移动推送

App SDK 手册

阿里云
aliyun.com

App SDK 手册

Android SDK手册

使用前必读：移动推送名词解释&约束

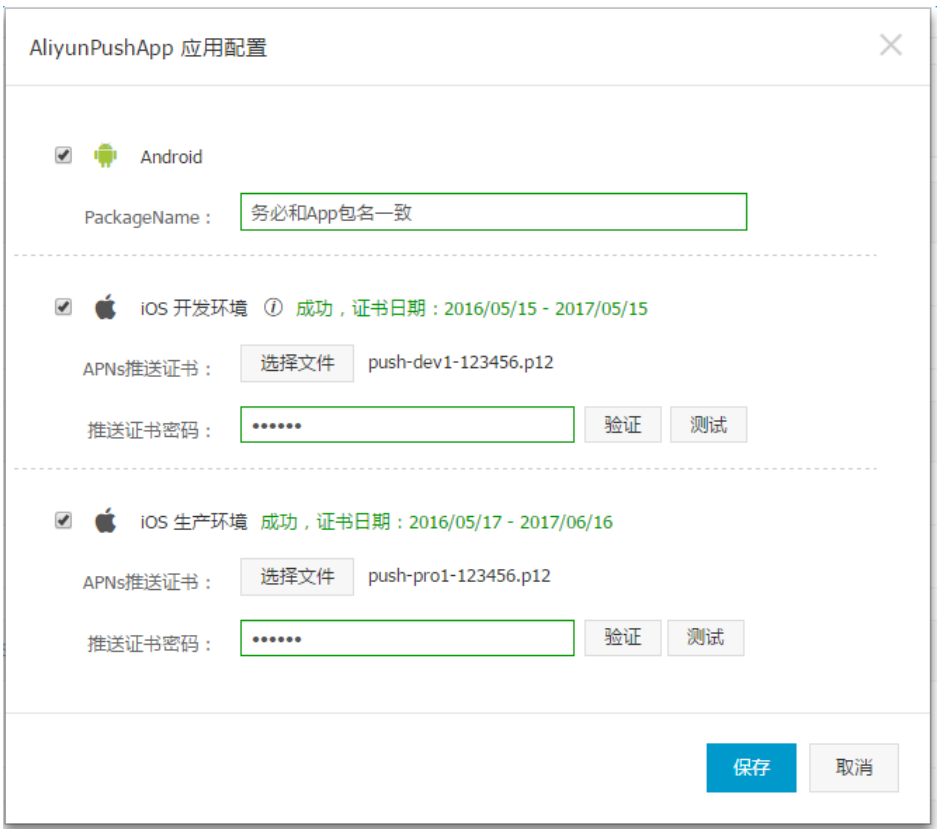
1. 创建应用

到阿里云移动推送控制台创建应用，应用创建完成以后，进入移动推送相关模块进行设置，具体操作请参见 创建APP。

在应用中完成应用配置，***请注意PackageName务必和App的包名一致***，否则推送将无法正确初始化。

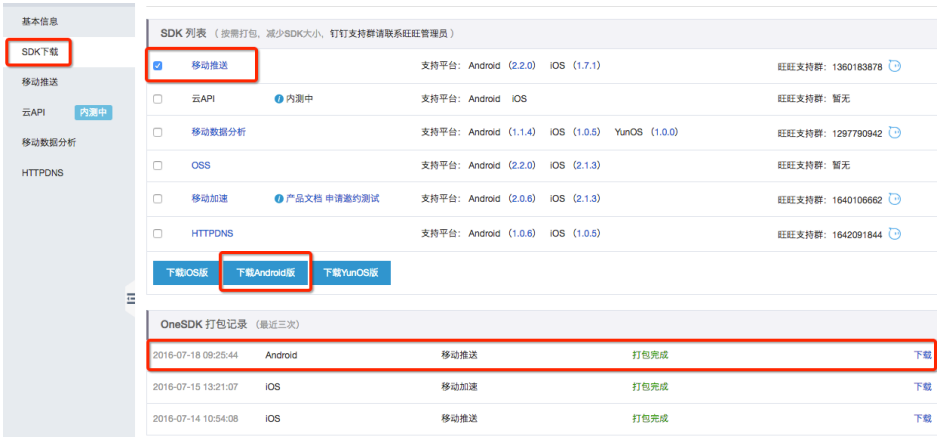
【注意】使用Gradle构建App时，PackageName的查看：

- 查看AndroidManifest.xml中根元素package属性；
- 查看工程build.gradle中applicationId设置，默认AndroidManifest.xml中的package属性保持一致，如果不一致，以applicationId为准。



2. SDK下载和集成

2.1 SDK下载



2.2 SDK目录结构

```
OneSDK
|-- AndroidManifest.xml
|-- build.gradle

|-- libs
```

```
| |-- armeabi
| | |-- libcocklogic.so -网络连接库及幽灵进程的辅助lib
| | |-- libtnet.so
| |-- armeabi-v7a
| | |-- libcocklogic.so
| | |-- libtnet.so
| |-- arm64-v8a
| | |-- libcocklogic.so
| | |-- libtnet.so
| |-- x86
| | |-- libcocklogic.so
| | |-- libtnet.so
| |-- arm64
| | |-- libcocklogic.so
| | |-- libtnet.so

| |-- alicloud-android-push-sdk.jar -移动推送主功能包
| |-- alisdk-ut.jar -UT基础包
| |-- utdid4all.jar -设备Id生成包

|-- project.properties
|-- src
```

2.3 SDK集成：

- 请在工程中添加android-support-v4.jar支持包（v2.3.0以上），关于v4支持包的说明请参考：https://developer.android.com/topic/libraries/support-library/features.html#v4；
- 手动拷贝下载SDK中的libs目录，手动拷贝后需要在APP工程的build.gradle中配置jniLibs的目录：

```
android {
...
sourceSets {
main {
jniLibs.srcDirs = ['libs']
}
}
}
```

3. 配置AndroidManifest.xml

3.1 appkey和appsecret配置

```
<meta-data android:name="com.alibaba.app.appkey" android:value="*****"/> <!-- 请填写你自己的- appKey -->
<meta-data android:name="com.alibaba.app.appsecret" android:value="****"/> <!-- 请填写你自己的appSecret -->
```

com.alibaba.app.appkey和com.alibaba.app.appsecret为您App的对应信息，在推送控制台APP列表页的应用证书中获取。

3.2 Permission 的配置

- 将以下uses-permission片段拷贝进你manifest中的Permission申明区域中：

```
<!--阿里移动推送相关权限-->
<!--Android 6.0版本可去除，用于选举信息（通道复用）的同步-->
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<!--进行网络访问和网络状态监控相关的权限声明-->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!--允许对sd卡进行读写操作-->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<!--网络库使用，当网络操作时需要确保事务完成不被杀掉-->
<uses-permission android:name="android.permission.WAKE_LOCK" />
<!--用于读取手机硬件信息等，用于机型过滤-->
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<!--选举使用，当应用有删除或者更新时需要重新选举，复用推送通道-->
<uses-permission android:name="android.permission.BROADCAST_PACKAGE_CHANGED" />
<uses-permission android:name="android.permission.BROADCAST_PACKAGE_REPLACED" />
<uses-permission android:name="android.permission.RESTART_PACKAGES" />
<!--补偿通道小米PUSH使用，不用可去除-->
<uses-permission android:name="android.permission.GET_TASKS" />
<!--补偿通道GCM使用，不使用可去除-->
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<!--允许监听启动完成事件-->
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<!--允许访问震动器-->
<uses-permission android:name="android.permission.VIBRATE" />
```

3.3 Service 的配置

- 将以下service片段拷贝进你Manifest中的service申明区域中：

```
<!-- 通道保持服务 -->
<service android:name="com.alibaba.sdk.android.push.ChannelService"
android:exported="true" android:process=":channel">
<intent-filter>
<action android:name="com.taobao.accs.intent.action.SERVICE"/>
</intent-filter>
<intent-filter>
<action android:name="org.agoo.android.intent.action.PING_V4" />
<category android:name="taobao" />
</intent-filter>
</service>

<!-- 消息接收服务 -->
<service android:name="com.alibaba.sdk.android.push.MsgService"
android:exported="false">
<intent-filter>
<action android:name="com.taobao.accs.intent.action.RECEIVE" />
</intent-filter>
```

```
<intent-filter>
<action android:name="com.alibaba.sdk.android.push.NOTIFY_ACTION" />
</intent-filter>
</service>
```

3.4 Receiver 的配置

将以下receiver片段拷贝进你manifest中的receiver申明区域中：

```
<!-- 连接心跳保持监听器 -->
<receiver android:name="anet.channel.heartbeat.HeartbeatManager$Receiver" >
<intent-filter>
<action android:name="anetwork.channel.intent.action.COMMAND" />
</intent-filter>
</receiver>

<!--消息接收监听器-->
<receiver android:name="com.alibaba.sdk.android.push.MessageReceiver">
<intent-filter>
<action android:name="com.alibaba.push2.action.NOTIFICATION_OPENED"/>
</intent-filter>
<intent-filter>
<action android:name="com.alibaba.push2.action.NOTIFICATION_REMOVED"/>
</intent-filter>
<intent-filter>
<action android:name="com.taobao.accs.intent.action.COMMAND" />
</intent-filter>
<intent-filter>
<action android:name="com.taobao.taobao.intent.action.COMMAND" />
</intent-filter>
<intent-filter>
<action android:name="org.agoo.android.intent.action.RECEIVE" />
</intent-filter>
<intent-filter>
<action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
</intent-filter>
<intent-filter>
<action android:name="android.intent.action.USER_PRESENT" />
</intent-filter>
<intent-filter>
<action android:name="android.intent.action.BOOT_COMPLETED"/>
</intent-filter>
<intent-filter>
<action android:name="android.intent.action.PACKAGE_REMOVED"/>
<data android:scheme="package"/>
</intent-filter>
</receiver>
```

