蚂蚁科技

消息推送
使用指南

文档版本：20240808

蚂蚁集团
ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

商标声明

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。蚂蚁集团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，并在蚂蚁集团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集团授权渠道下载、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。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

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，推送通道支持自建通道和厂商通道，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、推送通道以及推送方式。

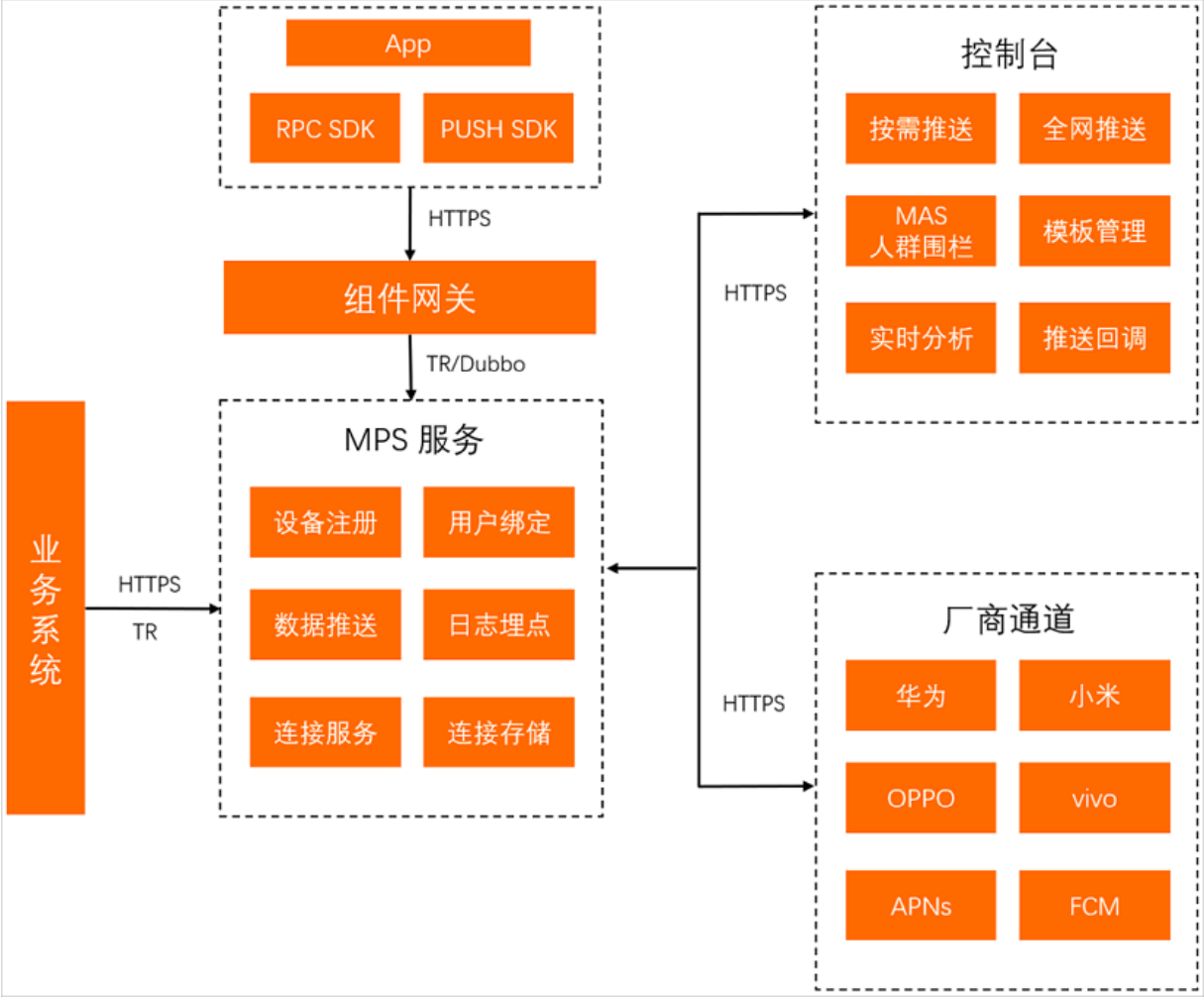
MPS 核心功能如下：

- **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。
- **使用分析**：基于客户端埋点上报数据，在平台、版本、推送通道、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。
- **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。
- **推送通道配置**：接入厂商推送通道，集成华为、小米等厂商通道推送功能，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。

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通道进行 **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、用户绑定以及厂商通道的关系绑定，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 等推送通道，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

Apache Dubbo（Dubbo）

Dubbo 是一个开源分布式服务框架，提供面向接口代理的高性能 RPC 调用、微服务治理等能力。

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。

B

绑定关系

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

又称为模板占位符，指消息模板中可被动态替换的部分。

模板参数值

指替换模板占位符的具体内容。

模板推送

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

一次消息推送请求标识为一次任务。

T

TaobaoRemoting（TR）

TR 框架指基于蚂蚁集团提供的供 RPC 使用的底层通信框架。

推送目标 ID

指要推送的目标，可能是 Android 设备的 Ad-token、iOS 设备的 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

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

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

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

由系统自动生成或用户自定义，用于在业务方系统中唯一标识消息。

应用 ID

应用标识，在创建应用时生成。

用户标识

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、用户绑定以及厂商通道的关系绑定，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。

- **厂商通道推送**

  厂商通道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 和厂商推送平台申请设备标识，在回调中分别返回 mPaaS 的设备标识和厂商的设备标识信息。

  若要使用厂商通道推送，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厂商的设备标识，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。

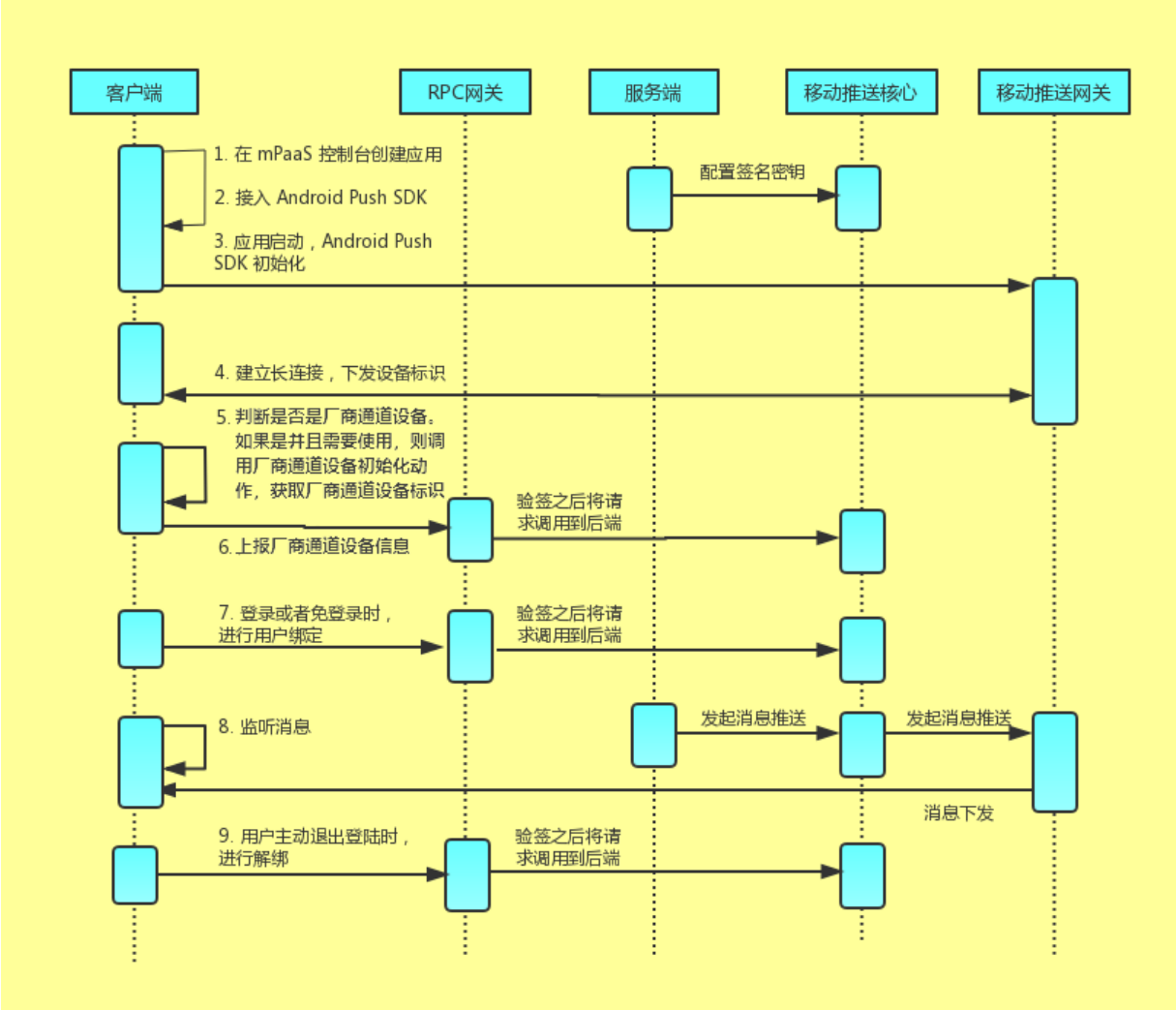
> **说明**
>
> 在请求设备标识 (token) 部分，若是小米、华为或其他已经接入厂商推送平台的手机，还会向厂商推送平台请求设备标识，需要等待两个设备标识返回，通过调用 report 接口将两者绑定，才能使用厂商的推送通道。普通手机只需要使用 mPaaS 返回的设备标识。

了解不同设备平台对应的消息推送接入流程：

- 中国内地安卓设备
- 苹果及国外安卓设备

中国内地安卓设备

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。针对中国内地的安卓设备，移动推送服务提供了自建网关。整个流程如下图所示：



其中：

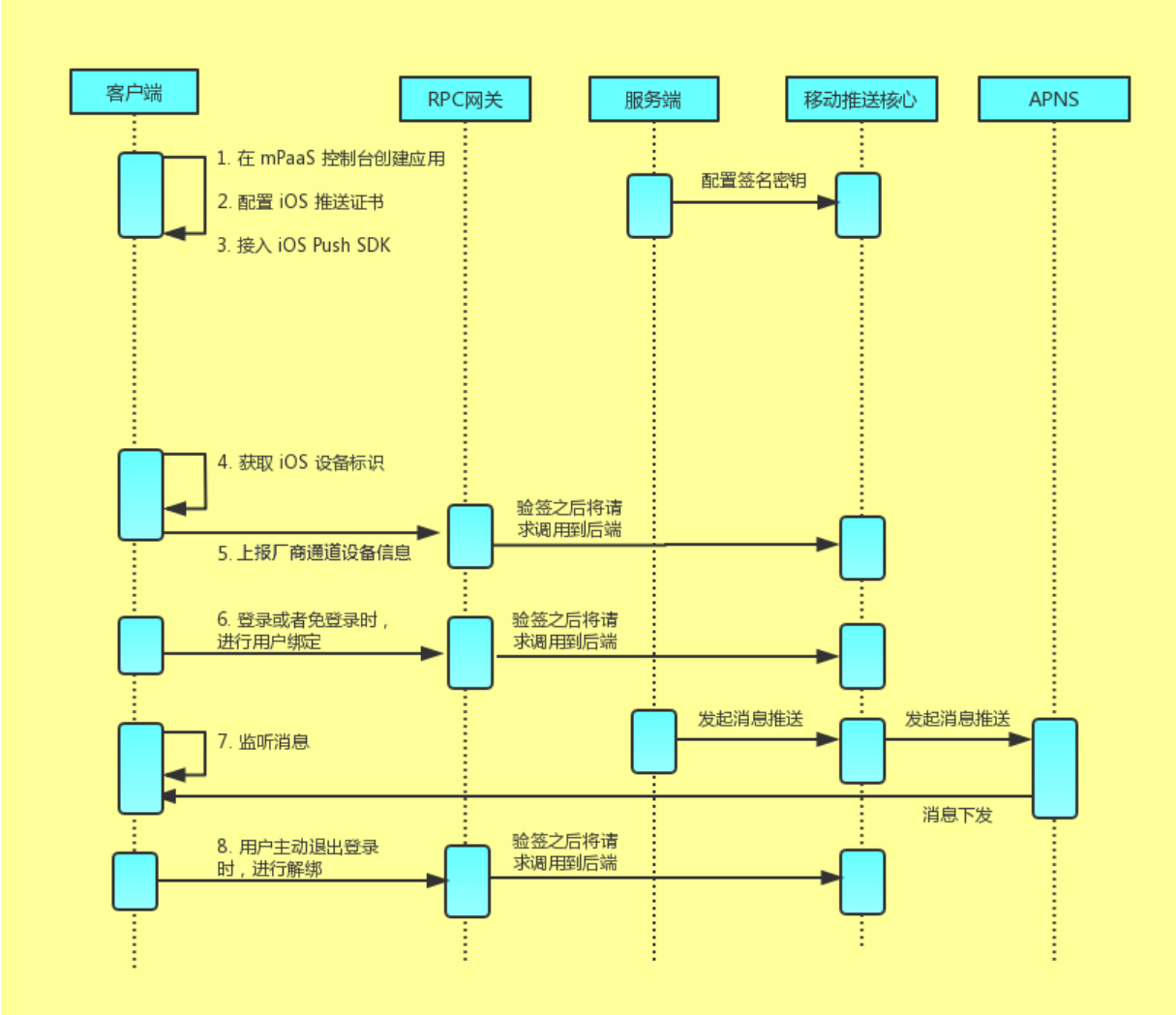
- 应用启动时，客户端同移动推送网关建立长链接，如果客户端建链信息中未携带设备标识，移动推送网关将下发设备标识。
- 如果用户开启小米、华为等厂商通道，且客户端属于这些厂商通道的机型，那么这些厂商通道的 SDK 会进行初始化动作，与对应厂商的推送网关建立长连接服务并获取厂商通道设备标识。
- 获取厂商通道设备标识后，客户端调用设备上报 RPC 接口，上报厂商通道设备信息。
- 应用用户在客户端上发起登录。
- 服务端收到用户登录请求，用户登录成功，可以选择在此时向移动推送核心发起用户和设备绑定请求。
- 服务端发起推送请求。
- 移动推送核心获取到推送请求，移动推送核心根据推送类型进行区分：
  - 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 和组件化（Portal & Bundle）两种接入方式。

前提条件

- 已将 mPaaS 接入到工程。
  - 若采用原生 AAR 方式接入，需要先 将 mPaaS 添加到您的项目中将 mPaaS 添加到项目。
  - 若采用组件化方式接入，需要先完成 接入流程组件化接入流程。
- 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 步骤 3 将配置文件添加到项目中将配置文件添加到项目。
- 本文中的 `MPPushMsgServiceAdapter` 方法仅适用于基线 10.1.68.32 及以上版本。若当前使用的基线版本低于 10.1.68.32，可参考 mPaaS 10.1.68 升级指南mPaaS 升级指南 升级基线版本。

> **说明**
> 旧版本中的 `AliPushRcvService` 方法仍可继续使用，如需查看旧版本文档，可 单击此处下载。

操作步骤

要使用消息推送服务，您需要完成以下接入步骤：

1. 添加推送 SDK。添加推送 SDK 依赖和 AndroidManifest 配置。
   i. 添加 SDK 依赖。根据不同的接入方式进行相应的操作：
      - 原生 AAR 方式：在工程中通过 **组件管理（AAR）** 安装 **消息推送（PUSH）** 组件。更多信息，请参考 管理组件依赖（原生 AAR）管理组件依赖（原生 AAR）。
      - 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 接入流程添加组件依赖。

ii. 添加 `AndroidManifest` 配置。在 `AndroidManifest.xml` 中添加以下内容：

> **说明**
> 如果是采用组件化方式接入，需要在 Portal 工程中添加 `AndroidManifest` 配置。

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

<service
    android:name="com.alipay.pushsdk.push.NotificationService"
    android:enabled="true"
    android:exported="false"
    android:label="NotificationService"
    android:process=":push">
    <intent-filter>
        <action android:name="${applicationId}.push.action.START_PUSHSERVICE" />
    </intent-filter>
</service>
<receiver
    android:name="com.alipay.pushsdk.BroadcastActionReceiver"
    android:enabled="true"
    android:exported="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>
```

iii. 推送 SDK 为了提高消息的到达率，内置了进程保活功能，包括上述 `com.alipay.pushsdk.BroadcastActionReceiver` 监听系统广播唤起推送进程，以及进程回收后的自动重启，接入时可以根据自身需求决定是否启用这些功能：

a. 如不需要监听系统启动广播，可删除：

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<action android:name="android.intent.action.BOOT_COMPLETED" />
```

b. 如不需要监听网络切换广播，可删除：

```
<action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
```

c. 如不需要监听用户唤醒广播，可删除：

```
<action android:name="android.intent.action.USER_PRESENT" />
```

d. 如不需要监听充电状态变化广播，可删除：

```
<action android:name="android.intent.action.ACTION_POWER_CONNECTED" />
```

e. 如不需要监听上述所有广播，可将 `com.alipay.pushsdk.BroadcastActionReceiver` 的 `android:enabled` 属性设置为 false 。

f. 如不需要推送进程回收后自动重启，可在 `application` 节点下添加以下配置：

```
<meta-data
  android:name="force.kill.push"
  android:value="on" />
```

> **说明**
> 该配置仅在基线版本 10.2.3.21 及以上版本中有效。

2. 初始化。初始化推送服务，建立客户端和移动推送网关之间的长连接，这个长连接由推送 SDK 维护，即自建通道。

根据不同的接入方式进行相应的操作：

- 原生 AAR 方式
  - 如已在 `Application` 中调用初始化 mPaaS，则在 `MP.init()` 方法之后调用：

JavaScript

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

  - 如未调用 mPaaS 初始化方法，则在 `Application` 中调用：

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

- 组件化方式

在 `LauncherApplicationAgent` 或 `LauncherActivityAgent` 的 `postInit` 方法中调用：

```
MPPush.init(context);
```

3. 创建 Service。创建 Service 继承 `MPPushMsgServiceAdapter`，重写 `onTokenReceive` 方法，接收自建通道下发的设备标识（token）。

```
public class MyPushMsgService extends MPPushMsgServiceAdapter {

    /**
     * 收到自建通道 token 的回调
     *
     * @param token 自建通道 token
     */
    @Override
    protected void onTokenReceive(String token) {
        Log.d("收到自建通道 token: " + token);
    }

}
```

在 `AndroidManifest.xml` 中声明该 Service：

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

完成此步骤后，就可以在控制台通过设备维度推送消息了，需要填写的设备标识就是收到的 token。

4. 绑定用户 ID。此用户 ID 由开发者自定义，既可以是真实用户系统的用户标识，也可以是能和每个用户形成映射关系的其他参数，例如账号、手机号等。

   在收到 token 后，可以将 token 和用户 ID 绑定：

```
String userId = "自定义 userId";
ResultPbPB bindResult = MPPush.bind(context, userId, token);
Log.d("绑定 userId " + (bindResult.success ? "成功" : ("错误：" + bindResult.code)));
```

   如已调用 `MPLogger` 设置过用户 ID，可以在绑定时不传入用户 ID，例如：

```
MPLogger.setUserId("自定义 userId");
ResultPbPB bindResult = MPPush.bind(context, token);
```

   如需解绑用户 ID，例如用户退出登录，可调用：

```
ResultPbPB unbindResult = MPPush.unbind(context, userId, token);
ResultPbPB unbindResult = MPPush.unbind(context, token);
```

   完成此步骤后，即可在控制台通过用户维度推送消息，需要填写的用户标识就是自定义的用户 ID。

5. （可选）绑定用户手机号。

> **重要**
> 目前，仅杭州非金融区提供短信补充服务。

在收到 token 后，还可以将 token 和用户的手机号码绑定，同时将 token 与用户 ID 和手机号绑定。绑定手机号码后，用户就可以通过该手机号码收到相关推送短信。

```
String userId = "自定义 userId";
String phoneNumber = "138xxxxxxxx"
ResultPbPB bindResult = MPPush.bind(context, userId, token,phoneNumber);
Log.d("绑定 userId " + (bindResult.success ? "成功" : ("错误：" + bindResult.code)));
```

相关操作

- 为提升消息推送的到达率，推荐接入 Android 手机厂商提供的推送通道，目前支持华为、小米、OPPO、vivo，具体操作参见 接入厂商通道。
- 收到消息后会自动发送通知，用户点击通知可自动打开网页，如需根据自定义的 DeepLink 跳转应用内页面，或自定义收到消息后的行为，请参见 通知点击处理。

更多其他功能，参见 高级功能 说明文档。

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 配置服务端。

4.1.2. 通知点击处理

对于已接入厂商通道，并且运行在对应厂商手机上的应用，服务端推送消息时默认优先推向厂商通道；对于其他应用，服务端推送消息时会推向自建通道。

- 自建通道收到消息后，推送 SDK 会自动发出通知，用户点击后可自动打开网页。

> **重要**
> 推送 SDK 使用的通知 ID 从 10000 开始，请确保您使用的其他通知 ID 不会与之冲突。

  - 如需跳转到应用内页面，请参考 跳转应用内页面 。
  - 如需处理收到的消息，请参考 自定义消息处理 。
- 厂商通道收到消息后手机系统会自动发出通知，推送 SDK 和开发者都无法干预，只有当用户点击通知后，推送 SDK 才能收到消息，并自动打开网页，如需：
  - 跳转到应用内页面，请参考 跳转应用内页面 。
  - 处理消息点击后的跳转，请参考 自定义消息处理 。

前提条件

- 本文中的 `MPPushMsgServiceAdapter` 方法仅适用于基线 10.1.68.32 及以上版本。若当前使用的基线版本低于 10.1.68.32，可参考 mPaaS 升级指南 升级基线版本。
- 旧版本中的 `AliPushRcvService` 方法仍可继续使用，如需查看旧版本文档，可 单击此处下载。

跳转应用内页面

如需跳转到应用内的指定页面，可以在推送消息的跳转地址里填入自定义的 DeepLink，例如 `mpaas://navigate`，同时在应用内设置一个路由 Activity 来接收 DeepLink，再分发到其他页面。

路由 Activity 需要在 `AndroidManifest.xml` 中添加相应的 `intent-filter`，例如：

```
<activity android:name=".push.LauncherActivity"
    android:launchMode="singleInstance">
    <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>
```

路由 Activity 中获取 URI 和消息：

```
Uri uri = intent.getData();
MPPushMsg msg = intent.getParcelableExtra("mp_push_msg");
```

自定义消息处理

如需处理消息，可以重写 `MPPushMsgServiceAdapter` 的 `onMessageReceive` 和 `onChannelMessageClick` 方法：

```
public class MyPushMsgService extends MPPushMsgServiceAdapter {

    /**
     * 自建通道收到消息的回调（非厂商通道）
     *
     * @param msg 收到的消息
     * @return 是否处理了消息：
     * 返回 true，则 SDK 不会处理消息，开发者需根据需求处理收到的消息，包括发通知和通知点击后的跳转
     * 返回 false，则 SDK 会自动发通知和添加通知点击后的跳转
     */
    @Override
    protected boolean onMessageReceive(MPPushMsg msg) {
        Log.d("从自建通道收到消息：" + msg.toString());
        // 处理消息，例如发自定义通知
        return true;
    }

    /**
     * 厂商通道的消息展示在通知栏，通知被点击后的回调
     *
     * @param msg 收到的消息
     * @return 是否处理了消息的点击：
     * 返回 true，则 SDK 不会处理厂商通道的通知点击，开发者需根据需求处理点击后的跳转
     * 返回 false，则 SDK 会自动处理通知点击后的跳转
     */
    @Override
    protected boolean onChannelMessageClick(MPPushMsg msg) {
        Log.d("从厂商通道到的消息被点击：" + msg.toString());
        // 处理被点击后的逻辑
        return true;
    }

}
```

其中 `MPPushMsg` 封装了消息的所有参数：

```
String id = msg.getId(); // 消息 ID
boolean isSilent = msg.isSilent(); // 是否静默消息

String title = msg.getTitle(); // 消息标题
String content = msg.getContent(); // 消息内容

String action = msg.getAction(); // 跳转类型，0：URL；1：自定义 DeepLink
String url = msg.getUrl(); // 跳转地址，URL 或 DeepLink

int pushStyle = msg.getPushStyle(); // 消息类型，0：普通消息，1：大文本消息，2：图文消息
String iconUrl = msg.getIconUrl(); // 图文消息的小图标
String imageUrl = msg.getImageUrl(); // 图文消息的图片

String customId = msg.getCustomId(); // 自定义的消息 ID
String params = msg.getParams(); // 扩展参数
```

如处理消息后，可能还需要上报消息埋点，否则控制台的推送使用分析将无法统计到正确的数据：

```
MPPush.reportPushOpen(msg); // 上报消息被打开
MPPush.reportPushIgnored(msg); // 上报消息被忽略
```

其中，对于自建通道的消息：

- 静默消息，无需上报。
- 非静默消息，需要上报被打开和被忽略的消息，可通过 `Notification.Builder` 的 `setContentIntent` 、`setDeleteIntent` 方法或其他有效方式来监听消息被用户打开和忽略。

对于厂商通道的消息，无需上报。

4.1.3. 接入厂商通道

4.1.3.1. 华为推送

本文介绍华为推送的接入流程，主要包括以下三个步骤。

1. 注册华为推送
2. 接入华为推送
3. 测试华为推送

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。

接入华为推送

推送 SDK 支持接入华为 HMS2 和 HMS5，但二者只能择其一接入。

- HMS2 版本过旧，如果您是首次接入，推荐接入 HMS5。
- 如果您是从 HMS2 升级到 HMS5，请先删除下文列出的 HMS2 `AndroidManifest` 配置。

接入华为推送 - HMS5.x 版本

1. 添加 **推送 - HMS5** 组件，方式与添加推送 SDK 相同，参见 添加推送 SDK 。

> **说明**
> 推送 - HMS5 组件仅包含适配代码，不包含 HMS SDK，请按照下文单独添加 HMS SDK 依赖。

2. 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。
3. 在项目根目录下的 `build.gradle` 文件中配置 HMS SDK 的 Maven 仓地址。

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

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

```
 dependencies {
     implementation 'com.huawei.hms:push:5.0.2.300'
 }
```

   - HMS SDK 版本会经常更新，最新版本号可以参考 HMS SDK 版本更新说明。
   - 当前适配的版本为 5.0.2.300，如需使用更高版本，可根据需求修改，通常来说厂商 SDK 都会向下兼容，如不兼容可加入钉钉群 41708565 反馈适配新版的需求。
5. 如需使用混淆，则要添加相关混淆配置：
   - 所有接入方式均需要添加 华为推送混淆规则。
   - 如采用的是 AAR 接入方式，还需要 添加 mPaaS 混淆规则。

接入华为推送 - HMS2.x 版本

1. 添加 **推送 - 华为2** 组件，方式与添加推送 SDK 相同，参见 添加 SDK 。 当前内置的 HMS2 SDK 版本为 2.5.2.201。
2. 配置 `AndroidManifest.xml` （组件化方式在 Portal 工程中添加），并替换其中的 `com.huawei.hms.client.appid` 的值。

```
<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"
            android:exported="true"
            >
            <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:exported="true"
            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>
```

