Ant Technology

Message Push Service
User Guide

Document Version: 20231226

蚂蚁集团
ANT GROUP

Legal disclaimer

Ant Group all rights reserved©2022.

Trademark statement

蚂蚁集团 ANT GROUP and other trademarks related to Ant Group are owned by Ant Group. The third-party registered trademarks involved in this document are owned by the right holder according to law.

Disclaimer

The content of this document may be changed due to product version upgrades, adjustments, or other reasons. Ant Group reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through channels authorized by Ant Group. You must pay attention to the version changes of this document as they occur and download and obtain the latest version of this document from Ant Group's authorized channels. Ant Group does not assume any responsibility for direct or indirect losses caused by improper use of documents.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.About Message Push Service

Message Push Service (MPS) provided by mPaaS is a professional mobile message push solution and supports various push types for different scenarios to cater to personalized push requirements. To improve the arrival rate of pushed messages, mPaaS integrates the push functions of Huawei, Xiaomi and other vendors in MPS. In addition to the capability of quickly pushing messages in the console, mPaaS provides server-side integration solutions. With these solutions, you can quickly integrate the function of pushing messages to mobile devices to keep interactions with app users, thereby effectively improving the user retention rate and user experience.

Features

You can initiate various types of message push through MPS. Both self-built and vendors' push channels are supported. In addition, messages can be pushed through the console or APIs. You can select push types, channels, and modes based on your requirements.

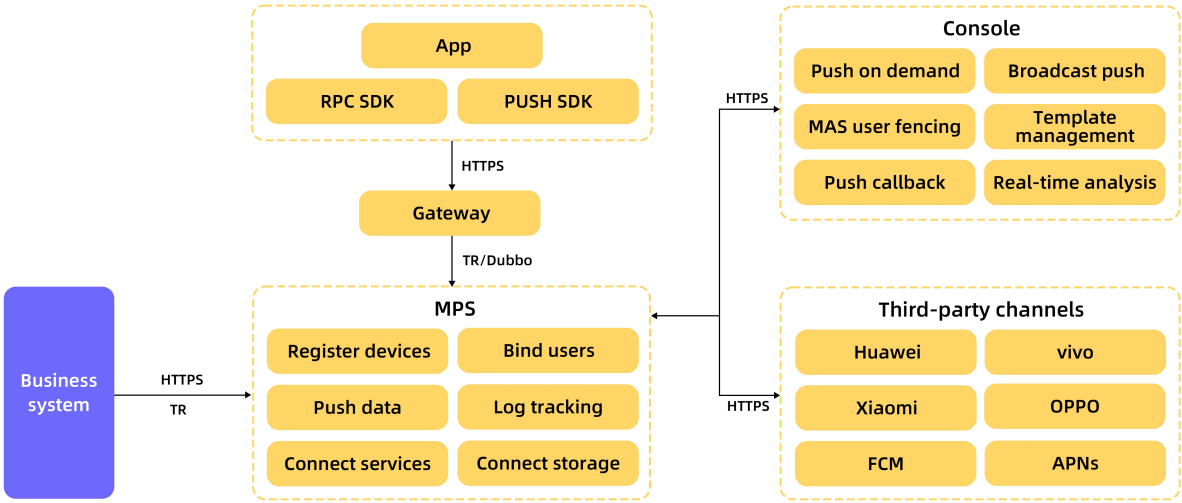
The core functions of MPS are described as follows:

- **Multiple push modes**: Messages can be precisely pushed to custom user groups, individual users, or all users through the MPS console or APIs.
- **Custom message validity period**: If a device is offline when a message is sent for the first time, the message can be resent when the device is connected or a user binding request is initiated within the validity period of the message.
- **Different types of push targets**: You can establish mapping between devices and login users to push messages by device or user ID.
- **Personalized message templates**: On the template management page, you can customize templates to meet your personalized push requirements.
- **Usage analysis**: Based on tracking logs reported by the client SDK, MPS collects and analyzes push data from various dimensions including platform, version, push channel, push type, and time, and generates analysis reports. You can view the statistics by minute or other granularity.
- **Push configuration**: On the push configuration page, you can configure a push certificate. For iOS devices, you can select an Apple APNs gateway based on your requirements.
- **Channel configuration**: You can configure third-party push channels to integrate the push functions provided by Huawei, Xiaomi, and other third-party vendors, thereby improving the arrival rate of pushed messages.
- **Key management**: All external APIs of MPS will sign the requests to ensure business security. On the key configuration page, you can configure keys based on your requirements. In addition, the message receipt function is provided for tracking the message delivery results.

Principle

In mPaaS, MPS is one of the core basic components that directly interact with clients. It transmits business data related to **message notifications** through TCP persistent connection channels or various phone vendors' push channels.

The client calls the Remote Procedure Call (RPC) gateway through mPaaS MGS for device registration, user binding, and third-party channel binding, thereby implementing message push by device and user. Client behavioral event tracking logs are collected and uploaded based on specifications. Based on the logs, the backend collects and analyzes push data in real time and generate statistical reports. MPS provides two push methods. You can either call APIs on your server based on the business logic to push personalized messages or directly push messages in the console. To improve the arrival rate of messages, MPS supports third-party push channels such as those provided by Huawei, Xiaomi, FCM, and APNs and keeps transparent to backend business systems. In this way, the business systems can focus on business function implementation, and don't need to pay attention to device models.



Advantages

MPS has the following advantages:

- **Quick and stable**: Messages are delivered quickly and arrive at targets stably.
- **Easy to access**: You can complete MPS access efficiently at a low cost.
- **Quantified push effect**: The push data statistics function is integrated to intelligently analyze the arrival rate and open rate of messages. This helps you clearly understand the push effects.
- **Precise personalized push**:
  - Personalized messages can be precisely pushed from various dimensions such as individual users and custom user groups.
  - A push console is provided to meet some simple push requirements. In addition, server-side integration solutions are provided to implement complex push requirements.
  - Message receipts are supported to track the message delivery results, improving the user retention rate and user activeness effectively.
  - Mapping between device IDs and app user IDs is established. The app user name can be directly used as the message recipient. In this way, messages can accurately arrive at any devices to which the user logs in.

Application scenarios

Typical application scenarios for MPS are as follows:

- **Marketing activities**

Push targeted messages to users, including marketing activities, business reminders, etc., to increase user stickiness. By calling the message push API, the app pushes targeted messages to target users to reach more users in a more active way, which attracts user, increases consumption, and improves the conversion effect of final marketing activities.

- **System notification**

  According to the business logic of the app server, specify the target user group, and directly push the message to the target device.

The following push modes are supported to accommodate different application scenarios:

- Simple Push: Quickly push messages to a single user or device with simple configuration.
- Template Push: Push messages to a single user or device, a message template can be specified, and the message body is obtained by replacing the template placeholder.
- Multiple Push: Push messages to a number of devices or users , you can specify a message template and set different placeholder variable values for different devices or users in the configuration file.
- Broadcast Push: Push to devices on the entire network, you can specify a message template, the message body is obtained by replacing the template placeholder.

2.Terminology

Terms are listed in an alphabetical order.

Ad-token

The unique identifier of Android device, mainly used in client SDK.

Apache Dubbo (Dubbo)

Dubbo is an open source distributed service framework developed by Alibaba, which provides high-performance RPC invocation, microservice governance and other capabilities for interface agents.

AppId

Application ID, generated when application is created.

Bind-info

The mapping relation between device token and user ID, in connection with two operations: binding and unbinding.

BroadcastPush

Used to push the same message to all devices. The message content is generated by replacing parameters in template.

Device Token

The unique identifier of Apple device, provided by iOS system.

Msgkey

Used to uniquely identify a message.

MultiplePush

Used to push customized message to a large number of targets. The message content is generated by using the same template and replacing parameters with different content according to different targets.

Push Cert

The certification, in iOS, used to establish connections with Apple's APNs servers.

SimplePush

Used to push the same message to individual target(s).

TaobaoRemoting (TR)

TaobaoRemoting (TR) framework refers to the underlying communication framework developed by Ant Group for RPC calls.

Target ID/Token

The target to push message to, which can be Ad-token of Android, Device Token of iOS or userId and is determined according to context.

TaskName

Each message push is identified as a task.

Template

The framework to generate a message, including attribute configuration of message, message content and placeholders which can be dynamically replaced.

Templatekv

"k" is the placeholder parameter in template; "v" is the parameter to be replaced.

Template Placeholder

The dynamically replaceable parameters in template configuration.

TemplatePush

Used to push the same message to individual target(s). The message content is generated by replacing parameters in template.

UserId/UsrId

Used to identify user, corresponding to device, normally used for binding.

3.Message push process

After integrating the Message Push Service (MPS), the client uses the mPaaS Mobile Gateway Service to call the Remote Procedure Call (RPC) gateway for device registration, user binding, and third-party channel binding, so as to implement message push by devices or users. The message push processes are different in different device platforms. The following sections introduce message push process through RPC on different device platforms.

Before acquainting yourself with the push process, you need to know some basic concepts involved in message push.

Basic concepts

- **Device ID (token)**: MPS assigns a unique identifier to each client device and determines the target of message push based on the identifier.
  - For Android devices, a persistent connection is established for message push.
  - For iOS devices, the Apple Push Notification service (APNs) is used for message push.
- **Push mode**: MPS provides the following push modes:
  - **Device ID-specific push**
  - **User ID-specific push**
  - **Broadcast push without specifying any identifiers**

    > **Note**
    >
    > No matter which mode is adopted, mapped device IDs will be eventually generated inside the system. User ID-specific message push offers convenience in interworking with your business systems. As user IDs are eventually mapped to device IDs, you must bind user IDs to device IDs. The recommended method is to bind the user ID to the corresponding device ID upon user login. When the user logs out, the binding relationship is removed.

- **Third-party push**: Third-party push refers to pushing by vendors, which can guarantee a high arrival rate. During the initialization process of calling the `init` method, the client applies for device IDs from both mPaaS and the third-party platform. Device IDs are then returned by mPaaS and the third-party platform in the callback.

  If you want to use a third-party push, you should call the `report` API to upload both mPaaS device ID and the third-party device ID to Mobile Push Core, and associate the two device IDs. After the above operation is completed, the third-party device ID can be truly used, otherwise the message push is a common mPaaS push.

Process

The MPS involves two backend systems:

- **Mobile Push Core (Pushcore)**: handles service logic and provides APIs to developers.
- **Mobile Push Gateway (Mcometgw)**: maintains persistent connections with Android devices.

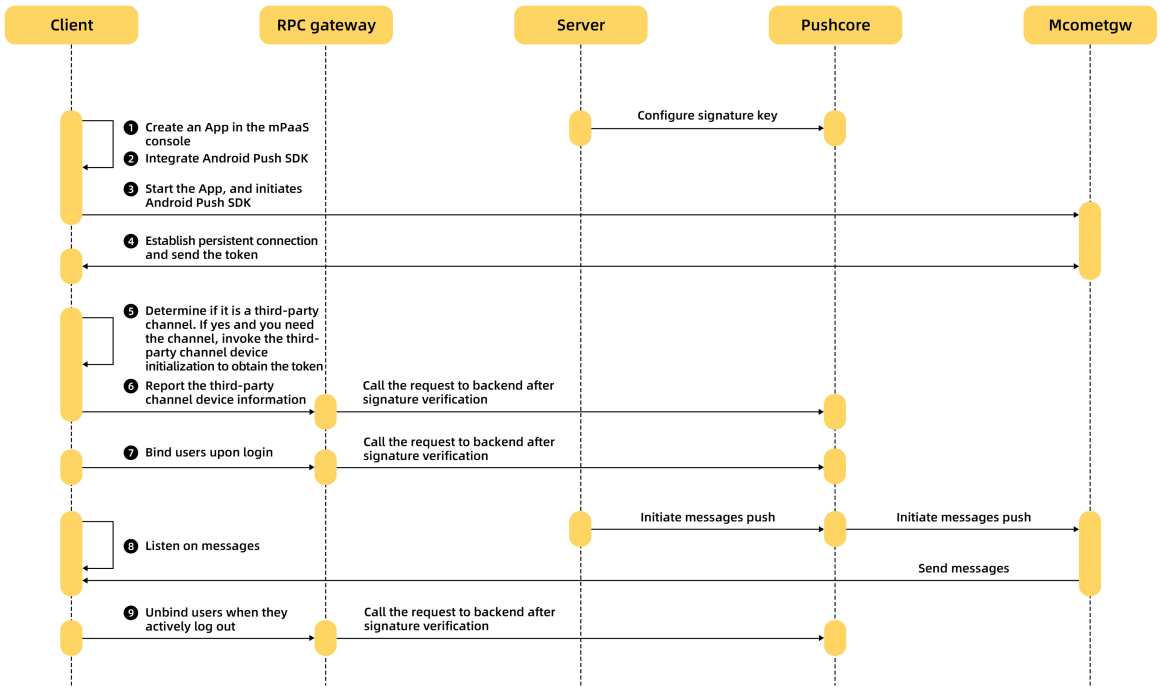
> **Note**
>
> For the devices configured with access to the third-party push platform, such as Xiaomi, Huawei or other vendors, the client also requests the device ID from the third-party platform. The third-party push channel is only available after you call the `report` API to bind the mPaaS device ID and third-party device ID returned. For general devices, only the device ID returned by mPaaS is used.

Learn about the process for integrating MPS on different device platforms:

- Android devices in Chinese mainland
- iOS devices and Android devices outside China

Android devices in Chinese mainland

The client uses RPC to directly interact with Mobile Push Core (Pushcore) through the RPC gateway. For Android devices in China, MPS provides a self-built gateway. The following figure shows the process.



Where,

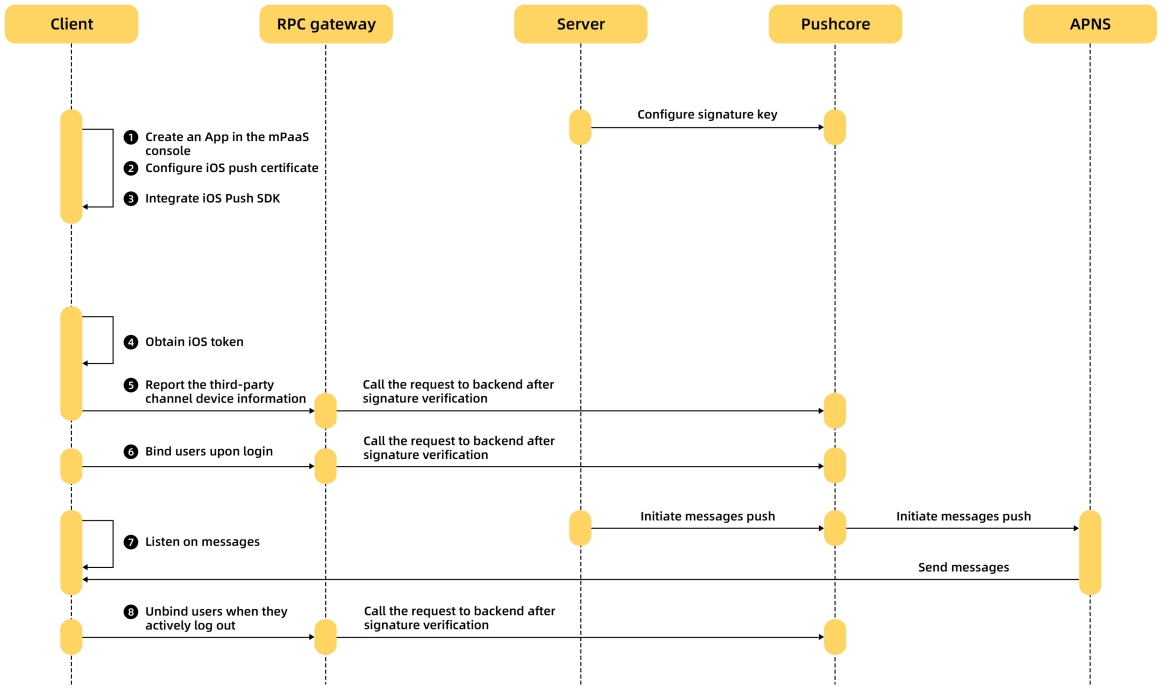
- When the app starts, the client establishes a persistent connection with Mcometgw. If the connection setup information of the client does not include the device identifier, Mcometgw issues the device identifier.
- If the user enables the MPS from a third-party channel such as Mi and Huawei, and the client is a third-party device, the third-party SDK initializes, establishes a persistent connection with the vendor’s push gateway, and obtains the device ID from the third-party channel.
- The app calls the device report RPC API and reports the third-party device information.
- The app user initiates a login request on the client.
- The server receives the user login request. When successfully logging in to the app, you can send a user-device binding request to Pushcore.
- The server initiates a push request.

- Pushcore receives the push request, and distinguishes the message push type.
  - If the message is pushed by device, Pushcore calls Mcometgw to send the message.
  - If the message is pushed by user, Pushcore obtains the device ID based on the user ID in the request and then calls Mcometgw to send the message.
- Mcometgw sends the message to the client.
- After the message is successfully sent, the client will confirm the receipt of the message with Mcometgw. If the user has configured a callback API, Pushcore will send a receipt to the server.
- When the user actively logs out of the app, the client calls the unbinding RPC API.

iOS devices and Android devices outside China

The push gateway for Android devices outside China uses Google Firebase Cloud Messaging (GCM/FCM) for Android, while the push gateway for iOS devices uses the Apple Push Notification service (APNs). The following takes the iOS device for example.

The client uses RPC to directly interact with Mobile Push Core (Pushcore) through the RPC gateway. The following figure shows the process.



Where,

- The client obtains the iOS device ID.
- The client calls the device report RPC API and reports the device ID to Pushcore through the RPC gateway.
- The app user initiates a login request on the client.
- When successfully logging in to the app, the user can call the binding RPC API to send a user-device binding request to the RPC gateway, which forwards the request to Pushcore.
- The server sends a push request to Pushcore.
- Pushcore receives the push request and distinguishes the message push type.
  - If the message is pushed by device, Pushcore directly calls the APNs to send the message.

- If the message is pushed by user, Pushcore obtains the device ID based on the user ID in the request and then calls the APNs to send the message.

- After the message is successfully sent, the client will confirm the receipt of the message with Pushcore. If the user has configured a callback API, Pushcore will send a receipt to the server.

4.Client-side development

4.1. Android

4.1.1. Quick start

This guide briefly describes how to fast integrate MPS to the Android client. You can integrate Message Push Service (MPS) through Native AAR or Portal & Bundle method.

The complete integration process mainly includes the following four steps:

1. Add SDK: Add the SDK dependencies and `AndroidManifest` configuration.
2. Initialize the SDK: Initialize the push service to establish persistent connection between the client and the mobile push gateway.
3. Create a service: Create a service to receive Android device IDs (Ad-tokens), so you can push messages based on device ID.
4. Bind user ID: Report user ID to the server to bind the user ID and the device ID, so you can push messages based on the user ID.

Prerequisites

- You have completed the basic configuration with reference to the general operations.
  - If you integrate MPS through Native AAR, ensure that you have added mPaaS to project.
  - If you integrate MPS in componentized integration mode (through Portal & Bundle projects), ensure that you have completed the componentized integration process.
- You have obtained the `.config` configuration file from the mPaaS console. For how to generate and download the configuration file, see Add configuration file to project.
- The `MPPushMsgServiceAdapter` method described in this guide only works in the baseline 10.1.68.32 or later version. If your current baseline version is lower than 10.1.68.32, please refer to mPaaS upgrade guide to upgrade the baseline version to 10.1.68.32.

> **Note**
>
> You can continue using the `AliPushRcvService` method in the earlier version. Click here to download the documentation about using `AliPushRcvService`.

Procedure

To use MPS, you should complete the following steps.

1. Add MPS SDK.

   Add the push SDK dependencies and `AndroidManifest` configuration.

   i. Add SDK dependencies. Choose an integration method, and complete the required steps accordingly.

   - Native AAR: Follow the instructions in AAR component management to install the **PUSH** component in the project through **Component management (AAR)**.
   - Componentized integration mode (Portal & Bundle): Install the **PUSH** component in the Portal and Bundle projects through **Component management (AAR)**. For more information, see Add component dependencies.

ii. Add `AndroidManifest` configuration. In the `AndroidManifest.xml` file, add the following content:

> **Note**
>
> If you add the SDK through Portal & Bundle, you should add the above content in the Portal project.

```
<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: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. In order to improve the arrival rate of messages, the push SDK has a built-in process keep-alive function, including the above-mentioned `com.alipay.pushsdk.BroadcastActionReceiver` to listen to the system broadcast to wake up the push process, and automatically restart after the process is recycled. When accessing, you can decide whether to enable these functions according to your own needs:

a. If you do not need to monitor the system startup broadcast, you can delete:

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

b. If you do not need to monitor the network switching broadcast, you can delete:

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

c. If you do not need to monitor the user wake-up broadcast, you can delete:

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

d. If you do not need to monitor the charging status change broadcast, you can delete:

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

e. If you do not need to monitor all the above broadcasts, you can set the `android:enabled` attribute of `com.alipay.pushsdk.BroadcastActionReceiver` to false .

f. If you do not need to automatically restart after the push process is recycled, you can add the following configuration under the `application` node:

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

> **Note**
> This configuration is only valid in baseline version 10.2.3.21 and above.

2. Initialize the SDK.

   Initialize the message push service to establish persistent connection between the client and the Mobile Push Gateway. The persistent connection is maintained by the SDK, and is regarded as the self-built channel.

   - Native AAR
     - If you have called the mPaaS initialization method in `Application`, you can call the following method behind `MP.init()`:

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

     - If you haven't called the mPaaS initialization method, you can call the following methods in `Application`:

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

   - Portal & Bundle

In `LauncherApplicationAgent` or `LauncherActivityAgent`, call the following method in `postInit`:

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

3. Create a service.

   Create a service to inherit `MPPushMsgServiceAdapter`, and override the `onTokenReceive` method to receive the device token delivered by the self-built channel.

```
public class MyPushMsgService extends MPPushMsgServiceAdapter {

    /**
     * Call back upon receiving the token delivered by the self-built channel
     *
     * @param token Device token delivered by the self-built channel
     */
    @Override
    protected void onTokenReceive(String token) {
        Log.d("Receive the token delivered by the self-built channel: " + token);
    }

}
```

   Declare the service in `AndroidManifest.xml`:

```
<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>
```

   After you complete this step, you can push messages by device on the console. The device ID required refers to the token.

4. Bind user ID.

   The user ID is customized by the developer. It can be the user ID of the real user system or other parameters that can form a mapping relationship with users, such as account and mobile phone number.

   After receiving the token, you can bind the token with the user ID:

```
String userId = "Custom userId";
ResultPbPB bindResult = MPPush.bind(context, userId, token);
Log.d("Bind userId " + (bindResult.success ? "Succeeded" : ("Error:" + bindResult.cod
e)));
```

   If you have already set the user ID by calling `MPLogger`, you don't have to pass the user ID when binding it. For example:

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

To unbind the user ID, for example, the user exits the app, you can call the following method:

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

After you complete this step, you can push messages by user on the console. The user ID required refers to the custom user ID.

Related operations

- To improve the message arrival rate, you are recommended to integrate the push channels provided by Android mobile phone vendors. Currently, MPS supports Huawei, Xiaomi, OPPO, and vivo push channels. For how to access the push channels of those vendors, see Integrate third-party channels.
- A notification will be sent automatically when the third-party channel receives the message. The users can click on the notification to open the Web page. If you need to jump to the in-app page according to a customized DeepLink, or customize the behavior after receiving the message, see Process notification click.

For more functions, see Advanced functions.

Sample code

Click here to download the sample code.

What to do next

After you successfully integrate MPS to your Android client, you can call the RESTful interface through the server. For more information, see Configure server > Push messages.

4.1.2. Process notification clicks

For the apps which have third-party channels integrated and run on the corresponding vendors' mobile phones, the server pushes messages through the third-party channels by default; for other apps, the server pushes messages through the self-built channel.

- When self-built channel receives a message, the push SDK automatically deliver a notification, and the user can click it to open the Web page.

> **Important**
>
> Message notification IDs used by the SDK start from 10000. Make sure that other notification IDs you use do not conflict with them.

- To jump to an in-app page, refer to Implement in-app page redirection.
- To process the received messages by yourself, refer to Implement custom message processing.
- After the third-party channel receives a message, the mobile system will automatically deliver a notification. Neither the push SDK nor developers can interfere. The push SDK can receive the message and open the Web page only when the user clicks the notification.
- To jump to an in-app page, refer to Implement in-app page redirection.
- To process the redirection upon click on message by yourself, refer to Implement custom message processing.

Prerequisites

- The `MPPushMsgServiceAdapter` method mentioned in this guide is only applicable for baseline 10.1.68.32 or later version. If your current baseline version is lower than 10.1.68.32, refer to mPaaS upgrade guide to upgrade the baseline.
- You can continue using the `AliPushRcvService` method in the earlier version. Click here to download the documentation about using `AliPushRcvService`.

Implement in-app page redirection

If you need to jump to a specific page in the app, you can fill in a custom DeepLink in the redirection address of the message, for example: `mpaas://navigate`, and set up a routing Activity in the app to receive the DeepLink and then distribute it to other pages.

You also need to add the corresponding `intent-filter` in `AndroidManifest.xml` for the routing Activity, for example:

```
<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>
```

Obtain URI and message from the routing Activity.

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

Implement custom message processing

To process the messages by yourself, you can override the `onMessageReceive` and `onChannelMessageClick` method of `MPPushMsgServiceAdapter`.

