蚂蚁金服金融科技产品手册

消息推送

产品版本： V20200101

文档版本： V20200101

蚂蚁金服金融科技文档

商标声明

及其他蚂蚁金服金融科技服务相关的商标均为蚂蚁金服金融科技所有。
本文档涉及的第三方的注册商标，依法由权利人所有。

免责声明

由于产品版本升级、调整或其他原因，本文档内容有可能变更。蚂蚁金服金融科技保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在蚂蚁金服金融科技授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过蚂蚁金服金融科技授权渠道下载、获取最新版的用户文档。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

目录

1 消息推送简介 1
2 推送流程 3
3 基础术语 10
4 接入 Android——基于 mPaaS 框架 10
4.1 添加 SDK 10
4.2 使用 SDK（版本 ≥ 10.1.32） 11
4.2.1 步骤概览 11
4.2.2 初始化 pushSDK 11
4.2.3 监听消息 12
4.2.4 接入第三方推送渠道 15
4.2.5 上报用户 ID 19
4.2.6 上报推送数据 19
4.3 使用 SDK（版本 < 10.1.32） 22
4.3.1 步骤概览 22
4.3.2 初始化 pushSDK 22
4.3.3 监听消息 22
4.3.4 接入第三方推送渠道 26
4.3.5 上报用户 ID 31
4.4 推送消息 31
4.5 常见问题 32
5 接入 Android——基于原生工程 33
5.1 Demo 33
5.2 接入 SDK 33
5.3 使用 SDK 34
5.4 接入华为推送渠道 38
5.5 接入小米推送渠道 39
5.6 使用 ProGuard 39
6 接入 iOS 46
7 配置服务端 53
8 管理控制台 54
8.1 消息列表 54
8.1.1 创建消息 54
8.1.2 管理消息 63
8.2 消息模板 65
8.2.1 创建模板 65
8.2.2 管理模板 66
8.3 渠道配置 67
8.4 推送配置 69
8.5 密钥管理 71
8.6 使用分析 73
9 参考 76
9.1 API 参考 76
9.2 制作 iOS 推送证书 87

1 消息推送简介

消息推送组件提供专业的移动消息推送方案，针对不同的场景推出多种推送类型，满足您的个性化推送需求。为了提升推送的到达率，mPaaS 集成了华为、小米等厂商的推送功能，在提供控制台快速推送能力的同时，也提供了服务端接入方案，方便用户快速集成移动终端推送功能，与 App 用户保持互动，从而有效地提高用户留存率，提升用户体验。

功能特性

您可通过 MPS 发起多种类型的消息推送，推送渠道支持自建渠道和三方渠道，推送方式支持控制台页面推送和 API 推送，基于实际业务场景，选择合适的推送类型、推送渠道以及推送方式。

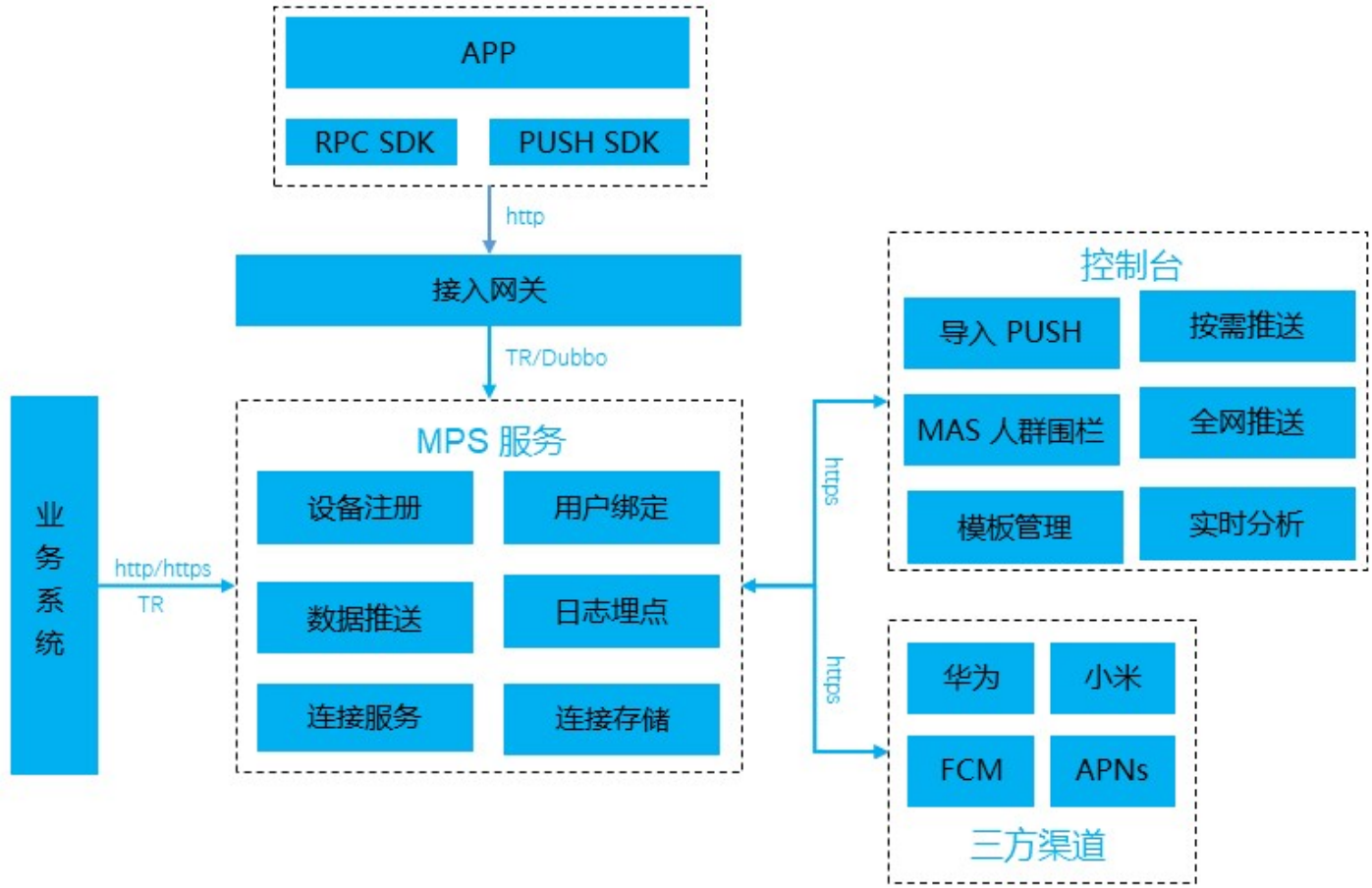
MPS 核心功能如下：

- **多种推送方式**：可以精准推送消息给自定义目标用户群体、单个用户、全部用户等多种方式，并可以从移动推送服务控制台页面发送消息，也可以利用 API 接口发送消息。
- **自定义消息有效期**：若初次下发消息时设备未在线，那么在消息有效期内，设备建链或者发起用户绑定均可触发消息再次下行，确保消息最终送达目标用户。
- **不同推送目标类型**：您可以建立设备与登录用户的对应关系，基于设备标识或用户标识推送消息。
- **个性化消息模板**：通过模板管理页面，您可以配置个性化模版，满足业务的个性化推送需求。
- **推送配置**：通过推送配置页面，配置证书，您可以选择 iOS 设备推送所对应的 APNs 网关。
- **渠道配置**：接入第三方推送渠道，集成华为、小米等第三方渠道推送功能，提升推送到达率。
- **密钥管理**：消息推送的所有对外接口都需要对请求进行签名，保证了业务的安全性，提供了密钥配置页面供用户配置自己的密钥。同时，提供消息回执功能，供您追踪消息的投递结果。
- **使用分析**：基于客户端埋点上报数据，在平台、版本、推送渠道、推送类型、时间等维度上，对推送数据进行统计分析，生成分析报表，可展示分钟级别的统计结果。

原理框架

MPS 推送服务为 mPaaS 体系内直接与客户端通讯的核心必备基础组件之一，其基础原理为基于 TCP 长连接通道或者手机厂商推送渠道进行 **消息通知** 相关业务数据传输。

客户端通过 mPaaS 移动网关（MGS）配合服务，调用 RPC 网关进行设备注册、用户绑定以及第三方渠道的关系绑定，实现基于设备维护和用户维度的消息推送。按照既定规范采集和上传客户端行为日志埋点，后端实时统计分析推送数据，生成统计报表。MPS 同时支持 API 推送与控制台页面推送，您可以在自己的服务端根据业务逻辑通过 API 调用推送个性化消息，也可以通过控制台页面直接推送消息。为了提升消息到达率，MPS 支持接入华为、小米、FCM 和 APNs 等推送渠道，并对后端业务系统保持透明，可让业务系统专注于完成业务功能，无需关注终端机型。



组件优势

作为消息推送的业务人员，您可以通过消息推送组件获取以下优势：

- **快速稳定**：消息下发速度快，保证稳定到达。
- **接入简单**：降低接入成本，更高效。
- **量化推送效果**：集成推送数据统计，更智能地分析消息送达率，打开率，明确推送效果。
- **精准个性化推送**：
  - 可以向单个用户、自定义用户分组等各种维度精准推送个性化信息。
  - 提供控制台推送页面推送，满足简单的推送需求。同时，也提供服务端接入方案，满足更为复杂的需求。
  - 提供消息回执，供您追踪消息下发结果，有效提升用户留存率跟活跃度。
  - 建立设备标识与 App 用户体系的对应关系，可把 App 用户名作为消息接收者直接发送消息，无论用户在哪台设备登录信息都能准确送达。

应用场景

针对不同应用场景，消息推送提供以下几种推送方式：

- **极简推送（Simple Push）**：针对单个用户或设备快速推送消息，配置简单。
- **模板推送（Template Push）**：针对单个用户或设备推送消息，可指定消息模版，消息正文由替换模

板占位符得到。

- **批量推送（Multiple Push）**：针对大量设备或者用户推送消息，可指定消息模版，在配置文件中针对不同设备或用户设置不同的占位符变量值。
- **群发推送（Broadcast Push）**：针对全网设备进行推送，可指定消息模版，消息正文由替换模板占位符得到。

2 推送流程

了解不同接入方式下的推送处理流程。

基本概念

设备标识 (token) 消息推送组件为每个客户端设备分配一个唯一标识，并根据该标识来确定消息推送的目标：

- Android 设备使用自建长连接进行消息推送。
- iOS 设备使用苹果提供的 APNs 服务进行消息推送。

推送模式
消息推送组件提供以下推送模式：

- **指定设备标识的推送**
- **指定用户标识的推送**
- **不指定任何标识的群发**

说明：无论采用哪种模式，最终在系统内部都会映射成设备标识。指定用户标识的推送是移动推送服务为方便与用户的业务系统对接而提供的推送方式。由于最终要映射成设备标识，需要应用开发者对用户标识和设备标识进行绑定。推荐在用户登录时，进行绑定，在用户登出时，进行解绑。

第三方推送
第三方推送是指厂商自己的推送，能够保证高到达率。在调用 push 的 init 进行初始化过程中，会分别向 mPaaS 和第三方平台申请设备标识，在回调中分别返回 mPaaS 的设备标识和第三方的设备标识信息。

若要使用第三方推送，需要等待以上两个设备标识返回后，再调用 report 接口将两个设备标识上传到移动推送核心，此时会将两者关联起来，完成上述操作后才能真正使用第三方的设备标识，否则就是普通的 mPaaS 推送。

处理流程

消息推送服务由两个后端系统组成：

- **移动推送核心（Pushcore）**：负责处理业务逻辑以及向开发者提供 API 接口。
- **移动推送网关（Mcometgw）**：负责保持与 Android 设备的长连接。

重要：在请求设备标识 (token) 部分，若是小米、华为或其他已经接入厂商推送平台的手机，还会向第三方平台请求设备标识，需要等待两个设备标识返回，通过调用 report 接口将两者绑定，才能使用厂商的推送通道。普通手机只需要使用 mPaaS 返回的设备标识。

了解不同设备平台对应的消息推送接入流程：

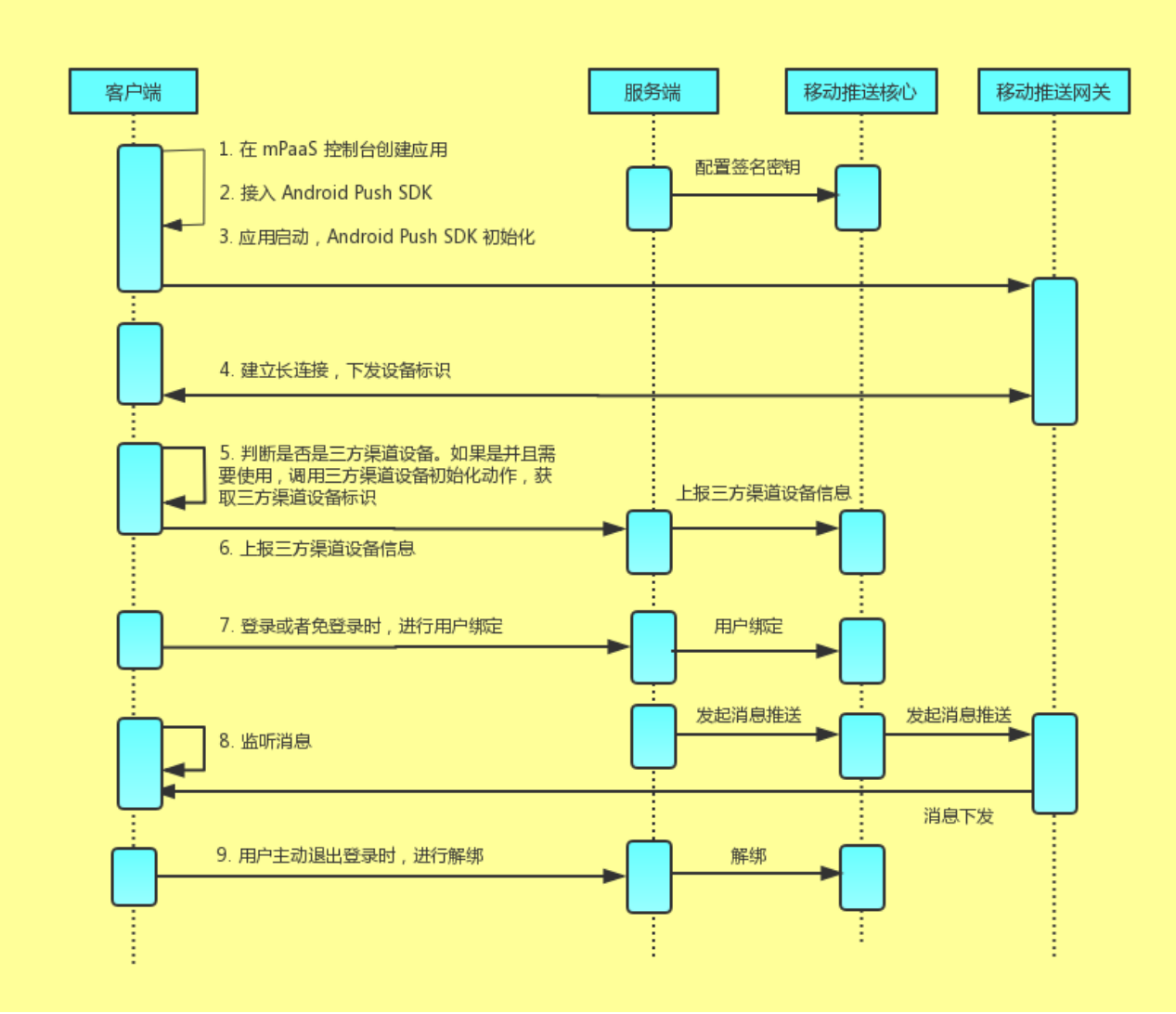
- 中国大陆安卓设备
- 苹果及国外安卓设备

中国大陆安卓设备

对于中国大陆的安卓系统设备，接入方式按照协议区分主要分为 HTTP 接入方式和 RPC 接入方式。

HTTP 接入方式

HTTP 接入方式是针对客户端不使用 RPC SDK 的情况下的接入方案，客户端不直接与移动推送核心进行交互，而是直接调用客户自己的服务端，由客户服务端通过 HTTP/HTTPS 的方式调用推送核心接口进行交互。针对中国大陆的安卓设备，移动推送服务提供了自建网关。整个流程如下图所示：



其中：

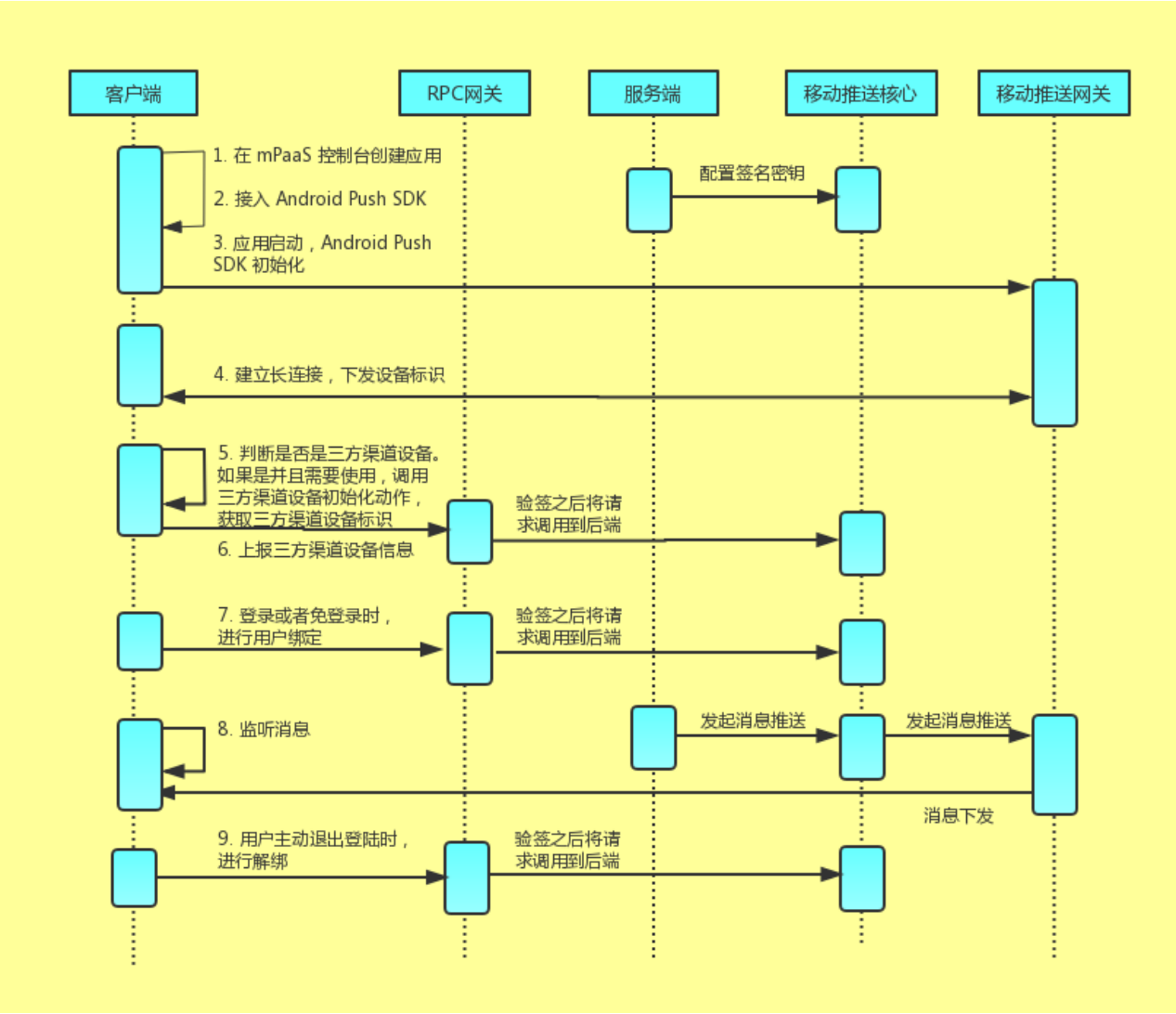
- 应用启动时，客户端同移动推送网关建立长链接，如果客户端建链信息中未携带设备标识，移动推送网关将下发设备标识。
- 如果用户开启小米、华为等三方渠道，且客户端属于这些三方渠道的机型，那么这些三方渠道的 SDK

会进行初始化动作，与对应厂商的推送网关建立长连接服务并获取三方渠道设备标识。

- 获取三方渠道设备标识后，通过用户方服务端调用设备上报 HTTP 接口，上报三方渠道设备信息至移动推送核心。
- 应用用户在客户端上发起登录。
- 用户服务端收到用户登录请求，用户登录成功，可以选择在此时向移动推送核心发起用户和设备绑定。
- 服务端发起推送请求。
- 移动推送核心获取到推送请求，移动推送核心根据推送类型进行区分：
    - 若按设备推送，直接调用移动推送网关下发消息。
    - 若按用户推送，根据请求中的用户标识获取与之绑定的设备标识，然后调用移动推送网关下发消息。
- 移动推送网关下发消息。
- 消息下发成功后，客户端会向移动推送网关确认已收到消息。如果租户配置了回调接口，移动推送核心会给服务端回执。
- 客户端在用户主动退出登录时，通过用户方服务端调用解绑 HTTP 接口。

RPC 接入方式

RPC 接入方式是针对客户端使用 RPC SDK 的情况下的接入方案，客户端使用 RPC 经由 RPC 网关直接与移动推送核心进行交互。针对国内安卓设备，移动推送服务提供了自建网关。整个流程如下图所示：



其中：

- 应用启动时，客户端同移动推送网关建立长链接，如果客户端建链信息中未携带设备标识，移动推送网关将下发设备标识。
- 如果用户开启小米、华为等等三方渠道，且客户端属于这些三方渠道的机型，那么这些三方渠道的SDK会进行初始化动作，与对应厂商的推送网关建立长连接服务并获取三方渠道设备标识。
- 获取三方渠道设备标识后，客户端调用设备上报 RPC 接口，上报三方渠道设备信息。
- 应用用户在客户端上发起登录。
- 服务端收到用户登录请求，用户登录成功，可以选择在此时向移动推送核心发起用户和设备绑定请求。
- 服务端发起推送请求。
- 移动推送核心获取到推送请求，移动推送核心根据推送类型进行区分：
    - 若按设备推送，则直接调用移动推送网关下发消息。
    - 若按按用户推送，则根据请求中的用户标识获取与之绑定的设备标识，然后调用移动推送网关下发消息。

- 移动推送网关下发消息。
- 消息下发成功后，客户端会向移动推送网关确认已收到消息，如果租户配置了回调接口，移动推送核心会给服务端回执。
- 客户端在用户主动退出登陆时调用解绑 RPC 接口。

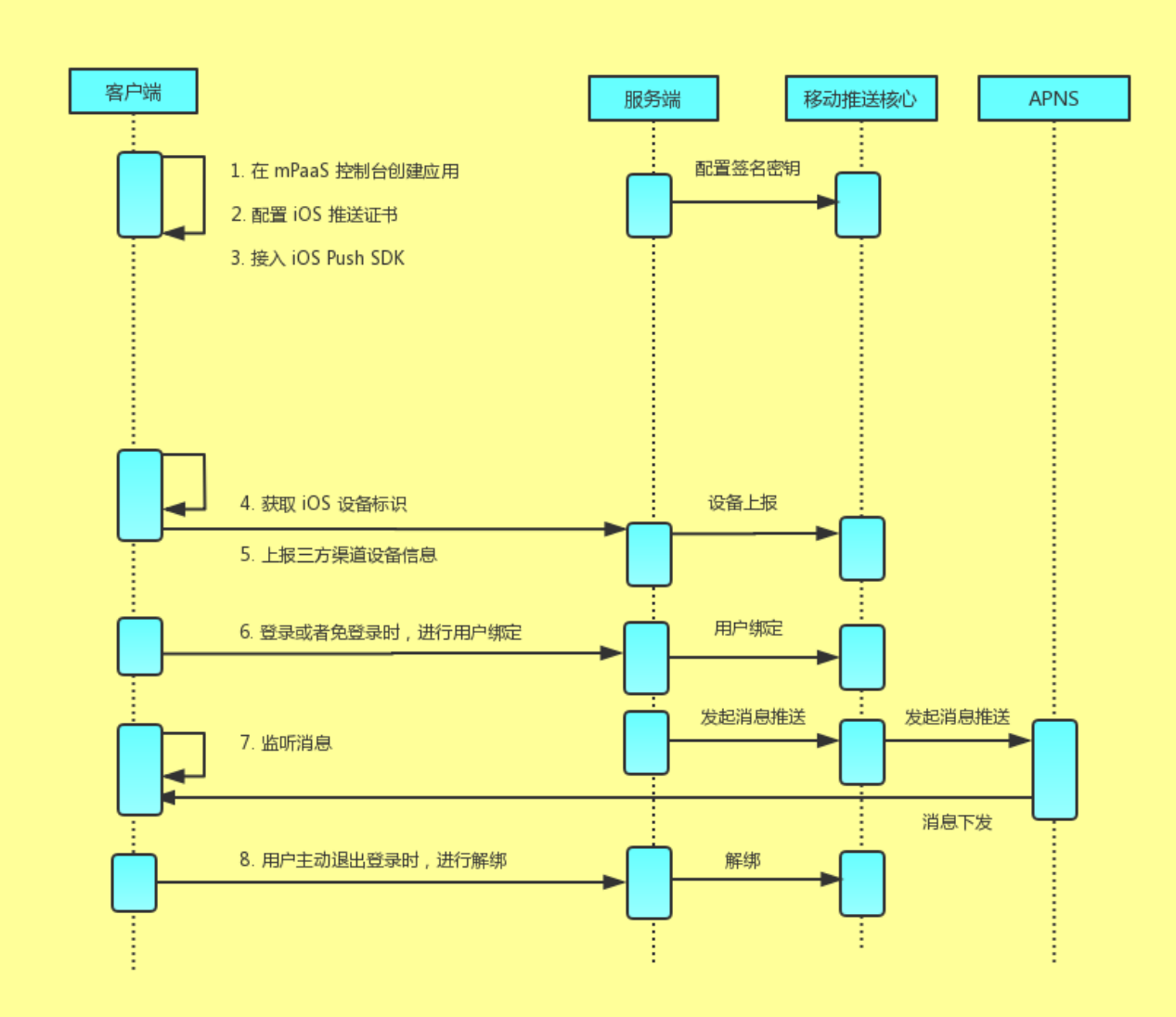
苹果及国外安卓设备

国外安卓的推送网关采用谷歌的 GCM/FCM 服务，苹果的推送网关采用苹果的 APNs 服务，此处以苹果设备为例。

接入方式按照协议区分主要分为 HTTP 接入方式和 RPC 接入方式。

HTTP 接入方式

HTTP 接入方式是针对客户端不使用 RPC SDK 的情况下的接入方案，客户端不直接与移动推送核心进行交互，而是直接调用客户自己的服务端，由客户服务端通过 HTTP/HTTPS 的方式调用推送核心接口进行交互。整个流程如下图所示：