3. 如需使用混淆，则要添加相关混淆配置：
   - 如采用的是 AAR 接入方式，需要 添加 mPaaS 混淆规则。
   - 如采用其他接入方式，则无需添加。

测试华为推送

1. 接入华为推送后，您可以在华为手机上启动应用并确保调用了初始化方法（参见 消息推送初始化），推送 SDK 会自动获取华为推送的厂商 token 并上报。
2. 您可以在杀掉应用进程的情况下推送测试消息：
   - 如果仍然能收到消息，说明您的应用成功接入华为推送。
   - 如果不能收到消息，请按照下文进行问题排查。

排查问题

1. 检查华为配置和参数是否和华为推送后台一致：
   - 接入 HMS2 请检查 `AndroidManifest.xml` 中相关配置是否添加，`com.huawei.hms.client.appid` 是否和华为推送后台一致。
   - 接入 HMS5 请检查 `agconnect-services.json` 是否存在，存放位置是否正确。
2. 检查 mPaaS 控制台是否开启了华为通道（参见 配置华为推送渠道），以及相关配置是否和华为推送后台一致。
3. 查看 logcat 日志进行排查：
   i. 选择 push 进程，过滤 `mPush.PushProxyFactory`，检查是否存在以下日志：

   ```
   D/mPush.PushProxyFactory: found proxy com.mpaas.push.external.hms.Creator (HMS2)
   D/mPush.PushProxyFactory: found proxy com.mpaas.push.external.hms5.Creator (HMS5)
   ```

   若无，说明添加 **推送 - 华为2** 或 **推送 - HMS5** 组件可能存在问题，请确认是否正确添加。

   ii. 选择主进程，过滤 `mHMS`，检查是否获取到了华为推送的厂商 token，如出现日志 `get token failed`：说明获取华为厂商 token 失败，错误码参见 华为推送错误码。

   iii. 选择主进程，过滤 `report channel token`，检查上报华为厂商 token 是否成功，如出现以下日志：

   ```
   report channel token error: xxxx
   ```

   说明上报厂商 token 失败，请检查 mPaaS 配置文件 的 `base64Code` 是否有值，以及获取配置文件时上传的 apk 签名和当前应用是否一致。

其他问题

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。

4.1.3.2. 荣耀推送

本文介绍荣耀推送的接入流程，主要包括以下三个步骤。

1. 注册荣耀推送
2. 接入荣耀推送
3. 测试荣耀推送

注册荣耀推送

登录荣耀开发官网，注册账号并且开启推送服务。详情请参见 荣耀推送开启步骤。

接入荣耀推送

1. 添加 **推送 > HONOR** 组件，方式与添加推送 SDK 相同，参考 添加推送 SDK 。

> **说明**
> 推送 > HONOR 组件仅包含适配代码，不包含荣耀推送 SDK，请按照下文单独添加荣耀推送 SDK 依赖。

2. 开发环境准备，开发环境需要符合荣耀推送集成的环境，具体可以参考 开发准备-环境信息。
3. 添加配置文件。 在 荣耀开发者服务平台 中下载 `mcs-services.json` 配置文件，具体参考 添加应用配置文件。
4. 配置 sdk 的仓库地址。具体参考官方链接文档 配置 sdk 的 Maven 仓库地址。
5. 添加依赖配置。在应用级的 `build.gradle` 文件中，在 dependencies 中添加如下编译依赖。

```
dependencies {
    // 添加如下配置
    implementation 'com.hihonor.mcs:push:7.0.61.302'
}
```

   - 具体请参考 添加依赖配置。
   - 如果需要更新版本，版本信息可以参考 版本信息。
   - mPaaS 当前适配的版本为 7.0.61.302，如需使用更高版本，可根据需求修改，通常来说厂商 SDK 都会向下兼容。
6. 如需使用混淆，则要添加相关混淆配置：
   - 所有接入方式均需要添加 荣耀推送混淆脚本。
   - 如采用的是 AAR 接入方式，还需要 添加混淆规则。

测试荣耀推送

> **重要**
> 请注意，荣耀 Magic OS 8.0 以下（不含 8.0）版本会继续使用华为推送适配层。

1. 接入荣耀推送后，您可以在荣耀手机上启动应用并确保调用了初始化方法（参见 快速开始），推送 SDK 动获取荣耀推送的厂商 token 并上报。
2. 您可以在杀掉应用进程的情况下推送测试消息：
   - 如果仍然能收到消息，说明您的应用成功接入荣耀推送。
   - 如果不能收到消息，请按照下文进行问题排查。

排查问题

1. 检查荣耀配置和参数是否和荣耀推送后台一致，检查 `AndroidManifest.xml` 中相关配置是否添加，`com.hihonor.push.app_id` 是否和荣耀推送后台一致。
2. 检查 `mcs-services.json` 文件是否存在，存放位置是否正确。
3. 检查 mPaaS 控制台是否开启了荣耀通道（参见 配置荣耀推送渠道），以及相关配置是否和荣耀推送后台一致。
4. 查看 logcat 日志进行排查：

i. 选择 push 进程，过滤 `mPush.PushProxyFactory` ，检查是否存在以下日志：

```
D/mPush.PushProxyFactory: found proxy com.mpaas.push.external.honor.Creator
```

ii. 选择主进程，过滤 mHonor，检查是否获取到了荣耀推送的厂商 token，如出现日志 `get token failed` 则说明获取荣耀厂商 token 失败，错误码参见 荣耀推送错误码。

iii. 选择主进程，过滤 `report channel token`，检查上报荣耀厂商 token 是否成功，如出现以下日志：

```
report channel token error: xxxx
```

说明上报厂商 token 失败，请检查 步骤 3 将配置文件添加到项目中 的 `base64Code` 是否有值，以及获取配置文件时上传的 apk 签名和当前应用是否一致。

若无，说明添加 **推送 > HONOR** 组件可能存在问题，请确认是否正确添加。

其他问题

推送支持哪些机型和系统版本

当前荣耀的厂商推送渠道支持 Magic OS 8.0 及以上版本系统的荣耀手机，Magic OS 8.0 版本以下（不含 8.0）的版本继续使用华为的厂商推送渠道。

4.1.3.3. OPPO 推送

本文介绍 OPPO 推送的接入流程，主要包括以下三个步骤。

1. 注册 OPPO 推送
2. 接入 OPPO 推送
3. 测试 OPPO 推送

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。

接入 OPPO 推送

1. 安装 **推送 - OPPO** 组件，方式与添加推送 SDK 相同，参见 添加 SDK。 **推送 - OPPO** 组件仅包含适配代码，不包含 OPPO Push SDK 。
2. 前往 OPPO SDK 文档 下载 SDK 并集成到主工程中。 当前适配的版本为 `3.4.0` ，如需使用更高版本，可根据需求修改，通常来说厂商 SDK 都会向下兼容，如不兼容可加入钉钉群 41708565 反馈适配新版的需求。
3. 配置 `AndroidManifest.xml` （组件化方式在 Portal 工程中添加），并替换其中的 `com.oppo.push.app_key` 、 `com.oppo.push.app_secret` 的值。

```
    <uses-permission android:name="com.coloros.mcs.permission.RECIEVE_MCS_MESSAGE" />
    <uses-permission android:name="com.heytap.mcs.permission.RECIEVE_MCS_MESSAGE"/>

    <application>
        <service

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

4. 如需使用混淆，则要添加相关混淆配置：
   - 所有接入方式均需要添加 OPPO 推送混淆规则。
   - 如采用的是 AAR 接入方式，还需要 添加 mPaaS 混淆规则。
5. 如果使用的是 OPPO 推送版本 `3.4.0` ，还需添加如下依赖：

```
implementation 'commons-codec:commons-codec:1.15'
```

测试 OPPO 推送

1. 接入 OPPO 推送后，您可以在 OPPO 手机上启动您的应用并确保调用了初始化方法（参见 消息推送初始化），推送 SDK 会自动获取 OPPO 推送的厂商 token 并上报。
2. 可以在杀掉应用进程的情况下推送测试消息：
   - 如果仍然能收到消息，说明应用成功接入 OPPO 推送。
   - 如果不能收到消息，请按照下文进行问题排查。

排查问题

1. 检查 `AndroidManifest.xml` 配置是否添加，以及其中 `com.oppo.push.app_key` 、`com.oppo.push.app_secret` 的值是否和 OPPO 开放平台一致。
2. 检查 mPaaS 控制台是否开启了 OPPO 通道（参见 配置 OPPO 推送通道），以及相关配置是否和 OPPO 开放平台一致。
3. 查看 logcat 日志进行排查：
   i. 选择 push 进程，过滤 `mPush.PushProxyFactory` ，检查是否存在以下日志：

   ```
   D/mPush.PushProxyFactory: found proxy com.mpaas.push.external.oppo.Creator
   ```

   若无，说明添加 **推送 - OPPO** 组件可能存在问题，请确认是否正确添加。
   ii. 选择 push 进程，过滤 `mOPPO` ，检查是否获取到了 OPPO 推送的厂商 token，如出现以下日志（“OPPO onRegister error”或“responseCode”不为 0）：说明 OPPO 推送注册失败，错误码参见 OPPO 推送错误码，下拉至错误码定义说明章节。
   iii. 选择主进程，过滤 `report channel token` , 检查上报 OPPO 厂商 token 是否成功，如出现以下日志：

   ```
   report channel token error: xxxx
   ```

   说明上报厂商 token 失败，请检查 mPaaS 配置文件 的 `base64Code` 是否有值，以及获取配置文件时上传的 apk 签名和当前应用是否一致。
   iv. 选择 push 进程，过滤 `mcssdk` ，查看 OPPO 推送内部日志。

其他问题

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。

4.1.3.4. vivo 推送

本文主要介绍 vivo 推送的接入流程，主要包括以下三个步骤。

1. 注册 vivo 推送
2. 接入 vivo 推送
3. 测试 vivo 推送

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。

接入 vivo 推送

1. 添加 **推送 - vivo** 组件，方式与添加推送 SDK 相同，参见 添加 SDK。**推送 - vivo** 组件仅包含适配代码，不包含 vivo Push SDK。
2. 前往 vivo SDK 文档 下载 SDK 并集成到主工程中。 当前适配版本为 v2.3.4，如需使用更高版本，可根据需求修改，通常来说厂商 SDK 都会向下兼容，如不兼容可加入钉钉群 41708565 反馈适配新版的需求。
3. 配置 `AndroidManifest.xml` （组件化方式在 Portal 工程中添加），并替换其中的 `com.vivo.push.api_key` 、`com.vivo.push.app_id` 的值。

```
<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" />
</application>
```

4. 如需使用混淆，则要添加相关混淆配置：
   - 所有接入方式均需要添加 vivo 推送混淆规则。
   - 如采用的是 AAR 接入方式，还需要 添加 mPaaS 混淆规则。

测试 vivo 推送

1. 接入 vivo 推送后，您可以在 vivo 手机上启动您的应用并确保调用了初始化方法（参见 消息推送初始化），推送 SDK 会自动获取 vivo 推送的厂商 token 并上报。
2. 可以在杀掉应用进程的情况下推送测试消息：
   - 如果仍然能收到消息，说明应用成功接入 vivo 推送。
   - 如果不能收到消息，请按照下文排查。

排查问题

1. 检查 `AndroidManifest.xml` 配置是否添加，以及其中 `com.vivo.push.api_key` 、`com.vivo.push.app_id` 的值是否和 vivo 开放平台一致。
2. 检查 mPaaS 控制台是否开启了 vivo 通道（参见 配置 vivo 推送通道），以及相关配置是否和 vivo 开放平台一致。
3. 查看 logcat 日志进行排查：
   i. 选择 push 进程，过滤 `mPush.PushProxyFactory` ，检查是否存在以下日志：

   ```
   D/mPush.PushProxyFactory: found proxy com.mpaas.push.external.vivo.Creator
   ```

   若无，说明添加 **推送 - vivo** 组件可能存在问题，请确认是否正确添加。

   ii. 选择 push 进程，过滤 `mVIVO` ，检查是否获取到了 vivo 推送的厂商 token，如出现日志 `"fail to turn on vivo push"` ：说明 vivo 推送注册失败，错误码（state）参见 vivo 推送错误码 。

   iii. 选择主进程，过滤 `report channel token` , 检查上报 vivo 厂商 token 是否成功，如出现以下日志：

   ```
   report channel token error: xxxx
   ```

   说明上报厂商 token 失败，请检查 mPaaS 配置文件 的 `base64Code` 是否有值，以及获取配置文件时上传的 apk 签名和当前应用是否一致。

常见问题

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

4.1.3.5. 小米推送

本文主要介绍小米推送的接入流程，主要包括以下三个步骤。

1. 注册小米推送
2. 接入小米推送
3. 测试小米推送

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

接入小米推送

1. 添加 **推送 - 小米** 组件。添加方式与添加推送 SDK 相同，参见 添加推送 SDK。

   当前内置的小米推送 SDK 版本为 4.0.2，历史版本参见 发布说明。

2. 配置 `AndroidManifest.xml`（组件化方式在 Portal 工程中添加），并替换其中的 `xiaomi_appid`、`xiaomi_appkey` 的值。

```
<permission
    android:name="${applicationId}.permission.MIPUSH_RECEIVE"
    android:protectionLevel="signature"/>
<uses-permission android:name="${applicationId}.permission.MIPUSH_RECEIVE"/>
<application>

    <!-- value 斜杠空格要保留 -->
    <meta-data
        android:name="xiaomi_appid"
        android:value="\ 2xxxxxxxxxxxxxx" />
    <!-- value 斜杠空格要保留 -->
    <meta-data
        android:name="xiaomi_appkey"
        android:value="\ 5xxxxxxxxxxxxxx" />

</application>
```

测试小米推送

1. 接入小米推送后，您可以在小米手机上启动应用并确保调用了初始化方法（参见 消息推送初始化），推送 SDK 会自动获取小米推送的厂商 token 并上报。
2. 可以在杀掉应用进程的情况下推送测试消息：
   - 如果仍然能收到消息，说明应用成功接入小米推送。
   - 如果不能收到消息，请按照下文进行问题排查。

排查问题

1. 检查 `AndroidManifest.xml` 配置是否添加，以及其中 `xiaomi_appid`、`xiaomi_appkey` 的值是否和小米开放平台一致。
2. 检查 mPaaS 控制台是否开启了小米通道（参见 配置小米推送通道），以及相关配置是否和小米开放平台一致。
3. 查看 logcat 日志进行排查：

   i. 选择 push 进程，过滤 `mPush.PushProxyFactory`，检查是否存在以下日志：

   ```
   D/mPush.PushProxyFactory: found proxy com.mpaas.push.external.mi.Creator
   ```

   若无，说明添加 **推送 - 小米** 组件可能存在问题，请确认小米推送组件是否正确添加。

   ii. 选择 push 进程，过滤 `mMi`，检查是否获取到了小米推送的厂商 token。

   如出现以下日志（`register_fail`），说明小米推送注册失败。错误原因（`reason`）参见 小米推送错误码。如果错误原因显示 UNKNOWN，通常是 `xiaomi_appid` 或 `xiaomi_appkey` 错误。结果码（`resultCode`）参见 小米推送服务端错误码。

iii. 选择主进程，过滤 `report channel token`，检查上报小米厂商 token 是否成功，如出现以下日志：

```
report channel token error: xxxx
```

说明上报厂商 token 失败，请检查 mPaaS 配置文件 的 `base64Code` 是否有值，以及获取配置文件时上传的 apk 签名和当前应用是否一致。

4.1.3.6. FCM 推送

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 推送通道的接入过程。

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 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：

1. 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。

2. 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。

3. 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
    // ...
  }
}
```

4. 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 {
  // ...
}
```

5. 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 librar
ies
    // When using the BoM, you don't specify versions in Firebase library dependencie
s
    implementation 'com.google.firebase:firebase-messaging'
    implementation 'com.google.firebase:firebase-analytics'
}
```

接入 mPaaS

操作步骤如下：

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

```
dependencies {
    implementation 'com.mpaas.push:fcm-adapter:0.0.2'
}
```

2. 接入消息推送组件，mPaaS 基线版本要求如下：
   - 对于 `com.mpaas.push:fcm-adapter:0.0.2` ，基线版本不能低于 10.1.68.34。
   - 对于 `com.mpaas.push:fcm-adapter:0.0.1` ，基线版本不能低于 10.1.68.19。
3. 接收推送消息。由于 FCM SDK 本身的特性，通过控制台或服务端向 FCM 通道推送的消息，客户端并不总是会从 FCM 通道接收到，也可能会从自建通道接收到。具体规则为：
   - 当应用在后台或应用进程被杀死时，消息会从 FCM 通道收到，直接展示在通知栏，和其他厂商通道一样。
   - 当应用在前台时，消息会被 FCM 透传给应用，应用会从自建通道收到消息。
4. （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), To
ast.LENGTH_SHORT).show();
        }
    }
}
```

4.1.4. 厂商消息分类

为了改善终端用户推送体验、营造良好可持续的通知生态，各大厂商陆续对推送的消息根据分类进行限额限频。

简介

根据推送内容对消息进行分类管理，可以自定义 Channel ID。

- 适用所有 Android 渠道
- 创建客户端自定义渠道
- 推送时下发对应的渠道 ID

参数名称	类型	是否必填	示例	描述
channelId	String	否	channelId: "channelIdTest"	Android 通知 channelId

- 若需要下发厂商通道重要级别消息，请参考下文各个厂商消息分类的使用指南

华为分类

厂商针对消息分类的说明

根据消息内容，华为推送将通知分类为 **服务与通讯、资讯营销** 两大类别，并对不同类别消息的提醒方式、消息样式进行差异化管理，具体如下：

消息类型	服务与通讯	资讯营销
推送内容	包括社交通讯类消息和服务提醒类消息。	包括资讯类消息和营销类消息，指的是运营人员向用户发送的活动信息、内容推荐、资讯等
提醒方式（EMUI 10.0及以上）	锁屏、铃声、振动	静默通知，仅在下拉通知栏展示消息
消息样式	文本+小图	仅有文本
推送数量	不限量	自 **2023.01.05** 起，资讯营销类消息根据应用类型对每日推送数量进行上限管理，具体要求参见 不同应用类别的推送数量上限要求
配置方式	需要向华为申请自分类权益，审核通过后将信任开发者提供的分类信息，消息不经过智能分类。	默认

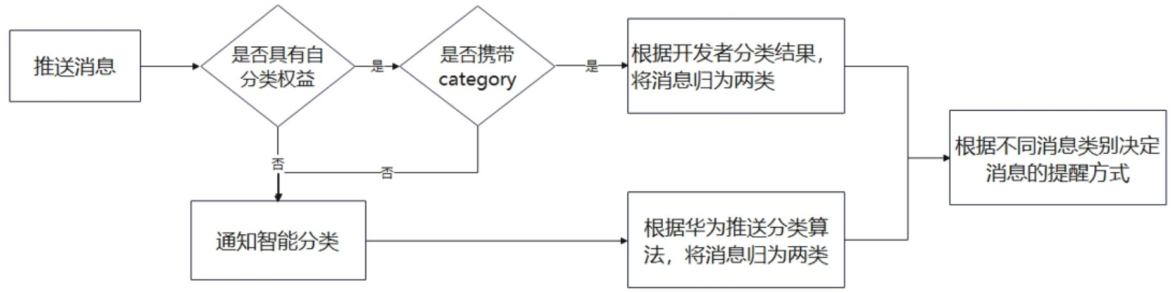
分类方式

消息智能分类

智能分类算法将根据您发送的内容等多个维度因素，自动将您的消息按照分类标准进行归类。

消息自分类

2021.07.01 起，华为推送服务开始接收开发者自分类权益的申请。申请成功后，允许开发者根据华为推送分类规范，自行对消息进行分类。



华为消息分类申请

自分类申请详情参见 华为消息分类管理方案。

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

在 mPaaS MPS 上对接华为消息分类传参枚举（thirdChannelCategory.hms）

传参（String）	含义
1	IM：即时聊天
2	VOIP：音视频通话
3	SUBSCRIPTION：订阅
4	TRAVEL：旅行类
5	HEALTH：健康
6	WORK：工作事项提醒
7	ACCOUNT：账号动态
8	EXPRESS：订单&物流
9	FINANCE：财务
10	DEVICE_REMINDER：设备提醒
11	SYSTEM_REMINDER：系统提示

12	MAIL：邮件
13	PLAY_VOICE：语音播报（仅透传消息支持）
14	MARKETING：内容推荐、新闻、财经动态、生活资讯、社交动态、调研、产品促销、功能推荐、运营活动（仅对内容进行标识，不会加快消息发送）

传参示例

参数名称	类型	是否必填	示例	描述
thirdChannelCategory	Map	否	thirdChannelCategory: {"hms": "9"}	示例传值为“9”表示华为 FINANCE 财务类型消息。其他取值详情请参见 厂商消息分类

荣耀分类

厂商针对消息分类的说明

根据消息内容，华为推送将通知分类为 **服务与通讯、资讯营销** 两大类别，具体如下：

消息类型	服务与通讯	资讯营销
推送内容	包括社交通讯类消息和服务提醒类消息。	包括资讯类消息和营销类消息，指的是运营人员向用户发送的活动信息、内容推荐、资讯等
提醒方式	锁屏展示+下拉通知栏展示，支持铃声、震动	静默通知，仅在下拉通知栏展示消息
消息样式	文本+小图	仅有文本
推送数量	不限量	资讯营销类消息根据应用类型对每日推送数量进行上限管理： • 新闻类（三级分类为新闻类）：5 条 • 其他应用类型：2 条 具体要求参见 不同应用类别的推送数量上限要求

分类方式

消息智能分类

智能分类算法将根据您发送的内容等多个维度因素，自动将您的消息按照分类标准进行归类。

消息自分类

允许开发者根据消息分类规范，自行对消息进行分类。

在 mPaaS MPS 上对接荣耀消息分类传参枚举（thirdChannelCategory.honor）

传参（String）	含义
1	服务通讯类
2	资讯营销类

传参示例

参数名称	类型	是否必填	示例	描述
thirdChannelCategory	Map	否	thirdChannelCategory: {"honor": "1"}	示例传值为“1”表示荣耀服务通讯类消息

小米消息分类

厂商针对消息分类的说明

根据《小米推送消息分类新规》，小米推送将消息分为 **私信消息** 和 **公信消息** 两个类别，若应用选择不接入私信或公信，则会接入 **默认** 通道。

消息类型	默认	公信消息	私信消息
推送内容	可按照小米的 公信场景说明	热点新闻、新品推广、平台公告、社区话题、有奖活动等，多用户普适性的内容	聊天消息、个人订单变化、快递通知、交易提醒、IoT 系统通知等与私人通知相关的内容
提醒方式	无	无	响铃、震动
推送数量限制	1 倍	2-3 倍，具体规则请参见“公信限制规则	不限量
用户接收数量限制	单个应用单个设备单日 1 条	单个应用单个设备单日 5-8 条	不限量
申请方式	无需申请	需在小米推送平台申请，详情请参见 channel 申请及接入方式	

小米消息分类申请

申请方式请参见小米官方文档 channel 申请及接入方法。

在 mPaaS MPS 上对接小米消息分类传参示例

参数名称	类型	是否必填	示例	描述
miChannelId	String	否	miChannelId:"miChannelIdTest"	小米厂商推送渠道的channelId

OPPO 消息分类

厂商针对消息分类的说明

消息类型	私信	公信
推送内容	针对用户有一定关注度，且希望能及时接收的信息，如即时聊天信息、个人订单变化、快递通知、订阅内容更新、评论互动、会员积分变动等。	公信是针对用户关注度不高，且对于接收这类信息并无心理预期，如热点新闻、新品推广、平台公告、社区话题、有奖活动等，多用户普适性的内容
推送数量限制	不限量	有公信类通道共享推送次数，当日达到推送量限制后，所有公信类通道将不能再推送消息； 推送限量：当累计用户数<50000时，按100000计算；当累计用户数≥50000时，按累计用户数*2计算
单用户推送限制（条/日）	不限量	• 新闻类（三级分类为新闻类）：5 条 • 其他应用类型：2 条 应用类别以创建应用时应用基本信息所提交的“软件分类”为准；若需修改应用类别，可在移动应用列表-应用详情内进行应用资料更新
配置方式	• 客户端创建自定义渠道。 • 私信申请邮件通过后，需要在OPPO 推送平台上登记该通道，并将通道对应属性设置为“私信”。	默认开通

OPPO 私信通道申请

- 私信通道权益申请
- 私信申请邮件通过后，需要在 OPPO 推送平台 上登记该通道，并将通道对应属性设置为 **私信**

在 mPaaS MPS 上对接 OPPO 消息分类传参示例

参数名称	类型	是否必填	示例	描述
channelId	String	否	channelId:"channelIdTest"	OPPO 私信通道channelId

vivo 消息分类

厂商针对消息分类的说明

- 通知开启的有效用户：应用集成的 push-sdk 订阅成功，且设备近 14 天内有联网的通知权限开启用户。
- 通知开启有效用户数＜10000，则运营消息量级默认为 10000。
- 通知开启的有效用户数及可发送运营消息量级，可在推送运营后台查询。
- 推送限额数以 **到达量** 计算，当日到达量超限则计入管控。

消息类型	系统消息	运营消息
推送内容	用户需要及时知道的消息，如：即时消息、邮件、用户设置的提醒、物流等通知	用户关注程度较低的消息，如：内容推荐、活动推荐、社交动态等通知
通知栏权限	• 默认响铃、震动、消息外显 • 默认锁屏、悬浮	• 默认无响铃、无震动、应用不存活时消息收纳进盒子 • 默认无悬浮、无锁屏
推送数量限制	3 倍通知开启有效用户数（可邮件申请消息不限量权限，详见 推送消息限制说明）	• 新闻类（三级分类为新闻类）：3 倍通知开启有效用户数 • 其他类：2 倍通知开启有效用户数
用户接收数量限制	无限制	• 新闻类（三级分类为新闻类）：5 条 • 其他类：2 条

在 mPaaS MPS 上对接 vivo 二级消息分类传参枚举（thirdChannelCategory.vivo）

传参（String）	含义
1	IM 用户间点对点聊天消息（私信、群聊等），包括聊天消息中的图片、文件传输、音频（或视频）通话，不包括未关注人的私信、官方号或者商家批量推送给用户的私信或广告。以及邮件提醒。
2	ACCOUNT 账号变动：账号上下线、状态变化、信息认证、会员到期、续费提醒、余额变动等。 资产变动：账户下的真实资产变动，交易提示、话费余额、流量、语音时长、短信额度等典型运营商提醒。
3	TODO 与个人日程安排相关，需要提醒用户需要处理某件事项。 • 会议提醒，开课提醒，预约提醒，差旅航班等出行相关消息。 • 推送对象为服务提供方：工单处理、状态流程提醒等工作流程消息；商家接单/发货/售后提醒等订单消息。 • 库存不足、售罄提醒、商品下架通知、限制提现、客诉警告、店铺限制、商品黑名单、交易违规、涉假/涉欺诈发货通知等商家运营提醒消息。

4	DEVICE_REMINDER • IoT设备发出的设备状态/信息/提示/告警等提醒消息 • 健康设备的提醒，包括运动量（步数、骑行里程、游泳距离等）、身体数据（心率、体重、体脂、消耗卡路里等 • 手机运行相关的提示及状态提醒
5	ORDER 电商购物、美食团购等各类商品服务中的订单相关信息，推送对象为用户。 • 下单成功、订单详情、订单状态、售后进度等 • 快递已发货、配送中，签收，取件等物流消息
6	SUBSCRIPTION 用户主动订阅关注，并有预期在特定时机接收到消息： • 主动订阅的专题内容、预约活动提醒、主动设置的直播开播提醒、书籍更新 • 设置的商品或机票降价、商品开团提醒 • 主动关注的行情动态提醒 • 主动设置的签到打卡提醒 • 付费订阅内容更新提醒等 **重要** 申请订阅类消息需要满足如下条件并提供完整证明： • 应用内支持用户“订阅/取消订阅”，用户界面需要至少出现“订阅”或“预约”等字样 • 订阅是用户的主动行为，在用户未订阅的情况下，不向用户推送消息 • 用户订阅后，应用内用户界面有明确提示，用户将收到订阅相关的推送消息。例如：您将收到 xx 消息推送 • 订阅消息的范围不宜过于宽泛、不具体。例如：订阅行情资讯，则过于宽泛、不具体 • 推送内容中需要体现该条推送是用户的订阅消息。例如：在消息标题或正文中携带“订阅消息”、“您订阅的……”等字样
7	NEWS 新近发生的、有价值的事实新闻内容。
8	CONTENT 内容型的信息推荐，包含热搜、点评、广告、书籍、音乐、视频、直播、课程、节目、游戏宣传、社区话题等。以及： • 各垂直类目的相关内容资讯 • 天气预报：包括各类天气预报、天气预警提醒等 • 出行资讯：包括交规公告、驾考信息、导航路况、铁路购票公告、疫情消息，道路管控等

9	MARKETING • 非用户主动设置，需用户参与的活动提醒、小游戏提醒、服务或商品评价提醒等。如：抽奖、积分、签到、任务、分享、偷菜、领金币等 • 商品推荐，包括红包折扣、商家服务更新、店铺上新等。如：可能感兴趣、商品达到最低价、满减、促销、返利、优惠券、代金券、送红包、信用分增加等相关的通知 • 其他消息：用户调研问卷、功能介绍、邀请推荐、版本更新等
10	SOCIAL • 用户之间的社交互动提醒，如：好友动态、新增粉丝、添加好友、被赞、被@、被收藏、评论、留言、关注、回复、转发、陌生人消息等 • 用户推荐：附近的人、大 V、主播、异性、可能认识的人等

在 mPaaS MPS 上对接 vivo 消息分类传参示例

参数名称	类型	是否必填	示例	描述
classification	String	否	classification:"1"	用于传递 vivo 推送通道的消息类型： • 0 - 运营类消息 • 1 - 系统类消息 不填则默认为 1
thirdChannelCategory	Map	否	thirdChannelCategory: {"vivo": "1"}	示例传值为“1”表示 vivo IM 类型消息

> **说明**
>
> classification 参数传“0”代表运营消息，不经过智能分类二次修正，直接从运营消息总量扣除额度，并受用户接收条数限制的频控。
>
> classification 参数传“1”代表系统消息，经过智能分类二次修正，若智能分类识别出不是系统消息，会自动修正为运营消息，并扣除运营消息额度；若识别为系统消息，则从系统消息总量扣除额度。

MPS 对接厂商消息分类 Java 示例代码

厂商消息分类推送参数推荐都上传，MPS 会根据设备类型进行对应厂商分类参数的封装。

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // 创建 DefaultAcsClient 实例并初始化
        // 阿里云账号AccessKey拥有所有API的访问权限，建议您使用RAM用户进行API访问或日常运维。
        // 强烈建议不要把AccessKey ID和AccessKey Secret保存到工程代码里，否则可能导致AccessKey泄露，威胁您账号下所有资源的安全。
        // 本示例以将AccessKey ID和AccessKey Secret保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里。
        // 建议先完成环境变量配置
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
```

```
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // 地域 ID
            accessKeyId,
            accessKeySecret);

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);

request.setPushStyle(2);
String imageUrls = "{\"defaultUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"oppoUrl\":\"https://pre-mpaas.oss-cn-hangzhou.ali
yuncs.com/tmp/test.png\",\"miuiUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"fcmUrl\":\"https://pre-mpaas.oss-cn-hangzhou.aliy
uncs.com/tmp/test.png\",\"iosUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\"}";
String iconUrls = "{\"defaultUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"hmsUrl\":\"https://pre-mpaas.oss-cn-hangzhou.aliy
uncs.com/tmp/test.png\",\"oppoUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"miuiUrl\":\"https://pre-mpaas.oss-cn-hangzhou.ali
yuncs.com/tmp/test.png\"}";
request.setImageUrls(imageUrls);
request.setIconUrls(iconUrls);

request.setStrategyType(2);
request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circl
ype\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}");

Map<String,String> target = new HashMap<String, String>();
String msgKey = String.valueOf(System.currentTimeMillis());
target.put("user1024",msgKey);
request.setTargetMsgkey(JSON.toJSONString(target));

//厂商消息分类字段

//封装VIVO消息分类一级分类
request.setClassification("1");
//封装华为消息分类、荣耀消息分类和VIVO消息分类二级分类
Map<String, String> map = new HashMap<>();
map.put("hms", "2");
map.put("vivo", "3");
map.put("honor", "1");
pushSimpleReq.setThirdChannelCategory(map);
//封装小米消息分类
```

```
pushSimpleReq.setMiChannelId("miChannelIdTest");
//封装OPPO消息分类
pushSimpleReq.setChannelId("channelIdTest");


// 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();
}
```

