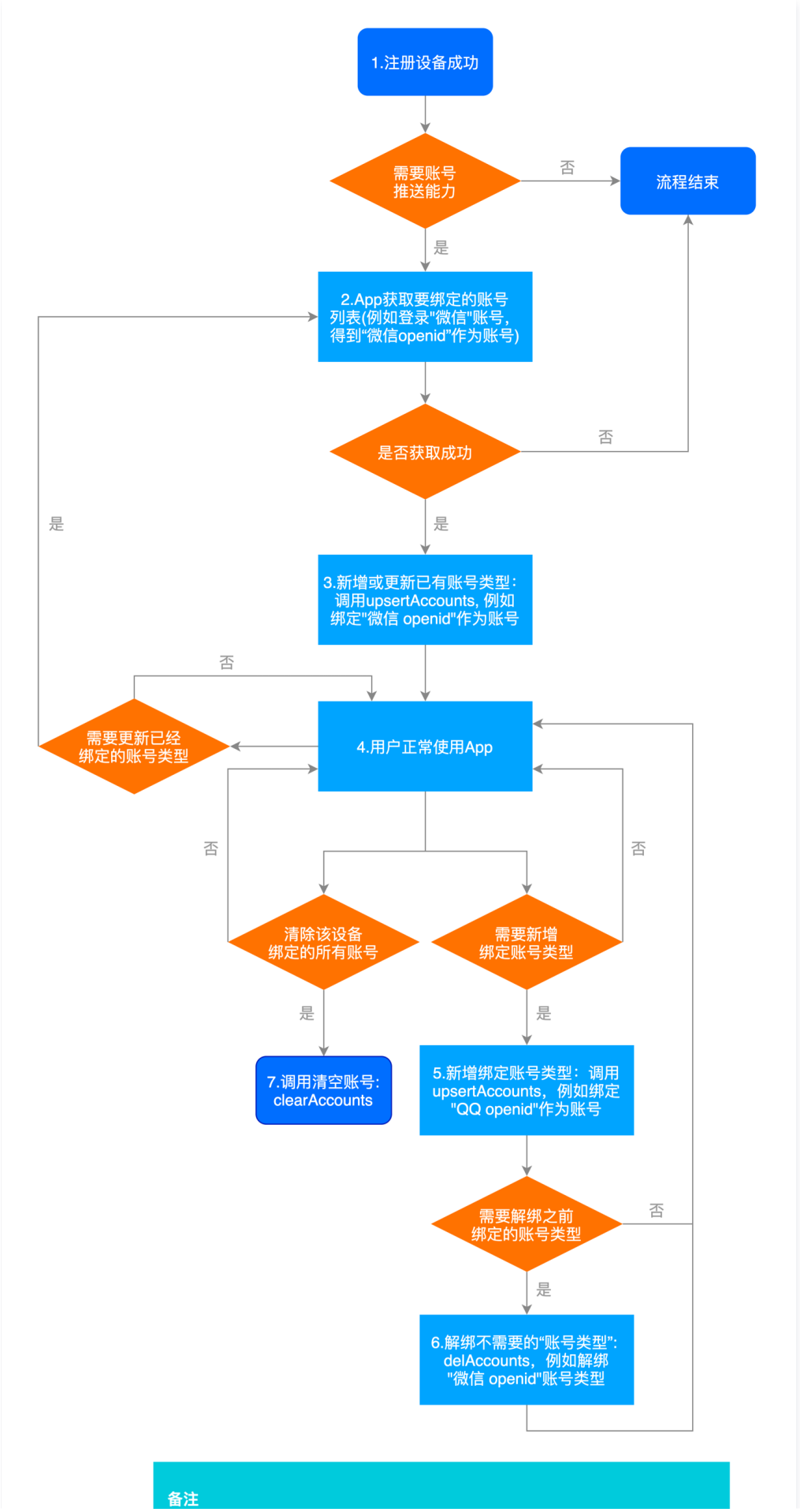


设备反注册流程

下图为设备反注册相关流程，具体接口方法请查看 反注册 。

账号相关流程

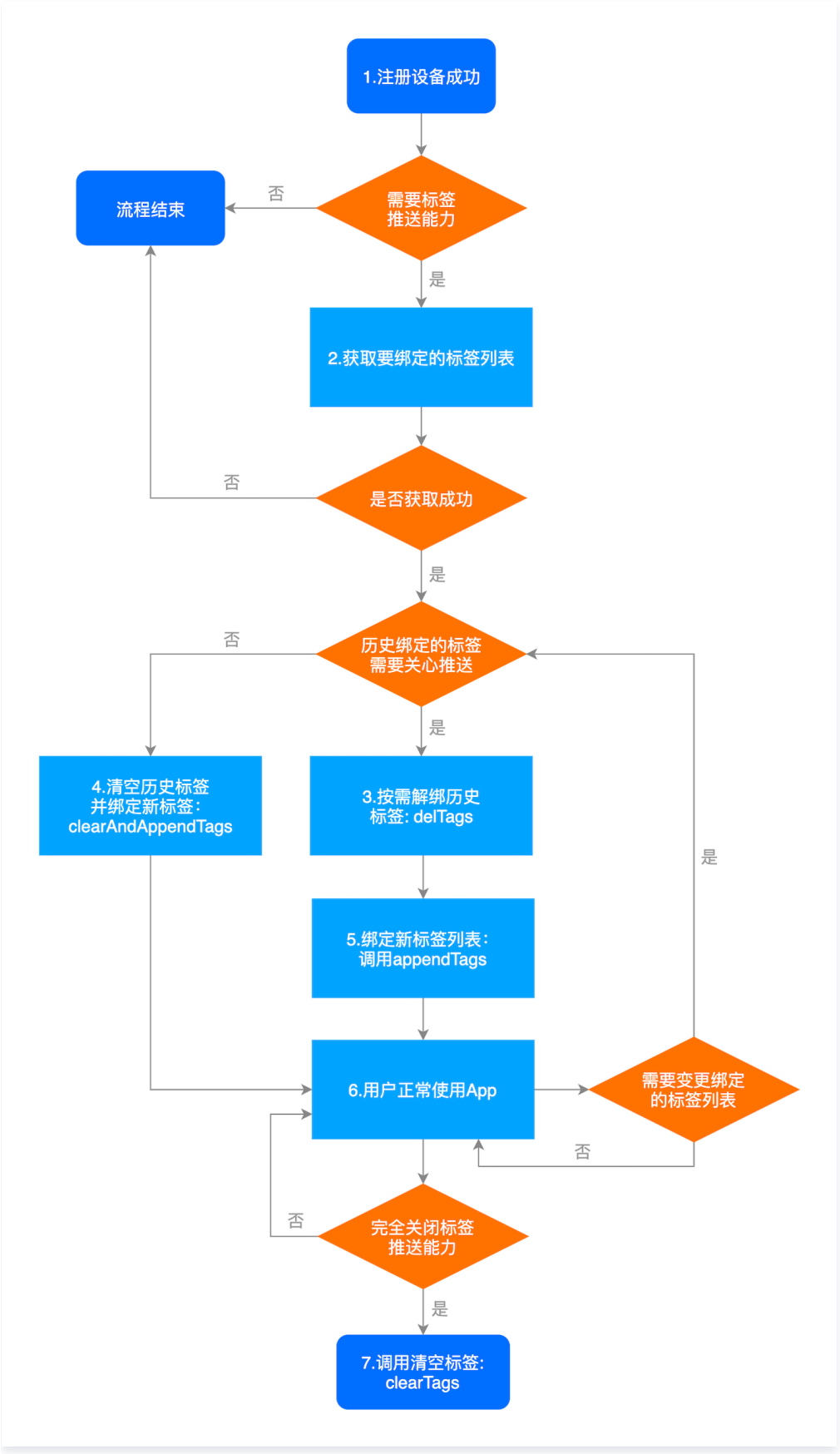
下图为账号相关流程，具体接口方法请查看 账号管理 。


1.注册设备成功
需要账号
推送能力
否
流程结束
是
2.App获取要绑定的账号
列表(例如登录"微信"账号，
得到"微信openid"作为账号)
是否获取成功
否
是
是
3.新增或更新已有账号类型：
调用upsertAccounts, 例如
绑定"微信 openid"作为账号
否
需要更新已经
绑定的账号类型
4.用户正常使用App
否
否
清除该设备
绑定的所有账号
需要新增
绑定账号类型
是
是
7.调用清空账号：
clearAccounts
5.新增绑定账号类型：调用
upsertAccounts，例如绑定
"QQ openid"作为账号
需要解绑之前
绑定的账号类型
否
是
6.解绑不需要的"账号类型"：
delAccounts，例如解绑
"微信 openid"账号类型
备注

同一种账号类型只支持绑定一个账号

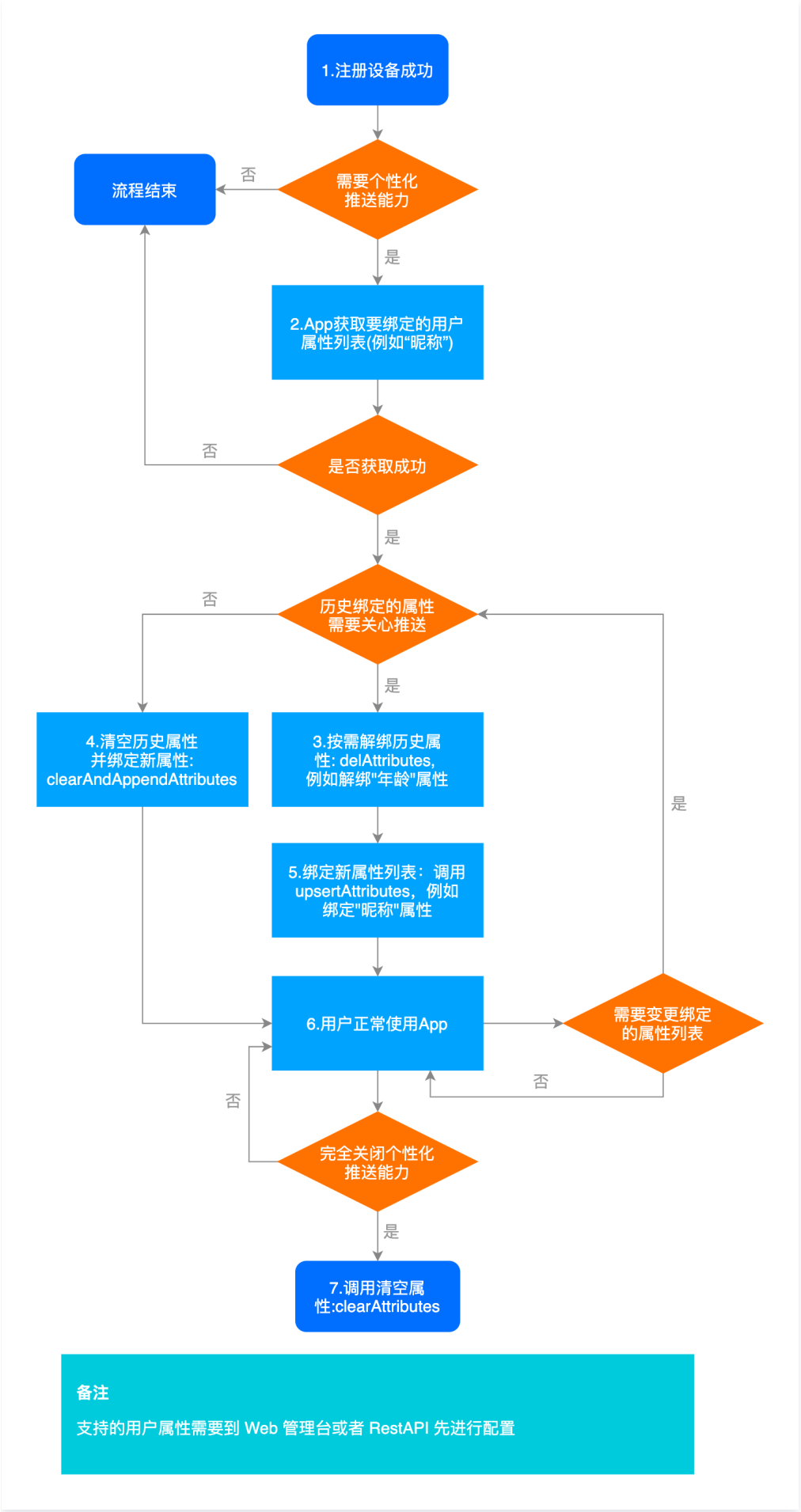
标签相关流程

下图为标签相关流程，具体接口方法请查看 标签管理 。



用户属性相关流程

下图为用户属性相关流程，具体接口方法请查看 用户属性管理 。


1.注册设备成功
需要个性化推送能力
否
流程结束
是
2.App获取要绑定的用户属性列表(例如"昵称")
是否获取成功
否
是
历史绑定的属性需要关心推送
否
是
4.清空历史属性并绑定新属性: clearAndAppendAttributes
3.按需解绑历史属性: delAttributes, 例如解绑"年龄"属性
5.绑定新属性列表：调用upsertAttributes，例如绑定"昵称"属性
是
6.用户正常使用App
需要变更绑定的属性列表
否
否
完全关闭个性化推送能力
是
7.调用清空属性:clearAttributes
备注
支持的用户属性需要到 Web 管理台或者 RestAPI 先进行配置

SDK 合规使用指南

最近更新时间：2024-10-11 15:57:42

注意：

根据《个人信息保护法》、《数据安全法》、《网络安全法》等法律法规和监管部门规章要求，App 开发运营者（以下简称为“开发者”）在提供网络产品服务时应尊重和保护最终用户的个人信息，不得违法违规收集使用个人信息，保证和承诺就个人信息处理行为获得最终用户的授权同意，遵循最小必要原则，且应当采取有效的技术措施和组织措施确保个人信息安全。为帮助开发者在使用移动推送 SDK 的过程中更好地落实用户个人信息保护相关要求，避免出现侵害最终用户个人信息权益的情形，特制定本合规使用说明，开发者使用 移动推送 SDK 时必须在《隐私政策》中告知终端用户 SDK 使用用途，并且在终端用户未同意《隐私政策》前不得初始化任何 SDK。请您务必按照以下步骤做好合规自查，避免被监管部门通报或下架您的应用。

1. 升级至最新版本移动推送 SDK

请务必确保您已经将移动推送 SDK 升级至满足监管新规的最新版本。请前往 Android SDK 发布动态 查看和下载最新版本 SDK。

2. 隐私政策中添加移动推送相关说明

请您确保您开发或运营的应用有符合监管要求的《隐私政策》文本。同时请您务必明确告知终端用户您的应用使用了移动推送服务。建议您在《隐私政策》对应的章节、列表中添加关于移动推送的说明，推荐条款如下：

SDK名称：移动推送 SDK
第三方名称：深圳市腾讯计算机系统有限公司
SDK 用途：在移动终端设备进行消息推送
收集个人信息类型：
(1)、 设备信息(手机型号，系统类型、系统版本等)用于标签化推送以及识别是否是真机、网络信息(网络类型)支持根据不同网络类型进行不同类型推送、应用数据（推送流程中产生的送达、点击、曝光等数据）用于推送业务数据统计。
(2)、（可选信息，依赖开发者设置）账号绑定信息(根据您所选用的不同推送渠道，QQ号、微信Union ID、 手机号、邮箱等)用于根据账号信息进行推送
数据处理方式：通过去标识化、加密传输及其他安全方式
官网链接：https://cloud.tencent.com/product/tpns
隐私政策链接：https://privacy.qq.com/document/preview/8565a4a2d26e480187ed86b0cc81d727

若您的应用接入了厂商通道，请务必根据真实集成情况在《隐私政策》向终端用户披露第三方 SDK 详情。提供以下信息供您参考，但请您务必参照各厂商官网就个人信息处理的说明进行操作并且以该官网的描述为准：

第三方 SDK 名称	第三方公司名称	个人信息类型	使用第三方 SDK 的目的	使用第三方 SDK 的使用场景	第三方个人信息处理规则
VIVO PUSH SDK	维沃移动通信有限公司	• 应用基本信息：appid，appkey，应用包名，应用版本号，pushSDK版本 • 应用内设备标识符：vpush的regId • 设备的硬件信息：设备类型（如手机，平板等） • 系统基本信息：系统类型（如Android，OS等）和系统版本	• 用于支持推送消息以及用于统计vivo推送SDK接口调用成功率。 • 用于支持推送消息。	使用 VIVO 厂商推送时	相关说明
OPPO PUSH SDK	广东欢太科技有限公司	应用基本信息（MCS应用包名、应用版本号、OPUSH SDK版本号），应用内设备标识符（RegistraterID、appKey、appSecret）	消息推送服务。	使用 OPPO 厂商推送时	相关说明
小米推送 SDK	北京小米移动软件有限公司	设备标识(OAID、加密的Android ID)、推送消息内容、设备信息(设备厂商、型号、归属地、运营商名称等)、通知栏设置	用于向您推送消息。	进行小米厂商推送时	• 相关说明 • 小米推送开发者应用合规指南
魅族推送 SDK	珠海市魅族通讯设备有限公司	设备相关信息（手机品牌、手机型号、系统版本、系统语言，以及设备标识符 PUSHID）和应用信息、推送状态	提供魅族手机实时消息推送、优化推送体验和统计分析。	使用魅族厂商推送	• 相关说明 • 魅族推送合规指南
华为推送 SDK	华为软件技术有限公司	应用信息、设备信息（设备的硬件信息、系统基本信息和系统设置）	用于消息推送	使用华为厂商推送时	相关说明

Google FCM SDK	Google LLC	IP 地址、移动广告 IDFV、androidID、FireBase 安装 ID、分析应用等实例、设备信息、推送消息相关信息	移动推送包含 Google FCM 商推送，以提升消息抵达率。	使用 FCM 进行推送时（仅面向境外用户）	相关说明
荣耀推送 SDK	荣耀终端有限公司	设备信息：设备标识符（AAID、PushToken） 应用的基本信息	向用户推送通知消息。	使用 honor 厂商推送时	相关说明

3. 隐私接口确认与移动推送 SDK 初始化、自启动配置及业务功能调用时机

要求内容：《SDK 合规使用说明》应详细说明 SDK 初始化及各项业务功能接口合规调用时机。App 应当在 App 登录注册页面及 App 首次运行时，通过弹窗、文本链接及附件等简洁明显且易于访问的方式，向最终用户告知涵盖个人信息处理主体、处理目的、处理方式、处理类型、保存期限等内容的个人信息处理规则，并且获得最终用户授权同意后才能处置最终用户数据。
接入说明：请务必确保终端用户有效同意您 App 中的隐私政策后，再进行移动推送 SDK 初始化。用户同意隐私政策之前，避免动态申请涉及用户个人信息的敏感设备权限；用户同意隐私政策前，您应避免私自采集和上报个人信息。当您的 App 未向用户提供服务时，例如 App 在后台运行时，请勿请求移动推送 SDK 的相关服务接口。
强烈建议您参见 Android SDK 发布动态 升级使用最新版本 SDK。
同时，SDK 提供以下接口配置，建议用户按实际需求配置使用。

3.1 初始化相关说明

Provider 组件初始化时机配置

移动推送 SDK 在后续 App 启动时移动推送 Android SDK 1.3.3.3 起新增以下接口配置是否自动初始化 SDK 的 Provider 组件功能，在终端用户有效同意您 App中的隐私政策且 App 调用了移动推送业务功能之后默认开启；

> ⚠ **注意：**
> 如不需要自动开启，请在应用工程 `Application.attachBaseContext()` 方法内调用此接口并传入 false。

```
/**
     * 配置 SDK 的 Provider 组件功能是否自动初始化，默认开启
     *
     * @param autoInitParam true: 开启; false: 关闭
     */
    public static void setAutoInit(boolean autoInitParam)
```

示例代码

```
      XGPushConfig.setAutoInit(false);
```

3.2 业务接口调用说明

请务必确保终端用户有效同意《隐私政策》后再调用移动推送 SDK 业务功能。SDK 功能业务接口链接：
https://cloud.tencent.com/document/product/548/36659
注册业务代码示例

```
/**
 * 在用户隐私政策协议同意后，再做调用移动推送TPNS SDK接口
 * 建议在
 */
private void callTPNSPushSDKRegister() {
        /*
         * 当用户同意隐私政策协议时，再调用移动推送 SDK业务功能；
         * 当用户未同意隐私政策协议时，需等用户同意后，再通过callTPNSPushSDKRegister(...)方法进行注册TPNS
* 应用只有在完成移动推送TPNS的启动与注册后才可以使用TPNS提供Push服务，在这之前请确保配置AccessId和AccessKey
         */
```

```
        boolean agreed = ...
        if (agreed) {
                //建议在线程中执行调用
                new Thread(new Runnable() {
                        @Override
                        public void run() {
                                XGPushManager.registerPush(getApplicationContext(), new
XGIOperateCallback() {
                                    @Override
                                    public void onSuccess(Object data, int flag) {
                                            Log.d("TPush","onSuccess token:" + data);
                                    }

                                    @Override
                                    public void onFail(Object data, int errCode, String msg) {
                                            Log.d("TPush","onFail token: " + data + ", errCode: " +
errCode + ", msg: " + msg);
                                    }
                                });
                        }
                }).start();
        }
}
```

4. SDK可选信息配置开关

要求内容：《SDK 合规使用说明》应详细说明SDK各项可选个人信息使用目的、场景及对应关闭的配置方式、示例。
接入说明：移动推送 SDK 向您提供了可选个人信息及权限的控制开关，您可以根据 App 所需的 SDK 功能服务自行配置打开或关闭隐私信息请求开关，对于移动推送 SDK 可选收集的个人信息的控制，开发者可以参考如下配置指引进行配置操作。因相关信息的不收集将会对其对应的功能造成影响，请开发者结合业务实际需要进行合理配置。

4.1. 配置可选个人信息

基于账号推送功能，第三方开发者需要通过预埋账号类型绑定共享给移动推送 SDK，**根据第三方开发者选择的推送通道的不同所共享给移动推送 SDK 的信息也会有所不同。如果第三方开发者未使用账号推送功能则不需要共享此类信息。**
如需清除已收集账号数据，可以参考 SDK 账号接口进行清除，SDK 业务功能接口链接：https://cloud.tencent.com/document/product/548/36659

可选个人信息类型及字段		使用目的	使用场景
设备信息（可选）	*OAID（仅Android）*	用 OAID 作为移动推送的账号，账号推送时可以使用 OAID 进行账号推送。	根据 OAID 进行账号推送时使用
	微信UnionID	使用微信 UnionID 作为移动推送的账号，账号推送时可以使用微信 UnionID 进行账号推送。	根据 UnionID 进行账号推送时使用
	QQ openID	使用 QQ openID 作为移动推送的账号，账号推送时可以使用 QQ openID 进行账号推送。	根据 QQ openID 进行账号推送时使用
账号信息（可选）	邮箱账号、新浪微博账号、支付宝账号、淘宝账号、豆瓣账号、FaceBook 账号、Twitter 账号、Google 账号、百度账号、京东账号、linkedin 账号	使用设置的账号类型和账号信息作为移动推送的账号，账号推送时可以使用该账号类型和账号信息进行账号推送。	根据设置账号类型进行账号推送
用户基本信息（可选）	*手机号*	广告定向投放及反作弊	在进行广告投放和广告投放效果分析时使用

4.2. SDK申请系统权限说明

要求内容：《SDK 合规使用说明》应详细说明SDK所需的系统权限与各业务功能间的关系，并说明权限申请时机。
接入说明：对于移动推送 SDK 申请的系统权限，您可以参考相关如下表格的内容，详细了解相关权限与各业务功能的关系及其申请时机，因相关权限的不申请将会对其对应的功能造成影响，您可以结合业务实际需要进行合理配置。配置文档链接：https://cloud.tencent.com/document/product/548/36652

操作系统	权限名称	使用目的	功能场景（申请时机）	是否可选
Android	应用包名.permission.XGPUSH_RECEIVE	使用移动推送的权限	使用推送服务 SDK 时	必选
Android	android.permission.INTERNET	进行与推送服务进行网络访问，进行业务数据上报	使用推送服务 SDK 时	必选
Android	android.permission.ACCESS_WIFI_STATE	获取 Wi-Fi 连接状态，以识别用户可下发推送图标的分辨率	使用推送服务 SDK 时	必选
Android	android.permission.ACCESS_NETWORK_STATE	获取网络连接状态，用户是否要进行网络交互	使用推送服务 SDK 时	必选
Android	android.permission.SCHEDULE_EXACT_ALARM	设置系统定时闹钟，用于保持推送长链接	调用推送 SDK 业务功能之前，由开发者自己申请	必选
Android	android.permission.WAKE_LOCK	发送保持长链接时避免短时间迅速休眠	使用推送服务 SDK 时	可选
Android	android.permission.POST_NOTIFICATIONS	用于给应用展示通知	调用推送 SDK 业务功能之前，由开发者自己申请	必选
Android	com.huawei.android.launcher.permission.CHANGE_BADGE	用于展示华为设备应用图标角标	使用推送服务 SDK 时	可选
Android	com.vivo.notification.permission.BADGE_ICON	用于展示 vivo 设备应用图标角标	使用推送服务 SDK 时	可选
Android	android.permission.VIBRATE	用于自定义通知震动	使用推送服务 SDK 时	可选
Android	android.permission.RECEIVE_USER_PRESENT	用户监控用户划屏，用户快速回复推送长链接	使用推送服务 SDK 时	可选
Android	android.permission.WRITE_EXTERNAL_STORAGE	用于缓存日志文件至外部存储中	调用推送 SDK 业务功能之前，由开发者自己申请	可选
Android	android.permission.RESTART_PACKAGES	用于重启当前应用的推送进程	使用推送服务 SDK 时	可选
Android	android.permission.GET_TASKS	获取开发者集成推送SDK应用的进程名称	使用推送服务 SDK 时	可选

5. SDK扩展业务功能的配置说明

要求内容：《SDK 合规使用指南》应详细说明SDK各项扩展业务功能介绍及对应关闭的配置方式、示例。
接入说明：移动推送 SDK 提供的主要扩展业务功能为自启动和关联启动（关联启动为老版本功能， SDK1.3.4.0 版本起不提供拉活功能，关联启动功能已下线），移动推送 SDK 为开发者提供关闭自启动，关联启动接口，开发者可以调用接口，向最终用户提供关闭自启动、关联启动的能力。退出后，最终用户接收推送的实时性会降低。开发者需遵守相关法律法规的要求，在 App 内为最终用户提供退出个性化广告的功能，保证在最终用户点击退出功能后调用移动推送 SDK 的能力接口。相关配置指引参考如下说明。

5.1. 自启动配置说明

自启动配置

移动推送 Android SDK 1.3.4.3 起新增以下接口配置是否开启自启动，默认关闭。

> **注意:**
> 如需开启，请一定参考 移动推送隐私保护指引 ，在应用自己的隐私政策中披露自启动行为，并确认在用户同意隐私政策后才配置开启。

```
 /**
 * 配置是否开启自启动，默认关闭。
 * 即使配置为开启，在不同厂商系统的电源管理方案下自启动行为可能同样受限。
 *
 * @param context 应用上下文
 * @param enable true：开启自启动；false：关闭自启动
 * @since 1.3.4.3
 */
public static void enableAutoStart(Context context, boolean enable)
```

示例代码

```
XGPushConfig.enableAutoStart(context, false);
```

说明：

- 移动推送 SDK 在测试观察部分厂商系统自带的应用启动记录工具时发现，当前移动推送 SDK 版本在某些厂商的部分设备上仍然有被记录存在自启动情况，但属于系统导致的自启动非 SDK 行为。该情况我们已经和厂商直接沟通，其反馈后续的新系统不会再有此问题。如您对此有相关疑问，请参考本文第7点提及的联系方式与我们联系。
- 关于第三方 SDK 自启动的特别提示：基于推送的目的，小米推送 SDK 、FCM 推送 SDK、华为推送 SDK 可能存在自启动行为。

关联启动配置

配置是否允许拉起其他集成移动推送服务的应用的后台推送进程，默认关闭。

注意：
如需开启，请一定参考 移动推送隐私保护指引 ，在应用自己的隐私政策中披露关联启动行为，并确认在用户同意隐私政策后才配置开启。受监管合规法规影响，从 SDK 1.3.4.0 版本起不提供拉活功能，关联启动功能已下线。

```
/**
 * 设置是否允许拉起其他集成 TPNS 服务的应用的后台推送进程，默认关闭。
 * 即使配置为开启，在不同厂商系统的电源管理方案下关联启动行为可能同样受限。
 *
 * @param context 应用上下文
 * @param pullUp true：开启；false：关闭
 * @since 1.1.5.4
 */
public static void enablePullUpOtherApp(final Context context, final boolean pullUp)
```

示例代码

```
XGPushConfig.enablePullUpOtherApp(context, false);
```

6. SDK 可按照不同频次、精度收集个人信息的配置说明

要求内容：如果 SDK 可按照不同频次、精度收集个人信息的，《SDK 合规使用说明》应说明不同频次、精度的使用目的、场景及对应选择的配置方式、示例。
接入说明：收集频次方面，移动推送 SDK 的数据采集仅在 App 调用/最终用户触发相关功能时触发，不涉及定时逻辑等频次控制选项。收集精度方面不涉及精度相关控制，相关业务接口务必确保终端用户有效同意《隐私政策》后调用，可参考 SDK 接口文档链接：
https://cloud.tencent.com/document/product/548/36659

7. 最终用户同意方式的示例

要求内容：《SDK 合规使用说明》应详细说明 App 获取最终用户授权同意的建议方式，其中需要取得最终用户单独同意的，应显著提示并给出示例。
接入说明：App 首次运行时应当有隐私弹窗，隐私弹窗中应公示简版隐私政策内容并附完整版隐私政策链接，并明确提示最终用户阅读并选择是否同意隐私政策；隐私弹窗应提供同意按钮和拒绝同意的按钮，并由最终用户主动选择。
披露示例：

8. 撤回同意机制说明

请您告知终端用户其享有选择关闭相应权限或行使退出（opt-out）权利，一旦终端用户行使前述权利，其个人信息将不再会被处理。我们建议您在终端用户撤销同意处理其个人信息的授权时，参见腾讯推送服务 移动推送Android 接入指南 - 接口文档 “反注册接口说明”部分进行处理，以便用户更便捷行使退出的选择权。

9. 隐私保护机制

如果您对 SDK 权限有任何疑问、意见和建议，或者需要腾讯协助关闭某项权限采集能力，可通过以下联系方式与我们联系：

- 电子邮件：tpns_team@tencent.com
- 联系地址：深圳市南山区粤海街道麻岭社区科技中一路腾讯大厦

您还可以随时通过访问移动推送官网在线客服系统与我们联系，我们将及时为您提供咨询和服务，确保各项隐私机制的落实和执行。

SDK 集成

最近更新时间：2025-03-25 15:34:42

简介

本文内容引导集成移动推送 SDK 在线通道推送能力，提供 AndroidStudio Gradle 自动集成和 Android Studio 手动集成两种方式指引。如需在应用进程被杀时也能收到推送，请在完成本文的集成操作后，参考 厂商通道接入指南 文档，完成各厂商通道的接入。

> **注意**
> 为了避免您的 App 被监管部门通报或下架，请您在接入 SDK 之前务必按照 Android 的SDK 合规使用指南 在《隐私政策》中增加移动推送相关说明，并且在用户同意《隐私政策》后再初始化移动推送SDK。

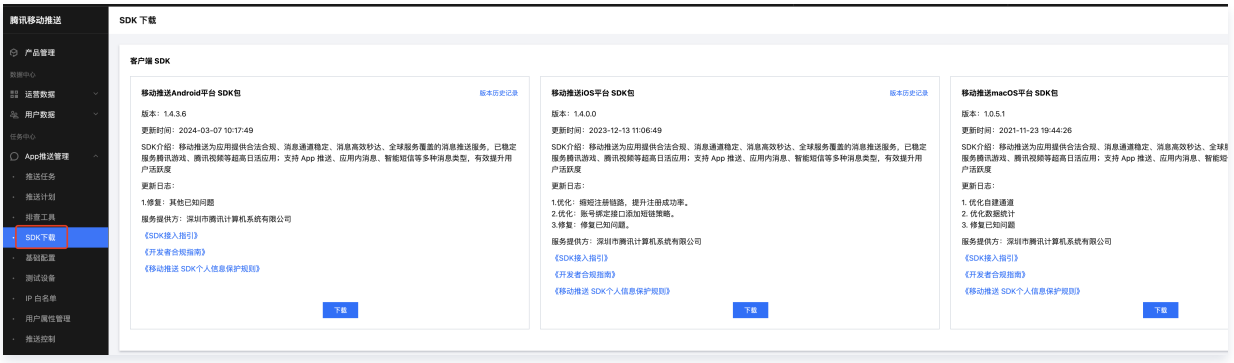
SDK 集成（二选一）

AndroidStudio Gradle 自动集成

操作步骤

> **注意**
> 在配置 SDK 前，确保已创建 Android 平台的应用。

1. 登录 移动推送控制台，在**产品管理>配置管理**页面获取应用的 AccessID、AccessKey。
2. 在 SDK 下载 页面，获取当前最新版本号。



3. 在 app build.gradle 文件下，配置以下内容：

```
android {
 ......
 defaultConfig {

     //控制台上注册的包名.注意application ID 和当前的应用包名以及控制台上注册应用的包名必须一致。
     applicationId "您的包名"
     ......

     ndk {
         //根据需要 自行选择添加的对应cpu类型的.so库。
         abiFilters 'armeabi', 'armeabi-v7a', 'arm64-v8a'
         // 还可以添加 'x86', 'x86_64', 'mips', 'mips64'
     }

     manifestPlaceholders = [

         XG_ACCESS_ID : "注册应用的accessid",
         XG_ACCESS_KEY : "注册应用的accesskey",
     ]
     ......
```

```
  }
  ......
}

dependencies {
  ......
  //添加以下依赖
  implementation 'com.tencent.tpns:tpns:[VERSION]-release'
        // TPNS 推送 [VERSION] 为最新 SDK 版本号，即为上述步骤2获取的版本号
}
```

注意

- 如果您的应用服务接入点为广州，SDK 默认实现该配置。
- 如果您的应用服务接入点为上海、新加坡或中国香港，请按照下文步骤完成其他服务接入点域名配置，否则会导致注册推送服务失败并返回-502或1008003错误码。
  在 AndroidManifest 文件 application 标签内添加以下元数据：

```
<application>
    // 其他安卓组件
    <meta-data
            android:name="XG_SERVER_SUFFIX"
            android:value="其他服务接入点域名" />
</application>
```

- 其他服务接入点域名如下：
  - 上海：`tpns.sh.tencent.com`
  - 新加坡：`tpns.sgp.tencent.com`
  - 中国香港：`tpns.hk.tencent.com`

注意事项

- 如在添加以上 abiFilter 配置后， Android Studio 出现以下提示：
  NDK integration is deprecated in the current plugin. Consider trying the new experimental plugin，则在 Project 根目录的 gradle.properties 文件中添加 `android.useDeprecatedNdk=true` 。
- 如需监听消息请参考 XGPushBaseReceiver 接口或 Demo（在 SDK 压缩包内，可前往 [SDK 下载](https://console.cloud.tencent.com/tpns/sdkdownload) 页面获取 ）的 MessageReceiver 类。自行继承 XGPushBaseReceiver 并且在配置文件中配置如下内容（请勿在 receiver 里处理耗时操作）：

```
<receiver android:name="com.tencent.android.xg.cloud.demo.MessageReceiver">
    <intent-filter>
        <!-- 接收消息透传 -->
        <action android:name="com.tencent.android.xg.vip.action.PUSH_MESSAGE" />
        <!-- 监听注册、反注册、设置/删除标签、通知被点击等处理结果 -->
        <action android:name="com.tencent.android.xg.vip.action.FEEDBACK" />
    </intent-filter>
</receiver>
```

- 如需兼容 Android P，需要添加使用 Apache HTTP client 库，在 AndroidManifest 的 application 节点内添加以下配置即可。

```
<uses-library android:name="org.apache.http.legacy" android:required="false"/>
```

Android Studio 手动集成

前往 SDK 下载 页面获取最新版 SDK，并参考以下步骤将 SDK 导入到您的 Android 工程中。

工程配置

将 SDK 导入到工程的步骤为：

1. 创建或打开 Android 工程。
2. 将移动推送 SDK 目录下的 libs 目录所有 .jar 文件拷贝到工程的 libs（或 lib）目录下。
3. .so 文件是移动推送必须的组件，支持 armeabi、armeabi-v7a、arm64-v8a、mips、mips64、x86、x86_64平台，请根据自己当前 .so 支持的平台添加。
4. 打开 AndroidManifest.xml，添加以下配置（建议参考下载包 Demo 中的 Merged Manifest 修改），其中 “APP的AccessId ” 和 “APP的AccessKey” 替换为 App 对应的 AccessId 和 AccessKey，请确保按照要求配置，否则可能导致服务不能正常使用。

权限配置

移动推送 SDK 正常运行所需要的权限。示例代码如下：

```
<!-- 【必须】 移动推送 TPNS SDK VIP版本所需权限 -->
<permission
        android:name="应用包名.permission.XGPUSH_RECEIVE"
        android:protectionLevel="signature" />
<uses-permission android:name="应用包名.permission.XGPUSH_RECEIVE" />

<!-- 【必须】 移动推送 TPNS SDK 所需权限 -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.SCHEDULE_EXACT_ALARM" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>

<!-- 【常用】 移动推送 TPNS SDK所需权限 -->
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.RECEIVE_USER_PRESENT" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.GET_TASKS" />
```

权限	是否必选	说明
android.permission.INTERNET	必选	允许程序访问网络连接，可能产生 GPRS 流量
android.permission.ACCESS_WIFI_STATE	必选	允许程序获取当前 Wi-Fi 接入的状态以及 WLAN 热点的信息
android.permission.ACCESS_NETWORK_STATE	必选	允许程序获取网络信息状态
android.permission.WAKE_LOCK	必选	允许程序在手机屏幕关闭后，后台进程仍然运行
android.permission.SCHEDULE_EXACT_ALARM	必选	允许定时广播
android.permission.VIBRATE	可选	允许应用震动
android.permission.RECEIVE_USER_PRESENT	可选	允许应用可以接收点亮屏幕或解锁广播
android.permission.WRITE_EXTERNAL_STORAGE	可选	允许程序写入外部存储
android.permission.RESTART_PACKAGES	可选	允许程序结束任务
android.permission.GET_TASKS	可选	允许程序获取任务信息

组件和应用信息配置

> ⚠ **注意**
> 移动推送 Android SDK 1.1.6.3 及之前版本请参考文档 1.1.6.3 及之前版本组件和应用信息配置 。

```
<application>
```

```
    <activity android:name="com.tencent.android.tpush.TpnsActivity"
            android:theme="@android:style/Theme.Translucent.NoTitleBar"
            android:launchMode="singleInstance"
            android:exported="true">
            <intent-filter>
                <action android:name="${applicationId}.OPEN_TPNS_ACTIVITY" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                <data
                    android:scheme="tpns"
                    android:host="${applicationId}"/>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.BROWSABLE" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action" />
            </intent-filter>
        </activity>

        <activity
            android:name="com.tencent.android.tpush.InnerTpnsActivity"
            android:exported="false"
            android:launchMode="singleInstance"
            android:theme="@android:style/Theme.Translucent.NoTitleBar">
            <intent-filter>
                <action android:name="${applicationId}.OPEN_TPNS_ACTIVITY_V2" />

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                <data
                    android:host="${applicationId}"
                    android:scheme="stpns" />

                <action android:name="android.intent.action.VIEW" />

                <category android:name="android.intent.category.BROWSABLE" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action" />
            </intent-filter>
        </activity>

        <!-- 【必须】 信鸽receiver广播接收 -->
        <receiver
            android:name="com.tencent.android.tpush.XGPushReceiver"
            android:exported="false"
            android:process=":xg_vip_service">

            <intent-filter android:priority="0x7fffffff">


                <!-- 【必须】 信鸽SDK的内部广播 -->
                <action android:name="com.tencent.android.xg.vip.action.SDK" />
                <action android:name="com.tencent.android.xg.vip.action.INTERNAL_PUSH_MESSAGE" />
                <action android:name="com.tencent.android.xg.vip.action.ACTION_SDK_KEEPALIVE" />
            </intent-filter>
```

```
        </receiver>

    <!-- 【必须】移动推送 TPNS service -->
    <service
        android:name="com.tencent.android.tpush.service.XGVipPushService"
        android:exported="false"
        android:process=":xg_vip_service">
    </service>

    <!-- 【必须】通知 service ，android:name 部分改为包名.XGVIP_PUSH_ACTION -->
        <service android:name="com.tencent.android.tpush.rpc.XGRemoteService"
            android:exported="false">
            <intent-filter>
                <!-- 【必须】请修改为当前APP名包.XGVIP_PUSH_ACTION -->
                <action android:name="应用包名.XGVIP_PUSH_ACTION" />
            </intent-filter>
        </service>

    <!-- 【必须】【注意】authorities 修改为包名.XGVIP_PUSH_AUTH -->
    <provider
        android:name="com.tencent.android.tpush.XGPushProvider"
        android:authorities="应用包名.XGVIP_PUSH_AUTH" />

    <!-- 【必须】【注意】authorities 修改为包名.TPUSH_PROVIDER -->
    <provider
        android:name="com.tencent.android.tpush.SettingsContentProvider"
        android:authorities="应用包名.TPUSH_PROVIDER" />

    <!-- 【可选】用于增强保活能力 -->
    <provider
        android:name="com.tencent.android.tpush.XGVipPushKAProvider"
        android:authorities="应用包名.AUTH_XGPUSH_KEEPALIVE"
        android:exported="true" />

    <!-- 【可选】APP实现的Receiver，用于接收消息透传和操作结果的回调，请根据需要添加 -->
    <!-- YOUR_PACKAGE_PATH.CustomPushReceiver需要改为自己的Receiver： -->
    <receiver android:name="应用包名.MessageReceiver"
        android:exported="false">
        <intent-filter>
            <!-- 接收消息透传 -->
            <action android:name="com.tencent.android.xg.vip.action.PUSH_MESSAGE" />
            <!-- 监听注册、反注册、设置/删除标签、通知被点击等处理结果 -->
            <action android:name="com.tencent.android.xg.vip.action.FEEDBACK" />
        </intent-filter>
    </receiver>

    <!-- MQTT START -->
    <service android:exported="false"
            android:process=":xg_vip_service"
            android:name="com.tencent.tpns.mqttchannel.services.MqttService" />

    <provider
            android:exported="false"
            android:name="com.tencent.tpns.baseapi.base.SettingsContentProvider"
            android:authorities="应用包名.XG_SETTINGS_PROVIDER" />

    <!-- MQTT END-->

    <!-- 【必须】 请修改为 APP 的 AccessId，“15”开头的10位数字，中间没空格 -->
    <meta-data
        android:name="XG_V2_ACCESS_ID"
```

```
        android:value="APP的AccessId" />
    <!-- 【必须】 请修改为APP的AccessKey，“A”开头的12位字符串，中间没空格 -->
    <meta-data
        android:name="XG_V2_ACCESS_KEY"
        android:value="APP的AccessKey" />

</application>

<!-- 【必须】 移动推送 TPNS SDK 5.0版本所需权限 -->
<permission
    android:name="应用包名.permission.XGPUSH_RECEIVE"
    android:protectionLevel="signature" />
<uses-permission android:name="应用包名.permission.XGPUSH_RECEIVE" />

<!-- 【必须】 移动推送 TPNS SDK所需权限 -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>

<!-- 【常用】 移动推送 TPNS SDK所需权限 -->
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.RECEIVE_USER_PRESENT" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

注意

- 如果您的应用服务接入点为广州，SDK 默认实现该配置。
- 如果您的应用服务接入点为上海、新加坡或中国香港，请按照下文步骤完成其他服务接入点域名配置，否则会导致注册推送服务失败并返回-502或1008003错误码。
  在 AndroidManifest 文件 application 标签内添加以下元数据：

```
<application>
    // 其他安卓组件
    <meta-data
            android:name="XG_SERVER_SUFFIX"
            android:value="其他服务接入点域名" />
</application>
```

- 其他服务接入点域名如下：
  - 上海：`tpns.sh.tencent.com`
  - 新加坡：`tpns.sgp.tencent.com`
  - 中国香港：`tpns.hk.tencent.com`

调试及设备注册

开启 Debug 日志数据

注意：
上线时请设置为 false。

```
XGPushConfig.enableDebug(this,true);
```

Token 注册

在需要启动推送服务的地方调用推送服务注册接口：

> **注意**
> 建议仅在 App 的主进程内调用注册接口。

```
XGPushManager.registerPush(this, new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        //token在设备卸载重装的时候有可能会变
        Log.d("TPush", "注册成功，设备token为：" + data);
    }

    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.d("TPush", "注册失败，错误码：" + errCode + ",错误信息：" + msg);
    }
});
```

过滤 "TPush" 注册成功的日志如下：

```
TPNS register push success with token : 6ed8af8d7b18049d9fed116a9db9c71ab44d5565
```

关闭日志打印

调用 XGPushConfig.enableDebug(context, false) 关闭 SDK debug 日志开关时，SDK 默认仍会打印部分日常运行日志（包含移动推送 Token）。您可以通过在 Application.onCreate 内调用如下方法，来关闭这些日常运行日志在控制台的输出打印：

```
new XGPushConfig.Build(context).setLogLevel(Log.ERROR);
```

代码混淆

如果您的项目中使用 proguard 等工具，已做代码混淆，请保留以下选项，否则将导致移动推送服务不可用：

```
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep class com.tencent.android.tpush.** {*;}
-keep class com.tencent.tpns.baseapi.** {*;}
-keep class com.tencent.tpns.mqttchannel.** {*;}
-keep class com.tencent.tpns.dataacquisition.** {*;}

-keep class com.tencent.bigdata.baseapi.** {*;}    // 1.2.0.1 及以上版本不需要此条配置
-keep class com.tencent.bigdata.mqttchannel.** {*;}  // 1.2.0.1 及以上版本不需要此条配置
```

> **注意**
> 如果移动推送 SDK 被包含在 App 的公共 SDK 里，即使公共 SDK 有增加配置混淆规则，主工程 App 也必须要同时增加配置混淆规则。

高级配置（可选）

关闭联合保活

如需关闭联合保活功能，请在应用初始化的时候，例如 Application 或 LauncherActivity 的 onCreate 中调用如下接口，并传递 false 值：

> **注意**
> - 仅 1.1.6.0 之后版本支持关闭联合保活功能，1.1.6.0 之前版本移动推送默认开启联合保活能力，且不可关闭。
> - 1.2.6.0 起默认关闭联合保活功能，可不再调用此接口。在 SDK 1.3.4.0 及以上的版本关联启动功能已下线。

```
XGPushConfig.enablePullUpOtherApp(Context context, boolean pullUp);
```

若您使用 gradle 自动集成方式，请在自身应用的 AndroidManifest.xml 文件 <application> 标签下配置如下结点，其中 `xxx` 为任意自定义名称；如果使用手动集成方式，请修改如下节点属性：

```
<!-- 在自身应用的AndroidManifest.xml文件中添加如下结点，其中 xxx 为任意自定义名称： -->
<!-- 关闭与 TPNS 应用的联合保活功能，请配置 -->
<provider
        android:name="com.tencent.android.tpush.XGPushProvider"
        tools:replace="android:authorities"
        android:authorities="应用包名.xxx.XGVIP_PUSH_AUTH"
        android:exported="false" />
```

若控制台有以下日志打印，则表明联合保活功能已经关闭：`I/TPush: [ServiceUtil] disable pull up other app` 。

获取移动推送 Token 交互建议

建议您完成 SDK 集成后，在 App 的关于、意见反馈等比较不常用的 UI 中，通过手势或者其他方式显示移动推送 Token，控制台和 Restful API 推送需要根据移动推送 Token 进行 Token 推送，后续问题排查也需要根据移动推送 Token 进行定位。
示例代码如下：

```
//获取 Token
XGPushConfig.getToken(getApplicationContext());
```

获取移动推送运行日志交互建议

SDK 提供日志上报接口。如用户在应用上线后遇到推送相关问题，可以通过引导用户操作触发此接口，上传 SDK 运行日志并获取回调返回的日志文件下载地址，方便问题排查。详情参见 日志上报接口 。
示例代码如下：

```
XGPushManager.uploadLogFile(context, new HttpRequestCallback() {
    @Override
```

```
    public void onSuccess(String result) {
        Log.d("TPush", "上传成功，文件地址：" + result);
    }
        @Override
    public void onFailure(int errCode, String errMsg) {
        Log.d("TPush", "上传失败，错误码：" + errCode + ",错误信息：" + errMsg);
    }
});
```

隐私协议声明建议

您可在申请 App 权限使用时，使用以下内容声明授权的用途：

> **我们使用 腾讯云移动推送 TPNS 用于实现产品信息的推送，在您授权我们“访问网络连接”和“访问网络状态”权限后，表示您同意 腾讯 SDK 隐私协议 。您可以通过关闭终端设备中的通知选项来拒绝接受此 SDK 推送服务。**

其中上述声明授权的两个链接如下：

- **腾讯云移动推送：** `https://cloud.tencent.com/product/tpns`
- **腾讯 SDK 隐私协议：** `https://cloud.tencent.com/document/product/548/50955`

接口文档

最近更新时间：2025-02-24 10:41:32

说明

本文档中账号功能、删除标签功能适用于 SDK 1.2.3.0 或更高版本，1.2.3.0 及之前版本请参见 接口文档 。
所有 API 接口的包名路径前缀都是：`com.tencent.android.tpush`，其中有以下几个重要的对外提供接口的类名，如下表所示：

类名	说明
XGPushManager	Push 服务推送
XGPushConfig	Push 服务配置项接口
XGPushBaseReceiver	接收消息和结果反馈的 Receiver，需要开发者在 AndroidManifest.xml 自主完成静态注册

启动与注册

- App 只有在完成移动推送的启动与注册后才可以移动推送 SDK 提供 Push 服务，在这之前请确保配置 AccessId 和 AccessKey。
- 新版的 SDK 已经将启动移动推送和 App 注册统一集成在注册接口中，即只需调用注册接口便默认完成启动和注册操作。
- 注册成功后，会返回设备 Token，Token 用于标识设备唯一性，同时也是移动推送维持与后台连接的唯一身份标识。关于如何获取 Token 请参考 获取 Token 。

注册接口通常提供简版和带 callback 版本的接口，请根据业务需要决定选择接口。

设备注册

以下为设备注册相关接口方法，若需了解调用时机及调用原理，可查看 设备注册流程 。

> **说明：**建议您每次打开 App 时触发一次注册接口，当失败时会增加重试。

接口说明

普通注册只注册当前设备，后台能够针对不同的设备 Token 发送推送消息，以下有2个版本的 API 接口方法：

```
public static void registerPush(Context context)
```

参数说明

context：当前应用上下文对象，不能为 null。

示例代码

```
XGPushManager.registerPush(getApplicationContext());
```

接口说明

为方便用户获取注册是否成功的状态，提供带 callback 的版本。

```
public static void registerPush(Context context,final XGIOperateCallback callback)
```

参数说明

- context：当前应用上下文对象，不能为 null。
- callback：callback 调用，主要包括操作成功和失败的回调，不能为 null。

示例代码

```
XGPushManager.registerPush(this, new XGIOperateCallback() {
```

```
    @Override
    public void onSuccess(Object data, int flag) {
        Log.d("TPush", "注册成功，设备token为：" + data);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.d("TPush", "注册失败，错误码：" + errCode + ",错误信息：" + msg);
    }
})
```

获取注册结果

有2种途径可以获取注册是否成功。
使用 Callback 版本的注册接口。
XGIOperateCallback 类提供注册成功或失败的处理接口，请参考注册接口里面的示例。

示例代码

