移动开发平台

腾讯移动推送（信鸽）

产品文档

腾讯云

文档目录

腾讯移动推送（信鸽）产品介绍

最近更新时间：2018-04-02 18:49:02

简介

腾讯移动推送（信鸽）是一款专业的移动 App 推送平台，支持百亿级的通知/消息推送，秒级触达移动用户，现已全面支持 Android 和 iOS 两大主流平台。开发者可以方便地通过嵌入 SDK，通过 API 调用或者 Web 端可视化操作，实现对特定用户推送，大幅提升用户活跃度，有效唤醒沉睡用户，并实时查看推送效果。

功能介绍

功能	简介
自定义推送通知	提供自定义通知、消息推送平台，您可以通过控制台或接口操作，将信息无延时的推送给目标用户并支持多种提示方式。
细分目标用户	精细化目标用户分类，为目标用户打上画像标签，不论地理位置、是否活跃用户，还是各个版本用户均能准确定位。
运营数据全面监控	推送完成后，可及时查看推送效果，推送量、抵达量、点击量等关键运营数据一目了然，为您的运营提供充分数据支撑。
灵活的服务端接口	支持多种推送接口，包括全量、单个、批量推送等。方便业务结合自身需求灵活调用。提供Java、PHP、Python、Node.js等多种SDK，方便开发者使用。

优势

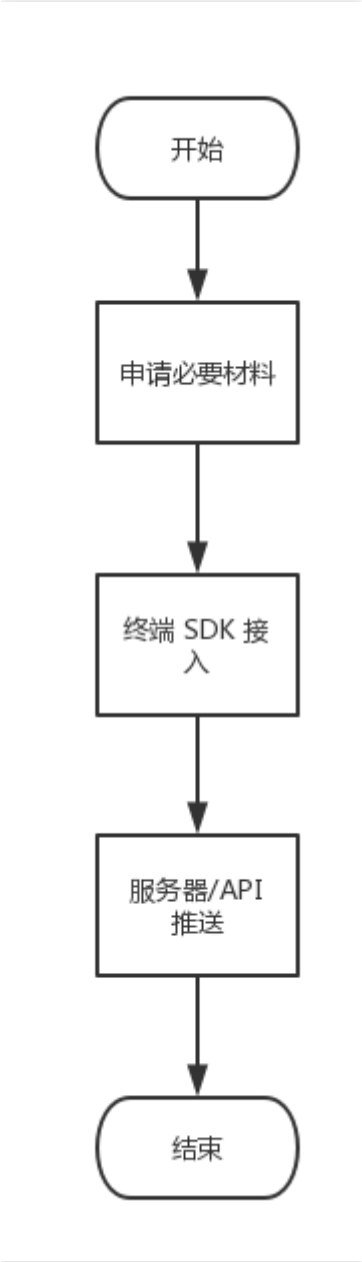
功能	简介
极速接入	2 分钟极速接入推送服务，与数亿移动智能终端建立稳定的长连接
精准触达	多维度传递价值信息，每天可发送百亿级的通知/消息，精确抵达目标用户
接口灵活	开放推送能力，提供多种语言 API ，包括Java/PHP等，业务自由集成
全面监控	实时监控通知/消息的抵达用户量、转化量、转化率，推送效果一目了然

客户案例

服务接入总览

最近更新时间：2019-04-18 15:40:27

接入腾讯移动推送模块主要分为几步，首先是申请一些必要的资料，例如 iOS 所需要的推送证书。然后是进行终端 SDK 的接入，最后是使用服务器端调用 API 或者直接使用控制台进行推送。



申请必要资料

iOS 申请必要资料

对于 iOS 端的开发者而言，需要先向苹果申请推送证书才能使用。申请推送证书步骤可参见：申请推送证书指南。

终端 SDK 接入

Android SDK 接入

详见 Android SDK 使用入门。

iOS SDK 接入

详见 iOS SDK 使用入门。

服务器端推送

可以采取服务器调用 API 的方式进行推送，详细的接口描述可以参见 服务器端推送指南。

控制台推送

如果没有使用服务器端调用 API 进行推送的需求，直接使用 控制台推送 是便捷的选择。

控制台使用手册
通知推送控制台文档

最近更新时间：2018-03-26 16:15:05

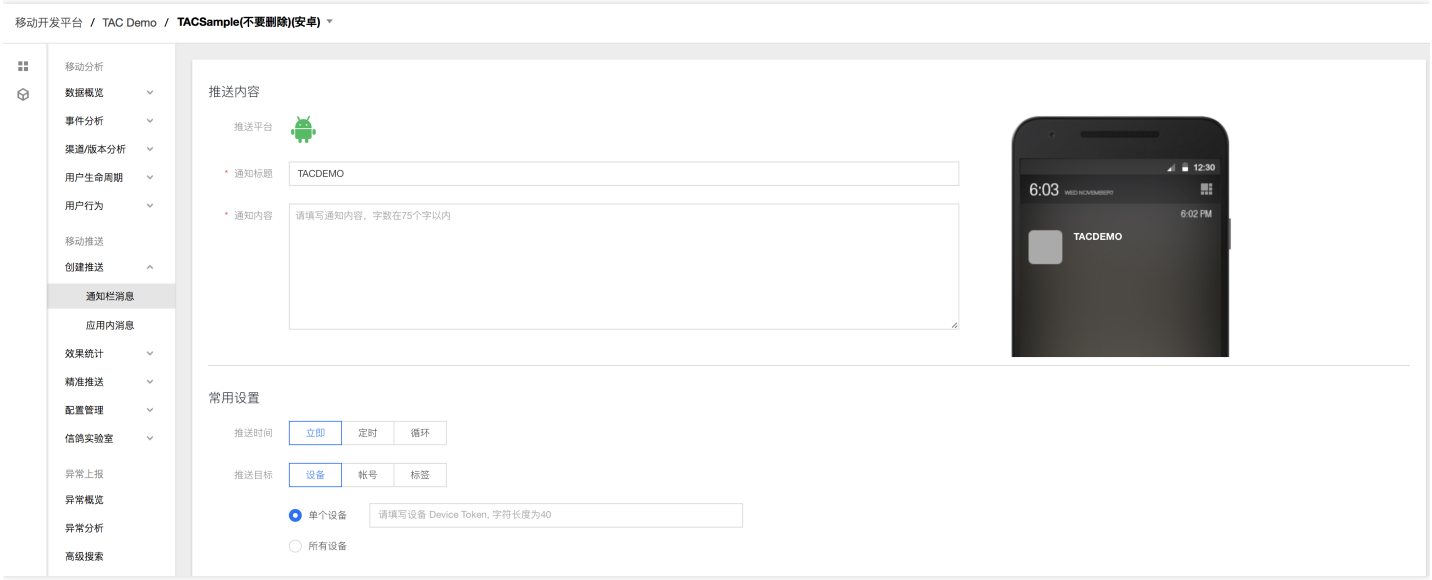
创建推送

Android

推送通知栏消息

通知栏信息是指在系统推送通知栏中弹出的信息，Android 应用可以在控制台上面直接创建通知栏消息推送。推送时可以设置推送时间，推送目标（针对特定的设备、账号或者是标签进行推送），并且可以附带额外的附加参数。

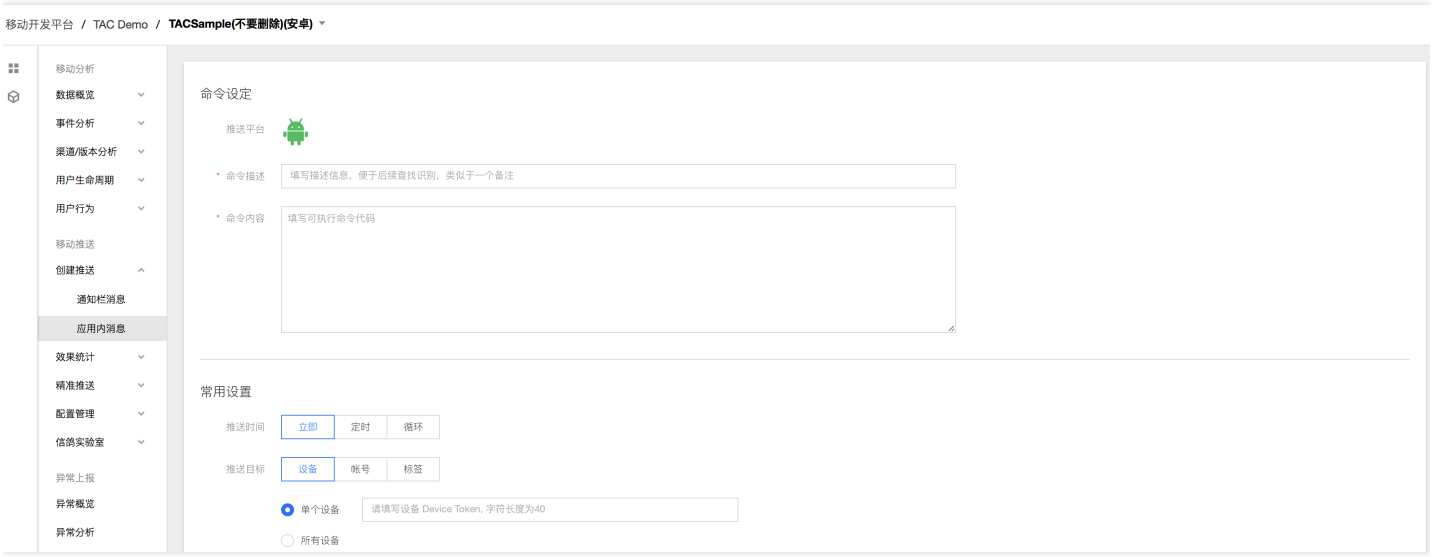
填好带*号也就是必填的选项后，点击"确认推送"既可以推送到相应的设备。



推送应用内消息

当 APP 在前台运行时，可以接受应用内信息并进行一些特殊的处理。与创建通知栏消息的过程类似。具体参数含义可以参照下面的说明表：

参数名字	含义
命令描述	描述信息，用于后续查找识别
命令内容	该推送的具体内容，用于携带需要的自定义信息。



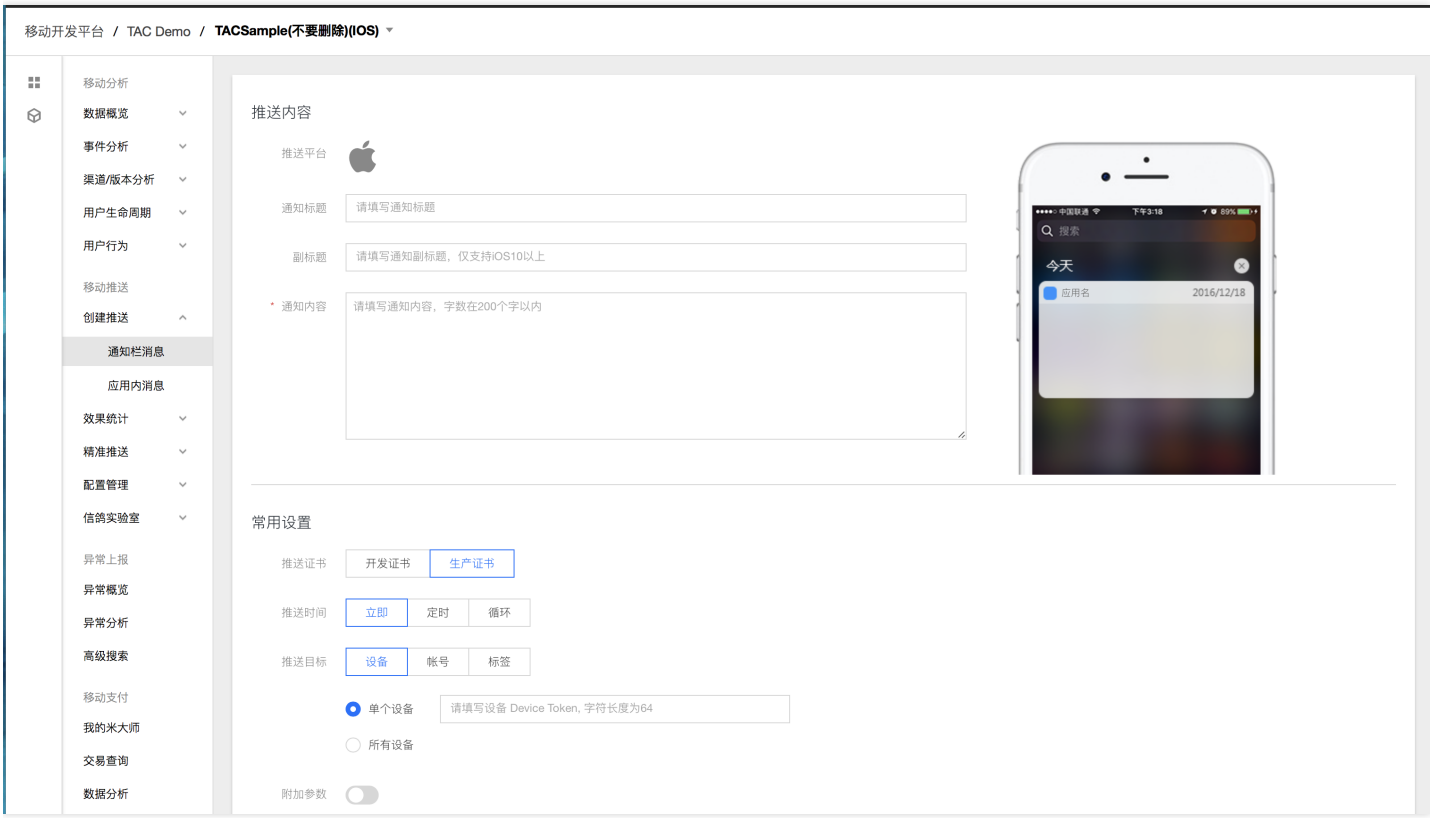
iOS

iOS 并不支持应用内消息，只能创建通知栏消息。

推送通知栏消息

推送时需要填入通知的内容，如果填了标题或者副标题的话，会在弹出消息中以对应的形式显示。推送时需要选择环境来选择是使用开发证书还是生产证书推送。

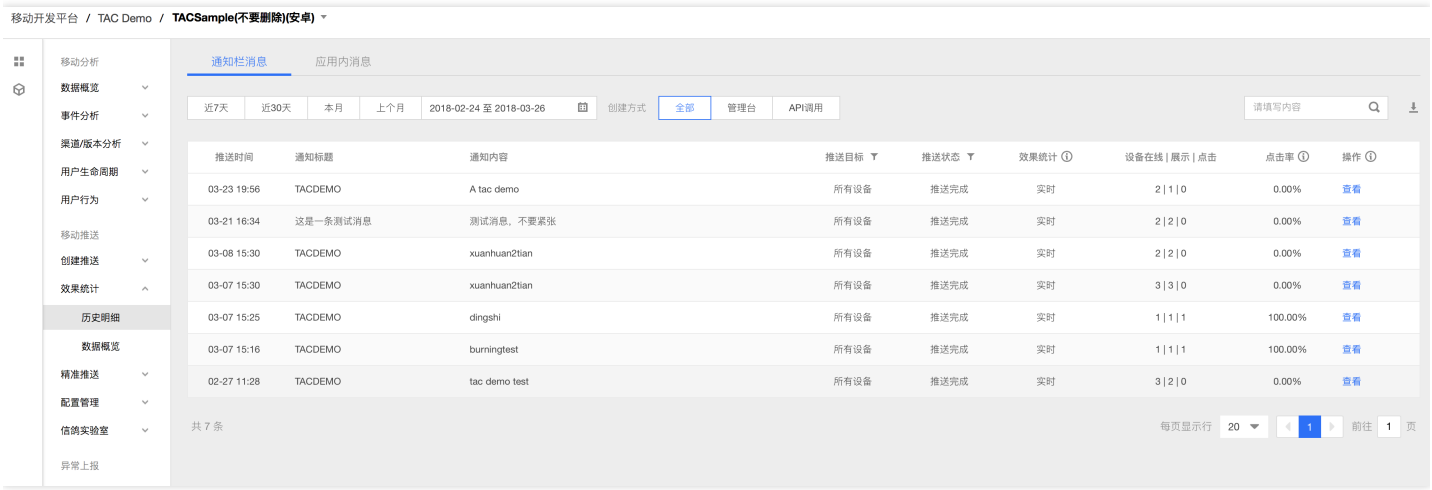
填好带*号也就是必填的选项后，点击“确认推送”既可以推送到相应的设备。注意在推送前，需要设置好推荐使用的开发证书或者生产证书，具体设置方法可以参考下文中的应用配置。



推送效果统计

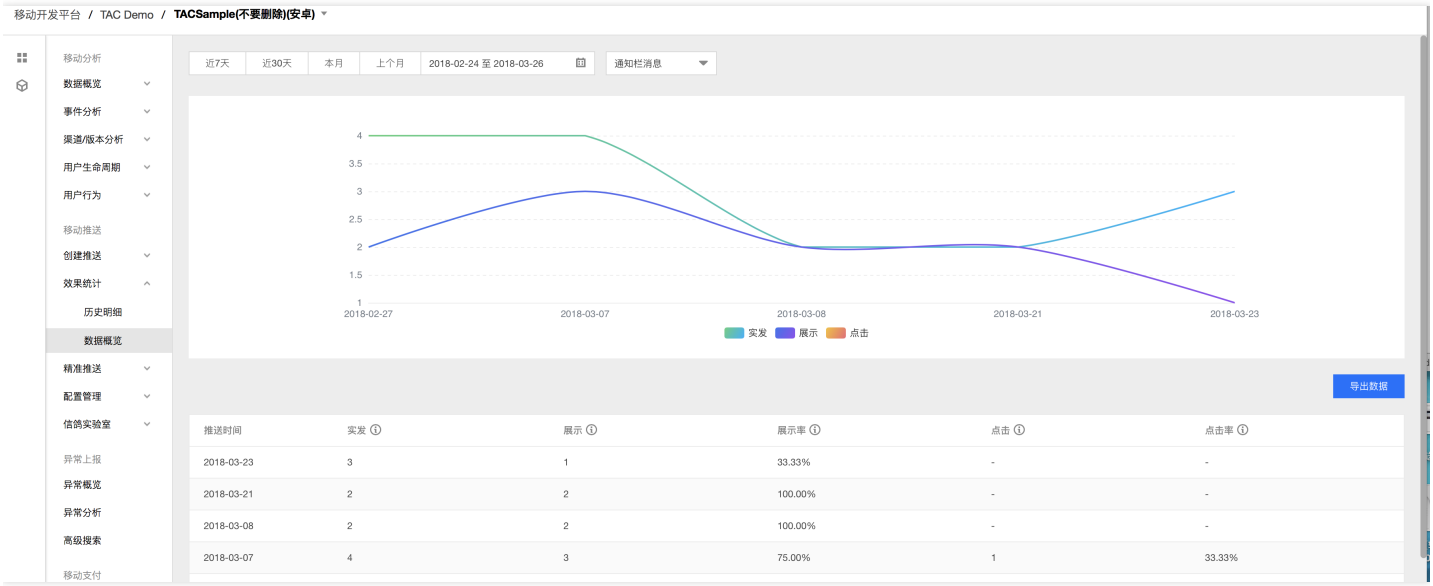
在推送效果统计中，可以查看历史推送消息的一些情况，并且进行统计，便于分析。

> 注意只有推送所有设备的消息才能在这里预览, 针对单个设备推送的消息无法在这里统计。



数据概览

数据概览中可以以图表的形式直观判断。



精准推送

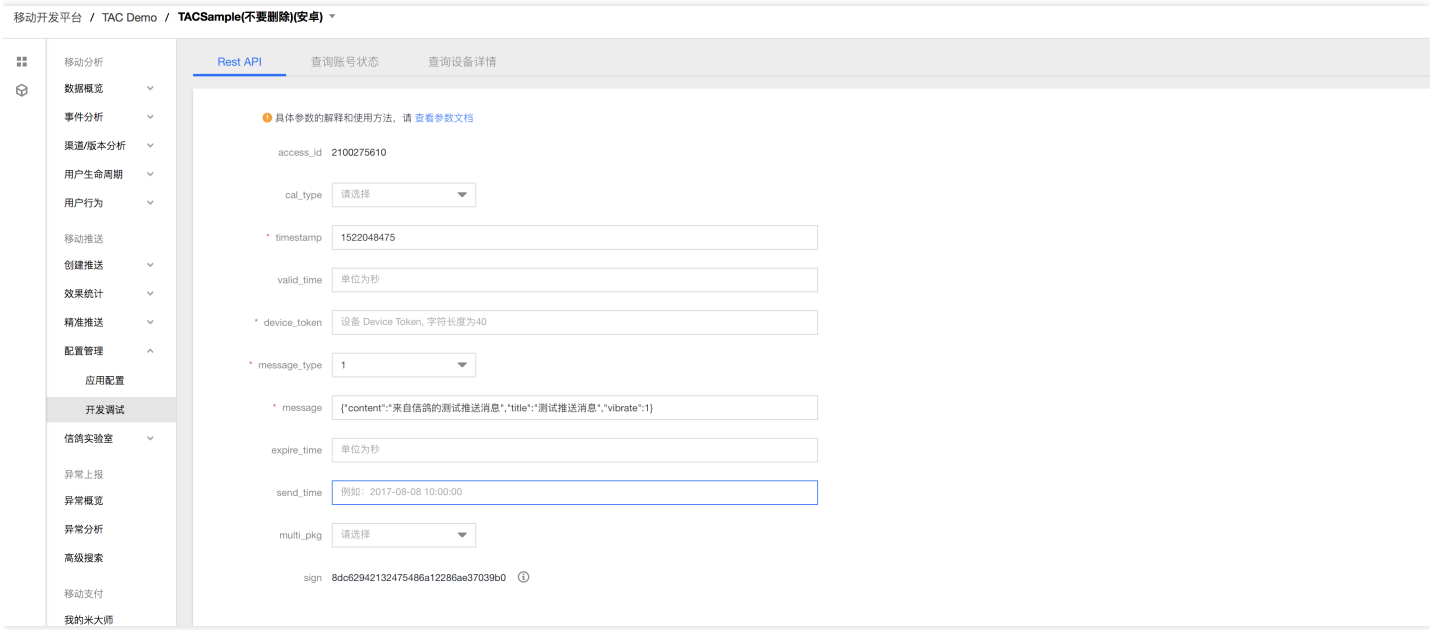
标签管理

可以对特定的设备或者用户添加标签（在这里或者SDK中），进行分组管理的作用。添加标签后，可以实现对特定标签的用户推送的功能，实现精准推送。

配置管理

开发调试 RestAPI

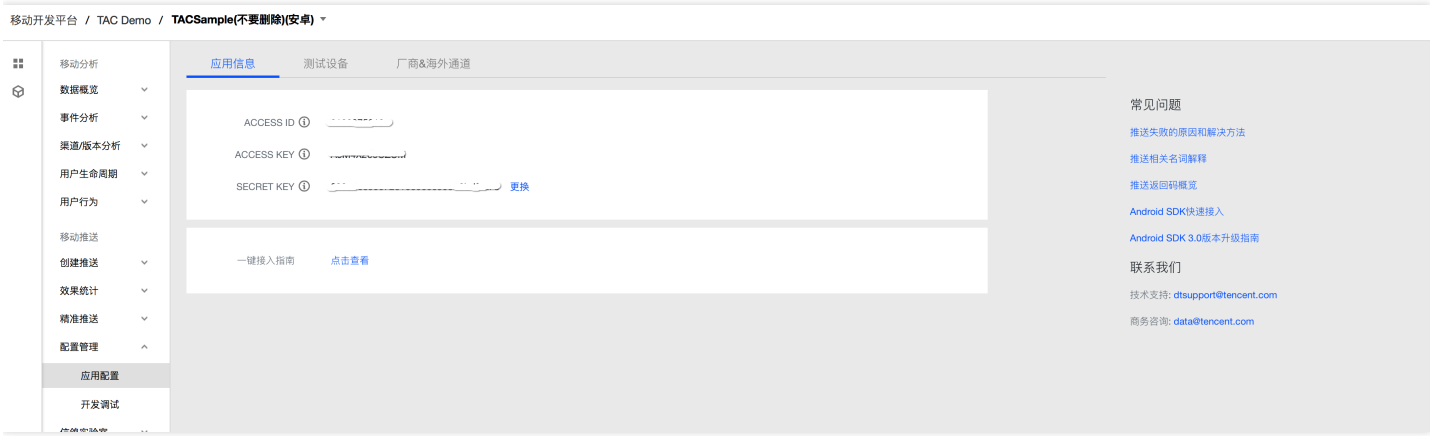
遇到推送无效等问题时，可以使用控制台提供的开发调试工具来调试，具体使用方式可以点击控制台中的查看参数文档查看。



Android

应用配置

应用配置中可以查看一些基本的应用信息，如应用推送使用的 ACCESS ID, ACCESS KEY, 和 SECRET KEY。



厂商和海外通道

对于厂商提供推送通道的，可以通过具体厂商的推送通道进行推送。通过厂商推送的话可以使用系统级的推送通道进行推送，有更好的及时性。

iOS

应用配置

iOS 应用的配置里包括了一些推送的基本信息，包括推送使用的 ACCESS ID, ACCESS KEY 和 SECRET KEY等。

其中如果想要向 iOS设备进行推送的话，就需要先设置开发证书（用于调试推送）和生产证书（用于Release版本的推送）。如果对于证书的来源或者用法有不清晰的地方，可以点击控制台右边的"iOS推送证书设置指南"进入了解具体的证书的生成和使用方法。

信鸽实验室

行业洞察

行业洞察里，可以观察最近一段时间内(7天，14天或者30天)，从推送获得的一些基本情况，包括联网用户、活跃用户、累计推送等。与此同时还可以看到与自己同类型的应用的平均情况，进行比较。

Android 文档
Android 快速入门

最近更新时间：2018-08-16 16:31:40

准备工作

您首先需要一个 Android 工程，这个工程可以是您现有的工程，也可以是您新建的一个空的工程。

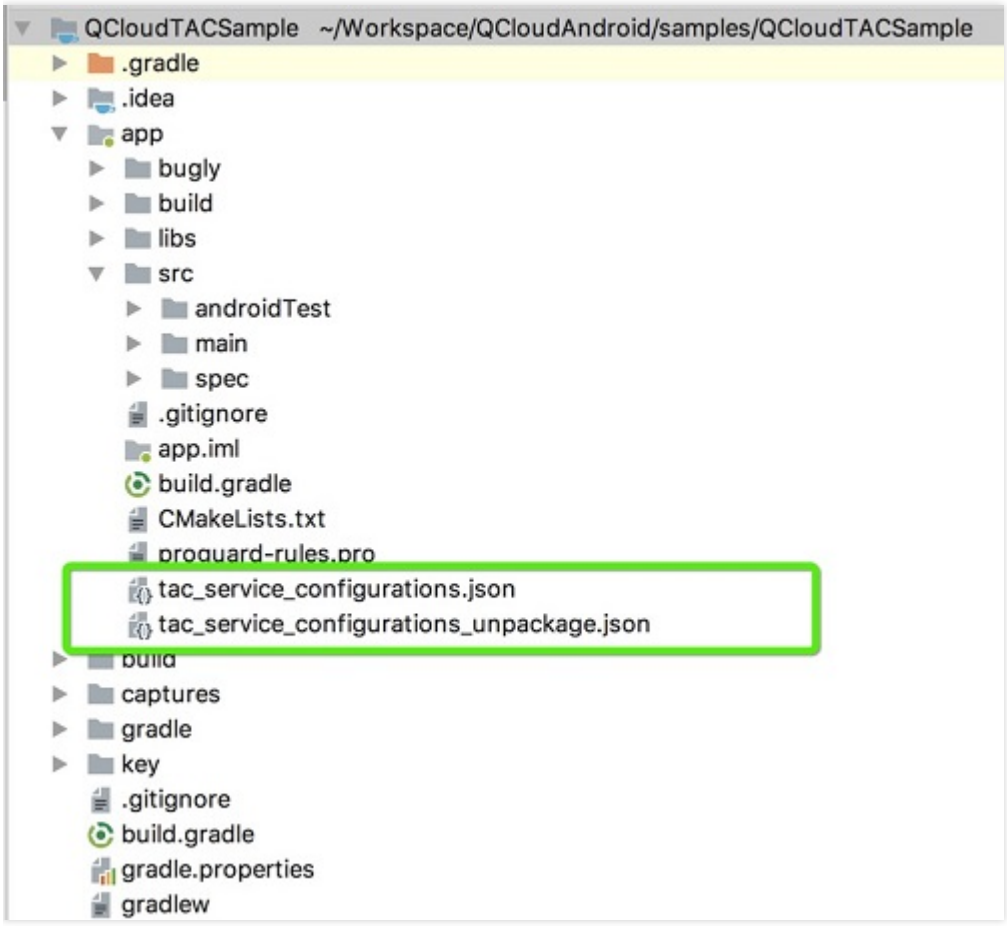
第一步：创建项目和应用（已完成请跳过）

在使用我们的服务前，您必须先在 MobileLine 控制台上 创建项目和应用。

第二步：添加配置文件（已完成请跳过）

在您创建好的应用上单击【下载配置】按钮来下载该应用的配置文件的压缩包：

解压该压缩包，您会得到 tac_service_configurations.json 和 tac_service_configurations_unpackage.json 两个文件，请您如图所示添加到您自己的工程中去。



注意：
请您按照图示来添加配置文件，`tac_service_configurations_unpackage.json` 文件中包含了敏感信息，请不要打包到 APK 文件中，MobileLine SDK 也会对此进行检查，防止由于您误打包造成的敏感信息泄露。

第三步：集成 SDK

您需要在工程级 build.gradle 文件中添加 SDK 插件的依赖：

```
buildscript {
...
dependencies {
classpath 'com.android.tools.build:gradle:3.0.1'
// 添加这行
classpath 'com.tencent.tac:tac-services-plugin:1.3.+'
}
}
```

在应用级 build.gradle 文件（通常是 app/build.gradle）中添加 messaging 服务依赖，并使用插件：

```
dependencies {
// 增加这两行
compile 'com.tencent.tac:tac-core:1.3.+'
compile 'com.tencent.tac:tac-messaging:1.3.+'
}
...

// 在文件最后使用插件
apply plugin: 'com.tencent.tac.services'
```

> `com.tencent.tac:tac-messaging` 默认引入了厂商通道推送包，如果不需要集成厂商推送，您可以改用 `com.tencent.tac:tac-messaging-lite`

到此您已成功接入了 MobileLine 移动推送服务。

验证服务

查看服务启动情况

安装并运行 App 后，SDK 会自动在 Messaging 后台进行注册，注册成功后会打印如下日志：

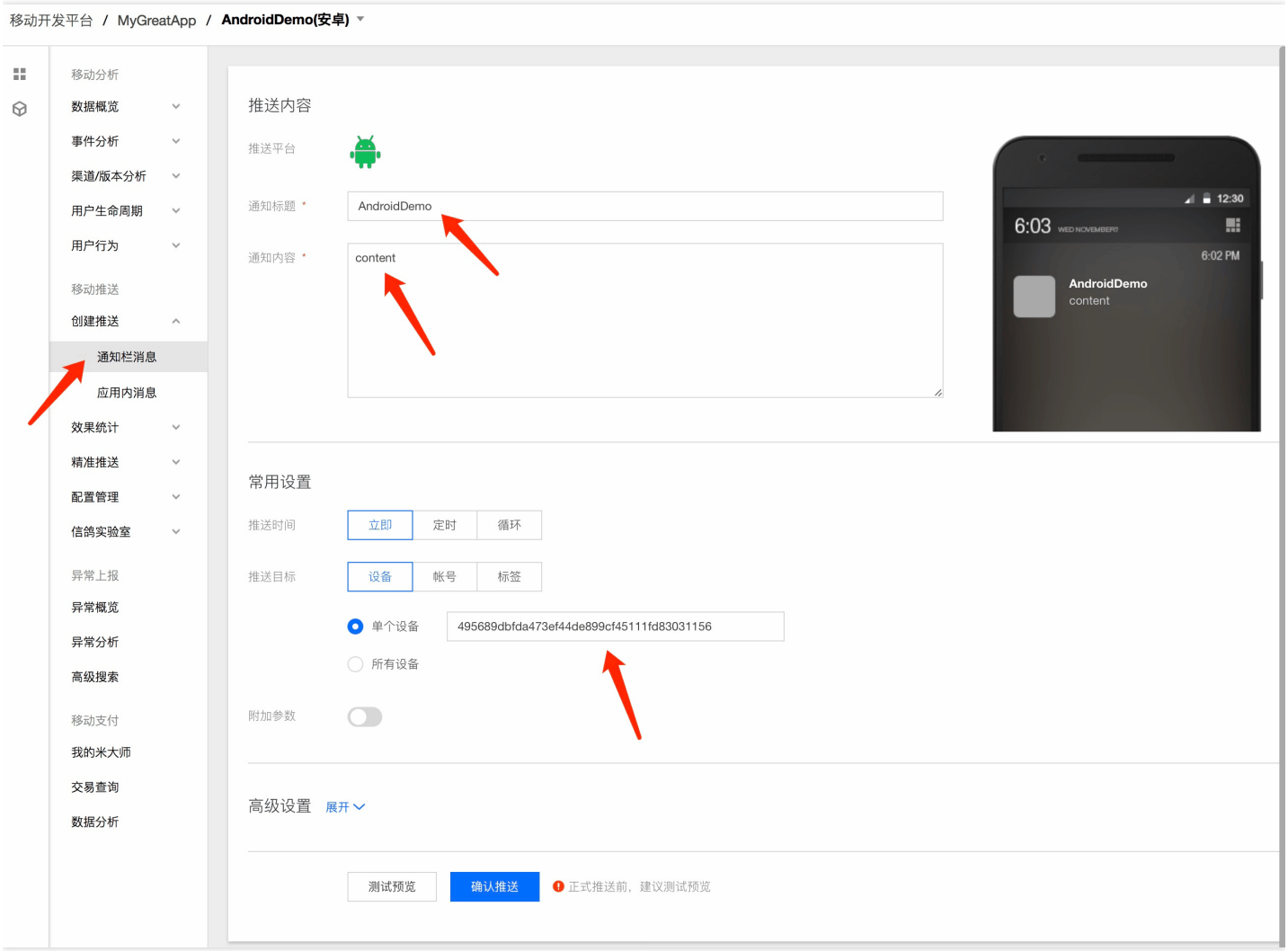
```
I/tacApp: TACMessagingService register success, code is 0, token is 495689dbfda473ef44de899cf45111fd83031156
```