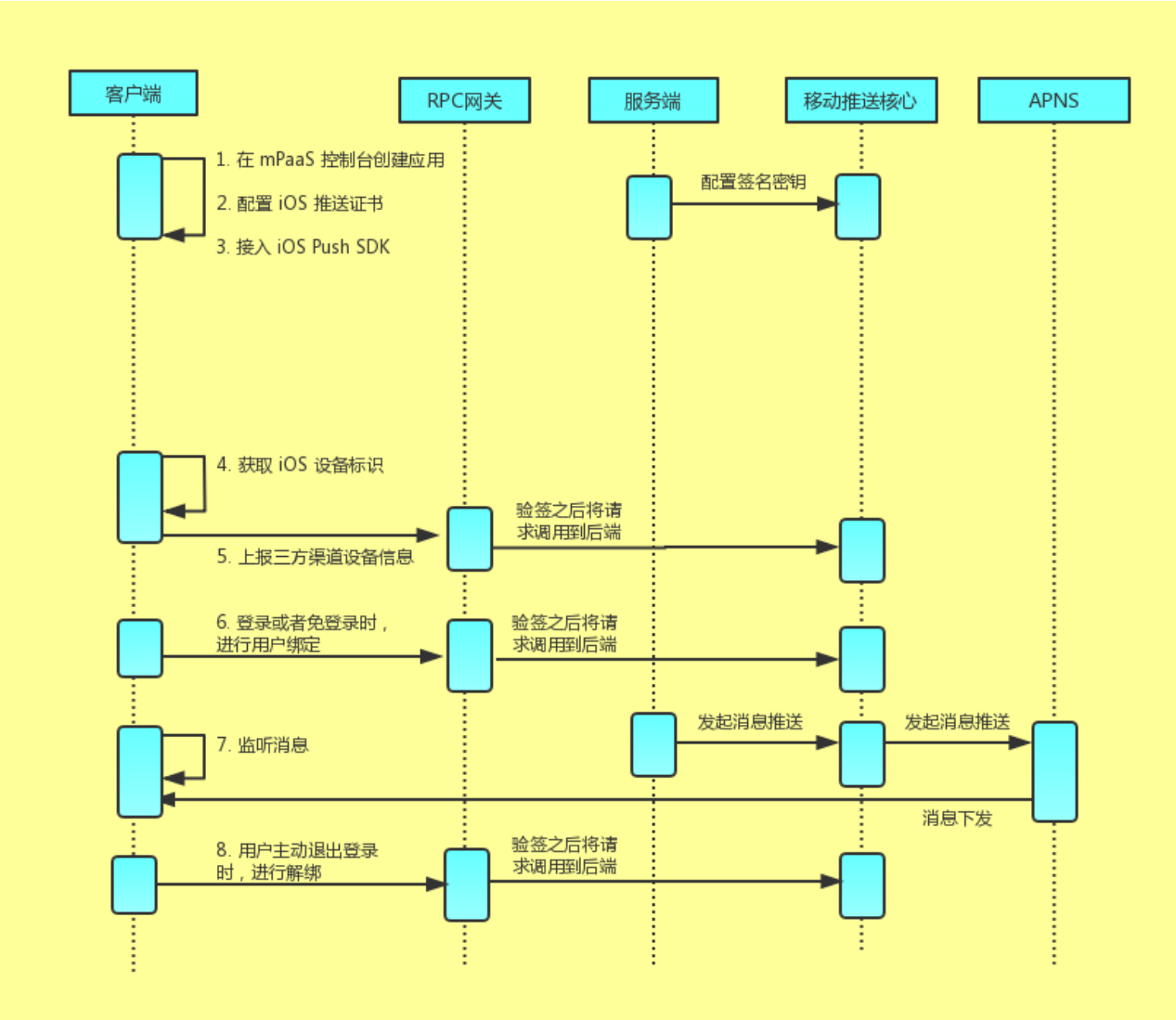
其中：

- 客户端获取苹果下发的设备标识。

- 客户端调用服务端，向服务端上报设备信息。
- 用户方服务端调用设备上报 HTTP 接口，上报设备信息至移动推送核心。
- 应用用户在客户端上发起登录。
- 服务端收到用户登录请求，用户登录成功，可以选择在此时向移动推送服器发起用户和设备绑定请求。
- 服务端发起推送请求。
- 移动推送核心获取到推送请求，并根据推送类型进行区分：
    - 若按设备推送，则直接调用 APNs 服务下发消息。
    - 若按用户推送，则根据请求中的用户标识获取与之绑定的设备标识，然后调用 APNs 服务下发消息。
- 消息下发成功后，客户端会向移动推送核心确认已收到消息，如果租户配置了回调接口，移动推送核心会给服务端回执。

RPC 接入方式

RPC 接入方式是针对客户端使用 RPC SDK的情况下的接入方案，客户端使用 RPC 经由 RPC 网关直接与移动推送核心进行交互。整个流程如下图所示：



其中：

- 客户端获取苹果下发的设备标识。
- 客户端调用上报设备 RPC 接口经由 RPC 网关向移动推送核心上报设备信息。
- 应用用户在客户端上发起登录。
- 用户登录成功后，可以选择在此时调用绑定 RPC 接口经由 RPC 网关向移动推送服发起用户和设备绑定请求。
- 服务端向移动推送核心发起推送请求。
- 移动推送核心获取到推送请求，并根据推送类型进行区分：
    - 若按设备推送，则直接调用 APNs 服务下发消息。
    - 若按用户推送，则根据请求中的用户标识获取与之绑定设备标识，然后调用 APNs 服务下发消息。
- 消息下发成功后，客户端会向移动推送核心确认已收到消息，如果租户配置了回调接口，移动推送核心会给服务端回执。

3 基础术语

中文	英文	解释
应用 ID	AppID	应用标识，在创建应用时生成
任务名称	Task Name	一次消息推送请求标识为一次任务
安卓设备标识	Ad-token	特指安卓设备的唯一标识，主要见于客户端 SDK 中
苹果设备标识	Device Token	特指苹果设备的唯一标识，由苹果系统提供
用户标识	userId、usrId	标识某个用户，与设备有对应关系，一般用于绑定关系
绑定关系	Bind-info	指设备与用户标识的映射关系，对应 **绑定** 和 **解绑** 两个操作
推送目标 ID	Target ID、Token	指要推送的目标，可能是 Android 的 Ad-token、iOS 的 Device Token、用户标识（userId），需要联系上下文的判断是哪种类型
业务方消息标识	Msgkey	由系统自动生成，用于在业务方系统中唯一标识消息
消息标识	pushMsgId	由系统自动生成，为 MPS 对消息的唯一标识，用于唯一标识一条消息
消息模板	Template	生成消息的框架，包含消息的属性配置，以及确定的消息内容和可被动态替换的占位参数
模板参数、模板占位符	Template Placeholder	指消息模板中可被动态替换的部分
模板参数值	Templatekv	指替换模板占位符的具体内容
极简推送	SimplePush	针对单个推送目标 ID，推送一条消息的方式
模板推送	TemplatePush	针对单个推送目标 ID，推送一条消息的方式，消息的内容是由模板进行参数替换得到的
批量推送	MultiplePush	针对大量推送目标 ID，推送个性化消息的方式，消息的内容是由同一个模板根据不同的推送目标 ID，使用各自的参数替换值得到的
群发推送	BroadcastPush	针对全网设备，推送相同消息的方式，消息的内容是由模板进行参数替换得到的
推送证书	Push Cert	特指苹果平台下，用于与苹果 APNs 服务器建立连接

4 接入 Android——基于 mPaaS 框架

4.1 添加 SDK

本文将引导您添加 SDK。

前置条件

- 您已参考 通用步骤说明 完成基础配置。
- 如果您已经接入过 pushSDK，请检查一下 Portal 工程是否引入了 pushservice 依赖。如果已经引入，请去掉该依赖，然后再接入。

操作步骤

1. 生成加密图片并添加到项目中。更多信息，请参考 加密图片 。

添加 SDK 依赖：

使用 mPaaS 插件，分别在 Portal 和 Bundle 工程中添加 **PUSH** 和 **RPC** 模块依赖。更多信息，请参考 管理组件依赖 > 增删组件依赖 。

后续步骤

初始化 pushSDK，建立客户端和移动推送网关的长链接，获取设备标识。

相关链接

常见问题

4.2 使用 SDK（版本 ≥ 10.1.32）

4.2.1 步骤概览

在 Android 客户端中使用 pushSDK，您需要进行如下操作：

1. 初始化 pushSDK ：建立客户端和移动推送网关的长链接，获取设备标识。
2. 监听消息 ：为接收消息、获取 Android 设备标识（Ad-token）编写 service，实现设备维度的消息推送。
3. 接入第三方推送渠道 ：可选，提升消息推送的到达率。
4. 上报用户 ID ：将用户标识上传到服务端，进行用户和设备绑定，实现用户维度的消息推送。
5. 上报推送数据 ：将消息到达、打开和忽略的数据上传至服务端进行统计分析，您可在控制台页面上查看统计分析结果。

相关操作

推送消息 ：按设备维度或用户维度推送消息。

相关链接

- 示例代码
- 常见问题

4.2.2 初始化 pushSDK

初始化 pushSDK 过程中，客户端和移动推送网关建立长链接，并获取设备标识。

关于此任务

客户端通过调用 mPaaS 中间层的 MPPush 类来实现对 pushSDK 的初始化：

```
MPPush.init(Context context)
```

初始化过程中， pushSDK 会用设定好的地址和端口尝试建立链接。如果客户端建链信息中未携带设备标识，移动推送网关在建链成功后将下发设备标识（Ad-token）。

在 pushSDK 成功获取到设备标识后，您可以通过 App 中配置好的 service 来获取这个设备标识。具体方法，参考 监听消息 中的示例。

相关链接

- 示例代码
- 常见问题

4.2.3 监听消息

在接收到推送消息，或者获取到 Ad-token 后，pushSDK 都会启动一个 service 来对该消息或该 Ad-token 进行处理。为接收消息、获取 Android 设备标识（Ad-token）编写 service，实现设备维度的消息推送。

关于此任务

通过编写和配置继承自 AliPushRcvService 的 service，用来接收移动推送网关下发的设备标识和消息。

操作步骤

编写 service。客户端可以分别为接收消息、获取 Ad-token 各编写一个 service，也可以共用同一个 service。下面是共用同一个 service 的示例代码：

```
public class RecvMsgIntentService extends AliPushRcvService {
public RecvMsgIntentService() {
super();
}

// 调用时机：
// 华为、小米手机：接入推送后，点击通知栏时进行调用
// 其他手机：收到消息时进行调用
@Override
protected void handleActionReceived(String s, String s1, boolean clicked) {
//示例代码
Intent intent1 = new Intent("tt-action");
String extra ="key:="+ s +"data:="+ s1;
intent1.putExtra("intent :", extra);
LocalBroadcastManager.getInstance(this).sendBroadcast(intent1);
Log.v("push-test","onHandleIntent sendLocalBroadcast :" + intent1.toString());
if (TextUtils.isEmpty(s1)) {
return;
}
if(clicked){
//华为小米采用通知栏消息，所以一定点击过
}
BDataBean bDataBean = BDataBean.create(s1);
// BDataBean 含有自定义的Param
if (bDataBean == null) {
return;
```

```
}
// 处理默认的跳转等事件
// 可以不调用以下方法，自行处理
bDataBean.action(this);
}

public static String adToken ="";
public static String thirdToken ="";
public static int platform = 0;

// 自建渠道 AdToken 返回
/** 调用时机：完成初始化后，收到自建渠道标识时调用
*@param s adtoken 自建渠道标识
*/
@Override
protected void handleActionId(String s) {
// 示例代码
String registrationId = s;
PushAppInfo pushAppInfo = new PushAppInfo(getApplicationContext());
pushAppInfo.setAppToken(registrationId);
Log.e("push-test","registrationId =" + registrationId);
Intent intent1 = new Intent("tt-action");
String extra ="adToken:="+ s;
intent1.putExtra("intent :", extra);
LocalBroadcastManager.getInstance(this).sendBroadcast(intent1);
Log.e("push-test","onHandleIntent sendLocalBroadcast :" + intent1.toString());
adToken = s;

}
/**
*调用时机：收到第三方渠道标识时调用
*@param s thirdToken 第三方渠道标识
*@param i platform 第三方渠道类型
*/
@Override
protected void handleActionThirdId(String s, int i) {
// 示例代码
Intent intent1 = new Intent("tt-action");
String extra ="third token:="+ s +"channel"+ i;
intent1.putExtra("intent :", extra);
LocalBroadcastManager.getInstance(this).sendBroadcast(intent1);
Log.e("push-test","onHandleIntent sendLocalBroadcast :" + intent1.toString());
thirdToken = s;
platform = i;
}
}
```

在 AndroidManifest.xml 配置此 service。

说明：基于 mPaaS 框架的工程，在 portal 或 bundle 的 AndroidManifest.xml 中都可以配置。

通配符说明：你需要将代码中 ${} 替换为真实的值。

${applicationId}：包名。示例：

将 ${applicationId}.push.action.CHECK 替换为 com.mpaas.demo.push.action.CHECK。

${appId}与${workspaceId}：应用 ID 与工作空间 ID。在 **mPaaS 控制台** > **代码管理** > **代码配置** 页面 下载配置文件 ，.config 文件包含了两者的值。

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.BROADCAST_STICKY"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<!-- RecvMsgIntentService 是指前文代码中定义的 service -->
<service android:name="com.mpaas.demo.push.RecvMsgIntentService"
android:exported="false">
<intent-filter>
//MESSAGE_RECEIVED 表示处理接收到的消息
<action android:name="${applicationId}.push.action.MESSAGE_RECEIVED"/>
//REGISTRATION_ID 表示处理获取到的设备标识
<action android:name="${applicationId}.push.action.REGISTRATION_ID"/>

<category android:name="${applicationId}"/>
</intent-filter>
</service>
<!-- 需要配置您的 alipush appId-->
<meta-data
android:name="ALIPUSH_APPID"
android:value="${appId}-${workspaceId}"/>
<service
android:name="com.alipay.pushsdk.push.NotificationService"
android:enabled="true"
android:exported="false"
android:process=":push"
android:label="NotificationService">
<intent-filter>
<action android:name="${applicationId}.push.action.START_PUSHSERVICE"/>
</intent-filter>
</service>

<service
android:name="com.alipay.pushsdk.push.AppInfoRecvIntentService"
android:exported="false"
android:process=":push">
</service>

<receiver
android:name="com.alipay.pushsdk.BroadcastActionReceiver"
android:enabled="true"
android:process=":push">
<intent-filter android:priority="2147483647">
<action android:name="android.intent.action.BOOT_COMPLETED"/>
<action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
<action android:name="android.intent.action.USER_PRESENT"/>
<action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>
</intent-filter>
```

```
</receiver>

<receiver
android:name="com.alipay.mobile.logmonitor.ClientMonitorWakeupReceiver"
android:enabled="true"
android:process=":push">
<intent-filter>
<action android:name="android.intent.action.BOOT_COMPLETED"/>
<action android:name="${applicationId}.push.action.CHECK"/>
<action android:name="${applicationId}.monitor.command"/>
</intent-filter>
</receiver>
```

您可在控制台上创建极简推送类型的消息，测试基于设备维度的消息推送是否成功。
创建消息的操作方法，参见 创建消息 > 操作方法 。

说明：

- 收到消息后不会弹出通知栏。
- 为了防止主进程被杀死后通知栏无法弹出，建议开发者在消息接收服务的 handleActionReceived(String s, String s1, boolean clicked) 方法中添加通知栏弹出代码。

后续步骤

- 为了提升消息推送的到达率，您可以选择是否接入 Android 客户端三方推送渠道，具体方法参考 第三方推送渠道 。
- pushSDK 支持基于用户标识的推送，你可以选择将用户标识和设备标识进行绑定，具体方法参考 上报用户 ID 。

相关链接

- 示例代码
- 常见问题

4.2.4 接入第三方推送渠道

为了提升推送的到达率，mPaaS 集成了华为、小米等厂商的推送功能。采用 小米通知栏消息、 华为通知消息 实现推送，在进程挂起时，依然可以发送通知，用户点击通知栏可以激活进程。

说明：接入厂商自有的推送渠道后，能够帮助应用获得稳定的推送性能，因此建议您将第三方推送渠道接入您的应用。

本文档分为 客户端接入第三方推送渠道 、 接入 MpaaSNcActivity 、和 FAQ 三部分，引导开发者接入三方渠道的推送功能。在 客户端接入第三方推送渠道 中，分别介绍了接入华为、小米两家厂商的推送服务所需要进行的客户端配置。

客户端接入第三方推送渠道

接入华为渠道

注册华为推送

登录华为开发官网，注册账号并且开启推送服务。详情请参见 华为推送开启步骤。

客户端接入华为推送

1. 接入 MPS 依赖。第三方渠道的接入与 MPS 自建通道的接入完全相同，更多信息请参见 添加 SDK 。

   配置 AndroidManifest.xml。

```
<activity
android:name="com.huawei.hms.activity.BridgeActivity"
android:configChanges="orientation|locale|screenSize|layoutDirection|fontScale"
android:excludeFromRecents="true"
android:exported="false"
android:hardwareAccelerated="true"
android:theme="@android:style/Theme.Translucent">
<meta-data
android:name="hwc-theme"
android:value="androidhwext:style/Theme.Emui.Translucent"/>
</activity>
<!--为了防止低版本 dex 崩溃，动态开启 provider，enabled 设置为 false-->
<provider
android:name="com.huawei.hms.update.provider.UpdateProvider"
android:authorities="${applicationId}.hms.update.provider"
android:exported="false"
android:enabled="false"
android:grantUriPermissions="true">
</provider>

<!-- value 的值“appid”用实际申请的应用 ID 替换，来源于开发者联盟网站应用的服务详情。-->
<meta-data
android:name="com.huawei.hms.client.appid"
android:value="your huawei appId"/>
<receiver
android:name="com.huawei.hms.support.api.push.PushEventReceiver"
>
<intent-filter>
<!-- 接收通道发来的通知栏消息，兼容老版本PUSH -->
<action android:name="com.huawei.intent.action.PUSH"/>
</intent-filter>
</receiver>

<receiver
android:name="com.alipay.pushsdk.thirdparty.huawei.HuaweiPushReceiver">
<intent-filter>
<!-- 必须,用于接收 TOKEN -->
<action android:name="com.huawei.android.push.intent.REGISTRATION"/>
<!-- 必须，用于接收消息 -->
<action android:name="com.huawei.android.push.intent.RECEIVE"/>
<!-- 可选，用于点击通知栏或通知栏上的按钮后触发 onEvent 回调 -->
<action android:name="com.huawei.android.push.intent.CLICK"/>
<!-- 可选，查看PUSH通道是否连接，不查看则不需要 -->
<action android:name="com.huawei.intent.action.PUSH_STATE"/>
</intent-filter>
</receiver>
```

3. 接入 MpaaSNcActivity 。

接入小米渠道

注册小米推送

参考以下小米官方文档，完成小米推送注册：

- 小米开发者注册
- 启用小米推送

客户端接入小米推送

1. 接入 MPS 依赖。第三方渠道的接入与 MPS 自建通道的接入完全相同，更多信息请参见 添加 SDK 。

   配置 AndroidManifest.xml。

```
<!--由于涉及到签名，权限放在 Portal 中-->
<permission
android:name="${applicationId}.permission.MIPUSH_RECEIVE"
android:protectionLevel="signature"/>
<uses-permission android:name="${applicationId}.permission.MIPUSH_RECEIVE"/>
<!-- value 斜杠空格要保留 -->
<meta-data
android:name="xiaomi_appid"
android:value="\ 2xxxxxxxxxx"/>
<!-- value 斜杠空格要保留 -->
<meta-data
android:name="xiaomi_appkey"
android:value="\ 5xxxxxxxxxxx"/>

<service
android:name="com.xiaomi.push.service.XMPushService"
android:enabled="true"
android:process=":push"/>
<service
android:name="com.xiaomi.mipush.sdk.PushMessageHandler"
android:enabled="true"
android:exported="true"
android:process=":push"/>
<service
android:name="com.xiaomi.mipush.sdk.MessageHandleService"
android:enabled="true"
android:process=":push"/>

<receiver
android:name="com.xiaomi.push.service.receivers.NetworkStatusReceiver"
android:exported="true"
android:process=":push">
<intent-filter>
<action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
<category android:name="android.intent.category.DEFAULT"/>
```

```
</intent-filter>
</receiver>
<receiver
android:name="com.xiaomi.push.service.receivers.PingReceiver"
android:exported="false"
android:process=":push">
<intent-filter>
<action android:name="com.xiaomi.push.PING_TIMER"/>
</intent-filter>
</receiver>
<receiver
android:name="com.alipay.pushsdk.thirdparty.xiaomi.XiaoMiMsgReceiver"
android:exported="true"
android:process=":push">
<intent-filter>
<action android:name="com.xiaomi.mipush.RECEIVE_MESSAGE"/>
</intent-filter>
<intent-filter>
<action android:name="com.xiaomi.mipush.ERROR"/>
</intent-filter>
<intent-filter>
<action android:name="com.xiaomi.mipush.MESSAGE_ARRIVED"/>
</intent-filter>
</receiver>
```

3. 接入 MpaaSNcActivity 。

接入 MpaaSNcActivity

```
<activity
android:name="com.alipay.pushsdk.thirdparty.MPaaSNcActivity"
android:exported="true"
android:theme="@android:style/Theme.Translucent">
<intent-filter>
<action android:name="android.intent.action.VIEW"/>
<category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
</activity>
```

FAQ

实现 PUSH 通知栏消息，对 EMUI 和华为移动服务是否有版本限制？
有版本限制，详细版本要求请参见 华为消息推送服务 EMUI 版本限制说明。

华为手机无法打印日志
在手机拨号界面输入 ***#*#2846579#*#*** 进入工程菜单 > **后台设置** > **LOG 设置** > 选中 **AP 日志**。重启手机后，logcat 开始生效。

华为手机推送错误码
如果在日志中出现 huawei client onConnectionFailed 记录，请至华为官网查看相关 错误码 及相应的处理建议。

常见问题排查步骤

- 检查 Manifest 文件是否配置正确。
- 检查 appId（华为、小米）、appSecret （小米）、ALIPUSH_APPID（mPaaS） 是否与对应开发平台的注册应用一致。
- 查看 tag 为 mpush 的 logcat 日志。

后续操作

在完成上述客户端接入第三方推送渠道的配置后，需要在消息推送控制台上配置对应三方渠道的相关参数。更多信息，参见 控制台渠道配置 。

4.2.5 上报用户 ID

如果需要用户维度的推送，当用户登录时，将该用户标识上传到服务端，由服务端向移动推送服务发出请求，将用户和设备进行绑定。当用户登出时，将该绑定关系解绑。

服务端的相关处理请参考 服务端 文档。

操作步骤

1. 为了方便开发者进行用户绑定，提供接口进行绑定和解除绑定操作，示例代码如下：

```
// 具体代码可以参考 demo
// 这里的用户 userId 不一定为开发者用户系统的真实标识，但是一定是可以与用户形成一一映射关系
// 参数 userId，adtoken (会在 service handleActionId 中接收到)
ResultPbPB resultBean = MPPush.bind(getApplicationContext(), your userId, ad-token);
ResultPbPB resultBean = MPPush.unbind(getApplicationContext(), your userId, ad-token);
```

选择绑定时机，可选的关键时机如下：

- 用户登录（有登录态 app）和 push 收到 Ad-token 后，为了登录后首次接口收到 push。
- App 冷启动，为了防止出现未注册情形。

后续步骤

推送消息 ：按设备维度或用户维度推送消息。

相关链接

- 示例代码
- 常见问题

4.2.6 上报推送数据

PushSDK 定义消息到达、打开和忽略的接口，调用这些接口，上报相关数据至服务端后，您可在 **控制台** > **后台服务管理** > **消息推送** > **使用分析** 页面上查看各数据的统计分析结果。

到达

在接入 PushSDK 之后，您无需执行任何操作，MPS 将自动完成消息到达相关数据的统计分析。

对于通过不同渠道推送的消息，MPS 采用不同的方式统计其到达量和到达率。

- **自建渠道**
  通过自建渠道推送的消息，在其到达客户端时，PushSDK 将自动上报消息到达事件，服务端自动统计消息的到达量以及到达率。
- **三方渠道**
  通过三方渠道推送的消息，由相应渠道的后端服务返回推送结果数据，MPS 对此数据进行统计分析，得到消息的到达量和到达率。

打开

调用 reportPushOpen 接口，上报消息打开事件，服务端将统计消息被打开的数量以及打开率。