```
/**
* 操作回调接口
*/
public interface XGIOperateCallback {
    /**
    * 操作成功时的回调。
    * @param data 操作成功的业务数据，如注册成功时的token信息等。
    * @param flag 标记码
    */
    public void onSuccess(Object data, int flag);
    /**
    * 操作失败时的回调
    * @param data 操作失败的业务数据
    * @param errCode 错误码
    * @param msg 错误信息
    */
    public void onFail(Object data, int errCode, String msg);
}
```

继承 XGPushBaseReceiver
可通过重写 XGPushBaseReceiver 的 onRegisterResult 方法获取。

> **说明**
> 继承 XGPushBaseReceiver 的子类 需要配置在 AndroidManifest.xml，请参考下文 消息配置 。

示例代码

```
/**
*
* @param context 当前上下文
* @param errorCode 0 为成功，其它为错误码
* @param message 注册结果返回
*/
@Override
public void onRegisterResult(Context context, int errorCode, XGPushRegisterResult message) {
            if (context == null || message == null) {
                return;
            }
            String text = "";
            if (errorCode == XGPushBaseReceiver.SUCCESS) {        // 注册成功
                // 在这里拿token
```

```
                String token = message.getToken();
                text = "注册成功，token: " + token;
            } else {
                text = message + "注册失败，错误码: " + errorCode;
            }
            Log.d(LogTag, text);
}
```

类方法列表

方法名	返回值	默认值	描述
getToken()	String	无	设备的 Token，即设备唯一识别 ID
getAccessId()	long	0	获取注册的 AccessId
getAccount	String	无	获取注册绑定的账号
getTicket()	String	无	登录态票据
getTicketType()	short	0	票据类型

反注册

以下为反注册接口方法，若需了解调用时机及调用原理，可查看 设备反注册流程 。

> **注意**
> 调用反注册接口后，需要重新调用注册接口才可以接收到推送。

接口说明

当用户已退出或 App 被关闭，不再需要接收推送时，可以取消注册 App，即反注册（一旦设备反注册，直到这个设备重新注册成功期间内，下发的消息该设备都无法收到）。

```
public static void unregisterPush(Context context)
```

参数说明

context： App 的上下文对象。

示例代码

```
XGPushManager.unregisterPush(getApplicationContext(), new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int i) {
        Log.d("TPush", "反注册成功");
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.d("TPush", "反注册失败，错误码: " + errCode + ",错误信息: " + msg);
    }
});
```

获取反注册结果

可通过重写 XGPushBaseReceiver的onUnregisterResult 方法获取。

> **说明**
> - 反注册操作切勿过于频繁，可能会造成后台同步延时。

- 切换账号无需反注册，多次注册自动会以最后一次为准。

示例代码

```
/**
* 反注册结果
* @param context 当前上下文
* @param errorCode  为成功，其它为错误码
*/
@Override
public void onUnregisterResult(Context context, int errorCode) {
    if (context == null) {
           return;
     }
    String text = "";
    if (errorCode == XGPushBaseReceiver.SUCCESS) {
         text = "反注册成功";
    } else {
         text = "反注册失败" + errorCode;
    }
    Log.d(LogTag, text);
}
```

推送通知（展现在通知栏）

指的是在设备的通知栏展示的内容，由移动推送 SDK 完成所有的操作，App 可以监听通知被打开的行为，即在前台下发的通知，无需 App 做任何处理，默认会展示在通知栏。

说明

- 成功注册移动推送服务后，通常不需要任何设置便可下发通知。
- 通常来说，结合自定义通知样式，常规的通知，能够满足大部分业务需求，如果需要更灵活的方式，请考虑使用消息。

获取通知

接口说明

移动推送 SDK 提供回调接口供开发者获取抵达的通知内容，可以通过重写XGPushBaseReceiver 的 `onNotificationShowedResult(Context, XGPushShowedResult)` 方法实现。其中，XGPushShowedResult 对象提供读取通知内容的接口。

注意

因部分厂商通道 SDK 未提供通知抵达回调方法，且当 App 进程未运行时，厂商通道的抵达回调方法无法触发。因此 SDK 内提供的回调接口 `onNotificationShowedResult` 仅支持移动推送自建通道下发通知抵达的监听，不支持厂商通道消息抵达的监听。

```
public abstract void onNotificationShowedResult(Context context,XGPushShowedResult notifiShowedRlt);
```

参数说明

- context：当前应用上下文。
- notifiShowedRlt： 抵达的通知对象。

获取通知点击结果

通知回调监听和自定义参数解析

使用移动推送 SDK 默认已经统计通知/消息的抵达量、通知的点击和清除动作。SDK 提供回调接口供开发者监听通知点击事件，通过重写 XGPushBaseReceiver 的 `onNotificationClickedResult(Context, XGPushClickedResult)` 方法实现。

说明

- 自 SDK 版本 v1.2.0.1 起，支持各厂商通道、移动推送自建通道下发的通知点击事件的监听。
- 请不要在此回调接口内另外做页面跳转动作，SDK 会自动按照推送任务配置的跳转动作进行通知点击跳转。如需下发并获取推送自定义参数，推荐使用 Intent 方式，请参考文档 通知点击跳转 。

接口说明

```
public abstract void onNotificationClickedResult(Context context, XGPushClickedResult notifiClickedRlt);
```

示例代码

```
// 通知点击回调，actionType=0 为该消息被点击，actionType=2 为该消息被清除
@Override
public void onNotificationClickedResult(Context context, XGPushClickedResult message) {
    if (context == null || message == null) {
        return;
    }
    String text = "";
    if (message.getActionType() == NotificationAction.clicked.getType()) {
        // 通知在通知栏被点击
        // APP自己处理点击的相关动作
        text = "通知被打开 :" + message;
    } else if (message.getActionType() == NotificationAction.delete.getType()) {
        // 通知被清除
        // APP自己处理通知被清除后的相关动作
        text = "通知被清除 :" + message;
    }

    // APP自主处理的过程。

    Log.d(LogTag, "广播接收到通知:" + text);
}
```

参数说明

- context：当前应用上下文。
- XGPushClickedResult：被打开的通知对象。

`XGPushClickedResult` 类成员方法列表：

方法名	返回值	默认值	描述
getMsgId()	long	0	消息 ID
getTitle()	String	无	通知标题
getContent()	String	无	通知正文内容
getActionType()	String	无	• 0：表示该通知被点击 • 2：表示该通知被清除
getPushChannel()	String	100	被点击通知的所下发通道标识。 • 100：移动推送自建通道。 • 101：FCM 通道。 • 102：华为通道。 • 103：小米通道。 • 104：vivo 通道。 • 105：OPPO 通道。 • 106：魅族通道。

清除所有通知

接口说明

清除本 App 在通知栏上的所有通知。

```
public static void cancelAllNotifaction(Context context)
```

参数说明

context：Context 对象。

示例代码

```
XGPushManager.cancelAllNotifaction(context);
```

创建通知渠道

接口说明

开发者可以创建通知 channel。

```
public static void createNotificationChannel(Context context, String channelId, String channelName,
boolean enableVibration, boolean enableLights, boolean enableSound, Uri soundUri)
```

> **说明**
> 此接口仅适用于1.1.5.4及以上版本。

参数说明

- context：当前应用上下文。
- channelId：通知渠道 Id。
- channelName：通知渠道名称。
- enableVibration：是否震动。
- enableLights：是否有呼吸。
- enableSound：是否有铃声。
- soundUri：铃声资源 Uri，enableSound 为 true 才有效，若使用系统默认铃声，则设置为 null。

示例代码

```
// 请将铃声文件放置在 Android 工程资源目录 raw 下，此处以文件 ring.mp3  为例
String uri = "android.resource://" + context.getPackageName() + "/" + R.raw.ring;
Uri soundUri = Uri.parse(uri);
XGPushManager.createNotificationChannel(context,"default_message", "默认通知", true, true, true, soundUri);
```

推送透传消息（消息不展示到通知栏）

指的是由移动推送下发给 App 的内容，需要 App 继承 XGPushBaseReceiver 接口实现并自主处理所有操作过程，也就是说，下发的消息默认是不会展示在通知栏的，移动推送只负责将消息从移动推送服务器下发到 App 这个过程，不负责消息的处理逻辑，需要 App 自己实现。

- 消息指的是由开发者通过前台或后台脚本下发的文本消息，移动推送只负责将消息传递给 App，App 完全自主负责消息体的处理。
- 消息具有灵活性强和高度定制性的特点，更适合 App 自主处理个性化业务需求，例如下发 App 配置信息、自定义处理消息的存储和展示等。

消息配置

请自行继承 XGPushBaseReceiver ，并且在配置文件中配置如下内容：

示例代码

```
<receiver android:name="com.tencent.android.xg.cloud.demo.MessageReceiver">
    <intent-filter>
        <!-- 接收消息透传 -->
        <action android:name="com.tencent.android.xg.vip.action.PUSH_MESSAGE" />
        <!-- 监听注册、反注册、设置/删除标签、通知被点击等处理结果 -->
        <action android:name="com.tencent.android.xg.vip.action.FEEDBACK" />
    </intent-filter>
</receiver>
```

获取透传消息

开发者在前台下发消息，需要 App 继承 XGPushBaseReceiver 重写 onTextMessage 方法接收，成功接收后，再根据特有业务场景进行处理。

> **说明**
> 请确保在 AndroidManifest.xml 已经注册过该 receiver，即设 YOUR_PACKAGE.XGPushBaseReceiver。

```
public void onTextMessage(Context context,XGPushTextMessage message)
```

参数说明

- context：应用当前上下文。
- message：接收到消息结构体。

类方法列表

方法名	返回值	默认值	描述
getContent()	String	无	消息正文内容，通常只需要下发本字段即可
getCustomContent()	String	无	消息自定义 key-value
getTitle()	String	无	消息标题（从前台下发应用内消息字中的描述不属于标题）

本地通知

增加本地通知

本地通知由用户自定义设置，保存在本地。当应用打开，移动推送 SDK Service 会根据网络心跳，判断当前是否有通知（5分钟一次），本地通知需要 Service 开启才能弹出，可能存在5分钟左右延时。（当设置的时间小于当前设备时间通知弹出）

示例代码

```
//新建本地通知
XGLocalMessage local_msg = new XGLocalMessage();
//设置本地消息类型，1:通知，2:消息
local_msg.setType(1);
// 设置消息标题
local_msg.setTitle("qq");
//设置消息内容
local_msg.setContent("ww");
//设置消息日期，格式为：20140502
local_msg.setDate("20140930");
//设置消息触发的小时(24小时制)，例如：22代表晚上10点
local_msg.setHour("19");
//获取消息触发的分钟，例如：05代表05分
local_msg.setMin("31");
//设置消息样式，默认为0或不设置
local_msg.setBuilderId(0);
```

```
//设置动作类型：1打开activity或App本身，2打开浏览器，3打开Intent ，4通过包名打开应用
local_msg.setAction_type(1);
//设置拉起应用页面
local_msg.setActivity("com.qq.xgdemo.SettingActivity");
// 设置URL
local_msg.setUrl("http://www.baidu.com");
// 设置Intent
local_msg.setIntent("intent:10086#Intent;scheme=tel;action=android.intent.action.DIAL;S.key=value;end");
// 是否覆盖原先build_id的保存设置。1覆盖，0不覆盖
local_msg.setStyle_id(1);
// 设置音频资源
local_msg.setRing_raw("mm");
// 设置key,value
HashMap<String, Object> map = new HashMap<String, Object>();
map.put("key", "v1");
map.put("key2", "v2");
local_msg.setCustomContent(map);
//添加通知到本地
XGPushManager.addLocalNotification(context,local_msg);
```

清除本地通知

接口说明

清除本 App 已经创建但未弹出的本地通知。

```
public static void clearLocalNotifications(Context context)
```

参数说明

context：Context 对象。

示例代码

```
XGPushManager.clearLocalNotifications(context);
```

账号管理

以下为账号管理相关接口方法，若需了解调用时机及调用原理，可查看 账号相关流程 。

添加账号

接口说明

添加或更新账号。若原来没有该类型账号，则添加；若原来有，则覆盖。可以同时添加多个账号，一个账号对应一个账号类型。

```
public static void upsertAccounts(Context context, List<AccountInfo> accountInfoList, XGIOperateCallback
callback)
```

参数说明

- context： Context 对象。
- accountInfoList： 账号列表：账号信息包含一个账号类型和账号名称。
- callback： 绑定账号操作的回调。

示例代码

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
```

```
        Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};
List<XGPushManager.AccountInfo> accountInfoList = new ArrayList<>();
accountInfoList.add(new XGPushManager.AccountInfo(XGPushManager.AccountType.UNKNOWN.getValue(), "account-
test"));
XGPushManager.upsertAccounts(context, accountInfoList, xgiOperateCallback);
```

说明

- 每个账号最多支持绑定100个 token。
- 账号可以是邮箱、QQ 号、手机号、用户名等任意类别的业务账号，账号类型取值可参考 账号类型取值表 。
- 同一个账号绑定多个设备时，后台将默认推送消息到最后绑定的设备，如需推送所有绑定的设备可查看 Rest API 文档中 account_push_type 参数设置。

添加手机号

接口说明

添加或更新手机号码。若原来没有绑定手机号码，则绑定；若原来有，则覆盖（SDK 1.2.5.0+）

说明

手机号格式为 `+[国家或地区码][手机号]` ，例如+8613711112222（其中前面有一个+号 ，86为国家或地区码，13711112222为手机号）。若绑定的手机号不带**国家或地区码**，则移动推送下发短信时自动增加+86的前缀；若带上**国家或地区码**，则按照指定的号码绑定。如需删除绑定的手机号，则需调用 `delAccountsByKeys` 接口并设置 `accountTypeSet` 为 `1002` 。

```
public static void upsertPhoneNumber(Context context, String phoneNumber, XGIOperateCallback callback)
```

参数说明

- context： Context 对象。
- phoneNumber：phoneNumber E.164标准，格式为+[国家或地区码][手机号]，例如+8613711112222。SDK 内部加密传输。
- callback： 绑定手机号操作的回调。

示例代码

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
  }
   @Override
  public void onFail(Object data, int errCode, String msg) {
      Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
  }
};
XGPushManager.upsertPhoneNumber(context, phoneNumber, xgiOperateCallback);
```

账号解绑

接口说明

对已绑定的账号进行解绑。

```
//解绑指定账号（有注册回调）
void delAccount(Context context, final String account, XGIOperateCallback callback)
//解绑指定账号（无注册回调）
void delAccount(Context context, final String account )
```

> **说明**
> 账号解绑只是解除 Token 与 App 账号的关联，若使用全量/标签/Token 推送，仍然能收到通知/消息。

参数说明

- context：当前应用上下文对象，不能为 null。
- account：账号。

示例代码

```
XGPushManager.delAccount(getApplicationContext(),"test");
```

账号类型解绑

接口说明

对一个或多个账号类型的账号进行解绑。（SDK 1.2.3.0+）

```
public static void delAccounts(Context context, final Set<Integer> accountTypeSet, XGIOperateCallback
callback)
```

参数说明

- context： Context 对象。
- accountTypeSet： 需解绑账号的账号类型。
- callback： 账号解绑操作的回调。

示例代码

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};

Set<Integer> accountTypeSet = new HashSet<>();
accountTypeSet.add(XGPushManager.AccountType.CUSTOM.getValue());
accountTypeSet.add(XGPushManager.AccountType.IMEI.getValue());
XGPushManager.delAccounts(context, accountTypeSet, xgiOperateCallback);
```

清空所有账号

> **说明**
> SDK 1.2.2.0 版本废弃 delAllAccount 接口，推荐使用 clearAccounts 接口。

接口说明

对的所有已绑定账号进行解绑。

```
//解绑所有的账号信息（有注册回调）
void clearAccounts(Context context, XGIOperateCallback callback)
//解绑所有的账号信息（无注册回调）
void clearAccounts(Context context)
```

> **说明**
> 账号解绑只是解除 Token 与 App 账号的关联，若使用全量/标签/Token 推送，仍然能收到通知/消息。

参数说明

context：当前应用上下文对象，不能为 null。

示例代码

```
XGPushManager.clearAccounts(getApplicationContext());
```

标签管理

以下为标签管理相关接口方法，若需了解调用时机及调用原理，可查看 标签相关流程 。

预设标签

目前移动推送平台提供的预设标签包括：App 版本，系统版本，省份，活跃信息，系统语言，SDK 版本，国家&地区，手机品牌，手机机型。预设标签会在 SDK 内部自动上报。

覆盖多个标签

接口说明

一次设置多个标签，会覆盖这个设备之前设置的标签。
开发者可以针对不同的用户设置标签，然后根据标签名群发通知。一个应用最多有10000个 tag，每个 Token 在一个应用下最多100个 tag，如需提高该限制，请联系 在线客服 。每个自定义 tag 可绑定的设备 Token 数量无限制，tag 中不准包含空格。

```
public static void clearAndAppendTags(Context context, String operateName, Set<String> tags)
```

参数说明

- context：Context 对象。
- operateName：用户定义的操作名称，回调结果会原样返回，用于标识回调属于哪次操作。
- tags：标签名集合，每个标签是一个 String。限制：每个 tag 不能超过50字节（超过会抛弃），不能包含空格（含有空格会删除空格）。最多设置100个 tag，超过部分会抛弃。

处理结果

可通过重写 XGPushBaseReceiver的onSetTagResult 方法获取。

示例代码

```
String[] tags = "tag1 tag2".split(" ");
Set<String> tagsSet = new HashSet<>(Arrays.asList(tags));
XGPushManager.clearAndAppendTags(getApplicationContext(), "clearAndAppendTags :" +
System.currentTimeMillis(), tagsSet);
```

新增多个标签

> **说明**

SDK 1.2.2.0 版本废弃 addTags 接口，推荐使用 appendTags 接口。

接口说明

- 如果新覆盖的标签都带有 `:` 号，例如 `test:2, level:2`，则会删除这个设备已绑定的所有 `test:*` 和 `level:*` 标签，再新增 `test:2` 和 `level:2` 。
- 如果新增的标签有部分不带 `:` 号，例如 `test:2  level`，则会删除这个设备的全部历史标签，再新增 `test:2` 和 `level` 标签。

> **说明**
> 新增的 tags 中， `:` 号为后台关键字，请根据具体的业务场景使用。

- 此接口调用的时候需要间隔一段时间（建议大于5s），否则可能造成更新失败。

```
public static void appendTags(Context context, String operateName, Set<String> tags)
```

参数说明

- context：Context 对象。
- operateName：用户定义的操作名称，回调结果会原样返回，用于标识回调属于哪次操作。
- tags：标签名集合，每个标签是一个 String。限制：每个 tag 不能超过50字节（超过会抛弃），不能包含空格（含有空格会删除空格）。最多设置100个 tag，超过部分会抛弃。

处理结果

可通过重写 XGPushBaseReceiver的onSetTagResult 方法获取。

示例代码

```
String[] tags = "tag1 tag2".split(" ");
Set<String> tagsSet = new HashSet<>(Arrays.asList(tags));
XGPushManager.appendTags(getApplicationContext(), "appendTags:" + System.currentTimeMillis(), tagsSet);
```

删除多个标签

> **说明**
> SDK 1.2.2.0 版本废弃 deleteTags 接口，推荐使用 delTags 接口。

接口说明

一次删除多个标签。

```
public static void delTags(Context context, String operateName, Set<String> tags, XGIOperateCallback
callback)
```

参数说明

- context：Context 对象。
- operateName：用户定义的操作名称，回调结果会原样返回，用于标识回调属于哪次操作。
- tags：标签名集合，每个标签是一个 String。限制：每个 tag 不能超过50字节（超过会抛弃），不能包含空格（含有空格会删除空格）。最多设置100个 tag，超过部分会抛弃。
- callback： 删除标签操作的回调

处理结果

可通过重写 XGPushBaseReceiver 的 onSetTagResult 方法获取。

示例代码

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};

Set<String> tagSet = new HashSet<>();
tagSet.add("tag1");
tagSet.add("tag2");
XGPushManager.delTags(context, "delTags", tagSet, xgiOperateCallback);
```

清除所有标签

> **说明**
> SDK 1.2.2.0 版本开始废弃 cleanTags 接口，推荐使用 clearTags 接口。

接口说明

清除这个设备的所有标签。

```
public static void clearTags(Context context, String operateName, XGIOperateCallback callback)
```

参数说明

- context：Context 对象。
- operateName：用户定义的操作名称，回调结果会原样返回，用于标识回调属于哪次操作。
- callback： 清理所有标签操作的回调。

处理结果

可通过重写 XGPushBaseReceiver 的 onSetTagResult 方法获取。

示例代码

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};

XGPushManager.clearTags(context, "clearTags", xgiOperateCallback);
```

查询标签

> **说明**
> 查询设备关联的标签（此接口仅适用于1.2.5.0及以上版本）。

接口说明

获取这个设备的标签。

```
    public static void queryTags(final Context context, final String operateName, final int offset, final
int limit, final XGIOperateCallback callback)
```

参数说明

- context：Context 对象
- operateName：用户定义的操作名称，回调结果会原样返回，用于给用户区分是哪个操作
- offset：开始的位置
- limit：获取标签的数量，最多为100个
- callback：获取标签操作的回调

处理结果

可通过重写 XGPushBaseReceiver 的 onQueryTagsResult 方法获取。

示例代码

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};
XGPushManager.queryTags(context, 0, 100, xgiOperateCallback);
```

用户属性管理

开发者可以针对不同的用户设置属性，然后在管理平台推送的时候进行个性化推送。以下为用户属性相关接口方法，若需了解调用时机及调用原理，可查看 用户属性相关流程 。

新增用户属性

接口说明

添加属性（带回调）：有则覆盖，无则添加。

```
public static void upsertAttributes(Context context, String operateName, Map<String, String> attributes,
XGIOperateCallback callback)
```

参数说明

- context：Context 对象。
- operateName：用户定义的操作名称，回调结果会原样返回，用于给用户区分是哪个操作。
- attributes：属性集合，每个属性通过 key-value 标识。
- callback：添加属性操作的回调。

> ⚠ 注意
> - 属性使用键值对传输，都只接受 string 字符串类型，非空串。
> - 属性个数限制50个。
> - 属性 key，value 长度都限制50个字符以内。

示例代码

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        log("action - onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        log("action - onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};

Map<String,String> attr = new HashMap<>();
attr.put("name", "coding-test");
attr.put("gender", "male");
attr.put("age", "100");
XGPushManager.upsertAttributes(context, "addAttributes-test", attr, xgiOperateCallback);
```

删除用户属性

接口说明

删除指定的属性。

```
public static void delAttributes(Context context, String operateName, Set<String> attributes,
XGIOperateCallback callback)
```

参数说明

- context：Context 对象。
- operateName：用户定义的操作名称，回调结果会原样返回，用于给用户区分是哪个操作。
- attributes：属性集合，每个属性通过 key-value 标识。
- callback：删除属性操作的回调。

> ⚠ 注意
> - 属性使用键值对传输，都只接受 string 字符串类型，非空串。
> - 属性个数限制50个。
> - 属性 key，value 长度都限制50个字符以内。

示例代码

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        log("action - onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        log("action - onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};
Set<String> stringSet = new HashSet<>();
stringSet.add("name");
stringSet.add("gender");

XGPushManager.delAttributes(context, "delAttributes-test", stringSet, xgiOperateCallback);
```

清空已有用户属性

接口说明

删除已设置的所有属性。

```
public static void clearAttributes(Context context, String operateName, XGIOperateCallback callback)
```

参数说明

- context：Context 对象。
- operateName：用户定义的操作名称，回调结果会原样返回，用于给用户区分是哪个操作。
- callback：清理所有属性操作的回调。

示例代码

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        log("action - onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        log("action - onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};

XGPushManager.clearAttributes(context, "cleanAttributes-test", xgiOperateCallback);
```

更新用户属性

接口说明

设置属性（带回调），会覆盖这个设备之前设置的所有属性（即清理并设置）。

> ⚠ 注意
> - 属性使用键值对传输，都只接受 string 字符串类型，非空串。
> - 属性个数限制50个。
> - 属性 key，value 长度都限制50个字符以内。

```
public static void clearAndAppendAttributes(Context context, String operateName, Map<String, String>
attributes, XGIOperateCallback callback)
```

参数说明

- context：Context 对象。
- operateName：用户定义的操作名称，回调结果会原样返回，用于给用户区分是哪个操作。
- attributes：属性集合，每个属性通过 key-value 标识。
- callback：设置属性操作的回调。

示例代码

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        log("action - onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
```

```
        log("action - onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};

Map<String,String> attr = new HashMap<>();
attr.put("name", "coding-test");
attr.put("gender", "male");
attr.put("age", "100");
XGPushManager.clearAndAppendAttributes(context, "setAttributes-test", attr, xgiOperateCallback);
```

配置接口

所有的配置相关接口在 XGPushConfig 类中，为了使配置及时生效，开发者需要保证配置接口在启动或注册移动推送之前被调用。

关闭联合保活能力（1.1.6.1+）

移动推送默认开启联合保活能力，若需要关闭联合保活能力，请在应用初始化的时候，例如 Application 或 LauncherActivity 的 onCreate 中调用如下接口，并传递 false。

```
XGPushConfig.enablePullUpOtherApp(Context context, boolean pullUp);
```

> **说明**
> 1.2.6.0 起默认关闭联合保活功能，可不再调用此接口。在 SDK 1.3.4.0 及以上的版本关联启动功能已下线。

参数说明

- context：应用上下文
- pullUp：true（开启联合保活）；false（关闭联合保活）

> **说明**
> 若有以下日志打印，则表明联合保活功能已经关闭：I/TPNS: [ServiceUtil] disable pull up other app。

示例代码

```
XGPushConfig.enablePullUpOtherApp(context, false); // 默认为 true: 开启保活
```

Debug 模式

接口说明

为保证数据的安全性，请在发布时确保已关闭 Debug 模式。

```
public static void enableDebug(Context context, boolean debugMode)
```

参数说明

- context：App 上下文对象。
- debugMode：默认为 false。如果要开启 Debug 日志，设为 true。

示例代码

```
XGPushConfig.enableDebug(context, true); // 默认为 false: 不打开
```

获取设备 Token

接口说明

Token 是移动推送保持与后台长连接的唯一身份标识，是 App 接收消息的唯一 ID，只有设备注册成功后才能获取 Token，获取方法如下。（移动推送的 Token 在应用卸载重新安装的时候有可能会变。）

1. 通过带 callback 的注册接口获取

带 XGIOperateCallback 的注册接口的 onSuccess(Object data, int flag) 方法中，参数 data 便是 Token，具体可参考注册接口的相关示例。

2. 继承 XGPushBaseReceiver

重写 XGPushBaseReceiver 的 onRegisterResult (Context context, int errorCode,XGPushRegisterResult registerMessage) 方法，通过参数 registerMessage 提供的 getToken 接口获取，具体请参见 获取注册结果 章节。

3. XGPushConfig.getToken(context)

当设备一旦注册成功后，便会将 Token 存储在本地，之后可通过 XGPushConfig.getToken(context) 接口获取。
Token 是一个设备的身份识别 ID，由服务器根据设备属性随机产生并下发到本地，同一 App 在不同设备上的 Token 不同。

```
public static String getToken(Context context)
```

> **说明**
> App 第一次注册会产生 Token，之后一直存储在手机上，不论之后是否进行注销注册操作，该 Token 一直存在。当 App 完全卸载重装后，Token 会发生变化。不同 App 之间的 Token 不同。

参数说明

context：App 上下文对象。

示例代码

```
XGPushConfig.getToken(context);
```

返回值

成功时返回正常的 Token；失败时返回 null 或0。

获取第三方厂商 Token

接口说明

第三方厂商 Token 是厂商设备的身份识别 ID，由厂商下发到本地，同一 App 在不同设备上的 Token 不同。

```
public static String getOtherPushToken(Context context)
```

> **说明**
> 需要注册成功之后才能调用，不然返回为 NULL。

参数说明

context：App 上下文对象。

示例代码

```
XGPushConfig.getOtherPushToken(context);
```

返回值

成功时返回正常的 Token；失败时返回 null 或0。

在通知点击目标页面内获取随通知下发的自定义参数（custom_content）内容

SDK 1.3.2.0 新增，当通知被点击打开时，可以在通知设置的目标页面内，通过此接口直接获取创建推送任务时配置的自定义参数（custom_content）内容；
详细使用方式参见 通知点击跳转 。

```
public static String getCustomContentFromIntent(Context context, Intent intent)
```

返回值

随推送下发的自定义参数（custom_content）字符串

参数说明

- context：Context 对象， 不能为 null。
- intent：activity intent；在 onCreate 内请直接传入 this.getIntent()；在 onNewIntent 内请直接传入回调的 intent。

示例代码

```
String customContent = XGPushManager.getCustomContentFromIntent(this, intent);
```

设置 AccessID

接口说明

如果已在 AndroidManifest.xml 配置过，无需再次调用；如果二者都存在，则以本接口为准。

```
public static boolean setAccessId(Context context, long accessId)
```

参数说明

- Context：对象。
- accessId：前台注册得到的 accessId。

示例代码

```
long accessId = 0L; // 当前应用的 accessId
XGPushConfig.setAccessId(context, accessId);
```

返回值

- true：成功。
- false：失败。

> **说明**
> 通过本接口设置的 accessId 会同时存储在文件中。

设置 AccessKey

接口说明

如果已在 AndroidManifest.xml 配置过，无需再次调用；如果二者都存在，则以本接口为准。

```
public static boolean setAccessKey(Context context, String accessKey)
```

参数说明

- Context：对象。
- accessKey：前台注册得到的 accesskey。

示例代码

```
String accessKey = ""; // 您应用的 accessKey
XGPushConfig.setAccessKey(context, accessKey);
```

返回值

- true：成功。
- false：失败。

> **说明**
> 通过本接口设置的 accessKey 会同时存储在文件中。

新增日志上报接口

接口说明

开发者如果发现 TPush 相关功能异常，可以调用该接口，触发本地 Push 日志的上报，反馈问题时，请联系 在线客服 将文件地址给到我们，便于我们排查问题。

```
public static void uploadLogFile(Context context, HttpRequestCallback httpRequestCallback)
```

参数说明

- context：Context 对象， 不能为 null。
- httpRequestCallback： 上传日志结果回调，主要包括操作成功和失败，不能为 null。

示例代码

```
XGPushManager.uploadLogFile(context, new HttpRequestCallback() {
    @Override
    public void onSuccess(String result) {
            Log.d("TPush", "上传成功，文件地址：" + result);
    }
    @Override
    public void onFailure(int errCode, String errMsg) {
            Log.d("TPush", "上传失败，错误码：" + errCode + ",错误信息：" + errMsg);
    }
});
```

> **说明**
> 首先需要开启 `XGPushConfig.enableDebug(this, true);` 。

通知分类说明

最近更新时间：2025-01-14 15:26:52

简介

目前，厂商逐步对 App 开发者的本地通知根据分类进行限额限频，也以此保证终端用户不被过度骚扰，不同的消息分类主要通过通知 Category 进行区分。移动推送结合厂商的要求，对于本地通知和自建通道下发通知增加了通知分类和 importance 的支持。

> **注意：**
> 从2023年9月15日开始，华为推送服务将对本地通知进行灰度管控，主要包括对应用发送本地通知进行分类管理，以及对资讯营销消息统一进行频次管控，有需求可以参见《 华为本地通知配置指引 》，未接入消息自分类的应用，消息通知类型将会默认归为资讯营销类消息，资讯营销消息每日推送数量上限要求请参见《 推送数量管理细则 》。

> **说明：**
> 华为本地通知包含移动推送本地通知和移动推送自建通道下发的通知。

通知消息的分类

移动推送的通知主要来自三种：

通知来源	备注
本地通知	通过移动推送 SDK 接口直接弹出的通知
移动推送自建通道下发通知	通过移动推送自建通道下发的推送通知
移动推送下发给厂商通道展示的通知	移动推送下发给厂商通道展示出来的通知， 厂商通道消息分类参见：《 厂商消息分类功能使用说明 》

本地通知支持分类

移动推送 Android SDK 1.4.3.1 版本在增加本地通知时，可以指定通知 category 和通知 channel importance。
其中 Category: 具体取值可参见 Android Notification 中相关 Category 的定义，具体分类参见《 Android 官方通知分类定义 》。
代码示例：

```
XGLocalMessage localMessage = new XGLocalMessage();
//设置通知类型: 1 通知, 2 透传消息
localMessage.setType(1);
localMessage.setTitle("一条本地通知");
localMessage.setContent("我是一条本地通知");
localMessage.setDate("20230729");
localMessage.setHour("19");
localMessage.setMin("31");
localMessage.setAction_type(4);
localMessage.setChannelId("local_123");
//设置本地通知分类
localMessage.setNotificationCategory(Notification.CATEGORY_CALL);
//设置通知Channel importance
localMessage.setNotificationImportance(NotificationManager.IMPORTANCE_HIGH);
localMessage.setActivity("com.tencent.android.duoduo.MainActivity");
localMessage.setIntent("tpns://com.tpns.push/notify_detail?param1=key");
HashMap<String, Object> custom = new HashMap<String, Object>();
custom.put("key", "v1");
custom.put("key2", "v2");
localMessage.setCustomContent(custom);
XGPushManager.addLocalNotification(context, localMessage);
```

移动推送自建通道下发通知支持分类

对于移动推送自建通道下发的推送通知，在 Rest API 请求参数 Android 结构体中设置 n_category 参数，可实现移动推送通道消息分类下发，详情请参见 PushAPI 参数说明。

> ⚠ **注意：**
> 移动推送 Android SDK 1.4.3.1及以上开始支持 tpns 自建通道通知分类。

n_category 字段的取值及参考使用场景说明：

参考类型	Category取值	参考场景说明
即时聊天	CATEGORY_MESSAGE	用户间点对点聊天消息（或私信）、群聊天消息。
音频、视频通话	CATEGORY_CALL	语音通话邀请、视频通话邀请、来电提醒等。
出行	CATEGORY_NAVIGATION	用户出行产生的通知提醒，推送对象为消费者。 • 行程提醒。 • 班车/航班变动提醒。 • 酒店入住前提醒。 • 公交到站提醒。
工作事项提醒	CATEGORY_REMINDER	用户下一步需要做某件事项的提醒、待处理的业务流程。 • 工作提醒：会议提醒、待办提醒、日程安排、教学任务/课程提醒等。 • 待处理业务流程，推送对象为服务提供方：审核进度提醒、认证状态流程提醒、工单处理、卖家收到订单提醒、卖家收到售后提醒、催促卖家发货提醒、司机接单提醒。 • 商家运营：库存不足、售罄提醒、商品下架通知、限制提现、客诉警告、店铺限制、商品黑名单、交易违规、涉假/涉欺诈发货通知。
财务	CATEGORY_SERVICE	金额变化的交易提示，仅限金融银行类、支付类 App 使用。 收付款、银行到账&扣款、交易提醒、催缴&退款信息、充值、待支付账单、贷款受理进度、还款/逾期提醒、资金冻结提醒、资金限制提醒、缴纳保证金提醒。
邮件	CATEGORY_EMAIL	新收到的邮件，例如邮箱类 App、办公软件 App 使用。
闹钟/计时器	CATEGORY_ALARM	闹钟提醒、计时器消息。
秒表	CATEGORY_STOPWATCH	秒表计时提醒。
进度	CATEGORY_PROGRESS	后台运行中的任务处理进展，如文件下载进度、传输进度等。
位置共享	CATEGORY_LOCATION_SHARING	临时共享位置信息。
内容推荐	CATEGORY_RECOMMENDATION	非用户主动订阅，应用向用户推送的内容。 包括点评、书籍、广告、视频、音频、节目、课程、直播、社区话题、游戏宣传等。
新闻		及时地报道新近发生的、有价值的事实。 包括新闻、经济新闻、法律新闻、新闻、科技新闻、文教新闻、体育新闻、社会新闻等。
财经动态		股票、彩票、期货、期权、外汇类通知，包括交易信息、行业公告等。
生活资讯		• 天气：天气提醒、天气异常变化，包括气温、雾霾、台风、灾情、预警等。 • 导航：行驶路线、路况规划、路线中的交通状况（拥堵提醒）、位置使用、调用地图类应用进行定位等相关的通知。 • 各垂直行业相关信息。
社交动态	CATEGORY_SOCIAL	用户之间的社交互动提醒，如：好友动态、新增粉丝、添加好友、被赞、被@、被收藏、评论、留言、关注、回复、转发等。 用户推荐：附近的人、大 V、主播、异性、可能认识的人等。

调研	CATEGORY_PROMO	发送问卷以获得受访者的态度和意见，包括使用习惯、产品满意度、服务满意度等。
其他		面向广大用户发布的产品功能推荐、平台公告、应用更新升级提醒等。
产品促销		产品信息相关推广、产品优惠，例如满减、低至、促销、买一送一、返利、优惠券、代金券、送红包相关的通知。
功能推荐		推荐用户使用当前产品的某一个功能。
运营活动		非用户主动设置，由应用发起需由用户参与的运营活动、提醒消息、小游戏提醒、服务等。

代码示例：

```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a***2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
     "title": "移动推送自建通道通知分类",
     "content": "测试内容",
     "android": {
            "n_category":"CATEGORY_CALL",
            "n_importance":3
     }
   }
}
```

移动推送下发给厂商通道展示的通知支持消息分类

目前，厂商会逐步对 App 开发者的通知消息根据分类进行限额限频，以此保证终端用户不被过度骚扰；
各厂商通道消息分类请参见《 厂商消息分类功能使用说明 》

通知 Channel优先级 importance

在 Rest API 请求参数 Android 结构体中设置 n_importance 参数，可实现移动推送通道消息分类下发，详情请参见 PushAPI 参数说明。
参考值：

```
1 //对应android IMPORTANCE_MIN
2 //对应android IMPORTANCE_LOW
3 //对应android IMPORTANCE_DEFAULT
4 //对应android IMPORTANCE_HEIGHT
```

代码示例：

```
{
    "audience_type": "token",
    "token_list": ["******"],
    "message_type": "notify",
    "message": {
     "title": "移动推送自建通道通知Channel优先级",
     "content": "测试内容",
     "android": {
            "n_category":"CATEGORY_CALL",
            "n_importance":3
     }
   }
}
```

> ⚠ **注意：**

n_importance 仅在首次创建通知的 Channel 时生效，通过移动推送的本地通知和自建通道下发的通知创建通知 Channel 支持设置优先级；如应用本地已经创建好了对应 Channel ID的 Channel 则 n_importance 字段不生效。

适配华为本地通知分类

从2023年9月15日开始，华为推送服务将对本地通知进行灰度管控，主要包括对应用发送本地通知进行分类管理，以及对资讯营销消息统一进行频次管控。本地通知的消息提醒方式取决于importance级别，应用需根据《华为本地通知分类和优先级规范》正确配置；
华为通知（云端通知和本地通知）的消息分类方式统一为消息自分类。如未申请自分类默认通知都为公信消息（有限额限频），如您希望消息分类能更精准地符合业务需要，可申请自分类权益。

注意：
- 应用若已申请云端相关消息类型，将自动生效对应的本地通知类型，本地通知无需重复申请，但应用仍需适配开发。
- 未接入消息自分类的应用，消息通知类型将会默认归为资讯营销类消息，资讯营销消息每日推送数量上限要求请参见《推送数量管理细则》。

申请华为自分类权益成功后，在集成移动推送 Android SDK 版本为1.4.3.1及以上，可对移动推送通道下发的通知和移动推送 Android SDK 触发的本地通知配置申请的本地通知分类，即可完成适配。

1. 移动推送通知下发的通知可以参见：《配置移动推送通道下发通知分类》

   示例：
   在移动推送 Rest API 请求参数 Android 结构体中设置 n_category 和 n_importance参 数，即可对通过移动推送自建通道的通知适配华为本地通知分类，详情请参见 PushAPI 参数说明。

   注意：
   n_importance 仅在首次创建通知的 Channel 时生效，通过移动推送的本地通知和自建通道下发的通知创建通知 Channel 支持设置优先级；如应用本地已经创建好了对应 Channel ID 的 Channel 则 n_importance 字段不生效。

```
{
    "audience_type": "token",
    "token_list": ["******"],
    "message_type": "notify",
    "message": {
        "title": "华为本地通知消息",
        "content": "自分类推送。",
        "android": {
            "n_category":"CATEGORY_PROMO",
            "n_importance":1
        }
    }
}
```

2. 移动推送本地通知触发的通知可以参见：《配置移动推送 SDK 触发本地通知分类》

   代码示例：

```
XGLocalMessage localMessage = new XGLocalMessage();
//设置通知类型：1 通知，2 透传消息
localMessage.setType(1);
localMessage.setTitle("一条本地通知");
localMessage.setContent("我是一条本地通知");
localMessage.setDate("20230729");
localMessage.setHour("19");
localMessage.setMin("31");
localMessage.setAction_type(4);
localMessage.setChannelId("local_123");
//设置本地通知分类
localMessage.setNotificationCategory(Notification.CATEGORY_CALL);
```

```
//设置通知Channel importance
localMessage.setNotificationImportance(NotificationManager.IMPORTANCE_HIGH);
localMessage.setActivity("com.tencent.android.duoduo.MainActivity");
localMessage.setIntent("tpns://com.tpns.push/notify_detail?param1=key");
HashMap<String, Object> custom = new HashMap<String, Object>();
custom.put("key", "v1");
custom.put("key2", "v2");
localMessage.setCustomContent(custom);
XGPushManager.addLocalNotification(context, localMessage);
```

厂商通道接入指南
华为通道 V5 接入

最近更新时间：2025-01-21 17:13:41

操作场景

移动推送跟进各厂商通道推送服务的更新进度，提供集成华为推送 HMS Core Push SDK 的插件依赖包供用户选择使用。

> **注意**
> - 华为推送只有在签名发布包环境下，才可注册厂商通道成功并通过厂商通道进行推送。
> - 华为通道不支持抵达回调，支持点击回调。

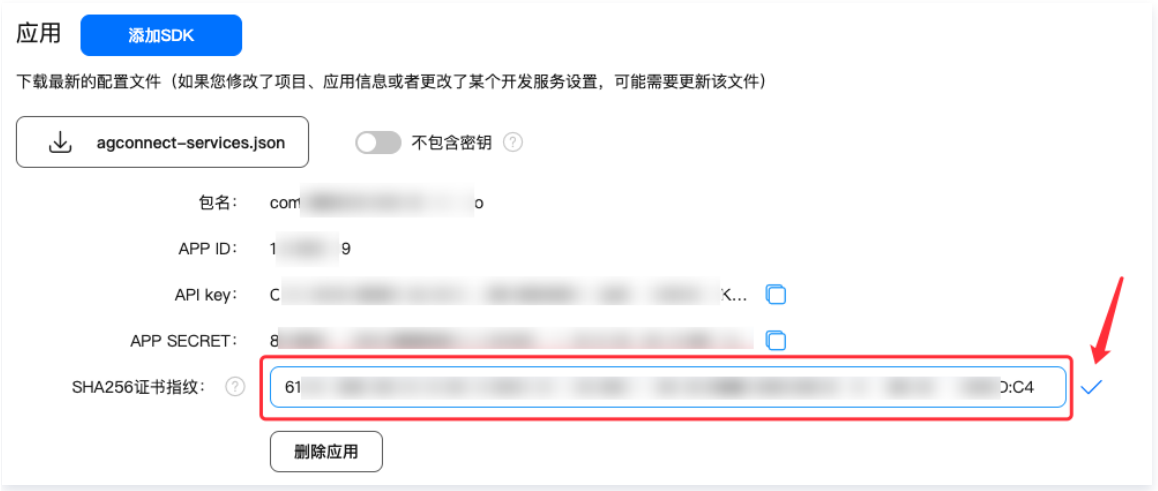
在华为推送平台配置应用

获取密钥

1. 进入 华为开放平台 。
2. 注册和登录开发者账号，详情参见 账号注册认证（如果您是新注册账号，需进行实名认证）。
3. 在华为推送平台中新建应用，详情参见 创建应用（应用包名需跟您在移动推送平台填写的一致）。
4. 进入**我的项目** > **项目设置** > **常规**中的应用获取并复制APPID和Client Secret，填入 移动推送控制台 > **App 推送管理** > **基础配置** > **华为官方推送通道**栏目的字段AppId、SecretKey中。
5. 

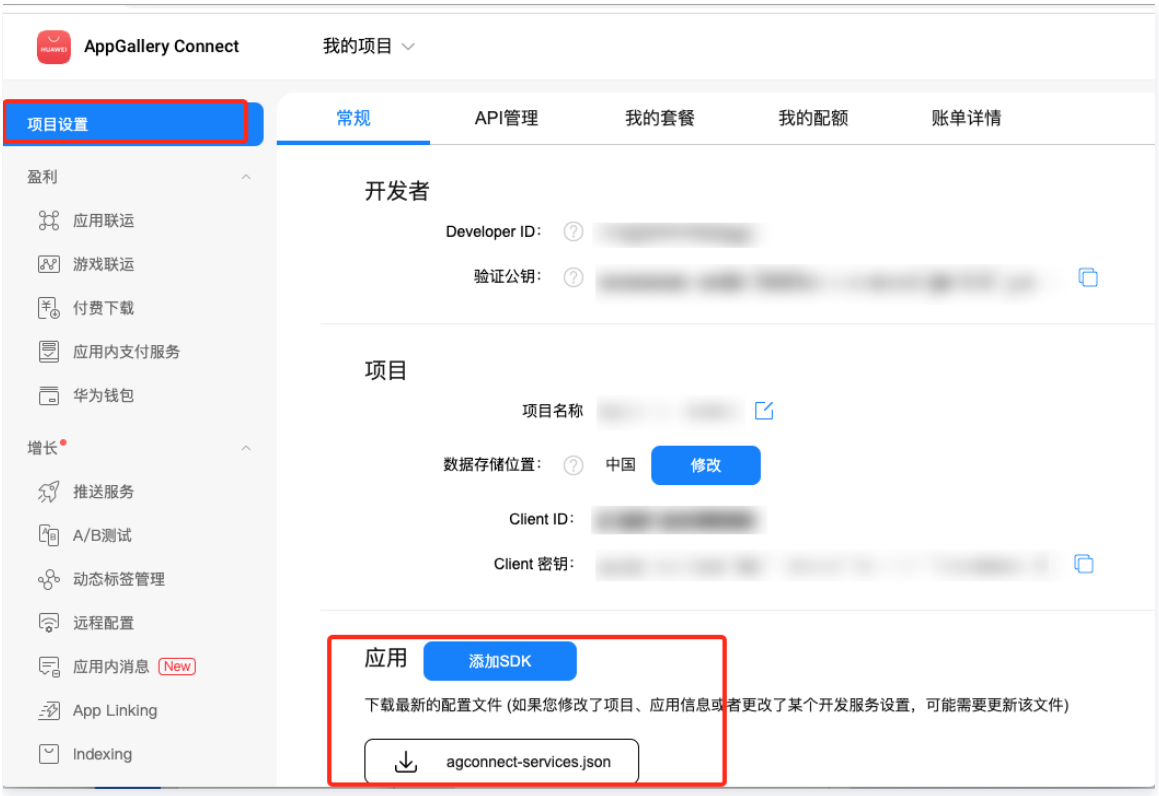
配置 SHA256 证书指纹

获取 SHA256 证书指纹，并在华为推送平台中配置证书指纹，**单击** ✓ **保存**。证书指纹获取可参见 生成签名证书指纹 。



获取华为推送配置文件

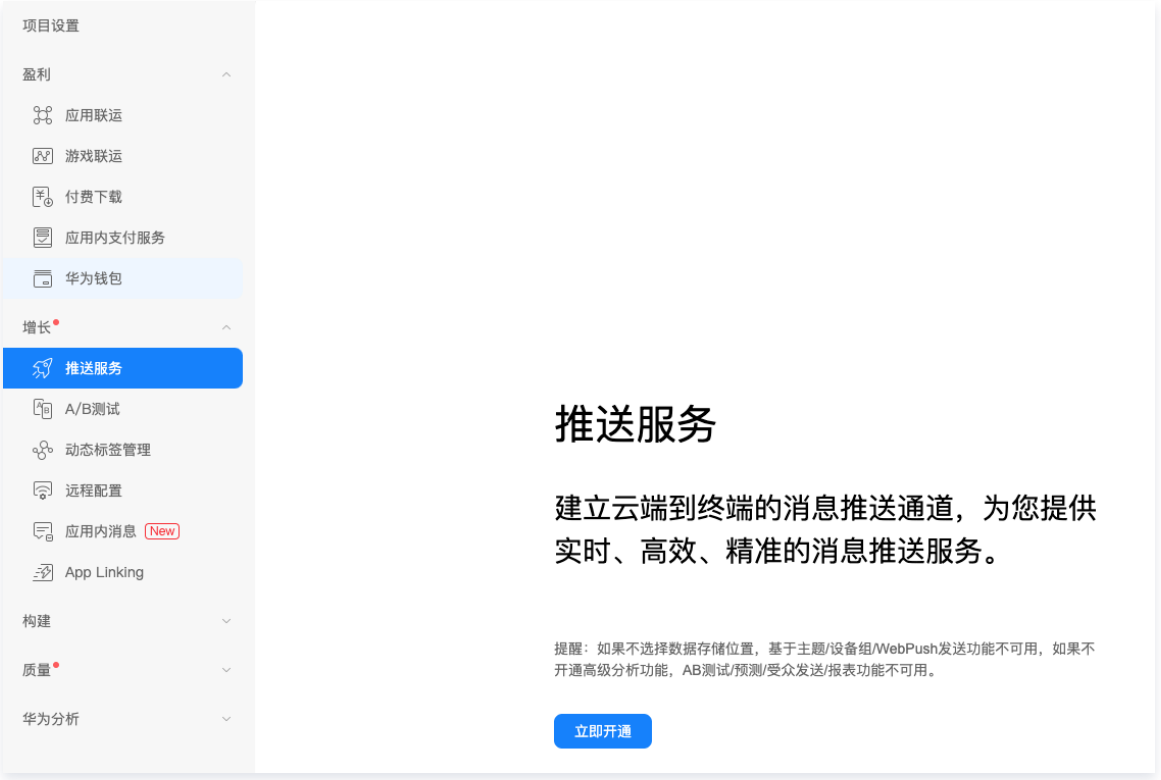
登录华为开放平台，进入**我的项目** > 选择项目 > **项目设置**，下载华为应用最新配置文件 agconnect-services.json。



打开推送服务开关

1. 在华为推送平台，单击**全部服务>推送服务**，进入推送服务页面。

2. 在“推送服务”页面，单击**立即开通**，详情请参见 打开推送服务开关 。



SDK 集成（二选一）

Android Studio Gradle 自动集成

Gradle 7.0 以下版本

1. 在 Android 项目级目录 build.gradle 文件，**buildscript > repositories & dependencies** 下分别添加华为仓库地址和 HMS gradle 插件依赖：

```
buildscript {
        repositories {
                mavenCentral()
                maven {url 'https://developer.huawei.com/repo/'}      // 华为 maven 仓库地址
```

```
        }
        dependencies {
                // 其他classpath配置
                classpath 'com.huawei.agconnect:agcp:1.9.1.301'      // 华为推送 gradle 插件依赖
        }
}
```

2. 在 Android 项目级目录 build.gradle 文件，**allprojects > repositories** 下添加华为依赖仓库地址：

```
allprojects {
 repositories {
     mavenCentral()
     maven {url 'https://developer.huawei.com/repo/'}      // 华为 maven 仓库地址
 }
}
```

Gradle 7.0 版本

1. 在 Android 项目级目录 build.gradle 文件，**buildscript > repositories & dependencies** 下分别添加华为仓库地址和 HMS gradle 插件依赖：

```
buildscript {
 repositories {
     mavenCentral()
     maven {url 'https://developer.huawei.com/repo/'}      // 华为 maven 仓库地址
 }
 dependencies {
     // 其他classpath配置
     classpath 'com.huawei.agconnect:agcp:1.9.1.301'      // 华为推送 gradle 插件依赖
 }
}
```

2. 在 Android 项目级目录 settings.gradle 文件，**dependencyResolutionManagement > repositories** 下添加华为依赖仓库地址：