> **注意：**
> 这里日志打印的 token 信息标识推送时的唯一 ID，您可以通过 token 信息给该设备发送通知。

如果没有打印以上日志，请查看 常见问题。

在控制台上推送通知栏消息

打开 MobileLine 控制台，选择【创建推送】下的【通知栏消息】，并填写好 **通知标题** 和 **通知内容**，然后选择单选框中的【单个设备】，然后将注册成功后打印的设备唯一标识 token 信息拷贝到编辑框中（示例这里为 495689dbfda473ef44de899cf45111fd83031156 ），然后单击【确认推送】。



推送通知栏消息成功后，App 在运行状态下会收到通知栏消息。

> **注意：**您也可以选择推送给所有的设备，设备收到消息可能会有一定的延时。

Proguard 配置

如果您的代码开启了混淆，为了 SDK 可以正常工作，请在 `proguard-rules.pro` 文件中添加如下配置：

```
# MobileLine Core

-keep class com.tencent.qcloud.core.** { *;}
-keep class bolts.** { *;}
-keep class com.tencent.tac.** { *;}
```

```
-keep class com.tencent.stat.*{*;}
-keep class com.tencent.mid.*{*;}
-dontwarn okhttp3.**
-dontwarn okio.**
-dontwarn javax.annotation.**
-dontwarn org.conscrypt.**

# MobileLine Messaging

-keep class com.tencent.android.tpush.** {* ;}
-keep class com.qq.taf.jce.** {*;}

# MobileLine Vendor Messaging

-keepattributes *Annotation*
-keepattributes Exceptions
-keepattributes InnerClasses
-keepattributes Signature
-keepattributes SourceFile,LineNumberTable
-keep class com.hianalytics.android.**{*;}
-keep class com.huawei.updatesdk.**{*;}
-keep class com.huawei.hms.**{*;}
-keep class com.huawei.gamebox.plugin.gameservice.**{*;}
-keep public class com.huawei.android.hms.agent.** extends android.app.Activity { public *; protec
ted *; }
-keep interface com.huawei.android.hms.agent.common.INoProguard {*;}
-keep class * extends com.huawei.android.hms.agent.common.INoProguard {*;}
-keep class com.meizu.cloud.pushsdk.**{*;}
-keepclasseswithmembernames class com.xiaomi.**{*;}
-dontwarn com.huawei.android.hms.**
-dontwarn com.xiaomi.push.**
-dontwarn com.meizu.cloud.pushsdk.**
```

后续步骤

注册回调接口

注册回调接口非常重要，您可以注册回调接口来接收推送服务在不同状态下给您的回调，具体有：

- `onRegisterResult()` ：注册 Messaging 服务后回调。
- `onUnregisterResult()` ：反注册 Messaging 服务后回调。
- `onMessageArrived()` ：收到透传消息（即控制台上的应用内消息）后回调。
- `onNotificationArrived()` ：收到通知栏消息后回调。

- `onNotificationClicked()` ：单击通知栏消息后回调。
- `onNotificationDeleted()` ：删除通知栏消息后回调。
- `onBindTagResult()` ：绑定标签后回调。
- `onUnbindTagResult()` ：解绑标签后回调。

如何注册回调接口，请参见 这里。

集成厂商推送通道

我们建议您集成厂商推送通道，通过集成厂商官方提供的系统级推送通道，在对应厂商手机上，推送消息能够通过系统通道抵达终端，并且无需打开应用就能够收到推送，目前支持华为、小米和魅族三个厂商通道，具体集成方式请参考 这里。

给设备推送消息

您可以通过控制台给设备推送消息(具体请参考 这里)，您也可以通过我们的后台接口来发送消息，具体请参考 Rest API 使用指南 或者 服务端 SDK。除了通过设备 token 来指定用户外，我们还支持通过标签推送消息（具体请参考 这里）或者通过账户推送消息（具体请参考 这里）。

注册状态回调通知

最近更新时间：2018-04-26 09:53:10

在使用 messaging 服务时，您可以注册 messaging 服务回调接口，用于接收消息在不同状态下的通知：

继承 TACMessagingReceiver 类

您必须创建一个 TACMessagingReceiver 子类用于接收我们的消息回调，例如：

```
public class MyReceiver extends TACMessagingReceiver {

// 启动 Messaging 服务后，会自动向 Messaging 后台注册，注册完成后会回调此接口。
@Override
public void onRegisterResult(Context context, int errorCode, TACMessagingToken token) {

Toast.makeText(context, "注册结果返回：" + token, Toast.LENGTH_SHORT).show();
Log.i("messaging", "MyReceiver::OnRegisterResult : code is " + errorCode + ", token is " + token.getTo
kenString());
}

// 反注册后回调此接口。
@Override
public void onUnregisterResult(Context context, int code) {

Toast.makeText(context, "取消注册结果返回：" + code, Toast.LENGTH_SHORT).show();
Log.i("messaging", "MyReceiver::onUnregisterResult : code is " + code);
}

// 收到透传消息后回调此接口。
@Override
public void onMessageArrived(Context context, TACMessagingText tacMessagingText, PushChannel
channel) {

Toast.makeText(context, "收到透传消息：" + tacMessagingText, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::OnTextMessage : message is " + tacMessagingText+ " pushChannel "
+ channel);
}

// 收到通知栏消息后回调此接口。
@Override
public void onNotificationArrived(Context context, TACNotification tacNotification, PushChannel pu
shChannel) {
```

```
Toast.makeText(context, "收到通知消息：" + pushChannel, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::onNotificationArrived : notification is " + tacNotification + " pushChannel " + pushChannel);

}

// 点击通知栏消息后回调此接口。
@Override
public void onNotificationClicked(Context context, TACNotification tacNotification, PushChannel pushChannel) {

Log.i("messaging", "MyReceiver::onNotificationClicked : notification is " + tacNotification + " pushChannel " + pushChannel);

}

// 删除通知栏消息后回调此接口
@Override
public void onNotificationDeleted(Context context, TACNotification tacNotification, PushChannel pushChannel) {

Log.i("messaging", "MyReceiver::onNotificationDeleted : notification is " + tacNotification + " pushChannel " + pushChannel);

}

// 绑定标签回调
@Override
public void onBindTagResult(Context context, int code, String tag) {
Toast.makeText(context, "绑定标签成功：tag = " + tag, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::onBindTagResult : code is " + code + " tag " + tag);

}

// 解绑标签回调
@Override
public void onUnbindTagResult(Context context, int code, String tag) {
Toast.makeText(context, "解绑标签成功：tag = " + tag, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::onUnbindTagResult : code is " + code + " tag " + tag);
}
}
```

注意：
回调的详细说明请参考这里

在 AndroidManifest.xml 文件中注册

在创建好 TACMessagingReceiver 的子类后，您需要在工程的 AndroidManifest.xml 文件中注册该类：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.tac">
<application
...>
...
<!-- 这里替换成你自己的 TACMessagingReceiver 子类 -->
<receiver android:name=".MyReceiver">
<intent-filter>
<action android:name="com.tencent.tac.messaging.action.CALLBACK" />
</intent-filter>
</receiver>

</application>
</manifest>
```

注册回调信息详细说明

最近更新时间：2018-04-24 18:49:18

通过注册 Messaging 服务广播接收器，您可以收到 Messaging 服务的通知，您需要继承 `TACMessagingReceiver` 类，然后在 `AndroidManifest.xml` 文件中进行如下注册：

```
<receiver android:name="com.yourpackage.MessagingReceiver">
<intent-filter>
<action android:name="com.tencent.tac.messaging.action.CALLBACK" />
</intent-filter>
</receiver>
```

`TACMessagingReceiver` 的回调方法说明如下，您可以根据业务需求，在回调时执行业务逻辑。

注册结果回调

```
/**
* 注册结果回调。
*
* <p>
* 用户可以在这里获取注册的状态，如果注册成功，则会返回 token 。
* </p>
*
* @param context 上下文
* @param errorCode 错误码
* @param tacMessagingToken 注册结果
*/
void onRegisterResult(Context context, int errorCode, TACMessagingToken tacMessagingToken);
```

注册成功后，可以获得 `TACMessagingToken` 对象，您可以通过该对象获取设备标识 token。

```
@Override
public void onRegisterResult(Context context, int errorCode, TACMessagingToken tacMessagingTok
en) {

if (tacMessagingToken != null) {
String token = tacMessagingToken.getTokenString();
}
}
```

相关错误码说明请看文档最后的列表。

反注册结果回调

```
/**
* 反注册结果回调
*
* @param context 上下文
* @param code 错误码
*/
void onUnregisterResult(Context context, int code);
```

相关错误码说明请看文档最后的列表。

收到透传消息

```
/**
* 收到透传消息回调。
*
* <p>
* 信鸽通道所有参数均有效。
*
* 魅族的透传消息到达通知没有走自己的回调，而是走的信鸽的回调，因此回调的参数和信鸽保持一致。
*
* 华为的透传消息只有 content，没有 title 和 自定义内容（华为的官方控制台上只允许填写 content 参数）。
*
* 小米的透传消息有 title、content 和 extra。
* </p>
*
* @param context 上下文
* @param message 应用内消息
* @param channel 渠道信息：信鸽、小米、魅族、华为
*/
void onMessageArrived(Context context, TACMessagingText message, PushChannel channel);
```

`TACMessagingText` 代表一个透传消息，包含了如下几个属性

参数名称	类型	意义	备注
title	String	消息标题	有些情况下 title 可能会空值
content	String	消息内容	所有的通道下均会有 content 内容
extra	String	自定义参数	返回信鸽或者相应厂商未经过解析的字符串
customContent	Map	将 extra 字符串解析为 map	如果自定义参数包含了","，那么在小米通道下 customContent 可能会解析出错。

收到通知栏消息

```
/**
* 收到通知消息，信鸽、小米和魅族推送渠道下回调，华为推送渠道不回调。
*
* <p>
* 信鸽通道所有参数均有效。
*
* 魅族回调时，{@link TACNotification} 中只有参数 {@link TACNotification#getText()} 返回的参数是有有效的，包括通知消息的
* 标题、内容和自定义信息，其他字段均无效。
*
* 小米回调时，{@link TACNotification} 中参数 {@link TACNotification#getText()} 返回的参数是有有效的，包括通知消息的
* 标题、内容和自定义信息，同时 {@link TACNotification#getId()} 也是有效的，其他字段均无效。
* </p>
*
* @param context 上下文
* @param notification 通知消息
* @param channel 当前的推送渠道：信鸽、小米、魅族
*/
void onNotificationArrived(Context context, TACNotification notification, PushChannel channel);
```

`TACNotification` 代表一个通知栏消息，包含了如下几个属性

参数名称	类型	意义
id	String	通知栏消息 ID
activity	String	设置点击通知栏消息后拉起 Acitivty 时对应的 Activity
notificationActionType	NotificationActionType	点击通知栏消息后的操作
text	TACMessagingText	通知栏消息的基本内容，包含了 title，content，extra 和 customContent

`NotificationActionType` 是一个枚举类型，包含了如下几种值：

参数名称	意义
NOTIFICATION_ACTION_ACTIVITY	点击消息后拉起 Activity
NOTIFICATION_ACTION_URL	点击消息后拉起浏览器

参数名称	意义
NOTIFICATION_ACTION_INTENT	点击消息后启动 Intent
NOTIFICATION_ACTION_PACKAGE	点击消息后，启动应用

点击通知栏消息

```
/**
* 通知消息被点击，华为通道只有包含自定义参数的通知才会回调，其他渠道均会回调。
*
* <p>
* 信鸽通道所有参数均有效。
*
* 魅族回调时，{@link TACNotification} 中只有参数 ）{@link TACNotification#getText()} 返回的参数是有效的，包括通知消息的
* 标题、内容和自定义信息，其他字段均无效。
*
* 小米回调时，{@link TACNotification} 中参数 {@link TACNotification#getText()} 返回的参数是有有效的，包括通知消息的
* 标题、内容和自定义信息，同时 {@link TACNotification#getId()} 也是有效的，其他字段均无效。
*
* 华为回调时，{@link TACNotification} 中参数 {@link TACNotification#getText()} 返回的参数中只有 customContent 参数是有效的，
* 同时 {@link TACNotification#getId()} 也是有效的，其他字段均无效。
* 如果在发送通知栏消息时没有携带自定义参数，则不会回调。
* </p>
*
* @param context 上下文
* @param notification 通知消息
* @param channel 当前的推送渠道：信鸽、华为、小米、魅族
*/
void onNotificationClicked(Context context, TACNotification notification, PushChannel channel);
```

删除通知栏消息

```
/**
* 通知消息被删除，信鸽和魅族推送渠道下回调，小米和华为推送渠道不回调。
*
* <p>
* 信鸽通道所有参数均有效。
*
* 魅族回调时，{@link TACNotification} 中只有参数 ）{@link TACNotification#getText()} 返回的参数是有有效的，包括通知消息的
```

```
     * 标题、内容和自定义信息。
     * </p>
     *
     * @param context 上下文
     * @param notification 通知消息
     * @param channel 当前的推送渠道：信鸽、魅族
     */
    void onNotificationDeleted(Context context, TACNotification notification, PushChannel channel);
```

集成厂商推送通道

最近更新时间：2018-07-10 13:36:09

Messaging 服务支持将华为、小米、魅族三个手机厂商推送通道集成到到信鸽推送中。

准备工作

获取华为推送密钥

华为推送通道是由华为官方提供的系统级推送通道。在华为手机上，推送消息能够通过华为的系统通道抵达终端，并且无需打开应用就能够收到推送。

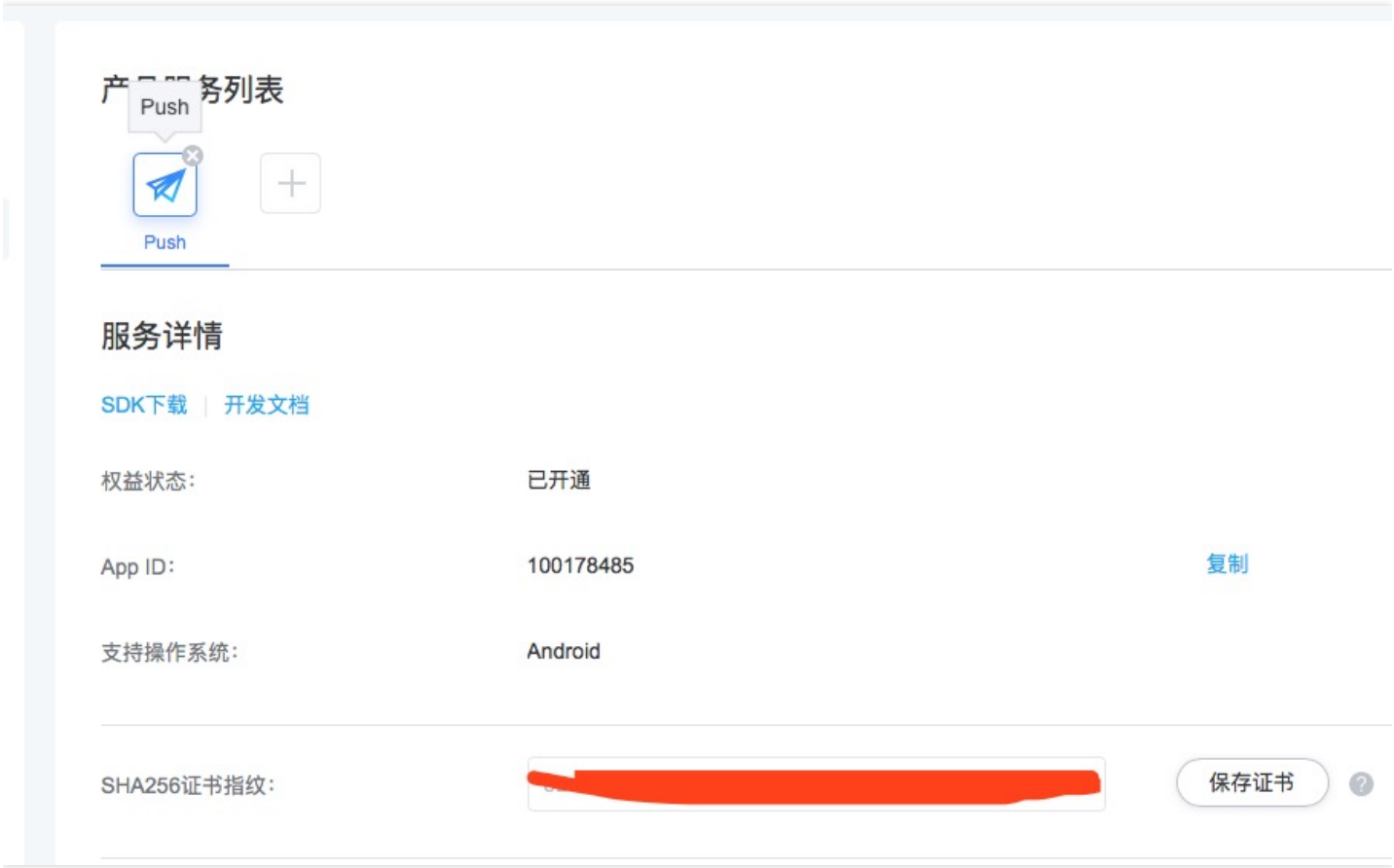
1. 打开 华为开放平台。

2. 注册/登录开发者账号（如果您是新注册账号，需进行实名认证）。

3. 在华为推送平台中新建应用（「应用包名」需跟您在移动开发平台上填写的包名保持一致）。

4. 配置 SHA256 证书指纹。

[配置示例]



获取 SHA256 证书指纹方法请参照 华为推送接入文档。

> **注意：**华为推送只有在签名发布包环境下才可以收到推送消息。

获取小米推送密钥

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

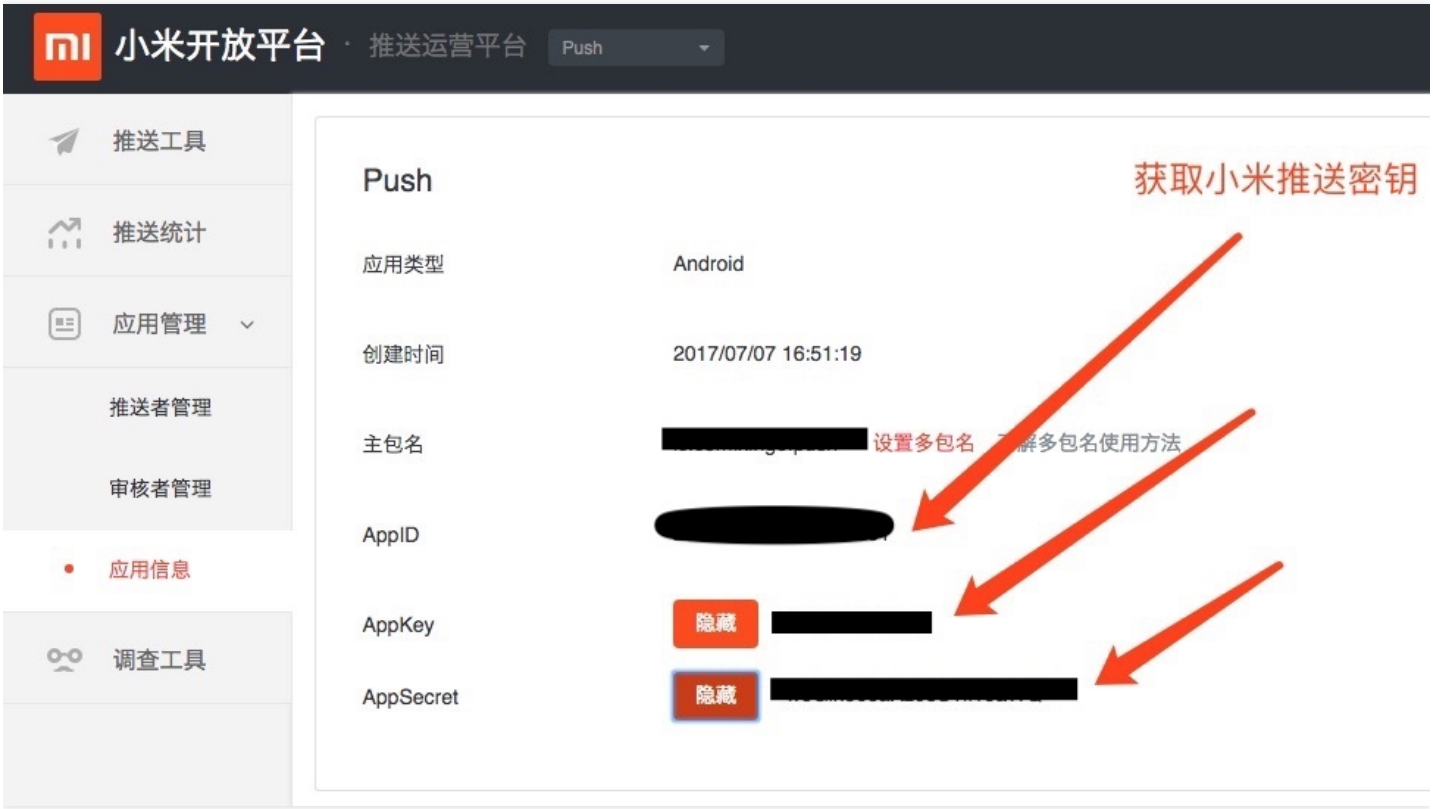
1. 打开 小米开放平台。

2. 注册/登录开发者账号（如果您是新注册账号，需进行实名认证）。

3. 注册应用并获取小米推送的密钥。

认证小米开发者：

获取小米推送密钥：



获取魅族推送密钥

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

> **注意：**魅族推送通道通知标题不超过32字符，通知内容不超过 100 字符。

1. 打开 魅族推送官网。

2. 注册/登录开发者账号（如果您是新注册账号，进行实名认证大约需要2天左右时间，具体请咨询魅族侧）。

3. 在 魅族推送平台 中新建应用（「应用包名」需跟您在信鸽填写的包名保持一致）。

4. 获取应用相关的信息。

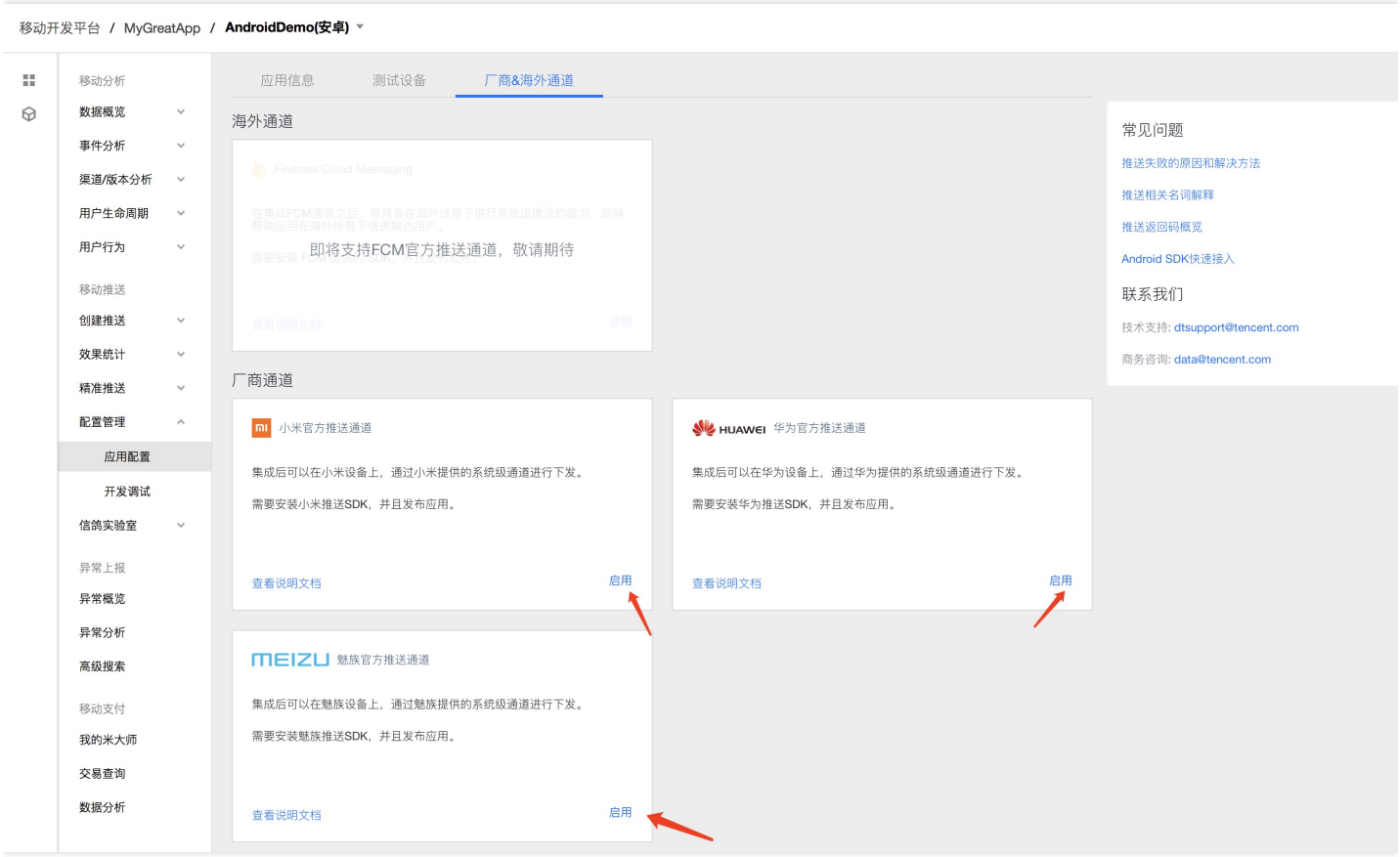
更多详情请参照 魅族开发文档。

集成 Messaging 服务（已做请跳过）

在集成第三方推送前，您必须首先 集成 Messaging 服务。

在 MobileLine 控制台上配置厂商推送信息

进入 MobileLine 控制台，选择【移动推送】>【配置管理】>【应用配置】，您可以分别配置小米、华为、魅族厂商推送通道的相关信息。



配置厂商推送的信息

在您的应用模块（通常是 App）目录下，新建一个名为 `tac_service_configurations_messaging.json` 的文件，内容如下：

```
{
"services": {
"messaging": {
"huaweiAppId": "申请的华为 appid",
"xiaomiAppId": "申请的小米 appid",
"xiaomiAppKey": "申请的小米 appkey",
```

```
"meizuAppId": "申请的魅族 appid",
"meizuAppKey": "申请的魅族 appkey"
}
}
}
```

验证厂商通道是否成功

验证方式一 ：通过回调函数中的 PushChannel 参数判断通道

首先您可以通过控制台发送**透传消息**，推送透传消息成功后，若 Messaging SDK 接收到了通知，则会回调在接入 Messaging 服务时配置的 TACMessagingReceiver 子类的 onMessageArrived() 方法，该方法中包含了 PushChannel 枚举类型参数，共有四个枚举值：

名称	含义
XINGE	Messaging 服务自带的信鸽通道
HUAWEI	华为厂商通道
XIAOMI	小米厂商通道
MEIZU	魅族厂商通道

您可以在回调时打印该参数来判断当前的推送通道：

```
// 收到透传消息后回调此接口
@Override
public void onMessageArrived(Context context, TACMessagingText tacMessagingText, PushChannel channel) {

Toast.makeText(context, "收到通知消息：" + pushChannel, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::onMessageArrived : pushChannel " + channel);
}
```

如果应用运行在华为手机上，则会打印如下日志：

```
I/messaging: MyReceiver::onMessageArrived : pushChannel HUAWEI
```

如果应用运行在小米手机上，则会打印如下日志：

```
I/messaging: MyReceiver::onNotificationArrived : pushChannel XIAOMI
```

如果应用运行在魅族手机上，则会打印如下日志：

```
I/messaging: MyReceiver::onNotificationArrived : pushChannel MEIZU
```

如果应用运行在其他手机上，则会打印如下日志：

```
I/messaging: MyReceiver::onNotificationArrived : pushChannel XINGE
```

验证方式二 ：通过厂商通道无需打开应用就能够收到推送的特性判断通道

1. 保持应用在前台运行，并通过控制台发送通知栏消息。

2. 如果应用收到消息，将应用退到后台，并且杀掉所有 App 进程。

3. 再次控制台发送通知栏消息，如果能够收到推送，则表明厂商通道集成成功。

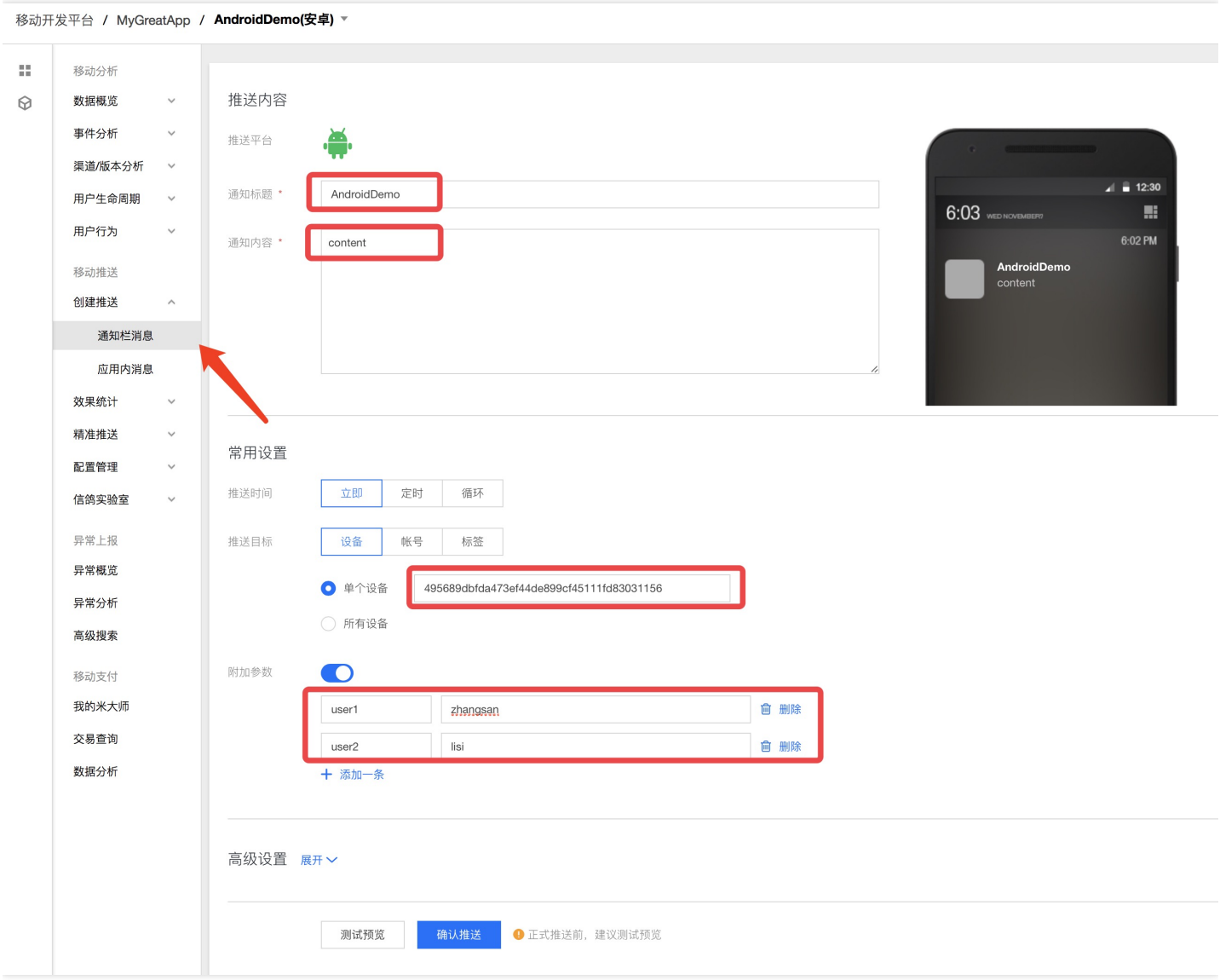
通过控制台推送消息

最近更新时间：2018-04-26 09:40:14

推送通知栏消息

在控制台上推送消息

打开 MobileLine 控制台，选择【创建推送】下的【通知栏消息】，并填写好 **通知标题** 和 **通知内容**，然后选择单选框中的【单个设备】，并将注册成功后回调时打印的设备唯一标识 token 信息拷贝到编辑框中（获取 token 信息请参见 这里），您也可以在推送时添加自定义参数，然后点击【确认推送】。



验证消息是否推送成功

推送通知栏消息成功后，若 Messaging SDK 接收到了通知，则会在通知栏上展示，并且会回调在接入 Messaging 服务时配置的 `TACMessagingReceiver` 子类的 `onNotificationArrived()` 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到通知栏消息后回调此接口。
@Override
public void onNotificationArrived(Context context, TACNotification tacNotification, PushChannel pushChannel) {

Toast.makeText(context, "收到通知消息：" + pushChannel, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::onNotificationArrived : notification is " + tacNotification + " pushChannel " + pushChannel);

}
```