```
public class MyPushMsgService extends MPPushMsgServiceAdapter {

    /**
     * Callback after the self-built channel receives the message
     *
     * @param msg Message received
     * @return Whether the message has been processed:
     * If true is returned, the SDK will not process the message; the developer needs to process the message, including notification delivery and redirection upon click on notification.
     * If false is returned, the SDK will automatically deliver a notification and add the redirection upon click on notification.
     */
    @Override
    protected boolean onMessageReceive(MPPushMsg msg) {
        Log.d("Receive message through self-built channel:" + msg.toString());
        // Process the message by yourself, such as delivering custom notification
        return true;
    }

    /**
     * Callback after the notification is clicked. The messages delivered through the third-party channels are displayed on the notification bar.
     *
     * @param msg Message received
     * @return Whether the click on message has been processed:
     * If true is returned, the SDK will not process the click on notification delivered through the third-party channel; the developer needs to process the redirection upon click on notification.
     * If false is returned, the SDK will automatically process the redirection upon click on notification.

     */
    @Override
    protected boolean onChannelMessageClick(MPPushMsg msg) {
        Log.d("Message through the third-party channel is clicked:" + msg.toString());
        // Process the logic after the message is clicked by yourself
        return true;
    }

}
```

`MPPushMsg` encapsulates all the parameters of the message:

```
String id = msg.getId(); // Message ID
boolean isSilent = msg.isSilent(); // Whether to silence the message

String title = msg.getTitle(); // Message title
String content = msg.getContent(); // Message body

String action = msg.getAction(); // Redirection type, 0: URL, 1: Custom DeepLink
String url = msg.getUrl(); // Redirection address, URL or DeepLink

int pushStyle = msg.getPushStyle(); // Message type, 0: Normal message, 1: Big text, 2:
Rich text
String iconUrl = msg.getIconUrl(); // Icon of rich text message
String imageUrl = msg.getImageUrl(); // Large image of rich text message

String customId = msg.getCustomId(); // Custom message ID
String params = msg.getParams(); // Extension parameters
```

After you process the message, you may need to report the following message tracking, otherwise the MPS usage analysis module on the mPaaS console will not get accurate statistical data.

```
MPPush.reportPushOpen(msg); // Report that the message was opened
MPPush.reportPushIgnored(msg); // Report that the message was ignored
```

For the messages delivered through self-built channel:

- For silent messages, there is no need to report the message tracking.
- For non-silent messages, it is required to report the opened and ignored messages. You can listen the message opening and ignorance by calling the `SetContentIntent` and `setDeleteIntent` methods of `Notification.Builder` or through other effective methods.

For the messages delivered through the third-party channels, there is no need to report the message tracking by yourself.

4.1.3. Integrate third-party push channels

4.1.3.1. Integrate HUAWEI Push

This guide mainly introduces the process of integrating HUAWEI Push. The process falls into three steps:

1. Register HUAWEI Push
2. Integrate HUAWEI Push
3. Test HUAWEI Push

Register HUAWEI Push

Visit the Huawei Developer official website, register an account, and enable the push service. For more information, see Enable HUAWEI Push.

Integrate HUAWEI Push

MPS supports access to Huawei HMS2 and HMS5. However, you can only select HMS2 or HMS5 in the process of integrating Huawei push component.

- The HMS2 is obsolete. If you are integrating HUAWEI Push for the first time, you are recommended to integrate HMS5.
- If you have upgraded from HMS2 to HMS5, you need to delete the HMS2 `AndroidManifest` configuration listed below first.

The following describes the integration methods of Huawei HMS2 and HMS5 respectively.

HUAWEI Push - HMS5.x version

1. Add **Push - HMS5** component in the IDE plugin. The steps are roughly the same as adding MPS SDK, see Add SDK.

   > **Note**
   >
   > The Push - HMS5 component only contains adaptive codes, without HMS SDK. You can add the HMS SDK dependencies separately by following the steps below.

2. Download the configuration file `agconnect-services.json` in the Huawei App Service Console and place it under the `assets` directory of the main project.
3. Configure the Maven warehouse address of HMS SDK in the `build.gradle` file in the project root directory.

   ```
   allprojects {
       repositories {
           // Other repos are ignored
           maven {url 'https://developer.huawei.com/repo/'}
       }
   }
   ```

4. Add HMS SDK dependencies in the `build.gradle` file of the main project.

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

   - The HMS SDK version is updated frequently. For the latest version, refer to HMS SDK Version Change History.
   - The current adaptable version is 5.0.2.300. If you need to use a higher version, you can change it by yourself. Generally, the vendor's SDK is downward compatible. If it is not compatible, you can give feedback to adapt to the needs of the new version

5. To use obfuscation, you need to add the related obfuscation configurations.
   - No matter which integration method is used in integrating HUAWEI push SDK, you need to add Huawei push obfuscation rules.
   - If you integrated HUAWEI push SDK through Native AAR, you need to add mPaaS obfuscation rules.

HUAWEI Push - HMS2.x version

1. Add **Push - HMS2** component in the IDE plugin. The steps are roughly the same as adding MPS SDK, see Add SDK.

   The current HMS2 SDK version is V2.5.2.201.

2. Configure `AndroidManifest.xml`, and replace the value of `com.huawei.hms.client.appid`. If you integrate the MiPush SDK through Portal & Bundle projects, please configure the `AndroidManifest.xml` in the Portal project.

```
 <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>
  <!--To prevent dex crashing in an earlier version, dynamically enable provider, and
set "enabled" to 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>
        <!-- Replace the "appid" of value with the actual app ID in the service deta
ils of the app on Huawei Developer. Keep the slash (\) and space in the value. -->
   <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>
                <!-- Receive the notification bar message sent by the channel. It is
compatible with earlier versions of 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>
                <!-- Required, used for receiving TOKEN. -->
                <action android:name="com.huawei.android.push.intent.REGISTRATION" />
                <!-- Required, used for receiving messages -->
                <action android:name="com.huawei.android.push.intent.RECEIVE" />
                <!-- Optional, used for triggering onEvent callback upon a click on t
he notification bar or the button on the notification bar -->
                <action android:name="com.huawei.android.push.intent.CLICK" />
                <!-- Optional, used for checking whether the PUSH channel is
connected. You do not need to configure this parameter if access check is not require
d -->
                <action android:name="com.huawei.intent.action.PUSH_STATE" />
            </intent-filter>
    </receiver>
```

3. To use obfuscation, you need to add the related obfuscation configurations.
   - If you integrated HUAWEI push SDK through Native AAR, you need to add mPaaS obfuscation rules.
   - If you integrated HUAWEI push SDK through other methods, you don't have to add any obfuscation configuration.

Test HUAWEI Push

1. After integrating HUAWEI Push, you can start the app on your Huawei phone, and the MPS SDK will automatically get the HUAWEI Push token and report it. Before you start the app, make sure that you have called the initialization method, see Message push initialization.
2. Push test messages when the app process is killed:
   - If you can still receive the messages, it means that the app has successfully integrated HUAWEI Push.
   - If you cannot receive the messages, you can follow the steps below for troubleshooting.

Troubleshooting

1. Check if the Huawei configuration and parameters are consistent with that in the Huawei push backend.
   - For HMS2, check if `AndroidManifest.xml` has related configurations added, and check if `com.huawei.hms.client.appid` is same as that in Huawei push backend.
   - For HMS5, check if `agconnect-services.json` exists, and the file is correctly placed.
2. Check if HUAWEI Push is enabled in the mPaaS console (see Configure HUAWEI Push), and the relevant configurations are consistent with that on Huawei push backend.
3. View the logs in Logcat to troubleshoot:
   i. Select the `push` process, filter `mPush.PushProxyFactory`, and check if the following log exists:

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

   If not, it means that there may be a problem with the Push - HMS2 or Push - HMS5 component. Check if the component has been correctly added.

   ii. Select the main process, filter `mHMS`, and check if the channel token of HUAWEI Push has been obtained. If the following log `get token failed` appears:

   It means the system failed to get the channel token, see HUAWEI Push Result Codes for the failure reason.

   iii. Select the main process, filter `report channel token`, check if the channel token of HUAWEI Push has been successfully reported. If the following log appears:

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

   It means the channel token reporting failed, you need to check if the `base64Code` in the mPaaS configuration file has a value, and check if the apk signature that you uploaded when obtaining the configuration file is consistent with the app.

Other questions

Does MPS has any version restrictions on EMUI and Huawei mobile services

There are version restrictions on Emotion UI (EMUI for short, it is an Android-based emotional operating system developed by Huawei) and Huawei mobile services.

For detailed version requirements, see Conditions for devices to receive Huawei notifications.

Failed to print logs for Huawei mobile phones

On the dialing interface of the Huawei mobile phone, enter **##2846579##** to enter **Project** menu > **Background settings** > **LOG settings** and select **AP Logs**. After the phone restarts, Logcat will start to take effect.

4.1.3.2. Integrate OPPO Push

This guide mainly introduces the process of integrating OPPO Push. The process falls into three steps:

1. Register OPPO Push
2. Integrate OPPO Push
3. Test OPPO Push

Register OPPO Push

Register an account on the OPPO Developers Platform and request to Integrate the push service with reference to Opening OPPO PUSH Service Guideline.

Integrate OPPO Push

1. Add **Push - OPPO** component in the IDE plugin. The steps are roughly the same as adding MPS SDK, see Add SDK.

   The component only contains adaptive codes, without OPPO Push SDK. You can add the OPPO Push SDK dependencies separately by following the steps below.

2. Download OPPO Push SDK from OPPO PUSH Client SDK Interface Document and integrate the SDK to the main project.

   The current adaptable version is V2.1.0. If you need to use a higher version, you can change it by yourself. Generally, the vendor's SDK is downward compatible. If it is not compatible, you can submit a ticket about the adaption issue.

3. Configure `AndroidManifest.xml`, and replace the value of `com.oppo.push.app_key`. If you integrate the MiPush SDK through Portal & Bundle projects, please configure the `AndroidManifest.xml` in the Portal project.

```
    <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="Obtained from the OPPO Developers platform"
            />
        <meta-data
            android:name="com.oppo.push.app_secret"
            android:value="Obtained from the OPPO Developers platform"
            />
    </application>
```

4. To use obfuscation, you need to add the related obfuscation configurations.
   - No matter which integration method is used in integrating OPPO push SDK, you need to add OPPO push obfuscation rules.
   - If you integrated OPPO push SDK through Native AAR, you need to add mPaaS obfuscation rules..

Test OPPO Push

1. After integrating OPPO Push, you can start the app on your OPPO phone, and the MPS SDK will automatically get the OPPO Push token and report it. Before you start the app, make sure that you have called the initialization method, see Message push initialization.
2. Push test messages when the app process is killed:
   - If you can still receive the messages, it means that the app has successfully integrated OPPO Push.
   - If you cannot receive the messages, you can follow the steps below for troubleshooting.

Troubleshooting

1. Check if `AndroidManifest.xml` has related configurations added, and check if the values of `com.oppo.push.app_key` and `com.oppo.push.app_secret` are the same as that on OPPO Developers Platform.
2. Check if OPPO Push is enabled in the mPaaS console (see Configure OPPO Push), and the relevant configurations are consistent with that on OPPO Developers Platform.
3. View the logs in Logcat to troubleshoot:
   i. Select the `push` process, filter `mPush.PushProxyFactory`, and check if the following log exists:

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

   If not, it means that there may be a problem with the Push - OPPO component. Check if the component has been correctly added.

   ii. Select the `push` process, filter `mOPPO`, and check if the channel token of OPPO Push has been obtained. If the following log (neither OPPO onRegister error nor responseCode is 0) appears:

   It means the OPPO Push registration failed. See OPPO Push Error Codes for the failure reason.

   iii. Select the main process, filter `report channel token`, check if the channel token of OPPO Push has been successfully reported. If the following log appears:

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

   It means the channel token reporting failed, you need to check if the `base64Code` in the mPaaS configuration file has a value, and check if the apk signature that you uploaded when obtaining the configuration file is consistent with the app.

   iv. Select the `push` process, filter `mcssdk`, and view the internal logs of OPPO Push.
4. If the above steps do not resolve the issue, please search for the group number 31591197 with DingTalk to join DingTalk group for further communication.

Other questions

Mobile models and OS versions supported by OPPO Push

Currently, OPPO phone models running ColorOS 3.1 and newer systems, OnePlus 5/5T and newer phone models, and all realme phone models are supported by OPPO Push.

ColorOS is a Android-based highly-customized, efficient, intelligent, and richly-designed mobile operating system developed by OPPO.

4.1.3.3. Integrate vivo Push

This guide mainly introduces the process of integrating vivo Push. The process falls into three steps:

1. Register vivo Push
2. Integrate vivo Push
3. Test vivo Push

Register vivo Push

Register an account on the vivo Developers Platform and request to integrate the push service with reference to vivo Push Platform Operation Guide.

Integrate vivo Push

1. Add **Push - vivo** component in the IDE plugin. The steps are roughly the same as adding MPS SDK, see Add SDK.

   The component has integrated the vivo Push SDK V2.3.4. You can upgrade the vivo Push SDK on demand. Generally, the vendor's SDK is downward compatible. If it is not compatible, you can submit a ticket about the adaption issue.

2. Configure `AndroidManifest.xml`, and replace the values of `com.vivo.push.api_key` and `com.vivo.push.app_id`. If you integrate the vivo Push SDK through Portal & Bundle projects, please configure the `AndroidManifest.xml` in the Portal project.

```
<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="Provided by vivo Developers Platform" />
    <meta-data
        android:name="com.vivo.push.app_id"
        android:value="Provided by vivo Developers Platform" />
</application>
```

3. To use obfuscation, you need to add the related obfuscation configurations.
   - No matter which integration method is used in integrating vivo push SDK, you need to add vivo push obfuscation rules.
   - If you integrated vivo push SDK through Native AAR, you need to add mPaaS obfuscation rules.

Test vivo Push

1. After integrating vivo Push, you can start the app on your vivo phone, and the MPS SDK will automatically get the OPPO Push token and report it. Before you start the app, make sure that you have called the initialization method, see Message push initialization.
2. Push test messages when the app process is killed:
   - If you can still receive the messages, it means that the app has successfully integrated vivo Push.
   - If you cannot receive the messages, you can follow the steps below for troubleshooting.

Troubleshooting

1. Check if `AndroidManifest.xml` has related configurations added, and check if the values of `com.vivo.push.api_key` and `com.vivo.push.app_id` are the same as that on vivo Developers Platform.
2. Check if vivo Push is enabled in the mPaaS console (see Configure vivo Push), and the relevant configurations are consistent with that on vivo Developers Platform.
3. View the logs in Logcat to troubleshoot:

i. Select the `push` process, filter `mPush.PushProxyFactory`, and check if the following log exists:

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

If not, it means that there may be a problem with the Push - vivo component. Check if the component has been correctly added.

ii. Select the `push` process, filter `mVIVO`, and check if the channel token of vivo Push has been obtained. If the following log `“fail to turn on vivo push”` appears:

It means the vivo Push registration failed, see vivo Push Error Codes.

iii. Select the main process, filter `report channel token`, check if the channel token of vivo Push has been successfully reported. If the following log appears:

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

It means the channel token reporting failed, you need to check if the `base64Code` in the mPaaS configuration file has a value, and check if the apk signature that you uploaded when obtaining the configuration file is consistent with the app.

4. If the above steps do not resolve the issue, please search for the group number 31591197 with DingTalk to join DingTalk group for further communication.

FAQ

Models and OS versions supported by vivo Push

The models and earlier system versions supported by vivo Push are listed in the following table. For other questions on vivo Push, see vivo Push FAQs.

Device model	Android version	Version for system test	Minimum version supported
Android 9.0 and later versions are supported by default			
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 Standard	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 Z1Youth	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	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

4.1.3.4. Integrate MiPush

This guide mainly introduces the process of integrating MiPush. The process falls into three steps:

1. Register MiPush
2. Integrate MiPush
3. Test MiPush

Register MiPush

Complete MiPush registration with reference to the following official Xiaomi documents:

- Register a Xiaomi developer account
- Enable MiPush

Integrate MiPush

1. Add **Push - Xiaomi** component in the IDE plugin. The steps are roughly the same as adding MPS SDK, see Add SDK. Currently, the built-in MiPush SDK is V4.0.2. You can see Release notes to learn the historical versions.

2. Configure `AndroidManifest.xml`, and replace the values of `xiaomi_appid` and `xiaomi_appkey`. If you integrate the MiPush SDK through Portal & Bundle projects, please configure the `AndroidManifest.xml` in the Portal project.

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

    <!-- Keep the slash (\) and space in the value -->
    <meta-data
        android:name="xiaomi_appid"
        android:value="\ 2xxxxxxxxxxxxxx" />
    <!-- Keep the slash (\) and space in the value -->
    <meta-data
        android:name="xiaomi_appkey"
        android:value="\ 5xxxxxxxxxxxxxx" />

</application>
```

Test MiPush

1. After integrating MiPush, you can start the app on your Xiaomi phone, and the MPS SDK will automatically get the MiPush token and report it. Before you start the app, make sure that you have called the initialization method, see Message push initialization.
2. Push test messages when the app process is killed:
   - If you can still receive the messages, it means that the app has successfully integrated MiPush.
   - If you cannot receive the messages, you can follow the steps below for troubleshooting.

Troubleshooting

1. Check if `AndroidManifest.xml` has been configured, and the values of `xiaomi_appid` and `xiaomi_appkey` in the file are consistent with that on Mi Developer Platform.
2. Check if MiPush is enabled in the mPaaS console (see Channel configuration), and the relevant configurations are consistent with that on Mi Developer Platform.
3. View the logs in Logcat to troubleshoot:
   i. Select the `push` process, filter `mPush.PushProxyFactory`, and check if the following log exists:

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

   If not, it means that there may be a problem with the Push - Xiaomi component. Check if the component has been correctly added.

   ii. Select the `push`, filter `mMi`, and check if the MiPush channel token has been obtained.

   If the following log (register_fail) appears, it means the MiPush registration failed. See MiPush error codes for the failure reason (`reason`). If the value of `reason` is UNKNOWN, it is generally due to incorrect `xiaomi_appid` or `xiaomi_appkey`. To learn about the result codes (`resultCode`), see MiPush server error codes.

iii. Select the main process, filter `report channel token`, check if the MiPush channel token has been successfully reported. If the following log appears:

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

It means the channel token reporting failed, you need to check if the `base64Code` in the mPaaS configuration file has a value, and check if the apk signature that you uploaded when obtaining the configuration file is consistent with the app.

4.1.3.5. Integrate FCM push channel

MPS supports integrating the Firebase Cloud Messaging (FCM) push channel to satisfy the message push requirements on overseas Android devices.

The following sections describe how to integrate the FCM push channel.

Prerequisites

Before you integrate FCM, ensure that the following conditions are met:

- Adopt native AAR integration mode. Portal & Bundle integration modes don't work for FCM.
- Gradle must be 4.1 or later versions.
- AndroidX is used.
- `com.android.tools.build:gradle` must be 3.2.1 or a later version.
- `compileSdkVersion` must be 28 or a later version.

Integrate FCM SDK

Perform the following steps:

1. Add your app in the Firebase console.

   Log on to the Firebase console and register your app. See Firebase documentation.

2. Add the Firebase Android configuration file to your app.

   Download the configuration file `google-services.json` and move the file to the main module of your project.

3. Add the Google service plug-in to the `buildScript` dependency in the root-level `build.gradle` file.

```
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. Apply the Google service plug-in in the `build.gradle` file of the main module.

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

android {
  // ...
}
```

5. Add the FCM SDK dependency to the `build.gradle` file of the main module.

```
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 libra
ries
    // When using the BoM, you don't specify versions in Firebase library
dependencies
    implementation 'com.google.firebase:firebase-messaging'
    implementation 'com.google.firebase:firebase-analytics'
}
```

Integrate mPaaS

Perform the following steps:

1. Add the FCM Adapter dependency to the `build.gradle` file of the main module.

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

2. Integrate the MPS SDK, with reference to the requirements on mPaaS baseline:
   - For `com.mpaas.push:fcm-adapter:0.0.2`, the baseline must be 10.1.68.34 or later version.
   - For `com.mpaas.push:fcm-adapter:0.0.1`, the baseline must be 10.1.68.19 or later version.
3. Receive push messages.

   Due to the features of FCM SDK, the messages pushed through the FCM channel may not always be received by the client through the FCM channel, but may be received through the self-built channel. The specific rules are:
   - If the app is in frontend, the messages are passed through to the app by FCM, and the app will receive the message through the self-built channel.
   - If the app is in backend or the app is killed, the messages are sent through FCM channel, and are displayed on the notification bar.
4. (Optional) You can register a message receiver to obtain an error message when the FCM initialization fails. For details, see Error codes.

   Refer to the following sample code:

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

```
package com.mpaas.demo.push;

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

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

4.1.4. Vendor message classification

Major vendors have successively limited the quantity and frequency of push messages by category to improve the end user experience and build a sound notification ecosystem.

Message Push Service (MPS) supports the vivo message classification feature, which allows you to specify the message category while calling the push API. To integrate this feature, refer to instructions in the following section.

About vivo message classification

The vivo message classification feature classifies push messages into system messages and operational messages. The classification principle is as follows:

- **System messages**: The user is expected to receive such messages timely. If the user misses the messages, adverse effects may occur.
- **Operational messages**: The user is not expected to receive such messages and therefore is less concerned with the messages.

Message category	Content allowed	Quantity	Quantity increase request	Push limit	Message notification display
System messages	The following eight message types are supported. For more information, see vivo message classification scenarios. • IM • Email • Reminder set by the user • Logistics • Order • To-do • Finance • Feature reminder	Number of SDK subscriptions × 2	Request an increase by email free of charge. For the application template and requirements, see the following section.	The number of messages received by a single user per app and per day is not limited.	The message is displayed regardless of whether the app is alive.
Operational messages	Notifications such as content recommendations, activity recommendations, and social contact dynamics that cannot be pushed as system messages. For more information, see vivo message classification scenarios.	Number of SDK subscriptions × 2	To request an increase, contact vivo business.	Up to five messages can be received by a single user per app and per day.	The message is collapsed or displayed if the app is not alive or alive.

> **Important**
>
> - Funtouch OS _10 and later versions do not provide a message box. The message is displayed in the notification bar if the app is not alive.
> - The quantity of both system messages and operational messages changes automatically with the number of SDK subscriptions. To increase the quantity of system messages, see Request an increase in vivo system messages.

Request an increase in vivo system messages

You can only pass in a parameter by calling a server-side API as the MPS console does not provide such an API. To be specific, add the request field `classification` to the push API. "0" indicates operational messages, and "1" indicates system messages. If not specified, the default "0" is used. For more information, see vivo server-side API document.

By default, the quantity of system messages is twice the number of SDK subscriptions. To request an increase in system messages, send an application email based on the following template to push@vivo.com.

```
Subject: Request an increase in system messages for the xxx app
Body: ...
App name: ...
App ID: ...
Package name: ...
App introduction: ...
The required quantity (unit: 10,000) of system messages: ...
Specific push scenarios: such as user personal chat and merchant chat
```

Use vivo system messages

Currently, vivo message notifications can only be delivered through server-side APIs rather than the MPS console.

> **Important**
>
> - The vivo push platform conducts daily inspections of system messages based on the message classification criteria. It checks whether developers send operational messages using system message channels. If so, the platform imposes penalties in light of the degree and frequency of violations. In case of severe violations, the message push feature may be disabled.
> - Developers must refer to vivo message classification and operate strictly in accordance with the platform message classification.

4.1.5. Advanced functions

After integrating the push SDK, you can also perform the following client configuration:

- Clear notification badges
- Report third-party channel tokens
- Customize NotificationChannel

Prerequisites

- The `MPPushMsgServiceAdapter` method mentioned in this guide is only applicable for baseline 10.1.68.32 or later version. If your current baseline version is lower than 10.1.68.32, refer to mPaaS upgrade guide to upgrade the baseline.
- You can continue using the `AliPushRcvService` method in the earlier version. Click here to download the documentation about using `AliPushRcvService`.

Clear notification badges

For third-party channels, you can enable that a badge appears on the app icon when users receive messages. Currently, the push SDK only supports automatic badge clearance for Huawei channel.

- To clear the app badge automatically when users click the notification, perform the following settings:

```
// Set whether to clear automatically
boolean autoClear = true;
MPPush.setBadgeAutoClearEnabled(context, autoClear);
// Set the Activity name of the app entrance. If not set, badges cannot be cleared
String activityName = "com.mpaas.demo.push.LauncherActivity";
MPPush.setBadgeActivityClassName(context, activityName);
```

- In the scenario that the badge cannot be cleared automatically, for example, the user actively clicks the app icon to enter the app, you can call the following method in `Application` to actively clear the badge:

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

Report third-party channel tokens

With the third-party channels integrated, the push SDK will receive the third-party channel token after initialization, automatically bind it with the self-built channel token, and report the tokens.

If necessary, you can override the `onChannelTokenReceive` and `onChannelTokenReport` methods of `MPPushMsgServiceAdapter` to listen the delivery and report of third-party channel tokens.