```
dependencyResolutionManagement {
 repositories {
     google()
 mavenCentral()
     maven {url 'https://developer.huawei.com/repo/'}      // 华为 maven 仓库地址
 }
}
```

Gradle 7.1 及以上版本

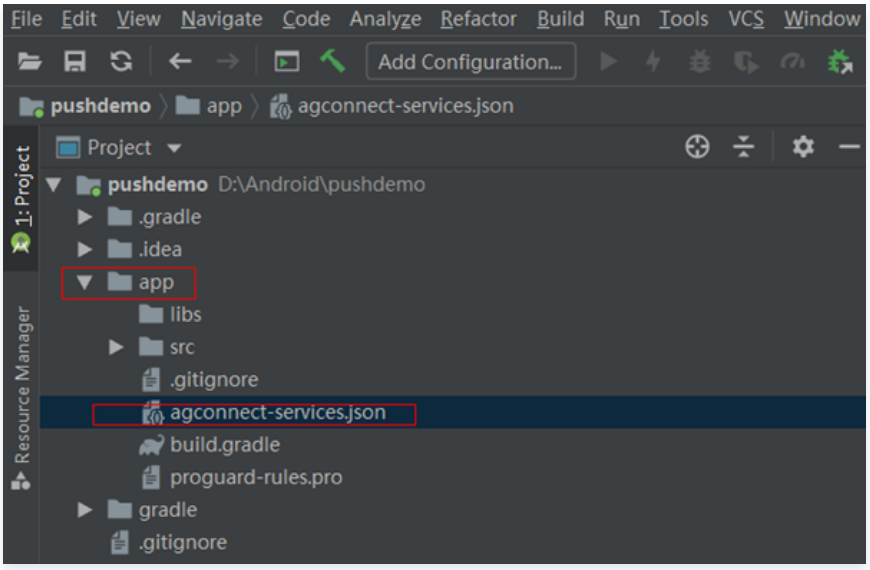
1. 在 Android 项目级目录 build.gradle 文件，**buildscript > dependencies** 下添加 HMS gradle 插件依赖：

```
buildscript {
 dependencies {
     // 其他classpath配置
     classpath 'com.huawei.agconnect:agcp:1.9.1.301'      // 华为推送 gradle 插件依赖
 }
}
```

2. 在 Android 项目级目录 settings.gradle 文件，**pluginManagement & dependencyResolutionManagement > repositories** 下分别添加华为依赖仓库地址：

```
pluginManagement {
 repositories {
     mavenCentral()  // 此处请调整 mavenCentral() 在仓库配置列表最前置位置
     google()
     gradlePluginPortal()
     maven {url 'https://developer.huawei.com/repo/'} // 华为仓，如果不集成华为可以不加
 }
}
dependencyResolutionManagement {
 repositories {
     google()
 mavenCentral()
     maven {url 'https://developer.huawei.com/repo/'}      // 华为 maven 仓库地址
 }
}
```

1. 将从华为推送平台获取的应用配置文件 agconnect-services.json 拷贝到 app 模块目录下（请勿放在子模块下）。



2. 在 app 模块下 build.gradle 文件头部添加以下配置（请勿放在子模块 build.gradle 下）：

```
// app 其他 gradle 插件
apply plugin: 'com.huawei.agconnect'      // HMS SDK gradle 插件
android {
 // app 配置内容
}
```

3. 在 app 模块下 build.gradle 文件内导入华为推送相关依赖：

```
dependencies {
     // ... 程序其他依赖
     implementation 'com.tencent.tpns:huawei:[VERSION]-release'      //  华为推送 [VERSION] 为当前最新 SDK 版本号，版本号可在 Android SDK 发布动态 查看
     implementation 'com.huawei.hms:push:6.12.0.300'       // HMS Core Push 模块依赖包
     }
```

> **说明**
> - 华为推送 hms:push 依赖自6.1.300起适配 Android 11预置 `<queries>` 标签，请注意升级 Android Studio 至3.6.1或更高版本、Android Gradle 插件至 3.5.4或更高版本，否则可能导致工程构建出错。
> - 华为推送 [VERSION] 为当前最新 SDK 版本号，版本号可在 Android SDK 发布动态 查看。
> - 移动推送 Android SDK 自1.2.1.3版本起正式支持华为推送 V5 版本，请使用1.2.1.3及以上版本的移动推送华为依赖以避免集成冲突问题。

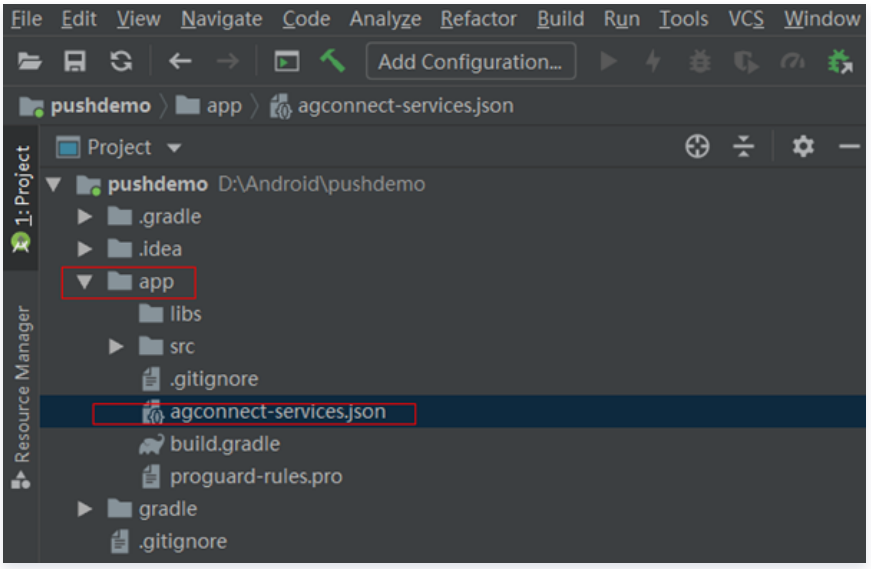
Android Studio 手动集成

针对开发者内部构建环境无法访问华为 maven 仓库的情况，提供以下手动集成方法。

1. 下载 SDK 安装包 。
2. 打开 Other-Push-jar 文件夹， 导入 huaweiv5 推送相关依赖包，将全部 jar、aar 包复制到项目工程中。
3. 在 Android 项目级目录 build.gradle 文件，**buildscript > dependencies**下添加 HMS gradle 插件的依赖：

```
buildscript {
 repositories {
     google()
     jcenter()
 }
 dependencies {
     // 其他 classpath 配置
     classpath files('app/libs/agcp-1.4.1.300.jar')      // 华为推送 gradle 插件依赖
 }
}
```

4. 将从华为推送平台获取的应用配置文件 agconnect-services.json 拷贝到 app 模块目录下。



5. 在 app 模块下 build.gradle 文件头部添加以下配置：

```
// app 其他 gradle 插件
apply plugin: 'com.huawei.agconnect'      // HMS SDK V4 gradle 插件
android {
 // app 配置内容
}
```

6. 在 app 模块下 build.gradle 文件内导入华为推送相关依赖：

```
dependencies {
     // ... 程序其他依赖
     implementation files('libs/tpns-huaweiv5-1.2.1.1.jar')      // 适用于 HMS Core 版本的 TPNS 插件
```

```
    implementation fileTree(include: ['*.aar'], dir: 'libs')     // HMS Core Push 模块依赖包
}
```

7. 在 manifest 文件 `<application> </application>` 标签内添加以下组件：

```
<application>
    <service
        android:name="com.huawei.android.hms.tpns.HWHmsMessageService"
        android:exported="false">
        <intent-filter>
            <action android:name="com.huawei.push.action.MESSAGING_EVENT" />
        </intent-filter>
    </service>
</application>
```

启动华为推送

1. 在调用移动推送注册接口 `XGPushManager.registerPush` 前，开启第三方推送接口：

```
//打开第三方推送
XGPushConfig.enableOtherPush(getApplicationContext(), true);
```

2. 注册成功的日志如下：

```
V/TPush: [XGPushConfig] isUsedOtherPush:true
E/xg.vip: get otherpush errcode: errCode : 0 , errMsg : success
V/TPush: [XGPushConfig] isUsedOtherPush:true
I/TPush: [OtherPushClient] handleUpdateToken other push token is :
IQAAAACy0PsqAADxfCrWG3kupbOraeAiYoo9n2B-bAfb2d--kctc8E_UnY_mrIdg9ionukZvC******dVD8GlJi_5-
0rpskunnNMcat35HA other push type: huawei
```

代码混淆

1. 将以下混淆规则添加在 App 项目级别的 proguard-rules.pro 文件中。

```
-ignorewarnings
-keepattributes *Annotation*
-keepattributes Exceptions
-keepattributes InnerClasses
-keepattributes Signature
-keepattributes SourceFile,LineNumberTable
-keep class com.hianalytics.android.**{*;}
-keep class com.huawei.updatesdk.**{*;}
-keep class com.huawei.hms.**{*;}
-keep class com.huawei.agconnect.**{*;}
```

2. 如应用使用了 AndResGuard 插件，请在 AndResGuard 配置白名单中添加以下内容。如果未使用 AndResGuard 插件，则请忽略该步骤。

```
whiteList = [
 "R.string.hms*",
 "R.string.connect_server_fail_prompt_toast",
 "R.string.getting_message_fail_prompt_toast",
 "R.string.no_available_network_prompt_toast",
 "R.string.third_app_*",
 "R.string.upsdk_*",
 "R.layout.hms*",
 "R.layout.upsdk_*",
 "R.drawable.upsdk*",
```

```
  "R.color.upsdk*",
  "R.dimen.upsdk*",
  "R.style.upsdk*",
  "R.string.agc*"
]
```

> **说明**
> 更多混淆配置请参见 华为推送配置混淆脚本 。

高级配置（可选）

华为通道抵达回执配置

华为通道抵达回执需要开发者自行配置，您可参见 华为厂商通道回执配置指引 进行配置。完成后，可在推送记录中查看华为推送通道的抵达数据。

通道类型	计划发送	实际发送	抵达数	抵达率
总计	1	1	1	100.00%
华为官方推送通道	1	1	1	100.00%

华为设备角标适配

华为设备支持设置应用角标，需要开发者申请应用内角标设置和设置应用启动类权限，详情请参见 角标适配指南 文档。

常见问题排查

华为推送注册错误码查询方法

华为推送服务接入过程配置要求较严格，若观察到如下类似日志则说明华为厂商通道注册失败，开发者可以通过以下方式获取华为推送注册错误码：

```
[OtherPushClient] handleUpdateToken other push token is :  other push type: huawei
```

推送服务 debug 模式下，过滤关键字“OtherPush”或“HMSSDK”，查看相关返回码日志（例如 `[OtherPushHuaWeiImpl] other push huawei onConnect code:907135702` ），并前往 厂商通道注册失败排查指南 查找对应原因，获取解决办法。

通过华为通道下发的通知，为什么没有通知提醒？

根据 华为消息分类标准，华为推送服务将通知消息分为资讯营销、服务与通讯两大类别。资讯营销类消息的每日推送数量自2023年01月05日起根据应用类型对推送数量进行上限管理，服务与通讯类消息每日推送数量不受限，资讯营销类消息单个应用每日每设备通知推送数量默认为2/5条。若推送消息属于服务与通讯类型的，请前往华为开发平台 申请自分类权益，开发者自行对推送进行分类，推送时把自分类参数带下来即可。详情请参见 华为推送限制规则 。

荣耀通道接入

最近更新时间：2025-03-19 10:14:52

操作场景

移动推送SDK 1.3.3.0 起支持荣耀推送通道接入。荣耀通道是由荣耀官方提供的系统级推送通道。在荣耀手机上，推送消息能够通过荣耀的系统通道抵达终端，并且无需打开应用，即可收到推送，更多详情请参见 荣耀推送业务介绍 。

说明：
- SDK 1.3.8.0版本起支持 Magic UI 4.0 及以上荣耀设备使用。
- 申请开通荣耀推送服务需荣耀开发者企业账户。
- 荣耀推送需要在签名包环境下才可注册厂商通道成功并通过厂商通道进行推送。

警告：
由于荣耀计划在2024年开始独立运营商业内容，并逐步将各项系统能力从华为转移至自身，无法继续通过华为通道下发消息。因此，请您尽快接入荣耀通道，以实现荣耀的离线推送功能。

操作步骤

步骤1：获取密钥

1. 进入 荣耀开发者服务平台 注册和登录开发者账号，详情参见 账号注册认证（使用荣耀推送需注册企业开发者账号）。
2. 在荣耀开发者服务平台新建应用并申请开通推送服务，填写应用包名和证书指纹等应用信息，详情参见 申请开通推送服务 。其中获取应用证书指纹可参考 证书指纹生成指南 。
3. 复制应用的 AppId、Client ID 和 Client Secret 参数，分别填入 移动推送控制台 > **App 推送管理** > **基础配置** > **荣耀官方推送通道**栏目的字段 AppId、ClientId、SecretKey 中。

步骤2：配置内容

以下集成方式二选一。

Android Studio 集成方法

在 App 模块下的 build.gradle 文件内，完成移动推送所需的配置后，再增加以下节点：

1. 配置荣耀推送的 AppID，示例代码如下：

```
manifestPlaceholders = [
 HONOR_APPID   : "xxxx"
    ]
```

2. 导入荣耀推送相关依赖，示例代码如下：

```
implementation 'com.tencent.tpns:honor:[VERSION]-release'
```

说明：
- 荣耀推送 [VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态 查看，1.3.3.0 起支持。
- 荣耀推送适配 Android 11 预置 `<queries>` 标签，请注意升级 Android Studio 至 3.6.1 或更高版本、Android gradle build tools 插件至 3.5.4 或更高版本，否则可能导致工程构建出错。

Eclipes 集成方法

获取移动推送荣耀通道 SDK 包后，按照移动推送官网手动集成方法，在配置好移动推送主版本的基础下，进行以下设置：

1. 下载 SDK 安装包 。
2. 打开 Other-Push-jar 文件夹，导入荣耀推送相关 jar 包。

3. 在 `Androidmanifest.xml` 文件中，新增如下配置：

```
<uses-permission android:name="com.hihonor.push.permission.READ_PUSH_NOTIFICATION_INFO" />

 <application>

    <!-- 自定义荣耀推送回调 service -->
    <service
        android:name="com.tencent.android.tpush.honor.HonorMessageService"
        android:exported="false">
        <intent-filter>
            <action android:name="com.hihonor.push.action.MESSAGING_EVENT" />
        </intent-filter>
    </service>

    <meta-data
        android:name="com.hihonor.push.sdk_version"
        android:value="7.0.41.301" />

    <!-- 荣耀推送 appId -->
    <meta-data
        android:name="com.hihonor.push.app_id"
        android:value="${HONOR_APPID}" />
 </application>
```

步骤3：开启荣耀推送

在调用移动推送 `XGPushManager.registerPush` 之前，开启第三方推送接口：

```
//打开第三方推送
XGPushConfig.enableOtherPush(getApplicationContext(), true);


//注册成功的日志如下
I/TPush: [OtherPushClient] handleUpdateToken other push token is : IQAAAACy0Ps******GlJi_5-
0rpskunnNMcat35HA  other push type: honor
I/TPush: [PushServiceBroadcastHandler] >> bind OtherPushToken success ack with [accId = 150000****  , rsp
= 0]  token = 01a22******ed343 otherPushType = honor otherPushToken = IQAAAACy0Ps******GlJi_5-
0rpskunnNMcat35HA
```

步骤4：代码混淆

```
-ignorewarnings
-keepattributes *Annotation*
-keepattributes Exceptions
-keepattributes InnerClasses
-keepattributes Signature
-keepattributes SourceFile,LineNumberTable

-keep class com.hihonor.push.framework.aidl.**{*;}
-keep class com.hihonor.push.sdk.**{*;}
```

> **说明：**
> 混淆规则需要放在 App 项目级别的 proguard-rules.pro 文件中。

步骤5：荣耀通道抵达回执配置

荣耀通道抵达回执需要开发者自行配置，您可参照 荣耀厂商通道回执配置指引 进行配置，完成后，可在推送记录中查看荣耀推送通道的抵达数据。

常见问题排查

荣耀推送注册错误码查询方法

若观察到如下类似日志则说明荣耀厂商通道注册失败，开发者可以通过以下方式获取荣耀推送注册错误码：

```
[OtherPushClient] handleUpdateToken other push token is :  other push type: honor
```

推送服务 debug 模式下，过滤关键字 “OtherPush”，查看相关返回码日志，并前往 厂商通道注册失败排查指南 查找对应原因，获取解决办法。

小米通道接入

最近更新时间：2025-02-24 10:41:32

操作场景

小米推送通道是由**小米官方提供**的系统级推送通道。在小米手机上，推送消息能通过小米的系统通道抵达终端，并且无需打开应用就能够收到推送。

> **注意**
> 在测试小米通道推送消息时，避免使用“test”、“测试”等字眼，否则可能会被小米拦截进入“非重要消息”中。

操作步骤

开启小米推送服务

前往 小米开放平台 >推送运营平台，开启应用的消息推送服务：

获取密钥

1. 进入 小米推送运营平台 ，开通小米开发者账号，并获取 AppId、AppKey、AppSecret 三个密钥参数，详情请参见 快速接入指南 。
2. 复制应用的 AppId、AppKey 和 AppSecret 参数填入 移动推送控制台 >**配置管理>基础配置>小米官方推送通道**栏目中。

配置内容

使用 jcenter 依赖接入

AS 开发建议使用 jcenter 依赖接入。引入小米推送的 jar 包。

```
implementation 'com.tencent.tpns:xiaomi:[VERSION]-release'//小米推送 [VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态查看
```

> **说明**
> 小米推送 [VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态 查看。

使用 Eclipse 开发接入

1. 下载 SDK 安装包 。
2. 打开 Other-Push-jar 文件夹， 导入小米推送相关 jar 包，将 xm4tpns1.1.2.1.jar 导入项目工程中。
3. 在配置好移动推送的基础上 ，新增小米推送的配置：

```
<application>
 <activity
     android:name="com.xiaomi.mipush.sdk.NotificationClickedActivity"
     android:theme="@android:style/Theme.Translucent.NoTitleBar"
     android:launchMode="singleInstance"
     android:exported="true"
     android:enabled="true">
     <meta-data
      android:name="supportStyle"
      android:value="scene|voip" />
 </activity>
```

```
<meta-data
   android:name="supportGetAndroidID"
   android:value="false"
   />
<service
    android:name="com.xiaomi.push.service.XMPushService"
    android:enabled="true"
    android:process=":pushservice" />
<service
    android:name="com.xiaomi.push.service.XMJobService"
    android:enabled="true"
    android:exported="false"
    android:permission="android.permission.BIND_JOB_SERVICE"
    android:process=":pushservice" />
<!-- 注：此service必须在3.0.1版本以后（包括3.0.1版本）加入 -->
<service
    android:name="com.xiaomi.mipush.sdk.PushMessageHandler"
    android:enabled="true"
    android:exported="true" />
<service
    android:name="com.xiaomi.mipush.sdk.MessageHandleService"
    android:enabled="true" />
<!-- 注：此service必须在2.2.5版本以后（包括2.2.5版本）加入 -->
<receiver
    android:name="com.xiaomi.push.service.receivers.NetworkStatusReceiver"
    android:exported="true" >
</receiver>
<receiver
    android:name="com.xiaomi.push.service.receivers.PingReceiver"
    android:exported="false"
    android:process=":pushservice" >
    <intent-filter>
        <action android:name="com.xiaomi.push.PING_TIMER" />
    </intent-filter>
</receiver>
</application>

<!-- 注：小米push 需要的权限 begin -->
<permission
 android:name="应用包名.permission.MIPUSH_RECEIVE"
 android:protectionLevel="signature" />
<!-- 这里 应用包名 改成app的包名 -->

<uses-permission android:name="应用包名.permission.MIPUSH_RECEIVE" />
<!-- 这里 应用包名 改成app的包名 -->
<!-- 注：小米push 需要的权限 end -->
```

4. 在 `AndroidManifest.xml` 增加 `Receiver` ，配置如下：

```
<receiver
 android:exported="true"
 android:name="com.tencent.android.mipush.XMPushMessageReceiver">
 <intent-filter>
     <action android:name="com.xiaomi.mipush.RECEIVE_MESSAGE" />
 </intent-filter>
 <intent-filter>
     <action android:name="com.xiaomi.mipush.MESSAGE_ARRIVED" />
 </intent-filter>
 <intent-filter>
     <action android:name="com.xiaomi.mipush.ERROR" />
```

```
  </intent-filter>
</receiver>
```

开启小米推送

设置小米 AppID 和 AppKey。

```
XGPushConfig.setMiPushAppId(getApplicationContext(), "APPID");
XGPushConfig.setMiPushAppKey(getApplicationContext(), "APPKEY");
//打开第三方推送
XGPushConfig.enableOtherPush(getApplicationContext(), true);


//注册成功的日志如下

I/TPush: [OtherPushClient] handleUpdateToken other push token is :
3CvDLfyPRArAGnv****dvQ7rYko+OthWo90rW+Edeqn53RUudp6U1dhySpV35 other push type: xiaomi
I/TPush: [PushServiceBroadcastHandler] >> bind OtherPushToken success ack with [accId = 1500001048  , rsp
= 0]  token = 03be2036762f******33bce72d40eb5e677a otherPushType = xiaomi otherPushToken =
3CvDLfyPRArAGnv****dvQ7rYko+OthWo90rW+Edeqn53RUudp6U1dhySpV35G
```

如需通过点击回调获取参数或者跳转自定义页面，可以通过使用 Intent 来实现，详情请参见 Android 常见问题 。

代码混淆

```
-keep class com.xiaomi.**{*;}
-keep public class * extends com.xiaomi.mipush.sdk.PushMessageReceiver
```

> **说明**
> 混淆规则需要放在 App 项目级别的 proguard-rules.pro 文件中。

常见问题排查

小米推送注册失败错误码查询方法

若您观察到如下类似日志则说明小米厂商通道注册失败，开发者可以通过以下方式获取小米推送注册错误码：

```
[OtherPushClient] handleUpdateToken other push token is :  other push type: xiaomi
```

推送服务 debug 模式下，过滤关键字 “OtherPush”，查看相关返回码日志（例如 `[OtherPush_XG_MI] register failed, errorCode: 22022, reason: Invalid package name: com.xxx.xxx` ），并前往 厂商通道注册失败排查指南 查找对应原因，获取解决办法。

魅族通道接入

最近更新时间：2024-05-09 11:34:53

操作场景

魅族推送通道是由**魅族官方提供**的系统级推送通道。在魅族手机上，推送消息能够通过魅族的系统通道抵达终端，并且无需打开应用，即可收到推送。

> **说明**
> - 魅族推送通道通知标题不超过32字符，通知内容不超过100字符。
> - 魅族推送通道不支持透传消息。

操作步骤

获取密钥

1. 进入 魅族推送官网，注册并登录开发者账号，获取 AppID、AppKey、AppSecret 三个密钥参数。更多详情请参见 魅族开发文档 。
2. 复制应用的 AppId、AppKey 和 AppSecret 参数填入 **移动推送控制台** > **配置管理** > **基础配置** > **魅族官方推送通道**栏目中。

集成方式

AndroidStudio 集成

```
implementation 'com.tencent.tpns:meizu:[VERSION]-release'//meizu 推送 [VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态查看
```

> **说明**
> 魅族推送 [VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态 查看。

Eclipse 集成

1. 下载 SDK 安装包 。
2. 打开 Other-Push-jar 文件夹， 导入魅族推送相关 jar 包，将 mz4tpns1.1.2.1.jar 导入项目工程中。
3. 请在主工程添加以下类资源文件：

```
package com.meizu.cloud.pushinternal;
public class R {
 public static final class drawable {
        // 资源文件 stat_sys_third_app_notify.png 请从 TPNS SDK 压缩包魅族厂商依赖目录的 flyme-notification-res 文件夹获取，并复制到应用自己的资源目录下
     public static final int stat_sys_third_app_notify =
com.tencent.android.tpns.demo.R.drawable.stat_sys_third_app_notify;
 }
 public static final class string {
   // 可更改为自定义字符串资源ID,对应字符串值为:MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC8hzUojzHX8jDL+97pqr7CaLiKSsZ0aOES7FUcX7vh9PoEDbCKNCTakRXdS5Eiu
rPk3QpvsAGbfyIs7JWKm4py9KcIdJsZRh9onknVeAVlU++jnrGFGEYfQb8iKzClN059gYeeJBs9mwi7RGU9tj0KHUG659v5sMBxv7zN
se3fJQIDAQAB
   public final static int security_public_key =
com.tencent.android.tpns.demo.R.string.security_public_key;
 }
}
```

4. 在 AndroidManifest 文件添加以下配置：

```
<application>
  <!-- 注：魅族push 需要的 begin -->
```

```
 <service
     android:name="com.meizu.cloud.pushsdk.NotificationService"
     android:exported="true"
     android:permission="com.meizu.cloud.push.permission.MESSAGE" />
  <!-- version 4.1.0-->
 <receiver
     android:name="com.meizu.cloud.pushsdk.MzPushSystemReceiver"
     android:exported="false"
     >
     <intent-filter>
         <action android:name="com.meizu.flyme.push.intent.PUSH_SYSTEM" />
     </intent-filter>
 </receiver>
</application>
<!-- version 4.1.0-->
<uses-permission android:name="com.meizu.flyme.permission.PUSH" />
<!-- version 3.9.0-->
<!-- 注：魅族push 需要的权限 begin -->
<!-- 兼容flyme5.0以下版本，魅族内部集成pushSDK必填，不然无法收到消息-->
<uses-permission android:name="com.meizu.flyme.push.permission.RECEIVE"></uses-permission>
<permission
 android:name="${applicationId}.push.permission.MESSAGE"
 android:protectionLevel="signature" />
<uses-permission android:name="${applicationId}.push.permission.MESSAGE"></uses-permission>
<!--  兼容flyme3.0配置权限-->
<uses-permission android:name="com.meizu.c2dm.permission.RECEIVE" />
<permission
 android:name="${applicationId}.permission.C2D_MESSAGE"
 android:protectionLevel="signature"></permission>
<uses-permission android:name="${applicationId}.permission.C2D_MESSAGE" />
<!-- 注：魅族push 需要的权限 end -->
```

5. 魅族消息 receiver：在 `AndroidManifest.xml` 增加 `Receiver` 配置如下：

```
<receiver
        android:name="com.tencent.android.mzpush.MZPushMessageReceiver"
        android:exported="true">

 <intent-filter>
     <!-- 接收push消息 -->
     <action android:name="com.meizu.flyme.push.intent.MESSAGE" />
     <!-- 接收register消息-->
      <action android:name="com.meizu.flyme.push.intent.REGISTER.FEEDBACK"/>
     <!-- 接收unregister消息-->
      <action android:name="com.meizu.flyme.push.intent.UNREGISTER.FEEDBACK"/>
      <action android:name="com.meizu.c2dm.intent.REGISTRATION" />
      <action android:name="com.meizu.c2dm.intent.RECEIVE" />
      <category android:name="应用包名"></category>
  </intent-filter>
</receiver>
```

6. Flyme 6.0 及以下版本的魅族手机，使用手动集成方式，需要在 drawable 不同分辨率的文件夹下对应放置一张名称必须为 stat_sys_third_app_notify 的图片，详情请参见 移动推送 Android SDK 中魅族厂商依赖目录的 flyme-notification-res 文件夹。

开启魅族推送

在启动移动推送，调用 `XGPushManager.registerPush` 之前，配置如下代码：

```
//设置魅族APPID和APPKEY
XGPushConfig.enableOtherPush(context, true);
```

```
XGPushConfig.setMzPushAppId(this, APP_ID);
XGPushConfig.setMzPushAppKey(this, APP_KEY);
```

注册成功的日志如下：

```
//成功的获取到 TPNS 的token和魅族的token，并且绑定成功说明注册成功
I/TPush: [OtherPushClient] handleUpdateToken other push token is :
V5R5b7c02********47744c6b635e464b527e487802 other push type: meizu
I/TPush: [PushServiceBroadcastHandler] >> bind OtherPushToken success ack with [accId = 150000****  , rsp
= 0]  token = 0398291156ce7d2f****66bd0952c87c372f otherPushType = meizu otherPushToken =
V5R5b7c02********47744c6b635e464b527e487802
```

如需通过点击回调获取参数或者跳转自定义页面，您可使用 Intent 来实现。更多详情请参见 Android 常见问题 。

代码混淆

1. 将以下混淆规则添加在 App 项目级别的 proguard-rules.pro 文件中。

```
-dontwarn com.meizu.cloud.pushsdk.**
-keep class com.meizu.cloud.pushsdk.**{*;}
```

> **说明**
> 若出现 App Debug 版本注册魅族 token 正常，但在 Release 版本中无法获取，请升级 SDK 1.3.2.0 及以上版本。

2. 如应用使用了 AndResGuard 插件，请在 AndResGuard 配置白名单中添加以下内容。如果未使用 AndResGuard 插件，则请忽略该步骤。

```
whiteList = [
"R.drawable.stat_sys_third_app_notify"
]
```

魅族通道抵达回执配置

魅族通道抵达回执需要开发者自行配置，您可参照 魅族厂商通道回执配置指引 进行配置，完成后，可在推送记录中查看魅族推送通道的抵达数据。

数据明细 收起 ▲ 数据下载

通道类型 ⓘ	计划发送 ⓘ	实际发送 ⓘ	抵达数 ⓘ	抵达率 ⓘ	展示数 ⓘ	点击/清除 ⓘ	点击率 ⓘ
总计	1	1	1	100.00%	0	0 / 0	0.00%
魅族官方推送通道	1	1	1	100.00%	N/A	N/A	N/A

N/A表示该厂商不提供该指标统计

常见问题排查

魅族推送注册失败错误码查询方法

若您观察到如下类似日志则说明魅族厂商通道注册失败，开发者可以通过以下方式获取魅族推送注册错误码：

```
[OtherPushClient] handleUpdateToken other push token is :  other push type: meizu
```

推送服务 debug 模式下，过滤关键字 “OtherPush” ，查看相关返回码日志（例如 `[OtherPush_XG_MZ] onRegisterStatus BasicPushStatus{code='110000', message='appId不合法'}` ），并前往 厂商通道注册失败排查指南 查找对应原因，获取解决办法。

厂商通道下发失败：客户端/服务端错误码：500，是什么原因?

该问题是因为魅族抵达回执地址配置不正确导致，可参照 魅族厂商通道回执配置指引 重新进行配置。

FCM 通道接入

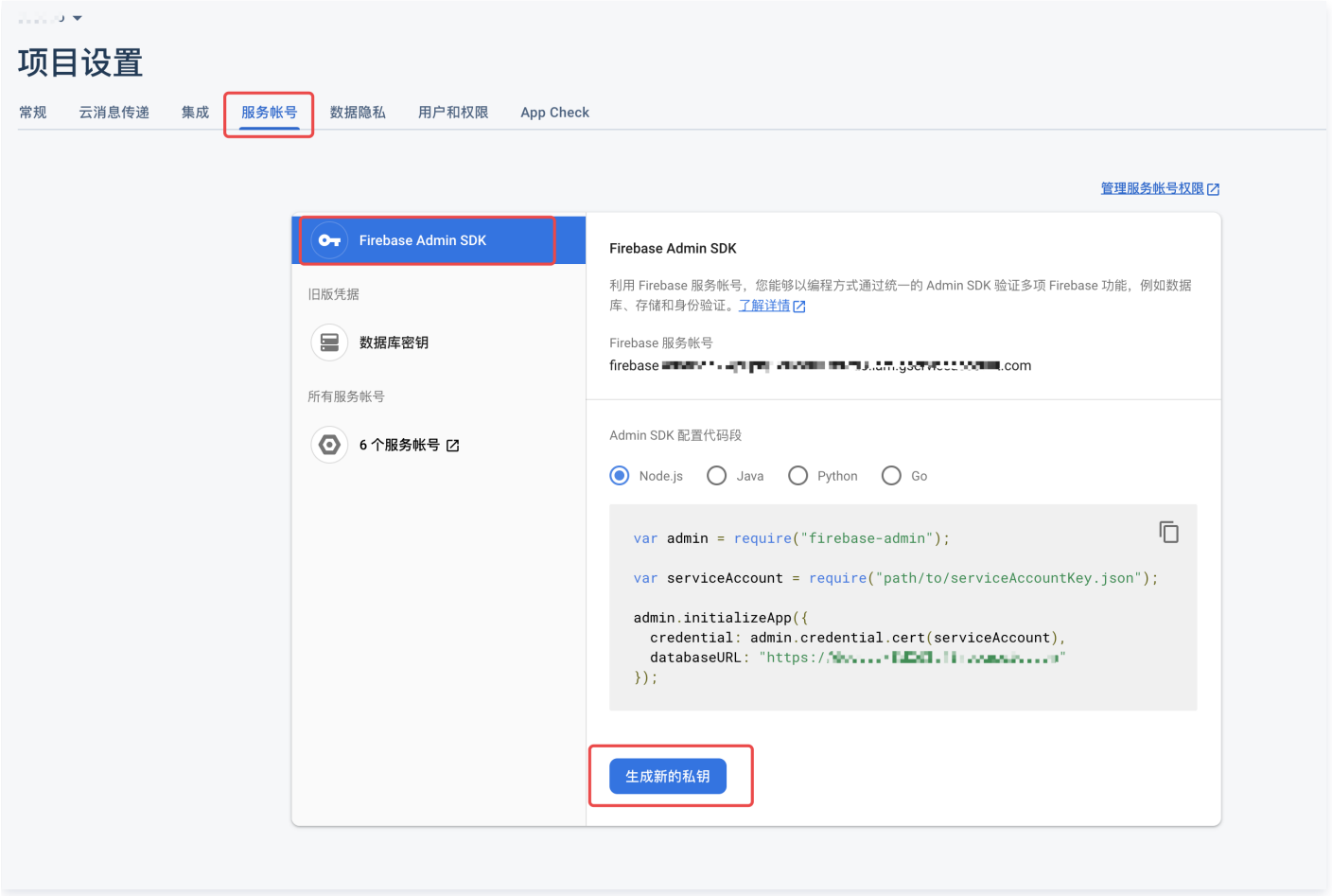
最近更新时间：2025-03-11 16:12:52

操作场景

FCM 通道是谷歌推出的系统级推送通道，在国外具备谷歌 Service 框架的手机上，鉴于其较宽松的后台进程管理方式，在应用进程未被强制停止的情况下，可以收到推送消息。FCM 通道不支持国内集群。

操作步骤

进入 Firebase 官网，注册应用信息。在 **Firebase 项目 > 选择具体的项目应用 > 设置 > 服务账号 > Firebase Admin SDK**，单击**生成新的私钥**，获取到 Firebase 服务器私钥 json 文件。然后进入 **移动推送控制台 > 配置管理 > 基础配置 > FCM 官方推送通道** 栏目中，选中“（推荐）服务器私钥”，单击**点击上传**，上传获取到的 json 文件。

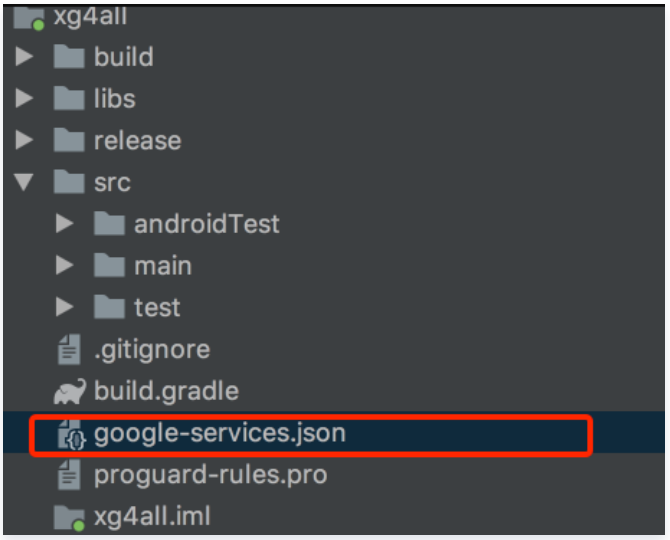


注意：

fcm 不支持旧版服务器密钥，强制使用新私钥。若您当前使用旧版服务器密钥请及时切换。

配置内容

1. 配置 google-services.json 文件。如图所示：



2. 配置 gradle，集成谷歌 service。

2.1 在项目级的 build.gradle 文件中的 dependencies 节点中添加下面代码：

```
classpath 'com.google.gms:google-services:4.2.0'
```

> ⚠ 注意
> 如果使用低于4.2.0版本出现
> `FCM Register error! java.lang.IllegalStateException: Default FirebaseApp is not initialized in this process com.qq.xg4all. Make sure to call FirebaseApp.initializeApp(Context) first.`
> ，建议在 res/values 文件夹下的 string.xml， 加上 YOUR_GOOGLE_APP_ID。

2.2 在应用级的 build.gradle 文件中，添加依赖：

```
    implementation 'com.tencent.tpns:fcm:[VERSION]-release' // FCM 推送 [VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态查看
     implementation  'com.google.firebase:firebase-messaging:24.0.1'
     //在应用级的 gradle 文件的最后一行代码中新增并将 google-services.json 放进您应用 module 的根路径下
apply plugin: 'com.google.gms.google-services'
```

> ⚠ 注意：
> FCM 推送 [VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态 查看。
> Google 配置 google-play-services（建议版本 24.0.1+，较低版本有可能出现无法注册 FCM 风险）。

启用 FCM 推送

在调用移动推送注册代码 XGPushManager.registerPush 前，添加以下代码设置：

```
XGPushConfig.enableOtherPush(this, true);
```

注册 FCM 成功的日志如下：

```
V/TPush: [XGPushConfig] isUsedOtherPush:true
I/TPush: [OtherPush] checkDevice pushClassNamecom.tencent.android.tpush.otherpush.fcm.impl.OtherPushImpl
I/TPush: [XGPushManager] other push token is :
dSJA5n4fSZ27YeDf2rFg1A:APA91bGiqSPCMZTuyup**********f1fBIahZKYkth2OoDpixDPQmEZkQ11fX06mw_1kEaW5-
jFmT4YwlER4qfX66h_BIoUxOyj_tKqZSUg7oHigIKaOrDWmMQfMAqGoT8qSfg  other push type: fcm
```

代码混淆

```
-keep class com.google.firebase.** {*;}
```

> **说明**
> 混淆规则需要放在 App 项目级别的 proguard-rules.pro 文件中。

常见问题排查

推送 FCM 推送收不到，是什么原因?

1. 在境外具备谷歌 Service 框架的手机上，鉴于其较宽松的后台进程管理方式，在应用进程未被强制停止的情况下， FCM 消息抵达较为稳定。
2. 在大陆发行的国内品牌手机，其后台进程管理普遍较为严格，谷歌 service 后台服务同样也会受到限制，这些手机上 FCM 消息抵达可能会受到影响，FCM 无法进行下发和接收，建议保持 App 在前台接收。

什么是强制停止应用进程?

在手机设置-应用管理-具体应用-点击“结束运行”/“强制停止”等按钮停止了应用。大部分国内品牌手机，在多任务页面划掉应用进程，也可认为是强制停止了应用进程（境外手机不会）。

vivo 通道接入

最近更新时间：2024-10-11 15:31:21

操作场景

vivo 通道是由 vivo 官方提供的系统级推送通道。在 vivo 手机上，推送消息能够通过 vivo 的系统通道抵达终端，并且无需打开应用，即可收到推送，更多详情请参见 vivo 推送官网 。

说明

- 如遇到 debug 版本点击通知后无法拉起 App，请在权限设置中找到后台弹出界面，并开启当前应用的权限开关。
- vivo 通道暂不支持应用内消息，此类型的消息将通过移动推送自建通道下发。
- vivo 通道对应用的每日推送量有额度限制，详情请参见 厂商通道限额说明 ，超过限制部分将走移动推送自建通道进行补推发送。
- vivo 通道7:00 - 23:00允许推送消息，其他时间只能推送系统消息，系统消息申请详情请参见 vivo 系统消息申请指南 。
- vivo 通道单应用单用户每天接收运营消息条数上限为2/5条，系统消息不限接收条数。

操作步骤

获取密钥

1. 开发者需向 vivo 申请开通推送权限，获取到 AppID 、AppKey、AppSecret 三个密钥参数。详情请参见 快速接入指引 。

   说明
   开发者的应用必须要在 vivo 开放平台通过审核上架后，才会通过消息推送服务审核。

2. 复制应用的 AppId、AppKey 和 AppSecret 参数填入 移动推送控制台 > **配置管理** > **基础配置** > **vivo 官方推送通道**栏目中。

配置内容

AndroidStudio 集成方法

在 App 模块下的 build.gradle 文件内，完成移动推送所需的配置后，再增加以下节点：

1. 配置 vivo 的 AppID 和 AppKey。示例代码如下：

```
manifestPlaceholders = [
 VIVO_APPID:"xxxx",
 VIVO_APPKEY:"xxxxx",
    ]
```

2. 导入 vivo 推送相关依赖。示例代码如下：

```
implementation 'com.tencent.tpns:vivo:[VERSION]-release' // vivo 推送 [VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态查看
```

说明
vivo 推送 [VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态 查看。

Eclipse 集成方法

获取移动推送 vivo 通道 SDK 包后，按照移动推送官网手动集成方法，在配置好移动推送主版本的基础下，进行以下设置：

1. 下载 SDK 安装包 。
2. 打开 Other-Push-jar 文件夹， 导入 vivo 推送相关 jar 包。
3. 在 `AndroidManifest.xml` 文件中，新增如下配置：

```
<application>
 <service
```

```
        android:name="com.vivo.push.sdk.service.CommandClientService"
        android:exported="true"
        android:permission="com.push.permission.UPSTAGESERVICE" />
    <activity
        android:name="com.vivo.push.sdk.LinkProxyClientActivity"
        android:exported="false"
        android:screenOrientation="portrait"
        android:theme="@android:style/Theme.Translucent.NoTitleBar" />
    <!-- push应用定义消息receiver声明 -->
    <receiver
        android:name="com.tencent.android.vivopush.VivoPushMessageReceiver"
        android:exported="false">
        <intent-filter>
            <!-- 接收push消息 -->
            <action android:name="com.vivo.pushclient.action.RECEIVE" />
        </intent-filter>
    </receiver>
    <!-- version 3.0.0.3 -->
    <!-- 该字段大于等于480，则表示支持点击消息直接打开应用activity，优化启动慢问题。 -->
    <meta-data
        android:name="sdk_version_vivo"
        android:value="483" />
    <meta-data
        android:name="local_iv"

android:value="MzMsMzQsMzUsMzYsMzcsMzgsMzksNDAsNDEsMzIsMzgsMzcsMzYsMzUsMzQsMzMsI0AzNCwzMiwzMywzNywzMywz
NCwzMiwzMywzMywzMywzNCw0MSwzNSwzNSwzMiwzMiwjQDMzLDM0LDM1LDM2LDM3LDM4LDM5LDQwLDQxLDMyLDM4LDM3LDMzLDM1LDM1LDM
0LDMzLCNAMzQsMzIsMzMsMzMsMzcsMzMsMzQsMzIsMzMsMzMsMzMsMzMsMzQsNDEsMzUsMzIsMzIsMzI=" />
    <!-- version 3.0.0.3 end-->
    <meta-data
        android:name="com.vivo.push.api_key"
        android:value="${VIVO_APPKEY}" />
    <meta-data
        android:name="com.vivo.push.app_id"
        android:value="${VIVO_APPID}" />
</application>
```

开启 vivo 推送

在调用移动推送 `XGPushManager.registerPush` 之前，开启第三方推送接口：

```
//打开第三方推送
XGPushConfig.enableOtherPush(getApplicationContext(), true);


//注册成功的日志如下
I/TPush: [OtherPushClient] handleUpdateToken other push token is : 160612459******08955218 other push
type: vivo
I/TPush: [PushServiceBroadcastHandler] >> bind OtherPushToken success ack with [accId = 150000****  , rsp
= 0]  token = 01a22fb503a33******66b89fad6be3ed343 otherPushType = vivo otherPushToken =
160612459******08955218
```

代码混淆

```
-dontwarn com.vivo.push.**
-keep class com.vivo.push.**{*; }
-keep class com.vivo.vms.**{*; }
```

```
-keep class com.tencent.android.vivopush.VivoPushMessageReceiver{*;}
```

> **说明**
> 混淆规则需要放在 App 项目级别的 proguard-rules.pro 文件中。

常见问题排查

vivo 推送注册错误码查询方法

若观察到如下类似日志则说明 vivo 厂商通道注册失败，开发者可以通过以下方式获取 vivo 推送注册错误码：

```
[OtherPushClient] handleUpdateToken other push token is :  other push type: vivo
```

推送服务 debug 模式下，过滤关键字 “OtherPush”，查看相关返回码日志（例如 `[OtherPushVivoImpl] vivoPush Register or UnRegister fail, code = 10003`），并前往 厂商通道注册失败排查指南 查找对应原因，获取解决办法。

推送 vivo 响应失败 code:10045，是什么原因?

应用审核中不可发送正式消息，请前往 vivo 平台确认推送权限审核进度。

OPPO 通道接入

最近更新时间：2024-10-11 15:31:21

操作场景

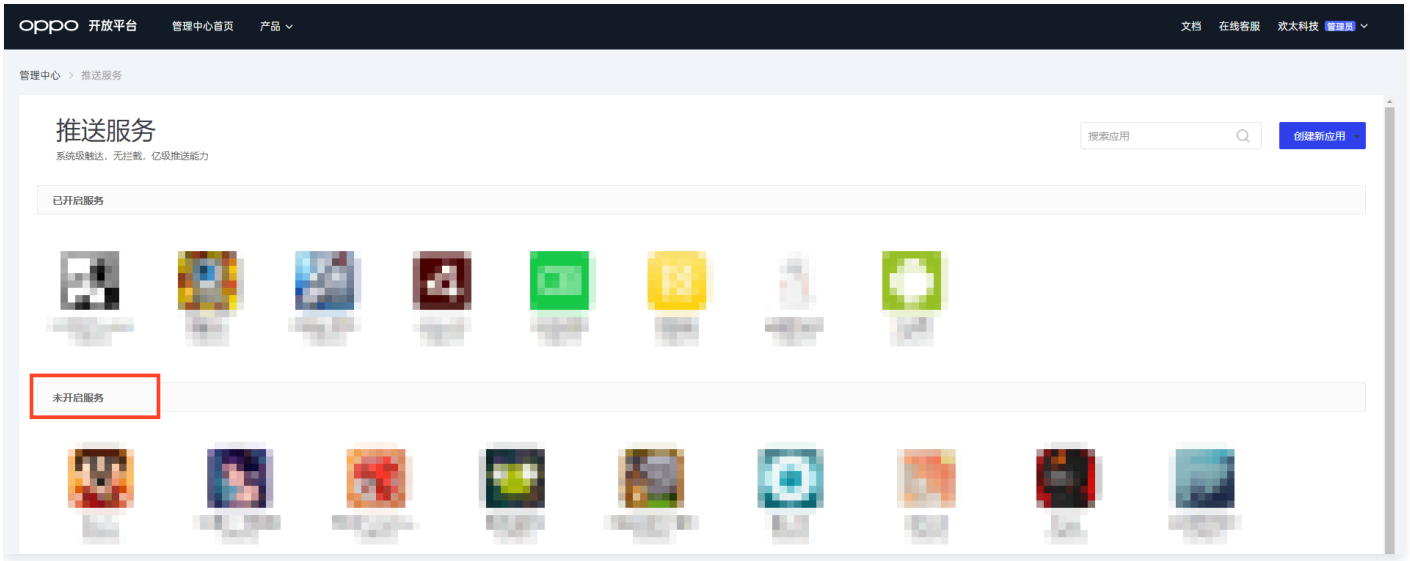
OPPO 通道是由 OPPO 官方提供的系统级推送通道。在 OPPO 手机上，推送消息能够通过 OPPO 的系统通道抵达终端，无需打开应用就能够收到推送。详情请参见 OPPO 推送官网 。

> **说明**
> - OPPO 通道暂不支持应用内消息的发送，此类型的消息会通过移动推送自建通道进行下发。
> - OPPO 通道对应用的每日推送量有额度限制，详情请参见 厂商通道限额说明 ，超过限制部分将走移动推送自建通道进行补推发送。
> - OPPO 通道需要 OPPO 手机系统 ColorOS V3.1 及以上支持。

操作步骤

开通权限

使用 OPPO 企业开发者账号，登录 OPPO 开发平台 ，在**管理中心** > **服务分发** > **推送服务**中为未开启服务的应用申请 OPPO PUSH 推送服务权限。

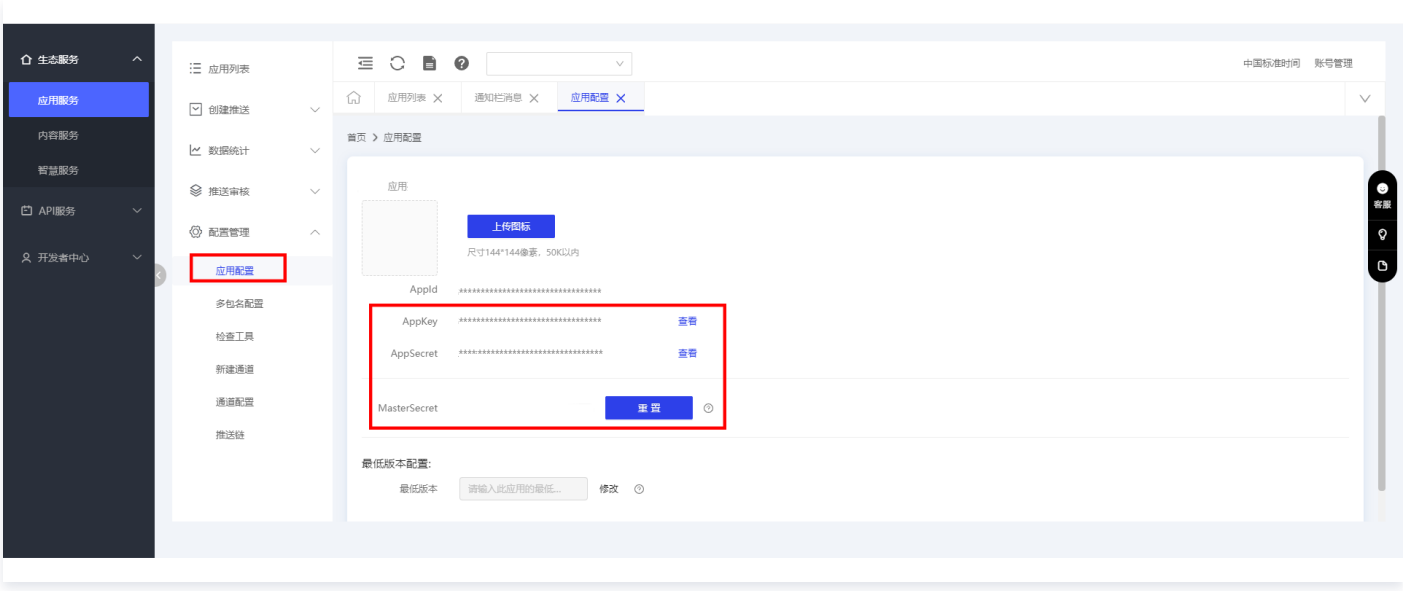


> **说明**
> 通知栏推送权限申请需要应用在 OPPO 软件商店上架才可通过，且主营业务不为借贷类的应用。

获取密钥

> **说明**
> 仅开发者账号（主账号）可查看。

1. Opush 申请开通成功后，您可在 **OPPO 推送平台** > **配置管理** > **应用配置页面**，查看 AppKey、AppSecret 和 MasterSecret。



2. 复制应用的 AppKey、AppSecret 和 MasterSecret 参数填入 **移动推送控制台** > **配置管理** > **基础配置** > **OPPO 官方推送通道** 栏目中。

配置推送通道

为兼容安卓8.0及以上版本的 OPPO 手机的通道配置，用户需在 OPPO 管理台上，创建一个移动推送推送的默认通道。详情请参见 OPPO 官方文档 。
具体内容为：

- “通道 ID”：“default_message”
- “通道名称”：“默认通知”

配置内容

AndroidStudio 集成方法

导入 OPPO 推送相关依赖。示例代码如下：

