蚂蚁集团
ANT GROUP

蚂蚁科技产品手册

消息推送

产品版本：V20210331
文档版本：V20210331
蚂蚁科技技术文档

蚂蚁科技集团有限公司版权所有 © 2020 ，并保留一切权利。

未经蚂蚁科技事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

商标声明

蚂蚁集团
ANT GROUP

及其他蚂蚁科技服务相关的商标均为蚂蚁科技所有。
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 基本概念 3
3 推送流程 4
4 接入客户端 8
4.1 接入 Android 8
4.1.1 快速开始 8
4.1.2 接入第三方推送渠道 16
4.1.3 接入 FCM 推送渠道 22
4.1.4 使用内建消息通知 25
4.2 接入 iOS 26
5 配置服务端 32
6 使用控制台 33
6.1 消息列表 33
6.1.1 创建消息 33
6.1.2 管理消息 42
6.1.3 推送消息 44
6.2 使用分析 46
6.3 消息模板 50
6.3.1 创建模板 50
6.3.2 管理模板 51
6.4 渠道配置 52
6.5 推送配置 55
6.6 密钥管理 57
7 常见问题 61
8 参考 64
8.1 API 说明 - 客户端 64
8.2 API 说明 - 服务端 67
8.3 制作 iOS 推送证书 87

1 消息推送简介

消息推送服务（Message Push Service，简称 MPS）是移动开发平台 mPaaS 提供的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 在 MPS 中集成了华为、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

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。

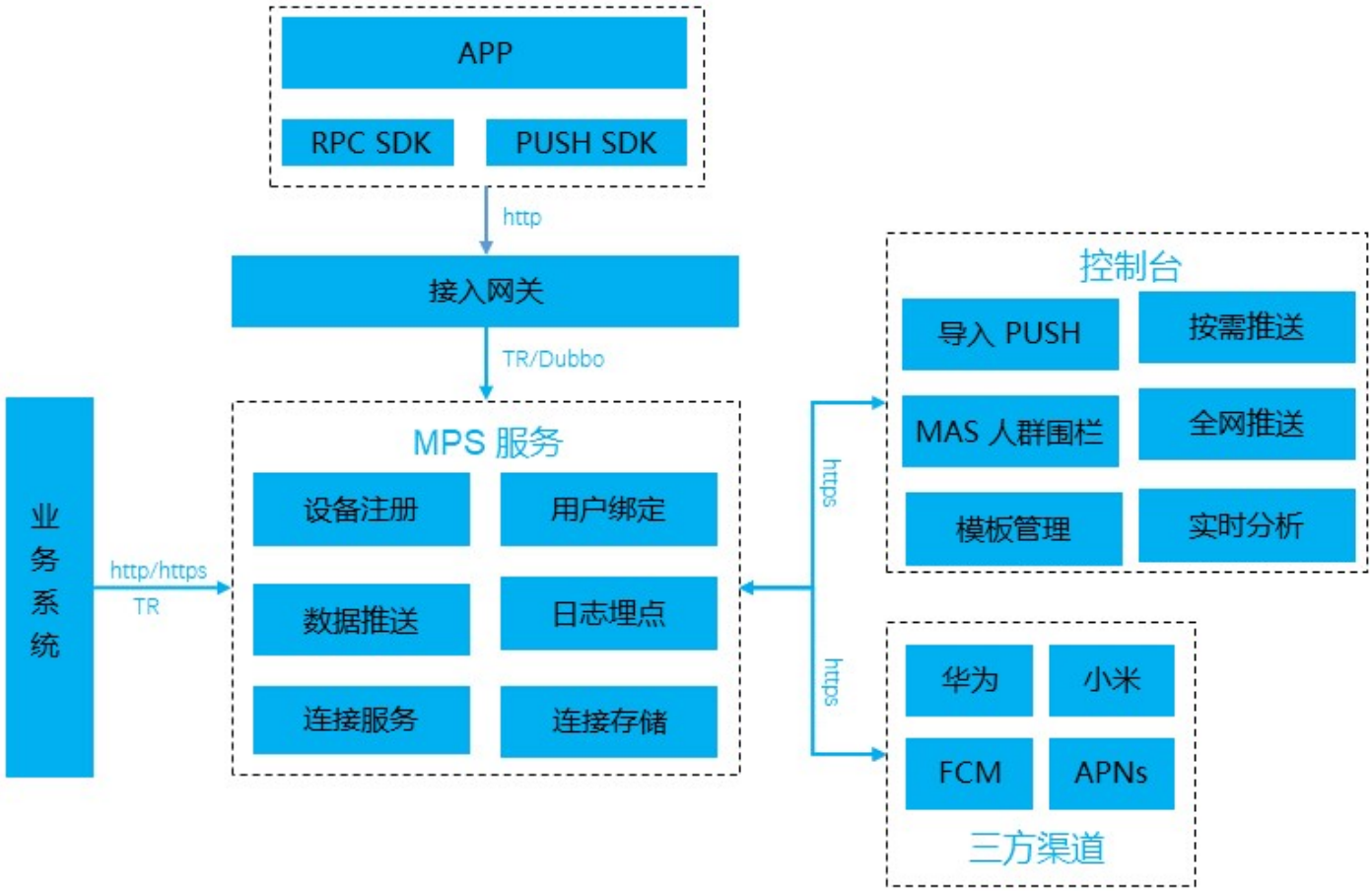
MPS 核心功能如下：

- **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，设备建连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**：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板，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**：MPS 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，以便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。

客户端通过使用 mPaaS 移动网关服务（MGS），调用远程过程调用（Remote Procedure Call，简称 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，实现基于设备维度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

MPS 具备以下优势：

- **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

消息推送的典型应用场景如下：

营销活动

向用户推送针对性的消息，包括营销活动、业务提醒等，以提升用户粘度。App 通过调用消息推送 API，对目标用户进行定向消息推送，以更主动的方式触达更多用户，吸引用户消费，从而提升最终营销活动转化效果。

系统通知

按照 App 服务端业务逻辑指定推送人群，直接将消息推送给目标设备。

针对不同应用场景，MPS 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，可指定消息模板，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，可指定消息模板，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，可指定消息模板，消息正文由替换模板占位符得到。

2 基本概念

本术语表按拼音首字母对术语进行排序。

A

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。

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。

J

极简推送（SimplePush）
针对单个推送目标 ID，推送一条消息的方式。

M

模板参数（Template Placeholder）
又成为模板占位符，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。

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。

P

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。

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。

Q

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。

R

任务名称（Task Name）
一次消息推送请求标识为一次任务。

T

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。

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。

X

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。

Y

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。

应用 ID（AppID）
应用标识，在创建应用时生成。

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。

3 推送流程

接入消息推送组件后，客户端通过使用 mPaaS 移动网关服务，调用远程过程调用（Remote Procedure Call，简称 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，实现基于设备维度和用户维度的消息推送。不同的设备平台对应的消息推送流程会有所差异，下面将介绍不同设备平台在 RPC 接入方式下的推送处理流程。

在了解推送流程之前，需要先了解消息推送涉及的一些基本概念。

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。

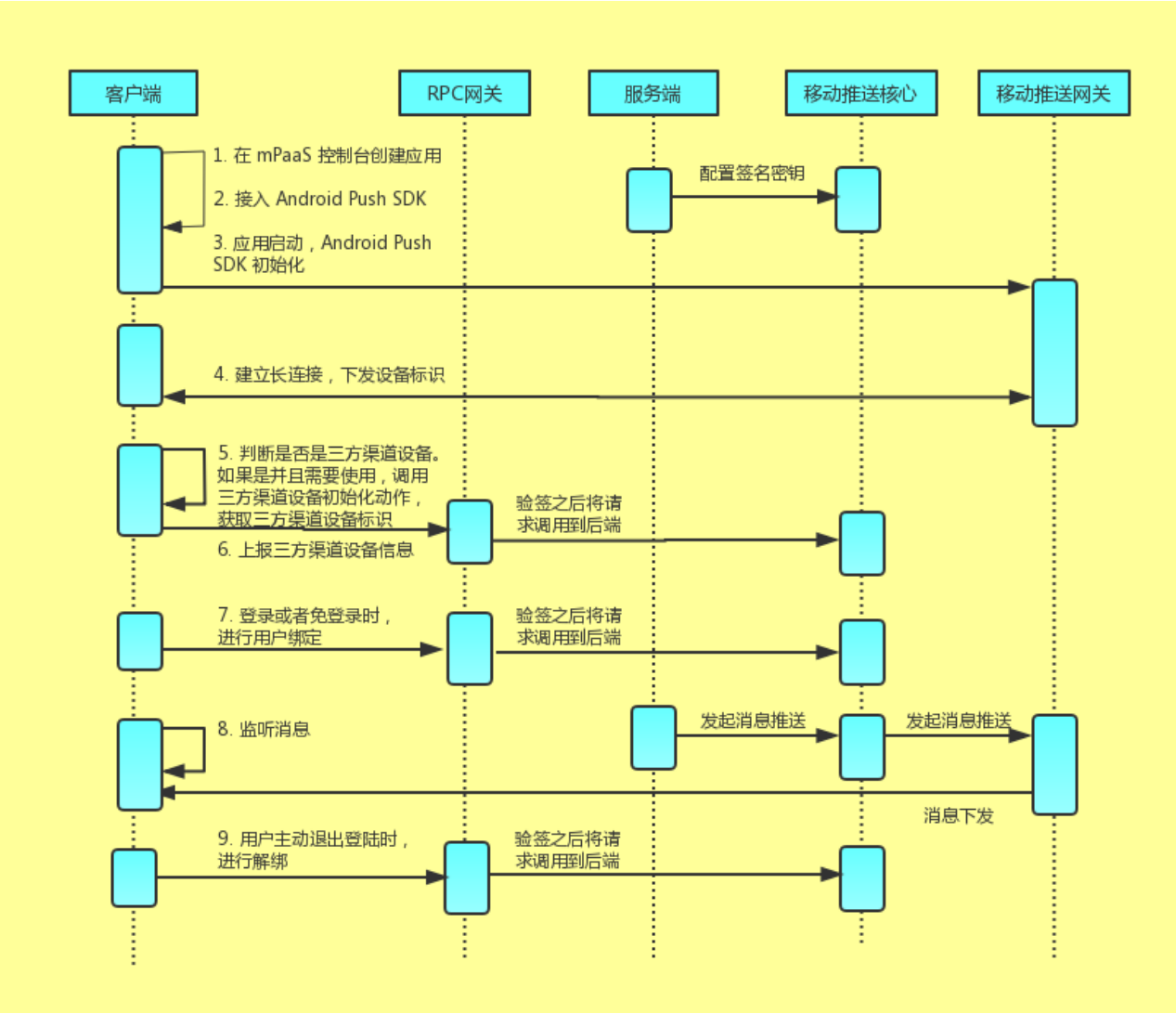
重要：在请求设备标识 (token) 部分，若是小米、华为或其他已经接入厂商推送平台的手机，还会向第三方平台请求设备标识，需要等待两个设备标识返回，通过调用 report 接口将两者绑定，才能使用厂商的推送通道。普通手机只需要使用 mPaaS 返回的设备标识。

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

客户端使用 RPC SDK 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：

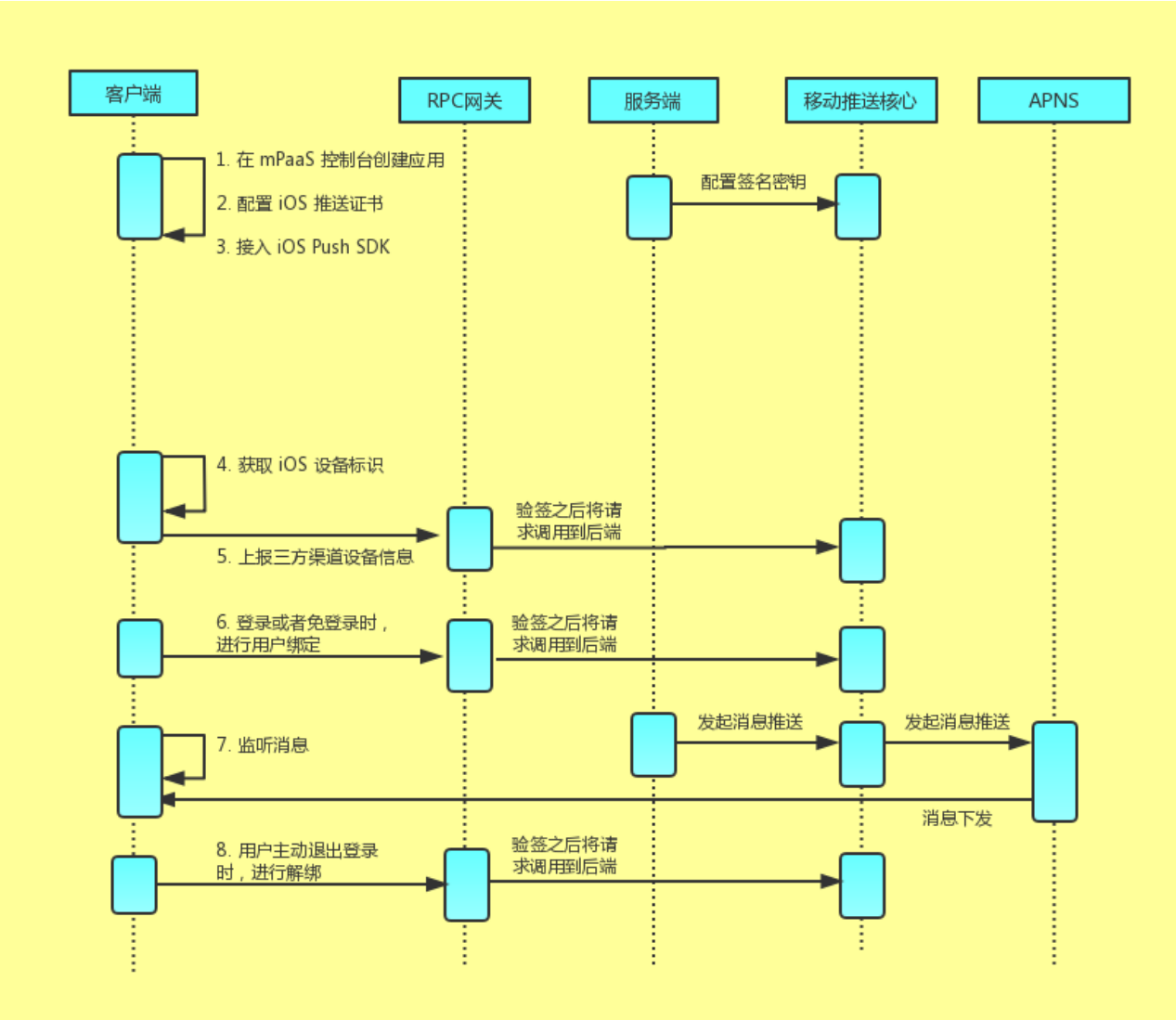
- 应用启动时，客户端同移动推送网关建立长链接，如果客户端建链信息中未携带设备标识，移动推送网关将下发设备标识。
- 如果用户开启小米、华为等等三方渠道，且客户端属于这些三方渠道的机型，那么这些三方渠道的SDK会进行初始化动作，与对应厂商的推送网关建立长连接服务并获取三方渠道设备标识。
- 获取三方渠道设备标识后，客户端调用设备上报 RPC 接口，上报三方渠道设备信息。
- 应用用户在客户端上发起登录。
- 服务端收到用户登录请求，用户登录成功，可以选择在此时向移动推送核心发起用户和设备绑定请求。
- 服务端发起推送请求。
- 移动推送核心获取到推送请求，移动推送核心根据推送类型进行区分：
 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

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 网关向移动推送核心发起用户和设备

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，如果用户配置了回调接口，移动推送核心会给服务端回执。

4 接入客户端

4.1 接入 Android

4.1.1 快速开始

本文介绍如何快速将消息推送组件接入到 Android 客户端。消息推送支持 **原生 AAR**、**mPaaS Inside** 和 **组件化（Portal & Bundle）** 三种接入方式。

完整的接入过程分为以下五步：

1. 添加 pushSDK ：添加 SDK 依赖。
2. 初始化 pushSDK ：建立客户端和移动推送网关的长连接，获取设备标识。
3. 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。
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，可在控制台页面上查看统计分析结果。

前置条件

- 已参考 通用步骤说明 完成基础配置。
    - 若采用原生 AAR 方式接入，需要先 将 mPaaS 添加到项目 。
    - 若采用 mPaaS Inside 方式接入，需要先完成 mPaaS Inside 接入流程 。
    - 若采用组件化方式接入，需要先完成 组件化接入流程 。
- 如果已经接入过 pushSDK，请检查一下工程是否引入了 pushservice 依赖。如果已经引入该依赖，则需要将其移除。
- 已通过 mPaaS 控制台获取 .config 配置文件。关于如何生成并下载配置文件，请参考 将配置文件添加到项目 。