4. Proguard配置

```
-keepclasseswithmembernames class ** {
native <methods>;
}
-keepattributes Signature

-keep class sun.misc.Unsafe { *; }

-keep class com.taobao.** {*;}
-keep class com.alibaba.** {*;}
-keep class com.alipay.** {*;}
-dontwarn com.taobao.**
-dontwarn com.alibaba.**
-dontwarn com.alipay.**

-keep class com.ut.** {*;}
-dontwarn com.ut.**

-keep class com.ta.** {*;}
-dontwarn com.ta.**

-keep class anet.**{*;}
-keep class org.android.spdy.**{*;}
-keep class org.android.agoo.**{*;}
-dontwarn anet.**
-dontwarn org.android.spdy.**
-dontwarn org.android.agoo.**
```

5. 在应用中注册和启动移动推送

首先通过PushServiceFactory获取到CloudPushService，然后调用register()初始化并注册云推送通道，并确保Application上下文中进行初始化工作。

请参照以下代码段进行初始化：

```
import android.app.Application;
import android.content.Context;
import android.util.Log;
import com.alibaba.sdk.android.AlibabaSDK;
import com.alibaba.sdk.android.callback.InitResultCallback;
import com.alibaba.sdk.android.push.CloudPushService;
import com.alibaba.sdk.android.push.CommonCallback;
import com.alibaba.sdk.android.push.noonesdk.PushServiceFactory;

public class MainApplication extends Application {
private static final String TAG = "Init";
@Override
public void onCreate() {
super.onCreate();
initCloudChannel(this);
}
```

```
/**
* 初始化云推送通道
* @param applicationContext
*/
private void initCloudChannel(Context applicationContext) {
PushServiceFactory.init(applicationContext);
CloudPushService pushService = PushServiceFactory.getCloudPushService();
pushService.register(applicationContext, new CommonCallback() {
@Override
public void onSuccess(String response) {
Log.d(TAG, "init cloudchannel success");
}

@Override
public void onFailed(String errorCode, String errorMessage) {
Log.d(TAG, "init cloudchannel failed -- errorcode:" + errorCode + " -- errorMessage:" + errorMessage);
}
});

}
}
```

【注意】：

- 如果设备成功注册，将回调callback.onSuccess()方法。
- 但如果注册服务器连接失败，则调用callback.onFailed方法，并且自动进行重新注册，直到onSuccess为止。（重试规则会由网络切换等时间自动触发。）
- 请在网络通畅的情况下进行相关的初始化调试，如果网络不通，或者App信息配置错误，在onFailed方法中，会有相应的错误码返回，可参考错误处理。

启动正常确认方法：

- 回调方法中日志打印正常（以上边接入代码为例）

```
11-24 12:55:51.096  15235-15535/com.alibaba.xxxx D/YourApp： init cloudchannel success
```

- 确认cloudchannel初始化正常，在logcat日志中：输入awcn关键字：

```
11-24 12:53:51.036  15235-15556/com.alibaba.xxxx E/awcn： |[seq:AWCN1_1] AUTH httpStatusCode: 200
11-24 12:53:51.036 15235-15556/com.alibaba.xxxx E/awcn： |[seq:AWCN1_1] status:AUTH_SUCC
```

- 确认DeviceId获取正常:在初始化成功后使用 cloudPushService.getDeviceId() 获取deviceId，应该能够成功获取。

Android API

Android SDK最新版本v2.3.0。

1. CloudPushService接口

以下接口调用时，如有回调，回调不能为空。

SDK注册

- 初始化推送SDK，关联到云通道。

参数

- context 应用上下文（需要ApplicationContext）
- callback 回调

```
void register(Context context, CommonCallback callback);
```

启动信息统计

- 统计App启动信息。

```
void onAppStart();
```

绑定账号

- 将应用内账号和推送通道相关联，可以实现按账号的定点消息推送；
- 设备只能绑定一个账号，多次绑定操作仅最后一个生效；
- 账户名设置支持32字节。

参数

- account 绑定账号名
- callback 回调

```
void bindAccount(String account, CommonCallback callback);
```

解绑账号

- 将应用内账号和推送通道取消关联。

参数

- callback 回调

```
void unbindAccount(CommonCallback callback);
```

绑定标签

- 绑定标签到指定目标；
- 支持向设备、账号和别名绑定标签，绑定类型由参数target指定；
- 绑定标签后，第二天服务端可按该标签推送，即（T + 1）天生效；
- App最多支持绑定128个标签，【请谨慎使用，避免标签绑定达到上限】，单个标签最大支持40字节。

参数

- target 目标类型，1：本设备； 2：本设备绑定账号； 3：别名
- tags 标签（数组输入）
- alias 别名（仅当target = 3时生效）
- callback 回调

```
void bindTag(int target, String[] tags, String alias, CommonCallback callback);
```

解绑标签

- 解绑指定目标标签；
- 支持解绑设备、账号和别名标签，解绑类型由参数target指定；
- 解绑标签后，当天服务端可继续按该标签推送，解绑生效时间在第二天，即（T + 1）天生效；
- 解绑标签不等同于删除标签，目前不支持标签的删除。

参数

- target 目标类型，1：本设备； 2：本设备绑定账号； 3：别名
- tags 标签（数组输入）
- alias 别名（仅当target = 3时生效）
- callback 回调

```
void unbindTag(int target, String[] tags, String alias, CommonCallback callback);
```

查询标签

- 查询目标绑定标签，当前仅支持查询设备标签；
- 查询结果可从回调onSuccess(response)的response获取；
- 标签绑定成功后即可查询。

参数

- target 目标类型，1: 本设备
- callback 回调

```
void listTags(int target, CommonCallback callback);
```

添加别名

- 设备添加别名；
- 别名支持128字节。

参数

- alias 别名
- callback 回调

```
void addAlias(String alias, CommonCallback callback);
```

删除别名

- 删除设备别名；
- 支持删除指定别名和删除全部别名（alias = null || alias.length = 0）。

参数

- alias 别名（alias = null or alias.length = 0时，删除设备全部别名）
- callback 回调

```
void removeAlias(String alias, CommonCallback callback);
```

别名查询

- 查询设备别名；
- 查询结果可从回调onSuccess(response)的response中获取

参数

- callback 回调

```
void listAliases(CommonCallback callback);
```

设置通知声音

- 设置推送通知声音文件路径；
- 若不调用本接口，默认获取资源id为R.raw.alicloud_notification_sound的资源文件；
- 若没有获取到指定声音文件，取设备设置的消息声音。

参数

- filePath 通知声音文件路径

```
void setNotificationSoundFilePath(String filePath);
```

设置通知栏图标

- 设置推送通知栏图标资源Bitmap。
- 若不调用本接口，默认获取id为R.drawable.alicloud_notification_largeIcon的资源文件；
- 若没有获取到指定图标文件，取App启动图标。

参数

- icon 图标资源Bitmap

```
void setNotificationLargeIcon(Bitmap icon);
```

设置状态栏图标

- 设置推送状态栏图标资源Id；
- 若不调用本接口，默认获取id为R.drawable.alicloud_notification_smallIcon的资源文件；
- 若没有获取到指定资源文件Id，取App启动图标。

参数

- iconId 图标资源Id

```
void setNotificationSmallIcon(int iconId);
```

获取设备标识

- 获取设备唯一标识。

返回

- 设备唯一标识。

```
String getDeviceId();
```

设置日志等级

- 需要在通道初始化之前设置；
- 默认日志等级为CloudPushService.ERROR；

参数

- logLevel 支持设置：CloudPushService.ERROR | CloudPushService.INFO | CloudPushService.DEBUG | CloudPushService.OFF（关闭Log）

```
void setLogLevel(int logLevel);
```

设置免打扰时段

- 设置免打扰时间段，过滤所有通知与消息；
- 免打扰时段仅支持设置一次，多次调用以最后一次调用设置时段为准；
- 免打扰时段设置对小米辅助弹窗通知无效。

参数

- startHour 免打扰的起始时间（小时），24小时制，取值范围：0-23
- startMinute 免打扰起始时间（分钟），取值范围：0-59
- endHour 免打扰的结束时间（小时），24小时制，取值范围：0-23
- endMinute 免打扰结束时间（分钟），取值范围：0-59

```
void setDoNotDisturb(int startHour, int startMinute, int endHour, int endMinute, CommonCallback callback);
```

2. MessageReceiver

- 通过继承MessageReciever，可以拦截通知，接收消息，获取推送中的扩展字段。或者在通知打开或删除的时候，切入进行后续处理。

使用方法：

- 继承com.alibaba.sdk.android.push.MessageReceiver；
- 在Manifest中找到原来MessageReceiver的配置，将上边的class替换成你自己的receiver[不要配置多个]。

```
<!--消息接收监听器-->
```

```
<receiver android:name="com.alibaba.sdk.android.push.MessageReceiver <-- 把这里替换成你自己的receiver">
<intent-filter>
<action android:name="com.alibaba.push2.action.NOTIFICATION_OPENED"/>
</intent-filter>
... ...
</receiver>
```

消息接收回调

- 用于接收服务端推送的消息。
- 消息不会弹窗，而是回调该方法。

参数

- context 上下文环境
- message CPushMessage类型，可以获取消息Id、消息标题和内容。

```
void onMessage(Context context, CPushMessage message);
```