```
public static void reportPushOpen(String pushMsgId, String pushMsgValue)
```

调用时机

对于三方渠道的通知栏消息以及需要自定义展示的消息，调用时机不同，具体如下：

- 三方渠道通知栏消息：消息被打开后，PushSDk 将自动调用 reportPushOpen 接口，上报相关数据。
- 自定义展示的消息：您需要添加消息打开操作的监听，监听到打开操作时，调用 reportPushOpen 接口，上报相关数据。

参数说明

pushMsgId 和 pushMsgValue 可从推送的消息体中获取，如下图所示。

- pushMsgId：对应于以上消息体中的 k 值，为消息 ID。
- pushMsgValue：对应于以上消息体中的 bData 值。

忽略

调用 reportPushIgnored 接口，上报消息的忽略事件，服务端将统计消息的忽略量以及忽略率。

```
public static void reportPushIgnored(String pushMsgId, String pushMsgValue)
```

调用时机

对于需要自定义展示的消息，您需要添加消息被忽略的监听，监听到消息被忽略时，调用 reportPushIgnored 接口，上报相关数据。

说明：对于无法自定义展示的消息，MPS 将无法统计其忽略量。

参数说明

参见 参数说明 。

4.3 使用 SDK（版本 < 10.1.32）

4.3.1 步骤概览

在 Android 客户端中使用 pushSDK，您需要进行如下操作：

1. 初始化 pushSDK ：建立客户端和移动推送网关的长链接，获取设备标识。
2. 监听消息 ：为接收消息、获取 Android 设备标识（Ad-token）编写 service，实现设备维度的消息推送。
3. 接入第三方推送渠道 ：可选，提升消息推送的到达率。
4. 上报用户 ID ：将用户标识上传到服务端，进行用户和设备绑定，实现用户维度的消息推送。

相关操作

推送消息 ：按设备维度或用户维度推送消息。

相关链接

- 示例代码
- 常见问题

4.3.2 初始化 pushSDK

初始化 pushSDK 过程中，客户端和移动推送网关建立长链接，并获取设备标识。

关于此任务

客户端通过 AliPushInterface 类来实现对 pushSDK 的控制。其中，appkey 为 alipush_appid：

```
AliPushInterface.init(Context context, String appKey)
```

初始化过程中， pushSDK 会用设定好的地址和端口尝试建立链接。如果客户端建链信息中未携带设备标识，移动推送网关将下发设备标识（Ad-token）。

在获取到设备标识后，pushSDK 会启动 App 中配置好的 service 来对设备标识进行处理。您可以通过这个 service 来获取这个 Ad-token。具体方法，参考 监听消息 中的示例。

相关链接

- 示例代码
- 常见问题

4.3.3 监听消息

在接收到推送消息，或者获取到 Ad-token 后，pushSDK 都会启动一个 service 来对该消息或该 Ad-token 进行处理。为接收消息、获取 Android 设备标识（Ad-token）编写 service，实现设备维度的消息推送。

操作步骤

编写 service。客户端可以分别为接收消息、获取 Ad-token 各编写一个 service，也可以共用同一个 service。下面是共用同一个 service 的示例代码：

```
public class RecvMsgIntentService extends AliPushRcvService {
public RecvMsgIntentService() {
super();
}

// 调用时机：
// 华为、小米手机：接入推送后，点击通知栏时进行调用
// 其他手机：收到消息时进行调用
@Override
protected void handleActionReceived(String s, String s1, boolean clicked) {
//示例代码
Intent intent1 = new Intent("tt-action");
String extra ="key:="+ s +"data:="+ s1;
intent1.putExtra("intent :", extra);
LocalBroadcastManager.getInstance(this).sendBroadcast(intent1);
Log.v("push-test","onHandleIntent sendLocalBroadcast :" + intent1.toString());
if (TextUtils.isEmpty(s1)) {
return;
}
if(clicked){
//华为小米采用通知栏消息，所以一定点击过
}
BDataBean bDataBean = BDataBean.create(s1);
// BDataBean 含有自定义的Param
if (bDataBean == null) {
return;
}
// 处理默认的跳转等事件
// 可以不调用以下方法，自行处理
bDataBean.action(this);
}

public static String adToken ="";
public static String thirdToken ="";
public static int platform = 0;

// 自建渠道 AdToken 返回
/** 调用时机：完成初始化后，收到自建渠道标识时调用
*@param s adtoken 自建渠道标识
*/
@Override
protected void handleActionId(String s) {
// 示例代码
String registrationId = s;
PushAppInfo pushAppInfo = new PushAppInfo(getApplicationContext());
pushAppInfo.setAppToken(registrationId);
Log.e("push-test","registrationId =" + registrationId);
Intent intent1 = new Intent("tt-action");
String extra ="adToken:="+ s;
intent1.putExtra("intent :", extra);
LocalBroadcastManager.getInstance(this).sendBroadcast(intent1);
Log.e("push-test","onHandleIntent sendLocalBroadcast :" + intent1.toString());
adToken = s;
```

```
}
/**
*调用时机：收到第三方渠道标识时调用
*@param s thirdToken 第三方渠道标识
*@param i platform 第三方渠道类型
*/
@Override
protected void handleActionThirdId(String s, int i) {
// 示例代码
Intent intent1 = new Intent("tt-action");
String extra ="third token:="+ s +"channel"+ i;
intent1.putExtra("intent :", extra);
LocalBroadcastManager.getInstance(this).sendBroadcast(intent1);
Log.e("push-test","onHandleIntent sendLocalBroadcast :" + intent1.toString());
thirdToken = s;
platform = i;
}
}
```

在 AndroidManifest.xml 配置此 service。

说明：基于 mPaaS 框架的工程，在 portal 或 bundle 的 AndroidManifest.xml 中都可以配置。

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.BROADCAST_STICKY"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<!-- RecvMsgIntentService 是指前文代码中定义的 service，这里推荐些绝对路径-->
<service android:name="xxx.push.RecvMsgIntentService"
android:exported="false">
<intent-filter>
<action android:name="${applicationId}.push.action.MESSAGE_RECEIVED"/>
<action android:name="${applicationId}.push.action.REGISTRATION_ID"/>

<category android:name="${applicationId}"/>
</intent-filter>
</service>
<!-- 需要配置您的 alipush appId-->
<meta-data
android:name="ALIPUSH_APPID"
android:value="${appId}-${workspaceId}"/>
<service
android:name="com.alipay.pushsdk.push.NotificationService"
android:enabled="true"
android:exported="false"
android:process=":push"
android:label="NotificationService">
<intent-filter>
<action android:name="${applicationId}.push.action.START_PUSHSERVICE"/>
```

```
</intent-filter>
</service>

<service
android:name="com.alipay.pushsdk.push.AppInfoRecvIntentService"
android:exported="false"
android:process=":push">
</service>

<receiver
android:name="com.alipay.pushsdk.BroadcastActionReceiver"
android:enabled="true"
android:process=":push">
<intent-filter android:priority="2147483647">
<action android:name="android.intent.action.BOOT_COMPLETED"/>
<action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
<action android:name="android.intent.action.USER_PRESENT"/>
<action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>
</intent-filter>
</receiver>

<receiver
android:name="com.alipay.mobile.logmonitor.ClientMonitorWakeupReceiver"
android:enabled="true"
android:process=":push">
<intent-filter>
<action android:name="android.intent.action.BOOT_COMPLETED"/>
<action android:name="${packageName}.push.action.CHECK"/>
<action android:name="${packageName}.monitor.command"/>
</intent-filter>
</receiver>
```

您可在控制台上创建极简推送类型的消息，测试基于设备维度的消息推送是否成功。
创建消息的操作方法参见 创建消息 > 操作方法 。

说明：

- 收到消息后不会弹出通知栏。
- 为了防止主进程被杀死后通知栏无法弹出，建议开发者在消息接收服务的 handleActionReceived(String s, String s1, boolean clicked) 方法中添加通知栏弹出代码。

后续步骤

- 为了提升消息推送的到达率，您可以选择是否接入 Android 客户端三方推送渠道，具体方法参考 第三方推送渠道 。
- pushSDK 支持基于用户标识的推送，你可以选择将用户标识和设备标识进行绑定，具体方法参考 上报用户 ID 。

相关链接

- 示例代码
- 常见问题

4.3.4 接入第三方推送渠道

为了提升推送的到达率，mPaaS 集成了华为、小米等厂商的推送功能。采用 小米通知栏消息 和 华为通知消息 的方式实现推送，在进程挂起时，依然可以发送通知，用户点击通知栏可以激活进程。

本文档会分为 接入华为渠道 和 接入小米渠道 两个部分，引导开发者接入三方渠道的推送功能。

接入华为渠道

注册华为推送

需要开发者登录华为开发官网，注册账号并且开启推送服务。可参见 华为推送开启步骤。

接入华为推送

1. 接入 MPS 依赖，三方渠道的接入与 MPS 自建通道的接入，在这一点完全相同。可参见 接入 pushSDK 。

   配置 AndroidManifest.xml：

```
<activity
android:name="com.huawei.hms.activity.BridgeActivity"
android:configChanges="orientation|locale|screenSize|layoutDirection|fontScale"
android:excludeFromRecents="true"
android:exported="false"
android:hardwareAccelerated="true"
android:theme="@android:style/Theme.Translucent">
<meta-data
android:name="hwc-theme"
android:value="androidhwext:style/Theme.Emui.Translucent"/>
</activity>
<!--为了防止低版本 dex 崩溃，动态开启 provider，enabled 设置为 false-->
<provider
android:name="com.huawei.hms.update.provider.UpdateProvider"
android:authorities="${applicationId}.hms.update.provider"
android:exported="false"
android:enabled="false"
android:grantUriPermissions="true">
</provider>

<!-- value 的值“appid”用实际申请的应用 ID 替换，来源于开发者联盟网站应用的服务详情。-->
<meta-data
android:name="com.huawei.hms.client.appid"
android:value="your huawei appId"/>
<receiver
android:name="com.huawei.hms.support.api.push.PushEventReceiver"
>
<intent-filter>
<!-- 接收通道发来的通知栏消息，兼容老版本PUSH -->
<action android:name="com.huawei.intent.action.PUSH"/>
</intent-filter>
</receiver>

<receiver
android:name="com.alipay.pushsdk.thirdparty.huawei.HuaweiPushReceiver">
<intent-filter>
```

```
<!-- 必须,用于接收 TOKEN -->
<action android:name="com.huawei.android.push.intent.REGISTRATION"/>
<!-- 必须，用于接收消息 -->
<action android:name="com.huawei.android.push.intent.RECEIVE"/>
<!-- 可选，用于点击通知栏或通知栏上的按钮后触发 onEvent 回调 -->
<action android:name="com.huawei.android.push.intent.CLICK"/>
<!-- 可选，查看PUSH通道是否连接，不查看则不需要 -->
<action android:name="com.huawei.intent.action.PUSH_STATE"/>
</intent-filter>
</receiver>
```

3. 接入 MpaaSNcActivity 。

接入小米渠道

注册小米推送

参看以下第三方文档，完成小米推送注册：

- 小米开发者注册
- 启用小米推送

接入小米推送

1. 接入 MPS 依赖，三方渠道的接入与 MPS 自建通道的接入，在这一点完全相同。可参见 接入 pushSDK 。

   配置 AndroidManifest.xml：

```
<!--由于涉及到签名，权限放在 Portal 中-->
<permission
android:name="${applicationId}.permission.MIPUSH_RECEIVE"
android:protectionLevel="signature"/>
<uses-permission android:name="${applicationId}.permission.MIPUSH_RECEIVE"/>
<!-- value 斜杠空格要保留 -->
<meta-data
android:name="xiaomi_appid"
android:value="\ 2xxxxxxxxx"/>
<!-- value 斜杠空格要保留 -->
<meta-data
android:name="xiaomi_appkey"
android:value="\ 5xxxxxxxxxxx"/>

<service
android:name="com.xiaomi.push.service.XMPushService"
android:enabled="true"
android:process=":push"/>
<service
android:name="com.xiaomi.mipush.sdk.PushMessageHandler"
android:enabled="true"
android:exported="true"
android:process=":push"/>
```

```
<service
android:name="com.xiaomi.mipush.sdk.MessageHandleService"
android:enabled="true"
android:process=":push"/>

<receiver
android:name="com.xiaomi.push.service.receivers.NetworkStatusReceiver"
android:exported="true"
android:process=":push">
<intent-filter>
<action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
<category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
</receiver>
<receiver
android:name="com.xiaomi.push.service.receivers.PingReceiver"
android:exported="false"
android:process=":push">
<intent-filter>
<action android:name="com.xiaomi.push.PING_TIMER"/>
</intent-filter>
</receiver>
<receiver
android:name="com.alipay.pushsdk.thirdparty.xiaomi.XiaoMiMsgReceiver"
android:exported="true"
android:process=":push">
<intent-filter>
<action android:name="com.xiaomi.mipush.RECEIVE_MESSAGE"/>
</intent-filter>
<intent-filter>
<action android:name="com.xiaomi.mipush.ERROR"/>
</intent-filter>
<intent-filter>
<action android:name="com.xiaomi.mipush.MESSAGE_ARRIVED"/>
</intent-filter>
</receiver>
```

3. 接入 MpaaSNcActivity 。

接入 MpaaSNcActivity

```
<activity
android:name="com.alipay.pushsdk.thirdparty.MPaaSNcActivity"
android:exported="true"
android:theme="@android:style/Theme.Translucent">
<intent-filter>
<action android:name="android.intent.action.VIEW"/>
<category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
</activity>
```

配置控制台

完成各平台渠道接入后，需要在消息推送控制台配置华为和小米推送配置信息。

1. 进入移动开发平台后，选择需要配置消息推送的应用。

2. 在左侧导航栏中，点击 **后台服务管理** > **消息推送** > **渠道配置**。

配置渠道。

华为推送渠道

- 点击 **华为推送渠道** 栏中的 **配置**。
- 打开 **状态** 开关。
- 在 **包名** 栏中，输入华为的应用包名。
- 在 **华为应用ID** 栏中，输入华为应用的 App ID。
- 在 **华为应用密钥** 栏中，输入华为应用的密钥（App Secret）。

  说明：应用包名、应用 App ID、密钥可从 **华为开发者联盟** > **管理中心** > **我的产品** > **移动应用详情** 中获取。

小米推送渠道

- 点击 **小米推送渠道** 栏中的 **配置**。
- 打开 **状态** 开关。
- 在 **包名** 栏中，输入小米应用的主包名。
- 在 **密码** 栏中，输入小米应用的密钥（AppSecret）。

  说明：主包名和密钥可从 **小米开放平台** > **应用管理** > **应用信息** 中获取。

点击 **确定**，完成渠道配置。

FAQ

实现 PUSH 通知栏消息，对 EMUI 和华为移动服务是否有版本限制？
有版本限制，详细版本要求请参见 华为消息推送服务 EMUI 版本限制说明。

华为手机无法打印日志
在手机拨号界面输入 ***#*#2846579#*#*** 进入工程菜单 > **后台设置** > **LOG 设置** > 选中 **AP 日志**。重启手机后，logcat 开始生效。

华为手机推送错误码
如果在日志中出现 huawei client onConnectionFailed 记录，请至华为官网查看相关 错误码 及相应的处理建议。

问题排查步骤

- 检查 Manifest 文件是否配置正确。
- 检查 appId（华为、小米）、appSecret （小米）、ALIPUSH_APPID（mPaaS） 是否与对应开发平台的注册应用一致。
- 查看 tag 为 mpush 的 logcat 日志。

4.3.5 上报用户 ID

如果需要用户维度的推送，当用户登录时，将该用户标识上传到服务端，由服务端向移动推送服务发出请求，将用户和设备进行绑定。当用户登出时，将该绑定关系解绑。

服务端的相关处理请参考 服务端 文档。

操作步骤

1. 为了方便开发者进行用户绑定，提供接口进行绑定和解除绑定操作，示例代码如下：

```
// 具体代码可以参考demo
// 这里的用户userId 不一定是开发者用户系统的真实标识，但是一定是可以形成一一映射关系
BIND bind = new BIND();
ResultPbPB resultBean = bind.bind(getApplicationContext(), your userId, ad-token);
ResultPbPB resultBean = bind.unbind(getApplicationContext(), your userId, ad-token);
```

2. 选择绑定时机，可选的关键时机如下：
   - 用户登录（有登录态 app）和 push 收到 Ad-token 后，为了登录后首次接口收到 push。
   - App 冷启动，为了防止出现未注册情形。

后续步骤

推送消息 ：按设备维度或用户维度推送消息。

相关链接

- 示例代码
- 常见问题

4.4 推送消息

获取到设备标识（Ad-token）后，就可以按设备维度向应用推送消息。上报用户 ID 并由服务端绑定用户和设备后，就可以按用户维度推送。

关于此任务

目前支持的推送方式如下：

- 极简推送

- 模板推送
- 批量推送
- 群发推送

注意：**消息类型** 提供以下的选择：

- **通知栏消息**：在接入第三方渠道的情况下，选择 **通知栏消息** 会优先使用第三方渠道。当使用第三方渠道，您无法自定义消息弹出时的展示样式。
- **透传消息**：不使用第三方渠道，您可以使用自建渠道，自定义展示样式。

相关链接

- 在控制台手动推送消息，参考 创建消息 。
- 在服务端利用代码调用接口推送消息，参考 配置服务端 。

4.5 常见问题

关于权限的说明
Android 6.0 后，需要用户手动授予手机权限，如读写 SD 卡。为了更加精准的发送推送，建议开发者引导获取消息推送所需权限。

接入华为、小米等第三方渠道后，无法发送推送
这是由于没有打开 mPaaS 推送控制台的渠道的设置开关。

示例代码：参见 获取代码示例 以获取代码示例以及使用方法和注意事项。

关于日志
使用魅族手机测试时，如 log.d 和 log.i 日志无法打印，您可通过在 **设置** > **辅助功能** > **开发者选项** 中打开 **高级日志输出**。

如遇开发问题，可设置 tag=mpush，对日志进行过滤。

关于 push ad-token (deviceId) 的生成
服务端依赖 IMSI 和 IMEI 生成 deviceId。因此，建议开发引导用户获取所需的 READ_PHONE_STATE 权限。

5 接入 Android——基于原生工程

5.1 Demo

请参考 代码示例 。

5.2 接入 SDK

本文将引导您接入消息推送 SDK。包括：

- 前置条件
- 通用基础配置
- 添加 aar 依赖
- 添加 ProGuard 混淆规则
- 后续步骤

前置条件

您首先需要 在控制台创建应用并下载配置文件 。注意在下载配置文件时，必须上传签名后的 APK（应和 **发布版本** 的 APK 使用相同的签名文件），此时下载到的配置文件 xx.config 中 base64Code 的值应该非空。

通用基础配置

请参考 文档 完成通用基础配置。

添加 aar 依赖

在 build.gradle 中添加如下依赖：

```
compile"com.mpaas.aar:push_xiaomi:3.6.12_simplified"
compile"com.mpaas.aar:push_huawei:2.5.2.201_simplified"
compile"com.mpaas.aar:common:10.1.20_adapter"
compile"com.mpaas.aar:rpc:10.1.20.4"
compile 'com.mpaas.aar:logging:10.1.32.0'
compile 'com.mpaas.aar:monitor:10.1.32.0'
compile 'com.mpaas.aar:pushsdk-build:10.1.32.2'
```

添加 ProGuard 混淆规则

ProGuard 混淆可以保护 App 免于破解。为了使用 ProGuard，您需要添加特殊的混淆规则。更多信息，请参见 使用 ProGuard 。

后续步骤

使用 SDK

5.3 使用 SDK

为了使用 SDK，推荐您先了解消息推送的 原理架构 。重点关注消息推送的以下流程：

- 初始化 。
- 网关向客户端下发设备标识。更多信息，请参见下文关于 AliPushRcvService#handleActionId() 的说明。
- 小米、华为等三方渠道初始化。更多信息，请参见下文关于 AliPushRcvService#handleActionThirdId() 的说明。
- 绑定用户标识与设备标识 。
- 客户端接收推送的消息。更多信息，请参见下文关于 AliPushRcvService#handleActionReceived() 的说明 。

初始化

在应用启动时，调用以下方法初始化 SDK：

```
MPPush.init(this);
```

监听消息

您需要编写监听消息的 Service，并在 AndroidManifest 中配置。

编写 Service

该 Service 需要继承 AliPushRcvService。

代码示例

```
public class PushReceiver extends AliPushRcvService {
public static final String TAG ="mpaas_push";
static ArrayList<WeakReference<IPushReceiver>> mListeners = new ArrayList<>();

@Override
protected void handleActionId(String adToken) {
LoggerFactory.getTraceLogger().debug(TAG,"AdToken:"+ adToken);
Iterator<WeakReference<IPushReceiver>> iter = mListeners.iterator();
while (iter.hasNext()) {
WeakReference<IPushReceiver> value = iter.next();
IPushReceiver receiver = value.get();
if (receiver == null) {
iter.remove();
} else {
receiver.handleActionId(adToken);
}
}
}

@Override
protected void handleActionThirdId(String thirdToken, int channel) {
LoggerFactory.getTraceLogger().debug(TAG,"ThirdToken:"+ thirdToken);
Iterator<WeakReference<IPushReceiver>> iter = mListeners.iterator();
while (iter.hasNext()) {
WeakReference<IPushReceiver> value = iter.next();
IPushReceiver receiver = value.get();
if (receiver == null) {
iter.remove();
} else {
receiver.handleActionThirdId(thirdToken, channel);
}
}
}

@Override
protected void handleActionReceived(String msgkey, String msgdata, boolean isFromThirdChannel) {
Iterator<WeakReference<IPushReceiver>> iter = mListeners.iterator();
while (iter.hasNext()) {
WeakReference<IPushReceiver> value = iter.next();
IPushReceiver receiver = value.get();
if (receiver == null) {
iter.remove();
} else {
PushData data = new PushData();
data.mMsgKey = msgkey;
data.mMsgValue = msgdata;
receiver.handleActionReceived(data);
}
}
}

public static void register(IPushReceiver receiver) {
mListeners.add(new WeakReference<IPushReceiver>(receiver));
}
```

```
public static void unregister(IPushReceiver receiver) {
Iterator<WeakReference<IPushReceiver>> iter = mListeners.iterator();
while (iter.hasNext()) {
WeakReference<IPushReceiver> value = iter.next();
if (value.get() == null || value.get() == receiver) {
iter.remove();
}
}
}
}
```

接口说明

handleActionId(String adToken)

初始化完成之后，网关会向客户端下发设备标识（即 adToken）。

一般地，您需要记录该设备标识，并在用户登录时，将设备标识与用户标识绑定。更多信息，请参见下文 绑定用户标识和设备标识 。

handleActionThirdId(String thirdToken, int channel)

如果接入了 小米 、 华为 等三方渠道，在三方渠道初始化之后，会触发该回调。其中，

- thirdToken：第三方渠道对应的设备标识。消息推送组件内部会将 mPaaS 设备标识（对应上文 handleActionId(String adToken) 的 adToken）与三方渠道的设备标识（即 thirdToken）打通。
- channel：渠道标识。

您可以通过此方法验证三方渠道是否接入成功。只有收到了三方渠道的设备标识，才说明三方渠道接入成功。

handleActionReceived(String msgkey, String msgdata, boolean isFromThirdChannel)

- 接口描述：接收推送的消息。
- 回调时机：
    - mPaaS 自建渠道：
      当收到推送消息时会立即回调到，收到消息后用户自动处理，比如弹通知栏等
    - 三方渠道（小米或华为）：
      mPaaS 推送后台最终通过三方渠道的推送后台进行消息推送。目前三方渠道推送的消息由系统处理必然会显示通知栏，此时并不会回调到本函数，只有当用户点击了通知栏才会回调到本函数。
- 参数说明：
    - isFromThirdChannel：是否来自于三方渠道。
    - msgkey：如下图所示，对应于 Android 消息体中的 “k” 的值。
    - msgdata: 如下图所示，对应于 Android 消息体中的 “bData” 的值。

配置 AndroidManifest

如下所示，配置 AndroidManifest：

```
<service
android:name="com.mpaas.pushaar.demo.demo_pushaar.PushReceiver"
android:exported="false">
<intent-filter>
<action android:name="${applicationId}.push.action.MESSAGE_RECEIVED"/>
<action android:name="${applicationId}.push.action.REGISTRATION_ID"/>

<category android:name="${applicationId}"/>
</intent-filter>
</service>
```

其中，action 中使用了 ${applicationId} 占位符，mPaaS 定义的 gradle 插件会将其替换成应用的 applicationId。您也可以直接将 ${applicationId} 替换成应用的 applicationId。

绑定用户标识和设备标识

通过控制台推送消息时，您可以指定接受者的 **用户标识（ID）** 或 **设备标识**。

但本质上，mPaaS 按照设备标识进行消息推送。因此，当用户登录时，您需要将该用户标识绑定到对应的设备标识上。用户登出时，将该绑定关系解绑。

绑定

当用户登录时，调用以下方法将 用户标识 和 设备标识 绑定：

```
ResultPbPB resultBean = MPPush.bind(context, userId, adToken);
```

其中，

- userId：用户标识。
- adToken：设备标识。即通过 AliPushRcvService#handleActionId(String adToken) 回调收到的 adToken。更多信息，请参见上文。

解绑

当用户登出时，调用以下方法解绑 用户标识 和 设备标识：

```
ResultPbPB resultBean = MPPush.unbind(context, userId, adToken);
```

5.4 接入华为推送渠道

第三方推送是指厂商自己的推送，能够保证高到达率。本文将引导您接入华为推送渠道。包括：

- 添加依赖
- 获取华为消息推送 AppId
- 配置 AndroidManifest

添加依赖

在 build.gradle 中添加如下依赖：

```
compile"com.mpaas.aar:push_huawei:2.5.2.201_simplified"
```

获取华为消息推送 AppId

参考 华为推送开启步骤 获取华为消息推送 AppId。

配置 AndroidManifest

在 AndroidManifest 中添加如下配置：

> 提醒：您需要将 value 替换为真实的 华为消息推送 AppId。

```
<meta-data
android:name="com.huawei.hms.client.appid"
android:value="\ 111111111"/>
```

