移动推送

SDK 文档

产品文档

腾讯云

文档目录

SDK 文档
Android 接入指南
简介

最近更新时间：2024-01-16 17:39:39

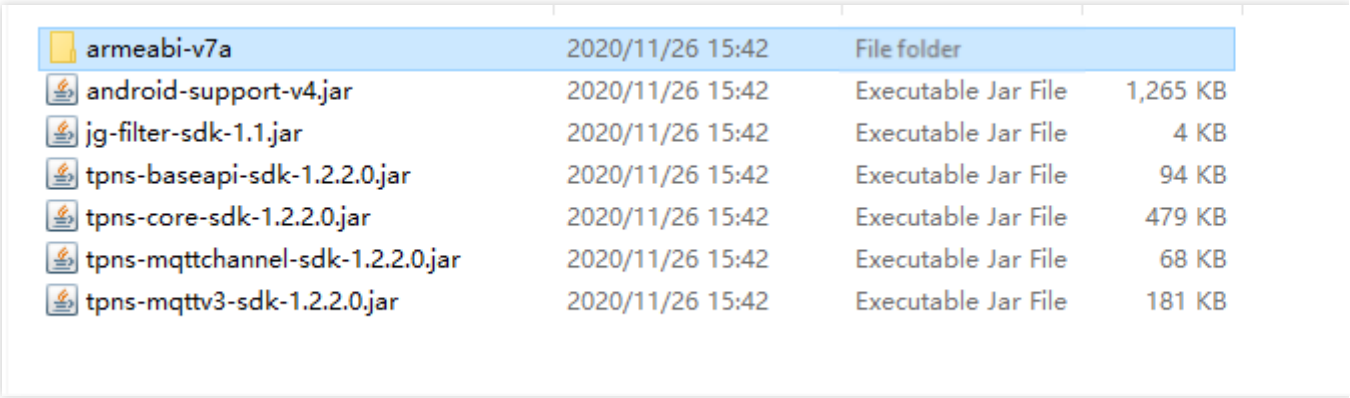
移动推送（Tencent Push Notification Service）是一款专业的移动 App 推送平台，支持百亿级的通知和消息推送，秒级触达移动用户，现已全面支持 Android 和 iOS 两大主流平台，开发者可以方便地通过嵌入 SDK，通过 API 调用或 Web 端可视化操作，实现对特定用户推送，大幅提升用户活跃度，有效唤醒沉睡用户，并实时查看推送效果。

功能说明

Android SDK 是移动推送服务为客户端实现消息推送而提供给开发者的接口，主要负责完成以下功能：
提供通知和消息二种推送形式，方便用户使用。
账号、标签与设备的绑定接口，以便开发者实现特定群组的消息推送，丰富推送方式。
点击量上报，统计消息被用户点击的次数。
提供多厂商通道集成功能，方便用户集成多厂商推送。

SDK 说明

从官网上下载下来的包，解压后内容如下：

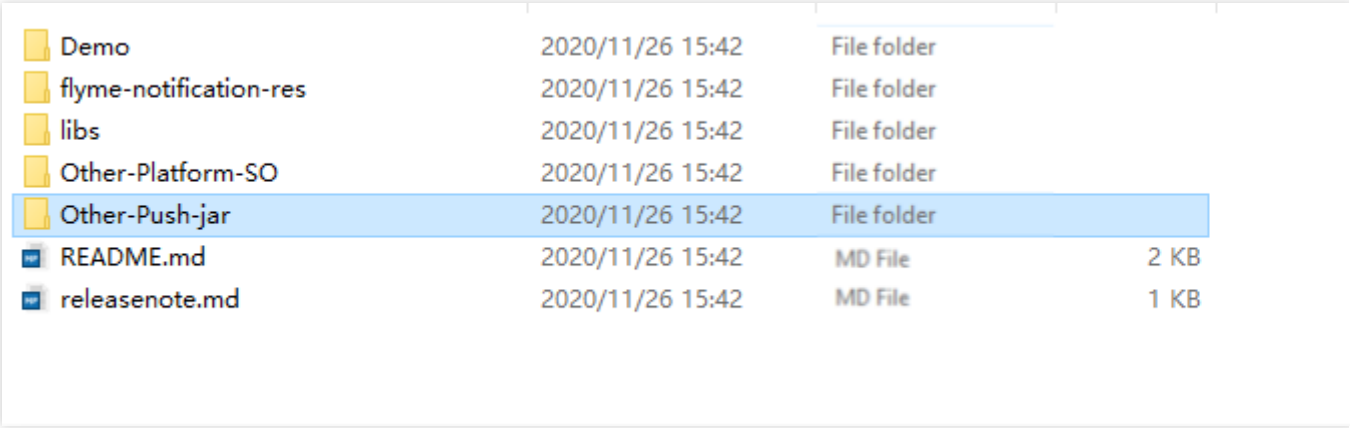


解压后根目录五个文件夹内容：

Demo 文件夹：移动推送官方 Demo 程序，用户可以参考相关配置。

flyme-notification-res 文件夹：魅族推送通道的资源文件，用于低版本魅族手机兼容，Flyme 6.0 及以下版本的魅族手机用户需要将对应的文件复制到 App 的 res 目录下。
libs 文件夹：包含移动推送的 jar 和 so 文件。
Other-Platform-SO 文件夹：包含其它不常用 CPU 架构的 so 文件。
Other-Push-jar 文件夹：移动推送封装的华为、魅族、小米、OPPO、VIVO、FCM 的 jar 包。