4.1.5. 高级功能

在接入推送 SDK 后，您还可以进行以下客户端配置：

- 清除角标
- 上报厂商通道 token
- 自定义通知渠道（NotificationChannel）

前提条件

- 本文中的 `MPPushMsgServiceAdapter` 方法仅适用于基线 10.1.68.32 及以上版本。若当前使用的基线版本低于 10.1.68.32，可参考 mPaaS 升级指南 升级基线版本。
- 旧版本中的 `AliPushRcvService` 方法仍可继续使用，如需查看旧版本文档，可 单击此处下载。

清除角标

厂商通道收到的消息，可在应用图标上显示消息数量的角标。目前推送 SDK 仅支持华为通道自动清除角标。

- 设置用户点击通知时自动清除应用角标：

```
// 设置是否自动清除
boolean autoClear = true;
MPPush.setBadgeAutoClearEnabled(context, autoClear);
// 设置应用入口 Activity 类名，不设置无法清除角标
String activityName = "com.mpaas.demo.push.LauncherActivity";
MPPush.setBadgeActivityClassName(context, activityName);
```

- 在角标无法自动清除的场景下，例如用户主动点击应用图标进入应用时，可以在 `Application` 中调用以下方法主动清除角标：

```
MPPush.clearBadges(context);
```

上报厂商通道 token

如已接入厂商通道，推送 SDK 初始化后会收到厂商通道的 token，推送 SDK 会自动将厂商通道 token 和自建通道 token 绑定上报。

如有需要，可通过重写 `MPPushMsgServiceAdapter` 的 `onChannelTokenReceive` 和 `onChannelTokenReport` 方法监听厂商通道 token 的下发和上报：

```
public class MyPushMsgService extends MPPushMsgServiceAdapter {

    /**
     * 收到厂商通道 token 的回调
     *
     * @param channelToken 厂商通道 token
     * @param channel      厂商通道类型
     */
    @Override
    protected void onChannelTokenReceive(String channelToken, PushOsType channel) {
        Log.d("收到厂商通道 token: " + channelToken);
        Log.d("厂商: " + channel.getName());
    }

    /**
     * 厂商通道 token 上报结果的回调
     *
     * @param result 上报结果
     */
    @Override
    protected void onChannelTokenReport(ResultBean result) {
        Log.d("上报厂商 token " + (result.success ? "成功" : ("错误：" + result.code)));
    }

    /**
     * 是否自动上报厂商 token
     *
     * @return 返回 false，可根据需求上报
     */
    @Override
    protected boolean shouldReportChannelToken() {
        return super.shouldReportChannelToken();
    }

}
```

如需绑定上报，可重写 `shouldReportChannelToken` 方法并返回 false，并在确保收到两个 token 后调用：

```
MPPush.report(context, token , channel.value(), channelToken);
```

自定义 NotificationChannel

如需自定义自建通道的 `NotificationChannel` 的名称和说明，可在 `AndroidManifest.xml` 中添加：

```
<meta-data
    android:name="mpaas.notification.channel.default.name"
    android:value="名称" />
<meta-data
    android:name="mpaas.notification.channel.default.description"
    android:value="说明" />
```

调整推送通道优先级顺序

自基线 10.2.3.43 起，支持在部分特定设备上调整相关厂商通道优先级，如需使用该功能，请在工程的 `assets` 目录下创建 `mpaas_push_config.properties` 文件，并按照下文按需开启。

在华为/荣耀设备上优先使用荣耀通道

如需在华为或者荣耀设备上优先使用荣耀推送通道，请在文件 `mpaas_push_config.properties` 中添加如下代码：

```
// 将在华为/荣耀设备上优先使用荣耀通道
isHonorBeforeHms=true
```

在具备 FCM 推送能力的设备上优先使用设备厂商通道

如需在具备 FCM 推送能力的设备上优先使用设备厂商的通道，请在文件 `mpaas_push_config.properties` 中添加如下代码：

```
// 将在具备 FCM 推送能力的设备上优先使用设备厂商的通道
isFcmEnd=true
```

4.2. 接入 iOS

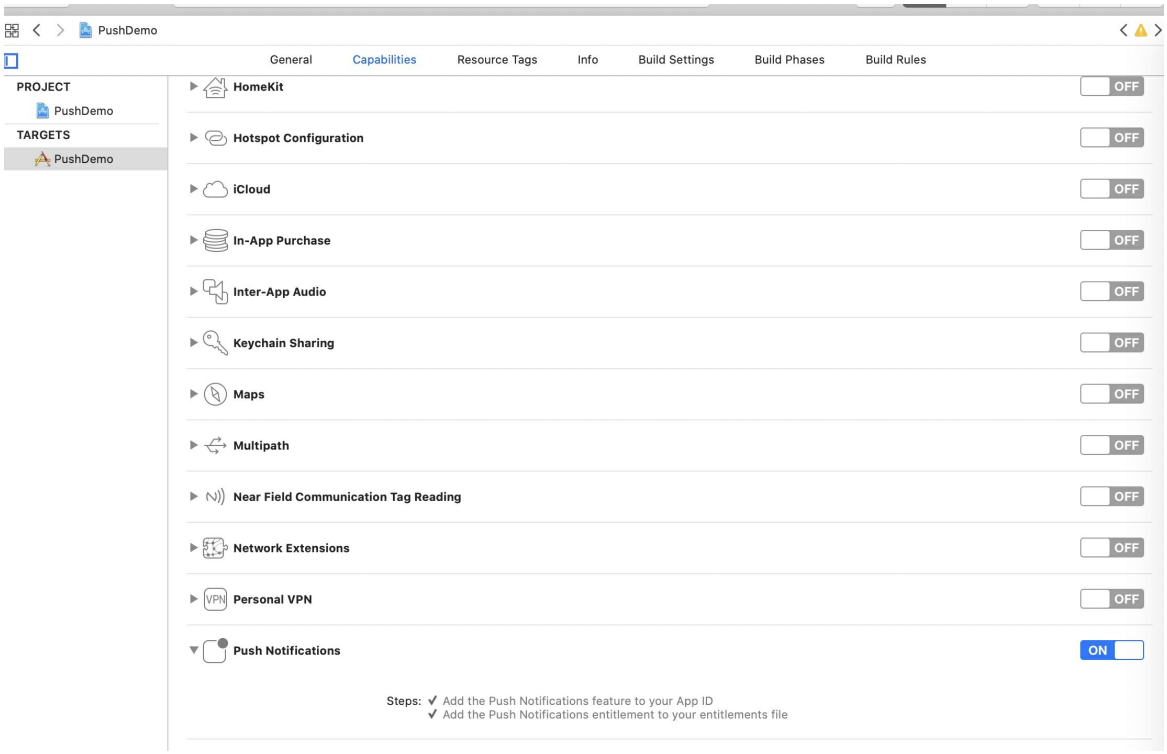
本文将向您详细介绍将消息推送服务接入 iOS 客户端的接入流程。

操作步骤

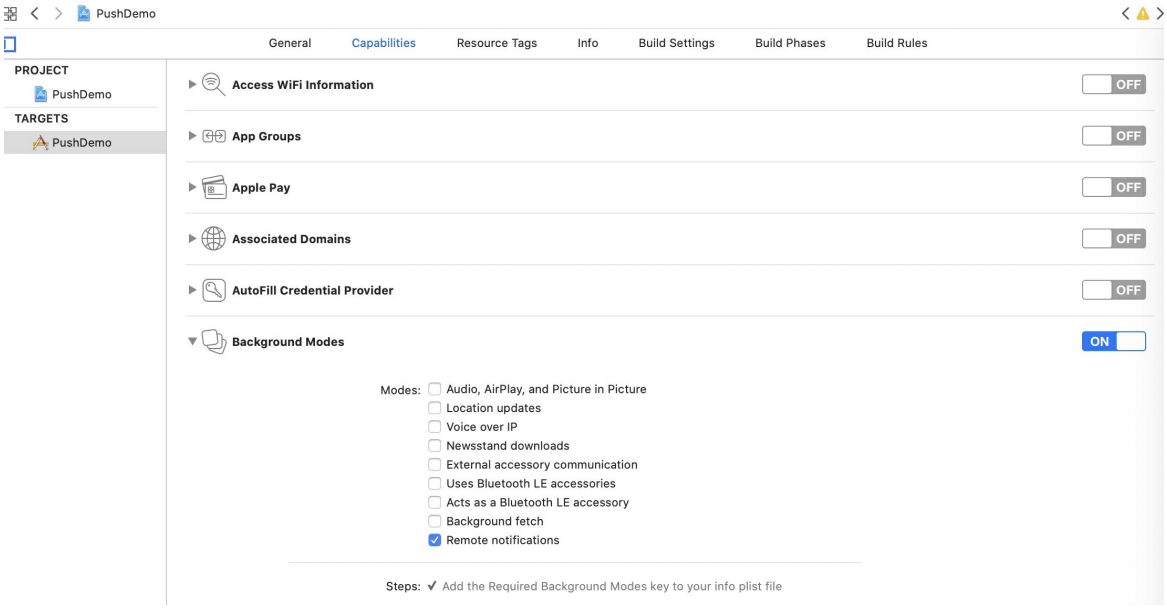
要使用消息推送服务，您需要完成以下接入步骤：

1. 参考 基于已有工程且使用 CocoaPods 接入基于已有工程且使用 CocoaPods 接入 文档，添加消息推送 SDK 完成接入。
2. 配置工程。需要在工程的 **TARGETS** 设置中开启以下两项配置：

- **Capabilities > Push Notifications**



- **Capabilities > Background Modes > Remote notifications**



3. 使用 SDK。

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

i. 注册 deviceToken（非必需）。

消息推送 SDK 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 set
tings) {

                [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 settingsF
orTypes:(UIUserNotificationTypeBadge
|UIUserNotificationTypeSound|UIUserNotificationTypeAlert) categories:nil];
        [[UIApplication sharedApplication]
registerUserNotificationSettings:settings];
        [[UIApplication sharedApplication] registerForRemoteNotifications];
    }
}
```

ii. 获取 deviceToken 并绑定 userId。

mPaaS 提供的消息推送 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) {
    }];

}
```

消息推送 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。

iii. 接收推送的消息。

客户端收到推送的消息后，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:(N
SDictionary *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult re
sult))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收
  - (void)userNotificationCenter:(UNUserNotificationCenter *)center willPresentNot
ification:(UNNotification *)notification withCompletionHandler:(void (^)(UNNotific
ationPresentationOptions options))completionHandler
  {
      NSDictionary *userInfo = notification.request.content.userInfo;

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

      } else {
          //应用处于前台时的本地推送接收

      }
      completionHandler(UNNotificationPresentationOptionNone);
  }

  //应用处于后台或者活冷启动时远程推送接收
  - (void)userNotificationCenter:(UNUserNotificationCenter *)center didReceiveNoti
ficationResponse:(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收

      } else {
          //应用处于前台时的本地推送接收

      }
      completionHandler();

  }
```

iv. **统计消息的打开率。**

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

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

4. 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 配置 iOS 推送通道。

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。

代码示例

点击此处 下载示例代码包。

相关链接

- 创建消息
- 配置服务端

Live Activity 消息推送

iOS 在 16.1 版本中推出了一个新功能：Live Activity（实时活动）。该功能可以将实时活动展示在锁屏界面上，帮助用户从锁定的屏幕实时获知各种活动的进展。在主工程中，可以使用 ActivityKit 框架来开启、更新、结束实时活动。其中，更新和结束实时活动还可以使用远程推送来实现。在 widget extension 中，可以使用 SwiftUI 和 WidgetKit 来创建 Live Activity 的界面。其中，Live Activity 远程推送更新功能，不支持 `.p12` 证书，因此需要用户配置 `.p8` 证书。

同一个项目中可以同时开启多个 Live Activity，不同的 Live Activity，其 token 是不同的。

Live Activity 苹果官方文档

- 实时活动
- 通过实时活动显示实时数据
- 使用 ActivityKit 推送通知启动和更新实时活动

Live Activity 使用限制

- 只能在 iOS 16.1 版本以上的系统上运行。
- 只支持 iPhone 设备，不支持 iPadOS，macOS，tvOS，watchOS。
- 单个 Live Activity 最多可运行 8 个小时，超过 8 小时系统会自动停止 Live Activity 的运行，但是不会立即从屏幕上被移除。
- 停止运行超过 4 小时后系统会自动将其从屏幕上移除。
- 资源文件尺寸需符合要求，详情请参见 苹果开发者文档。
- 推送的内容不能超过 4KB。

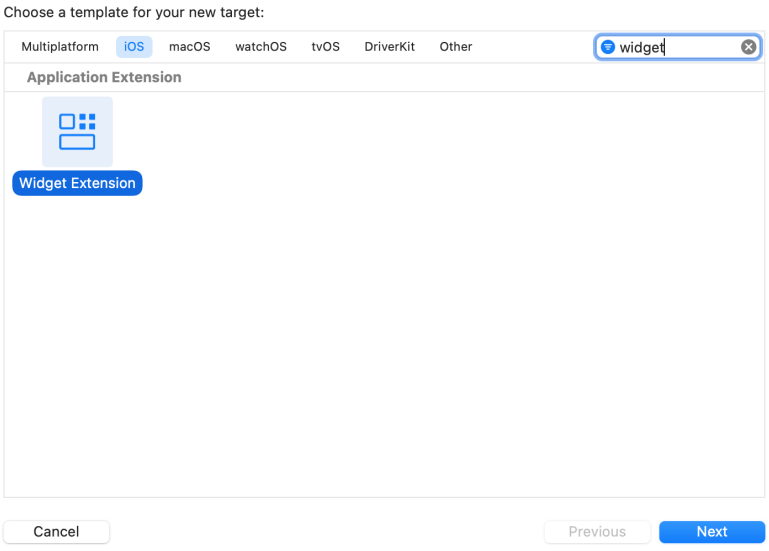
接入客户端

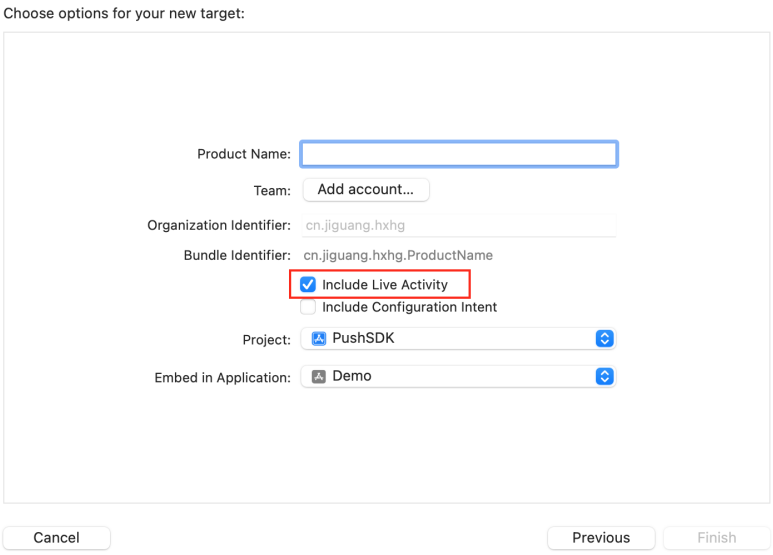
配置工程支持 Live Activity

1. 在主工程的 `Info.plist` 文件中添加一个键值对，key 为 `NSSupportsLiveActivities`，值为 `YES`。

Key	Type	Value
Information Property List	Dictionary	(26 items)
NSSupportsLiveActivities	Boolean	YES

2. 新建 Widget Extension，如果项目中已有，可跳过此步骤。





代码实现

1. 创建 model。
   - 在主工程代码里新建一个 swift 文件，在其中定义 `ActivityAttributes` 以及 `Activity.ContentState`。以下代码为示例代码，请按照实际业务编写。

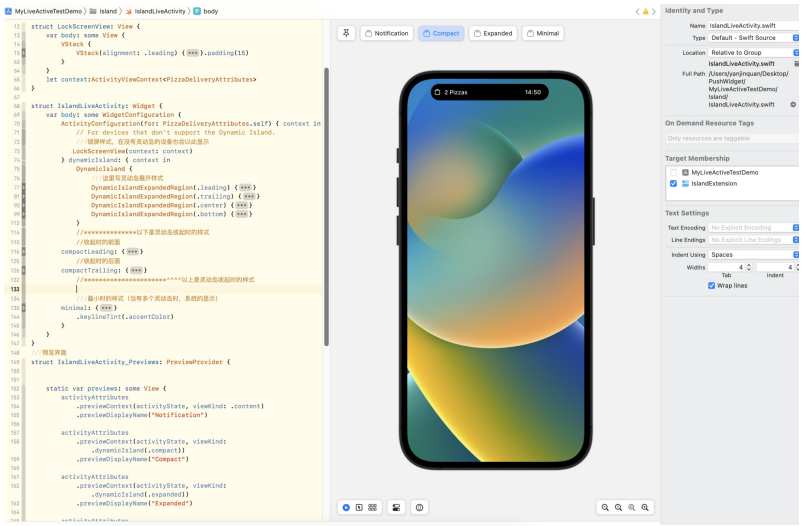
```
import SwiftUI
import ActivityKit

struct PizzaDeliveryAttributes: ActivityAttributes {
    public typealias PizzaDeliveryStatus = ContentState

    public struct ContentState: Codable, Hashable {
        var driverName: String
        var estimatedDeliveryTime: ClosedRange<Date>

        init(driverName: String, estimatedDeliveryTime: ClosedRange<Date>) {
            self.driverName = driverName
            self.estimatedDeliveryTime = estimatedDeliveryTime
        }
        init(from decoder: Decoder) throws {
            let container:
KeyedDecodingContainer<PizzaDeliveryAttributes.ContentState.CodingKeys> = try decoder.
container(keyedBy: PizzaDeliveryAttributes.ContentState.CodingKeys.self)
            self.driverName = try container.decode(String.self, forKey:
PizzaDeliveryAttributes.ContentState.CodingKeys.driverName)
            if let deliveryTime = try? container.decode(TimeInterval.self, forKey: Piz
zaDeliveryAttributes.ContentState.CodingKeys.estimatedDeliveryTime) {
                self.estimatedDeliveryTime =
Date()...Date().addingTimeInterval(deliveryTime * 60)
            } else if let deliveryTime = try? container.decode(String.self, forKey: Pi
zzaDeliveryAttributes.ContentState.CodingKeys.estimatedDeliveryTime) {
                self.estimatedDeliveryTime =
Date()...Date().addingTimeInterval(TimeInterval.init(deliveryTime)! * 60)
            } else {
                self.estimatedDeliveryTime = try
container.decode(ClosedRange<Date>.self, forKey:
PizzaDeliveryAttributes.ContentState.CodingKeys.estimatedDeliveryTime)
            }
        }
    }

    var numberOfPizzas: Int
    var totalAmount: String
}
```

- 主工程 target 和 Activity 都要选上。
- 收到的推送消息由系统处理，开发者不能拦截。
- `ContentState` 中为可以动态更新的数据，推送 Live Activity 通知时，动态更新的参数名和类型要和 `ContentState` 里配置的对应上。
- 如果有些数据需要经过加工，需要重写 `ActivityAttributes.ContentState` 的 `decoder` 方法。

2. 创建界面。

   在 Widget Extension 中创建实时活动的界面。创建 Widget 并返回一个 `ActivityConfiguration`。

   具体 UI 请按照自己的业务编写。



3. 使用 WidgetBundle。

   如果目标 App 既支持小组件也支持实时活动，请使用 WidgetBundle。

```
import WidgetKit
import SwiftUI

@main
structIslandBundle: WidgetBundle {
varbody: someWidget {
Island()
IslandLiveActivity()
}
}
```

4. 开启实时活动。

```
func startDeliveryPizza() {
    let pizzaDeliveryAttributes = PizzaDeliveryAttributes(numberOfPizzas: 1, totalAmou
nt:"$99")
    let initialContentState = PizzaDeliveryAttributes.PizzaDeliveryStatus(driverName:
"TIM", estimatedDeliveryTime: Date()...Date().addingTimeInterval(15 * 60))
    do {
        let deliveryActivity = try Activity<PizzaDeliveryAttributes>.request(
            attributes: pizzaDeliveryAttributes,
            contentState: initialContentState,
            pushType: .token)
    } catch (let error) {
        print("Error requesting pizza delivery Live Activity \
(error.localizedDescription)")
    }
}
```

5. 提交 Token。

   开启实时活动成功后，通过 `pushTokenUpdates` 方法拿到系统返回的 Live Activity 的推送 Token。调用 PushService 的 `liveActivityBindWithActivityId:pushToken:filter:completion:` 方法上报。

在上报 Token 的同时，需要将该实时活动的标识一起上报。实时活动推送时需要用到该标识，服务器根据该标识确认推送目标。该实时活动的标识请自定义，不同 Live Activity，其 id 不同（如果唯一会导致推送出现问题），同一个 Live Activity，在 Token 更新时不要更换 id。

> **说明**
>
> ActivityKit 为 swift 语言框架，且不支持直接 OC 调用，使用该框架 API 的时候，请在 swift 文件里面调用。由于 MPPushSDK 是 OC 语言，涉及到 swift 调用 OC，需要创建桥接文件。并在桥接文件里导入：`#import <MPPushSDK/MPPushSDK.h>`。