添加 pushSDK

添加 SDK 依赖。

原生 AAR 方式

参考 AAR 组件管理 ，通过 **组件管理（AAR）**在工程中安装 **消息推送（PUSH）**组件。

mPaaS Inside 方式

在工程中通过 **组件管理** 安装 **消息推送（PUSH）**组件。

更多信息，请参考 添加组件依赖 。

组件化方式

在 Portal 和 Bundle 工程中通过 **组件管理** 安装 **消息推送（PUSH）**组件。

更多信息，请参考 添加组件依赖 。

初始化 pushSDK

初始化 pushSDK，建立客户端和移动推送网关的长连接，获取设备标识。

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

```
MPPush.setup(Application application);
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连接。如果客户端建连信息中未携带设备标识，移动推送网关在建连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。

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 PushMsgService extends AliPushRcvService {

private static final String TAG ="pushTag";
public static final String PUSH_SERVICE_ACTION ="tt-action";
//推送消息的类型，用户自定义即可
```

```
public static final int TYPE_MSG = -1;
public static final int TYPE_INNER_PUSH_INIT = -2;
public static final int TYPE_THIRD_PUSH_INIT = -3;
//自建渠道推送标识
public static String mAdToken ="";
//第三方渠道推送标识
public static String mThirdToken ="";

public static String mUserId ="mpaas_push_demo";

public static int platformType = 0;

public static boolean useDefault = false;

public PushMsgService() {
super();
}

/**
* 判断展示类消息是否使用内建通知。
* 返回true，则所有展示类（非静默）消息由mPaaS进行处理，handleActionReceived不再被调用。
* 返回false，表示透传所有消息。
*/
@Override
protected boolean useDefaultNotification(String msgKey, String msgValue) {
Intent intent = new Intent(PUSH_SERVICE_ACTION);
intent.putExtra("push_type", TYPE_MSG);
intent.putExtra("push_key", msgKey);
intent.putExtra("push_value", msgValue);
LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
LoggerFactory.getTraceLogger().debug(TAG,"onHandleIntent sendLocalBroadcast:" + intent.toString());
return useDefault;
}

/**
* 消息处理回调，自建消息需要接入方自行处理弹出通知等工作。
* @param msgKey 消息推送的 key
* @param msgValue 消息推送的 value
* @param clicked 是否已经点击了
* 对于三方渠道来说clicked = true，展示通知栏
* 对于自建渠道来说clicked = false，没有展示通知栏
*/
@Override
protected void handleActionReceived(String msgKey, String msgValue, boolean clicked) {
if (TextUtils.isEmpty(msgValue)) {
return;
}
BDataBean data = BDataBean.create(msgValue);
if (clicked) {
try {
Uri uri = Uri.parse(data.getUrl());
Intent actionIntent = new Intent(Intent.ACTION_VIEW);
actionIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
actionIntent.setData(uri);
actionIntent.putExtra("data", data.getParams());
startActivity(actionIntent);
```

```
} catch (Exception e) {
Log.e(TAG,"Unable start activity due to wrong format uri", e);
}
} else {
// show your notification and handle action
NotificationHelper helper = new NotificationHelper(getApplicationContext());
helper.notify("(注意：非内建消息)"+ data.getTitle(), data.getContent());
}

}

/**
* @param adToken 自建渠道的推送标识
*/

@Override
protected void handleActionId(String adToken) {
PushAppInfo pushAppInfo = new PushAppInfo(getApplicationContext());
pushAppInfo.setAppToken(adToken);
LoggerFactory.getTraceLogger().debug(TAG,"自建渠道的adToken:" + adToken);
Intent intent = new Intent(PUSH_SERVICE_ACTION);
intent.putExtra("push_type", TYPE_INNER_PUSH_INIT);
intent.putExtra("push_token", adToken);
LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
mAdToken = adToken;
}

/**
* @param thirdToken 三方渠道的推送标识
* @param platformType 三方渠道的类型 华为=5 小米=4 OPPO=7 VIVO=8
*/
@Override
protected void handleActionThirdId(String thirdToken, int platformType) {
LoggerFactory.getTraceLogger().debug(TAG,"第三方渠道的adToken:" + thirdToken +"platformType:" +
platformType);
Intent intent = new Intent(PUSH_SERVICE_ACTION);
intent.putExtra("push_type", TYPE_THIRD_PUSH_INIT);
intent.putExtra("push_thirdToken", thirdToken);
intent.putExtra("push_channel", platformType);
LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
mThirdToken = thirdToken;
PushMsgService.platformType = platformType;
}
}
```

在 AndroidManifest.xml 配置此 service。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：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** > **总览** 页面，点击 **代码配置** > 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"/>
<!-- PushMsgService 是指前文代码中定义的 service -->
<service android:name="com.mpaas.demo.push.PushMsgService"
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="${applicationId}"/>
</intent-filter>
</service>
<!-- 需要配置 alipush appId-->
<meta-data
android:name="ALIPUSH_APPID"
android:value="${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>
```

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 。

后续步骤

- 10.1.60 基线已支持内建通知消息，以满足展示类消息推送场景，具体方法参考 内建消息通知 。
- 为了提升消息推送的到达率，您可以选择接入 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，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 。

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 配置服务端 。

操作步骤如下：

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
// 此处的用户 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);
// 如果已经通过调用 MPLogger.reportUserLogin() 或 MPLogger.setUserId() 设置了 userId，也可以直接使用以下方法（版本须为 10.1.60.9 及以上，或 10.1.68.3 及以上）
ResultPbPB resultBean = MPPush.bind(getApplicationContext(), ad-token);
ResultPbPB resultBean = MPPush.unbind(getApplicationContext(), 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。

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。

到达

在接入 PushSDK 之后，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

参见上述 参数说明 。

代码示例

点击此处 下载示例代码包。

后续操作

接入完成后，可以通过服务端调用 RESTful 接口，推送消息。具体内容请参考 服务端配置 > 推送消息 。

4.1.2 接入第三方推送渠道

为提升推送的到达率，mPaaS 集成了华为、小米、OPPO 和 vivo 等厂商的推送功能。采用 小米通知栏消息、 华为通知消息 、OPPO 通知栏消息 和 vivo 通知栏消息 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 和 接入 MpaaSNcActivity 两部分，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、OPPO 和 vivo 四家厂商的推送服务所需要进行的客户端配置。

- 接入华为渠道
- 接入小米渠道
- 接入 OPPO 渠道
- 接入 vivo 渠道

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

MPS 支持接入华为 HMS2 和 HMS5，但在接入华为推送组件过程中，您只能选择其中一个版本进行接入。下面分别对华为 HMS2 和 HMS5 这两个版本的接入方式进行说明。

接入华为推送 - HMS2.x 版本

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 。
2. 在 IDE 插件中添加 **推送 - 华为2** 组件。

   配置 AndroidManifest.xml。

   如果是基于 mPaaS 框架的工程，请在 Portal 工程的 AndroidManifest.xml 配置文件中进行配置。

```
<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。注意，value 中的斜杠（\）及空格要保留。-->
<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"
android:process=":push">
<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>
```

4. 接入 MpaaSNcActivity 。

接入华为推送 - HMS5.x 版本

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 。

2. 在 IDE 插件中添加 **推送 - HMS5** 组件。
3. 在华为应用服务控制台下载配置文件 agconnect-services.json 并放置到应用主工程的 assets 目录下。
4. 在项目根目录下的 build.gradle 文件中配置 HMS Core SDK 的 Maven 仓地址。

```
allprojects {
repositories {
// 其他repo已省略
maven {url 'https://developer.huawei.com/repo/'}
}
}
```

在主工程的 build.gradle 文件中添加 HMS Core SDK 依赖。

```
dependencies {
implementation 'com.huawei.hms:push:5.0.2.300'
// 华为HMS Core SDK版本会经常更新，最新版本号可以参考
https://developer.huawei.com/consumer/cn/doc/HMSCore-Guides-V5/version-change-history-
0000001050040072-V5
}
```

6. 接入 MpaaSNcActivity 。
7. 如果您是从 2.x 版本升级到 5.x 版本，请卸载已安装的 **推送 - 华为2** 组件，并移除相应的 AndroidManifest.xml 文件中的华为推送配置。

应用角标处理

mPaaS 自 10.1.60.13 基线版本起支持应用角标处理。当服务端推送携带角标信息的消息到客户端时，SDK 提供接口处理应用图标上的角标信息。

MPPush 类提供以下三个处理应用角标的接口：

- 设置是否允许点击通知栏消息时自动清除应用角标消息。enable 为 true 时表示允许，false 表示不允许。

```
void setBadgeAutoClearEnabled(Context context, boolean enabled)
```

- 设置角标展示所在的应用图标对应的 Activity 名称。必须设置，否则无法清除角标信息。

```
void setBadgeActivityClassName(Context context, String name)
```

- 清除应用角标信息。SDK 不能处理角标信息的情况下可以使用此方法主动清除角标信息，例如用户主动点击应用图标进入应用时清除角标信息。

```
void clearBadges(Context context)
```

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 。
2. 在 IDE 插件中添加 **推送 - 小米** 组件。

   配置 AndroidManifest.xml。

   如果是基于 mPaaS 框架的工程，请在 Portal 工程的 AndroidManifest.xml 配置文件中进行配置。

```
<!--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 中的斜杠（\）及空格要保留 -->
<meta-data
android:name="xiaomi_appid"
android:value="\ 2xxxxxxxxxx"/>
<!-- value 中的斜杠（\）及空格要保留 -->
<meta-data
android:name="xiaomi_appkey"
android:value="\ 5xxxxxxxxxxxx"/>
<service android:name="com.xiaomi.push.service.XMJobService"
android:enabled="true"
android:exported="false"
android:permission="android.permission.BIND_JOB_SERVICE"
android:process=":push"/>
<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>
```

4. 接入 MpaaSNcActivity 。

接入 OPPO 渠道

说明：仅 10.1.60 及以上基线版本支持接入 OPPO 推送渠道。如果您使用了较早版本基线提供的 AliPushInterface 类进行初始化，请替换成使用 MPPush 类。

注册 OPPO 推送

参考 OPPO推送平台使用指南 在 OPPO 开放平台 注册账号并申请接入推送服务。

客户端接入 OPPO 推送

1. mPaaS 不提供 OPPO Push SDK 的库，请前往 OPPO SDK 文档 下载 SDK 并集成到主工程中。

说明：当前适配的版本为 2.0.2。

2. 在 IDE 插件中添加 **推送 - OPPO** 组件。

   配置 AndroidManifest.xml。

   如果是基于 mPaaS 框架的工程，请在 Portal 工程的 AndroidManifest.xml 配置文件中进行配置。

```
<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.mcssdk.PushService"
android:permission="com.coloros.mcs.permission.SEND_MCS_MESSAGE"
android:process=":push">
<intent-filter>
<action android:name="com.coloros.mcs.action.RECEIVE_MCS_MESSAGE"/>
</intent-filter>
</service>

<service
android:name="com.heytap.mcssdk.AppPushService"
android:permission="com.heytap.mcs.permission.SEND_MCS_MESSAGE"
android:process=":push">
<intent-filter>
<action android:name="com.heytap.mcs.action.RECEIVE_MCS_MESSAGE"/>
</intent-filter>
</service>
<meta-data
android:name="com.oppo.push.app_key"
android:value="OPPO开放平台获取"
/>
<meta-data
android:name="com.oppo.push.app_secret"
android:value="OPPO开放平台获取"
/>
<meta-data
android:name="mpaas.notification.channel.default.name"
android:value="默认"
/>
<meta-data
android:name="mpaas.notification.channel.default.description"
android:value="默认消息通知"
/>
</application>
```

4. 接入 MpaaSNcActivity 。

接入 vivo 渠道

说明：仅 10.1.60 及以上基线版本支持接入 vivo 推送渠道。

注册 vivo 推送

参考 vivo 推送平台使用指南，在 vivo 开放平台 上注册账号，并申请接入推送服务。

客户端接入 vivo 推送

1. mPaaS 不提供 vivo Push SDK 的库，请前往 vivo SDK 文档 下载 SDK 并集成到主工程中。
2. 在 IDE 插件中添加 **推送 - vivo** 组件。

   配置 AndroidManifest.xml。

如果是基于 mPaaS 框架的工程，请在 Portal 工程的 AndroidManifest.xml 配置文件中进行配置。

```
<application>
<service
android:name="com.vivo.push.sdk.service.CommandClientService"
android:process=":push"
android:exported="true"/>
<activity
android:name="com.vivo.push.sdk.LinkProxyClientActivity"
android:exported="false"
android:process=":push"
android:screenOrientation="portrait"
android:theme="@android:style/Theme.Translucent.NoTitleBar"/>
<meta-data
android:name="com.vivo.push.api_key"
android:value="VIVO开放平台提供"/>
<meta-data
android:name="com.vivo.push.app_id"
android:value="VIVO开放平台提供"/>
<receiver
android:name="com.alipay.pushsdk.thirdparty.vivo.PushMessageReceiver"
android:process=":push">
<intent-filter>
<!-- 接收push消息 -->
<action android:name="com.vivo.pushclient.action.RECEIVE"/>
</intent-filter>
</receiver>
</application>
```

4. 接入 MpaaSNcActivity 。

接入 MpaaSNcActivity

在不接入 MpaaSNcActivity 的情况下，第三方渠道消息将无法跳转。

```
<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>
```

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 。

4.1.3 接入 FCM 推送渠道

消息推送支持集成 Firebase 云信息传递（Firebase Cloud Messaging，简称 FCM）通道，以满足 App 在海外安卓设备上的使用需求。

下面介绍 FCM 推送渠道的接入过程。

前提条件

接入 FCM 前，需要满足以下几个条件：

- 采用原生 AAR 方式接入。FCM 仅支持以原生 AAR 方式接入，不支持 mPaaS Inside 和 Portal & Bundle 接入。
- Gradle 版本大于 4.1。
- 使用 AndroidX。
- com.android.tools.build:gradle 为 3.2.1 或以上版本。
- compileSdkVersion 为 28 或以上版本。

接入 FCM SDK

操作步骤如下：

在 Firebase 控制台添加应用。

登录 Firebase 控制台，参考 Firebase 文档 完成应用注册。

将 Firebase Android 配置文件添加到您的应用。

下载 google-services.json 配置文件，将配置文件放置到项目主 module 目录下。

在根目录build.gradle 的 buildScript 依赖中加入 Google 服务插件。

```
buildscript {

repositories {
// Check that you have the following line (if not, add it):
google() // Google's Maven repository
}

dependencies {
// ...

// Add the following line:
classpath 'com.google.gms:google-services:4.3.4' // Google Services plugin
}
}

allprojects {
// ...

repositories {
```

```
// Check that you have the following line (if not, add it):
google() // Google's Maven repository
// ...
}
}
```

在主 module 工程的 build.gradle 中应用 Google 服务插件。

```
apply plugin: 'com.android.application'
// Add the following line:
apply plugin: 'com.google.gms.google-services' // Google Services plugin

android {
// ...
}
```

在主 module 工程的 build.gradle 中加入 FCM SDK 依赖。

```
dependencies {
// Import the BoM for the Firebase platform
implementation platform('com.google.firebase:firebase-bom:26.1.1')

// Declare the dependencies for the Firebase Cloud Messaging and Analytics libraries
// When using the BoM, you don't specify versions in Firebase library dependencies
implementation 'com.google.firebase:firebase-messaging'
implementation 'com.google.firebase:firebase-analytics'
}
```

接入 mPaaS

操作步骤如下：

1. 接入 MPS 依赖，mPaaS 基线版本不低于 10.1.68.19。

   在主 module 工程的 build.gradle 中加入 FCM Adapter 依赖。

```
dependencies {
implementation 'com.mpaas.push:fcm-adapter:0.0.1@aar'
}
```

在AndroidManifest.xml 文件中加入 MPaaSNcActivity。如果已加入，请检查是否与下方配置一致。

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

```
</intent-filter>
</activity>
```

接收推送消息。

当应用在前台时，FCM 消息以透传消息方式传递给应用，您可以在自己实现的 AliPushRcvService 派生类的 handleActionReceived 方法中收到消息对象，此时 clicked 参数值为 false。

当应用在后台时，FCM 消息以通知栏消息展示，点击通知栏后 handleActionReceived 方法被调用，此时 clicked 参数值为 true。

（可选）可通过注册消息接收器来获取 FCM 初始化失败的错误信息，具体参见 错误码说明文档。

示例如下：

```
<receiver android:name=".push.FcmErrorReceiver"android:exported="false">
<intent-filter>
<action android:name="action.mpaas.push.error.fcm.init"/>
</intent-filter>
</receiver>


package com.mpaas.demo.push;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class FcmErrorReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
String action = intent.getAction();
if ("action.mpaas.push.error.fcm.init".equalsIgnoreCase(action)) {
Toast.makeText(context,"fcm error" + intent.getIntExtra("error", 0), Toast.LENGTH_SHORT).show();
}
}
}
```

4.1.4 使用内建消息通知

在新版本的消息推送 SDK 中，mPaaS 支持内建消息通知，即：

- 当展示类型消息（非静默消息）到达时 SDK 将自动弹出系统通知并处理跳转事件。
- 消息到达、打开、忽略埋点由 SDK 自行处理，无需您手动调用。

启用内建消息通知

默认情况下，内建消息通知功能是关闭的。打开该功能需要您在其实现的 AliPushRcvService 子类中覆写

useDefaultNotification 方法并返回 true。

```
protected abstract boolean useDefaultNotification(String msgKey, String msgValue);
```

启用内建消息通知后，展示类型消息推送至客户端时，handleActionReceived 方法不再被调用，如果您仍希望获取消息内容，可在 useDefaultNotification 中进行处理。

通知消息跳转

当用户点击消息时，消息推送 SDK 将根据消息携带的 URL 参数进行跳转。当 URL 为 Web 网页时，浏览器将被启动被打开此 URL。当 URL 为自定义 URI 时，接入应用必须能够接收此 URI 并进行正确处理。步骤如下：

在 AndroidManifest.xml 中声明可接收自定义 URI 的 Activity。下方代码示例中表明 LauncherActivity 可以接收 mpaas:// 的 URI。