libs 目录详细介绍

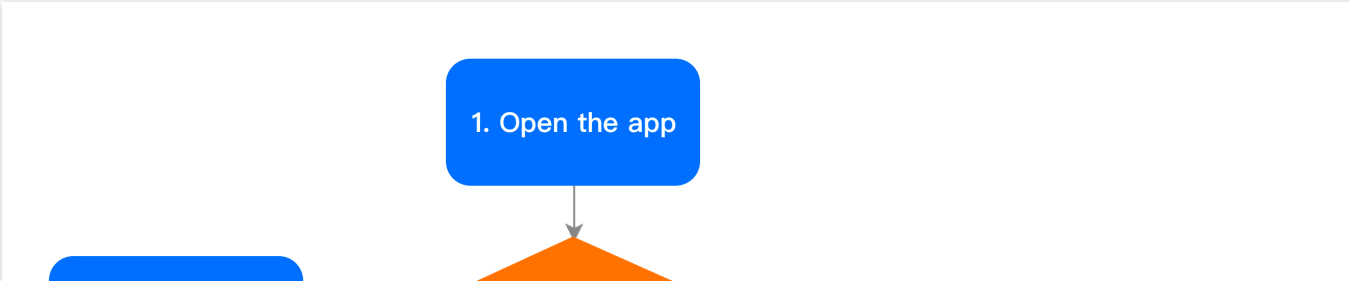


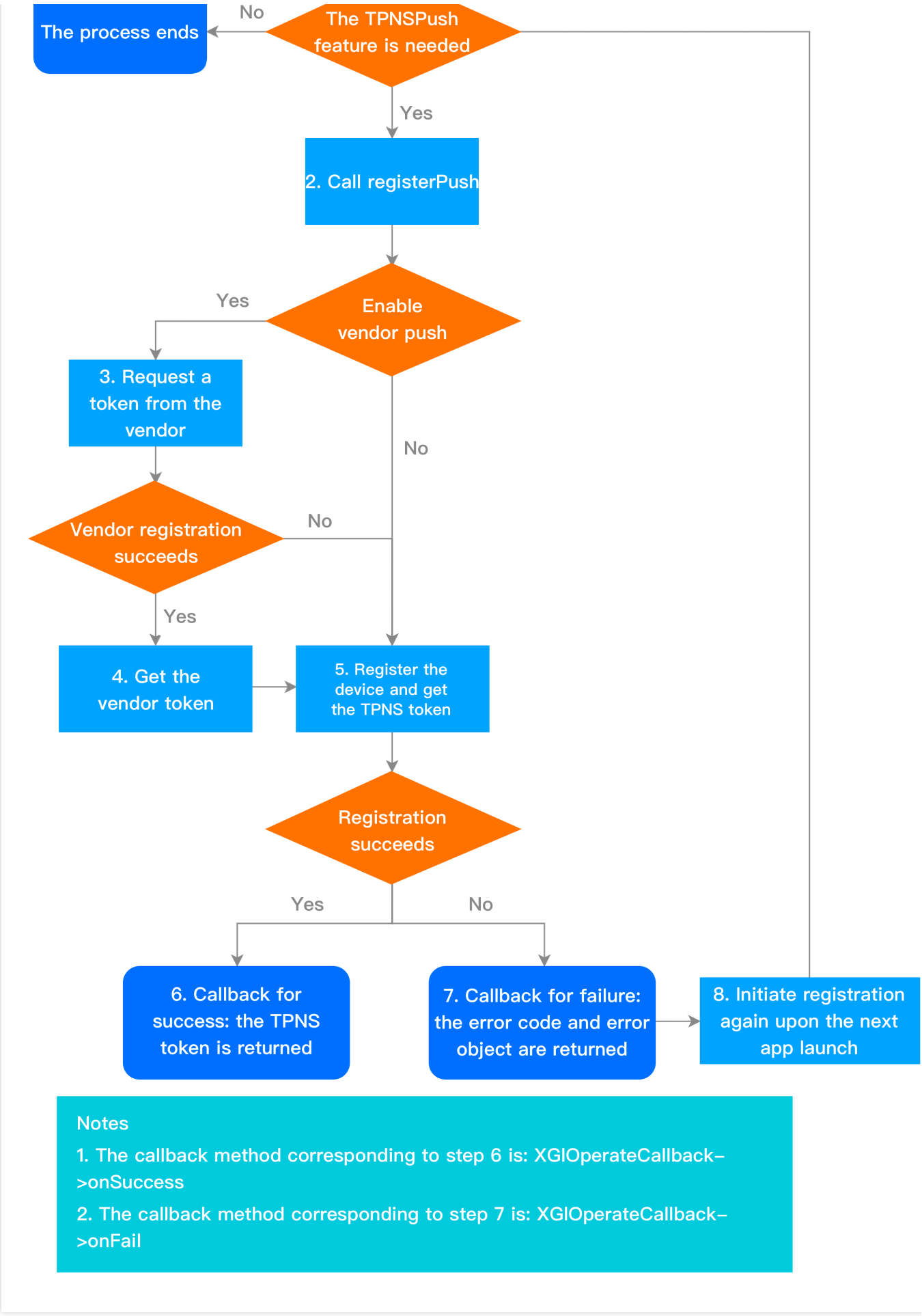
android-support-v4.jar： 谷歌推出的兼容包，兼容 Android1.6 以上的系统。
jg-filter-sdk-1.1.jar：金刚扫描的 jar 包，使用腾讯 SDK 的产品必须带上。
tpns-baseapi-sdk-x.x.x.x.jar：移动推送提供的部分底层公共 API。
tpns-core-sdk-x.x.x.x.jar：移动推送 SDK 核心模块代码，存放所有对外的类、API 和组件。
tpns-mqttchannel-sdk-x.x.x.x.jar：移动推送上层实现基于 MQTT 协议的通信功能，长连接独立在一个进程中。
tpns-mqttv3-sdk-x.x.x.x.jar：移动推送改造后的 MQTT 协议包，提供多厂商通道集成功能，方便用户集成多厂商推送。

常用场景流程说明

设备注册流程

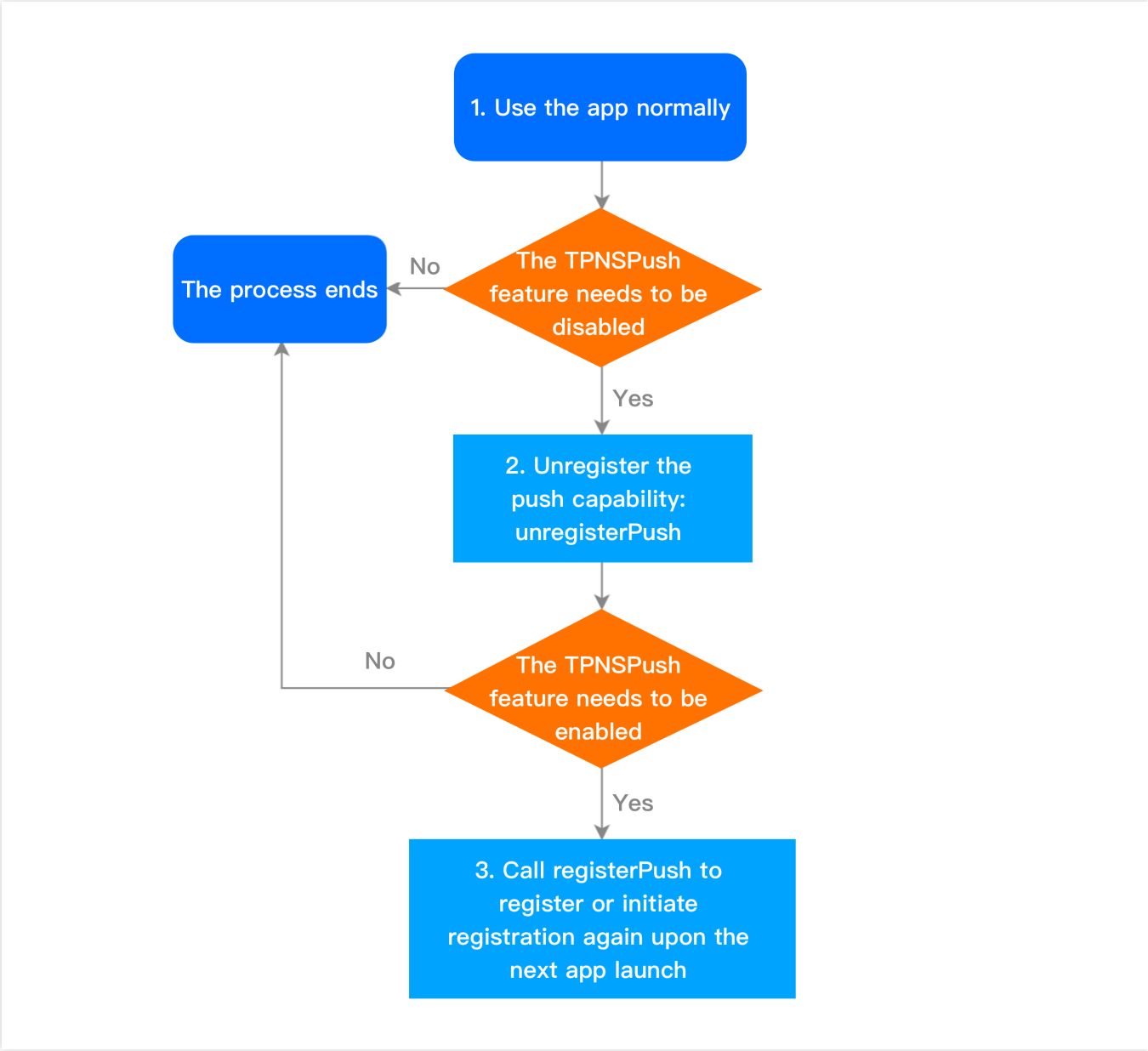
下图为设备注册相关流程，具体接口方法请查看 启动与注册。




The process ends
No
The TPNSPush feature is needed
Yes
2. Call registerPush
Yes
Enable vendor push
3. Request a token from the vendor
No
Vendor registration succeeds
No
Yes
4. Get the vendor token
5. Register the device and get the TPNS token
Registration succeeds
Yes
No
6. Callback for success: the TPNS token is returned
7. Callback for failure: the error code and error object are returned
8. Initiate registration again upon the next app launch
Notes
1. The callback method corresponding to step 6 is: XGIOperateCallback->onSuccess
2. The callback method corresponding to step 7 is: XGIOperateCallback->onFail