```
//  OPPO 推送 SDK，[VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态查看
implementation 'com.tencent.tpns:oppo:[VERSION]-release'

// 自 SDK 1.3.2.0 起，需一并加入以下依赖语句，否则可能导致 OPPO 推送注册失败
implementation 'com.google.code.gson:gson:2.6.2'
implementation 'commons-codec:commons-codec:1.15'
```

> **说明**
> OPPO 推送 [VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态 查看。

Eclipse 集成方法

获取移动推送 OPPO 通道 SDK 包后，按照移动推送官网手动集成方法，在配置好移动推送主版本的基础下，进行以下设置。
1. 打开 Other-push-jar 文件夹，将 OPPO 推送相关 jar 导入项目工程中。
2. 在主工程添加类资源文件，代码如下：

```
package com.pushsdk;

class R {
 public static final class string {
 public final static int system_default_channel = com.tencent.android.tpns.demo.R.string.app_name; // 可更改为自定义字符串资源ID
 }
```

```
}
```

3. 在 `Androidmanifest.xml` 文件中新增如下配置：

```
<!--OPPO 推送服务必须权限-->
<uses-permission android:name="com.coloros.mcs.permission.RECIEVE_MCS_MESSAGE"/>
<uses-permission android:name="com.heytap.mcs.permission.RECIEVE_MCS_MESSAGE"/>
<application>
     <service
         android:name="com.heytap.msp.push.service.CompatibleDataMessageCallbackService"
         android:permission="com.coloros.mcs.permission.SEND_MCS_MESSAGE"
         android:exported="true">
         <intent-filter>
         <action android:name="com.coloros.mcs.action.RECEIVE_MCS_MESSAGE" />
         </intent-filter>
     </service>

     <service
         android:name="com.heytap.msp.push.service.DataMessageCallbackService"
         android:permission="com.heytap.mcs.permission.SEND_PUSH_MESSAGE"
         android:exported="true">
         <intent-filter>
         <action android:name="com.heytap.mcs.action.RECEIVE_MCS_MESSAGE" />
         <action android:name="com.heytap.msp.push.RECEIVE_MCS_MESSAGE" />
         </intent-filter>
     </service>
</application>
```

开启 OPPO 推送

在调用移动推送 `XGPushManager.registerPush` 之前，调用以下代码：

```
// 注意这里填入的是 Oppo 的 AppKey，不是AppId
XGPushConfig.setOppoPushAppId(getApplicationContext(), "Oppo的AppKey");
// 注意这里填入的是 Oppo 的 AppSecret，不是 AppKey
XGPushConfig.setOppoPushAppKey(getApplicationContext(), "Oppo的AppSecret");
//打开第三方推送
XGPushConfig.enableOtherPush(getApplicationContext(), true);

//注册成功的日志如下
I/TPush: [RegisterReservedInfo] Reservert info: other push token is : CN_fc0f0b38220cba7a1bcbda20857e021b
other push type: oppo
I/TPush: [PushServiceBroadcastHandler] >> bind OtherPushToken success ack with [accId = 150000****  , rsp
= 0]  token = 007a4105425********52ac1e1360c6780f3 otherPushType = oppo otherPushToken =
CN_fc0f0b3822****7a1bcbda20857e021b
```

代码混淆

```
-keep public class * extends android.app.Service
-keep class com.heytap.mcssdk.** {*;}
-keep class com.heytap.msp.push.** { *;}
```

> **说明**
> 混淆规则需要放在 App 项目级别的 proguard-rules.pro 文件中。

常见问题排查

OPPO 推送注册错误码查询方法

若观察到如下类似日志则说明 OPPO 厂商通道注册失败，开发者可以通过以下方式获取 OPPO 推送注册错误码：

```
[OtherPushClient] handleUpdateToken other push token is :  other push type: OPPO
```

推送服务 debug 模式下，过滤关键字 “OtherPush”，查看相关返回码日志（例如 `[OtherPushOppoImpl] OppoPush Register failed, code=14, msg=INVALID_APP_KEY`），并前往 厂商通道注册失败排查指南 查找对应原因，获取解决办法。

推送 OPPO 响应失败 code:30，是什么原因?

应用审核中不可发送正式消息，请前往 OPPO 平台确认推送权限审核进度。

厂商通道高级功能
角标适配指南

最近更新时间：2024-09-04 16:19:41

Android 阵营各厂商机型角标开放能力不同，移动推送对推送角标的支持程度做以下说明，供开发者参考使用。

概览

厂商	是否支持角标/红点显示	是否需要配置	适配说明
华为/荣耀	支持角标	是	请参见下文 华为手机角标适配说明
小米	支持角标	否	遵从系统默认逻辑，感应通知栏通知数目，按 1 自动增减
魅族	支持红点	否	遵从系统默认逻辑，仅支持红点展示，有通知则展示，无则不展示
OPPO	支持红点	否	• 圆点展示需由用户在通知设置中手动开启，遵从系统默认逻辑，有通知则展示，无则不展示 • 数值展示只对指定应用开启，例如 QQ、微信，需向官方进行权限申请，暂无明确适配说明
vivo	支持角标	是	请参见下文 vivo 手机角标适配说明

服务端下发角标设置

您可以通过移动推送控制台或 Push API 设置服务端下发角标：

方式1：通过控制台推送页面设置

1. 登录 移动推送控制台 。
2. 找到您需要配置的 Android 产品，单击**创建消息**，进入推送任务页面。
3. 单击**新建推送**，进入推送配置页面。
4. 在**高级设置**配置项中，开启角标数字：

方式2：通过 Push API 设置

在推送消息体 `body.message.android` 下添加字段 "badge_type"，属性如下：

参数名	类型	父项目	是否必需	默认值	描述
badge_type	Int	android	否	-1	通知角标： • -2：自动增加1，支持华为、荣耀和鸿蒙设备 • -1：不变，支持华为、荣耀、鸿蒙和vivo 设备 • [0, 100)：直接设置，支持华为、荣耀、鸿蒙和vivo 设备

> **说明**
> 不同厂商设备的角标适配能力不同，详情参考下方各厂商的角标适配说明。

消息体示例：

```
{
    "audience_type": "token",
    "expire_time": 3600,
    "message_type": "notify",
    "message": {
            "android": {
                    "badge_type": -2,
                    "clearable": 1,
                    "ring": 1,
                    "ring_raw": "xtcallmusic",
                    "vibrate": 1,
                    "lights": 1,
                    "action": {
                            "action_type": 1,
                            "activity": "com.qq.xg4all.JumpActivity",
                            "aty_attr": {
                            "if": 0,
                            "pf": 0
                    }
                    }
            },
            "title": "android test",
            "content": "android test 21"
    },
    "token_list": [
        "01f6ac091755a79015b4a30c9c4c7ddba1ea"
    ],
    "multi_pkg": true,
    "platform": "android",
}
```

终端通用 API

直接设置角标数值（ SDK v1.2.0.1 起）

直接设置角标数值；当前支持华为、OPPO、vivo 手机角标展示，其中 OPPO 需另外向厂商申请角标展示权限。

```
/**
 * @param context 应用上下文
 * @param setNum  修改角标值
 * @since v1.2.0.1
 */
XGPushConfig.setBadgeNum(Context context, int setNum);
```

示例：在收到透传消息时，调用 `XGPushConfig.setBadgeNum(context, 8)` 设置角标数值为 8。

清除角标数值（ SDK v1.2.0.1 起）

设置手机应用角标归零；当前支持华为、OPPO、vivo 手机角标展示，其中 OPPO 需另外向厂商申请角标展示权限。

```
    /**
     * @param context 应用上下文
     * @since v1.2.0.1
     */
    XGPushConfig.resetBadgeNum(Context context);
```

示例：在透传消息已阅读或打开应用时，调用 `XGPushConfig.resetBadgeNum(context)` 清除角标数值。

> **注意**
> 因厂商通道抵达的通知不支持通知清除时角标数值自动减1，建议您在恰当时机调用此接口来清除角标数值，如重新从桌面打开应用时。

华为手机角标适配说明

使用限制

华为手机角标展示支持 EMUI 8.0 及以上手机。
受限于华为手机角标能力的开放程度，在不同的推送场景下角标功能有所不同，详见下表。请按照指导方式使用华为手机角标功能。

推送形式	角标能力	实现方式
华为通道通知	支持角标自动加1、直接设置或不变，支持通知点击的自动减1，不支持通知清除的自动减1	通过管理台或 Push API 关键字设置
移动推送自建通道通知	支持角标自动加1、直接设置或不变，支持通知点击/清除的自动减1	通过管理台或 Push API 关键字设置
透传消息	开发者自行处理设置、加减逻辑	调用移动推送 SDK 开放接口

配置内容

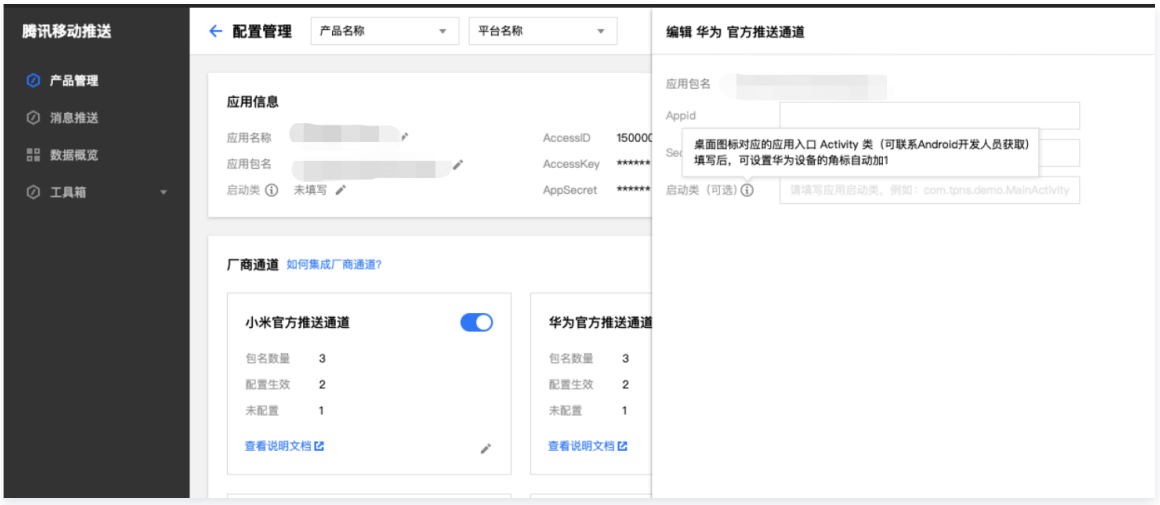
应用内角标设置权限申请

为能实现角标修改的正确效果，请首先为应用添加华为手机上的角标读写权限，具体实现为在应用 AndroidManifest.xml 文件的 manifest 标签下添加以下权限配置：

```
<uses-permission android:name="com.huawei.android.launcher.permission.CHANGE_BADGE" />
<!-- 兼容荣耀手机 -->
<uses-permission android:name="com.hihonor.android.launcher.permission.CHANGE_BADGE" />
```

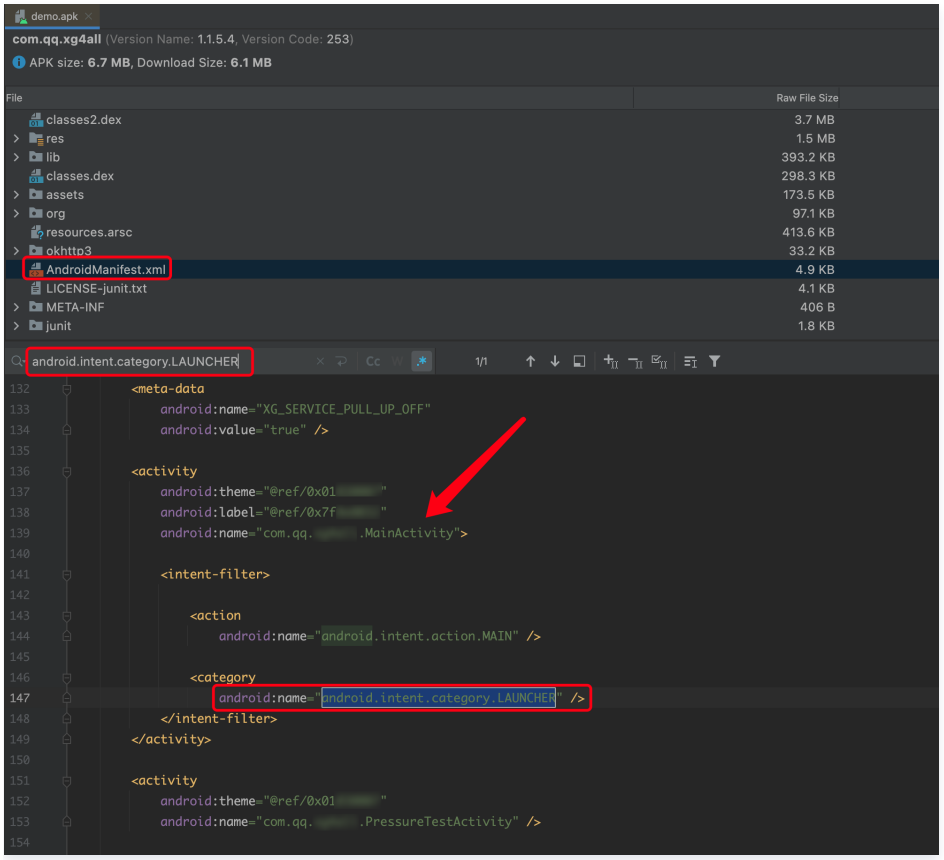
通知下发角标设置

请一定先在管理台华为通道开启及参数配置处填写桌面图标对应的应用入口 Activity 类，如“com.test.badge.MainActivity”，否则华为通道下发通知的角标设置将不生效，如图所示：



启动类名称获取指引：

请将打包完成的 apk 文件拖入 AndroidStudio，进入其中的 AndroidManifest 文件，搜索关键字 “android.intent.category.LAUNCHER”，所属的 activity.name 属性即为启动类名称。



华为手机终端设置角标自增减

华为手机支持角标自动增减1，接口如下：

```
/**
 * 华为手机角标修改接口
 *
 * @param context    应用上下文
 * @param changeNum 改变的数字，修改效果为累加；例如先前角标为5，入参为1，则角标被设置为6。
 *       当前支持 1：角标加1；-1：角标-1
 */
XGPushConfig.changeHuaweiBadgeNum(Context context, int changeNum);
```

示例：在收到透传消息时，调用 `XGPushConfig.changeHuaweiBadgeNum(context, 1)` 实现角标加1；在需要清除该消息的角标时调用 `XGPushConfig.changeHuaweiBadgeNum(context, -1)` 实现角标减1。

vivo 手机角标适配说明

使用限制

受限于 vivo 手机角标能力的开放程度，当前 vivo 手机角标仅支持直接设置角标数值，不支持自动增减；仅支持移动推送自建通道下发的通知。

推送形式	角标能力	实现方式
vivo 通道通知	不支持	不支持
移动推送自建通道通知	支持角标直接设置或不变，不支持自动增减	通过管理台或 Push API 关键字设置
透传消息	开发者自行处理设置逻辑	调用移动推送 SDK 开放接口

配置内容

应用内角标设置权限申请

为能实现角标修改的正确效果，请首先为应用添加 vivo 手机上的角标读写权限，具体实现为在应用 AndroidManifest.xml 文件的 manifest 标签下添加以下权限配置：

```
<uses-permission android:name="com.vivo.notification.permission.BADGE_ICON" />
```

手机设置内开启“桌面应用图标”

接入成功后，“桌面图标角标”默认关闭，需要用户手动开启。
开启路径：**设置 > 通知与状态栏 > 应用通知管理 > 应用名称 > 桌面图标角标**。
未成功接入“桌面图标角标”的应用，无“桌面图标角标”选项。

> **说明**
> 视 OS 版本差异，“桌面图标角标”名称可能为“应用图标标记”或“桌面角标”。

厂商通道测试方法

最近更新时间：2025-03-25 15:34:42

> **说明**
> 此测试方法适用所有版本的手机厂商通道。

1. 在您的 App 中集成移动推送V1.0.9以上版本的 SDK，并且按照厂商通道集成指南，集成所需的厂商 SDK。
2. 确认已在 移动推送控制台 > **配置管理** > **基础配置** > **厂商通道**中填写相关的应用信息。通常相关配置将在1个小时后生效，请您耐心等待，在生效后再进行下一个步骤。
3. 将集成好的 App（测试版本）安装在测试机上，并且运行 App。
4. 保持 App 在前台运行，尝试对设备进行推送（支持管理台/API推送）。
5. 如果应用收到消息，将 App 退到后台，并且停止所有 App 进程。
6. 再次进行推送，如果能够收到推送，则表明厂商通道集成成功。

> **注意：**
> 各厂商对推送有新的规则，每日单设备的公信消息接收量受限，详情可参见 厂商消息分类功能使用说明 。

厂商通道注册失败排查指南

最近更新时间：2024-09-04 14:46:41

问题描述

如您的应用接入了厂商通道，但在应用运行日志中观察到如下类似日志：

```
[OtherPushClient] handleUpdateToken other push token is :  other push type: huawei
```

表示您的应用注册该厂商通道失败。您可以通过获取厂商通道注册失败的返回码来进行问题定位和排查。

排查步骤

获取厂商通道注册返回码

移动推送 Android SDK 提供以下方式获取厂商通道注册返回码：
在应用运行日志中通过过滤关键字 `OtherPush`，找到如下类似日志来定位厂商通道注册返回码：

```
// 华为通道
// 如果过滤关键字 `OtherPush` 找不到返回码，可以过滤关键字 `HMSSDK`，并注意查看 onResult 或 onConnect 后的返回码
[OtherPushHuaWeiImpl] other push huawei onConnect code:907135702

// 小米通道
[OtherPush_XG_MI] register failed, errorCode: 22022, reason: Invalid package name: com.xxx.xxx

// 魅族通道
[OtherPush_XG_MZ] onRegisterStatus BasicPushStatus{code='110000', message='appId不合法'}

// OPPO 通道
[OtherPushOppoImpl] OppoPush Register failed, code=14, msg=INVALID_APP_KEY

// vivo 通道
[OtherPushVivoImpl] vivoPush Register or UnRegister fail, code = 10003
```

返回码问题排查

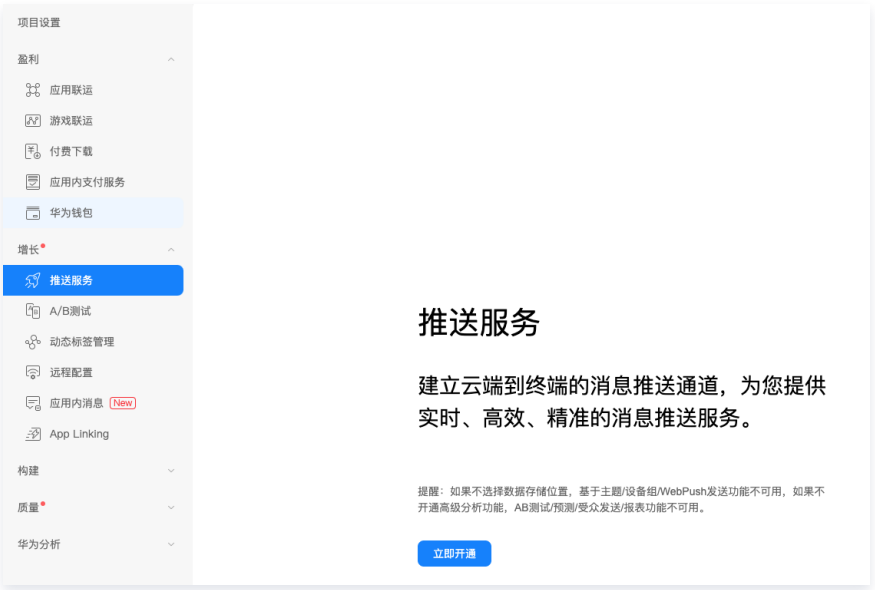
您可以前往各厂商推送的官方文档获取返回码具体含义并进行问题排查。部分常见错误码可参考下表：

厂商通道	返回码	含义	解决建议	厂商通道官方返回码地址
华为	1001	请确认手机中安装有应用 “华为移动服务” 或 “HMS-Core”，华为推送必须	前往华为应用商店下载安装应用 “HMS-Core”	华为返回码参考
	6003	应用 APK 未打签名或与华为开放平台登记签名信息不一致，华为推送必须	为 APK 文件打上签名或检查签名信息配置是否一致：华为通道配置 App 签名证书指纹	
	907135000	appId 不合法	• 检查华为推送配置文件 agconnect-services.json 文件内 appId 字段内容是否和应用包名匹配 • 检查配置文件位置是否在工程 app 模块的根目录下（和 app build.gradle 文件同级）	
	907135702	签名文件的 SHA256 值与在华为推送平台上配置的不一致	前往华为推送平台检查填写的签名文件 SHA256 值是否配置一致（华为支持添加多个）	
	907135003	ApiClient对象无效。	• 请检查手机网络是否可以正常访问互联网，或重新进行网络连接	

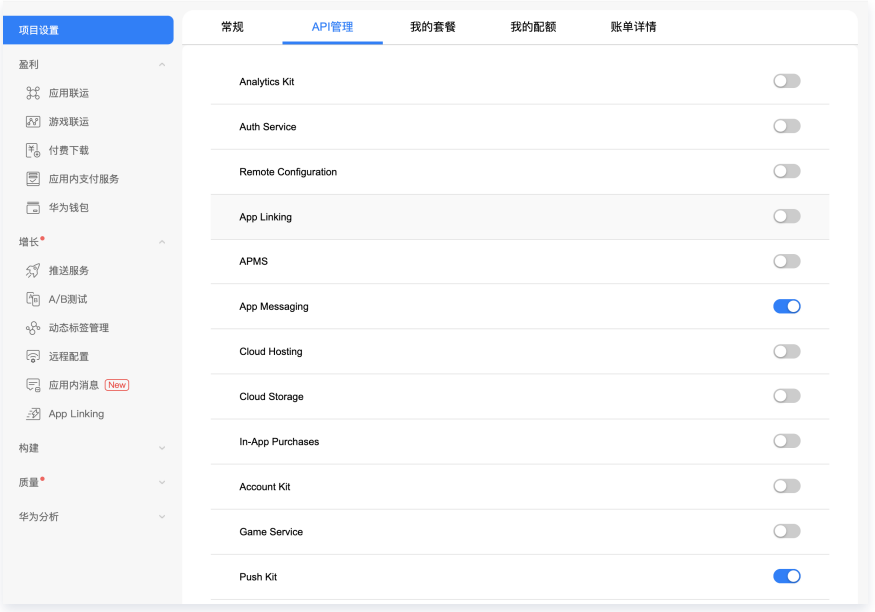
			• 此问题多和华为手机系统应用华为移动服务 HMS-Core 版本不兼容有关，您可以尝试进入华为应用商店搜索 HMS-Core 或华为移动服务，检查是否最新版本并进行升级	
荣耀	8001000	设备不支持荣耀推送	• 请使用荣耀设备测试 • 请更新包含推送服务Magic UI的版本	荣耀 返回码参考
	8001003	应用证书指纹获取失败	请配置好您的证书指纹	
小米	22006	应用程序 ID 不合法	前往小米推送平台检查应用的包名、appId、appKey 是否匹配	小米返回码参考
	22007	应用程序 Key 不合法	前往小米推送平台检查应用的包名、appId、appKey 是否匹配	
	22022	应用程序 package name 不合法	前往小米推送平台检查应用的包名、appId、appKey 是否匹配	
魅族	110000	appId 不合法	前往魅族推送平台检查应用的包名、appId、appKey 是否匹配，是 Flyme 推送平台 的应用信息	魅族返回码参考
	110001	appKey 不合法	前往魅族推送平台检查应用的包名、appId、appKey 是否匹配	
OPPO	14	无效的 AppKey 参数	请注意 setOppoPushAppId 填入的是 OPPO 的 AppKey，不是 AppId；setOppoPushAppKey 填入的是 OPPO 的 AppSecret，不是 AppKey	OPPO 返回码参考
	15	缺少 AppKey 参数	补充 AppKey 参数	
vivo	10003	App 包名与配置不匹配	前往 vivo 推送平台检查应用的包名、appId、appKey 是否匹配	vivo 返回码参考
	10004	appkey 不匹配	前往 vivo 推送平台检查应用的包名、appId、appKey 是否匹配	
	10005	appid 传入错误	前往 vivo 推送平台检查应用的包名、appId、appKey 是否匹配	

其他排查

- **华为推送需要在华为推送平台开启推送服务**
  如您在华为设备上无法获取华为 Token，但获取到厂商推送注册返回码为0，请前往 华为推送平台，进入开发 > 推送服务页面，确认应用的推送开关是否开启；进入开发 > 项目设置 > API 管理页面，确认 `Push Kit、App Messaging` 开关是否开启。
  推送服务页面显示如下：



API 管理页面显示如下：



- **小米推送需要在小米推送平台开启推送服务**
  如您未找到小米通道注册返回码，请前往 小米开放平台 > **推送运营平台**，确认应用的消息推送服务是否启用。

- **OPPO 推送需要申请推送功能开通后才能进行正式推送**
  在 OPPO 开放平台 的推送服务界面可以看到已开启服务应用和未开启服务应用，在未开启服务中单击需要申请 Push 权限的应用，进入 Push 服务并单击申请开通。
- **vivo 推送需要申请推送功能开通后才能进行正式推送**
  进入 vivo 开放平台 > **推送运营平台**，在**消息推送** > **全部应用**中，所创建应用将会列入在应用名称里，单击**应用名称**选择要申请的应用后单击**提交申请**。

> **说明**
> 部分厂商推送开关开启生效有约5分钟延迟，若开启开关后仍遇到注册失败，可以稍等片刻再进行尝试。

- **华为移动服务版本过低**
  搜索日志关键字“HMSSDK”，如观察如下类似日志，即 `connect versionCode` 小于 `connect minVersion`，表示系统应用“华为移动服务”或“HMS_Core”版本较低，请做升级后尝试重新注册。

```
I/HMSSDK_HuaweiApiClientImpl: ====== HMSSDK version: 20601301 ======
I/HMSSDK_HuaweiApiClientImpl: Enter connect, Connection Status: 1
E/HMSSDK_Util: In getHmsVersion, Failed to read meta data for the HMS VERSION.
I/HMSSDK_HuaweiApiClientImpl: connect minVersion:20600000
I/HMSSDK_HuaweiMobileServicesUtil: connect versionCode:20301306
D/HMSAgent: connect end:-1001
```

- **vivo 部分机型不支持推送服务**
  vivo 推送仅支持部分较新的机型和对应的系统及以上系统，详情请参见 vivo 推送常见问题汇总 。

厂商消息分类功能使用说明

最近更新时间：2025-06-04 17:08:52

简介

目前，厂商会逐步对 App 开发者的通知消息根据分类进行限额限频，以此保证终端用户不被过度骚扰，不同的消息分类主要通过渠道 ID（ChannelID）进行区分。移动推送综合各厂商的分类能力，支持将消息分为两类：

- 公信消息（默认）：适用于推送全员公告、运营活动、热点新闻等，多为用户普适性的内容，推送数量限制每日2/5条。
- 私信消息：适用于推送聊天消息、个人订单变化、交易提醒等与私人通知相关的内容，通知消息的推送数量不受限制。

消息分类支持在调用推送 API 时指定，同时管理台也支持消息分类。
魅族暂不支持消息分类，且不限额。

使用步骤

1. 若需要使用厂商通知消息，按以下说明申请或创建通知消息的 Channel ID：
   - OPPO 申请指南
   - 小米申请指南
   - vivo 申请指南
   - 华为使用指南
   - 荣耀使用指南

> **说明**
> 从 Android 8.0（API 级别 26）开始，弹出通知栏通知必须先为应用创建通知渠道，并为要弹出的通知分配渠道，否则通知将不会显示。通过将通知分配给特定的通知渠道，则该通知将以该通知渠道已被开启的行为功能展示在通知栏中。用户可以为应用的每个通知渠道进行个性化控制，而非直接管理应用的所有通知，例如控制每个渠道的开闭、视觉和听觉选项等。
> 一个应用可以有多个通知渠道，建议设置不超过7个通知渠道。应用的每个通知渠道按照通知渠道 ID（channel_id）区分，通知渠道以通知渠道名称（channel_name）定义的文本展示在应用的通知设置中。
> 通知渠道一旦创建，设备用户拥有完全控制权，开发者便无法更改通知行为。对同一个通知渠道 ID（channel_id）进行重复创建的代码调用，仅不同的通知渠道名称（channel_name）和渠道描述参数会生效，其他的视觉、听觉、重要性等选项无法改变。

2. 若需要使用厂商做渠道分类管理，自定义 Channel ID，从而做到根据 App 自身的业务消息类别进行消息分类，可根据不同厂商对应进行配置：

推送通道	配置说明
移动推送自建通道	• App 端，调用 Android SDK 创建 Channel ID 接口创建 Channel ID。 • 调用移动推送 服务端 API 时，指定对应的 Channel ID（不限额度）。 • 控制台创建推送任务时，选定 TPNS 消息类型标识传值。
华为	• 在华为管理台 申请自分类权益，自分类权益生效后，应用的推送消息将根据 hw_category 字段进行归类。 • 调用移动推送 服务端 API 时，指定 hw_category 参数。 • 华为控制台创建推送任务时，选定华为消息类型标识参数传值。 • 华为的 ChannelID 作为自定义的渠道策略展示消息提醒方式，不作消息分类。
小米	• 在小米开放平台管理台上申请 Channel ID 或通过小米服务端 API 创建。 • 调用移动推送 服务端 API 时，指定对应的 Channel ID。 • 控制台创建推送任务时，选定小米渠道 ID 参数传值。
OPPO（新/旧分类）	1.新分类方式 • OPPO 管理台按旧分类方式申请通道，请参照旧 私信通道使用 规范。 • 调用移动推送 服务端 API 时，指定对应的 oppo_category 分类参数。 2.旧分类方式 • App 端，调用 Android SDK 创建 Channel ID。 • 在 OPPO 管理台申请登记该 Channel ID，保持一致性。 • 调用移动推送 服务端 API 时，指定对应的 Channel ID参数。 • 控制台创建推送任务时，选定 OPPO 渠道 ID 参数传值，新分类暂不支持控制台推送。

魅族	无 Channel 相关说明。
vivo	• 支持配置使用 vivo 系统消息/运营消息，不支持自定义通知渠道 Channel 。 • 无需单独申请消息分类，可参见 vivo 消息分类场景 直接使用。 • 调用移动推送 服务端 API 时，指定 vivo_category 参数并正确赋值。 • 控制台创建推送任务时，选定vivo_category 参数传值。
荣耀	• 支持配置使用 荣耀 服务通讯/资讯营销消息，不支持自定义通知渠道 Channel 。 • 无需单独申请消息分类，可参见 荣耀消息分类场景 直接使用。 • 调用移动推送 服务端 API 时，指定 honor_importance 参数正确赋值。 • 暂不支持控制台推送。

3. 若既不需要使用厂商通知消息，也不需要自定义 Channel ID，则无需做任何处理，移动推送会为 App 的所有消息指定一个默认的 Channel ID，消息归到默认类别中。
4. 在控制台创建推送任务需要使用渠道分类时，请打开 厂商通道分类 开关指定对应的渠道传值。

OPPO 消息分类申请指南

OPPO 新消息分类介绍（新）

OPPO PUSH 将对消息发送通道和消息分类进行调整优化。根据消息的内容，将通知分类为**通讯与服务**和**内容与营销**两个大类别，并固定创建对应通道，根据不同类别消息明确发送的通道、内容、发送量级和提醒方式。
接入新消息分类能力后，当应用通知开关为开启状态时，二级开关默认开启（当前为默认关闭）。规范使用新消息分类能力，提升用户对不同消息类型的提醒体验。

> **说明：**
> 新消息分类对通讯与服务类消息具备更强的提醒能力，提升了用户消息接收体验，同时平台在逐步加强对通道的监管力度，会逐步收回自建通道能力，建议开发者按照平台规则做新消息分类能力接入。

消息类型	定义范围	推送内容方向	消息提醒方式	推送量级
通讯与服务	• 用户间的聊天消息、通话等信息。 • 与用户自身息息相关的重要通知提醒，用户对接收此类消息有预期。	• 用户间点对点聊天消息（或私信）、群聊天消息、视频语音提醒。 • 个人账号与资产变化；个人设备提醒；个人订单/物流状态变化等；更详细场景请参见 OPPO 分类细则 。	• 默认为**通知栏、锁屏**。 • 可升级为**通知栏、锁屏、横幅、铃声、震动**强提醒方式（需申请，申请流程请参见 OPPO 通道提醒指引 ）	发送量级和接收量级均不限
内容与营销	开发者主动向用户发送的对内容或产品推广的通知	内容推荐、平台活动、社交动态等。	仅下拉通知栏展示	限制每日推送量级与单用户接收条数，具

				体请参见 推送服务受限说明 。

OPPO 通讯与服务申请

参照旧 私信通道使用 规范，开发者如有**通讯与服务**消息类别发送诉求，则走线上权限申请，申请通过后自动增加**通讯与服务**消息类别。（如之前已开通私信通道权限，则无需再申请。如有新增消息模板或强提醒诉求则再申请。）

OPPO 新通道权限开启

> **警告：**
> 使用OPPO新分类，需要打开新渠道权限。新通道权限开关打开后不可逆有风险，请谨慎操作！（如下图配置）详细说明请参见 OPPO 分类细则 。

OPPO 通讯与服务使用

> **注意：**
> 因 OPPO 新消息分类能力当前支持系统版本OS13及以上，在使用新消息分类能力时，必须同时填写 oppo_category 和选择原有通道渠道，来兼顾低版本用户的触达，否则会影响低版本用户接收。

自分类消息目前仅支持 API 进行下发，控制台暂不支持，可通过以下方式使用：
在 Rest API 请求参数 `Android` 结构体中设置 `oppo_category` 参数，可实现 OPPO 新自分类消息下发，若申请强提醒消息，再设置 `oppo_notify_level`参数实现， 详情请参见 PushAPI 参数说明。
推送示例如下：

```
{
    "audience_type": "account",
    "account_list": ["123"],
    "message_type": "notify",
    "message": {
    "title": "测试标题",
    "content": "测试内容",
    "android": {
        "oppo_ch_id": "私信通道id",
        "oppo_category":"IM",
        "oppo_notify_level":16}
  }
}
```

OPPO 通知渠道介绍（旧）

OPush 平台上默认的是公信通道，目前在原有基础上新增“私信”通道，对单个用户推送个性化信息时，不再受推送数量限制。以下是“公信”和“私信”的对比：

类型		公信	私信
推送内容		热点新闻、新品推广、平台公告、社区话题、有奖活动等，多用户普适性的内容。	个人订单变化、快递通知、订阅内容更新、评论互动、会员积分变动等，与单个用户信息强相关的内容。
单用户推送限制（条/日）	新闻类（三级分类为新闻类）	5条	不限量。
	其他应用类型	2条	不限量。
推送数量限制		所有公信类通道共享推送次数，当日达到次数限制后，所有公信类通道将不能再推送消息，目前当日推送数量为：累计注册用户数 * 2 。未上架应用市场的测试权限仅提供1000条/日	不限量。
配置方式		默认。	需要在 OPPO PUSH 运营平台上登记该通道，并将通道对应属性设置为“私信” 。

警告：
OPPO 官方提醒：切记！一定不要利用私信通道用于普适性消息推送（例如热点新闻、新品推广等），后台会实时监控，如违反运营规则，OPush 有权关闭您的私信通道权限。由此产生的后果，如调用接口异常，或使用私信通道发送的消息没到达用户等，由业务方自行承担。

OPPO 私信通道申请

1. 进入 OPPO 开放平台，在**应用配置** > **新建通道**中新建通道，通道 ID 与通道名称必填且需要与应用客户端保持一致，其他选项可不填。

注意
- 通道 ID 一旦确定下来不能随意变更或删除。
- OPPO 私信通道必须**邮件申请**成功后才能生效，请参见 OPPO 邮件私信模板 发送邮件给 OPush 平台，等待OPPO审核通过后即可使用。

OPPO 私信通道使用

1. 客户端创建通知渠道（必须与 OPPO 平台申请的渠道 ID 保持一致），请选择以下任意一种方式创建：

1.1 使用 Android API 创建通知渠道，详情请参见 Android 官方文档 创建和管理通知渠道 。

1.2 使用移动推送 SDK（1.1.5.4及以上的版本）创建通知渠道，详情请参见文档 创建通知渠道 。

2. 支持 Rest API 和控制台创建推送。

2.1 在 Rest API 请求参数 Android 结构体中设置 `oppo_ch_id` 参数，可实现根据渠道ID分类下发，具体参见 PushAPI 参数说明 。
推送示例如下：

```
{
    "audience_type": "token",
    "token_list": ["**********"],
    "message_type": "notify",
    "message": {
    "title": "测试标题",
    "content": "测试内容",
    "android": {
        "oppo_ch_id": "私信通道id"}
  }
}
```

2.2 控制台推送，打开厂商通道分类功能，选定OPPO渠道ID参数传值。

小米通知渠道申请指南

小米通知渠道介绍

小米推送（Mipush）的通知渠道分为“私信消息”和“公信消息”两类，不同类别对应不同的权限，详情请参见 小米推送消息分类新规 。

- 公信消息适用于推送热点新闻、新品推广、平台公告、社区话题、有奖活动等，多为用户普适性的内容。
- 私信消息适用于推送聊天消息、个人订单变化、快递通知、交易提醒、IOT 系统通知等与私人通知相关的内容，通知消息的推送数量不受限制。
  小米推送对推送消息数量、推送速率 QPS 进行了统一管理，详情请参见 小米推送消息限制说明 。

公信消息与私信消息限制说明：

消息类型	消息内容	推送数量限制	用户接收数量限制	申请方式

公信消息	热点新闻、新品推广、平台公告、社区话题、有奖活动等，多用户普适性的内容。	2-3倍，具体规则请参见“公信限制规则”。	单个应用单个设备单日5-8条，具体规则请参见“公信限制规则”	需在小米推送平台申请，详情请参见 channel 申请及接入方式。
私信消息	聊天消息、个人订单变化、快递通知、交易提醒、IoT系统通知等与私人通知相关的内容。	不限量	不限量	

注意：
- 必须要收到审核通过的邮件私信通道才生效，且在推送时必须传递 channel_id 字段，否则厂商会返回推送失败返回错误码：27001。
- 为了规范使用小米推送，请遵守小米的 推送运营规则 。

小米额外提升推送量级申请

若特殊情况需要额外提升推送量级，开发者可以向小米推送进行申请，详情请参见 小米推送通知消息 。

小米通知消息使用

支持 Rest API和 控制台创建推送。

1. 推送在 Rest API 请求参数 `Android` 结构体中设置 `xm_ch_id` 参数，可实现小米通知渠道下发，详情请参见 PushAPI 参数说明。
   推送示例如下：

```
{
    "audience_type": "token",
    "token_list": ["************************"],
    "message_type": "notify",
    "message": {
     "title": "小米通知消息",
     "content": "测试内容",
     "android": {
        "xm_ch_id": "小米通知消息的channel_id"
     }
       }
}
```

2.控制台推送，选定小米渠道ID参数传值。

vivo 系统消息申请指南

vivo 消息分类介绍

vivo 消息分类功能将推送消息类型分为运营消息和系统消息。
为提升用户消息通知体验，营造良好推送生态，vivo 推送服务于2023年4月3日起，针对不同应用类别的消息进行统一管理，开发者需根据自身应用的通知场景，将消息内容按照对应消息类别发送。详情请参见 vivo 的 推送消息限制说明 。

消息类别	划分原则	允许发送的内容	应用类别	推送数量限制	用户接收数量限制	提醒方式	接入方式
系统消息	用户对收到此类消息有预期，并需要及时知道的消息，如果错过可能会导致不良影响。	• 即时消息 • 账号与资产 • 日程待办 • 设备信息 • 订单与物流 • 订阅提醒	/	3倍通知开启有效用户数（可申请消息不限量权限）	无限制	• 悬浮 • 锁屏 • 响铃 • 震动	开发者接口设置参数（vivo_category）,开发文档请参见《服务端 API 接口文档》
运营消息	用户对收到此类消息无预期，关注程度较低。	• 新闻 • 内容推荐 • 运营活动 • 社交动态	新闻资讯类	3倍通知开启有效用户数	5条	无提醒，仅在下拉通知栏展示	
			其他类	2倍通知开启有效用户数	2条		

推送数量超过当日限制时，返回错误码10070或10073：

- 10070：运营消息发送量总量超出限制。
- 10073：系统消息发送量总量超出限制，如需要申请无限量推送可参考的vivo系统消息申请 。

说明

- 2020年6月1日前，无论是否接入消息分类，频控规则不变，均按每个应用单用户“公共类消息（全推，群推，标签推）”每天接收上限为5条，不限制单推条数。2020年6月1日起，频控规则变更为按每个应用单用户“运营消息”接收条数上限5条进行频控，若出现用户体验类投诉，将会根据实际情况调整条数。
- Funtouch OS_10 及以上版本没有消息盒子，应用不存活时窄条展示，具体样式以实际为准。
- 若消息内容满足订阅类消息条件，可按特殊场景消息归档模板提供相应材料向 vivo 申请，申请方法详见下方 vivo 系统消息申请 。
- 若某 vivo 用户当前接收运营消息超过2/5条，则当天触发限额后的运营消息均会通过移动推送自建通道下发，不再通过 vivo 通道下发。
- 为避免造成用户打扰，目前vivo手机接收的消息为7:00-23:00，服务器允许推送时间为7:00-23:00，系统消息不受此时间限制，系统消息申请详情请参见 vivo 系统消息申请指南 。
- vivo消息分类不需要另外申请，可直接使用。

警告：

vivo 推送平台将按消息分类标准，对系统消息进行每日巡检，巡查开发者以系统消息渠道发送运营消息的情况，并按违规程度及频次进行相应处罚，最高将关闭消息推送功能。

vivo 系统消息使用

支持 Rest API和 控制台创建推送。

1.如在 Rest API 请求参数 `Android` 结构体中设置 `vivo_category` 参数为"IM"，可实现 vivo 系统消息下发，具体参见 PushAPI 参数说明 参数说明。

说明

- 2023年4月3日起由 `vivo_ch_id` 调整为 `vivo_category` 字段,若您在二级分类方案调整前，已经使用 vivo_ch_id 进行消息分类，请您尽快完成新方案的适配，补充 vivo_category 的传值。按以上表格申请 vivo 二级分类新方案 vivo 系统消息申请 。

推送示例如下：

```
{
    "audience_type": "token",
    "token_list": ["************************"],
    "message_type": "notify",
    "message": {
     "title": "vivo系统通知",
     "content": "测试内容",
     "android": {
        "vivo_category": "IM"
     }
```

```
    }
}
```

2.控制台推送，指定 vivo_category 参数并正确赋值。

华为消息分类使用指南

华为消息分类介绍

华为推送从 EMUI 10.0版本开始将通知消息智能分成两个级别：**服务与通讯**和**资讯营销**。EMUI 10.0之前的版本没有对通知消息进行分类，只有一个级别，消息全部通过“默认通知”渠道展示，等价于 EMUI 10.0的服务与通讯。资讯营销类消息的每日推送数量自2023年01月05日起根据应用类型对推送数量进行上限管理，服务与通讯类消息每日推送数量不受限。
华为回执 256 表示当日的发送量超出资讯营销类消息的限制，请您调整发送策略，各消息类型详情请参见华为的 推送数量关系细则 。
如图所示：

不同消息级别呈现样式对比：

消息分类	提醒方式	类型说明	推送数量限制

服务与通讯消息	锁屏、铃声、振动	社交通讯：即时聊天，音频、视频通话。 服务提醒：订阅，出行，健康，工作事项提醒，账号动态，订单&物流，财务，设备提醒，系统提示，邮件，闹钟/计时器，秒表，进度，位置共享。 具体类别请参见 服务与通讯类消息场景说明 。	无限制。	
资讯营销消息	静默通知，仅在通知栏展示消息	内容资讯：内容推荐，新闻，财经动态，生活资讯，社交动态，调研，其他。 营销活动：产品促销，功能推荐，运营活动。 具体类别请参见 资讯营销类消息场景说明 。	新闻类（需具备《互联网新闻信息服务许可证》）	5条
			其他类	2条 根据应用类别限制每日推送数量，具体要求参见 不同应用类别的推送数量上限要求 。

若您希望服务与通讯类消息也按照静默的方式发送，可以添加 `hw_importance` 字段且传值为“1”，详情请参见 PushAPI 参数说明。

警告：
遵守《华为消息分类标准》，如出现分类错误，违反消息分类标准的场景，将被华为判为违规。

华为自分类消息权益申请

华为通知消息自分类权限需要申请并激活后才能生效，申请流程详情请参见 华为自分类权益申请 。

注意
- 若应用没有自分类权益，则应用的推送消息将通过智能分类进行自动归类。
- 若应用有自分类权益，将信任开发者提供的分类信息，消息不经过智能分类。

华为自分类消息使用

支持 Rest API和 控制台创建推送。

> **注意：**
> hw_category字段取值为大写的英文单词且仅可填写在分类权益页面中已在华为平台审批通过的消息类型所对应的category值 。

1.在 Rest API 请求参数 `Android` 结构体中设置 `hw_category` 参数，可实现自分类消息下发，详情请参见 PushAPI 参数说明。
推送示例如下：

```
{
    "audience_type": "token",
    "token_list": ["**********"],
    "message_type": "notify",
    "message": {
        "title": "账号下线：",
        "content": "您的账号出现异地登录，已经下线。",
        "android": {
            "hw_category":"VOIP"
        }
    }
}
```

2.华为自分类控制台推送，指定 hw_category 参数并正确赋值。

华为通知渠道创建

华为推送支持应用自定义通知渠道分组，客户端创建通知渠道，请选择以下任意一种方式创建：
1. 使用 Android API 创建通知渠道，详情请参见 Android 官方文档 创建和管理通知渠道 。
2. 使用移动推送 SDK（1.1.5.4及以上的版本）创建通知渠道，详情请参见文档 创建通知渠道 。

华为通知渠道使用

目前自定义渠道只能通过 Rest API 进行下发，控制台暂不支持，在您创建完成通知渠道后，可通过以下方式使用：
在 Rest API 请求参数 `Android` 结构体中设置 `hw_ch_id` 参数，可实现华为通知渠道分组，详情请参见 PushAPI 参数说明。

> **注意**
> - 如果您的应用在华为推送控制台申请开通华为推送服务时，选择的数据处理位置为中国区，自定义渠道功能将不再适用于您的应用。您的推送消息将按照智能分类系统或消息自分类权益确认的消息级别，归类为服务与通讯类或资讯营销类消息。详见 自定义通知渠道 。
> - 自定义渠道功能需要您的应用具有消息自分类权益，请参见上文进行申请。

推送示例如下：