通知接收回调

- 客户端接收到通知后，回调该方法。
- 可获取到并处理通知相关的参数。

参数

- context 上下文环境
- title 通知标题
- summary 通知内容
- extraMap 通知额外参数

```
void onNotification(Context context, String title, String summary, Map<String, String> extraMap)
```

通知打开回调

- 打开通知时会回调该方法。

参数

- context 上下文环境
- title 通知标题
- summary 通知内容
- extraMap 通知额外参数

```
void onNotificationOpened(Context context, String title, String summary, String extraMap);
```

通知删除回调

- 删除通知时回调该方法。

参数

- context 上下文环境
- messageId 删除通知的Id

```
void onNotificationRemoved(Context context, String messageId);
```

小米/华为系统推送支持

1. 配置应用

- 在 小米开放平台 注册你的App, 得到相应的小米AppID，小米AppKey，小米AppSecert。在控制台App详情设置你的小米AppSecert。（注意：小米开发者平台的消息推送开关记得打开）
- 同理在 华为开发者联盟 注册App，应用审核通过后，能够得到华为的AppID和AppSecert。在控制台App详情中设置你的华为AppID和AppSecert。(注意，您的app不能是草稿状态，必须是审核中，或者通过审核的状态，不然通道不会生效。请确保您在华为控制台激活了推送通道功能)

2. 下载扩展包

将小米华为扩展包拷贝到你项目的Lib目录下，下载地址。

3. 配置Manifest

- 为小米特别通道新增配置Permission：

```
<!--小米通道相关权限 -->
<permission
android:name="你的包名.permission.MIPUSH_RECEIVE"
android:protectionLevel="signatureOrSystem"/>
<uses-permission android:name="你的包名.permission.MIPUSH_RECEIVE"/>
```

【注意】：请确保上述Permission配置是在manifest标签域，而非application标签域。关于AndroidManifest.xml应用清单文件的说明参考：
https://developer.android.com/guide/topics/manifest/manifest-intro.html#filec

- 为小米特别通道新增配置Manifest：

```
<!-- 小米通道官方配置 -->
<service android:name="com.xiaomi.push.service.XMPushService" android:enabled="true"
android:process=":channel" />
<service android:name="com.xiaomi.mipush.sdk.PushMessageHandler" android:enabled="true"
android:exported="true" />
<service android:enabled="true" android:name="com.xiaomi.mipush.sdk.MessageHandleService" />
<receiver android:name="com.alibaba.sdk.android.push.MiPushBroadcastReceiver" android:exported="true" >
<intent-filter>
<action android:name="com.xiaomi.mipush.RECEIVE_MESSAGE" />
</intent-filter>
<intent-filter>
<action android:name="com.xiaomi.mipush.MESSAGE_ARRIVED" />
</intent-filter>
<intent-filter>
<action android:name="com.xiaomi.mipush.ERROR" />
</intent-filter>
</receiver>
<receiver android:name="com.xiaomi.push.service.receivers.NetworkStatusReceiver" android:exported="true" >
<intent-filter>
<action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
<category android:name="android.intent.category.DEFAULT" />
</intent-filter>
</receiver>
<receiver android:name="com.xiaomi.push.service.receivers.PingReceiver" android:exported="false"
android:process=":channel" >
<intent-filter>
<action android:name="com.xiaomi.push.PING_TIMER" />
</intent-filter>
</receiver>
```

注意，com.xiaomi.push.service.XMPushService加入Manifest后，这个类名会在Android Studio中显示成红色，请忽略该错误，不要删除这行配置。（该应用在小米机器上会找到对应的Service）

- 为华为特别通道新增配置Manifest：

```
<!-- 华为通道官方配置 -->
<receiver android:name="com.alibaba.sdk.android.push.HuaWeiReceiver">
<intent-filter>
<action android:name="com.huawei.android.push.intent.REGISTRATION"/>
```

```
<action android:name="com.huawei.android.push.intent.RECEIVE"/>
<action android:name="com.huawei.intent.action.PUSH"/>
<action android:name="com.huawei.intent.action.PUSH_STATE"/>
</intent-filter>
</receiver>
```

4. 在应用中初始化小米华为通道

将以下代码加入你application.onCreate()方法中初始通道：

```
// 注册方法会自动判断是否支持小米系统推送，如不支持会跳过注册。
MiPushRegister.register(applicationContext, "小米AppID", "小米AppKey");

// 注册方法会自动判断是否支持华为系统推送，如不支持会跳过注册。
HuaWeiRegister.register(applicationContext);
```

本方法会自动判断是否支持小米系统推送，如不支持会跳过注册。

5. 在日常中查看初始化情况

华为通道初始化成功，可以看到以下日志：

```
11-11 22:21:33.671  30248-30324/com.xxx E/MPS:HuaWeiRegister: HuaWeiRegister checkDevice flag=true //确认是华为的手机
11-11 22:21:33.674 30248-30324/com.xxx E/MPS:HuaWeiRegister： Register huawei push..... //开始注册华为手机
11-11 22:21:33.714 29643-30328/com.xxx E/MPS:HuaWeiReceiver： huawei register success，token = 086574302431254720000000411000001
11-11 22:21:33.714 29643-30328/com.xxx E/MPS:HuaWeiReceiver： report huaweiPushId intent... //完成华为注册和信息上报
```

小米通道初始化成功，可以看到以下日志：

```
12-09 22:20:39.710 19566-19566/com.xxx E/MPS:MiPushRegister: MiPushRegister checkDevice flag=true //确认是小米的手机
12-09 22:20:39.712 19566-19566/com.xxx E/MPS:MiPushRegister: Register mipush. //开始注册小米
12-09 22:20:40.596 19566-19733/com.xxx E/MPS:MiPushReceiver: XiaoMi register success. //小米注册成功
regid=d//igwEhgBGCI2TG6lWqlCesc0I6xE1wUhNCBXQ8uNOi/dDZioYXVysbrVrvRmyEVPn9nWz92D28IzYbA1RzoGDyTzYZwXKfBHEQkrey4G8=
```

收到小米通道下行的消息：(需要将sdk日志等级设置到DEBUG)

```
12-09 22:24:34.065 19566-25042/com.xxx D/MPS:MiPushReceiver: onReceiveMessage,msg=[{"f":262,"b":"{\"content\"\ ... ... ,"i":"f__-rnje3_OH74gE|VG0g3kwMnGADAGrXZku1FFW5"}]
```

注：如果控制台配置了小米/华为的信息，app需要加对应的jar包依赖，不然会有crash的风险。

6. 小米系统辅助弹窗

小米设备管控严格，接入推送功能的App进程在后台被清理后，收不到推送通知；接入小米辅助弹窗后，可走MIUI系统的通道进行通知的推送，保证App后台被清理后，仍能收到推送通知（v2.3.0以上支持）。

6.1 客户端

- 该功能的使用需要接入小米推送辅助通道，确保使用最新的小米华为扩展包，具体参考上文；
- 小米辅助弹窗送达的通知展示效果，和普通通知相同；
- 服务端指定小米辅助弹窗通道推送时，一定要指定通知点击后要打开的Activity，该Activity需继承自抽象类MiPushSystemNotificationActivity，否则无法获取到通知的相关信息，并且会影响通知到达率的统计；

  MiPushSystemNotificationActivity中提供抽象方法onMiPushSysNoticeOpened()，实现该方法后可获取到小米辅助弹窗通知的标题、内容和额外参数，在通知点击时触发，原本的通知回调onNotification()和onNotificationOpened()不适用于小米辅助弹窗；

  接入如下所示：

```
import com.alibaba.sdk.android.push.MiPushSystemNotificationActivity;

public class XiaoMiPushActivity extends MiPushSystemNotificationActivity {
static final String TAG = "XiaoMiPushActivity";

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
}

/**
* 实现通知打开回调方法，获取通知相关信息
* @param title 标题
* @param summary 内容
* @param extMap 额外参数
*/
@Override
protected void onMiPushSysNoticeOpened(String title, String summary, Map<String, String> extMap) {
Log.d("OnMiPushSysNoticeOpened, title: " + title + ", content: " + summary + ", extMap: " + extMap);
}
}
```

6.2 服务端

- OpenAPI提供setXiaoMiActivity()接口，参数为小米辅助弹窗通知打开时跳转的Activity，setStoreOffline为true时生效；
- 配置setXiaoMiActivity()后，仅对后台进程被清理的小米设备生效，对非小米设备和在线小米设备不

生效；
- 推送通知标题、内容和额外参数的设置和普通推送接口相同，即setTitle()、setBody()和setAndroidActivity()，额外参数设置中的功能性设置如声音、震动等对小米辅助弹窗通知不生效；

```
PushRequest pushRequest = new PushRequest();
// 其余设置省略
// ...
// 0:表示消息(默认为0), 1:表示通知
pushRequest.setType(1);
// 标题
pushRequest.setTitle(dateFormat.format(new Date()));
// 内容
pushRequest.setBody("PushRequest body");
// 额外参数
pushRequest.setAndroidExtParameters("{\"k1\":\"android\",\"k2\":\"v2\"}");
// 设置小米辅助弹窗打开Activity
pushRequest.setXiaomiActivity("*****");
```

6.3 场景解析

以下几种普通推送结合小米辅助弹窗推送的场景，帮助您快速理解小米辅助弹窗功能的配置。

客户端配置

客户端有Main、Second两个Activity，MainActivity为App打开主页面，SecondActivity extends MiPushSystemNotificationActivity；

- 普通通知回调配置：

```
public class MyMessageReceiver extends MessageReceiver {
/**
* 推送通知的回调方法
* @param context
* @param title
* @param summary
* @param extraMap
*/
@Override
public void onNotification(Context context, String title, String summary, Map<String, String> extraMap) {
Log.d(TAG, "Receive notification, title: " + title + ", content: " + summary + ", extraMap: " + extraMap);
}
}
```