```
public class MyPushMsgService extends MPPushMsgServiceAdapter {

    /**
     * Callback for the reception of third-party channel token
     *
     * @param channelToken Third-party channel token
     * @param channel      Third-party channel type
     */
    @Override
    protected void onChannelTokenReceive(String channelToken, PushOsType channel) {
        Log.d("Receive third-party channel token: " + channelToken);
        Log.d("Vendor: " + channel.getName());
    }

    /**
     * Callback for the report of third-party channel token
     *
     * @param result Report result
     */
    @Override
    protected void onChannelTokenReport(ResultBean result) {
        Log.d("Report third-party channel token " + (result.success ? "Succeeded" : ("E
rror:" + result.code)));
    }

    /**
     * Whether to report the third-party channel token automatically
     *
     * @return If false is returned, you can report the token by yourself
     */
    @Override
    protected boolean shouldReportChannelToken() {
        return super.shouldReportChannelToken();
    }

}
```

To bind the tokens and report them by yourself, you can override the `shouldReportChannelToken` method and return `false`. You must call the following method after receiving the two tokens.

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

Customize NotificationChannel

To customize the name and description of `NotificationChannel` for the self-built channel, you can add the following content in the `AndroidManifest.xml` file:

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

4.2. iOS

This guide introduces how to integrate MPS to iOS client. You can integrate MPS to iOS client based on native project with CocoaPods.

> **Note**
>
> Since June 28, 2020, mPaaS has stopped support for the baseline 10.1.32. Please use 10.1.68 or 10.1.60 instead. For how to upgrade the baseline from version 10.1.32 to 10.1.68 or 10.1.60, see mPaaS 10.1.68 upgrade guide or mPaaS 10.1.60 upgrade guide.

Prerequisites

You have integrated your project to mPaaS. For more information, refer to Integrate based on native framework and using Cocoapods.

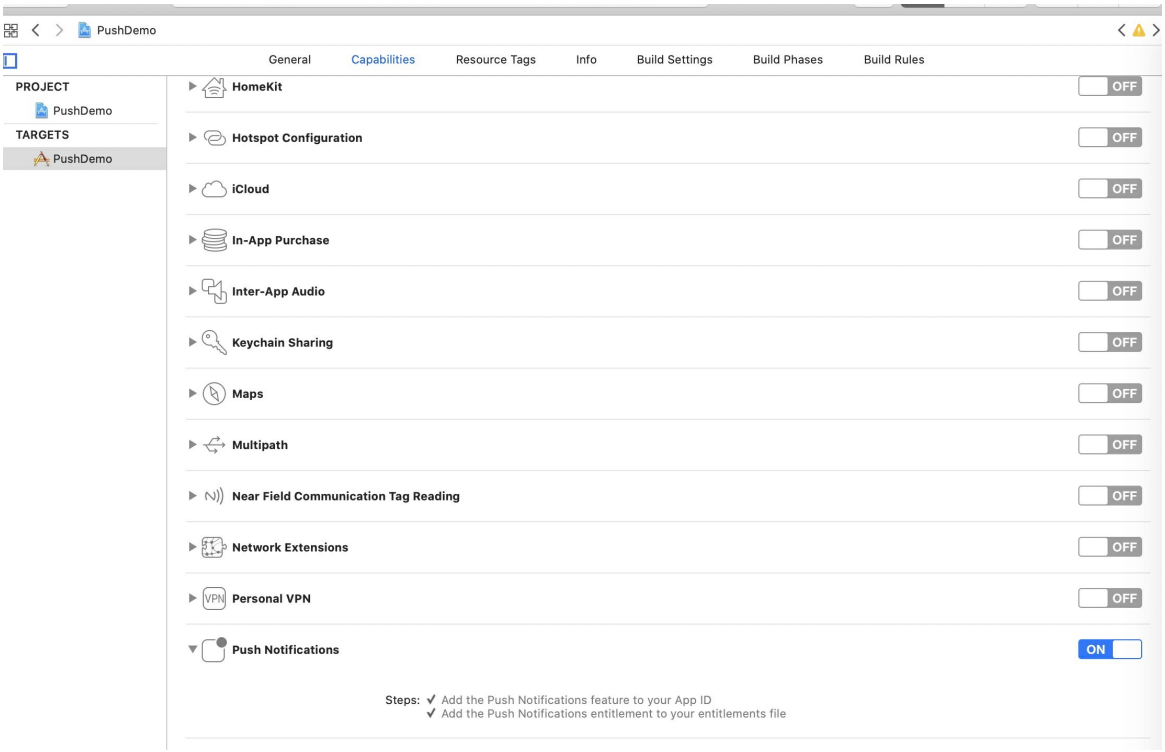
Procedure

To use MPS, you should complete the following steps.

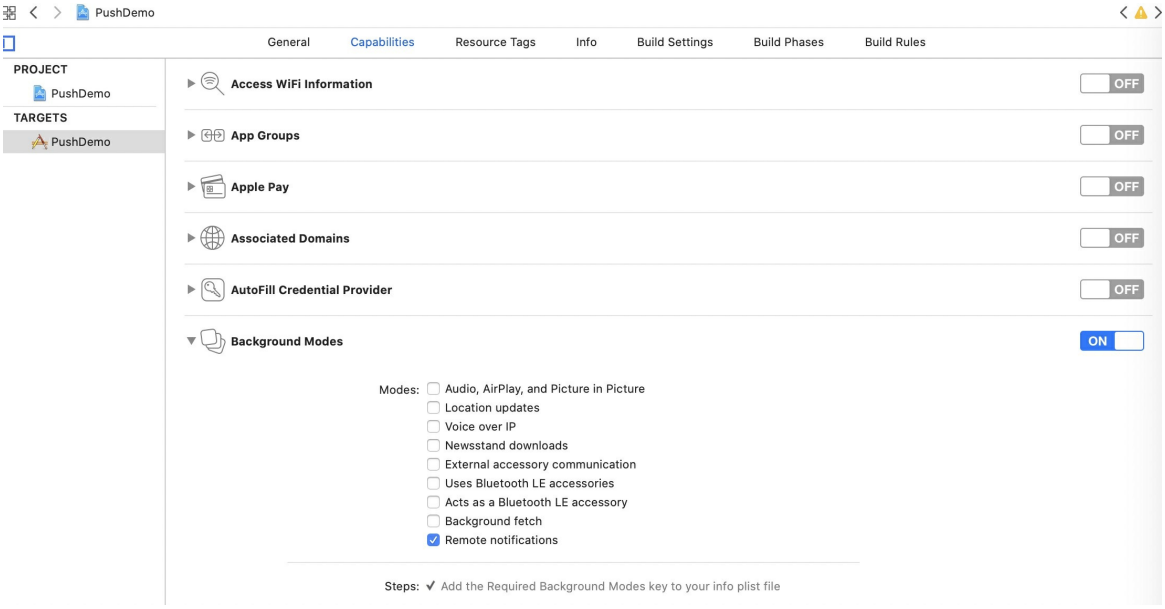
1. Use CocoaPods plugin to add the MPS SDK.
   i. In the Podfile file, use `mPaaS_pod "mPaaS_Push"` to add dependency.
   ii. Execute `pod install` to complete integrating the SDK.
2. Configure the project.

   Enable the following functions in the **TARGETS** directory of your project:

- **Capabilities > Push Notifications**



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



3. Use the SDK. In the case of using CocoaPods to access the iOS client based on an existing project, you need to complete the following operations.

i. (Optional) Register device token.

The message push SDK will automatically request the registration of deviceToken when the application is started. Generally, you do not need to request the registration of deviceToken. But in special cases (such as when there is privacy control at startup, when all network requests are blocked), you need to trigger the registration of deviceToken again after the control and authorization. The sample code is as follows:

```
- (void)registerRemoteNotification
{
    // Push notification registration
    if ([[[UIDevice currentDevice] systemVersion] floatValue] >= 10.0) {// 10.0+
        UNUserNotificationCenter* center = [UNUserNotificationCenter
currentNotificationCenter];
        center.delegate = self;
        [center
getNotificationSettingsWithCompletionHandler:^(UNNotificationSettings * _Nonnull se
ttings) {

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

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

ii. **Obtain the device token and bind it with user ID.**

The message push SDK provided by mPaaS encapsulates the logic of registering with the APNs server. After the program starts, the Push SDK automatically registers with the APNs server. You can obtain the deviceToken issued by APNs in the callback method of successful registration, and then call the interface method of PushService to report the binding userId to the mobile push core.

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

}
```

The push SDK also provides the API `- (void)pushUnBindWithUserId:(NSString *)userId completion:(void (^)(NSException *error))completion;` for unbinding the device token from the user ID of the app. For example, you can call the unbind API after the user switches to another account.

iii. **Receive push messages.**

After the client receives the pushed message, if the user clicks to view it, the system will start the corresponding application. The logic processing after receiving the push message can be done in the callback method of `AppDelegate`.

- **In the system versions earlier than iOS 10, the methods of processing notification bar messages or silent messages are as follows:**

```
 // Cold start for push messages in system versions earlier than iOS 10
  - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions {
  NSDictionary *userInfo = [launchOptions objectForKey:
UIApplicationLaunchOptionsRemoteNotificationKey];
  if ([[[UIDevice currentDevice] systemVersion] doubleValue] < 10.0) {
  // Cold start for push messages in system versions earlier than iOS 10
  }

  return YES;
  }

  // When the app runs in the foreground, adopt the method of processing common p
ush messages; when the app runs in the background or foreground, adopt the method
of processing silent messages ; when the app version is earlier than iOS 10, adop
t the method of processing notification bar messages
  -(void)application:(UIApplication *)application didReceiveRemoteNotification:(N
SDictionary *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult r
esult))completionHandler
  {
  // Process received messages
  }
```

- On iOS 10 and above, you need to implement the following delegate methods to listen for notification bar messages:

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

   // Receive remote push messages when the app runs in the foreground
  - (void)userNotificationCenter:(UNUserNotificationCenter *)center willPresentNo
tification:(UNNotification *)notification withCompletionHandler:(void (^)(UNNotif
icationPresentationOptions options))completionHandler
  {
      NSDictionary *userInfo = notification.request.content.userInfo;

      if([notification.request.trigger isKindOfClass:[UNPushNotificationTrigger c
lass]]) {
          // Receive remote push messages when the app runs in the foreground

      } else {
          // Receive local push messages when the app runs in the foreground

      }
      completionHandler(UNNotificationPresentationOptionNone);
  }

  // Receive remote push messages when the app runs in the background or uses col
d start mode
  - (void)userNotificationCenter:(UNUserNotificationCenter *)center didReceiveNot
ificationResponse:(UNNotificationResponse *)response withCompletionHandler:(void(
^)(void))completionHandler
  {
      NSDictionary *userInfo = response.notification.request.content.userInfo;

      if([response.notification.request.trigger isKindOfClass:
[UNPushNotificationTrigger class]]) {
          // Receive remote push messages when the app runs in the background or
uses cold start mode

      } else {
          // Receive local push messages when the app runs in the foreground

      }
      completionHandler();

  }
```

iv. Calculate message open rate.

In order to count the open rate of messages on the client side, you need to call the `pushOpenLogReport` interface of `PushService` (available in versions 10.1.32 and above) to report the message open event when the app message is opened by the user. After the event is reported, you can view the statistics of the message open rate on the **Message Push** > **Overview** page in the mPaaS console.

```
/**
 * Enable the API for reporting push messages so that the message open rate can be
calculated.
 * @param  userInfo userInfo of a message
 * @return
 */
- (void)pushOpenLogReport:(NSDictionary *)userInfo;
```

4. Configure a push certificate.

To push messages through the MPS console of mPaaS, you need to configure an APNs push certificate in the console. This certificate must match the signature on the client. Otherwise, the client cannot receive push messages.

For more information about the configuration, see Configure an iOS push certificate.

Follow-up steps

- After an APNs certificate is configured on the MPS console of mPaaS, messages can be pushed to applications in **device** dimension. MPS pushes messages to clients through Apple APNs. For more information, see Push process for Apple devices and Android devices outside China.
- After user IDs are reported and the server binds them with device tokens, messages can be pushed to applications in **user** dimension.

Code sample

Click here to download the code sample.

Related topics

- Create a message
- Configure the server

5.Server-side configuration

After learning about the message push process of Mobile Push Service, you need to configure signature verification, bind users and devices, and push messages.

Prerequisites

- You have activated mPaaS.
- You have a server-side application.
- You have reported the user ID and device ID on client.

Procedure

Step 1: Bind users and devices

When obtaining the user ID and device ID reported by client, the server calls the interface provided by mobile push service to complete binding.

For more information about interfaces, see Client APIs or Server APIs.

Step 2: Push messages

Server can push the following four types of messages by calling interfaces:

- Simple Push: Push simple messages.
- Template Push: Push templated messages.
- Multiple Push: Push different messages to different targets.
- Broadcast Push: Push message to all users.

6.Console operations

6.1. Data overview

Important: Since June 28, 2020, mPaaS has stopped support for the baseline 10.1.32. Please use 10.1.68 or 10.1.60 instead. For how to upgrade the baseline from version 10.1.32 to 10.1.68 or 10.1.60, see mPaaS 10.1.68 upgrade guide (Android/iOS) or mPaaS 10.1.60 upgrade guide (Android/iOS).

MPS provides statistics on message push data including pushed messages, successfully pushed messages, message arrivals, opened messages, and ignored messages, and supports filtering the data by platform, version, push channel, push type, and other criteria, and exporting the data reports.

Prerequisites

- You have integrated MPS SDK based on the mPaaS framework.
- You have completed client tracking by referring to the following topics. All data involved in usage analysis are collected from the SDK tracking logs.
  - Android: Report push data
  - iOS: Calculate message open rate

> **Note**
>
> For iOS devices, currently you can only collect the message open rate.

View push data

To view the statistical data about MPS usage, you should complete the following steps:

1. Log in to the mPaaS console, select the target app, and enter the **Message Push Service** > **Overview** page.
2. Set filter criteria to query statistical data. You can filter by **platform**, **version**, **push channel**, **push type**, and **time**, or input a complete task ID to search.

   > **Note**
   >
   > Searching data with task ID only works for messages delivered through multiple push. You can view the task ID on the **Multiple push records** page.

   - Platform: The options include **All platforms**, **Android - workspaceId**, and **iOS - workspaceId**. Available options depend on the existing push platforms with message push and the push console which launches message push. For example, if no message has been pushed to iOS devices, the **iOS - workspaceId** option is unavailable. In these options, workspaceId indicates the workspace ID of the push console.
   - Version: The value depends on tracking log reported by the client SDK. MPS gets the app version based on MAS statistics.
   - Push channel: The options include **All push channels**, **MPS self-built channel**, and **Third-party channel** (such as MIUI, HMS, vivo, OPPO and iOS). Only when any message push through the push channel occurred, the corresponding option is available. For example, if no message has been pushed through MIUI (MiPush) channel, the **MIUI** option is unavailable.

- Push type: The options include **All push types**, **Simple push - non-template based**, **Simple push - template based**, **Multiple push - all devices**, and **Multiple push - not all devices**. Only when message push of the push type occurred, the corresponding option is available. For example, if no template-based simple push occurred, the corresponding option is unavailable.
- Time range: A maximum of 90 days is allowed.

Core metrics

Display the critical push data within a certain period, including the pushed messages, successfully pushed messages, message arrivals, opened messages, ignored messages, etc.

Metrics	Description
Pushed messages	The total number of messages pushed by the backend, which is counted by backend.
Successfully pushed messages	MPS automatically collects statistics on the actual number of messages that have been pushed in the specified time period, which is counted by backend. The statistics doesn't care whether the messages were pushed within the specified time period. • One push task may contain multiple target IDs, and MPS needs to push a message to each of these targets. • If a token has expired or a user binding relationship does not exist, the target ID is invalid and MPS will not count the messages pushed to this target.
Message arrivals	The actual number of messages that have arrived at the client, which is counted by client. The statistics doesn’t care whether the messages were pushed within the specified time period. For example, if the message arrivals during 2021.8.1 ~ 2021.8.7 is 100, it means 100 pieces of messages arrived at client during the period. Among those 100 pieces of messages, some may be pushed before August 1. The data statistics varies with the push channels: • Android self-built channel: After messages are successfully pushed to devices, statistics are collected based on tracking log data reported by the client SDK. • iOS and Android third-party channels: After messages are pushed through specified channels, statistics are collected based on push results returned by backend services of these channels.
Arrival rate	Arrival rate = (Message arrivals/Pushed messages) × 100%.

Opened messages	The actual number of messages that have been opened on the client, which is counted by client. The value depends on tracking log data reported by the client SDK. MPS obtains the number of opened messages based on MAS statistics. The statistics doesn't care whether the messages arrived at client within the specified time period. For example, if the number of opened messages during 2021.8.1 ~ 2021.8.7 is 88, it means 88 pieces of messages were opened by users during the period. Among those 88 pieces of messages, some may have arrived at client before August 1.
Open rate	Open rate = (Opened messages/Message arrivals) × 100%
Ignored messages	The number of messages that are manually ignored by users on the client. The statistics doesn't care whether the messages arrived at client within the specified time period. The value depends on tracking log data reported by the client SDK. MPS obtains the number of ignored messages based on MAS statistics. For example, if the number of ignored messages during 2021.8.1 ~ 2021.8.7 is 66, it means 66 pieces of messages were manually ignored by users during the period. Among those 66 pieces of messages, some may have arrived at client before August 1.
Ignorance rate	Ignorance rate = (Ignored messages/Message arrivals) × 100%

Data trend

Message push statistical data is presented in a line chart. You can click the metric legend under the chart to hide or display the curve of a metric.

In the upper left corner of the chart, you can select **Query by quantity** or **Query by rate** to view the metric statistics in quantity or rate curves.

- Query by quantity: Displays curves of pushed, arrived, opened, and ignored messages.
- Query by rate: Displays curves of the arrival rate, open rate, and ignorance rate.

In the upper right corner of the chart, you can select a granularity to display the chart by minute, hour, or day.

- **Minutes**: The horizontal axis displays the time points (accurate to minutes) of pushed, arrived, opened, and ignored messages.
- **Hours**: The horizontal axis displays the time points (accurate to hours) of pushed, arrived, opened, and ignored messages.
- **Days**: The horizontal axis displays the time points (accurate to days) of pushed, arrived, opened, and ignored messages.

> **Note**
>
> If you set a duration longer than one day, **Minutes** and **Hours** will be unavailable.

Push details

Daily or hourly push details listed in the table are consistent with data displayed in the core metric chart.

- The time points in the **Time** column are obtained from the horizontal axis of the core metric chart.
- The list contains the following core metrics: pushed messages, successfully pushed messages, message arrivals (arrival rate), opened messages (open rate), and Ignored messages (ignorance rate).

Click **Export** in the upper right corner to download the corresponding data.

6.2. Message management

6.2.1. Create a message - Simple push

> **Important**
>
> Since March 18th, 2022, mPaaS MPS console has been upgraded. On the new console, the push types have been integrated and optimized from the previous four types (simple push, template push, multiple push and broadcast push) to two types (simple push and multiple push). The upgraded simple push covers the capabilities of the original simple push and template push; the upgraded multiple push covers the capabilities of the original multiple push and broadcast push.

Simple push refers to pushing a message to an individual user or device. When you pushing messages in this mode, you can either customize messages or create messages based on a predefined message template.

Customizing message is applicable for the scenarios of pushing messages to a few targets, such as verifying the validity of Apple Push certificate and checking whether the Android Push SDK is correctly integrated. The message template is suitable for the scenario of pushing messages to multiple targets in multiple times. That is to verify and test the template function by creating a template-based message through the console before the template function is automatically or widely used.

> **Note**
>
> - The messages are pushed immediately after they are created. You cannot delete or modify them.
> - Since manual operations are required, we recommend you push messages through the console in the scenarios requiring low-frequency message push, such as system verification, operation support, and temporary emergency requirement.

The following sections describe how to create a simple push message in the console.

Prerequisites

- To push messages to iOS devices, you should have integrated MPS iOS SDK (see Integrate iOS SDK) and configured the iOS push certificate on the **Channel configuration** page in the mPaaS console. For more information, see Configure iOS push channel.
- To push messages through the Android vendor channels (also known as third-party channels), you should have integrated MPS Android SDK (see Integrate Android SDK), integrated relevant vendor channels (see Integrate vendor push channels) and completed corresponding push channel setting on the **Channel configuration** page in the mPaaS console. For more information, see Channel configuration.

Procedure

1. Log in to the mPaaS console, select the target app, and go to the **Message Push Service** > **Message management** page.
2. Click the **Create message push task** button, and in the pop-up dialog box, select the **Simple push** tab.
3. On the simple push tab page, configure the basic information of the message. The configuration items are as follows:

Parameter	Required	Description
Message type: silent message	Yes	Whether to display the message: ◦ **Yes**: Indicates that the message will not be displayed in any form on the target device, and user has no sense about it. ◦ **No**: Indicates that the message will be displayed in the notification bar. For Android devices, you need to perform different operations according to the push channel that you have selected: ◦ **MPS channel**: This parameter is sent to the client as a reference field. You need to parse the message body and get the content of this field, then control the display of the message. ◦ **Vendor channel**: This parameter is sent to the target device as a field. The device vendor's system will then parse the content of this field, and control the display of the message. You do not need to perform any other operations. For iOS devices, the display of messages is controlled by the device vendor’s system. You do not need to perform any other operations.
Message content creation method	Yes	Create the message in either of the following ways: ◦ **Create**: Customizes message content, including message title, body and the presentation style. ◦ **Use a template**: Uses the predefined template.
Template	Yes	Choose a message template from templates listed on the **Message templates** page. **Note** It is required only when you choose to create the message based on a template.
Template placeholder	Yes	Enter variable values in the template. The system provides configuration options for placeholders in the selected template.

Parameter	Required	Description
Push dimension	Yes	Select the message delivery mode: ◦ **Users**: Push messages by user ID. You need to call the bind API to bind the user ID with device ID. For more information about the binding API, see Client APIs. ◦ **Android**: Push messages by Android device ID. ◦ **iOS**: Push messages by iOS device ID.
User ID/Device ID	Yes	Input the corresponding user ID or device ID according to the push dimension you chose. ◦ When the push dimension is Android, input the Ad-token. ◦ When the push dimension is iOS, input the Device Token. ◦ When the push dimension is user, input the actual user ID, that is the value of `userid` passed in when you called the binding API. ◦ If there is any space in the device ID obtained from sources such as logs, you need to delete the space.
Push priority of Android message channels	Yes	Only available for Android push platform. ◦ **Vendor channels preferred**: Vendor channels are preferred. If vendor channels are integrated, messages are pushed through the corresponding vendor channels; if no vendor channel is integrated to the app, the messages are pushed through MPS self-built channel. ◦ **MPS channel**: MPS uses the self-built channel to push messages. For Android devices, this parameter specifies whether to push messages through an MPS self-built channel or vendor channel. For iOS devices, you do not need to set this parameter (iOS push belongs to vendor channel push).

Parameter	Required	Description
Display style	Yes	The style that how the message is displayed on the client. You can choose any one of the following three styles: Default (short text), Big text, and Rich text. ◦ **Default**: This style is suitable for messages with concise and clear content. The message of this style contains title and text only. It is recommended to keep the length of the message text within 100 characters, including custom parameters and symbols. ◦ **Big text**: This style is suitable for messages with long text, such as information and news messages, so users can quickly obtain information without opening the application. The message of this style contains title and text only. It is recommended to keep the length of the message text within 256 characters, including custom parameters and symbols. ◦ **Rich text**: This style supports the messages containing icon and image, suitable for the messages with various content. To ensure good message presentation effect, it is better to keep the text within two lines.
Message title	Yes	Enter the title of the message with no more than 200 characters. The message display effect can be previewed in the preview area.
Message content	Yes	Enter the message boy with no more than 200 characters. The message display effect can be previewed in the preview area.
Icon	No	The icon displayed on the right of the message, which can be JPG, JPEG or PNG image. Enter the public accessible URL of the icon here. If you only provide the default icon URL while no materials are uploaded for the corresponding vendor channels, the default icon will be automatically pulled and used for the messages pushed through the vendor channels. Since the vendor channels have different requirements on the icon material, it is suggested to upload the material for each vendor channel separately according to their requirements. ◦ **Default icon**: The suggested size is 140 * 140px, not exceeding 50 KB. ◦ **OPPO icon**: The suggested size is 140 * 140px, not exceeding 50 KB. ◦ **Xiaomi icon**: The suggested size is 120 * 120px, not exceeding 50 KB. ◦ **Huawei icon**: The suggested size is 40 * 40dp, not exceeding 512 KB. ◦ **FCM icon**: If no specific requirement applies, the default icon will be automatically used.