```
{
    "audience_type": "token",
    "token_list": ["************************"],
    "message_type": "notify",
    "message": {
        "title": "华为通知消息",
        "content": "测试内容",
        "android": {
            "hw_ch_id": "华为通知消息的channel_id"
        }
    }
}
```

荣耀消息分类使用指南

荣耀消息分类介绍

荣耀推送服务将根据应用类型、消息内容和消息发送场景，将推送消息分成服务通讯和资讯营销两大类别，资讯营销类消息的每日推送数量根据应用类型对推送数量进行上限管理，服务与通讯类消息每日推送数量不受限，详细说明请参见 荣耀推送数量管理细则 。
根据消息分类，对不同类别消息的默认展示方式、推送数量限制、消息样式进行差异化管理，具体如下：

消息分类	提醒方式	类型说明	推送数量限制	
服务与通讯消息	锁屏展示+下拉通知栏展示，支持铃声、震动	社交通讯：即时聊天，音频、视频通话。 服务提醒：订阅，出行，健康，工作事项提醒，账号动态，订单&物流，财务，设备提醒，系统提示，邮件。 具体类别请参见 服务与通讯类消息场景说明 。	无限制	
资讯营销消息	静默通知，仅在下拉通知栏展示	内容资讯：内容推荐，新闻，财经动态，生活资讯，社交动态，调研，其他。 营销活动：产品促销，功能推荐，运营活动。 具体类别请参见 资讯营销类消息场景说明 。	新闻类（需具备《互联网新闻信息服务许可证》）	5条
			其他类	2条 根据应用类别限制每日推送数量，具体要求参见 不同应用类别的推送数量上限要求

警告：

请遵守荣耀的推送分类规则，违规者将受到荣耀对应的处罚，违规行为及相应的处罚措施请参见 消息违规处罚标准 。

荣耀自分类消息权益申请

注意：

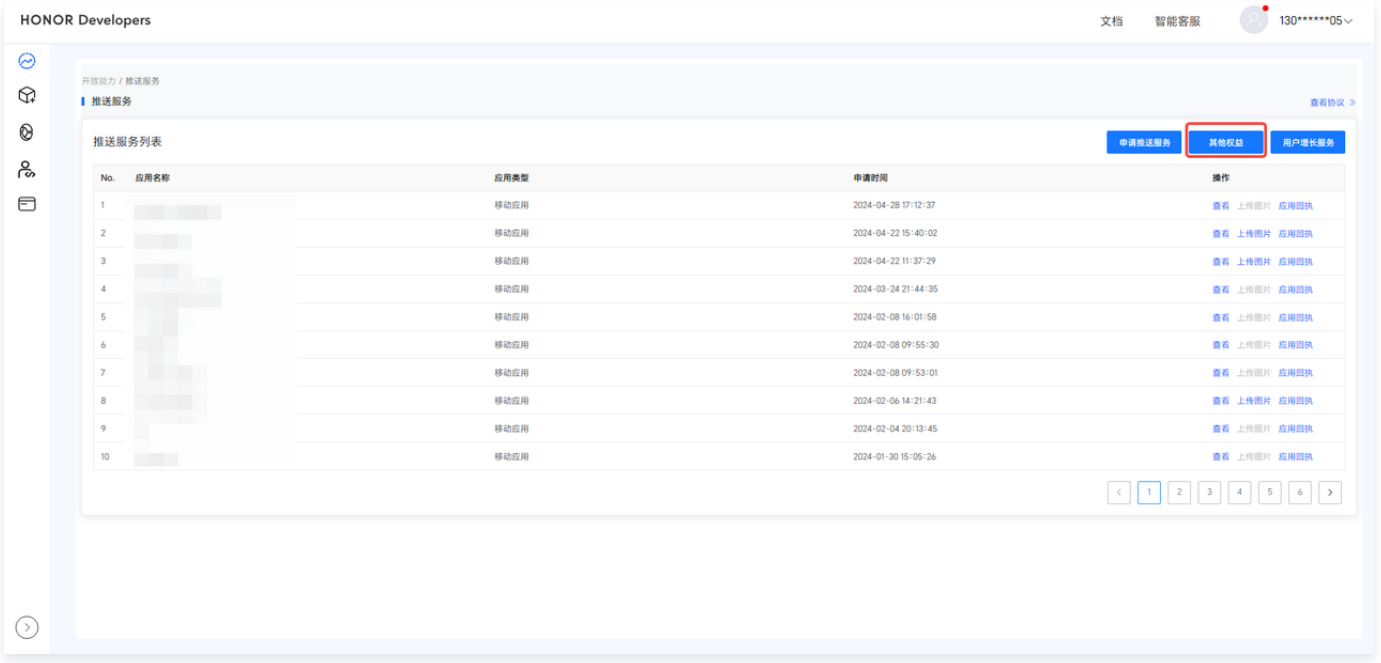
荣耀推送服务的消息分类方式统一为消息自分类，您可通过申请自分类权益，对推送服务消息进行分类管理。

- 荣耀推送服务接受开发者自分类权益的申请。当您申请成功后，允许开发者根据《荣耀消息分类标准》，自行对消息进行分类。
- 未接入消息自分类的应用，消息通知类型将会默认归为资讯营销类消息，资讯营销消息每日推送数量上限要求请参考《荣耀推送数量管理细则》。

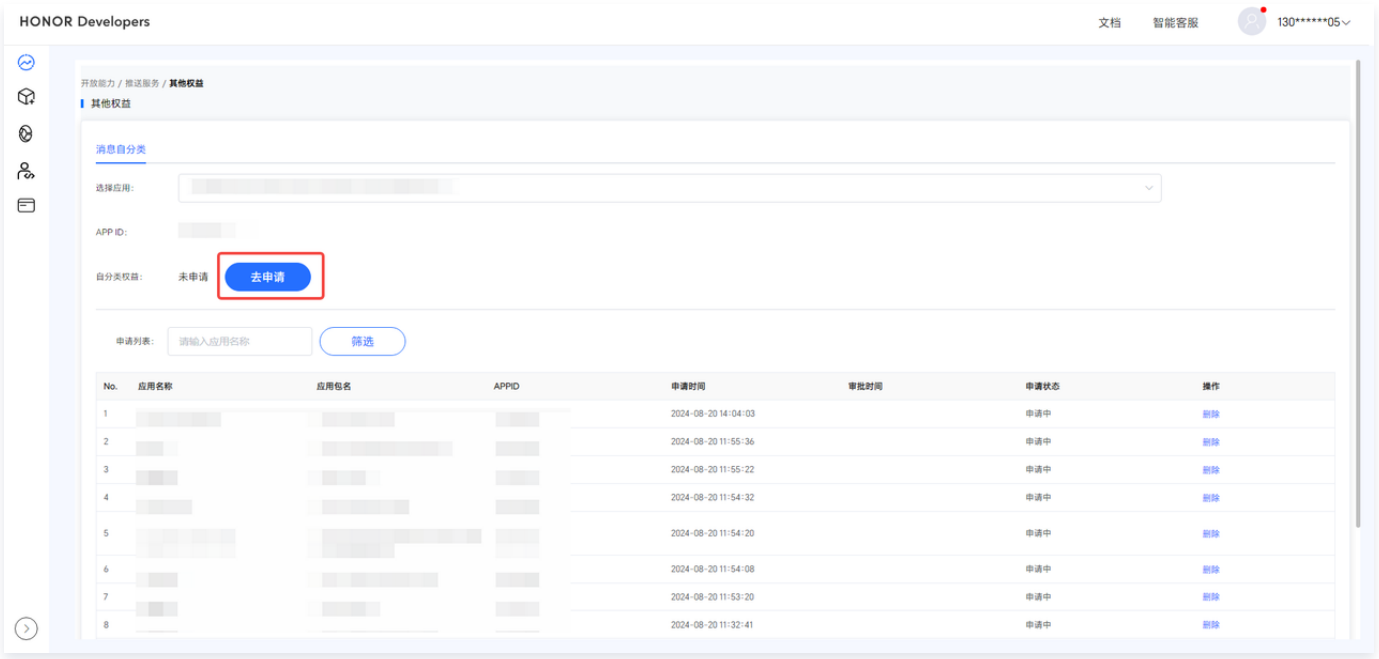
荣耀通知消息自分类权限申请成功后立即生效，申请流程详情请参见 荣耀自分类权益申请 。

申请流程：

1. 登录 荣耀开发者服务平台 ，单击**推送服务**。
2. 在推送服务列表右侧单击**其他权益**。



3. 开发者选择对应应用，单击自分类权益后的“**去申请**”。



4. 申请须知勾选同意，单击**确定**。

5. 审核周期为15个工作日内，您可以在荣耀开发者服务平台**申请状态**中查看申请进度。

荣耀自分类消息使用

> **说明：**
> - honor_importance 字段值为“1”时：表示消息为资讯营销类（限额），默认展示方式为静默通知，仅在下拉通知栏展示。
> - honor_importance 字段值为“2”时：表示消息为服务通讯类（无限额），默认展示方式为锁屏展示+下拉通知栏展示。

自分类消息仅支持 API 进行下发，控制台暂不支持，可通过以下方式使用：
在 Rest API 请求参数 `Android` 结构体中设置honor_importance参数，可实现荣耀自分类消息下发，详情请参见 PushAPI 参数说明。
推送示例如下：

```
{
    "audience_type": "token",
    "token_list": ["************************"],
    "message_type": "notify",
    "message": {
        "title": "荣耀: ",
        "content": "自分类推送。",
        "android": {
            "honor_importance":2
        }
    }
}
```

厂商通道抵达回执获取指南

最近更新时间：2025-03-25 15:34:42

为了帮助客户全链路分析推送效果，移动推送提供厂商通道抵达数据展示。国内不同厂商通道对抵达数据回执的支持程度不同，部分通道抵达数据不能直接获取，需要开发者进行相应配置。
配置成功后，开发者可在移动推送控制台推送详情中查看推送转化数据，也可通过移动推送 Rest API 获取推送转化数据。

概览

厂商通道	是否支持抵达	是否需要配置
华为通道	是	是
荣耀通道	是	是
魅族通道	是	是
小米通道	是	否
OPPO 通道	是	否
vivo 通道	是	是
FCM 通道	是	否

说明

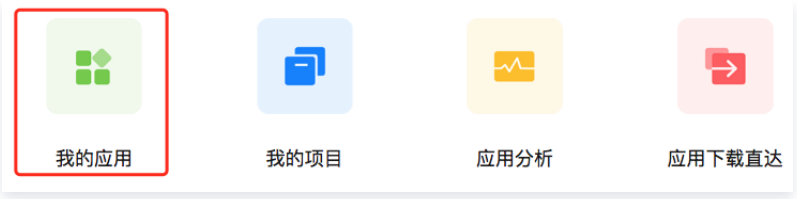
- 厂商通道抵达回执数据仅供参考。
- 建议配置 vivo 抵达回执，vivo 将消息状态以回执消息形式发送给您的应用回执服务端。

华为厂商通道回执配置指引

完成华为厂商通道 SDK 集成后，需要开发者在华为开放平台开通并配置v1版本的消息回执，才能获取到华为通道抵达数据。具体配置方法可参见华为开放平台消息回执，配置流程如下：

开通回执权益

1. 登录 AppGallery Connect 网站，选择**我的应用**。



2. 选择需要开通服务的应用所属产品的名称，进入应用信息页面。

3. 在“应用信息”页面，选择**全部服务 > 推送服务**。

4. 在“推送服务”页面，找到**应用回执状态**，单击**开通**。

回执参数配置

1. 配置消息回执地址。请在 产品管理 页面查看您的应用的服务接入点，并复制该应用的 AccessID，选择对应服务接入点的回执地址进行配置：

服务接入点	回执地址
广州服务接入点	https://stat.tpns.tencent.com/log/statistics/hw/AccessID
中国香港服务接入点	https://stat.tpns.hk.tencent.com/log/statistics/hw/AccessID
新加坡服务接入点	https://stat.tpns.sgp.tencent.com/log/statistics/hw/AccessID
上海服务接入点	https://stat.tpns.sh.tencent.com/log/statistics/hw/AccessID

> **说明**
> AccessID 替换为您应用的 AccessID。例如应用为广州服务接入点，则回执地址配置为：`https://stat.tpns.tencent.com/log/statistics/hw/1500016691`。

2. 根据您的应用服务接入点，下载对应的 HTTPS 证书，下载完成后，用文本打开该证书，将内容复制到文本框中。

服务接入点	下载地址
广州服务接入点	点击下载

中国香港服务接入点	点击下载
新加坡服务接入点	点击下载
上海服务接入点	点击下载

3. 配置用户名和密钥（非必填）进行身份验证。
4. 单击**测试回执**，可以对回执地址进行功能测试。

> **注意**
> - 目前单击**测试回执**，会提示“测试回调地址失败”，请忽略并直接单击**提交**。
> - 请配置华为的 v1 版本的抵达回执。

5. 单击**提交**，即可完成服务的开通。

魅族厂商通道回执配置指引

完成魅族厂商通道 SDK 集成后，需要开发者在 Flyme 推送平台中新建回执，并在移动推送控制台完成激活后，才能获取到魅族通道抵达数据，配置方法如下：

> **注意**
> 以下两步缺一不可，否则可能导致魅族通道下发失败。

配置回执

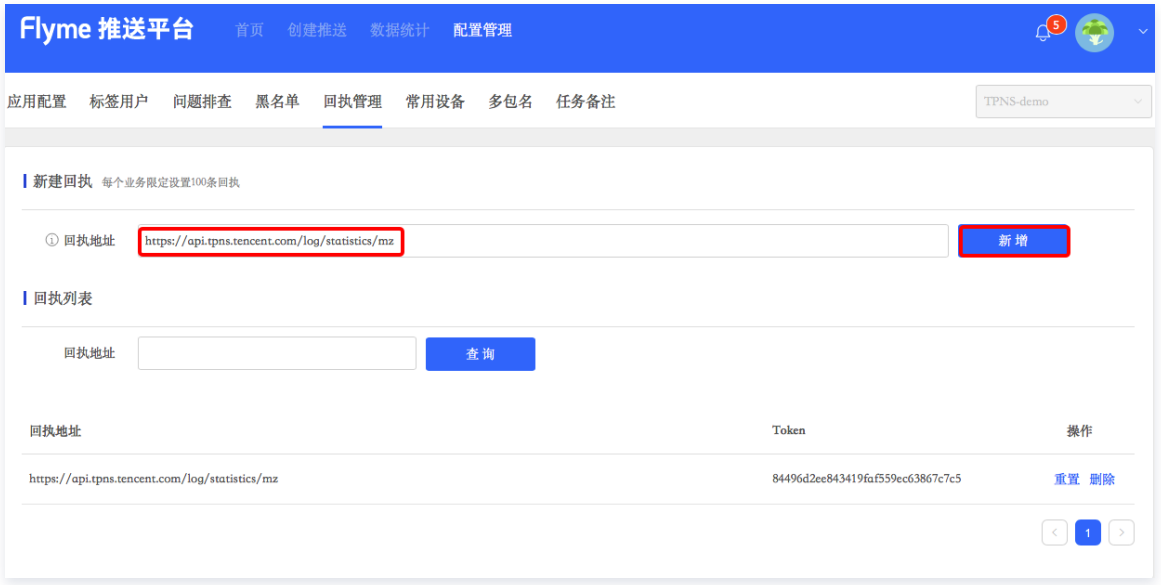
1. 登录 Flyme 推送平台 ，选择需要配置回执的应用，单击**打开应用**。

2. 在推送通知页面，单击**配置管理** > **回执管理**。
3. 在**新建回执**中填写回执地址。请在 产品管理 页面查看您的应用的服务接入点，并选择对应服务接入点的回执地址进行配置：

服务接入点	回执地址
广州服务接入点	https://api.tpns.tencent.com/log/statistics/mz
	https://stat.tpns.tencent.com/log/statistics/mz
中国香港服务接入点	https://stat.tpns.hk.tencent.com/log/statistics/mz
新加坡服务接入点	https://stat.tpns.sgp.tencent.com/log/statistics/mz
上海服务接入点	https://stat.tpns.sh.tencent.com/log/statistics/mz

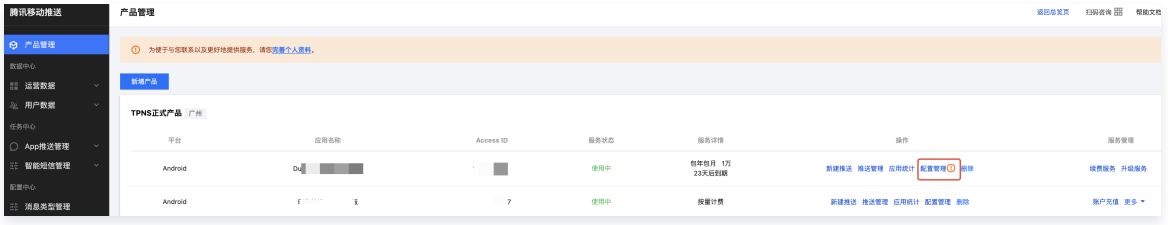
> **注意**
> 广州服务接入点的两个回执地址都需要填写。

4. 填写回执地址后，单击右侧**新增**，**回执列表**中正确显示新建的回执即完成配置。



激活回执

1. 进入 **产品管理** 页面，选择需要激活的应用，单击**配置管理**。



2. 在配置管理页面，**厂商通道** > **魅族官方推送通道** > 单击编辑图标。

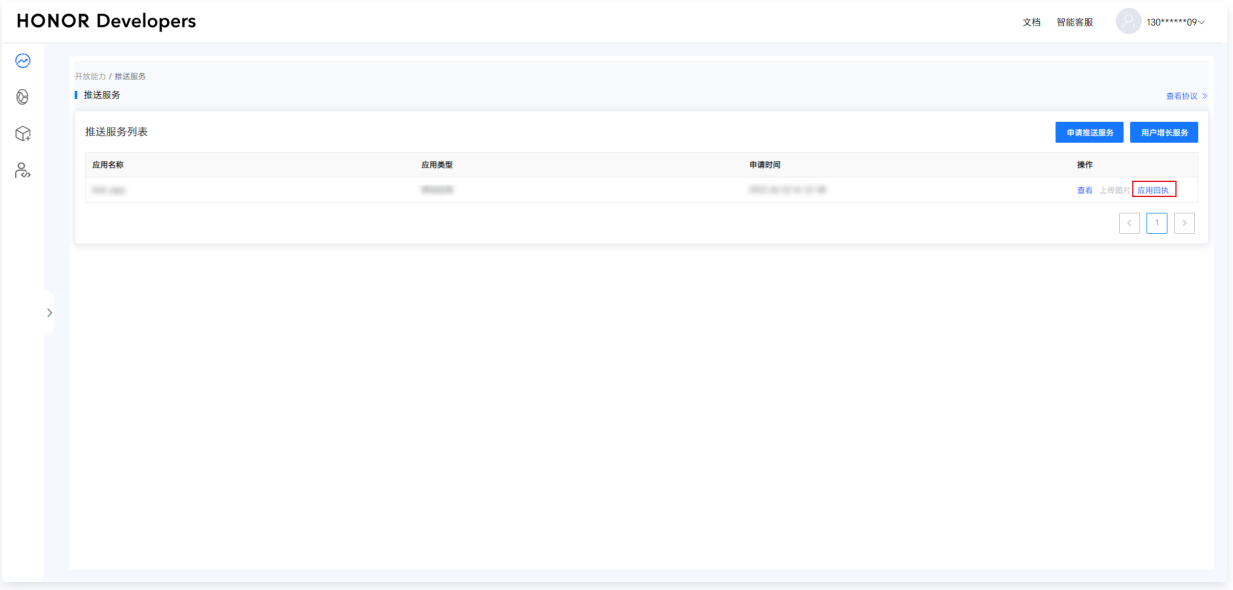
3. 在编辑魅族官方推送通道页面，按照提示单击**点击激活**。

荣耀厂商通道回执配置指引

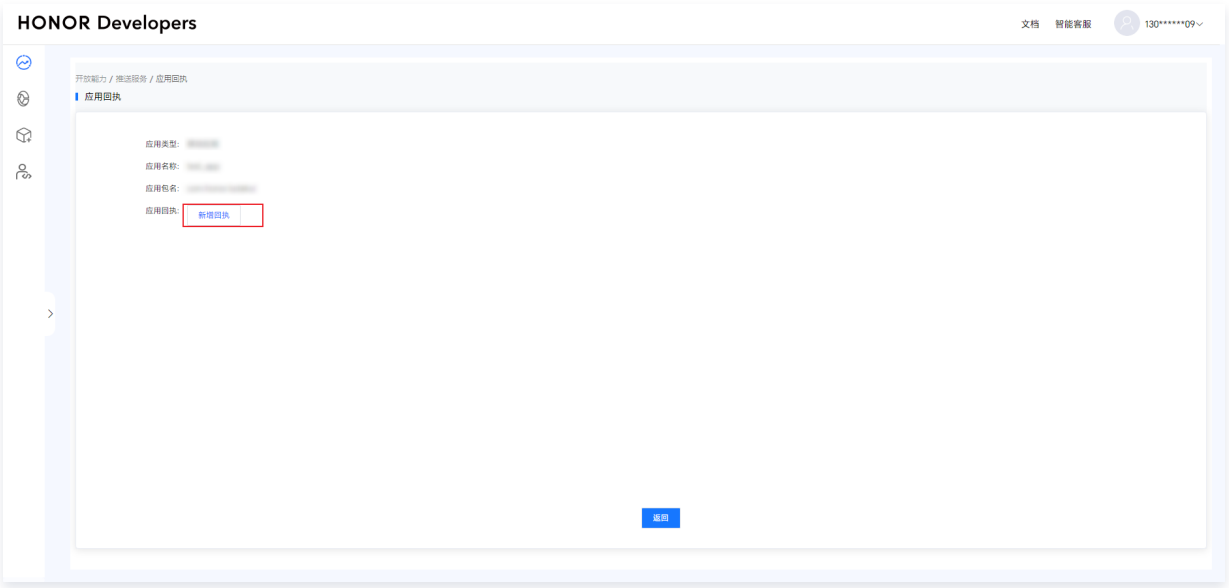
完成荣耀厂商通道 SDK 集成后，需要开发者在荣耀平台开通并配置消息回执，才能获取到荣耀通道抵达数据，具体配置方法可参考荣耀平台 荣耀推送回执指南 。

开通回执权益

1. 登录 荣耀开发者服务平台 。
2. 选择**管理中心** > **生态服务** > **开发服务** > **推送服务** ，进入推送服务页面。
3. 选择需要配置回执的应用，点击 应用回执 ，进入应用回执页面。



4.单击**新增回执** ，进入回执配置页面。



回执参数配置

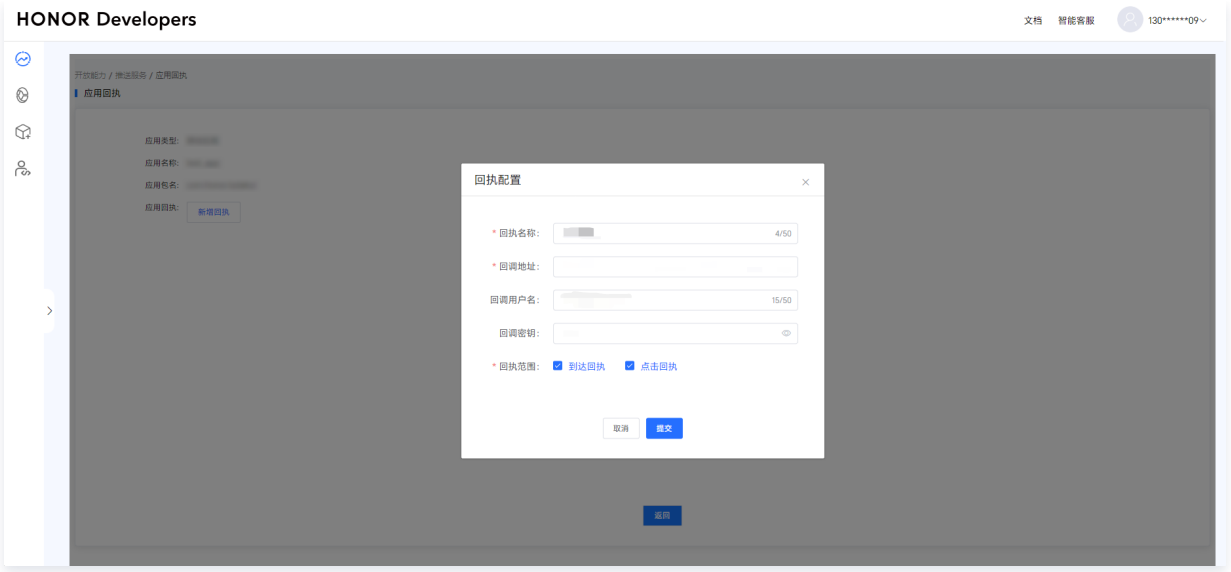
1. 请在 **产品管理** 页面查看您的应用的服务接入点，并复制该应用的 AccessID，选择对应服务接入点的回执地址进行配置。

服务接入点	回执地址
广州服务接入点	https://stat.tpns.tencent.com/log/statistics/honor/AccessID
中国香港服务接入点	https://stat.tpns.hk.tencent.com/log/statistics/honor/AccessID
新加坡服务接入点	https://stat.tpns.sgp.tencent.com/log/statistics/honor/AccessID
上海服务接入点	https://stat.tpns.sh.tencent.com/log/statistics/honor/AccessID

说明

AccessID 替换为您应用的 AccessID。例如应用为广州服务接入点，则回执地址配置为：`https://stat.tpns.tencent.com/log/statistics/honor/1500016691`。

2. 进入回执配置页面，需要配置参数，*为必填项。
3. 单击**提交**，即可完成服务的开通。



vivo 厂商通道回执配置指引

为协助开发者了解推送数据折损情况，优化推送体验，vivo 推送平台上线新回执能力。新回执补充了未达到、未展示回执数据，支持单推、群推回执，开发者可参照以下步骤接入。

> **注意：**
> - 新老回执不可同时使用，使用新回执后，老回执将失效。
> - vivo 回执功能暂未完全对外开放，可提供您的账号UIN加白名单即可配置回执。

完成 vivo 厂商通道 SDK 集成后，需要开发者在 vivo 推送平台中开通回执地址，并在移动推送控制台完成配置后，才能获取到 vivo 通道抵达数据，配置方法如下：

开通回执权益

1. 登录 vivo 的 推送运营平台 。
2. 单击**应用信息**。在 **App 回执能力** > 单击**开通** > 添加回执参数。

回执参数配置

1. 配置消息回执地址。请在 产品管理 页面查看您的应用的服务接入点，并复制该应用的 AccessID，选择对应服务接入点的回执地址进行配置：

服务接入点	回执地址
广州服务接入点	https://stat.tpns.tencent.com/log/statistics/vivo/AccessID
中国香港服务接入点	https://stat.tpns.hk.tencent.com/log/statistics/vivo/AccessID
新加坡服务接入点	https://stat.tpns.sgp.tencent.com/log/statistics/vivo/AccessID
上海服务接入点	https://stat.tpns.sh.tencent.com/log/statistics/vivo/AccessID

说明

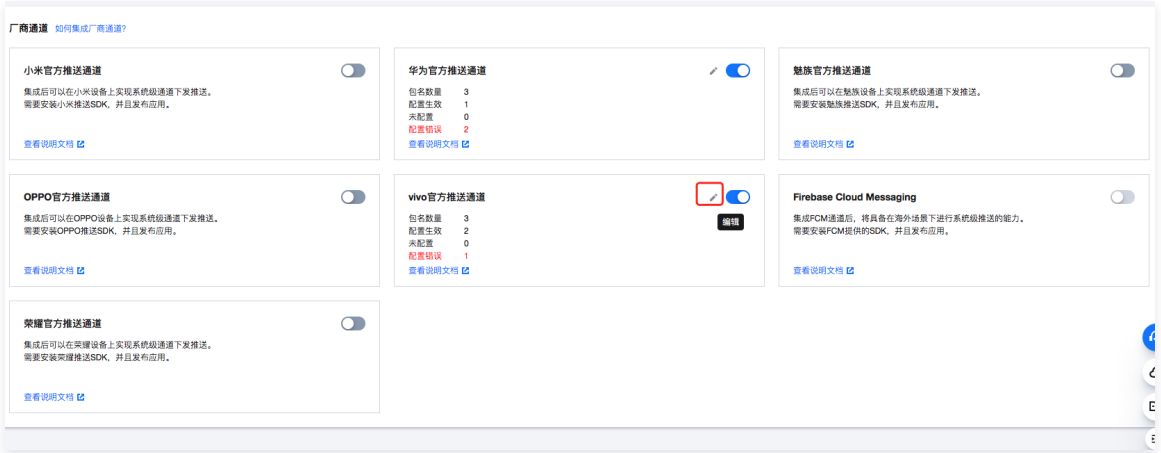
AccessID 替换为您应用的 AccessID。例如应用为广州服务接入点，则回执地址配置为：`https://stat.tpns.tencent.com/log/statistics/vivo/1500016691`。

2. 添加回执地址后，会生成对应的回执ID。

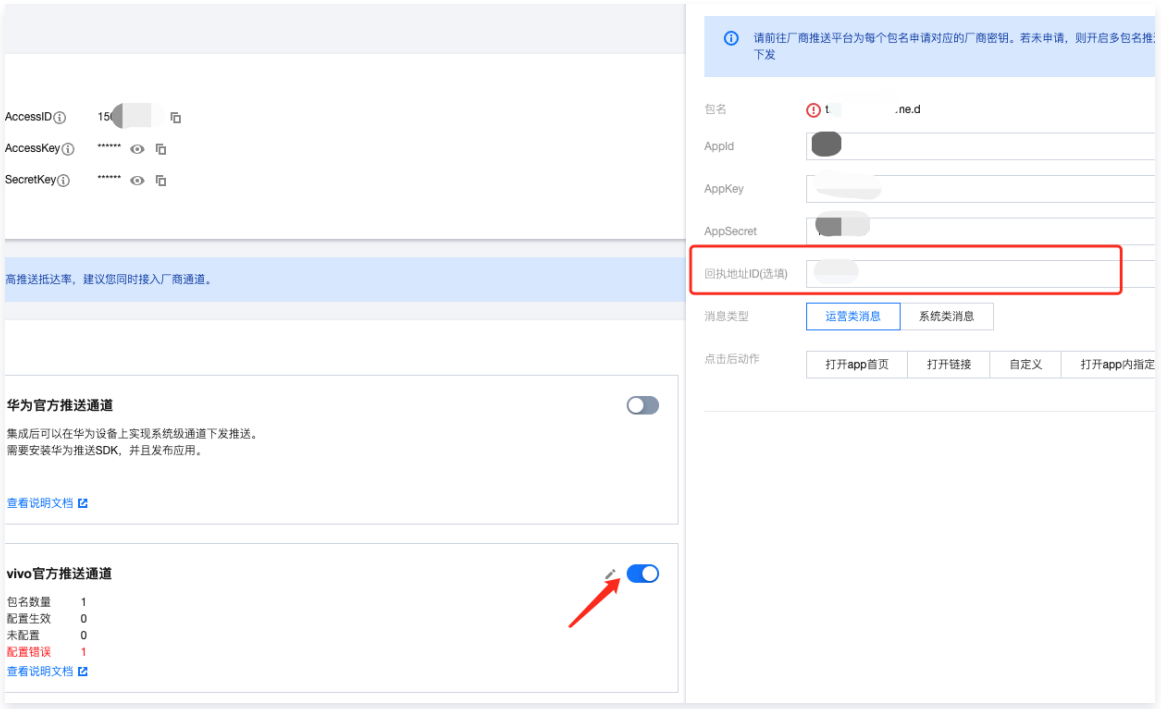
配置回执

1. 进入 产品管理 页面，选择需要配置的应用，单击**配置管理**。

2. 在配置管理页面，**厂商通道 > vivo 官方推送通道 > 单击编辑图标**。



3. 在编辑 vivo 官方推送通道页面，配置 vivo 平台生成的回执 ID，最后点击 保存 即可。



厂商通道限额说明

最近更新时间：2025-01-14 15:26:52

vivo 平台限制

限额说明

推送消息限额说明

- 正式消息分为系统消息和运营消息，两者每日限制发送量均根据 SDK 订阅数推算，SDK 订阅数小于10000，按10000计数；大于10000，则等于 SDK 订阅数。如特殊情况需额外提升系统消息量级，申请方法详见 vivo 系统消息申请 。
  - 系统消息：包括邮件、用户设置的提醒、物流、订单、待办待阅读、财务、功能提醒、即时消息8类消息。
  - 运营消息：包括但不限于广告、推荐、推广、活动等对用户有主动运营作用的推送，或者其他非用户主动触发的信息；未订阅的影音视听内容、商品推广、宣传，或者折扣、红包、领　优惠信息等。
- 通过 API 发送的测试消息每日限制发送量为运营消息100条，系统消息10000条，测试设备设置上限20台。
- 目前不限制单推和群推的比例，可发送的单推和群推消息指定的用户量不得超过每日限制的推送总量。

接收消息限额说明

- 用户单应用每日运营消息接收条数上限2/5条，系统消息无限制。后续会结合平台对用户体验的要求和用户反馈情况对条数进行调整，具体以 vivo 官方公告为准。
- 用户单应用接收条数限制以“到达量”是否超过2/5条为准，在发送时校验单用户是否到达2/5条，超限则计入管控量。

额度查询指引

在 vivo 开放平台 > **推送统计** > **推送数据**中可以查看 SDK 订阅数和可发送的消息总量，详情请参见 vivo 推送平台使用指南 。

OPPO 平台限制

限额说明

- 公信通道（适用于默认的多用户普适性消息推送）中累计用户数 < 50000，可推送量为100000，累计用户数 ⩾ 50000，可推送总数量为累计用户数 × 2。
- 私信通道（适用于单个用户的私人消息推送）的推送数量不受限制，详情请参见 OPPO 通知渠道介绍 。
- 用户接收数量限制：通过 OPPO 推送通道下发的消息（包含公私信），单用户接收上限2000条/日。
  单设备推送条数限制，详情请参见 OPPO 推送服务受限说明 。

类型		公信	私信
单用户推送限制（条/日）	新闻类（三级分类为新闻类）	5条	不限量。
	其他应用类型	2条	不限量。
推送数量限制		所有公信类通道共享推送次数，当日达到次数限制后，所有公信类通道将不能再推送消息，目前单日推送数量为：累计注册用户数 * 2 。	不限量。

额度查询指引

- 管理台查询：累计用户数在 OPPO 推送运营平台 可查询，数据每天刷新。
- API 查询：请参见 OPPO PUSH 服务端 API 。

小米平台限制

限额说明

- 公信消息（默认的多用户普适性消息）的单日可推送数量总量：为应用在MIUI上安装且通知开启数x倍数。默认倍数为2倍，具备《互联网新闻信息服务许可证》的应用为3倍，具体如表1所示。通知开启数小于10000的按10000计数。
- 私信消息（单个用户的私人消息）的推送数量不受限制，详情请参见 小米通知渠道介绍 。

公信消息限制倍数,于2023年2月1日生效，详情请参见 小米推送消息限制说明 。

是否具备《互联网新闻信息服务许可证》	单个应用单日单设备通知推送数量限制倍数（单位：倍）	单个设备单日单应用接收通知数量（单位：条）
有	3	8
无	2	5

说明
若厂商通道推送量超过当日限制，超量后的推送任务将会通过移动推送自建通道补发。

额度查询指引

- 管理台查询：在 小米开放平台 > **推送运营平台** > **推送统计** > **用户数据** > **数据详情**，可查询 MIUI 日联网设备数。
- API 查询：请参见 小米推送消息限制说明 查询当日可下发总量和当日已送达数。

魅族平台限制

限额说明

- 单个业务的推送有速率限制，默认 App 为500条/秒。
- 单个业务每天的推送有次数限制，默认为1000次/天。
- 单个业务订阅标签的个数不超过100个。
- 单个设备单个业务推送消息 ⩾ 4条会被折叠展示，消息多次不点击后有可能会被收纳于右上角消息收纳盒。
- 单个设备1个月内不活跃，将取消订阅。
- 一个 IP 地址每小时请求 API 接口有次数限制，具体次数魅族官方未给出说明。
- 单个业务每天累计请求 API 接口有次数限制，具体次数魅族官方未给出说明。
- 单个业务每天推送的消息总量有限制，具体次数魅族官方未给出说明。

华为平台限制

限额说明

- 发送条数限制：资讯营销类消息的每日推送数量自2023年01月05日起根据应用类型对推送数量进行上限管理，服务与通讯类消息每日推送数量不受限，详情请参见华为的 推送数量管理细则 。
- 发送速率限制：华为推送对推送速度的分配，主要依据 App 在华为渠道的月活、App 在华为应用市场上架时的应用类型这两个要素进行计算分配。

荣耀平台限制

限额说明

发送条数限制：根据消息分类标准，荣耀推送服务将通知消息分为资讯营销、服务与通讯两大类别。资讯营销类消息的每日推送数量根据应用类型对推送数量进行上限管理，服务与通讯类消息每日推送数量不受限。不同应用类别的推送数量上限要求，详情请参见 荣耀的推送数量管理细则 。

厂商通道 QPS 限制说明

最近更新时间：2025-03-25 15:34:43

各个手机厂商的推送通道在发送速率上有一定的限制，如目前速率无法满足您的运营需求，可参考以下说明和申请流程向厂商申请调整 QPS。(仅部分厂商可申请)

华为推送

QPS 限制规则

发送速率（QPS）= App 在华为渠道 MAU x 应用类别权重系数 x 0.00072

名词解释

MAU：在华为渠道推送应用的每月最后一个自然日的值作为当月的 MAU。
分类规则：依据在华为应用市场上架的应用分类。

分组名称	应用分类	权重系数
IM类	社交通讯	5
金融类	金融理财	5
新闻类	新闻阅读；资讯生活	4
内容类	图书阅读；影音娱乐；拍摄美化	3
电商类	购物比价	3
衣食住行类	便捷生活；出行导航；美食；旅游住宿	3
商务类	商务	3
游戏类	网络游戏；休闲益智；经营策略；棋牌	2
工具类	实用工具	1
运动健康类	医疗健康；运动健康	1
其他类	儿童；教育；主题个性；汽车	1
Default	Default	1

说明
如果没在华为应用市场发布，按 Default 分类。
如您的应用通过 QPS 公式计算所得的值不足6000，将执行默认6000的 QPS。
如全网流量较高时，也会出现系统级流控。
另外，无论何种类别的应用，针对单个设备每天不能推送超过10万条/天，否则将进行推送权益限制，需要整改并申报整改方案重新申请 push 权益。

QPS 提升申请

如您对华为产品有任何相关疑问和建议，请反馈至华为邮箱 hwpush@huawei.com，格式如下：
标题：【QPS】咨询及建议+应用名称

华为消息推送QPS咨询及建议表	
应用名称：	
企业名称：	
应用包名：	
在应用市场应用分类：	

问题类型：	申请 QPS 提升、QPS 具体值咨询描述等。
需求背景：	
具体诉求：	
联系方式（邮箱）	

小米推送

QPS 限制规则

小米推送对推送速率（QPS）的分配主要依据 App 的 MIUI 日联网设备数进行分级计算。
QPS：表示1秒可调用的请求数。1个请求里最多可以携带1000个目标设备。例如：3000 QPS时，1秒内最多可推送300万设备。

> **说明**
> MIUI 日联网设备数的查询路径：推送运营平台 > 推送统计 > 用户数据 > 数据详情。

不同量级的 MIUI 日联网设备数分配不同的QPS：

MIUI 日联网设备数	QPS
≥1000万	3000
≥500万且<1000万	2500
≥100万且<500万	2000
≥10万且<100万	1000
<10万	500

QPS 提升申请

暂不开放申请。

OPPO 推送

QPS 限制规则

OPPO 推送对 QPS 的分配主要依据应用的累计用户数，应用类别权重和平台推送系数三个值进行加权计算，其中累计用户数在 OPUSH 平台上的查询路径是 OPPO PUSH 推送运营平台—我要推送消息—应用列表中的累计用户数。

计算公式

应用 QPS = 推送 QPS 参考值 × 应用类别权重 × 平台推送系数

累计用户数	QPS	应用类别权重	平台推送系数(默认 = 1)
≥10000万	30000	1	1
≥5000万且<10000万	20000	1	1
≥1000万且<5000万	10000	1	1
<1000万	5000	1	1

QPS 提升申请

暂不开放申请。

vivo 推送

QPS 限制规则

QPS：推送 QPS 根据 SDK 订阅数自动调整，默认最低值为500条/秒。

QPS 提升申请

暂不开放申请。

魅族推送

QPS 限制规则

QPS：默认 App 是 500 条/秒，如果超过此速率可以联系魅族进行调整。

QPS 提升申请

可以联系魅族进行调整 QPS。

- 推送服务联系人 QQ：2442575011。
- 推送服务咨询邮箱：push_support@meizu.com。

荣耀推送

QPS 限制规则

QPS：单应用 QPS 统一限制为3000，达到 QPS 限制则会返回错误码80200021，本次请求发送失败。

QPS 提升申请

暂不开放申请。
建议应用：

- 降低并发请求量，将每秒请求数控制在 QPS 配额内。
- 优化发送效率，在一次请求中携带更多 token。

> **注意**
> QPS（Quantity Per Second）：每秒推送请求数。每次请求最多可携带1000个 token。

错误码

最近更新时间：2024-10-11 15:40:01

客户端返回码

错误码	原因以及解决办法
0	调用成功
2	参数错误，例如，已绑定单字符的别名，或是 iOS 的 Token 长度不正确，应为64个字符
-1	SDK 内部错误，请保存日志 联系我们
-2	等待服务器回包超时
-3	资源已经被销毁，需要重新购买服务，并且需要升级 App 版本（即：App 版本号需要比之前的版本号大）才可恢复
-4	连接超出套餐限制，连接会随机拒绝，需要升级服务即可恢复
-5	获取 Guid 出错，请检查网络，以及 AccessId 和 AccessKey 是否正确，资源是否已经购买
-7	发送请求包出错，请检查网络，或保存日志 联系我们
-11	连接 MQTT 服务器失败，请检查网络
-101	SDK 出现某些内容 JSON 格式错误，请保存日志 联系我们
-102	获取不到 Token，检查网络和 App 配置
-502	获取 Guid 出错，请检查网络，以及服务接入点域名配置是否正确，详情参见 SDK 集成文档
-701	发送请求包时出现网络异常
-702	发送请求包超时
-1101	连接 MQTT 服务器出现网络异常，请检查网络
-1102	连接 MQTT 服务器出现异常，请检查网络或 联系我们
-1103	连接 MQTT 服务器超时，请检查网络或检查配置是否正确
-10005	AIDL 配置出错
20	鉴权错误，Access ID 或者 Access Key 配置错误
10000	起始错误
10001	操作类型错误码，例如，参数错误时将会发生该错误
10002	正在执行注册操作时，又有一个注册操作，则回调此错误码
10003	权限配错或者缺少所需权限
10004	so 库没有正确导入（Androidstudio 可在 main 文件目录下，添加 jniLibs 命名的文件夹，将 SDK 文档中的 Other-Platform-SO 下的7个 so 库文件夹，添加至该目录）
10005	AndroidManifest 文件的 XGRemoteService 节点没有配置或者的该节点的 action 包名配错
10008	没有配置正确的 ContentProvider，请检查 AndroidManifest 文件
10009	jce JAR 错误或者缺少 jce JAR（检查是否已将 wup 包编译进去了，如果是混淆打包过后出现，请检查混淆代码）
10101	创建链路失败（切换网络重试）
10102	请求处理过程中， 链路被主动关闭（切换网络重试）

10103	请求处理过程中，服务器关闭链接（切换网络重试）
10104	请求处理过程中，客户端产生异常（切换网络重试）
10105	请求处理过程中，发送或接收报文超时（切换网络重试）
10106	请求处理过程中， 等待发送请求超时（切换网络重试）
10107	请求处理过程中， 等待接收请求超时（切换网络重试）
10108	服务器返回异常报文
10109	未知异常，切换网络或者重启设备
10110	创建链路的 handler 为 null
其他	如出现其他未知错误，请记录错误日志并 联系我们
10006	AccessKey 或者 AccessID 错误
10007	初始化移动推送 Service 错误
10008	AccessKey 或者 AccessID 错误
10110	认证过程错误
10115	短时间内重复注册
10300	跑马策略相关返回码
10400	SDK 参数错误
20002	无有效的网络连接，请确认该应用是否付费
10030009	应用不存在，SDK 集成时需要根据接入的服务接入点配置域名，详情参见 SDK 集成文档

服务端返回码

错误码	含义
1010001	资源未部署，请确认应用是否已购买推送资源
1008001	参数解析错误
1008002	必填参数缺失
1008003	认证失败
1008004	调用服务失败
1008006	Token 无效，请检查设备 Token 是否注册成功
1008007	参数校验失败
1008011	文件上传失败
1008012	上传文件为空
1008013	证书解析失败
1008015	推送任务 ID 不存在
1008016	日期时间参数格式不对
1008019	被内容安全服务判定不和谐

1008020	证书包名校验失败
1008021	p12 证书格式内容校验失败
1008022	p12 证书密码不对
1008025	创建应用失败，产品下已存在该平台的应用
1008026	批量操作，部分失败
1008027	批量操作，全部失败
1008028	超出限频
1008029	Token 校验非法
1008030	App 未付费
1008031	App 资源已销毁
10110008	查询的 Token ， 账号不存在
10010005	推送目标不存在
10010012	非法的推送时间，请更改推送时间。 定时推送时 send_time 传入的值如果是过去的时间，具体规则如下： • 如果 | send_time − 当前时间 | <= 10min， 推送任务会创建，接收任务后会立即调度 • 如果 | send_time − 当前时间 | > 10min， 推送任务会被拒绝， 接口返回失败
10010018	重复推送
10030002	AccessID 和 AccessKey 不匹配

iOS 接入指南
简介

最近更新时间：2025-03-25 15:34:43

iOS 端实现推送消息的服务涉及到三个角色：终端应用（Client App），APNs（Apple Push Notification service），移动推送服务器（移动推送 Provider）。在使用移动推送服务实现给客户端推送消息，需要这三个角色在整个流程中相互配合，任何一个角色出现异常，都可能会导致消息推送无法收到。

SDK 说明

文件组成

- `XGPush.h`，`XGPushPrivate.h`（SDK 提供接口的头文件）
- `libXG-SDK-Cloud.a`（主 SDK 文件）
- `libXGExtension.a`、`XGExtension.h`（“抵达和富媒体”扩展插件库及接口头文件）
- `XGMTACloud.framework`（“点击上报”组件）
- `XGInAppMessage.framework`（应用内消息）

版本说明

- 支持 iOS 8.0+。
- 针对 iOS 10.0+ 以上版本。
- 需要额外引入 UserNotification.framework。
- 建议使用 Xcode 8.0 +。
- 如果使用 Xcode7 及其以下的版本，需要自行配置 iOS SDK 来支持 UserNotification 框架的编译。

功能说明

iOS SDK 是移动推送服务为客户端实现消息推送而提供给开发者的接口，主要负责完成：

- 设备 Token 的自动化获取和注册，降低接入门槛。
- 账号、标签与设备的绑定接口，以便开发者实现特定群组的消息推送，丰富推送方式。
- 点击量上报，统计消息被用户点击的次数。

推送通道

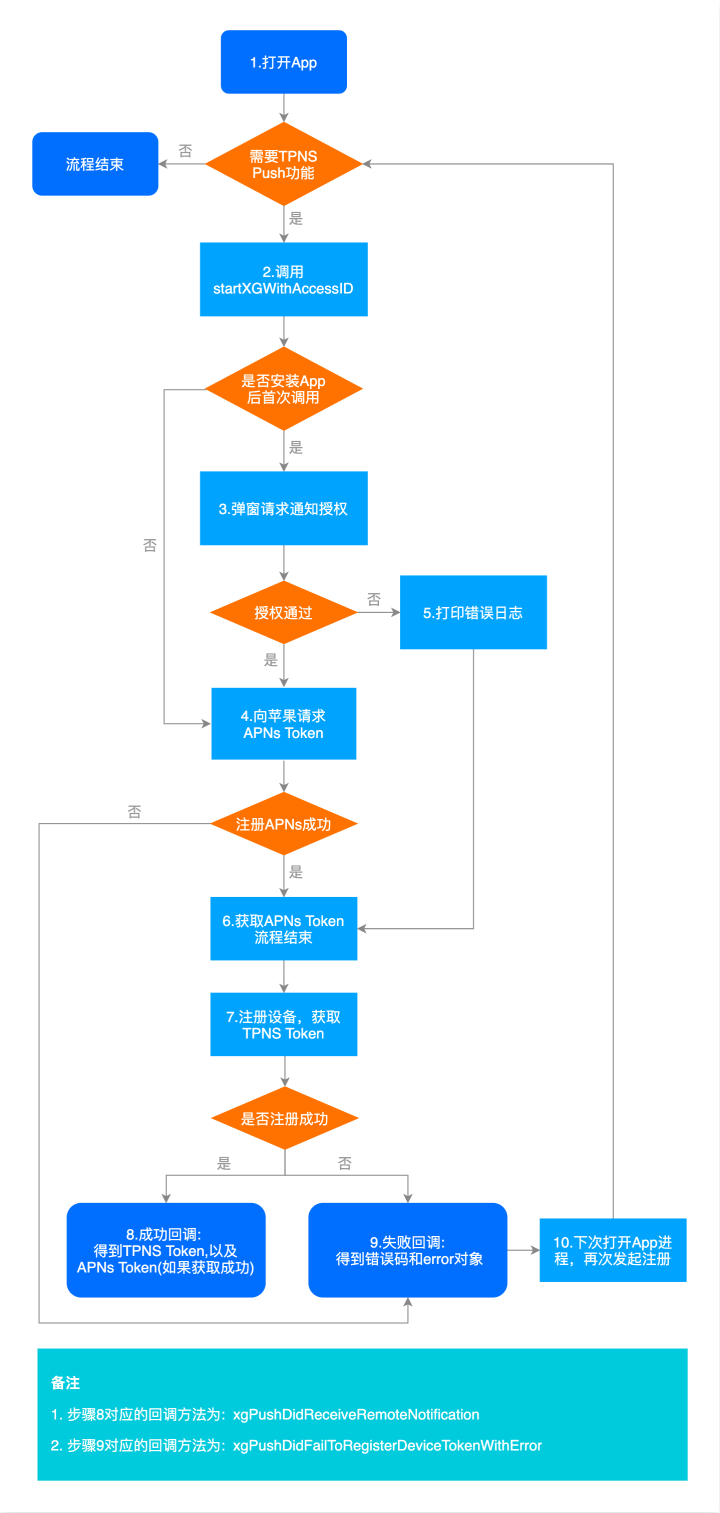
移动推送使用的消息下发通道：

- 移动推送在线通道：移动推送在线通道是移动推送的自建通道，依赖移动推送 Service 在线（与移动推送后台服务器保持长连接）才能下发消息；需要 SDK 1.2.8.0 及以上版本支持。
- APNs 通道：苹果官方提供的消息推送服务，详情请参见 APNs。

常用场景流程说明

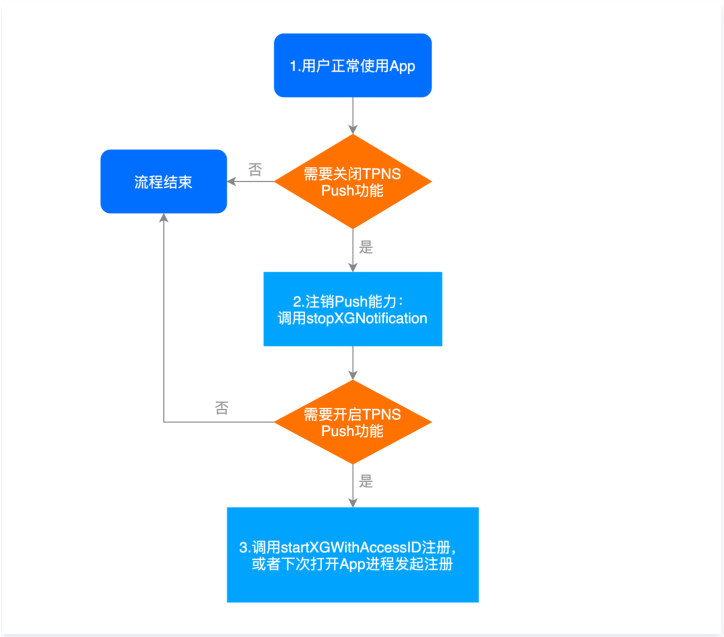
设备注册流程

下图为设备注册相关流程，具体接口方法请查看 启动腾讯移动推送服务 。



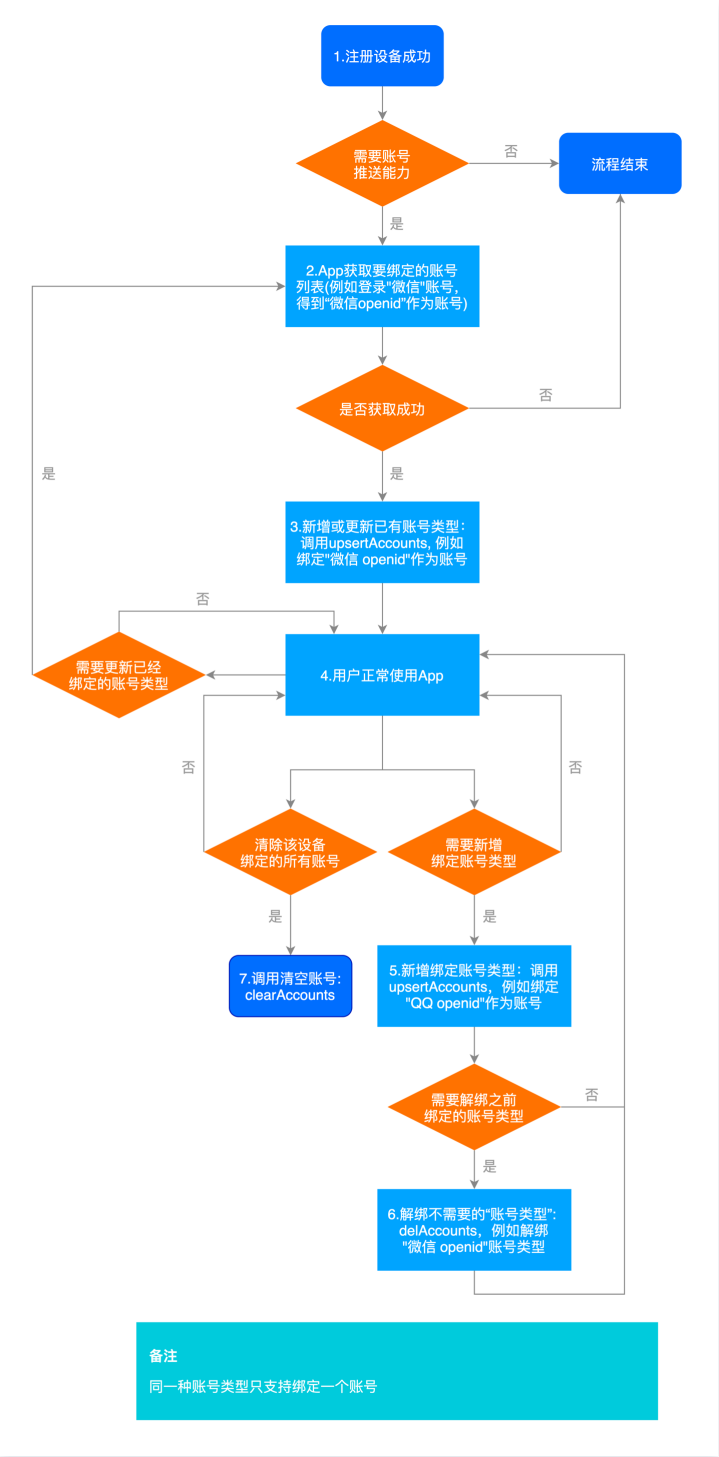
设备反注册流程

下图为设备反注册相关流程，具体接口方法请查看 终止腾讯移动推送服务 。



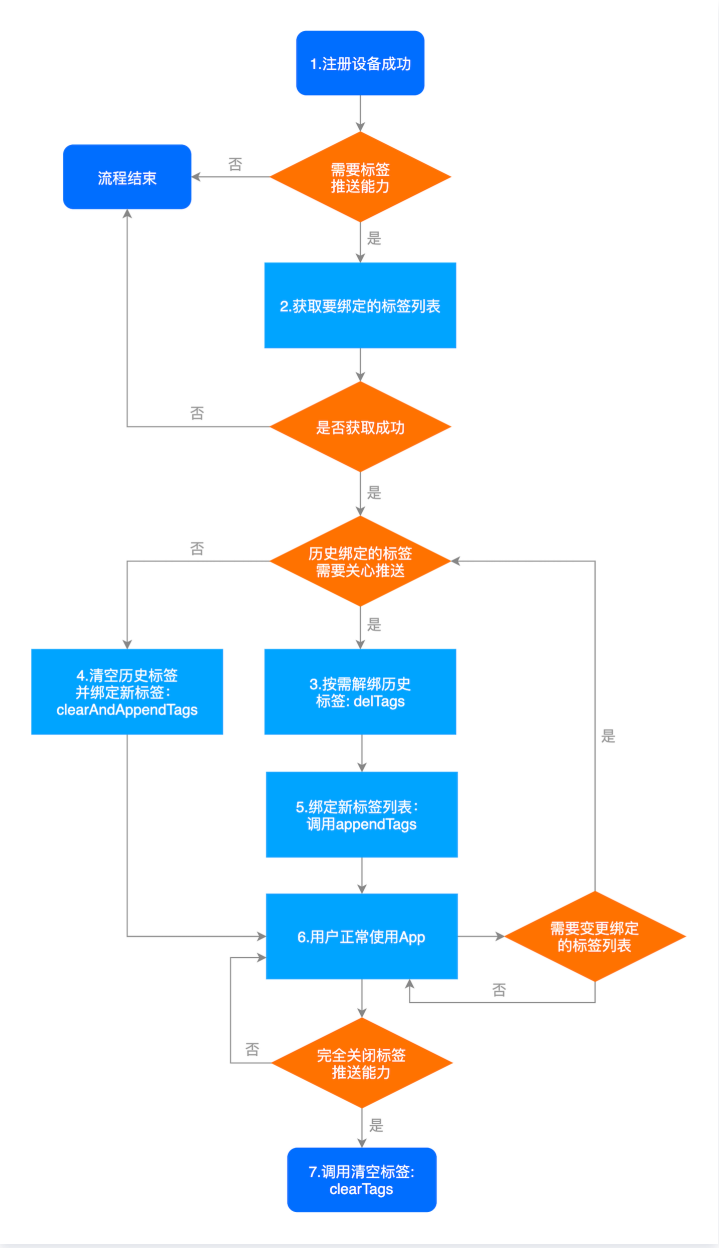
账号相关流程

下图为账号相关流程，具体接口方法请查看 账号管理 。



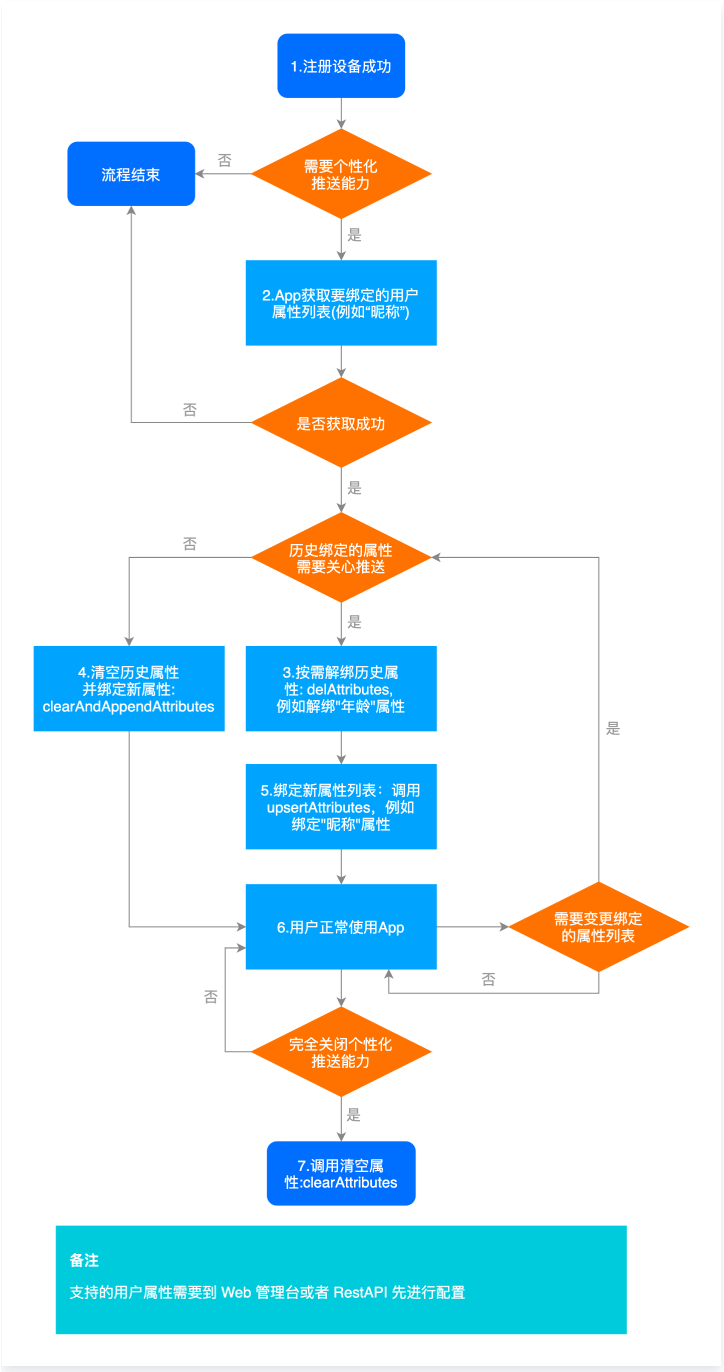
标签相关流程

下图为标签相关流程，具体接口方法请查看 标签管理 。



用户属性相关流程

下图为用户属性相关流程，具体接口方法请查看 用户属性管理 。



SDK 合规使用指南

最近更新时间：2024-10-11 15:31:21

> **注意**
> 根据《个人信息保护法》、《数据安全法》、《网络安全法》等法律法规和监管部门规章要求，App 开发运营者（以下简称为“开发者”）在提供网络产品服务时应尊重和保护最终用户的个人信息，不得违法违规收集使用个人信息，保证和承诺就个人信息处理行为获得最终用户的授权同意，遵循最小必要原则，且应当采取有效的技术措施和组织措施确保个人信息安全。为帮助开发者在使用移动推送 SDK 的过程中更好地落实用户个人信息保护相关要求，避免出现侵害最终用户个人信息权益的情形，特制定本合规使用说明，开发者使用 移动推送 SDK 时必须在《隐私政策》中告知终端用户 SDK 使用用途，并且在终端用户未同意《隐私政策》前不得初始化任何 SDK。请您务必按照以下步骤做好合规自查，避免被监管部门通报或下架您的应用。

1 请务必确保您已经将移动推送 SDK 升级至满足监管新规的最新版本

请务必确保您已经将移动推送 SDK 升级至满足监管新规的最新版本。请前往 iOS SDK 发布动态 查看和下载最新版本 SDK。

2 隐私政策中添加移动推送相关说明

请您确保您开发或运营的应用有符合监管要求的《隐私政策》文本。同时请您务必明确告知终端用户您的应用使用了移动推送服务。建议您在《隐私政策》对应的章节、列表中添加关于移动推送的说明，推荐条款如下：