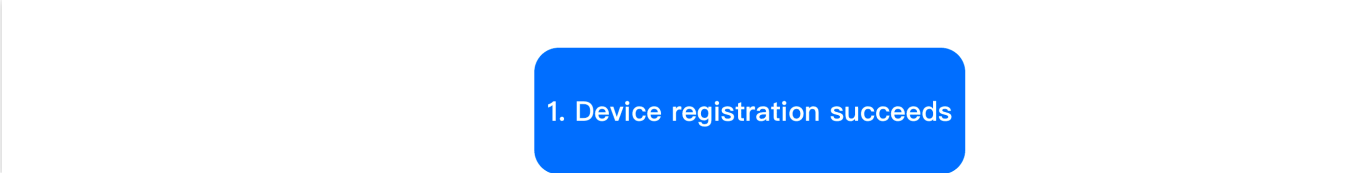
设备反注册流程

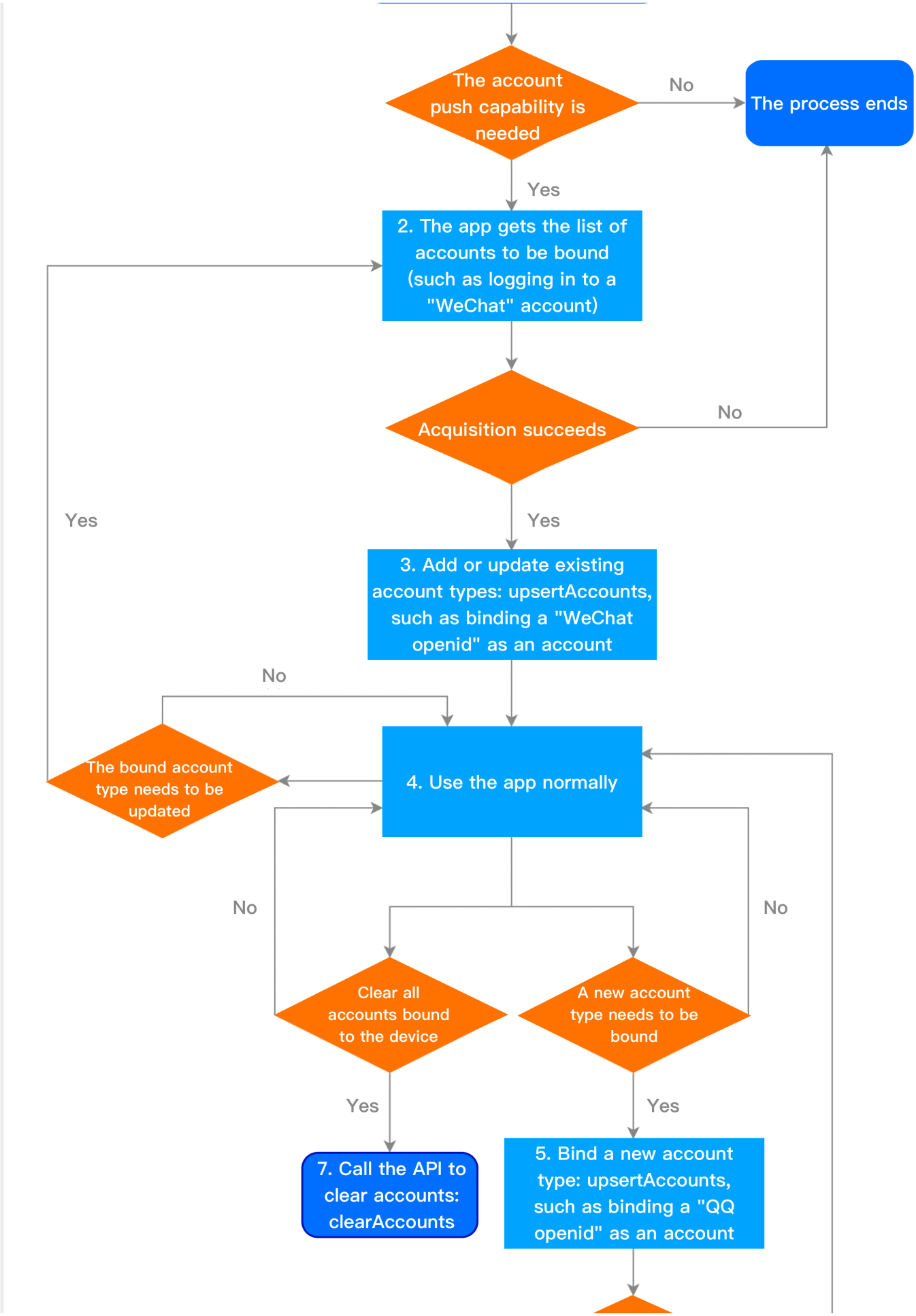
下图为设备反注册相关流程，具体接口方法请查看 反注册。

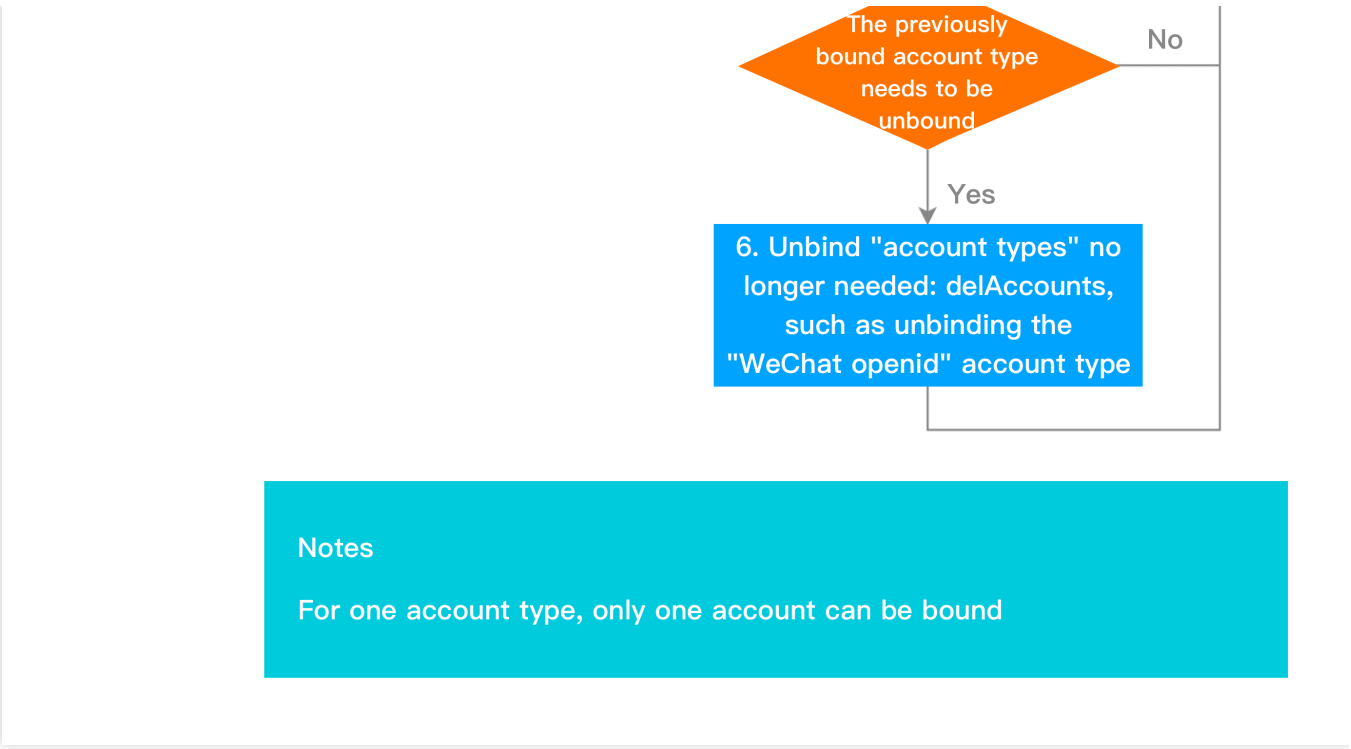


账号相关流程

下图为账号相关流程，具体接口方法请查看 账号管理。

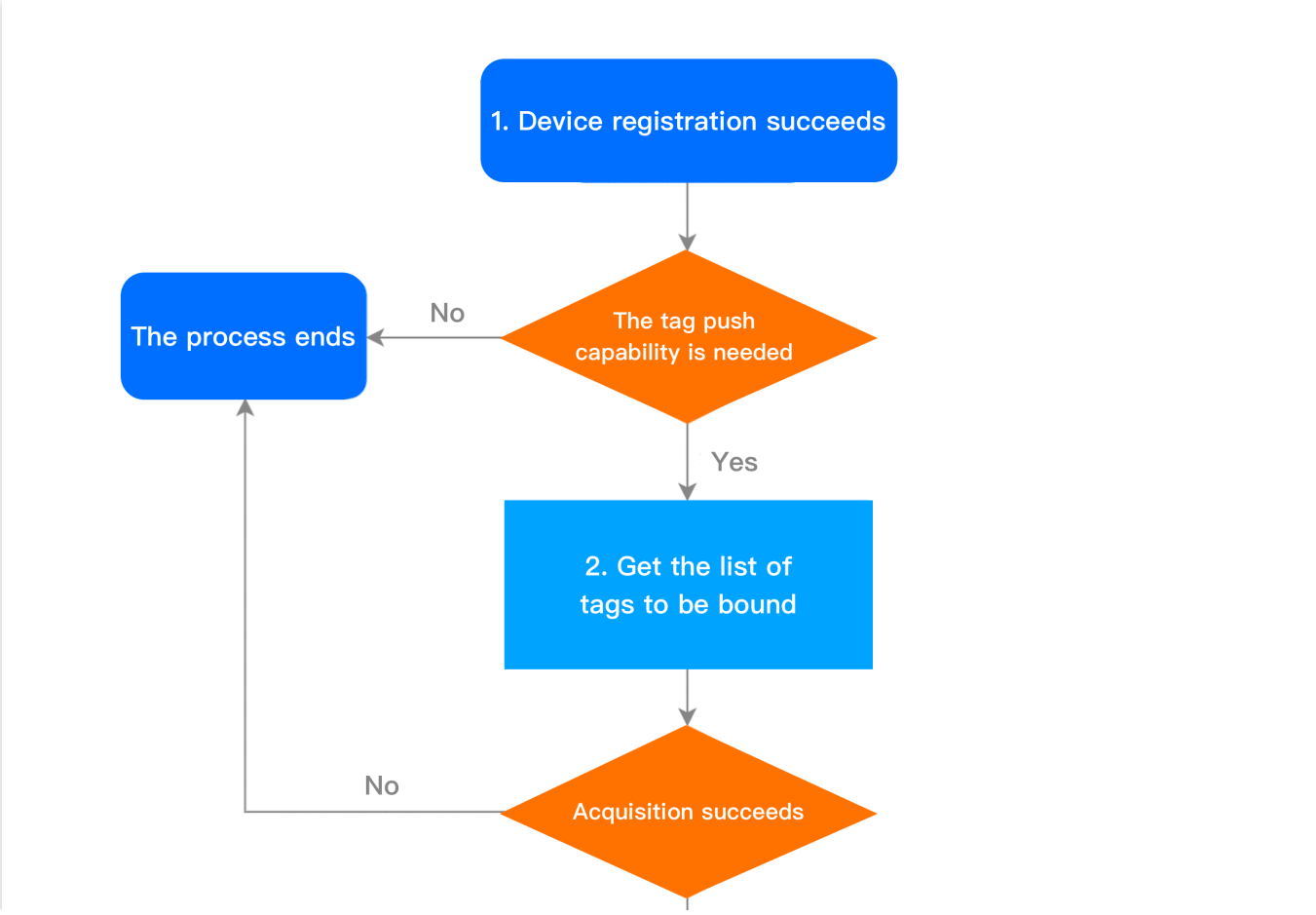


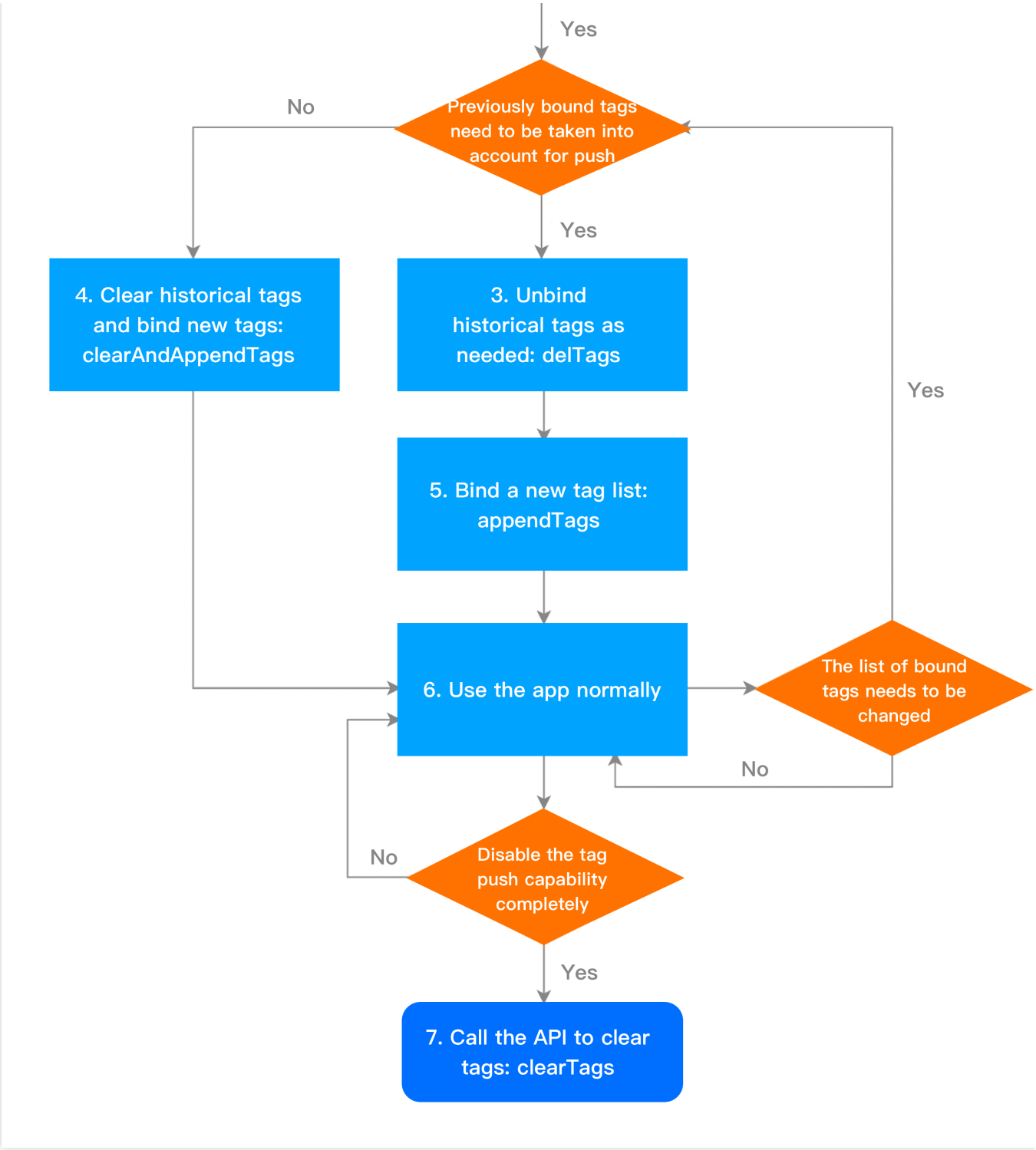

The account push capability is needed
No
The process ends
Yes
2. The app gets the list of accounts to be bound (such as logging in to a "WeChat" account)
Acquisition succeeds
No
Yes
Yes
3. Add or update existing account types: upsertAccounts, such as binding a "WeChat openid" as an account
No
The bound account type needs to be updated
4. Use the app normally
No
No
Clear all accounts bound to the device
A new account type needs to be bound
Yes
Yes
7. Call the API to clear accounts: clearAccounts
5. Bind a new account type: upsertAccounts, such as binding a "QQ openid" as an account