- MainActivity定义：

```
package com.alibaba.push.testdemo;

public class MainActivity extends Activity {
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
Log.d(TAG, "Main");
}
}
```

- SecondActivity定义：

```
package com.alibaba.push.testdemo;

import com.alibaba.sdk.android.push.MiPushSystemNotificationActivity;

public class SecondActivity extends MiPushSystemNotificationActivity {
/**
* 小米辅助弹窗指定打开Activity回调
* @param title 标题
* @param content 内容
* @param extraMap 额外参数
*/
@Override
protected void onMiPushSysNoticeOpened(String title, String content, Map<String, String> extraMap) {
Log.d(TAG, "Receive XiaoMi notification, title: " + title + ", content: " + content + ", extraMap: " + extraMap);
}
}
```

场景1：普通推送打开App + 小米辅助弹窗

服务端配置如下：

```
PushRequest pushRequest = new PushRequest();
// 其余设置省略
// ...
// 0:表示消息(默认为0), 1:表示通知
pushRequest.setType(1);
// 标题
pushRequest.setTitle("hello");
// 内容
pushRequest.setBody("PushRequest body");
// 点击通知后动作,1:打开应用 2: 打开应用Activity 3:打开 url
pushRequest.setAndroidOpenType("1");
// 设置小米辅助弹窗打开Activity
pushRequest.setXiaomiActivity("com.alibaba.push.testdemo.SecondActivity");
// 设定android类型设备通知的扩展属性
pushRequest.setAndroidExtParameters("{\"k1\":\"android\",\"k2\":\"v2\"}");;
```

推送结果：

非小米设备和在线小米设备

- 收到普通推送通道弹出的通知，点击后打开App，进入首页MainActivity，如果设备在前台，保持当前界面不变；

- onNotification()回调输出Receive notification, title: hello, content: PushRequest body, extraMap: {k1=android, k2=v2}；

清理进程后的小米设备

- 小米辅助弹窗通道弹出通知，点击后跳转到SecondActivity；
- onMiPushSysNoticeOpened()回调输出Receive XiaoMi notification, title: hello, content: PushRequest body, extraMap: {k1=android, k2=v2}；

场景2：普通推送打开Activity + 小米辅助弹窗

服务端配置如下：

```
PushRequest pushRequest = new PushRequest();
// 其余设置省略
// ...
// 0:表示消息(默认为0), 1:表示通知
pushRequest.setType(1);
// 标题
pushRequest.setTitle("hello");
// 内容
pushRequest.setBody("PushRequest body");
// 点击通知后动作,1:打开应用 2: 打开应用Activity 3:打开 url
pushRequest.setAndroidOpenType("2");
// 指定普通推送要打开的Activity
pushRequest.setAndroidActivity("com.alibaba.push.testdemo.SecondActivity");
// 设置小米辅助弹窗打开Activity
pushRequest.setXiaomiActivity("com.alibaba.push.testdemo.SecondActivity");
// 设定android类型设备通知的扩展属性
pushRequest.setAndroidExtParameters("{\"k1\":\"android\",\"k2\":\"v2\"}");
```

推送结果：

非小米设备和在线小米设备

- 收到普通推送通道弹出的通知，点击后跳转到SecondActivity；
- onNotification()回调输出Receive notification, title: hello, content: PushRequest body, extraMap: {k1=android, k2=v2}；

清理进程后的小米设备

- 小米辅助弹窗通道弹出通知，点击后跳转到SecondActivity；
- onMiPushSysNoticeOpened()回调输出Receive XiaoMi notification, title: hello, content: PushRequest body, extraMap: {k1=android, k2=v2}；

错误处理

- 调用CloudPushService的相关接口时，如果发生错误，可以在CommonCallback的onFailed()回调中可以获取到errorCode和errorMessage。

常见错误码

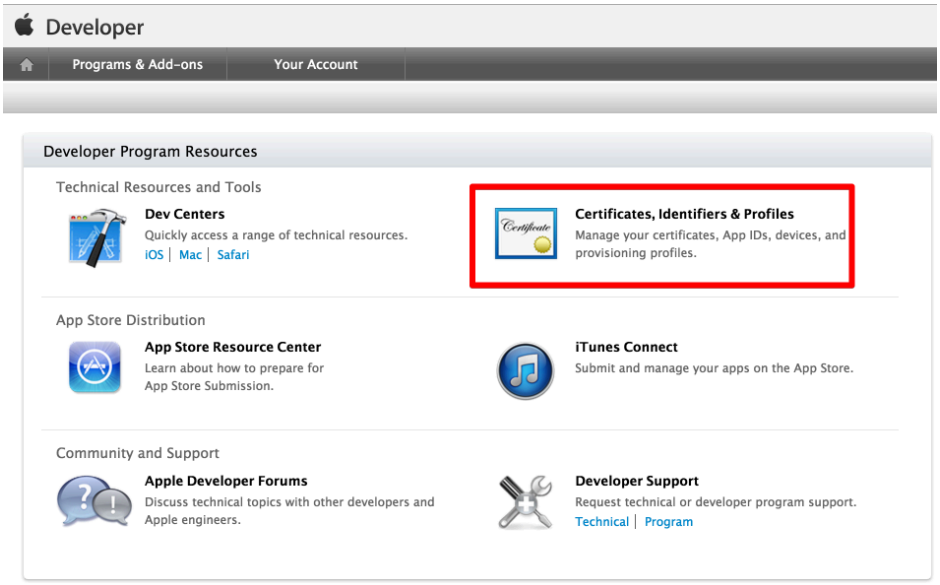
错误名称	错误码（Error Code）	错误描述和解决办法（Error Message）
NO_NETWORK	1101	网络不可用
REG_FAIL	1056	注册/鉴权失败(请检查AppSecret配置)
INVAILD_APPKEY	1052	AppKey不存在
INVAILD_PACKAGENAME	1053	包名与配置的不符
INVAILD_APPSECRET	1054	Appsecret不合法
NETWORK_UNSTABLE	1105	网络不稳定或连接异常
INVAILD_SERVER_RETRUN	1115	不合法的服务端返回（请检查返回是否被篡改）
SYSTEM_UNKNOWN_ERROR	1108	系统未知异常