```
SDK名称：移动推送 SDK
第三方名称：深圳市腾讯计算机系统有限公司
SDK 用途：在移动终端设备进行消息推送
收集个人信息类型：
(1)、 设备信息(手机型号，系统类型、系统版本等)用于标签化推送以及识别是否是真机、网络信息(网络类型)支持根据不同网络类型进行不同类型推送、应用数据（推送流程中产生的送达、点击、曝光等数据）用于推送业务数据统计。
(2)、（可选信息，依赖开发者设置）账号绑定信息(根据您所选用的不同推送渠道，QQ号、微信Union ID、 手机号、邮箱等)用于根据账号信息进行推送
数据处理方式：通过去标识化、加密传输及其他安全方式
官网链接：https://cloud.tencent.com/product/tpns
隐私政策链接：https://privacy.qq.com/document/preview/8565a4a2d26e480187ed86b0cc81d727
```

3《隐私政策》弹出条件

您需要确保 App 有《隐私政策》，且在用户首次启动 App 时弹出《隐私政策》取得用户同意。

4 隐私接口确认与移动推送 SDK 初始化、自启动配置及业务功能调用时机

要求内容：《SDK 合规使用说明》应详细说明SDK初始化及各项业务功能接口合规调用时机。App 应当在 App 登录注册页面及 App 首次运行时，通过弹窗、文本链接及附件等简洁明显且易于访问的方式，向最终用户告知涵盖个人信息处理主体、处理目的、处理方式、处理类型、保存期限等内容的个人信息处理规则，并且获得最终用户授权同意后才能处置最终用户数据。

接入说明：请务必确保终端用户有效同意您 App 中的隐私政策后，再进行移动推送 SDK 初始化。用户同意隐私政策之前，避免动态申请涉及用户个人信息的敏感设备权限；用户同意隐私政策前，您应避免私自采集和上报个人信息。当您的 App 未向用户提供服务时，例如 App 在后台运行时，请勿请求移动推送 SDK 的相关服务接口。

4.1 初始化相关说明

移动推送 SDK 无初始化接口，开发者您无需调用 SDK 初始化接口。

4.1 业务接口调用说明

- 请务必确保终端用户有效同意您 App 中的隐私政策后再调用 `startXGWithAccessID` 来使用移动推送 SDK 推送业务。
- 如果用户已经同意《隐私政策》，建议在 App 启动后，尽早调用移动推送SDK注册接口 `startXGWithAccessID` ，得到完整的推送服务。
- 如果用户未同意《隐私政策》，请勿调用 `startXGWithAccessID` 。

更多 SDK 功能业务接口连接：https://cloud.tencent.com/document/product/548/48835

相关代码示例，可参见 移动推送-Demo 。该 Demo 可通过移动推送官网 手动下载（下载并解压后，即可查看 Demo）。

5. SDK可选信息配置开关

要求内容：《SDK 合规使用说明》应详细说明SDK各项可选个人信息使用目的、场景及对应关闭的配置方式、示例。

接入说明：移动推送 SDK 向您提供了可选个人信息及权限的控制开关，您可以根据 App 所需的 SDK 功能服务自行配置打开或关闭隐私信息请求开关，对于移动推送 SDK 可选收集的个人信息的控制，开发者可以参考如下配置指引进行配置操作。因相关信息的不收集将会对其对应的功能造成影响，请开发者结合业务实际需要进行合理配置。

5.1、配置可选个人信息

基于账号推送功能，第三方开发者需要通过预埋账号类型绑定共享给移动推送 SDK，**根据第三方开发者选择的推送通道的不同所共享给移动推送 SDK的信息也会有所不同。如果第三方开发者未使用账号推送功能则不需要共享此类信息**。
如需清除已收集账号数据，可以参考SDK账号接口进行清除，SDK 业务功能接口链接：https://cloud.tencent.com/document/product/548/48835 。

可选个人信息类型及字段		使用目的	使用场景
设备信息（可选）	*OAID（仅Android）*	用 OAID 作为移动推送的账号，账号推送时可以使用 OAID 进行账号推送。	根据 OAID 进行账号推送时使用
	微信 UnionID	使用微信 UnionID 作为移动推送的账号，账号推送时可以使用微信 UnionID 进行账号推送。	根据 UnionID 进行账号推送时使用
	QQ openID	使用 QQ openID 作为移动推送的账号，账号推送时可以使用 QQ openID 进行账号推送。	根据 QQ openID 进行账号推送时使用
账号信息（可选）	邮箱账号、新浪微博账号、支付宝账号、淘宝账号、豆瓣账号、Facebook 账号、Twitter 账号、Google 账号、百度账号、京东账号、linkedin 账号	使用设置的账号类型和账号信息作为移动推送的账号，账号推送时可以使用该账号类型和账号信息进行账号推送。	根据设置账号类型进行账号推送
用户基本信息（可选）	*手机号*	广告定向投放及反作弊	在进行广告投放和广告投放效果分析时使用

5.2、SDK申请系统权限说明

要求内容：《SDK 合规使用说明》应详细说明SDK所需的系统权限与各业务功能间的关系，并说明权限申请时机。
接入说明：对于移动推送 SDK 申请的系统权限，您可以参考相关如下表格的内容，详细了解相关权限与各业务功能的关系及其申请时机，因相关权限的不申请将会对其对应的功能造成影响，您可以结合业务实际需要进行合理配置。配置文档链接：https://cloud.tencent.com/document/product/548/48835 。

操作系统	权限名称	使用目的	功能场景（申请时机）	是否可选
iOS/Mac OS	APNS	用于给应用展示通知	调用推送 SDK 业务功能之前	必选
iOS/Mac OS	网络权限	用户网络访问	用推送服务 SDK 时	必选

6. SDK可按照不同频次、精度收集个人信息的配置说明

要求内容：如果SDK可按照不同频次、精度收集个人信息的，《SDK 合规使用说明》应说明不同频次、精度的使用目的、场景及对应选择的配置方式、示例。
接入说明：收集频次方面，移动推送 SDK 的数据采集仅在 App 调用/最终用户触发相关功能时触发，不涉及定时逻辑等频次控制选项。收集精度方面不涉及精度相关控制，相关业务接口务必确保终端用户有效同意《隐私政策》后调用，可参考 SDK 接口文档链接：
https://cloud.tencent.com/document/product/548/48835 。

7、最终用户同意方式的示例

要求内容：《SDK 合规使用说明》应详细说明 App 获取最终用户授权同意的建议方式，其中需要取得最终用户单独同意的，应显著提示并给出示例。
接入说明：App 首次运行时应当有隐私弹窗，隐私弹窗中应公示简版隐私政策内容并附完整版隐私政策链接，并明确提示最终用户阅读并选择是否同意隐私政策；隐私弹窗应提供同意按钮和拒绝同意的按钮，并由最终用户主动选择。
披露示例：

8. 撤回同意机制说明

请您告知终端用户其享有选择关闭相应权限或行使退出（opt-out）权利，一旦终端用户行使前述权利，其个人信息将不再会被处理。我们建议您在终端用户撤销同意处理其个人信息的授权时，参见腾讯推送服务 移动推送 iOS 接入指南 - 接口文档 “反注册接口说明”部分进行处理，以便用户更便捷行使退出的选择权。

9. 隐私保护机制

如果您对 SDK 权限有任何疑问、意见和建议，或者需要腾讯协助关闭某项权限采集能力，可通过以下联系方式与我们联系：

- 电子邮件：tpns_team@tencent.com
- 联系地址：深圳市南山区粤海街道麻岭社区科技中一路腾讯大厦

您还可以随时通过访问移动推送官网在线客服系统与我们联系，我们将及时为您提供咨询和服务，确保各项隐私机制的落实和执行。

SDK 集成

最近更新时间：2025-03-25 15:34:43

简介

本文档提供关于 SDK 接入以及开启推送服务的示例代码（SDK 版本：V1.0+ 版本）。

> **注意**
> 为了避免您的 App 被监管部门通报或下架，请您在接入 SDK 之前务必按照iOS的 SDK 合规使用指南 在《隐私政策》中增加移动推送相关说明，并且在用户同意《隐私政策》后再初始化移动推送SDK。

SDK 组成

- doc 文件夹：腾讯移动推送 iOS SDK 开发指南。
- demo 文件夹：包含样例工程，腾讯移动推送SDK（仅包含 OC demo，Swift Demo 请前往 腾讯工蜂 进行下载）。

SDK 集成

接入前准备

1. 接入 SDK 之前，请前往 移动推送控制台 创建产品和 iOS 应用，详细操作可参见 创建产品和应用 。

2. 单击**配置管理**，进入管理页面。

3. 单击**上传证书**，完成上传操作。推送证书获取详情请参见 证书获取指引 。

4. 证书上传成功后，在应用信息栏中，获取应用 Access ID 和 Access KEY。

导入 SDK（二选一）

方式一：Cocoapods 导入

通过 Cocoapods 下载地址：

```
pod 'TPNS-iOS', '~> 版本'  // 如果不指定版本则默认为本地 pod TPNS-iOS 最新版本
```

说明

- 首次下载需要登录 仓库地址，并在**账户**菜单栏中 设置用户名和密码。设置成功后，在 Terminal 输入对应的用户名和密码，后续即可正常使用，当前 PC 不需要再次登录。
- 由于仓库地址变更，如果 pod 提示 `Unable to find a specification for 'TPNS-iOS'`，那么需要执行以下命令，并更新仓库确认版本：

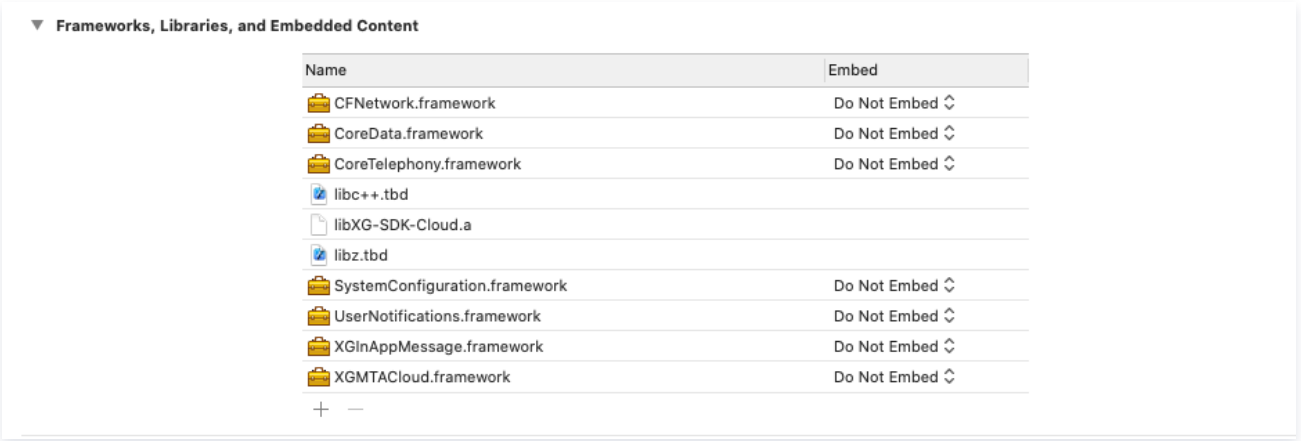
```
pod repo update
pod search TPNS-iOS
pod install //安装 SDK
```

方式二：手动导入

1. 进入腾讯移动推送 控制台，单击左侧菜单栏 SDK 下载，进入下载页面，选择需要下载的 SDK 版本，单击操作栏中**下载**即可。
2. 打开 demo 目录下的 SDK 文件夹，将 XGPush.h 及 libXG-SDK-Cloud.a 添加到工程，打开 XGPushStatistics 文件夹，获取 XGMTACloud.framework。
3. 将 InAppMessage 文件夹导入到工程并在 **Build Setting** > **Framework Search Paths** 添加查找路径（若您 SDK 版本低于1.2.8.0，则可以忽略此步骤）。
4. 在 Build Phases 下，添加以下 Framework：

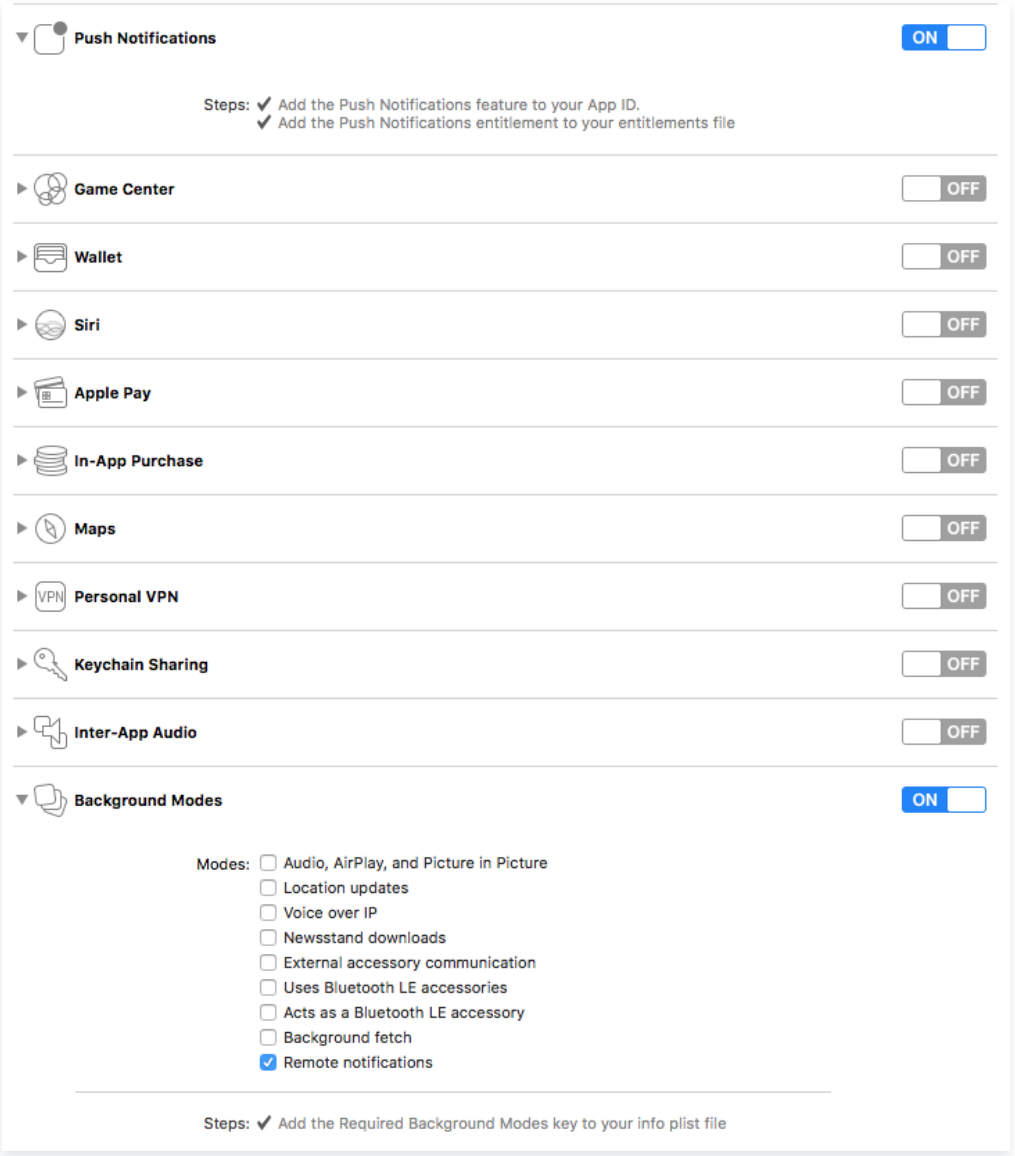
```
* XGInAppMessage.framework
* XGMTACloud.framework
* CoreTelephony.framework
* SystemConfiguration.framework
* UserNotifications.framework
* libXG-SDK-Cloud.a
* libz.tbd
* CoreData.framework
* CFNetwork.framework
* libc++.tbd
```

5. 添加完成后，库的引用如下：

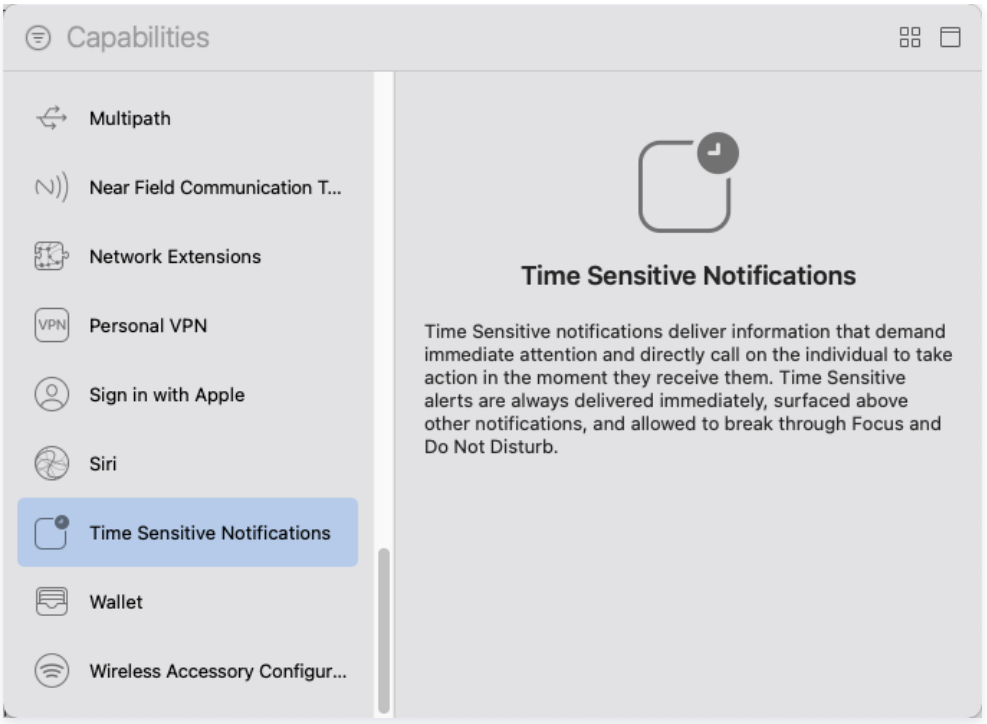


工程配置

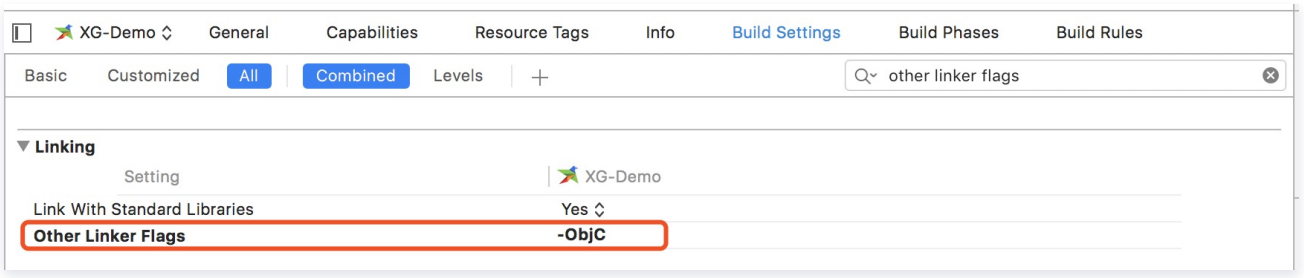
1. 在工程配置和后台模式中打开推送，如下图所示：



1.1 如需使用 iOS15 新增的"时效性通知功能"，请在 `Capabilities` 中开启 `Time Sensitive Notifications` 。



2. 添加编译参数 `-ObjC` 。



如果 checkTargetOtherLinkFlagForObjc 报错，是因为 build setting 中，Other link flags 未添加 -ObjC。

> **注意**
> 如果您的应用服务接入点为广州，SDK 默认实现该配置，广州域名为 `tpns.tencent.com` 。

如果您的应用服务接入点为上海、新加坡或者中国香港，请按照下文步骤完成其他服务接入点域名配置。

2.1 解压 SDK 文件包，将 SDK 目录下的 XGPushPrivate.h 文件添加到工程中并在需要配置域名的类中引用(#import "XGPushPrivate.h")。

2.2 在 `startXGWithAccessID:accessKey:delegate:` 方法之前调用头文件中的配置 域名 接口：

- 如需接入上海服务接入点，则将域名设置为 `tpns.sh.tencent.com` 。
  示例

```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.sh.tencent.com"];
```

- 如需接入新加坡服务接入点，则将域名设置为 `tpns.sgp.tencent.com` 。
  示例

```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.sgp.tencent.com"];
```

- 如需接入中国香港服务接入点，则将域名设置为 `tpns.hk.tencent.com` 。
  示例

```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.hk.tencent.com"];
```

如需接入中国广州服务接入点，则将域名设置为 `tpns.tencent.com` 。
示例

```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.tencent.com"];
```

接入样例

调用启动腾讯移动推送的 API，并根据需要实现 `XGPushDelegate` 协议中的方法，开启推送服务。
1. 启动腾讯移动推送服务， `AppDelegate` 示例如下：

```
@interface AppDelegate () <XGPushDelegate>
@end
/**
@param AccessID 通过 TPNS 管理台申请的 AccessID
@param AccessKey 通过 TPNS 管理台申请的 AccessKey
@param delegate 回调对象
**/
-(BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
{
  [[XGPush defaultManager] startXGWithAccessID:<your AccessID> accessKey:<your AccessKey>
delegate:self];
return YES;
}
```

2. 在 `AppDelegate` 中，选择实现 `XGPushDelegate` 协议中的方法：

```
/// 统一接收消息的回调
/// @param notification 消息对象(有2种类型NSDictionary和UNNotification具体解析参考示例代码)
/// @note 此回调为前台收到通知消息及所有状态下收到静默消息的回调（消息点击需使用统一点击回调）
/// 区分消息类型说明：xg字段里的msgtype为1则代表通知消息msgtype为2则代表静默消息
- (void)xgPushDidReceiveRemoteNotification:(nonnull id)notification withCompletionHandler:(nullable
void (^)(NSUInteger))completionHandler{
 /// code
}
 /// 统一点击回调
/// @param response 如果iOS 10+/macOS 10.14+则为UNNotificationResponse，低于目标版本则为NSDictionary
- (void)xgPushDidReceiveNotificationResponse:(nonnull id)response withCompletionHandler:(nonnull void
(^)(void))completionHandler {
  /// code
}
```

通知服务扩展插件集成

SDK 提供了 Service Extension 接口，可供客户端调用，从而可以使用以下扩展功能：

- 精准统计 APNs 通道消息抵达。
- 接收 APNs 通道图片、音视频富媒体消息。

接入步骤请参见文档 通知服务扩展的使用说明 。

> **注意**
> 如果未集成此接口，则无法统计 APNs 通道“抵达数”。

未集成通知服务扩展插件：

数据明细 收起 ▲ 数据下载

通道类型	计划发送	实际发送数	抵达数	点击	点击率
总计（去重）	101,065	99,753	0	1,495	--
TPNS通道	1,007	0	0	0	--
APNs通道	100,058	99,753	0	1,495	--

集成通知服务扩展插件后：

数据明细 收起 ▲ 数据下载

通道类型	计划发送	实际发送数	抵达数	点击	点击率
总计（去重）	1,174,168	1,154,036	404,834	5,626	1.39%
TPNS通道	39	0	0	0	--
APNs通道	1,174,129	1,154,036	404,834	5,626	1.39%

调试方法

开启 Debug 模式

打开 Debug 模式，即可在终端查看详细的腾讯移动推送 Debug 信息，方便定位问题。

示例代码

```
//打开 debug 开关
[[XGPush defaultManager] setEnableDebug:YES];
```

实现 `XGPushDelegate` 协议

在调试阶段建议实现协议中的此方法，即可获取更详细的调试信息：

```
/**
@brief 注册推送服务回调
@param deviceToken APNs 生成的 Device Token
@param xgToken TPNS 生成的 Token，推送消息时需要使用此值。TPNS 维护此值与 APNs 的 Device Token 的映射关系
@param error 错误信息，若 error 为 nil 则注册推送服务成功
@note TPNS SDK1.2.6.0+
*/
- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nullable NSString
*)xgToken error:(nullable NSError *)error;

/// 注册推送服务失败回调
/// @param error 注册失败错误信息
/// @note TPNS SDK1.2.7.1+
- (void)xgPushDidFailToRegisterDeviceTokenWithError:(nullable NSError *)error {
}
```

观察日志

如果 Xcode 控制台，显示如下相似日志，表明客户端已经正确集成 SDK。

```
[TPNS] Current device token is 9298da5605c3b242261b57****376e409f826c2caf87aa0e6112f944
```

```
[TPNS] Current TPNS token is 00c30e0aeddff1270d8****dc594606dc184
```

> **注意**
> 在推送单个目标设备时请使用 TPNS 36位的 Token。

统一接收消息及点击消息回调说明

统一接收消息回调说明：

- 通知消息：应用处于前台会走此回调，后台/关闭状态下无消息回调，需要点击通知走点击回调。
- 静默消息：应用处于前台/后台会走此回调，关闭状态下需要打开应用后才走此回调。

```
- (void)xgPushDidReceiveRemoteNotification:(nonnull id)notification withCompletionHandler:(nullable void
(^)(NSUInteger))completionHandler;
```

> **说明**
> - 如何让通知消息在前台展示
>
> 当您前台收到通知时默认不弹横幅，通知中心不会展示的，如需展示请参考如下代码：
>
> ```
> if ([notification isKindOfClass:[UNNotification class]]) {
>  completionHandler(UNNotificationPresentationOptionBadge | UNNotificationPresentationOptionSound
> | UNNotificationPresentationOptionAlert);
> }
> ```
>
> - 如何区分通知消息和静默消息
>
> 当应用在前台收到通知消息以及所有状态下收到静默消息时，会触发统一接收消息回调 xgPushDidReceiveRemoteNotification。
> 区分前台收到通知消息和静默消息示例代码如下：
>
> ```
> NSDictionary *tpnsInfo = notificationDic[@"xg"];
> NSNumber *msgType = tpnsInfo[@"msgtype"];
> if (msgType.integerValue == 1) {
>       /// 前台收到通知消息
>   } else if (msgType.integerValue == 2) {
>       /// 收到静默消息
>   } else if (msgType.integerValue == 9) {
>       /// 收到本地通知（TPNS本地通知）
>   }
> ```

统一点击消息回调说明：
应用所有状态（前台、后台、关闭）下点击通知消息时会走此回调。

```
/// 统一点击回调
/// @param response 如果 iOS 10+/macOS 10.14+ 则为 UNNotificationResponse，低于目标版本则为 NSDictionary
/// @note TPNS SDK1.2.7.1+
- (void)xgPushDidReceiveNotificationResponse:(nonnull id)response withCompletionHandler:(nonnull void (^)
(void))completionHandler;
```

> **注意**
> - 移动推送统一消息回调 `xgPushDidReceiveRemoteNotification` 会处理消息接收，并自动后续调用 `application:didReceiveRemoteNotification:fetchCompletionHandler` 方法。然而，该方法也可能被其他 SDK 也进行 hook 调用。
> - 如果您只集成了移动推送平台，我们不推荐再去实现系统通知回调方法，请统一在移动推送通知回调中进行处理。

- 如果您集成了多推送平台，并且需要在 `application:didReceiveRemoteNotification:fetchCompletionHandler` 方法处理其他推送平台的业务，请参照如下指引，避免业务重复：
  - 您需要区分平台消息，在两个消息回调方法中分别拿到消息字典后通过“xg”字段来区分是否是移动推送平台的消息，如果是移动推送的消息则在 `xgPushDidReceiveRemoteNotification` 方法进行处理，非移动推送消息请统一在 `application:didReceiveRemoteNotification:fetchCompletionHandler` 方法处理。
  - `xgPushDidReceiveRemoteNotification` 和 `application:didReceiveRemoteNotification:fetchCompletionHandler` 如果都执行，总共只需要调用一次 `completionHandler`。如果其他 SDK 也调用 `completionHandler`，确保整体的 `completionHandler` 只调用一次。这样可以防止由于多次 `completionHandler` 而引起的 crash。

高级配置（可选）

获取移动推送 Token 交互建议

建议您完成 SDK 集成后，在 App 的**关于、意见反馈**等比较不常用的 UI 中，通过手势或者其他方式显示移动推送Token，控制台和 Restful API 推送需要根据移动推送Token 进行 Token 推送，后续问题排查也需要根据移动推送Token 进行定位。

示例代码

```
//获取 TPNS 生成的 Token
[[XGPushTokenManager defaultTokenManager] xgTokenString];
```

获取 "移动推送运行日志" 交互建议

建议您完成 SDK 集成后，在 App 的**关于、意见反馈**等比较不常用的 UI 中，通过手势或者其他方式显示"移动推送运行日志"，方便后续问题排查。
效果如下图所示：

示例代码

```
[[XGPush defaultManager] uploadLogCompletionHandler:^(BOOL result, NSString * _Nullable errorMessage) {
NSString *title = result ? NSLocalizedString(@"report_log_info", nil) : NSLocalizedString(@"failed", nil);
if (result && errorMessage.length>0) {
UIPasteboard *pasteboard = [UIPasteboardgeneralPasteboard];
pasteboard.string = errorMessage;
}
[TPNSCommonMethodshowAlert:title message:errorMessage viewController:selfcompletion:nil];
}];
```

隐私协议声明建议

您可在申请 App 权限使用时，使用以下内容声明授权的用途：

> **我们使用 腾讯云移动推送 TPNS 用于实现产品信息的推送，在您授权我们“访问网络连接”和“访问网络状态”权限后，表示您同意 腾讯 SDK 隐私协议 。您可以通过关闭终端设备中的通知选项来拒绝接受此 SDK 推送服务。**

其中上述声明授权的两个链接如下：

- **腾讯云移动推送：** `https://cloud.tencent.com/product/tpns`
- **腾讯 SDK 隐私协议：** `https://cloud.tencent.com/document/product/548/50955`

接口文档

最近更新时间：2025-03-19 10:14:52

说明

本文档中账号功能、标签功能及用户属性功能适用于 SDK 1.2.9.0或更高版本，1.2.7.2及之前版本请参见 接口文档 。

启动腾讯移动推送服务

以下为设备注册相关接口方法，若需了解调用时机及调用原理，可查看 设备注册流程 。

接口说明

通过使用在腾讯移动推送官网注册的应用信息，启动腾讯移动推送服务。
（此接口为 SDK 1.2.7.2版本新增，1.2.7.1及之前版本请参考 SDK 包内 XGPush.h 文件 `startXGWithAppID` 接口）。

```
/// @note TPNS SDK1.2.7.2+
- (void)startXGWithAccessID:(uint32_t)accessID accessKey:(nonnull NSString *)accessKey delegate:(nullable id<XGPushDelegate>)delegate;
```

参数说明

- accessID：通过前台申请的 AccessID。
- accessKey：通过前台申请的 AccessKey。
- Delegate：回调对象。

> **注意**
> 接口所需参数必须要正确填写，否则腾讯移动推送服务将不能正确为应用推送消息。

示例代码

```
[[XGPush defaultManager] startXGWithAccessID:<your AccessID> accessKey:<your AccessKey> delegate:self];
```

终止腾讯移动推送服务

以下为设备注册相关接口方法，若需了解调用时机及调用原理，可查看 设备反注册流程 。

接口说明

终止腾讯移动推送服务后，将无法通过腾讯移动推送服务向设备推送消息，如再次需要接收腾讯移动推送服务的消息推送，则必须再次调用 `startXGWithAccessID:accessKey:delegate:` 方法重启腾讯移动推送服务。

```
- (void)stopXGNotification;
```

示例代码

```
[[XGPush defaultManager] stopXGNotification];
```

移动推送 Token 及注册结果

查询移动推送 Token

接口说明

查询当前应用从腾讯移动推送服务器生成的 Token 字符串。

```
@property (copy, nonatomic, nullable, readonly) NSString *xgTokenString;
```

示例代码

```
NSString *token = [[XGPushTokenManager defaultTokenManager] xgTokenString];
```

> ⚠ 注意
> token 的获取应该在 xgPushDidRegisteredDeviceToken:error: 返回正确之后被调用。

注册结果回调

接口说明

SDK 启动之后，通过此方法回调来返回注册结果及 Token。

```
- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nullable NSString
*)xgToken error:(nullable NSError *)error
```

返回参数说明

- deviceToken：APNs 生成的 Device Token。
- xgToken：移动推送生成的 Token，推送消息时需要使用此值。移动推送维护此值与 APNs 的 Device Token 的映射关系。
- error：错误信息，若 error 为 nil，则注册推送服务成功。

注册失败回调

接口说明

SDK 1.2.7.2 新增，当注册推送服务失败会走此回调。

```
/// @note TPNS SDK1.2.7.2+
- (void)xgPushDidFailToRegisterDeviceTokenWithError:(nullable NSError *)error
```

通知授权弹窗的回调

接口说明

SDK 1.3.1.0 新增，通知弹窗授权的结果会走此回调。

```
- (void)xgPushDidRequestNotificationPermission:(bool)isEnable error:(nullable NSError *)error;
```

返回参数说明

- isEnable：是否同意授权。
- error：错误信息，若 error 为 nil，则获取弹窗结果成功。

账号功能

以下为账号相关接口方法，若需了解调用时机及调用原理，可查看 账号相关流程 。

添加账号

接口说明

若原来没有该类型账号，则添加；若原来有，则覆盖。（移动推送 SDK1.2.9.0+ 新增）。

```
- (void)upsertAccountsByDict:(nonnull NSDictionary<NSNumber *, NSString *> *)accountsDict;
```

> **说明**
> 此接口应该在 xgPushDidRegisteredDeviceToken:error: 返回正确之后被调用。

参数说明

accountsDict：账号字典。

> **说明**
> - 账号类型和账号名称一起作为联合主键。
> - 需要使用字典类型，key 为账号类型，value 为账号，示例：@{@(accountType):@"account"}。
> - Objective-C的写法：@{@(0):@"account0",@(1):@"account1"}；Swift的写法：[NSNumber(0):@"account0",NSNumber(1):@"account1"]。
> - 更多 accountType 请参照 SDK Demo 包内的 XGPushTokenAccountType 枚举或 账号类型取值表 。

示例代码

```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;
NSString *account = @"account";
[[XGPushTokenManager defaultTokenManager] upsertAccountsByDict:@{ @(accountType):account }];
```

添加手机号

接口说明

添加或更新用户手机号，等于调用 `upsertAccountsByDict:@{@(1002):@"具体手机号"}` 。

```
/// @note TPNS SDK1.3.2.0+
- (void)upsertPhoneNumber:(nonnull NSString *)phoneNumber;
```

参数说明

- phoneNumber：E.164标准，格式为+[国家或地区码][手机号],例如：+8613711112222。SDK内部加密传输。

示例代码

```
[[XGPushTokenManager defaultTokenManager] upsertPhoneNumber:@"+8613712345678"];;
```

> **注意**
> - 此接口应该在 `xgPushDidRegisteredDeviceToken:error:` 返回正确之后被调用
> - 如需要删除手机号，调用 `delAccountsByKeys:[[NSSet alloc] initWithObjects:@(1002), nil]`

删除账号

接口说明

接口说明：删除指定账号类型下的所有账号。（移动推送 SDK1.2.9.0+ 新增）。

```
- (void)delAccountsByKeys:(nonnull NSSet<NSNumber *> *)accountsKeys;
```

> **说明**
> 此接口应该在 xgPushDidRegisteredDeviceToken:error: 返回正确之后被调用。

参数说明

- accountsKeys：账号类型组成的集合。

> **说明**
> - 使用集合且 key 是固定要求。
> - 更多 accountType 请参照 SDK 包内 XGPush.h 文件中的 XGPushTokenAccountType 枚举。

示例代码

```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;

NSSet *accountsKeys = [[NSSet alloc] initWithObjects:@(accountType), nil];

[[XGPushTokenManager defaultTokenManager] delAccountsByKeys:accountsKeys];
```

清除账号

接口说明

清除所有设置的账号。

```
- (void)clearAccounts;
```

> **说明**
> 此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码

```
[[XGPushTokenManager defaultTokenManager] clearAccounts];
```

标签功能

以下为标签相关接口方法，若需了解调用时机及调用原理，可查看 标签相关流程 。

绑定/解绑标签

接口说明

开发者可以针对不同的用户绑定标签，然后对该标签进行推送。

```
- (void)appendTags:(nonnull NSArray<NSString *> *)tags
- (void)delTags:(nonnull NSArray<NSString *> *)tags
```

> **说明**
> - 此接口为追加方式。
> - 此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。
> - 单个应用最多可以有10000个自定义 tag， 每个设备 Token 最多可绑定100个自定义 tag，如需提高该限制，请联系 在线客服 ，每个自定义 tag 可绑定的设备 Token 数量无限制。

参数说明

tags：标签数组。

> **说明**
> 标签操作 tags 为标签字符串数组（标签字符串不允许有空格或者是 tab 字符）。

示例代码

```
//绑定标签:
[[XGPushTokenManager defaultTokenManager] appendTags:@[ tagStr ]];

//解绑标签
[[XGPushTokenManager defaultTokenManager] delTags:@[ tagStr ]];
```

更新标签

接口说明

清空已有标签，然后批量添加标签。

```
- (void)clearAndAppendTags:(nonnull NSArray<NSString *> *)tags
```

> **说明**
> - 此接口应在 `xgPushDidRegisteredDeviceToken:error:` 返回正确后被调用。
> - 此接口会将当前 Token 对应的旧有的标签全部替换为当前的标签。

参数说明

tags：标签数组。

> **说明**
> 标签操作 tags 为标签字符串数组（标签字符串不允许有空格或者是 tab 字符）。

示例代码

```
[[XGPushTokenManager defaultTokenManager] clearAndAppendTags:@[ tagStr ]];
```

清除全部标签

接口说明

清除所有设置的标签。

```
- (void)clearTags
```

> **说明**
> 此接口应在 `xgPushDidRegisteredDeviceToken:error:` 返回正确后被调用。

示例代码

```
[[XGPushTokenManager defaultTokenManager] clearTags];
```

查询标签

接口说明

SDK 1.3.1.0 新增，查询设备绑定的标签。

```
- (void)queryTags:(NSUInteger)offset limit:(NSUInteger)limit;
```

> **说明**

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

参数说明

- offset：此次查询的偏移大小。
- offset：limit 此次查询的分页大小, 最大200。

示例代码

```
[[XGPushTokenManager defaultTokenManager] queryTags:0 limit:100];
```

查询标签的回调

接口说明

SDK 1.3.1.0 新增，查询标签的结果会走此回调。

```
- (void)xgPushDidQueryTags:(nullable NSArray<NSString *> *)tags totalCount:(NSUInteger)totalCount error:
(nullable NSError *)error;
```

返回参数说明

- tags：查询条件返回的标签。
- totalCount：设备绑定的总标签数量。
- error：错误信息，若 error 为 nil，则查询成功。

用户属性功能

以下为用户属性相关接口方法，若需了解调用时机及调用原理，可查看 用户属性相关流程 。

新增用户属性

接口说明

添加或更新用户属性（key-value 结构，若原来没有该 key 的用户属性 value，则新增；若原来有该 key 的用户属性 value，则更新该 value）。

```
- (void)upsertAttributes:(nonnull NSDictionary<NSString *,NSString *> *)attributes
```

说明
此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

参数说明

attributes：用户属性字符串字典，字符串不允许有空格或者是 tab 字符。

说明
- 需要先在管理台配置用户属性的键，才能操作成功。
- key，value 长度都限制50个字符以内。
- 需要使用字典且 key 是固定要求。
- Objective-C 的写法 : @{@"gender": @"Female", @"age": @"29"};
- Swift 的写法：["gender":"Female", "age": "29"]

示例代码

```
[[XGPushTokenManager defaultTokenManager] upsertAttributes:attributes];
```

删除用户属性

接口说明

删除用户已有的属性。

```
- (void)delAttributes:(nonnull NSSet<NSString *> *)attributeKeys
```

> **说明**
> 此接口应在 `xgPushDidRegisteredDeviceToken:error:` 返回正确后被调用。

参数说明

attributeKeys：用户属性 key 组成的集合，字符串不允许有空格或者是 tab 字符。

> **说明**
> 使用集合且 key 是固定要求。

示例代码

```
[[XGPushTokenManager defaultTokenManager] delAttributes:attributeKeys];
```

清空已有用户属性

接口说明

清空已有用户属性。

```
- (void)clearAttributes;
```

> **说明**
> 此接口应在 `xgPushDidRegisteredDeviceToken:error:` 返回正确后被调用。

示例代码

```
[[XGPushTokenManager defaultTokenManager] clearAttributes];
```

更新用户属性

接口说明

清空已有用户属性，然后批量添加用户属性。

```
- (void)clearAndAppendAttributes:(nonnull NSDictionary<NSString *,NSString *> *)attributes
```

> **说明**
> 此接口应在 `xgPushDidRegisteredDeviceToken:error:` 返回正确后被调用。

示例代码

```
[[XGPushTokenManager defaultTokenManager] clearAndAppendAttributes:attributes];
```

角标功能

同步角标

接口说明

当应用本地角标值更改后，需调用此接口将角标值同步到移动推送服务器，下次推送时以此值为基准，此功能在管理台位置（【新建推送】>【高级设置】>【角标数字】）。

```
- (void)setBadge:(NSInteger)badgeNumber;
```

参数说明

badgeNumber：应用的角标数。

> **注意**
> 当本地应用角标设置后需调用此接口同步角标值到移动推送服务器，并在下次推送时生效，此接口必须在移动推送长链接建立后调用（xgPushNetworkConnected）。

示例代码

```
/// 冷启动调用时机
- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nullable NSString
*)xgToken error:(nullable NSError *)error {
    /// 在注册完成后上报角标数目
    if (!error) {
     /// 重置应用角标，-1不清空通知栏，0清空通知栏
     [XGPush defaultManager].xgApplicationBadgeNumber = -1;
        /// 重置服务端自动+1基数
        [[XGPush defaultManager] setBadge:0];
    }
}

/// 热启动调用时机
/// _launchTag热启动标识，业务自行管理
- (void)xgPushNetworkConnected {
    if (_launchTag) {
        /// 重置应用角标，-1不清空通知栏，0清空通知栏
        [XGPush defaultManager].xgApplicationBadgeNumber = -1;
        /// 重置服务端自动+1基数
        [[XGPush defaultManager] setBadge:0];
        _launchTag = NO;
    }
}
```

实时活动 LiveActivity

接口说明

iOS 应用端 ActivityKit 初始化实时活动成功后上报 activityId 及 pushToken 给 TPNS Server 或者用来结束实时活动。

```
/**
 @brief 实时活动回调结果
 @param result 返回请求的liveActivity的id和token @{@"activityId":@"idValue", @"token":@"tokenValue"}
 */
typedef void(^XGPushLiveActivityCompletion)(NSDictionary * _Nonnull result, NSError *_Nullable error);

/**
 @brief 开始实时活动，pushToken有变化时需重新调用

 @param activityId liveActivity的名字
 @param token liveActivity对应的pushToken，该token有变化时需要及时调用此方法更新
 @param completionHandler 请求回执，error为nil代表请求成功
```

```
 @note 实时活动结束时需要及时调用该方法且pushToken传nil
 */
- (void)startLiveActivityWithId:(nonnull NSString *)activityId pushToken:(nullable NSString *)token
withCompletionHandler:(nullable XGPushLiveActivityCompletion)completionHandler __API_AVAILABLE(ios(16.1));
```

参数说明

- activityId：liveActivity 的名字，例如某场活动或比赛。
- token：liveActivity 对应的 pushToken，该 token 有变化时需要及时调用此方法更新。
- completionHandler：请求回执，error 为 nil 代表请求成功，参考 XGPushLiveActivityCompletion 类型。

> **说明：**
> 实时活动仅支持 P8 证书鉴权方式，推送实时活动前，您需要在 Apple 开发者中心申请 P8 证书，并上传到 TPNS 管理台，请参见 API 协议文档 进行推送。

示例代码：

```
XGPush.defaultManager().startLiveActivity(withId: orderActivity.id, pushToken: DeviceToken(data:
data).hexString, withCompletionHandler: {(p1, p2) in})
```

查询设备通知权限

接口说明

查询设备通知权限是否被用户允许。