标签相关流程

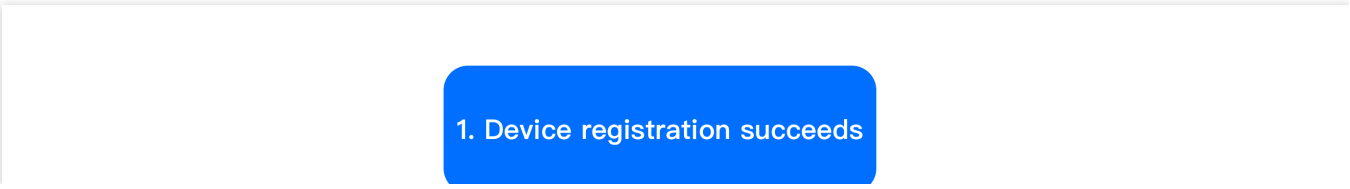
下图为标签相关流程，具体接口方法请查看 标签管理。

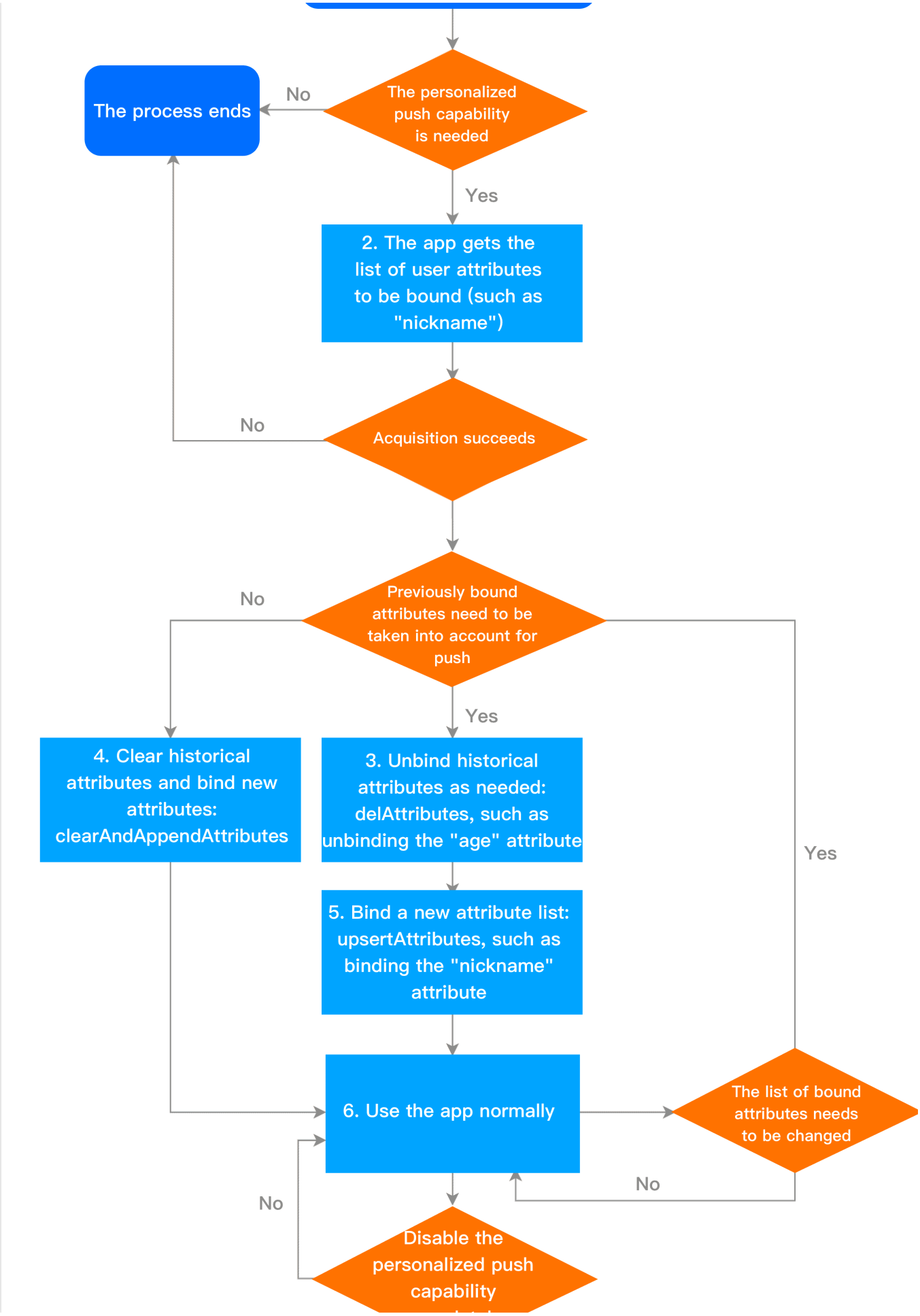


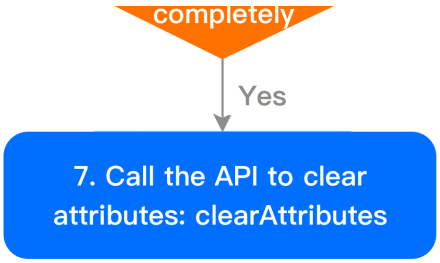


用户属性相关流程

下图为用户属性相关流程，具体接口方法请查看 用户属性管理。




The personalized push capability is needed
No
The process ends
Yes
2. The app gets the list of user attributes to be bound (such as "nickname")
Acquisition succeeds
No
Previously bound attributes need to be taken into account for push
No
Yes
4. Clear historical attributes and bind new attributes: clearAndAppendAttributes
3. Unbind historical attributes as needed: delAttributes, such as unbinding the "age" attribute
5. Bind a new attribute list: upsertAttributes, such as binding the "nickname" attribute
Yes
6. Use the app normally
The list of bound attributes needs to be changed
No
No
Disable the personalized push capability



Notes

Supported user attributes need to be configured in the web console or through RESTful APIs first

SDK 集成

最近更新时间：2024-01-16 17:39:39

简介

本文内容引导集成移动推送 SDK 在线通道推送能力，提供 AndroidStudio Gradle 自动集成和 Android Studio 手动集成两种方式指引。如需在应用进程被杀时也能收到推送，请在完成本文的集成操作后，参考 厂商通道接入指南 文档，完成各厂商通道的接入。

SDK 集成（二选一）

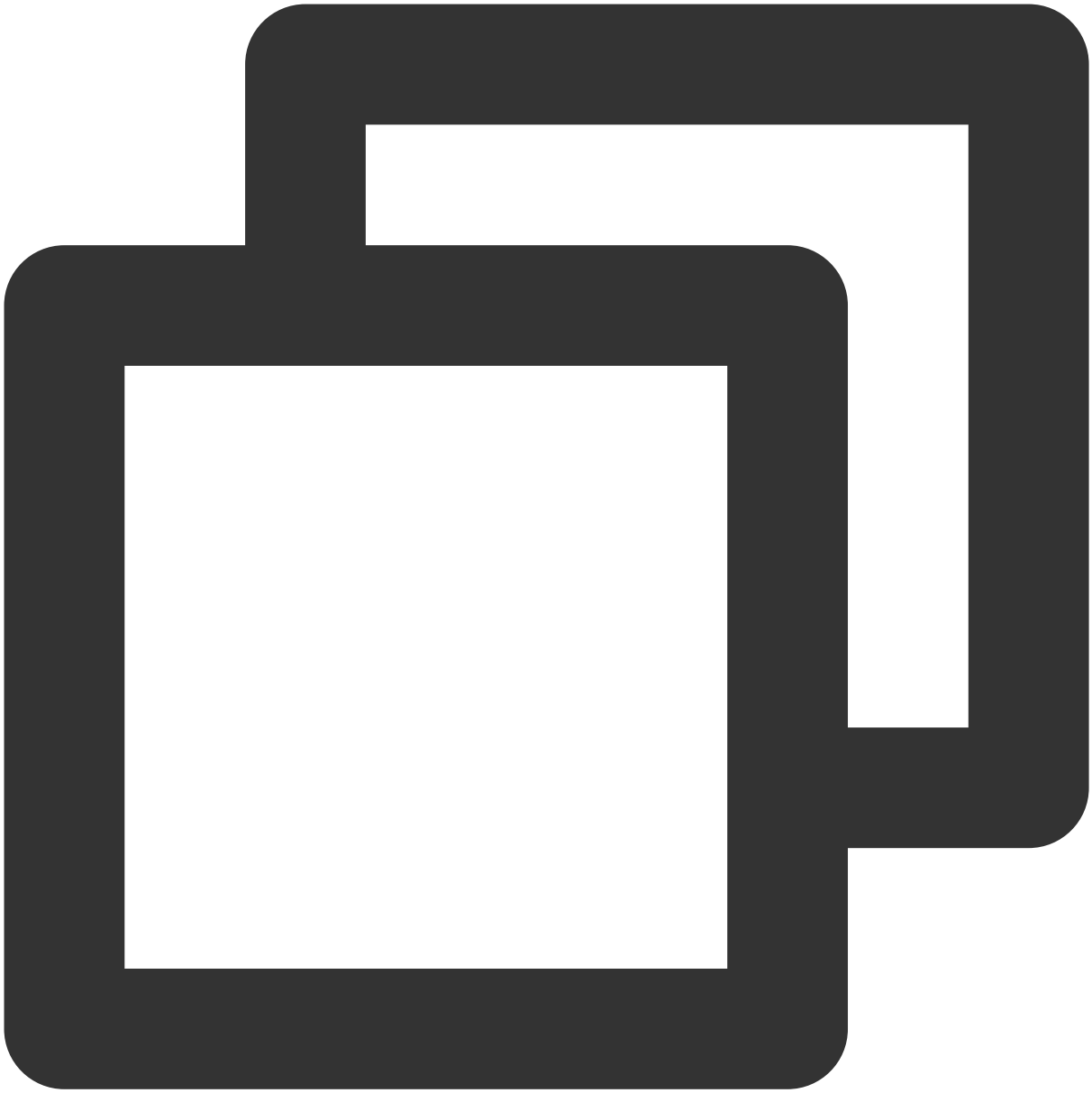
AndroidStudio Gradle 自动集成

操作步骤

注意：

在配置 SDK 前，确保已创建 Android 平台的应用。

1. 登录 移动推送控制台，在**产品管理>配置管理**页面获取应用的 AccessID、AccessKey。
2. 在 SDK 下载 页面，获取当前最新版本号。
3. 在 app build.gradle 文件下，配置以下内容：



```
android {
 ......
 defaultConfig {

     //控制台上注册的包名.注意application ID 和当前的应用包名以及控制台上注册应用的包名必须一致
     applicationId "您的包名"
     ......

     ndk {
         //根据需要 自行选择添加的对应cpu类型的.so库。
         abiFilters 'armeabi', 'armeabi-v7a', 'arm64-v8a'
```