Parameter	Required	Description
Large image	No	The image displayed at the lower part of the message, which can be JPG, JPEG or PNG image. Enter the public accessible URL of the image here. If you only provide the default image URL while no materials are uploaded for the corresponding vendor channels, the default large image will be automatically pulled and used for the messages pushed through the vendor channels. Since the vendor channels have different requirements on the image, it is suggested to upload the material for each vendor channel separately according to their requirements. ◦ **Default large image**: The suggested size is 876 * 324px, not exceeding 1 MB. ◦ **OPPO large image**: The suggested size is 876 * 324px, not exceeding 1 MB. ◦ **Xiaomi large image**: The suggested size is 876 * 324px, not exceeding 1 MB. ◦ **iOS large image**: Supports custom images, without limitation on image size. ◦ **FCM large image**: If no specific requirement applies, the default image will be automatically used.
Push mode	Yes	Select the time to push message: ◦ **Now**: Push the message immediately once the message push task is created. ◦ **Scheduled**: Specify a time to push the message. For example, push the message at 8:00 am on June 19th. ◦ **Cyclic**: Push the message at a specific time cyclically within a period. For example, push the message at 8:00 am every Friday from June 1st to September 30th.

The preview area is on the right side of the message creation window. To preview the message display effects for different platforms respectively, click **Notification**, **iOS message body** and **Android message body**.

4. (Optional) Configure the advanced information on demand. In the **Advanced information** area, complete the following configurations:
   - **Redirect upon click**: Specify the operation to be performed after a user taps the message on the phone. This parameter is sent to the client as a reference field. You need to implement subsequent operations by referring to the field.
     - **Web page**: Users will be redirected to a Web page.
     - **Custom page**: Users will be redirected to a native page.
   - **Redirection address**: The page to be visited after a user taps the message on the mobile phone. Enter the address according to the option you chose.
     - For Web page, enter the URL of the web page to be visited.
     - For custom page, enter the address of the native page to be visited (Android: ActivityName; iOS: VCName).

- **Custom message ID**: Custom message ID is automatically generated by the system to uniquely identify the message in the client's system. It can be customized and a maximum of 64 characters are allowed.

  > **Note**
  >
  > Custom message ID is required for silent message only.

- **Valid period**: Specify the valid period of the message in seconds. To ensure the message arrival rate, when a message fails to be sent because the device is offline or the user is logged out, MPS will resend it after the device is connected or a user binding request is initiated within the validity period of the message. It is 180 seconds by default.

  > **Note**
  >
  > The valid period cannot be shorter than 180 seconds or longer than 72 hours.

- **Extension parameters**: Turn the switch on, click **Add parameter**, set the key/value, and left click on any area of the page to complete setting. The extension parameters are passed to the client together with the message body for your use.

  Extension parameters include the following three types:

  - System extension parameters

    These extension parameters are occupied by the system, and cannot be modified. System extension parameters include `notifyType`, `action`, `silent`, `pushType`, `templateCode`, `channel`, and `taskId`.

- System extension parameters with some significance

  These extension parameters are occupied by the system and have some significance. You can configure values of these extension parameters.

  For more information about these parameters, see the following table.

Parameter	Description
sound	The custom ringtone of the message. The value of this parameter is the path of the ringtone. This parameter is only valid for Xiaomi phones and iPhones.
badge	Badge number. Its value is a specific number. This extension parameter will be passed to the client together with the message body. ▪ For Android devices, you need to implement the badge logic by yourself. ▪ For iOS devices, iOS system automatically implements the badge logic. When a message is pushed to the target mobile phone, the number that you specified in value appears in the badge of the app icon.
mutable-content	The APNs custom push identifier. If a pushed message carries this parameter, it indicates that the `UNNotificationServiceExtension` of iOS10 is supported, otherwise it is a normal push. The value is set to 1.
badge_add_num	Accumulative badge number, only available in Huawei channel.
badge_class	Activity class corresponding to the desktop app icon in Huawei channel.
big_text	Big text style, the value is fixed to 1, and other values are invalid. This parameter is only valid for Xiaomi and Huawei phones.

- User-defined extension parameters

  All other parameters than system extension parameters and system extension parameters with some significance are user-defined extension parameters. User-defined extension parameters are passed to the client together with the message body for your use.

5. Click **Submit** to complete creating the message. The new message will appear in the simple push records.

In addition to console operation, you can also push messages by calling relevant APIs. For more information, see Server APIs.

Relevant operations

- Create a message – Multiple push

- Call API to push messages
- Manage messages

6.2.2. Create a message – Multiple push

> **Important**
>
> Since March 18th, 2022, mPaaS MPS console has been upgraded. On the new console, the push types have been integrated and optimized from the previous four types (simple push, template push, multiple push and broadcast push) to two types (simple push and multiple push). The upgraded simple push covers the capabilities of the original simple push and template push; the upgraded multiple push covers the capabilities of the original multiple push and broadcast push.

Multiple push is mainly used to push messages to a large number of users to meet some operation needs.

The multiple push falls into network-wide push and non network-wide push.

- Network-wide push refers to pushing the same template-based message to all Android and iOS networking devices, which only supports pushing by devices.

  When you push a message to Android devices, all the Android devices that are connected in the message validity period can receive the message; when you push a message to iOS devices, all the iOS devices that are bound in the message validity period can receive the message.

- Non network-wide push refers to pushing the same template-based message to specified user groups.

  You can manually upload a group of message receivers, customize tagged user groups, or use the MAS groups.

> **Note**
>
> - The messages are pushed immediately after they are created. You cannot delete or modify them.
> - Since manual operations are required, we recommend you push messages through the console in the scenarios requiring low-frequency message push, such as system verification, operation support, and temporary emergency requirement.

The following sections describe how to create a multiple push message in the console.

Prerequisites

- To push messages to iOS devices, you should have integrated MPS iOS SDK (see Integrate iOS SDK) and configured the iOS push certificate on the **Channel configuration** page in mPaaS console. For more information, see Configure iOS push channel.
- To push messages through the Android vendor channels (also known as third-party channels), you should have integrated MPS Android SDK (see Integrate Android SDK), accessed relevant vendor channels (see Integrate vendor push channels) and completed corresponding push channel setting on the **Channel configuration** page in mPaaS console. For more information, see Channel configuration.
- Before creating a multiple push task, you need to prepare a template. For how to create a template, see Create a message template.

- When you create a multiple push task, if you choose to call the MAS group as the target audiences, you should create a MAS group in advance. For details, see Create user groups. If you choose a tagged user group as the target audiences, you should create a tagged user group in advance. For details, see Create a user tag.

Procedure

1. Log in to the mPaaS console, select the target app, and go to the **Message Push Service** > **Message management** page.
2. Click the **Create message push task** button, and in the pop-up dialog box, select the **Multiple push** tab.
3. On the multiple push tab page, configure the basic information of the message. The configuration items are as follows:

Parameter	Required	Description
Message type: silent message	Yes	Whether to display the message: ◦ **Yes**: Indicates that the message will not be displayed in any form on the target device, and user has no sense about it. ◦ **No**: Indicates that the message will be displayed in the notification bar. For Android devices, you need to perform different operations according to the push channel that you have selected: ◦ **MPS channel**: This parameter is sent to the client as a reference field. You need to parse the message body and get the content of this field, then control the display of the message. ◦ **Vendor channel**: This parameter is sent to the target device as a field. The device vendor's system will then parse the content of this field, and control the display of the message. You do not need to perform any other operations. For iOS devices, the display of messages is controlled by the device vendor’s system. You do not need to perform any other operations.
Push dimension	Yes	Select the message delivery mode: ◦ **Users**: Push messages by user ID. You need to call the bind API to bind the user ID with device ID. For more information about the binding API, see Client APIs. ◦ **Devices**: Push messages by device ID.

Push platform	Yes	When you choose the push dimension as **Devices**, you need to select a push platform to specify the type of the target device. ◦ **Android**: MPS provides vendor channels and MPS self-build channel to push the message to the network-wide online Android devices (in valid period) or specified Android devices. The message will be pushed only once for each device. ◦ **iOS**: Use the vendor channel to push the message to the network-wide or specified iOS devices. The message will be pushed only once for each device.
Select push targets	Yes	◦ When you choose the push dimension as **Users**, you have the following options: ▪ **Upload a group**: Upload the file containing target IDs and the personalized configuration of each target ID based on the selected template. Every data record in the file represents a message, which is identified by a customer message ID. Requirements for the file format are as follows: ▪ The format of each data record: `target ID,customer message ID,placeholder 1=XXX;placeholder 2=XXX…` , where the customer message ID can be user customized. ▪ The file encoding type must be UTF-8 and the maximum file size is 200 MB. Separate multiple data records with line breaks. Each data record must be 1~250 characters in length. Only one file can be uploaded in one push task. After a file is successfully uploaded, its icon is displayed below the **Upload** button. You can preview up to 10 data records of the file by clicking the icon. ▪ **MAS group**: Call the MAS group and push the same message to the specified group users. You need to create a MAS group first. For details, see Create user groups. If the message template includes any placeholder, this option is unavailable. ▪ **User tags**: Select the target groups by tag. You should create a tagged user group first. For details, see Create a user tag. ◦ When you choose the push dimension as **Devices**, you have the following options: ▪ **All devices**: Push the message to all devices of the selected platform.

		▪ **Partial devices**: Upload the file containing target IDs and the personalized configuration of each target ID based on the selected template. Every data record in the file represents a message, which is identified by a customer message ID. Requirements for the file format are as follows: ▪ The format of each data record: `target ID,customer message ID,placeholder 1=XXX;placeholder 2=XXX…` , where the customer message ID can be user customized. ▪ The file encoding type must be UTF-8 and the maximum file size is 200 MB. Separate multiple data records with line breaks. Each data record must be 1~250 characters in length. Only one file can be uploaded in one push task. After a file is successfully uploaded, its icon is displayed below the **Upload** button. You can preview up to 10 data records of the file by clicking the icon. ▪ **MAS group**: Call the MAS group and push the same message to the specified group users. You need to create a MAS group first. For details, see Create user groups. If the message template includes any placeholder, this option is unavailable.
Template	Yes	Choose a message template from templates listed on the **Message templates** page.
Template placeholder	Yes	Enter variable values in the template. The system provides configuration options for placeholders in the selected template.
Push priority of Android message channels	Yes	Only available for Android push platform. ◦ **Vendor channels preferred**: Vendor channels are preferred. If vendor channels are integrated, messages are pushed through the corresponding vendor channels; if no vendor channel is integrated to the app, the messages are pushed through MPS self-built channel. ◦ **MPS channel**: MPS uses the self-built channel to push messages. For Android devices, this parameter specifies whether to push messages through an MPS self-built channel or vendor channel. For iOS devices, you do not need to set this parameter (iOS push belongs to vendor channel push).
Push mode	Yes	Select the time to push message: ◦ **Now**: Push the message immediately once the message push task is created. ◦ **Scheduled**: Specify a time to push the message. For example, push the message at 8:00 am on June 19th. ◦ **Cyclic**: Push the message at a specific time cyclically within a period. For example, push the message at 8:00 am every Friday from June 1st to September 30th.

The preview area is on the right side of the message creation window. To preview the message display effects for different platforms respectively, click **Notification**, **iOS message body** and **Android message body**.

4. (Optional) Configure the advanced information on demand. In the **Advanced information** area, complete the following configurations:
   - **Redirect upon click**: Specify the operation to be performed after a user taps the message on the phone. This parameter is sent to the client as a reference field. You need to implement subsequent operations by referring to the field.
     - **Web page**: Users will be redirected to a Web page.
     - **Custom page**: Users will be redirected to a native page.
   - **Redirection address**: The page to be visited after a user taps the message on the mobile phone. Enter the address according to the option you chose.
     - For Web page, enter the URL of the web page to be visited.
     - For custom page, enter the address of the native page to be visited (Android: ActivityName; iOS: VCName).
   - **Login status**: Specify target users according to login status. When you select the login/logout period, **Permanent** means no time limit, namely pushing messages to all login/logout users.

     > **Important**
     >
     > Login status is unconfigurable when you use Android push platform and push messages through MPS self-built channel.

     - If you select **Login users**, MPS will push messages to the users who logged in to the App in the specified time period. For example, if the login period is 15 days, it means pushing messages to the users who logged in to the App in recent 15 days.
     - If you select **Logout users**, MPS will push messages to the users who logged out from the App in the specified time period. For example, if the logout period is 15 days, it means pushing messages to the users who logged out in recent 15 days.
     - If you select both **Login users** and **Logout users**, MPS will push messages to the users who logged in to the App and logged out in the specified time period. For example, if the login period is permanent while the logout period is 7 days, it means pushing messages to all login users and the users who logged out in recent 7 days.
   - **Custom message ID**: Custom message ID is automatically generated by the system to uniquely identify the message in the client's system. It can be customized and a maximum of 64 characters are allowed.
   - **Valid period**: Specify the valid period of the message in seconds. It is 180 seconds by default. To ensure the message arrival rate, when a message fails to be sent because the device is offline or the user is logged out, MPS will resend it after the device is connected or a user binding request is initiated within the validity period of the message.
   - **Extension parameters**: Turn the switch on, click **Add parameter**, set the key/value, and left click on any area of the page to complete setting. The extension parameters are passed to the client together with the message body for your use.

     Extension parameters include the following three types:

- System extension parameters

  These extension parameters are occupied by the system, and cannot be modified. System extension parameters include `notifyType`, `action`, `silent`, `pushType`, `templateCode`, `channel`, and `taskId`.

- System extension parameters with some significance

  These extension parameters are occupied by the system and have some significance. You can configure values of these extension parameters.

  For more information about these parameters, see the following table.

Parameter	Description
sound	The custom ringtone of the message. The value of this parameter is the path of the ringtone. This parameter is only valid for Xiaomi phones and iPhones.
badge	Badge number. Its value is a specific number. This extension parameter will be passed to the client together with the message body. ▪ For Android devices, you need to implement the badge logic by yourself. ▪ For iOS devices, iOS system automatically implements the badge logic. When a message is pushed to the target mobile phone, the number that you specified in value appears in the badge of the App icon.
mutable-content	The APNs custom push identifier. If a pushed message carries this parameter, it indicates that the `UNNotificationServiceExtension` of iOS10 is supported, otherwise it is a normal push. The value is set to 1.
badge_add_num	Accumulative badge number, only available in Huawei channel.
badge_class	Activity class corresponding to the desktop App icon in Huawei channel.
big_text	Big text style, the value is fixed to 1, and other values are invalid. This parameter is only valid for Xiaomi and Huawei phones.

- User-defined extension parameters

  All other parameters than system extension parameters and system extension parameters with some significance are user-defined extension parameters. User-defined extension parameters are passed to the client together with the message body for your use.

5. Click **Submit** to complete creating the message. The new message will appear in the multiple push records.

In addition to console operation, you can also push messages by calling relevant APIs. For more information, see Server APIs.

Relevant operations

- Create a message – Simple push
- Call API to push messages
- Manage messages

6.2.3. Manage simple push messages

The **Simple push records** tab page shows the relevant information of simple push messages created in the last 30 days., and you can query the historical messages. The list only displays the messages pushed through the console. For the messages pushed by calling simple push API, you can query the message details by device/user ID or custom message ID.

View push details

1. Log in to the mPaaS console, select your app, and enter the **Message Push Service** > **Message management** > **Simple push records** page.
2. In the search box displayed in the upper right corner, enter a complete device ID, user ID or customer message ID to search for the message. The message with the specified target ID and customer message ID will be displayed in the message list.

   > **Note**
   >
   > You can only search for simple push messages created in the last 30 days.

   Messages are sorted in descending order by creation time by default. The information displayed in the list includes:

   - **Customer message ID**: It is customized by user or automatically generated by system.
   - **Push time**: It refers to the time when the message was pushed, accurate to seconds.
   - **Push mode**: It indicates that the message was pushed immediately upon creation or was pushed in schedule.
   - **Push dimension**: It indicates that the message was pushed by user, Android device or iOS device.
   - **Target ID**: user ID or device ID.
   - **Message title**: the title of a message.
   - **Creation time**: The time when the message was successfully created, accurate to seconds.
   - **Push status**: Shows the push status of a message. To learn the status codes and corresponding description, see Message push status codes.
3. To view the push details of a message, click the **Expand** button (**+**) of the target message on the list.

   Then the following information appears:

   - **Message ID**: It refers to the unique identifier of a message automatically generated by MPS.
   - **Offline retention period**: It refers to the time when a message expires. If a message has not been sent successfully, MPS will resend it after the device is connected or a user binding request is initiated. However, if the message expires, MPS will not resend it.
   - **Display type**: Shows that the message is a plain text message, a big text message or a rich text message.

- **Extension parameters**: Shows the extension parameters added during message creation.
- **Message content**: message body.

Revoke messages

It is supported to revoke the messages that have been pushed in past 7 days. For more information, see Message revocation.

Silent messages will be immediately withdrawn once you revoke them, and the client-side users have no sense about that. For non-silent messages, stop pushing the ones not arriving user devices, and cancel presenting the ones that have arrived the user devices but not appeared.

> **Note**
>
> The messages with "Failed" push status cannot be revoked.

6.2.4. Manage multiple push messages

Message Push Service (MPS) provides real-time statistics on the multiple-push and broadcast-push tasks that are created through MPS console or triggered by calling API to help you get the message push status.

View push tasks

1. Log in to the mPaaS console, select your app, and enter the **Message Push Service** > **Message management** > **Multiple push records** page.
2. In the search box displayed in the upper right corner, enter a complete push task ID or task name, and specify the time range to search the tasks. The eligible tasks will appear in the task list.

   In the task list, the tasks are sorted in descending order by creation time. The task information displayed includes:

   - **Task ID**: The unique identifier of the push task, which is automatically generated by the system.
   - **Task name (API)**: If the push task is delivered through the MPS console, the task name is automatically generated by the system, usually named in the format “console + time”, for example, “Console Wed Mar 24 14:47: 23 CST 202”; if the task is triggered by calling an API, the task name is the name filled in by the caller.
   - **Push type**: It indicates that the message was pushed immediately upon creation or was pushed in schedule.
3. To view the push details, click the **Expand** button (**+**) of the target task on the list.
   - **Pushed messages**: Refers to the total number of messages pushed by message push backend, which is counted by the backend.
   - **Successfully pushed messages**: Refers to the total number of messages successfully pushed by message push backend, which is counted by the backend.
   - **Message arrivals**: The number of messages that actually arrive the device. For iOS channel or Android third-party channels (such as Xiaomi and Huawei), the statistics relies on the result returned from the corresponding third-party channel’s backend after the messages are pushed to the third-party channels. For the Android self-built channel, the statistics relies on the tracking report after the messages are pushed the client.

- **Offline retention period**: Indicates the validity period of the message. In the validity period, MPS delivers the message to the target devices or users once the target devices get connected or the users initiate a binding request till the message is pushed successfully. Once the message expires, the MPS will no longer deliver the message.

Revoke messages

It is supported to revoke the messages that have been pushed in past 7 days. For more information, see Message revocation.

Silent messages will be immediately withdrawn once you revoke them, and the client-side users have no sense about that. For non-silent messages, stop pushing the ones not arriving user devices, and cancel presenting the ones that have arrived the user devices but not appeared.

> **Note**
>
> The messages with "Failed" push status cannot be revoked.

6.2.5. Manage scheduled push task

All scheduled push tasks and cyclic push tasks created through the mPaaS console and triggered by calling APIs are displayed in the scheduled push task list. One cyclic push task may contain one or more scheduled push tasks.

View a scheduled push task

1. Log in to the mPaaS console, and select a target app. In the navigation pane on the left, choose **Message Push Service** > **Message management** > **Scheduled push tasks**.
2. In the search bars in the upper right of the displayed **Scheduled push task** tab page, specify the scheduled push time and the push type, enter a push task ID, and click the **Search** button ( ) to search. Or you can press Enter to search. The tasks that are found will be displayed in the list.

   By default, scheduled push tasks are sorted by creation time in descending order. The information displayed in the list includes:
3. Specify the push type and the scheduled push time to filter messages, and enter a push task ID to search for messages. The results that are found will be displayed in the message list. Note that the push type can be mPaaS console or API and all push types are displayed by default. By default, messages in the message list are sorted by creation time in descending order. The information displayed in the list includes:
   - **Scheduled push time**: push time specified when you create a push task.
   - **Task ID**: unique ID of a scheduled push task. The task ID is generated automatically by the system.
   - **Push mode**: scheduled and cyclic.
   - **Push dimension**: the push dimension of a message, which can be users or devices.
   - **Message title**: the title of a message.
   - **Message body**: the body content of a message.
   - **Push type**: simple push and multiple push.
   - **Creation method**: the creation mode of a message. You can push a message through the mPaaS console or by calling APIs.
   - **Push status**: indicates whether a scheduled push task has been implemented.

Cancel a scheduled push task

A scheduled push task that has not been implemented can be canceled. Each cyclic push task contains one or more scheduled push tasks. When you cancel a cyclic push task, you need to confirm whether to cancel the latest scheduled push task or all scheduled push tasks.

With Message Push Service (MPS), you can cancel a scheduled push task by the mPaaS console or by calling APIs. For more details, see section Cancel a scheduled push task.

6.3. Message templates

6.3.1. Create a message template

A template consists of the body, placeholders and some other attributes. You can use placeholders to specify dynamic content in the template. Only templates with placeholders can be used to send personalized messages.

You can use templates to flexibly configure messages and eliminate input of repeated content.

In a template, you can mark the dynamic part in the **title**, **body**, and **redirection URL** by using the format of **#placeholder name#**.

Procedure

1. Log in to the mPaaS console, select your app, and enter the **Message Push Service** > **Message templates** page.
2. On the right page, click the **Create template** button, and in the pop-up template creation window, configure template information. The following table describes related parameters.

Parameter	Required	Description
Template name	Yes	The name of the template. The name must be 1 to 200 characters in length, and can contain letters, digits, and underscores (_). The name must be unique, and it will be used to identify the template in API calling.
Description	Yes	The description of the template. The description must be 1 to 200 characters in length, and can contain letters, numbers, and underscores (_).
Template title	Yes	The title of the template. The title must be 1 ~ 200 characters in length.
Template body	Yes	The body of the template. The text must be 1 ~ 200 characters in length.

Message type: silent message	Yes	Whether to display the message: ◦ **Yes**: Indicates that the message will not be displayed in any form on the target device, and user has no sense about it. ◦ **No**: Indicates that the message will be displayed in the notification bar. For Android devices, you need to perform different operations according to the push channel that you have selected: ◦ **MPS channel**: This parameter is sent to the client as a reference field. You need to parse the message body and get the content of this field, then control the display of the message. ◦ **Vendor channel**: This parameter is sent to the target device as a field. The device vendor's system will then parse the content of this field, and control the display of the message. You do not need to perform any other operations. For iOS devices, the display of messages is controlled by the device vendor’s system. You do not need to perform any other operations.
Display style	Yes	The style that how the message is displayed on the client. You can choose any one of the following three styles: Default (short text), Big text, and Rich text. ◦ **Default**: This style is suitable for messages with concise and clear content. The message of this style contains title and text only. It is recommended to keep the length of the message text within 100 characters, including custom parameters and symbols. ◦ **Big text**: This is style is suitable for messages with long text, such as information and news messages, so users can quickly obtain information without opening the application. The message of this style contains title and text only. It is recommended to keep the length of the message text within 256 characters, including custom parameters and symbols. ◦ **Rich text**: This style supports the messages containing icon and image, suitable for the messages with various content. To ensure good message presentation effect, it is better to keep the text within two lines.