```
- (void)deviceNotificationIsAllowed:(nonnull void (^)(BOOL isAllowed))handler;
```

参数说明

handler：查询结果的返回方法。

示例代码

```
[[XGPush defaultManager] deviceNotificationIsAllowed:^(BOOL isAllowed) {
        <#code#>
    }];
```

查询 SDK 版本

接口说明

查询当前 SDK 的版本。

```
- (nonnull NSString *)sdkVersion;
```

示例代码

```
[[XGPush defaultManager] sdkVersion];
```

日志上报接口

接口说明

开发者如果发现推送相关功能异常，可以调用该接口，触发本地 push 日志的上报，通过联系 在线客服 反馈问题时，请将文件地址提供给我们，便于排查问题。

```
/// @note TPNS SDK1.2.4.1+
```

```
- (void)uploadLogCompletionHandler:(nullable void(^)(BOOL result,  NSString * _Nullable
errorMessage))handler;
```

参数说明

- @brief：上报日志信息（SDK1.2.4.1+）。
- @param handler：上报回调。

示例代码

```
[[XGPush defaultManager] uploadLogCompletionHandler:nil];
```

移动推送日志托管

接口说明

可以在此方法获取移动推送的 log 日志。此方法和 XGPush > enableDebug 无关。

参数说明

logInfo：日志信息。

示例代码

```
- (void)xgPushLog:(nullable NSString *)logInfo;
```

自定义通知栏消息行为

创建消息支持的行为

接口说明

在通知消息中创建一个可以点击的事件行为。

```
+ (nullable id)actionWithIdentifier:(nonnull NSString *)identifier title:(nonnull NSString *)title
options:(XGNotificationActionOptions)options;
```

参数说明

- identifier：行为唯一标识。
- title：行为名称。
- options：行为支持的选项。

示例代码

```
XGNotificationAction *action1 = [XGNotificationAction actionWithIdentifier:@"xgaction001"
title:@"xgAction1" options:XGNotificationActionOptionNone];
```

> ⚠ 注意
> 通知栏带有点击事件的特性，只有在 iOS8.0+ 以上支持，iOS 7.x or earlier 的版本，此方法返回空。

创建分类对象

接口说明

创建分类对象，用以管理通知栏的 Action 对象。

```
+ (nullable id)categoryWithIdentifier:(nonnull NSString *)identifier actions:(nullable NSArray<id>
*)actions intentIdentifiers:(nullable NSArray<NSString *> *)intentIdentifiers options:
(XGNotificationCategoryOptions)options
```

参数说明

- identifier：分类对象的标识。
- actions：当前分类拥有的行为对象组。
- intentIdentifiers：用以表明可以通过 Siri 识别的标识。
- options：分类的特性。

> **注意**
> 通知栏带有点击事件的特性，只有在 iOS8+ 以上支持，iOS 8 or earlier的版本，此方法返回空。

示例代码

```
XGNotificationCategory *category = [XGNotificationCategory categoryWithIdentifier:@"xgCategory"
actions:@[action1, action2] intentIdentifiers:@[] options:XGNotificationCategoryOptionNone];
```

创建配置类

接口说明

管理推送消息通知栏的样式和特性。

```
+ (nullable instancetype)configureNotificationWithCategories:(nullable NSSet<id> *)categories types:
(XGUserNotificationTypes)types;
```

参数说明

- categories：通知栏中支持的分类集合。
- types：注册通知的样式。

示例代码

```
XGNotificationConfigure *configure = [XGNotificationConfigure configureNotificationWithCategories:[NSSet
setWithObject:category]
types:XGUserNotificationTypeAlert|XGUserNotificationTypeBadge|XGUserNotificationTypeSound];
```

本地推送

本地推送相关功能请参见 苹果开发者文档 。

推送证书获取指引

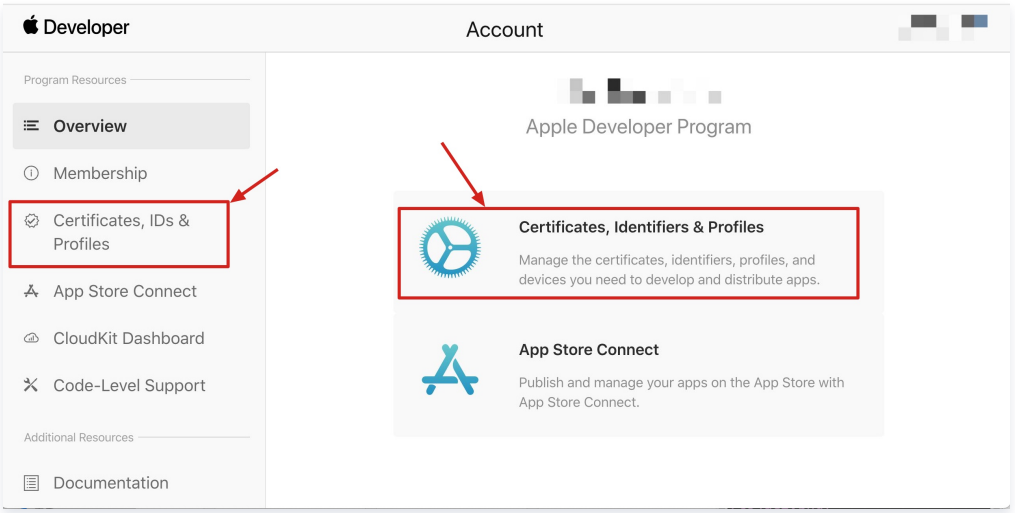
最近更新时间：2024-05-09 11:34:53

本文档主要指导您如何生成 iOS 消息推送证书及上传证书。

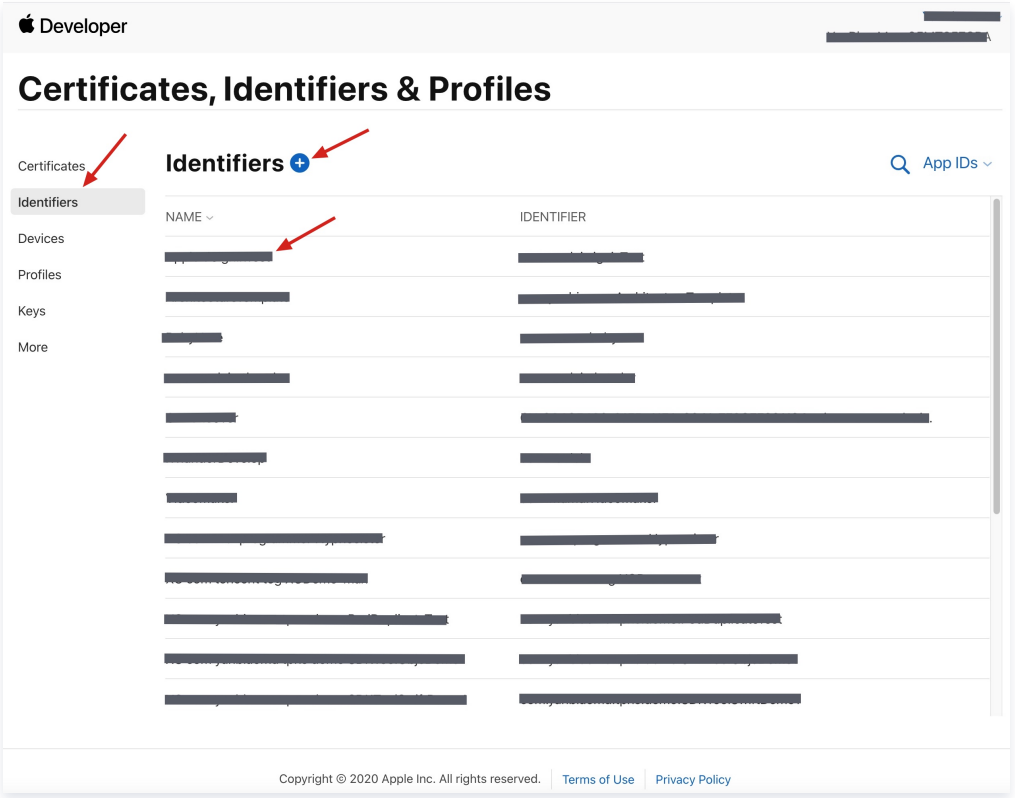
> **说明**
> 移动推送更推荐您使用 p12 来分别管理您的应用的推送服务；虽然 p8 比 p12 有更长的有效期，但是同时也有更大的推送权限和范围，若泄露，可能会造成更加严重的影响。

步骤1：为 App 开启远程推送服务

1. 登录 苹果开发者中心 网站，单击**Certificates,Identifiers & Profiles**或者侧栏的**Certificates, IDs & Profiles**，进入 Certificates, IDs & Profiles 页面。

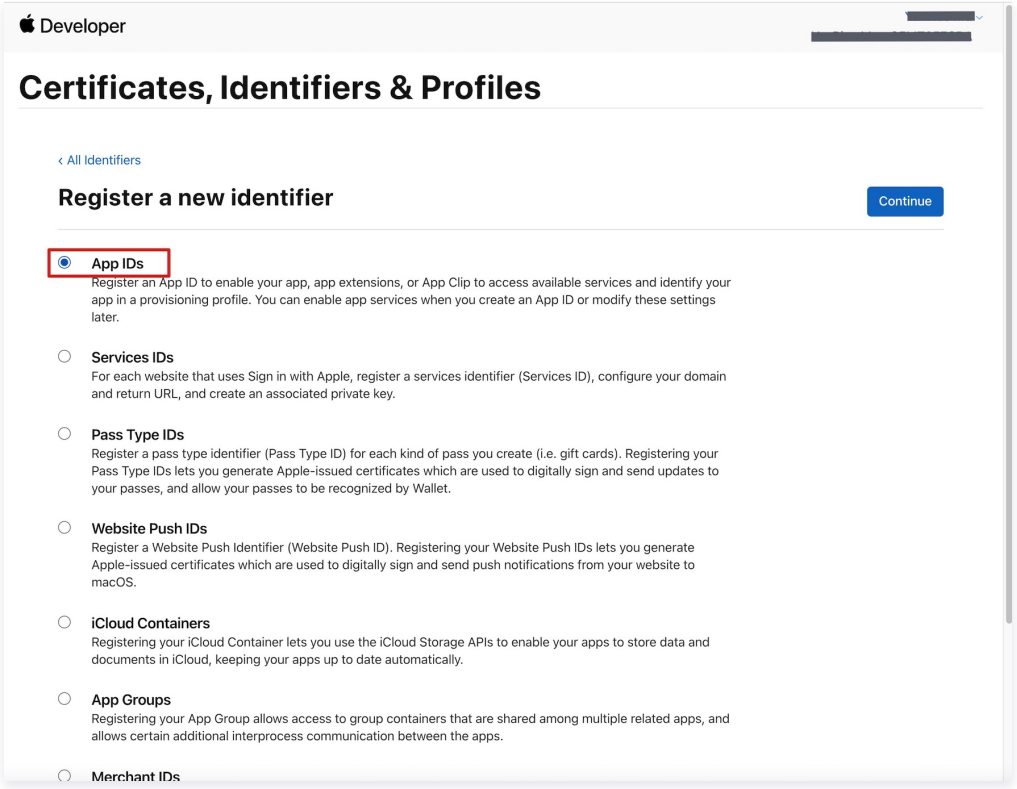


2. 单击 Identifiers 右侧的 +。

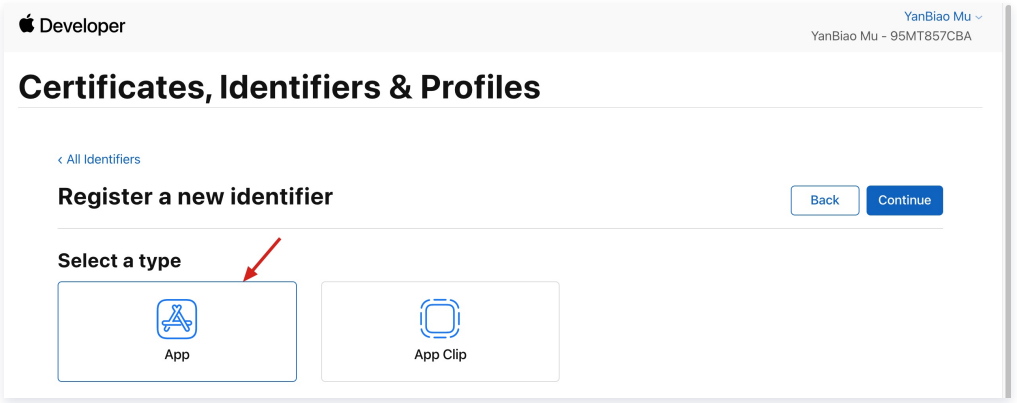


3. 您可以参考如下步骤新建一个 AppID，或者在您原有的 AppID 上增加 `Push Notification` 的 `Service`。需要注意的是，您 App 的 `Bundle ID` 不能使用通配符 `*`，否则将无法使用远程推送服务。

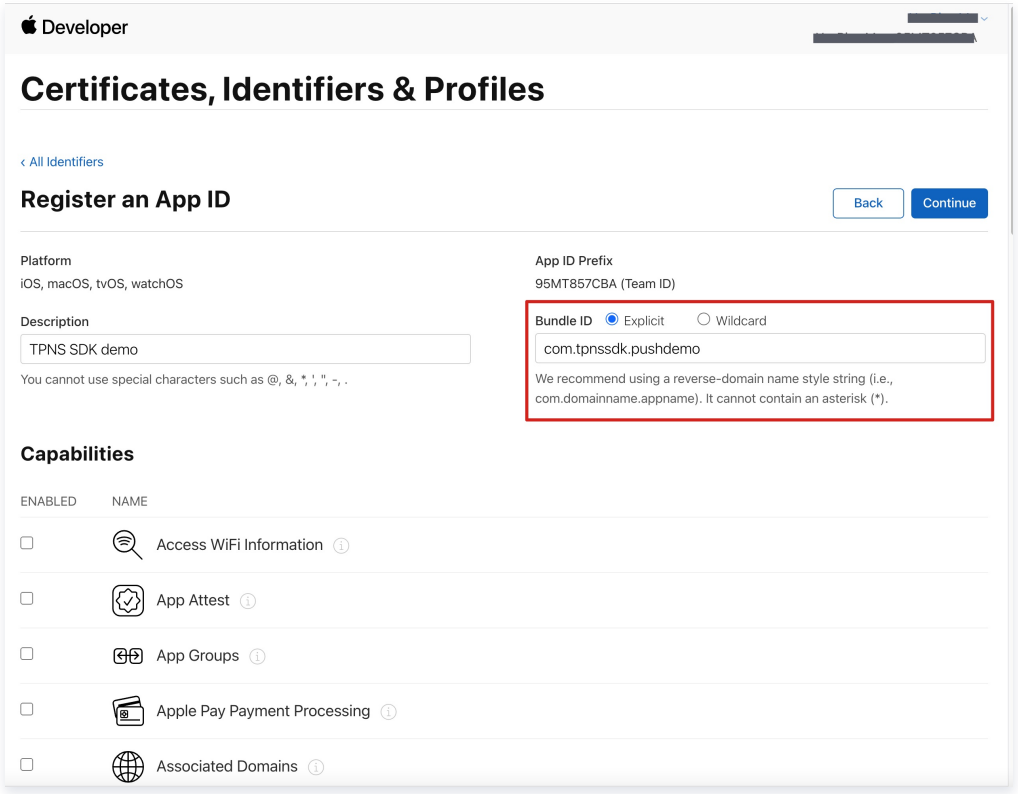
3.1 勾选 App IDs，单击 Continue 进行下一步。



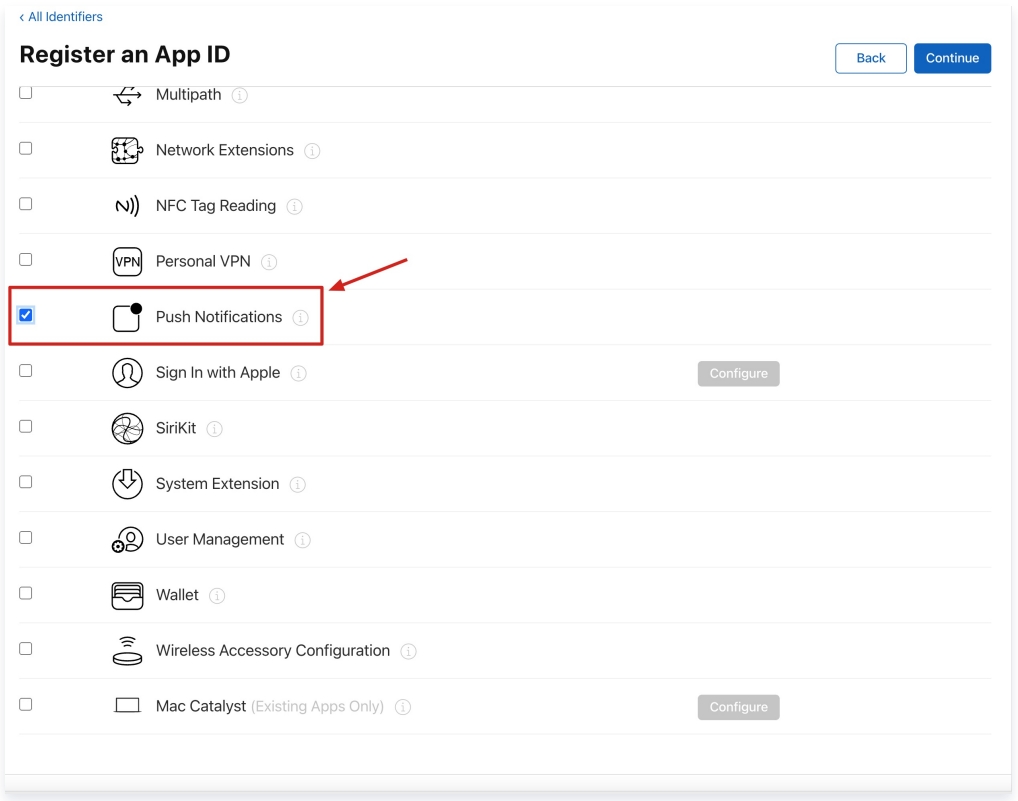
3.2 选择App，单击 Continue 进行下一步。



3.3 配置 `Bundle ID` 等其他信息，单击Continue进行下一步。

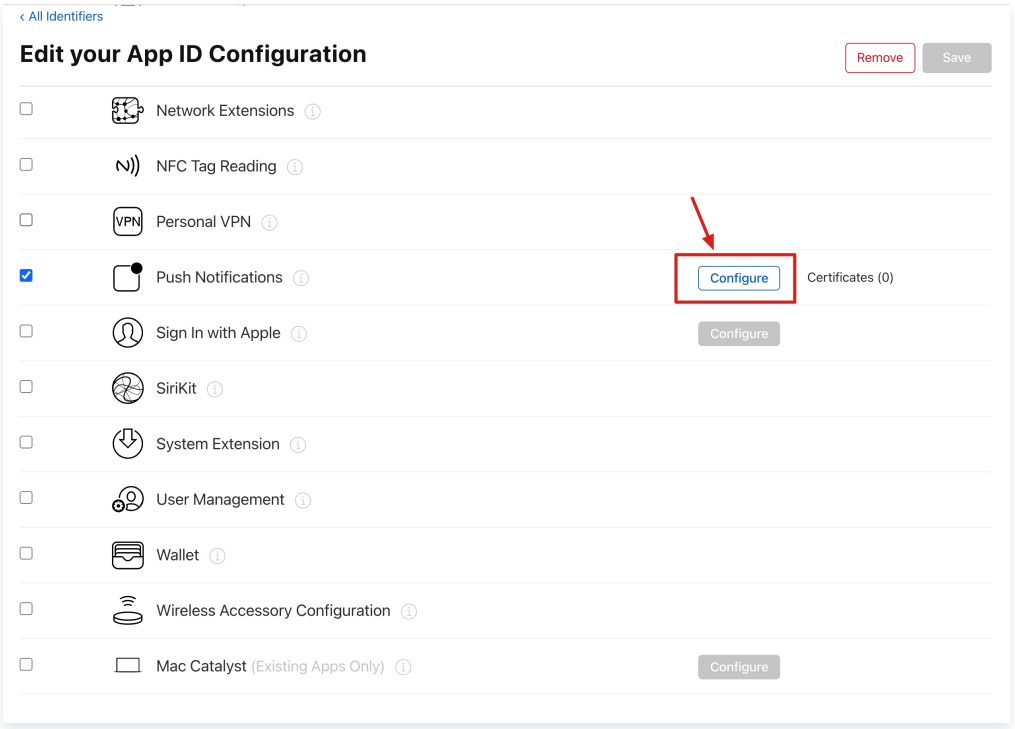


3.4 勾选Push Notifications，开启远程推送服务。

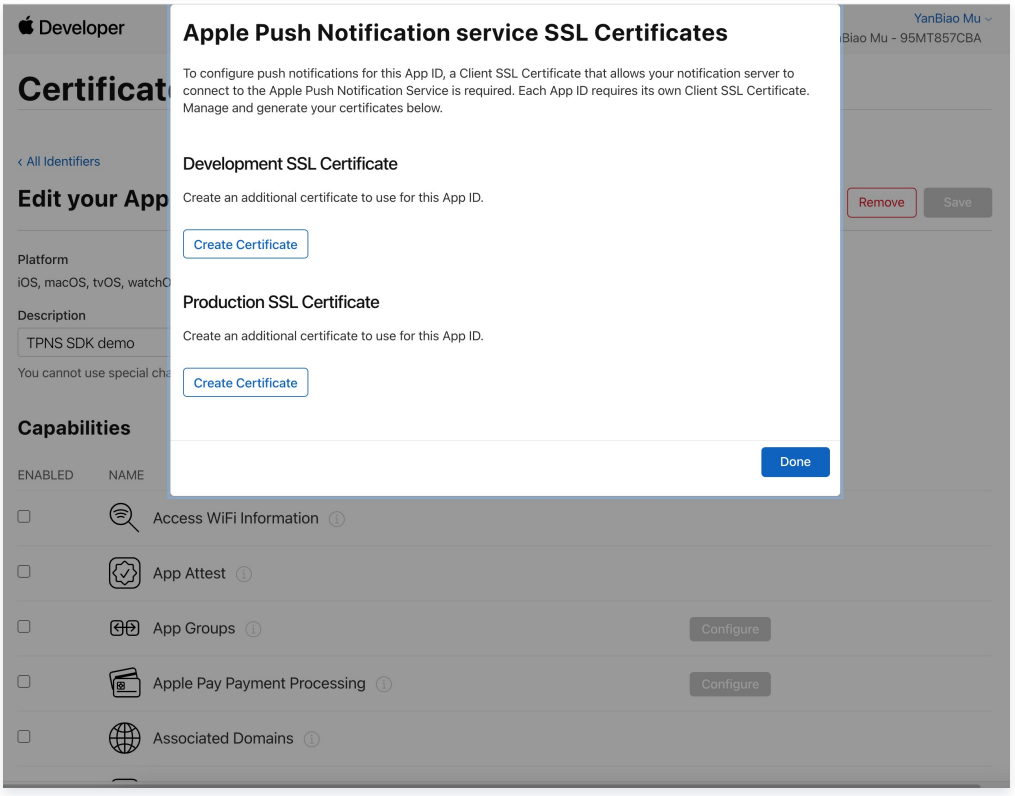


步骤2：生成并上传 P12 证书

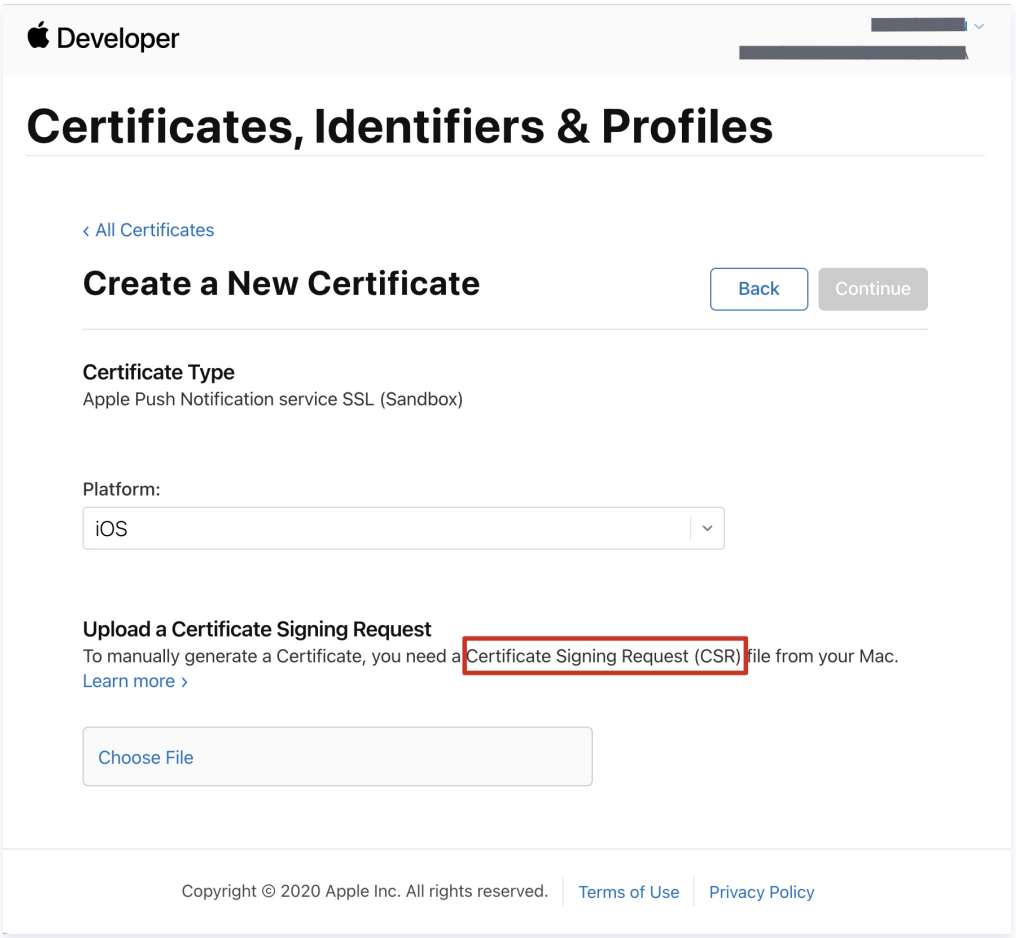
1. 选中您的 AppID，选择 **Configure**。



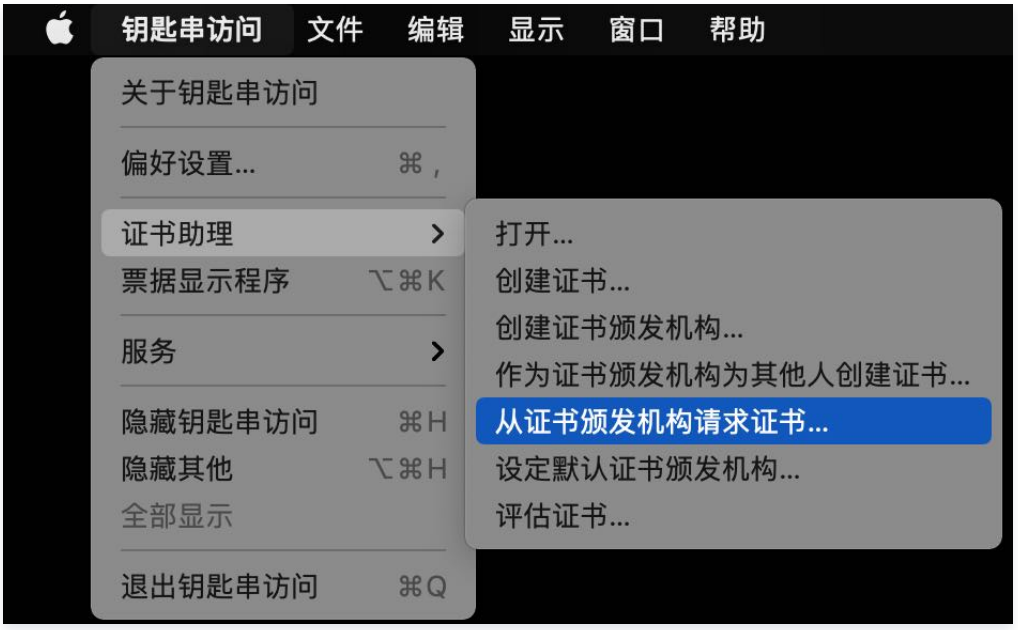
2. 可以看到在 **Apple Push Notification service SSL Certificates** 窗口中有两个 `SSL Certificate` ，分别用于开发环境（ `Development` ）和生产环境（ `Production` ）的远程推送证书，如下图所示：



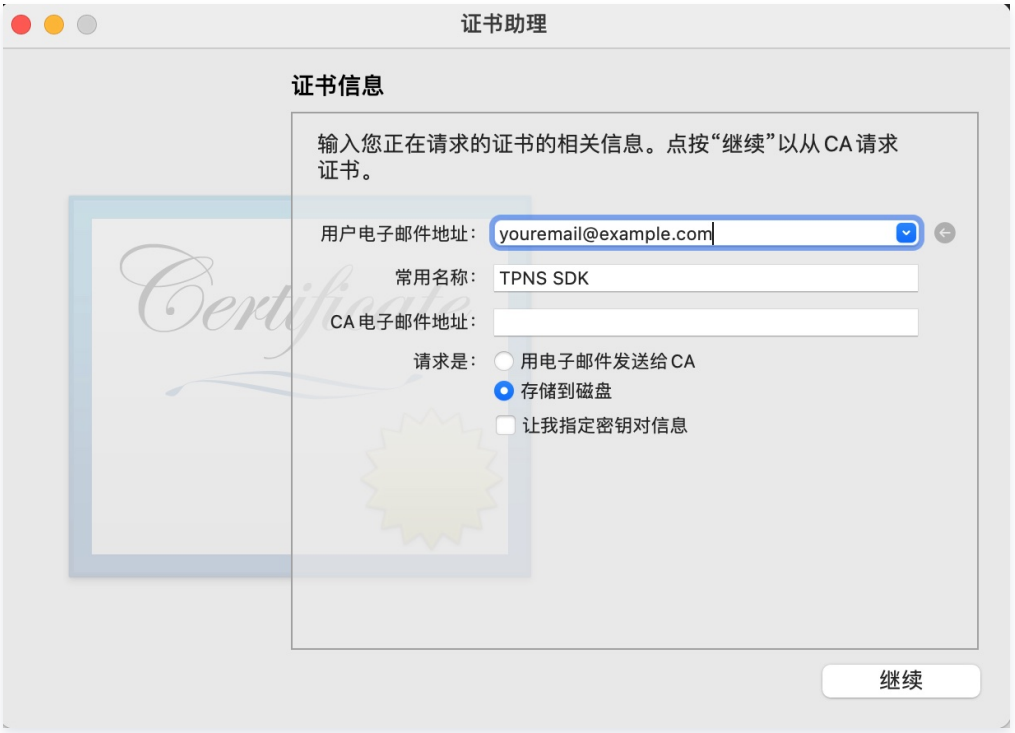
3. 我们先选择开发环境（Development）的**Create Certificate**，系统将提示我们需要一个 Certificate Signing Request（CSR）。



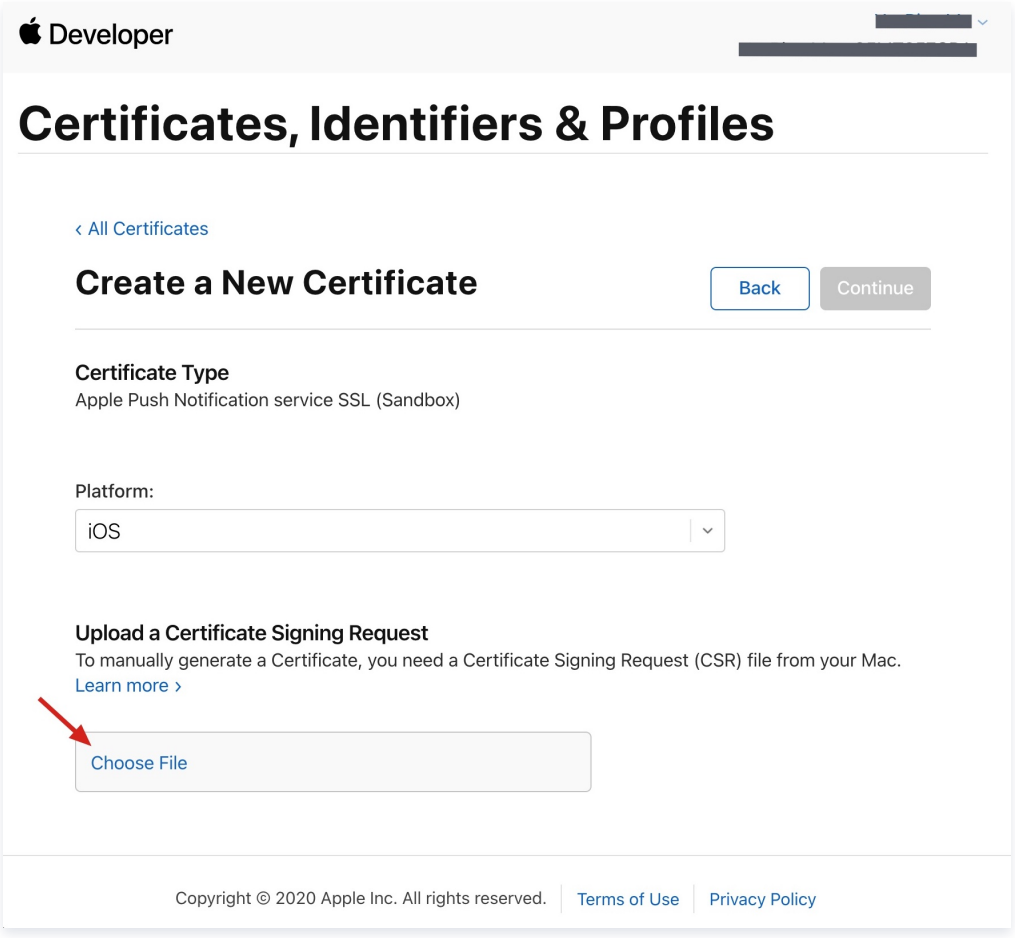
4. 在 Mac 上打开**钥匙串访问工具（Keychain Access）**，在菜单中选择**钥匙串访问>证书助理>从证书颁发机构请求证书**（`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`）。



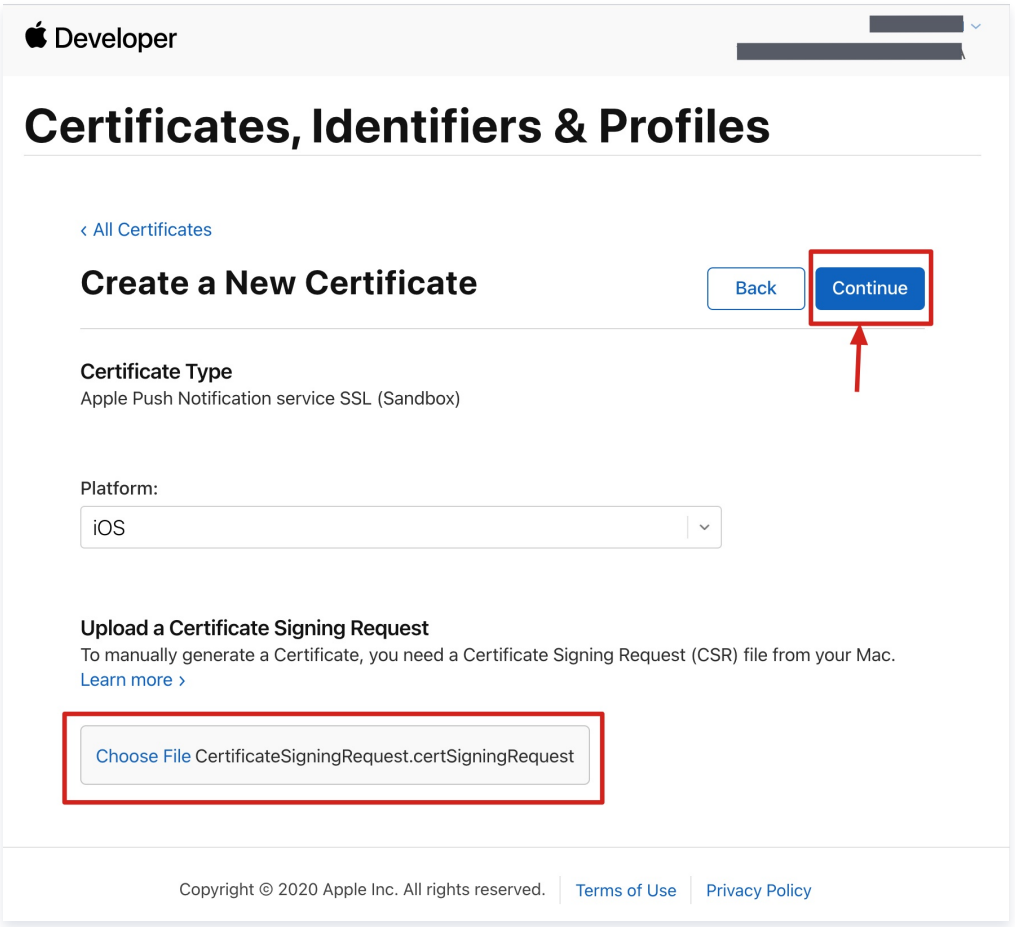
5. 输入用户电子邮件地址（您的邮箱）、常用名称（您的名称或公司名），选择**存储到磁盘**，单击继续，系统将生成一个 `*.certSigningRequest` 文件。



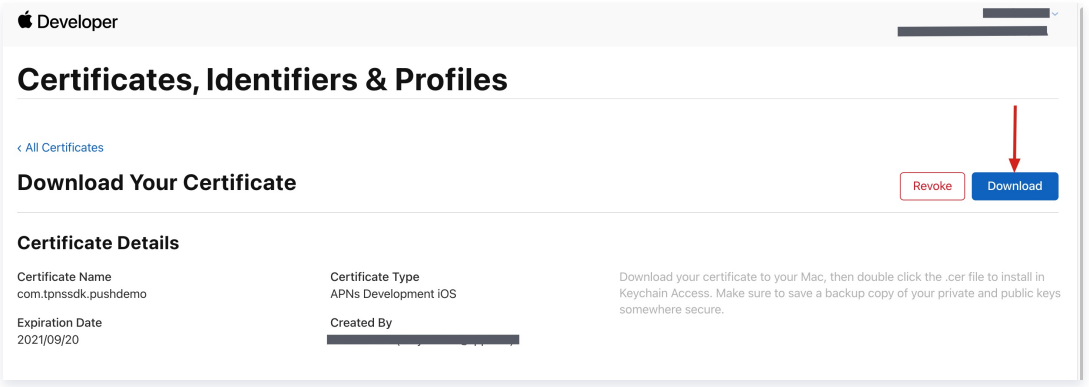
6. 返回上述 第3步骤 中 `Apple Developer` 网站刚才的页面，单击**Choose File**上传生成的 `*.certSigningRequest` 文件。



7. 单击 **Continue**，即可生成推送证书。



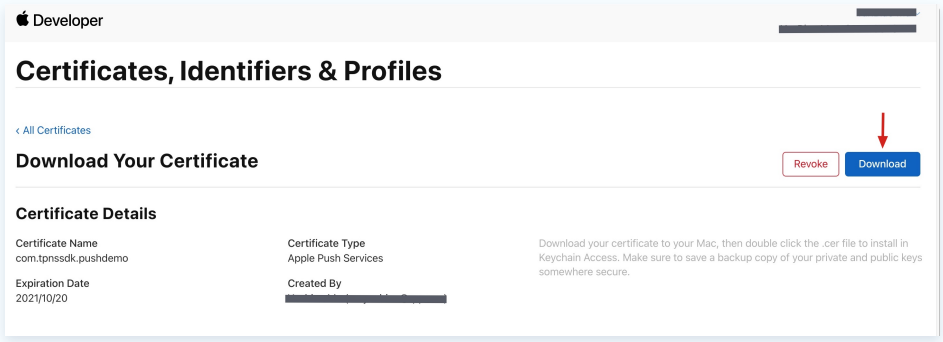
8. 单击 **Download** 下载开发环境的 `Development SSL Certificate` 到本地。



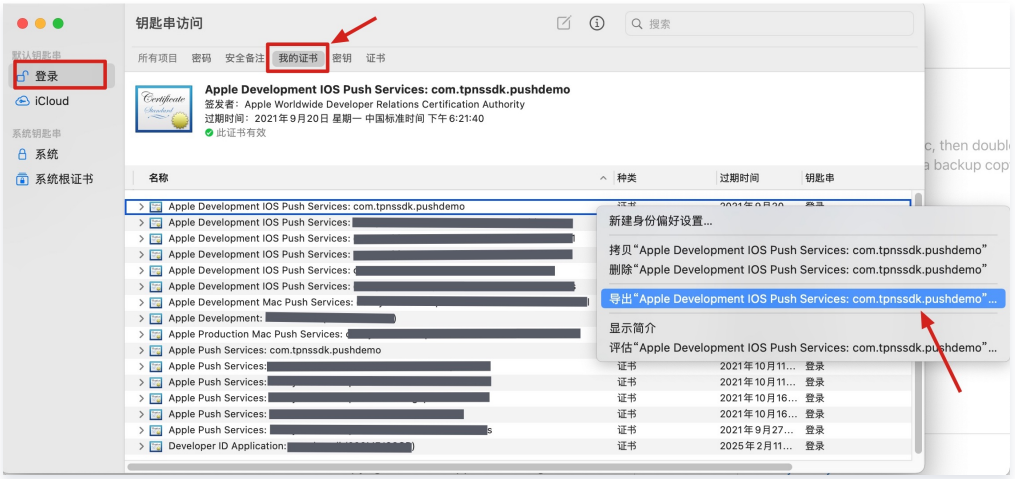
9. 按照上述步骤，将生成生产环境的 `Production SSL Certificate` 并下载到本地。

> **说明**

生产环境的证书实际是开发(`Sandbox`)+生产(`Production`)的合并证书，可以同时作为开发环境和生产环境的证书使用。



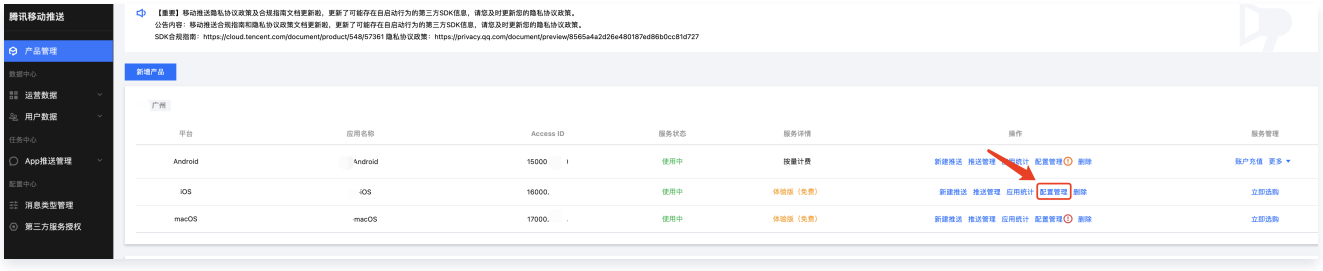
10. 双击打开下载的开发环境和生产环境的 `SSL Certificate` ，系统会将其导入钥匙串中。
11. 打开钥匙串应用，在**登录 > 我的证书**，右键分别导出开发环境（ `Apple Development IOS Push Service: com.tpnssdk.pushdemo` ）和生产环境（ `Apple Push Services: com.tpnssdk.pushdemo` ）的 `P12` 文件。



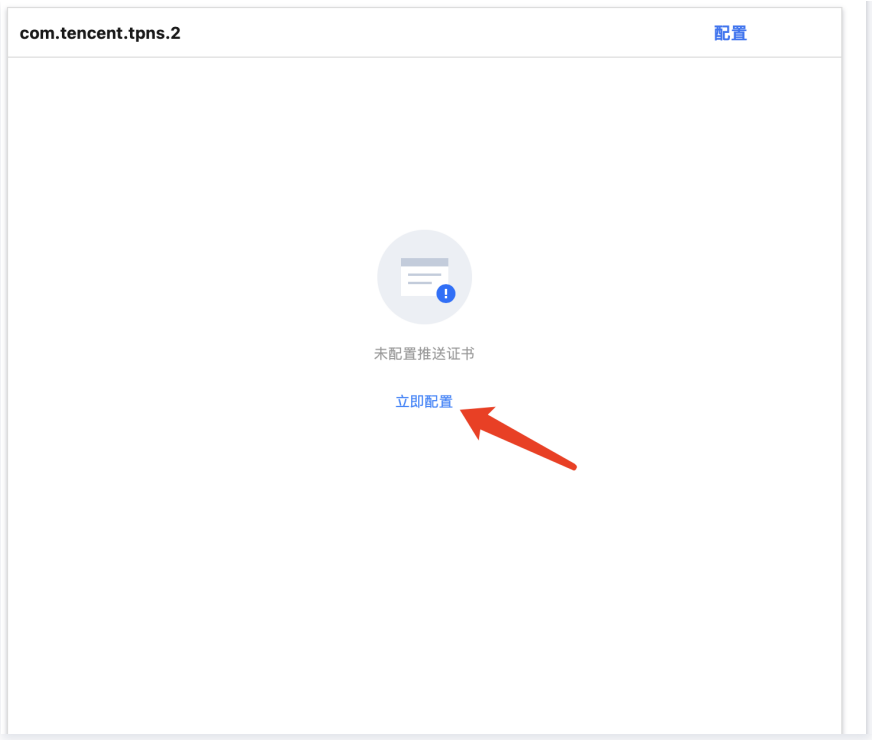
> **注意**
> 保存 `P12` 文件时，请务必要为其设置密码。

步骤3：上传证书到移动推送控制台

1. 登录 移动推送控制台，选择**产品管理 > 基础配置**。



2. 进入证书上传页面，单击**立即配置**，上传 p12 **生产&开发环境**公用或者上传**环境独立**证书，若使用实时活动需上传 **p8证书**。



3. 输入对应的证书密码，单击**点击选择**，选择您的证书，最后单击**提交**即可。

推送环境选择说明

最近更新时间：2023-05-22 11:03:54

管理台推送消息时，测试推送消息有两种环境可以选择。
开发环境：需要确定 App 已使用开发环境的签名证书打包，使用 `Xcode` 直接编译安装到设备。
生产环境：需要确定 App 已使用生产环境的签名证书打包，生产环境的 App 有以下3种打包方式： `Ad-Hoc` `TestFlight` `AppStore` 。

服务端指定推送环境

当您使用 `RESTAPI` 推送消息时，需要在 `PushAPI` 中指定 `environment` 字段，此字段中有两个可选值 `product` 和 `dev` 。
开发环境：需要指定 `environment` 为 `dev` 。
生产环境：需要指定 `environment` 为 `product` 。

推送证书说明

在管理台中需要上传开发环境和生产环境的二合一推送证书（Apple Push Notification service SSL (Sandbox & Production)）。此证书可以推送生产环境和开发环境，根据 App 实际使用的签名证书来进行选择，选择的方式参照上文。

> **说明**
> - App 签名证书，分为开发环境（对应 `xxx Developer:xxx` 字样的签名证书）和生产环境（对应 `xxx Distribution:xxx` 字样的签名证书），请根据实际情况选择。
> - App 推送证书为合并证书，已兼容开发环境与生产环境。

错误码

最近更新时间：2025-03-25 15:34:43

客户端返回码

错误码	含义
101	Guid 请求超时
102	Guid 请求错误
701	SDK 异常
801	长链接超时
802	长链接超时，与当天启动数有关
901	长链接错误
903	长链接 checksum 错误
904	长链接 checksum 反解错误
1001	deviceToken 为空，厂商 Token 未拿到，请检查是否开启消息推送能力
1101	设备网络错误
1102	未注册
1103	App 信息或路由配置错误
1104	业务接口操作类型传入错误
1105	业务接口参数传入错误
1106	业务接口参数为空
1107	系统不支持
1108	插件引入错误
1109	插件版本不匹配
1110	启动失败
1111	内存不足
1112	云控限制
1401	短连接拉取离线消息失败
1402	短连接拉取离线消息大于1
1501	建立长链接失败
1502	建立长链接失败，应用非前台
1503	限频错误
1504	长链接业务请求超时（最小30秒）

服务端返回码

错误码	含义
1010001	资源未部署，请确认应用是否已购买推送资源

1008001	参数解析错误
1008002	必填参数缺失
1008003	认证失败
1008004	调用服务失败
1008006	Token 无效，请检查设备 Token 是否注册成功
1008007	参数校验失败
1008011	文件上传失败
1008012	上传文件为空
1008013	证书解析失败
1008015	推送任务 ID 不存在
1008016	日期时间参数格式不对
1008019	被内容安全服务判定不和谐
1008020	证书包名校验失败
1008021	p12 证书格式内容校验失败
1008022	p12 证书密码不对
1008025	创建应用失败，产品下已存在该平台的应用
1008026	批量操作，部分失败
1008027	批量操作，全部失败
1008028	超出限频
1008029	Token 校验非法
1008030	App 未付费
1008031	App 资源已销毁
10110008	查询的 Token， 账号不存在
10010005	推送目标不存在
10010012	非法的推送时间，请更改推送时间。 定时推送时 send_time 传入的值如果是过去的时间，具体规则如下： • 如果 | send_time − 当前时间 | <= 10min， 推送任务会创建，接收任务后会立即调度 • 如果 | send_time − 当前时间 | > 10min， 推送任务会被拒绝， 接口返回失败
10010018	重复推送
10030002	AccessID 和 AccessKey 不匹配

拓展功能
通知服务扩展的使用说明

最近更新时间：2025-03-25 15:34:43

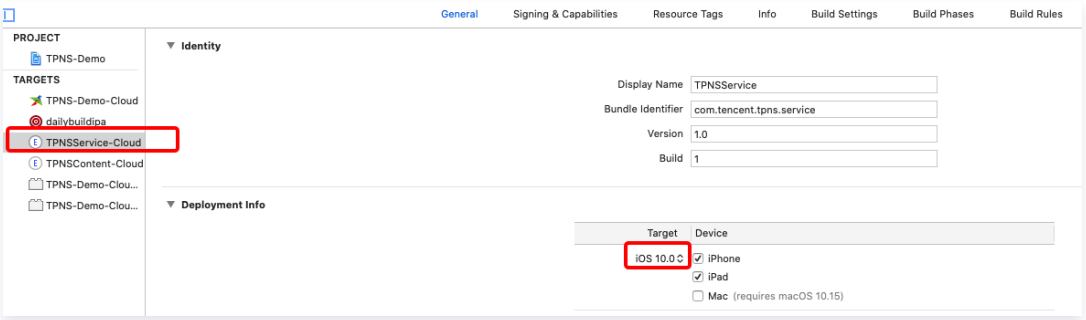
简介

为了**精准统计消息抵达率和接收富媒体消息**，SDK 提供了 Service Extension 接口，可供客户端调用，从而可以监听消息的到达和接收富媒体消息，您可以按以下指引使用此功能。

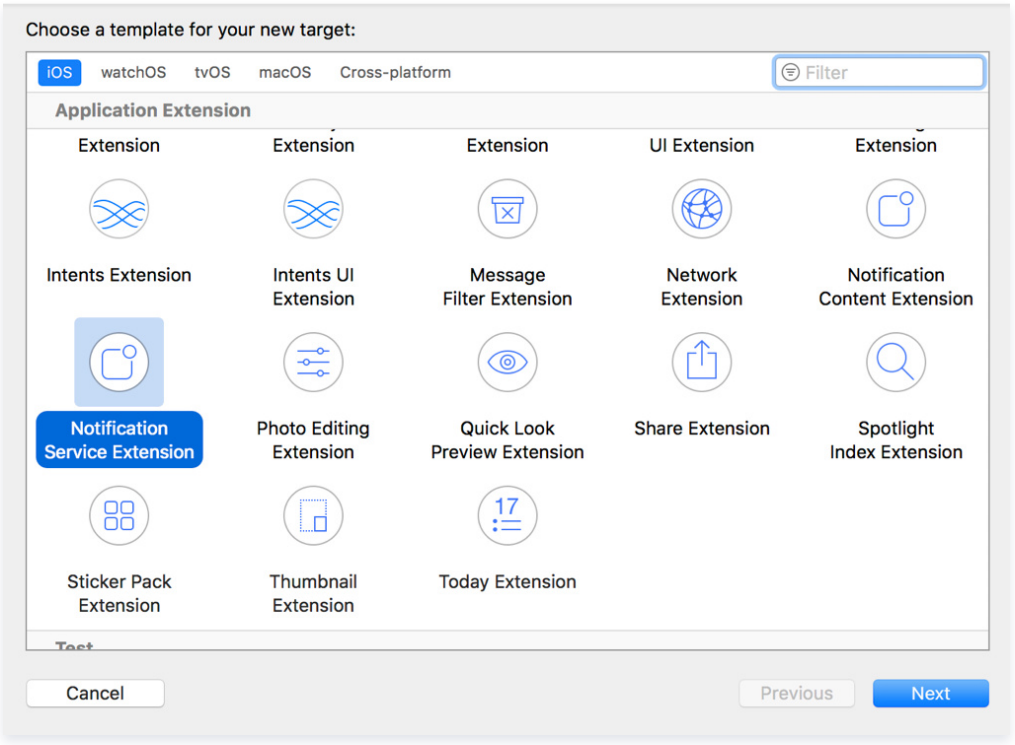
创建通知拓展 Target

1. 在 xcode 菜单栏，选择 **File>New> Target**。

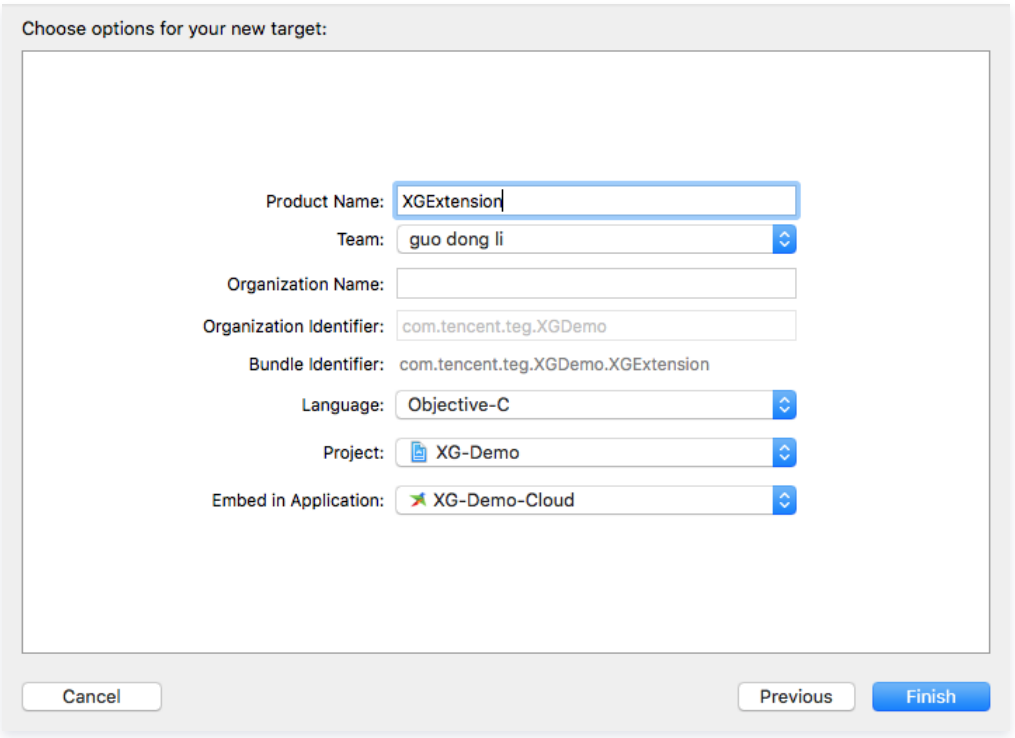
> **说明**
> - 主工程的 Bundle Id 和 Service 的 Bundle Id 必须不同，且 Service 的 Bundle Id 必须以主工程的 Bundle Id 为前缀（例如，主工程的 Bundle Id：com.tencent.tpns，Service 的 Bundle Id：com.tencent.tpns.service）。
> - 若主工程 Target 最低支持版本小于10.0，扩展 Target 系统版本设置为10.0。
> - 若主工程 Target 最低支持版本大于10.0，则扩展 Target 系统版本与主工程 Target 版本一致。



2. 进入 Target 页面，选择**Notification Service Extension**， 单击**Next**。



3. 输入 Product Name，单击 **Finish**。



添加移动推送扩展库（三选一）

方式一：Cocoapods 集成

通过 Cocoapods 下载地址：