```
let liveactivityId = UserDefaults.standard.string(forKey: "pushTokenUpdates_id") ?? "d
efloutliveactivityId"
Task {
    for await tokenData in deliveryActivity.pushTokenUpdates {
        let newToken = tokenData.map { String(format: "%02x", $0) }.joined()
        PushService.shared().liveActivityBind(withActivityId: liveactivityId,
pushToken: newToken, filter: .call) { excpt in
            guard let excpt = excpt else {
                ///上报成功
                return
            }
            if "callRepeat" == excpt.reason {
                ///重复调用，请忽略
                print("pushTokenUpdates_id——重复调用")
            } else {
                ///上报失败
            }
        }
    }
}
```

上报成功后，则可以使用实时活动的标识推送更新。

> **说明**
>
> 由于 iPhone 的 `pushTokenUpdates` 会同时被调用两次，即在多个 Live Activity 的场景中，新创建 Live Activity 时之前的 `LiveActivity pushTokenUpdates` 又会被重新唤醒一次，所以 SDK 提供了过滤功能，并由参数 filter 控制：
>
> - filter 为 `MPPushServiceLiveActivityFilterAbandon` 时，SDK 会自动直接抛弃重复的调用，不给回调。
> - filter 为 `MPPushServiceLiveActivityFilterCall` 时，SDK 会自动过滤掉本次请求，给失败回调（callRepeat），此时 `error.reason` 为 `@"callRepeat"`，请忽略。
> - filter 为 `MPPushServiceLiveActivityFilterReRefuse` 时，SDK 内部不做过滤。
> - 重复地调用相同的 activityId，相同的 pushToken 时，如果上报失败，客户端重新上报不会被认为是相同的调用。

下面是 `MPPushServiceLiveActivityFilterType` 的定义：

```
typedef NS_ENUM(NSInteger, MPPushServiceLiveActivityFilterType){
    MPPushServiceLiveActivityFilterAbandon,//直接抛弃，不给回调
    MPPushServiceLiveActivityFilterCall,//过滤掉本次请求，给失败回调（callRepeat）
    MPPushServiceLiveActivityFilterRefuse//不做过滤
};
```

4.3. 接入 HarmonyOS NEXT（beta）

4.3.1. 添加 SDK

本文介绍如何将移动推送组件接入到 HarmonyOS NEXT 客户端。您可以基于已有工程使用 ohpmrc 方式接入推送 SDK 到客户端。

前置条件

添加移动推送 SDK 前，请您确保已经将工程接入到 mPaaS。更多信息请参见 基于已有工程使用 ohpmrc 接入。

添加 SDK

在 `oh-package.json5` 中配置所需依赖：

```
{
  "license": "",
  "devDependencies": {},
  "author": "",
  "name": "entry",
  "description": "Please describe the basic information.",
  "main": "",
  "version": "1.0.0",
  "dependencies": {
    "@mpaas/push": "0.0.2"
  }
}
```

配置权限

需要 App 具备以下三种权限。

```
{
 "name" : "ohos.permission.GET_NETWORK_INFO",
},
{
 "name" : "ohos.permission.SET_NETWORK_INFO",
},
{
 "name" : "ohos.permission.INTERNET",
}
```

4.3.2. 使用 SDK

初始化与接入 SDK

前置条件

- 已在华为的应用平台上开通 App 的推送服务，详情请参考 华为的 Push 服务。

组件接入

1. 需在 `Ability` 的 `onCreate` 方法中设置调用。

   代码如下：

   ```
   import { MpaasPushServiceImpl, MPPush, CallResp } from '@mpaas/push';

   let tokenGet: CallResp
   try {
    tokenGet = await MPPush.init()
    console.info("CallResp="+tokenGet)
    if (tokenGet.success) {
     this.tokenVal = tokenGet.msg
    } else {
     console.info("MPPush.init failed=" + tokenGet.msg)
    }
   } catch (e) {
    console.info("MPPush.init err=" + e.message)
   }
   ```

   该方法会初始化 `Push` 的服务组件，同时会获取一个鸿蒙推送的 `token`，然后将其上报到服务器。

   - 如果成功，则该方法会返回 `token` 值。
   - 如果失败，则会返回空字符串，并且打印日志。

2. 编写一个承接 MPS 组件的 `Ability`，继承 `MpaasNcAbility`。

   代码如下：

```
import { MpaasNcAbility } from '@mpaas/push'

export default class MpaasBridgeMsgAbility extends MpaasNcAbility {
 static tag: string = "MpaasBridgeMsgAbility"
 onCreate(want: Want, launchParam: AbilityConstant.LaunchParam): void {
 hilog.info(0x0000, MpaasBridgeMsgAbility.tag, "start MpaasBridgeMsgAbility onCreate")
 super.onCreate(want, launchParam)
 }

  onForeground() {
    console.log('MpaasBridgeMsgAbility onBackground');
      try {
        // 销毁掉中间过渡的 Ability
        this.context.terminateSelf((err: BusinessError) => {
          if (err.code) {
            // 处理业务逻辑错误
            console.error(`terminateSelf failed, code is ${err.code}, message is
${err.message}`);
            return;
          }
          // 执行正常业务
          console.info('terminateSelf succeed');
        });
      } catch (err) {
        // 捕获同步的参数错误
        let code = (err as BusinessError).code;
        let message = (err as BusinessError).message;
        console.error(`terminateSelf failed, code is ${code}, message is ${message}`);
      }
  }
}
```

3. 在 `module.json5` 文件中，描述 `MpaasBridgeMsgAbility` 。

   代码如下：

```
{
 "name": "MpaasBridgeMsgAbility",
 "srcEntry": "./ets/pushability/MpaasBridgeMsgAbility.ets",
 "description": "$string:EntryAbility_desc",
 "icon": "$media:icon",
 "label": "$string:BridgeAbility_label",
 "startWindowIcon": "$media:startIcon",
 "startWindowBackground": "$color:start_window_background",
 "exported": true,
 "removeMissionAfterTerminate": true,
 "skills": [
 {
 "actions": ["com.mpaas.harmony.push"]
 }
 ]
}
```

> **重要**

- `actions` 必须和代码中的描述保持一致。
- `removeMissionAfterTerminate` 字段必须设置为 `true`，否则 `MpaasBridgeMsgAbility` 不会被彻底销毁，从而造成应用中有一个多余且无效的 `UIAbility`。

使用 SDK

绑定/解绑 token

用户调用绑定接口，将 App 的用户 id 和 token 发送到后端即可进行绑定，解绑即为取消绑定。

- 绑定接口如下：

```
import { MpaasPushServiceImpl, MPPush, CallResp } from '@mpaas/push';

// 第一个入参： phoneNumber，如果没有电话号码，填入 ""
// 第二个入参： token 就是 init 方法返回的 token 值，必填
// 第三个入参： userId，用户 ID，如果不填，则从框架层获取
let res: CallResp = await MpaasPushServiceImpl.getInstance().bind("12345678900", this.
tokenVal, "mpaas_user_id")
console.log("bind res="+res.msg)
if (res.success) {
 this.bindState = "bind state: " + "binded"
}
```

返回值 `res.success` 表示绑定是否成功。

- 解绑接口如下：

```
import { MpaasPushServiceImpl, MPPush, CallResp } from '@mpaas/push';

// 第一个入参： token 就是 init 方法返回的 token 值，必填
// 第二个入参： userId，用户 ID，如果不填，则从框架层获取
let res: CallResp = await MpaasPushServiceImpl.getInstance().unbind(this.tokenVal, "mp
aas_user_id")
console.log("unbind res="+res.msg)
if (res.success) {
 this.bindState = "bind state: " + "unbind"
}
```

返回值 `res.success` 表示解绑是否成功。

配置跳转消息 Ability

在客户端的 `module.json5` 文件中配置目标应用内的 `Ability`，例如下面的 `PushLandingAbility`：

```
{
 "name": "PushLandingAbility",
 "srcEntry": "./ets/pushability/PushLandingAbility.ets",
 "description": "$string:EntryAbility_desc",
 "icon": "$media:icon",
 "label": "$string:LandingAbility_label",
 "startWindowIcon": "$media:startIcon",
 "startWindowBackground": "$color:start_window_background",
 "exported": true,
 "skills": [
 {
 "actions": [""],
 "uris": [
 {
 "scheme": "jump",
 "host": "com.mpaas.harmony.push",
 "path": "landing"
 }
 ]
 }
 ]
}
```

在控制台推送消息时，需要填入跳转的 URI 为 `jump://com.mpaas.harmony.push/landing` ，和上述代码中的 `uris` 对应。在控制台推送消息结束之后，可以点击客户端的通知栏，即可跳转至配置好的 `PushLandingAbility` 所对应的页面。用户也可以点击通知栏，打开消息中包含的 HTTP 链接地址，如果该消息的 URL 字段中 HTTP 链接合法，那么应用会打开本地默认的浏览器应用，同时跳转到 URL 对应的链接地址。

配置落地页面的 Ability 示例代码

```
import UIAbility from '@ohos.app.ability.UIAbility';
import Want from '@ohos.app.ability.Want';
import AbilityConstant from '@ohos.app.ability.AbilityConstant';
import window from '@ohos.window';
import hilog from '@ohos.hilog';
import { MpaasPushServiceImpl } from '@mpaas/push';
import pushService from '@hms.core.push.pushService';
import { BusinessError } from '@ohos.base';

export class PushLandingAbility extends UIAbility {
  private static TAG: string = "PushLandingAbility"

  private pushData: object = JSON.parse("{}");
  private pushKey: string = "";
  public  para: Record<string,string> = { 'msg_id': "default", 'msg_data': "default"};
  public storageData: LocalStorage = new LocalStorage(this.para);
  landingWindowStage: window.WindowStage | undefined = undefined;

  async onCreate(want: Want, launchParam: AbilityConstant.LaunchParam): Promise<void> {
    hilog.info(0x0000, PushLandingAbility.TAG, 'PushLandingAbility create. Data: %{publi
c}s', JSON.stringify(want.parameters) ?? '');
    let k: string = ""
    let v: object = JSON.parse("{}")
```

```
    if (want.parameters) {
      if (want.parameters["msg_id"]) {
        k = want.parameters["msg_id"] as string
        hilog.info(0x0000, PushLandingAbility.TAG, 'onNewWant k: %{public}s', k);
      }
      if (want.parameters["msg_data"]) {
        v = want.parameters["msg_data"]
        hilog.info(0x0000, PushLandingAbility.TAG, 'onNewWant k: %{public}s', v);
      }
    }

    this.storageData.set('msg_id', k)
    this.storageData.set('msg_data', JSON.stringify(v))
    hilog.info(0x0000, PushLandingAbility.TAG, 'onCreate push_msgkey from storage: %{pub
lic}s', this.storageData.get("msg_id"));
    hilog.info(0x0000, PushLandingAbility.TAG, 'onCreate push_msgdata from storage: %{pu
blic}s', JSON.stringify(this.storageData.get("msg_data")));
  }

  async onNewWant(want: Want, launchParam: AbilityConstant.LaunchParam): Promise<void> {
    let k: string = ""
    let v: object = JSON.parse("{}")
    if (want.parameters) {
      if (want.parameters["msg_id"]) {
        k = want.parameters["msg_id"] as string
        hilog.info(0x0000, PushLandingAbility.TAG, 'onNewWant k: %{public}s', k);
      }
      if (want.parameters["msg_data"]) {
        v = want.parameters["msg_data"]
        hilog.info(0x0000, PushLandingAbility.TAG, 'onNewWant k: %{public}s', v);
      }
    }
    this.storageData.set('msg_id', k)
    this.storageData.set('msg_data', JSON.stringify(v))

    hilog.info(0x0000, PushLandingAbility.TAG, 'onNewWant push_msgkey: %{public}s', this
.storageData.get("msg_id"));
    hilog.info(0x0000, PushLandingAbility.TAG, 'onNewWant push_msgdata: %{public}s', JSO
N.stringify(this.storageData.get("msg_data")));

  // 进入该阶段需要手动加载一下页面
  if (this.landingWindowStage != null) {
      await this.landingWindowStage.loadContent('pushpages/pushLandingPage',
this.storageData)
    }

  }

  onDestroy(): void {
    hilog.info(0x0000, PushLandingAbility.TAG, '%{public}s', 'Ability onDestroy');
  }

  async onWindowStageCreate(windowStage: window.WindowStage): Promise<void> {
    // Main window is created, set main page for this ability
    // 创建新实例并使用给定对象初始化
```

```
    this.landingWindowStage = windowStage
    hilog.info(0x0000, PushLandingAbility.TAG, 'this.pushKey= %
{public}s',this.storageData.get("msg_id"));
    hilog.info(0x0000, PushLandingAbility.TAG, 'this.pushData= %{public}s',
JSON.stringify(this.storageData.get("msg_data")));

    try {
      await windowStage.loadContent('pushpages/pushLandingPage', this.storageData)
      hilog.info(0x0000, PushLandingAbility.TAG, 'Succeeded in loading the content. Key:
%{public}s, Data: %{public}s',this.storageData.get("msg_id"),
JSON.stringify(this.storageData.get("msg_data")) ?? '');
    } catch (e) {
      console.log("err msg="+e.message)
      return
    }
  }

  onWindowStageDestroy(): void {
    hilog.info(0x0000, PushLandingAbility.TAG, '%{public}s', 'Ability
onWindowStageDestroy');
  }

  onForeground(): void {
    hilog.info(0x0000, PushLandingAbility.TAG, '%{public}s', 'Ability onForeground');
  }

  onBackground(): void {
    hilog.info(0x0000, PushLandingAbility.TAG, '%{public}s', 'Ability onBackground');
  }
}
```

配置透传消息 Ability

后端可以向客户端推送 App 的后台消息，而不展示通知栏。但开发者仍然可以通过特定的方法获取推送的数据。MPS 已经实现了不拉起应用子进程的后台消息。

1. 开发者需继承 `MpaasExtDefaultNcAbility` 类实现自己的 `Ability` ，如下所示：

```
import { MpaasExtDefaultNcAbility } from '@mpaas/push';

export default class MpaasBridgeExtMsgAbility extends MpaasExtDefaultNcAbility {

  async onCreate(): Promise<void> {
    // 收到eventId为9999的事件后执行回调函数
    emitter.on(innerEvent, (data) => {
      if (data.data != null) {
        console.log("onCreate receivedMsg="+JSON.stringify(data.data["receivedMsg"]))
      } else  {
        console.log("onCreate receivedMsg="+data.data)
      }

    });
    await super.onCreate()
  }

}
```

在 `onCreate()` 方法中执行 `super.onCreate()`，之后即可通过 `MpaasExtDefaultNcAbility.getReceivedMsg()` 获取后台推送的消息。

2. 需要在 `module.json5` 文件中配置 `MpaasBridgeExtMsgAbility`，如下所示：

```
{
 "name": "MpaasBridgeExtMsgAbility",
 "srcEntry": "./ets/pushability/MpaasBridgeExtDeMsgAbility.ets",
 "launchType": "singleton",
 "startWindowIcon": "$media:icon",
 "startWindowBackground": "$color:start_window_background",
 "skills": [
 {
 "actions": [
 "action.ohos.push.listener"
 ]
 }
 ]
}
```

其中，`skills` 里配置 `actions` 内容为 `action.ohos.push.listener`。

> **重要**
>
> 有且只能有一个 `ability` 定义该 `action`，若同时添加 `uris` 参数，则 `uris` 内容需为空。

角标清理

可以调用 MPS 提供的方法，实现在打开 App 的时候将应用的角标数字清理掉，方法如下所示：