Icon	No	The icon displayed on the right of the message, which can be JPG, JPEG or PNG image. Enter the public accessible URL of the icon here. If you only provide the default icon URL while no materials are uploaded for the corresponding third-party channels, the default icon will be automatically pulled and used for the messages pushed through the third-party channels. Since the third-party channels have different requirements on the icon material, it is suggested to upload the material for each third-party channel separately according to their requirements. ◦ **Default icon**: The suggested size is 140 * 140px, not exceeding 50 KB. ◦ **OPPO icon**: The suggested size is 140 * 140px, not exceeding 50 KB. ◦ **Xiaomi icon**: The suggested size is 120 * 120px, not exceeding 50 KB. ◦ **Huawei icon**: The suggested size is 40 * 40dp, not exceeding 512 KB. ◦ **FCM icon**: If no specific requirement applies, the default icon will be automatically used.
Large image	No	The image displayed at the lower part of the message, which can be JPG, JPEG or PNG image. Enter the public accessible URL of the image here. If you only provide the default image URL while no materials are uploaded for the corresponding third-party channels, the default large image will be automatically pulled and used for the messages pushed through the third-party channels. Since the third-party channels have different requirements on the image, it is suggested to upload the material for each third-party channel separately according to their requirements. ◦ **Default large image**: The suggested size is 876 * 324px, not exceeding 1 MB. ◦ **OPPO large image**: The suggested size is 876 * 324px, not exceeding 1 MB. ◦ **Xiaomi large image**: The suggested size is 876 * 324px, not exceeding 1 MB. ◦ **iOS large image**: Support custom images, without limitation on image size. ◦ **FCM large image**: If no specific requirement applies, the default image will be automatically used.
Redirect upon click	Yes	This parameter is sent to the client as a reference field. You need to implement subsequent operations by referring to the field. ◦ **Web page**: Users will be redirected to a Web page. It is required to enter the URL of the web page to be visited. ◦ **Custom page**: Users will be redirected to a native page. It is required to enter the address of the native page to be visited (Android: ActivityName; iOS: VCName).

Redirection address	No	The page to be visited after a user taps the message on the mobile phone. This parameter will be sent to the client as a reference. You need to develop the implementation logic by yourself. Set this parameter based on the value of **Redirect upon click**.

3. Click **Submit** to create the template. When the template is created successfully, the **Message templates** page is displayed, with the new template listed at the top.

6.3.2. Manage message templates

The template list displays information about existing message templates. You can view or delete them as required.

View the template list

1. Log in to the mPaaS console, select your app, and enter the **Message Push Service** > **Message templates** page.

   Templates are listed in descending order by creation time . You can view the name, description, body, and creation time of the template.

2. Click **View** in the **Operations** column of the target template to view detailed information about the template.

Delete a template

The procedure is as follows:

1. On the template list, click **Delete** in the **Operations** column of the target template.
2. In the dialog box that appears, click **OK**. Then the template is deleted.

> **Note**
>
> Before deleting a template, ensure that it is not used for any messages to be sent. Otherwise, the corresponding messages cannot be sent.

6.4. Message revocation

Message Push Service (MPS) enables you to revoke messages that have been pushed. With this function, notifications that have been sent but not viewed or cleared will disappear from the device notification bar. To reduce business loss and related impacts, this function mainly applies to the following two scenarios: 1. Wrong messages are pushed due to misoperations; 2. Messages that have been pushed but need to be revoked urgently in case of temporary business changes.

You can query the message status and revoke messages through the mPaaS console. In addition, MPS supports backend APIs. You can revoke messages by calling APIs in the business system.

The mode of implementing message revocation varies with the push channel. The following table describes the specific details.

Push channel		Revocation supported or not	How it works
Vendor channel	Huawei	Yes	Overlap a message. After the client receives the command of revoking a message, the message displayed in the notification bar will be cleared. The "Message revoked" message is displayed.
	Xiaomi	Yes	Overlap a message. After the client receives the command of revoking a message, the message displayed in the notification bar will be cleared. The "Message revoked" message is displayed.
	OPPO	Yes	Overlap a message. After the client receives the command of revoking a message, the message displayed in the notification bar will be cleared. The "Message revoked" message is displayed.
	Vivo	Yes	Revoke a message. After the client receives the command of revoking a message, the message displayed in the notification bar will be directly cleared. That is, the message will disappear from the notification bar.
	Apple (iOS)	Yes	Overlap a message. After the client receives the command of revoking a message, the message displayed in the notification bar will be cleared. The "Message revoked" message is displayed.
MPS self-built channel		Yes	Overlap a message. After the client receives the command of revoking a message, the message displayed in the notification bar will be cleared. The "Message revoked" message is displayed.
SMS push		No	The SMS messages that have been sent cannot be revoked.

Revoke a message by the mPaaS console

1. Log in to the mPaaS console, and select a target app. In the navigation pane on the left, choose **Message Push Service** > Message management.

2. Select a message push task type to enter the message list page.
3. Select a message to be revoked, click **Revoke**, and click OK. After you perform the revocation operation, a message that is being pushed will not be pushed. A message that has been pushed but is not displayed will not be displayed.

Revoke a message by calling APIs

A message pushed in the simple push mode can be revoked by the message ID. A message pushed in the multiple push mode can be revoked by the task ID. Only messages in recent 7 days can be revoked.

For how to revoke a message by calling APIs, see the documentation listed in Message revocation API.

6.5. User tag management

With Message Push Service (MPS), you can set tags to customize user groups to whom messages are pushed to facilitate user management. If you set a user tag when you push a message, you can push the message to all the users marked with such tag.

A tag is one attribute that describes the basic attribute, hobbies, and behavior characteristics of a user. After you set one tag for users, you can use such tag to select the user group with the same characteristic. In this way, messages are accurately pushed to targeted users. For example, you can set one tag called "Female" for female users. Then, you can select the user group marked with such tag and push messages to the group on International Women's Day.

Users have a many-to-many relationship with tags. That is, one user can correspond to multiple tags, and one tag can also correspond to multiple users.

Create a user tag

To create a user tag is to tag a group of users with the same characteristic.

The procedure is as follows:

1. Log in to the mPaaS console, and select a target app. In the navigation pane on the left, choose **Message Push Service** >

   Settings > **User tag management**.
2. Click **Create user tag**. In the displayed Create user tag page, enter a tag name and add a group. Two ways of adding a group are as follows:
   - **Tag name**: presents the group characteristic directly to facilitate user management. Any character is supported. A maximum of 30 characters are allowed. The tag name should be unique in an app.
   - **Add a group**: supports adding users directly and importing a file including user IDs.
     - **Add directly**: enter one or more user IDs in a text box. User IDs are separated with ",". Each record cannot exceed 60 characters in length; otherwise, the excess content will not be added. A maximum of 10,000 characters are allowed.
     - **Import file**: upload a .txt file that contains the user ID. The file size cannot exceed 100 MB. User IDs are separated with a line break in a file. Each record cannot exceed 60 characters in length; otherwise, the excess content will not be added. A maximum of 500,000 user IDs can be uploaded. When you import user IDs, the system automatically deduplicates the IDs.
3. After you complete the configuration, click **Submit**. A new user tag is created. The new user tag will be displayed in the list.

View a user tag

All user tags in the list are displayed by creation time in descending order. The tag name, tag ID, users, creation time, and update time are displayed in the user tag list. Where:

- Tag ID: generated automatically by the system after you create a user tag successfully.
- Users: the number of user IDs contained in the user group.

In the user tag list, click **Details** in the **Operations** column to view the user tag information.

Edit a user tag

In the user tag list, click **Edit** in the **Operations** column to edit the tag name or modify the user information that corresponds to the tag.

For detailed operations of modifying the user information corresponding to a tag, see the content of adding a group described in Create a user tag.

Delete a user tag

In the user tag list, click **Delete** in the **Operations** column to delete the user tag. When you delete a user tag, all the user information corresponding to the user tag will be deleted.

Export a user list

In the user tag list, click **Export** in the **Operations** column to download the user list that corresponds to the tag.

6.6. Device status query

Message Push Service (MPS) supports querying the status of the target devices to which the messages are pushed by user ID (UserId) or device ID (DeviceId). You can check device status to facilitate troubleshooting in case of any pushing problems.

Complete the following steps to query device status:

1. Log in to the mPaaS console, select the target app, and go to the **Message Push Service** > **Query tool** page from the left navigation pane to enter the device status query page.
2. Set the query criteria to query the status of the target device.

   Select the query dimension, **User ID** or **Device ID**, enter the corresponding user ID or device ID, and then press **Enter** or click the search icon to query the relevant information of the device. The queried information includes user ID , device ID, self-built Token, vendor Token, platform, device manufacturer, and self-built channel status.

   Where,

   - **User ID**: It refers to the `userid` value passed in when the user calls the binding interface.
   - **Device ID**: For Android device, it refers to the self-built channel token; for iOS device, it refers to the APNS token.
   - **Self-built Token**: It refers to the identifier of self-built channel.
   - **Vendor Token**: It refers to the identifier of the vendor channel.
   - **Self-built channel status**: It indicates whether the self-built channel of the current device is online.
     - For Android device, the device status is either **Online** or **Offline**.
     - For iOS device, since the iOS platform completes message push through the third-party channel, so the device status is always **Unknown**.

6.7. Channel configuration

This topic describes how to configure the iOS and Android push channels through console.

Configure iOS push channel

When you push messages to an iPhone, the APNs service is used as the message push gateway. You must upload the iOS push certificate on the console to access APNs service.

Complete the following steps to configure the iOS push certificate:

1. Log in to the mPaaS console, select the target app, and enter the **Message Push Service** > **Settings** page from the left-side navigation pane.
2. Click the **Push configuration** tab, and in the **iOS channel** area, configure the iOS push certificate.
   - **Certificate file**: Select and upload the pre-prepared iOS push certificate. The backend will obtain the certificate environment and BundleId by parsing the uploaded certificate. For how to generate iOS push certificate, see Create an iOS push certificate.
   - **Certificate password**: Enter the certificate password, which refers to the password you set when exporting the .p12 certificate.
3. Click the **Upload** button to save the settings. If the certificate is in a correct format, you can view the details. You can use Simple push to test whether a certificate is valid and is compatible with the environment.

> **Note**
>
> The iOS push certificate is only valid for a certain period, please renew the certificate before it becomes invalid to keep the message push working normally. You will be prompted to replace the certificate 15 days before the certificate expires. To replace the certificate, you can click **Re-upload** at the bottom to upload a new one.

Configure Android push channels

To improve the arrival rate of pushed messages, mPaaS integrates the push functions some mainstream mobile phone manufacturers such as Huawei, Xiaomi, OPPO, and vivo. Xiaomi Push, Huawei Push, OPPO Push, and vivo Push are used for message push. Notifications can still be sent even if the app is not running. The process can be activated when users tap the notification bar.

> **Note**
>
> Mobile phone manufacturers' proprietary push channels can help apps achieve stable push performance. Therefore, we recommend that you integrate third-party push channels to your app.

This topic describes the console-side configurations required for integrating push channels of Xiaomi, Huawei, OPPO, and vivo.

- Configure Huawei push channel
- Configure Xiaomi push channel
- Configure OPPO push channel
- Configure vivo push channel
- Configure FCM push channel

Configure Huawei push channel

1. On the left-side navigation pane, select the **Message Push Service** > **Settings** menu, and then enter the **Channel configuration** tab page.
2. Click **Configure** in the upper right corner of the **Huawei push channel** area.

Parameter	Required	Description
Status	Yes	Channel integration status, if the status switch is turned on, MPS integrates Huawei Push based on the configurations. If this switch is turned off, the integration is cancelled.
Package name	Yes	Huawei app package name, you can customize the package name. If you don't fill the field, the Xiaomi app package name that you register will be applied by default.
Huawei app ID	Yes	Huawei app ID.
App key	Yes	The secret key (AppSecret) of the Huawei app.

> **Note**
>
> To obtain the App package name, App ID, and secret key, go to Huawei Developer, log in to **Console**, and choose **My Products** > **Mobile App details**.

3. Click **OK** to save the settings.

Configure Xiaomi push channel

1. On the left-side navigation pane, select the **Message Push Service** > **Settings** menu, and then enter the **Channel configuration** tab page.
2. Click **Configure** in the upper right corner of the **Xiaomi push channel** area.

Parameter	Required	Description
Status	Yes	Channel integration status, if the status switch is turned on, MPS integrates MiPush based on the configurations. If this switch is turned off, the integration is cancelled.
Package name	Yes	The name of the main Xiaomi app package.
AppSecret	Yes	The secret key (AppSecret) of the Xiaomi app.

> **Note**
>
> To obtain the app package name and secret key, go to **Xiaomi Developer**, and choose **App management** > **App info**.

3. Click **OK** to save the settings.

Configure OPPO push channel

1. On the left-side navigation pane, select the **Message Push Service** > **Settings** menu, and then enter the **Channel configuration** tab page.
2. Click **Configure** in the upper right corner of the **OPPO push channel** area.

Parameter	Required	Description
Status	Yes	Channel integration status, if the status switch is turned on, MPS integrates OPPO Push based on the configurations. If the switch is turned off, the integration is cancelled.
Package name	Yes	OPPO app package name, you can customize the package name which must be consistent with that on the OPPO PUSH Operation Platform. If you don't fill the field, the Xiaomi app package name that you register will be applied by default.
AppKey	Yes	AppKey is the client ID used when you initialize the client SDK.
MasterSecret	Yes	MasterSecret is the identifier used by developers to authenticate their identities when using server APIs.

> **Note**
>
> After activating OPPO Push on the OPPO PUSH Operation Platform, you can view the AppKey and MasterSecret of the app by logging into OPPO PUSH Operation Platform, and choosing **Push management** > **App configuration**.

3. Click **OK** to save the settings.

Configure vivo push channel

1. On the left-side navigation pane, select the **Message Push Service** > **Settings** menu, and then enter the **Channel configuration** tab page.
2. Click **Configure** in the upper right corner of the **vivo push channel** area.

Parameter	Required	Description
Status	Yes	The integration status switch of the channel. If the switch is turned on, MPS integrates vivo Push based on the configurations. If the switch is turned off, the integration is cancelled.

Package name	Yes	vivo app package name, you can customize the package name which must be consistent with that on the vivo Developers Platform. If you don't fill the field, the Xiaomi app package name that you register will be applied by default.
AppID	Yes	AppID is the client ID used when you initialize the client SDK.
AppKey	Yes	AppKey is the client ID used when you initialize the client SDK.
MasterSecret	Yes	MasterSecret is the identifier used by developers to authenticate their identities when using server APIs.

> **Note**
> After activating the Push service for an app on the vivo Developers Platform, you can obtain the AppId, AppKey, and MasterSecret of the app.

3. Click **OK** to save the settings.

Configure FCM push channel

When you push messages to an Android device outside China, the FCM service is used as the message push gateway. You must configure FCM on the console.

Prerequisites

You have obtained the FCM server key from the Firebase console (**Project settings** > **Cloud messaging** > **Server key**).

Procedure

1. On the left-side navigation pane, select the **Message Push Service** > **Settings** menu, and then enter the **Channel configuration** tab page.
2. Click **Configure** in the upper right corner of the **FCM push channel** area.
   i. Turn on the **Status** switch, MPS accesses the FCM service. If the switch is turned off, the FCM service is disabled in MPS.
   ii. Enter the **FCM server key**. Make sure that this is the server key. Android, iOS, and browser keys will be rejected by FCM.
3. Click **OK** to save the settings.

6.8. Key management

To enhance interaction security between MPS and your business system, MPS will sign and verify all data passed through APIs. In addition, MPS provides a key management page, on which you can perform key configuration.

- Configure push API

MPS provides RESTful APIs. To ensure data security, MPS will verify the caller's identity. Therefore, before calling an API, you must use the RSA algorithm to sign the request and configure a key for identity verification in the **Push API configuration** area on the **Key management** page of the MPS console.

- Configure callback API

  To receive a receipt of the message sending result, configure the URL of the target RESTful callback API in the **Callback API configuration** area on the **Key management** page of the MPS console, and obtain the public key. This is because MPS will sign request parameters when calling a callback API. You need to use the public key to verify the request signature.

Configure push API

Prerequisites

Before configuring the push API, you have used the RSA algorithm to generate a 2048-bit public key.

- RSA public key generation method is as follows:
  i. Download and install the OpenSSL tool (version 1.1.1 or above) from OpenSSL official website.
  ii. Open the OpenSSL tool and use the following command line to generate a 2048-bit RSA private key.

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

  iii. Generate an RSA public key based on the RSA private key.

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

- The signing rules are as follows:
  - Use the SHA-256 signature algorithm.
  - Convert the signature to a base64 string.
  - Replace the plus sign (`+`) and forward slash (`/`) in the base64 string with a minus sign (`-`) to get the final signature.

Procedure

Complete the following steps to configure the push API:

1. Log in to the mPaaS console, select the target app, and go to the **Message Push Service** > **Settings** page.
2. On the right side of the page, click the **Key management** tab to enter the key management page.
3. Click **Configure** in the upper right corner of the **Push API configuration** area.

Field	Required	Description
Status	Yes	Specifies whether to enable the push API. When it is on, the API provided by MPS can be called. When it is off, the API cannot be called.
Encryption method	No	Only the RSA algorithm is available.

RSA public key	No	Enter a 2048-bit public key. After you use a private key to sign request parameters, MPS will use the public key to decrypt them to verify the caller's identity.

> **Important**
>
> Ensure that the public key is set correctly and does not contain spaces. Otherwise, the API call will fail. For more information about API calls, see API reference.

4. Click **OK** to save the settings.

Configure callback API

Log in to the mPaaS console, select the target app, and perform the following steps to configure the callback API:

1. On the **Key management** page, click **Configure** in the upper right corner of the **Callback API configuration** area.

Field	Required	Description
Status	Yes	Specifies whether to enable the callback API. MPS will send a receipt to your server according to the configuration only after the API is enabled.
Callback API URL	Yes	Enter the URL of the callback API. The URL must be an HTTP request URL that can be visited in the public network. MPS uses the private key to sign the POST request body and passes the signed content as the `sign` parameter.
Encryption method	No	MPS uses the RSA algorithm to sign the POST request body.
RSA public key	No	The system automatically sets this parameter and you cannot modify it. After obtaining the POST request body and the `sign` parameter, your server needs to use the public key to verify that the request is sent by MPS and has not been tampered with during data transmission. For more information about signature verification, see API reference > HTTP call.

2. Click **OK** to save the settings.

   The time when MPS executes a callback varies with the push channel.

> **Note**
>
> - Vendor channels (FCM/APNs/Xiaomi/Huawei/OPPO/vivo): A callback is executed when the third-party service is called successfully.
> - MPS self-built channel: A callback is executed when a message is pushed successfully.

Code sample

```
/**
 * 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 2020.03.22 11:20 AM yqj Exp $
 */
@Controller
public class PushCallbackController {

    /**
     * Copy the RSA public key configured for the callback API on the message push cons
ole.
     */
    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);
        // Signature verification
        sign = sign.replace('-','+');
        sign = sign.replace('_','/');
        if(!SignUtil.check(callbackJson,sign,pubKey,"UTF-8")){
            System.out.println("Signature verification failed");
            return;
        }
        System.out.println ("Signature verification succeeded");
        // JSON message body
        System.out.println(callbackJson);

    }

}
```

`callbackJson` specifies the JSON request body. An example is as follows:

```
{
    "extInfo":{
        "adToken":"da64bc9d7d448684ebaeecfec473f612c57579008343a88d4dbdd145dad20e84",
        "osType":"ios"
    },
    "msgId":"console_1584853300103",
    "pushSuccess":true,
    "statusCode":"2",
    "statusDesc":"Acked",
    "targetId":"da64bc9d7d448684ebaeecfec473f612c57579008343a88d4dbdd145dad20e84"
}
```

The following table describes each field in `callbackJson`. You can click here to download the callback code sample.

Field	Description
msgId	The ID of the service message to be pushed.
pushSuccess	Indicates whether the message is pushed successfully.
statusCode	The message status code.
statusDesc	The description of the message status code.
targetId	The target ID.

7.API reference

7.1. Client APIs

Message Push Service involves the following client APIs.

Call method	API	Description
RPC call	Bind	Bind the user ID and device ID (Ad-token).
	Unbind	Unbind the user ID and device ID (Ad-token).
	Report third-party channel devices	Bind the third-party channel device ID (Ad-token).

The `MPPush` class in the intermediate layer of mPaaS encapsulates all the APIs of MPS, including the interfaces for binding users, unbinding users, and reporting three-party channel device information. The API calls are implemented through the mobile gateway SDK.

Bind

- **Method definition**

  This method is used to bind user ID and device ID. After the binding is completed, messages can be pushed in user dimension.

  > **Note**
  > The interface must be called in the child thread.

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

  This method is used to bind the user ID with device ID. Once the user IDs and device IDs are bound, MPS push messages from user dimension.

- **Request parameters**

Parameter	Type	Description
ctx	Context	It must be a non-empty Context.
userId	String	The unique identifier of a user. The user ID is not always the actual identifier in the business system, but there must be one-to-one mapping between the user ID and user.

Parameter	Type	Description
token	String	The device token distributed by the push gateway.

- **Response parameters**

Parameter	Description
success	Whether the interface call is successful or not. ◦ true: Successful ◦ false: Failed
code	Operation result code. For the common operation codes and the corresponding description, see the following Result codes table.
name	Name of the result code
message	Description corresponding to the result code

- **Result codes**

Code	Name	Message	Description
3012	NEED_USERID	need userid	The parameter `userId` is empty when client calls the interface.
3001	NEED_DELIVERYTOKEN	need token	The parameter `token` is empty when client calls the interface.

- **Code sample**

```
  private void doSimpleBind() {
      final ResultPbPB resultPbPB = MPPush.bind(getApplicationContext(), mUserId, PushMsgService.mAdToken);
      handlePbPBResult("Bind users", resultPbPB);
  }
```

Unbind

- **Method definition**

  This method is used to unbind user ID and device ID.

> **Note**
>
> The interface must be called in the child thread.

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

- **Request parameters**

Parameter	Type	Description
ctx	Context	It must be a non-empty Context.
userId	String	The unique identifier of a user. The user ID is not always the actual identifier in the business system, but there must be one-to-one mapping between the user ID and user.
token	String	The device token distributed by the push gateway.

- **Response parameters**

  Refer to the response parameters of Bind API.

- **Code sample**

```
    private void doSimpleUnBind() {
        final ResultPbPB resultPbPB = MPPush.unbind(getApplicationContext()
                , mUserId, PushMsgService.mAdToken);
        handlePbPBResult("Unbind users", resultPbPB);
    }
```

Report third-party channel devices

- **Method definition**

  This method is used to bind the third-party channel device ID and the Ad-token. That is, the third-party channel device identifier and mPaaS device identifier (the Ad-token issued by the MPS gateway) are reported to the mobile push core, and the mobile push core will bind these two identifiers. After completing this process, you can use third-party channels to push messages.

> **Note**
>
> This method will be called once by the framework. To avoid SDK call failure, it is recommended that you call it again manually.

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

- **Request parameters**

Parameter	Type	Description
ctx	Context	It must be a non-empty Context.
deliveryToken	String	The device ID (Ad-token) issued by MPS gateway.
thirdChannel	int	The third-party channel. Valid values include: ◦ 2: Apple ◦ 4: Xiaomi ◦ 5: Huawei ◦ 6: FCM ◦ 7: OPPO ◦ 8: vivo
thirdChannelDeviceToken	String	The ID of a device connected to a third-party channel.

- **Response parameters**

  Refer to the response parameters of Bind API.

- **Code sample**

```
    private void doSimpleUploadToken() {
        final ResultPbPB resultPbPB = MPPush.report(getApplicationContext(),
PushMsgService.mAdToken
                , PushOsType.HUAWEI.value(), PushMsgService.mThirdToken);
        handlePbPBResult("report 3rd-party device ID", resultPbPB);
```

Troubleshooting

If an exception occurs in the process of initiating RPC requests for resources, refer to Security guard result codes.

7.2. Server APIs

Message Push Service (MPS) provides the following OpenAPIs for the server to implement the functions of message push (simple push, template push, multiple push, and broadcast push), message revocation, message statistics and analysis, and scheduled push. As for message push, MPS supports immediate push, timed push, and scheduled push three push strategies to meet the push requirements in different scenarios and reduce repetitive work.

API	Description
Push message - simple push	Pushes one message to one target ID.

Push message - template push	Pushes one message to one target ID. The message is created based on a template.
Push message - multiple push	Pushes different messages to multiple target IDs. Based on the template, configure different template placeholders for the target IDs to Implement personalized message push by use template placeholders based on the template.
Push message - broadcast push	Pushes the same message to all devices. The message is created based on a template.
Revoke messages	Withdraws the pushed messages. Messages pushed through simple push or template push can be withdrawn through message ID; messages pushed through the multiple push or broadcast push can be withdrawn through task ID.
Analyze message push	Queries message push statistical data, including pushed messages, successfully pushed messages, message arrivals, opened messages, and ignored messages, and query the multiple/broadcast push tasks created on MPS console or triggered by calling API as well as the task details.
p	Supports querying the scheduled push task list and canceling the scheduled push task. Scheduled push fall into two types: timed push and cyclic push: • Scheduled push: Pushes messages at a specified time. For example, push messages at 8:00 AM on June 19. • Cyclic push: Pushes messages repeatedly within a specified time period. For example, push messages at 8:00 AM every Friday from June 1 to September 30. A cyclic push task may generate one or more scheduled push tasks.

SDK preparations

MPS supports four programming languages: Java, Python, Node.js, and PHP. Before you call the preceding APIs for message push, you should make different preparations for different programming languages.

The following examples describe the preparations needed before implementing the SDK for different programming languages.