5.5 接入小米推送渠道

第三方推送是指厂商自己的推送，能够保证高到达率。本文将引导您接入小米推送渠道。包括：

- 添加依赖
- 获取小米消息推送 AppId 和 AppKey
- 配置 AndroidManifest

添加依赖

在 build.gradle 中添加如下依赖：

```
compile"com.mpaas.aar:push_xiaomi:3.6.12_simplified"
```

获取小米消息推送 AppId 和 AppKey

参考 小米开发者注册 和 启用小米推送 获取小米消息推送 AppId 和 AppKey。

配置 AndroidManifest

在 AndroidManifest 中配置 小米消息推送 AppId 和 AppKey。

```
<meta-data
android:name="xiaomi_appid"
android:value="\ 2111111111111111111"/>

<meta-data
android:name="xiaomi_appkey"
android:value="\ 5111111111111"/>
```

5.6 使用 ProGuard

ProGuard 用于压缩、优化和混淆 Java 字节码文件。本文引导您使用 ProGuard。

开启混淆

在 gradle.build 中添加 ProGuard 配置。示例如下：

```
android {
buildTypes {
release {
// 是否混淆。true 或 false
minifyEnabled true
// 混淆文件列表，混淆规则配置
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
}
}
}
```

热修复混淆规则

使用热修复组件，需要添加特殊的混淆规则：

> 推荐您查看 Demo mpaas_aar_push_demo 工程中的 proguard-rules.pro 文件。

```
-optimizationpasses 5
-dontusemixedcaseclassnames
-dontskipnonpubliclibraryclasses
-dontpreverify
-verbose
-ignorewarnings
-optimizations !code/simplification/arithmetic,!field/*,!class/merging/*

-keepattributes SourceFile,LineNumberTable

-keep public class * extends android.app.Activity
-keep public class * extends android.support.v4.app.FragmentActivity
-keep public class * extends android.support.v4.app.Fragment
-keep public class * extends android.app.Application{
!private <fields>;
!private <methods>;
}
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
-keep public class com.android.vending.licensing.ILicensingService


-keepclasseswithmembernames class * {
native <methods>;
}

-keepclasseswithmembernames class * {
public <init>(android.content.Context, android.util.AttributeSet);
}

-keepclasseswithmembernames class * {
public <init>(android.content.Context, android.util.AttributeSet, int);
}

-keepclassmembers enum * {
public static **[] values();
public static ** valueOf(java.lang.String);
}

-keep class * implements android.os.Parcelable {
public static final android.os.Parcelable$Creator *;
}

-keep class com.eg.android.AlipayGphone.R* {
*;
}
```

```
-keep class com.alipay.mobile.quinox.BundleContext {
!private <fields>;
!private <methods>;
}

-keep class com.alipay.mobile.quinox.apk.FragmentActivityShell{
!private <fields>;
!private <methods>;
}

-keeppackagenames com.alipay.mobile.quinox.**
-keeppackagenames com.eg.android.**

-keep class com.alipay.mobile.quinox.engine.** {
!private <fields>;
!private <methods>;
}

-keep class com.alipay.mobile.quinox.shell.** {
!private <fields>;
!private <methods>;
}

-keep class com.alipay.mobile.quinox.apk.** {
!private <fields>;
!private <methods>;
}

-keep class com.alipay.mobile.apk.common.**{
!private <fields>;
!private <methods>;
}

-keep class com.alipay.mobile.quinox.startup.** {
!private <fields>;
!private <methods>;
}

-keep class com.eg.android.AlipayGphoneRC.R* {
*;
}

-keep class com.alipay.android.phone.automation.testui.MainTestActivity{
!private <fields>;
!private <methods>;
}

-keep class com.alipay.mobile.framework.service.ext.openplatform.persist.AllAppInfoDaoImpl{
!private <fields>;
!private <methods>;
}

-keepclassmembers class com.alipay.mobile.quinox.classloader.QuinoxClassLoader{
*;
}
```

```
-keepclassmembernames class com.alipay.mobile.quinox.classloader.BundleClassLoader {
private com.alipay.mobile.quinox.bundle.Bundle mBundle;
}

-keepclassmembernames class com.alipay.mobile.quinox.bundle.Bundle {
private java.lang.String mName;
}

-keep class com.alipay.mobile.quinox.LauncherActivity*{
*;
}

-keep class com.alipay.mobile.quinox.utils.FileUtil{
*;
}
-keep class com.alipay.android.launcher.StartupRuler{*;}

-keep class com.alipay.mobile.quinox.utils.**{ *; }

-keep class **{ *; }

-keep class com.alipay.android.phone.devtool.leakreporter.ApplicationInjector {
*;
}

-keepattributes SourceFile,LineNumberTable

-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
-keep public class * extends android.os.IInterface
-keep public class com.android.vending.licensing.ILicensingService

-keep class * implements android.os.Parcelable {
!private <methods>;
public static final android.os.Parcelable$Creator *;
}

-keep class * extends android.os.Binder{
!private <fields>;
!private <methods>;
}
-keep class * implements java.io.Serializable{
<fields>;
public <methods>;
}

-keep class * extends android.os.IInterface{
!private <fields>;
!private <methods>;
}

-keep public class com.alipay.pushsdk.AliPushInterface{
```

```
!private <fields>;
!private <methods>;
}

-keep public class com.alipay.pushsdk.util.ZipPushInit{
<fields>;
!private <methods>;
}

-keep public class com.alipay.pushsdk.PushExtConstants{
public protected <fields>;
}
-keep public class com.alipay.pushsdk.content.*{
!private <methods>;
}
-keep public class com.alipay.pushsdk.BroadcastActionReceiver{
public void onReceive(android.content.Context, android.content.Context);
}

-keep public class com.alipay.pushsdk.push.PushAppInfo{
public <fields>;
public <methods>;
}

-keep class com.alipay.pushsdk.util.log.LogUtil{
*;
}
-keep interface com.alipay.pushsdk.util.log.*{
*;
}

-keep class com.alipay.pushsdk.push.PushAddrConfig{
*;
}


#这里com.xiaomi.mipushdemo.DemoMessageRreceiver改成app中定义的完整类名
-keep class com.alipay.pushsdk.thirdparty.xiaomi.XiaoMiMsgReceiver {
*;
}
#可以防止一个误报的 warning 导致无法成功编译，如果编译使用的 Android 版本是 23。
-dontwarn com.xiaomi.push.**

# 华为push防止混淆
-ignorewarning
-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 alipay.yunpushcore.rpc.* {
*;
}
```

```
-keep class alipay.yunpushcore.rpc.bind.* {
*;
}
-keep class alipay.yunpushcore.rpc.bind.pb.* {
*;
}
-keep class alipay.yunpushcore.rpc.report.*{
*;
}
-keep class alipay.yunpushcore.rpc.report.pb.*{
*;
}
-keep class com.alipay.pushsdk.rpc.*{
*;
}
-keep class com.google.protobuf.micro.*{
*;
}
-keep class com.xiaomi.channel.commonutils.android.**{
*;
}
-keep class com.xiaomi.channel.commonutils.file.**{
*;
}
-keep class com.xiaomi.channel.commonutils.network.**{
*;
}
-keep class com.xiaomi.channel.commonutils.misc.**{
*;
}
-keep class com.xiaomi.channel.commonutils.stats.**{
*;
}
-keep class com.xiaomi.channel.commonutils.string.**{
*;
}
-keep class com.xiaomi.mipush.sdk.**{
*;
}
-keep class com.xiaomi.mipush.network.**{
*;
}
-keep class com.xiaomi.mipush.push.**{
*;
}
-keep class com.xiaomi.mipush.smack.**{
*;
}
-keep class com.xiaomi.mipush.stats.**{
*;
}
-keep class com.xiaomi.xmpush.thrift.**{
*;
}
-keep class org.apache.thrift.**{
*;
```

```
}
-keep class org.apache.thrift.protocol.**{
*;
}
-keep class org.apache.thrift.meta_data.**{
*;
}
-keep class org.apache.thrift.transport.**{
*;
}
-keep class com.xiaomi.measite.smack.**{
*;
}
-keep class com.xiaomi.measite.smack.**{
*;
}
-keepclasseswithmembernames class com.xiaomi.**{*;}
-keep public class * extends com.xiaomi.mipush.sdk.PushMessageReceiver

-keep class alipay.yunpushcore.rpc.bind.pb.**{
*;
}
-keep class alipay.yunpushcore.rpc.bind.**{
*;
}
-keep class alipay.yunpushcore.rpc.report.**{
*;
}
-keep class alipay.yunpushcore.rpc.report.pb.**{
*;
}
-keep class com.alipay.pushsdk.content.**{
*;
}
-keep class com.alipay.pushsdk.data.**{
*;
}
-keep class com.alipay.pushsdk.deliver.**{
*;
}
-keep class com.alipay.pushsdk.exception.**{
*;
}
-keep class com.alipay.pushsdk.exception.**{
*;
}
-keep class com.alipay.**{
*;
}
-keep class com.hianalytics.**{
*;
}
-keep class com.huawei.**{
*;
}
```

相关链接

ProGuard 帮助手册

6 接入 iOS

消息推送服务使用苹果的 APNs 服务向客户端推送消息，请参见 推送流程 > 苹果及国外安卓设备 。

为了使用消息推送服务，您需要在客户端添加消息推送 SDK 并完成相关配置。

关于此任务

消息推送组件提供多种接入方式，您可以自由选择，请参见 接入方式简介 。下面将简述各种接入方式的接入流程。

基于 mPaaS 框架

1. 接入 mPaaS
   如果您的应用已接入其他 mPaaS 组件，那么这些流程已经完成。即要使用 mPaaS 组件，以下流程是必需且通用的：
   - 配置开发环境
   - 在控制台创建应用
   - 基于 mPaaS 框架接入
2. 接入消息推送组件
   - 使用 mPaaS 插件添加 SDK
   - 配置推送证书
   - 配置工程
   - 基于 mPaaS 框架使用 SDK

基于原生框架、使用 CocoaPods

目前 CocoaPods 只支持 消息推送 和 移动分析 组件。接入消息推送的流程如下：

1. 通用接入流程
   如果您的应用已接入其他 mPaaS 组件，那么这些流程已经完成。即要使用 mPaaS 组件，以下流程是必需且通用的：
   - 在控制台创建应用
   - 基于原生框架、使用 CocoaPods 创建工程
2. 接入消息推送组件
   - 使用 CocoaPods 添加 SDK
   - 配置推送证书
   - 配置工程
   - 基于原生框架使用 SDK

基于原生框架、使用 mPaaS 插件

1. 通用接入流程
   如果您的应用已接入其他 mPaaS 组件，那么这些流程已经完成。即要使用 mPaaS 组件，以下流程是必需且通用的：
   - 在控制台创建应用
   - 配置开发环境
   - 基于原生框架、使用 mPaaS 插件创建工程
2. 接入消息推送组件
   - 使用 mPaaS 插件添加 SDK
   - 配置推送证书
   - 配置工程
   - 基于原生框架使用 SDK

添加 SDK

根据 接入方式 的不同，添加 SDK 的方法分为 **使用 mPaaS 插件** 或 **CocoaPods** 2 种。

使用 mPaaS 插件添加 SDK

使用 Xcode 插件添加 Push 模块到工程中。

在当前工程的 Build Phases > Link Binary With Libraries 中，添加系统库 CoreMotion.framework。

使用 CocoaPods 添加 SDK

参考 使用 CocoaPods 接入 文档，确保在 Podfile 文件中添加如下依赖：

```
mPaaS_pod"mPaaS_Push","10.1.32"
```

在当前工程的 Build Phases > Link Binary With Libraries 中，添加系统库 CoreMotion.framework。

配置推送证书

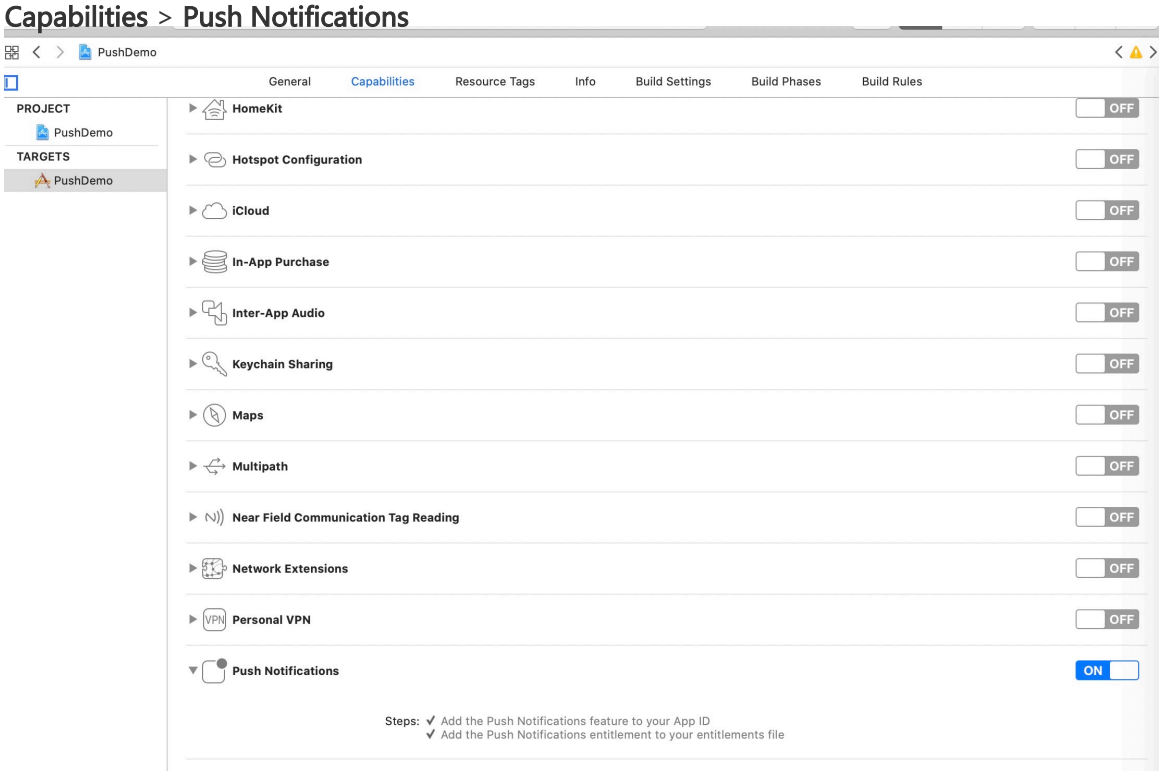
要使用 mPaaS 消息推送控制台推送消息，您需要在控制台中配置 APNs 推送证书。该证书必须是与客户端签名对应的推送证书，否则客户端会收不到推送消息。

有关详细的配置说明，查看 iOS 推送证书配置 。

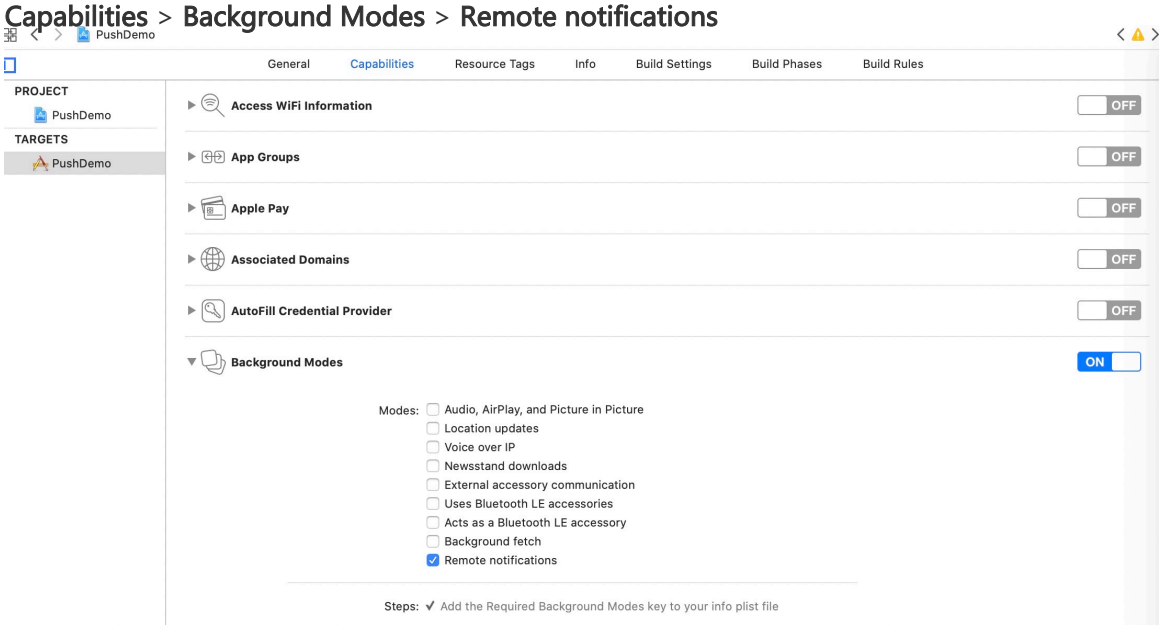
配置工程

需要在工程 target 设置中开启以下两处：

- Capabilities > Push Notifications



- Capabilities > Background Modes > Remote notifications



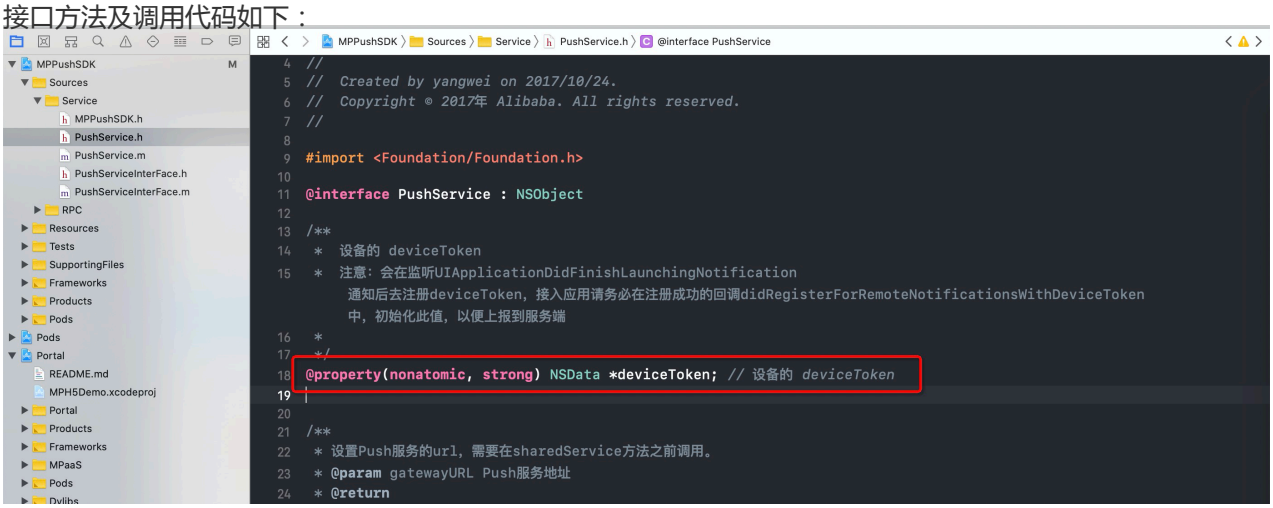
基于原生框架使用 SDK

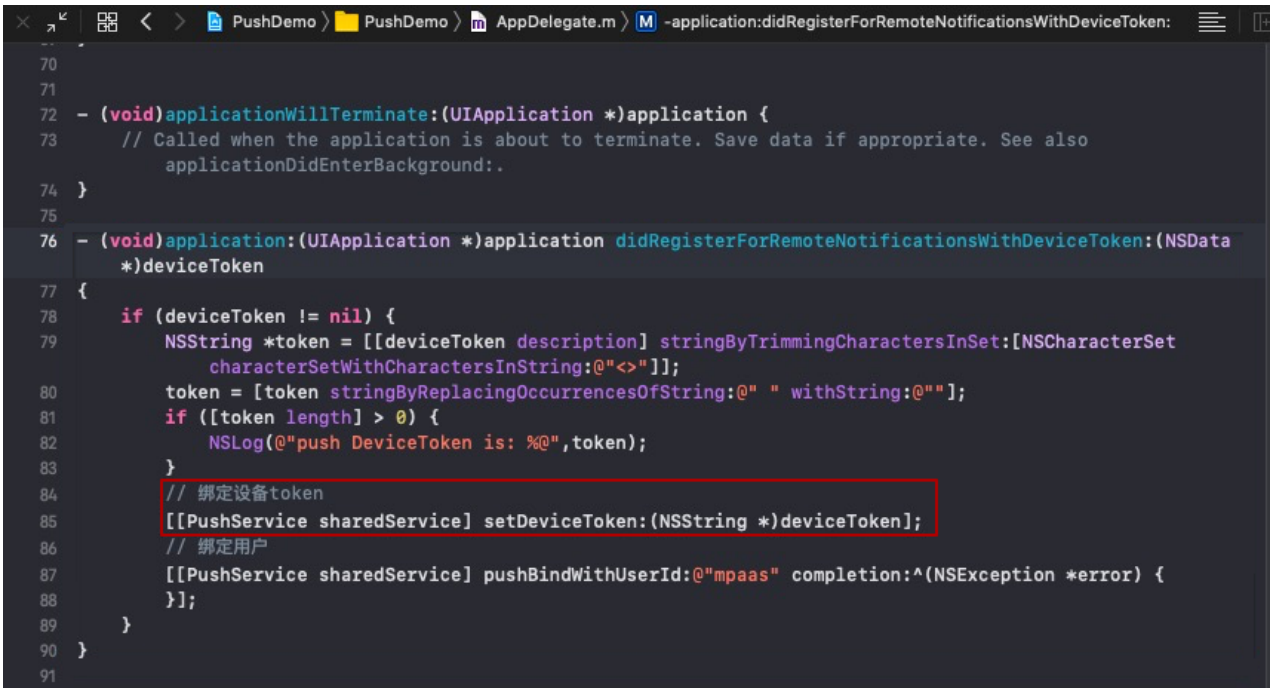
获取 DeviceToken

mPaaS 提供的 Push SDK 中封装了向 APNs 服务器注册的逻辑，在程序启动后，Push SDK 自动向 APNs 服

务器注册，您可在注册成功的回调方法中获取 APNs 下发的 DeviceToken，然后调用 PushService 的接口方法，将该 DeviceToken 转化为 64 位纯字符串（删除特殊字符和空格），并上报至移动推送核心。

接口方法及调用代码如下：

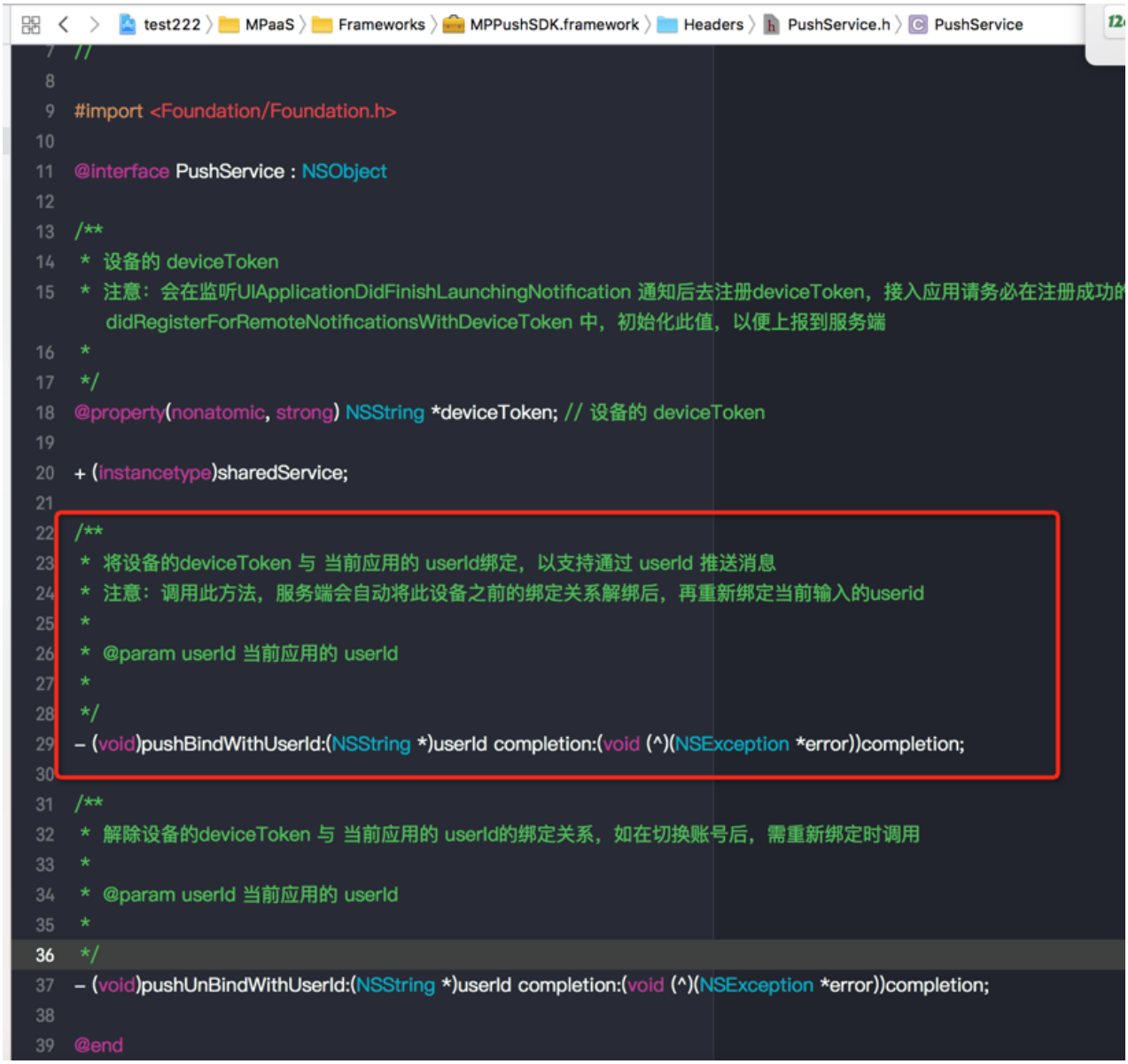




说明：MPS 支持通过调用推送接口和在控制台上创建消息的方式来推送消息，两种方式均使用转化后的 DeviceToken（64 位纯字符串）来指定推送目标设备。

上报 DeviceToken

除直接使用 DeviceToken 向客户端推送消息外，mPaaS push 服务支持通过当前应用的 userId 推送消息，需要客户端将设备的 DeviceToken 与当前应用的 userId 绑定，接口方法如下：