```
<activity android:name=".push.LauncherActivity"
android:launchMode="singleInstance">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
<intent-filter>
<action android:name="android.intent.action.VIEW"/>
<category android:name="android.intent.category.BROWSABLE"/>
<category android:name="android.intent.category.DEFAULT"/>
<data android:scheme="mpaas"/>
</intent-filter>
</activity>
```

在 Activity 中处理通知消息的 Intent 对象，可通过 Intent.getData() 方法获取 URI 并进行跳转。通知消息携带的额外参数可通过 Intent.getStringExtra("data") 获取。

注意事项

SDK 使用的消息通知 ID 从 10000 开始，请避免您使用的其他通知栏消息 ID 与 SDK 使用的消息通知 ID 冲突。

4.2 接入 iOS

> **重要**：自 2020 年 6 月 28 日起，mPaaS 停止维护 10.1.32 基线。请使用 10.1.68 或 10.1.60 系列基线。可以参考 mPaaS 10.1.68 升级指南 或 mPaaS 10.1.60 升级指南 进行基线版本升级。

本文将向您详细介绍将消息推送服务接入 iOS 客户端的接入流程。为了使用消息推送服务，您需要完成以下流程：

1. 选择接入方式并接入
2. 配置工程

3. 使用 SDK
4. 配置推送证书

选择接入方式并接入

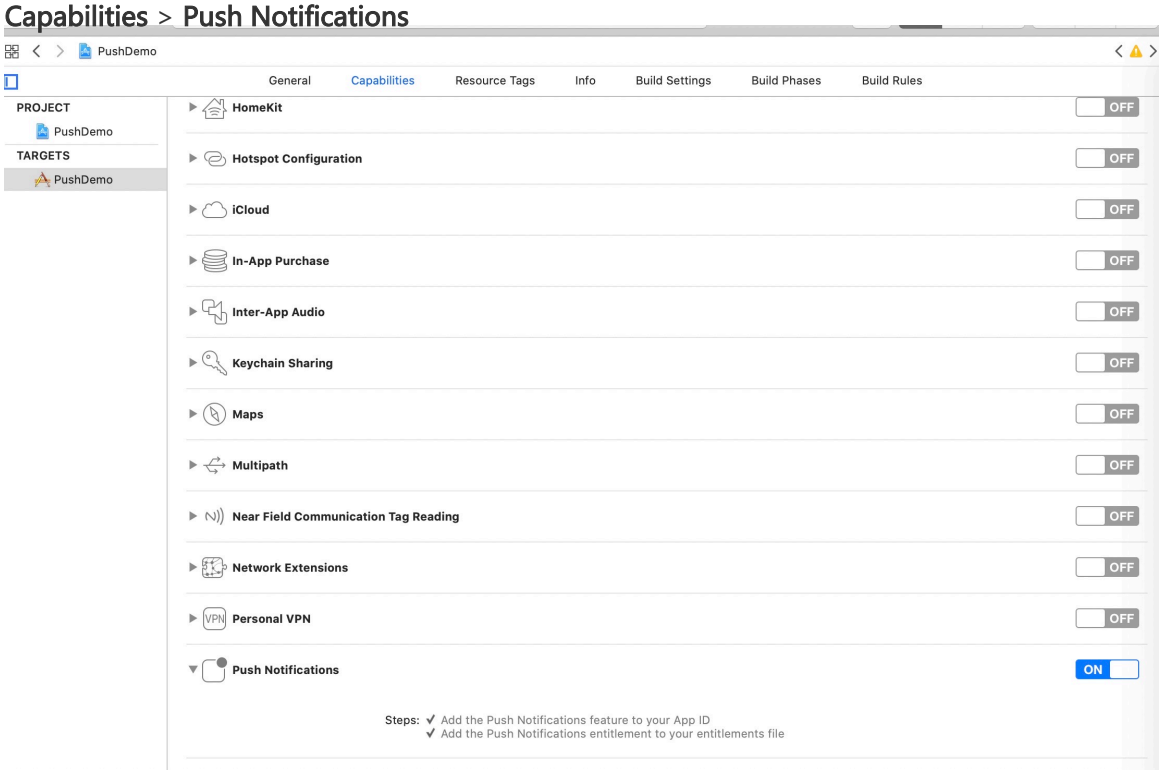
消息推送服务支持以下三种接入方式，您可以根据实际情况进行选择。关于接入方式的更多信息，请参见 接入方式简介 。在确定接入方式后，请您参考各接入方式的接入文档，添加消息推送 SDK 完成接入。

- 基于 mPaaS 框架接入
- 基于已有工程且使用 mPaaS 插件接入
- 基于已有工程且使用 CocoaPods 接入

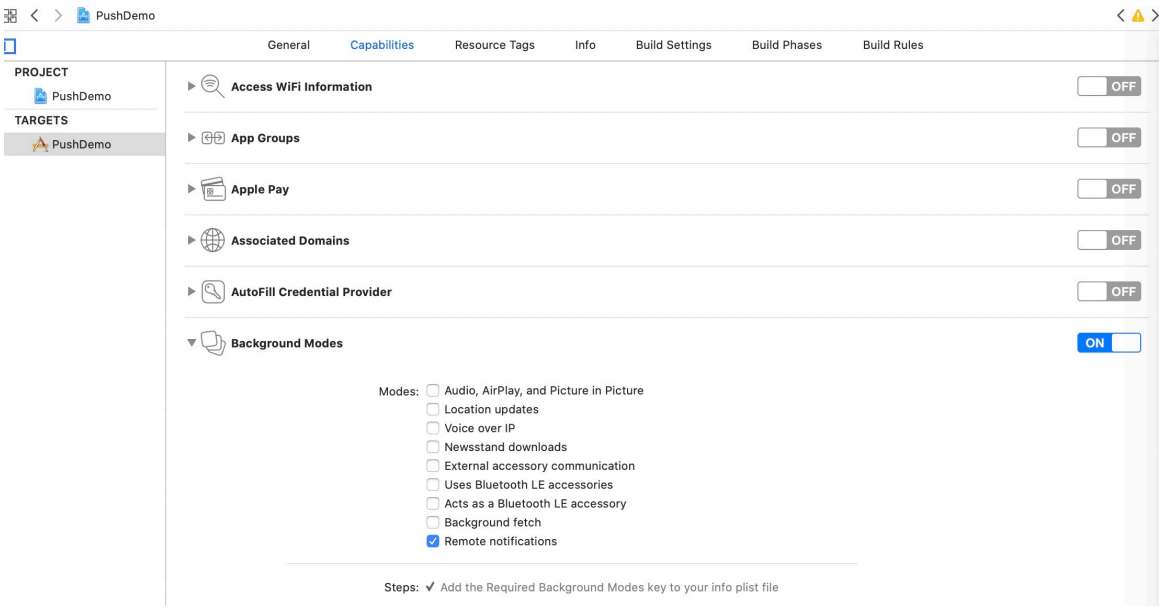
配置工程

需要在工程 target 设置中开启以下两处：

- Capabilities > Push Notifications



- Capabilities > Background Modes > Remote notifications



`

使用 SDK

下面对不同接入方式的 SDK 使用进行说明。

基于 mPaaS 框架

注册 deviceToken（非必需）

PushSDK 在应用启动完成时，会自动请求注册 deviceToken，一般情况下您无需请求注册 deviceToken。但是当特殊情况下（比如启动时有隐私管控，阻止一切网络请求时）您需要在管控授权后，再次触发注册 deviceToken，示例代码如下：

```
- (void)registerRemoteNotification
{
// 注册推送
if ([[[UIDevice currentDevice] systemVersion] floatValue] >= 10.0) {// 10.0+
UNUserNotificationCenter* center = [UNUserNotificationCenter currentNotificationCenter];
center.delegate = self;
[center getNotificationSettingsWithCompletionHandler:^(UNNotificationSettings * _Nonnull settings) {

[center
requestAuthorizationWithOptions:(UNAuthorizationOptionAlert|UNAuthorizationOptionSound|UNAuthorizationOptionBadge)
completionHandler:^(BOOL granted, NSError * _Nullable error) {
// Enable or disable features based on authorization.
if (granted) {
dispatch_async(dispatch_get_main_queue(), ^{
[[UIApplication sharedApplication] registerForRemoteNotifications];
});
}
}];
```

```
}];
} else {// 8.0，9.0
UIUserNotificationSettings *settings = [UIUserNotificationSettings settingsForTypes:(UIUserNotificationTypeBadge
|UIUserNotificationTypeSound|UIUserNotificationTypeAlert) categories:nil];
[[UIApplication sharedApplication] registerUserNotificationSettings:settings];
[[UIApplication sharedApplication] registerForRemoteNotifications];
}
}
```

获取 deviceToken 并绑定 userId

与基于 iOS 原生框架相比，基于 mPaaS 框架的应用的生命周期被 mPaaS 框架接管，应用获取 deviceToken 的回调方法有所不同，代码示例如下：

```
// import <PushService/PushService.h>
// 在 DTFrameworkInterface 分类中重写如下方法
- (DTFrameworkCallbackResult)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
[[PushService sharedService] setDeviceToken:deviceToken];
[[PushService sharedService] pushBindWithUserId:@"your userid(需替换)"completion:^(NSException *error) {
}];

return DTFrameworkCallbackResultContinue;
}
```

说明：Push SDK 同时提供了解绑的接口 - (void)pushUnBindWithUserId:(NSString *)userId completion:(void (^)(NSException *error))completion;，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。

接收推送消息

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，收到消息的回调方法不同，代码示例如下：

```
// import <PushService/PushService.h>
// 在 DTFrameworkInterface 分类中重写如下方法
- (DTFrameworkCallbackResult)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
*)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler
{
// userInfo 为推送消息内容，业务解析处理
return DTFrameworkCallbackResultContinue;
}
```

基于已有工程

在基于已有工程且使用 mPaaS 插件或 CocoaPods 接入 iOS 客户端的情况下，您需要完成以下操作。

注册 deviceToken（非必需）

PushSDK 在应用启动完成时，会自动请求注册 deviceToken，一般情况下您无需请求注册 deviceToken。但是当特殊情况下（比如启动时有隐私管控，阻止一切网络请求时）您需要在管控授权后，再次触发注册 deviceToken，示例代码如下：

```
- (void)registerRemoteNotification
{
// 注册推送
if ([[[UIDevice currentDevice] systemVersion] floatValue] >= 10.0) {// 10.0+
UNUserNotificationCenter* center = [UNUserNotificationCenter currentNotificationCenter];
center.delegate = self;
[center getNotificationSettingsWithCompletionHandler:^(UNNotificationSettings * _Nonnull settings) {

[center
requestAuthorizationWithOptions:(UNAuthorizationOptionAlert|UNAuthorizationOptionSound|UNAuthorizationOptionBadge)
completionHandler:^(BOOL granted, NSError * _Nullable error) {
// Enable or disable features based on authorization.
if (granted) {
dispatch_async(dispatch_get_main_queue(), ^{
[[UIApplication sharedApplication] registerForRemoteNotifications];
});
}
}];

}];
} else {// 8.0，9.0
UIUserNotificationSettings *settings = [UIUserNotificationSettings settingsForTypes:(UIUserNotificationTypeBadge
|UIUserNotificationTypeSound|UIUserNotificationTypeAlert) categories:nil];
[[UIApplication sharedApplication] registerUserNotificationSettings:settings];
[[UIApplication sharedApplication] registerForRemoteNotifications];
}
}
```

获取 deviceToken 并绑定 userId

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，上报绑定 userId 至移动推送核心。

```
// import <PushService/PushService.h>
- (void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
[[PushService sharedService] setDeviceToken:deviceToken];
[[PushService sharedService] pushBindWithUserId:@"your userid(需替换)"completion:^(NSException *error) {
}];

}
```

说明：Push SDK 同时提供了解绑的接口 - (void)pushUnBindWithUserId:(NSString *)userId completion:(void (^)(NSException *error))completion;，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。

接收推送消息

客户端收到 push 消息后，如果用户点击查看，系统将启动相应应用。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：

```
// iOS 10 以下 Push 冷启动处理
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {
NSDictionary *userInfo = [launchOptions objectForKey:
UIApplicationLaunchOptionsRemoteNotificationKey];
if ([[[UIDevice currentDevice] systemVersion] doubleValue] < 10.0) {
// iOS 10 以下 Push 冷启动处理
}

return YES;
}

// 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，您需要实现以下代理方法来监听通知栏消息：

```
// 注册 UNUserNotificationCenter delegate
if ([[[UIDevice currentDevice] systemVersion] doubleValue] >= 10.0) {
UNUserNotificationCenter* center = [UNUserNotificationCenter currentNotificationCenter];
center.delegate = self;
}

//应用处于前台时的远程推送接受
- (void)userNotificationCenter:(UNUserNotificationCenter *)center willPresentNotification:(UNNotification
*)notification withCompletionHandler:(void (^)(UNNotificationPresentationOptions options))completionHandler
{
NSDictionary *userInfo = notification.request.content.userInfo;

if([notification.request.trigger isKindOfClass:[UNPushNotificationTrigger class]]) {
//应用处于前台时的远程推送接受

} else {
//应用处于前台时的本地推送接受

}
completionHandler(UNNotificationPresentationOptionNone);
}

//应用处于后台或者活冷启动时远程推送接受
- (void)userNotificationCenter:(UNUserNotificationCenter *)center
didReceiveNotificationResponse:(UNNotificationResponse *)response
withCompletionHandler:(void(^)(void))completionHandler
{
NSDictionary *userInfo = response.notification.request.content.userInfo;

if([response.notification.request.trigger isKindOfClass:[UNPushNotificationTrigger class]]) {
```

```
//应用处于后台或者活冷启动时远程推送接受

} else {
//应用处于前台时的本地推送接受

}
completionHandler();

}
```

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 。

代码示例

点击此处 下载示例代码包。

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。消息推送服务使用苹果的 APNs 服务向客户端推送消息，更多信息请参见 苹果及国外安卓设备的推送流程 。
- 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

关于各推送类型的定义，参见 推送类型 。

相关链接

- 创建消息
- 配置服务端

5 配置服务端

通过 推送流程 文档了解移动推送服务的总体流程后，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

- 已经开通 mPaaS 产品。
- 已经有一个服务端应用。

- 已经在客户端上报用户 ID 和 设备 ID。

操作步骤

1. 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接口来完成绑定。
接口文档参见 API 说明 - 客户端 或 API 说明 - 服务端 。

2. 推送消息

服务端可以通过调用接口，推送以下类型的消息：

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

6 使用控制台

6.1 消息列表

6.1.1 创建消息

消息推送服务（Message Push Service，简称 MPS）支持创建多种推送类型的消息，并基于不同维度（用户标识或设备标识）推送消息。下面对如何通过控制台创建推送消息进行具体说明。

推送类型

MPS 支持极简推送、模板推送、批量推送以及群发推送四种推送类型。

- 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 推送 SDK 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 支持直接调用移动分析人群作为推送目标，同时也支持通过上传文件的方式，指定推送目标。
    - 调用移动分析人群：支持对移动分析人群推送相同消息，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下，通过控制台页面推送消息。
- 消息一旦创建成功即进行推送，您将无法删除或修改。

针对不同的推送类型以及不同的推送平台，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
    - 当安卓消息通道为 **MPS 自建通道** 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 推送平台：提供 **默认通道**（即安卓厂商通道） 和 **MPS 自建通道** 推送消息。
  - iOS 推送平台：使用三方渠道推送消息。

前置条件

- 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，若选择移动分析人群作为推送目标人群，则需要事先创建好移动分析人群，操作参见 创建用户群组 。
- 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 。

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 **消息推送**，然后进入 **消息列表** 标签页。
2. 点击 **新建推送消息** 按钮，页面上弹出 **新建推送消息** 对话框。

3. 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，配置推送消息的相关信息。具体参见 配置项说明 - 基础信息 和 配置项说明 - 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。

4. 点击 **提交** 按钮，移动推送核心即推送消息至推送 ID。

配置项说明

在 **新建推送消息** 对话框内创建消息，对话框分三个区域：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**：推送目标设备为安卓手机。 • **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 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与所选 **推送平台** 一致，否则将导致推送失败。当目标 ID 类型为 **UserId** 时，目标 ID 为用户标识，即用户调用绑定接口时传入的 userid 值。 • 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。
安卓消息通道	是	可选： • **默认通道**：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。 • **MPS 自建通道**：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（通知栏消息）**：指在通知栏展示消息。 • **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	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**：推送目标设备为安卓手机。 • **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 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与所选 **推送平台** 一致，否则将导致推送失败。当目标 ID 类型为 **UserId** 时，目标 ID 为用户标识，即用户调用绑定接口时传入的 userid 值。 • 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。
安卓消息通道	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（通知栏消息）** 时：提供 **默认通道** 和 **MPS 自建通道** 选项。 • 模板展示类型为 **静默消息** 时：系统默认选择 **MPS 自建通道**，您无法修改。 各选项说明如下： • **默认通道**：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。 • **MPS 自建通道**：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（通知栏消息）**：指在通知栏展示消息。 • **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**：推送目标设备为安卓手机。 • **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：推送 ID,业务方消息 ID,占位符1= XXX,占位符2=XXX.....，其中推送 ID 对应目标 ID（与目标 ID 类型保持一致），业务方消息 ID 由用户自定义。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，点击图标，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** • 支持调用移动分析人群，您需要先创建移动分析用户群组，操作方法参见 创建用户群组 。 支持对移动分析人群推送相同消息，所选推送模板中包含占位符时，移动分析人群 将不可选。当所选目标 ID 类型为设备 ID 时，**移动分析人群** 选项不可选。
安卓消息通道	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（通知栏消息）** 时：提供 **默认通道** 和 **MPS 自建通道** 选项。 • 模板展示类型为 **静默消息** 时：系统默认选择 **MPS 自建通道**，您无法修改。 各选项说明如下： • **默认通道**：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。 • **MPS 自建通道**：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（通知栏消息）**：指在通知栏展示消息。