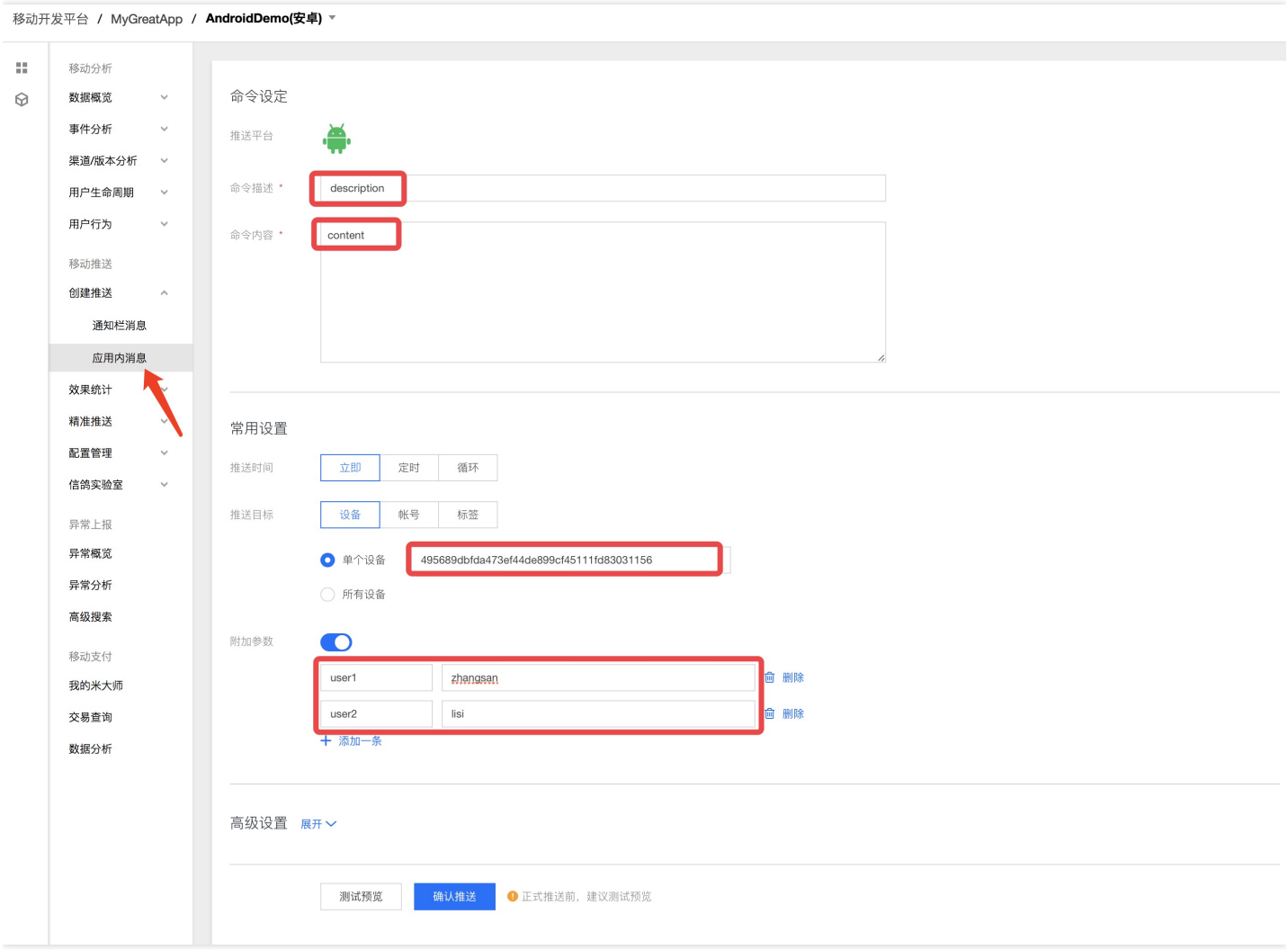
收到通知后，会弹出 toast 并打印如下日志：

```
I/messaging: MyReceiver::onNotificationArrived : notification is TACNotification [msgId=1463713536, title=AndroidDemo, content=content, customContent={}, activity=com.android.demo.MainActivity, notificationActionType1] pushChannel XINGE
```

推送透传消息

在控制台上推送消息

打开 MobileLine 控制台，选择【创建推送】下的【应用内消息】，并填写好 **命令描述** 和 **命令内容**，然后选择单选框中的【单个设备】，并将注册成功后回调时打印的设备唯一标识 token 信息拷贝到编辑框中（获取 token 信息请参见 这里），您也可以在推送时添加自定义参数，然后点击【确认推送】。



验证消息是否推送成功

推送透传消息成功后，若 Messaging SDK 接收到了通知，不会直接在通知栏上展示，仅仅会回调在接入 Messaging 服务时配置的 `TACMessagingReceiver` 子类的 `onMessagingArrived()` 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到透传消息后回调此接口。
@Override
public void onMessageArrived(Context context, TACMessagingText tacMessagingText, PushChannel channel) {

Toast.makeText(context, "收到透传消息：" + tacMessagingText, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::OnTextMessage : message is " + tacMessagingText+ " pushChannel " + channel);
}
```

收到通知后，会弹出 toast 并打印如下日志：

I/messaging: MyReceiver::OnTextMessage : message is TACMessagingText [title=**null**, content=content, customContent={user2=lisi, user1=zhangsan}, extra={"user1":"zhangsan","user2":"lisi"}] pushChannel XINGE

注意：在控制台上推送透传时，【命令描述】并不是 SDK 中 `TACMessagingText` 的 title 字段，该字段只有通过 api 推送时才有值。

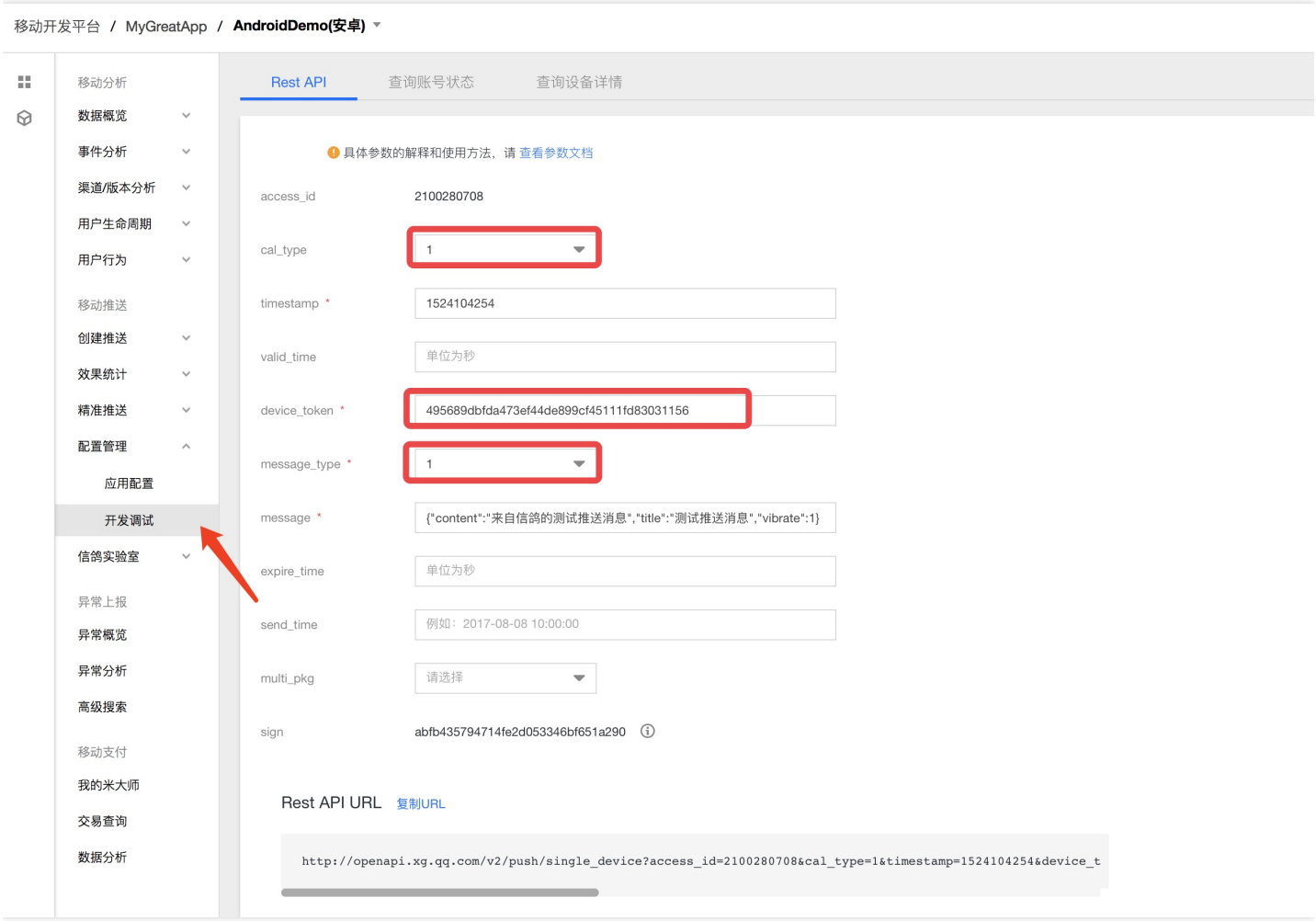
通过后台接口推送消息

最近更新时间：2018-04-26 09:40:45

推送通知栏消息

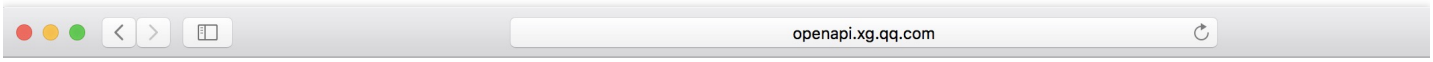
获取推送 URL

打开 MobileLine 控制台，选择【配置管理】下的【开发调试】，选择 cal_type 为 1（表示实时统计），然后将注册成功后回调时打印的设备唯一标识 token 信息拷贝到 device_token 编辑框中（获取 token 信息请参见 这里），最后选择 message_type 为 1（表示推送通知栏消息），然后点击【复制 URL】。



发起推送请求

将复制好的 URL 黏贴到浏览器中，并通过浏览器发起请求，请求成功后浏览器返回如下：



{"ret_code":0}

验证推送是否成功

发起推送请求后，若 Messaging SDK 接收到了通知，则会在通知栏上展示，并且会回调在接入 Messaging 服务时配置的 TACMessagingReceiver 子类的 onNotificationArrived() 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到通知栏消息后回调此接口。
@Override
public void onNotificationArrived(Context context, TACNotification tacNotification, PushChannel pushChannel) {

Toast.makeText(context, "收到通知消息：" + pushChannel, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::onNotificationArrived : notification is " + tacNotification + " pushChannel " + pushChannel);

}
```

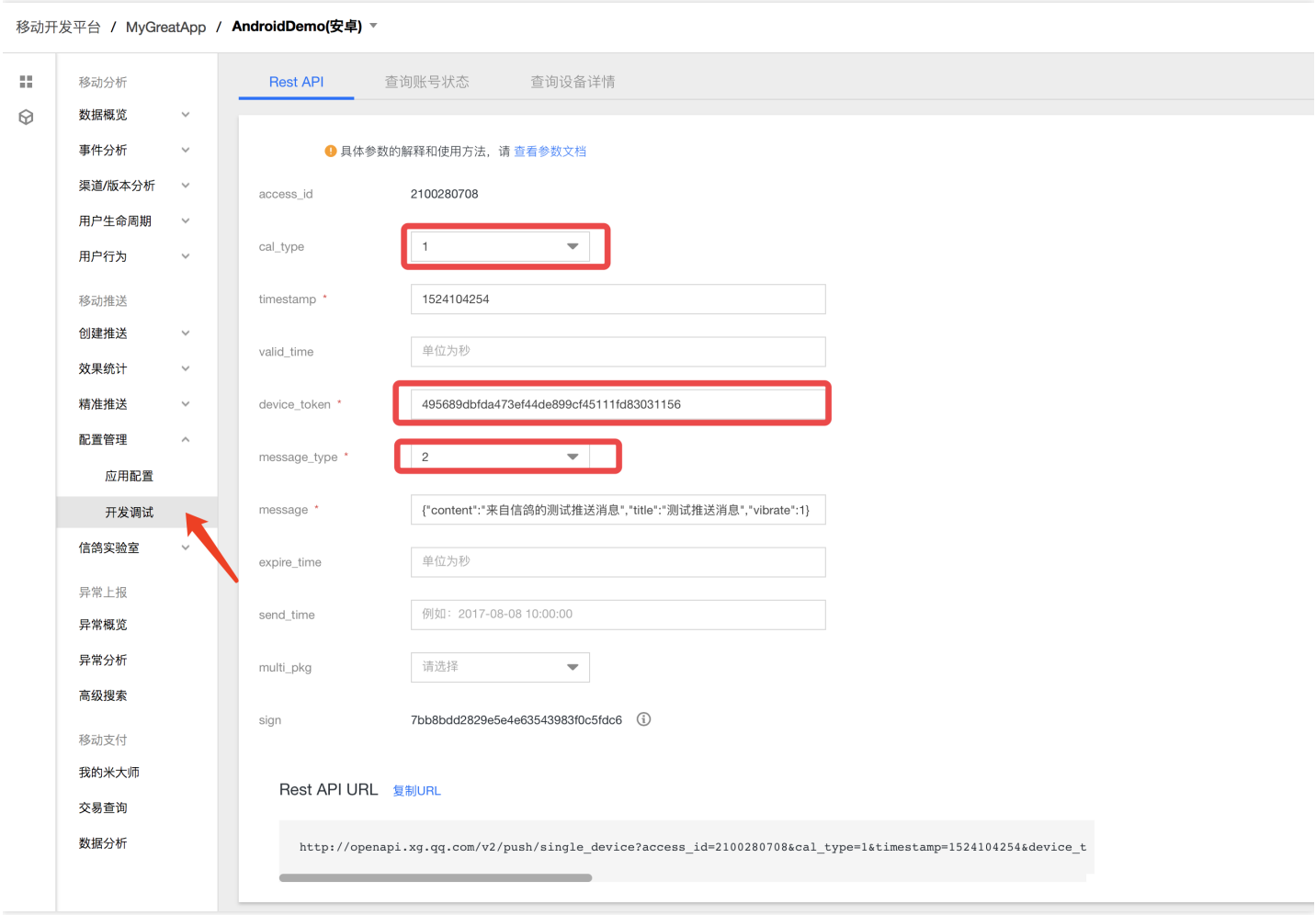
收到通知后，会弹出 toast 并打印如下日志：

```
I/messaging: MyReceiver::onNotificationArrived : notification is TACNotification [msgId=1056503921, title=测试推送消息, content=来自信鸽的测试推送消息, customContent={}, activity=com.android.demo.MainActivity, notificationActionType1] pushChannel XINGE
```

推送透传消息

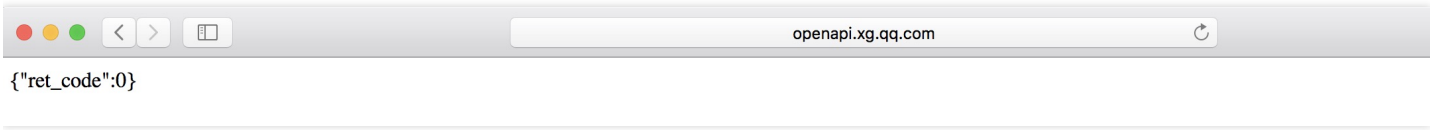
获取推送 URL

打开 MobileLine 控制台，选择【配置管理】下的【开发调试】，选择 cal_type 为 1（表示实时统计），然后将注册成功后回调时打印的设备唯一标识 token 信息拷贝到 device_token 编辑框中（获取 token 信息请参见 这里），最后选择 message_type 为 2（表示推送透传消息），然后点击【复制 URL】。



发起推送请求

将复制好的 URL 黏贴到浏览器中，并通过浏览器发起请求，请求成功后浏览器返回如下：



验证推送是否成功

推送透传消息成功后，若 Messaging SDK 接收到了通知，不会直接在通知栏上展示，仅仅会回调在接入 Messaging 服务时配置的 `TACMessagingReceiver` 子类的 `onMessagingArrived()` 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到透传消息后回调此接口。
@Override
public void onMessageArrived(Context context, TACMessagingText tacMessagingText, PushChannel
```

```
channel) {

Toast.makeText(context, "收到透传消息：" + tacMessagingText, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::OnTextMessage : message is " + tacMessagingText+ " pushChannel "
+ channel);
}
```

收到通知后，会弹出 toast 并打印如下日志：

```
I/messaging: MyReceiver::OnTextMessage : message is TACMessagingText [title=测试推送消息, conten
t=来自信鸽的测试推送消息, customContent={}, extra=null] pushChannel XINGE
```

后台接口说明

以上【推送通知栏消息】和【推送透传消息】两部分可用仅仅作为您的调试使用，在生产环境下可以使用我们提供的 API 接口 和 SDK 来推送消息。

根据标签推送消息

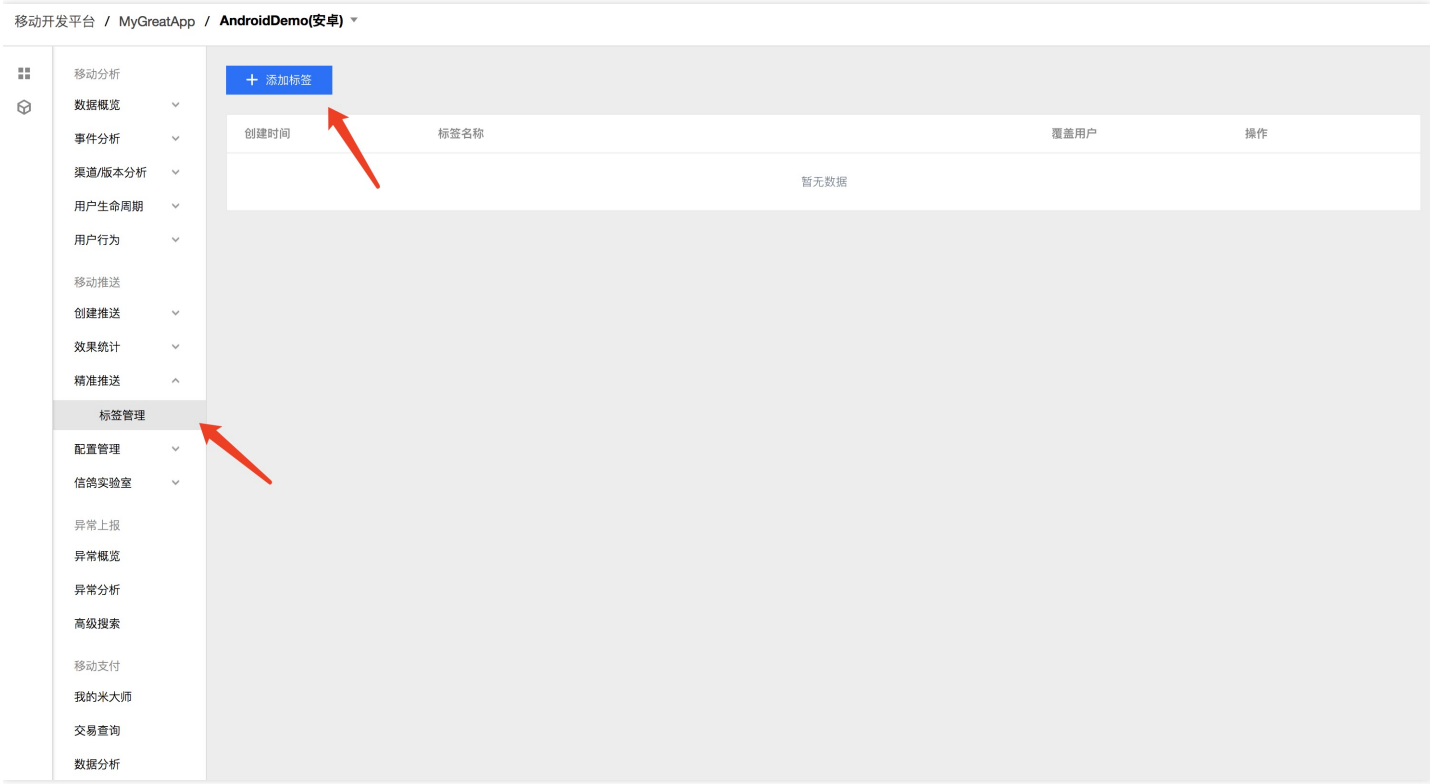
最近更新时间：2018-04-26 09:41:19

根据预设标签发送通知

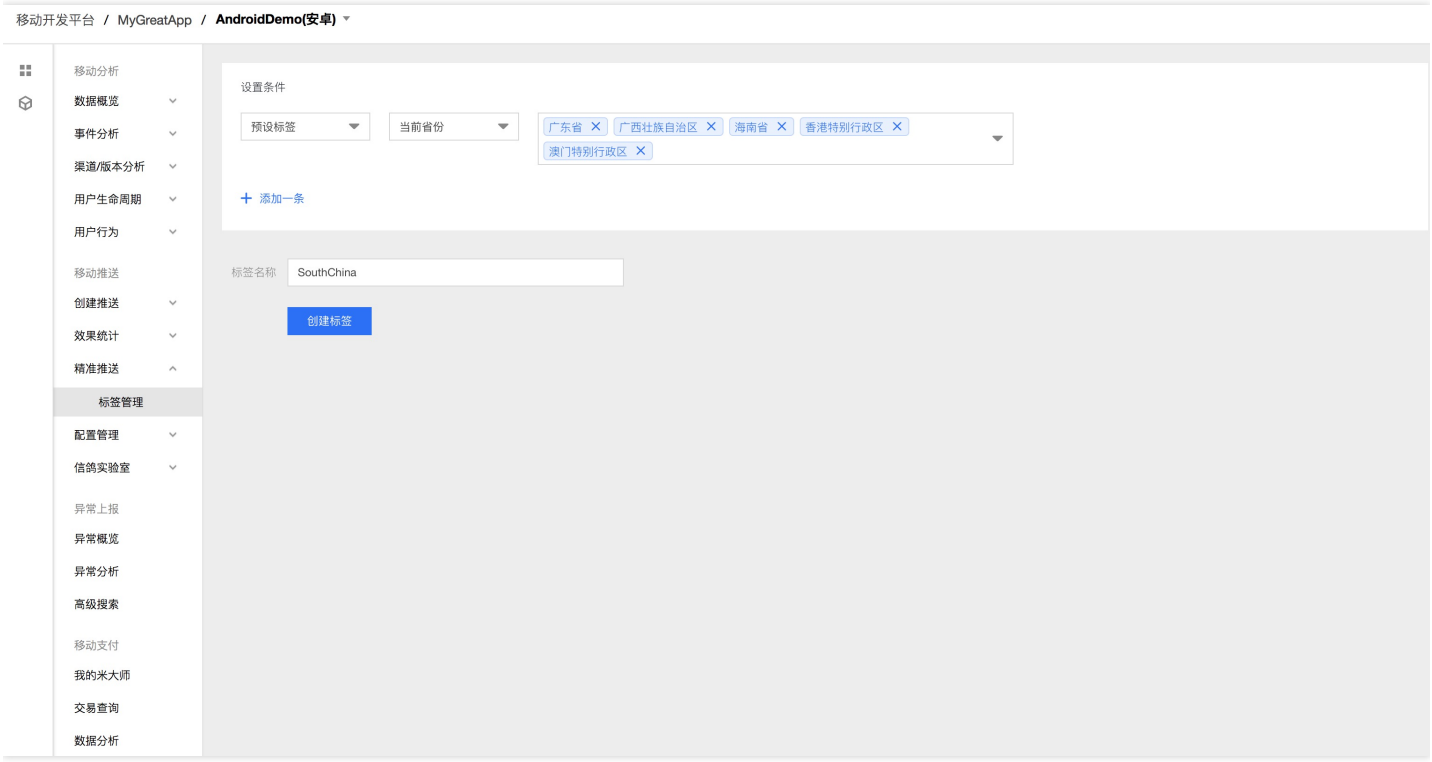
添加预设标签

预设标签是指我们给您预定义的标签，您可以在控制台上通过组合预设标签来定义新的标签规则：

首先登录 MobileLine 控制台，点击【移动推送】下的【精准推送】，并选择【标签管理】，然后点击【添加标签】按钮：

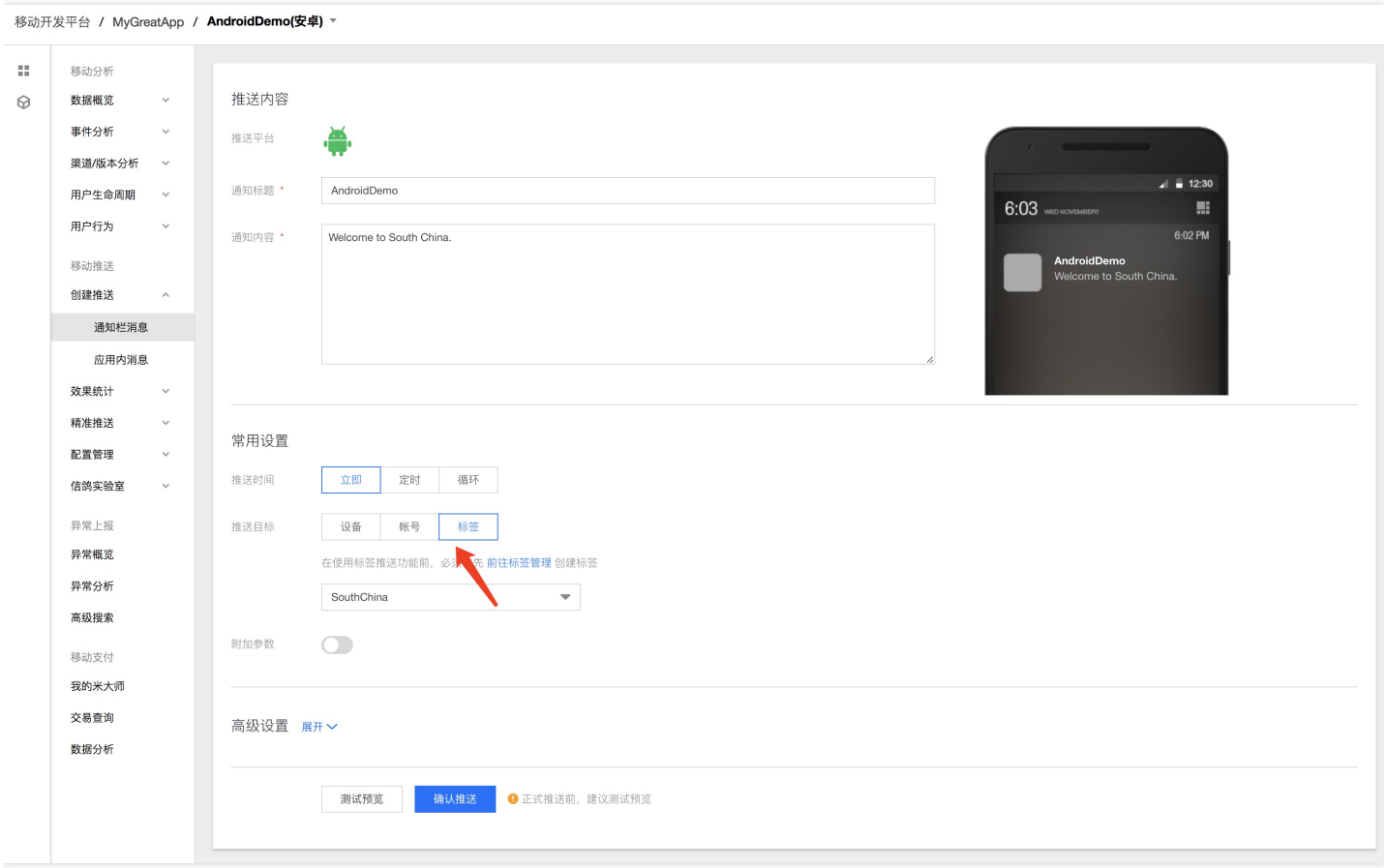


如图这里添加 SouthChina 标签，该标签下包含了广东省、广西壮族自治区、海南省、香港特别行政区和澳门特别行政区。



根据标签发送通知

打开 MobileLine 控制台，选择【创建推送】下的【通知栏消息】，并填写好 通知标题 和 通知内容，然后在推送目标下选择【标签】，并在下拉栏中选择 `SouthChina` 标签，最后点击【确认推送】。



验证通知是否发送成功

推送通知栏消息成功后，若 Messaging SDK 接收到了通知，则会在通知栏上展示，并且会回调在接入 Messaging 服务时配置的 TACMessagingReceiver 子类的 onNotificationArrived() 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到通知栏消息后回调此接口。
@Override
public void onNotificationArrived(Context context, TACNotification tacNotification, PushChannel pushChannel) {

Toast.makeText(context, "收到通知消息：" + pushChannel, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::onNotificationArrived : notification is " + tacNotification + " pushChannel " + pushChannel);

}
```

收到通知后，会弹出 toast 并打印如下日志：

```
I/messaging: MyReceiver::onNotificationArrived : notification is TACNotification [msgId=3887741210, title=AndroidDemo, content=Welcome to South China., customContent={}, activity=com.android.de
```

```
mo.MainActivity, notificationActionType1] pushChannel XINGE
```

> 注意：选择按标签推送只会推送到绑定了该标签的手机，如 `SouthChina` 标签只会推送给华南用户

根据自定义标签发送通知

添加自定义标签

自定义标签是指您可以根据您自己的业务场景给不同的用户打上不同的标签，如 `programer` 等。

```
// 在注册成功回调接口 onRegisterResult() 方法中会返回 TACMessagingToken 的实例，需要保存该实例。
TACMessagingToken tacMessagingToken = ...

String tag = "programer";
tacMessagingToken.bindTag(this, tag);
```

绑定标签成功后，会回调在接入 Messaging 服务时配置的 TACMessagingReceiver 子类的 onBindTagResult() 方法，您可以在该方法中调用如下方法打印日志：

```
// 绑定标签成功后回调此接口
@Override
public void onBindTagResult(Context context, int code, String tag) {

Toast.makeText(context, "绑定标签成功：tag = " + tag, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::onBindTagResult : code is " + code + " tag " + tag);

}
```

收到通知后，会弹出 toast 并打印如下日志：

```
I/messaging: MyReceiver::onBindTagResult : code is 0 tag programer
```

> 您也可以通过后台接口来绑定标签

根据标签发送通知

控制台上暂时不支持通过自定义标签来发送通知，您可以通过调用我们的后台接口来进行发送，详情请参见 标签推送 API 接口 或者 SDK 接口。

验证通知是否发送成功

推送通知栏消息成功后，若 Messaging SDK 接收到了通知，则会在通知栏上展示，并且会回调在接入 Messaging 服务时配置的 TACMessagingReceiver 子类的 onNotificationArrived() 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到通知栏消息后回调此接口。
@Override
public void onNotificationArrived(Context context, TACNotification tacNotification, PushChannel pushChannel) {

Toast.makeText(context, "收到通知消息：" + pushChannel, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::onNotificationArrived : notification is " + tacNotification + " pushChannel " + pushChannel);

}
```

收到通知后，会弹出 toast 并打印如下日志：

```
I/messaging: MyReceiver::onNotificationArrived : notification is TACNotification [msgId=3887741210, title=AndroidDemo, content=Welcome to South China., customContent={}, activity=com.android.demo.MainActivity, notificationActionType1] pushChannel XINGE
```

> 注意：选择按标签推送只会推送到绑定了该标签的手机，如 `programer` 标签只会推送给您主动打上标签的用户。

解绑自定义标签

您可以通过 SDK 来解绑自定义标签

```
// 在注册成功回调接口 onRegisterResult() 方法中会返回 TACMessagingToken 的实例，需要保存该实例。
TACMessagingToken tacMessagingToken = ...