Push SDK 同时提供了解绑的接口，用于解除设备的 deviceToken 与当前应用的 userId 的绑定。如在用户切换账号后，可以调用解绑接口。

监听消息

客户端收到 push 消息后，如果用户点击查看，系统将根据证书启动相应应用，可在 AppDelegate 的回调方法中完成收到 push 消息后的逻辑处理。

- 在 iOS 10 以下系统中，通知栏消息或静默消息的处理方法如下：

```
// App 在前台时，普通推送的处理方法；App 在前台或后台时，静默推送的处理方法；iOS 10 以下系统，通知栏消息处理方法
- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult result))completionHandler
{
//处理接受到的消息
```

```
}
```

- 在 iOS 10 及以上系统中，您还需要实现以下代理方法来监听通知栏消息：

```
// iOS 10 通知栏消息
- (void)userNotificationCenter:(UNUserNotificationCenter *)center
didReceiveNotificationResponse:(UNNotificationResponse *)response
withCompletionHandler:(void(^)())completionHandler
{
// 处理接受到的消息
completionHandler();
}
```

统计消息的打开率

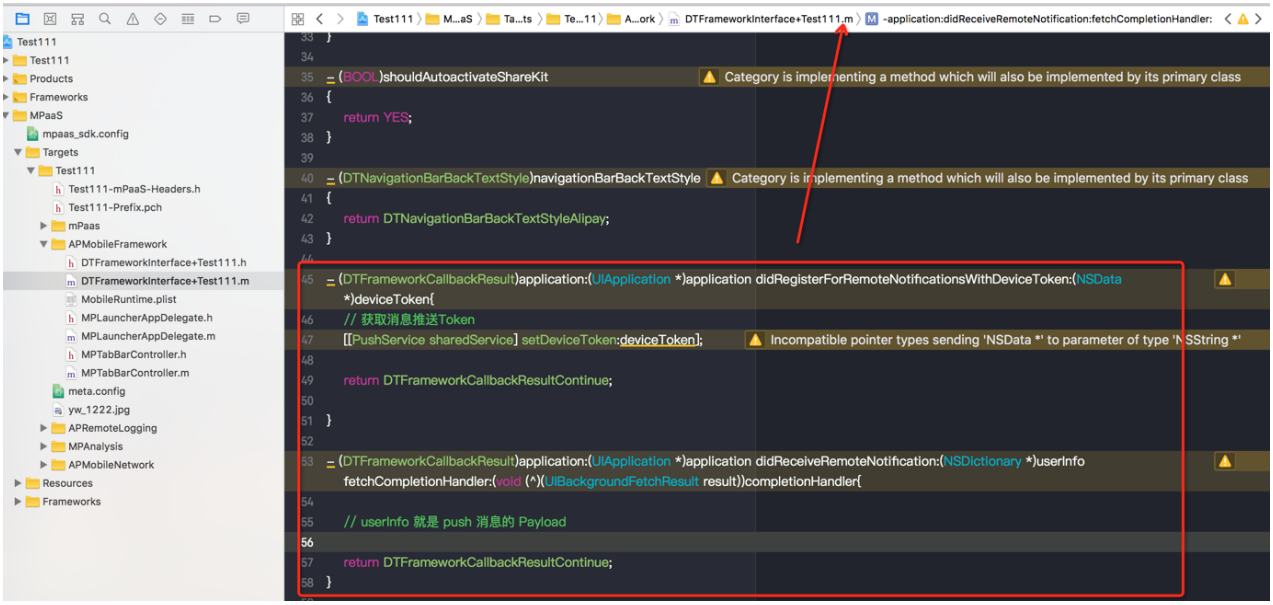
为了统计消息在客户端的打开率，您需要在 App 消息被用户打开时，调用 PushService 的 pushOpenLogReport 接口（10.1.32 及以上版本可用）上报消息打开事件。该事件上报后，您可以在 mPaaS 控制台中的 **后台服务管理 > 消息推送 > 使用分析** 中查看消息打开率的统计数据。

```
/**
* 打开推送消息的上报接口，用于统计推送消息的打开率
* @param userInfo 消息的 userInfo
* @return
*/
- (void)pushOpenLogReport:(NSDictionary *)userInfo;
```

基于 mPaaS 框架使用 SDK

获取 DeviceToken

基于 mPaaS iOS 框架的应用，由于其生命周期被 mPaaS 框架接管，与基于 iOS 原生框架相比，只是获取 DeviceToken 和收到消息的回调方法不同，Push SDK 相关接口调用逻辑一样。获取 DeviceToken 和收到消息的回调方法如下：



统计消息的打开率

统计消息打开率的操作方法和基于原生框架的操作方法完全相同。

后续操作

- 在 mPaaS 消息推送控制台配置完 APNs 证书后，可以按 **设备** 维度向应用推送消息。
- 上报用户 ID 并由服务端绑定用户和设备后，可以按 **用户** 维度向应用推送消息。

支持的推送类型如下：

- 极简推送
- 模板推送
- 批量推送
- 群发推送

关于推送类型的定义，查看 创建消息 。

相关链接

- 创建消息
- 配置服务端

7 配置服务端

你已经通过 推送流程 文档了解移动推送服务的总体流程。作为用户，您需要在自己的业务服务端配置验签、绑定用户和设备、推送消息。

前置条件

- 您已经开通 mPaaS 产品。
- 您已经有一个服务端应用。

- 您已经在客户端上报用户 ID 和 设备 ID。

操作步骤

1. 配置验签

由于 REST 调用跨越公网，为保证安全性，需要对调用的身份进行验证。调用者在进行 REST 调用时，需要根据规定的方式对参数进行加签。移动推送组件提供了两种签名方式，分别为 MD5 签名规则和 RSA 签名规则。配置验签的流程为：

1. 控制台上配置验签。具体内容，参考 密钥管理 。
2. 服务端对请求进行签名：
   RSA 签名规则如下，RSA 验签的公钥需要由业务方提供：
   - RSA 公钥长度为 2048 位
   - 使用 RSA 私钥对 POST 体进行签名，签名算法选择 SHA256 签名算法
   - 将签名结果转换成 base64 的字符串
   - 做 URL 安全的 base64 替换：将 、 + 替换为 - ，将 / 替换为 - 后，即为签名

2. 绑定用户和设备

服务端获取客户端上报的用户 ID 和 设备 ID 后，调用移动推送服务提供的 RESTful 接口来完成绑定。

接口文档参见 绑定 。

3. 推送消息

服务端可以通过调用 RESTful 接口，推送以下类型的消息：

- 推送简单消息 简单推送 。
- 使用模板推送消息 模板推送 。
- 针对不同目标推送不同消息 批量推送 。
- 针对全网用户推送消息 群发推送 。

8 管理控制台

8.1 消息列表

8.1.1 创建消息

关于此任务

您可通过 **消息列表** 页面，创建多种类型的消息，并基于不同维度（用户标识或设备标识）推送消息。MPS 支持的推送类型包括：

- 极简推送

主要用于对少数几个目标进行推送的场景，比如测试苹果推送证书的有效性，Andriod pushSDK 接入的正确性等。无需使用模板，在创建消息时，直接添加消息内容。推送时可以选择按照指定用户或指定设备推送。

模板推送
主要用于对多个目标进行多次推送的场景。可以在自动化或大范围使用模板功能之前，通过在控制台页面创建模板推送类型的消息进行模板功能的校验和测试。

当消息具有普适性，即针对多个用户、多次下发的场景下，可以从消息的标题、正文中提取出变量，配置成一个模板，采用模板进行推送。

批量推送
批量推送主要用于对大量目标（非全网）进行推送的场景，通常用来支持一些运营需求。

在创建批量推送类型的消息时，MPS 支持直接调用 MAS 人群作为推送目标，同时也支持通过上传文件的方式，指定推送目标。

- 调用 MAS 人群：支持对 MAS 人群推送相同消息，不支持消息的个性化推送。
- 手动上传人群：支持通过上传文件的方式来指定推送目标。您可基于消息使用的模板，在文件中为各推送 ID 配置不同的占位符内容，从而实现消息的个性化推送。

群发推送
群发推送用于进行全网推送的场景，对全网安卓或苹果设备每次推送相同模板消息，通常用来支持一些运营需求。

群发不支持用户标识的推送目标类型，只支持安卓设备和苹果设备两种类型。对安卓设备进行群发时，所有在消息有效期内建链的安卓设备都将收到群发消息；对苹果设备进行群发时，所有在消息有效期内处于绑定状态的设备都将收到群发消息。

说明：由于需要人工在页面上进行操作，故建议在系统验证、运营支持以及紧急临时需求等小频次推送场景时，通过控制台页面推送消息。

重要：消息一旦创建成功后即进行推送，您将无法删除或修改。

前置条件

- 在对 iOS 设备进行推送时，要先在 **推送配置** 页面配置好苹果设备的推送证书，操作参见 iOS 推送证书配置 。
- 创建批量推送、模板推送和群发推送消息之前，需要先创建好模板，操作参见 创建模板 。
- 创建批量推送时，若选择 MAS 人群作为推送目标人群，则需要先创建好 MAS 人群，操作参见 创建用户群组 。

操作方法

登录 mPaaS 控制台，选择目标应用，创建消息的操作方法如下：

1. 在左侧导航栏中，选择 **后台服务管理** > **消息推送**，进入 **消息推送** 页面。
2. 在右侧页面上，点击 **消息列**表 标签，进入 **消息列表** 标签页。
3. 点击 **新建推送消息** 按钮，页面上弹出 **新建推送消息** 对话框。

4. 点击对话框上方的页签，进入不同推送类型的消息创建页面，配置 基础信息 和 高级信息 。
   在高级信息配置区域，添加扩展参数操作方法如下：
   - 打开 **扩展参数** 的开关，**高级信息** 区域内展示 **扩展参数** 配置区域。
   - 点击 **增加参数** 按钮，在扩展参数列表中即增加一行 key 和 value 配置区域。

   - 配置 key 和 value 值，在页面任意区域处点击鼠标左键，完成配置。

说明：点击 删除，可删除对应的扩展参数。

5. 点击 **提交** 按钮，移动推送核心即推送消息至推送 ID。

不同推送类型以及不同推送平台，MPS 使用的推送渠道不同，具体如下：

- 极简推送、模板推送和批量推送
  - Android 推送平台
    - 当消息类型为 **透传消息** 或 **红点消息**（配置了扩展参数 badge）时，使用自建渠道推送消息。
    - 其他场景下，当接入三方推送平台（小米、华为、OPPO、vivo、FCM 等）且推送目标设备为相应厂商的机型时，使用三方渠道推送消息，否则，使用自建渠道推送消息。
  - iOS 推送平台
    使用三方渠道推送消息。
- 群发推送
  - Android 平台：使用自建渠道推送消息。
  - iOS 平台：使用三方渠道推送消息。

配置项说明

在 **新建推送消息** 对话框内创建消息，对话框分 3 个区域：基础信息配置区域、高级信息配置区域以及推送预览区域。您需要在前两个区域内配置推送任务的参数，并在预览区域内实时查看消息展示效果。

基础信息

配置推送的基础信息，包括消息标题、正文、推送 ID等。不同推送类型的消息，其配置项有所不同。

极简推送

需配置的基础信息参数说明见下表。

参数	是否必填	说明
目标 ID 类型	是	选择消息下发模式，可选： • **UserId**：基于用户维度推送消息。需要调用绑定接口，绑定用户标识和设备标识，绑定接口说明参见 API 参考 。 • **DeviceId**：基于设备维度推送消息。
推送平台	是	基于设备维度推送消息时，需要选择推送平台，明确推送设备类型。可选： • **Android**：推送目标设备为 Android 手机。 • **iOS**：推送目标设备为苹果手机。
业务方消息 ID	是	系统自动生成，用于在业务方系统中唯一标识消息。支持自定义，最多可输入 64 个字符。
目标 ID	是	填写用户标识和设备标识。 • 需要根据选择的 **目标 ID 类型** 进行填写，否则将导致推送失败。 • 当目标类型 ID 类型为 **DeviceId** 时，**目标 ID** 为 **设备标识**（Android 设备填写 Ad-token，iOS 设备填写 Device Token），其所对应的手机操作系统应于所选 **推送平台** 一致，否则将导致推送失败。 • 若通过日志等途径获取的设备标识包含空格，您需要删除其中的空格。
消息类型	是	可选： • **通知栏消息**：使用三方渠道推送消息。未接入三方渠道或已接入但推送设备非三方渠道机型时，使用自建渠道推送消息。 • **透传消息**：使用自建渠道推送消息。 对于 Android 推送平台，本参数为自建渠道和三方渠道推送的选择入口。对于 iOS 推送平台，您无需配置本参数（iOS 推送为三方渠道推送）。
展示类型	是	可选： • **展示消息**：指需要展示的消息。 • **静默消息**：指无感知消息，即在目标设备上不以任何形式展示的消息。**消息类型** 为 **透传消息** 时，本选项可选。 对于 Android 推送平台，您需要根据不同的推送渠道，执行不同后续操作： • **自建渠道**：本参数作为参考字段发送至客户端，您需要解析消息体，在获取本字段内容后，自行控制消息的展示。 • **三方渠道**：本参数作为字段发送至目标设备后，由厂商系统解析字段内容并控制消息的展示，您无需执行其他操作。

		对于 iOS 推送平台，消息的展示为厂商系统行为，您无需执行其他操作。
点击后操作	是	选择在手机上点击消息内容后的操作。本参数仅作为参考字段发送至客户端，您需要参考字段内容，自行实现后续操作。 可选： • **打开 Intent Activity**：点击消息后，页面跳转至原生页面。 • **打开 Web URL**：点击消息后，页面跳转至网页。
推送标题	是	填写消息的标题，最多可输入 200 字符。在 **新建推送消息** 文本框右侧的预览区域，可预览消息下发后的展示效果。
推送文本	是	填写消息的文本内容，最多可输入 200 字符。在 **新建推送消息** 文本框右侧的预览区域，可预览消息下发后的展示效果。

模板推送

需配置的基础信息参数说明见下表。

参数	是否必填	说明
目标 ID 类型	是	选择消息下发模式，可选： • **UserId**：基于用户维度推送消息。需要调用绑定接口，绑定用户标识和设备标识，绑定接口说明参见 API 参考 。 • **DeviceId**：基于设备维度推送消息。
推送平台	是	基于设备维度推送消息时，需要选择推送平台，明确推送设备类型。可选： • **Android**：推送目标设备为 Android 手机。 • **iOS**：推送目标设备为苹果手机。
业务方消息 ID	是	系统自动生成，用于在业务方系统中唯一标识消息。支持自定义，最多可输入 64 个字符。
目标 ID	是	填写用户标识或设备标识。 • 需要根据选择的 **目标 ID 类型** 进行填写，否则将导致推送失败。 • 当目标类型 ID 类型为 **DeviceId** 时，**目标 ID** 为 **设备标识**（Android 设备填写 Ad-token，iOS 设备填写 Device Token），其所对应的手机操作系统应于所选 **推送平台** 一致，否则将导致推送失败。 • 若通过日志等途径获取的设备标识包含空格，您需要删除其中的空格。
推送模板	是	选择消息模板，可选 **消息模板** 页面上列表中的所有模板。
消息类型	是	系统根据所选消息模板的展示类型，提供不同选项： • 模板展示类型为 **展示消息** 时：提供 **通知栏消息** 和 **透传消息** 选项。 • 模板展示类型为 **静默消息** 时：系统默认选择 **透传消息**，您无法修改。 各选项说明如下： • **通知栏消息**：使用三方渠道推送消息。未接入三方渠道或已接入但推送设备非三方渠道机型时，使用自建渠道推送消息。

		• **透传消息**：使用自建渠道推送消息。 对于 Android 推送平台，本参数为自建渠道和三方渠道推送的选择入口。对于 iOS 推送平台，您无需配置本参数（iOS 推送为三方渠道推送）。
展示类型	是	系统自动选择所选模板中的展示类型，您无法修改。 • **展示消息**：指需要展示的消息。 • **静默消息**：指无感知消息，即在目标设备上不以任何形式展示的消息。 对于 Android 推送平台，您需要根据不同的推送渠道，执行不同后续操作： • **自建渠道**：本参数作为参考字段发送至客户端，您需要解析消息体，在获取本字段内容后，自行控制消息的展示。 • **三方渠道**：本参数作为字段发送至目标设备后，由厂商系统解析字段内容并控制消息的展示，您无需执行其他操作。 对于 iOS 推送平台，消息的展示为厂商系统行为，您无需执行其他操作。
点击后操作	是	系统根据所选模板中的配置自动进行选择，您无法修改。
模板占位符	是	填写模板中的变量值。系统根据所选模板中的占位符，提供配置入口。

批量推送

需配置的基础信息参数说明见下表。

参数	是否必填	说明
目标 ID 类型	是	选择消息下发模式，可选： • **UserId**：基于用户维度推送消息。需要调用绑定接口，绑定用户标识和设备标识，绑定接口说明参见 API 参考 。 • **DeviceId**：基于设备维度推送消息。
推送平台	是	基于设备维度推送消息时，需要选择推送平台，明确推送设备类型。可选： • **Android**：推送目标设备为 Android 手机。 • **iOS**：推送目标设备为苹果手机。
推送模板	是	选择消息模板，可选 **消息模板** 页面上列表中的所有模板。
选择推送人群	是	选择推送目标人群，可选： • **手动上传人群** • 手动上传推送目标文件，文件内包含了推送目标标识以及针对所选模板对各推送目标的个性化配置。文件内一条数据代表一条消息，每条消息使用业务方消息 ID 进行标识。文件格式要求如下： 每条数据格式为：推送 ID,业务方消息 ID,占位符1= XXX,占位符2=XXX……。文件编码类型要求为 UTF-8，文件大小上限为 200 MB，多条数据之间使用换行符分隔，每条数据不要超过 250 字符。一

		个推送任务中最多可上传 1 个文件。文件上传成功后，**手动上传人群** 按钮下方将显示已上传文件的图标，点击图表，可对文件中的内容进行预览，最多可预览 10 条数据。 • **移动分析人群** • 支持调用 MAS 人群，您需要先创建 MAS 用户群组，操作方法参见 创建用户群组 。 支持对 MAS 人群推送相同消息，所选推送模板中包含占位符时，移动分析人群 将不可选。无所选目标 ID 类型的 MAS 人群时，**移动分析人群** 将不可选。
消息类型	是	系统根据所选消息模板的展示类型，提供不同选项： • 模板展示类型为 **展示消息** 时：提供 **通知栏消息** 和 **透传消息** 选项。 • 模板展示类型为 **静默消息** 时：系统默认选择 **透传消息**，您无法修改。 各选项说明如下： • **通知栏消息**：使用三方渠道推送消息。未接入三方渠道或已接入但推送设备非三方渠道机型时，使用自建渠道推送消息。 • **透传消息**：使用自建渠道推送消息。 对于 Android 推送平台，本参数为自建渠道和三方渠道推送的选择入口。对于 iOS 推送平台，您无需配置本参数（iOS 推送为三方渠道推送）。
展示类型	是	系统自动选择所选模板中的展示类型，您无法修改。 • **展示消息**：指需要展示的消息。 • **静默消息**：指无感知消息，即在目标设备上不以任何形式展示的消息。 对于 Android 推送平台，您需要根据不同的推送渠道，执行不同后续操作： • **自建渠道**：本参数作为参考字段发送至客户端，您需要解析消息体，在获取本字段内容后，自行控制消息的展示。 • **三方渠道**：本参数作为字段发送至目标设备后，由厂商系统解析字段内容并控制消息的展示，您无需执行其他操作。 对于 iOS 推送平台，消息的展示为厂商系统行为，您无需执行其他操作。
点击后操作	是	系统根据所选模板中的配置自动进行选择，您无法修改。

说明：推送目标人群中，以下不符合要求的推送目标 ID 将无法收到消息：

- 非所选目标 ID 类型的 ID。
- 对应手机操作系统非所选推送平台的的 ID。

群发推送

需配置的基础信息参数说明见下表。

参数	是否必填	说明
推送平台	是	选择推送平台，明确推送设备类型。可选： • **Android**：使用自建渠道，对全网（在消息有效期内）在线的 Android 设备推送消息，对每个设备仅推送一次，不重复推送。 • **iOS**：使用三方渠道，对全网 iOS 客户端用户（当前登录或者历史登录过且当前登出的用户

		）推送消息，对每个用户仅推送一次，不重复推送。
业务方消息 ID	是	系统自动生成，用于在业务方系统中唯一标识消息。支持自定义，最多可输入 64 个字符。
推送模板	是	选择消息模板，可选 **消息模板** 页面上列表中的所有模板。
消息类型	是	群发推送时，由于 Android 推送平台采用自建渠道推送，故本参数无特殊作用，您无需配置。
展示类型	是	系统自动选择所选模板中的展示类型，您无法修改。 • **展示消息**：指需要展示的消息。 • **静默消息**：指无感知消息，即在目标设备上不以任何形式展示的消息。 对于不同的推送平台，您需要为消息的展示执行不同的操作： • Android 推送平台：本参数作为参考字段发送指客户端，您需要解析消息体，在获取本字段内容后，自行控制消息的展示。 • iOS 推送平台：消息的展示为厂商系统行为，您无需执行其他操作。
点击后操作	是	系统根据所选模板中的配置自动进行选择，您无法修改。
模板占位符	是	填写模板中的变量值。系统根据所选模板中的占位符，提供配置入口。

高级信息

配置推送任务的更多信息，包括：

- **跳转地址**：在手机上点击消息后访问的页面。本参数仅作为参考字段下发至客户端，仅供您参考使用，但并不生效。您需要自行实现跳转逻辑。
  根据 **点击后操作** 填写不同内容：
  - **打开 Intent Activity**：填写需要访问的原生页面地址（Android：ActivityName；iOS：VCName）。
  - **打开 Web URL**：填写需要访问的网页地址。

说明：创建模板推送、批量推送以及群发推送消息时，**跳转地址** 在所选模板中配置，高级信息配置区域中无本参数的配置入口。

- **消息有效期**：设置消息的有效期，单位为秒。由于设备未在线或者用户登出导致消息下发失败时，在消息有效期内，设备建链或发起用户绑定请求后，MPS 将重新下发消息，确保消息触达率。

  扩展参数：会跟随消息体到达客户端，供用户自定义处理。
  扩展参数包含以下 3 类：
  - 系统扩展参数
    这些扩展参数被系统占用，注意不要修改此类参数的 value 值。系统扩展参数包括：
    - notifyType
    - action
    - silent
    - pushType

- templateCode
- channel
- taskId

系统具有一定意义的扩展参数
这些扩展参数被系统占用，且具有一定的意义。您可以配置此类扩展参数的 value 值，但配置结果不会随消息体中的扩展参数到达客户端。系统具有一定意义的扩展参数及其说明参见下表。

key	说明
min_version	客户端最小版本，配置 value 后，即指定最小版本，MPS推送消息时将检查应用的版本号，仅对的大于最小版本的应用推送消息。
max_version	客户端最大版本，配置 value 后，即指定最大版本，MPS推送消息时将检查应用的版本号，仅对的小于最大版本的应用推送消息。
sound	自定义铃声，value 配置为铃声的路径，本参数仅对小米和苹果手机有效。
badge	应用图标角标，value 配置为具体数值。本扩展参数会跟随消息体到达客户端。 • 对于 Android 手机，您需要自行处理角标的实现逻辑。 • 对于苹果手机，手机系统将自动实现角标。消息推送至目标手机后，应用图标的角标即会显示为 value 中配置的数值。
mutable-content	APNs 自定义推送标识，推送的时候携带本参数即表示支持 iOS10 的 UNNotificationServiceExtension；若不携带本参数，则为普通推送。Value 配置为 1。
Icon	FCM 消息（依赖 FCM 服务推送的消息）的图标，Value 配置为图标的路径。
fcm	针对国外安卓用户进行群发时，需要配置扩展参数 fcm = ture，否则即为对国内安卓用户的群发。

用户自定义扩展参数
除了系统扩展参数和系统具有一定意义的扩展参数，其他的参数 key 都属于用户扩展参数。用户自定义扩展参数会随消息体中的扩展参数到达客户端，供用户自定义处理。

推送登录状态：选择 iOS 平台群发推送的目标用户类型。

- **登录用户**：对全网登录的用户推送消息。
- **登录用户+登出用户**：对全网登录，或者历史登录过且当前登出的用户推送消息。

- **退出登录时长**：选择 **推送登录状态** 为 **登录用户+登出用户** 时，需要设置本参数，用于圈定推送的登出用户人群。可选：
  - **7 天**：对在 7 天内登出的用户推送消息。
  - **15 天**：对在 15 天内登出的用户推送消息。
  - **60 天**：对在 60 天内登出的用户推送消息。
  - **永久**：对所有登出的用户推送消息。

推送预览

新建推送消息 对话框右侧区域为 **推送预览** 区域。点击 **通知**、**苹果消息体**、**安卓消息体**，可分别预览消息的展

示效果以及下发至不同平台的消息体。

消息内容由替换模板占位符得到时，预览中 **#占位符名称#** 将根据对应的 **模板占位符** 内容进行替换，方便您验证消息配置是否正确。

8.1.2 管理消息

消息列表中展示了最近 30 天内创建的 **极简推送** 和 **模板推送** 消息相关信息，您可通过消息列表，定位至目标消息，查看其推送详情。

查看消息列表

登录 mPaaS 控制台，选择 App，通过以下步骤，查看消息列表：

1. 在左侧导航栏中，选择 **后台服务管理** > **消息推送**，进入 **消息推送** 页面。
2. 在右侧页面上，点击 **消息列表** 标签，进入 **消息列表** 标签页。