		• **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 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**：提供安卓厂商渠道和 MPS 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。
安卓消息通道	是	• **MPS 自建通道**：向推送消息有效期内的所有在线安卓用户推送消息。 • **默认通道**：推送对象为所有安卓用户。如果接入了厂商通道，则通过厂商通道推送、否则通过自建通道推送。如果选择默认通道，请先接入安卓各厂商渠道。 注意：由于默认通道会对所有安卓用户推送，推送数量会远大于消息有效期内在线用户，同时可能会造成计费增长。如果未接入厂商渠道，使用默认通道，实际上是对所有用户尝试使用自建渠道推送，实际收到消息的用户就是消息有效期内在线的用户，效果和 MPS 自建通道一致。
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（通知栏消息）**：指在通知栏展示消息。

		• **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，设备建连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。
  扩展参数包含以下三类：

  - 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。系统具有一定意义的扩展参数及其说明参见下表。

参数	说明
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。
badge_add_num	华为渠道推送角标增加数。
badge_class	华为渠道桌面图标对应的应用入口 Activity 类。
big_text	大文本样式，value 固定为 1，填写其他值无效。本参数仅对小米和华为手机有效。

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。

6.1.2 管理消息

消息列表默认展示最近 30 天内创建的 **极简推送** 和 **模板推送** 消息相关信息，支持查询最近 30 天的极简推送、模板推送以及批量推送类型的历史消息。您可通过消息列表，定位目标消息，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 **消息推送**，进入 **消息推送** 页面。

   在右侧页面上，点击 **消息列表** 标签，进入 **消息列表** 标签页。

消息默认按创建时间倒序排列，列表中展示的信息包括：

- **业务方消息 ID**：用户自定义或系统自动生成。

  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**：具体的用户 ID 或设备 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**：由用户自定义或系统自动生成，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连接断开）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，出现该状态的原因如下： • 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。在开发环境（控制台配置为开发环境证书）下，需要使用个人开发证书打包App进行测试。否则会出现BadDeviceToken。
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**），目标设备建连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。

   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。

6.1.3 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

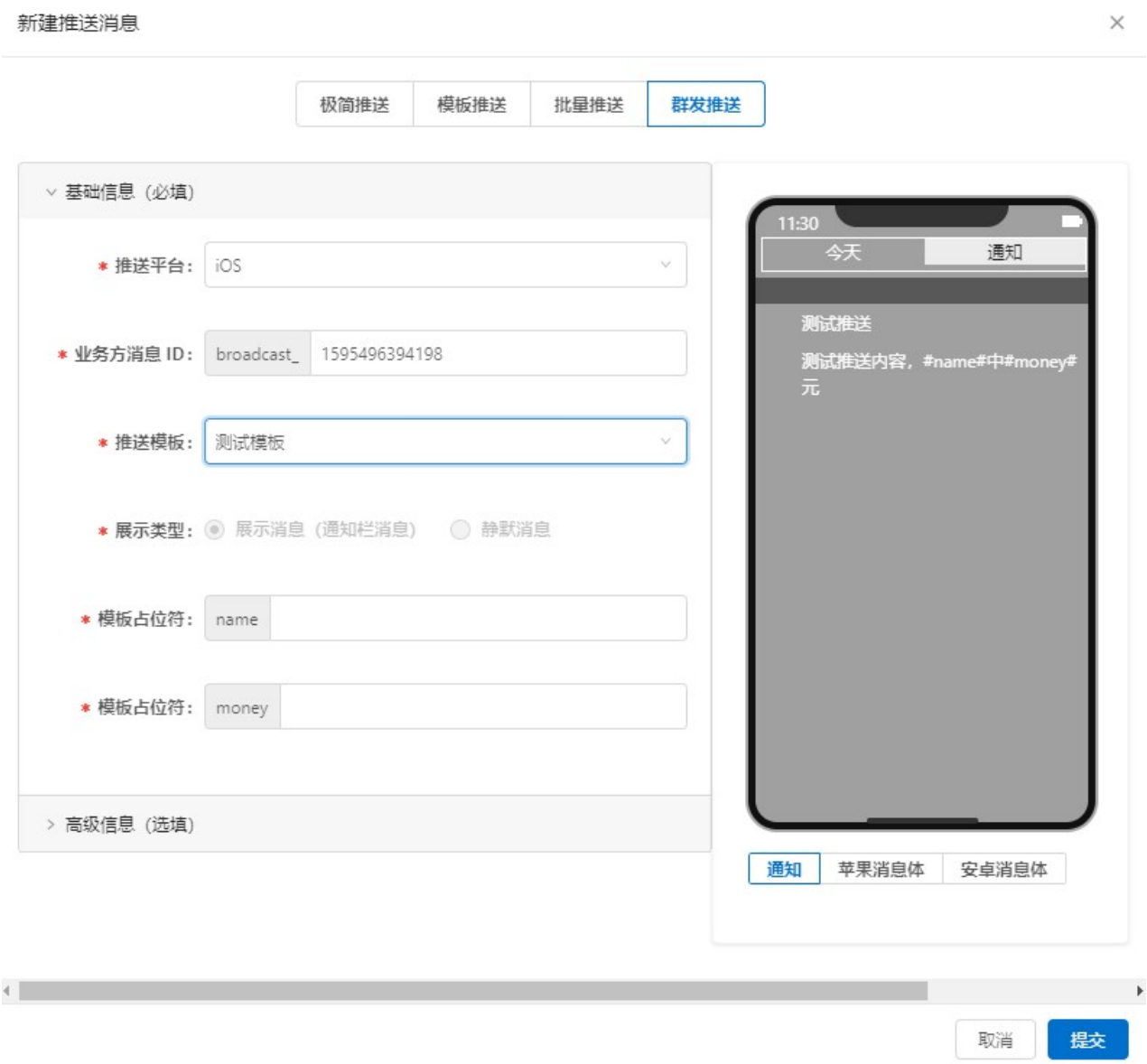
目前支持的推送方式如下：

- 极简推送
- 模板推送
- 批量推送

- 群发推送

极简推送、模板推送、批量推送是根据目标 ID 类型来推送消息；群发推送则直接按平台来推送消息。

- 当目标 ID 类型为用户标识（UserId）时，需要调用绑定接口，绑定用户标识和设备标识，具体信息参见 API 参考 。
  按 UserId 推送消息时：
  - 如果目标 UserId 绑定的是安卓设备，则根据安卓消息通道配置进行推送；
  - 如果目标 UserId 绑定的是 iOS 设备，则统一使用三方渠道推送。
- 当目标 ID 类型为设备标识（DeviceId）时，需要选择推送平台，明确推送设备类型。



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 。

6.2 使用分析

> **重要**：自 2020 年 6 月 28 日起，mPaaS 停止维护 10.1.32 基线。请使用 10.1.68 或 10.1.60 系列基线。可以参考 mPaaS 10.1.68（ Andorid / iOS ）或 mPaaS 10.1.60（ Andorid / iOS ）升级指南进行基线版本升级。

MPS 依赖客户端 SDK 埋点上报的数据，在推送成功、到达、打开和忽略四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。

查看使用分析报告

完成以下步骤，查看消息推送使用分析数据：

1. 登录 mPaaS 控制台，选择目标应用。
2. 从左侧导航栏进入 **消息推送** 页面。
3. 在右侧页面上，单击 **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，则可选项中将不包含 iOS–workspaceId 相关选项。WorkspaceId 为发起推送的控制台所在工作空间。
- **版本**：依赖客户端 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

统计图表分两部分呈现：

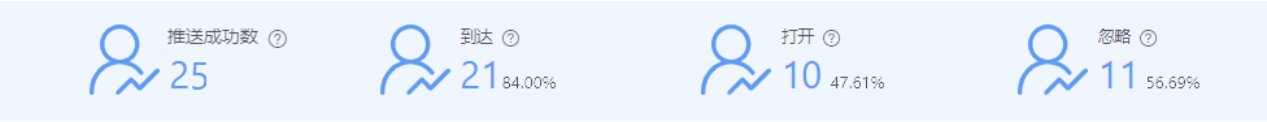
- 推送核心指标
- 推送详细数据

推送核心指标

MPS 分数据和图表两种形式来展示推送核心指标，包括推送成功、到达、打开和忽略四个指标。

数据展示

以数字形式展示消息推送的统计结果。



- **推送成功数**
  MPS 服务端自动统计，指实际推送成功的消息数量。
  - 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%

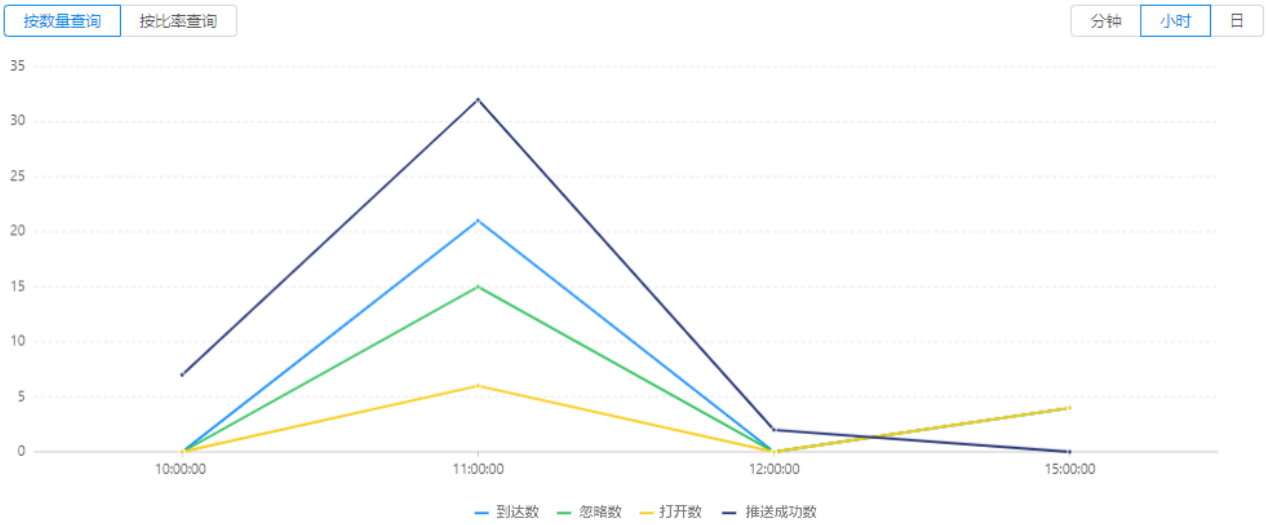
  对于不同的推送渠道，到达数据的统计方式不同：

  - 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：依赖客户端 SDK 埋点上报，MPS 调用移动分析统计结果。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 调用移动分析统计结果。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

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。

详细数据　　导出

时间	推送成功数	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%)

- 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 **推送成功数**、**到达数**（**到达率**）、**打开数**（**打开率**） 和 **忽略数**（**忽略率**）四个指标。

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。

6.3 消息模板

6.3.1 创建模板

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。

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。模板推送、批量推送以及群发推送都需要用到模板。

下面对如何配置推送模板进行详述。配置模板时，可以通过 **#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。

操作步骤

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 **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，访问原生页面。 • **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（通知栏消息）**：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。

6.3.2 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。

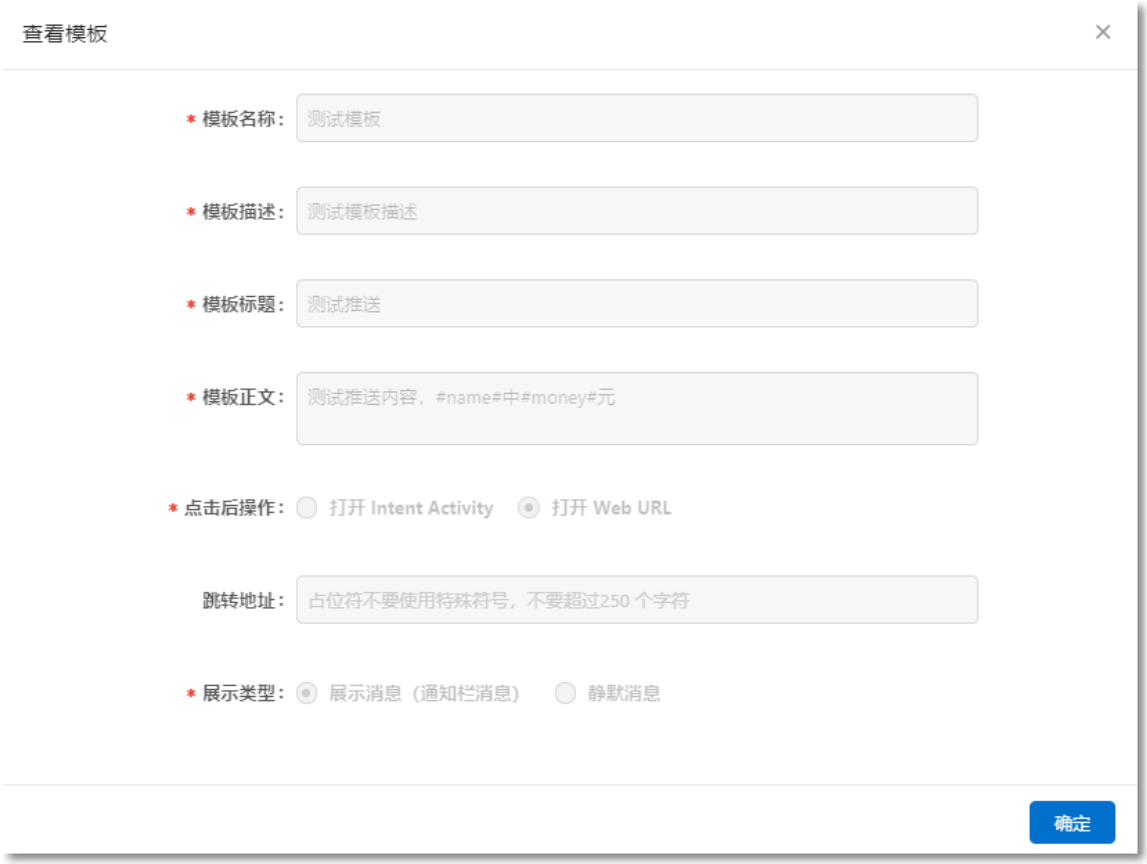
查看模板

登录 mPaaS 控制台，选择 App，通过以下步骤，查看模板信息：

1. 在左侧导航栏中，选择 **消息推送**，进入 **消息推送** 页面。

   在右侧页面上，点击 **消息模板** 标签，进入消息模板列表页面。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。

   点击 **操作** 列下的 **查看**，可查看对应模板详情。



删除模板

操作方法如下：

1. 在列表中，点击目标模板对应的 **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：删除模板前，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。

6.4 渠道配置

为提升推送的到达率，mPaaS 集成了华为、小米、OPPO 和 vivo 等厂商的推送功能。采用 **小米通知栏消息**、**华为通知栏消息**、**OPPO 通知栏消息** 和 **vivo 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小米、华为、OPPO、和 vivo 推送渠道时需要进行的控制台侧配置。

- 配置华为推送渠道
- 配置小米推送渠道
- 配置 OPPO 推送渠道
- 配置 vivo 推送渠道

前置条件

您需要先完成客户端侧的接入配置，操作参见 接入第三方推送渠道 。

操作方法

配置华为推送渠道

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 **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，进入渠道配置页面。

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。
华为应用密钥	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。

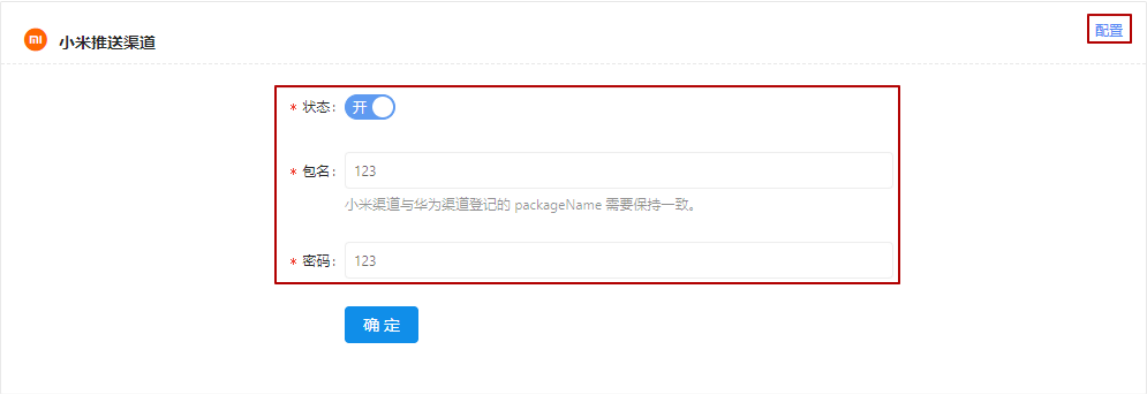
4. 点击 **确定** 按钮，保存配置。

配置小米推送渠道

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 **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，进入渠道配置页面。