```
BadgeUtil.clearBadge()
```

5.配置服务端

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

步骤一：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 客户端 API 或 服务端 API。

步骤二：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. 数据概览

概览页面从推送数、推送成功数、到达数、打开数和忽略数五个维度对消息推送情况进行统计分析，并支持通过平台、版本、推送渠道、推送类型等条件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

- 已基于 mPaaS 框架完成消息推送（MPS）SDK 接入。
- 使用分析中的数据都是根据客户端 SDK 埋点上报的日志统计得出，确保已完成客户端埋点。
  - 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，暂不支持统计到达和忽略数据。

查看推送数据

完成以下步骤，查看消息推送分析数据：

1. 登录 mPaaS 控制台，进入目标应用后，从左侧导航栏进入 **消息推送** > **概览** 页面。
2. 设置筛选条件，筛选统计数据。可根据需要选择平台、版本、推送通道、推送类型，或输入完整任务 ID 进行搜索。

   > **说明**
   > 使用任务 ID 搜索仅适用于批量推送方式下发的消息。批量推送任务 ID 可在消息推送控制台的 **批量消息记录** 页面中查看。

   - **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。
   - **推送通道**：可选 **所有推送通道**、**自建通道** 以及 **厂商通道**（例如小米、华为、苹果等）。当前存在相应通道的推送后，方提供对应可选项，例如当前不存在小米通道推送时，则可选项中将不包含 **厂商通道 – 小米**。
   - **推送类型**：可选 **所有推送类型**、**极简推送 - 非模板**、**极简推送 - 模板**、**批量推送 - 所有设备**，**批量推送 - 非所有设备**。当前存在相应类型的推送时，才会展示对应的可选项。例如当前不存在基于模板的简单推送，则没有 **极简推送 - 模板** 这个选项。
   - **时间范围**：最长可选时间跨度为 90 天。

核心指标概览

展示指定时间范围内的推送核心指标数据，包括推送数、推送成功数、到达数、打开数、忽略数。

指标	说明
推送数	指所选时间段内消息推送后端推出的消息总数，由后端统计。

推送成功数	指所选时间段内实际推送成功的消息数量，由后端统计。消息推送成功数的统计不考虑消息是否是在指定时间段内推送的。 • 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。
到达数	指所选时间段内实际到达客户端本地的消息数量。消息到达数的统计不考虑消息是否是在指定时间段内推送的。例如，8.1 ~ 8.7 日的到达数为 100，则表示这 7 天内有 100 条消息到达了用户手机客户端，其中可能包含 8.1 日之前推送的消息。对于不同的推送通道，到达数据的统计方式不同： • Android 自建通道推送：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 厂商通道推送：在消息推送至对应通道后，通过其后端服务返回推送结果数据的方式进行统计。
到达率	到达率 =（到达数/推送数）* 100%
打开数	指在客户端本地被实际打开的消息条数，由客户端统计。消息打开数的统计不考虑消息是否是在指定时间段内到达的。例如，8.1 ~ 8.7 日的打开数为 88，则表示这 7 天内共有 88 条消息被用户打开，其中可能包含 8.1 日之前就已经到达客户端但用户一直未打开查阅的消息。
打开率	打开率 =（打开数/到达数）* 100%
忽略数	指在客户端本地被手动忽略的消息条数，由客户端统计。消息忽略数的统计不考虑消息是否是在指定时间段内到达的。例如，8.1 ~ 8.7 日的忽略数为 66，则表示这 7 天内共有 66 条消息被用户手动忽略，其中可能包含 8.1 日之前就已经到达客户端的消息。
忽略率	忽略率 =（忽略数/到达数）* 100%

推送数据趋势

以趋势折线图的形式展示指定时间段内的消息推送统计结果。点击位于图表下方的图例，可隐藏或显示对应指标的曲线。

在折线图左上方，选择 **按数量查询** 或 **按比率查询** 选项卡，分别查看推送指标的数量或比率变化趋势。

- **按数量查询**：展示推送数、推送成功数、到达数、打开数和忽略数的数据变化趋势。
- **按比率查询**：展示到达率、打开率和忽略率的数据变化趋势。

在折线图右上方，选择 **分钟**、**小时** 或 **日** 选项，可分别按分钟、小时和日展示推送数据变化趋势。

- **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。

推送详细数据

以列表形式展示选定时间段内每小时或每天的推送详细数据，随核心指标曲线变化而变化。

- 列表中的 **时间** 继承自核心指标曲线横轴的时间。
- 列表中包含 **推送数**、**推送成功数**、**到达数（到达率）**、**打开数（打开率）** 和 **忽略数（忽略率）** 五个指标。

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。

6.2. 消息管理

6.2.1. 创建消息 - 极简推送

> **重要**
>
> 即日起，mPaaS 消息推送将使用新版控制台。在新控制台界面上，创建消息推送窗口提供的推送方式由之前的极简推送、模板推送、批量推送、群发推送四种整合优化为极简推送、批量推送两种。重构后，现极简推送方式覆盖了原极简推送和模板推送功能；现批量推送方式覆盖了原批量推送和群发推送功能。

极简推送指针对单个推送目标推送一条消息。采用该推送方式进行消息推送时，您既可以自定义消息内容，也可以使用预先创建的消息模板。

自定义消息内容适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，Android 推送 SDK 接入的正确性等。消息模板适用于对多个目标进行多次推送的场景，即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。

> **说明**
>
> - 消息一旦创建成功即进行推送，您将无法删除或修改。
> - 由于需要人工在页面上进行操作，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。

下面主要介绍如何通过控制台创建极简推送类型的消息。

前置条件

- 对 iOS 设备进行消息推送前，确保已完成消息推送 iOS SDK 接入，并通过控制台的 **通道配置** 页面配置好苹果设备的推送证书，具体操作参见 配置 iOS 推送通道。
- 使用 Android 厂商通道进行消息推送前，确保已完成消息推送 Android SDK 接入，接入相应的厂商通道，并通过控制台的 **通道配置** 页面完成相应的推送通道配置，具体操作参见 配置 Android 推送通道。

操作步骤

1. 登录 mPaaS 控制台，选择目标应用后，从左侧导航栏进入 **消息推送** > **消息管理** 页面。
2. 单击 **创建消息推送任务** 按钮，在弹出的消息推送任务创建窗口中，选择 **极简推送** 标签。
3. 在极简推送标签页上，配置推送消息的基础信息。各配置项说明如下：

参数	是否必填	说明

消息类型：是否静默	是	是否展示消息： ◦ **是**：表示静默消息，即用户对消息无感知，在目标设备上不以任何形式展示的消息。 ◦ **否**：指在通知栏展示消息。 对于 Android 推送平台，需要根据不同的推送通道，执行不同后续操作： ◦ **自建通道**：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，根据需求控制消息的展示。 ◦ **厂商通道**：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，无需执行其他操作。
消息内容创建方式	是	支持两种创建方式： ◦ **新建**：自定义消息内容，包括消息标题、正文以及展示样式。 ◦ **使用模板**：使用事先创建好的推送模板。
推送模板	是	选择消息模板，可选当前应用的 **消息模板** 页面上列表中的所有模板。 **说明** 仅当消息内容创建方式为 **使用模板** 时需要配置。
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。
推送维度	是	选择消息下发模式，可选择： ◦ **用户维度**：基于用户标识推送消息。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 客户端 API。 ◦ **Android 维度**：基于 Android 设备标识推送消息。 ◦ **iOS 维度**：基于 iOS 设备标识推送消息。
用户/设备标识	是	根据所选的推送维度，填写相应的用户标识或设备标识。 ◦ 当推送维度为 Android 维度时，填写 Ad-token。 ◦ 当推送维度为 iOS 维度时，填写 Device Token。 ◦ 当推送维度为用户维度时，填写用户标识，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。
安卓消息通道推送优先级	是	仅针对 Android 推送平台，可选择： ◦ **优先厂商通道**：优先使用厂商通道推送消息。对于已接入的厂商通道，消息走对应的厂商通道服务；对于未接入的厂商通道，消息走 MPS 自建通道。 ◦ **MPS 通道**：使用 MPS 自建通道推送消息。 对于 Android 推送平台，本参数为 MPS 通道和厂商通道推送的选择入口。 对于 iOS 推送平台，您无需配置本参数（iOS 推送为厂商通道推送）

展示样式	是	消息在客户端的展示样式，支持默认（小文本）、大文本、图文三种样式。 ◦ **默认**：该样式展示的内容包含推送标题和文本，适用于内容简洁明了的消息。消息文本长度建议不超过 100 个字符（包括自定义参数和符号）。 ◦ **大文本**：该样式展示的内容包含推送标题和文本，适用于文字内容较多的消息（例如资讯类、新闻类消息），让用户无需打开应用，也能快速获取信息。消息文本长度建议不超过 256 个字符（包括自定义参数和符号）。 ◦ **图文**：该样式支持消息中带图标和大图，适用于除普通文本之外更丰富的内容。出于展示效果考虑，建议消息文本长度不超过两行。 **说明** 仅当消息内容创建方式为 **新建** 时需要配置。
推送标题	是	填写消息的标题。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。 **说明** 仅当消息内容创建方式为 **新建** 时需要配置。
推送内容	是	填写消息的文本内容。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。 **说明** 仅当消息内容创建方式为 **新建** 时需要配置。
消息图标	否	展示在通知栏消息内容右侧的消息图标，支持 jpg、jpeg、png 格式的图片。此处填写公网可访问的图标物料 URL 地址。若各厂商通道未上传相应的物料，仅上传默认物料 URL时，系统会自动拉取默认物料去支持各厂商通道的图标显示，但因为各厂商通道对物料要求不尽相同，为避免效果不好，建议按各通道要求分别上传物料。 ◦ **默认图标**：建议尺寸为 140 * 140px，大小 50 KB 以内 ◦ **OPPO 通道图标**：建议尺寸为 140 * 140px，大小 50 KB 以内 ◦ **小米通道图标**：建议尺寸为 120 * 120px，大小 50 KB 以内 ◦ **华为通道图标**：建议尺寸为 40 * 40dp，大小 512 KB 以内 ◦ **FCM 图标**：无明确配置要求，系统自动拉取默认图标适配。 **说明** 仅当消息内容创建方式为 **新建** 且展示样式为 **图文** 时需要配置。

消息大图	否	展示在通知栏消息内容下方的图片。此处填写公网可访问的大图物料 URL 地址。若各厂商通道未上传相应的物料，仅上传默认物料 URL 时，系统会自动拉取默认物料去支持各厂商通道的大图显示，但因为各厂商通道对物料要求不尽相同，为避免效果不好，建议按各通道要求分别上传物料。 ◦ **默认大图**：建议尺寸为 876 * 324px，大小 1 MB 以内，支持 jpg、jpeg、png 格式 ◦ **OPPO 通道大图**：建议尺寸为 876 * 324px，大小 1 MB 以内，支持 jpg、jpeg、png 格式 ◦ **小米通道大图**：建议尺寸为 876 * 324px，大小 1 MB 以内，支持 jpg、jpeg、png 格式 ◦ **iOS 大图**：由用户自定义图片，无尺寸限制 ◦ **FCM 大图**：无明确配置要求，系统自动拉取默认图标适配。 **说明** 仅当消息内容创建方式为 **新建** 且展示样式为 **图文** 时需要配置。
推送时间	是	选择何时推送消息： ◦ **立即**：推送时间为当前时间，即消息推送任务创建完成后立即推送。 ◦ **定时**：在指定时间推送消息。例如，指定在 6.19 日早上 8:00 推送消息。 ◦ **循环**：在指定时间范围内重复循环推送消息，例如指定在 6.1 ~ 9.30 期间，每周五早上 8:00 推送消息。

对话框右侧区域为 **推送预览** 区域。单击 **通知**、**苹果消息体**、**安卓消息体** 和 **鸿蒙消息体**，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。

4. （选填）根据实际需要，配置高级信息。在高级信息配置区域，完成以下配置：
   - **单击后跳转**：选择在手机上单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，根据需求实现后续操作。
     - **客户端自定义**：单击消息后，页面跳转至原生页面。
     - **网页**：单击消息后，页面跳转至网页。
   - **跳转地址**：在手机上单击消息后访问的页面。根据 **单击后跳转** 选项，填写具体的页面地址：
     - 如果是 **客户端自定义**，填写需要访问的原生页面地址（Android：ActivityName；iOS：VCName）。
     - 如果是 **网页**：填写需要访问的网页地址。
   - **自定义消息 ID**：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。

     说明
     仅当选择消息类型为静默时需要配置自定义 ID。
   - **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。如不设置，则默认有效期为 180 秒。

> **说明**
> 消息有效期不能短于 180 秒，也不得超过 72 小时。

- **扩展参数**：打开 **扩展参数** 开关，单击 **增加参数** 按钮，在 key/value 配置区域中配置 key/value 后，在页面任意区域处单击鼠标左键，完成配置。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

    这类扩展参数被系统占用，参数值不可修改。参数包括 notifyType、action、silent、pushType、templateCode、channel、taskId。
  - 系统具有一定意义的扩展参数

    这类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参数值。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，参数值配置为铃声的路径。该参数仅对小米和苹果手机有效。
badge	应用图标角标，参数值配置为具体数值。该参数会跟随消息体到达客户端。 ■ 对于 Android 手机，您需要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所配置的参数值。
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。参数值配置为 `1` 。
badge_add_num	华为通道推送角标增加数。
badge_class	华为通道桌面图标对应的应用入口 Activity 类。
big_text	大文本样式，参数值固定为 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。

5. 单击 **提交** 按钮完成创建。创建的消息将展示在极简消息记录列表中。

   除了通过控制台推送消息外，还支持通过调用 API 推送消息。具体操作，参见 服务端 API 说明。

相关操作

- 创建消息 - 批量推送
- 管理极简推送消息

- 管理定时推送任务

6.2.2. 创建消息 - 批量推送

> **重要**
>
> 即日起，mPaaS 消息推送将使用新版控制台。在新控制台界面上，创建消息推送窗口提供的推送方式由之前的极简推送、模板推送、批量推送、群发推送四种整合优化为极简推送、批量推送两种。重构后，现极简推送方式覆盖了原极简推送和模板推送功能；现批量推送方式覆盖了原批量推送和群发推送功能。

批量推送指对大量目标进行消息推送，通常用来支持一些运营需求。

批量推送分为以下两种类型：

- 全网推送：对全网 Android 或 iOS 设备推送相同的模板消息，仅支持按设备维度推送。对 Android 设备进行群发消息时，所有在消息有效期内建连的 Android 设备都将收到消息；对 iOS 设备进行群发消息时，所有在消息有效期内处于绑定状态的 iOS 设备都将收到消息。
- 非全网推送：对指定人群推送相同的模板消息。支持手动上传人群、自定义推送人群或直接调用移动分析人群做为推送人群。

> **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。

下面主要介绍如何通过控制台创建批量推送消息。

前置条件

- 对 iOS 设备进行消息推送前，确保已完成消息推送 iOS SDK 接入，并通过控制台的 **通道配置** 页面配置好苹果设备的推送证书，具体操作参见 配置 iOS 推送通道。
- 使用 Android 厂商通道进行消息推送前，确保已完成消息推送 Android SDK 接入，接入相应的厂商通道，并通过控制台的 **通道配置** 页面完成相应的推送通道配置，具体操作参见 配置 Android 推送通道。
- 创建批量推送消息之前，需要先创建好模板，操作参见 创建模板。
- 创建批量推送消息时，若选择调用移动分析人群做为目标推送人群，则需要事先创建好移动分析人群，具体操作参见 创建用户群组。若选择用户标签人群做为目标推送人群，则需要事先创建用户标签人群，具体操作参见 创建用户标签。

操作步骤

1. 登录 mPaaS 控制台，选择目标应用后，从左侧导航栏进入 **消息推送** > **消息管理** 页面。
2. 单击 **创建消息推送任务** 按钮，在弹出的消息推送任务创建窗口中，选择 **批量推送** 标签。
3. 在批量推送标签页上，配置推送消息的基础信息。各配置项说明如下：

参数	是否必填	说明

消息类型：是否静默	是	是否展示消息： ◦ 是：表示静默消息，即用户对消息无感知，在目标设备上不以任何形式展示的消息。 ◦ 否：指在通知栏展示消息。 对于 Android 推送平台，需要根据不同的推送通道，执行不同后续操作： ◦ 自建通道：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，控制消息的展示。 ◦ 厂商通道：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，无需执行其他操作。
批量推送方式	是	选择消息下发模式，可选择： ◦ **用户维度推送**：基于用户标识推送消息。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 客户端 API。 ◦ **设备维度推送**：基于设备标识推送消息。
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。 ◦ **Android**：提供安卓厂商通道和 MPS 自建通道，对全网（在消息有效期内）在线的或者指定的 Android 设备推送消息，对每个设备仅推送一次，不重复推送。 ◦ **iOS**：使用厂商通道，对全网或指定的 iOS 客户端用户推送消息，对每个用户仅推送一次，不重复推送。

推送目标	是	◦ 当为用户推送维度时，可选择： ▪ **手动上传人群**：手动上传推送目标文件，文件内需包含推送目标 ID 以及针对所选模板对各推送目标的个性化配置。文件内一条数据代表一条消息，每条消息使用业务方消息 ID 进行标识。文件格式要求如下： ▪ 每条数据格式为 `用户 ID,业务方消息 ID,占位符1= XXX;占位符2=XXX……`，其中业务方消息 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，单击图标，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**：调用移动分析人群，对所选移动分析人群推送相同消息。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。当所选推送模板中包含占位符时，移动分析人群不可选。 ▪ **用户标签**：根据用户标签选择人群。选择已打上标签的人群之前，需要先 创建用户标签。 ◦ 当为设备推送维度时，可选择： ▪ **所有设备**：对所选平台的所有设备进行推送。 ▪ **批量账号**：手动上传推送目标文件，文件内需包含推送目标 ID 以及针对所选模板对各推送目标的个性化配置。文件内一条数据代表一条消息，每条消息使用业务方消息 ID 进行标识。文件格式要求如下： ▪ 每条数据格式为 `设备 ID,业务方消息 ID,占位符1= XXX,占位符2=XXX……`，其中业务方消息 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。 数据示例： mpaas_push_demo,123456,title=111,content=222。 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，单击图标，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**：调用移动分析人群，对所选移动分析人群推送相同消息。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。当所选推送模板中包含占位符时，移动分析人群不可选。
推送模板	是	选择消息模板，可选当前应用的 **消息模板** 页面上的所有模板。

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。
安卓消息通道推送优先级	是	仅针对 Android 推送平台，可选择： ◦ **优先厂商通道**：优先使用厂商通道推送消息。对于已接入的厂商通道，消息走对应的厂商通道服务；对于未接入的厂商通道，消息走 MPS 自建通道。 ◦ **MPS 通道**：使用 MPS 自建通道推送消息。 对于 Android 推送平台，本参数为自建通道和厂商通道推送的选择入口。对于 iOS 推送平台，您无需配置本参数（iOS 推送为厂商通道推送）。
推送时间	是	选择何时推送消息： ◦ **立即**：推送时间为当前时间，即消息推送任务创建完成后立即推送。 ◦ **定时**：在指定时间推送消息。例如，指定在 6.19 日早上 8:00 推送消息。 ◦ **循环**：在指定时间范围内重复循环推送消息，例如指定在 6.1 ~ 9.30 期间，每周五早上 8:00 推送消息。 **重要** 当推送目标为移动分析人群或自定义标签人群时，不支持定时和循环推送。

对话框右侧区域为 **推送预览** 区域。单击 **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。

4. （选填）根据实际需要，配置高级信息。在高级信息配置区域，完成以下配置：
   - **单击后跳转**：选择在手机上单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，实现后续操作。
     - **客户端自定义**：单击消息后，页面跳转至原生页面。
     - **网页**：单击消息后，页面跳转至网页。
   - **跳转地址**：在手机上单击消息后访问的页面。根据 **单击后跳转** 选项，填写具体的页面地址：
     - 如果是 **客户端自定义**，填写需要访问的原生页面地址（Android：ActivityName；iOS：VCName）。
     - 如果是 **网页**：填写需要访问的网页地址。
   - **推送登录状态**：根据用户的登录状态进一步圈定目标推送人群。指定登录/登出时段时，**永久** 表示对所有登录/登出的用户推送消息，没有时间限制。

     重要

     仅当群发/批量推送方式为设备维度推送时，推送登录状态可配置。

     - 如选择 **登录用户**，将会对在指定时段内登录 App 的用户推送消息。例如，选择登录时段为 15 天，则表示对最近 15 天内登录的用户推送消息。
     - 如选择 **登出用户**，将会对在指定时段内登出 App 用户推送消息。例如，选择登出时段为 15 天，则表示对最近 15 天内登出的用户推送消息。

- 如同时选择 **登录用户** 和 **登出用户**，将会对在指定时段内登录和登出的用户推送消息。例如，选择登录时段为永久，登出时段为 7 天，则表示对所有登录 App 的用户以及最近 7 天内登出 App 的用户推送消息。

- **自定义 ID**：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。
- **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。如不设置，则默认有效期为 180 秒。

  > **说明**
  >
  > 消息有效期不能短于 180 秒，也不得超过 72 小时。

- **扩展参数**：打开 **扩展参数** 开关，单击 **增加参数** 按钮，在 key/value 配置区域中配置 key/value 后，在页面任意区域处单击鼠标左键，完成配置。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

    这类扩展参数被系统占用，参数值不可修改。参数包括 notifyType、action、silent、pushType、templateCode、channel、taskId。

  - 系统具有一定意义的扩展参数

    这类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参数值。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，参数值配置为铃声的路径。该参数仅对小米和苹果手机有效。
badge	应用图标角标，参数值配置为具体数值。该参数会跟随消息体到达客户端。 - 对于 Android 手机，您需要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所配置的参数值。
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。参数值配置为 1。
badge_add_num	华为通道推送角标增加数。
badge_class	华为通道桌面图标对应的应用入口 Activity 类。
big_text	大文本样式，参数值固定为 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。

5. 单击 **提交** 按钮完成创建。创建的消息将展示在批量消息记录列表中。

除了通过控制台推送消息外，还支持通过调用 API 推送消息。具体操作，参见 服务端 API 说明。

相关操作

- 创建消息 - 极简推送
- 管理批量推送消息
- 管理定时推送任务

6.2.3. 管理极简推送消息

极简消息记录列表默认展示最近 30 天的历史消息记录，且支持消息查询。当前消息列表仅展示通过控制台推送的消息记录，调用 API 触发的极简推送消息可通过搜索设备/用户 ID 或自定义消息 ID 来查询消息详情。

查看推送详情

1. 登录 mPaaS 控制台，选择目标应用后，从左侧导航栏进入 **消息推送 > 消息管理 > 极简消息记录** 标签页。
2. 在 **极简消息记录** 页面右上方的搜索框中，输入完整的设备/用户 ID 或自定义消息 ID 搜索消息，消息列表将展示相应的搜索结果。

   > **说明**
   >
   > 仅支持查询最近 30 天内的极简推送消息。

   在消息列表中，消息默认按创建时间倒序排列，列表中展示的信息包括：

   - **自定义消息 ID**：由用户自定义或系统自动生成，为消息的唯一标识，用于唯一标识一条消息。
   - **消息推送时间**：推送消息的时间，精确到秒。
   - **推送时间**：分为立即、定时、循环三种。
   - **推送维度**：展示消息的推送维度，可以是用户、iOS 设备、Android 设备。
   - **推送目标 ID**：具体的用户 ID 或设备 ID。
   - **推送标题**：消息的标题。
   - **消息推送状态**：显示消息的推送状态，成功或失败。各消息推送状态码及其含义详见 消息推送状态码。
3. 在列表中单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。
   - **离线保留时长**：指消息的有效期。在消息未推送成功之前，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。
   - **展示类型**：展示消息是小文本、大文本还是图文消息。
   - **扩展参数**：展示创建消息时添加的扩展参数。
   - **消息内容**：展示消息的正文内容。

撤回消息

支持对已推送成功的消息进行撤回。仅支持撤回最近 7 天内的消息。更多关于消息撤回的说明，参考 消息撤回 文档。

对于静默消息，执行撤回操作后，消息将直接被撤回，用户无感知；对于非静默消息，推送中的消息将会停止推送，已推送但未展示的消息将会取消展示。

> **说明**
>
> 推送状态为“失败”的消息不可撤回。

6.2.4. 管理批量推送消息

消息推送服务提供推送任务统计功能，支持实时统计通过控制台创建和调用 API 触发的批量推送任务，方便用户了解消息的推送情况。

查看推送任务

1. 登录 mPaaS 控制台，选择目标应用后，从左侧导航栏进入 **消息推送 > 消息管理 > 批量消息记录** 标签页。
2. 在 **批量消息记录** 标签页右上方的搜索框中，输入完整的推送任务 ID 或推送任务名称，指定推送开始时间搜索任务，任务列表将展示相应的筛选或搜索结果。

   在任务列表中，任务默认按创建时间倒序排列，列表中展示的信息包括：

   - **推送任务 ID**：推送任务的唯一标识，系统自动生成。
   - **推送任务名称（API）**：如果是通过控制台下发的推送任务，任务名称由系统自动生成，一般为“控制台 + 时间”，例如“控制台 Wed Mar 24 14:47:23 CST 202”；如果是通过调用 API 触发的推送任务，则任务名称为调用方所填写的名称。
   - **推送类型**：立即、定时或循环。
3. 在列表中单击目标任务的展开图标（+）查看相应消息的推送详情。其中：
   - **推送量**：指消息推送后端推出的消息总条数，由后端统计。
   - **推送成功量**：指消息推送后端实际成功推出的消息条数，由后端统计。
   - **推送到达量**：实际到达设备的消息条数。
     - 对于 iOS 或 Android 厂商通道（如小米、华为），依赖消息推送至对应通道后，其后端服务成功返回结果统计。
     - 对于 Android 自建通道，依赖消息推送至客户端本地后埋点上报。
   - **离线保留时长**：指消息的有效期。在消息未推送成功之前，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。

撤回消息

支持对已推送成功的消息进行撤回。仅支持撤回最近 7 天内的消息。更多关于消息撤回的说明，参考 消息撤回 文档。

对于静默消息，执行撤回操作后，消息将直接被撤回，用户无感知；对于非静默消息，推送中的消息将会停止推送，已推送但未展示的消息将会取消展示。

> **说明**
>
> 推送状态为“失败”的消息不可撤回。

6.2.5. 管理定时推送任务

定时推送任务列表展示通过控制台创建和通过调用 API 触发的所有定时推送任务和循环推送任务。一个循环推送任务可能生成一个或多个定时推送任务。

查看定时推送任务

1. 登录 mPaaS 控制台，选择目标应用后，从左侧导航栏进入 **消息推送 > 消息管理 > 定时推送任务** 标签页。

2. 在 **定时推送任务** 标签页右上方的搜索框中，选择日期范围，推送方式，输入推送任务 ID，单击 **搜索** 按钮（ ），或按回车键进行搜索。列表中将显示搜索的任务结果。

3. 选择消息推送类型（控制台推送类型或 API 推送类型，默认展示全部推送）、时间范围筛选消息，或者输入推送任务 ID 搜索消息，列表将展示相应的筛选或搜索结果。在任务列表中，任务默认按创建时间倒序排列，列表中展示的信息包括：
   - **预定推送时间**：对应您在创建推送任务时指定的推送时间。
   - **定时任务 ID**：定时推送任务的唯一标识，系统自动生成。
   - **消息分类**：定时或循环。
   - **推送维度**：展示消息的推送维度，用户维度或设备维度。
   - **推送标题**：消息的标题。
   - **推送文本**：消息的正文内容。
   - **推送类型**：极简推送或批量推送。
   - **创建形式**：消息的创建方式，可以是控制台推送或 API 推送。
   - **推送状态**：显示定时任务是否已执行。

取消定时推送任务

对于未执行的定时推送任务，支持取消推送。每个循环推送任务下都包含了一个或多个定时推送任务，因此取消循环推送任务时，需确认是取消当次推送任务，还是取消全部推送任务。

除了通过控制台取消定时任务，消息推送还支持通过 API 调用的方式取消定时推送任务，具体参见 取消定时推送任务。

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，减少重复内容的输入。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，选择目标应用后，从左侧导航栏进入 **消息推送** > **消息模板** 页面。
2. 在右侧页面中，单击 **创建模板** 按钮，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。
模板正文	是	模板的正文。
展示类型：是否静默	是	是否展示消息。 ◦ **是**：表示静默消息，即用户对消息无感知，在目标设备上不以任何形式展示的消息。 ◦ **否**：指在通知栏展示消息。 对于 Android 推送平台，需要根据不同的推送通道，执行不同后续操作： ◦ **自建通道**：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，控制消息的展示。 ◦ **厂商通道**：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 和鸿蒙推送平台，消息的展示为厂商系统行为，无需执行其他操作。
展示样式	是	消息在客户端的展示样式，支持默认（小文本）、大文本、图文消息三种样式。 ◦ **默认**：该样式展示的内容包含推送标题和文本，适用于内容简洁明了的消息。消息文本长度建议不超过 100 个字符（包括自定义参数和符号）。 ◦ **大文本**：该样式展示的内容包含推送标题和文本，适用于文字内容较多的消息（例如资讯类、新闻类消息），让用户无需打开应用，也能快速获取信息。消息文本长度建议不超过 256 个字符（包括自定义参数和符号）。 ◦ **图文**：该样式支持消息中带图标和大图，适用于除普通文本之外更丰富的内容。出于展示效果考虑，建议消息文本长度不超过两行。

消息图标	否	展示在通知栏消息内容右侧的消息图标，支持 jpg、jpeg、png 格式的图片。此处填写公网可访问的图标物料 URL 地址。若各厂商通道未上传相应的物料，仅上传默认物料 URL时，系统会自动拉取默认物料去支持各厂商通道的图标显示，但因为各厂商通道对物料要求不尽相同，为避免效果不好，建议按各通道要求分别上传物料。 ◦ **默认图标**：建议尺寸为 140 * 140px，大小 50 KB 以内 ◦ **OPPO 通道图标**：建议尺寸为 140 * 140px，大小 50 KB 以内 ◦ **小米通道图标**：建议尺寸为 120 * 120px，大小 50 KB 以内 ◦ **华为通道图标**：建议尺寸为 40 * 40dp，大小 512 KB 以内 ◦ **FCM 通道图标**：无明确配置要求，系统自动拉取默认图标适配。
消息大图	否	展示在通知栏消息内容下侧的图片。此处填写公网可访问的大图物料 URL 地址。若各厂商通道未上传相应的物料，仅上传默认物料 URL 时，系统会自动拉取默认物料去支持各厂商通道的大图显示，但因为各厂商通道对物料要求不尽相同，为避免效果不好，建议按各通道要求分别上传物料。 ◦ **默认大图**：建议尺寸为 876 * 324px，大小 1 MB 以内，支持 jpg、jpeg、png 格式 ◦ **OPPO 通道大图**：建议尺寸为 876 * 324px，大小 1 MB 以内，支持 jpg、jpeg、png 格式 ◦ **小米通道大图**：建议尺寸为 876 * 324px，大小 1 MB 以内，支持 jpg、jpeg、png 格式 ◦ **iOS 通道大图**：由用户自定义图片，无尺寸限制 ◦ **FCM 通道大图**：无明确配置要求，系统自动拉取默认大图适配。 ◦ **鸿蒙通道大图**：建议尺寸小于 25000px，支持 png、jpg、jpeg、heif、gif、bmp 格式。
单击后跳转	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处理。可选： ◦ **网页**：单击消息后，跳转至网页。选择该项后，需要填写要访问的网页 URL。 ◦ **客户端自定义**：单击消息后，跳转至原生页面。选择该项后，需要填写要访问的原生页面地址（Android：ActivityName；iOS：VCName）。
跳转地址	否	单击消息后将要访问的页面地址。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。您需要实现跳转逻辑。

3. 单击 **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

1. 登录 mPaaS 控制台，选择目标应用后，从左侧导航栏进入 **消息推送** > **消息模板** 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。

2. 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. 消息撤回

消息推送提供消息撤回功能，即对已经推送的消息进行撤回，使已发送但未被单击或清除的通知在终端设备通知栏消失。消息撤回功能主要针对由于误操作导致推送错误消息内容、或由于业务临时变更等情况需要紧急撤回已经推送的消息的场景，以减少业务损失和影响范围。

mPaaS 控制台提供消息状态查询、消息撤回功能。除此之外，消息推送提供后端 API，支持业务系统通过调用 API 的方式进行消息撤回。

针对不同的推送通道，消息撤回的实现方式有所差异，具体说明见下表。

推送通道		是否支持撤回	撤回方式
厂商通道	华为	是	覆盖消息。客户端接收到消息撤回的指令后，会将通知栏中展示的消息删除，同时显示“消息已撤回”。
	小米	是	覆盖消息。客户端接收到消息撤回的指令后，会将通知栏中展示的消息删除，同时显示“消息已撤回”。
	OPPO	是	覆盖消息。客户端接收到消息撤回的指令后，会将通知栏中展示的消息删除，同时显示“消息已撤回”。
	vivo	是	撤回消息。客户端接收到消息撤回的指令后，直接将通知栏中展示的消息删除，即消息从通知栏消失。

	苹果（iOS）	是	覆盖消息。客户端接收到消息撤回的指令后，会将通知栏中展示的消息删除，同时显示“消息已撤回”。
	荣耀	是	覆盖消息。客户端接收到消息撤回的指令后，会将通知栏中展示的消息删除，同时显示“消息已撤回”。
	鸿蒙	是	撤回消息。客户端接收到消息撤回的指令后，直接将通知栏中展示的消息删除，即消息从通知栏消失。
MPS 自建通道		是	覆盖消息。客户端接收到消息撤回的指令后，会将通知栏中展示的消息删除，同时显示“消息已撤回”。
短信推送		否	下发的短信消息将无法撤回。

通过控制台撤回

1. 登录 mPaaS 控制台，选择目标应用后，从左侧导航栏进入 **消息推送 > 消息管理** 页面。
2. 选择消息推送任务类型，进入相应的消息列表页。
3. 选择要撤回的消息，单击 **撤回** 并确认即可。 执行撤回操作后，推送中的消息将会停止推送，已推送但未展示的消息将会取消展示。

调用 API 撤回

通过极简推送方式推送的消息可通过消息 ID 撤回；通过批量推送方式推送的消息可通过任务 ID 撤回。仅支持撤回最近 7 天内的消息。

具体如何调用 API 实现消息撤回功能，参见 消息撤回 API 说明文档。

6.5. 用户标签管理

消息推送支持通过设置标签来自定义推送人群，方便用户管理。推送消息时，指定某一用户标签，即可向所有打上该标签的用户发送消息。

标签是用户的一种属性，用于描述用户的基础属性、兴趣爱好、行为特征等。在给用户设置某类标签后，就可以通过标签圈选具有共同特征的用户群，实现精准推送。例如，可以为女性用户打上“女性”标签，打标后，就可以通过圈选拥有“女性”标签的人群，在 3.8 妇女节向该人群推送女士专属消息。

用户与标签是多对多的关系，即一个用户可以对应多个标签，一个标签也可以对应多个用户。

创建用户标签

创建用户标签的过程就是给具有共同特征的一群用户打上标签的过程。

操作步骤如下：

1. 登录 mPaaS 控制台，选择目标应用后，从左侧导航栏进入 **消息推送 > 设置 > 用户标签管理** 页面。
2. 单击 **创建用户标签**，在创建面板中设置标签名称，添加人群。
   - **标签名称**：标签名称尽量能够直接反应人群特征，以方便管理。支持输入任意字符，最多可输入 30 个字符，标签名称需要在应用内保证唯一。
   - **添加人群**：有直接添加和人群导入两种方式。

- **直接添加**：直接在文本框中输入一个或多个用户 ID。多个用户 ID 之间使用英文逗号（,）分隔，每条数据的长度不能超过 60 个字符，超过 60 字符的数据将不予以录入，总长度不超过 10,000 字符。
- **文件导入**：上传包含用户 ID 信息的 .txt 文件（不超过 100 MB）。文件中多个用户 ID 之间用换行符隔开。每条数据的长度不能超过 60 个字符，超过 60 字符的数据将不予以录入。最大可上传的用户 ID 数量为 500,000 个。在导入 ID 过程中，系统将自动对 ID 进行去重。

3. 配置完毕后，单击 **提交** 完成标签创建。新建的用户标签将展示在列表中。

查看用户标签

用户标签列表按创建时间倒序展示标签，包括标签名称、标签 ID、人群数量、创建时间、更新时间。其中：

- 标签 ID：在标签创建成功后，由系统自动生成。
- 人群数量：该标签人群中包含的用户 ID 数量。

在用户标签列表中，单击 **操作** 列下的 **详情**，可查看该标签的具体信息。

编辑用户标签

在用户标签列表中，单击 **操作** 列下的 **编辑**，编辑标签名称，或修改该标签对应的用户信息。

修改标签对应的用户的操作可参考 创建用户标签 中的添加人群部分。

删除用户标签

在用户标签列表中，单击 **操作** 列下的 **删除**，删除该用户标签。删除标签的同时，将会删除该标签对应的所有用户信息。

导出标签人群

在用户标签列表中，单击 **操作** 列下的 **导出**，下载该标签对应的用户列表。

6.6. 设备状态查询

消息推送支持根据用户 ID（UserId）或设备 ID（DeviceId）来查询推送的目标设备状态。方便您在功能开发调试时，通过查看设备状态来进行问题排查。

查询设备状态的步骤如下：

1. 登录 mPaaS 控制台，选择目标应用后，从左侧导航栏进入 **消息推送** > **查询工具** 页面。
2. 设置查询条件，查询目标设备状态。

   选择查询维度用户 ID 或设备 ID，输入相应的用户 ID 或设备 ID 后进行搜索，即可查询出推送设备的相关信息，包括用户 ID、设备 ID、自建 Token、厂商 Token、平台系统、设备厂商、自建通道在线状态信息。其中：

   - **用户 ID**：指用户调用绑定接口时传入的 `userid` 值。
   - **设备 ID**：如果是 Android 设备，则指自建通道 Token；如果是 iOS 设备，则指 APNS Token；如果是鸿蒙设备，则指鸿蒙 Token。
   - **自建 Token**：表示自建通道推送标识。
   - **厂商 Token**：表示厂商推送通道标识。
   - **自建渠道在线状态**：表示当前设备的自建通道是否在线。
     - 如果是 Android 设备，其状态为 Online（在线）或 Offline（离线）。
     - 如果是 iOS 设备，由于 iOS 平台使用第三方通道推送消息，因此其状态固定为 Unknown（未知）。
     - 如果是鸿蒙设备，由于鸿蒙平台使用第三方通道推送消息，因此其状态固定为 Unknown（未知）。

6.7. 通道配置

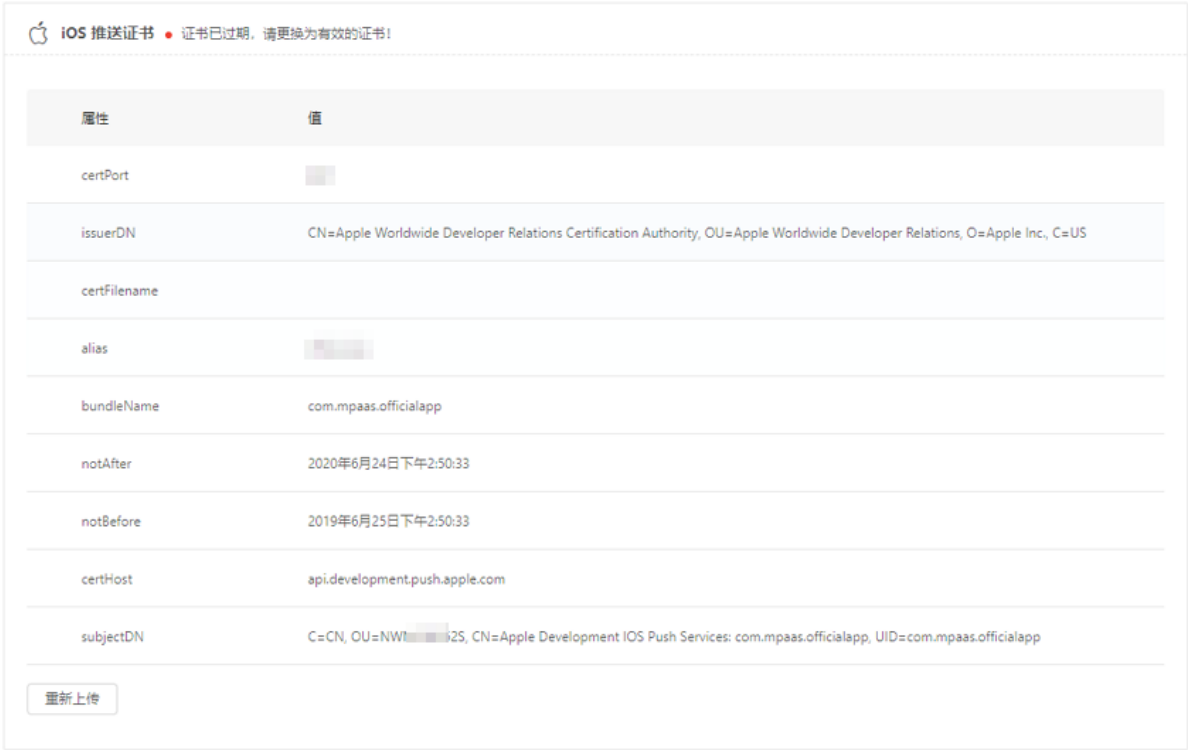
本文介绍如何进行 iOS 和 Android 推送通道配置。

配置 iOS 推送通道

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。

完成以下步骤配置 iOS 推送证书：

1. 登录 mPaaS 控制台，选择目标应用后，从左侧导航栏进入 **消息推送 > 设置** 页面。
2. 在右侧的设置页面上，选择 **通道配置** 标签，在 **iOS 通道** 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，即导出 .p12 证书时所设置的密码。
3. 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，可通过在控制台推送消息进行测试。



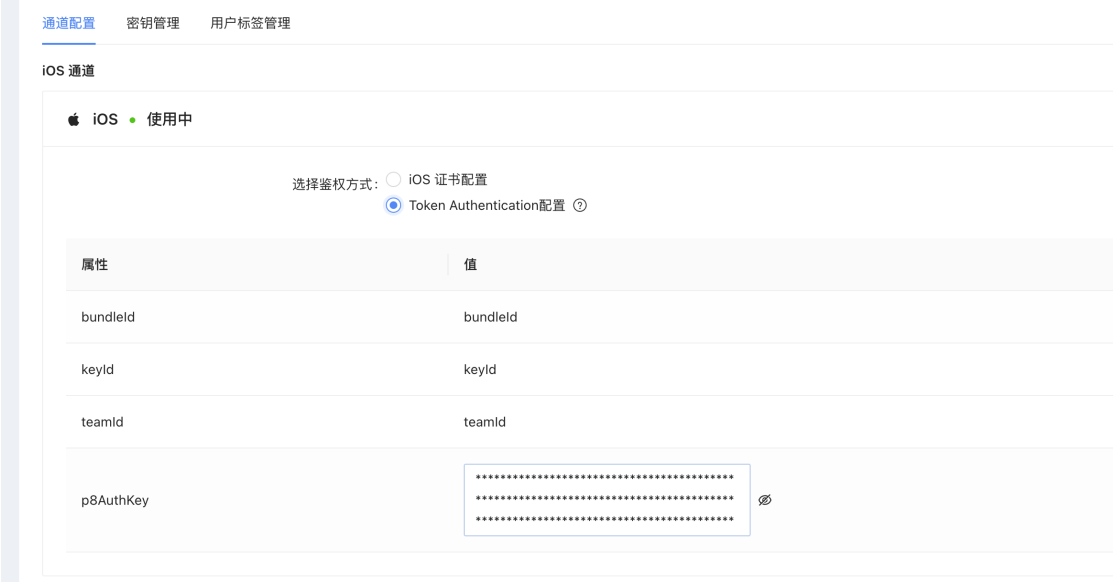
> **说明**
>
> iOS 推送证书具有有效期，请在推送证书失效前及时更新证书，以免消息推送无法正常工作。系统会在证书失效前 15 天开始提醒更换。如需更换证书，单击证书信息下方的 **重新上传** 上传新的证书即可。

配置 iOS 实时活动消息推送证书

> **重要**
>
> 在配置 iOS 实时活动消息推送证书之前，首先要确定 iOS 原推送证书，即 `.p12` 证书，已经配置完毕，否则将无法配置实时活动消息证书。

完成以下步骤配置 iOS 实时活动消息推送证书：

1. 登录 mPaaS 控制台，选择目标应用后，从左侧导航栏进入 **消息推送 > 设置** 页面。
2. 在 **iOS 通道** 的设置页面上，勾选 **Token Authentication 配置**，配置好 bundleId、keyId、teamId 后，上传 p8AuthKey 私钥文件，即 .p8 文件，而后点击 **保存**。



> **重要**
>
> 实时活动消息推送的环境和原 iOS 证书之间绑定，因此使用效果如下：
>
> - 若原 iOS 证书为测试环境 sandbox 证书，则推送测试环境实时活动消息。
> - 若原 iOS 证书为生产环境证书，则推送生产环境实时活动消息。

配置 Android 推送通道

为提升推送的到达率，mPaaS 集成了华为、小米、OPPO 和 vivo 等厂商推送通道。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通道后，能够帮助应用获得稳定的推送性能，因此建议您将厂商推送通道接入应用。

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
- 配置荣耀推送渠道
- 配置小米推送通道
- 配置 OPPO 推送通道
- 配置 vivo 推送通道
- 配置 FCM 推送通道
- 配置新版 FCM 通道

前置条件

您需要先完成客户端侧的接入配置，操作参见 接入厂商推送通道。

操作方法

配置华为推送渠道

1. 从左侧导航栏进入 **消息推送 > 设置 > 通道配置** 标签页。

2. 单击 **华为推送通道** 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**
>
> 可登录 华为开发者联盟 官网，进入 **管理中心** > **我的产品** > **移动应用详情** 页面中获取应用包名、应用 App ID 和密钥。

3. 点击 **确定** 按钮，保存配置。

配置荣耀推送渠道

1. 从左侧导航栏进入 **消息推送** > **设置** > **通道配置** 标签页。
2. 单击 **荣耀推送通道** 配置区域右上角的 **配置**，页面上展示配置入口。

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 将根据配置接入荣耀推送渠道；关闭开关，即取消接入。
包名	是	支持自定义荣耀应用包名。

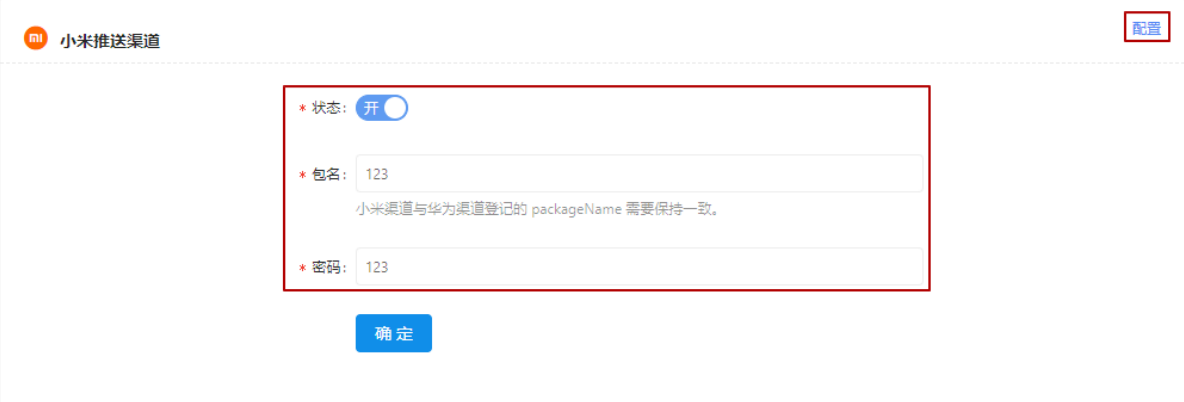
荣耀 AppID	是	唯一应用标识符，在开发者平台开通对应应用的荣耀推送服务时生成。
荣耀应用 ID	是	应用的客户 ID，用于获取发送消息令牌的 ID，在开发者平台开通对应应用 PUSH 服务时生成。
荣耀应用密钥	是	输入荣耀应用的密钥（App Secret）。

> **说明**
>
> 可登录 荣耀开发者联盟 官网，进入 **管理中心** > **我的产品** > **移动应用详情** 页面中获取应用包名、应用 App ID 和密钥。

3. 点击 **确定** 按钮，保存配置。

配置小米推送通道

1. 从左侧导航栏进入 **消息推送** > **设置** > **通道配置** 标签页。
2. 单击 **小米推送通道** 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	通道的接入状态开关。打开开关，MPS 将根据配置接入小米推送通道；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**
>
> 可登录 小米开放平台，进入 **应用管理** > **应用信息** 页面获取包名和密钥。

3. 点击 **确定** 按钮，保存配置。

配置 OPPO 推送通道

1. 从左侧导航栏进入 **消息推送 > 设置 > 通道配置** 标签页。
2. 单击 **OPPO 推送通道** 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	通道的接入状态开关。打开开关，MPS 将根据配置接入 OPPO 推送通道；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。

3. 点击 **确定** 按钮，保存配置。

配置 vivo 推送通道

1. 从左侧导航栏进入 **消息推送 > 设置 > 通道配置** 标签页。
2. 单击 **VIVO 推送通道** 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	通道的接入状态开关。打开开关，MPS 将根据配置接入 vivo 推送通道；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。
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。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）。

3. 点击 **确定** 按钮，保存配置。

配置 FCM 推送通道

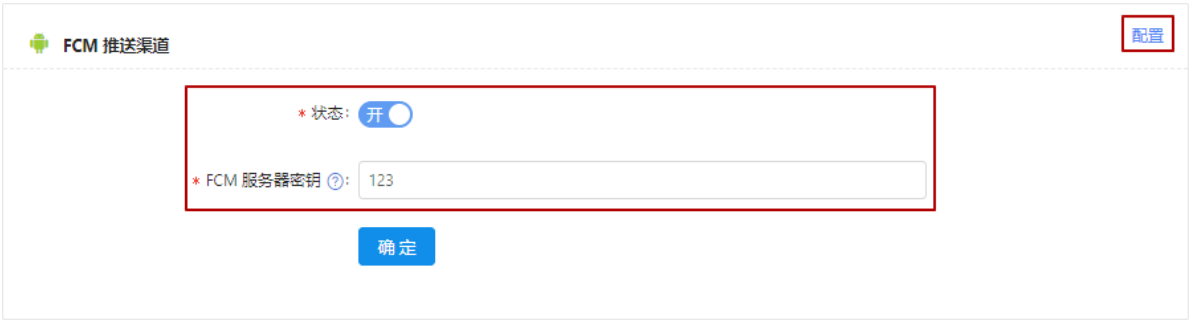
接入海外安卓设备时，依赖谷歌的 FCM 服务作为消息推送网关，需要在控制台侧配置 FCM 推送通道。

前提条件

进行 FCM 推送通道配置前，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。

操作步骤

1. 从左侧导航栏进入 **消息推送 > 设置 > 通道配置** 标签页。
2. 单击 **FCM 推送通道** 配置区域右上角的 **配置**，配置通道信息，如下图所示。



3. 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。
4. 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。
5. 单击 **确定**，保存配置。

配置新版 FCM 通道

> **重要**
>
> 自 2024 年 6 月 20 日起，FCM 旧版 API 不再受到支持，并且会停用。为避免推送通知服务发生任何中断，请您尽快迁移到新版 FCM API。

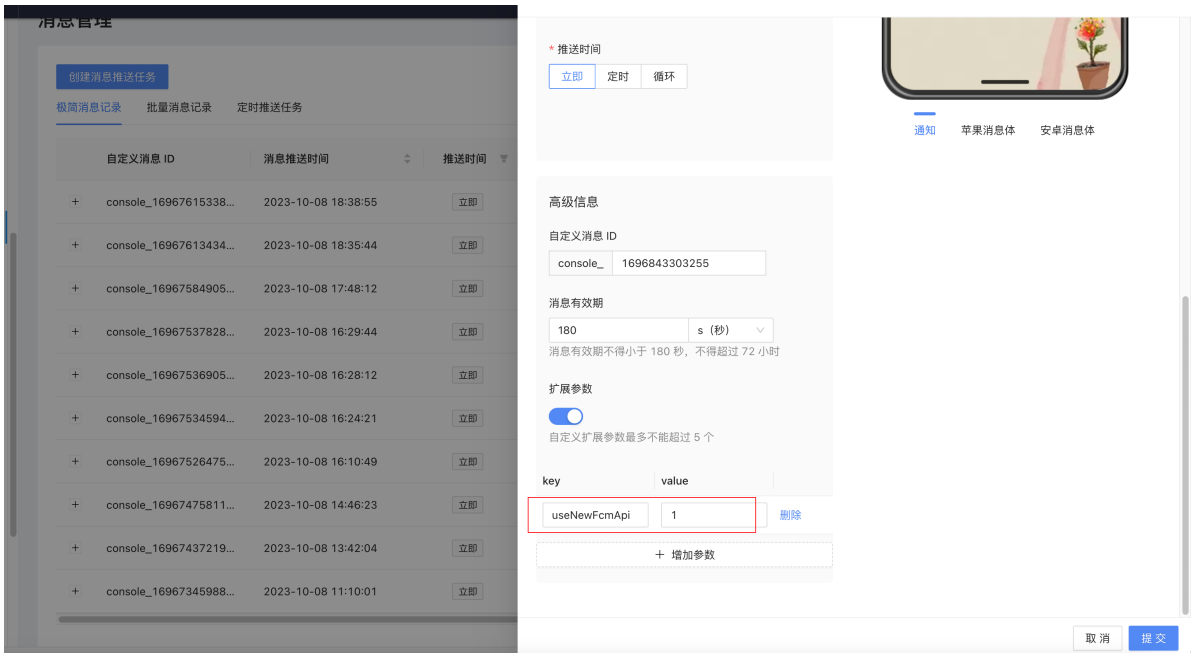
1. 通过控制台上传 FCM 鉴权文件。

Firebase 项目支持 Google 服务账号，您可以使用这些账号从应用服务器或受信任环境调用 Firebase 服务器 API。如果您在本地编写代码，或在本地部署您的应用，则可以通过此服务账号获取的凭据来对服务器请求进行授权。

> **说明**
>
> 如需对服务账号进行身份验证并授予其访问 Firebase 服务的权限，您必须生成 JSON 格式的私钥文件，操作步骤如下：
>
> i. 在 Firebase 控制台中，选择 **设置** > **服务账号**。
>
> ii. 单击 **生成新的私钥**，而后通过单击 **生成密钥** 按钮进行确认。
>
> iii. 妥善存储包含密钥的 JSON 文件。

2. **推送链路切换方式。**

   通过新版 FCM 逻辑提供的链路切换方式是增加扩展参数（extended_params）配置，增加一个键值对 `useNewFcmApi=1` 表示通过新链路进行消息推送。



   推送消息时，需要添加扩展字段：

   - 旧版：`useNewFcmApi`，0；
   - 新版：`useNewFcmApi`，1；

   不添加扩展参数时默认为旧版。

6.8. 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，并在消息推送控制台的 **密钥管理** 页面上的 **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，则需要在消息推送控制台的 **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：
  i. 通过 OpenSSL 官网 下载并安装 OpenSSL 工具（1.1.1 或以上版本）。
  ii. 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
  ```