消息列表以消息的创建时间进行倒序排列，列表中展示的信息包括：

- 业务方消息 ID

  推送维度：展示消息的推送维度。

   ：基于用户维度

   ：基于设备维度，且推送平台为 Android

   ：基于设备维度，且推送平台为 iOS

- 推送目标 ID
- 推送标题：消息的标题。
- 消息创建时间：消息创建成功的时间，精确到秒。

查看推送详情

在列表中点击目标消息的 **展开** 按钮（ ），可查看相应消息的推送详情。

包括：

- 消息 ID：由系统自动生成，为 MPS 对消息的唯一标识，用于唯一标识一条消息。

  消息推送状态：显示消息的推送状态，其中常见的状态及其含义详见下表。

状态码	含义
DeviceNotOnlineOrNoResponse	等待设备上线（推送目标设备与移动推送网关长链接断开）或正在发送流程中。
NoBindInfo	无绑定关系。基于用户标识维度推送消息时，确认推送目标（userId）已绑定设备标识。
Acked	使用自建渠道推送消息时，表示消息已成功推送至客户端；使用三方渠道推送消息时，表示已成功调用三方推送网关。
ConnClosed	本状态仅出现在对 iOS 设备推送的消息中，产生本状态的原因如下： • 在控制台上配置的苹果推送证书环境与推送的 Device Token 不匹配。

	• 在 App 安装包中打包的证书和在控制台上配置的证书不匹配。 • 工程中的 BundleId 和 在控制台上配置的 BundleId 不一致。 关于在控制台上配置 iOS 推送证书、证书环境以及 BundleId 的详细操作，参见 iOS 推送证书配置 。
BadDeviceToken	设备标识无效、格式错误或不存在。当基于用户维度推送消息，且出现本状态时，您需要检查在绑定时所使用的设备标识是否正确。建议在绑定完成后，在消息推送控制台上创建极简推送类型的消息进行测试。
DeviceTokenNotForTopic	客户端的 Bundle ID 和配置 iOS 推送证书时填写的 Bundle ID 不一致。

消息过期时间：指消息的过期时间，由系统根据消息的创建时间和消息有效期自动计算得到。在消息未推送成功之前（推送状态为 **DeviceNotOnlineOrNoResponse / NoBindInfo** ），目标设备建链或用户发起绑定请求时，MPS 将下发消息。一旦消息过期后，MPS 将不再下发消息。

- 消息变更时间：消息推送状态变更的时间，精确到秒。

搜索消息

消息列表支持根据 **设备标识/用户标识** 和 **业务方消息 ID** 搜索消息，操作方法如下：

1. 在 **消息列表** 页面右上方的搜索框中，输入完整的设备标识/用户标识和业务方消息 ID。
2. 点击 **搜索** 按钮（  ），或按回车键，列表中将只显示相应推送目标 ID和业务方消息 ID 的消息。

说明：

- 仅支持查询最近 30 天内创建的 **极简推送**、**模板推送** 或 **批量推送** 类型的消息。
- 列表中默认不展示批量推送类型的消息，您可通过搜索操作查看。
- 列表中不展示群发推送类型的消息，且您无法通过搜索操作查看。

8.2 消息模板

8.2.1 创建模板

关于此任务

模板功能可以增加消息配置的灵活性，减少重复内容的传输。

消息模板由模板主体、占位符，以及其它一些消息属性组成。占位符为模板中的动态化内容，不包含占位符的模板将不具有个性化消息推送的能力。

通过 **#占位符名称#** 的写法来标识模板中的动态化部分，占位符可以出现在 **模板标题**、**模板正文** 和 **跳转地址** 中。

操作方法

操作方法如下：

1. 登录 mPaaS 控制台，在应用列表页面选择目标应用，进入目标应用的控制台操作页面。
2. 在左侧导航栏中，选择 **后台服务管理** > **消息推送**，进入 **消息推送** 页面。
3. 在右侧页面上，点击 **消息模板** 标签，进入 **消息模板** 标签页。
4. 点击 **创建模板** 按钮，页面上弹出 **创建模板** 对话框。
5. 配置模板信息，参数说明见下表。

参数	是否必填	说明
模板名称	是	模板的名称，最多可输入 200 字符，可为字母、数字、下划线的组合。不能与已有名称重复，将作为模板的唯一标识用于 API 调用中。
模板描述	是	模板的描述，可输入字母、数字以及下划线，最多可输入 200 字符。
模板标题	是	模板的标题，最多可输入 200 字符。
模板正文	是	模板的正文，最多可输入 200 字符。
点击后操作	是	本参数作为参考字段下发至客户端，供您参考使用，具体实现逻辑需要您自行处理。可选： • **打开 Internet Activity**：在手机上点击消息，访原生页面。 • **打开 Web URL**：在手机上点击消息，访问网页。
跳转地址	否	本参数作为参考字段下发至客户端，供您参考使用，具体实现逻辑需要您自行处理。 根据 **点击后操作**，填写不同的内容： • 打开 Internet Activity：填写需要访问的原生页面地址（Android：ActivityName；iOS：VCName）。 • 打开 Web URL：填写需要访问的网页 URL。
展示类型	是	可选： • **展示消息**：指需要在手机上展示消息，消息的展示样式由手机系统决定，您无法自定义消息的展示样式。 • **静默消息**：指无感知消息，即在手机上不以任何形式展示消息。

6. 点击 **创建** 按钮，创建消息。创建成功后，页面将返回 **消息模板** 页面，最新创建的模板将显示在列表最上方。

8.2.2 管理模板

模板列表中展示了已成功创建的消息模板信息，您可通过模板列表，定位至目标模板，查看或删除模板。

查看模板列表

登录 mPaaS 控制台，选择 App，通过以下步骤，查看模板列表：

1. 在左侧导航栏中，选择 **后台服务管理** > **消息推送**，进入 **消息推送** 页面。
2. 在右侧页面上，点击 **消息模板** 标签，进入 **消息模板** 标签页。

模板名称	模板描述	模板正文	创建日期	操作
文档测试03	文档测试使用	#用户名#您好，有您的消息，请注意查收	2019/7/24下午7:17	删除
文档测试02	文档测试使用---静默消息	#正文测试02#	2019/7/24下午6:02	删除
文档测试	文档测试使用	#测试正文#	2019/7/24下午6:01	删除
#s1#模板	#s1#模板	#s1#模板	2019/6/26上午10:44	删除
18	18	18	2019/4/28下午4:10	删除
17	17	17	2019/4/28下午4:10	删除
16	16	16	2019/4/28下午4:10	删除
15	15	15	2019/4/28下午4:00	删除
14	14	14	2019/4/28下午4:00	删除
13	13	13	2019/4/28下午3:55	删除

< 1 2 3 >

模板列表根据模板的创建日期进行倒序排列，列表中展示的信息包括：

- 模板名称
- 模板描述
- 模板正文
- 创建日期
- 操作：点击 **删除**，可删除对应模板。详细操作参见 删除模板 。

删除模板

操作方法如下：

1. 在列表中，点击与目标模板对应的 **操作** 列的 **删除**。
2. 在弹出的提示框中，点击 **确定**，删除模板。

重要：确保目标模板未被待推送的消息使用，否则将导致相应消息推送失败。

8.3 渠道配置

关于此任务

为提升推送的到达率，mPaaS 集成了华为、小米等厂商的推送功能。采用 **小米通知栏消息** 和 **华为通知栏消息** 实现消息推送。在应用未运行时，依然可以发送通知，用户点击通知栏即可激活进程。

说明：接入厂商自有的推送渠道后，能够帮助应用获得稳定的推送性能，因此建议您将第三方推送渠道接入您的应用。

本文将引导您完成在接入华为和小米推送渠道时需要进行的控制台侧配置。

前置条件

您需要先完成客户端侧的接入配置，操作参见：

- 版本 ≥ 10.1.32 > 接入第三方推送渠道
- 版本 < 10.1.32 > 接入第三方推送渠道

操作方法

华为推送渠道配置

登录 mPaaS 控制台，选择 App，通过以下步骤，配置华为推送渠道：

1. 在左侧导航栏中，选择 **后台服务管理** > **消息推送**，进入 **消息推送** 页面。
2. 在右侧页面上，点击 **渠道配置** 标签，进入 **渠道配置** 标签页。

点击 **华为推送渠道** 配置区域右上角的 **配置**，页面上展示配置入口，如下图所示。

参数	是否必填	说明
状态	是	渠道的接入状态开关。打开开关，MPS 将根据配置接入华为推送渠道；关闭开关，即取消接入。
包名	是	输入华为应用包名。
华为应用 ID	是	输入华为应用的 App ID。
华为应用密码	是	输入华为应用的密钥（App Secret）。

说明：应用包名、应用 App ID、密钥可从 **华为开发者联盟** > **管理中心** > **我的产品** > **移动应用详情** 中获取。

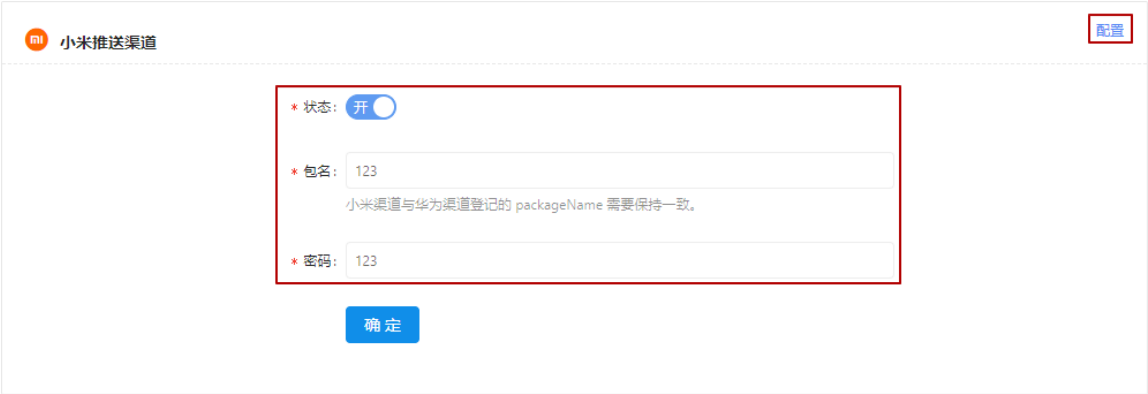
4. 点击 **确定** 按钮，保存配置。

小米推送渠道配置

登录 mPaaS 控制台，选择 App，通过以下步骤，配置小米推送渠道：

1. 在左侧导航栏中，选择 **后台服务管理** > **消息推送**，进入 **消息推送** 页面。
2. 在右侧页面上，点击 **渠道配置** 标签，进入 **渠道配置** 标签页。

点击 **小米推送渠道** 配置区域右上角的 **配置**，页面上展示配置入口，如下图所示。



参数	是否必填	说明
状态	是	渠道的接入状态开关。打开开关，MPS 将根据配置接入小米推送渠道；关闭开关，即取消接入。
包名	是	输入小米应用的主包名。
密码	是	输入小米应用的密钥（AppSecret）。

说明：主包名和密钥可从 **小米开放平台** > **应用管理** > **应用信息** 中获取。

4. 点击 **确定** 按钮，保存配置。

8.4 推送配置

关于此任务

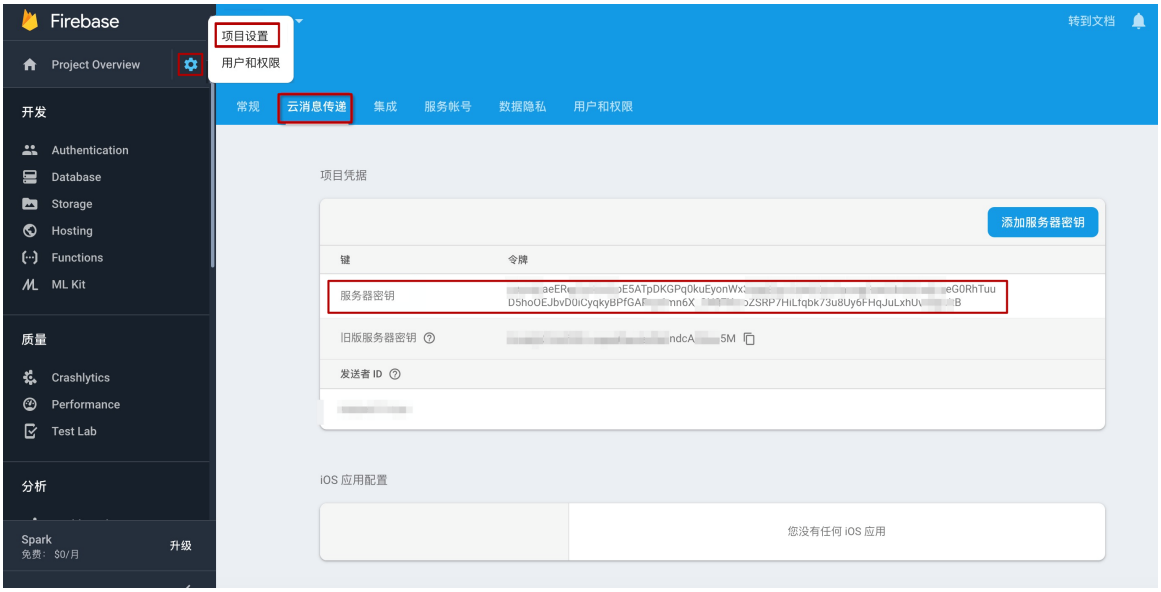
推送配置包括 **FCM 推送渠道配置** 和 **iOS 推送证书配置**。

FCM 推送渠道配置
接入国外安卓设备时，依赖谷歌的 FCM 服务作为消息推送网关，需要在控制台侧配置 FCM 推送渠道。

iOS 推送证书配置
接入苹果手机时，依赖 APNs 服务作为消息推送网关，需要在控制台侧上传 iOS 推送证书，用于连接 APNs 服务。

前置条件

- 进行 FCM 推送渠道配置时，您需要先在 Firebase 控制台上获取 FCM 服务器密钥，获取方法如下图所示。



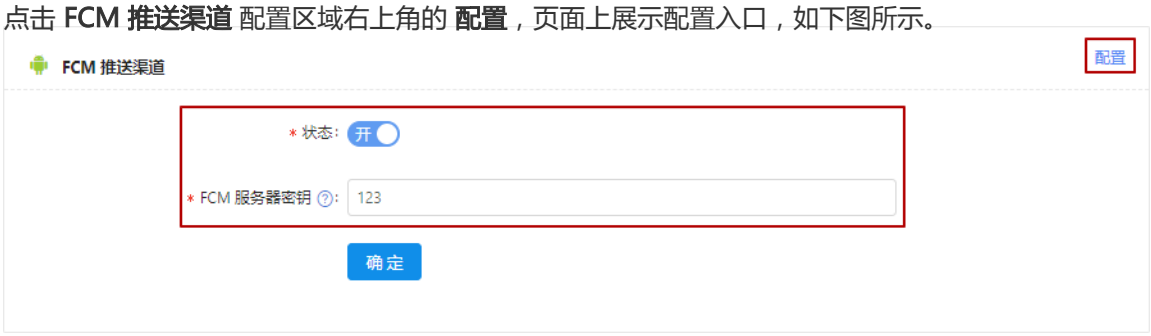
- 进行 iOS 推送证书配置时，您需要先 制作 iOS 推送证书 。

操作方法

FCM 推送渠道配置

登录 mPaaS 控制台，选择 App，通过以下步骤，配置 FCM 推送渠道：

1. 在左侧导航栏中，选择 **后台服务管理** > **消息推送**，进入 **消息推送** 页面。
2. 在右侧页面上，点击 **推送配置** 标签，进入 **推送配置** 标签页。
3. 点击 **FCM 推送渠道** 配置区域右上角的 **配置**，页面上展示配置入口，如下图所示。



   - 点击 状态 开关，打开开关时，MPS 将接入 FCM 服务；关闭开关时，MPS 不接入 FCM 服务。
   - 填写 **FCM 服务器密钥**，确保填写的是 server（服务器）的密钥，Android 密钥、iOS 密钥和浏览器密钥会被 FCM 拒绝。
4. 点击 **确定** 按钮，保存配置。

iOS 推送证书配置

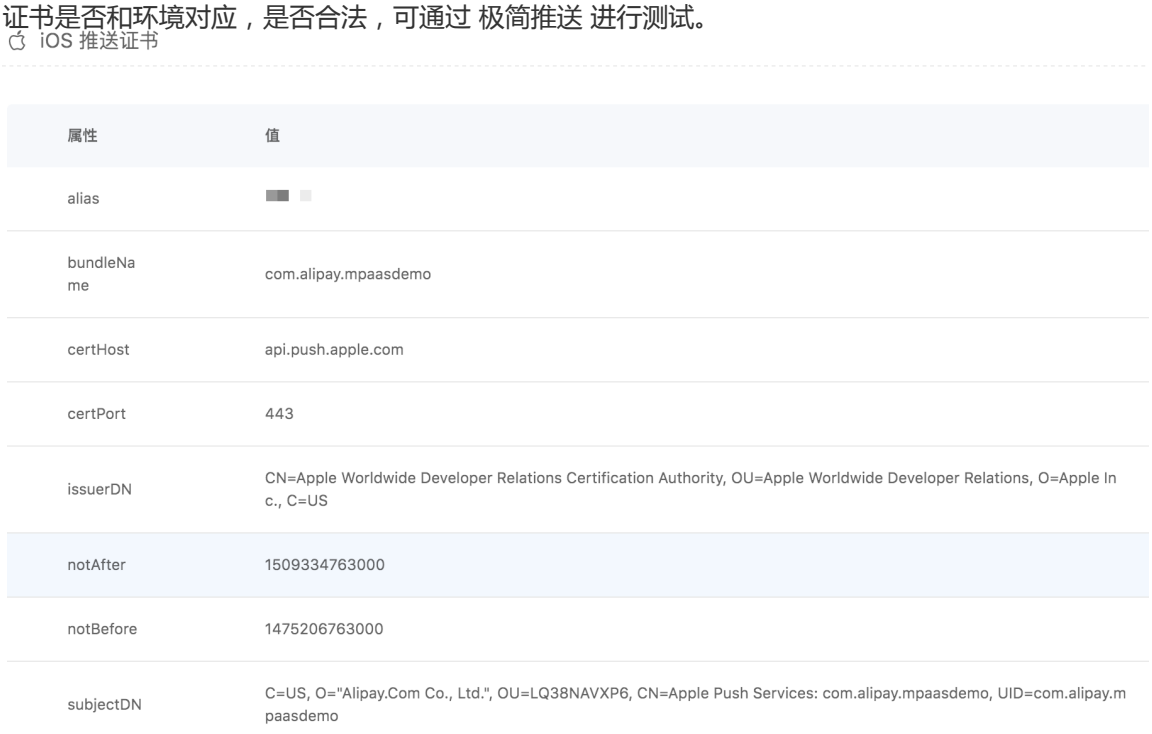
登录 mPaaS 控制台，选择 App，通过以下步骤，配置 iOS 推送证书：

1. 在左侧导航栏中，选择 **后台服务管理** > **消息推送**，进入 **消息推送** 页面。
2. 在右侧页面上，点击 **推送配置** 标签，进入 **推送配置** 标签页。

3. 在 **iOS 推送证书** 配置区域，配置 **iOS 推送证书**，各配置项说明参见下表。

参数	说明
选择证书文件	点击 **点击上传** 按钮，选择预先准备好的 iOS 推送证书并上传。 需根据所选 **证书环境**，上传相应环境的推送证书，否则，您将无法对 iOS 设备推送消息。不同环境证书的区别，参见 证书类型 。
证书密码	填写证书密码。
证书环境	可选 **APNs 2.0 生产环境** 和 **APNs 2.0 开发环境**。 移动推送服务在对 iOS 设备下发消息时，会根据证书所对应的环境选择对应的 APNs 网关服务器。请根据您实际业务需求，选择正确的证书环境。
Bundle Id	应用的全球唯一标识，在苹果开发平台的 **App IDs** 页面上查看您应用的 BundleId。
主机	APNs 服务器的地址，系统根据 **证书环境** 自动选择，您无法修改。
端口	APNs 服务器的端口，系统根据 **证书环境** 自动选择，您无法修改。

4. 点击 **上传** 按钮，保存配置，若证书格式正确，可以看到证书的详细内容，如下图所示。若需要验证证书是否和环境对应，是否合法，可通过 极简推送 进行测试。



8.5 密钥管理

关于此任务

为了提高 MPS 与用户系统之间的交互的安全性，MPS 会对所有接口进行加签与验证，并且提供了密钥管理界面，供您进行密钥配置。

推送 API 接口配置
MPS 提供 REST 接口供您调用。为确保安全性，MPS 需要对调用者的身份进行验证。在调用 API 之

前，您需要使用 RSA 算法，对请求进行签名，并在消息推送的控制台的 **密钥管理** 界面上的 **推送 API 接口配置** 区域内，配置密钥，供 MPS 验证调用者身份所用。

推送回调接口配置
若您需要获取消息下发结果的回执，则需要在 控制台 **密钥管理** 页面上的 **推送回调接口配置** 区域中，配置 MPS 回调时用到的 REST 接口地址，并获取公钥。MPS 在回调用户接口时，会对请求参数进行签名。您需要使用获取的公钥，对请求进行验签，以验证是否为 MPS 回调。

前置条件

进行推送 API 接口配置时，您需要先使用 RSA 算法生成 2048 位公钥，算法签名规则如下：

- 使用 SHA 256 签名算法。
- 将签名结果转换成 base64 字符串。
- 将 base64 字符串中的 + 替换为 - ，/ 替换为 - ，得到最终签名结果。

操作方法

推送 API 接口配置

登录 mPaaS 控制台，选择 App，通过以下步骤，配置推送接口：

1. 在左侧导航栏中，选择 **后台服务管理** > **消息推送**，进入 **消息推送** 页面。
2. 在右侧页面上，点击 **密钥管理** 标签，进入 **密钥管理** 标签页。

点击 **推送 API 接口配置** 区域右上角的 **配置**，页面上展示配置入口，如下图所示。

参数	是否必填	说明
状态	是	推送接口的可调用状态。打开开关，即可调用MPS 提供的接口；关闭开关，则不能调用 MPS 提供的接口。
调用接口加密方式	否	仅可选 RSA 算法。
RSA 算法公钥	否	填写 2048 位公钥。 使用私钥对请求参数进行签名后，MPS 使用公钥对签名后的请求参数进行解密，验证调用者身份。

重要：确保公钥填写正确无空格，否则将导致调用接口失败，调用接口说明参见 API 参考 。

4. 点击 **确定** 按钮，保存配置。

推送回调接口配置

登录 mPaaS 控制台，选择 App，通过以下步骤，配置推送回调接口：

1. 在左侧导航栏中，选择 **后台服务管理** > **消息推送**，进入 **消息推送** 页面。
2. 在右侧页面上，点击 **密钥管理** 标签，进入 **密钥管理** 标签页。

点击 **推送回调接口配置** 区域右上角的 **配置**，页面上展示配置入口，如下图所示。

参数	是否必填	说明
状态	是	回调状态。打开开关，移动推送核心将根据配置，给用户服务端回执；关闭开关，移动推送核心将不给用户服务端回执。
回调接口 URL 地址	是	填写回调接口地址。MPS 对POST请求体以私钥进行签名，将签名后的内容作为 sign 参数进行回调。
回调接口加密方式	否	MPS 使用 RSA 算法对 POST 请求体进行签名。
RSA 算法公钥	否	系统自动填写，您无法修改。用户服务端获取 POST 请求体和 sign 参数后，使用公钥验证是否为 MPS 请求，确保数据传输过程中未被篡改，关于回调验签的说明，参见 API 参考 > HTTP 调用 。

4. 点击 **确定** 按钮，保存配置。

说明：使用不同渠道推送消息时，移动推送核心回调时机不同，具体如下所示。

- 三方渠道（FCM/APNs/小米/华为）：调用三方服务成功时，发起回调。
- 自建渠道：推送消息成功时，发起回调。

8.6 使用分析

MPS 依赖客户端 SDK 埋点上报的数据，在 **推送**、**到达**、**打开** 和 **忽略** 4 个维度上对消息推送情况进行统计分析，并提供多种筛选条件供您筛选生成多种形式的统计报表，统计结果数据支持导出。

前置条件

- 基于 mPaaS 框架接入，且 SDK 版本≥10.1.32 时支持本功能。

- 确保已完成客户端埋点，参考文档：
    - Android： 上报推送数据
    - iOS： 统计消息的打开率

说明：对于 iOS 设备，目前仅支持统计其消息的打开数据，到达和忽略数据的统计暂不支持。

操作方法

在 **使用分析** 页面上可查看统计分析数据，登录 mPaaS 控制台，选择 App，通过以下步骤，进入 **使用分析** 页面。

1. 在左侧导航栏中，选择 **后台服务管理** > **消息推送**，进入 **消息推送** 页面。
2. 在右侧页面上，点击 **使用分析** 标签，进入 **使用分析** 标签页。

页面上提供了多种筛选条件，如下图所示，包括 **平台**、**版本**、**推送渠道**、**推送类型** 和 **时间**，供您对统计数据进行筛选。

- **平台**：可选 **所有平台**、**Android – workspaceId**、**iOS – workspaceId**。根据当前已有推送的推送平台以及发起推送的控制台提供不同选项。例如当前未对 iOS 设备推送消息时，则可选项将中不包含 iOS–workspaceId 相关选项。WorkspaceId 为发起推送的控制台的 workspaceId。
- **版本**：赖客户端SDK埋点上报数据，MPS 直接调用 MAS 统计得到的应用版本号。
- **推送渠道**：可选 **所有推送渠道**、**自建渠道** 以及 **三方渠道**（例如小米、华为、苹果等）。当前存在相应渠道的推送后，方提供对应可选项，例如当前不存在小米渠道推送时，则可选项中将不包含 **三方渠道 – 小米**。
- **推送类型**：可选 **所有推送类型**、**SIMPLE**（简单推送）、**TEMPLATE**（模板推送）、**MULTIPLE**（批量推送）和**BROADCAST**（群发推送）。当前存在相应类型的推送后，方提供对应可选项。例如当前不存在简单推送时，则可选项中将不包含 **SIMPLE**。
- **时间**：最长可选时间跨度为 90 天。