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。

配置 OPPO 推送渠道

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

1. 在左侧导航栏中，选择 **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，进入渠道配置页面。

点击 **OPPO 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 OPPO 推送渠道；关闭开关，即取消接入。
AppKey	是	AppKey 是客户端的身份标识，在客户端 SDK 初始化时使用。
MasterSecret	是	MasterSecret 是开发者在使用服务端 API 接口时，用于校验身份的标识。

说明：在 OPPO 开放平台 上，开通 OPPO PUSH 权限后，即可在 OPPO 推送平台 > **配置管理** > **应用配置** 页面上查看应用的 AppKey 和 MasterSecret 信息。

4. 点击 **确定** 按钮，保存配置。

配置 vivo 推送渠道

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

1. 在左侧导航栏中，选择 **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，进入渠道配置页面。

点击 **VIVO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 vivo 推送渠道；关闭开关，即取消接入。
APP ID	是	AppId 是客户端的身份标识，在客户端 SDK 初始化时使用。
AppKey	是	AppKey 是客户端的身份标识，在客户端 SDK 初始化时使用。
Master Secret	是	MasterSecret 是开发者在使用服务端 API 接口时，用于校验身份的标识。该参数对应您从 vivo 开发者平台申请获取的 AppSecret。

说明：在 vivo 开放平台 上为应用申请 Push 服务通过后，即可获取应用的 AppId，AppKey 和 MasterSecret（AppSecret）。

4. 点击 **确定** 按钮，保存配置。

6.5 推送配置

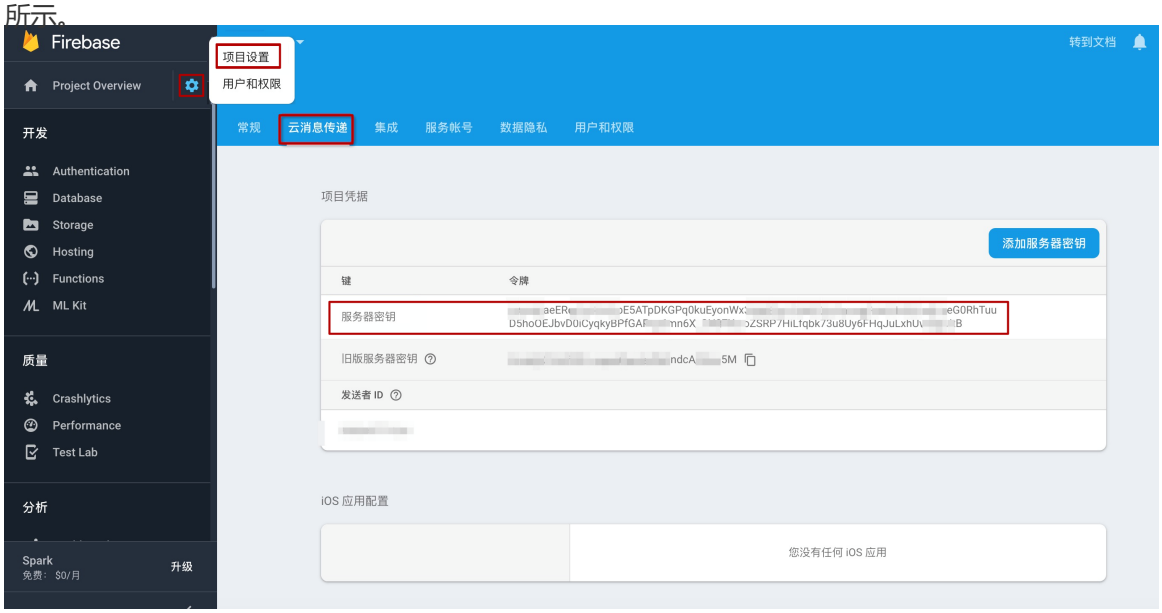
推送配置包括 **FCM 推送渠道配置** 和 **iOS 推送证书配置**。

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 推送渠道配置前，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。



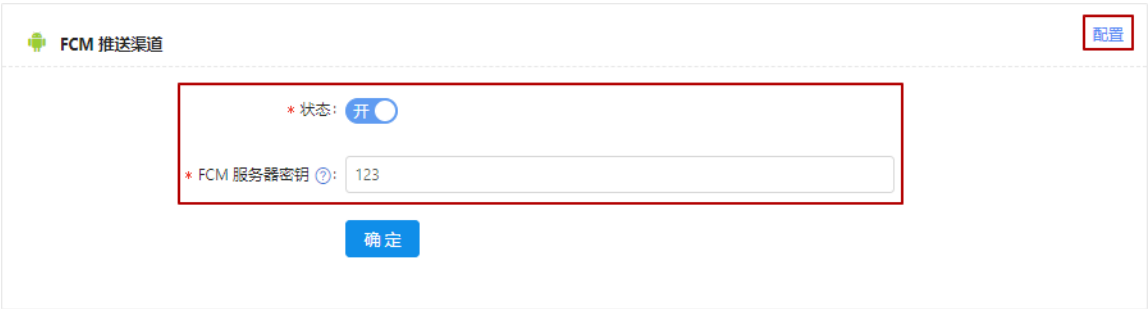
- 进行 iOS 推送证书配置时，您需要先 制作 iOS 推送证书 。

配置 FCM 推送渠道

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 **消息推送**，进入 **消息推送** 页面。
2. 在右侧页面上，单击 **推送配置** 标签，进入推送配置页面。

   单击 **FCM 推送渠道** 配置区域右上角的 **配置**，配置渠道信息，如下图所示。



   - 单击 **状态** 开关，打开开关后，MPS 将接入 FCM 服务；关闭开关后，MPS 不接入 FCM 服务。
   - 填写 **FCM 服务器密钥**，确保填写的是服务器（server）的密钥，Android 密钥、iOS 密钥和浏览器密钥会被 FCM 拒绝。

4. 单击 **确定**，保存配置。

配置 iOS 推送证书

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 **消息推送**，进入 **消息推送** 页面。
2. 在右侧页面上，单击 **推送配置** 标签，进入推送配置页面。
3. 在 **iOS 推送证书** 配置区域，配置 iOS 推送证书。
   - **选择证书文件**：选择并上传预先准备好的 iOS 推送证书。后端会通过解析上传的证书获得证书环境和 BundleId。如需了解 iOS 推送证书的制作步骤，参考 制作 iOS 推送证书 。
   - **证书密码**：填写证书密码。
4. 单击 **上传**，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。

iOS 推送证书

属性	值
alias	
bundleName	com.alipay.mpaasdemo
certHost	api.push.apple.com
certPort	443
issuerDN	CN=Apple Worldwide Developer Relations Certification Authority, OU=Apple Worldwide Developer Relations, O=Apple Inc., C=US
notAfter	1509334763000
notBefore	1475206763000
subjectDN	C=US, O="Alipay.Com Co., Ltd.", OU=LQ38NAVXP6, CN=Apple Push Services: com.alipay.mpaasdemo, UID=com.alipay.mpaasdemo

6.6 密钥管理

为了提高 MPS 与用户系统之间的交互的安全性，MPS 会对所有的服务端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。

配置推送 API 接口

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。

- RSA 公钥生成方法如下：
  - 通过 OpenSSL 官网 下载并安装 OpenSSL 工具（1.1.1 或以上版本）。
  - 打开 OpenSSL 工具，使用以下命令行生成 2048 位的 RSA 私钥。

```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048
```

  - 根据 RSA 私钥生成 RSA 公钥。

```
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

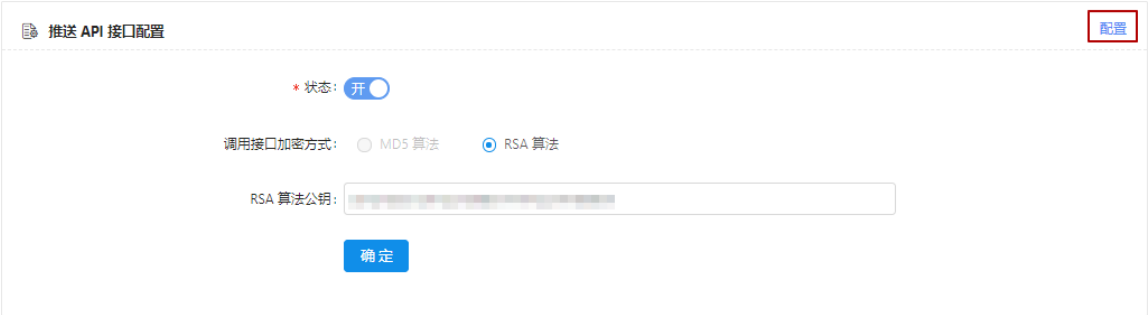
- 算法签名规则如下：
  - 使用 SHA 256 签名算法。
  - 将签名结果转换成 base64 字符串。
  - 将 base64 字符串中的 + 替换为 -，/ 替换为 -，得到最终签名结果。

操作步骤

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，进入密钥管理页面。

   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。



字段	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 。

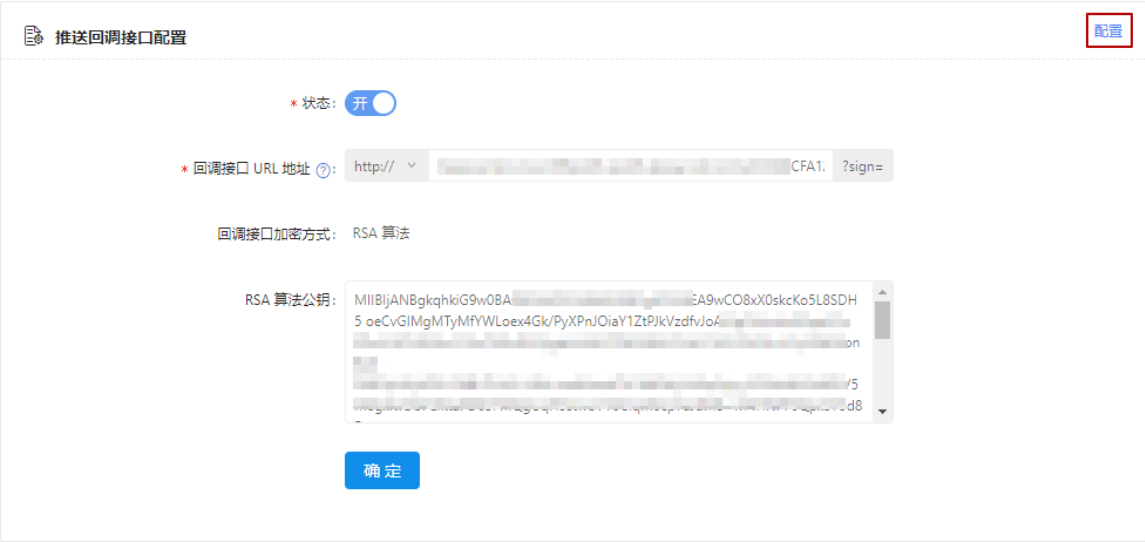
4. 点击 **确定** 按钮，保存配置。

配置推送回调接口

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，进入密钥管理页面。

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。



字段	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，应为公网可访问的 HTTP 请求地址。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/OPPO/vivo）：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。

代码示例

```
/**
* Alipay.com Inc. Copyright (c) 2004-2020 All Rights Reserved.
*/
package com.callback.demo.callbackdemo;

import com.callback.demo.callbackdemo.util.SignUtil;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

/**
*
* @author yqj
* @version $Id: PushCallbackController.java, v 0.1 2020年03月22日 11:20 AM yqj Exp $
*/
@Controller
public class PushCallbackController {

/**
* 拷贝控制台中推送回调接口配置的RSA 算法公钥
*/
private static final String pubKey ="";


@RequestMapping(value ="/push/callback",method = RequestMethod.POST)
public void callback(@RequestBody String callbackJson, @RequestParam String sign) {
System.out.println(sign);
//验签
sign = sign.replace('-','+');
sign = sign.replace('_','/');
if(!SignUtil.check(callbackJson,sign,pubKey,"UTF-8")){
System.out.println("验签失败");
return;
}
System.out.println("验签成功");
//json 消息体
System.out.println(callbackJson);

}

}
```

callbackJson 为消息请求体，为 JASON 格式，参考如下：

```
{
"extInfo":{
"adToken":"da64bc9d7d448684ebaeecfec473f612c57579008343a88d4dbdd145dad20e84",
"osType":"ios"
},
"msgId":"console_1584853300103",
```

```
"pushSuccess":true,
"statusCode":"2",
"statusDesc":"Acked",
"targetId":"da64bc9d7d448684ebaeecfec473f612c57579008343a88d4dbdd145dad20e84"
}
```

下表为 callbackJson 中各字段的说明。您可以通过此链接获取回调的 回调示例代码。

字段	说明
msgId	请求的业务消息 ID
pushSuccess	推送是否成功
statusCode	消息状态码
statusDesc	消息状态码对应的描述
targetId	目标 ID

7 常见问题

- 通用问题 关于权限的说明日志无法打印在 10.1.60.5 ~ 10.1.60.7 版本基线中存在的端口解析问题
- Android 相关问题 接入华为、小米等第三方渠道后，无法发送推送关于 push ad-token (deviceId) 的生成实现 PUSH 通知栏消息，对 EMUI 和华为移动服务是否有版本限制华为手机无法打印日志华为手机推送错误码OPPO 推送支持哪些机型和系统版本OPPO 手机推送错误码vivo 推送支持哪些机型和系统版本vivo 手机推送错误码Android 常见问题排查步骤
- iOS 相关问题 当 App 处于前台时， 消息推送是否会有横幅或声音提示消息状态是 NoBindInfo消息状态是 BadDeviceToken消息状态是 DeviceTokenNotForTopiciOS 手机无法收到消息，但消息状态是 ACKED

通用问题

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。

日志无法打印

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。

在 10.1.60.5 ~ 10.1.60.7 版本基线中存在的端口解析问题

如果是专有云环境，对于非 443 端口的推送服务器配置会出现解析失败导致连接错误。

解决方法：

如果使用 config 文件打包，请在 config 文件中按如下方式修改：

```
//config 文件中其他部分省略，在自定义端口号前加上\\{空格}
{
"pushPort":"\\ 8000",
}
```

如果不使用 config 文件打包，请在 AndroidManifest.xml 中将 rome.push.port 的值按如下方式修改：