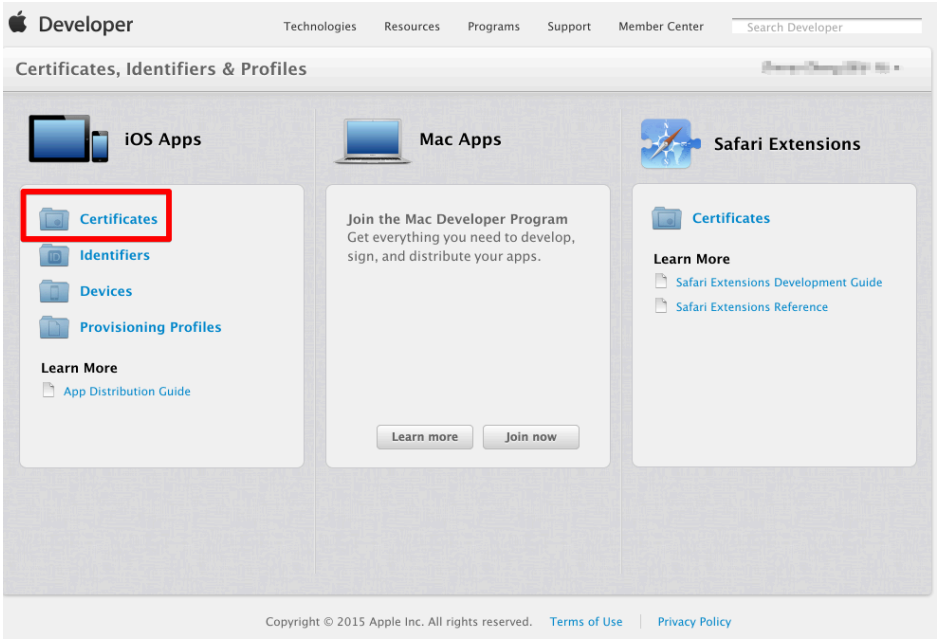
iOS SDK手册

iOS 推送证书设置指南

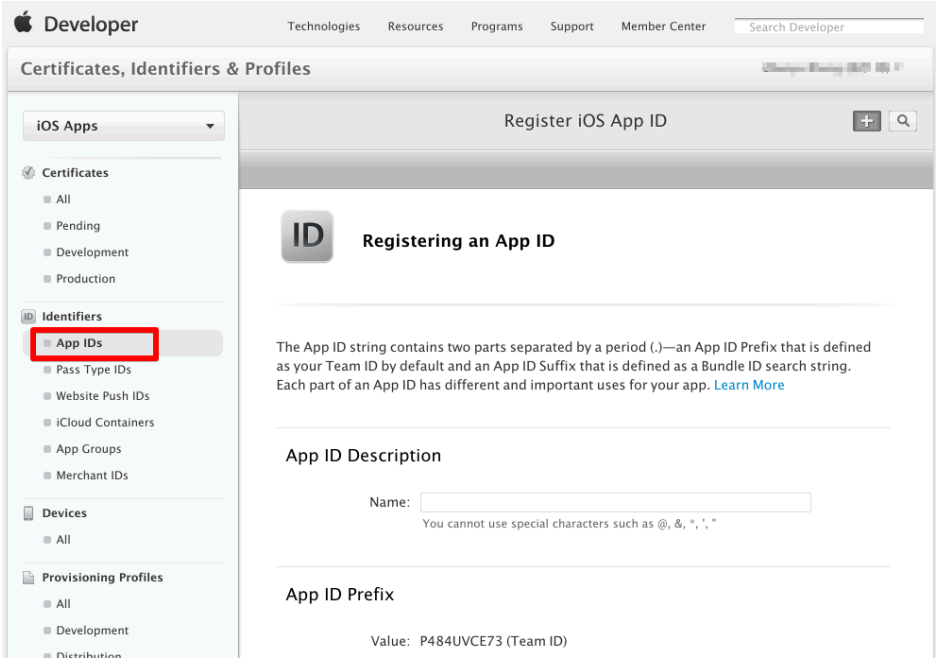
1. 创建应用程序 ID

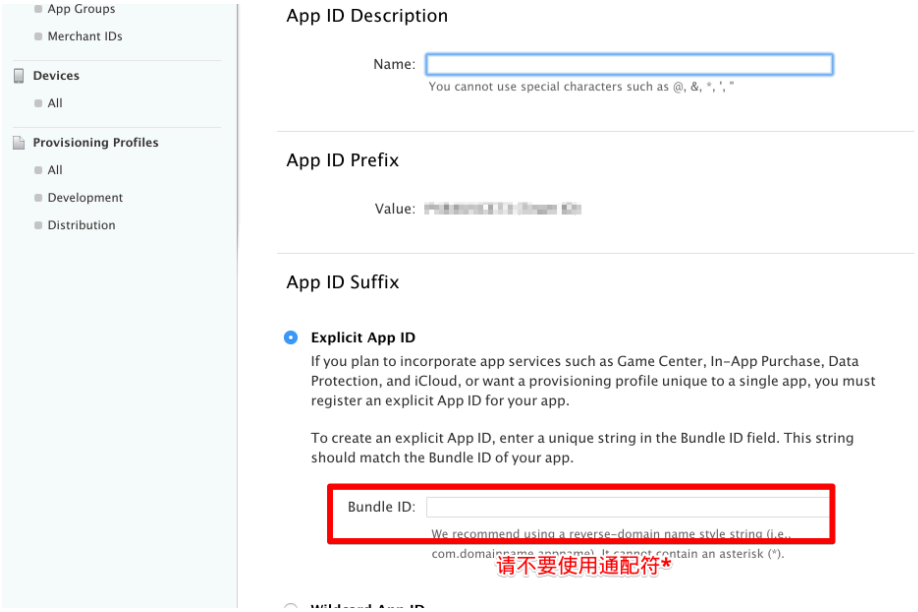
- 苹果开发者平台登录地址


Developer
Programs & Add-ons
Your Account
Developer Program Resources
Technical Resources and Tools
Dev Centers
Quickly access a range of technical resources.
iOS | Mac | Safari
Certificates, Identifiers & Profiles
Manage your certificates, App IDs, devices, and provisioning profiles.
App Store Distribution
App Store Resource Center
Learn about how to prepare for App Store Submission.
iTunes Connect
Submit and manage your apps on the App Store.
Community and Support
Apple Developer Forums
Discuss technical topics with other developers and Apple engineers.
Developer Support
Request technical or developer program support.
Technical | Program

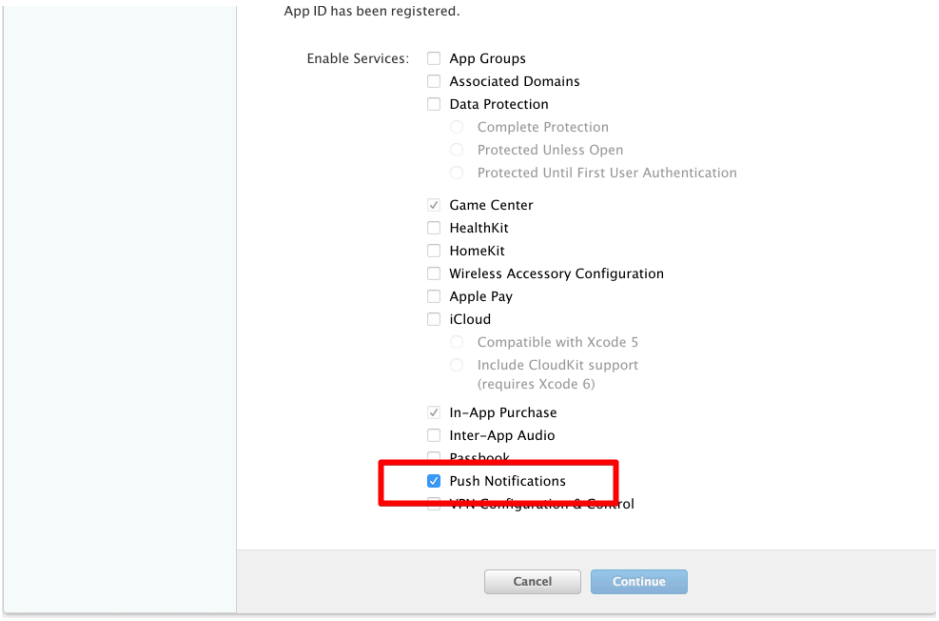

Developer
Technologies
Resources
Programs
Support
Member Center
Search Developer
Certificates, Identifiers & Profiles
iOS Apps
Certificates
Identifiers
Devices
Provisioning Profiles
Learn More
App Distribution Guide
Mac Apps
Join the Mac Developer Program
Get everything you need to develop, sign, and distribute your apps.
Learn more
Join now
Safari Extensions
Certificates
Learn More
Safari Extensions Development Guide
Safari Extensions Reference
Copyright © 2015 Apple Inc. All rights reserved.
Terms of Use
Privacy Policy

2. 创建 App ID


Developer
Technologies
Resources
Programs
Support
Member Center
Search Developer
Certificates, Identifiers & Profiles
iOS Apps
Certificates
All
Pending
Development
Production
Identifiers
App IDs
Pass Type IDs
Website Push IDs
iCloud Containers
App Groups
Merchant IDs
Devices
All
Provisioning Profiles
All
Development
Register iOS App ID
ID
Registering an App ID
The App ID string contains two parts separated by a period (.)—an App ID Prefix that is defined as your Team ID by default and an App ID Suffix that is defined as a Bundle ID search string. Each part of an App ID has different and important uses for your app. Learn More
App ID Description
Name:
You cannot use special characters such as @, &, *, ', "
App ID Prefix
Value: P484UVCE73 (Team ID)

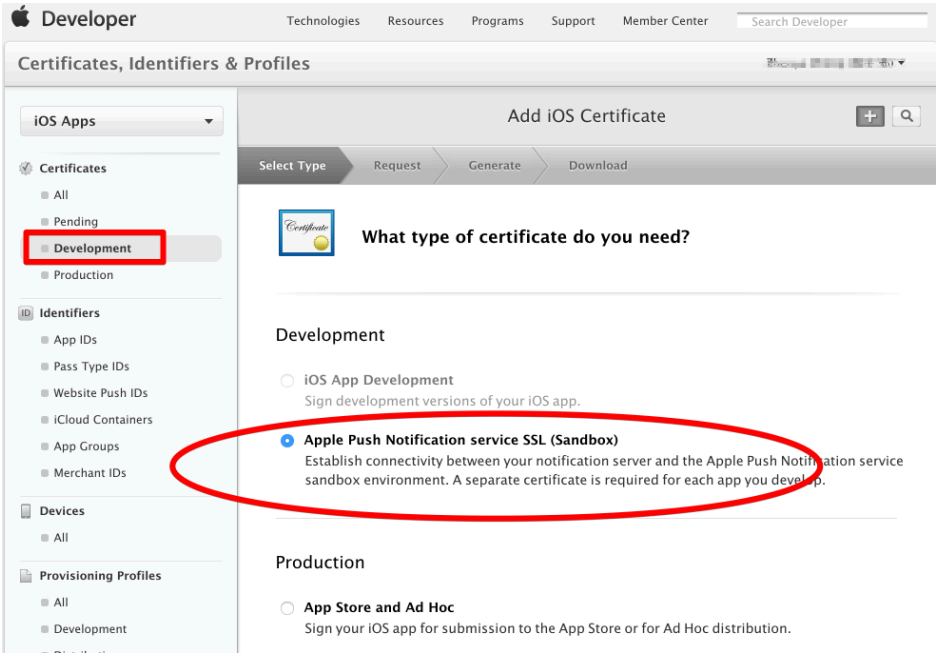

App Groups
Merchant IDs
Devices
All
Provisioning Profiles
All
Development
Distribution
App ID Description
Name:
You cannot use special characters such as @, &, *, ', "
App ID Prefix
Value:
App ID Suffix
Explicit App ID
If you plan to incorporate app services such as Game Center, In-App Purchase, Data Protection, and iCloud, or want a provisioning profile unique to a single app, you must register an explicit App ID for your app.
To create an explicit App ID, enter a unique string in the Bundle ID field. This string should match the Bundle ID of your app.
Bundle ID:
We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (*).
请不要使用通配符*