图表说明

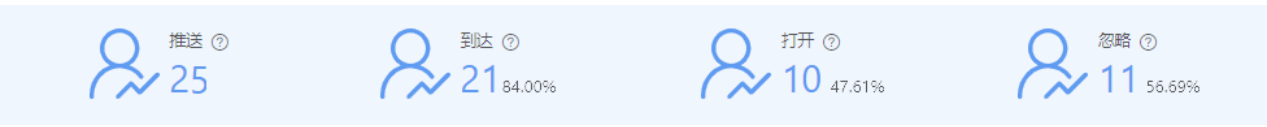
统计图表分 2 部分呈现：

- 推送核心指标
- 推送详细数据

推送核心指标

MPS 分数据和图表 2 种形式来展示推送核心指标，包括 **推送**、**到达**、**打开** 和 **忽略** 4 个指标

数据展示



以数字形式展示消息 **推送**、**到达**、**打开** 和 **忽略** 的统计结果。

- **推送**
  MPS 服务端自动统计，指实际推送的消息数量。
  - 一个推送任务，可能会存在多个推送目标 ID，MPS 对每一个目标推送消息。
  - token 失效或用户绑定关系不存在时，此类推送目标 ID为无效 ID，MPS 对无效 ID推送的消息数量将不计入推送数中。

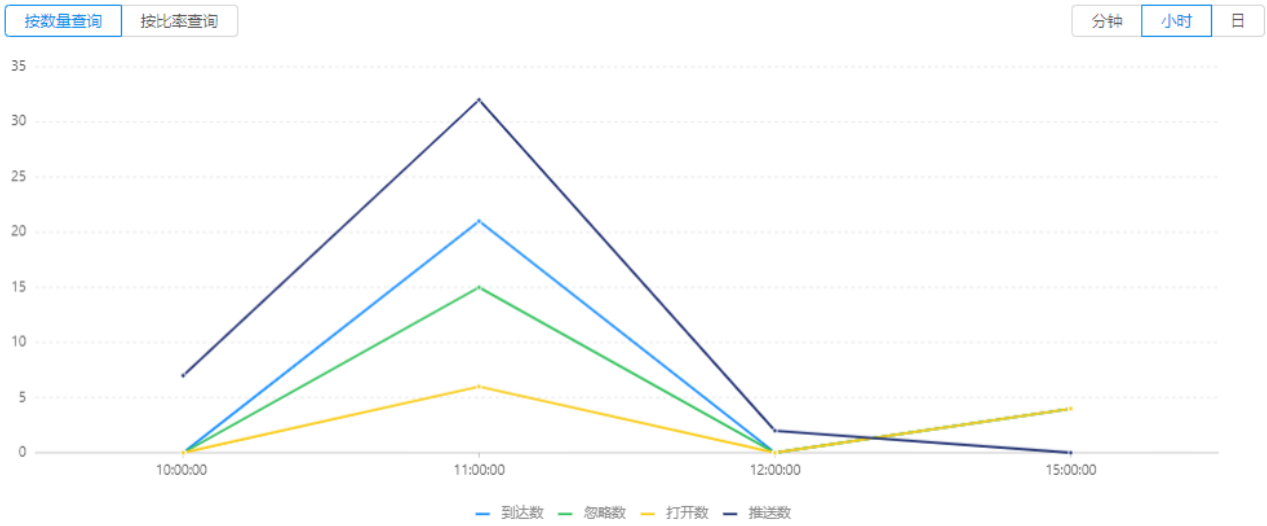
  到达
  到达数：指实际到达客户端本地的消息条数。
  到达率：（到达数/推送数）*100%

  对于不同的推送渠道，到达数据的统计方式不同：

  - Android 自建渠道推送：在消息成功推送至设备后，通过客户端 SDK 埋点上报的方式进行统计。
  - iOS 和 Android 三方渠道推送：在消息推送至对应渠道后，通过其后端服务返回推送结果数据的方式进行统计。

- **打开**
  打开数：赖客户端SDK埋点上报，MPS 调用 MAS统计结果。指在客户端本地被实际打开的消息条数。
  打开率：（打开数/到达数）*100%
- **忽略**
  仅统计自建渠道推送消息的忽略数。
  忽略数：依赖客户端SDK埋点上报，MPS 调用 MAS统计结果。指在客户端本地被手动忽略的消息条数。
  忽略率：（忽略数/到达数）*100%

图表展示



以图表形式呈现消息 **推送**、**到达**、**打开** 和 **忽略** 的统计结果。点击位于图表下方的图例，可隐藏或显示对应指标的曲线。

- 展示形式
  在曲线右上方，选择 分钟、小时 和 日 选项，可分别按分钟、小时和日展示曲线。
  - 分钟：横轴展示存在推送/到达/打开/忽略数据的时间点（精确到分钟）。
  - 小时：横轴展示存在推送/到达/打开/忽略数据的时间点（精确到小时）。
  - 日：横轴展示存在推送/到达/打开/忽略数据的日期。

说明：所选时间跨度超过 1 天时，**分钟/小时** 将不可选。

- 展示内容
  在曲线左上方，选择 **按数量查询/按比率查询** 选项，可分别查看指标的数量曲线和比率曲线。
  - 按数量查询：展示推送数量、到达数量、打开数量和忽略数量曲线。
  - 按比率查询：展示到达率、打开率和忽略率曲线。

推送详细数据


详细数据 导出

时间	推送数	到达数（到达率）	打开数（打开率）	忽略数（忽略率）
2019-06-26 15:00:00	0	4(0%)	4(100.00%)	4(100.00%)
2019-06-26 12:00:00	2	0(0%)	0(0%)	0(0%)
2019-06-26 11:00:00	32	21(65.63%)	6(28.57%)	15(71.43%)
2019-06-26 10:00:00	7	0(0%)	0(0%)	0(0%)

以列表形式展示推送详细数据，随核心指标曲线变化而变化。

- 列表中的 **时间** 继承自核心指标曲线横轴的时间。
- 列表中包含 **推送数**、**到达数**（**到达率**）、**打开数**（**打开率**）和 **忽略数**（**忽略率**）4 个指标。

点击列表右上方的 **导出** 按钮，以 Excel 表格的形式导出列表中的数据。

9 参考

9.1 API 参考

消息推送包含以下 API 接口，具体描述见下表：

调用方式	API	描述
RPC 调用	绑定	绑定用户标识和设备标识（Ad-token）。
	解绑	解绑用户标识和设备标识（Ad-token）。
	三方渠道设备上报	绑定三方渠道设备标识与 Ad-token。
HTTP 调用	绑定	绑定用户标识和设备标识（Ad-token）。

	解绑	解绑用户标识和设备标识（Ad-token）。
	三方渠道设备上报	绑定三方渠道设备标识与 Ad-token。
	极简推送	对一个目标 ID 推送一条消息。
	模板推送	对一个目标 ID 推送一条消息，消息通过模板创建。
	批量推送	对多个目标 ID 推送不同消息。基于模板，为各推送 ID 配置不同的模板占位符内容，从而实现消息的个性化推送。
	群发	对全网设备推送相同消息，消息通过模板进行创建。

重要：绑定、解绑和三方渠道设备上报 3 个接口，因其即将不支持 HTTP 调用方式，所以建议您优先使用 RPC 方式进行调用。

RPC 调用

mPaaS 中间层的 MPPush 类封装了移动推送组件所有 API，包括绑定、解绑以及三方渠道设备上报等接口。

绑定

方法定义

```
public static ResultPbPB bind(Context ctx, String userId, String token)
```

本方法用于绑定用户标识和设备标识。绑定完成后，可基于用户维度推送消息。

说明：本接口需要在子线程中进行调用。

参数说明

ctx	Context：一个不为空的 Context。
userId	String：用户唯一标识，该标识不一定为接入方用户系统中的真实标识，但一定可以与用户形成一一映射关系。
token	String：由移动推送网关下发的设备标识。

返回值

sucess	接口调用是否成功，ture：成功；false：失败。
code	操作结果码，常见的操作结果码及其含义参见下表。
name	操作结果码的名称。
message	操作结果码对应的描述信息。

操作结果码说明

code	name	message	说明
3012	NEED_USERID	need userid	调用接口时，入参 userId 为空。
3001	NEED_DELIVERYTOKEN	need token	调用接口时，入参 token 为空。

使用示例

```
private void doSimpleBind() {
final ResultPbPB resultPbPB = MPPush.bind(getApplicationContext(), mUserId, PushMsgService.mAdToken);
handlePbPBResult("绑定用户操作", resultPbPB);
}
```

解绑

方法定义

```
public static ResultPbPB unbind(Context ctx, String userId, String token)
```

本方法用于解绑用户标识和设备标识。

说明：本接口需要在子线程中进行调用。

参数说明

ctx	Context：一个不为空的 Context。
userId	String：用户唯一标识，该标识不一定为接入方用户系统中的真实标识，但一定可以与用户形成一一映射关系。
token	String：由移动推送网关下发的设备标识。

返回值

调用本方法的返回值参见 返回值 。

使用示例

```
private void doSimpleUnBind() {
final ResultPbPB resultPbPB = MPPush.unbind(getApplicationContext()
, mUserId, PushMsgService.mAdToken);
handlePbPBResult("解绑定用户操作", resultPbPB);
}
```

三方渠道设备上报

方法定义

```
public static ResultPbPB report(Context context, String deliveryToken, int thirdChannel, String thirdChannelDeviceToken)
```

该方法用于同步绑定三方渠道设备标识与本机设备标识。

说明：本方法用于上报三方渠道设备标识和 mPaaS 设备标识（移动推送网关下发的 Ad-token）至移动推送核心，移动推送核心将绑定这两个标识。完成此过程后，方可使用三方渠道推送消息。

参数说明

ctx	Context：一个不为空的 Context。
deliveryToken	String：由移动推送网关下发的设备标识（Ad-token）。
thirdChannel	int：三方渠道厂商。4：小米；5：华为。
thirdChannelDeviceToken	String：三方渠道设备标识

返回值
调用本方法的返回值参见 返回值 。

使用示例

```
private void doSimpleUploadToken() {
final ResultPbPB resultPbPB = MPPush.report(getApplicationContext(), PushMsgService.mAdToken
, PushOsType.HUAWEI.value(), PushMsgService.mThirdToken);
handlePbPBResult("第三方push标识上报操作", resultPbPB);
```

HTTP 调用

调用 MPS 服务端接口时，需要先在消息推送控制台侧进行 推送 API 接口配置 ，填写公钥，MPS 服务端使用公钥对请求体进行验签，验证调用者身份。

MPS 回调时，需要先在控制台侧进行 推送回调接口配置 ，填写回调接口的 url 地址，用户服务端收到回调请求后，使用公钥对请求进行验签，验证是否为 MPS 请求。验签代码在代码示例的 com.mpaas.mps.demo.callback.CallbackHandler 类中，对应代码如下：

```
//验签
sign = sign.replace('-','+');
sign = sign.replace('_','/');
if(!SignUtil.check(jsonStr,sign,pubKey,"UTF-8")){
responseJson.put("error","sign verify error");
ResponseJson(ctx, req, responseJson.toString());
return;
```

RSA 算法签名规则如下：

- 采用 SHA 256 签名算法。
- 公钥长度为 2048 位。
- 将签名结果转换成 base64 字符串。
- 将 base64 字符串中的 + 替换为 - ，/ 替换为 - ，得到最终签名结果。

说明：

- 对于公有云环境，您可以使用 https://cn-hangzhou-mps-api.cloud.alipay.com 作为域名。
- 调用接口报签名错误时，确认加签使用的私钥以及在控制台侧配置的公钥正确无空格（参见 推送 API 接口配置 ），接口地址中的 appId 和 workspaceId 正确。

您可以点击 代码示例，下载服务端代码示例，了解 **简单推送**、**模板推送**、**批量推送**、**群发** 的 API 代码。其中：

- PushSimpleDemo.java 为简单推送。

- PushTemplateDemo.java 为模板推送。
- PushMultipleDemo.java 为批量推送。
- PushBroadcastDemo.java 为群发。

绑定

路径

POST /bind/bind/{APPID}/{WORKSPACEID}/{signature}

调用本接口，绑定用户标识和设备标识，绑定成功后，可基于用户维度推送消息。

参数说明

参数说明	说明
userId	要绑定的用户标识。
deliveryToken	目标设备 token。
osType	设备类型，1 为 Android，2 为 iOS。

使用示例

```
POST /bind/bind/9156D89171050/sit/BEwgZbDmeTmUIGfgsv27YwI7EazkUmpdJiSjNUo2D0qNM33wsI3T3gLh

{
 “deliveryToken”:” 2a0a7683d042611a3db830ce1be2804bc996e4962c087ff48117b53950427adb”,
 “osType”:1,
 “userId”: “asdgqwer1234”
}
```

解绑

路径

POST /bind/unbind/{APPID}/{WORKSPACEID}/{signature}

调用本接口，解绑用户标识与设备标识。

- **参数说明**

参数说明	说明
userId	要解绑的用户标识。
deliveryToken	目标设备 token。

使用示例

```
POST /bind/unbind/9156D89171050/sit/BEwgZbDmeTmUIGfgsv27YwI7EazkUmpdJiSjNUo2D0qNM33wsI3T3gLh

{
 “deliveryToken”： “2a0a7683d042611a3db830ce1be2804bc996e4962c087ff48117b53950427adb”，
 “userId”： “asdgqwer1234”
}
```

三方渠道设备上报

路径

POST /deviceinfo/report/{APPID}/{WORKSPACEID}/{signature}

调用本接口，上报三方渠道设备标识至移动推送核心和 mPaaS 设备标识（移动推送网关下发的 Ad-token）等信息至移动推送核心，移动推送核心将绑定这两个标识。完成此过程后，方可使用三方渠道推送消息。

参数说明

参数名称	说明
deliveryToken	目标设备 token。
osType	设备类型，1 为 Android，2 为 iOS。
imei	国际移动设备识别码。
imsi	国际移动用户识别码。
appVersion	客户端应用版本。
pushVersion	pushSDK 版本。
connectType	设备的网络连接类型，例如 wifi、cmwap。
model	设备型号，例如 iPad、iPhone、Android。
thirdChannelDeviceToken	第三方渠道设备标识。
thirdChannel	第三方渠道，4 为 小米；5 为 华为；6 为 FCM。
channel	应用的安装渠道，例如 App Store 、mhs-channel、Google Play。

使用示例

```
POST
/deviceinfo/report/9156D89171050/sit/BEwgZbDmeTmUIGfgsv27YwI7EazkUmpdJiSjNUo2D0qNM33wsI3T3gLh

{
"appVersion":"9.0.1",
"channel":"channel",
"connectType":"WIFI",
"deliveryToken":"2a0a7683d042611a3db830ce1be2804bc996e4962c087ff48117b53950427adb",
"imei":"imei",
"imsi":"imsi",
"model":"model",
"osType": 1,
"pushVersion":"2.1.0",
"connectType":"wifi"
```

```
}
```

极简推送

对一个推送 ID 推送一条消息。

- **路径**

  POST /push/pushsimple/{APPID}/{WORKSPACEID}/{signature}

参数说明

参数名称	说明
notifyType	消息类型，默认为通知栏消息。
action	点击消息后的跳转方式，0 为 web url ，1 为 Intent activity，默认为 web url。
extended_params	扩展参数。
taskName	推送任务名称。
title	消息的标题。
content	消息的正文。
Url	点击消息后的跳转地址。
deliveryType	传送方式，决定了 PUSH 服务如何解析 target_msgkey 字段中的目标 token： • 1：按 Android 设备 token 解析 • 2：按 iOS 设备Token 解析 • 3：按用户标识解析
expiredSeconds	消息的有效期，单位为秒。
target_msgkey	为 map 格式，包含该消息推送的目标 token 和赋予这条消息的业务方消息 ID。目标 token 可以是 Android 设备 Ad-token、iOS 设备的 Device Token 以及用户标识（userId）。

返回值

推送失败时，接口返回的错误码参见下表。

错误码	名称	说明
8001	FLOW_CONTROL_ERROR	由于推送限流控制，当前推送请求被抛弃。

使用示例

参考服务端代码示例：com.mpaas.mps.demo.PushSimpleDemo

```
POST
/push/pushsimple/9156D89171050/sit/BEwgZbDmeTmUIGfgsv27YwI7EazkUmpdJiSjNUo2D0qNM33wsI3T3gLh

{
"action":0,
"content":"简单推送样例-内容",
```

```
"deliveryType":2,
"expiredSeconds":600,
"extended_params":{
"test":"自定义扩展参数"
},
"silent":0,
"target_msgkey":{
"db295d8c016b1000831d30300ba0d05c564":"1562901468701"
},
"taskName":"测试任务",
"title":"简单推送样例-标题",
"uri":"http://127.0.0.1"
}
```

模板推送

说明：在调用本接口之前，您需要先在消息推送控制台上创建好目标模板，详细操作参见 创建模板 。

对多个推送 ID 推送相同消息，消息通过模板创建。

- **路径**

  POST /push/pushtemplate/{APPID}/{WORKSPACEID}/{signature}

- **参数说明**

参数名称	说明
notifyType	消息类型，默认为通知栏消息。
action	点击消息后的跳转方式，0 为 web url ，1 为 Intent activity，默认为 web url。
silent	是否是静默消息，1 为静默消息，0 为非静默消息。
extended_params	扩展参数。
taskName	推送任务名称。
templateName	使用的模板名称。
templatekv	模板参数，为 map 格式，和 templateName 指定的模板对应，key 为占位符名称，value 为要替换的值，例如模板内容为（两个 # 之间为占位符名称） 恭喜#name#中了#money#元，则对应的 templatekv 为： “templatekv”:{ “name”: “您”, “money”: “50” }
expiredSeconds	消息的有效期，单位为秒。
deliveryType	传送方式，决定了 PUSH 服务如何解析 target_msgkey 字段中的目标 token： • 1：按 Android 设备 token 解析 • 2：按 iOS 设备Token 解析 • 3：按用户标识解析
target_msgkey	为 map 格式，表示该消息推送的目标 token 和赋予这条消息的业务方消息 ID，目标 token 可以是 Android 设备 token、iOS deviceToken 以及用户标识。

- **返回值**

  推送失败时，接口返回的错误码参见下表。

错误码	名称	说明
8001	FLOW_CONTROL_ERROR	由于推送限流控制，当前推送请求被抛弃。

使用示例
参考服务端代码示例：com.mpaas.mps.demo.PushTemplateDemo。

```
POST
/push/pushtemplate/9156D89171050/sit/BEwgZbDmeTmUIGfgsv27YwI7EazkUmpdJiSjNUo2D0qNM33wsI3T3gLh

{
"action":0,
"deliveryType":1,
"expiredSeconds":600,
"extended_params":{
"test":"自定义扩展参数"
},
"silent":0,
"target_msgkey":{
"1fe792f00163100080a730300a19467a":"1562901468701"
},
"taskName":"测试任务",
"templateName":"template",
"templatekv":{
"name":"您",
"money":"50"
}
}
```

批量推送

说明：在调用本接口之前，您需要先在消息推送控制台上创建好目标模板，并确保模板中存在占位符，否则将无法实现消息的个性化推送（对不同推送 ID 推送不同消息）。关于创建模板的详细操作方法，参见 创建模板。

对各个推送 ID 推送不同消息。通过替换模板占位符的方式，创建针对某一推送 ID 的个性化消息。与模板推送的区别在于，每一个推送ID 可以收到内容不相同的消息。

- **路径**

  POST /push/pushmultiple/{APPID}/{WORKSPACEID}/{signature}

- **参数说明**

参数名称	说明
notifyType	消息类型，默认为通知栏消息。
action	点击消息后的跳转方式，0 为 web url ，1 为 Intent activity，默认为 web url。
silent	是否是静默消息，1 为静默消息，0 为非静默消息。
extended_params	扩展参数。
taskName	推送任务名称。
templateName	使用的模板名称。

templatekv	模板参数，为 map 格式，和 templateName 指定的模板对应，key 为占位符名称，value 为要替换的值，例如模板内容为（两个 # 之间为占位符名称） 恭喜#name#中了#money#元，则对应的 templatekv 为： “templatekv”:{ “name”: “您”, “money”: “50” }
expiredSeconds	消息的有效期，单位为秒。
deliveryType	传送方式，决定了 PUSH 服务如何解析 target_msgkey 字段中的目标 token： • 1：按 Android 设备 token 解析 • 2：按 iOS 设备Token 解析 • 3：按用户标识解析
target_msginfo	以推送目标 token 为 key，消息相关的内容为 value。其中目标 token 如何解析由 deliveryType 决定；value 部分由业务方消息 ID 和模板参数组成。

- **返回值**

  推送失败时，接口返回的错误码参见下表。

错误码	名称	说明
8001	FLOW_CONTROL_ERROR	由于推送限流控制，当前推送请求被抛弃。

使用示例

参考服务端代码示例：com.mpaas.mps.demo.PushMultipleDemo。

```
POST
/push/pushmultiple/9156D89171050/sit/BEwgZbDmeTmUIGfgsv27YwI7EazkUmpdJiSjNUo2D0qNM33wsI3T3gLh

{
"action":0,
"deliveryType":1,
"expiredSeconds":600,
"extended_params":{
"test":"自定义扩展参数"
},
"notifyType":"notify",
"silent":0,
"target_msginfo":{
"1fe792f00163100080a730300a19467b":{
"msgkey":"1562901762262121",
"templatekv":{
"name":"张三",
"money":"100"
}
},
"1fe792f00163100080a730300a19467a":{
"msgkey":"1562901762262",
"templatekv":{
"name":"小明",
"money":"50"
}
}
},
```

```
"taskName":"测试任务",
"templateName":"template"
}
```

群发

说明：在调用本接口之前，您需要先在消息推送控制台上创建好目标模板，详细操作参见 创建模板 。

对全网设备推送相同消息，消息通过模板创建。

- **路径**

  POST /push/pushbroadcast/{APPID}/{WORKSPACEID}/{signature}

- **参数说明**

参数名称	说明
notifyType	消息类型，默认为通知栏消息。
action	点击消息后的跳转方式，0 为 web url ，1 为 Intent activity，默认为 web url。
silent	是否是静默消息，1 为静默消息，0 为非静默消息。
extended_params	扩展参数。
taskName	推送任务名称。
templateName	使用的模板名称。
templatekv	模板参数，为 map 格式，和 templateName 指定的模板对应，key 为占位符名称，value 为要替换的值，例如模板内容为（两个 # 之间为占位符名称） 恭喜#name#中了#money#元，则对应的 templatekv 为： “templatekv”:{ “name”: “您”, “money”: “50” }
msgkey	业务方消息 ID。
expiredSeconds	消息的有效期，单位为秒。
deliveryType	传送方式，决定了 PUSH 服务判断推送平台： • 1：Android 群发 • 2：iOS 群发

- **返回值**

  推送失败时，接口返回的错误码参见下表。

错误码	名称	说明
8001	FLOW_CONTROL_ERROR	由于推送限流控制，当前推送请求被抛弃。

使用示例

参考服务端代码示例：com.mpaas.mps.demo.PushBroadcastDemo。

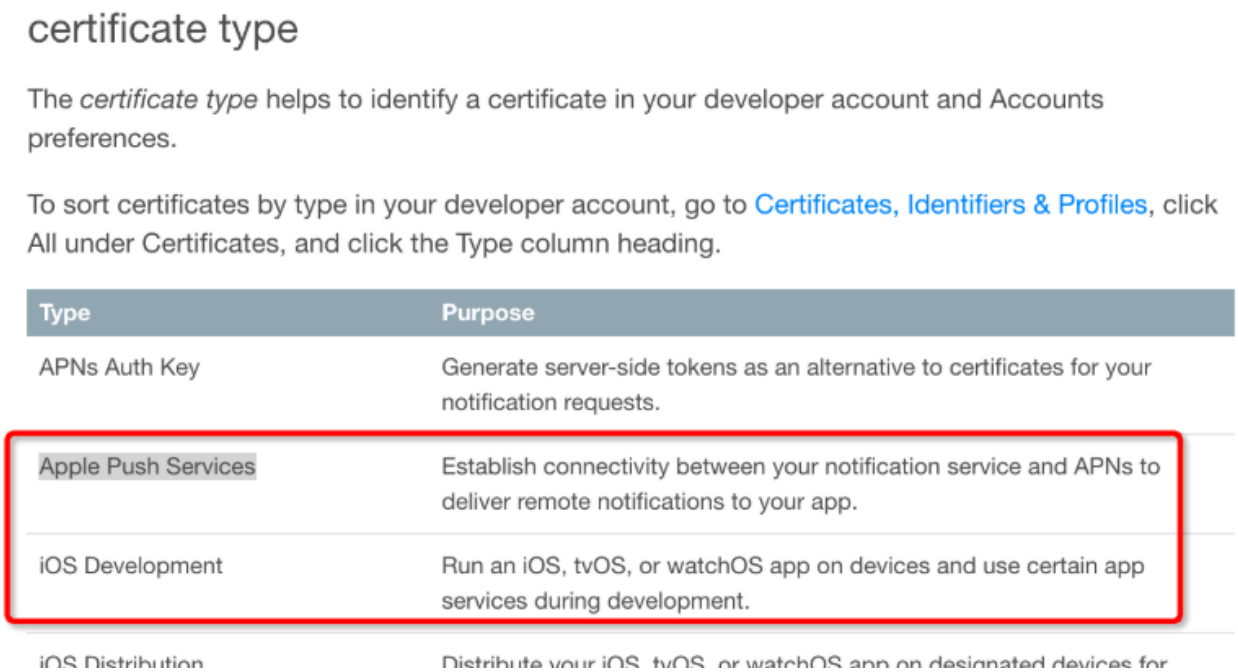
```
POST
/push/pushbroadcast/9156D89171050/sit/BEwgZbDmeTmUIGfgsv27YwI7EazkUmpdJiSjNUo2D0qNM33wsI3T3gLh
```

```
{
"action":0,
"deliveryType":1,
"expiredSeconds":600,
"extended_params":{
"test":"自定义扩展参数"
},
"msgkey":"1562902268610",
"silent":0,
"taskName":"测试任务",
"templateName":"template",
"templatekv":{
"name":"您",
"money":"100"
}
}
```

9.2 制作 iOS 推送证书

为了向 iOS 设备推送数据，您首先需要在移动推送控制台配置 iOS 推送证书。本文将介绍移动推送服务支持的证书类型，并引导您制作证书。

证书类型


certificate type

The *certificate type* helps to identify a certificate in your developer account and Accounts preferences.