```
//在端口号前加上\{空格}
<meta-data
android:name="rome.push.port"
android:value="\ 8000"/>
```

Android 相关问题

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。请参见 代码示例 以获取代码示例以及使用方法和注意事项。

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。

实现 PUSH 通知栏消息，对 EMUI 和华为移动服务是否有版本限制

对 Emotion UI（简称 EMUI，是华为基于 Android进行开发的情感化操作系统）和华为移动服务有版本限制，详细版本要求请参见 设备接收华为推送消息的条件。

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

如需了解错误码详情，请至华为官网查看 客户端错误码详解 及 服务端错误码详解。

OPPO 推送支持哪些机型和系统版本

目前支持 **ColorOS 3.1** 及以上系统的 **OPPO** 机型，**一加 5/5T** 及以上机型以及 **realme 所有** 机型。

ColorOS 是由 OPPO 推出的基于 Android 系统深度定制并优化的手机操作系统。

OPPO 手机推送错误码

当 OPPO 推送不生效时，您可在客户端日志中搜索 OPPO onRegister error =，获取错误码，并对照 OPPO 错误码 查询相应的错误原因。

vivo 推送支持哪些机型和系统版本

目前，SDK 支持的机型和最低系统版本如下表所示。有关 vivo 推送的其它相关问题，请参见 vivo 推送常见问题汇总。

机型名	Android版本	系统测试推送版本	第一个推送的版本号
Android9.0以及以上的版本默认支持			
Y93	Android 8.1	PD1818_A_1.9.6	PD1818_A_1.9.6
Y91	Android 8.1	PD1818E_A_1.7.5	PD1818E_A_1.7.5
Y93 标准版	Android 8.1	PD1818B_A_1.5.25	PD1818B_A_1.5.25
Y93s	Android 8.1	PD1818C_A_1.9.10	PD1818C_A_1.9.10
vivo Z1青春版	Android 8.1	PD1730E_A_1.13.27	PD1730E_A_1.13.27
Y97	Android 8.1	PD1813_A_1.10.6	PD1813_A_1.10.6
Z3	Android 8.1	PD1813B_A_1.5.19	PD1813B_A_1.5.19
Y81	Android 8.1	PD1732D_A_1.14.5	PD1732D_A_1.14.5
X23 幻彩版	Android 8.1	PD1816_A_1.10.2	PD1816_A_1.10.2
X21s	Android 8.1	PD1814_A_1.5.4	PD1814_A_1.5.4
X23	Android 8.1	PD1809_A_1.14.0	PD1809_A_1.14.1
NEX S	Android 8.1	PD1805_A_1.18.3	PD1805_A_1.18.4
NEX A	Android 8.1	PD1806B_A_2.17.1	PD1806B_A_2.17.1
NEX A	Android 8.1	PD1806_A_2.16.0	PD1806_A_2.17.1
X21i	Android 8.1	PD1801_A_1.15.0	PD1801_A_1.15.1
X21	Android 8.1	PD1728_A_1.21.0	PD1728_A_1.21.7
X20	Android 8.1	PD1709_A_8.8.1	PD1709_A_8.8.2
Y81s	Android 8.1	PD1732_A_1.12.2	PD1732_A_1.12.9
Y83A	Android 8.1	PD1803_A_1.20.5	PD1803_A_1.20.10
x9sp_8.1	Android 8.1	PD1635_A_8.15.0_Beta	PD1635_A_8.15.0_Beta
x9s_8.1	Android 8.1	PD1616B_A_8.15.0_Beta	PD1616B_A_8.15.0_Beta
Z1	Android 8.1	PD1730C_A_1.9.6	PD1730C_A_1.9.8
Y71	Android 8.1	PD1731_A_1.9.5	PD1731_A_1.9.5
Y73	Android 8.1	PD1731C_A_1.8.0	PD1731C_A_1.8.0
X20 Plus	Android 8.1	PD1710_A_8.3.0	PD1710_A_8.4.0
Y85	Android 8.1	PD1730_A_1.13.10	PD1730_A_1.13.11
x9_8.1	Android 8.1	PD1616_D_8.6.15	PD1616_D_8.6.16
x9Plus_8.1	Android 8.1	PD1619_A_8.12.1	PD1619_A_8.12.1
Y75A	Android 7.1	PD1718_A_1.12.6	PD1718_A_1.12.6
Y79A	Android 7.1	PD1708_A_1.23.10	PD1708_A_1.23.10
Y66i A	Android 7.1	PD1621BA_A_1.8.5	PD1621BA_A_1.8.5
X9	Android 7.1	PD1616_D_7.15.5	PD1616_D_7.15.5
x9s	Android 7.1	PD1616BA_A_1.13.5	PD1616BA_A_1.13.5
x9P	Android 7.1	PD1619_A_7.14.10	PD1619_A_7.14.10
x9sp	Android 7.1	PD1635_A_1.21.5	PD1635_A_1.21.6
xplay6	Android 7.1	PD1610_D_7.11.1	PD1610_D_7.11.1
Y69A	Android 7.0	PD1705_A_1.11.15	PD1705_A_1.11.15
Y53	Android6.0	PD1628_A_1.16.20	PD1628_A_1.16.20
Y67A	Android6.0	PD1612_A_1.11.27	PD1612_A_1.11.27
Y55	Android6.0	PD1613_A_1.19.11	PD1613_A_1.19.11
Y66	Android6.0	PD1621_A_1.12.36	PD1621_A_1.12.36

vivo 手机推送错误码

当 vivo 推送不生效时，您可在客户端日志中搜索 fail to turn on vivo push state =，获取状态码，并对照 公共状态码 定位具体原因。

Android 常见问题排查步骤

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

iOS 相关问题

当 App 处于前台时，消息推送是否会有横幅或声音提示

苹果的默认机制是，当 App 在前台时，消息可以达到，但是不会展示。如果需要在前台实现展示，需要自己做处理。

消息状态是 NoBindInfo

NoBindInfo 表示用户通过 UserId 去推送，但根据 UserId 没有找到对应的信息。请先确认客户端是否有调用绑定接口，并且对应的 appId 和 workspaceId 是否一致。

消息状态是 BadDeviceToken

此状态只会出现在 iOS 的推送，表示实际推送的 token 非法。先检查证书的环境是否正确。

- 如果 App 打包使用开发证书，那么 push 控制台配置需使用开发环境证书；Xcode 连真机调试，需要使用开发者证书。
- 如果 App 打包使用生产证书，那么 push 控制台配置需要使用生产环境证书。

消息状态是 DeviceTokenNotForTopic

此状态只会出现在 iOS 的推送，表示此 token 与推送的证书的 BundleId 不匹配。先检查证书是否正确，并且与客户端打包的 BundleId 是否一致。

iOS 手机无法收到消息，但消息状态是 ACKED

对于 iOS 的推送，如果消息状态是 ACKED，表示已经成功推送给苹果的推送服务。请先确认是否开启推送权限，是否有将应用切到后台。

苹果的默认机制是，当 App 在前台时，消息可以达到，但是不会展示。如果需要在前台实现展示，需要自己做处理。

8 参考

8.1 API 说明 - 客户端

消息推送包含以下客户端 API 接口，具体描述见下表：

调用方式	API	描述
RPC 调用	绑定 - RPC	绑定用户标识和设备标识（Ad-token）。

	解绑 - RPC	解绑用户标识和设备标识（Ad-token）。
	三方渠道设备上报 - RPC	绑定三方渠道设备标识（Ad-token）。

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 - RPC

方法定义

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。

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

参数说明

参数	类型	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

参数	说明
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。

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 - RPC

方法定义

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。

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

参数说明

参数	类型	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同 绑定接口返回值 。

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 - RPC

方法定义

该方法用于同步绑定三方渠道设备标识与本机设备标识，即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。

说明：框架内部会调用一次该方法，为避免 SDK 调用失败，建议您再手动调用一次。

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

参数说明

参数	类型	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同 绑定接口返回值 。

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);
```

8.2 API 说明 - 服务端

消息推送提供以下 OpenAPI 接口以实现用户服务端调用消息推送的极简推送、模板推送、批量推送、群发推送功能，具体描述见下表：

API	描述
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。
群发推送	对全网设备推送相同消息，消息通过模板进行创建。

SDK 准备

消息推送支持 Java、Python、Node.js、PHP 四种语言版本。针对不同的语言版本，在调用上述推送方式前，需要进行相应的推送准备。

下面分别对各个语言版本的 SDK 准备工作进行说明。

Java

在调用上述四种推送方式前，需引入 Maven 配置，在主控 pom 中引入如下依赖：

```
<dependency>
<groupId>com.aliyun</groupId>
<artifactId>aliyun-java-sdk-mpaas</artifactId>
```

```
<version>1.0.0.0</version>
</dependency>

<dependency>
<groupId>com.aliyun</groupId>
<artifactId>aliyun-java-sdk-core</artifactId>
<optional>true</optional>
<version>[4.3.2,5.0.0)</version>
</dependency>
```

Python

执行以下命令添加 SDK 相关依赖。

```
## 阿里云SDK
pip install aliyun-python-sdk-core
## mpaas SDK
pip install aliyun-python-sdk-mpaas
```

Node.js

执行以下命令添加 SDK 相关依赖。

```
npm i @alicloud/mpaas20190821
```

PHP

执行以下命令添加 SDK 相关依赖。

```
composer require alibabacloud/sdk
```

极简推送

对一个推送 ID 推送一条消息。

说明：在调用本接口之前，您需要引入依赖，详见 SDK 准备 。

参数说明

参数名称	类型	是否必填	示例	描述
taskName	String	是	simpleTest	推送任务名称。
title	String	是	测试	消息的标题。
content	String	是	测试	消息的正文。
appId	String	是	ONEX570DA89211721	mPaaS App ID
workspaceId	String	是	test	mPaaS 工作空间

deliveryType	Long	是	3	目标 ID 类型，数值选择如下： • 1 为 Android 设备维度 • 2 为 iOS 设备维度 • 3 为用户维度
targetMsgkey	String	是	{“user1024”:”1578807462788”}	推送目标，为 map 形式： • key：为目标，配合 deliveryType。 如果 deliveryType 为 1 ，则 key 为 Android 设备 ID。 如果 deliveryType 为 2，则 key 为 iOS 设备 ID。 如果 deliveryType 为 3 ，则 key 为 userid。 • value：消息业务 ID，用户自定义，必须保持唯一。 需注意，推送目标不可以超过 10 个。
expiredSeconds	Long	是	300	消息有效期，单位为秒。
extendedParams	String	否	{“key1”:”value1”}	扩展参数，为 map 形式。
pushAction	Long	否	0	点击消息后的跳转方式： • 0 为 Web URL • 1 为 Intent Activity 默认为 Web URL。
uri	String	否	http://www	点击消息后的跳转地址。
silent	Long	否	1	是否静默： • 1 为静默 • 0 为非静默

Java 代码示例

点击这里 查看下方代码示例中 AccessKeyId 与 AccessKeySecret 的获取方式 。

```
DefaultProfile.addEndpoint("cn-hangzhou","mpaas","mpaas.cn-hangzhou.aliyuncs.com");
// 创建DefaultAcsClient实例并初始化
DefaultProfile profile = DefaultProfile.getProfile(
"cn-hangzhou", // 地域ID
"******", // RAM账号的AccessKey ID
"*****"); // RAM账号AccessKey Secret

IAcsClient client = new DefaultAcsClient(profile);
```

```
// Create an API request and set parameters
PushSimpleRequest request = new PushSimpleRequest();
request.setAppId("ONEX570DA89211721");
request.setWorkspaceId("test");
request.setTaskName("测试任务");
request.setTitle("测试");
request.setContent("测试");
request.setDeliveryType(3L);
Map<String,String> extendedParam = new HashMap<String, String>();
extendedParam.put("key1","value1");
request.setExtendedParams(JSON.toJSONString(extendedParam));
request.setExpiredSeconds(300L);

Map<String,String> target = new HashMap<String, String>();
String msgKey = String.valueOf(System.currentTimeMillis());
target.put("user1024",msgKey);
request.setTargetMsgkey(JSON.toJSONString(target));
// Initiate the request and handle the response or exceptions
PushSimpleResponse response;
try {
response = client.getAcsResponse(request);
System.out.println(response.getResultCode());
System.out.println(response.getResultMessage());
} catch (ClientException e) {
e.printStackTrace();
}
```

Python 代码示例

```
from aliyunsdkcore.client import AcsClient
from aliyunsdkmpaas.request.v20190821 import PushSimpleRequest
import json

# Initialize AcsClient instance
client = AcsClient(
"***",
"***",
"cn-hangzhou"
);

# Initialize a request and set parameters
request = PushSimpleRequest.PushSimpleRequest()
request.set_endpoint("mpaas.cn-hangzhou.aliyuncs.com")
request.set_AppId("ONEX570DA89211721")
request.set_WorkspaceId("test")
request.set_Title("python测试")
request.set_Content("测试2")
request.set_DeliveryType(3)
request.set_TaskName("python测试任务")
request.set_ExpiredSeconds(600)
target = {"user1024":str(time.time())}
request.set_TargetMsgkey(json.dumps(target))

# Print response
response = client.do_action_with_exception(request)
```

```
print response
```

Node.js 代码示例

```
const sdk = require('@alicloud/mpaas20190821');

const { default: Client, PushSimpleRequest } = sdk;
// 创建客户端
const client = new Client({
accessKeyId: '****',
accessKeySecret: '*****',
endpoint: 'mpaas.cn-hangzhou.aliyuncs.com',
apiVersion: '2019-08-21'
});
// 初始化 request
const request = new PushSimpleRequest();
request.appId ="ONEX570DA89211721";
request.workspaceId ="test";
request.title ="Node测试";
request.content ="测试";
request.deliveryType = 3;
request.taskName ="Node测试任务";
request.expiredSeconds=600;
const extendedParam = {
test: '自定义扩展参数'
};
request.extendedParams = JSON.stringify(extendedParam);
// value 为业务方消息id，请保持唯一
const target = {
"userid1024": String(new Date().valueOf())
};
request.targetMsgkey = JSON.stringify(target);

// 调用 API
try {
client.pushSimple(request).then(res => {
console.log('SUCCESS', res);
}).catch(e => {
console.log('FAIL', e);
});
} catch(e) {
console.log('ERROR', e);
}
```

PHP 代码示例

```
<?php

use AlibabaCloud\Client\AlibabaCloud;
use AlibabaCloud\MPaaS\MPaaS;
AlibabaCloud::accessKeyClient('accessKeyId', 'accessKeySecret')
->regionId('cn-hangzhou')
->asDefaultClient();
```

```
class Demo {
public function run() {
try {
$this->simplePush();
} catch (\Exception $e) {
}
}


public function simplePush() {
$request = MPaaS::v20190821()->pushSimple();
$result = $request->withAppId("ONEX570DA89211721")
->withWorkspaceId("test")
->withTitle("PHP 测试")
->withContent("测试3")
->withDeliveryType(3)
->withTaskName("PHP 测试任务")
->withExpiredSeconds(600)
->withTargetMsgkey(
json_encode(["userid1024"=>"".time() ]
))
// endpoint
->host("mpaas.cn-hangzhou.aliyuncs.com")
// 是否开启 debug 模式
->debug(true)
->request();
}
}
```

模板推送

模板推送指针对单个目标 ID 的消息推送，消息通过模板创建。多个 ID 可以使用同一个模板。

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 。
- 在调用本接口之前，您需要引入依赖，详见 SDK 准备 。

参数说明

参数名称	类型	是否必填	示例	描述
taskName	String	是	模板测试	推送任务名称。
appId	String	是	ONEX570DA89211721	mPaaS App ID
workspaceId	String	是	test	mPaaS 工作空间
deliveryType	Long	是	3	目标 ID 类型，数值选择如下： • 1为 Android 设备维度 • 2 为 iOS 设备维度 • 3 为用户维度

targetMsgkey	String	是	{“user1024”:”1578807462788”}	推送目标，为 map 形式： • key：为目标，配合 deliveryType。 如果 deliveryType 为 1 ，则 key 为 Android 设备 ID。 如果 deliveryType 为 2，则 key 为 iOS 设备 ID。 如果 deliveryType 为 3 ，则 key 为 userid。 • value：消息业务 ID，用户自定义，必须保持唯一。 需注意，推送目标不可以超过 10 个。
expiredSeconds	Long	是	300	消息有效期，单位为秒。
templateName	String	是	测试模板	模板名称，在控制台创建模板。
templateKeyValue	String	否	{“money”:”200”,”name”:”张三”}	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。
extendedParams	String	否	{“key1”:”value1”}	扩展参数，为 map 形式。

Java 代码示例

点击这里 查看下方代码示例中 AccessKeyId 与 AccessKeySecret 的获取方式 。

```
DefaultProfile.addEndpoint("cn-hangzhou","mpaas","mpaas.cn-hangzhou.aliyuncs.com");
// 创建DefaultAcsClient实例并初始化
DefaultProfile profile = DefaultProfile.getProfile(
"cn-hangzhou", // 地域ID
"******", // RAM账号的AccessKey ID
"******"); // RAM账号AccessKey Secret

IAcsClient client = new DefaultAcsClient(profile);
// Create an API request and set parameters
PushTemplateRequest request = new PushTemplateRequest();
request.setAppId("ONEX570DA89211721");
request.setWorkspaceId("test");
request.setTemplateName("测试模板");
//你好#name#,恭喜中奖#money#元
Map<String,String> templatekv = new HashMap<String, String>();
templatekv.put("name","张三");
templatekv.put("money","200");
request.setTemplateKeyValue(JSON.toJSONString(templatekv));
request.setExpiredSeconds(600L);
request.setTaskName("模板测试");
request.setDeliveryType(3L);
Map<String,String> target = new HashMap<String, String>();
String msgKey = String.valueOf(System.currentTimeMillis());
```

```
target.put("userid1024",msgKey);
request.setTargetMsgkey(JSON.toJSONString(target));

PushTemplateResponse response;
try {
response = client.getAcsResponse(request);

System.out.println(response.getResultCode());
System.out.println(response.getResultMessage());
} catch (ClientException e) {
e.printStackTrace();
}
```

Python 代码示例

```
from aliyunsdkcore.client import AcsClient
from aliyunsdkmpaas.request.v20190821 import PushTemplateRequest
import json
import time

# Initialize AcsClient instance
client = AcsClient(
"AccessKey ID",
"AccessKey Secret",
"cn-hangzhou"
);

# Initialize a request and set parameters
request = PushTemplateRequest.PushTemplateRequest()
request.set_endpoint("mpaas.cn-hangzhou.aliyuncs.com")
request.set_AppId("ONEX570DA89211721")
request.set_WorkspaceId("test")
request.set_TemplateName("template1024")
templatekv = {"name":"张三","money":"200"}
request.set_TemplateKeyValue(json.dumps(templatekv))
request.set_DeliveryType(3)
request.set_TaskName("python模板测试任务")
request.set_ExpiredSeconds(600)
target = {"userid1024":str(time.time())}
request.set_TargetMsgkey(json.dumps(target))

# Print response
response = client.do_action_with_exception(request)
print response
```

Node.js 代码示例

```
const sdk = require('@alicloud/mpaas20190821');

const { default: Client, PushTemplateRequest } = sdk;
// 创建客户端
const client = new Client({
accessKeyId: 'accessKeyId',
```

```
accessKeySecret: 'accessKeySecret',
endpoint: 'mpaas.cn-hangzhou.aliyuncs.com',
apiVersion: '2019-08-21'
});
// 初始化 request
const request = new PushTemplateRequest();
request.appId ="ONEX570DA89211721";
request.workspaceId ="test";
request.templateName="template1024";
const templatekv = {
name: '张三',
money:'300'
};
request.templateKeyValue = JSON.stringify(templatekv);
request.deliveryType = 3;
request.taskName ="Node测试任务";
request.expiredSeconds=600;
const extendedParam = {
test: '自定义扩展参数'
};
request.extendedParams = JSON.stringify(extendedParam);
const target = {
"userid1024": String(new Date().valueOf())
};
request.targetMsgkey = JSON.stringify(target);

// 调用 API
try {
client.pushTemplate(request).then(res => {
console.log('SUCCESS', res);
}).catch(e => {
console.log('FAIL', e);
});
} catch(e) {
console.log('ERROR', e);
}
```