String tag = "programer";
tacMessagingToken.unbindTag(this, tag);
```

解绑标签成功后，会回调在接入 Messaging 服务时配置的 TACMessagingReceiver 子类的 onUnbindTagResult() 方法，您可以在该方法中调用如下方法打印日志：

```
// 解绑标签成功后回调此接口
@Override
```

```
public void onUnbindTagResult(Context context, int code, String tag) {

Toast.makeText(context, "解绑标签成功：tag = " + tag, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::onUnbindTagResult : code is " + code + " tag " + tag);
}
```

收到通知后，会弹出 toast 并打印如下日志：

```
I/messaging: MyReceiver::onUnbindTagResult : code is 0 tag programer
```

> 您也可以通过后台接口来解绑标签

根据账户推送消息

最近更新时间：2018-04-26 09:41:45

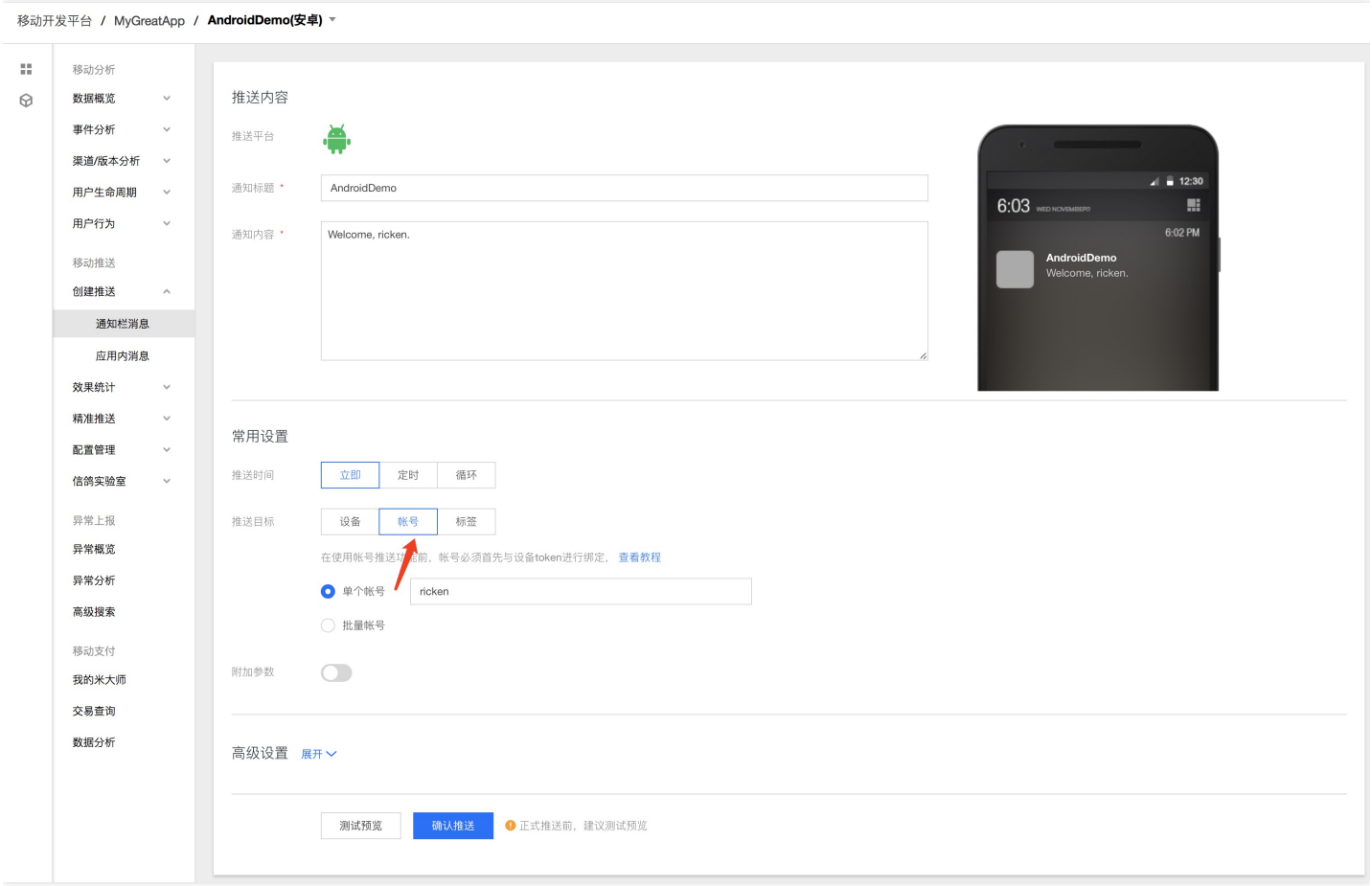
绑定账户

您可以在 Android 编程手册 中绑定账户。

> 帐号可以是邮箱、QQ号、手机号、用户名等任意类别的业务帐号

通过账户发送通知

打开 MobileLine 控制台，选择【创建推送】下的【通知栏消息】，并填写好 通知标题 和 通知内容，然后在推送目标下选择账户，并填写账户名称。最后点击【确认推送】。



您也可以使用我们的后台接口来推送通知，示例中的账户名称为 ricken

验证通知是否发送成功

推送通知栏消息成功后，若 Messaging SDK 接收到了通知，则会在通知栏上展示，并且会回调在接入 Messaging 服务时配置的 TACMessagingReceiver 子类的 onNotificationArrived() 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到通知栏消息后回调此接口。
@Override
public void onNotificationArrived(Context context, TACNotification tacNotification, PushChannel pushChannel) {

Toast.makeText(context, "收到通知消息：" + pushChannel, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::onNotificationArrived : notification is " + tacNotification + " pushChannel " + pushChannel);

}
```

收到通知后，会弹出 toast 并打印如下日志：

```
I/messaging: MyReceiver::onNotificationArrived : notification is TACNotification [msgId=1629316924, title=AndroidDemo, content=Welcome, ricken., customContent={}, activity=com.android.demo.MainActivity, notificationActionType1] pushChannel XINGE
```

本地推送消息

最近更新时间：2018-04-26 09:42:18

本地通知栏消息

本地推送消息

和普通通知栏消息类似，接收到本地通知栏消息后会在通知栏上展示，并且会回调在接入 Messaging 服务时配置的 TACMessagingReceiver 子类的 onNotificationArrived() 方法。

```
//新建本地通知
TACMessagingLocalMessage localMsg = new TACMessagingLocalMessage();

//设置本地消息类型，1:通知，2:消息
localMsg.setType(MessageType.NOTIFICATION);

// 设置消息标题
localMsg.setTitle("MobileLine");

//设置消息内容
localMsg.setContent("协助开发者快速搭建稳定高质量的移动应用");

//设置消息日期，
localMsg.setNotificationTime(new NotificationTime.NotificationTimeBuilder()
.setDate(2018, 3,1)
.setHourAndMinute(12,30)
.build());

//设置动作类型：
localMsg.setActionType(NotificationActionType.ACTION_OPEN_BROWSER);

// 设置URL
localMsg.setUrl("https://cloud.tencent.com/document/product/666");

// 是否覆盖原先build_id的保存设置。1覆盖，0不覆盖
localMsg.setStyleId(1);

//添加通知到本地
TACMessagingService messagingService = TACMessagingService.getInstance();
messagingService.addLocalNotification(this, localMsg);
```

验证消息是否推送成功

推送通知栏消息成功后，若 Messaging SDK 接收到了通知，则会在通知栏上展示，并且会回调在接入 Messaging 服务时配置的 `TACMessagingReceiver` 子类的 `onNotificationArrived()` 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到通知栏消息后回调此接口。
@Override
public void onNotificationArrived(Context context, TACNotification tacNotification, PushChannel pushChannel) {

Toast.makeText(context, "收到通知消息：" + pushChannel, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::onNotificationArrived : notification is " + tacNotification + " pushChannel " + pushChannel);

}
```

收到通知后，会弹出 toast 并打印如下日志：

```
I/messaging: MyReceiver::onNotificationArrived : notification is TACNotification [msgId=-1524115352319, title=MobileLine, content=协助开发者快速搭建稳定高质量的移动应用, customContent={}, activity=https://cloud.tencent.com/document/product/666, notificationActionType2] pushChannel XINGE
```

推送本地透传消息

本地推送消息

和普通透传消息类似，接收到本地透传消息后会回调在接入 Messaging 服务时配置的 TACMessagingReceiver 子类的 onMessageArrived() 方法。

```
//新建本地通知
TACMessagingLocalMessage localMsg = new TACMessagingLocalMessage();

//设置本地消息类型
localMsg.setType(MessageType.MESSAGE);

// 设置消息标题
localMsg.setTitle("MobileLine");

//设置消息内容
localMsg.setContent("协助开发者快速搭建稳定高质量的移动应用");
```

```
//设置消息日期，
localMsg.setNotificationTime(new NotificationTime.NotificationTimeBuilder()
.setDate(2018, 3,1)
.setHourAndMinute(12,30)
.build());

// 是否覆盖原先build_id的保存设置。1覆盖，0不覆盖
localMsg.setStyleId(1);

//添加通知到本地
TACMessagingService messagingService = TACMessagingService.getInstance();
messagingService.addLocalNotification(this, localMsg);
```

验证消息是否推送成功

推送透传消息成功后，若 Messaging SDK 接收到了通知，不会直接在通知栏上展示，仅仅会回调在接入 Messaging 服务时配置的 `TACMessagingReceiver` 子类的 `onMessagingArrived()` 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到透传消息后回调此接口。
@Override
public void onMessageArrived(Context context, TACMessagingText tacMessagingText, PushChannel channel) {

Toast.makeText(context, "收到透传消息：" + tacMessagingText, Toast.LENGTH_LONG).show();
Log.i("messaging", "MyReceiver::OnTextMessage : message is " + tacMessagingText+ " pushChannel " + channel);
}
```

收到通知后，会弹出 toast 并打印如下日志：

```
I/messaging: MyReceiver::OnTextMessage : message is TACMessagingText [title=MobileLine, content=协助开发者快速搭建稳定高质量的移动应用, customContent={}, extra=null] pushChannel XINGE
```

获取设备推送 token

最近更新时间：2018-04-26 09:42:35

获取设备推送 token

注册成功后，会返回设备 token，token 用于标识设备唯一性，同时也是 Messaging 服务维持与后台连接的唯一身份标识。在 Android SDK 中，您可以在接入 Messaging 服务时，通过重写注册的 `TACMessagingReceiver` 子类的 `onRegisterResult()` 方法来获取 token。

```
public class MyReceiver extends TACMessagingReceiver {

// 启动 Messaging 服务后，会自动向 Messaging 后台注册，注册完成后会回调此接口。
@Override
public void onRegisterResult(Context context, int errorCode, TACMessagingToken tacMessagingTok
en) {

if (tacMessagingToken != null) {
String token = tacMessagingToken.getTokenString(); // 获取推送 token
}
}

...
}
```

接入 Messaging 服务请参考 这里

添加通知样式

最近更新时间：2018-04-26 09:43:11

您可以根据自行需要设置通知样式，然后在后台推送通知的时候，可以设置样式的 ID。由于目前的定制 ROM 的限制，部分接口无法适配全部机型。

初始化通知样式

```
TACMessagingNotificationBuilder build = new TACMessagingNotificationBuilder();
build.setSound(RingtoneManager.getActualDefaultRingtoneUri(getApplicationContext(), RingtoneManager.TYPE_ALARM))
.setDefaults(Notification.DEFAULT_VIBRATE) // 振动
.setFlags(Notification.FLAG_NO_CLEAR); // 是否可清除

// 设置状态栏的通知图标
build.setNotificationLargeIcon(R.drawable.tencent_cloud); // 将腾讯云图片作为通知图标
build.setIcon(R.drawable.tencent_cloud);
```

将通知样式保存到本地

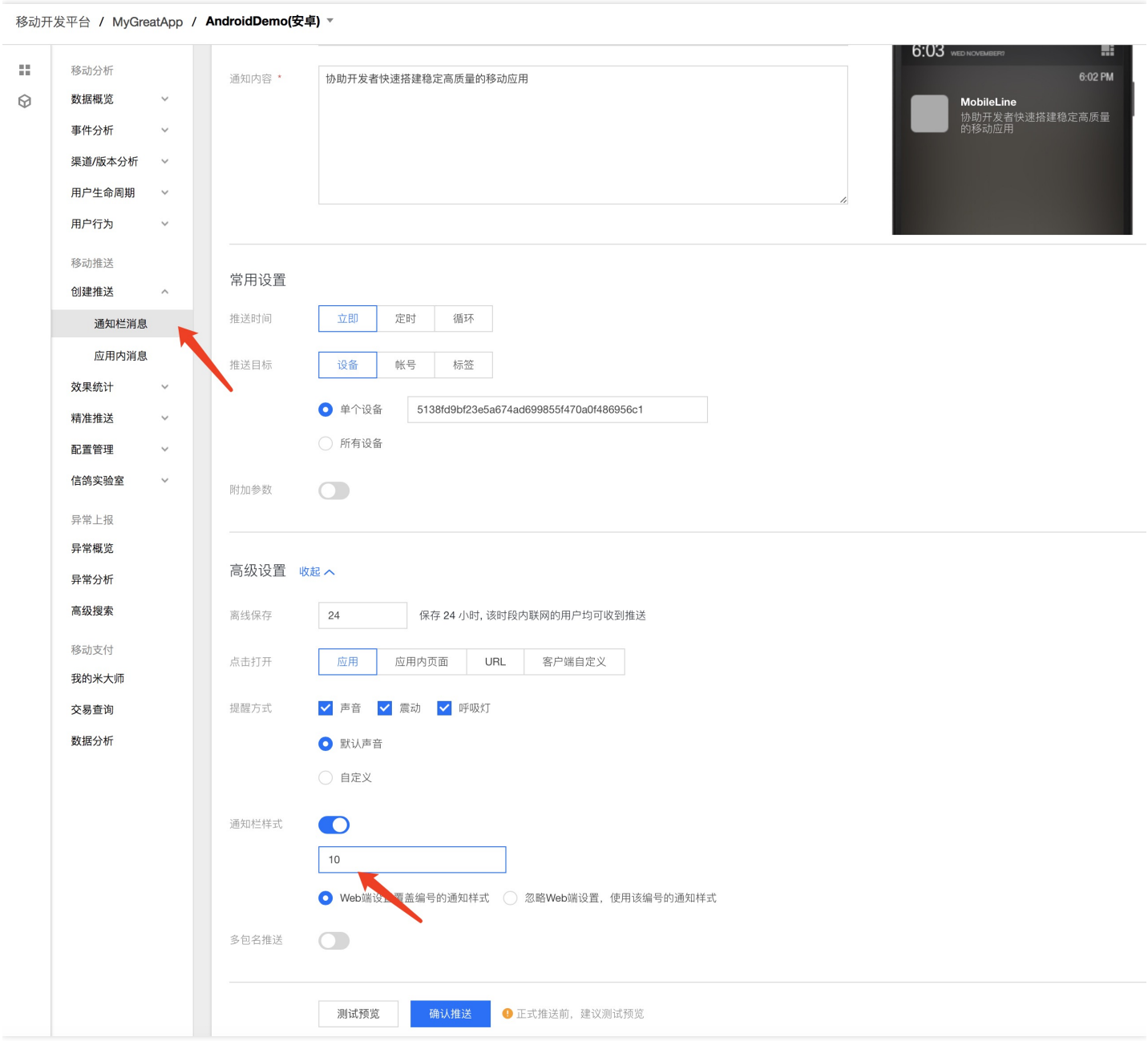
初始化好通知样式后，你可以将通知样式保存到本地，之后通过通知样式的 ID 号即可发送对应的通知。

```
// notificationBuilderId 是样式的 id，id 的范围为 0 ~ 4096
int notificationBuilderId = 10;
TACMessagingService messagingService = TACMessagingService.getInstance();
messagingService.addNotificationBuilder(this, notificationBuilderId, build);
```

验证通知样式是否添加成功

在控制台上发送通知栏消息

打开 MobileLine 控制台，选择【创建推送】下的【通知栏消息】，并填写好 **通知标题** 、 **通知内容** 以及 **token** 后，选择【高级设置】并【点击通知栏样式】，然后填写设置好的通知样式 ID（示例这里为10），最后点击【确认发送】。



查看通知栏消息

在控制台发送通知后，通知栏上会展示一个和该 ID 对应样式的通知栏消息：

通知样式其他设置

```
TACMessagingNotificationBuilder build = new TACMessagingNotificationBuilder();

// 设置声音
```

```
build.setSound(RingtoneManager.getActualDefaultRingtoneUri(getApplicationContext(), RingtoneM
anager.TYPE_ALARM))

build..setDefaults(Notification.DEFAULT_VIBRATE) // 振动

build..setFlags(Notification.FLAG_NO_CLEAR); // 是否可清除

// 设置自定义通知layout,通知背景等可以在layout里设置
build.setLayoutId(R.layout.notification);

// 设置自定义通知内容id
build.setLayoutTextId(R.id.content);

// 设置自定义通知标题id
build.setLayoutTitleId(R.id.title);

// 设置自定义通知图片资源
build.setLayoutIconDrawableId(R.drawable.logo);

// 设置状态栏的通知小图标
build.setIcon(R.drawable.right);

// 设置时间id
build.setLayoutTimeId(R.id.time);

// 若不设定以上自定义layout，又想简单指定通知栏图片资源
build.setNotificationLargeIcon(R.drawable.ic_action_search);
```

启动和停止服务

最近更新时间：2018-04-26 09:43:25

MoblieLine SDK 会在您的应用启动时自动启动 Messaging 服务，**一般情况下不需要您自己启动和停止 Messaging 服务。**

启动 Messaging 服务

手动启动服务

您可以调用如下代码启动 Messaging 服务：

```
TACMessagingService.getInstance().start(context);
```

> 您必须在手动停止 Messaging 服务后才能重新启动 Messaging 服务，请不要重复启动。

验证服务是否启动

启动成功后，Messaging SDK 会回调在接入 Messaging 服务时配置的 `TACMessagingReceiver` 子类的 `onRegisterResult()` 方法，您可以在该方法中调用如下方法打印日志：

```
// 启动 Messaging 服务后，会自动向 Messaging 后台注册，注册完成后会回调此接口。
@Override
public void onRegisterResult(Context context, int errorCode, TACMessagingToken token) {

Toast.makeText(context, "注册结果返回：" + token, Toast.LENGTH_SHORT).show();
Log.i("messaging", "MyReceiver::OnRegisterResult : code is " + errorCode + ", token is " + token.getTo
kenString());
}
```

收到通知后，会弹出 toast 并打印如下日志：

```
I/messaging: MyReceiver::OnRegisterResult : code is 0, token is 495689dbfda473ef44de899cf45111fd8
3031156
```

停止 Messaging 服务

手动停止服务

您可以调用如下代码启动 Messaging 服务：

```
TACMessagingService.getInstance().stop(context);
```

> 您必须在手动停止 Messaging 服务后才能重新启动 Messaging 服务，请不要重复启动。

验证服务是否停止

停止服务成功后，Messaging SDK 会回调在接入 Messaging 服务时配置的 `TACMessagingReceiver` 子类的 `onUnregisterResult()` 方法，您可以在该方法中调用如下方法打印日志：

```
// 停止服务后回调此接口。
@Override
public void onUnregisterResult(Context context, int code) {

Toast.makeText(context, "取消注册结果返回：" + code, Toast.LENGTH_SHORT).show();
Log.i("messaging", "MyReceiver::onUnregisterResult : code is " + code);
}
```

收到通知后，会弹出 toast 并打印如下日志：

```
I/messaging: MyReceiver::onUnregisterResult : code is 0
```

iOS 文档
iOS 快速入门

最近更新时间：2018-06-12 15:50:49

移动开发平台（MobileLine）使用起来非常容易，只需要简单的 4 步，您便可快速接入移动崩溃监测。接入后，您即可获得我们提供的各项能力，减少您在开发应用时的重复工作，提升开发效率。

准备工作

为了使用移动开发平台（MobileLine）iOS 版本的 SDK，您首先需要一个 iOS 工程，这个工程可以是您现有的工程，也可以是您新建的一个空的工程。

第一步：创建项目和应用

在使用我们的服务前，您必须先在 MobileLine 控制台上 创建项目和应用。

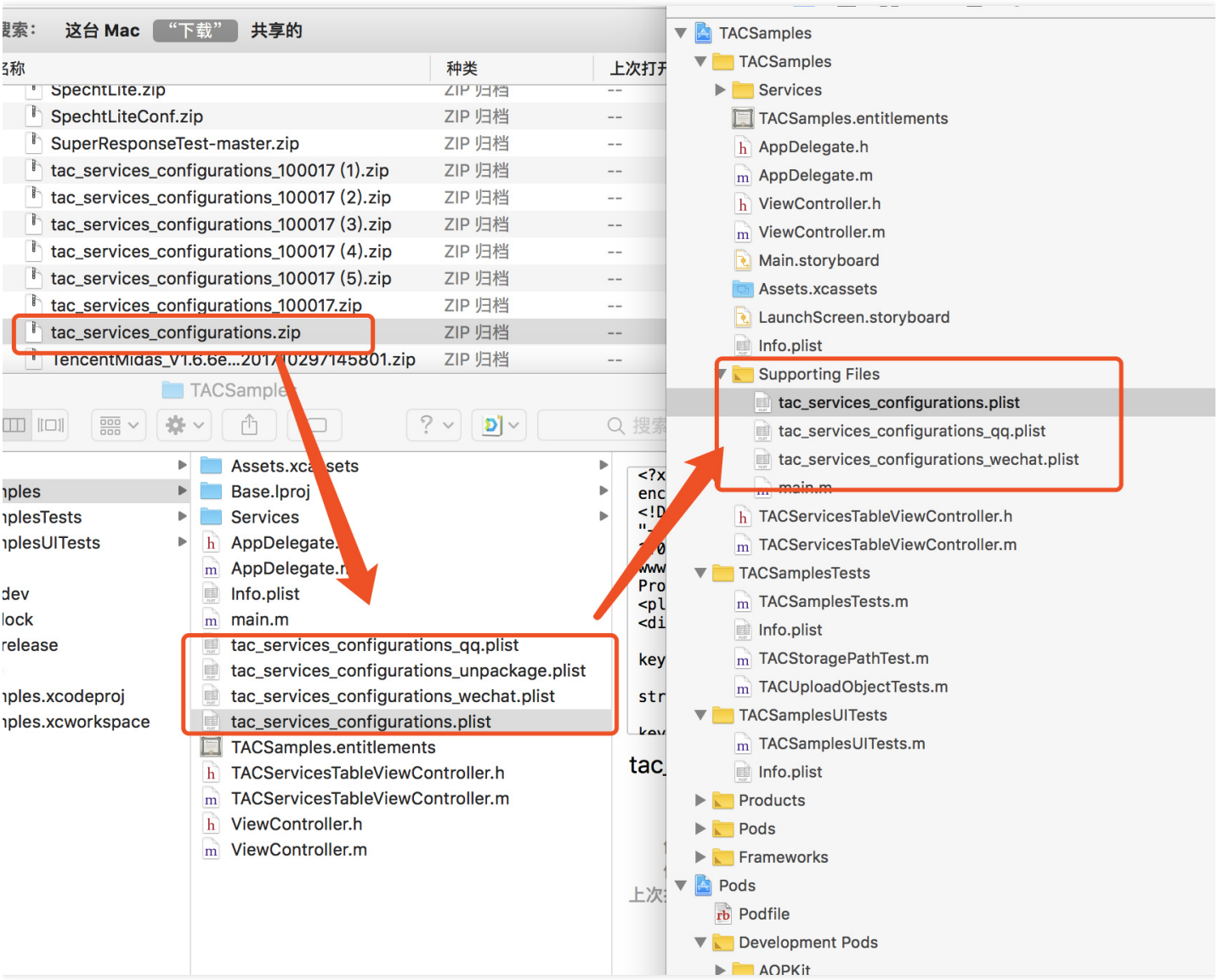
> 如果您已经在 MobileLine 控制台上创建过了项目和应用，请跳过此步。

第二步：添加配置文件

> 如果您已经添加过配置文件，请跳过此步。

创建好应用后，您可以点击红框中的【下载配置】来下载该应用的配置文件的压缩包：

解压后将 tac_services_configurations.plist 文件集成进项目中。其中有一个 tac_services_configurations_unpackage.plist 文件，请将该文件放到您工程的根目录下面(**切记不要将改文件添加进工程中**)。 添加好配置文件后，继续点击【下一步】。



注意：

请您按照图示来添加配置文件， `tac_service_configurations_unpackage.plist` 文件中包含了敏感信息，请不要打包到 apk 文件中，MobileLine SDK 也会对此进行检查，防止由于您误打包造成的敏感信息泄露。

第三步：集成 SDK

如果还没有 Podfile，请创建一个。

```
$ cd your-project directory
$ pod init
```

并在您的 Podfile 文件中添加移动开发平台（MobileLine）的私有源：

```
source "https://git.cloud.tencent.com/qcloud_u/cocopoads-repo"
source "https://github.com/CocoaPods/Specs"
```

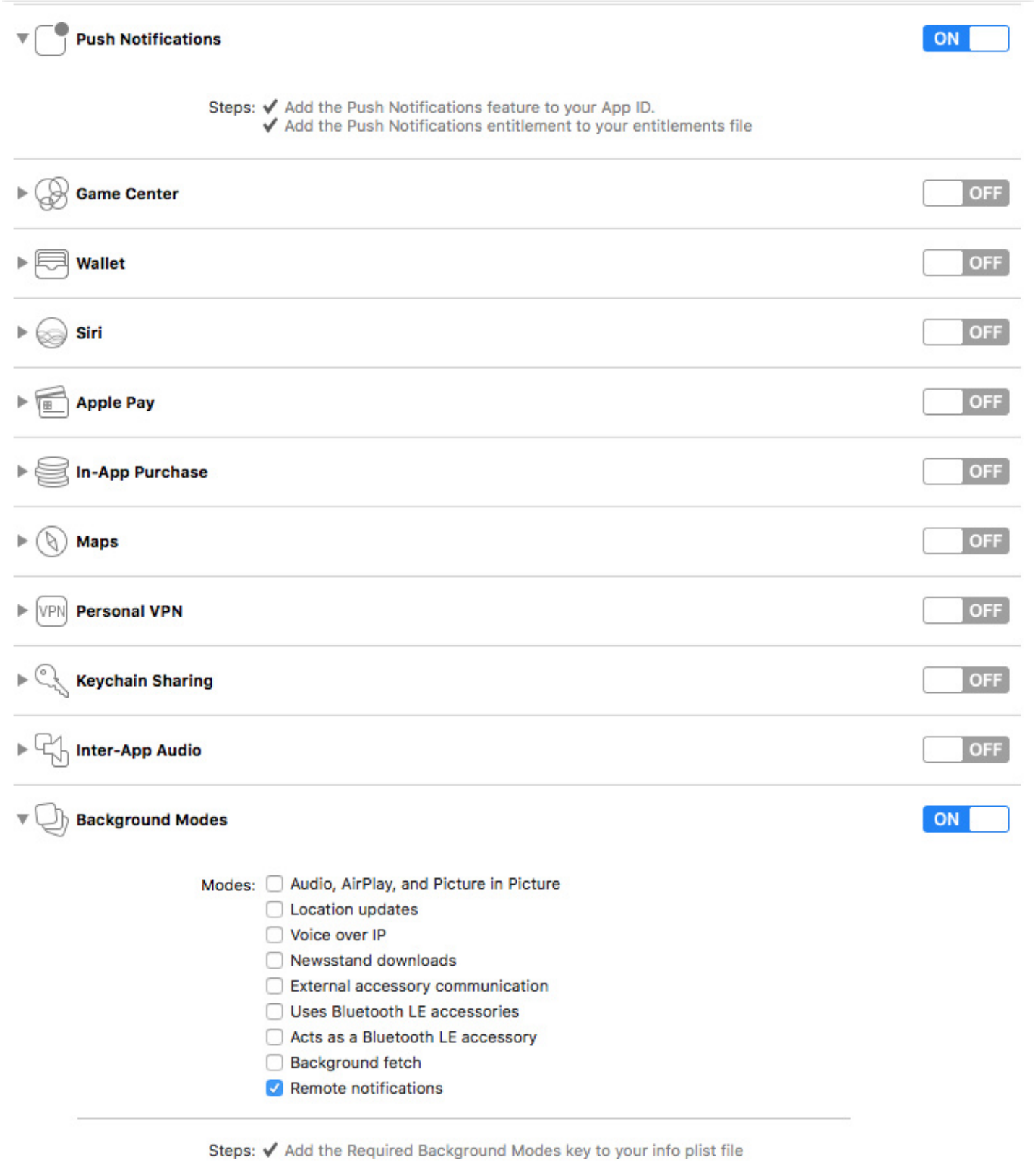
添加 Podfile 依赖：

```
pod 'TACMessaging'
```

并运行命令

```
pod update
```

并且在工程配置和后台模式中打开推送,如下图所示：



启动服务

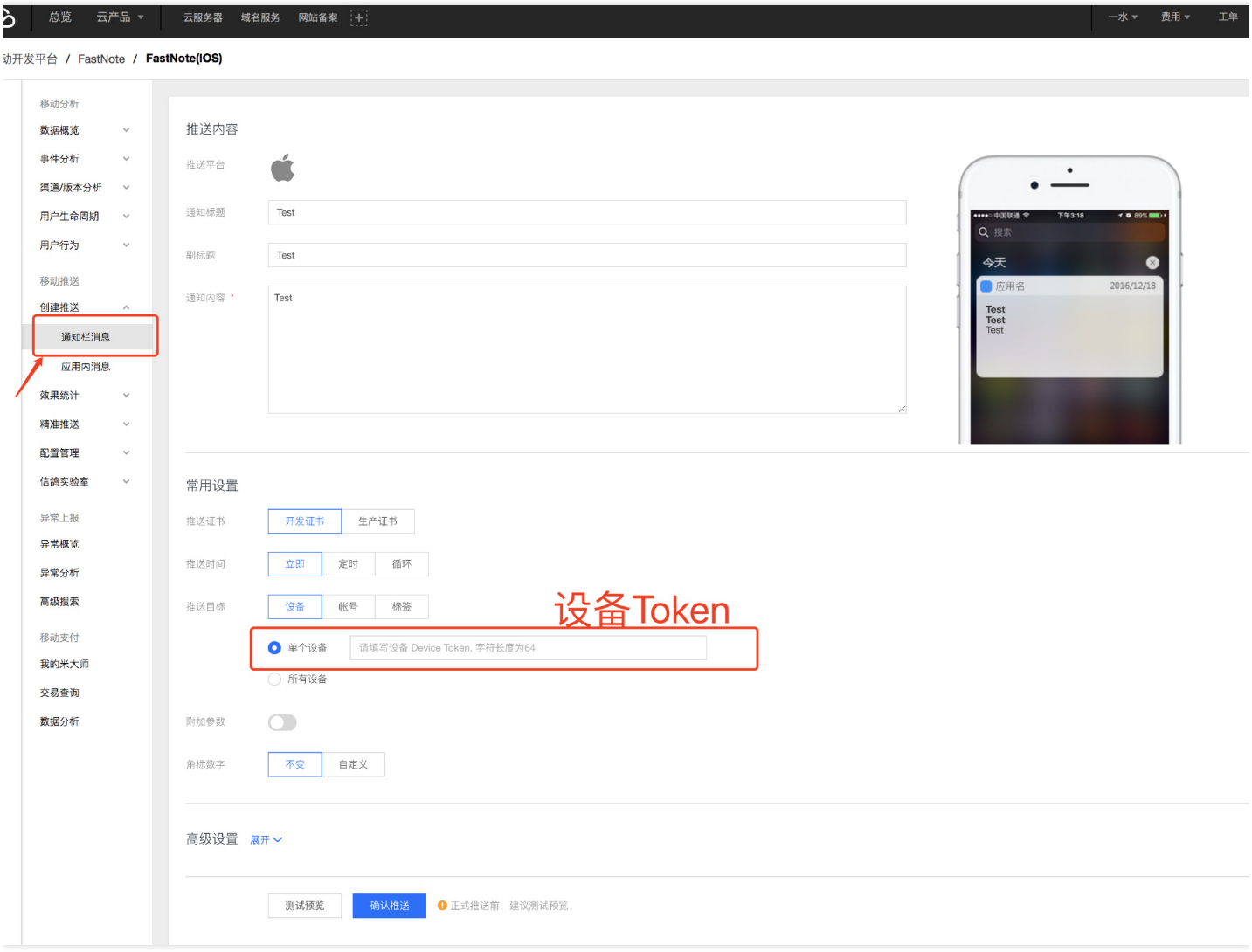
移动推送 服务无需启动，到此您已经成功接入了 MobileLine 移动推送服务。