Java

Before you call the preceding four APIs for message push, introduce the Maven configuration. Import the following dependencies to the main `pom` file:

```
<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

Run the following commands to add relevant dependencies.

```
## Aliyun SDK
pip install aliyun-python-sdk-core
## mPaaSs SDK
pip install aliyun-python-sdk-mpaas
```

Node.js

Run the following commands to add relevant dependencies.

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

PHP

Run the following commands to add relevant dependencies.

```
composer require alibabacloud/sdk
```

Push message - simple push

Push one message to one target ID. Before you call this API, you must introduce the required dependencies. For more information, see SDK preparations.

Request parameters

Parameter	Data type	Required	Example	Description
classification	String	No	1	Indicates the type of the messages pushed through vivo push channel: • 0 - Operational message • 1 - System message If not filled, it defaults to 1.

taskName	String	Yes	simpleTest	The name of push task
title	String	Yes	Test	Message title
content	String	Yes	Test	Message body
appId	String	Yes	ONEX570DA89211721	mPaaS app ID
workspaceId	String	Yes	test	mPaaS workspace
deliveryType	Long	Yes	3	The type of target ID. Valid values: • 1 - Android device • 2 - iOS device • 3 - User
targetMsgkey	String	Yes	{“user1024”:”1578807462788”}	Targets to which the message will be pushed, in the map format: • key: The target, which depends on the value of `deliveryType`. ◦ If the value of `deliveryType` is 1, the key is Android device ID. ◦ If the value of `deliveryType` is 2, the key is iOS device ID. ◦ If the value of `deliveryType` is 3, the key is user ID which is the value of `userid` passed in when you called the binding API. • value: The business ID of the message, which is user-defined and must be unique. Note that the number of the targets cannot exceed 10.
expiredSeconds	Long	Yes	300	The validity period of message, in seconds.
pushStyle	Integer	Yes	0	Push style: • 0 - Default • 1 - Big text • 2 - Image and text

extendedParams	String	No	{“key1”:”value1”}	The extension parameters, in the map format.
pushAction	Long	No	0	The redirection method upon a tap on the message. Valid values: • 0: Web URL • 1 - Intent Activity The default redirection method is Web URL.
uri	String	No	http://www	The URL to be redirected to upon a tap on the message.
silent	Long	No	1	Specify whether the message is silent. Valid values: • 1 - Silent • 0 - Not silent
notifyType	String	No		Message push channel: • transparent - MPS self-built channel • notify - Default channel
imageUrls	String	No		Large image link (JSON string), supported in OPPO, HMS, MIUI, FCM and iOS push channels. You can use `defaultUrl` as the default value.
iconUrls	String	No		Icon link (JSON string), supported in OPPO, HMS, MIUI, FCM and iOS push channels. You can use `defaultUrl` as the default value.
strategyType	int	No	1	Push strategy: • 0 - Immediately • 1 - Timed • 2 - Cyclic It is 0 by default.
StrategyContent	String	No		Push strategy details (JSON string). This parameter is required when the value of `strategyType` is not 0. See the following description of the StrategyContent fields.

StrategyContent fields

JSON value is converted to String and passed in.

Parameter	Data type	Required	Example	Description
fixedTime	long	No	1630303126000	Scheduled push timestamp (in ms, accurate to second). When the push strategy is Timed (the value of `strategyType` is 1), `fixedTime` is required.
startTime	long	No	1640966400000	Cycle period start timestamp (in ms, accurate to day). When the push strategy is Cyclic (the value of `strategyType` is 2), `startTime` is required.
endTime	long	No	1672416000000	Cycle period end timestamp (in ms, accurate to day). The end time cannot exceed 180 days after the current day. When the push strategy is Cyclic (the value of `strategyType` is 2), `endTime` is required.
circleType	int	No	3	Loop type: • 1 - Daily • 2 - Weekly • 3 - Monthly When the push strategy is Cyclic (the value of `strategyType` is 2), `circleType` is required.

circleValue	int[]	No	[1,3]	Cycle value: • If the loop type is daily: Empty • If the loop type is weekly: Set the cyclic push time every week. For example, `[1,3]` means pushing the message every Monday and Wednesday. • If the loop type is monthly: Set the cyclic push time every month. For example, `[1,3]` means pushing the message on the 1st and 3rd day every month. When the push strategy is Cyclic (the value of `strategyType` is 2 and the value of `circleType` is not daily), `circleValue` is required.
time	String	No	09:45:11	Cyclic push time (time format: HH:mm:ss). When the push strategy is Cyclic (the value of `strategyType` is 2), `time` is required.

Note

- The upper limit of unexecuted timed or cyclic push tasks is 100 by default.
- The cycle period is from 00:00 at the start date to 24:00 at the end date.
- Neither the cycle start time nor the end time can be earlier than 00:00 of the day, and the end time cannot be earlier than the start time.

Response parameters

Parameter	Data type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	Request ID
ResultCode	String	OK	Request result code
ResultMessage	String	param is invalid	Error description
PushResult	JSON		Request result

Success	boolean	true	Request status. The value of `Success` is contained in the `PushRresult` JSON string.
ResultMsg	String	param is invalid	Error content. The value of `ResultMsg` is contained in the `PushRresult` JSON string.
Data	String	903bf653c1b5442b9ba07684767bf9c2	Scheduled push task ID. When `strategyType` is not 0, this field is not empty.

Java sample code

Click here for information about how to obtain the AccessKey ID and AccessKey secret in the following sample code.

```
 DefaultProfile.addEndpoint("cn-hongkong", "mpaas", "mpaas.cn-hongkong.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize it
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            "*******",      // The AccessKey ID of the RAM account
            "******"); // The AccessKey secret of the RAM account

        IAcsClient client = new DefaultAcsClient(profile);
        // Create an API request and set parameters
        PushSimpleRequest request = new PushSimpleRequest();
        request.setAppId("ONEX570DA89211721");
        request.setWorkspaceId("test");
        request.setTaskName("Test task");
        request.setTitle("Test");
        request.setContent("Test");
        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.al
iyuncs.com/tmp/test.png\",\"miuiUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"fcmUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.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.aliyuncs.com/tmp/test.png\",\"oppoUrl\":\"https://pre-mpaas.oss-cn-hangzhou.al
iyuncs.com/tmp/test.png\",\"miuiUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\"}";
        request.setImageUrls(imageUrls);
        request.setIconUrls(iconUrls);

        request.setStrategyType(2);
```

```
        request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circ
Type\":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 sample code

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

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

# Initialize a request and set parameters
request = PushSimpleRequest.PushSimpleRequest()
request.set_endpoint("mpaas.cn-hongkong.aliyuncs.com")
request.set_AppId("ONEX570DA89211721")
request.set_WorkspaceId("test")
request.set_Title( "Python test")
request.set_Content( "Test 2")
request.set_DeliveryType(3)
request.set_TaskName("The test task of Python template push")
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 sample code

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

const { default: Client, PushSimpleRequest } = sdk;
// Create a client
const client = new Client({
  accessKeyId: '****',
  accessKeySecret: '*****',
  endpoint: 'mpaas.cn-hongkong.aliyuncs.com',
  apiVersion: '2019-08-21'
});
// Initialize the request.
  const request = new PushSimpleRequest();
  request.appId = "ONEX570DA89211721";
  request.workspaceId = "test";
  request.title = "Node test";
  request.content = "Test";
  request.deliveryType = 3;
  request.taskName = "Node test task";
  request.expiredSeconds=600;
  const extendedParam = {
    test: 'Custom extension parameter'
  };
  request.extendedParams = JSON.stringify(extendedParam);
// The value is the ID of the business message. Make sure that the ID is unique.
  const target = {
    "userid1024": String(new Date().valueOf())
  };
  request.targetMsgkey = JSON.stringify(target);

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

PHP sample code

```
<?php

use AlibabaCloud\Client\AlibabaCloud;
use AlibabaCloud\MPaaS\MPaaS;
AlibabaCloud::accessKeyClient('accessKeyId', 'accessKeySecret')
    ->regionId('cn-hongkong')
    ->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 test")
            ->withContent("Test 3")
            ->withDeliveryType(3)
            ->withTaskName("PHP test task")
            ->withExpiredSeconds(600)
            ->withTargetMsgkey(
                json_encode(["userid1024" => "".time() ]
            ))
            // endpoint
            ->host("mpaas.cn-hongkong.aliyuncs.com")
            // Specify whether to enable the debug mode
            ->debug(true)
            ->request();
    }
}
```

Push message - template push

Template push refers to pushing one message to a single target ID. The message is created based on a template. Multiple IDs can share the same template.

Before you call the interface, ensure that you have completed the following operations:

- You have created a template in the MPS console. For more information, see Create a template.
- You have introduced the required dependencies. For more information, see SDK preparations.

Request parameters

Parameter	Data type	Required	Example	Description

classification	String	No	1	Indicates the type of the messages pushed through vivo push channel: • 0 - Operational message • 1 - System message If not filled, it defaults to 1.
taskName	String	Yes	templateTest	The name of push task
appId	String	Yes	ONEX570DA89211721	mPaaS app ID
workspaceId	String	Yes	test	mPaaS workspace
deliveryType	Long	Yes	3	The type of target ID. Valid values: • 1 - Android device • 2 - iOS device • 3 - User
targetMsgkey	String	Yes	{“user1024”:”1578807462788”}	Targets to which the message will be pushed, in the map format: • key: The target, which depends on the value of `deliveryType`. ◦ If the value of `deliveryType` is 1, the key is Android device ID. ◦ If the value of `deliveryType` is 2, the key is iOS device ID. ◦ If the value of `deliveryType` is 3, the key is user ID which is the value of `userid` passed in when you called the binding API. • value: The business ID of the message, which is user-defined and must be unique. Note that the number of the targets cannot exceed 10.
expiredSeconds	Long	Yes	300	The validity period of message, in seconds.
templateName	String	Yes	testTemplate	The name of template. Create a template in the MPS console.

templateKeyValue	String	No	{“money”:”200”,”name”:”Bob”}	The parameters of template, in the map format. The parameters depend on the template specified by `templateName`. Key refers to the placeholder while value refers to the specific value that is used to replace the placeholder. For example, the content of a template can be `Congratulations to #name# for winning RMB #money#`. The string between two number signs “#” is the name of the placeholder.
extendedParams	String	No	{“key1”:”value1”}	The extension parameters, in the map format.
notifyType	String	No		Message push channel: • transparent - MPS self-built channel • notify - Default channel
strategyType	int	No	1	Push strategy: • 0 - Immediately • 1 - Timed • 2 - Cyclic It is 0 by default.
StrategyContent	String	No		Push strategy details (JSON string). This parameter is required when the value of `strategyType` is not 0. See the following description of the StrategyContent fields.

StrategyContent fields

JSON value is converted to String and passed in.

Parameter	type	Required	Example	Description
fixedTime	long	No	1630303126000	Scheduled push timestamp (in ms, accurate to second). When the push strategy is Timed (the value of `strategyType` is 1), `fixedTime` is required.

startTime	long	No	1640966400000	Cycle period start timestamp (in ms, accurate to day). When the push strategy is Cyclic (the value of `strategyType` is 2), `startTime` is required.
endTime	long	No	1672416000000	Cycle period end timestamp (in ms, accurate to day). The end time cannot exceed 180 days after the current day. When the push strategy is Cyclic (the value of `strategyType` is 2), `endTime` is required.
circleType	int	No	3	Loop type: • 1 - Daily • 2 - Weekly • 3 - Monthly When the push strategy is Cyclic (the value of `strategyType` is 2), `circleType` is required.
circleValue	int[]	No	[1,3]	Cycle value: • If the loop type is daily: Empty • If the loop type is weekly: Set the cyclic push time every week. For example, `[1,3]` means pushing the message every Monday and Wednesday. • If the loop type is monthly: Set the cyclic push time every month. For example, `[1,3]` means pushing the message on the 1st and 3rd day every month. When the push strategy is Cyclic (the value of `strategyType` is 2 and the value of `circleType` is not daily), `circleValue` is required.
time	String	No	09:45:11	Cyclic push time (time format: HH:mm:ss). When the push strategy is Cyclic (the value of `strategyType` is 2), `time` is required.

Note

- The upper limit of unexecuted timed or cyclic push tasks is 100 by default.
- The cycle period is from 00:00 at the start date to 24:00 at the end date.
- Neither the cycle start time nor the end time can be earlier than 00:00 of the day, and the end time cannot be earlier than the start time.

Response parameters

Parameter	Data type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	Request ID
ResultCode	String	OK	Request result code
ResultMessage	String	param is invalid	Error description
PushResult	JSON		Request result
Success	boolean	true	Request status. The value of `Success` is contained in the `PushRresult` JSON string.
ResultMsg	String	param is invalid	Error content. The value of `ResultMsg` is contained in the `PushRresult` JSON string.
Data	String	903bf653c1b5442b9ba07684767bf9c2	Scheduled push task ID. When `strategyType` is not 0, this field is not empty.

Java sample code

Click here for information about how to obtain the AccessKey ID and AccessKey secret in the following sample code.

```
 DefaultProfile.addEndpoint("cn-hongkong", "mpaas", "mpaas.cn-hongkong.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize it.
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // region ID
            "******",      // The AccessKey ID of the RAM account
            "******"); // The AccessKey secret of the RAM account

        IAcsClient client = new DefaultAcsClient(profile);
        // Create an API request and set parameters
        PushTemplateRequest request = new PushTemplateRequest();
        request.setAppId("ONEX570DA89211721");
        request.setWorkspaceId("test");
        request.setTemplateName("testTemplate");
        // Hello #name#. Congratulations to you for winning RMB #money#.
        Map<String,String> templatekv = new HashMap<String, String>();
        templatekv.put("name"," Bob");
        templatekv.put("money","200");
        request.setTemplateKeyValue(JSON.toJSONString(templatekv));
        request.setExpiredSeconds(600L);
        request.setTaskName("templateTest");
        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,\"circ
Type\":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 sample code

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

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

# Initialize a request and set parameters
request = PushTemplateRequest.PushTemplateRequest()
request.set_endpoint("mpaas.cn-hongkong.aliyuncs.com")
request.set_AppId("ONEX570DA89211721")
request.set_WorkspaceId("test")
request.set_TemplateName("template1024")
templatekv = {"name":"Bob","money":"200"}
request.set_TemplateKeyValue(json.dumps(templatekv))
request.set_DeliveryType(3)
request.set_TaskName("The test task of Python template push")
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 sample code

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

const { default: Client, PushTemplateRequest } = sdk;
// Create a client.
const client = new Client({
  accessKeyId: 'accessKeyId',
  accessKeySecret: 'accessKeySecret',
  endpoint: 'mpaas.cn-hongkong.aliyuncs.com',
  apiVersion: '2019-08-21'
});
// Initialize the request.
  const request = new PushTemplateRequest();
  request.appId = "ONEX570DA89211721";
  request.workspaceId = "test";
  request.templateName= "template1024";
  const templatekv = {
    name: 'Bob',
    money:'300'
  };
  request.templateKeyValue = JSON.stringify(templatekv);
  request.deliveryType = 3;
  request.taskName = "Node test task";
  request.expiredSeconds=600;
  const extendedParam = {
    test: 'Custom extension parameter'
  };
  request.extendedParams = JSON.stringify(extendedParam);
  const target = {
    "userid1024": String(new Date().valueOf())
  };
  request.targetMsgkey = JSON.stringify(target);

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

PHP sample code

```
<?php

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

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

    public function templatePush() {
        $request = MPaaS::v20190821()->pushTemplate();
        $result = $request->host("mpaas.cn-hongkong.aliyuncs.com")
            // Specify whether to enable the debug mode.
            ->debug(true)
            ->withAppId("ONEX570DA89211721")
            ->withWorkspaceId("test")
            ->withTemplateName("template1024")
            ->withTemplateKeyValue(json_encode(["name" => "Bob", "money" => "200"]))
            ->withDeliveryType(3)
            ->withTaskName("PHP test task")
            ->withExpiredSeconds(600)
            ->withTargetMsgkey(
                json_encode(["userid1024" => "".time() ])
            )
            ->request();
    }
}
```

Push message - multiple push

You can call this API to push different messages to different target IDs. This API allows you to create a personalized message for a target ID by replacing the template placeholders. Different from template push, multiple push allows you to send messages of different content to different target IDs.

Before you call the interface, ensure that you have completed the following operations:

- You have created a template in the MPS console, and the template contains placeholders. Otherwise, you can't implement personalized message push, that is, push different messages to different target IDs. For more information, see Create a template.
- You have introduced the required dependencies. For more information, see SDK preparations.

Request parameters

Parameter	Data type	Required	Example	Description

classification	String	No	1	Indicates the type of the messages pushed through vivo push channel: • 0 - Operational message • 1 - System message If not filled, it defaults to 1.
taskName	String	Yes	multipleTest	The name of push task
appId	String	Yes	ONEX570DA89211721	mPaaS app ID
workspaceId	String	Yes	test	mPaaS workspace
deliveryType	Long	Yes	3	The type of target ID. Valid values: • 1 - Android device • 2 - iOS device • 3: User
templateName	String	Yes	testTemplate	Template name. The template can be created in the MPS console.
targetMsgs	List	Yes	targetMsgs object list	The list of TargetMsg objects. | The list of push targets. For information about the parameters of each object, see targetMsgs objects.
expiredSeconds	Long	Yes	300	The validity period of message, in seconds.
extendedParams	String	No	{“key1”:”value1”}	The extension parameters, in the map format.
notifyType	String	No		Message push channel: • transparent - MPS self-built channel • notify - Default channel

strategyType	int tra	No	1	Push strategy: • 0 – Immediately • 1 - Scheduled • 2 - Cyclic It is 0 by default.
StrategyContent	String	No		Push strategy details (JSON string). This parameter is required when the value of `strategyType` is not 0. See the following description of the StrategyContent fields.

targetMsgs objects

Parameter	Data type	Required	Example	Description
target	String	Yes	userid1024	The target ID, which depends on the value of the `deliveryType` parameter.
msgKey	String	Yes	1578807462788	The ID of business message. The ID is used for message troubleshooting. The ID is user defined and must be unique.
templateKeyValue	String	No	{“money”:”200”,”name”:”Bob”}	The parameters of template, in the map format. The parameters depend on the template specified by `templateName`. Key refers to the placeholder while value refers to the specific value that is used to replace the placeholder. For example, the content of a template can be `Congratulations to #name# for winning RMB #money#`. The string between two number signs “#” is the name of the placeholder.
extendedParams	String	No	{“key1”:”value1”}	The extension parameters, in the map format. Different messages have different extension parameters.

StrategyContent fields

JSON value is converted to String and passed in.

Parameter	Data type	Required	Example	Description

fixedTime	long	No	1630303126000	Scheduled push timestamp (in ms, accurate to second). When the push strategy is Timed (the value of `strategyType` is 1), `fixedTime` is required.
startTime	long	No	1640966400000	Cycle period start timestamp (in ms, accurate to day). When the push strategy is Cyclic (the value of `strategyType` is 2), `startTime` is required.
endTime	long	No	1672416000000	Cycle period end timestamp (in ms, accurate to day). The end time cannot exceed 180 days after the current day. When the push strategy is Cyclic (the value of `strategyType` is 2), `endTime` is required.
circleType	int	No	3	Loop type: • 1 - Daily • 2 - Weekly • 3 - Monthly When the push strategy is Cyclic (the value of `strategyType` is 2), `circleType` is required.
circleValue	int[]	No	[1,3]	Cycle value: • If the loop type is daily: Empty • If the loop type is weekly: Set the cyclic push time every week. For example, `[1,3]` means pushing the message every Monday and Wednesday. • If the loop type is monthly: Set the cyclic push time every month. For example, `[1,3]` means pushing the message on the 1st and 3rd day every month. When the push strategy is Cyclic (the value of `strategyType` is 2 and the value of `circleType` is not daily), `circleValue` is required.
time	String	No	09:45:11	Cyclic push time (time format: HH:mm:ss). When the push strategy is Cyclic (the value of `strategyType` is 2), `time` is required.

> **Note**
>
> - The upper limit of unexecuted timed or cyclic push tasks is 100 by default.
> - The cycle period is from 00:00 at the start date to 24:00 at the end date.
> - Neither the cycle start time nor the end time can be earlier than 00:00 of the day, and the end time cannot be earlier than the start time.

Response parameters

Parameter	Data type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	Request ID
ResultCode	String	OK	Request result code
ResultMessage	String	param is invalid	Error description
PushResult	JSON		Request result
Success	boolean	true	Request status. The value of `Success` is contained in the `PushRresult` JSON string.
ResultMsg	String	param is invalid	Error content. The value of `ResultMsg` is contained in the `PushRresult` JSON string.
Data	String	903bf653c1b5442b9ba07684767bf9c2	Scheduled push task ID. When `strategyType` is not 0, this field is not empty.

Java sample code

Click here for information about how to obtain the AccessKey ID and AccessKey secret in the following sample code.

```
 DefaultProfile.addEndpoint("cn-hongkong", "mpaas", "mpaas.cn-hongkong.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize it
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            "******",      // The AccessKey ID of the RAM account
            "******"); // he AccessKey secret of the RAM account

        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("multipleTest");
        request.setTemplateName("testTemplate");
        // Hello #name#. Congratulations to you for winning RMB #money#.
        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", "Bob");
        templatekv.put("money", "200");
        targetMsg.setTemplateKeyValue(JSON.toJSONString(templatekv));
        // The number of TargetMsg objects can be up to 400
        targetMsgs.add(targetMsg);
        request.setTargetMsgs(targetMsgs);
        request.setExpiredSeconds(600L);

        request.setStrategyType(2);
        request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circ
Type\":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());  // Push task ID or
scheduled push task ID
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

Python sample code

```
# -*- coding: utf8 -*-
from aliyunsdkcore.client import AcsClient
from aliyunsdkmpaas.request.v20190821 import PushMultipleRequest
import json
import time

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

# Initialize a request and set parameters
request = PushMultipleRequest.PushMultipleRequest()
request.set_endpoint("mpaas.cn-hongkong.aliyuncs.com")
request.set_AppId("ONEX570DA89211721")
request.set_WorkspaceId("test")
request.set_TemplateName("template1024")
request.set_DeliveryType(3)
request.set_TaskName("The test task of Python template push")
request.set_ExpiredSeconds(600)
msgkey = str(time.time())
targets = [
  {
    "Target": "user1024",
    "MsgKey": msgkey,
    "TemplateKeyValue": {
      "name": "Bob",
      "money": "200"
    }
  }
]
request.set_TargetMsgs(targets)
# Print response
response = client.do_action_with_exception(request)
print response
```

Node.js sample code

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

const { default: Client, PushMultipleRequest,PushMultipleRequestTargetMsg } = sdk;
// Create a client
const client = new Client({
  accessKeyId: 'accessKeyId',
  accessKeySecret: 'AccessKey Secret',
  endpoint: 'mpaas.cn-hongkong.aliyuncs.com',
  apiVersion: '2019-08-21'
});
// Initialize request
  const request = new PushMultipleRequest();
  request.appId = "ONEX570DA89211721";
  request.workspaceId = "test";
  request.templateName= "template1024";
  const templatekv = {
    name: 'Bob',
    money:'300'
  };
  //request.templateKeyValue = JSON.stringify(templatekv);

  request.deliveryType = 3;
  request.taskName = "Node test task";
  request.expiredSeconds=600;
  const extendedParam = {
    test: 'Custom extension parameter'
  };
  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];

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

PHP sample code

```
<?php

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

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


   public function multiPush() {
        $request = MPaaS::v20190821()->pushMultiple();
        $result = $request->host("mpaas.cn-hongkong.aliyuncs.com")
            // Specify whether to enable the debug mode
            ->debug(true)
            ->withAppId("ONEX570DA89211721")
            ->withWorkspaceId("test")
            ->withTemplateName("template1024")
            ->withDeliveryType(3)
            ->withTaskName("The test task of PHP multiple push")
            ->withExpiredSeconds(600)
            ->withTargetMsg(
                [
                    [
                        "Target" => "userid1024",
                        "MsgKey" => "" . time(),
                        "TemplateKeyValue" => json_encode([
                            "name" => "Bob",
                            "money" => "200",
                        ])
                    ]
                ]
            )
            ->request();
    }
}
```

Push message - broadcast push

You can call this interface to push the same message to all devices. The message is created based on a template.

Before you call the interface, ensure that you have completed the following operations:

- You have created a template in the MPS console, and the template contains placeholders. Otherwise, you cann't implement personalized message push, that is, push different messages to different target IDs. For more information, see Create a template.

- You have introduced the required dependencies. For more information, see SDK preparations.