PHP 代码示例

```
<?php

use AlibabaCloud\Client\AlibabaCloud;
use AlibabaCloud\MPaaS\MPaaS;
AlibabaCloud::accessKeyClient('accessKeyId', 'accessKeySecret')
->regionId('cn-hangzhou')
->asDefaultClient();

class Demo {
public function run() {
try {
$this->templatePush();
} catch (\Exception $e) {
}
}
```

```
public function templatePush() {
$request = MPaaS::v20190821()->pushTemplate();
$result = $request->host("mpaas.cn-hangzhou.aliyuncs.com")
// 是否开启 debug 模式
->debug(true)
->withAppId("ONEX570DA89211721")
->withWorkspaceId("test")
->withTemplateName("template1024")
->withTemplateKeyValue(json_encode(["name"=>"张三","money"=>"200"]))
->withDeliveryType(3)
->withTaskName("PHP 测试任务")
->withExpiredSeconds(600)
->withTargetMsgkey(
json_encode(["userid1024"=>"".time() ])
)
->request();
}
}
```

批量推送

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。

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（即对不同推送 ID 推送不同消息）。详细操作参见 创建模板 。
- 在调用本接口之前，您需要引入依赖，详见 SDK 准备 。

参数说明

参数名称	类型	是否必填	示例	描述
taskName	String	是	批量测试	推送任务名称。
appId	String	是	ONEX570DA89211721	mPaaS App ID
workspaceId	String	是	test	mPaaS 工作空间
deliveryType	Long	是	3	目标 ID 类型，数值选择： • 1 为 Android 设备维度 • 2 为 iOS 设备维度 • 3 为用户维度
templateName	String	是	测试模板	模板名称，在控制台创建模板。
targetMsgs	List	是	TargetMsg 对象的list	目标对象列表，参数详见下方的 TargetMsg 对象。
expiredSeconds	Long	是	300	消息有效期，单位为秒。
extendedParam	Strin	否	{ “key1” :” value1” }	统一扩展参数，为 map 形式。

s	g			

TargetMsg 对象

参数名称	类型	是否必填	示例	描述
target	String	是	userid1024	目标 ID，根据 deliveryType 类型填写。
msgKey	String	是	1578807462788	业务消息 ID，用于消息的排查。由用户定义，不可重复。
templateKeyValue	String	否	{ “money” :” 200” ,” name” :” 张三” }	模板参数，为 map 形式，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。
extendedParams	String	否	{ “key1” :” value1” }	扩展参数，为 map 形式，针对每条消息的不同扩展参数。

Java 代码示例

点击这里 查看下方代码示例中 AccessKeyId 与 AccessKeySecret 的获取方式 。

```
DefaultProfile.addEndpoint("cn-hangzhou","mpaas","mpaas.cn-hangzhou.aliyuncs.com");
// 创建DefaultAcsClient实例并初始化
DefaultProfile profile = DefaultProfile.getProfile(
"cn-hangzhou", // 地域ID
"******", // RAM账号的AccessKey ID
"******"); // RAM账号AccessKey Secret

IAcsClient client = new DefaultAcsClient(profile);
// Create an API request and set parameters
PushMultipleRequest request = new PushMultipleRequest();
request.setAppId("ONEX570DA89211721");
request.setWorkspaceId("test");
request.setDeliveryType(3L);
request.setTaskName("批量测试");
request.setTemplateName("测试模板");
//你好#name#,恭喜中奖#money#元
List<PushMultipleRequest.TargetMsg> targetMsgs = new ArrayList<PushMultipleRequest.TargetMsg>();
PushMultipleRequest.TargetMsg targetMsg = new PushMultipleRequest.TargetMsg();
targetMsg.setTarget("userid1024");
targetMsg.setMsgKey(String.valueOf(System.currentTimeMillis()));
Map<String, String> templatekv = new HashMap<String, String>();
templatekv.put("name","张三");
templatekv.put("money","200");
targetMsg.setTemplateKeyValue(JSON.toJSONString(templatekv));
//目标数量不要超过400个
targetMsgs.add(targetMsg);
request.setTargetMsgs(targetMsgs);
request.setExpiredSeconds(600L);
PushMultipleResponse response;
try {
response = client.getAcsResponse(request);
System.out.println(response.getResultCode());
```

```
System.out.println(response.getResultMessage());
} catch (ClientException e) {
e.printStackTrace();
}
```

Python 代码示例

```
# -*- coding: utf8 -*-

from aliyunsdkcore.client import AcsClient
from aliyunsdkmpaas.request.v20190821 import PushMultipleRequest
import json
import time

# Initialize AcsClient instance
client = AcsClient(
"AccessKey ID",
"AccessKey Secret",
"cn-hangzhou"
);

# Initialize a request and set parameters
request = PushMultipleRequest.PushMultipleRequest()
request.set_endpoint("mpaas.cn-hangzhou.aliyuncs.com")
request.set_AppId("ONEX570DA89211721")
request.set_WorkspaceId("test")
request.set_TemplateName("template1024")
request.set_DeliveryType(3)
request.set_TaskName("python测试任务")
request.set_ExpiredSeconds(600)
msgkey = str(time.time())
targets = [
{
"Target":"user1024",
"MsgKey": msgkey,
"TemplateKeyValue": {
"name":"张三",
"money":"200"
}
}
]
request.set_TargetMsgs(targets)
# Print response
response = client.do_action_with_exception(request)
print response
```

Node.js 代码示例

```
const sdk = require('@alicloud/mpaas20190821');

const { default: Client, PushMultipleRequest,PushMultipleRequestTargetMsg } = sdk;
// 创建客户端
const client = new Client({
```

```
accessKeyId: 'accessKeyId',
accessKeySecret: 'AccessKey Secret',
endpoint: 'mpaas.cn-hangzhou.aliyuncs.com',
apiVersion: '2019-08-21'
});
// 初始化 request
const request = new PushMultipleRequest();
request.appId ="ONEX570DA89211721";
request.workspaceId ="test";
request.templateName="template1024";
const templatekv = {
name: '张三',
money:'300'
};
//request.templateKeyValue = JSON.stringify(templatekv);

request.deliveryType = 3;
request.taskName ="Node测试任务";
request.expiredSeconds=600;
const extendedParam = {
test: '自定义扩展参数'
};
request.extendedParams = JSON.stringify(extendedParam);

const targetMsgkey = new PushMultipleRequestTargetMsg();
targetMsgkey.target ="userid1024";
targetMsgkey.msgKey = String(new Date().valueOf());
targetMsgkey.templateKeyValue = JSON.stringify(templatekv);;
request.targetMsg = [targetMsgkey];

// 调用 API
try {
client.pushMultiple(request).then(res => {
console.log('SUCCESS', res);
}).catch(e => {
console.log('FAIL', e);
});
} catch(e) {
console.log('ERROR', e);
}
```

PHP 代码示例

```
<?php

use AlibabaCloud\Client\AlibabaCloud;
use AlibabaCloud\MPaaS\MPaaS;
AlibabaCloud::accessKeyClient('accessKeyId', 'accessKeySecret')
->regionId('cn-hangzhou')
->asDefaultClient();

class Demo {
public function run() {
try {
$this->multiPush();
```

```
} catch (\Exception $e) {
}
}


public function multiPush() {
$request = MPaaS::v20190821()->pushMultiple();
$result = $request->host("mpaas.cn-hangzhou.aliyuncs.com")
// 是否开启 debug 模式
->debug(true)
->withAppId("ONEX570DA89211721")
->withWorkspaceId("test")
->withTemplateName("template1024")
->withDeliveryType(3)
->withTaskName("PHP测试批量任务")
->withExpiredSeconds(600)
->withTargetMsg(
[
[
"Target"=>"userid1024",
"MsgKey"=>"". time(),
"TemplateKeyValue"=> json_encode([
"name"=>"张三",
"money"=>"200",
])
]
]
)
->request();
}
}
```

群发推送

对全网设备推送相同消息，消息通过模板创建。

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 。
- 在调用本接口之前，您需要引入依赖，详见 SDK 准备 。

参数说明

参数名称	类型	是否必填	示例	描述
taskName	String	是	群发测试任务	推送任务名称。
appId	String	是	ONEX570DA89211721	mPaaS App ID
workspaceId	String	是	test	mPaaS 工作空间
deliveryType	Long	是	1	目标 ID 类型，数值选择： • 1 为 Android 群发 • 2 为 iOS 群发

msgkey	String	是	1578807462788	业务方消息 ID，用户自定义，不可重复。
expiredSeconds	Long	是	300	消息有效期，单位为秒。
templateName	String	是	群发模板	模板名称，在控制台创建模板。
templateKeyValue	String	否	{“content”:”公告内容”}	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。
pushStatus	Long	否	0	针对 iOS 群发，群发时，推送登录状态： • 0 表示所有当前绑定的用户（默认） • 1 表示所有用户（包括解绑的用户）
unBindPeriod	Long	否	1	退出登录时长，当 pushStatus 为 1 时为必填： • 1 表示 7 天内解绑的用户 • 2 表示 15 天内解绑的用户 • 3 表示 60 天内解绑的用户 • 4 表示永久

Java 代码示例

点击这里 查看下方代码示例中 AccessKeyId 与 AccessKeySecret 的获取方式 。

```
DefaultProfile.addEndpoint("cn-hangzhou","mpaas","mpaas.cn-hangzhou.aliyuncs.com");
// 创建DefaultAcsClient实例并初始化
DefaultProfile profile = DefaultProfile.getProfile(
"cn-hangzhou", // 地域ID
"******", // RAM账号的AccessKey ID
"******"); // RAM账号AccessKey Secret

IAcsClient client = new DefaultAcsClient(profile);

PushBroadcastRequest request = new PushBroadcastRequest();
request.setAppId("ONEX570DA89211720");
request.setWorkspaceId("test");
request.setDeliveryType(2L);
request.setMsgkey(String.valueOf(System.currentTimeMillis()));
request.setExpiredSeconds(600L);
request.setTaskName("群发任务");
request.setTemplateName("群发测试");
//这是一个公告:#content#
Map<String, String> templatekv = new HashMap<String, String>();
templatekv.put("content","公告内容");
request.setTemplateKeyValue(JSON.toJSONString(templatekv));
PushBroadcastResponse response;
try {
response = client.getAcsResponse(request);
System.out.println(response.getResultCode());
System.out.println(response.getResultMessage());
```

```
} catch (ClientException e) {
e.printStackTrace();
}
```

Python 代码示例

```
# -*- coding: utf8 -*-

from aliyunsdkcore.client import AcsClient
from aliyunsdkmpaas.request.v20190821 import PushBroadcastRequest
import json
import time

# Initialize AcsClient instance
client = AcsClient(
"AccessKey ID",
"AccessKey Secret",
"cn-hangzhou"
);

# Initialize a request and set parameters
request = PushBroadcastRequest.PushBroadcastRequest()
request.set_endpoint("mpaas.cn-hangzhou.aliyuncs.com")
request.set_AppId("ONEX570DA89211720")
request.set_WorkspaceId("test")
request.set_TemplateName("broadcastTemplate")
templatekv = {"content":"这个一个公告"}
request.set_TemplateKeyValue(json.dumps(templatekv))
request.set_DeliveryType(1)
request.set_TaskName("python测试群发任务")
request.set_ExpiredSeconds(600)
request.set_Msgkey(str(time.time()))

# Print response
response = client.do_action_with_exception(request)
print response
```

Node.js 代码示例

```
const sdk = require('@alicloud/mpaas20190821');

const { default: Client, PushBroadcastRequest } = sdk;
// 创建客户端
const client = new Client({
accessKeyId: 'accessKeyId',
accessKeySecret: 'AccessKey Secret',
endpoint: 'mpaas.cn-hangzhou.aliyuncs.com',
apiVersion: '2019-08-21'
});
// 初始化 request

const request = new PushBroadcastRequest();
request.appId ="ONEX570DA89211720";
```

```
request.workspaceId ="test";
request.templateName="broadcastTemplate";
const templatekv = {
content: '这是公告哦',
};
request.templateKeyValue = JSON.stringify(templatekv);
request.deliveryType = 1;
request.taskName ="Node测试任务";
request.expiredSeconds=600;
const extendedParam = {
test: '自定义扩展参数'
};
request.extendedParams = JSON.stringify(extendedParam);

request.msgkey = String(new Date().valueOf())

// 调用 API
try {
client.pushBroadcast(request).then(res => {
console.log('SUCCESS', res);
}).catch(e => {
console.log('FAIL', e);
});
} catch(e) {
console.log('ERROR', e);
}
```

PHP 代码示例

```
<?php

use AlibabaCloud\Client\AlibabaCloud;
use AlibabaCloud\MPaaS\MPaaS;
AlibabaCloud::accessKeyClient('accessKeyId', 'accessKeySecret')
->regionId('cn-hangzhou')
->asDefaultClient();

class Demo {
public function run() {
try {
$this->broadcastPush();
} catch (\Exception $e) {
}
}


public function broadcastPush(){
$request = MPaaS::v20190821()->pushBroadcast();
$result = $request->host("mpaas.cn-hangzhou.aliyuncs.com")
// 是否开启 debug 模式
->debug(true)
->withAppId("ONEX570DA89211720")
->withWorkspaceId("test")
->withTemplateName("broadcastTemplate")
->withTemplateKeyValue(
```

```
json_encode(["content"=>"这是一个公告"])
)
->withDeliveryType(1)
->withTaskName("PHP 测试群发任务")
->withExpiredSeconds(600)
->withMsgkey(". time())
->request();
}
}
```

扩展参数

扩展参数会跟随消息体到达客户端，供用户自定义处理。

扩展参数包含以下三类：

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。系统具有一定意义的扩展参数及其说明参见下表。

key	说明
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 iOS 10 的 UNNotificationServiceExtension；若不携带本参数，则为普通推送。Value 配置为 1。
badge_add_num	华为渠道推送角标增加数。
badge_class	华为渠道桌面图标对应的应用入口 Activity 类。
big_text	大文本样式，value 固定为 1，填写其他值无效。本参数仅对小米和华为手机有效。

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。

API 调用结果码

结果码	结果消息	说明
100	SUCCESS	成功
-1	SIGNATURE_MISMATCH	签名不匹配
3001	NEED_DELIVERYTOKEN	deliveryToken 为空
3002	NEED_FILE	文件为空
3003	NEED_APPID_WORKSPACEID	appid 或 workspace 为空
3007	APPID_WRONG	appid 或 workspace 不合法
3008	OS_TYPE_NOT_SUPPORTED	推送平台类型不支持
3009	DELIVERY_TYPE_NOT_SUPPORTED	目标 ID 类型不支持
3012	NEED_USERID	UserId 为空
3019	TASKNAME_NULL	任务名称为空
3020	EXPIREDSECONDS_WRONG	消息超时时间非法
3021	TOKEN_OR_USERID_NULL	目标为空
3022	TEMPLATE_NOT_EXIST	模板不存在
3023	TEMPLATEKV_NOT_ENOUGH	模板参数不匹配
3024	PAYLOAD_NOT_ENOUGH	标题或内容为空
3025	NEED_TEMPLATE	模板为空
3026	EXPIREDTIME_TOO_LONG	消息有效期过长
3028	INVALID_PARAM	参数非法
3029	SINGLE_PUSH_TARGET_TOO_MUCH	推送目标过多
3030	BROADCAST_ONLY_SUPPORT_BY_DEVICE	仅支持设备维度的群发
303	REQUEST_SHOULD_BE_UTF8	请求体编码需为 UTF-8

1		
3032	REST_API_SWITCH_NOT_OPEN	推送 API 接口关闭
3033	UNKNOWN_REST_SIGN_TYPE	签名类型不支持
3035	EXTEND_PARAM_TO_MUCH	扩展字段太多，不能超过 20 个
3036	TEMPLATE_ALREADY_EXIST	模板已存在
3037	TEMPLATE_NAME_NULL	模板名称为空
3038	TEMPLATE_NAME_INVALID	模板名称非法
3039	TEMPLATE_CONTENT_INVALID	模板内容非法
3040	TEMPLATE_TITLE_INVALID	模板标题非法
3041	TEMPLATE_DESC_INFO_INVALID	模板描述非法
3042	TEMPLATE_URI_INVALID	模板 URI 非法
3043	SINGLE_PUSH_CONTENT_TOO_LONG	消息体过长
3044	INVALID_EXTEND_PARAM	扩展参数非法
3049	MULTIPLE_INNER_EXTEND_PARAM_TO_MUCH	批量推送内部扩展参数需小于 10 个
3050	MSG_PAYLOAD_TOO_LONG	消息体太长
3051	BROADCAST_ALL_USER_NEED_UNBIND_PERIOD	群发推送针对所有用户（登录用户 + 登出用户），必须传解绑参数
3052	BROADCAST_ALL_USER_UNBIND_PERIOD_INVALID	群发解绑参数非法
3053	BROADCAST_ALL_USER_NOT_SUPPORT_SELFCHANNEL_ANDROID	群发所有用户，不支持自建渠道群发
3054	DELIVERYTOKEN_INVALID	自建渠道 token 非法
3055	MULTIPLE_TARGET_NUMBER_TOO_MUCH	批量推送目标过多
3056	TEMPLATE_NUM_TOO_MUCH	模板数量过多
3057	ANDROID_CHANNEL_PARAM_INVALID	androidChannel 参数非法
3058	BADGE_ADD_NUM_INVALID	角标参数非法
3059	BADGE_ADD_NUM_NEED_BADGE_CLASS	badge_add_num 参数需要 badge_class 参数
900	SYSTEM_ERROR	系统错误