第四步 验证

请先参考 iOS 推送证书设置指南 设置开发和发布证书

在控制台上推送消息

打开 MobileLine 控制台，选择【创建推送】下的【通知栏消息】，并填写好 **通知标题** 和 **通知内容**，然后选择单选框中的【单个设备】，并将注册成功后回调时打印的设备唯一标识 token 信息拷贝到编辑框中，您也可以在推送时添加自定义参数，然后点击【确认推送】。



验证通知是否发送成功

推送通知栏消息成功后，若 Messaging SDK 接收到了通知，如果您的程序在前台则会调用 `AppDelegate` 的 `application:didReceiveRemoteNotification` 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到通知栏消息后回调此接口。
- (void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userI
nfo
{
NSLog(@"got messaging");
}
```

收到通知后终端将会输出日志：

```
2018-04-20 16:37:06.983857+0800 TACSamples[384:51189] got messaging
```

后续步骤

了解 MobileLine：

- 查看 MoblieLine 应用示例

向您的应用添加 MobileLine 功能：

- 借助 Analytics 深入分析用户行为。
- 借助 messaging 向用户发送通知。
- 借助 crash 确定应用崩溃的时间和原因。
- 借助 storage 存储和访问用户生成的内容（如照片或视频）。
- 借助 authorization 来进行用户身份验证。
- 借助 payment 获取微信和手 Q 支付能力

iOS 推送证书设置指南

最近更新时间：2018-04-24 19:28:24

iOS 证书命令

证书有效期

```
openssl x509 -in xxx.pem -noout -dates
```

连接 APNS 测试证书是否合法

1. 开发环境

```
openssl s_client -connect gateway.sandbox.push.apple.com:2195 -cert xxx.pem -key xxx.pem
```

2. 生产环境

```
openssl s_client -connect gateway.push.apple.com:2195 -cert xxx.pem -key xxx.pem
```

指南介绍

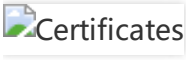
本指南用于介绍 iOS 证书如何设置

配置好证书后请前往 iOS SDK 完整接入

设置步骤

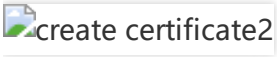
首先，登录 苹果开发者中心网站。然后点击 Certificates,Identifiers & Profiles

然后点击 Certificates

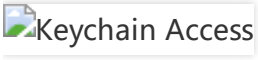


选中需要制作 Push 证书的应用，勾选 Push 服务

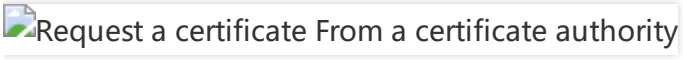
下面以制作开发证书为例演示。点击 Create Certificate...



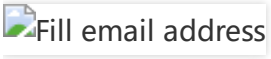
然后打开 Keychain Access 工具



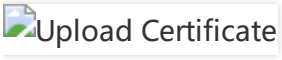
选择 Request a Certificate From a Certificate Authority...



填写邮件地址，其它留空，继续，证书将保存到本地

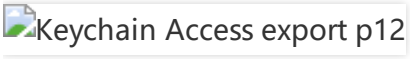


返回网站，选择刚才创建的文件上传



成功后，下载到本地

再次打开Keychain Access。选中 Push 证书导出，选中一行，导出的格式为 p12



生成 pem 格式的证书

完成上述操作后，打开终端，进入到 p12 文件所在执行以下命令:

```
openssl pkcs12 -in CertificateName.p12 -out CertificateName.pem -nodes
```

则生成了 CertificateName.pem 证书，上传到移动开发平台（MobileLine）控制台则可以进行消息推送。

启动和停止服务

最近更新时间：2018-04-24 19:28:49

MoblieLine SDK 会在您的应用启动时自动启动 Messaging 服务，**一般情况下不需要您自己启动和停止 Messaging 服务。**

服务启动与停止

当您集成了 Messaging 服务之后，系统将会在程序启动时默认启动该服务。

如果您不希望在启动的时候默认启动 Messaging 服务，可以在配置中将其关闭 (例如在 AppDelegate 中加入如下代码)：

Objective-C 代码示例：

```
TACApplicationOptions* options = [TACApplicationOptions defaultApplicationOptions];
options.messagingOptions.autoStart = NO;
```

Swift 代码示例：

```
let options = TACApplicationOptions.default()
options?.messagingOptions.autoStart = false
```

手动开启服务

Objective-C 代码示例：

```
[[TACMessagingService defaultService] startReceiveNotifications];
```

Swift 代码示例：

```
TACMessagingService.default().startReceiveNotifications()
```

手动关闭服务

Objective-C 代码示例：

```
[[TACMessagingService defaultService] stopReceiveNotifications];
```

Swift 代码示例：

```
TACMessagingService.default().stopReceiveNotifications()
```

如果您希望在特定的时候关闭服务。

根据标签推送消息

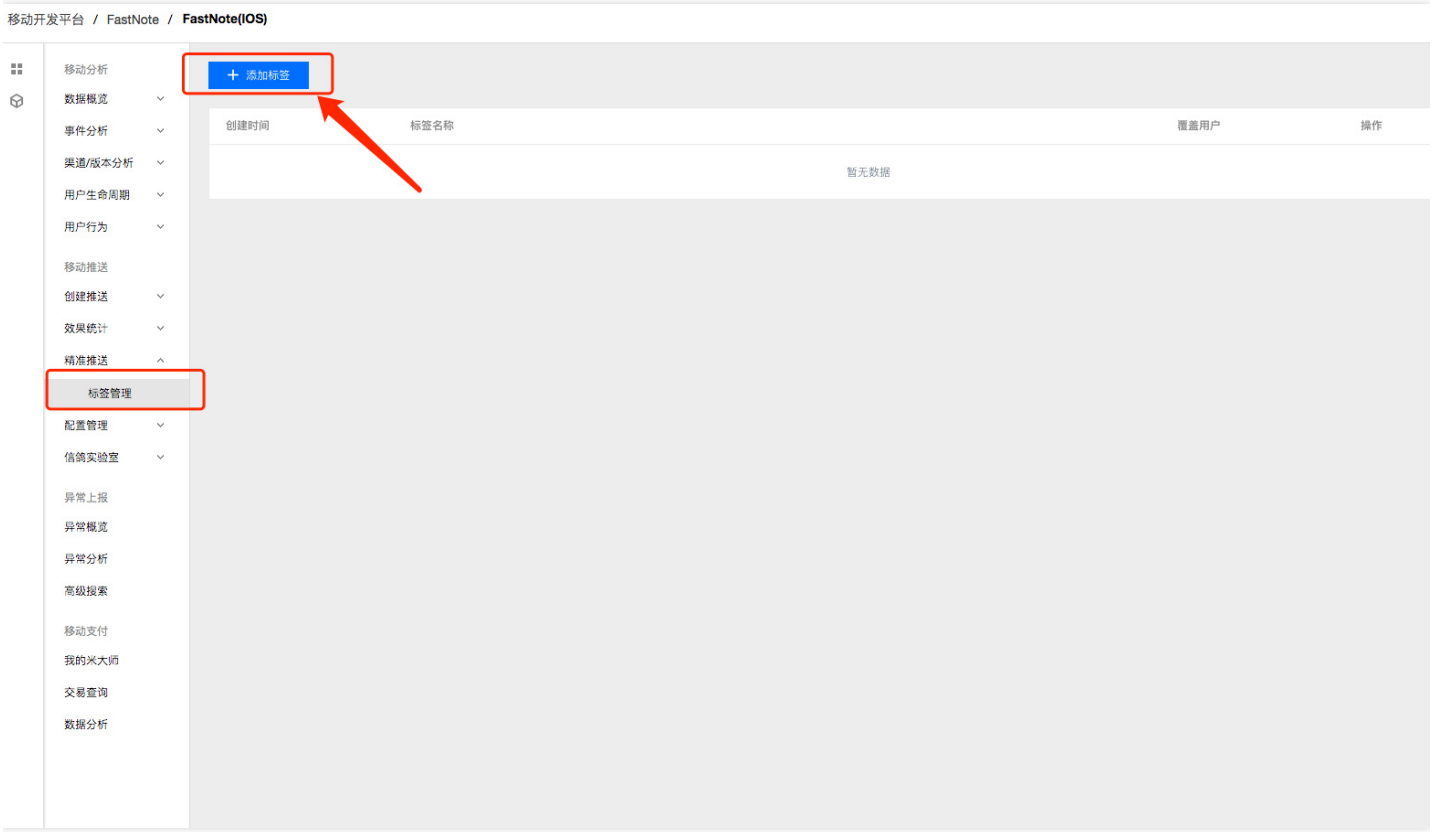
最近更新时间：2018-04-24 19:29:38

根据预设标签发送通知

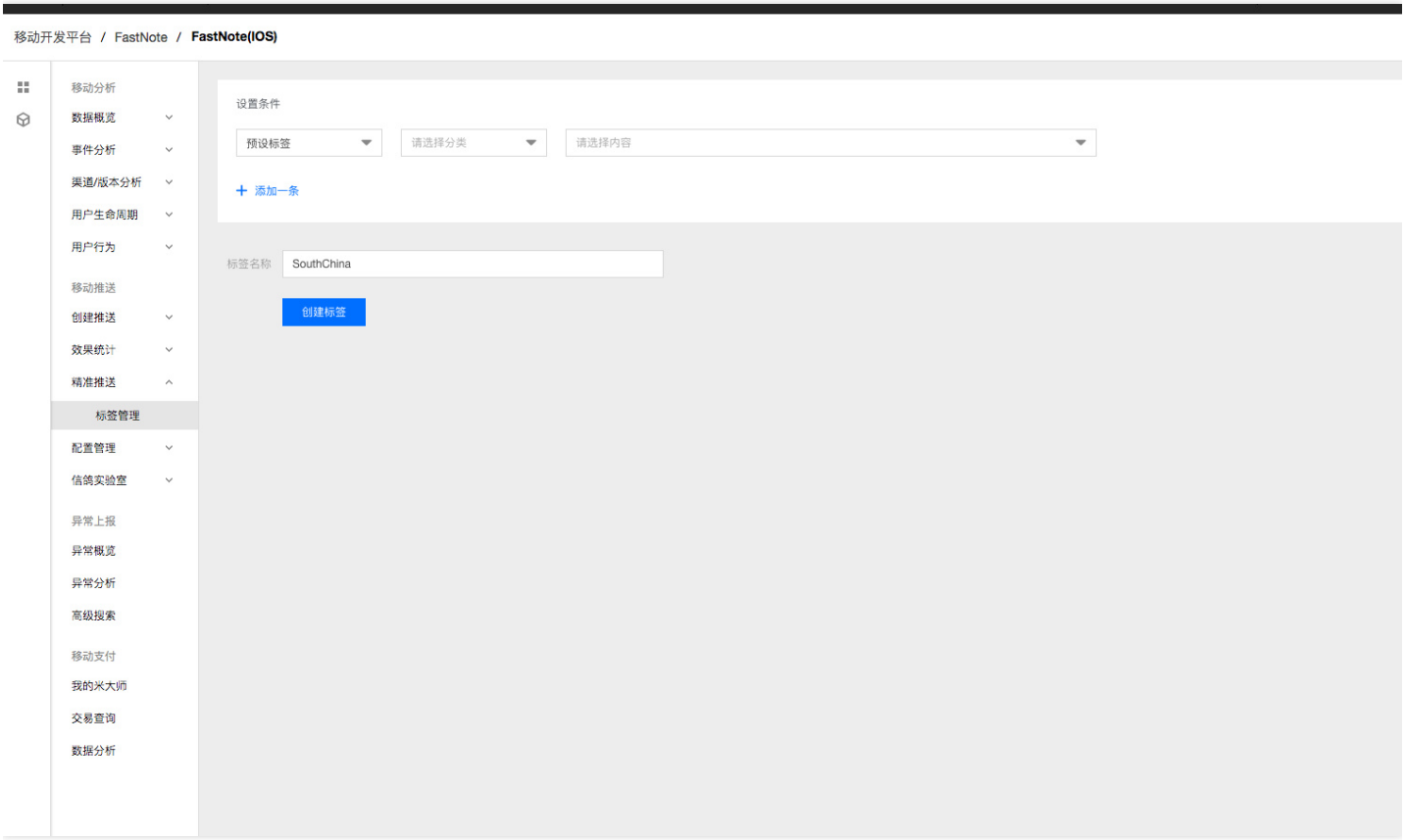
添加预设标签

预设标签是指我们给您预定义的标签，您可以在控制台上通过组合预设标签来定义新的标签规则：

首先登录 MobileLine 控制台，点击【移动推送】下的【精准推送】，并选择【标签管理】，然后点击【添加标签】按钮：

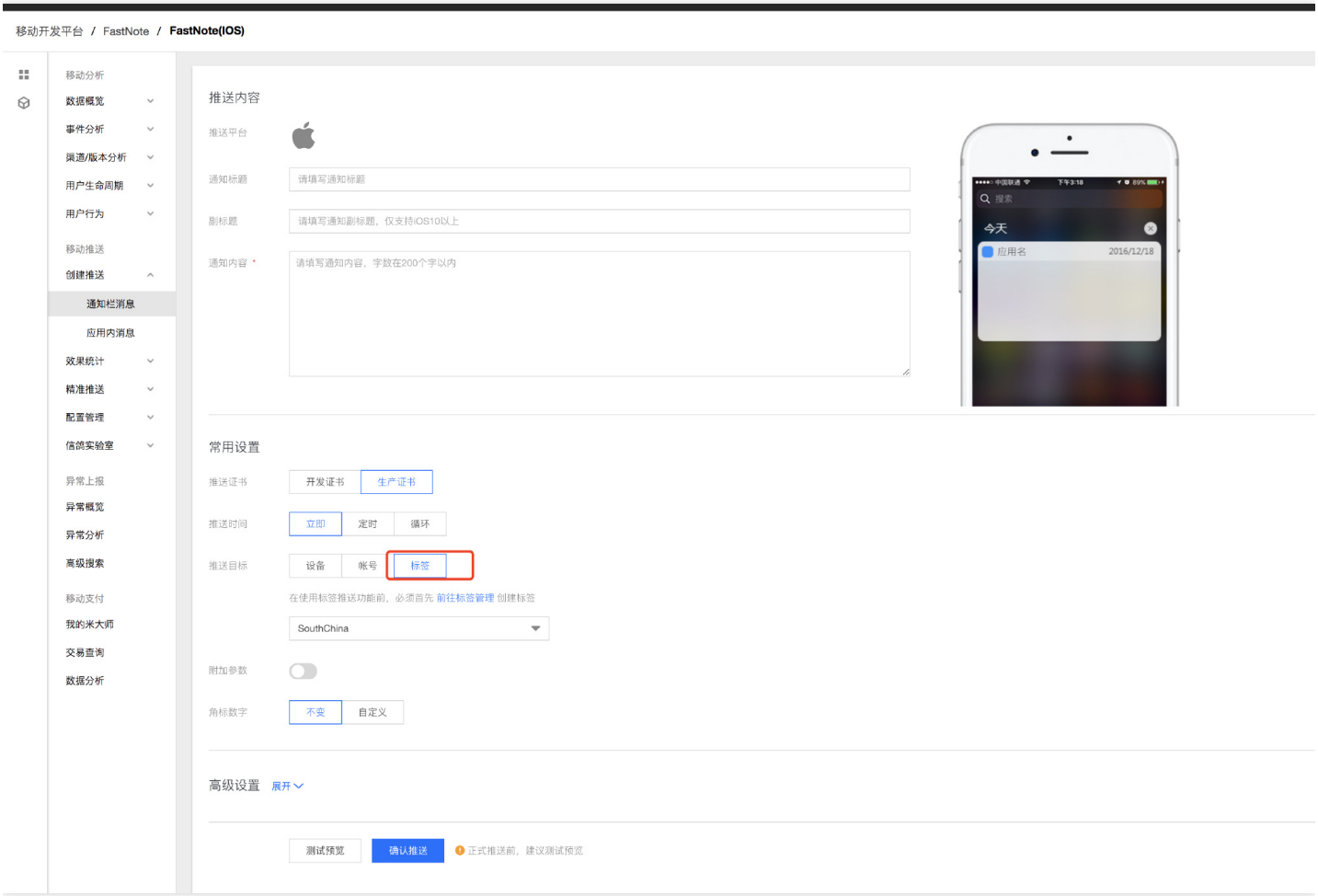


如图这里添加 SouthChina 标签，该标签下包含了广东省、广西壮族自治区、海南省、香港特别行政区和澳门特别行政区。



根据标签发送通知

打开 MobileLine 控制台，选择【创建推送】下的【通知栏消息】，并填写好 通知标题 和 通知内容，然后在推送目标下选择【标签】，并在下拉栏中选择 `SouthChina` 标签，最后点击【确认推送】。



验证通知是否发送成功

推送通知栏消息成功后，若 Messaging SDK 接收到了通知，如果您的程序在前台则会调用 `AppDelegate` 的 `application:didReceiveRemoteNotification` 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到通知栏消息后回调此接口。
- (void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userI
nfo
{
NSLog(@"got messaging");
}
```

注意：选择按标签推送只会推送到绑定了该标签的手机，如 `SouthChina` 标签只会推送给华南用户

根据自定义标签发送通知

添加自定义标签

自定义标签是指您可以根据您自己的业务场景给不同的用户打上不同的标签，如 programer 等。在您的 AppDelegate 中实现方法并调用绑定代码：

```
- (void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userI
nfo
{
[[TACMessagingService defaultService].token bindTag:@"SouthChina"];
}
```

> 请注意请一定要在该函数中调用绑定代码，如果您没有注册成功 APNS 则不会存在变量 [TACMessagingService defaultService].token
>
> 您也可以通过后台接口来绑定标签

根据标签发送通知

控制台上暂时不支持通过自定义标签来发送通知，您可以通过调用我们的后台接口来进行发送，详情请参见 标签推送 API 接口 或者 SDK 接口。

验证通知是否发送成功

推送通知栏消息成功后，若 Messaging SDK 接收到了通知，如果您的程序在前台则会调用 AppDelegate 的 application:didReceiveRemoteNotification 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到通知栏消息后回调此接口。
- (void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userI
nfo
{
NSLog(@"got messaging");
}
```

收到通知后终端将会输出日志：

```
2018-04-20 16:37:06.983857+0800 TACSamples[384:51189] got messaging
```

> 注意：选择按标签推送只会推送到绑定了该标签的手机，如 programer 标签只会推送给您主动打上标签的用户。

解绑自定义标签

您可以通过 SDK 来解绑自定义标签

```
String tag = "programer";
TACMessagingService.getInstance().unbindTag(this, tag);
```

解绑标签成功后，会回调在接入 Messaging 服务时配置的 TACMessagingReceiver 子类的 onUnbindTagResult() 方法，您可以在该方法中调用如下方法打印日志：

Objective-C 代码示例：

```
// 解绑标签成功后回调此接口
[[TACMessagingService defaultService].token unbindTag:@"paragramer"];
```

Swift 代码示例：

```
// 解绑标签成功后回调此接口
TACMessagingService.default().token.unbindTag("paragramer")
```

> 您也可以通过后台接口来解绑标签

根据账户推送消息

最近更新时间：2018-04-24 19:30:03

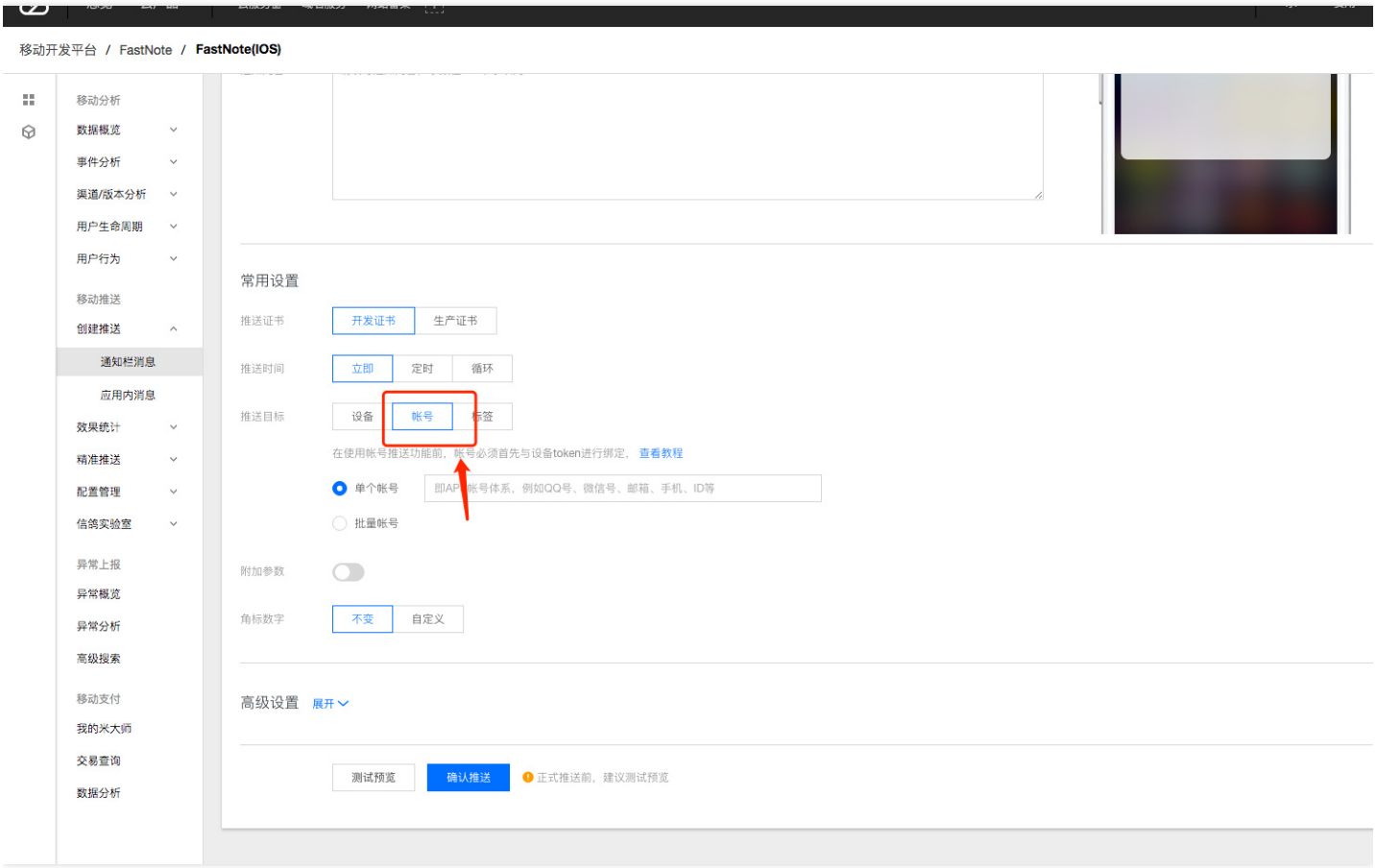
绑定账户

您可以在 iOS 编程手册 中绑定账户。

> 帐号可以是邮箱、QQ号、手机号、用户名等任意类别的业务帐号

通过账户发送通知

打开 MobileLine 控制台，选择【创建推送】下的【通知栏消息】，并填写好 通知标题 和 通知内容，然后在推送目标下选择账户，并填写账户名称。最后点击【确认推送】。



您也可以使用我们的后台接口来推送通知，示例中的账户名称为 ricken

验证通知是否发送成功

验证通知是否发送成功

推送通知栏消息成功后，若 Messaging SDK 接收到了通知，如果您的程序在前台则会调用 `AppDelegate` 的 `application:didReceiveRemoteNotification` 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到通知栏消息后回调此接口。
- (void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userI
nfo
{
NSLog(@"got messaging");
}
```

收到通知后终端将会输出日志：

```
2018-04-20 16:37:06.983857+0800 TACSamples[384:51189] got messaging
```

获取设备推送 token

最近更新时间：2018-04-24 19:30:36

获取设备推送 token

注册成功后，会返回设备 token，token 用于标识设备唯一性，同时也是 Messaging 服务维持与后台连接的唯一身份标识(APNS的token)。在 iOS SDK 中，您可以在接入 Messaging 服务时，通过重写 `AppDelegate application:didReceiveRemoteNotification` 方法来获取 token。

Objective-C 代码示例：

```
- (void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userI
nfo
{
[TACMessagingService defaultService].token ;
}
```

Swift 代码示例：

```
func application(_ application: UIApplication, didReceiveRemoteNotification userInfo: [AnyHashable :
Any]) {
TACMessagingService.default().token;
}
```

请注意请一定要在该函数中调用绑定代码，如果您没有注册成功 APNS 则不会存在变量 `[TACMessagingService defaultService].token`

接入 Messaging 服务请参考 这里

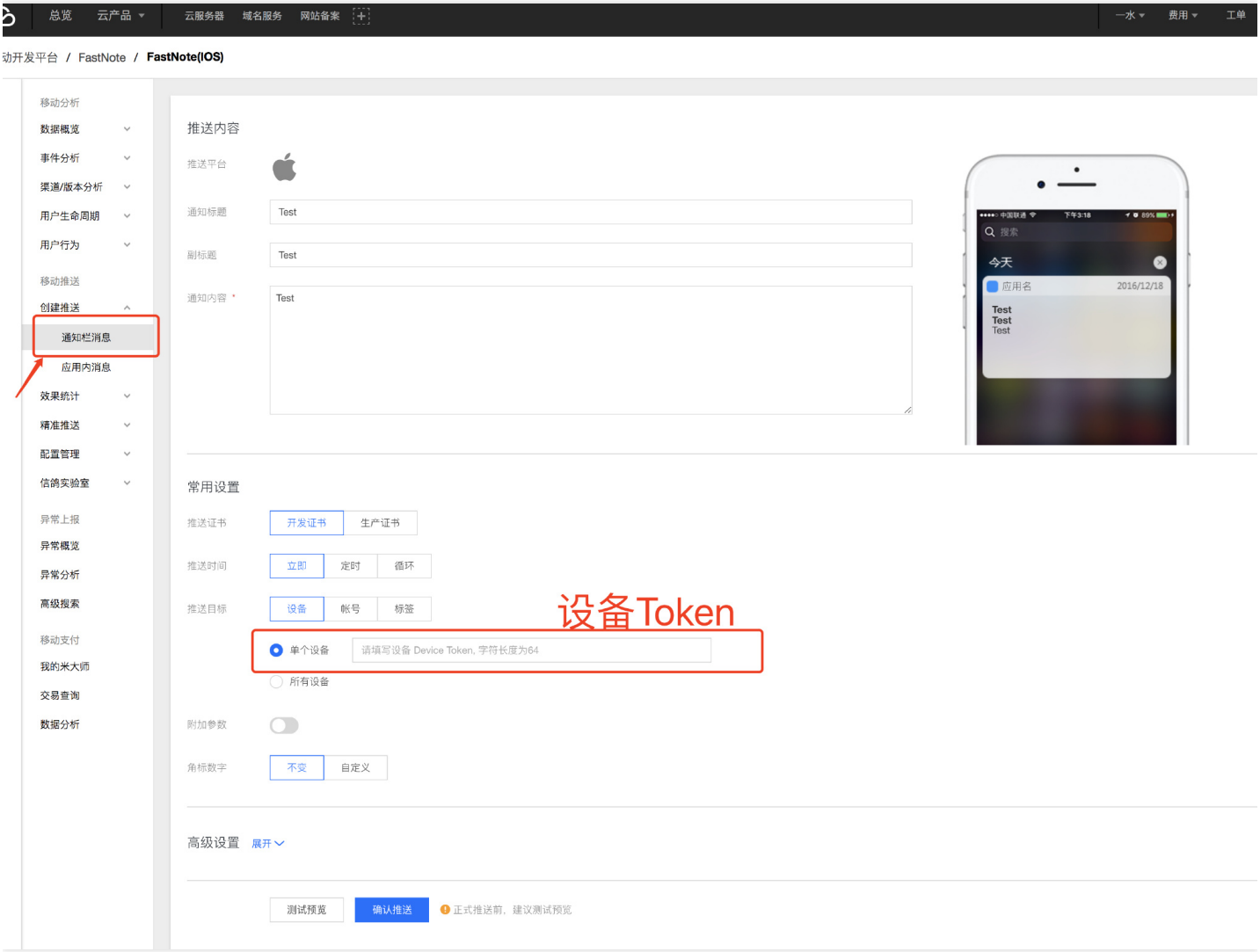
通过控制台推送消息

最近更新时间：2018-04-24 19:31:04

推送通知栏消息

在控制台上推送消息

打开 MobileLine 控制台，选择【创建推送】下的【通知栏消息】，并填写好 **通知标题** 和 **通知内容**，然后选择单选框中的【单个设备】，并将注册成功后回调时打印的设备唯一标识 token 信息拷贝到编辑框中，您也可以在推送时添加自定义参数，然后点击【确认推送】。



验证通知是否发送成功

推送通知栏消息成功后，若 Messaging SDK 接收到了通知，如果您的程序在前台则会调用 `AppDelegate` 的 `application:didReceiveRemoteNotification` 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到通知栏消息后回调此接口。
- (void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userI
nfo
{
NSLog(@"got messaging");
}
```

收到通知后终端将会输出日志：