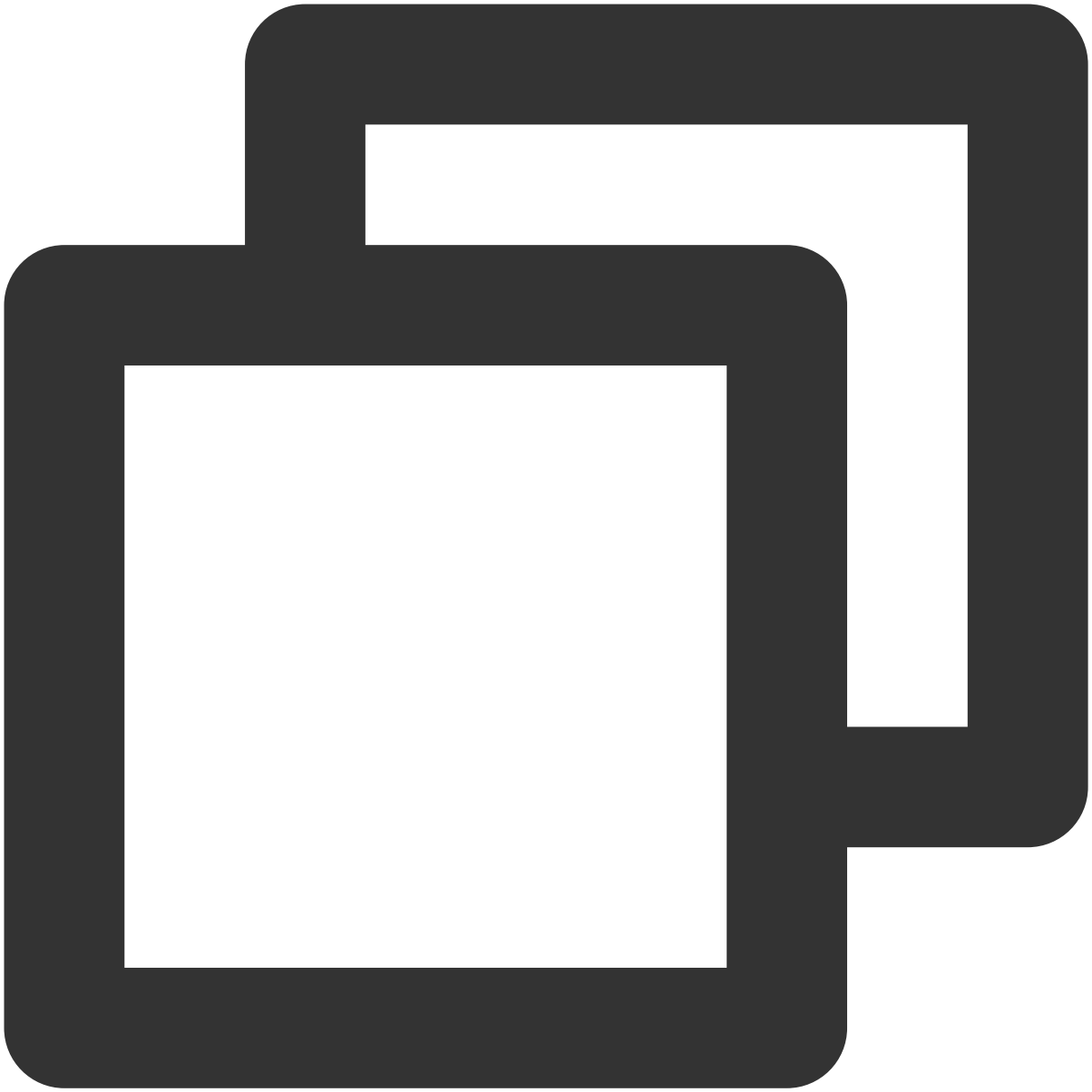
```
            // 还可以添加 'x86', 'x86_64', 'mips', 'mips64'
        }

        manifestPlaceholders = [

            XG_ACCESS_ID : "注册应用的accessid",
            XG_ACCESS_KEY : "注册应用的accesskey",
        ]
        ......
    }
    ......
}

dependencies {
    ......
    //添加以下依赖
    implementation 'com.tencent.tpns:tpns:[VERSION]-release'
          // TPNS 推送 [VERSION] 为最新 SDK 版本号，即为上述步骤2获取的版本号
}
```

注意：

如果您的应用服务接入点为广州，SDK 默认实现该配置。

如果您的应用服务接入点为上海、新加坡或中国香港，请按照下文步骤完成其他服务接入点域名配置，否则会导致注册推送服务失败并返回-502或1008003错误码。

在 AndroidManifest 文件 application 标签内添加以下元数据：



```
    <application>
    // 其他安卓组件
    <meta-data
            android:name="XG_SERVER_SUFFIX"
            android:value="其他服务接入点域名" />
</application>
```

说明：

其他服务接入点域名如下：

上海：`tpns.sh.tencent.com`

新加坡： `tpns.sgp.tencent.com`

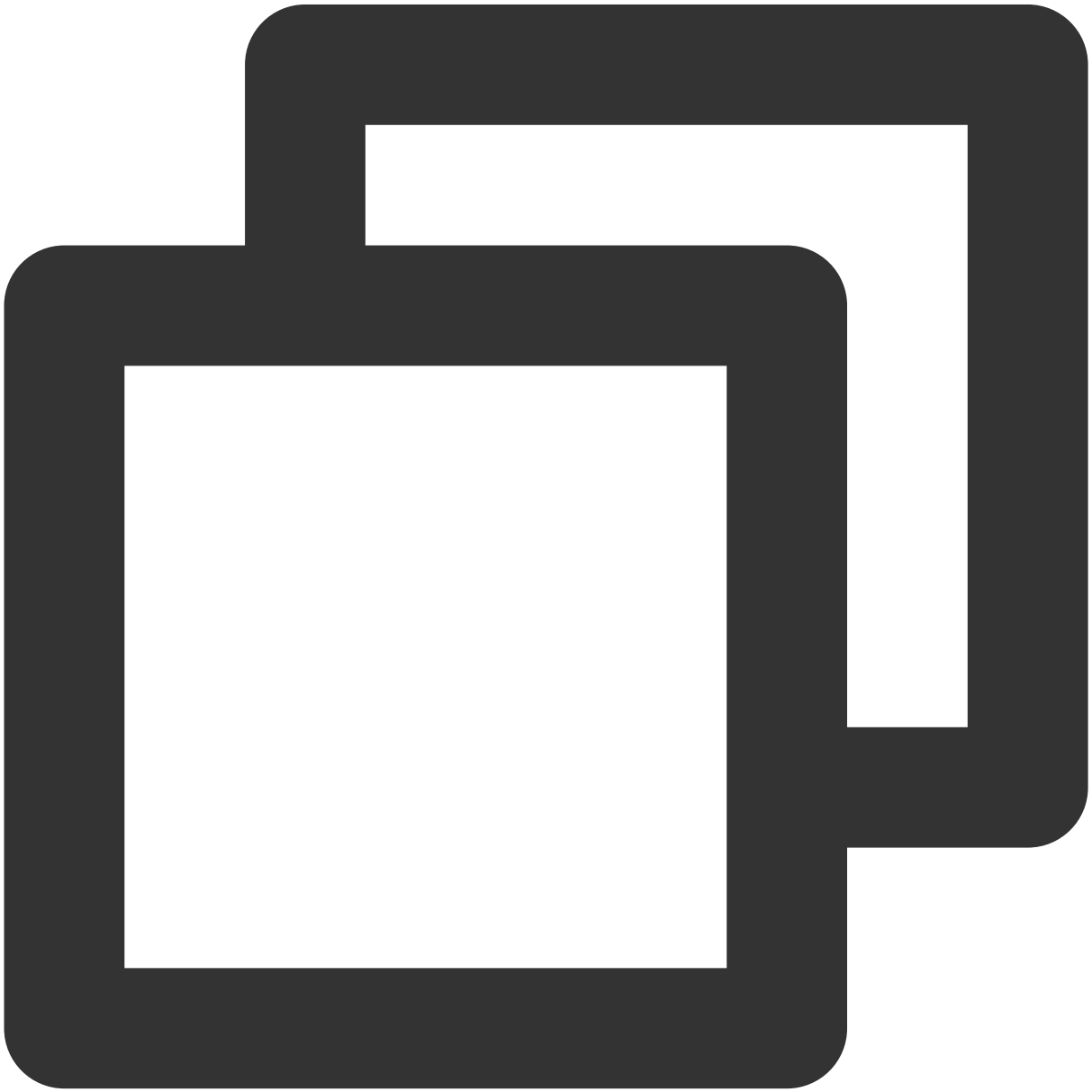
中国香港： `tpns.hk.tencent.com`

注意事项:

如在添加以上 abiFilter 配置后， Android Studio 出现以下提示：

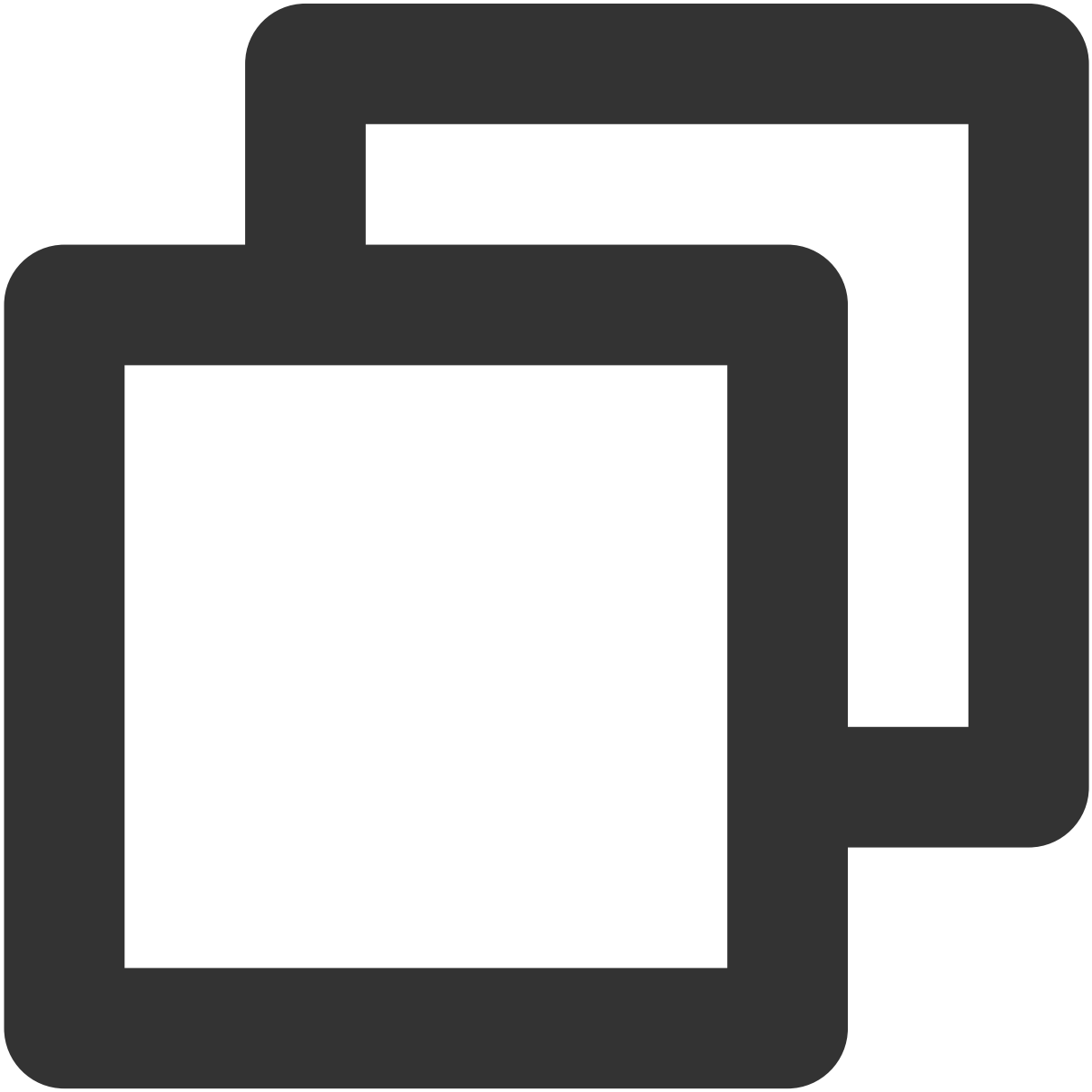
NDK integration is deprecated in the current plugin. Consider trying the new experimental plugin，则在 Project 根目录的 gradle.properties 文件中添加 android.useDeprecatedNdk=true。

如需监听消息请参考 XGPushBaseReceiver 接口或 Demo（在 SDK 压缩包内，可前往 SDK 下载 页面获取 ）的 MessageReceiver 类。自行继承 XGPushBaseReceiver 并且在配置文件中配置如下内容（请勿在 receiver 里处理耗时操作）：



```
<receiver android:name="com.tencent.android.xg.cloud.demo.MessageReceiver">
 <intent-filter>
     <!-- 接收消息透传 -->
     <action android:name="com.tencent.android.xg.vip.action.PUSH_MESSAGE" />
     <!-- 监听注册、反注册、设置/删除标签、通知被点击等处理结果 -->
     <action android:name="com.tencent.android.xg.vip.action.FEEDBACK" />
 </intent-filter>
</receiver>
```

如需兼容 Android P，需要添加使用 Apache HTTP client 库，在 AndroidManifest 的 application 节点内添加以下配置即可。



```
<uses-library android:name="org.apache.http.legacy" android:required="false"/>
```

Android Studio 手动集成

前往 SDK 下载 页面获取最新版 SDK，并参考以下步骤将 SDK 导入到您的 Android 工程中。

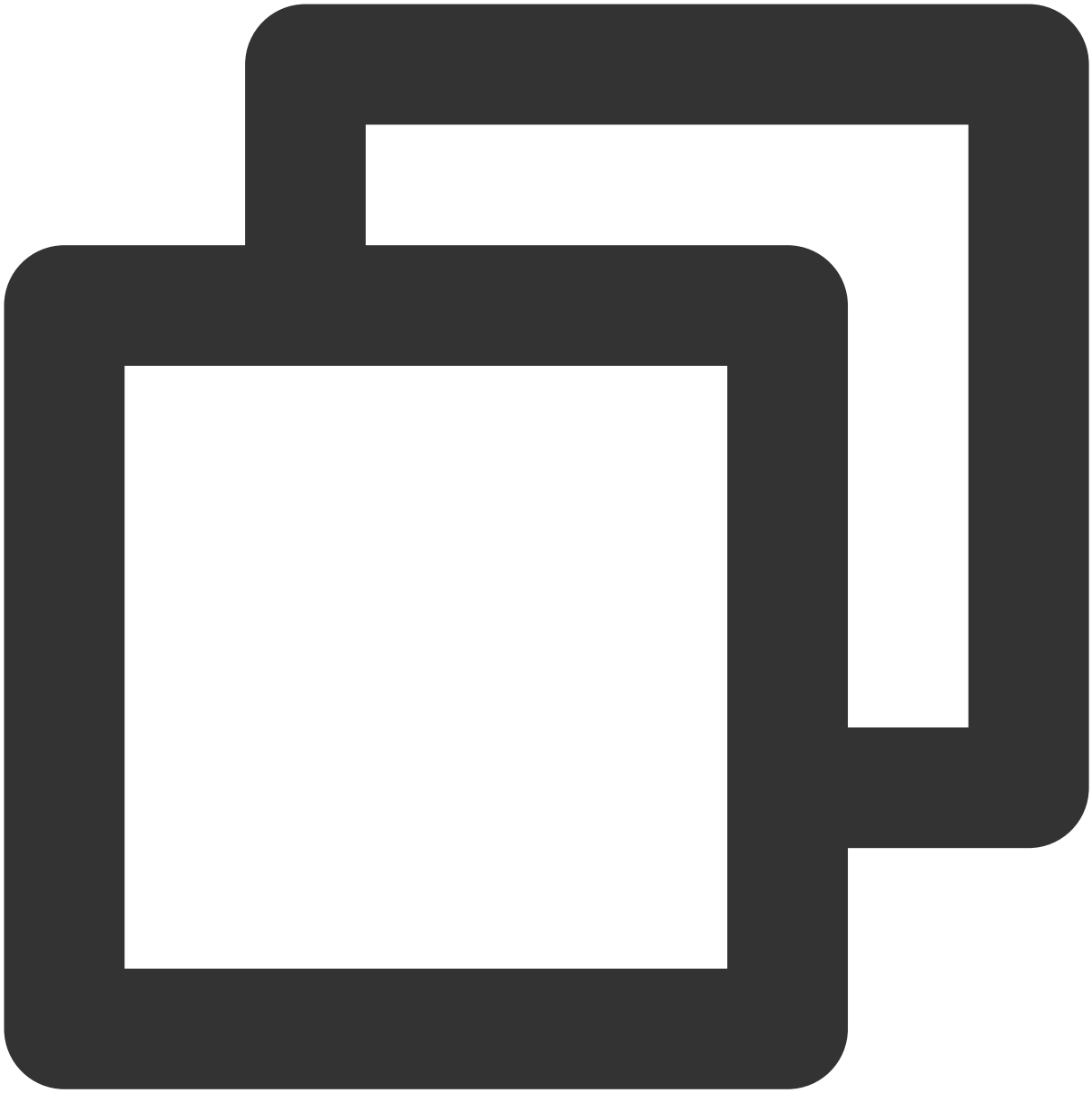
工程配置

将 SDK 导入到工程的步骤为：

1. 创建或打开 Android 工程。
2. 将移动推送 SDK 目录下的 libs 目录所有 .jar 文件拷贝到工程的 libs（或 lib）目录下。
3. .so 文件是移动推送必须的组件，支持 armeabi、armeabi-v7a、arm64-v8a、mips、mips64、x86、x86_64平台，请根据自己当前 .so 支持的平台添加
4. 打开 AndroidManifest.xml，添加以下配置（建议参考下载包 Demo 中的 Merged Manifest 修改），其中 “APP的AccessId ” 和 “APP的AccessKey” 替换为 App 对应的 AccessId 和 AccessKey，请确保按照要求配置，否则可能导致服务不能正常使用。