To sort certificates by type in your developer account, go to Certificates, Identifiers & Profiles, click All under Certificates, and click the Type column heading.

Type	Purpose
APNs Auth Key	Generate server-side tokens as an alternative to certificates for your notification requests.
Apple Push Services	Establish connectivity between your notification service and APNs to deliver remote notifications to your app.
iOS Development	Run an iOS, tvOS, or watchOS app on devices and use certain app services during development.
iOS Distribution	Distribute your iOS, tvOS, or watchOS app on designated devices for

苹果证书类型 如上图所示。移动推送服务只支持 Apple Push Service 类型的证书。

Apple Push Service 易和 iOS Development 类型的证书混淆。使用 iOS Development 证书会导致消息推送大量失败。下面将介绍如何通过 **MAC Key Store** 和 **移动推送控制台** 区分这两类证书。

MAC Key Store

双击已有的 .p12 证书，将证书导入 MAC 钥匙串中，您将看到证书名称等信息：

其中，

- iPhone Developer：苹果开发证书。移动推送不支持。
- Apple Push Service：生产环境苹果推送证书。移动推送支持。
- Apple Development IOS Push Services：开发环境苹果推送证书。移动推送支持。

移动推送控制台

在移动推送控制台导入证书后，您将看到以下证书信息：

属性	值
alias	
bundleName	
certFilename	
certHost	api.development.push.apple.com
certPort	443
issuerDN	CN=Apple Worldwide Developer Relations Certification Authority, OU=Apple Worldwide Developer Relations, O=Apple Inc., C=US
notAfter	1567063059000
notBefore	1535527059000
subjectDN	C=CN, OU=3SJ9H5532E, CN=Apple Development IOS Push Services

如上图，subjectDN 属性：

- Apple Development IOS Push Services：开发环境苹果推送证书。移动推送支持。
- Apple Push Service：生产环境苹果推送证书。移动推送支持。

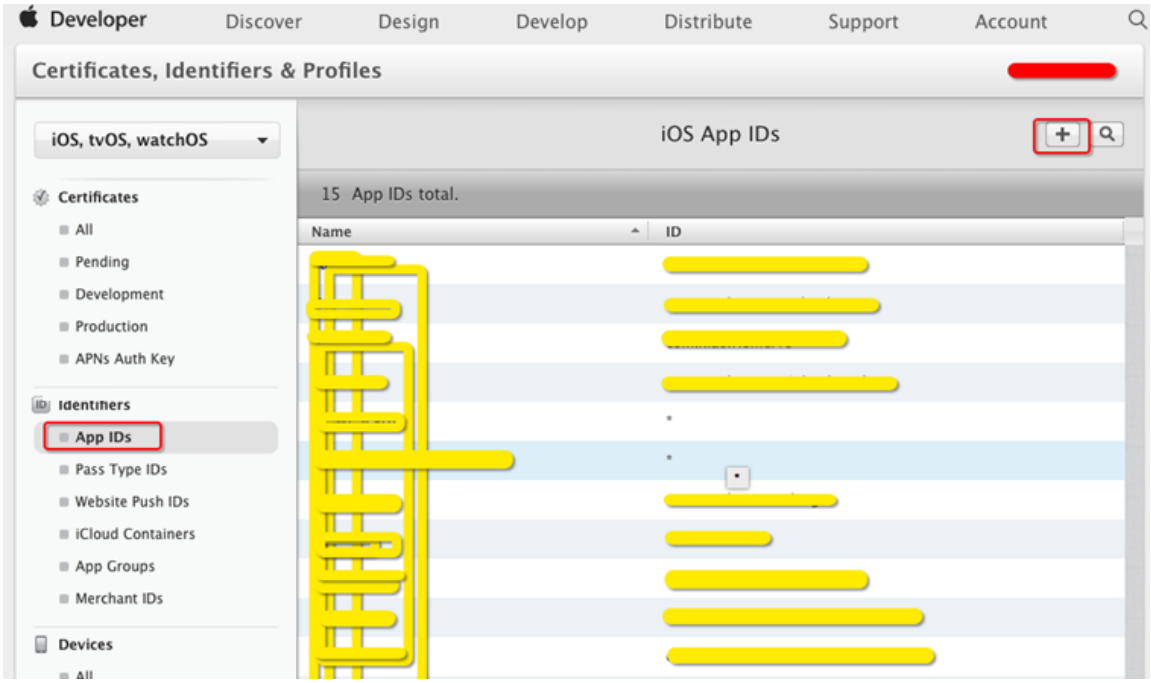
subjectDN	C=US, O= , OU= , CN=iPhone Developer: , UID=579328UB59

如上图，subjectDN 属性 iPhone Developer 表明是苹果开发证书，移动推送不支持。

制作证书

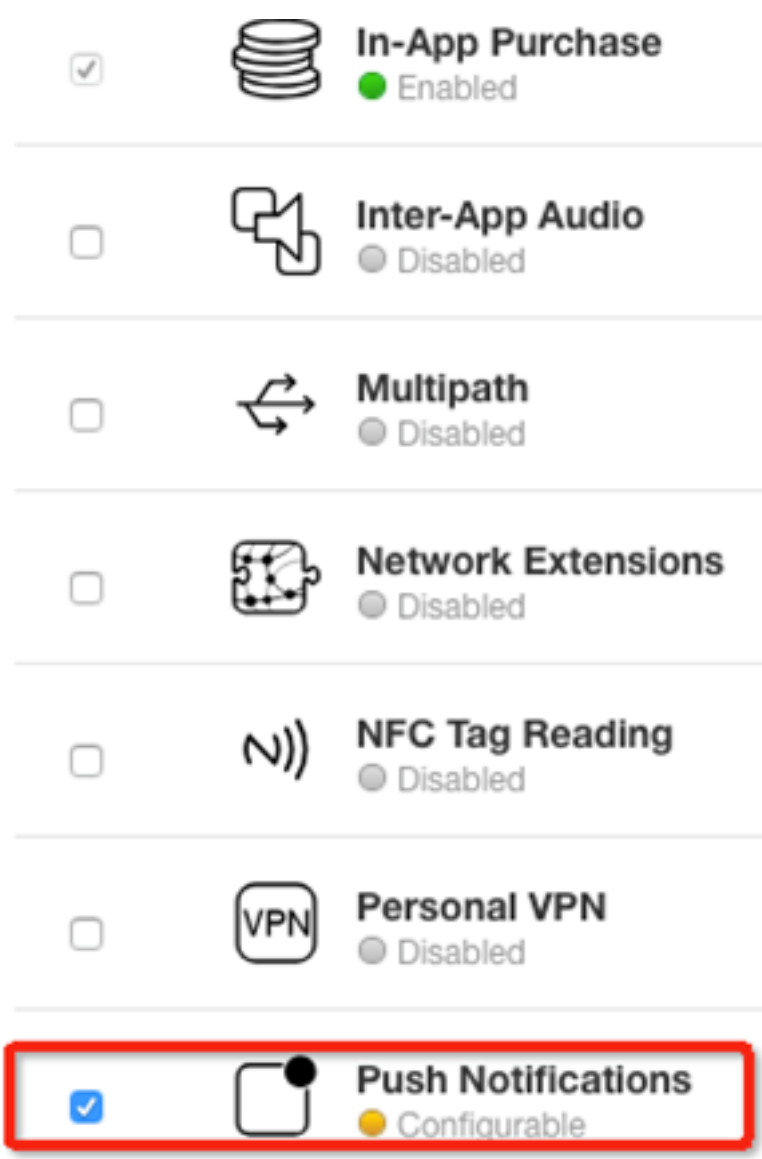
创建苹果 App ID

在苹果开发平台，点击左侧导航栏 **App IDs**，然后点击右上角 + 按钮。



2. 填写基础信息：
    - **App ID Description** > **Name**
    - **App ID Suffix** > **Bundle ID**： Bundle ID 需要具备唯一性。

    勾选 Push Notifications 能力。



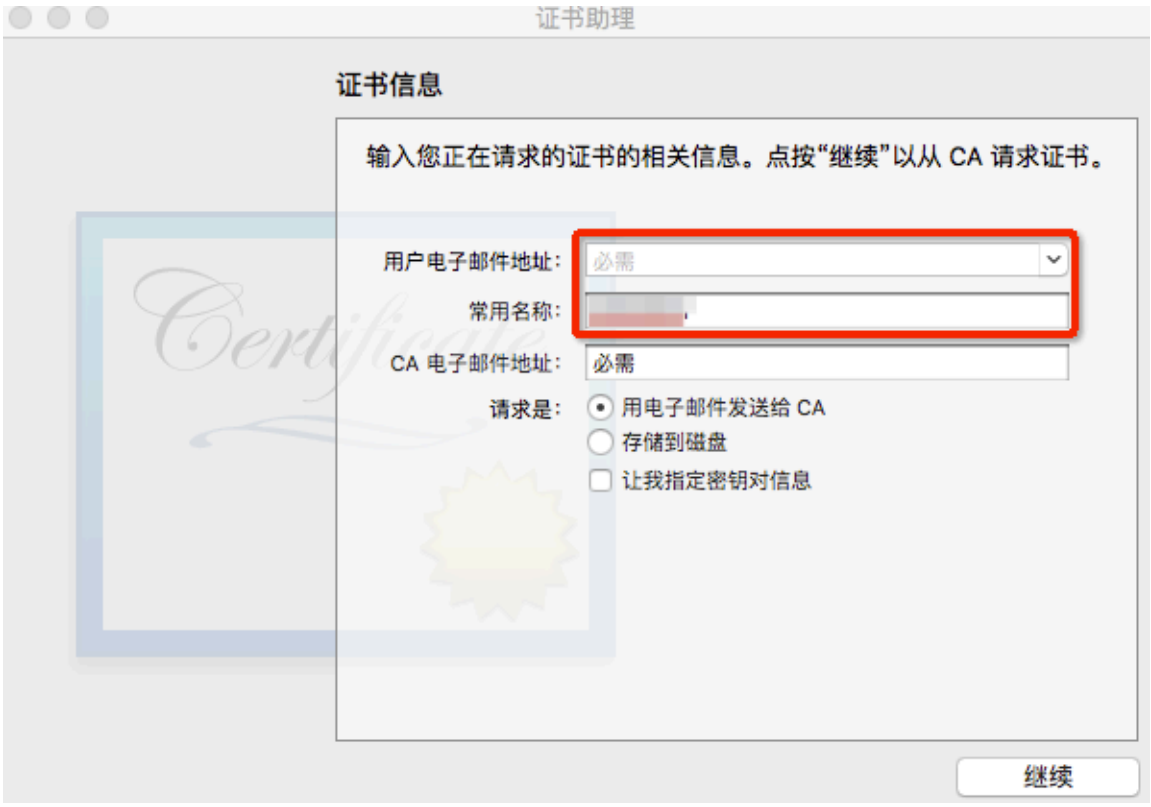
点击 **Continue** 按钮后，点击 **Register** 按钮完成创建。

制作 .certSigningRequest 文件

进入 Mac 中的钥匙串服务。

请求证书。

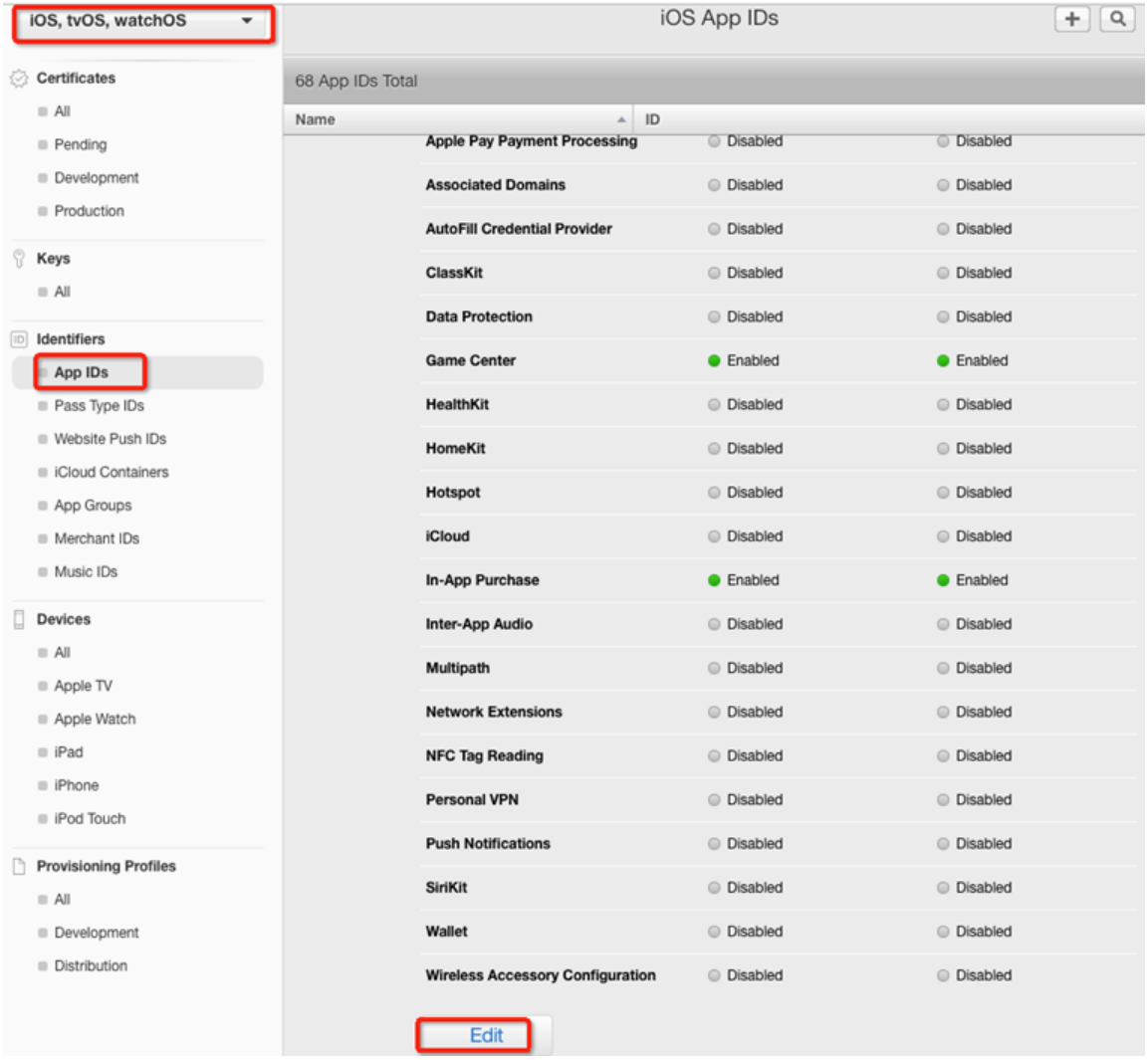
请求证书时，需要实际情况填写邮件地址和常用名称等相关信息：



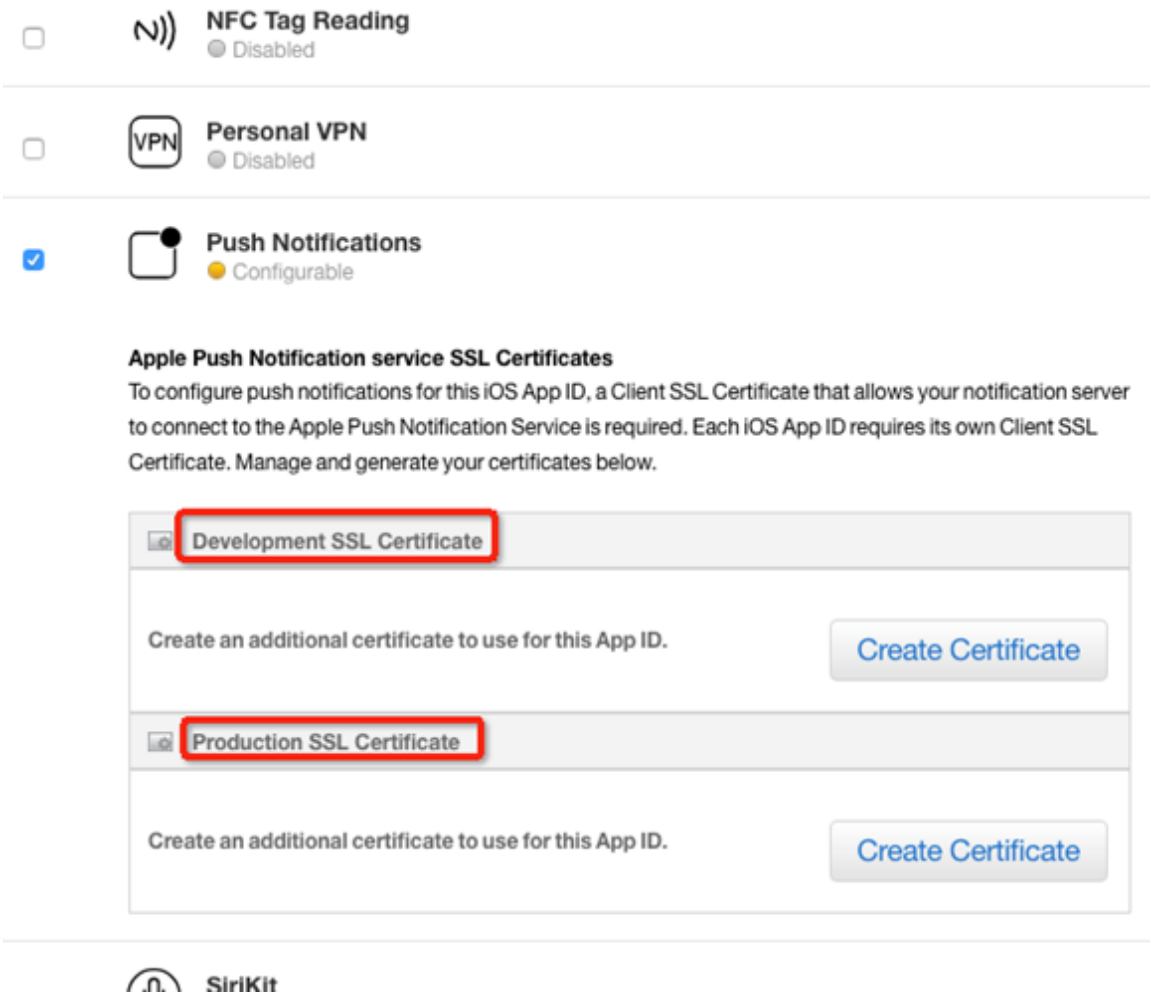
.certSigningRequest 文件制作成功，如图所示：

创建证书

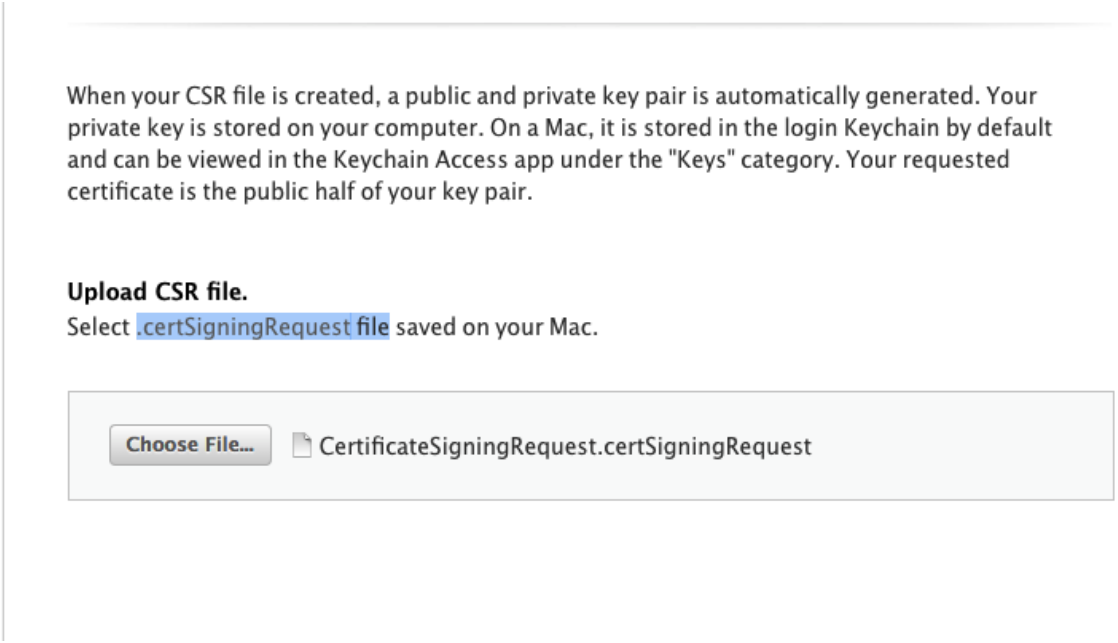
在苹果 **App IDs** 页面中，选中自己的 iOS App ID，点击 **Edit** 按钮。



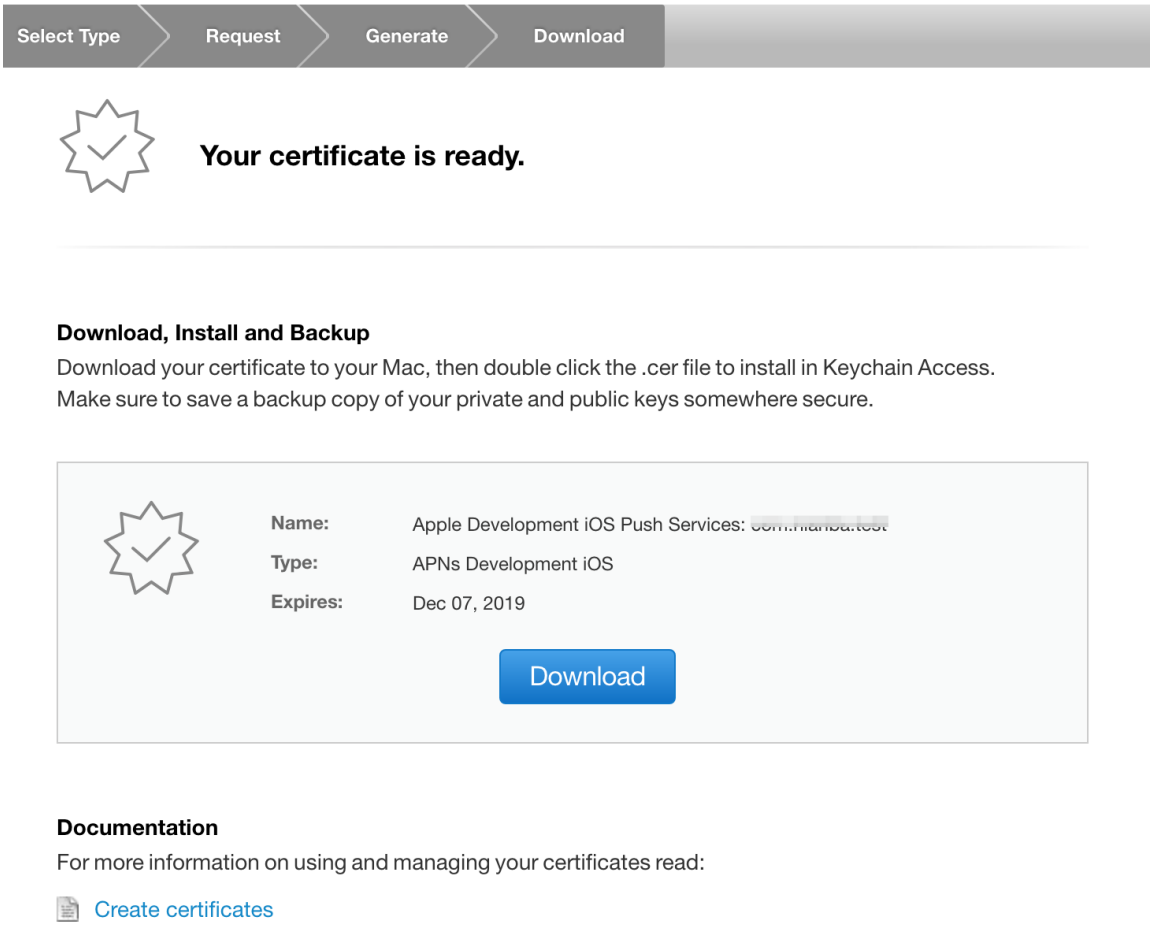
点击 **Development SSL Certificate** 或 **Production SSL Certificate** 卡片中的 **Create Certificate** 按钮，开始创建开发或生产环境下的证书。



在创建证书时，您需要上传前面制作的 .certSigningRequest 文件：



证书创建成功后，您将看到以下页面。点击 **Download** 按钮，您将得到 .cer 文件。



将 .cer 文件转换成 .p12 文件。

- 双击 .cer 文件，将文件导入 Key Store。
- 找到刚刚导入的证书，右键单击，选择 **导出** 功能。导出成功后您将获得 .p12 证书。

至此您已获得了 .p12 证书，可以到 **移动推送控制台** > **推送配置** 中配置 iOS 推送证书。


蚂蚁金服
ANT FINANCIAL
金融科技
FINANCIAL TECHNOLOGY