```
2018-04-20 16:37:06.983857+0800 TACSamples[384:51189] got messaging
```

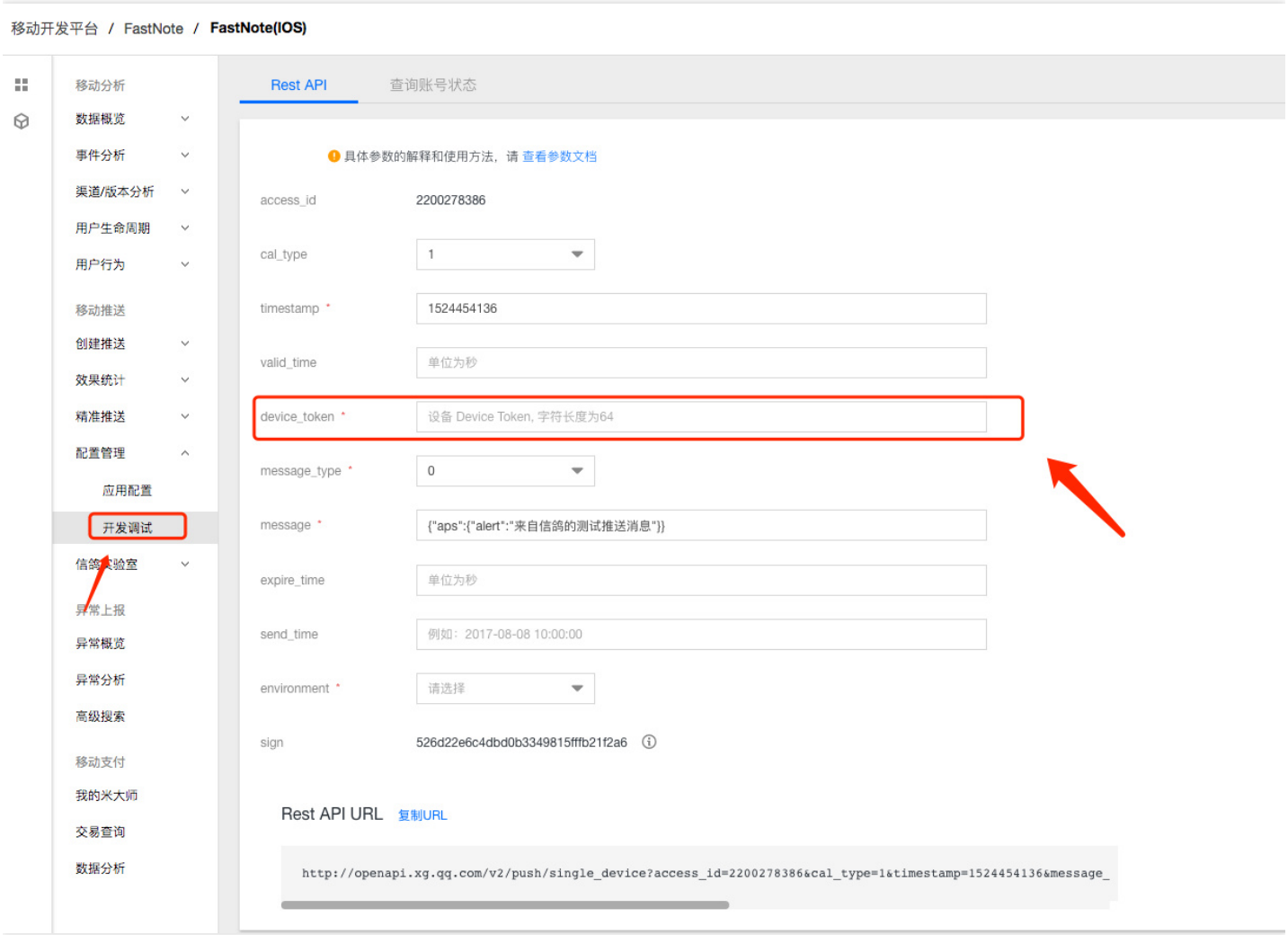
通过后台接口推送消息

最近更新时间：2018-04-24 19:31:25

推送通知栏消息

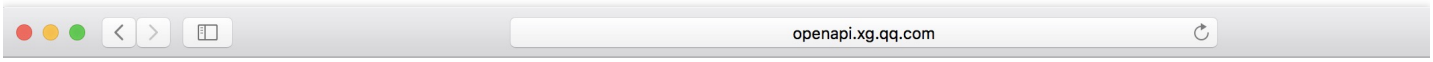
获取推送 URL

打开 MobileLine 控制台，选择【配置管理】下的【开发调试】，选择 cal_type 为 1（表示实时统计），然后将注册成功后回调时打印的设备唯一标识 token 信息拷贝到 device_token 编辑框中，最后选择 message_type 为 1（表示推送通知栏消息），然后点击【复制 URL】。



发起推送请求

将复制好的 URL 黏贴到浏览器中，并通过浏览器发起请求，请求成功后浏览器返回如下：



{"ret_code":0}

验证通知是否发送成功

推送通知栏消息成功后，若 Messaging SDK 接收到了通知，如果您的程序在前台则会调用 AppDelegate 的 application:didReceiveRemoteNotification 方法，您可以在该方法中调用如下方法打印日志：

```
// 收到通知栏消息后回调此接口。
- (void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userI
nfo
{
NSLog(@"got messaging");
}
```

收到通知后终端将会输出日志：

```
2018-04-20 16:37:06.983857+0800 TACSamples[384:51189] got messaging
```

后台接口说明

以上【推送通知栏消息】和【推送透传消息】两部分可用仅仅作为您的调试使用，在生产环境下可以使用我们提供的 API 接口 和 SDK 来推送消息。

iOS10 以上 UserNotifications 框架中的回调

最近更新时间：2018-06-12 20:10:32

在 iOS 10 及以上的系统中，系统使用了新的消息通知框架 `UserNotifications` 。而我们也在此基础上构建我们的功能。我们设置了 `[UNUserNotificationCenter currentNotificationCenter].delegate` 请不要重复设置该参数，避免冲突。如果您对 `UNUserNotificationCenterDelegate` 的回调关心，请实现 `TACMessagingDelegate` 协议，并实现对应的方法：

```
- (void)userNotificationCenter:(UNUserNotificationCenter *)center willPresentNotification:(UNNotification *)notification withCompletionHandler:(void (^)(UNNotificationPresentationOptions options))completionHandler __IOS_AVAILABLE(10.0) __TVOS_AVAILABLE(10.0) __WATCHOS_AVAILABLE(3.0);
- (void)userNotificationCenter:(UNUserNotificationCenter *)center didReceiveNotificationResponse:(UNNotificationResponse *)response withCompletionHandler:(void(^)(void))completionHandler __IOS_AVAILABLE(10.0) __WATCHOS_AVAILABLE(3.0) __TVOS_PROHIBITED;
```

这两个方式是 `UNUserNotificationCenterDelegate` 中对应的方法。

> **注意：**
> 在老版本中我们对该函数进行了重新，重新设计 API 后我们直接使用了系统的函数名称，方便您使用。请不要使用被标记为废弃的 API。

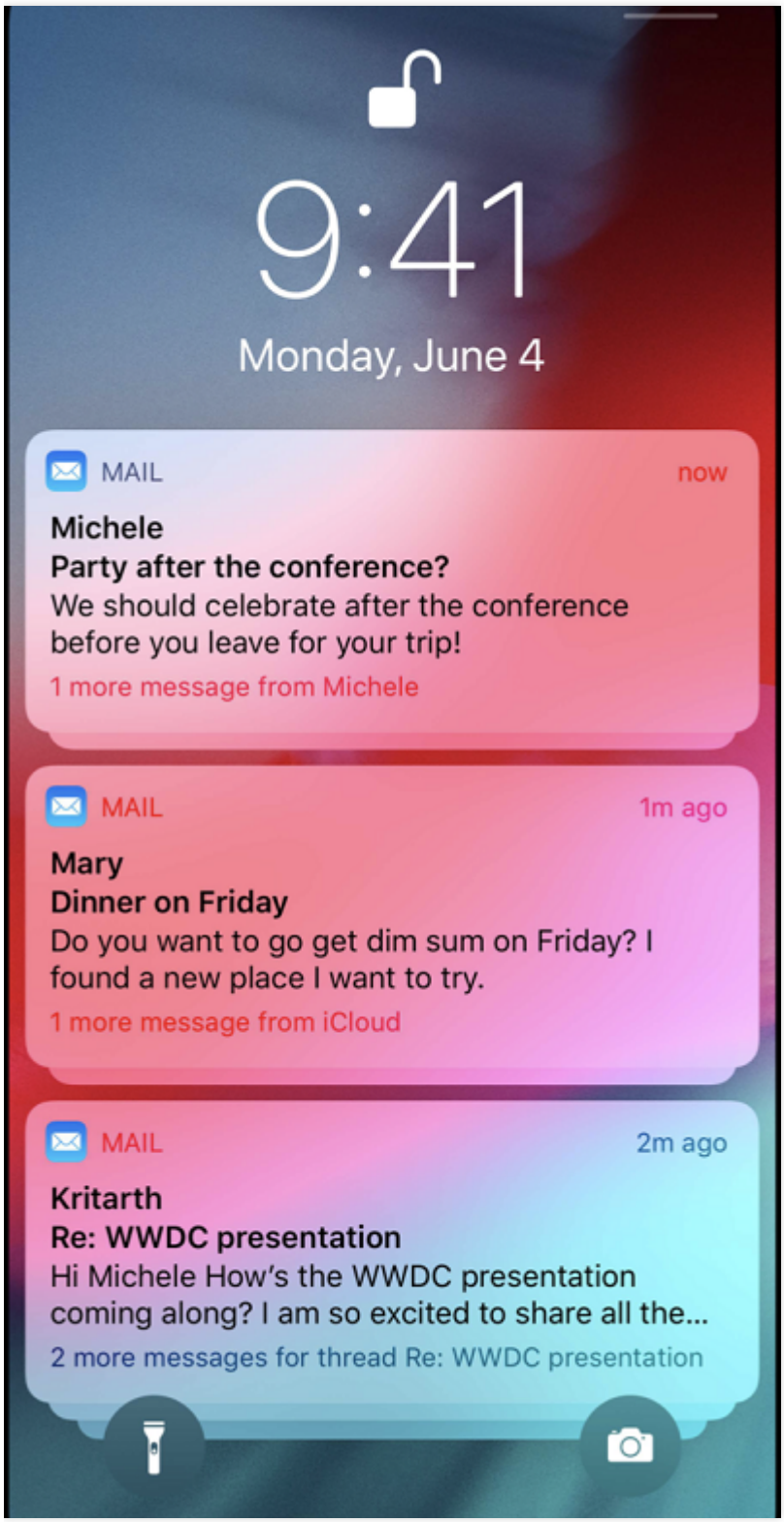
iOS 12 新特性指引

最近更新时间：2019-03-06 21:12:45

通知分组

通知分类是 iOS 12 以后实现的新功能，在推送的 payload 中可以指定一个 key 为 thread-id 的值，iOS 12 系统中，显示推送时将会根据 thread-id 进行分组。

如果不设置 thread-id 的话，那么默认根据 App 来进行分组，通知超过一个屏幕时，将会根据 App 进行折叠。



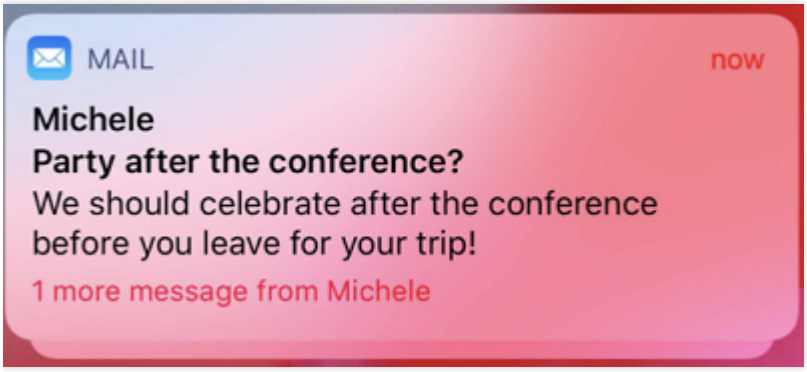
payload 示例：

```
{
"aps": {
```

```
"alert": {
"title": "New Photo",
"body": "Jane Doe posted a new photo"
}
},
"thread-id": "thread-identifier"
}
```

设置分组的简介

可以给每个推送指定一个简介，当它们被聚合到一个通知组里的时候，通知组的下端会显示有来多少个个来自谁的通知。例如像实现上图中的简介效果，那么同时指定 payload 中 summary-arg 这个字段的值为 Michele 即可。



参考示例：

```
{
"aps": {
"alert": {
"title": "Michele",
"body": "We should celebrate after the conference before you leave your trip!",
"summary-arg": "Michele"
}
},
"thread-id": "thread-id-that-you-want"
}
```

静默推送

通常情况下，请求推送需要先获取用户的同意。但如果在请求推送权限时指定了默认推送的选项，那么可以在不获得用户许可的情况下进行静默推送。静默推送没有声音与提示，用户只可以主动在通知中心看到。

请求静默推送示例：

Objective-C：

```
[[UNUserNotificationCenter currentNotificationCenter] requestAuthorizationWithOptions:UNAuthorizationOptionBadge|UNAuthorizationOptionSound|UNAuthorizationOptionSound|UNAuthorizationOptionProvisional completionHandler:^(BOOL granted, NSError * \_Nullable error) {

}];
```

Swift：

```
let notificationCenter = UNUserNotificationCenter.current()

notificationCenter.requestAuthorization(options:[.badge, .sound, .alert, .provisional]) {
}
```

紧急推送

紧急推送可以穿透静音与勿扰模式发出通知，权限相当高。所以如果想发出经济推送，那么需要满足几个条件：

- 在苹果开发者网站上申请对应的 Entitlements，只有几种应用可以申请：医药与健康、家庭与安全、公共安全。
- 用户需要允许 App 发出紧急推送。

满足了上面条件以后，就可以发出紧急推送了，在 payload 中将 critical 的值设置为 1 即可。
参考示例：

```
{
"aps": {
"sound": {
"critical": 1
}
},
"name": "warning-sound.aiff",
"volume": 1.0
}
```

服务端 API 接入
Rest API 使用指南
API 简介

最近更新时间：2018-04-16 18:05:25

请求 URL 结构为： `http://接口域名/v2/class/method?params`

字段名	用途	备注
接口域名	接口域名	统一使用 openapi.xg.qcloud.com
v2	表示当前 API 的版本号	无
class	提供的接口类别	无
method	每个接口大类提供的具体操作接口	如查询、设置、删除等
params	以 GET 方式调用接口时传递的参数	包括通用参数和 API 相关特定参数。所有的参数都必须为 utf 8 编码，params 字符串应进行 url encode

注意：
以 POST 方式调用接口时，参数应以 POST 参数形式传递，内容要求同 params 字段。HTTP HEADER 中“Content-type”字段要设置为“application/x-www-form-urlencoded”。

API 概览

最近更新时间：2018-04-16 19:13:55

推送 API

API 名称	描述
单个设备	给单个设备推送通知
批量设备	给批量设备推送通知
标签	可以针对设置过标签的设备进行推送
单个账号	给单个设备的账户推送通知
批量账号	给批量设备的账户推送通知
高级批量账号	给批量设备的账户推送通知（推送目标帐号数量 ≥10000）

标签管理 API

API 名称	描述
设置标签	批量设置标签信息
删除标签	批量删除标签

查询 API

API 名称	描述
查询消息	查询群发消息发送状态
查询设备数	查询应用覆盖的设备数（token 总数）
查询设备信息	查询应用的某个 token 的信息（查看是否有效）
查询映射设备	查询应用某帐号映射的 token（查看帐号-token 对应关系）

API 名称	描述
查询应用标签	查询应用设置的标签
查询设备标签	查询应用的某个设备上设置的标签
查询标签下设备数	查询应用某个标签下关联的设备数

删除和取消任务 API

API 名称	描述
删除任务	删除群发推送任务的离线消息（针对有任务 ID）
取消任务	取消尚未触发的定时群发任务（针对尚未发送的任务）

通用参数

最近更新时间：2018-11-16 16:41:30

各接口 URL 结构的 params 字段有共同参数：

参数名	类型	必选	参数描述
access_id	uint	是	应用的唯一标识符，在提交应用时管理系统返回。可在 移动开发平台控制台 查看
cal_type	int	否	0-使用离线计算，1-使用实时统计，默认情况下为 0
timestamp	uint	是	本请求的 UNIX 时间戳，用于确认请求的有效期。默认情况下，请求时间戳与服务器时间（北京时间）偏差大于 600 秒则会被拒绝
valid_time	uint	否	配合 timestamp 确定请求的有效期，单位为秒，最大值为 600。若不设置此参数或参数值非法，则按默认值 600 秒计算有效期
sign	string	是	内容签名

内容签名生成规则：

.qcloud

1. 提取请求方法 method（GET 或 POST）；
2. 提取请求 url 信息，包括 Host 字段的 IP 或域名和 URI 的 path 部分，注意不包括 Host 的端口和 Path 的 querystring。请在请求中带上 Host 字段，否则将视为无效请求。比如 `openapi.xg.qcloud.com/v2/push/single_device` 或者 `10.198.18.239/v2/push/single_device` ；
3. 将请求参数（不包括 sign 参数）格式化成 K=V 方式，注意：计算 sign 时所有参数不应进行 urlencode；
4. 将格式化后的参数以 K 的字典序升序排列，拼接在一起，注意字典序中大写字母在前；
5. 拼接请求方法、url、排序后格式化的字符串以及应用的 secret_key；
6. 将 E 形成字符串计算 MD 5 值，形成一个 32 位的十六进制（字母小写）字符串，即为本次请求 sign（签名）的值； Sign=MD5($http_method$url$k1=$v1$k2=$v2$secret_key)，该签名值基本可以保证请求是合法者发送且参数没有被修改，但无法保证不被偷窥。

> 例如： POST 请求到接口 `http://openapi.xg.qcloud.com/v2/push/single_device` ，有四个参数，access_id=123，timestamp=1386691200，Param1=Value1，Param2=Value2，secret_key为abcde。
> 则上述步骤 5 拼接出的字符串为： `POSTopenapi.xg.qcloud.com/v2/push/single_deviceaccess_id=123Param1=Value1Param2=Value2timestamp=1386691200abcde` ，注意字典序中大写在前。

计算出该字符串的 MD 5 为 83c1ed0d65c312ba6e90b0e524753d1c，以此作为 sign 参数的值。

通用返回结果

最近更新时间：2019-03-06 21:00:38

通用返回结果如下：

```
{
"ret_code": 0,
"erroeMsg": "ok",
"result": {
"status": 0
}
}
```

字段定义如下：

参数名	类型	是否必要	参数描述
ret_code	int	是	返回码
err_msg	string	否	请求出错时的错误信息
result	json	否	请求正确时，若有额外数据要返回，则结果封装在该字段的 json 中。若无额外数据，则可能无此字段特别

⚠ 注意：

- 参数和值都是大小写敏感，如没有特别注明，都是小写。
- 所有的 K 和 V 须 urlencode，避免里面有“&”或“=”之类字符影响解析。

通用返回码

最近更新时间：2018-04-16 18:20:38

描述如下：

值	含义	可采取措施
0	调用成功	
-1	参数错误	检查参数配置
-2	请求时间戳不在有效期内	检查设备当前时间
-3	sign 校验无效	检查 Access ID 和 Secret Key（注意不是 Access Key）
2	参数错误	检查参数配置
14	收到非法 token，例如 iOS 终端没能拿到正确的token	Android Token 长度为 40 位 iOS Token 长度为 64 位
15	信鸽逻辑服务器繁忙	稍后重试
19	操作时序错误。例如进行 tag 操作前未获取到 deviceToken	没有获取到 deviceToken 的原因：1. 没有注册信鸽或者苹果推送 2. provisioning profile 制作不正确
20	鉴权错误，可能是由于 Access ID 和 Access Key 不匹配	检查 Access ID 和 Access Key
40	推送的 token 没有在信鸽中注册	检查 token 是否注册
48	推送的账号没有绑定 token	检查 account 和 token 是否有绑定关系见推送指南：绑定/设置账号见热门问题解答：账号和设备未绑定的解答
63	标签系统忙	检查标签是否设置成功见推送指南：设置标签
71	APNS服务器繁忙	苹果服务器繁忙，稍后重试
73	消息字符数超限	iOS 目前是 1000 字节左右，苹果的额外推送设置如角标，也会占用字节数
76	请求过于频繁，请稍后再试	全量广播限频为每 3 秒一次
78	循环任务参数错误	-
100	APNS 证书错误。请重新提交正确的证书	证书格式是 pem 的，另外，注意区分生产证书、开发证书的区别

值	含义	可采取措施
其他	其他错误	-

推送 Android 平台

最近更新时间：2018-04-16 18:19:16

message 参数值应为如下所述的 json 字符串，其总长度不能超过 4096 字节。

推送通知定义示例

推送通知：默认展示在手机或设备通知栏。

```
{"content":"this is content","title":"this is title", "vibrate":1}
```

完整定义

```
{
"title ":"xxx", // 标题，必填
"content ":"xxxxxxxxx", // 内容，必填
"accept_time": //表示消息将在哪些时间段允许推送给用户，选填
[
{
 “start” :{ “hour” :” 13” , “min” :” 00” },
” end” : { “hour” :” 14” , “min” :” 00” }
},
{
 “start” :{ “hour” :” 00” ,” min” :” 00” },
” end” : { “hour” :” 09” , “min” :” 00” }
}
],
"n_id":0, //通知id，选填。若大于0，则会覆盖先前弹出的相同id通知；若为0，展示本条通知且不影响其他通知；若为-1，将清除先前弹出的所有通知，仅展示本条通知。默认为0
"builder_id":0, // 本地通知样式，必填
"ring":1 ， // 是否响铃，0否，1是，下同。选填，默认1
"ring_raw":"ring", // 指定应用内的声音（ring.mp3），选填
"vibrate":1, // 是否振动，选填，默认1
"lights":1// 是否呼吸灯，0否，1是，选填，默认1
"clearable":1, // 通知栏是否可清除，选填，默认1
"icon_type":0 //默认0，通知栏图标是应用内图标还是上传图标,0是应用内图标，1是上传图标,选填
"icon_res":"xg",// 应用内图标文件名（xg.png）或者下载图标的url地址，选填
"style_id":1 //Web端设置是否覆盖编号的通知样式，默认1，0否，1是,选填
"small_icon":"xg"指定状态栏的小图片(xg.png),选填
"action":{ // 动作，选填。默认为打开app
"action_type ": 1, // 动作类型，1打开activity或app本身，2打开浏览器，3打开Intent
```

```
"activity ": "xxx"
"aty_attr ": // activity属性，只针对action_type=1的情况
{
"if":0, // 创建通知时，intent的属性，如：intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED);
"pf":0, // PendingIntent的属性，如：PendingIntent.FLAG_UPDATE_CURRENT
}
"browser": {"url": "xxxx ","confirm": 1}, // url：打开的url，confirm是否需要用户确认
 “intent” : “xxx”
},
"custom_content":{ // 用户自定义的key-value，选填
"key1": "value1",
"key2": "value2"
}
}
```

透传消息定义示例

透传消息：可以由 App 识别的任意透传消息命令，比推送通知更灵活。

```
{"content":"this is content","title":"this is title"}
```

完整定义

```
{
"title":"xxx", // 标题，选填
"content ":"xxxxxxxxx", // 内容，选填
"accept_time": //表示消息将在哪些时间段允许推送给用户，选填
[
{
 “start” :{ “hour” :” 13” , “min” :” 00” },
” end” :{ “hour” :” 14” , “min” :” 00” }
},
{
 “start” :{ “hour” :” 00” ,” min” :” 00” },
” end” :{ “hour” :” 09” , “min” :” 00” }
}
],
"custom_content":{ // 用户自定义的key-value，选填
"key1": "value1",
"key2": "value2"
```

```
}
}
```

推送 iOS 平台

最近更新时间：2018-04-16 18:19:42

message 参数应为 APNS 规定的 payload（也是一个 json 字符串），详细定义参考 APNS 官方手册。信鸽在其基础上仅增添了两保留字段 xg 和 accept_time。payload 不能超过 800 字节。需要注意的是 accept_time 字段不会传递给 APNS，因此不占用 payload 容量。

普通通知示例

```
{
"aps" : { // apns规定的key-value
"alert" : { //设置消息通知栏的字段
"title": "this is a title", //通知标题
"body" : "Bob wants to play poker", //通知内容
},
"badge" : 5,
 “category” ： “INVITE_CATEGORY”，
},
"accept_time":[ //允许推送给用户的时段，选填。accept_time不会占用payload容量
{
"start":{"hour":"13","min":"00"},
"end": {"hour":"14","min":"00"}
},
{
"start":{"hour":"00","min":"00"},
"end": {"hour":"09","min":"00"}
}
] // 仅0~9点和13~14点这两个时段可推送
"custom1" : "bar", // 合法的自定义key-value，会传递给app
"custom2" : [ "bang", "whiz" ], // 合法的自定义key-value，会传递给app
"xg" : "oops" // 错误！xg为信鸽系统保留key，其value会被信鸽系统覆盖，应避免使用
}
```

静默通知示例

```
{
"aps" : { // apns规定的key-value
"badge" : 5,
```

```
"category" : “INVITE_CATEGORY”,
"content-available": 1, //静默通知的标识
},
"custom1" : "bar", // 合法的自定义key-value，会传递给app
"custom2" : [ "bang", "whiz" ], // 合法的自定义key-value，会传递给app
"xg" : "oops" // 错误！xg为信鸽系统保留key，其value会被信鸽系统覆盖，应避免使用
}
```

删除全部账号映射

最近更新时间：2018-05-04 19:38:42

删除应用中某 account 映射的所有 token
URL 路径： http://接口域名/v2/application/del_app_account_all_tokens?params

请求参数.qcloud

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
account	string	是	无	账号

响应结果

在通用返回结果参数中，result 字段的 json 为空。

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）

MD5 加密前：

```
GETopenapi.xg.qcloud.com/v2/application/del_app_account_all_tokensaccess_id=2100240957account=easonshipushtestaccounttimestamp=1502701471f255184d160bad51b88c31627bbd9530
```

Rest Api Url:

```
http://openapi.xg.qcloud.com/v2/application/del_app_account_all_tokens?access_id=2100240957&account=easonshipushtestaccount&timestamp=1502701471&sign=88fbcc8b5c29a3f5ae1dab99b7479439
```

删除单个账号映射

最近更新时间：2019-03-06 20:44:03

删除应用中某个 account 映射的某个 token。
URL 路径： http://接口域名/v2/application/del_app_account_tokens?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
account	string	是	无	账号，可以是邮箱号、手机号、QQ 号等任意形式的业务帐号
device_token	string	是	无	token，设备的唯一识别 ID

响应结果

在通用返回结果参数中，result 字段的 json 如下（即显示删除 device_token 后该 account 映射的剩余 token）：

```
{
"tokens":["token1","token2"]
}
```

示例.qcloud

MD5加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）

MD5 加密前：

```
GETopenapi.xg.qcloud.com/v2/application/del_app_account_tokensaccess_id=2100240957account=easonshipushtestaccountdevice_token=76501cd0277cdcef4d8499784a819d4772e0fddetimestamp=1502361905f255184d160bad51b88c31627bbd9530
```

Rest Api Url:

```
http://openapi.xg.qcloud.com/v2/application/del_app_account_tokens?access_id=2100240957&account=easonshipushtestaccount&device_token=76501cd0277cdcef4d8499784a819d4772e0fdde&timestamp=1502361905&sign=c8c86feab7a1d8b1a3064c733a76079a
```

全量设备

最近更新时间：2019-03-06 20:42:57

给全量设备推送通知。
后台对本接口的调用频率有限制，两次调用之间的时间间隔不能小于 3 秒。

URL 路径： http://接口域名/v2/push/all_device?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
message_type	uint	是	无	消息类型：1. 通知 2. 透传消息。iOS 平台请填 0
message	string	是		详细参阅 推送 Android 平台、推送 iOS 平台
expire_time	uint	否	3 天	消息离线存储时间（单位为秒），最长存储时间 3 天。若设置为 0，则使用默认值（3 天）
send_time	string	否	立即	指定推送时间,格式为 year-mon-day hour:min:sec 若小于服务器当前时间，则会立即推送
multi_pkg	uint	否	0	0表示按注册时提供的包名分发消息；1 表示按 access id 分发消息，所有以该 access id 成功注册推送的 App 均可收到消息。本字段对 iOS 平台无效
environment	uint	仅 iOS 必需		向 iOS 设备推送时必填，1 表示推送生产环境；2 表示推送开发环境。推送 Android 平台不填或填0
loop_times	uint	否		循环任务执行的次数，取值[1, 15]
loop_interval	uint	否		循环任务的执行间隔，以天为单位，取值[1, 14]。loop_times 和 loop_interval 一起表示任务的生命周期，不可超过 14 天

响应结果

在通用返回结果参数中，result 字段的 json 示例如下：

```
{
"push_id":"string" //(表示给app下发的任务id，如果是循环任务，返回的是循环父任务id)
}
```

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）

MD5 加密前：

GETopenapi.xg.qcloud.**com**/v2/push/all_deviceaccess_id=2100240957message={"title":"测试消息","content":"来自restapi的全量接口测试消息"}message_type=1timestamp=1502360486f255184d160bad51b88c31627bbd9530

Rest Api Url:

http://openapi.xg.qcloud.**com**/v2/push/all_device?access_id=2100240957&message={"title":"测试消息","content":"来自restapi的全量接口测试消息"}&message_type=1&timestamp=1502360486&**sign**=4813a111880885223b72229495508813

推送相关接口
单个设备

最近更新时间：2018-04-16 18:08:08

给单个设备推送通知。
URL 路径： http://接口域名/v2/push/single_device?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
device_token	string	是	无	针对某一设备推送，token 是设备的唯一识别 ID
message_type	uint	是	无	消息类型：1. 通知 2. 透传消息。iOS 平台请填 0
message	string	是	无	详细参阅 推送Android平台、推送 iOS 平台
expire_time	uint	否	3 天	消息离线存储时间（单位为秒），最长存储时间 3 天。若设置为 0，则使用默认值（3 天）
send_time	string	否	立即	指定推送时间，格式为 year-mon-day hour:min:sec， 若小于服务器当前时间，则会立即推送
multi_pkg	uint	否	0	0表示按注册时提供的包名分发消息；1 表示按 access id 分发消息，所有以该 access id 成功注册推送的 App 均可收到消息。本字段对 iOS 平台无效
environment	uint	仅 iOS 必需	无	向 iOS 设备推送时必填，1 表示推送生产环境；2 表示推送开发环境。推送 Android 平台不填或填 0

响应结果

在通用返回结果参数中，result 字段的 json 为空。
本接口不返回 push id

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）

MD5 加密前：

GETopenapi.xg.qcloud.com/v2/**push**/single_deviceaccess_id=2100250470device_token=76501cd0277cdcef4d8499784a819d4772e0fddemessage={"title":"测试消息","content":"来自restapi的单推接口测试消息"}message_type=1timestamp=1502356505f1fa8b11f540794bf13e10d499ac5c36

Rest Api Url

http://openapi.xg.qcloud.com/v2/**push**/single_device?access_id=2100250470&timestamp=1502356505&device_token=76501cd0277cdcef4d8499784a819d4772e0fdde&message_type=1&message={"title":"测试消息","content":"来自restapi的单推接口测试消息"}&sign=b7f5761d37fb352536e53db0c50ffcc6

批量设备

最近更新时间：2019-03-06 20:47:09

给批量设备推送通知。

- 首先，需要创建批量消息（获取 push_id）。
- 其次，按照已创建的批量推送消息推送给多个设备。

创建批量消息

URL 路径：`http://接口域名/v2/push/create_multipush?params`

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
message_type	uint	是	无	消息类型：1. 通知 2. 透传消息
message	string	是	无	详细参阅 推送 Android 平台、推送 iOS 平台
expire_time	uint	否	3 天	消息离线存储时间（单位为秒），最长存储时间 3 天。若设置为 0，则不存储；iOS 无需设置此参数
multi_pkg	uint	否	无	0 表示按注册时提供的包名分发消息；1 表示按 access id 分发消息，所有以该 access id 成功注册推送的 App 均可收到消息
environment	uint	仅 iOS 必需	无	向 iOS 设备推送时必填，1 表示推送生产环境；2 表示推送开发环境。推送 Android 平台不填或填 0

响应结果

在通用返回结果参数中，result 字段的 json 如下：

```
{
"push_id":"string" //(表示给app下发的任务id)
}
```

推送批量消息

URL 路径： http://接口域名/v2/push/device_list_multiple?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
device_list	string	是	无	Json 数组格式，每个元素是一个 token，string 类型，单次发送 token 不超过 1000 个。例：[“token 1”,”token 2”,”token 3”]
push_id	uint	是	无	创建批量推送消息 接口的返回值中的 push_id

响应结果

在通用返回结果参数中，result 字段的 json 为空。

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）

- **获取 push_id：**

  MD5 加密前：

  GETopenapi.xg.qcloud.**com**/v2/push/create_multipushaccess_id=2100264266message={"title":"测试消息","content":"来自restapi的批量接口测试消息","custom_content":{"key1":"value1","key2":"value2"}}message_type=2timestamp=1502694940d8fc29c627259a06452794e31dab5bb8

  Rest Api Url:

  http://openapi.xg.qcloud.**com**/v2/push/create_multipush?access_id=2100264266&message={"title":"测试消息","content":"来自restapi的批量接口测试消息","custom_content":{"key1":"value1","key2":"value2"}}&message_type=2&timestamp=1502694940&**sign**=e5ca158c01712fb185399e67b6a57d1f

- **进行推送：**

  MD5 加密前：

  GETopenapi.xg.qcloud.com/v2/push/device_list_multipleaccess_id=2100264266device_list=["78e8540853619eb14fb49fdd53274c0c82ca2025"]push_id=2854657652timestamp=1502694940d8fc29c627259a06452794e31dab5bb8

Rest Api Url:

http://openapi.xg.qcloud.**com**/v2/push/device_list_multiple?access_id=2100264266&device_list=["78e8540853619eb14fb49fdd53274c0c82ca2025"]&push_id=2854657652&timestamp=1502694940&**sign**=e4779a9173a1c51541800e76b8a25322

标签

最近更新时间：2019-03-06 20:53:22

可以针对设置过标签的设备进行推送。如：女、大学生、低消费等任意类型标签。
URL 路径： http://接口域名/v2/push/tags_device?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
message	string	是	无	详细参阅 推送 Android 平台、推送 iOS 平台
message_type	uint	是	1	消息类型：1. 通知 2. 透传消息。iOS 平台请填0
tags_list	json	是	无	[“tag 1”,”tag 2”,”tag 3”]
tags_op	string	是	无	取值为 AND 或 OR
expire_time	uint	否	3 天	消息离线存储时间（单位为秒），最长存储时间 3 天。若设置为 0，则使用默认值（3 天）
send_time	string	否	立即	指定推送时间，格式为 year-mon-day hour:min:sec 若小于服务器当前时间，则会立即推送
multi_pkg	uint	否	0	0表示按注册时提供的包名分发消息；1 表示按 access id 分发消息，所有以该 access id 成功注册推送的 App 均可收到消息。本字段对 iOS 平台无效
environment	uint	仅 iOS 必需	无	向 iOS 设备推送时必填，1 表示推送生产环境；2 表示推送开发环境。推送 Android 平台不填或填 0
loop_times	uint	否	无	循环任务执行的次数，取值[1, 15]
loop_interval	uint	否	无	循环任务的执行间隔，以天为单位，取值[1, 14]。loop_times 和 loop_interval 一起表示任务的生命周期，不可超过 14 天

响应结果

在通用返回结果参数中，result 字段的 json 示例如下：