  iii. 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

完成以下步骤，配置推送接口：

1. 登录 mPaaS 控制台，选择目标应用后，从左侧导航栏进入 **消息推送 > 设置** 页面。
2. 在右侧页面上，单击 **密钥管理** 标签，进入密钥管理页面。
3. 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。

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**
> 为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 说明。

4. 点击 **确定** 按钮，保存配置。

配置推送回调接口

1. 在密钥管理页面，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。

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 服务端 API。

2. 点击 **确定** 按钮，保存配置。

   使用不同通道推送消息时，移动推送核心回调时机不同，具体如下所示。

> **说明**
> - 厂商通道（FCM/APNs/小米/华为/OPPO/vivo）：调用第三方服务成功时，发起回调。
> - 自建通道：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 JSON 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。

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

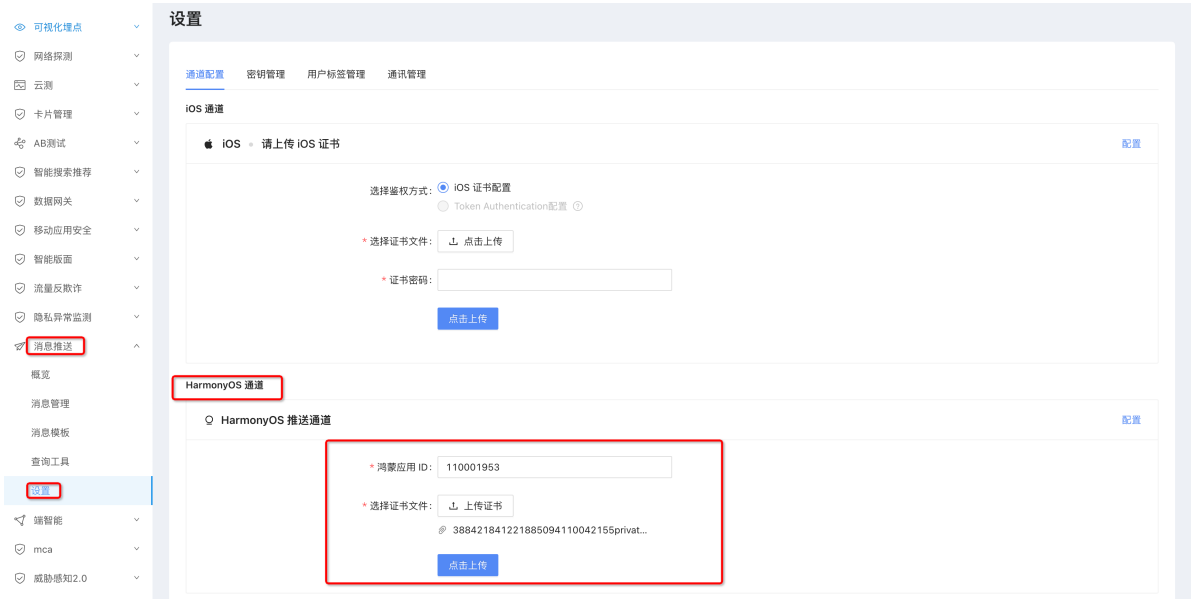
7.HarmonyOS NEXT 控制台使用（beta）

本文介绍了如何配置鸿蒙厂商渠道。

接入鸿蒙设备时，需要在控制台侧上传鸿蒙推送凭证，用于连接鸿蒙的 PUSH 服务。

完成以下步骤配置鸿蒙推送凭证：

1. 鸿蒙开放平台生成 Auth 凭证。
2. 上传凭证文件和鸿蒙应用 ID 到 MPS 控制台。



> **说明**
>
> 其中的鸿蒙应用 ID 可以在开放平台获取，如下图所示。

3. 单击 **点击上传** 即可完成鸿蒙厂商通道配置。

8.API 说明

8.1. 客户端 API

消息推送提供以下客户端 API，具体描述见下表。

调用方式	API	描述
RPC 调用	绑定	绑定用户标识和设备标识（Ad-token）。
	解绑	解绑用户标识和设备标识（Ad-token）。
	厂商通道设备上报	绑定厂商通道设备标识（Ad-token）。

mPaaS 中间层的 `MPPush` 类封装了移动推送组件所有 API，包括绑定、解绑以及厂商通道设备上报等。上述接口方法的 PRC 调用通过移动网关 SDK 来实现。

绑定

- **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。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
success	接口调用是否成功。 ◦ true：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

结果码	结果码名称	消息	说明
3012	NEED_USERID	need userid	调用接口时，入参 `userId` 为空。
3001	NEED_DELIVERY TOKEN	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, Push
MsgService.mAdToken);
      handlePbPBResult("绑定用户操作", resultPbPB);
  }
```

解绑

- **方法定义**

  本方法用于解绑用户标识和设备标识。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的返回值。

- **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);
  }
```

厂商通道设备上报

- **方法定义**

  该方法用于同步绑定厂商通道设备标识与本机设备标识，即上报厂商通道设备标识和 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，方可使用厂商通道推送消息。

  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 thirdChan
nel, 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	厂商通道。枚举值如下： ◦ 2：苹果 ◦ 4：小米 ◦ 5：华为 ◦ 6：FCM ◦ 7：OPPO ◦ 8：vivo
thirdChannelDeviceToken	String	厂商通道设备标识

- **返回值**

  调用本方法的返回值同绑定接口的返回值。

- **使用示例**

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

问题排查

如果通过 RPC 请求进行资源调用过程中出现异常，请参考 无线保镖结果码说明 进行排查。

8.2. 服务端 API

消息推送支持通过 API 接口调用的方式实现极简推送、模板推送、批量推送、群发推送、消息撤回、使用分析、定时推送功能。消息推送支持即时推送、定时推送、循环推送三种不同推送策略，以满足您在不同场景下的推送需求，减少重复工作量。同时，提供短信补充服务，即通过短信通道进行消息补充，以提升消息触达率。

> **重要**
>
> - 目前，仅杭州非金融区提供短信补充服务。
> - 使用短信业务，会产生额外的运营商费用。有关短信服务的计费方式和定价信息，请参考 什么是短信服务。

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。
消息撤回	对已推送的消息进行撤回。通过极简推送或模板推送方式推送的消息可通过消息 ID 撤回；通过批量推送和群发推送方式推送的消息可通过任务 ID 撤回。
使用分析	查询消息推送统计数据，包括总推送条数、推送成功数、推送到达数、消息打开数、消息忽略数等，通过控制台创建或通过调用 API 触发的批量推送任务和群发推送任务列表以及任务详情。
定时推送任务	支持查询定时推送任务列表、取消定时推送任务。定时推送任务分为定时推送和循环推送两种： • 定时推送：在指定时间推送消息。例如，指定在 6.19 日早上 8:00 推送消息。 • 循环推送：在指定时间范围内重复推送消息，例如指定在 6.1 ~ 9.30 期间，每周五早上 8:00 推送消息。一个循环推送任务可能生成一个或多个定时推送任务。

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

> **说明**
>
> 对于非金区（非金融区）用户，消息推送 SDK 最新版本为 3.0.10；对于金区用户，消息推送 SDK 最新版本为 2.1.9。

```
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-mpaas</artifactId>
  <version>3.0.10</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
```

环境变量配置

配置环境变量 MPAAS_AK_ENV 和 MPAAS_SK_ENV。

- Linux 和 macOS 系统配置方法执行以下命令：

```
export MPAAS_AK_ENV=<access_key_id>
export MPAAS_SK_ENV=<access_key_secret>
```

> **说明**
>
> `access_key_id` 替换为已准备好的 AccessKey ID，`access_key_secret` 替换为 AccessKey Secret。

- Windows 系统配置方法
  i. 新建环境变量，添加环境变量 **MPAAS_AK_ENV** 和 **MPAAS_SK_ENV**，并写入已准备好的 AccessKey ID 和 AccessKey Secret。
  ii. 重启 Windows 系统。

极简推送

对一个推送 ID 推送一条消息。

在调用本接口之前，您需要引入依赖，详见 SDK 准备。

请求参数

参数名称	类型	是否必填	示例	描述
classification	String	否	1	用于传递 vivo 推送通道的消息类型： • 0 - 运营类消息 • 1 - 系统类消息 不填则默认为 1。
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 - Android 设备维度 • 2 - iOS 设备维度 • 3 - 用户维度 • 5 - 实时活动 pushToken • 6 - 实时活动 activityId

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 为用户 ID，即用户调用绑定接口时传入的 `userid` 值。
				• value：消息业务 ID，用户自定义，必须保持唯一。
				说明
				推送目标不可以超过 10 个，即 `targetMsgkey` 参数最多可传入 10 个键值对。

expiredSeconds	Long	是	300	消息有效期，单位为秒。
pushStyle	Integer	是	0	推送样式： • 0 - 默认 • 1 - 大文本 • 2 - 图文消息
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 - Web URL • 1 - 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 - 静默 • 0 - 非静默

notifyType	String	否		表示消息通道类型： • transparent - MPS 自建通道 • notify - 默认通道
imageUrls	String	否	{\"defaultUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"oppoUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"miuiUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"fcmUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"iosUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\"}	大图链接（JSON 字符串），支持 OPPO、HMS、MIUI、FCM 和 iOS 推送通道，也可以使用 `defaultUrl` 作为默认值。
iconUrls	String	否	{\"defaultUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"hmsUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"oppoUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"miuiUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\"}	图标链接（JSON 字符串），支持 OPPO、HMS、MIUI、FCM 和 iOS 推送通道，也可以使用 `defaultUrl` 作为默认值。
strategyType	Integer	否	1	推送策略类型： • 0 - 立即 • 1 - 定时 • 2 - 循环 不填则默认为 0。

StrategyContent	String	否	{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circleType\":1,\"circleValue\":[1,7],\"time\":\"13:45:11\"}	推送策略详情（JSON 字符串）。`strategyType` 不等于 0 时必填。具体参数详见下方的 StrategyContent 字段说明。
smsStrategy	int	否	2	短信策略： • 0 - 无（默认） • 1 - 补发 • 2 - 并发
smsSignName	String	否	mPaaS 测试	短信签名
smsTemplateCode	String	否	SMS_216070269	短信模板 ID
smsTemplateParam	String	否	{\"code\": 123456}	短信模板变量对应的实际值，JSON 格式。

thirdChannelCategory	Map	否	thirdChannelCategory: {
			"hms": "9", //华为 FINANCE 财务类型消息
			"vivo": "1"
			//vivo IM 类型消息
			}
				用于传递华为消息分类以及 vivo 二级消息分类参数，详情请参见 厂商消息分类。

miChannelId	String	否	"123321"	小米厂商推送渠道的 channelId
activityEvent	String	否		实时活动事件，可选 update/end： • update - 更新事件 • end - 结束事件
activityContentState	JSONObject	否		实时活动消息的 `content-state`，需和客户端定义的参数保持一致。

dismissalDate	long	否		实时活动消息过期时间（秒级时间戳），可选字段，不传则按 iOS 系统默认失效时间 12h。

> **说明**
>
> 关于 `smsStrategy` 参数：
>
> - 如果 `smsStrategy` 的值不为 0，则 `smsSignName` 、 `smsTemplateCode` 和 `smsTemplateParam` 必填。
>
> 关于 `activityEvent` 参数：
>
> - 当 `activityEvent` 为 end 事件时， `dismissalDate` 配置的过期时间才会生效。
> - 当 `activityEvent` 为 update 事件时， `dismissalDate` 配置的过期时间不会生效。
> - 若传 end 事件但不传 `dismissalDate` ，iOS 系统默认 4h 后结束实时活动。