3. 配置推送功能

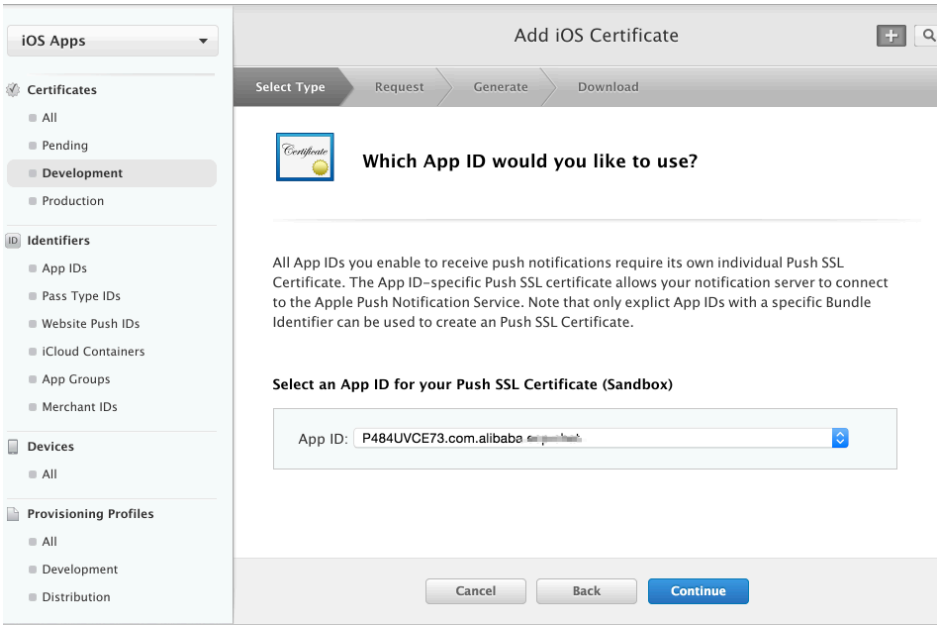


4. 配置推送证书

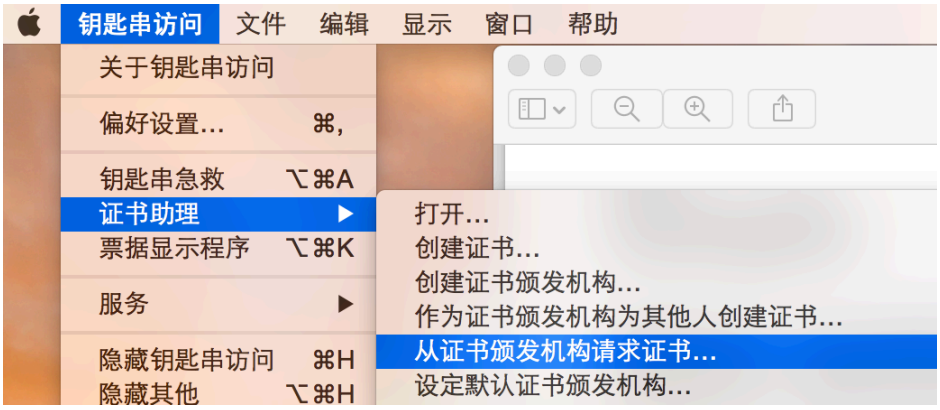
- 这里只演示配置开发证书，如是生产环境，请选择相应的生产环境证书。



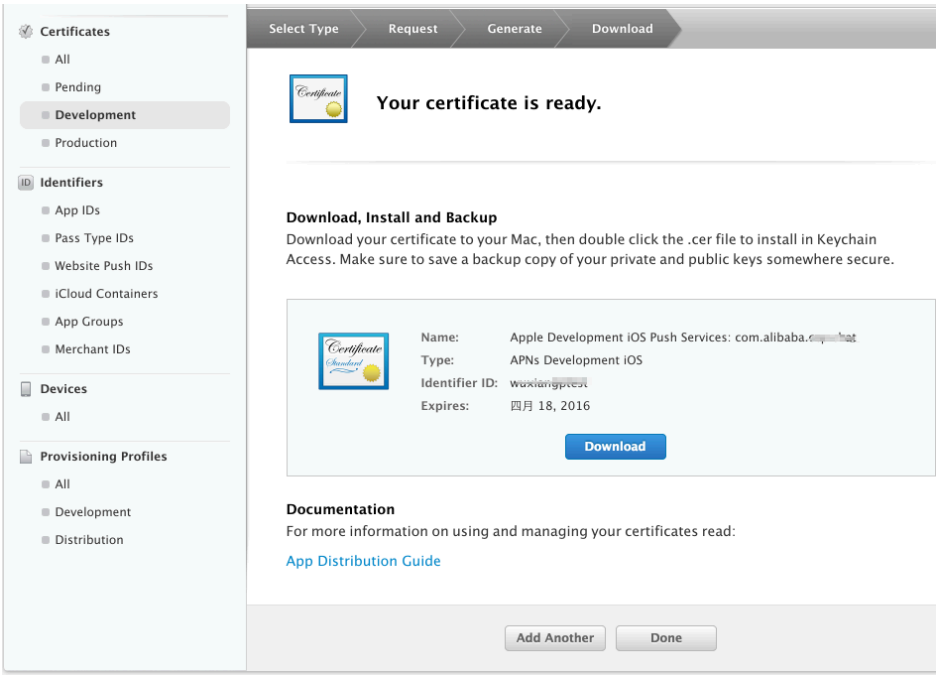
- 选择刚配好的App ID；



- 上传本地生成的CSR文件；

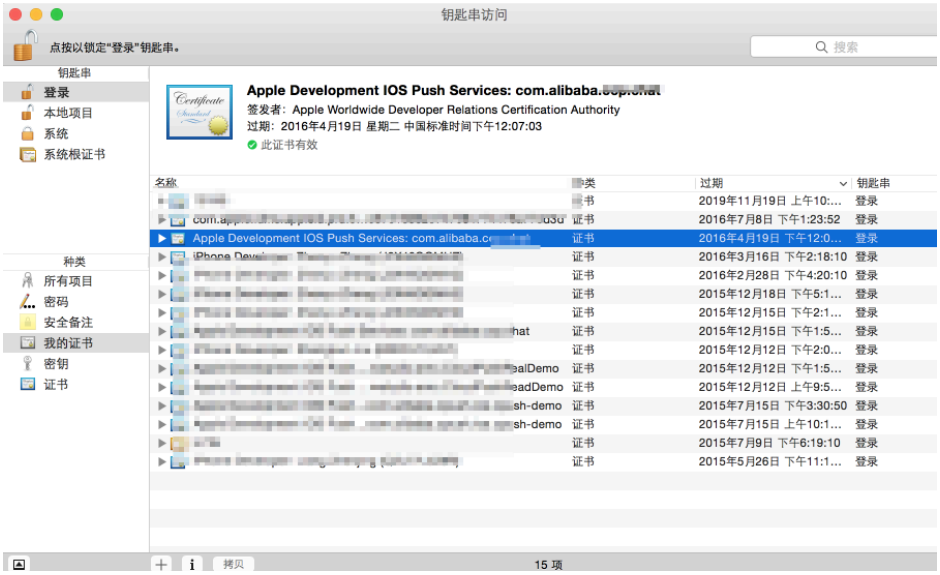


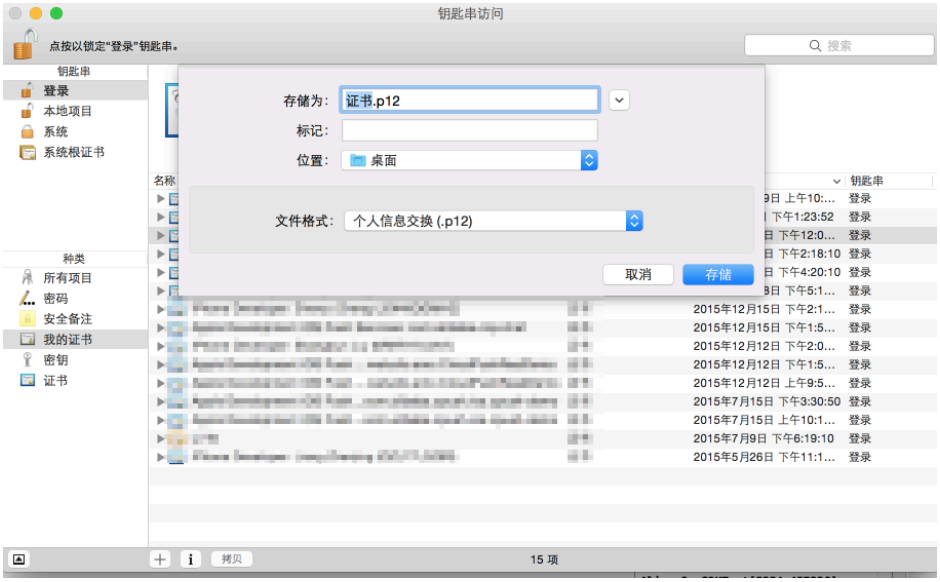
5. 下载推送证书



6. 安装推送证书并导出

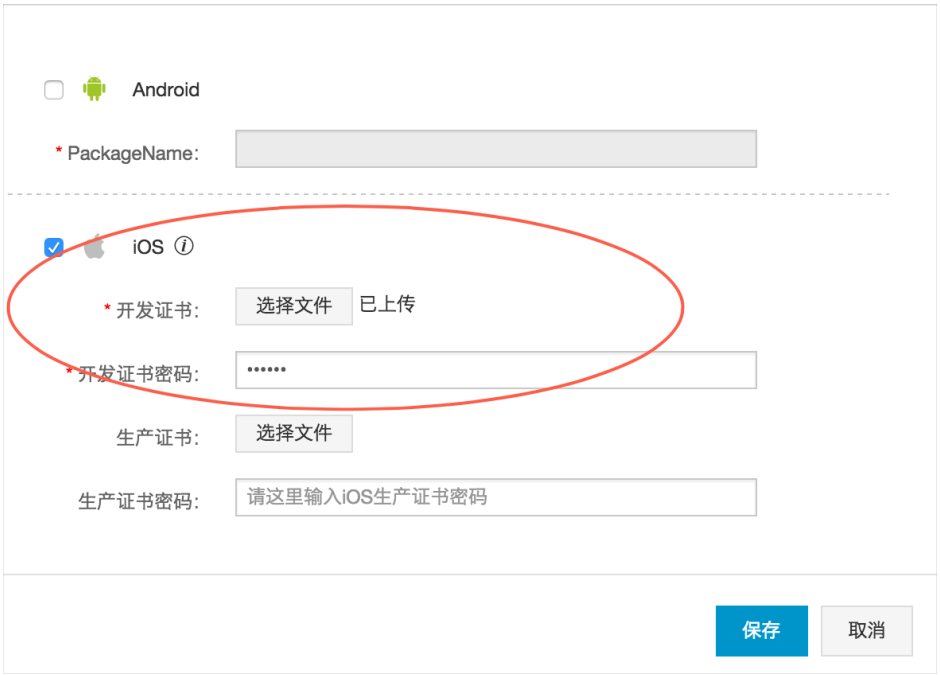
- 双击安装推送证书到本地；
- 在本地KeyChain（钥匙串访问）的我的证书中查看推送证书，并选择导出；