```
pod 'TPNS-iOS-Extension', '~> 版本'  // 如果不指定版本，则默认为本地 pod TPNS-iOS-Extension 最新版本
```

使用说明：

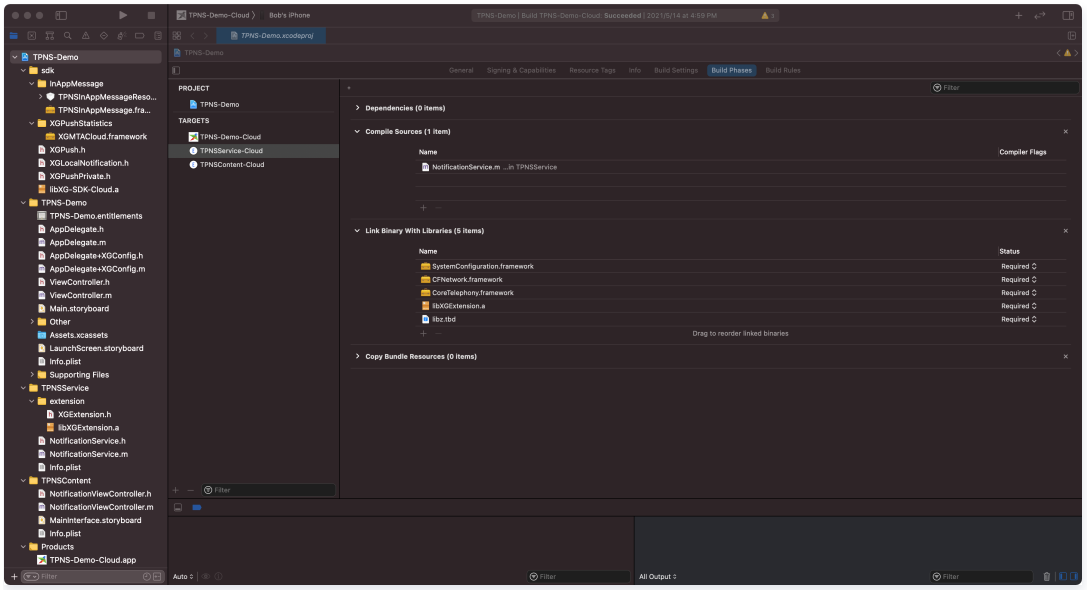
1. 创建类型为 `Application Extension` 的 `Notification Service Extension` TARGET，例如 `XXServiceExtension`。
2. 在 Podfile 新增 XXServiceExtension 的配置栏目。
   Podfile 中增加配置项目后展示效果，示例如下：

```
target 'XXServiceExtension'do
  platform:ios,'10.0'
  pod 'TPNS-iOS-Extension' , '~> 版本' // 需要与主SDK（TPNS-iOS）版本保持一致
end
```

方式二：手动集成

1. 登录 腾讯移动推送控制台 。
2. 在左侧导航栏中，单击**工具箱** > ** SDK 下载 **，进入 SDK 下载管理页面。
3. 在 SDK 下载管理页面，选择 iOS 平台 ，单击**下载**。
4. 解压缩 SDK 包，并依次打开 demo > sdk > XGPushStatistics > extension 目录，获取 `XGExtension.h` 及 `libXGExtension.a` 文件。
5. 将获取到的 `XGExtension.h` 及 `libXGExtension.a` 文件添加至通知扩展 Target：
   - 添加系统库：libz.tbd
   - 移动推送扩展库：libXGExtension.a

集成后的目录结构如下：



方式三：HomeBrew 集成

首次安装 `new_tpns_svc_ext`，请在终端执行下面的命令：

1. 关联 移动推送 `homebrew` 仓库。

```
brew tap tpns/serviceExtension https://github.com/TencentCloud/homebrew-tpnsServiceExtension.git
```

2. 安装 `new_tpns_svc_ext` 命令行。

```
brew install new_tpns_svc_ext
```

3. 安装移动推送通知扩展插件。

```
new_tpns_svc_ext  "AccessID" "AccessKey" "xxx.xcodeproj"
```

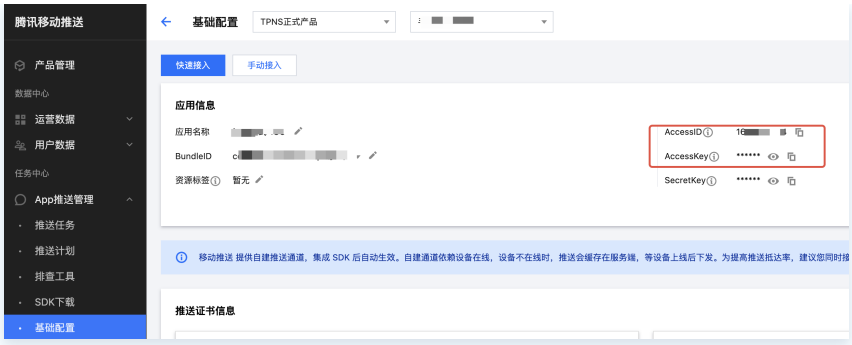
参数说明：

- 参数1：腾讯云-腾讯移动推送-您产品的 `AccessID`
- 参数2：腾讯云-腾讯移动推送-您产品的 `AccessKey`
- 参数3： `.xcodeproj` 的完整路径

使用示例：

```
new_tpns_svc_ext "1600013400" "IWRNAHX6XXK6"
"/Users/yanbiaomu/Developer/tencent/demo2/SDKToolObjcDemo2/SDKToolObjcDemo2.xcodeproj"
```

> **说明**
> 参数1、2的获取方式：请在 移动推送控制台 >产品管理>选中您要配置推送能力的产品>基础配置中，粘贴复制 `AccessID` 和 `AccessKey` 到命令行 `new_tpns_svc_ext` 的参数1、2中。



4. 执行 `new_tpns_svc_ext` 命令，进行结果验证。在终端执行 `new_tpns_svc_ext` 命令之后，如果输出如下结果，即说明集成通知扩展插件成功。

```
TPNS service auto coding done!
New TPNSService Extension Success
```

升级 `new_tpns_svc_ext` 版本

当有新版本 SDK 通知扩展插件发布，可在终端执行如下命令进行升级：

```
brew update && brew reinstall new_tpns_svc_ext
```

> **说明**
> - 您可以在 iOS 发布动态页 查看最新版本更新详情。
> - 当前 `homebrew` 命令 `new_tpns_svc_ext` 只支持集成通知扩展插件 `TPNSService`，暂不支持基础推送能力的集成。

使用方式

调用 SDK 统计上报接口

1. 在通知扩展类 NotificationService 引入头文件 XGExtension.h。
2. 在回调方法 didReceiveNotificationRequest:withContentHandler: 中调用如下示例代码：

```
/**
 @brief TPNS处理富媒体通知和抵达到终端的消息，即消息回执
 @param request 推送请求
 @param accessID TPNS应用 accessId
 @param accessKey TPNS应用 accessKey
 @param handler 处理消息的回调，回调方法中处理关联的富媒体文件
 */
  - (void)handleNotificationRequest:(nonnull UNNotificationRequest *)request
                           accessID:(uint32_t)accessID
                          accessKey:(nonnull NSString *)accessKey
                     contentHandler:(nullable void (^)(NSArray<UNNotificationAttachment *> *_Nullable
attachments, NSError *_Nullable error))handler;
```

示例代码：

```
- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentHandler:(void (^)
(UNNotificationContent *_Nonnull))contentHandler {
    self.contentHandler = contentHandler;
    self.bestAttemptContent = [request.content mutableCopy];
     /// 非广州集群，请开启对应集群配置（广州集群无需使用）
    //  [XGExtension defaultManager].reportDomainName = @"tpns.hk.tencent.com"; /// 中国香港集群
    //  [XGExtension defaultManager].reportDomainName = @"tpns.sgp.tencent.com";  /// 新加坡集群
    //  [XGExtension defaultManager].reportDomainName = @"tpns.sh.tencent.com";  /// 上海集群
```

```
    [[XGExtension defaultManager] handleNotificationRequest:request accessID:<your accessID> accessKey:
<your accessKey
        > contentHandler:^(NSArray<UNNotificationAttachment *> * _Nullable attachments, NSError *
_Nullable error) {
        self.bestAttemptContent.attachments = attachments;
        self.contentHandler(self.bestAttemptContent);  // 如果需要在弹出通知前增加业务逻辑，请在contentHandler
调用之前处理。
    }];
}
```

接入验证

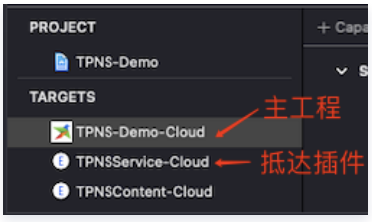
在您按以上流程完成接入后，可按如下步骤验证插件接入是否成功：

1. 关闭应用，给手机推送一条通知消息。
2. 在不点击的情况下，管理台查看消息抵达情况。
   若有抵达数据，表示集成成功。

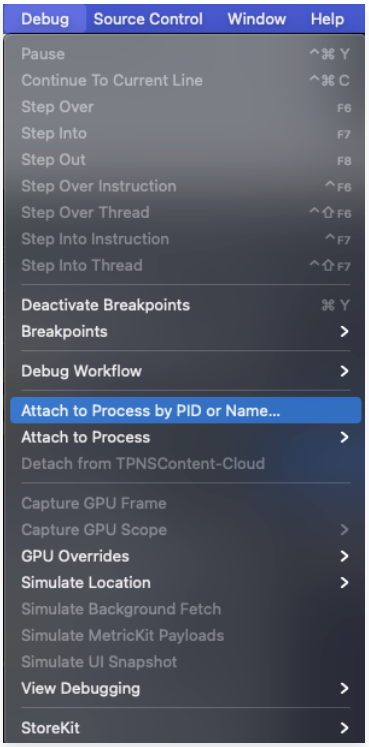
调试方式

若设备收到推送，但无抵达数据，可按照以下步骤进行调试：

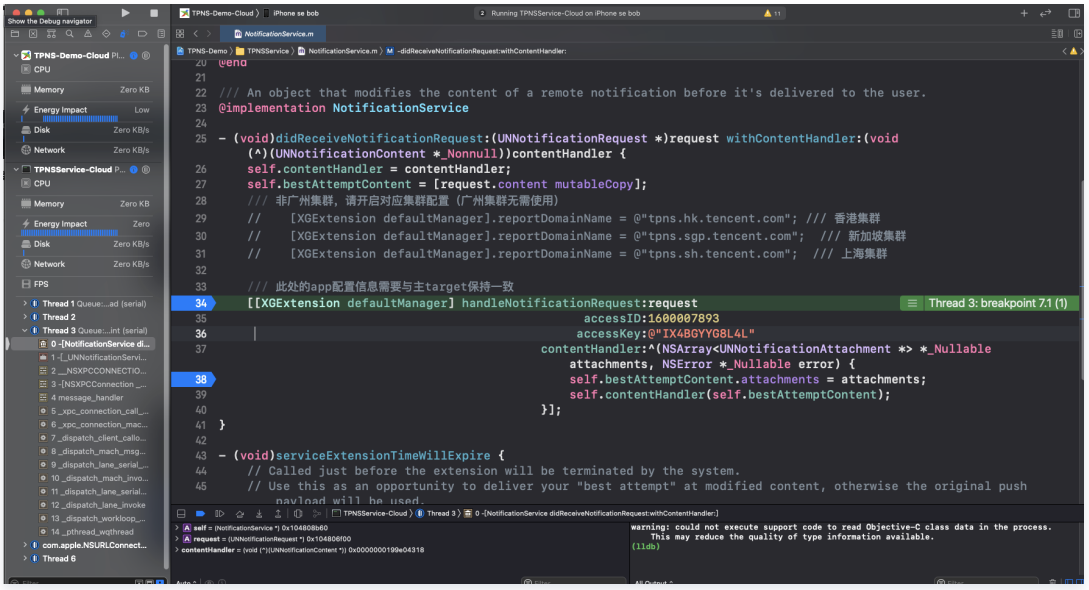
1. 运行主 Target（图中为 Demo 示例）



2. 通过 PID 或者 Name 将 UNNotificationServiceExtension 的实现 Target（Demo 中抵达插件为 TPNSService-Cloud）连接到主 Target。



3. 在如图代码34行、38行的位置加断点，发送一条通知配合调试，注意需要使用 APNs (Apple Push Notification service) 通道，通知的内容中 mutable-content 字段必须为1（移动推送SDK 1.2.8.0开始，后台默认使用 APNs 通道，前台使用移动推送自建通道，调试抵达插件需要将 App 退到后台后）。如果看到断点被执行了，说明调试成功，否则停止所有 Target，重新从第1步开始。



常见问题

为何发送通知后，却没有看到抵达上报?

客户端抵达上报需要集成**通知扩展服务插件**。若集成后，还没有抵达数据上报，则需要排查主工程与抵达插件的 ACCESS ID 和 ACCESS KEY 是否一致。且需要排查 Web 管理台或者 RestAPI 通知的内容中 mutable-content 字段是否为1。
只有主工程与抵达插件的 ACCESS ID 和 ACCESS KEY 一致，且 Web 管理台或者 RestAPI 通知的内容中 mutable-content 字段为1，才会运行客户端抵达插件，从而有抵达数据上报。

客户端集成插件

最近更新时间：2025-03-25 15:34:43

除了原生的 Android SDK 与 iOS SDK 之外，移动推送还提供主流的开发工具集成插件。

官方维护

官方维护的版本放在**腾讯工蜂 > tpns**上以开源的形式发布。如果需要下载打包版本，请单击相应项目页面中的**下载**，下载您需要的插件包。
官方插件地址包含：安装方法、demo（example 文件夹内）以及 API 说明，请开发者参阅。

项目	地址
Unity	官方地址
Flutter	官方地址
React-Native	官方地址
Swift Demo	官方地址
Cordova	官方地址

macOS 接入指南
简介

最近更新时间：2025-03-25 15:34:43

macOS 端实现推送消息的服务涉及三个角色：终端应用（Client App），APNs（Apple Push Notification service），移动推送服务器（移动推送 Provider）。在使用移动推送服务实现给客户端推送消息，需要这三个角色在整个流程中相互配合，任何一个角色出现异常都可能会导致消息无法推送。

文件组成

- XG_SDK_Cloud_macOS.framework (主 SDK 文件)
- XGMTACloud_macOS.framework（“点击上报”组件）

版本说明

- 支持 macOS 10.8+。
- 针对 macOS10.14+ 以上版本。
  - 需要额外引入 UserNotification.framework。
  - 建议使用 Xcode 10.0+。

主要功能

macOS SDK 是移动推送服务为客户端实现消息推送而提供给开发者的接口，主要负责完成：

- 设备 Token 的自动化获取和注册，降低接入门槛。
- 账号、标签与设备的绑定接口，以便开发者实现特定群组的消息推送，丰富推送方式。
- 点击量上报，统计消息被用户点击的次数。

移动推送macOS 与 iOS 的功能差异

> ⚠ 注意
> 以下特性由于苹果官方不支持，因此 macOS SDK 未提供。

功能描述	iOS	macOS	说明
通知扩展插件	✓	×	macOS 不支持通知扩展插件，不支持富媒体通知，不支持离线抵达统计
自定义通知声音	✓	×	macOS 不支持自定义通知声音
静默消息	✓	×	macOS 不支持静默消息
通知分组	✓	×	macOS 不支持通知分组

- 使用 Mac Catalyst 构建的 App 推荐使用移动推送 iOS SDK
- Big Sur 系统（11.3及以下）开发环境无法获取 DeviceToken
  此为苹果官方 Bug，已在11.4修复。

SDK 集成

最近更新时间：2025-02-24 10:41:32

操作场景

本文档提供关于 SDK 接入以及开启推送服务的示例代码。

SDK 组成

- doc 文件夹：移动推送 macOS SDK 开发指南。
- demo 文件夹：主要包含样例工程，移动推送 SDK。

集成步骤

接入前准备

1. 登录 移动推送控制台，单击左侧菜单栏**产品管理**。
2. 进入产品管理页面，单击**新增产品**。
3. 进入新增产品页面，填写产品名称、产品详情，选择产品分类，单击**确定**，即可完成产品新增。
4. 产品创建完成后，选择左侧菜单**配置管理 > 基础配置**，在应用信息一栏中获取应用 `AccessID` 和 `AccessKey` 。

导入 SDK（二选一）

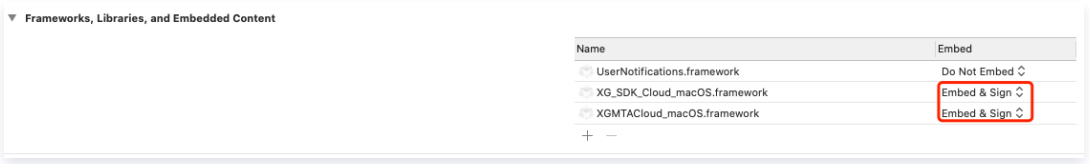
- **方式一：Cocoapods 导入**
  通过 Cocoapods 下载地址：

```
pod 'TPNS-macOS'
```

- **方式二：手动导入**
  进入控制台，单击左侧菜单栏 SDK 下载，进入下载页面，选择 macOS 平台，在其操作栏下单击**下载**即可导入。
  1.1 打开 demo 目录下的 XG-Demo-macOS 文件夹，将 XG_SDK_Cloud_macOS.framework 及 XGMTACloud_macOS.framework 添加到工程。
  1.2 在 Build Phases 下添加以下 Framework：

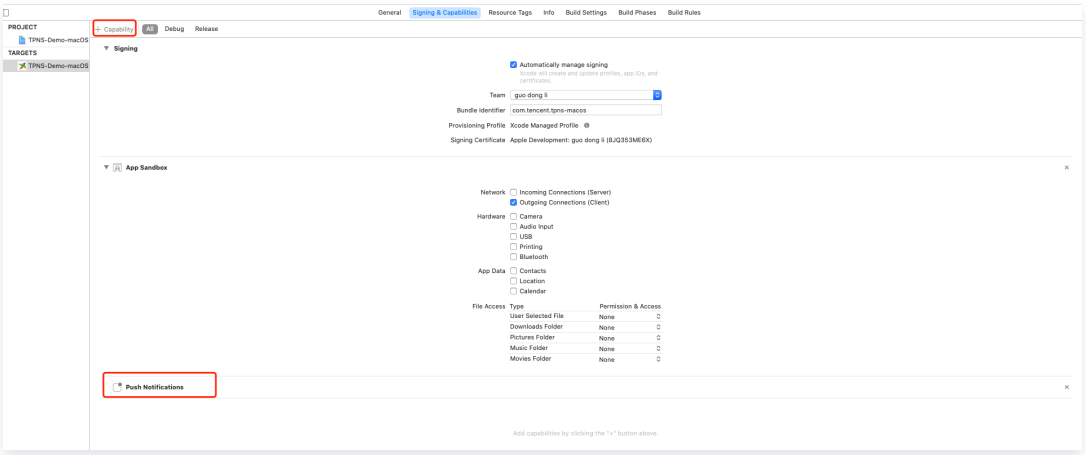
```
* XG_SDK_Cloud_macOS.framework
  * XGMTACloud_macOS.framework
    * UserNotifications.framework(10.14+)
```

  1.3 添加完成以后，`TARGETS->General->Frameworks,Libraries,and Embedded Content` 选项下 Embed 选择 Embed&Sign 如下图：

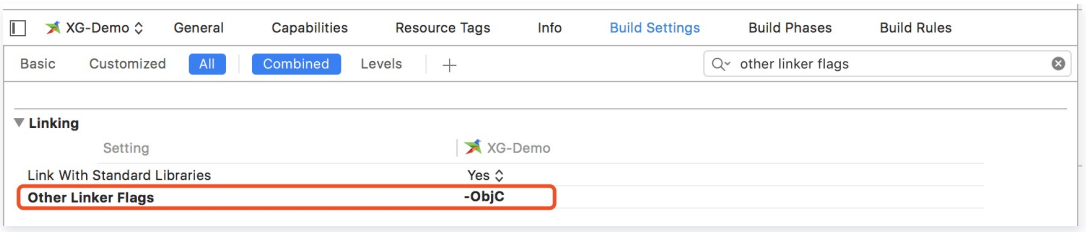


工程配置

1. 在工程配置中打开推送，如下图：



2. 在 `Build Settings->Other Linker Flags` 添加编译参数 `-ObjC` 。



> **注意**
> 如 checkTargetOtherLinkFlagForObjc 报错，是因为 build setting 中，Other link flags 未添加 -ObjC。

接入样例

调用启动移动推送的 API，并根据需要实现 XGPushDelegate 协议中的方法，开启推送服务。

1. 启动移动推送服务，以下是在 AppDelegate 中做演示：

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    /// 打开 Debug 模式，即可在终端查看详细的移动推送 TPNS Debug 信息，方便定位问题。
//    [[XGPush defaultManager] setEnableDebug:YES];
    [XGPush defaultManager].launchOptions = [[aNotification userInfo] mutableCopy];
    [[XGPush defaultManager] startXGWithAccessID:TPNS_ACCESS_ID accessKey:TPNS_ACCESS_KEY
delegate:self];
}
```

2. 在 `AppDelegate` 中选择实现 `XGPushDelegate` 协议中的方法：

```
/// 注册推送服务成功回调
/// @param deviceToken APNs 生成的Device Token
/// @param xgToken TPNS 生成的 Token，推送消息时需要使用此值。TPNS 维护此值与APNs 的 Device Token的映射关系
/// @param error 错误信息，若error为nil则注册推送服务成功
- (void)xgPushDidRegisteredDeviceToken:(NSString *)deviceToken xgToken:(NSString *)xgToken error:
(NSError *)error {
    if (!error) {
        NSLog(@"%s, register success, deviceToken:%@, xgToken:%@", __FUNCTION__, deviceToken, xgToken);
    } else {
        NSLog(@"%s, register failed:%@, deviceToken:%@, xgToken:%@", __FUNCTION__,error.description,
deviceToken, xgToken);
    }
}

/// 统一收到通知消息的回调
```

```
/// @param notification 消息对象
/// @param completionHandler 完成回调
/// 区分消息类型说明：xg字段里的msgtype为1则代表通知消息msgtype为2则代表静默消息
/// notification消息对象说明：有2种类型NSDictionary和UNNotification具体解析参考示例代码
- (void)xgPushDidReceiveRemoteNotification:(id)notification withCompletionHandler:(void (^)
(NSUInteger))completionHandler {
    NSLog(@"[TPNS Demo] receive notification: %@", notification);
}

/// 统一点击回调
/// @param response 如果iOS 10+/macOS 10.14+则为UNNotificationResponse，低于目标版本则为NSDictionary
/// 区分消息类型说明：xg字段里的msgtype为1则代表通知消息,msgtype为9则代表本地通知
- (void)xgPushDidReceiveNotificationResponse:(nonnull id)response withCompletionHandler:(nonnull void
(^)(void))completionHandler {
    if ([response isKindOfClass:[UNNotificationResponse class]]) {
        NSLog(@"[TPNS Demo] click notification: %@", ((UNNotificationResponse
*)response).notification.request.content.userInfo);
    } else if ([response isKindOfClass:[NSDictionary class]]) {
        NSLog(@"[TPNS Demo] click notification: %@", response);
    }
    completionHandler();
}
```

观察日志

如果 Xcode 控制台显示如下相似日志，表明客户端已经成功集成 SDK。

```
[TPNS]  Current device token is 2117b45c7e32bcdae2939f******57e420a376bdd44cf6f58613129d2065370
[TPNS]  Current TPNS token is 0304b8f5d4e*****0af06b37d8b850d95606
[TPNS]  The server responds correctly, registering device successfully
```

集成建议

获取 Token（非必选）

建议您完成 SDK 集成后，在 App 的【关于】、【意见反馈】等比较不常用的 UI 中，通过手势或者其他方式显示 Token，该操作便于我们后续进行问题排查。

示例代码

```
//获取 TPNS 生成的 Token
[[XGPushTokenManager defaultTokenManager] xgTokenString];
//获取 APNs 生成的 DeviceToken
[[XGPushTokenManager defaultTokenManager] deviceTokenString];
```

接口文档

最近更新时间：2025-03-25 15:34:43

启动腾讯移动推送服务

以下为设备注册相关接口方法，若需了解调用时机及调用原理，可查看 设备注册流程 。

接口说明

通过使用在腾讯移动推送官网注册的应用信息，启动腾讯移动推送服务。

```
- (void)startXGWithAccessID:(uint32_t)accessID accessKey:(nonnull NSString *)accessKey delegate:(nullable id<XGPushDelegate>)delegate;
```

参数说明

- accessID：通过前台申请的 AccessID。
- accessKey：通过前台申请的 AccessKey。
- Delegate：回调对象。

> **注意**
> 接口所需参数必须要正确填写，否则腾讯移动推送服务将不能正确为应用推送消息。

示例代码

```
[[XGPush defaultManager] startXGWithAccessID:<your AccessID> accessKey:<your AccessKey> delegate:self];
```

终止腾讯移动推送服务

以下为设备注册相关接口方法，若需了解调用时机及调用原理，可查看 设备反注册流程 。

接口说明

终止腾讯移动推送服务后，将无法通过腾讯移动推送服务向设备推送消息，如再次需要接收腾讯移动推送服务的消息推送，则必须再次调用 `startXGWithAccessID:accessKey:delegate:` 方法重启腾讯移动推送服务。

```
- (void)stopXGNotification;
```

示例代码

```
[[XGPush defaultManager] stopXGNotification];
```

移动推送 Token 及注册结果

查询移动推送Token

接口说明

查询当前应用从腾讯移动推送服务器生成的 Token 字符串。

```
@property (copy, nonatomic, nullable, readonly) NSString *xgTokenString;
```

示例代码

```
NSString *token = [[XGPushTokenManager defaultTokenManager] xgTokenString];
```

注册结果回调

接口说明

SDK 启动之后，通过此方法回调来返回注册结果及 Token。

```
- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nullable NSString *)xgToken error:(nullable NSError *)error
```

返回参数说明

- deviceToken：APNs 生成的 Device Token。
- xgToken：移动推送生成的 Token，推送消息时需要使用此值。移动推送维护此值与 APNs 的 Device Token 的映射关系。
- error：错误信息，若 error 为 nil，则注册推送服务成功。

注册失败回调

接口说明

当注册推送服务失败会走此回调。

```
- (void)xgPushDidFailToRegisterDeviceTokenWithError:(nullable NSError *)error
```

通知授权弹窗的回调

接口说明

通知弹窗授权的结果会走此回调。

```
- (void)xgPushDidRequestNotificationPermission:(bool)isEnable error:(nullable NSError *)error;
```

返回参数说明

- isEnable：是否同意授权。
- error：错误信息，若 error 为 nil，则获取弹窗结果成功。

账号功能

以下为账号相关接口方法，若需了解调用时机及调用原理，可查看 账号相关流程 。

添加账号

接口说明

若原来没有该类型账号，则添加；若原来有，则覆盖。

```
- (void)upsertAccountsByDict:(nonnull NSDictionary<NSNumber *, NSString *> *)accountsDict;
```

> **说明**
> 此接口应该在 xgPushDidRegisteredDeviceToken:error: 返回正确之后被调用。

参数说明

accountsDict：账号字典。

> **说明**
> - 账号类型和账号名称一起作为联合主键。
> - 需要使用字典类型，key 为账号类型，value 为账号，示例：@{@(accountType):@"account"}。

- Objective-C的写法：@{@(0):@"account0",@(1):@"account1"}；Swift的写法：[NSNumber(0):@"account0",NSNumber(1):@"account1"]。
- 更多 accountType 请参照 SDK Demo 包内的 XGPushTokenAccountType 枚举或 账号类型取值表 。

示例代码

```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;
NSString *account = @"account";
[[XGPushTokenManager defaultTokenManager] upsertAccountsByDict:@{ @(accountType):account }];
```

添加手机号

接口说明

添加或更新用户手机号，等于调用 `upsertAccountsByDict:@{@(1002):@"具体手机号"}` 。

```
- (void)upsertPhoneNumber:(nonnull NSString *)phoneNumber;
```

参数说明

- phoneNumber：E.164标准，格式为+[国家或地区码][手机号],例如+8613711112222。SDK内部加密传输。

示例代码

```
[[XGPushTokenManager defaultTokenManager] upsertPhoneNumber:@"13712345678"];;
```

注意

- 此接口应该在xgPushDidRegisteredDeviceToken:error:返回正确之后被调用。
- 如需要删除手机号，调用 `delAccountsByKeys:[[NSSet alloc] initWithObjects:@(1002), nil]`

删除账号

接口说明

接口说明：删除指定账号类型下的所有账号。

```
- (void)delAccountsByKeys:(nonnull NSSet<NSNumber *> *)accountsKeys;
```

说明

此接口应该在 xgPushDidRegisteredDeviceToken:error: 返回正确之后被调用。

参数说明

accountsKeys：账号类型组成的集合。

说明

- 使用集合且 key 是固定要求。
- 更多 accountType 请参照 SDK 包内 XGPush.h 文件中的 XGPushTokenAccountType 枚举。

示例代码

```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;
```

```
NSSet *accountsKeys = [[NSSet alloc] initWithObjects:@(accountType), nil];

[[XGPushTokenManager defaultTokenManager] delAccountsByKeys:accountsKeys];
```

清除账号

接口说明

清除所有设置的账号。

```
- (void)clearAccounts;
```

> **说明**
> 此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码

```
[[XGPushTokenManager defaultTokenManager] clearAccounts];
```

标签功能

以下为标签相关接口方法，若需了解调用时机及调用原理，可查看 标签相关流程 。

绑定/解绑标签

接口说明

开发者可以针对不同的用户绑定标签，然后对该标签进行推送。

```
- (void)appendTags:(nonnull NSArray<NSString *> *)tags
- (void)delTags:(nonnull NSArray<NSString *> *)tags
```

> **说明**
> - 此接口为追加方式。
> - 此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。
> - 单个应用最多可以有10000个自定义 tag， 每个设备 Token 最多可绑定100个自定义 tag，如需提高该限制，请联系 在线客服 ，每个自定义 tag 可绑定的设备 Token 数量无限制。

参数说明

tags：标签数组。

> **说明**
> 标签操作 tags 为标签字符串数组（标签字符串不允许有空格或者是 tab 字符）。

示例代码

```
//绑定标签:
[[XGPushTokenManager defaultTokenManager] appendTags:@[ tagStr ]];

//解绑标签
[[XGPushTokenManager defaultTokenManager] delTags:@[ tagStr ]];
```

更新标签

接口说明

清空已有标签，然后批量添加标签。

```
- (void)clearAndAppendTags:(nonnull NSArray<NSString *> *)tags
```

> **说明**
> - 此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。
> - 此接口会将当前 Token 对应的旧有的标签全部替换为当前的标签。

参数说明

tags：标签数组。

> **说明**
> 标签操作 tags 为标签字符串数组（标签字符串不允许有空格或者是 tab 字符）。

示例代码

```
[[XGPushTokenManager defaultTokenManager] clearAndAppendTags:@[ tagStr ]];
```

清除全部标签

接口说明

清除所有设置的标签。

```
- (void)clearTags
```

> **说明**
> 此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码

```
[[XGPushTokenManager defaultTokenManager] clearTags];
```

查询标签

接口说明

查询设备绑定的标签。

```
- (void)queryTags:(NSUInteger)offset limit:(NSUInteger)limit;
```

> **说明**
> 此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

参数说明

- offset：此次查询的偏移大小。
- offset：limit 此次查询的分页大小, 最大200。

示例代码

```
[[XGPushTokenManager defaultTokenManager] queryTags:0 limit:100];
```

查询标签的回调

接口说明

查询标签的结果会走此回调。

```
- (void)xgPushDidQueryTags:(nullable NSArray<NSString *> *)tags totalCount:(NSUInteger)totalCount error:
(nullable NSError *)error;
```

返回参数说明

- tags：查询条件返回的标签。
- totalCount：设备绑定的总标签数量。
- error：错误信息，若 error 为 nil，则查询成功。

用户属性功能

以下为用户属性相关接口方法，若需了解调用时机及调用原理，可查看 用户属性相关流程 。

新增用户属性

接口说明

添加或更新用户属性（key-value 结构，若原来没有该 key 的用户属性 value，则新增；若原来有该 key 的用户属性 value，则更新该 value）。

```
- (void)upsertAttributes:(nonnull NSDictionary<NSString *,NSString *> *)attributes
```

> **说明**
> 此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

参数说明

attributes：用户属性字符串字典，字符串不允许有空格或者是 tab 字符。

> **说明**
> - 需要先在管理台配置用户属性的键，才能操作成功。
> - key，value 长度都限制50个字符以内。
> - 需要使用字典且 key 是固定要求。
> - Objective-C 的写法：@{@"gender": @"Female", @"age": @"29"}。
> - Swift 的写法：["gender":"Female", "age": "29"]。

示例代码

```
[[XGPushTokenManager defaultTokenManager] upsertAttributes:attributes];
```

删除用户属性

接口说明

删除用户已有的属性。

```
- (void)delAttributes:(nonnull NSSet<NSString *> *)attributeKeys
```

> **说明**

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

参数说明

attributeKeys：用户属性 key 组成的集合，字符串不允许有空格或者是 tab 字符。

> **说明**
> 使用集合且 key 是固定要求。

示例代码

```
[[XGPushTokenManager defaultTokenManager] delAttributes:attributeKeys];
```

清空已有用户属性

接口说明

清空已有用户属性。

```
- (void)clearAttributes;
```

> **说明**
> 此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码

```
[[XGPushTokenManager defaultTokenManager] clearAttributes];
```

更新用户属性

接口说明

清空已有用户属性，然后批量添加用户属性。

```
- (void)clearAndAppendAttributes:(nonnull NSDictionary<NSString *,NSString *> *)attributes
```

> **说明**
> 此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码

```
[[XGPushTokenManager defaultTokenManager] clearAndAppendAttributes:attributes];
```

角标功能

同步角标

接口说明

当应用本地角标值更改后，需调用此接口将角标值同步到移动推送服务器，下次推送时以此值为基准，此功能在管理台位置（【新建推送】>【高级设置】>【角标数字】）。

```
- (void)setBadge:(NSInteger)badgeNumber;
```

参数说明

badgeNumber：应用的角标数。

> **注意**
> 当本地应用角标设置后需调用此接口同步角标值到移动推送服务器，并在下次推送时生效，此接口必须在移动推送注册成功后调用（xgPushDidRegisteredDeviceToken）。

示例代码

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {
    /// 每次启动 App 应用角标清零（本地应用角标设置需要在主线程执行）
    if ([XGPush defaultManager].xgApplicationBadgeNumber > 0) {
        [XGPush defaultManager].xgApplicationBadgeNumber = 0;
    }
    return YES;
}

- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nullable NSString
*)xgToken error:(nullable NSError *)error {
    /// 在注册完成后同步角标数到 TPNS
    if (!error) {
        [[XGPush defaultManager] setBadge:0];
    }
}
```

查询设备通知权限

接口说明

查询设备通知权限是否被用户允许。

```
- (void)deviceNotificationIsAllowed:(nonnull void (^)(BOOL isAllowed))handler;
```

参数说明

handler：查询结果的返回方法。

示例代码

```
[[XGPush defaultManager] deviceNotificationIsAllowed:^(BOOL isAllowed) {
        <#code#>
    }];
```

查询 SDK 版本

接口说明

查询当前 SDK 的版本。

```
- (nonnull NSString *)sdkVersion;
```

示例代码

```
[[XGPush defaultManager] sdkVersion];
```

日志上报接口

接口说明

开发者如果发现推送相关功能异常，可以调用该接口，触发本地 push 日志的上报，通过联系 在线客服 反馈问题时，请将文件地址提供给我们，便于排查问题。

```
- (void)uploadLogCompletionHandler:(nullable void(^)(BOOL result,  NSString * _Nullable
errorMessage))handler;
```

参数说明

- @brief：上报日志信息 。
- @param handler：上报回调。

示例代码

```
[[XGPush defaultManager] uploadLogCompletionHandler:nil];
```

移动推送日志托管

接口说明

可以在此方法获取移动推送的 log 日志。此方法和 XGPush > enableDebug 无关。

参数说明

logInfo：日志信息。

示例代码

```
- (void)xgPushLog:(nullable NSString *)logInfo;
```

本地推送

本地推送相关功能请参见 苹果开发者文档 。

推送证书说明

最近更新时间：2025-03-25 15:34:43

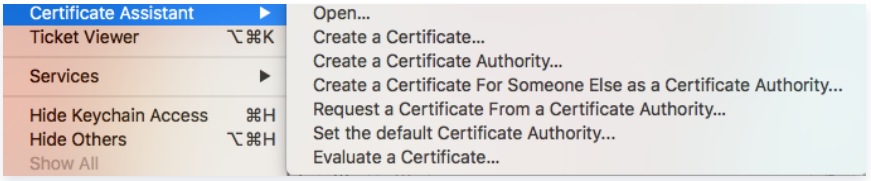
操作场景

macOS 推送证书分为开发环境的推送证书、发布环境的推送证书。本文档指导您如何制作开发环境的推送证书，发布环境操作步骤一致，在第5步选择 production 即可。

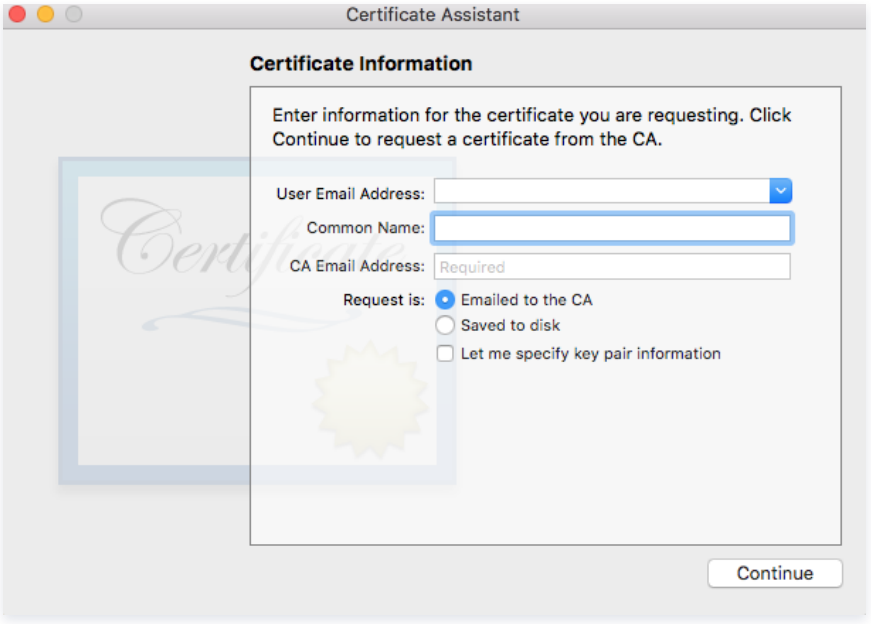
操作步骤

生成证书

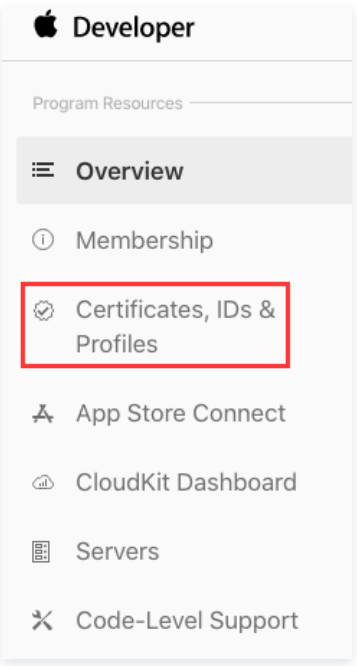
1. 在您的电脑中，打开 Keychain Access 工具，选择 **Request a Certificate From a Certificate Authority**。



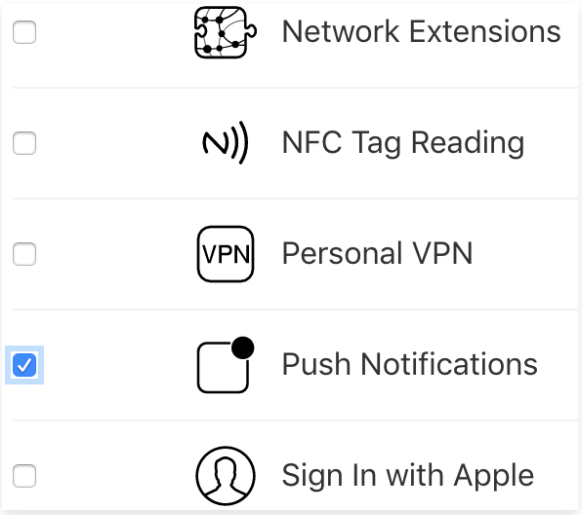
2. 填写邮件地址，其它留空，单击 **continue**，将证书保存到本地。



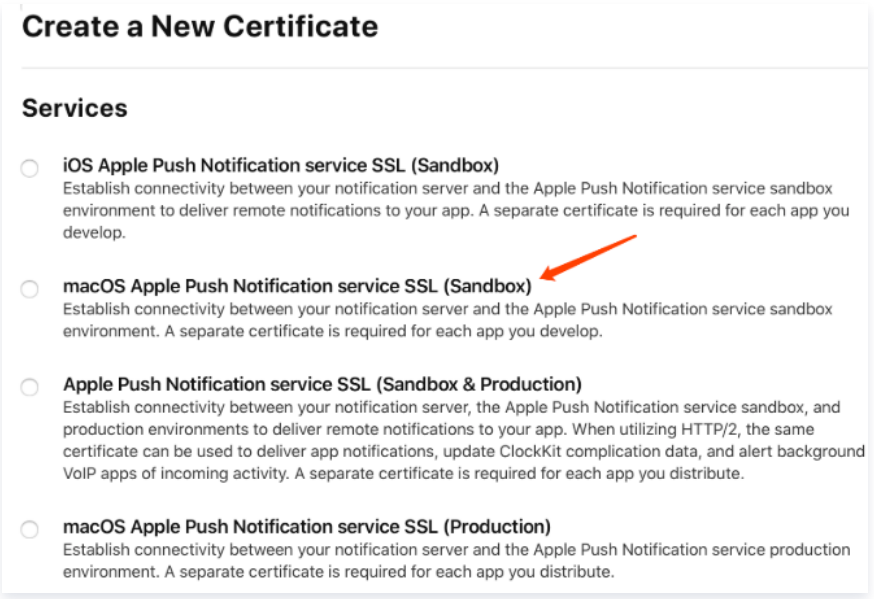
3. 登录苹果开发者中心网站，单击 **Certificates,Identifiers & Profiles**。



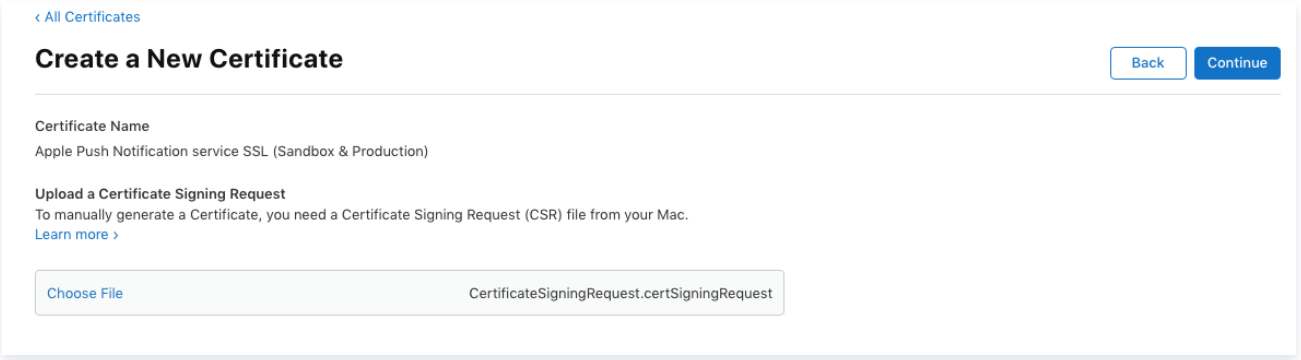
4. 选中需要制作消息推送证书的应用，勾选消息推送服务 **Push Notifications**。



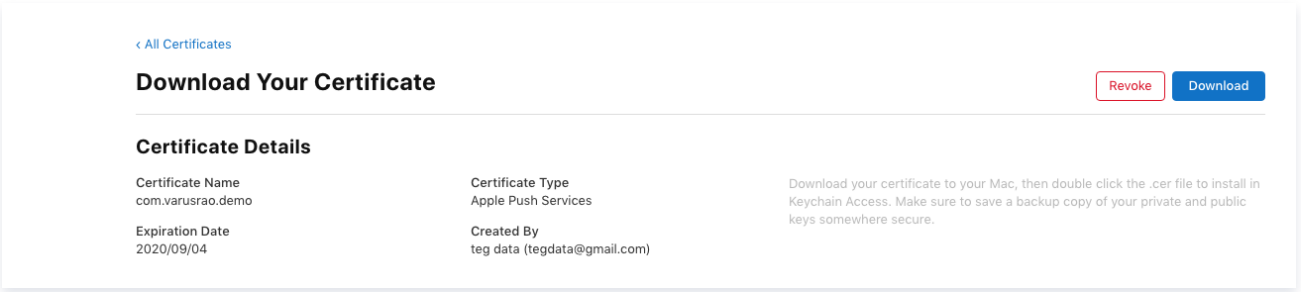
5. 勾选 **Create Certificate**，选择开发环境的推送证书。



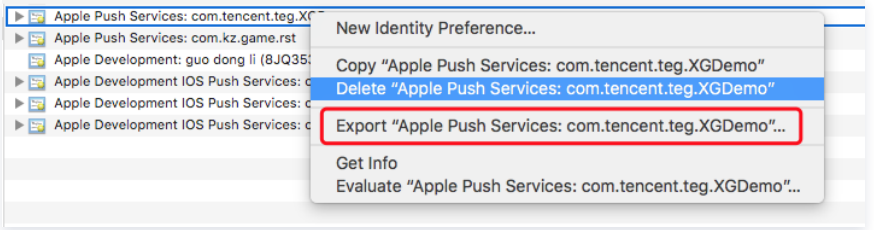
6. 选择第2步中创建的消息推送证书请求文件，上传完毕之后，单击 **continue**。



7. 将生成的消息推送证书下载到本地。



8. 双击上一步中下载的证书，将自动将消息推送证书安装到 Keychain 应用中。
9. 打开 Keychain Access， 选中需要导出的消息推送证书，右键选择导出证书，导出的格式为 P12，并设置密码。



上传证书

1. 登录 移动推送控制台，选择左侧菜单**基础配置**。
2. 进入**基础配置**页面，选择需要上传推送证书的应用。

3. 输入证书密码，单击**上传证书**，选择您的证书即可完成上传。