```
{
"push_id":"string" //(表示给app下发的任务id，如果是循环任务，返回的是循环父任务id)
}
```

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）

MD5 加密前：

```
GETopenapi.xg.qcloud.com/v2/push/tags_deviceaccess_id=2100240957message={"title":"测试消息","content":"来自restapi的标签接口测试消息"}message_type=1tags_list=["qwertyuiop"]tags_op=ORtimestamp=1502360486f255184d160bad51b88c31627bbd9530
```

Rest Api Url：

```
http://openapi.xg.qcloud.com/v2/push/tags_device?access_id=2100240957&message={"title":"测试消息","content":"来自restapi的标签接口测试消息"}&message_type=1&timestamp=1502360486&tags_list=["qwertyuiop"]&tags_op=OR&sign=95dbc4d1107a99d6824fda19e7ff09c9
```

单个账号

最近更新时间：2018-04-16 18:08:52

设备的账户或别名由终端 SDK 在调用推送注册接口时设置，详情参考终端 SDK 文档。

URL 路径： `http://接口域名/v2/push/single_account?params`

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
account	string	是	无	针对某一账号推送，帐号可以是 qq 号，邮箱号，openid，手机号等各种类型
message_type	uint	是	1	消息类型：1. 通知 2. 透传消息
message	string	是	无	详细参阅 推送Android平台、推送 iOS 平台
expire_time	uint	否	3 天	消息离线存储时间（单位为秒），最长存储时间 3 天。若设置为 0，则使用默认值（3 天）
send_time	string	否	立即	指定推送时间，格式为 year-mon-day hour:min:sec 若小于服务器当前时间，则会立即推送
multi_pkg	uint	否	0	0表示按注册时提供的包名分发消息；1 表示按 access id 分发消息，所有以该 access id 成功注册推送的 App 均可收到消息
environment	uint	仅 iOS 必需	无	向 iOS 设备推送时必填，1 表示推送生产环境；2 表示推送开发环境。推送 Android 平台不填或填 0

响应结果

在通用返回结果参数中，result 字段的 json 为空。本接口不返回 push id。

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）

MD5 加密前：

GETopenapi.xg.qcloud.com/v2/**push**/single_accountaccess_id=2100240957account=easonshipushtest accountmessage={"title":"测试消息","content":"来自restapi的单个账号接口测试消息"}message_type=1timestamp=1502361241f255184d160bad51b88c31627bbd9530

Rest Api Url：

http://openapi.xg.qcloud.com/v2/**push**/single_account?access_id=2100240957&account=easonshipushtestaccount&message={"title":"测试消息","content":"来自restapi的单个账号接口测试消息"}&message_type=1&timestamp=1502361241&sign=0d7bee840e87801e8a90b831ee87eefb

批量账号

最近更新时间：2018-04-16 18:11:07

设备的账户或别名由终端 SDK 在调用推送注册接口时设置，详情参考终端 SDK 文档。
账号数量≥10000 时建议使用 高级批量账号 接口进行推送。
URL 路径： http://接口域名/v2/push/account_list?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
account_list	string	是	无	Json 数组格式，每个元素是一个 account，string 类型，单次发送 account 不超过 100 个。例：[“account 1”,”account 2”,”account 3”]
message_type	uint	是	无	消息类型：1. 通知 2. 透传消息
message	string	是	无	详细参阅 推送 Android 平台、推送 iOS 平台
expire_time	uint	否	3 天	消息离线存储时间（单位为秒），最长存储时间 3 天。若设置为 0，则使用默认值（3 天）
multi_pkg	uint	否	0	0 表示按注册时提供的包名分发消息；1 表示按 access id 分发消息，所有以该 access id 成功注册推送的 App 均可收到消息
environment	uint	仅 iOS 必需	无	向 iOS 设备推送时必填，1 表示推送生产环境；2 表示推送开发环境。推送 Android 平台不填或填0

响应结果

在通用返回结果参数中，result 字段的 json 为每个 account 发送返回码。本接口不返回 push id 。

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）

MD5 加密前：

GETopenapi.xg.qcloud.com/v2/**push**/account_listaccess_id=2100240957account_list=["easonshipushtestaccount"]message={"title":"测试消息","content":"来自restapi的批量账号接口测试消息"}message_type

=1timestamp=1502361241f255184d160bad51b88c31627bbd9530

Rest Api Url：

http://openapi.xg.qcloud.com/v2/**push**/account_list?access_id=2100240957&account_list=["easonshipushtestaccount"]&message={"title":"测试消息","content":"来自restapi的批量账号接口测试消息"}&message_type=1&timestamp=1502361241&sign=cff802738763385db89aaadb49dbe345

高级批量账号

最近更新时间：2018-05-04 19:42:57

如果推送目标帐号数量很大（比如≥10000），推荐使用 account_list_multiple 接口。

- 首先，（如同推送批量设备）需要创建批量消息。
- 其次，选择推送批量帐号。

创建批量消息

URL 路径： http://接口域名/v2/push/create_multipush?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
message_type	uint	是	无	消息类型：1. 通知 2. 透传消息
message	string	是	无	详细参阅 推送 Android 平台、推送 iOS 平台
expire_time	uint	否	无	消息离线存储多久，单位为秒，最长存储时间 3 天。在超时时间内，可以发起此消息的批量推送
multi_pkg	uint	否	无	0 表示按注册时提供的包名分发消息；1 表示按 access id 分发消息，所有以该 access id 成功注册推送的 App 均可收到消息
environment	uint	仅 iOS 必需	无	向 iOS 设备推送时必填，1 表示推送生产环境；2 表示推送开发环境。推送 Android 平台不填或填 0

选择推送批量帐号

URL 路径： http://接口域名/v2/push/account_list_multiple?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述

参数名	类型	必需	默认值	描述
account_list	string	是	无	Json 数组格式，每个元素是一个 account，string 类型，单次发送 account 不超过 1000 个。例：[“account 1”,”account 2”,”account 3”]
push_id	uint	是	无	创建批量推送消息，接口的返回值中的 push_id

响应结果

在通用返回结果参数中，result 字段的 json 为空。

示例

请参考 批量设备示例。

标签管理相关
批量设置标签

最近更新时间：2018-04-16 18:18:49

批量设置标签信息。
URL 路径： http://接口域名/v2/tags/batch_set

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
tag_token_list	string	是	无	json 字符串，包含若干标签-token 对，后台将把每一对里面的 token 打上对应的标签。每次调用最多允许设置 20 对，每个对里面标签在前，token 在后。注意标签最长 50 字节，不可包含空格；真实 token 长度至少 40 字节。示例（其中 token 值仅为示意）： [[”tag 1”,”token 1”],[”tag 2”,”token 2”]]

响应结果

在通用返回结果参数中，result 字段的 json 为空。

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）

MD5 加密前：

```
GETopenapi.xg.qcloud.com/v2/tags/batch_setaccess_id=2100240957tag_token_list=[["easonshitag1","76501cd0277cdcef4d8499784a819d4772e0fdde"]]timestamp=1502361905f255184d160bad51b88c31627bbd9530
```

Rest Api Url:

```
http://openapi.xg.qcloud.com/v2/tags/batch_set?access_id=2100240957&tag_token_list=[["easonshitag1","76501cd0277cdcef4d8499784a819d4772e0fdde"]]&timestamp=1502361905&sign=3c0ea17401f02fed8397eef9230fb607
```

批量删除标签

最近更新时间：2018-04-16 18:18:22

批量删除标签。
URL 路径： http://接口域名/v2/tags/batch_del

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
tag_token_list	string	是	无	json 字符串，包含若干标签-token 对，后台将为每一对里面的 token 删除对应的标签。每次调用最多允许设置 20 对，每个对里面标签在前，token 在后。注意标签最长 50 字节，不可包含空格；真实 token 长度至少 40 字节。示例如下（其中 token 值仅为示意）：[[“tag 1”,”token 1”],[“tag 2”，“token 2”]]

响应结果

在通用返回结果参数中，result 字段的 json 为空。

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）

MD5 加密前：

```
GETopenapi.xg.qcloud.com/v2/tags/batch_delaccess_id=2100240957tag_token_list=[["easonshitag1","76501cd0277cdcef4d8499784a819d4772e0fdde"]]timestamp=1502361905f255184d160bad51b88c31627bbd9530
```

Rest Api Url:

```
http://openapi.xg.qcloud.com/v2/tags/batch_del?access_id=2100240957&tag_token_list=[["easonshitag1","76501cd0277cdcef4d8499784a819d4772e0fdde"]]&timestamp=1502361905&sign=301fd2e83a7f65223e1d9e38fb0b5864
```

查询相关接口
查询群发状态

最近更新时间：2019-03-06 21:08:36

查询群发消息发送状态。
URL 路径： http://接口域名/v2/push/get_msg_status?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
push_id	json	是		[{“push_id”: string}, {“push_id”:“xxxx”},]

响应结果

在通用返回结果参数中，result 字段的 json 形式为：

```
{
"list": [{
"push_id": "27ABC5486977"
"status": 0, //0（ 未处理 ），1（ 推送中 ），2（ 推送完成 ），3（ 推送失败 ）
"start_time": "year-mon-day hour:min:sec"
"finished": "xxxx", //（已发送 ）
"total": "xxxxx" //（共需要发送 ）
}]
}
```

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）

MD5 加密前：

```
GETopenapi.xg.qcloud.com/v2/push/get_msg_statusaccess_id=2100240957push_id=2841253998timestamp=1502698593f255184d160bad51b88c31627bbd9530
```

Rest Api Url:

http://openapi.xg.qcloud.**com**/v2/push/get_msg_status?access_id=2100240957&push_id=2841253998&timestamp=1502698593&**sign**=39b62ab54f08e7844ed1d86e00cec76a

查询应用设备数

最近更新时间：2019-03-06 20:30:32

查询应用覆盖的设备数（token 总数）。
URL 路径： `http://接口域名/v2/application/get_app_device_num?params`

请求参数

本接口仅包括 通用参数 。

响应结果

在通用返回结果参数中，result 字段的 json 形式为（若请求应用列表中某个应用信息非法，则不会在 result 中返回结果。）：

```
{
"device_num": 34567 //(设备数)
}
```

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）

MD5 加密前：

```
GETopenapi.xg.qcloud.com/v2/application/get_app_device_numaccess_id=2100240957timestamp=1502701471f255184d160bad51b88c31627bbd9530
```

Rest Api Url:

```
http://openapi.xg.qcloud.com/v2/application/get_app_device_num?access_id=2100240957&timestamp=1502701471&sign=e4385f856ddaa932170c181927965cb1
```

查询应用 token

最近更新时间：2019-03-06 20:55:15

查询应用的某个 token 的信息（查看是否有效）
URL 路径： http://接口域名/v2/application/get_app_token_info?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
device_token	string	是	无	无

响应结果

在通用返回结果参数中，result 字段的 json 如下：

```
{
"isReg":1, //（1为token已注册，0为未注册）
"connTimestamp":1426493097, //（最新活跃时间戳）
"msgsNum":2 //（该应用的离线消息数）
}
```

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）。

MD5 加密前：

```
GETopenapi.xg.qcloud.com/v2/application/get_app_token_infoaccess_id=2100240957device_token=76501cd0277cdcef4d8499784a819d4772e0fddetimestamp=1502698593f255184d160bad51b88c31627bbd9530
```

Rest Api Url:

```
http://openapi.xg.qcloud.com/v2/application/get_app_token_info?access_id=2100240957&device_token=76501cd0277cdcef4d8499784a819d4772e0fdde&timestamp=1502698593&sign=c4f650c6c468adba2e2b82a15ca68c3e
```

查询映射 token

最近更新时间：2019-03-06 20:32:01

查询应用某帐号映射的 token（查看帐号-token 对应关系）
URL 路径： http://接口域名/v2/application/get_app_account_tokens?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
account	string	是	无	帐号

响应结果

在通用返回结果参数中，result 字段的 json 如下：

```
{
"tokens":["token1","token2"]
}
```

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）。

MD5 加密前：

GETopenapi.xg.qcloud.**com**/v2/application/get_app_account_tokensaccess_id=2100240957account=easonshipushtestaccounttimestamp=1502699212f255184d160bad51b88c31627bbd9530

Rest Api Url:

http://openapi.xg.qcloud.com/v2/application/get_app_account_tokens?access_id=2100240957&account=easonshipushtestaccount&timestamp=1502699212&sign=015ef9e7fde208f2d12674f731e13e8c .qcloud

查询应用标签

最近更新时间：2019-03-06 20:29:00

查询应用设置的标签

URL 路径： http://接口域名/v2/tags/query_app_tags?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
start	uint	否	0	开始值
limit	uint	否	100	限制数量

响应结果

在通用返回结果参数中，result 字段的 json 格式如下：

```
{
"total": 2, //应用的tag总数，注意不是本次查询返回的tag数
"tags": ["tag1", "tag2"]
}
```

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）。

MD5 加密前：

```
GETopenapi.xg.qcloud.com/v2/tags/query_app_tagsaccess_id=2100240957timestamp=1502699212f255184d160bad51b88c31627bbd9530
```

Rest Api Url:

```
http://openapi.xg.qcloud.com/v2/tags/query_app_tags?access_id=2100240957&timestamp=1502699212&sign=5dbf914884378af6b62cba919e012b34
```

查询设备标签

最近更新时间：2018-04-16 18:17:53

查询应用的某个设备上设置的标签
URL 路径： http://接口域名/v2/tags/query_token_tags?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
device_token	string	是	无	无

响应结果

在通用返回结果参数中，result 字段的 json 格式如下：

```
{
 “tags” :[ “tag1” ,” tag2” ]
}
```

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）。

MD5 加密前：

```
GETopenapi.xg.qcloud.com/v2/tags/query_token_tagsaccess_id=2100240957device_token=76501cd0277cdcef4d8499784a819d4772e0fddetimestamp=1502699212f255184d160bad51b88c31627bbd9530
```

Rest Api Url:

```
http://openapi.xg.qcloud.com/v2/tags/query_token_tags?access_id=2100240957&device_token=76501cd0277cdcef4d8499784a819d4772e0fdde&timestamp=1502699212&sign=4cbd1b7a5553ed263f47a4a0b96402e9
```

查询标签下设备数

最近更新时间：2019-03-06 20:57:10

查询应用某个标签下关联的设备数
URL 路径： http://接口域名/v2/tags/ query_tag_token_num?params

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
tag	string	是	无	无

响应结果

在通用返回结果参数中，result 字段的 json 格式如下：

```
{
"device_num":589874
}
```

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）。

MD5 加密前：

```
GETopenapi.xg.qcloud.com/v2/tags/query_tag_token_numaccess_id=2100240957tag=easonmipushtesttimestamp=1502699920f255184d160bad51b88c31627bbd9530
```

Rest Api Url:

```
http://openapi.xg.qcloud.com/v2/tags/query_tag_token_num?access_id=2100240957&tag=easonmipushtest&timestamp=1502699920&sign=0ea7f16df1b59d69c9b81b385f938822
```

删除或取消任务接口

最近更新时间：2018-04-16 18:06:05

删除任务

删除群发推送任务的离线消息，针对有任务 ID（push ID），并且已发送任务可以删除离线消息。
该功能仅支持在 移动开发平台管理台 中使用，登录管理台，在任务管理中单击【停止】按钮即可。

取消任务

取消尚未触发的定时群发任务，针对尚未发送的任务，需要任务 ID。
URL 路径：`http://接口域名/v2/push/cancel_timing_task?params`

请求参数

除了 通用参数 外，还包括如下参数：

参数名	类型	必需	默认值	描述
push_id	string	是	无	要取消的任务 ID

响应结果

在通用返回结果参数中，result 字段的 json 格式如下：

```
{
  “status”：0, //0为成功，其余为失败
}
```

示例

MD5 加密前 url 用作生成 sign，Rest Api Url 为最终请求的 url（以下为 Android 推送示例，需替换通用参数后使用）。

MD5 加密前：

```
GETopenapi.xg.qcloud.com/v2/push/cancel_timing_taskaccess_id=2100240957push_id=2853333945timestamp=1502700856f255184d160bad51b88c31627bbd9530
```

Rest Api Url:

http://openapi.xg.qcloud.com/v2/**push**/cancel_timing_task?access_id=2100240957&push_id=2853333945&timestamp=1502700856&sign=1fb3b7846f79d0027542acd05effb4a3

服务端其他语言

最近更新时间：2018-06-20 16:54:46

服务端其它语言 SDK 及文档官方下载

- 服务端 Java
- 服务端 PHP
- 服务端 Python
- 服务端 C#

感谢第三方开发者（非官方）提供的其它语言版本

- 服务端 Node js

Rest API V3

最近更新时间：2018-08-27 17:44:47

Rest API V3 概述

为了进一步优化用户体验，我们目前发布了推送 Rest API V3，和 Rest API V2 相比，两者的区别如下：

特点	Rest API V3	Rest API V2
完备性	目前只支持 Push API，其他的 API，如标签 API 和查询 API 等接口目前正在开发中	支持全部接口
HTTP 协议	只支持 HTTPS	只支持 HTTP
HTTP Method	只支持 POST	支持 POST 和 GET
参数位置	Body 中	URL 中
鉴权方式	Authorization 头部	Url 的 sign 参数

Rest API V3 接口主要分为四大类：

API类型	描述	状态
Push API	包含多种消息推送的接口	正常
标签 API	主要完成标签的新增、删除、查询	实现中
账号 API	主要完成账号的查询、删除	实现中
工具类 API	用来定位问题和其他数据查询	实现中

本文档将只描述接口状态为正常的 API，其他状态的 API，请使用 v2 版本

申请使用 API V3

API V3 已经开放内测，您可以 **申请使用**。

请求方式

- 仅支持 HTTPs
- 仅支持 POST

权鉴方式

- 采用基础鉴权的方式，HTTP Header（头）里加一个字段（ Key/Value 对）：

  Authorization: Basic base64_auth_string

- 其中 base64_auth_string 的生成算法为： base64(APPID:SECRETKEY)

  即对 APPID 加上冒号，加上 SECRETKEY 拼装起来的字符串，再做 base64 转换

- APPID 和 SECRET KEY 可在 应用云控制台 下的移动推送的【配置管理】【应用配置】中获取。

协议描述

请求URL： https://openapi.xg.qq.com/v3/<class_path>/<method>

字段名	用途	备注
openapi.xg.qq.com	接口域名	无
v3	版本号	无
class_path	提供的接口类别	不同的接口有不同的路径名
method	功能接口名称	不同的功能接口有不同的名称

API限制

- 除去全量推送、标签推送接口有调用频率的限制外，其他均无限制
- 推送的消息体大小限制为 4K，此限制适用于 Push API 中的 message 字段

通用基础返回值

- 通用基础返回值，是所有请求的响应中都会包含的字段，格式如下：

```
{
"seq": 0,
"push_id": 123,
"ret_code": 0,
"environment": "product",
"err_msg": "",
"result": {
"": ""
}
}
```

- 返回值字段描述表：

参数名	类型	必需	参数描述
seq	int64	是	与请求包一致（如果请求包是非法json 该字段为0）
push_id	string	是	推送id
ret_code	int	是	错误码
environment	string	是	用户指定推送环境，仅支持iOS product： 生产环境 dev： 开发环境
err_msg	string	否	结果描述
result	JSON	否	请求正确时且有额外数据，则结果封装在该字段中

Push API

Push API 概述

- Push API 是所有推送接口的统称

- Push API 有多种推送目标，推送目标如下：

- 全量推送
- 标签推送
- 单设备推送
- 设备列表推送
- 单账号推送
- 账号列表推送

- 所有推送目标都采用相同的 URL 发起请求，URL： `https://openapi.xg.qq.com/v3/push/app`
- 所有请求参数通过 JSON 封装上传给后台，后台通过请求参数区分不同的推送目标

Push API 必要参数

推送必要参数指一条推送消息必需携带的参数

参数名	类型	是否必需	描述
audience_type	string	是	推送目标 all：全量推送 tag：标签推送 token：单设备推送 token_list ：设备列表推送 account：单账号推送 account_list：账号列表推送
platform	string	是	客户端平台类型 android：安卓 ios：苹果 all：安卓&&苹果，仅支持全量推送和标签推送
message	object	是	消息体，参见消息体格式
message_type	string	是	消息类型 notify：通知 message：透传消息/静默消息

audience_type：推送目标

推送目标，表示一条推送可以被推送到哪些设备

Push API 提供了多种推送目标的，比如：全量、标签、单设备、设备列表、单账号、账号列表

推送目标	描述	必需参数及使用说明
all	全量推送	无
tag	标签推送	`tag_list` 1. 推送 tag1 和 tag2 的设备 {“tags”:[“tag1”,”tag2”],”op”:”AND”} 2. 推送 tag1 或 tag2 的设备 {“tags”:[“tag1”,”tag2”],”op”:”OR”}
token	单设备推送	`token_list` 1. 如果该参数包含多个token 只会推送第一个token 2. 格式eg：[“token1”]

token_list	设备列表群推	`token_list`
		1. 最多1000 个token
		2. 格式eg：[“token1”,”token2”]
		`push_id`
		1. 第一次推送该值填0，系统会创建对应的推送任务，并且返回对应的pushid：123
		2. 后续推送push_id 填123(同一个文案）表示使用与123 id 对应的文案进行推送

account	单账号推送	`account_list` 1. 该参数有多个账号时，仅推送第一个账号 2. 格式eg：[“account1”]

account_list	账号列表群推	`account_list`
		1. 最多1000 个account
		2. 格式eg：[“account1”,”account2”]
		`push_id`
		1. 第一次推送该值填0，系统会创建对应的推送任务，并且返回对应的pushid：123
		2. 后续推送push_id 填123(同一个文案）表示使用与123 id 对应的文案进行推送

- 全量推送：推送给全部设备

```
{
"audience_type": "all"
}
```

- 标签推送：推送给同时设置了"tag1"和"tag2"标签的设备

```
{
"audience_type": "tag",
"tag_list": {
"tags": [
"tag1",
"tag2"
],
"op": "AND"
}
}
```

- 单设备推送：推送给token为"token1"的设备

```
{
"audience_type": "token",
"token_list": [
"token1"
]
}
```

- 设备列表推送，推送给token为"token1"和"token2"的设备

```
{
"audience_type": "token_list",
"token_list": [
"token1",
"token2"
],
"push_id": "0"
}
```

- 单账号推送：推送给账号为"account1"的设备

```
{
"audience_type": "account",
"account_list": [
"account1"
]
}
```

- 账号列表推送：推送账号为"account1"和"account2"的设备

```
{
"audience_type": "account_list",
"account_list": [
"account1",
"account2"
],
"push_id": "0"
}
```

platform：推送平台

当前支持 Android、iOS 两个平台的推送

其关键字分别为："android", "ios"，如果要同时推送两个平台，则关键字为："all"

- 推送到所有平台：

```
{
"platform": "all"
}
```

- 推送到安卓平台：

```
{
"platform": "android"
}
```

- 推送到iOS平台：

```
{
"platform": "ios"
}
```

message_type：消息体类型

针对不同平台，消息类型稍有不同，具体参照下表：

消息类型	描述	支持平台	特性说明
notify	通知栏消息	Android iOS	通知栏展示消息
message	透传消息/静默消息	Android(透传消息) iOS(静默消息)	通知栏不展示消息

message：消息体

消息体，即下发到客户端的消息

Push API 对 iOS 和 Android 两个平台的消息有不同处理，需要分开来实现对应平台的推送消息，推送的消息体是 JSON 格式

Android普通消息

Android平台具体字段如下表：

字段名	类型	默认值	必需	参数描述
title	string	无	是	消息标题
content	string	无	是	消息内容
accept_time	array	无	否	消息将在哪些时间段允许推送给用户，建议小于10个
n_id	int	0	否	通知消息对象的唯一标识 1. 大于0，会覆盖先前相同id的消息； 2. 等于0，展示本条通知且不影响其他消息； 3. 等于-1，将清除先前所有消息，仅展示本条消息
builder_id	int	无	是	本地通知样式标识
ring	int	1	否	是否有铃声 0：没有铃声 1：有铃声
ring_raw	string	无	否	指定Android工程里raw目录中的铃声文件名，不需要后缀名
vibrate	int	1	否	是否使用震动 0：没有震动 1：有震动

字段名	类型	默认值	必需	参数描述
lights	int	1	否	是否使用呼吸灯 0：使用呼吸灯 1：不使用呼吸灯
clearable	int	1	否	通知栏是否可清除
icon_type	int	0	否	通知栏图标是应用内图标还是上传图标 0：应用内图标 1：上传图标
icon_res	string	无	否	应用内图标文件名或者下载图标的url地址
style_id	int	1	否	设置是否覆盖指定编号的通知样式
small_icon	string	无	否	消息在状态栏显示的图标，若不设置，则显示应用图标
action	JSON	有	否	设置点击通知栏之后的行为，默认为打开app
custom_content	JSON	无	否	用户自定义的键值对

完整的消息示例如下：

```
{
"title": "xxx",
"content": "xxxxxxxxx",
"accept_time": [
{
"start": {
"hour": "13",
"min": "00"
},
"end": {
"hour": "14",
"min": "00"
}
},
{
"start": {
"hour": "00",
"min": "00"
},
"end": {
"hour": "09",
```

```
"min": "00"
}
}
],
"android": {
"n_id": 0,
"builder_id": 0,
"ring": 1,
"ring_raw": "ring",
"vibrate": 1,
"lights": 1,
"clearable": 1,
"icon_type": 0,
"icon_res": "xg",
"style_id": 1,
"small_icon": "xg",
"action": {
"action_type ": 1,
"activity": "xxx",
"aty_attr": {
"if": 0,
"pf": 0
},
"browser": {
"url": "xxxx ",
"confirm": 1
},
"intent": "xxx"
},
"custom_content": {
"key1": "value1",
"key2": "value2"
}
}
}
```


iOS普通消息

iOS平台具体字段如下表：

字段名	类型	默认值	必需	参数描述

字段名	类型	默认值	必需	参数描述
aps	JSON	无	是	苹果推送服务(APNs)特有的消息体字段 其中比较重要的键值对: alert：包含标题和消息内容(必选) badge：App显示的角标数(可选), category：下拉消息时显示的操作标识(可选) 详细介绍可以参照：Payload
custom	string/JSON	无	否	自定义下发的参数
xg	string	无	否	系统保留key，应避免使用

完整的消息示例如下：

```
{
"title": "xxx",
"content": "xxxxxxxxx",
"ios":{
"aps": {
"alert": {
"subtitle": "my subtitle"
},
"badge": 5,
"category": "INVITE_CATEGORY"
},
"custom1": "bar",
"custom2": [
"bang",
"whiz"
],
"xg": "oops"
}
}
```

Android透传消息

透传消息，Android平台特有，即不显示在手机通知栏中的消息，可以用来实现让用户无感知的向App下发带有控制性质的消息

Android平台具体字段如下表：

字段名	类型	默认值	是否必需	参数描述
title	string	无	是	消息标题

字段名	类型	默认值	是否必需	参数描述
content	string	无	是	消息内容
custom_content	JSON	无	否	自定义内容
accept_time	array	无	否	消息将在哪些时间段允许推送给用户，建议小于10个

具体完整示例：

```
{
"title": "this is title",
"content": "this is content",
"android": {
"custom_content": {
"key1": "value1",
"key2": "value2"
}
},
"accept_time": [
{
"start": {
"hour": "13",
"min": "00"
},
"end": {
"hour": "14",
"min": "00"
}
},
{
"start": {
"hour": "00",
"min": "00"
},
"end": {
"hour": "09",
"min": "00"
}
}
]
}
```

iOS静默消息

静默消息，iOS平台特有，类似Android中的透传消息，消息不展示，当静默消息到达终端时，iOS会在后台唤醒App一段时间(小于30s)，让App来处理消息逻辑

具体字段如下表：

字段名	类型	默认值	是否必要	参数描述
aps	JSON	无	是	苹果推送服务(APNs)特有的， 其中最重要的键值对: content-available：标识消息类型(必须为1) 且不能包含alert、sound、badge字段 详细介绍可以参照：Payload
custom	string/JSON	无	否	自定义下发的参数
xg	string	无	否	系统保留key，应避免使用

具体完整示例：

```
{
"ios":{
"aps": {
"content-available": 1
},
"custom": {
"key1": "value1",
"key2": "value2"
},
"xg": "oops"
}
}
```

Push API 可选参数

Push API 可选参数是除了 `audience_type` 、 `platform` 、 `message_type` 、 `message` 以外，可选的高级参数

参数名	类型	必需	默认值	描述
expire_time	int	否	259200	消息离线存储时间（单位为秒） 最长存储时间3天，若设置为0，则默认值（3天） 建议取值区间[600, 86400x3] 第三方通道离线保存消息不同厂商标准不同

参数名	类型	必需	默认值	描述
send_time	string	否	当前系统时间	指定推送时间 格式为yyyy-MM-DD HH:MM:SS 若小于服务器当前时间，则会立即推送 仅全量推送和标签推送支持此字段
multi_pkg	bool	否	false	多包名推送 当app存在多个不同渠道包（例如应用宝、豌豆荚等），推送时如果是希望手机上安装任何一个渠道的app都能收到消息那么该值需要设置为true
loop_times	int	否	0	循环任务重复次数 仅支持全推、标签推 建议取值[1, 15]
loop_interval	int	否	0	循环执行消息下发的间隔 必须配合loop_times使用 以天为单位，取值[1, 14] loop_times和loop_interval一起表示消息下发任务的循环规则，不可超过14天
environment	string	否	product	用户指定推送环境，仅限iOS平台推送使用 product： 推送生产环境 dev： 推送开发环境
stat_tag	string	否	无	统计标签，用于聚合统计 使用场景(示例)： 现在有一个活动id：active_picture_123,需要给10000个设备通过单推接口（或者列表推送等推送形式）下发消息，同时设置该字段为active_picture_123 推送完成之后可以使用v3统计查询接口，根据该标签active_picture_123 查询这10000个设备的实发、抵达、展示、点击数据
seq	int64_t	否	0	接口调用时，在应答包中信鸽会回射该字段，可用于异步请求 使用场景：异步服务中可以通过该字段找到server端返回的对应应答包
tag_list	object	仅标签推送必需	无	1. 推送 tag1 和 tag2 的设备： {“tags”:[“tag1”,”tag2”],”op”:”AND”} 2. 推送 tag1 或 tag2 的设备： {“tags”:[“tag1”,”tag2”],”op”:”OR”}

参数名	类型	必需	默认值	描述
account_list	array	单账号推送、账号列表推送时必需	无	若单账号推送 1. 要求 audience_type=account 2. 参数格式：["account1"] 若账号列表推送 1. 参数格式：["account1","account2"] 2. 最多1000 个account
account_type	int	单账号推送时可选	0	1. 账号类型，参考后面账号说明。 2. 必须与账号绑定时设定的账号类型一致
token_list	array	单设备推送、设备列表推送时必需	无	若单设备推送 1. 要求 audience_type=token 2. 参数格式：["token1"] 若设备列表推送 1. 参数格式：["token1","token2"] 2. 最多 1000 个 token
push_id	string	账号列表推送、设备列表推送时必需	无	账号列表推送和设备列表推送时，第一次推送该值填0，系统会创建对应的推送任务，并且返回对应的pushid：123，后续推送push_id 填123(同一个文案）表示使用与123 id 对应的文案进行推送。(注：文案的有效时间由前面的expire_time 字段决定）

Push API 请求完整示例

标签推送请求消息

```
POST /v3/push/app HTTP/1.1
Host: openapi.xg.qq.com
Content-Type: application/json
Authorization: Basic YTViNWYwNzFmZjc3YTplYTUxMmViNzcwNGQ1ZmI1YTZhOTM3Y2FmYTcwZTc3MQ==
Cache-Control: no-cache
```

```
Postman-Token: 4b82a159-afdd-4f5c-b459-de978d845d2f
{
"platform": "android",
"audience_type": "tag",
"tag_list": {
"tags": [
"tag1",
"tag2"
],
"op": "AND"
},
"message_type": "notify",
"message": {
"title": "this is title",
"content": "this is content",
"custom_content": {
"key1": "value1",
"key2": "value2"
},
"accept_time": [
{
"start": {
"hour": "13",
"min": "00"
},
"end": {
"hour": "14",
"min": "00"
}
}
]
}
}
```

标签推送应答消息