- 【注意】将推送证书上传到阿里云推送控制台时，需要填入证书密码，导出推送证书时一定要填写密码。

7. 上传证书至阿里移动推送



- 如果您想在生产环境测试iOS推送通知功能，请参考：
  https://help.aliyun.com/knowledge_detail/13382190.html?spm=0.0.0.0.GDdjXG。

- 使用前必读：移动推送名词解释&约束

1. 创建应用

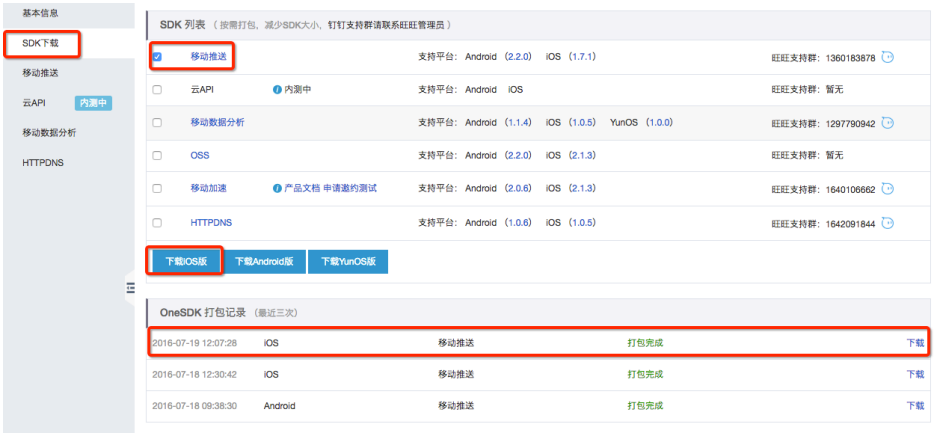
到阿里云移动推送控制台创建应用，应用创建完成后，进入移动推送相关模块进行设置，具体操作请参见 创建APP。

- iOS应用推送需配置开发环境/生产环境推送证书，具体可参见iOS推送证书设置。

2. SDK下载和集成

2.1 SDK下载

- 在移动推送控制台进行SDK下载；



2.2 SDK引用说明

2.2.1 公共包依赖

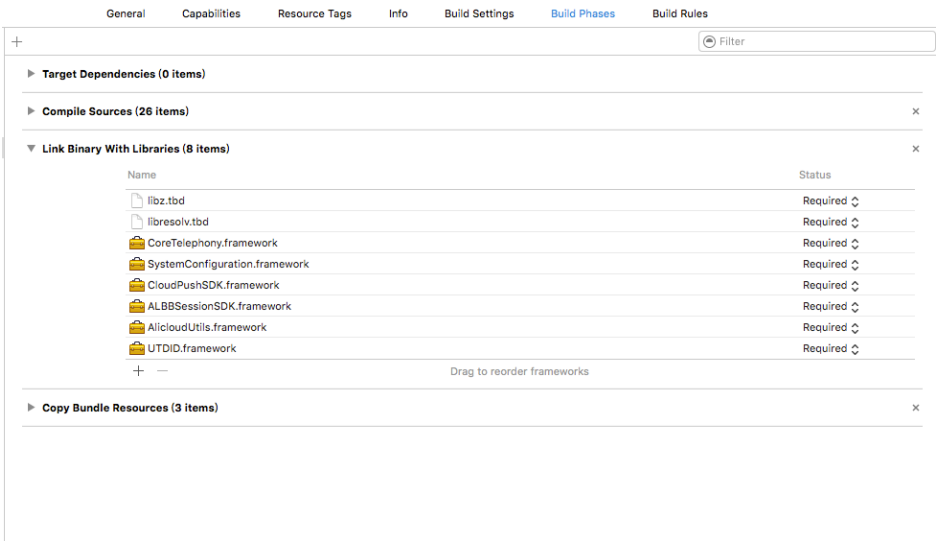
- libz.tbd
- libresolv.tbd
- CoreTelephony.framework
- SystemConfiguration.framework

2.2.2 SDK目录结构

- CloudPushSDK.framework
- ALBBSessionSDK.framework
- AlicloudUtils.framework
- UTDID.framework

2.2.3 引入Framework

- Xcode中，直接把下载SDK目录中的framework拖入对应Target下即可，在弹出框勾选Copy items if needed。
- 在 Build Phases -> Link Binary With Libraries中，引入2.2.1列出的公共包；



2.2.4 工程引入头文件

```
#import <CloudPushSDK/CloudPushSDK.h>
```

2.2.5 说明

- 应用的targets -> Build Settings -> Linking -> Other Linker Flags，请加上-ObjC这个属性，否则推送服务无法正常使用
- iOS 9引入了App Transport Security(ATS)机制，可参考App Transport Security(ATS)机制。 要求App内访问的网络必须使用HTTPS协议，现在阿里移动推送使用的是HTTP安全的加签访问机制来保证安全，未使用HTTPS，所以解决方法如下：
    - 在工程的Info.plist中添加NSAppTransportSecurity，添加后会自动转变为App Transport Security Setting，右击该选项选择Show Raw Key/Values，可显示原本添加名；
    - 将该选项点击铺开(黑色三角指向下)，右击选择Add Row，会自动显示Item Allow Arbitrary Loads，将Value值设为YES；
    - Build Setting中的Enable Bitcode需要设置为NO。

3. Push SDK使用

- 请参照以下代码完成SDK的初始化；

```
- (void)initCloudPush {
// SDK初始化
[CloudPushSDK asyncInit:@"*****" appSecret:@"*****" callback:^(CloudPushCallbackResult *res) {
if (res.success) {
NSLog(@"Push SDK init success, deviceId: %@.", [CloudPushSDK getDeviceId]);
} else {
NSLog(@"Push SDK init failed, error: %@", res.error);
}
```

```
}];
}
```

- 向苹果APNs注册获取deviceToken并上报到阿里云推送服务器；

```
/**
* 注册苹果推送，获取deviceToken用于推送
*
* @param application
*/
- (void)registerAPNS:(UIApplication *)application {
if ([[[UIDevice currentDevice] systemVersion] floatValue] >= 8.0) {
// iOS 8 Notifications
[application registerUserNotificationSettings:
[UIUserNotificationSettings settingsForTypes:
(UIUserNotificationTypeSound | UIUserNotificationTypeAlert | UIUserNotificationTypeBadge)
categories:nil]];
[application registerForRemoteNotifications];
}
else {
// iOS < 8 Notifications
[[UIApplication sharedApplication] registerForRemoteNotificationTypes:
(UIRemoteNotificationTypeAlert | UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeSound)];
}
}

/*
* 苹果推送注册成功回调，将苹果返回的deviceToken上传到CloudPush服务器
*/
- (void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData
*)deviceToken {
[CloudPushSDK registerDevice:deviceToken withCallback:^(CloudPushCallbackResult *res) {
if (res.success) {
NSLog(@"Register deviceToken success.");
} else {
NSLog(@"Register deviceToken failed, error: %@", res.error);
}
}];
}

/*
* 苹果推送注册失败回调
*/
- (void)application:(UIApplication *)application didFailToRegisterForRemoteNotificationsWithError:(NSError *)error {
NSLog(@"didFailToRegisterForRemoteNotificationsWithError %@", error);
}
```

- 推送消息到来监听；

```
/**
* 注册推送消息到来监听
*/
- (void)registerMessageReceive {
```

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMessageReceived:)
name:@"CCPDidReceiveMessageNotification"
object:nil];
}

/**
* 处理到来推送消息
*
* @param notification
*/
- (void)onMessageReceived:(NSNotification *)notification {
CCPSysMessage *message = [notification object];
NSString *title = [[NSString alloc] initWithData:message.title encoding:NSUTF8StringEncoding];
NSString *body = [[NSString alloc] initWithData:message.body encoding:NSUTF8StringEncoding];
NSLog(@"Receive message title: %@, content: %@.", title, body);
}
```

通知打开监听

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
// 点击通知将App从关闭状态启动时，将通知打开回执上报
[CloudPushSDK handleLaunching:launchOptions];
return YES;
}

/*
* App处于启动状态时，通知打开回调
*/
- (void)application:(UIApplication*)application didReceiveRemoteNotification:(NSDictionary*)userInfo {
NSLog(@"Receive one notification.");
// 取得APNS通知内容
NSDictionary *aps = [userInfo valueForKey:@"aps"];
// 内容
NSString *content = [aps valueForKey:@"alert"];
// badge数量
NSInteger badge = [[aps valueForKey:@"badge"] integerValue];
// 播放声音
NSString *sound = [aps valueForKey:@"sound"];
// 取得Extras字段内容
NSString *Extras = [userInfo valueForKey:@"Extras"]; //服务端中Extras字段，key是自己定义的
NSLog(@"content = [%@], badge = [%ld], sound = [%@], Extras = [%@]", content, (long)badge, sound, Extras);
// iOS badge 清0
application.applicationIconBadgeNumber = 0;
// 通知打开回执上报
[CloudPushSDK handleReceiveRemoteNotification:userInfo];
}
```