StrategyContent 字段说明

JSON 格式转化为 String 传值。

参数名称	类型	是否必填	示例	描述
fixedTime	long	否	1630303126000	定时推送时间戳（单位：毫秒，精确到秒）。推送策略类型为定时（ `strategyType` 值为 1）时， `fixedTime` 必填。
startTime	long	否	1640966400000	循环周期开始时间戳（单位：毫秒，精确到天）。推送策略类型为循环（ `strategyType` 值为 2）时， `startTime` 必填。
endTime	long	否	1672416000000	循环周期结束时间戳（单位：毫秒，精确到天），循环结束时间不得超过当天后的 180 天。推送策略类型为循环（ `strategyType` 值为 2）时， `endTime` 必填。
circleType	int	否	3	循环类型： • 1 - 每天 • 2 - 每周 • 3 - 每月 推送策略类型为循环（ `strategyType` 值为 2）时， `circleType` 必填。

circleValue	int[]	否	[1,3]	循环值： • 若循环类型为每天：空 • 若循环类型为每周：设置每周循环的时间，例如 `[1,3]` 表示每周 1 和周 3。 • 若循环类型为每月：设置每月循环推送的时间，例如 `[1,3]` 表示每月 1 号和 3 号。 推送策略类型为循环（ `strategyType` 值为 2）且循环类型（ `circleType` ）非每天时，`circleValue` 必填。
time	String	否	09:45:11	循环推送时间（时分秒，格式为 HH:mm:ss）。推送策略类型为循环（ `strategyType` 值为 2）时，`time` 必填。

> **说明**
> - 未执行的定时或循环推送任务总数上限默认为 100 条。
> - 循环周期为开始时间的 0 点到结束时间的 24 点。
> - 循环开始时间和结束时间均不可早于当天 0 点，且结束时间不得早于开始时间。

返回参数

参数名称	类型	示例	描述
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	请求 ID
ResultCode	String	OK	请求结果码
ResultMessage	String	param is invalid	请求错误描述
PushResult	JSON		请求结果
Success	boolean	true	请求状态。`Success` 参数值包含在 `PushRresult` JSON 字符串中。
ResultMsg	String	param is invalid	请求错误内容。`ResultMsg` 参数值包含在 `PushRresult` JSON 字符串中。

Data	String	903bf653c1b5442b9ba07684767bf9c2	定时推送任务 ID。`strategyType` 不等于 0 时，该字段不为空。

Java 代码示例

点击此处 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
        // 阿里云账号AccessKey拥有所有API的访问权限，建议您使用RAM用户进行API访问或日常运维。
        // 强烈建议不要把AccessKey ID和AccessKey Secret保存到工程代码里，否则可能导致AccessKey泄露，威胁您账号下所有资源的安全。
        // 本示例以将AccessKey ID和AccessKey Secret保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里。
        // 建议先完成环境变量配置
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",           // 地域 ID
            accessKeyId,
            accessKeySecret);

        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);

        request.setPushStyle(2);
        String imageUrls = "{\"defaultUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"oppoUrl\":\"https://pre-mpaas.oss-cn-hangzhou.ali
yuncs.com/tmp/test.png\",\"miuiUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"fcmUrl\":\"https://pre-mpaas.oss-cn-hangzhou.aliy
uncs.com/tmp/test.png\",\"iosUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\"}";
        String iconUrls = "{\"defaultUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"hmsUrl\":\"https://pre-mpaas.oss-cn-hangzhou.aliy
uncs.com/tmp/test.png\",\"oppoUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"miuiUrl\":\"https://pre-mpaas.oss-cn-hangzhou.ali
yuncs.com/tmp/test.png\"}";
        request.setImageUrls(imageUrls);
        request.setIconUrls(iconUrls);

        request.setStrategyType(2);
        request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circl
```

```
ype\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}");

        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

        // 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
        // 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
        // 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
        // 建议先完成环境变量配置
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
# Initialize AcsClient instance
client = AcsClient(
"cn-hangzhou",
accessKeyId,
accessKeySecret
);

# 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
// 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
// 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
// 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
// 建议先完成环境变量配置
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
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
<?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。

在调用本接口之前，确保已完成以下操作：

- 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。
- 引入 SDK 依赖，详见 SDK 准备。

请求参数

参数名称	类型	是否必填	示例	描述

classification	String	否	1	用于传递 vivo 推送通道的消息类型： • 0 - 运营类消息 • 1 - 系统类消息 不填则默认为 1。
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 - Android 设备维度 • 2 - iOS 设备维度 • 3 - 用户维度 • 5 - 实时活动 pushToken • 6 - 实时活动 activityId
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 为用户 ID，即用户调用绑定接口时传入的 `userid` 值。 • value：消息业务 ID，用户自定义，必须保持唯一。 **说明** 推送目标不可以超过 10 个，即 `targetMsgkey` 参数最多可传入 10 个键值对。
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。
notifyType	String	否		表示消息通道类型： • transparent - MPS 自建通道 • notify - 默认通道
strategyType	Integer	否	1	推送策略类型： • 0 - 立即 • 1 - 定时 • 2 - 循环 不填则默认为 0。
StrategyContent	String	否	{\”fixedTime\”:1630303126000,\”startTime\”:1625673600000,\”endTime\”:1630303126000,\”circleType\”:1,\”circleValue\”:[1,7],\”time\”:\”13:45:11\”}	推送策略详情（JSON 字符串）。 `strategyType` 不等于 0 时必填。具体参数详见下方的 StrategyContent 字段说明。
smsStrategy	int	否	2	短信策略： • 0 - 无（默认） • 1 - 补发 • 2 - 并发
smsSignName	String	否	mPaaS 测试	短信签名
smsTemplateCode	String	否	SMS_216070269	短信模板 ID
smsTemplateParam	StringString	否	{\"code\": 123456}	短信模板变量对应的实际值，JSON 格式。

thirdChannelCategory	Map	否	thirdChannelCategory: {
			"hms": "9", //华为 FINANCE 财务类型消息
			"vivo": "1"
			//vivo IM 类型消息
			}
				用于传递华为消息分类以及 vivo 二级消息分类参数，详情请参见 厂商消息分类。

miChannelId	String	否	"123321"	小米厂商推送渠道的 channelId
activityEvent	String	否		实时活动事件，可选 update/end： • update - 更新事件 • end - 结束事件
activityContentState	JSONObject	否		实时活动消息的 `content-state` ，需和客户端定义的参数保持一致。
dismissalDate	long	否		实时活动消息过期时间（秒级时间戳），可选字段，不传则按 iOS 系统默认失效时间 12h。

> **说明**
>
> 关于 `smsStrategy` 参数：
>
> - 如果 `smsStrategy` 的值不为 0，则 `smsSignName` 、 `smsTemplateCode` 和 `smsTemplateParam` 必填。
>
> 关于 `activityEvent` 参数：
>
> - 当 `activityEvent` 为 end 事件时， `dismissalDate` 配置的过期时间才会生效。
> - 当 `activityEvent` 为 update 事件时， `dismissalDate` 配置的过期时间不会生效。
> - 若传 `end` 事件但不传 `dismissalDate` ，iOS 系统默认 4h 后结束实时活动。

StrategyContent 字段说明

JSON 格式转化为 String 传值。

参数名称	类型	是否必填	示例	描述
fixedTime	long	否	1630303126000	定时推送时间戳（单位：毫秒，精确到秒）。推送策略类型为定时（ `strategyType` 值为 1）时， `fixedTime` 必填。

startTime	long	否	1640966400000	循环周期开始时间戳（单位：毫秒，精确到天）。推送策略类型为循环（`strategyType` 值为 2）时，`startTime` 必填。
endTime	long	否	1672416000000	循环周期结束时间戳（单位：毫秒，精确到天），循环结束时间不得超过当天后的 180 天。推送策略类型为循环（`strategyType` 值为 2）时，`endTime` 必填。
circleType	int	否	3	循环类型： • 1 - 每天 • 2 - 每周 • 3 - 每月 推送策略类型为循环（`strategyType` 值为 2）时，`circleType` 必填。
circleValue	int[]	否	[1,3]	循环值： • 若循环类型为每天：空 • 若循环类型为每周：设置每周循环的时间，例如 `[1,3]` 表示每周 1 和周 3。 • 若循环类型为每月：设置每月循环推送的时间，例如 `[1,3]` 表示每月 1 号和 3 号。 推送策略类型为循环（`strategyType` 值为 2）且循环类型（`circleType`）非每天时，`circleValue` 必填。
time	String	否	09:45:11	循环推送时间（时分秒，格式为 HH:mm:ss）。推送策略类型为循环（`strategyType` 值为 2）时，`time` 必填。

说明

- 未执行的定时或循环推送任务总数上限默认为 100 条。
- 循环周期为开始时间的 0 点到结束时间的 24 点。
- 循环开始时间和结束时间均不可早于当天 0 点，且结束时间不得早于开始时间。

返回参数

参数名称	类型	示例	描述

RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	请求 ID
ResultCode	String	OK	请求结果码
ResultMessage	String	param is invalid	请求错误描述
PushResult	JSON		请求结果
Success	boolean	true	请求状态。`Success` 参数值包含在 `PushRresult` JSON 字符串中。
ResultMsg	String	param is invalid	请求错误内容。`ResultMsg` 参数值包含在 `PushRresult` JSON 字符串中。
Data	String	903bf653c1b5442b9ba07684767bf9c2	定时推送任务 ID。`strategyType` 不等于 0 时，该字段不为空。

Java 代码示例

点击此处 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
        // 阿里云账号AccessKey拥有所有API的访问权限，建议您使用RAM用户进行API访问或日常运维。
        // 强烈建议不要把AccessKey ID和AccessKey Secret保存到工程代码里，否则可能导致AccessKey泄露，威胁您账号下所有资源的安全。
        // 本示例以将AccessKey ID和AccessKey Secret保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里。
        // 建议先完成环境变量配置
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",           // 地域 ID
            accessKeyId,
            accessKeySecret);

        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));

        request.setStrategyType(2);
        request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circl
ype\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}");

        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

        // 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
        // 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
        // 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
        // 建议先完成环境变量配置
# Initialize AcsClient instance
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
client = AcsClient(
accessKeyId,
accessKeySecret,
"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
// 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
// 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
// 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
// 建议先完成环境变量配置
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
const client = new Client({
  accessKeyId,
  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
<?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。

> **重要**
>
> 当推送目标为移动分析人群或自定义标签人群时，不支持定时和循环推送。

在调用本接口之前，确保已完成以下操作：

- 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。
- 引入 SDK 依赖，详见 SDK 准备。

请求参数

参数名称	类型	是否必填	示例	描述
classification	String	否	1	用于传递 vivo 推送通道的消息类型： • 0 - 运营类消息 • 1 - 系统类消息 不填则默认为 1。
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，数值选择如下： • 1 - Android 设备维度 • 2 - iOS 设备维度 • 3 - 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	targetMsgs 对象列表	目标对象列表，参数详见下方的 targetMsgs 对象说明。
expiredSeconds	Long	是	300	消息有效期，单位为秒。
extendedParams	String	否	{“key1”:”value1”}	统一扩展参数，为 Map 形式。
notifyType	String	否		表示消息通道类型： • transparent - MPS 自建通道 • notify - 默认通道

strategyType	Integer	否	1	推送策略类型： • 0 - 立即 • 1 - 定时 • 2 - 循环 不填则默认为 0。
StrategyContent	String	否	{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circleType\":1,\"circleValue\":[1,7],\"time\":\"13:45:11\"}	推送策略详情（ISON 字符串）。`strategyType` 不等于 0 时必填。具体参数详见下方的 StrategyContent 字段说明。

thirdChannelCategory	Map	否	thirdChannelCategory: {
			"hms": "9", //华为 FINANCE 财务类型消息
			"vivo": "1"
			//vivo IM 类型消息
			}
				用于传递华为消息分类以及 vivo 二级消息分类参数，详情请参见 厂商消息分类。

miChannelId	String	否	"123321"	小米厂商推送渠道的 channelId
activityEvent	String	否		实时活动事件，可选 update/end： • update - 更新事件 • end - 结束事件
activityContentState	JSONObject	否		实时活动消息的 `content-state`，需和客户端定义的参数保持一致
dismissalDate	long	否		实时活动消息过期时间（秒级时间戳），可选字段，不传则按 iOS 系统默认失效时间 12h。

说明

关于 `activityEvent` 参数：

- 当 `activityEvent` 为 end 事件时，`dismissalDate` 配置的过期时间才会生效。
- 当 `activityEvent` 为 update 事件时，`dismissalDate` 配置的过期时间不会生效。
- 若传 end 事件但不传 `dismissalDate`，iOS 系统默认 4h 后结束实时活动。

targetMsgs 对象说明

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。

StrategyContent 字段说明

JSON 格式转化为 String 传值。

参数名称	类型	是否必填	示例	描述
fixedTime	long	否	1630303126000	定时推送时间戳（单位：毫秒，精确到秒）。推送策略类型为定时（ `strategyType` 值为 1）时， `fixedTime` 必填。
startTime	long	否	1640966400000	循环周期开始时间戳（单位：毫秒，精确到天）。推送策略类型为循环（ `strategyType` 值为 2）时， `startTime` 必填。
endTime	long	否	1672416000000	循环周期结束时间戳（单位：毫秒，精确到天），循环结束时间不得超过当天后的 180 天。推送策略类型为循环（ `strategyType` 值为 2）时， `endTime` 必填。
circleType	int	否	3	循环类型： • 1 - 每天 • 2 - 每周 • 3 - 每月 推送策略类型为循环（ `strategyType` 值为 2）时， `circleType` 必填。

circleValue	int[]	否	[1,3]	循环值： • 若循环类型为每天：空 • 若循环类型为每周：设置每周循环的时间，例如 `[1,3]` 表示每周 1 和周 3。 • 若循环类型为每月：设置每月循环推送的时间，例如 `[1,3]` 表示每月 1 号和 3 号。 推送策略类型为循环（ `strategyType` 值为 2）且循环类型（ `circleType` ）非每天时，`circleValue` 必填。
time	String	否	09:45:11	循环推送时间（时分秒，格式为 HH:mm:ss）。推送策略类型为循环（ `strategyType` 值为 2）时，`time` 必填。

> **说明**
> - 未执行的定时或循环推送任务总数上限默认为 100 条。
> - 循环周期为开始时间的 0 点到结束时间的 24 点。
> - 循环开始时间和结束时间均不可早于当天 0 点，且结束时间不得早于开始时间。

返回参数

参数名称	类型	示例	描述
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	请求 ID
ResultCode	String	OK	请求结果码
ResultMessage	String	param is invalid	请求错误描述
PushResult	JSON		请求结果
Success	boolean	true	请求状态。`Success` 参数值包含在返回的 `PushRresult` JSON 字符串中。
ResultMsg	String	param is invalid	请求错误内容。`ResultMsg` 参数值包含在返回的 `PushRresult` JSON 字符串中。

Data	String	903bf653c1b5442b9ba07684767bf9c2	定时推送任务 ID。`strategyType` 不等于 0 时，该字段不为空。

Java 代码示例

点击此处 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
        // 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
        // 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
        // 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
        // 建议先完成环境变量配置。
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",           // 地域 ID
            accessKeyId,
            accessKeySecret);

        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);

        request.setStrategyType(2);
        request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circl
ype\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}");

        PushMultipleResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
            System.out.println(response.getPushResult().getData());  // 推送任务 ID 或定时推送任务 ID
        } 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

// 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
// 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
// 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
// 建议先完成环境变量配置
# Initialize AcsClient instance
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
client = AcsClient(
accessKeyId,
accessKeySecret,
"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
// 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
// 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
// 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
// 建议先完成环境变量配置
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
const client = new Client({
  accessKeyId,
  accessKeySecret,
  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。

> **重要**
>
> 当推送目标为移动分析人群或自定义标签人群时，不支持定时和循环推送。

在调用本接口之前，确保已完成以下操作：

- 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。
- 引入 SDK 依赖，详见 SDK 准备。

请求参数

参数名称	类型	是否必填	示例	描述
classification	String	否	1	用于传递 vivo 推送通道的消息类型： • 0 - 运营类消息 • 1 - 系统类消息 不填则默认为 1。
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 - Android 群发 • 2 - 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	群发时，推送登录状态： • 0 - 绑定的用户（默认） • 1 - 所有用户（包括绑定和解绑的用户） • 2 - 解绑的用户
bindPeriod	Integer	否		登录时长，当 `pushStatus` 值为 0 时必填： • 1 - 7 天内绑定的用户 • 2 - 15 天内绑定的用户 • 3 - 60 天内绑定的用户 • 4 - 永久 **说明** `bindPeriod` 参数仅在非金环境中可配置。
unBindPeriod	Long	否		退出登录时长，当 `pushStatus` 值为 1 或 2 时必填： • 1 - 7 天内解绑的用户 • 2 - 15 天内解绑的用户 • 3 - 60 天内解绑的用户 • 4 - 永久
androidChannel	Integer	否		安卓消息通道： • 1 - MPS 自建通道 • 2 - 默认通道
strategyType	int	否	1	推送策略类型： • 0 - 立即 • 1 - 定时 • 2 - 循环 不填则默认为 0。
StrategyContent	String	否	{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circleType\":1,\"circleValue\":[1,7],\"time\":\"13:45:11\"}	推送策略详情（JSON 字符串）。`strategyType` 不等于 0 时必填。具体参数详见下方的 StrategyContent 字段说明。

thirdChannelCategory	Map	否	thirdChannelCategory: {
			"hms": "9", //华为 FINANCE 财务类型消息
			"vivo": "1"
			//vivo IM 类型消息
			}
				用于传递华为消息分类以及 vivo 二级消息分类参数，详情请参见 厂商消息分类。

miChannelId	String	否	"123321"	小米厂商推送渠道的 channelId

StrategyContent 字段说明

JSON 格式转化为 String 传值。

参数名称	类型	是否必填	示例	描述
fixedTime	long	否	1630303126000	定时推送时间戳（单位：毫秒，精确到秒）。推送策略类型为定时（ `strategyType` 值为 1）时， `fixedTime` 必填。
startTime	long	否	1640966400000	循环周期开始时间戳（单位：毫秒，精确到天）。推送策略类型为循环（ `strategyType` 值为 2）时， `startTime` 必填。
endTime	long	否	1672416000000	循环周期结束时间戳（单位：毫秒，精确到天），循环结束时间不得超过当天后的 180 天。推送策略类型为循环（ `strategyType` 值为 2）时， `endTime` 必填。
circleType	int	否	3	循环类型： • 1 - 每天 • 2 - 每周 • 3 - 每月 推送策略类型为循环（ `strategyType` 值为 2）时， `circleType` 必填。

circleValue	int[]	否	[1,3]	循环值： • 若循环类型为每天：空 • 若循环类型为每周：设置每周循环的时间，例如 `[1,3]` 表示每周 1 和周 3。 • 若循环类型为每月：设置每月循环推送的时间，例如 `[1,3]` 表示每月 1 号和 3 号。 推送策略类型为循环（ `strategyType` 值为 2）且循环类型（ `circleType` ）非每天时，`circleValue` 必填。
time	String	否	09:45:11	循环推送时间（时分秒，格式为 HH:mm:ss）。推送策略类型为循环（ `strategyType` 值为 2）时，`time` 必填。

> **说明**
> - 未执行的定时或循环推送任务总数上限默认为 100 条。
> - 循环周期为开始时间的 0 点到结束时间的 24 点。
> - 循环开始时间和结束时间均不可早于当天 0 点，且结束时间不得早于开始时间。

返回参数

参数名称	类型	示例	描述
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	请求 ID
ResultCode	String	OK	请求结果码
ResultMessage	String	param is invalid	请求错误描述
PushResult	JSON		请求结果
Success	boolean	true	请求状态。`Success` 参数值包含在 `PushRresult` JSON 字符串中。
ResultMsg	String	param is invalid	请求错误内容。`ResultMsg` 参数值包含在 `PushRresult` JSON 字符串中。

Data	String	903bf653c1b5442b9ba07684767bf9c2	定时推送任务 ID。`strategyType` 不等于 0 时，该字段不为空。

Java 代码示例

点击此处 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
        // 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
        // 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
        // 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
        // 建议先完成环境变量配置
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",           // 地域 ID
            accessKeyId,
            accessKeySecret);

        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));

        request.setStrategyType(2);
        request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circl
ype\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}");

        PushBroadcastResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
            System.out.println(response.getPushResult().getData());  // 推送任务 ID或定时推送任务 ID
        } 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