Request parameters

Parameter	Data type	Required	Example	Description
classification	String	No	1	Indicates the type of the messages pushed through vivo push channel: • 0 - Operational message • 1 - System message If not filled, it defaults to 1.
taskName	String	Yes	broadcastTest	The name of push task
appId	String	Yes	ONEX570DA89211721	mPaaS app ID
workspaceId	String	Yes	test	mPaaS workspace
deliveryType	Long	Yes	1	The type of target ID. Valid values: • 1 - Android broadcast • 2 - iOS broadcast
msgkey	String	Yes	1578807462788	The ID of business message. The ID is used for message troubleshooting. The ID is user defined and must be unique.
expiredSeconds	Long	Yes	300	The validity period of message, in seconds.
templateName	String	Yes	broadcastTemplate	Template name. The template can be created in the MPS console.
templateKeyValue	String	No	{“content”:”Announcement”}	The parameters of template, in the map format. The parameters depend on the template specified by `templateName`. Key refers to the placeholder while value refers to the specific value that is used to replace the placeholder.

pushStatus	Long	No	0	Login status: • 0 - Login users (default) • 1 - All users (including login and logout users) • 2 - Logout users
bindPeriod	int	No		Login period, required when the value of `pushStatus` is 0: • 1 - Login users in recent 7 days • 2 - Login users in recent 15 days • 3 - Login users in recent 60 days • 4 - Permanent **Note** The `bindPeriod` parameter is only configurable in non-financial environment.
unBindPeriod	Long	No		Logout period, required when the value of `pushStatus` is 1 or 2: • 1 - Logout users in recent 7 days • 2 - Logout users in recent 15 days • 3 - Logout users in recent 60 days • 4 - Permanent
androidChannel	Integer	No		Android message channel: • transparent - MPS self-built channel • notify - Default channel
strategyType	int	No	1	Push strategy: • 0 - Immediately • 1 - Scheduled • 2 - Cyclic It is 0 by default.
StrategyContent	String	No		Push strategy details (JSON string). This parameter is required when the value of `strategyType` is not 0. See the following description of the StrategyContent fields.

StrategyContent fields

JSON value is converted to String and passed in.

Parameter	Data type	Required	Example	Description
fixedTime	long	No	1630303126000	Scheduled push timestamp (in ms, accurate to second). When the push strategy is Timed (the value of `strategyType` is 1), `fixedTime` is required.
startTime	long	No	1640966400000	Cycle period start timestamp (in ms, accurate to day). When the push strategy is Cyclic (the value of `strategyType` is 2), `startTime` is required.
endTime	long	No	1672416000000	Cycle period end timestamp (in ms, accurate to day). The end time cannot exceed 180 days after the current day. When the push strategy is Cyclic (the value of `strategyType` is 2), `endTime` is required.
circleType	int	No	3	Loop type: • 1 - Daily • 2 - Weekly • 3 - Monthly When the push strategy is Cyclic (the value of `strategyType` is 2), `circleType` is required.
circleValue	int[]	No	[1,3]	Cycle value: • If the loop type is daily: Empty • If the loop type is weekly: Set the cyclic push time every week. For example, `[1,3]` means pushing the message every Monday and Wednesday. • If the loop type is monthly: Set the cyclic push time every month. For example, `[1,3]` means pushing the message on the 1st and 3rd day every month. When the push strategy is Cyclic (the value of `strategyType` is 2 and the value of `circleType` is not daily), `circleValue` is required.

time	String	No	09:45:11	Cyclic push time (time format: HH:mm:ss). When the push strategy is Cyclic (the value of `strategyType` is 2), `time` is required.

> **Note**
>
> - The upper limit of unexecuted timed or cyclic push tasks is 100 by default.
> - The cycle period is from 00:00 at the start date to 24:00 at the end date.
> - Neither the cycle start time nor the end time can be earlier than 00:00 of the day, and the end time cannot be earlier than the start time.

Response parameters

Parameter	Data type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	Request ID
ResultCode	String	OK	Request result code
ResultMessage	String	param is invalid	Error description
PushResult	JSON		Request result
Success	boolean	true	Request status. The value of `Success` is contained in the `PushRresult` JSON string.
ResultMsg	String	param is invalid	Error content. The value of `ResultMsg` is contained in the `PushRresult` JSON string.
Data	String	903bf653c1b5442b9ba07684767bf9c2	Scheduled push task ID. When `strategyType` is not 0, this field is not empty.

Java sample code

Click here for information about how to obtain the AccessKey ID and AccessKey secret in the following sample code.

```
DefaultProfile.addEndpoint("cn-hongkong", "mpaas", "mpaas.cn-hongkong.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize it
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            "******",      // The AccessKey ID of the RAM account
            "******"); // The AccessKey Secret of the RAM account

        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("broadcastTest ");
        request.setTemplateName("broadcastTemplate ");
        // This is an announcement: #content#.
        Map<String, String> templatekv = new HashMap<String, String>();
        templatekv.put("content", " The content of the announcement ");
        request.setTemplateKeyValue(JSON.toJSONString(templatekv));

        request.setStrategyType(2);
        request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circ
Type\":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());  // push task ID or
scheduled task ID
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

Python sample code

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

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

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

# Initialize a request and set parameters
request = PushBroadcastRequest.PushBroadcastRequest()
request.set_endpoint("mpaas.cn-hongkong.aliyuncs.com")
request.set_AppId("ONEX570DA89211720")
request.set_WorkspaceId("test")
request.set_TemplateName("broadcastTemplate")
templatekv = {"content":"This is an announcement"}
request.set_TemplateKeyValue(json.dumps(templatekv))
request.set_DeliveryType(1)
request.set_TaskName("The test task of Python broadcast push")
request.set_ExpiredSeconds(600)
request.set_Msgkey(str(time.time()))

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

Node.js sample code

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

const { default: Client, PushBroadcastRequest } = sdk;
// Create a client.
const client = new Client({
  accessKeyId: 'accessKeyId',
  accessKeySecret: 'AccessKey Secret',
  endpoint: 'mpaas.cn-hongkong.aliyuncs.com',
  apiVersion: '2019-08-21'
});
// Initialize the request.

  const request = new PushBroadcastRequest();
  request.appId = "ONEX570DA89211720";
  request.workspaceId = "test";
  request.templateName= "broadcastTemplate";
  const templatekv = {
    content: 'This is an announcement',
  };
  request.templateKeyValue = JSON.stringify(templatekv);
  request.deliveryType = 1;
  request.taskName = "Node test task";
  request.expiredSeconds=600;
  const extendedParam = {
    test: 'Custom extension parameter'
  };
  request.extendedParams = JSON.stringify(extendedParam);

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

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

PHP sample code

```
<?php

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

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


   public function broadcastPush(){
        $request = MPaaS::v20190821()->pushBroadcast();
        $result = $request->host("mpaas.cn-hongkong.aliyuncs.com")
            // Specify whether to enable the debug mode.
            ->debug(true)
            ->withAppId("ONEX570DA89211720")
            ->withWorkspaceId("test")
            ->withTemplateName("broadcastTemplate")
            ->withTemplateKeyValue(
                json_encode(["content" => "This is an announcement"])
            )
            ->withDeliveryType(1)
            ->withTaskName("The test task of PHP broadcast push")
            ->withExpiredSeconds(600)
            ->withMsgkey(". time())
            ->request();
    }
}
```

Revoke messages

Messages pushed through simple push or template push can be withdrawn through message ID; messages pushed through the multiple push or broadcast push can be withdrawn through task ID. Only the messages pushed in recent 7 days can be revoked.

Revoke by message ID

Revoke the messages pushed through simple push mode or template push mode.

Request parameters

Parameter	Data type	Required	Example	Description

messageId	String	Yes	1578807462788	Message ID in business system, which can be customized by users and is used to uniquely identify the message in the business system.
targetId	String	Yes	user1024	Target ID. If the message was pushed by device, then the target ID refers to device ID; if the message was pushed by user, then the target ID refers to user ID.

Response parameters

Parameter	Data type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	Request ID
ResultCode	String	OK	Request result code
ResultMessage	String	param is invalid	Error description
PushResult	JSON		Request result
Success	boolean	true	Request status. The value of `Success` is contained in the `PushRresult` JSON string.
ResultMsg	String	param is invalid	Error content. The value of `ResultMsg` is contained in the `PushRresult` JSON string.

Sample code

```
DefaultProfile.addEndpoint("cn-hongkong", "mpaas", "mpaas.cn-hongkong.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize it
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            "******",      // The AccessKey ID of the RAM account
            "******"); // The AccessKey Secret of the RAM account

        IAcsClient client = new DefaultAcsClient(profile);

        RevokePushMessageRequest request = new RevokePushMessageRequest();
        request.setAppId("ONEX570DA89211720");
        request.setWorkspaceId("test");
        request.setMessageId("console_1624516744112");  // Message ID in business
system
        request.setTargetId("mpaas_push_demo");         // Target ID

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

Revoke by task ID

Revoke the messages pushed through multiple push mode or broadcast push mode.

Request parameters

Parameter	Data type	Required	Example	Example
taskId	String	Yes	20842863	Push task ID, which can be used to query push tasks in the MPS console.

Response parameters

Parameter	Data type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	Request ID
ResultCode	String	OK	Request result code

ResultMessage	String	param is invalid	Error description
PushResult	JSON		Request result
Success	boolean	true	Request status. The value of `Success` is contained in the `PushRresult` JSON string.
ResultMsg	String	param is invalid	Error content. The value of `ResultMsg` is contained in the `PushRresult` JSON string.

Sample code

```
DefaultProfile.addEndpoint("cn-hongkong", "mpaas", "mpaas.cn-hongkong.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize it
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            "*******",      // The AccessKey ID of the RAM account
            "*******"); // The AccessKey Secret of the RAM account

        IAcsClient client = new DefaultAcsClient(profile);

        RevokePushTaskRequest request = new RevokePushTaskRequest();
        request.setAppId("ONEX570DA89211720");
        request.setWorkspaceId("test");
        request.setTaskId("20842863");     // Push task ID

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

Analyze message push

Query statistical data

Query message push statistical data, including pushed messages, successfully pushed messages, message arrivals, opened messages, and ignored messages.

Request parameters

Parameter	Data type	Required	Example	Description

appId	String	Yes	ONEX570DA89211721	mPaaS app ID
workspaceId	String	Yes	test	mPaaS workspace
startTime	long	Yes	1619798400000	The start timestamp of the time period to be queried, in milliseconds and accurate to day.
endTime	long	Yes	1624358433000	The end timestamp of the time period to be queried, in milliseconds and accurate to day. The interval between the start time and end time cannot exceed 90 days.
platform	String	No	ANDROID	Push platform. It defaults to query all platforms if no value is passed in. Valid values: IOS, ANDROID
channel	String	No	ANDROID	Push channel. It defaults to query all channels if no value is passed in. Valid values: IOS, FCM, HMS, MIUI, OPPO, VIVO, ANDROID (self-built channel)
type	String	No	SIMPLE	Push mode. It defaults to query all types if no value is passed in. Valid values: SIMPLE, TEMPLATE, MULTIPLE, BROADCAST
taskId	String	No	20842863	Push task ID

Response parameters

Parameter	Data type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	Request ID
ResultCode	String	OK	Request result code
ResultMessage	String	param is invalid	Error description

ResultContent	JSON		Response content
data	JSON		Response content. The value of `data` is contained in the `ResultContent` JSON string.
pushTotalNum	float	100	The number of pushed messages
pushNum	float	100	The number of successfully pushed messages
arrivalNum	float	100	The number of messages that arrive client
openNum	float	100	The number of opened messages
openRate	float	100	Message open rate
ignoreNum	float	100	The number of ignored messages
ignoreRate	float	100	Message ignorance rate

Sample code

```
DefaultProfile.addEndpoint("cn-hongkong", "mpaas", "mpaas.cn-hongkong.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize it
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            "******",      // The AccessKey ID of the RAM account
            "******"); // The AccessKey Secret of the RAM account
        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();
        }
```

Query push tasks

Query the multiple/broadcast push tasks created on MPS console or triggered by calling API.

Request parameters

Parameter	Data type	Required	Example	Description
appId	String	Yes	ONEX570DA89211721	mPaaS app ID
workspaceId	String	Yes	test	mPaaS workspace
startTime	long	Yes	1619798400000	The start timestamp of the time period to be queried, in milliseconds and accurate to day.
taskId	String	No	20842863	Push task ID
taskName	String	No	Test task	Task name

pageNumber	int	No	1	Page number, 1 by default.
pageSize	int	No	10	The total number of pages, 500 by default.

Response parameters

Parameter	Data type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	Request ID
ResultCode	String	OK	Request result code
ResultMessage	String	param is invalid	Error description
ResultContent	JSON		Response content
data	JSON		Response content. The value of `data` is contained in the `ResultContent` JSON string.
taskId	String	20927873	Task ID
taskName	String	Test task	Task name
templateId	String	9108	Template ID
templateName	String	Test template	Template name
type	long	3	Push mode: • 2 – Multiple push • 3 – Broadcast push
gmtCreate	long	1630052750000	Creation time

Sample code

```
DefaultProfile.addEndpoint("cn-hongkong", "mpaas", "mpaas.cn-hongkong.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize it
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            "******",      // The AccessKey ID of the RAM account
            "******"); // The AccessKey Secret of the RAM account
        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("Tesk task");
        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();
        }
```

Query push task details

Query the details of multiple/broadcast push tasks created on MPS console or triggered by calling API.

Request parameters

Parameter	Data type	Required	Example	Description
appId	String	Yes	ONEX570DA89211721	mPaaS app ID
workspaceId	String	Yes	test	mPaaS workspace
taskId	String	Yes	20842863	Push task ID

Response parameters

Parameter	Data type	Example	Description

RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	Request ID
ResultCode	String	OK	Request result code
ResultMessage	String	param is invalid	Error description
ResultContent	JSON		Response content
data	JSON		Response content. The value of `data` is contained in the `ResultContent` JSON string.
taskId	long	20927872	Task ID
pushNum	float	10	The number of pushed messages
pushSuccessNum	float	10	The number of successfully pushed messages
pushArrivalNum	float	10	The number of messages that arrive client
startTime	long	1630052735000	Start time (ms)
endTime	long	1630052831000	End time (ms)
duration	string	00 hour 01 minute 36 seconds	Push duration

Sample code

```
DefaultProfile.addEndpoint("cn-hongkong", "mpaas", "mpaas.cn-hongkong.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize it
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            "******",      // The AccessKey ID of the RAM account
            "******"); // The AccessKey Secret of the RAM account
        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();
        }
```

Manage scheduled push tasks

Query scheduled push tasks

Query the created scheduled push tasks, including timed and cyclic push tasks.

Request parameters

Parameter	Data type	Required	Example	Description
appId	String	Yes	ONEX570DA89211721	mPaaS app ID
workspaceId	String	Yes	test	mPaaS workspace
startTime	long	Yes	1619798400000	The start timestamp when the scheduled push is triggered, not the task creation time.
endtTime	long	Yes	1630425600000	The end timestamp when the scheduled push is triggered.

type	int	No	0	Push mode: • 0 - Simple push • 1 – Template push • 2 – Multiple push • 3 – Broadcast push
uniqueId	String	No	49ec0ed5a2a642bcbe139a2d7a419d6d	The unique ID of the scheduled push task. If you pass the master task ID, then the information of all sub tasks will be returned. If you pass the sub task ID, then the corresponding sub task information will be returned.
pageNumber	int	No	1	Page number, 1 by default.
pageSize	int	No	10	The total number of pages, 500 by default.

Response parameters

Parameter	Data type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	Request ID
ResultCode	String	OK	Request result code
ResultMessage	String	param is invalid	Error description
ResultContent	JSON		Response content
data	JSON		Response content. The value of `data` is contained in the `ResultContent` JSON string.
totalCount	int	10	Total amount
list	JSONArray		Task array

uniqueId	String	56918166720e46e1bcc40195c9ca71db	Unique ID of the scheduled push task. • If the value of `strategyType` is 1, it refers to the master task ID of timed task. • If the value of `strategyType` is 2, it refers to the child task ID of cyclic task.
parentId	String	56918166720e46e1bcc40195c9ca71db	Master ID of the scheduled push task. • If the value of `strategyType` is 1, it refers to the master task ID of timed task. • If the value of `strategyType` is 2, it refers to the master task ID of cyclic task.
pushTime	Date	1630486972000	Scheduled push time
pushTitle	String	Test	Title of message
pushContent	String	Test text	Body content of message
type	int	0	Push mode: • 0 - Simple push • 1 – Template push • 2 – Multiple push • 3 – Broadcast push
deliveryType	int	1	Push type: • 1 - Android • 2 - iOS • 3 - UserId
strategyType	int	1	Push strategy: • 1 - Scheduled • 2 - Cyclic
executedStatus	int	0	Whether the task has been executed: • 0 - No executed • 1 - Executed

createType	int	0	Task creation method: • 0 - API • 1 - Console
gmtCreate	Date	1629971346000	Creation time

Sample code

```
DefaultProfile.addEndpoint("cn-hongkong", "mpaas", "mpaas.cn-hongkong.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize it
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            "******",      // The AccessKey ID of the RAM account
            "******"); // The AccessKey Secret of the RAM account
        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();
        }
```

Cancel scheduled push tasks

Cancel the scheduled push tasks (including cyclic push tasks) that haven't been pushed. You can cancel the tasks in batch.

Request parameters

Parameter	Data type	Required	Example	Description
appId	String	Yes	ONEX570DA89211721	mPaaS app ID

workspaceId	String	Yes	test	mPaaS workspace
type	int	No	0	Scheduled push task ID type. It is 0 by default. • 0 - Master task ID, corresponding to `parentId` • 1 - Sub task ID, corresponding to `uniqueId`
uniqueIds	String	Yes	714613eb,714613ec,714613ed	The unique ID of the scheduled push task. Multiple task IDs are separated with commas (,). You can input 30 IDs at most.

Response parameters

Parameter	Data type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	Request ID
ResultCode	String	OK	Request result code
ResultMessage	String	param is invalid	Error description
ResultContent	String	{714613eb=1,714613ed=0}	Cancellation result: • 1 - Successful • 0 - Failed

Sample code

```
DefaultProfile.addEndpoint("cn-hongkong", "mpaas", "mpaas.cn-hongkong.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize it
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            "******",      // The AccessKey ID of the RAM account
            "******"); // The AccessKey Secret of the RAM account
        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();
        }
```

Extension parameters

Extension parameters are passed to the client with message body. You can define or process these parameters.

Extension parameters include the following three types:

- **System extension parameters**

  These extension parameters are occupied by the system. Do not modify the values of these parameters. System extension parameters include `notifyType`, `action`, `silent`, `pushType`, `templateCode`, `channel`, and `taskId`.

- **System extension parameters with some significance**

  Extension parameters of this type are occupied by the system. Each parameter has a specific meaning. You can set the values of these extension parameters. The following table describes the extension parameters with specific meanings:

Key	Description
sound	The custom ringtone of the message. The value of this parameter is the path of the ringtone. This parameter only applies to Xiaomi phones and iPhones.

badge	The badge of the app icon. Its value is a specific number. This extension parameter will be passed to the client together with the message body. ◦ For Android devices, you need to implement the badge logic by yourself. ◦ For iOS devices, the system automatically implements the badge logic. When a message is pushed to the target mobile phone, the number that you specified in value appears in the badge of the app icon.
mutable-content	Custom push ID of Apple Push Notification service (APNs). A push notification carrying this parameter indicates the support of iOS 10 `UNNotificationServiceExtension`. If the push notification not carrying this parameter indicates a common push. Set the value to 1.
badge_add_num	Number of added push badges for Huawei push channel.
badge_class	Activity class corresponding to the desktop icon for Huawei push channel
big_text	Big text style. This parameter has a fixed value "1". Any other value is invalid. This parameter is only valid for Xiaomi and Huawei devices.

- **User-defined extension parameters**

  All the parameters other than the preceding system extension parameters are user-defined extension parameters. User-defined extension parameters are passed to the client together with a message body. You can define and process these parameters.

Result codes

Result code	Message	Description
100	SUCCESS	Succeeded
-1	SIGNATURE_MISMATCH	Signature mismatched.
3001	NEED_DELIVERYTOKEN	deliveryToken is empty.
3002	NEED_FILE	The file is empty.
3003	NEED_APPID_WORKSPACEID	The app ID or workspace is empty.
3007	APPID_WRONG	Invalid app ID or workspace.

3008	OS_TYPE_NOT_SUPPORTED	Push platform not supported.
3009	DELIVERY_TYPE_NOT_SUPPORTED	deliveryType not supported.
3012	NEED_USERID	UserId is empty.
3019	TASKNAME_NULL	Task name is empty.
3020	EXPIREDSECONDS_WRONG	Illegal message timeout length.
3021	TOKEN_OR_USERID_NULL	Target is empty.
3022	TEMPLATE_NOT_EXIST	Template doesn't exist.
3023	TEMPLATEKV_NOT_ENOUGH	Template parameter mismatched.
3024	PAYLOAD_NOT_ENOUGH	Title or content is empty.
3025	NEED_TEMPLATE	Template is empty.
3026	EXPIREDTIME_TOO_LONG	The validity period of message is too long.
3028	INVALID_PARAM	Illegal parameter.
3029	SINGLE_PUSH_TARGET_TOO_MUCH	Too many targets.
3030	BROADCAST_ONLY_SUPPORT_BY_DEVICE	Only broadcast push by device is supported.
3031	REQUEST_SHOULD_BE_UTF8	The request body must be UTF-8 encoded.
3032	REST_API_SWITCH_NOT_OPEN	The push API has been closed.
3033	UNKNOWN_REST_SIGN_TYPE	Signature type not supported.
3035	EXTEND_PARAM_TO_MUCH	Too many extension parameters. A maximum of 20 extension parameters are allowed.

3036	TEMPLATE_ALREADY_EXIST	The template already exists.
3037	TEMPLATE_NAME_NULL	Template name is empty.
3038	TEMPLATE_NAME_INVALID	Illegal template name.
3039	TEMPLATE_CONTENT_INVALID	Illegal template content.
3040	TEMPLATE_TITLE_INVALID	Illegal template title.
3041	TEMPLATE_DESC_INFO_INVALID	Illegal template description.
3042	TEMPLATE_URI_INVALID	Illegal template URI.
3043	SINGLE_PUSH_CONTENT_TOO_LONG	Message body is too long.
3044	INVALID_EXTEND_PARAM	Illegal extension parameter.
3049	MULTIPLE_INNER_EXTEND_PARAM_TO_MUCH	The number of internal extension parameters for multiple push cannot exceed 10.
3050	MSG_PAYLOAD_TOO_LONG	Message body is too long.
3051	BROADCAST_ALL_USER_NEED_UNBIND_PERIOD	Unbinding parameters are required for the broadcast push targeting at all users (including both login and logout users).
3052	BROADCAST_ALL_USER_UNBIND_PERIOD_INVALID	Illegal unbinding parameters for broadcast push.
3053	BROADCAST_ALL_USER_NOT_SUPPORT_SELFCHANNEL_ANDROID	MPS self-built push channel doesn't supports the broadcast push targeting at all users (including both login and logout users).
3054	DELIVERYTOKEN_INVALID	Illegal MPS self-built channel token.
3055	MULTIPLE_TARGET_NUMBER_TOO_MUCH	The number of push targets exceeds the threshold.

3056	TEMPLATE_NUM_TOO_MUCH	The number of message templates exceeds the threshold.
3057	ANDROID_CHANNEL_PARAM_INVALID	Invalid `androidChannel`.
3058	BADGE_ADD_NUM_INVALID	Invalid `badge_add_num`.
3059	BADGE_ADD_NUM_NEED_BADGE_CLASS	The parameter `badge_class` is required for `badge_add_num`.
9000	SYSTEM_ERROR	System error.

8.Message content restrictions

To ensure effective message delivery, you should create message push tasks with reference to the message content limits for different push channels in the process of pushing messages.

To ensure effective message delivery, you should create message push tasks with reference to the message content limits for different push channels in the process of pushing messages.

Android push channel

Push channel	Message title length limit	Message body length limit
MPS self-built channel	No limit	No limit
Mi	50 characters	128 characters
Huawei	40 characters	1024 characters
OPPO	32 characters	200 characters
vivo	40 characters	100 characters

Note

- Pushes through vendor channels will fail if corresponding length limits are exceeded.
- Pushes through vendor channels will fail if the message title or content is empty.
- For the pushes through Android push channel (no matter vendor channels or MPS self-built channel), the size of the pushed message cannot exceed 2 KB.

iOS push channel

Push channel	Message title length limit	Message body length limit

APNs	40 characters, excess parts will be displayed as an ellipsis.	• Up to 110 characters will be displayed in the Notification Center, and excess parts will be displayed as an ellipsis. • Up to 110 characters will be displayed when the phone screen is locked, and excess parts will be displayed as an ellipsis. • Up to 62 characters will be displayed in the top pop-up window, and excess parts will be displayed as an ellipsis.

Note

For the pushes through iOS push channel, the size of the pushed message cannot exceed 2 KB.

9.FAQ

This topic summarizes the common problems that may appear in the process of integrating and using Message Push Service, and provides the corresponding solutions to solve those problems.

General questions

Description on permissions

For Android 6.0 and later versions, users need to manually grant permissions to the phone, such as reading/writing SD cards. To send messages more precisely, we recommend that developers provide a guide to users on how to grant the required permissions for the notifications.

Logs cannot be printed

For Meizu phones, if `log.d` and `log.i` cannot be printed, you can choose **Settings** > **Accessibility Options** > **Developer Options** and turn on **Advanced Log Output**.

In case of development issues, you can set `tag=mpush` to filter logs.

Android related questions

Port resolution problems in baseline versions 10.1.60.5 ~ 10.1.60.7

In private cloud environments, for the message push using ports other than 443, the resolution of server configurations will fail, and cause connection errors.

Solution:

- If you use the config file for packaging, modify the config file as follows:

```
  //Ignore the rest of the config file and add \\{white space} before the custom port
number.
  {
      "pushPort":"\\ 8000",
  }
```

- If you do not use the config file for packaging, change the value of `rome.push.port` in `AndroidManifest.xml` as follows:

```
  //Add \{white space} before the port number.
  <meta-data
      android:name="rome.push.port"
      android:value="\ 8000" />
```

Failed to push messages after accessing Huawei, Xiaomi and other third-party channels

You need to turn on the settings for the corresponding channels in the mPaaS Message Push Service console. Refer to Code sample for sample code, usage and notes.

Notes on the generation of push ad-token (deviceId)

The server generates deviceId with dependency on IMSI and IMEI. So, you are suggested guide the users to grant the “READ_PHONE_STATE” permission.

Does message push on the notification bar have version restrictions for EMUI and Huawei mobile services?

There are version restrictions for Emotion UI and Huawei mobile services. Emotion UI, EMUI for short, is an emotional operating system based on Android and is developed by Huawei.

For detailed version requirements, see Conditions for devices to receive Huawei notifications.

Cannot print logs for Huawei phones

In the dialing UI of the phone, enter ***#*#2846579#*#*** to enter **Project** menu > **Background settings** > **LOG settings** and select **AP Logs**. After the phone restarts, Logcat will start to take effect.

What should I do when my Huawei phone receives a push error code?

For more information about error codes, see Client error code description and Server error code description on Huawei official website.

Models and system versions supported by OPPO Push

Currently, OPPO phone models running **ColorOS 3.1** and newer systems, **OnePlus 5/5T** and newer phone models, and **all realme** phone models are supported.

ColorOS is a highly-customized, efficient, intelligent, and richly-designed Android-based mobile OS by OPPO.

What should I do when my OPPO phone receives a push error code?

When OPPO push does not work, you can search for “OPPO onRegister error =” in client logs to obtain the error code. Then find the corresponding causes by referring to OPPO error codes.

Models and system versions supported by vivo Push

The models and oldest system versions supported by vivo Push are listed in the following table. For other questions on vivo push, see vivo Push FAQs.

Device model	Android version	Version for system test	Minimum version supported
Android 9.0 and later versions are supported by default			
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 Standard	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 Z1Youth	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	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

What should I do when my vivo phone receives a push error code?

When vivo Push does not work, you can search for "fail to turn on vivo Push state =" in client logs to obtain the status code and find the specific causes by referring to Public status codes.

Troubleshooting procedure for common Android problems

1. Check whether the `Manifest` file is configured correctly.
2. Check whether the appId (Huawei, Xiaomi, or vivo), appSecret (Xiaomi or OPPO), appKey (OPPO or vivo), and ALIPUSH_APPID (mPaaS) are consistent with the app registration information on the corresponding development platform.
3. Check the Logcat logs tagged as mpush.

iOS related questions

Whether there will be a banner or sound alert for messages when the app runs in the foreground

The default mechanism for Apple is that when an app is in foreground, the messages can arrive but will be not shown. In order to show messages in foreground, you need to implement it manually.

Message status is NoBindInfo

NoBindInfo means the user pushes messages by UserId, but no corresponding information is found based on the UserId. Please check if the client has called the binding API, and if the corresponding appId and workspaceId are consistent.

Message status is BadDeviceToken

This status will only appear for iOS pushes, indicating that the actually pushed token is invalid. First, check if the environment of the certificate is correct.

- If the app is packaged with a development certificate, the push console configuration requires a development environment certificate, while Xcode requires a developer certificate for debugging in real devices.
- If the app is packaged with a production certificate, the push console configuration requires a production environment certificate.

Message status is DeviceTokenNotForTopic

This status will only appear for iOS pushes, indicating that the token is inconsistent with the BundleId of the certificate used in the push. Please check if the certificate is correct and if the BundleId of the certificate is consistent with the BundleId used in client packaging.

The iOS phone cannot receive messages, but the message status is ACKED

For iOS pushes, if the message status is ACKED, it means that the message has been successfully pushed to Apple Push Notification service. Please check if the push permission is enabled and whether you have switched the app to the background.

The default mechanism for Apple is that when an app is in foreground, the messages can arrive but will be not shown. In order to show messages in foreground, you need to implement it mannually.

RPC call exceptions

If an exception occurs when you call a resource through a remote procedure call (RPC) request, troubleshoot the problem with reference to Security Guard error codes or Gateway result codes.

10.Appendix

10.1. Create an iOS push certificate

To send messages to an iOS device, you need to configure the iOS push certificate in the Message Push Service (MPS) console. iOS push certificate is used for message push. This topic describes types of certificates supported by the Message Push Service and the method of preparing a certificate.

Certificate types

Message Push Service only supports the Apple Push Service certificate. To learn more about Apple certificate types and related description, see Certificate type.

It is easy to confuse the Apple Push Service certificate with iOS Development certificate. Using **iOS Development** certificate may cause message push failure. The following sections describe how to distinguish between the two certificates through Key Store MAC and Message Push Service console.

Certificate type	Purpose
Apple Push Service	It is the Apple push certificate for production environment. It is used to establish connectivity between your notification service and APNs to deliver remote notifications to your app.
iOS Development	It is the Apple push certificate for development environment. It is used during development and testing.

MAC Key Store

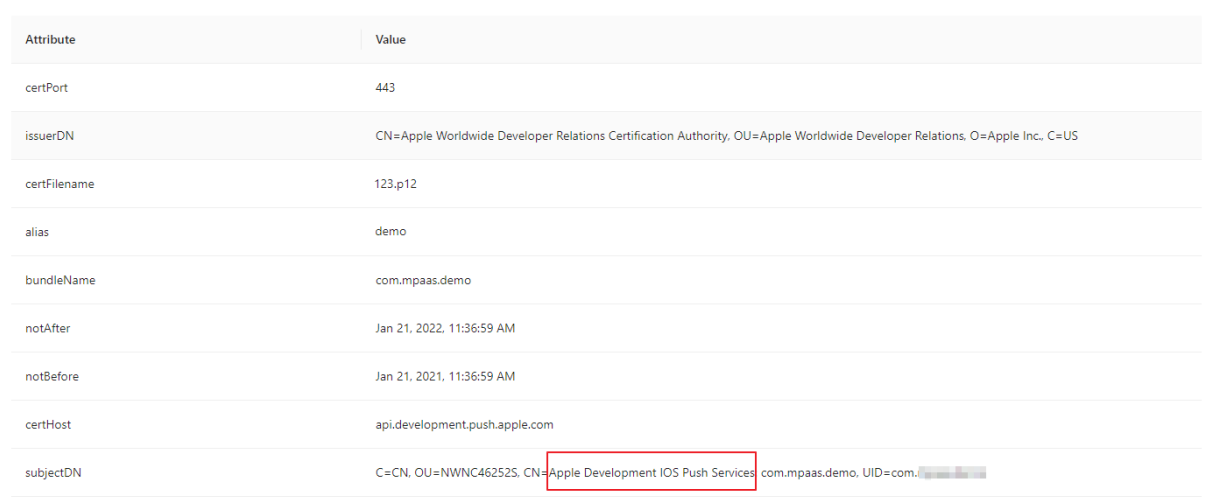
Double-click the existing `.p12` certificate and import the certificate into the MAC Keychain. The certificate information such as the name is displayed.

Among the certificates:

- **iPhone Developer**: Apple development certificate that is not supported by Message Push Service.
- **Apple Push Services**: Apple push certificate for the production environment that is supported by Message Push Service.
- **Apple Development IOS Push Services**: Apple push certificate for the development environment that is supported by Message Push Service.

MPS console

After the certificate is imported into the Message Push Service console, the following certificate information is displayed.



Attribute	Value
certPort	443
issuerDN	CN=Apple Worldwide Developer Relations Certification Authority, OU=Apple Worldwide Developer Relations, O=Apple Inc., C=US
certFilename	123.p12
alias	demo
bundleName	com.mpaas.demo
notAfter	Jan 21, 2022, 11:36:59 AM
notBefore	Jan 21, 2021, 11:36:59 AM
certHost	api.development.push.apple.com
subjectDN	C=CN, OU=NWNC46252S, CN=Apple Development IOS Push Services: com.mpaas.demo, UID=com.

Check the **subjectDN** attribute.

- **Apple Development IOS Push Services**: Apple push certificate for the development environment that is supported by Message Push Service.
- **Apple Push Service**: Apple push certificate for the production environment that is supported by Message Push Service.

In the preceding figure, the **subjectDN** attribute is **iPhone Developer**, indicating that it is an Apple development certificate, which is not supported by Message Push Service.

Prepare a certificate

Create an iOS app ID

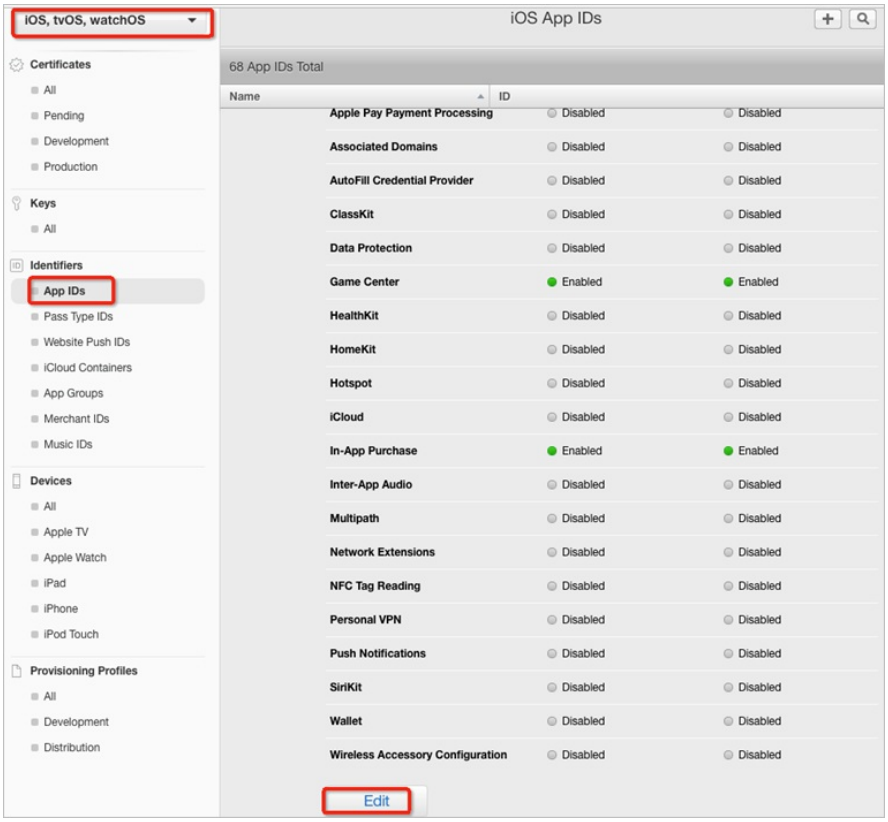
1. On Apple Developer, click **App IDs** in the left navigation pane, and click **+** in the upper right corner.
2. Enter the basic information.
   - **App ID Description** > **Name**
   - **App ID Suffix** > **Bundle ID** (The Bundle ID must be unique.)
3. Check **Push Notifications**.
4. Click **Continue**, and click **Register**. An iOS app ID is created.

Prepare a .certSigningRequest file

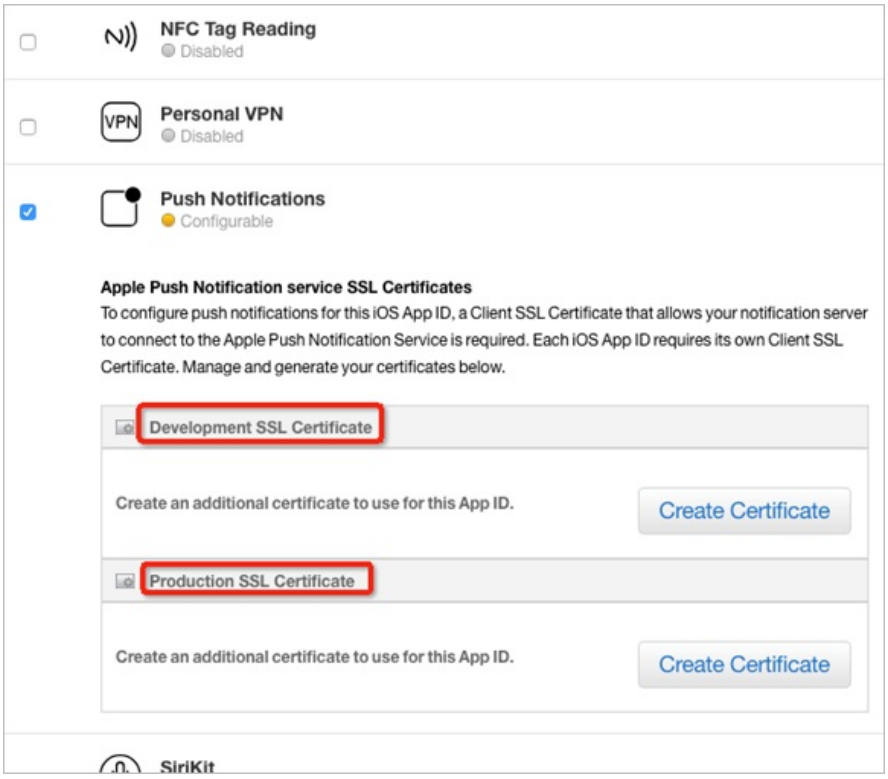
1. Access the MAC Keychain.
2. Request a certificate, choose **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority...**.
3. In the **Certificate Information** window, enter relevant information, such as the email address and name, based on actual situations.
4. A `.certSigningRequest` file is successfully generated.

Create a certificate

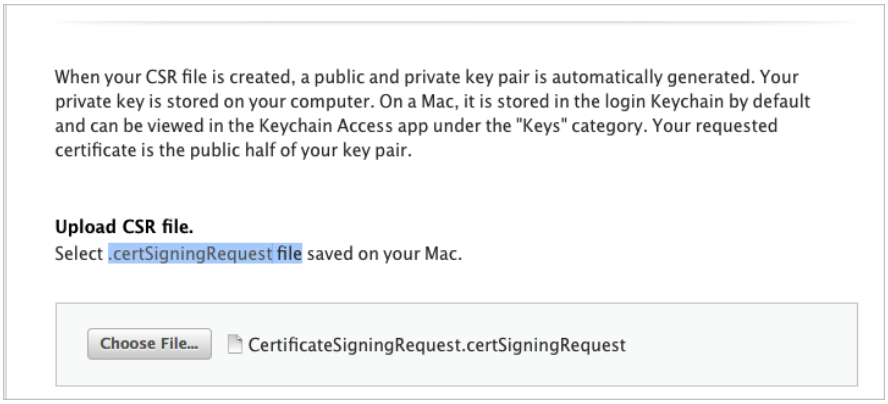
1. On the **iOS App IDs** page, select your iOS app ID and click **Edit**.



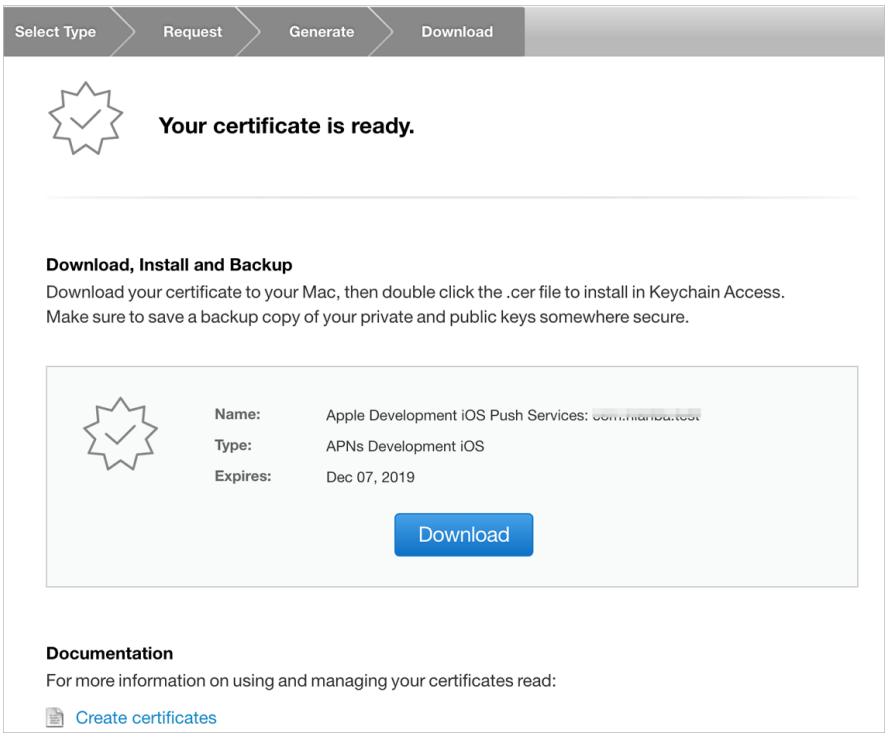
2. Click **Create Certificate** under **Development SSL Certificate** or **Production SSL Certificate** to create a certificate for the development or production environment.



3. Upload the `. certSigningRequest` file that you have prepared.



4. After a certificate is created successfully, the following page is displayed. Click **Download** to download the `.cer` file.



5. Convert the `.cer` file into a `.p12` file.
   i. Double-click the `.cer` file to import it into the MAC Key Store.
   ii. Right-click the file that you have imported, and **export** it. The file is exported as a `.p12` file.
6. After obtaining the `.p12` iOS push certificate, go to the mPaaS console, select the target App > **Message Push Service** > **Push configuration** to configure it.

10.2. Message push status codes

The following tables list the common status codes and the possible status codes for various push channels.

- Common status codes
- Apple Push
- Huawei Push

- MiPush
- OPPO Push
- vivo Push
- FCM

Common status codes

Status code	Message	Description
-1	WaitingForVerify	Waiting for verification.
0	DeviceNotOnlineOrNoResponse	Waiting for the device to go online (the persistent connection between the target device and the message push gateway is closed) or waiting for delivery confirmation.
1	NoBindInfo	There is no binding relationship. When you push a message based on the user ID, make sure that the target user ID has been bound with a device ID.
2	Acked	When you use an MPS self-built channel to push a message, this status indicates that the message has been successfully pushed to the client. When you use a vendor push channel to push a message, this status indicates that the vendor's push gateway has been successfully called.
99999999	NONE	Unknown status.

Apple Push

Status code	Message	Description
2001	PayloadEmpty	The message payload is empty.
2002	PayloadTooLarge	The message payload is too large.
2003	BadTopic	Incorrect bundleid in the certificate.
2004	TopicDisallowed	Illegal bundleid in the certificate.
2005	BadMessageId	Incorrect messageId.

2006	BadExpirationDate	Invalid expiration date.
2007	BadPriority	Invalid priority.
2008	MissingDeviceToken	Device token missed.
2009	BadDeviceToken	The device token is invalid or in incorrect format, or it does not exist. When you push a message based on the user dimension and receive this status code, you need to check whether the token used for binding is correct or not. We recommend that you create a simple push message in the MPS console as a test after completing the binding. In the development environment (the console is configured with a development environment certificate), you need to use your personal development certificate to package the app for testing. Otherwise, BadDeviceToken will appear.
2010	DeviceTokenNotForTopic	The device token doesn't match the specified topic.
2011	Unregistered	Invalid token.
2013	BadCertificateEnvironment	The client certificate is for the wrong environment.
2014	BadCertificate	The certificate is invalid.
2023	MissingTopic	No topic is specified.
2024	ConnClosed	APNS disconnected. This status may caused by the following reasons: • The iOS push environment configured in the console and the pushed device token do not match. • The certificate packaged in the app's installation package and the certificate configured in the console do not match. • The BundleId in the project is different from the BundleId configured in the console. For more information about how to configure the iOS push certificate, environment and BundleId in the console, see Channel configuration.
2025	ConnUnavailable	APNS connection is unavailable.

For more message push statuses of Apple Push, see Handling Notification Responses from APNs.

Huawei Push

Status code	Description
100	Invalid unknown parameter.
101	Invaid API_KEY.
102	Invaid SESSION_KEY.
106	The app or session has no permission to call the current service.
107	Obtain the client and secret again (e.g., in case of an updated algorithm).
109	Excessive nsp_ts difference
110	Interface internal exception.
111	Server is busy.
80000003	Terminal is not online.
80000004	The app has been uninstalled.
80000005	Response timed out.
80000006	No routing. No connection has been established between the terminal and Push.
80000007	The terminal is in other region, and doesn't use Push in Chinese mainland.
80000008	Incorrect routing. It may because that the terminal has switched the Push server.
80100000	Some parameters are incorrect.
80100002	Ilegal token list.

80100003	Ilegal payload.
80100004	Invalid timeout period.
80300002	No permission to send messages to the tokens listed in the parameter.
80300007	All tokens in the request are illegal tokens.
81000001	Internal error.
80300008	Authentication error (the request message body is too large).

MiPush

Status code	Description
1001	System error.
10002	Service suspended.
10003	Error in remote service.
10004	Cannot request this resource due to IP restriction.
10005	This resource requires authorized appkey.
10008	Incorrect parameters.
10009	The system is busy.
10012	Ilegal request.
10013	Ilegal user.
10014	Access to the app interface is restricted.
10017	Ilegal parameter value.

10018	The request exceeds the length limit.
10022	Requests to the IP exceed the frequency limit.
10023	User's requests exceed the frequency limit.
10024	User's requests for special interface exceed the frequency limit.
10026	The app is in the blacklist, and cannot call any APIs.
10027	The app API is called too frequently.
10029	Illegal device.
21301	Authentication failed.
22000	Illegal app.
22001	The app doesn't exist.
22002	The app has been revocated.
22003	Failed to update the app.
22004	App information missed.
22005	Invalid app name.
22006	Invalid app ID.
22007	Invalid app Key.
22008	Invalid app Secret .
22020	Illegal app description.
22021	The app hasn't been authorized by users.

22022	Invalid app package name.
22100	Incorrect data format for the app notification.
22101	Too many app notifications.
22102	Failed to send the app notification.
22103	Invalid app notification ID.
20301	Invalid target.

OPPO Push

Status code	Message	Description
-1	Service Currently Unavailable	The service is unavailable, please try again later.
-2	Service in Flow Control	The service is under traffic control.
11	Invalid Auth Token	Invalid AuthToken.
13	App Call Limited	App calling counts exceed limit, including the calling frequency limit.
14	Invalid App Key	Invalid AppKey.
15	Missing App Key	AppKey missed.
16	Invalid Signature	Invalid signature. Failed to pass signature verification.
17	Missing Signature17	Signature missed. Failed to pass signature verification.
28	App Disabled	The app is unavailable.
29	Missing Auth Token	AuthToken missed.

30	Api Permission Denied	The app has no permission to perform API push.
10000	Invalid RegistrationId	registration_id is in incorrect format.

vivo Push

Status code	Description
10000	Permission authentication failed.
10040	The resource has reached the upper limit, please try again later.
10050	Both alias and regId cannot be empty.
10055	The title cannot be empty.
10056	The title cannot exceed 40 characters in length.
10058	The content cannot exceed 100 characters in length.
10066	The number of custom key/value pairs cannot exceed 10.
10067	Invalid custom key/value pair.
10070	The total number of messages sent exceeds the limit.
10071	The sending time is out of the allowable time range.
10072	Message push is too fast, please try again later.
10101	The message content is unapproved.
10102	Unknown exception occured in vivo server.
10103	Pushed content contains sensitive information.
10110	Please set the frequency of sending commercial messages.

10302	Invalid regId.
10303	requestId already exists.
10104	Please send a formal message. Please check the content, and do not send test text. The content in a formal message should not be numbers only, letters only, symbols plus numbers, and cannot contain "test", braces, and square brackets.

FCM

Status code	Message	Description
90000002	InvalidRegistration	Invalid target.
90000003	NotRegistered	The target is unregistered.
90000004	InvalidPackageName	Invalid package name.
90000007	MessageTooBig	Message body is too large.
90000009	InvalidTtl	Invalid offline time-to-live.
90000011	InternalServerError	FCM service exception
90000401	Authentication	Failed to pass permission verification.