如果使用推送模块，请参考移动推送常见问题

iOS API

iOS SDK最新版本v1.7.1。

1. CloudPushSDK接口

打开调试日志

- 打开推送SDK日志；
- 测试时可选择打开，App上线后建议关闭。

```
+ (void)turnOnDebug;
```

获取SDK版本号

- 版本号也可以在CloudPushSDK.h中查看。

返回

- SDK版本号。

```
+ (NSString *)getVersion;
```

获取推送通道状态

- 查询推送应用内通道状态。

返回

- 推送通道是否打开。

```
+ (BOOL)isChannelOpened;
```

获取设备deviceId

- deviceId为阿里云移动推送过程中对设备的唯一标识；
- 推送通道正确打开后，可以获取。

返回

- 设备唯一标识deviceId。

```
+ (NSString *)getDeviceId;
```

绑定账号

- 将应用内账号和推送通道相关联，可以实现按账号的定点消息推送；
- 设备只能绑定一个账号，多次绑定操作仅最后一个生效；
- 账户名设置支持32字节。

参数

- account 绑定账号名
- callback 回调

```
+ (void)bindAccount:(NSString *)account
withCallback:(CallbackHandler)callback;
```

解绑账号

- 将应用内账号和推送通道取消关联。

参数

- callback 回调

```
+ (void)unbindAccount:(CallbackHandler)callback;
```

绑定标签

- 绑定标签到指定目标；
- 支持向设备、账号和别名绑定标签，绑定类型由参数target指定；
- 绑定标签后，第二天服务端可按该标签推送，即（T + 1）天生效；
- App最多支持绑定128个标签【请谨慎使用，避免标签绑定达到上限】，单个标签最大支持40字节。

参数

- target 目标类型，1：本设备；2：本设备绑定账号；3：别名
- tags 标签（数组输入）
- alias 别名（仅当target = 3时生效）
- callback 回调

```
+ (void)bindTag:(int)target
```

```
withTags:(NSArray *)tags
withAlias:(NSString *)alias
withCallback:(CallbackHandler)callback;
```

解绑标签

- 解绑指定目标标签；
- 支持解绑设备、账号和别名标签，解绑类型由参数target指定；
- 解绑标签后，当天服务端可继续按该标签推送，解绑生效时间在第二天，即（T + 1）天生效；
- 解绑标签不等同于删除标签，目前不支持标签的删除。

参数

- target 目标类型，1：本设备；2：本设备绑定账号；3：别名
- tags 标签（数组输入）
- alias 别名（仅当target = 3时生效）
- callback 回调

```
+ (void)unbindTag:(int)target
withTags:(NSArray *)tags
withAlias:(NSString *)alias
withCallback:(CallbackHandler)callback;
```

查询标签

- 查询目标绑定标签，当前仅支持查询设备标签；
- 查询结果可从callback的data中获取；
- 标签绑定成功后即可查询。

参数

- target 目标类型，1：本设备
- callback 回调

```
+ (void)listTags:(int)target
withCallback:(CallbackHandler)callback;
```

添加别名

- 设备添加别名；
- 别名支持128字节。

参数

- alias 别名
- callback 回调

```
+ (void)addAlias:(NSString *)alias
withCallback:(CallbackHandler)callback;
```

删除别名

- 删除设备别名；
- 支持删除指定别名和删除全部别名（alias为nil or length = 0）。

参数

- alias 别名（alias为nil or length = 0时，删除设备全部别名）
- callback 回调

```
+ (void)removeAlias:(NSString *)alias
withCallback:(CallbackHandler)callback;
```

查询别名

- 查询设备别名；
- 查询结果可从callback的data中获取。

参数

- callback 回调

```
+ (void)listAliases:(CallbackHandler)callback;
```

上报设备deviceToken

- 向阿里云推送注册该设备的deviceToken；
- 可在APNs注册成功回调中调用该接口。

参数

- deviceToken 苹果APNs服务器返回的deviceToken
- callback 回调

```
+ (void)registerDevice:(NSData *)deviceToken
withCallback:(CallbackHandler)callback;
```

获取设备deviceToken

- 返回获取APNs返回的deviceToken；
- 调用registerDevice()接口后可获取。

返回

- 设备deviceToken。

```
+ (NSString *)getApnsDeviceToken;
```

上报"通知点击事件"（App处于关闭状态）

- 上报"通知点击事件"到推送服务器；
- 点击通知将App从关闭状态拉起时，在didFinishLaunchingWithOptions回调中调用该接口。

参数

- launchOptions didFinishLaunchingWithOptions 回调中的launchOptions参数

```
+ (void)handleLaunching:(NSDictionary *)launchOptions;
```

上报"通知点击事件"（App处于打开状态）

- 上报"通知点击事件"到推送服务器；
- App处于打开状态（前台 or 后台），在didReceiveRemoteNotification回调中调用该接口；
- App处于前台，通知不弹窗，直接触发回调；App处于后台，通知弹窗并触发回调。

参数

- userInfo didReceiveRemoteNotification回调中的参数userInfo

```
+ (void)handleReceiveRemoteNotification:(NSDictionary *)userInfo;
```

2. 推送通道监听接口

监听推送通道建立

- 通知中心注册事件名为CCPDidChannelConnectedSuccess的广播监听；
- 推送通道成功建立后，发出事件名为CCPDidChannelConnectedSuccess的广播通知。

```
- (void)listenerOnChannelOpened {
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onChannelOpened:)
name:@"CCPDidChannelConnectedSuccess"
object:nil];
}

// 通道打开通知
- (void)onChannelOpened:(NSNotification *)notification {
}
```

消息接收监听

- 通知中心注册事件名为CCPDidReceiveMessageNotification的广播监听；
- 推送通道成功建立后，发出事件名为CCPDidReceiveMessageNotification的广播通知。

```
- (void) registerMessageReceive {
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onMessageReceived:)
name:@"CCPDidReceiveMessageNotification"
object:nil];
}

- (void)onMessageReceived:(NSNotification *)notification {
CCPSysMessage *message = [notification object];
NSString *title = [[NSString alloc] initWithData:message.title encoding:NSUTF8StringEncoding];
NSString *body = [[NSString alloc] initWithData:message.body encoding:NSUTF8StringEncoding];
NSLog(@"Receive message title: %@, content: %@.", title, body);
}
```

通知打开监听

- App处于关闭状态时，点击打开通知；

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
// 点击通知将App从关闭状态启动时，将通知打开回执上报
[CloudPushSDK handleLaunching:launchOptions];
return YES;
}
```

- App处于打开状态时，点击打开通知；

```
- (void)application:(UIApplication*)application didReceiveRemoteNotification:(NSDictionary*)userInfo {
NSLog(@"Receive one notification.");
// 取得APNS通知内容
NSDictionary *aps = [userInfo valueForKey:@"aps"];
// 内容
NSString *content = [aps valueForKey:@"alert"];
// badge数量
```

```
NSInteger badge = [[aps valueForKey:@"badge"] integerValue];
// 播放声音
NSString *sound = [aps valueForKey:@"sound"];
// 取得Extras字段内容
NSString *Extras = [userInfo valueForKey:@"Extras"]; //服务端中Extras字段，key是自己定义的
NSLog(@"content = [%@], badge = [%ld], sound = [%@], Extras = [%@]", content, (long)badge, sound, Extras);
// iOS badge 清0
application.applicationIconBadgeNumber = 0;
// 通知打开回执上报
[CloudPushSDK handleReceiveRemoteNotification:userInfo];
}
```

错误处理

- 调用CloudPushSDK的相关接口时，如果发生错误，可以从CallbackHandler回调对象中获取错误码和错误描述等信息。
- CallbackHandler定义如下，可从回调处理对象res中获取
    - success（接口调用是否成功）；
    - data（调用成功后返回相关数据）；
    - error（错误信息描述）。

```
typedef void (^CallbackHandler)(CloudPushCallbackResult *res);
```

常见错误码

错误名称	错误码	错误描述
INIT_INVALID_APPKEY_CODE	1011	appKey获取失败
INIT_INVALID_APPSECRET_CODE	1012	appSecret获取失败
INIT_SESSION_FAILED_CODE	1013	Session初始化失败
INIT_AS_ERROR_CODE	1014	连接AS错误
INIT_SID_ERROR_CODE	1015	SID缺失
TAG_INPUT_INVALID_CODE	2001	标签输入为空
TAG_APPID_INVALID_CODE	2002	appId错误
TAG_RPC_REQUEST_FAILED_CODE	2003	标签请求错误
ACCOUNT_INVALID_ACCOUNT_CODE	3001	account输入为空
ACCOUNT_CHANNEL_CLOSED_CODE	3002	推送通道关闭
ACCOUNT_REQUEST_TIMEO	3003	请求超时

UT_CODE		
ACCOUNT_ENCODER_STATUS_ERROR_CODE	3004	状态码错误
ALIAS_INPUT_INVALID_CODE	4001	别名输入为空
VIP_REQ_HTTP_ERROR_CODE	5001	VIP请求状态码错误
VIP_REQ_CONNECTION_ERROR_CODE	5002	VIP请求连接错误
VIP_REQ_SERVER_ERROR_CODE	5003	VIP请求服务错误
OTHER_ERROR_INVLIAD_PARA_CODE	6001	其他输入错误