// 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
// 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
// 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
// 建议先完成环境变量配置
# Initialize AcsClient instance
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
client = AcsClient(
accessKeyId,
accessKeySecret,
"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
// 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
// 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
// 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
// 建议先完成环境变量配置
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
const client = new Client({
  accessKeyId,
  accessKeySecret,
  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
<?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();
    }
}
```

消息撤回

通过极简推送或模板推送方式推送的消息可通过消息 ID 撤回；通过批量推送和群发推送方式推送的消息可通过任务 ID 撤回。仅支持撤回最近 7 天内的消息。

通过消息 ID 撤回

支持撤回通过极简推送和模板推送发送的消息。

请求参数

参数名称	类型	是否必填	示例	描述
messageId	String	是	1578807462788	业务方消息 ID，用户自定义，用于在业务方系统中唯一标识消息。

targetId	String	是	user1024	目标 ID，若原消息以设备维度推送，则目标 ID 为设备 ID；若原消息以用户维度推送，则目标 ID 为用户 ID。

返回参数

参数名称	类型	示例	描述
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	请求 ID
ResultCode	String	OK	请求结果码
ResultMessage	String	param is invalid	请求错误描述
PushResult	JSON		请求结果
Success	boolean	true	请求状态。`Success` 参数值包含在 `PushRresult` JSON 字符串中。
ResultMsg	String	param is invalid	请求错误内容。`ResultMsg` 参数值包含在 `PushRresult` JSON 字符串中。

使用示例

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // 创建 DefaultAcsClient 实例并初始化
        // 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
        // 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
        // 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
        // 建议先完成环境变量配置
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // 地域 ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);

        RevokePushMessageRequest request = new RevokePushMessageRequest();
        request.setAppId("ONEX570DA89211720");
        request.setWorkspaceId("test");
        request.setMessageId("console_1624516744112");  // 业务方消息 ID
        request.setTargetId("mpaas_push_demo");         // 目标 ID

        RevokePushMessageResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

通过任务 ID 撤回

支持撤回通过批量推送和群发推送发送的消息。

请求参数

参数名称	类型	是否必填	示例	描述
taskId	String	是	20842863	推送任务 ID，可在控制台推送任务列表中查询。

返回参数

参数名称	类型	示例	描述
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	请求 ID

ResultCode	String	OK	请求结果码
ResultMessage	String	param is invalid	请求错误描述
PushResult	JSON		请求结果
Success	boolean	true	请求状态。`Success` 参数值包含在 `PushRresult` JSON 字符串中。
ResultMsg	String	param is invalid	请求错误内容。`ResultMsg` 参数值包含在 `PushRresult` JSON 字符串中。

使用示例

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // 创建 DefaultAcsClient 实例并初始化
        // 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
        // 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
        // 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
        // 建议先完成环境变量配置
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",           // 地域 ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);

        RevokePushTaskRequest request = new RevokePushTaskRequest();
        request.setAppId("ONEX570DA89211720");
        request.setWorkspaceId("test");
        request.setTaskId("20842863");      // 推送任务 ID

        RevokePushTaskResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

使用分析

查询统计数据

查询消息推送统计数据，包括总推送条数、推送成功数、推送到达数、消息打开数、消息忽略数等数据。

请求参数

参数名称	类型	是否必填	示例	描述
appId	String	是	ONEX570DA89211721	mPaaS App ID
workspaceId	String	是	test	mPaaS 工作空间
startTime	long	是	1619798400000	要查询的时间范围的开始时间戳，以毫秒为单位，精确到天。
endTime	long	是	1624358433000	要查询的时间范围的结束时间戳，以毫秒为单位，精确到天。开始时间和结束时间间隔不能超过 90 天。
platform	String	否	ANDROID	平台，不传则表示查询全部。可选值：IOS，ANDROID
channel	String	否	ANDROID	推送通道，不传表示查询全部。可选值：IOS、FCM、HMS、MIUI、OPPO、VIVO、ANDROID（自建通道）
type	String	否	SIMPLE	推送类型，不传表示查询全部。可选值：SIMPLE、TEMPLATE、MULTIPLE、BROADCAST
taskId	String	否	20842863	推送任务 ID

返回参数

参数名称	类型	示例	描述
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	请求 ID
ResultCode	String	OK	请求结果码
ResultMessage	String	param is invalid	请求错误描述

ResultContent	JSON		响应内容
data	JSON		响应内容。该参数值包含在 `ResultContent` JSON 字符串中。
pushTotalNum	float	100	推送数
pushNum	float	100	推送成功数
arrivalNum	float	100	推送到达数
openNum	float	100	推送打开数
openRate	float	100	推送打开率
ignoreNum	float	100	推送忽略数
ignoreRate	float	100	推送忽略率

使用示例

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // 创建 DefaultAcsClient 实例并初始化
        // 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
        // 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
        // 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
        // 建议先完成环境变量配置
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",           // 地域 ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);

        QueryPushAnalysisCoreIndexRequest request = new
QueryPushAnalysisCoreIndexRequest();
        request.setAppId("ONEX570DA89211720");
        request.setWorkspaceId("test");
        request.setStartTime(Long.valueOf("1617206400000"));
        request.setEndTime(Long.valueOf("1624982400000"));
        request.setPlatform("ANDROID");
        request.setChannel("ANDROID");
        request.setType("SIMPLE");
        request.setTaskId("20842863");

        QueryPushAnalysisCoreIndexResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

查询推送任务列表

查询通过控制台创建或通过调用 API 触发的批量推送任务和群发推送任务信息。

请求参数

参数名称	类型	是否必填	描述	描述
appId	String	是	ONEX570DA89211721	mPaaS App ID
workspaceId	String	是	test	mPaaS 工作空间

startTime	long	是	1619798400000	开始时间戳，以毫秒为单位，精确到天。
taskId	String	否	20842863	推送任务 ID
taskName	String	否	测试任务	推送任务名称
pageNumber	int	否	1	页码，默认为 1。
pageSize	int	否	10	页数，默认为 500。

返回参数

参数名称	类型	示例	描述
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	请求 ID
ResultCode	String	OK	请求结果码
ResultMessage	String	param is invalid	请求错误描述
ResultContent	JSON		响应内容
data	JSONArray		响应内容。该参数值包含在 `ResultContent` JSON 字符串中。
taskId	String	20927873	任务 ID
taskName	String	测试任务	任务名称
templateId	String	9108	模板 ID

templateName	String	测试模板	模板名称
type	long	3	推送类型，其中： • 2 - 批量推送 • 3 - 群发推送
gmtCreate	long	1630052750000	创建时间

使用示例

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // 创建 DefaultAcsClient 实例并初始化
        // 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
        // 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
        // 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
        // 建议先完成环境变量配置
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",           // 地域 ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);

        QueryPushAnalysisTaskListRequest request = new
QueryPushAnalysisTaskListRequest();
        request.setAppId("ONEX570DA89211721");
        request.setWorkspaceId("default");
        request.setStartTime(Long.valueOf("1617206400000"));
        request.setTaskId("20845212");
        request.setTaskName("测试任务");
        request.setPageNumber(1);
        request.setPageSize(10);

        QueryPushAnalysisTaskListResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

查询推送任务详情

查询通过控制台创建或通过调用 API 触发的批量推送任务和群发推送任务的任务详情。

请求参数

参数名称	类型	是否必填	示例	描述
appId	String	是	ONEX570DA89211721	mPaaS App ID
workspaceId	String	是	test	mPaaS 工作空间
taskId	String	是	20842863	推送任务 ID

返回参数

参数名称	类型	示例	描述
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	请求 ID
ResultCode	String	OK	请求结果码
ResultMessage	String	param is invalid	请求错误描述
ResultContent	JSON		响应内容
data	JSON		响应内容。该参数值包含在 `ResultContent` JSON 字符串中。
taskId	long	20927872	任务 ID
pushNum	float	10	推送数
pushSuccessNum	float	10	推送成功数
pushArrivalNum	float	10	推送到达数

startTime	long	1630052735000	开始时间（毫秒）
endTime	long	1630052831000	结束时间（毫秒）
duration	string	00 小时 01 分 36 秒	持续时间

使用示例

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // 创建 DefaultAcsClient 实例并初始化
        // 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
        // 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
        // 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
        // 建议先完成环境变量配置
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",           // 地域 ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);

        QueryPushAnalysisTaskDetailRequest request = new
QueryPushAnalysisTaskDetailRequest();
        request.setAppId("ONEXPREF4F5C52081557");
        request.setWorkspaceId("default");
        request.setTaskId("20845212");

        QueryPushAnalysisTaskDetailResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

定时推送任务

查询定时推送任务列表

查询已创建的定时推送任务列表，包括定时推送任务和循环推送任务。

请求参数

参数名称	类型	是否必填	示例	描述

appId	String	是	ONEX570DA89211721	mPaaS App ID
workspaceId	String	是	test	mPaaS 工作空间
startTime	long	是	1619798400000	触发定时推送的开始时间戳，并非定时推送任务的创建时间。
endtTime	long	是	1630425600000	触发定时推送的结束时间戳。
type	int	否	0	推送方式，其中： • 0 - 极简推送 • 1 - 模板推送 • 2 - 批量推送 • 3 - 群发推送
uniqueId	String	否	49ec0ed5a2a642bcbe139a2d7a419d6d	定时推送任务的唯一 ID。若传主任务 ID，则返回主任务下的所有子任务的信息；若传子任务 ID，则返回子任务的信息。
pageNumber	int	否	1	页码，默认为 1。
pageSize	int	否	10	分页大小，默认为 500。

返回参数

参数名称	类型	示例	描述
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	请求 ID
ResultCode	String	OK	请求结果码
ResultMessage	String	param is invalid	请求错误描述
ResultContent	JSON		响应内容

data	JSON		响应内容。该参数值包含在 `ResultContent` JSON 字符串中。
totalCount	int	10	总数
list	JSONArray		任务数组
uniqueId	String	56918166720e46e1bcc40195c9ca71db	定时推送任务的唯一 ID。 • 若 `strategyType` 值为 1，表示定时推送任务主 ID。 • 若 `strategyType` 值为 2，表示循环任务子 ID。
parentId	String	56918166720e46e1bcc40195c9ca71db	定时推送任务主 ID。 • 若 `strategyType` 值为 1，表示定时推送任务主 ID； • 若 `strategyType` 值为 2，表示循环任务主 ID。
pushTime	Date	1630486972000	预计推送时间
pushTitle	String	测试标题	通知标题
pushContent	String	测试正文	通知内容
type	int	0	推送方式，其中： • 0 - 极简推送 • 1 - 模板推送 • 2 - 批量推送 • 3 - 群发推送
deliveryType	int	1	推送类型，其中： • 1 - Android • 2 - iOS • 3 - UserId
strategyType	int	1	推送策略类型，其中： • 1 - 定时 • 2 - 循环

executedStatus	int	0	是否执行，其中： • 0 - 未执行 • 1 - 已执行
createType	int	0	创建方式，其中： • 0 - API • 1 - 控制台
gmtCreate	Date	1629971346000	创建时间

使用示例

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // 创建 DefaultAcsClient 实例并初始化
        // 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
        // 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
        // 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
        // 建议先完成环境变量配置
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",           // 地域 ID
            accessKeyId,
            accessKeySecret);
        IAcsClient client = new DefaultAcsClient(profile);

        QueryPushSchedulerListRequest request = new QueryPushSchedulerListRequest();
        request.setAppId("ONEXPREF4F5C52081557");
        request.setWorkspaceId("default");
        request.setStartTime(Long.valueOf("1625068800000"));
        request.setEndTime(Long.valueOf("1630425600000"));
        request.setType(0);
        request.setUniqueId("49ec0ed5a2a642bcbe139a2d7a419d6d");
        request.setPageNumber(1);
        request.setPageSize(10);

        QueryPushSchedulerListResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

取消定时推送任务

取消未执行的定时推送任务（包括循环推送任务），支持批量取消。

请求参数

参数名称	类型	是否必填	示例	描述
appId	String	是	ONEX570DA89211721	mPaaS App ID
workspaceId	String	是	test	mPaaS 工作空间
type	int	否	0	定时推送任务 ID 类型，默认为 0。 • 0 - 主任务 ID，对应 `parentId` • 1 - 子任务 ID，对应 `uniqueId`
uniqueIds	String	是	714613eb,714613ec,714613ed	定时推送任务的唯一 ID，多个 ID 以“,”分隔，上限为 30 个。

返回参数

参数名称	类型	示例	描述
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	请求 ID
ResultCode	String	OK	请求结果码
ResultMessage	String	param is invalid	请求错误描述
ResultContent	String	{714613eb=1,714613ed=0}	取消结果，1 表示成功，0 表示失败。

使用示例

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // 创建 DefaultAcsClient 实例并初始化
        // 阿里云账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 用户进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 用户
        // 此处以把 AccessKey 和 AccessKeySecret 保存在环境变量为例说明。您也可以根据业务需要，保存到配置文件里
        // 强烈建议不要把 AccessKey 和 AccessKeySecret 保存到代码里，会存在密钥泄漏风险
        // 建议先完成环境变量配置
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // 地域 ID
            accessKeyId,
            accessKeySecret);
        IAcsClient client = new DefaultAcsClient(profile);

CancelPushSchedulerRequest request = new CancelPushSchedulerRequest();
        request.setAppId("ONEXPREF4F5C52081557");
        request.setWorkspaceId("default");
        request.setUniqueIds("49ec0ed5a2a642bcbe139a2d7a419d6d,
49ec0ed5a2a642bcbe139a2d7a419d6c");

        CancelPushSchedulerResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

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，参数值配置为铃声的路径，本参数仅对小米和苹果手机有效。

badge	应用图标角标，参数值配置为具体数值。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参数值中配置的数值。
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。参数值配置为 1。
badge_add_num	华为通道推送角标增加数。
badge_class	华为通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。
3031	REQUEST_SHOULD_BE_UTF8	请求体编码需为 UTF-8。
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，不支持自建通道群发。
3054	DELIVERYTOKEN_INVALID	自建通道 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。
8014	ACCOUNT_NO_PERMISSION	账号无权限。请检查 AK/SK 与 appId 和 workspaceId 是否一致。

9000	SYSTEM_ERROR	系统错误。

9.推送消息内容限制

为确保消息能够有效送达，在进行消息推送过程中，请参考各通道的推送消息内容限制创建消息推送任务。

Android 推送通道

推送通道	消息标题长度限制	消息内容长度限制
MPS 自建通道	无长度限制	无长度限制
小米	50 个字符，中英文均以 1 个字符计算	128 个字符，中英文均以 1 个字符计算
华为	40 个字符，中英文均以 1 个字符计算	1024 个字符，中英文均以 1 个字符计算
OPPO	32 个字符，中英文均以 1 个字符计算	200 个字符，中英文均以 1 个字符计算
vivo	20 个汉字（40 个英文字符），1 个汉字相当于 2 个英文字符	50 个汉字（100 个英文字符），1 个汉字相当于 2 个英文字符

说明

- 厂商通道在消息超过长度限制时会推送失败。
- 厂商通道在消息标题和内容为空时会推送失败。
- Android 通道（无论是厂商通道，还是 MPS 自建通道）推送的总消息体大小限制为 2KB。

iOS 推送通道

推送通道	消息标题长度限制	消息内容长度限制
APNs	20 个汉字（40 个英文字符），1 个汉字等于 2 个英文字符，超过部分会显示省略号。	• 消息中心最多显示 55 个汉字（110 个英文字符），1 个汉字等于 2 个英文字符，超过部分会显示省略号。 • 锁屏状态最多显示 55 个汉字（110 个英文字符），1 个汉字等于 2 个英文字符，超过部分会显示省略号。 • 顶部弹窗最多显示 31 个汉字（62 个英文字符），1 个汉字等于 2 个英文字符，超过部分会显示省略号。

说明

iOS 通道推送的总消息体大小限制为 2KB。

10.常见问题

本文汇总了接入及使用消息推送组件的过程中经常出现的一些问题及其解决方法。

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。

Android 相关问题

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" />
```

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。

RPC 调用问题

如果通过 RPC 请求进行资源调用的过程中出现异常，请参见 无线保镖结果码说明 或 网关结果码说明 进行排查。

11.参考

11.1. 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

消息推送服务仅支持 Apple Push Service 类型的证书。有关苹果证书类型及相关介绍，请参见 苹果证书类型。

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。

证书类型	用途
Apple Push Service	生产环境下的推送证书。用于在通知服务和 APNs 之间建立连接，以向 App 发送远程通知。
iOS Development	开发证书。用于真机调试和发布测试。

MAC Key Store

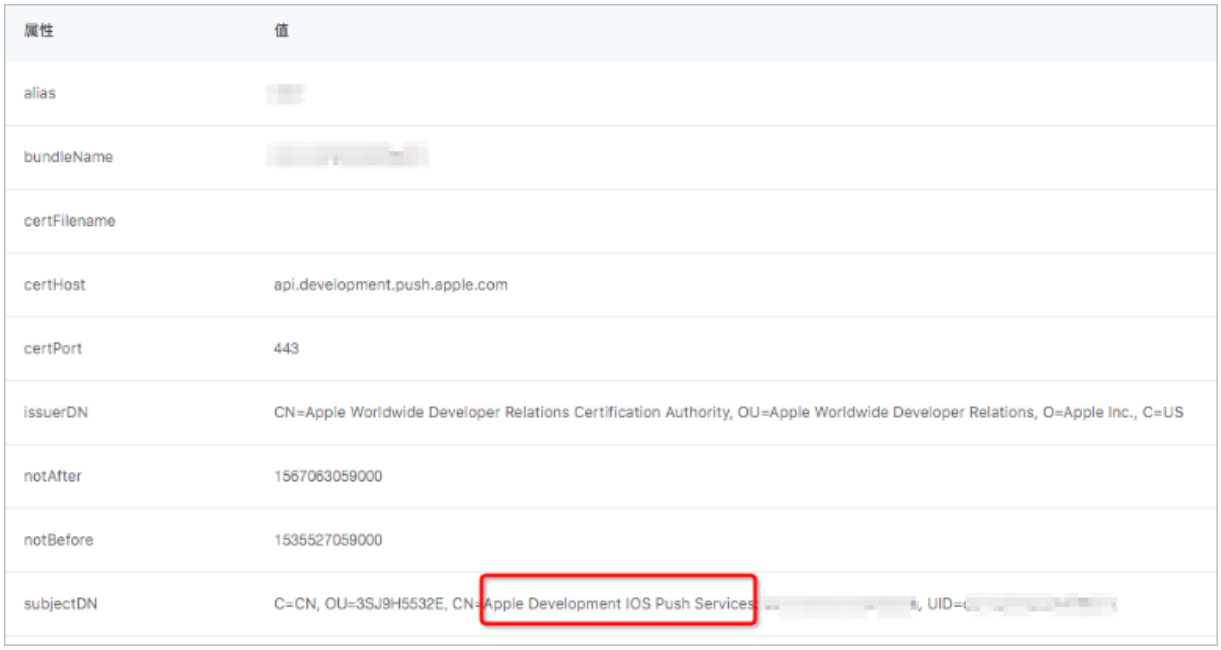
双击已有的 `.p12` 证书，将证书导入 MAC 钥匙串中，您将看到证书名称等信息：

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

如上图所示，`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。

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

如上图所示，`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

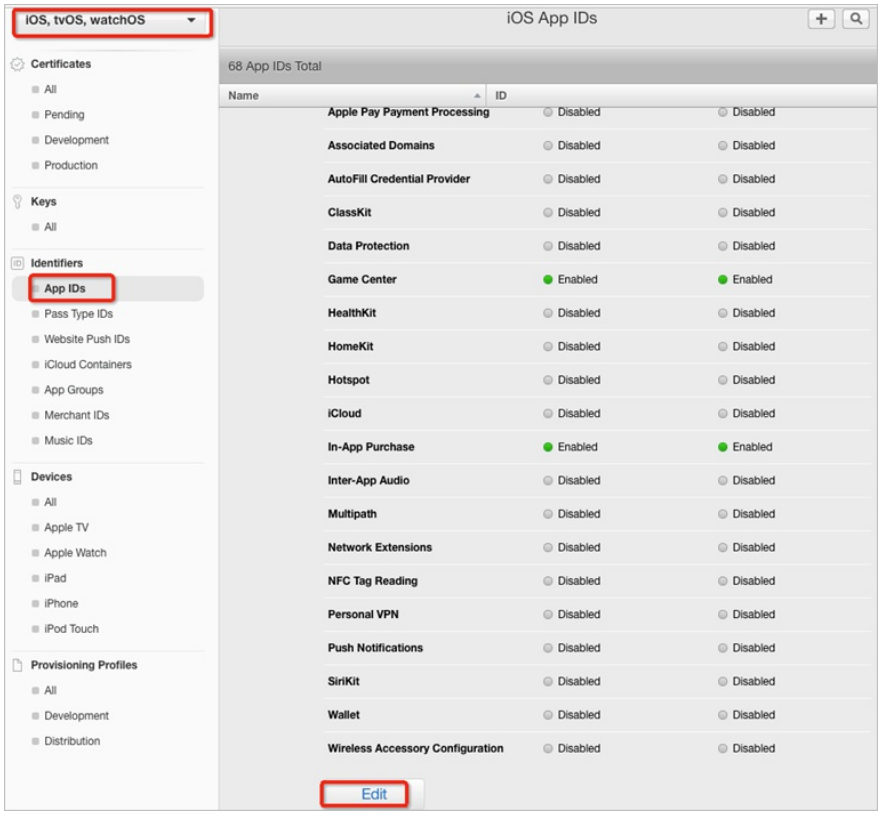
1. 在苹果开发平台，单击左侧导航栏 **App IDs**，然后单击右上角 + 按钮。
2. 填写基础信息。
   - **App ID Description** > **Name**
   - **App ID Suffix** > **Bundle ID**： Bundle ID 需要具备唯一性。
3. 勾选 **Push Notifications** 能力。
4. 单击 **Continue** 后，单击 **Register** 完成创建。

制作 .certSigningRequest 文件

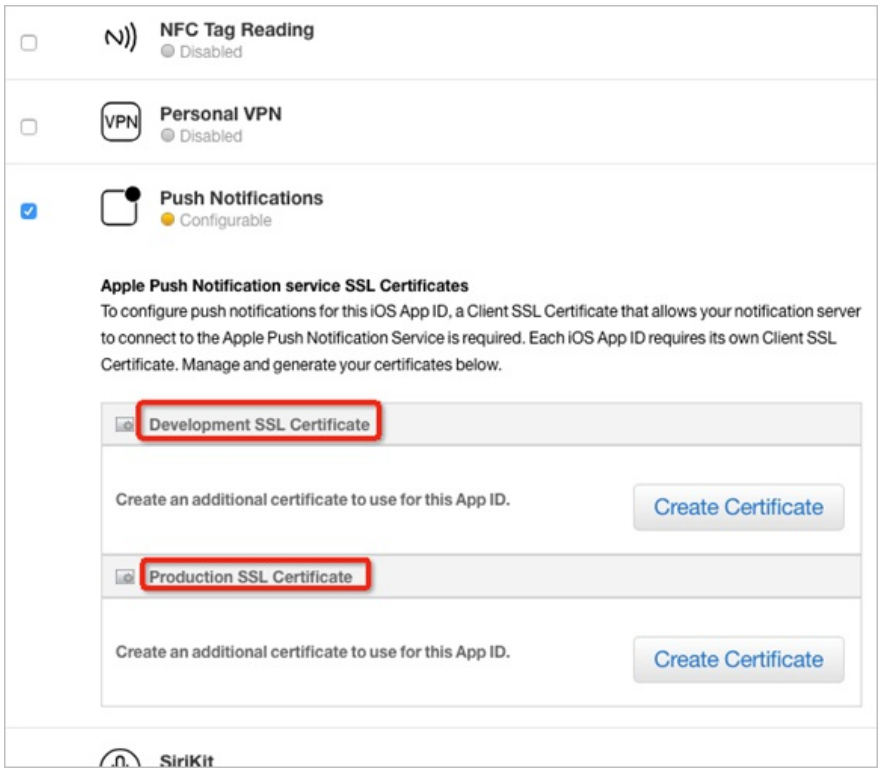
1. 进入 Mac 中的钥匙串服务。
2. 请求证书。选择 **钥匙串访问** > **证书助理** > **从证书颁发机构请求证书...** 。
3. 在打开的 **证书信息** 窗口中，根据实际情况填写邮件地址和常用名称等相关信息。
4. `.certSigningRequest` 文件制作成功。

创建证书

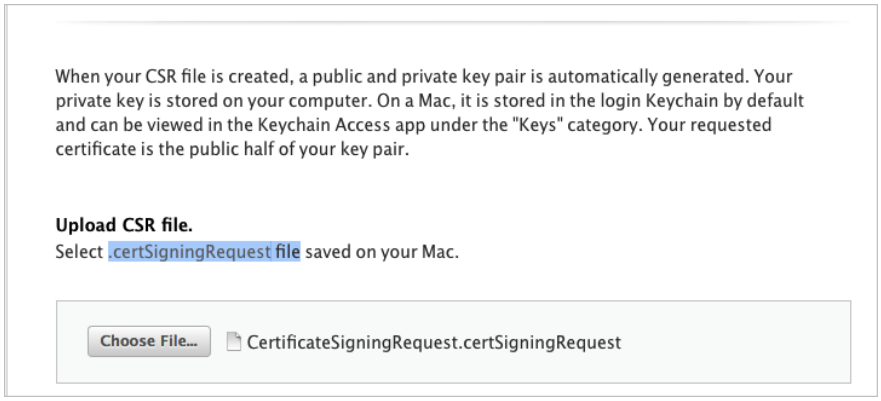
1. 在苹果 **App IDs** 页面中，选中自己的 iOS App ID，单击 **Edit**。



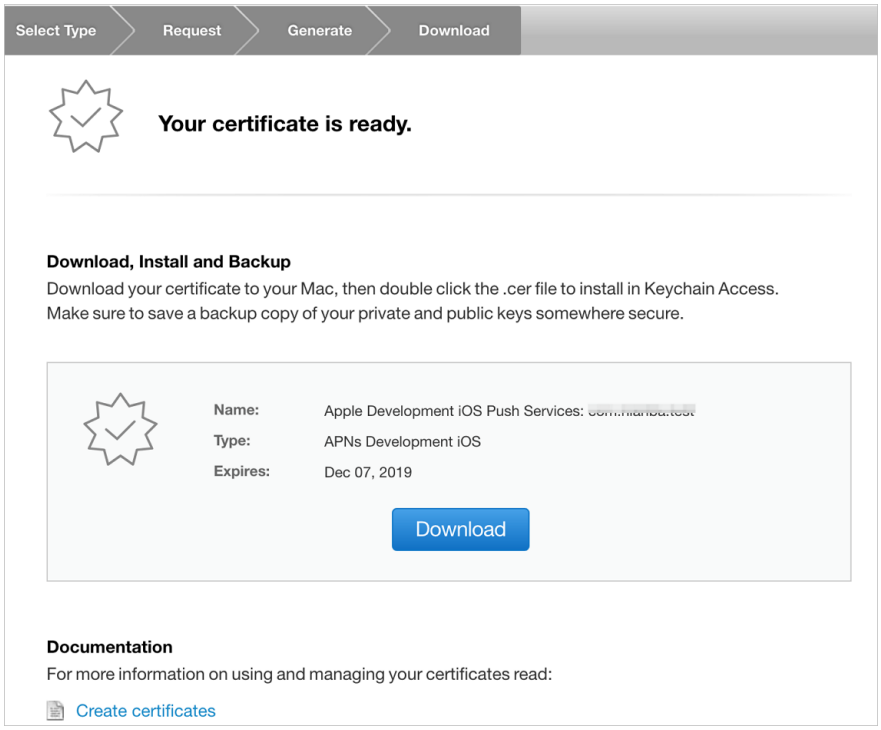
2. 单击 **Development SSL Certificate** 或 **Production SSL Certificate** 卡片中的 **Create Certificate**，开始创建开发或生产环境下的证书。



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



4. 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。



5. 将 `.cer` 文件转换成 `.p12` 文件。
   i. 双击 `.cer` 文件，将文件导入 Key Store。

ii. 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。

6. 至此您已获得了 `.p12` 证书，可以前往消息推送控制台的 **设置** > **渠道配置** 页面配置 iOS 推送证书。

11.2. 消息推送状态码

下面分别对公共的以及各推送通道对应的消息推送状态码进行说明。

- 公共消息推送状态码
- 苹果推送通道
- 华为推送通道
- 小米推送通道
- OPPO 推送通道
- vivo 推送通道
- FCM 推送通道

公共消息推送状态码

状态码	描述	解释
-1	WaitingForVerify	等待校验。
0	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）或等待发送确认。
1	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。
2	Acked	使用自建通道推送消息时，表示消息已成功推送至客户端；使用厂商通道推送消息时，表示已成功调用厂商推送网关。

99999999	NONE	未知状态

苹果推送通道

状态码	描述	解释
2001	PayloadEmpty	消息体为空。
2002	PayloadTooLarge	消息体太大。
2003	BadTopic	证书的 bundleid 错误。
2004	TopicDisallowed	证书的 bundleid 非法。
2005	BadMessageId	messageId 错误。
2006	BadExpirationDate	非法的有效期时间。
2007	BadPriority	非法的权重。
2008	MissingDeviceToken	缺少设备 token。
2009	BadDeviceToken	设备 token 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 token 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。
2010	DeviceTokenNotForTopic	设备 token 和证书不匹配。
2011	Unregistered	token 失效。
2013	BadCertificateEnvironment	非法的证书环境。
2014	BadCertificate	非法的证书。
2023	MissingTopic	未指定 Topic。

2024	ConnClosed	APNS 连接断开。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设备 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 配置 iOS 推送证书。
2025	ConnUnavailable	APNS 连接未完成。

华为推送通道

状态码	描述
100	无效未知参数。
101	无效的 API_KEY。
102	无效的 SESSION_KEY。
106	App 或者 Session 没有调用当前服务的权限。
107	client 和 secret 需要重新获取（如算法升级等）。
109	nsp_ts 偏差过大。
110	接口内部异常。
111	服务繁忙。
80000003	终端不在线。
80000004	应用已卸载。
80000005	响应超时。
80000006	无路由，终端未连接过 Push。

80000007	终端在其他大区，不在中国内地使用 Push。
80000008	路由不正确，可能终端切换 Push 服务器。
80100000	参数检查，部分参数错误。
80100002	不合法的 token 列表。
80100003	不合法的 payload。
80100004	不合法的超时时间。
80300002	无权限下发消息给参数中的 token 列表。
80300007	请求中所有的 token 都是非法 token。
81000001	内部错误。
80300008	认证类错误（请求消息体过大）。

小米推送通道

状态码	描述
10001	系统错误。
10002	服务暂停。
10003	远程服务错误。
10004	IP 限制不能请求该资源。
10005	该资源需要 appkey 拥有授权。
10008	参数错误。
10009	系统繁忙。
10012	非法请求。

10013	不合法的用户。
10014	应用的接口访问权限受限。
10017	参数值非法。
10018	请求长度超过限制。
10022	IP 请求频次超过上限。
10023	用户请求频次超过上限。
10024	用户请求特殊接口频次超过上限。
10026	应用被加入黑名单，不能调用 API。
10027	应用的 API 调用太频繁。
10029	不合法的设备。
21301	认证失败。
22000	非法应用。
22001	应用不存在。
22002	应用已经撤销。
22003	更新应用程序失败。
22004	缺少应用程序信息。
22005	应用程序名字不合法。
22006	应用程序 ID 不合法。

22007	应用程序 Key 不合法。
22008	应用程序 Secret 不合法。
22020	应用程序描述信息不合法。
22021	用户没有授权给应用程序。
22022	应用程序 package name 不合法。
22100	应用通知数据格式不合法。
22101	太多应用通知消息。
22102	发送应用通知消息失败。
22103	应用通知 ID 不合法。
20301	目标不合法。

OPPO 推送通道

状态码	描述	解释
-1	Service Currently Unavailable	服务不可用，此时请开发者稍后再试。
-2	Service in Flow Control	服务器流量控制。
11	Invalid Auth Token	不合法的 AuthToken。
13	App Call Limited	应用调用次数超限，包含调用频率超限。
14	Invalid App Key	无效的 AppKey 参数。
15	Missing App Key	缺少 AppKey 参数。
16	Invalid Signature	签名校验不通过，无效签名。

17	Missing Signature	签名校验不通过，缺少签名。
28	App Disabled	应用不可用。
29	Missing Auth Token	缺少 Auth Token 参数。
30	Api Permission Denied	该应用没有 API 推送的权限。
10000	Invalid RegistrationId	registration_id 格式不正确。

vivo 推送通道

状态码	描述
10000	权限认证失败。
10040	资源已达上限，稍后重试。
10050	alias 和 regId 不能都为空。
10055	title 不能为空。
10056	title 长度不能超过 40 个字符。
10058	content 长度不能超过 100 个字符。
10066	自定义 key 和 Value 键值对个数不能超过 10 个。
10067	自定义 key 和 value 键值对不合法。
10070	发送量总量超出限制。
10071	超出发送时间允许范围。
10072	推送速度过快，请稍后再试。
10101	消息内容审核不通过。

10102	vivo 服务器端未知异常。
10103	推送内容含敏感信息。
10110	请配置商业化消息发送频率。
10302	regId 不合法，regId 为无效的 regId，regId 可能已经失效。
10303	requestId 已存在。
10104	请发送正式信息。请检查 content，不要发送测试内容，正式信息发送的 content 里面不能是纯数字、纯英文、纯符号，符号加数字，不能包含“测试”字样、大括号、中括号。

更多 vivo 推送相关的错误码，请参见 vivo 推送错误码参考。

FCM 推送通道

状态码	描述	解释
90000002	nvalidRegistration	目标非法。
90000003	NotRegistered	目标未注册。
90000004	InvalidPackageName	包名非法。
90000007	MessageTooBig	消息体过大。
90000009	InvalidTtl	离线存活日期非法。
90000011	InternalServerError	FCM 服务异常。
90000401	Authentication	权限校验失败。