```
{
"seq": 0,
"environment": "product",
"ret_code": 0,
"push_id": "3895624686"
}
```

账号类型

账号类型是客户端调用SDK接口绑定，类型如下表所示：

账号类型	含义
0	未知
1	手机号
2	邮箱
1000	微信openid
1001	qq openid
1002	新浪微博
1003	支付宝
1004	淘宝
1005	豆瓣
1006	facebook
1007	twitter
1008	google
1009	百度
1010	京东
1011	linkin
1999	其他
2000	游客登录
2001以上	用户自定义

错误码

推送 REST API 接口较多，开发者使用过程中不可避难会遇到各种问题，这里提供了常见的错误码释义，对应着是通用基础返回值中的ret_code字段的可能值

错误码	含义
10100	系统繁忙请稍后重试!
10101	系统繁忙请稍后重试!
10102	缺少参数请检查后重试
10103	参数值非法，请检查后重试
10104	鉴权未通过，请检查secret key!
10105	证书无效!
10106	当前推送类型不支持多平台推送!
10107	消息体是非法json 格式
10108	内部错误,请稍后重试!
10109	内部错误,请稍后重试!
10110	设备未注册!
10111	内部错误,请稍后重试!
10112	内部错误,请稍后重试!
10113	内部错误,请稍后重试!
10114	内部错误,请稍后重试!
10115	帐号不能为空,帐号为空!
10116	帐号不存在
10117	推送内容太大
10201	创建推送任务失败,请稍后重试!
10202	推送消息内容转换APNs 失败!
10203	创建推送任务失败，请稍后重试!
10204	推送失败，请稍后重试!
10205	推送任务过期，请检查！
10206	获取消息副本失败，请稍后重试！

错误码	含义
10207	获取消息副本失败，请稍后重试！
10301	帐号列表推送失败，请稍后重试!
10302	帐号列表推送部分失败！
10303	帐号列表推送全部失败,请稍后重试!
10304	token 列表推送部分失败！
10305	token 列表推送全部失败,请稍后重试!
10401	内部错误，请稍后重试!
10402	内部错误，请稍后重试!
10403	内部错误，请稍后重试!
10404	内部错误，请稍后重试!
10405	内部错误，请稍后重试!
10406	内部错误，请稍后重试!
10407	内部错误，请稍后重试!
10501	内部错误，请稍后重试!
10502	内部错误，请稍后重试!
10503	内部错误，请稍后重试!
10504	内部错误，请稍后重试!
10505	内部错误，请稍后重试!
10506	内部错误，请稍后重试!
10507	内部错误，请稍后重试!
10601	内部错误，请稍后重试!
10602	内部错误，请稍后重试!
10603	内部错误，请稍后重试!
10604	内部错误，请稍后重试!

错误码	含义
10605	内部错误，请稍后重试!
10606	app 未注册，请注册后重试!
10701	内部错误,请稍后重试!
10702	内部错误，请稍后重试!
10707	内部错误，请稍后重试!
10708	内部错误，请稍后重试!
10709	内部错误，请稍后重试!
10710	内部错误，请稍后重试!
10711	内部错误，请稍后重试!
10712	内部错误，请稍后重试!
10713	内部错误，请稍后重试!
其他	未知错误，请稍后重试!

开发者手册
名词解释

最近更新时间：2018-04-26 09:28:03

通知栏消息

指的是会在设备的通知栏展示的消息，由 SDK 完成所有的操作，APP 可以监听通知被打开的行为，也就是说在前台下发的通知不需要 APP 做任何处理，默认会展示在通知栏。通常来说，结合自定义通知样式，常规的通知能够满足大部分业务需求，如果需要更灵活的方式请考虑使用透传消息（应用内消息）。

透传消息（应用内消息）

指的是由推送服务下发给 APP 的内容，需要 APP 注册状态通知回调接口，并自主处理所有操作过程，也就是说，下发的消息默认是不会展示在通知栏的，Messaging 服务只负责将消息从下发到 APP 这个过程，不负责消息的处理逻辑，需要APP自己实现。

透传消息具有灵活性强和高度定制性特点，因此更适合APP自主处理个性化业务需求，比如下发APP配置信息、自定义处理消息的存储和展示等。

例如：某游戏需要针对不同情景（用户升级提示、版本更新提示、活动营销提示等）提供不同的通知，可以把这些情景以json格式封装在消息，下发到APP，然后APP根据这些场景提供不同的提示，满足个性化需求。

我的标签

标签：通常是指给某个或某一群用户打上标签。自定义标签可以通过客户端或服务端调用进行设置，之后在前台进行使用。

高级数据标签：是指移动开发平台（MobileLine）通过数据整理，直接为特定用户打上标签。高级标签仅供使用，无法编辑或删除。

应用配置

应用包名：出于安全考虑，应用包名填写后不可更改；填写应用包名才能进行推送操作；应用包名是应用的 Package Name，用于AndroidManifest.xml配置，比如com.tencent.news。

测试设备：测试设备用于推送消息之前先进行特定设备推送，以测试实际情况，确定无误之后再正式推送。测试设备的ID就是DeviceToken，通过logcat获取，具体请查阅开发者手册。

ACCESS KEY：客户端鉴权密钥，与Access ID共同验证以确定调用合法；需要配置到客户端SDK中，无法更换。

SECRET KEY：用于验证API调用，与APPKEY共同验证以确定调用合法。如果泄露，需要立即更换并重新配置客户端SDK。

ACCESS ID：识别一个应用的唯一标识，不能更改，需要配置到客户端SDK中，调用后台接口时也需要提供。

创建推送

推送内容：消息命令是应用接收后去执行的代码，具体代码形式由应用开发者自己定义。利用消息命令，可以远程控制应用各种行为，比如：应用下载并更换启动闪屏；修改移动游戏中物品的价格；静默更新应用内文字或者图片内容。

立即/定时推送：立即推送多适用于测试推送。定时推送多用于正式对外推送。

自定义参数：您还可以自定义设置键值对(key-value)，根据不同的键值对实现自定义需求。

离线保存：用户如果不在线，下次上线时可以收到推送，这里需要设定一个离线保存过期的时间，超过这个时间如果用户仍不在线，则不会收到这条推送。如果没有限时活动，强烈建议保存72小时！！！

时段控制：设定用户可以接收推送的时段，您可以避免在夜间打扰用户，也可以设定在特定时间用户才能收到推送。

点击通知操作：设定用户点击通知之后的响应动作，可以直接打开应用，也可以指定打开应用的某个功能页面，还可以使用浏览器打开一个网址。

多包名推送：在多包名提示模式下，设备上所有使用这个access_id注册推送的app都会收到消息。该功能使用与区分不同渠道和包名的app。

推送列表

推送列表：展示的数据是全量/批量推送（属于广播类），点对点（单播类）推送不会展示在推送列表内。查看单播类推送，请前往"推送数据"页面。

推送时间：开始本条推送的时间。

Android-有效推送量：是指发送给当前在线设备（与服务器有连接的设备）的推送量。若消息设置了离线保存，随着时间的推移和用户的上线动作，有效推送量会有数值上的增加，即表示新上线的用户也收到了该条推送。

iOS-有效推送量：是指成功发送给苹果服务器的推送量。移动开发平台（MobileLine）负责将消息成功推送给苹果服务器，但由于苹果服务器限制，抵达量无法统计。

抵达量：本条推送成功推送的用户数，实时数据有5分钟左右延迟。

撤销定时：撤销已经设定的，尚未发生的定时推送，撤消后该条推送将不再执行。

删除离线：是指删除后台存储的，即将发送的离线消息。成功删除离线消息后，这部分消息将不再发送。但是！！已经发送成功的消息不能进行删除。

推送数据

Android-有效推送量：是指发送给当前在线设备（与服务器有连接的设备）的推送量。若消息设置了离线保存，随着时间的推移和用户的上线动作，有效推送量会有数值上的增加，即表示新上线的用户也收到了该条推送。

iOS-有效推送量：是指成功发送给苹果服务器的推送量。移动开发平台（MobileLine）负责将消息成功推送给苹果服务器，但由于苹果服务器限制，抵达量无法统计。

抵达量：本条推送成功推送的用户数，实时数据有5分钟左右延迟。

抵达率：抵达量/推送量×100%

点击量：成功的推送中，本条推送消息被点击次数的总计，点击量要求终端调用特定函数上报。

点击率：点击量/抵达量×100%

基础数据

日新增设备数：当天新注册的设备总数

日卸载设备数：当天卸载应用的设备总数

日连接设备数：当天连接过移动开发平台（MobileLine）服务器的设备总数

推送失败原因

最近更新时间：2018-08-27 17:33:40

1.1. 设备注册失败

新创建的app会有一分钟左右的数据同步过程，在此期间注册可能返回20错误码，稍后重试即可。其他情况返回20错误码，请检查配置文件中 Messaging 模块对应的 APPID 和 APPKEY 是否正确配置（大小写敏感并且不呢那个有空格）。

1.1.1. 帐号和设备未绑定

对于希望根据帐号进行推送的用户，首先需要将账号与token进行绑定，否则将无法推送成功。
注意：

Android Token长度为40位

iOS Token长度为64位

账号，又称别名，指带有账号登录功能的APP的用户账号，这里不仅仅是QQ或微信，只要是用户的账号都支持，比如手机QQ的账号就是QQ号码，gmail的账号就是邮箱，中国移动的账号就是手机号码。

Android绑定账号在注册时绑定，即：registerPush(context,account)接口，iOS通过setAccount设置。

选择帐号推送时，提示token not found,check registration，说明账号没和token关联上，这种情况有两种可能：

(1)账号或别名注销了，不一定是app调用，某些情况下可能会自动触发注销的

(2)该设备注册了别的账号或别名，这样会自动与原来的解绑。（一个设备只能对应一个别名如果当前别名下没有设备了，就not found了）

绑定账号后，可以通过指定别名（账号）下发通知。通常情况下，这个账号最近登录过的设备都可以收到通知。用户账号退出时，调用registerPush(context,"*")解除当前账号的绑定。

账号（别名）不允许单字符，一个token只能绑定一个账号，多次绑定时，以最后一次为准。

注意：一个帐号（即别名，account）下面最多可以绑定15台设备，当绑定满后，最新绑定的设备会随机顶掉之前绑定的一台设备token。

1.1.2. Service被终止

service被终止后，由系统、安全软件和用户操作限定是否能够再次启动。

SDK通过唯一的service与移动开发平台（MobileLine） Messaging 后台保持通讯，在android中，service被杀死后在没有被系统/安全软件禁止的条件下是能够自启动的，具体可自行网上搜索“android service onstartcommand START_STICKY”

目前，在某些定制的系统（如MIUI）或被安全软件禁止自启动后，只有用户再次打开APP才能重启移动开发平台（MobileLine）service

移动开发平台（MobileLine）service何时能够启动由系统调度确定

在锁屏触屏、网络切换、安装APP、系统重启等条件，移动开发平台（MobileLine）会主动尝试启动service

1.1.3. iOS证书问题

iOS推送提示failed to load certificate,check your APNS certificate，对应环境的apns证书没提交。 证书做得不对，请参照官方指南进行制作。

推送环境是否选择正确，测试预览请选择开发环境。

推送环境是否选择正确，开发证书对应测试环境，生产证书对应正式环境。

证书有效期判断

查看连接APNS测试证书是否合法

开发环境

生产环境

1.1.4. 消息延时

在Android平台下，推送服务和大多数互联网服务一样，受限于包括网络服务质量、运营商政策或用户需求差异等因素的影响，所以有可能出现延时。另外，如果终端上移动开发平台（MobileLine）service被系统或者安全软件杀死，也会有收不到或延时情况。

在iOS平台下，移动开发平台（MobileLine）负责将消息成功转发给APNS，但APNS是否会成功下发给用户，具体下发多少，折损多少，都受限于包括网络服务质量、运营商政策等因素的影响。

注意：APNS只为离线终端保存一条消息，所以离线终端上线后仅能收到离线期间最新的一条消息。

1.1.5. 排查方法

(1)设备是否正常联网？

(2)配置文件中 APPID、APPKey设置是否与前台注册的一致？

(3)当前APP包名是否与前台注册的一致？如果不一致，请在前台选中“使用多包名”选项。

(4)设备是否注册成功？

(5)前台下发通知时，“时段控制”选项里的时间段是否符合终端设备当前时间？

(6)请检查xml的receiver和service标签是否匹配，强烈建议请直接复制demo的例子再修改。

(7)请检查是否没有加“android.permission.GET_TASKS”权限的问题

1.1.6. 推送暂停

（1）相同的内容在推送给所有设备的时候，一小时内不能重复创建。

（2）每小时最多创建30条全量推送。

返回码一览

最近更新时间：2018-03-29 16:32:31

服务端返回码

值	含义	可采取措施
0	调用成功	
-1	参数错误	检查参数配置
-2	请求时间戳不在有效期内	检查设备当前时间
-3	recv 失败	稍后重试
-5	Action 处理超时	稍后重试
2	非法参数	检查参数配置
5	与 CMEM 通讯失败	稍后重试（推送超时）
6	设备 token 未成功注册	请检查终端设备注册是否成功
7	通用错误，账号超限	删除其他未使用的账号(调用账号解绑）
14	token 非法	Android Token 长度为 40 位，iOS Token 长度为 64 位
15	推送逻辑服务器繁忙	稍后重试
16	系统繁忙	稍后重试
19	操作时序错误。例如进行 tag 操作前未获取到 deviceToken	没有获取到 deviceToken 的原因：1.没有注册移动开发平台（MobileLine）或者苹果推送2.provisioning profile 制作不正确
20	鉴权错误，可能是由于 Access ID 和 Access Key 不匹配	检查 Access ID 和 Access Key（注意空格）
21	鉴权失败	检查 Access ID 和 Access Key
40	推送的 token 没有在信鸽中注册	检查 token 是否注册
48	推送的账号没有绑定 token	检查 account 和 token 是否有绑定关系见推送指南：绑定/设置账号见热门问题解答：账号和设备未绑定的解答

值	含义	可采取措施
53	设备未注册	反注册后重新注册
73	消息字符数超限	iOS 目前是 1000 字节左右，苹果的额外推送设置如角标，也会占用字节数
75	消息体格式不符合 json 格式	检查消息体即 message 字段内容
76	请求过于频繁，请稍后再试	全量广播限频为每 3 秒一次
78	循环任务参数错误	检查 loop time
90	设备离线	重新打开应用
91	设备 tag 过多	清理不使用的 tag
92	apptag 过多	清理不使用的 tag
100	APNS 证书错误，请重新提交正确的证书	证书格式是 pem 的，另外，注意区分生产证书、开发证书的区别
-101	参数错误	请检查参数
-102	请求 timestamp 字段超过了时间过期	请使用当前系统时间戳，确保时间同步
-103	sign 不合法	检查签名生成流程，生成 sign 是 METHOD 必须与请求时所使用的一致
-105	请求过于频繁	稍后重试
-106	证书错误	证书错误
-111	缺少公共参数：access_id\timestamp\sign	检查公共参数access_id\timestamp\sign
-112	参数取值非法	检查参数取值
其他	内部错误	稍后重试，如出现为标明且必现错误请及时与我们取得联系

客户端返回码

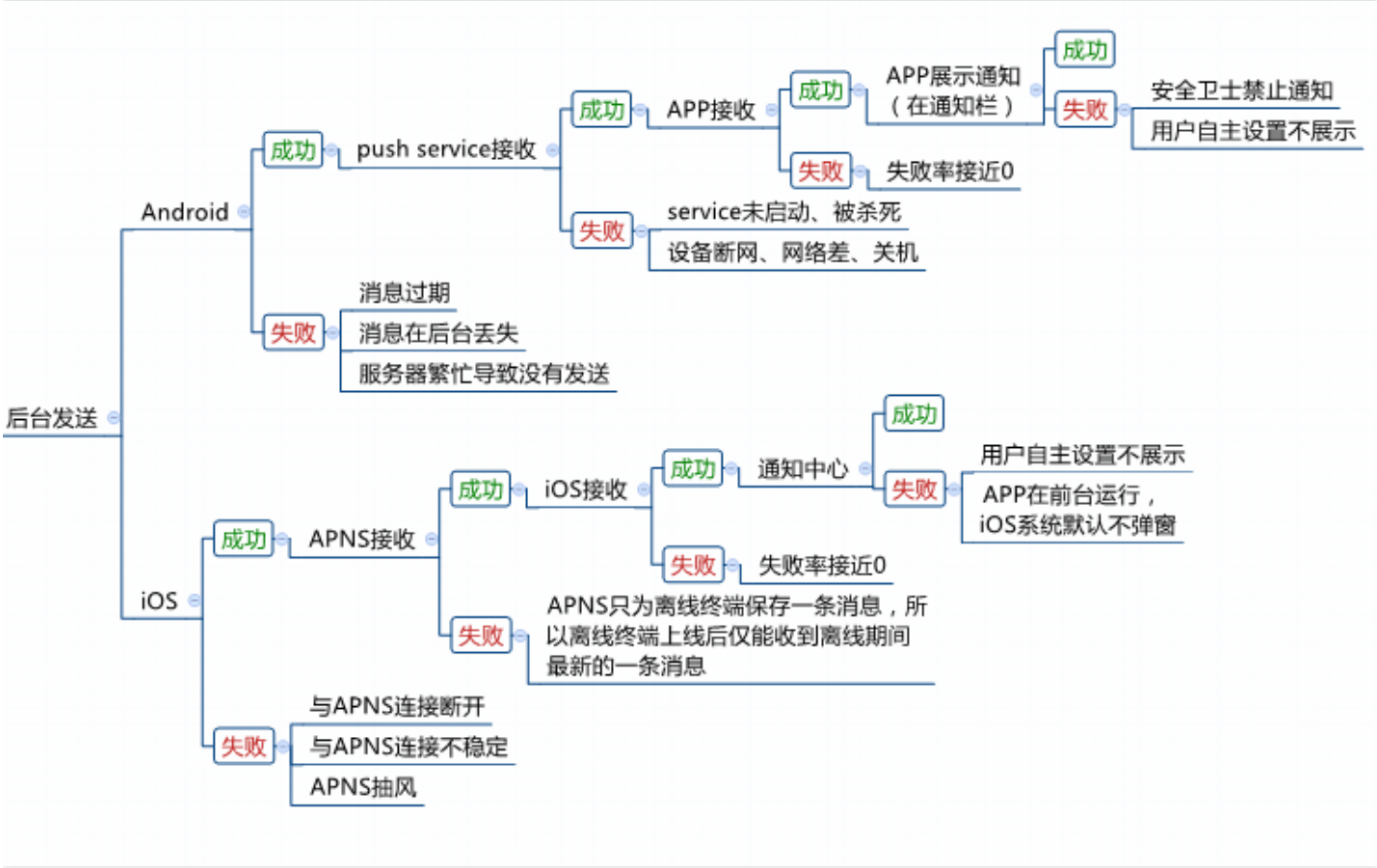
值	原因以及解决办法
0	调用成功

值	原因以及解决办法
2	参数错误，例如绑定了单字符的别名，或是 ios 的 token 长度不对，应为 64 个字符
20	鉴权错误,access id 或者 access key 配置错误
10000	起始错误
10001	操作类型错误码，例如参数错误时将会发生该错误
10002	正在执行注册操作时，又有一个注册操作到来，则回调此错误码
10003	权限配错或者缺少所需权限
10004	so 库没有正确导入（Androidstudio 可在 main 文件目录下 添加 jniLibs 命名的文件夹将 SDK 文档中的 Other-Platform-SO 下的 7 个 so 库文件夹添加至该目录）
10005	AndroidManifest 文件的 XGRemoteService 节点没有配置或者的该节点的 action 包名配错
10008	jce JAR 错误或者缺少 jce JAR（如果是混淆打包过后出现,请检查混淆代码）
10101	创建链路失败（切换网络重试）
10102	请求处理过程中， 链路被主动关闭（切换网络重试）
10103	请求处理过程中，服务器关闭链接（切换网络重试）
10104	请求处理过程中，客户端产生异常（切换网络重试）
10105	请求处理过程中，发送或接收报文超时（切换网络重试）
10106	请求处理过程中， 等待发送请求超时（切换网络重试）
10107	请求处理过程中， 等待接收请求超时（切换网络重试）
10108	服务器返回异常报文
10109	未知异常，切换网络 或者 重启设备）
10110	创建链路的 handler 为 null
其他	如出现其他未知错误请记录错误日志与我们取得联系

推送流程图

最近更新时间：2018-04-16 19:19:51

消息推送流程图



Messaging 常见问题
Android 常见问题

最近更新时间：2018-04-16 19:28:37

推送时效性问题

全量推送接口，批量推送接口（批量账号，批量 token，tag），会有一个三十秒左右的任务调度时间。单推接口基本上秒达。（单推账号，单推 token）秒达。

> 注意：
> 在中午 12 点。晚六点到八点为推送高峰期，部分消息可能会延时到达。

本地通知延时

需要保证应用在前台，信鸽 service 存活，正常运行。本地通知才能展示，关闭应用无法展示本地通知，本地通知是根据网络心跳来判断弹出通知，大约五分钟一次心跳，不能保证准时弹出推送前后可能会有一定的时间差。

收不到推送的问题

用获取到的 token，在控制台推送。如无法收到推送请根据以下情况进行排查（请确保 SDK 版本是最新的版本，如果是旧版本出现问题，在新版本可能已经修复，如遇到文本端推送报错，请刷新页面重试）。

注册成功无法收到推送：

1. 请查看当前的应用包名和注册信鸽应用时填写的应用包名是否一致。如果不一致，推送的时候建议开启多包名推送。
2. 查看设备是否开启通知栏权限，oppo，vivo 等手机，需要手动开启通知栏权限。
3. 信鸽推送分为 通知栏消息 和 应用内消息（透传消息），通知栏消息可以展示到通知栏，应用内消息不能展示到通知栏。
4. 确认手机当前模式是正常模式，部分手机在低电量，勿扰模式，省电模式下，会对后台信鸽进程进行一系列网络和活动的限制。

注册不成功无法收到推送：

1. 注册返回错误:

如 10004，20 等请参考错误码一览。

错误 10004

原因：.so 文件导入不全，so 是用来适配各种设备的不同型号的 CPU，如出现 10004 的错误，应该查看当前导入的

.so 库文件是否支持当前设备的 CPU。如果不支持需要添加对应的 .so 文件（完整的 .so 库在 SDK 文件夹下 Other-Platform-SO 目录内）。

eclipse 开发工具解决办法：
将需要的对应设备 CPU 的 .so 文件复制到 lib 目前中。

Androidstudio 的开发工具的解决办法：
Androidstudio 可在 .main 文件目录下添加 jniLibs 命名的文件夹将 SDK 文档中的 Other-Platform-SO 下的七个 .so 库文件夹添加至该目录 ，或者采用自动接入，无须手动导入 .so 文件。

2. 如注册无回调:
确认当前*网络情况*是否良好（建议使用 4G 网络测试，WiFi 由于使用人数过多可能造成网络带宽不足），是否添加 *wup 包*，以及*努比亚手机*（部分机型不支持第三方推送）在 2015 年下半年和 2016 年出的机器都无法注册，具体机型包括 nubia Z11 系列，nubiaZ11S 系列，nubiaZ9S 系列。可以的机器都是之前的机器，包括 Z7 系列，my 布拉格系列（在信鸽 2.47 和信鸽 3.X 上都有这个现象）。

关闭应用无法收到推送

目前第三方推送都无法保证关闭应用过后还可以收到推送消息，这个是手机定制 ROM 对信鸽 service 的限制问题，信鸽的一切活动都需要建立在信鸽的 service 能够正常联网运行。
QQ，微信是系统级别的应用白名单，相关的 service 不会由于应用关闭而退出，所以用户感知退出应用过后还可以收到消息，其实是因为相关的 service 还在后台存活。
Android 端在应用退出，信鸽 service 和信鸽的服务器断开连接后，这个时候给该设备下发的消息，会变成离线消息，离线消息最多保存 72 条消息，每个设备最多保存两条，如果有多条离线消息。在关闭应用期间推送的消息，如开启应用无法收到，请检查是否调用了反注册接口：XGPushManager.unregisterPush(this)。

账号推送收不到

每个账号最多可以绑定 15 个设备，超过 15 个设备，会自动顶掉最先绑定的一个账号。每个设备注册的有效账号为最后一次绑定的账号，如果多个设备同时绑定多个账号，则全部能收到推送。

tag 推送收不到

请确认 tag 标签是否绑定成功，一个应用最多有 10000 个标签（tag），每个 token 在一个应用下最多一百个标签（tag），标签（tag）中不准包含空格。

信鸽推送是否支持海外

只要能 ping 通信鸽服务器域名 openapi.xg.qcloud.com 就能够收到推送消息，信鸽海外服务器部署在香港，由于在海外地区网络延时较高，信鸽在海外的推送效果会略低于在国内的推送效果。
测试方法： 在想测试的网络环境，打开命令行，输入 ping openapi.xg.qcloud.com 再回车终端输出如下日志 表示能够成功连上信鸽服务器：

```
admin$ ping openapi.xg.qcloud.com
PING openapi.xg.qcloud.com (******* ip地址): 56 data bytes
```

```
64 bytes from 14.215.138.42: icmp_seq=0 ttl=54 time=4.364 ms
64 bytes from 14.215.138.42: icmp_seq=1 ttl=54 time=5.352 ms
64 bytes from 14.215.138.42: icmp_seq=2 ttl=54 time=4.514 ms
64 bytes from 14.215.138.42: icmp_seq=3 ttl=54 time=4.924 ms
64 bytes from 14.215.138.42: icmp_seq=4 ttl=54 time=4.447 ms
64 bytes from 14.215.138.42: icmp_seq=5 ttl=54 time=4.843 ms
64 bytes from 14.215.138.42: icmp_seq=6 ttl=54 time=5.946 ms
```

推送数据问题

推送暂停

1. 相同的内容的全量推送每小时只能推送一次，超过一次推送会被暂停。
2. 每小时最多推送 30 条全量推送，超过 30 次会被暂停。

效果统计

【次日】：推送完第二天才能看到推送数据；
【实时】：推送完马上可以看到推送数据。目前每周仅支持 14 次的实时数据统计。

实发

在消息离线保存时间内，有成功连接到信鸽服务器，并且有正常下发的量。（如：消息离线保存时间为三天，实发数据会在第四天稳定，数据会随着设备不断开启连接到信鸽服务器的数量而增加）。

历史明细

历史明细只展示全量推送，tag 推送和官网的号码包推送。（其他推送接口不展示推送详情）

数据概览

展示的是当天的数据，某天的数据是在那一天中各种推送行为的推送总量。（分为单推，广播也就是批量和全量推送，通知栏消息和应用内消息四类）

iOS 常见问题

最近更新时间：2018-04-16 19:28:50

iOS 是否支持离线保存？

苹果默认支持离线保存一条消息。关于离线保存的时长苹果官方文档没有明确的说明。

为何每天看到的全量推送的实发量会有波动，有时高有时低？

信鸽后台会根据每天推送时，apns 返回的错误来清理已经过期的无效 Token。这个清理每天都会执行一次，因此第二天的全量推送实发量是已经除去了前一天的过期 Token 的数量，可能会比前一天的实发量少。这是属于正常现象。

初始化信鸽接口，出现如下日志 2017-10-26 15:13:38.888951+0800 XG-Demo [2295:1737660] [xgpush] 服务器返回码: 20

在初始化信鸽的方法中 appid 和 appkey 不要使用宏定义。

什么情况会出现推送暂停

每小时最多可创建 30 条全量推送，超过 30 条的推送将被推送暂停，一小时内创建推送内容完全一样的推送，将被推送暂停，推送暂停的任务将不会下发，请视情况重新创建推送。

上传证书到管理台失败

验证失败，请刷新后重试

用编辑器打开证书文件，找到 friendlyname 字段如果同行有？，将其修改成别的，保存后重新上传。

不包含 push 参数

制作新的推送证书

文件大小为 0 kb，不能上传

重新转换 pem 格式，信鸽证书制作教程：iOS 推送证书设置指南 。

终端出现"Error Domain=NSCocoaErrorDomain Code=3000 "未找到应用程序的“aps-environment”的授权字符串" UserInfo=0x16545fc0 {NSLocalizedDescription=未找到应用程序的“aps-environment”的授权字符串}"错误

这是由于 App 证书没有推送权限引起的。请重新配置证书。

设备收到消息没有进入回调

iOS 10 会进静默通知的回调方法中。

设置/删除标签的时候出现如下错误 exception.name= WupSerializableException exception.reason= -[XGJceOutputStream writeAnything:tag:required:], 349: assert(0) fail!

请在 registerDevice 之后再 setTag/delTag.在 registerDevice 之前进行 tag 操作会出现这个错误。

Token 和别名（account）的对应关系

一个设备一个 token，token 在注册推送时由苹果下发，一个 token 最多绑定一个 account，一个 account 最多绑定 15 个 token，超出数量时会顶替之前绑定的 token。iOS 的 token 是会变化的，卸载，重装，刷机，重置都会导致 token 发生变化。

创建推送成功了，但推送列表没有该条推送的记录

推送列表只展示针对所有设备和批量设备的推送记录。

如何播放自定义通知音？

把音频文件放到 bundle 目录下，创建推送时，给 sound 字段传入音频文件名称。

使用信鸽服务端 SDK，怎么创建静默推送？

给参数 content-available 赋值 1，同时不使用 setalert。

常见问题

最近更新时间：2018-03-29 10:44:37

iOS SDK 中有没有使用热更新或者私有接口？

Messaging SDK 没有使用过热更新或私有接口，不会影响苹果审核。

推送数量/推送频率限制？

推送数量无限制。推送频率上，仅全量广播限频为每 3 秒一次，其他推送行为不限频。

对单个设备，保存多少条离线信息？保存时间？

离线消息 Android 最多保存 2 条，iOS 最多保存 1 条；保存时间最多 72 小时。

标签方面限制？

单个设备最多设置 100 个标签，单个 App 全局最多可以有 10000 个不同的标签。

当第一次注册成功后，没有反注册，以后使用还需要注册吗？

不需要，只要没反注册，就不需要再次注册。

应用关闭或结束进程后，还能收到推送消息吗？

Messaging 主要依赖 service 进行消息的收发，杀死进程之后 service 也被杀死，只能等待 service 被拉活或重启 App 才可以收到推送。若手机中有其他接入 Messaging 的 App 被打开，则可以利用其他 App 的 service 接收消息，但共享 service 通道也受手机 ROM 限制，无法保证百分之百的成功率。

设备注册为什么收不到回调信息？

注册操作中，后台只可能有三种出错行为：

- 不响应；
- 返回错误格式的数据包；
- 返回错误码。这三种行为终端应该都可以检测到并给出回调。

为什么我推送成功了，有了抵达量，点击量却等于 0 ？

iOS 点击量统计需要调用特殊代码，具体请参考 iOS 开发文档。

为什么会出现推送通知时，只有声音却没有文字信息的情况？

该问题与系统有很大关系，需要拿设备的 logcat 来进行特定分析。

token 与 Account 区别？

token 是一台设备（device）的标识，账号是一个用户（users）的标识。一个 token 只能绑定一个账号，多次绑定时，以最后一次为准。

账号在设备 A 上登录过，又在设备 B 上登录？给这个账号发信息会怎么样？

只要是没有注销，则两台设备都会收到。

标签与账号的区别？

标签是用于标识一个 token 或用户的一些属性，如广东省、男性、游戏玩家等。帐号是用户的账号，请勿用标签作为别名使用。

在应用列表中看到“覆盖设备数”，具体指的是什么？

是指该应用下处于注册状态的设备数/终端数，同时也是该应用在推送时可以覆盖到的最大设备数。终端若调用了 unregister 的接口，覆盖设备数会减少。

为什么在 Web 端推送出现服务器繁忙？

请先检查 token 以及所选推送环境是否正确，然后检查证书是否正确提交，若还出现相同错误可重新制作一份不带密码的证书提交再试。

推送过程中，非定时推送（立即推送）能否撤销？

不能，只有返回 push_id 的任务才可以做撤销操作。

推送后查看推送列表，已经推送完成了，状态却显示推送中，怎么办？

刷新再试试。

在推送时，如何向单个用户推送消息？

请参考开发手册，有关于“推送消息给单个设备”和“推送消息给单个账户或别名”的使用指南。

用户重连上线后收到多条 push 的顺序是怎样？

按照消息 ID 递增。客户端也是按照此规则收取消息，因此，收消息的顺序就是发消息的顺序。

我现在有安卓的用户和 iOS 的用户，那我 PHP 后台要写两个不同的接口分别推给安卓用户和 iOS 用户吗？

需要调用两次推送接口，也可以把两个封装为一个。

如果定时 push 选择的是过去的时间，是不是不会 push 出去？

不是，选择过去的时间系统则会立刻发送。