0		

8.3 制作 iOS 推送证书

如需向 iOS 设备推送数据，您首先需要在消息推送控制台上配置 iOS 推送证书。iOS 推送证书用于推送通知，本文将介绍消息推送服务支持的证书类型，并引导您制作 iOS 推送证书。

证书类型

苹果证书类型 如下图所示。消息推送服务仅支持 **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 **消息推送控制台** 区分这两类证书。

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

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。消息推送不支持。
- Apple Push Service：生产环境苹果推送证书。消息推送支持。
- Apple Development IOS Push Services：开发环境苹果推送证书。消息推送支持。

消息推送控制台

在消息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：

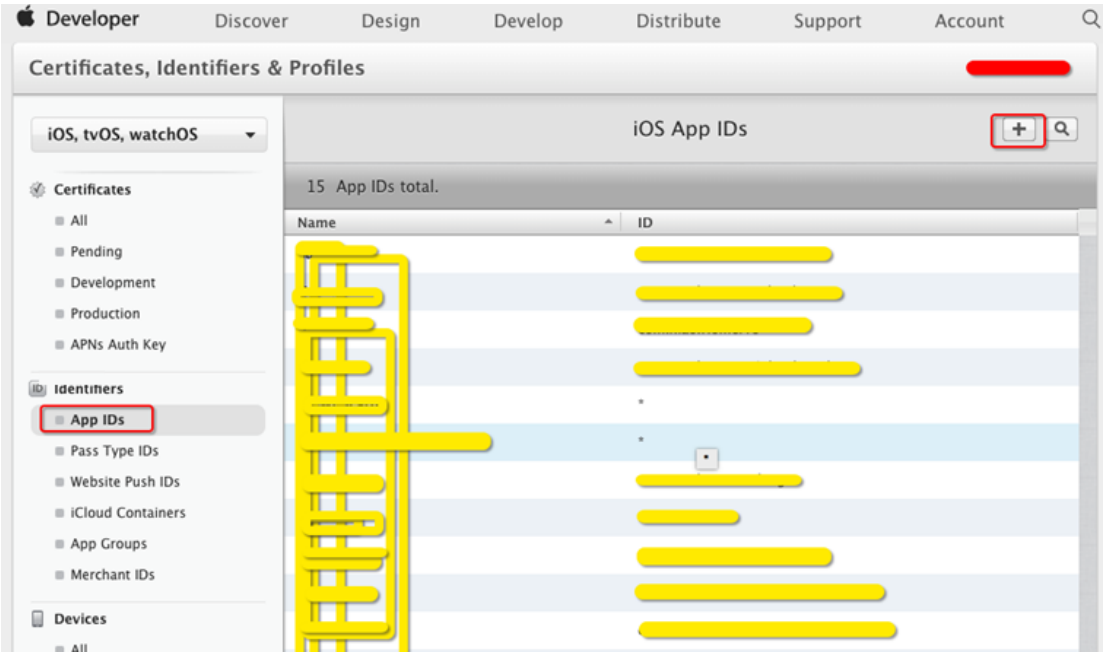
- Apple Development IOS Push Services：开发环境苹果推送证书。消息推送支持。
- Apple Push Service：生产环境苹果推送证书。消息推送支持。

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

制作证书

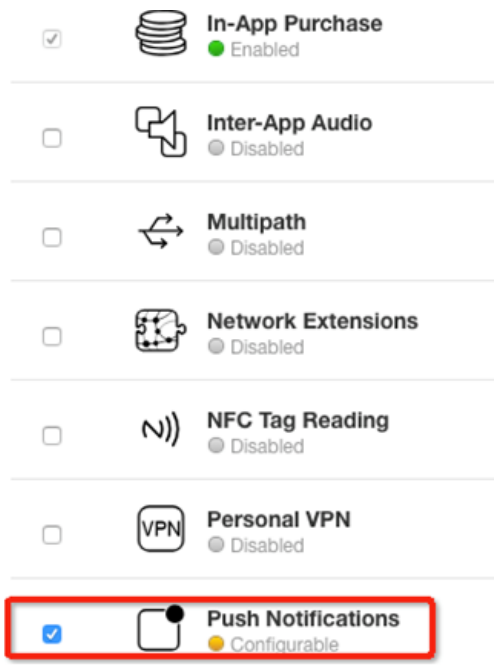
创建苹果 App ID

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



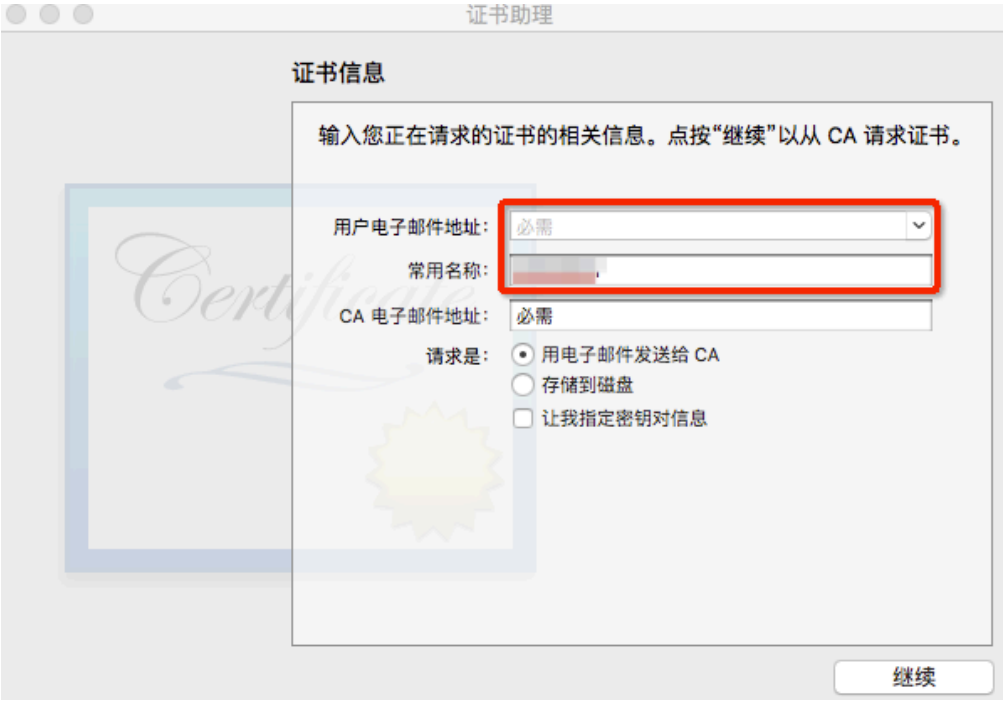
单击 **Continue** 后，单击 **Register** 完成创建。

制作 .certSigningRequest 文件

进入 Mac 中的钥匙串服务。

请求证书。选择 **钥匙串访问** > **证书助理** > **从证书颁发机构请求证书...**。

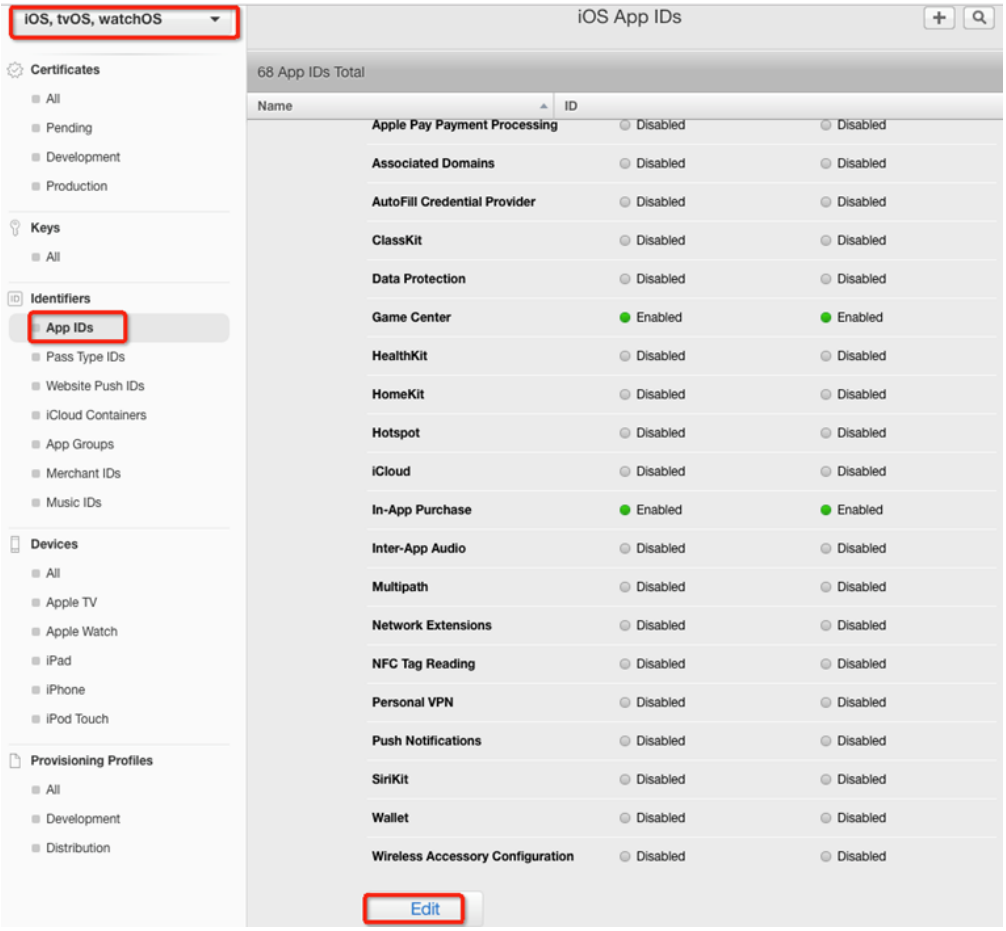
在打开的 **证书信息** 窗口中，根据实际情况填写邮件地址和常用名称等相关信息。



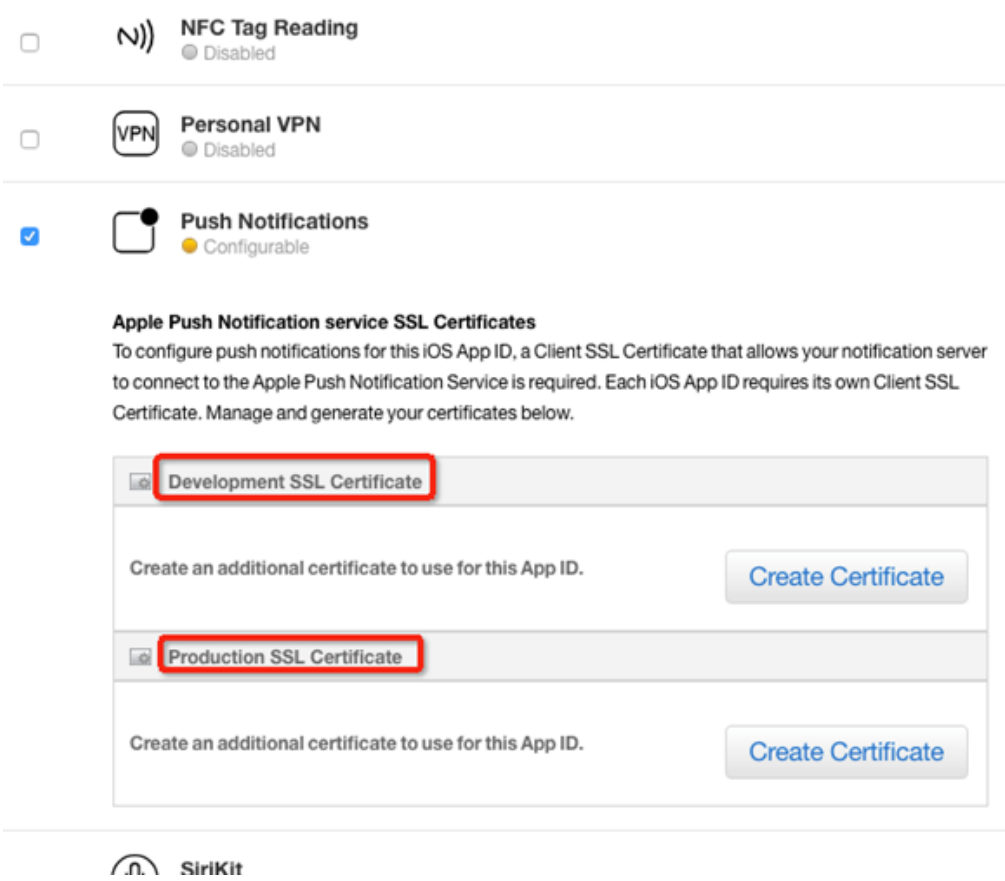
.certSigningRequest 文件制作成功，如下图所示。

创建证书

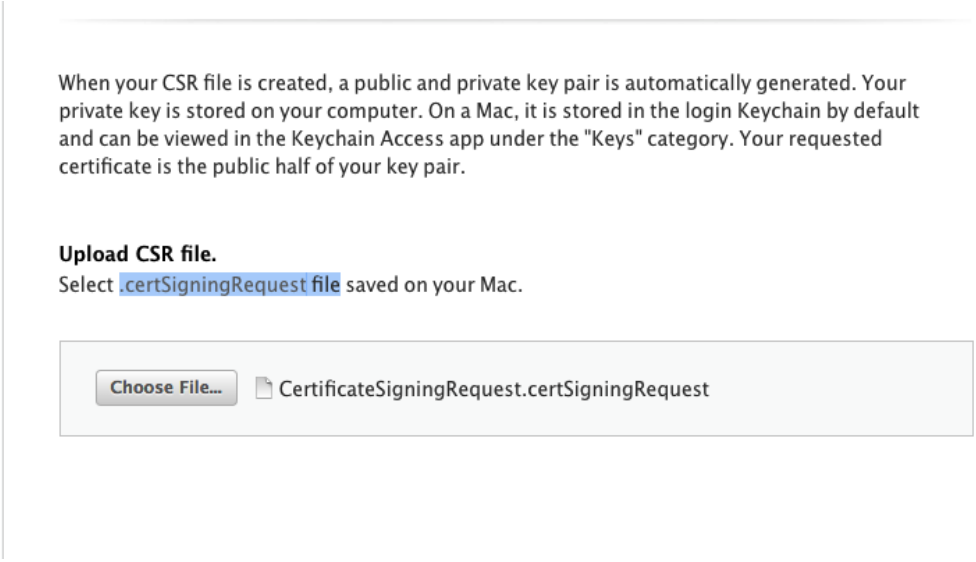
在苹果 **App IDs** 页面中，选中自己的 iOS App ID，单击 **Edit**。



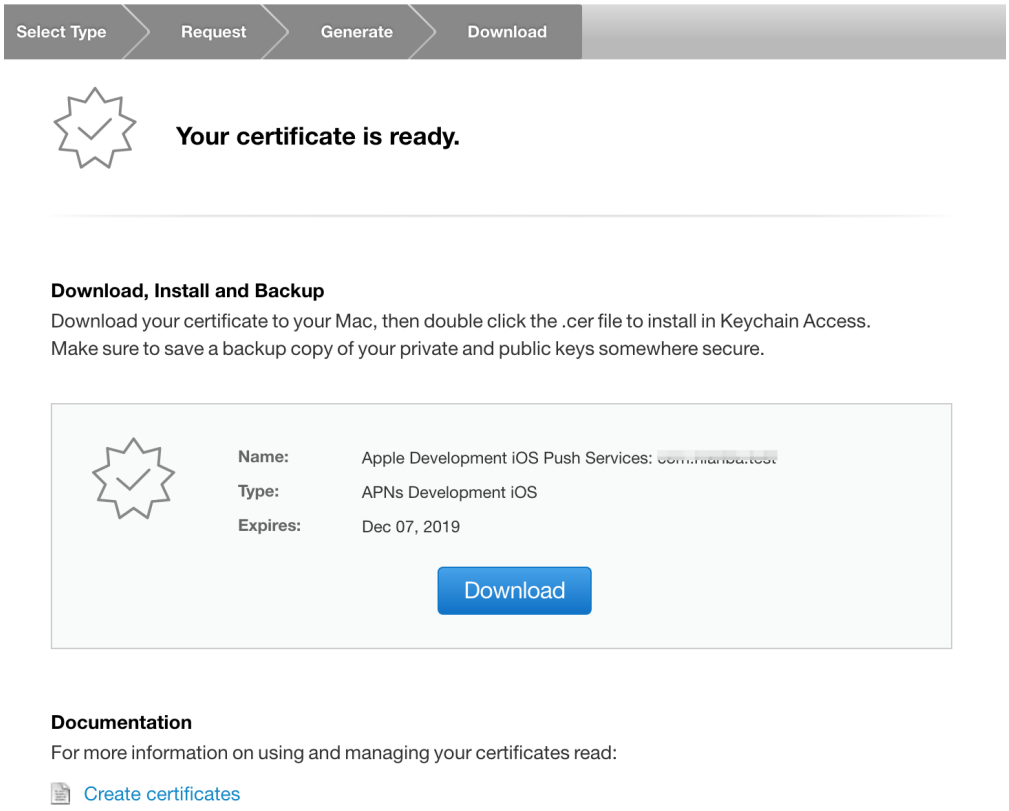
单击 Development SSL Certificate 或 Production SSL Certificate 卡片中的 Create Certificate，开始创建开发或生产环境下的证书。



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。单击 **Download**，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 **消息推送控制台** > **推送配置** 中配置 iOS 推送证书。

蚂蚁集团
ANT GROUP