权限配置

移动推送 SDK 正常运行所需要的权限。示例代码如下：

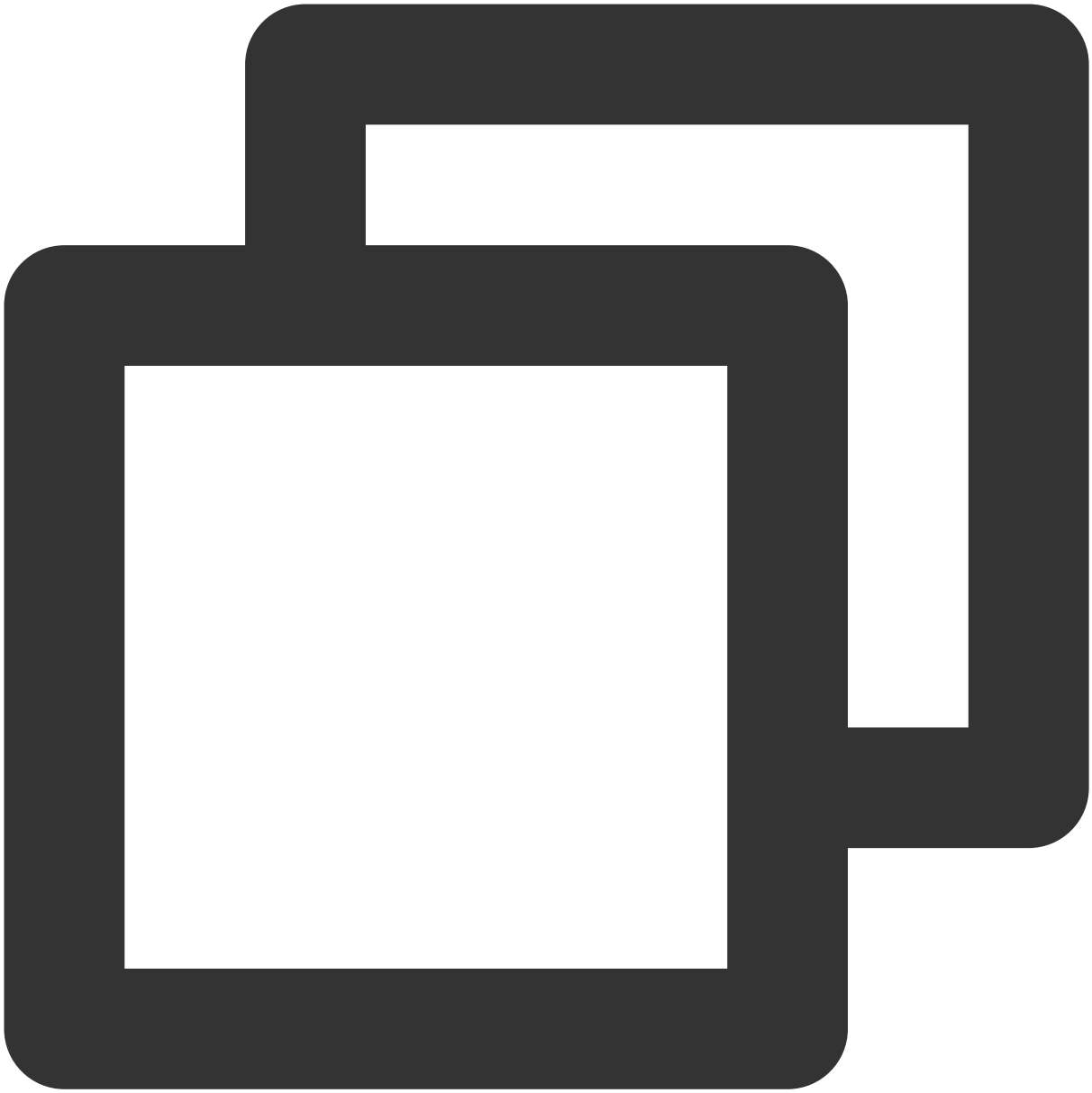


```
<!-- 【必须】 移动推送 TPNS SDK VIP版本所需权限 -->
<permission
        android:name="应用包名.permission.XGPUSH_RECEIVE"
        android:protectionLevel="signature" />
<uses-permission android:name="应用包名.permission.XGPUSH_RECEIVE" />

<!-- 【必须】 移动推送 TPNS SDK 所需权限 -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.SCHEDULE_EXACT_ALARM" />
```

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

<!-- 【常用】 移动推送 TPNS SDK所需权限 -->
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.RECEIVE_USER_PRESENT" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.GET_TASKS" />
```

权限	是否必选	说明
android.permission.INTERNET	必选	允许程序访问网络连接，可能产生 GPRS 流量
android.permission.ACCESS_WIFI_STATE	必选	允许程序获取当前 Wi-Fi 接入的状态以及 WLAN 热点的信息
android.permission.ACCESS_NETWORK_STATE	必选	允许程序获取网络信息状态
android.permission.WAKE_LOCK	必选	允许程序在手机屏幕关闭后，后台进程仍然运行
android.permission.SCHEDULE_EXACT_ALARM	必选	允许定时广播
android.permission.VIBRATE	可选	允许应用震动
android.permission.RECEIVE_USER_PRESENT	可选	允许应用可以接收点亮屏幕或解锁广播
android.permission.WRITE_EXTERNAL_STORAGE	可选	允许程序写入外部存储
android.permission.RESTART_PACKAGES	可选	允许程序结束任务
android.permission.GET_TASKS	可选	允许程序获取任务信息

组件和应用信息配置



```
<application>
    <activity android:name="com.tencent.android.tpush.TpnsActivity"
            android:theme="@android:style/Theme.Translucent.NoTitleBar"
            android:launchMode="singleInstance"
            android:exported="true">
            <intent-filter>
                <action android:name="${applicationId}.OPEN_TPNS_ACTIVITY" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                <data
```

```
                android:scheme="tpns"
                android:host="${applicationId}"/>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.BROWSABLE" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action" />
        </intent-filter>
    </activity>

    <activity
        android:name="com.tencent.android.tpush.InnerTpnsActivity"
        android:exported="false"
        android:launchMode="singleInstance"
        android:theme="@android:style/Theme.Translucent.NoTitleBar">
        <intent-filter>
            <action android:name="${applicationId}.OPEN_TPNS_ACTIVITY_V2" />

            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
        <intent-filter>
            <data
                android:host="${applicationId}"
                android:scheme="stpns" />

            <action android:name="android.intent.action.VIEW" />

            <category android:name="android.intent.category.BROWSABLE" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action" />
        </intent-filter>
    </activity>

    <!-- 【必须】 信鸽receiver广播接收 -->
    <receiver
        android:name="com.tencent.android.tpush.XGPushReceiver"
        android:exported="false"
        android:process=":xg_vip_service">

        <intent-filter android:priority="0x7fffffff">


            <!-- 【必须】 信鸽SDK的内部广播 -->
            <action android:name="com.tencent.android.xg.vip.action.SDK" />
```

```
                <action android:name="com.tencent.android.xg.vip.action.INTERNAL_PU
                <action android:name="com.tencent.android.xg.vip.action.ACTION_SDK_
            </intent-filter>

        </receiver>

    <!-- 【必须】移动推送 TPNS service -->
    <service
        android:name="com.tencent.android.tpush.service.XGVipPushService"
        android:exported="false"
        android:process=":xg_vip_service">
    </service>

    <!-- 【必须】通知 service ，android:name 部分改为包名.XGVIP_PUSH_ACTION -->
        <service android:name="com.tencent.android.tpush.rpc.XGRemoteService"
            android:exported="false">
            <intent-filter>
                <!-- 【必须】请修改为当前APP名包.XGVIP_PUSH_ACTION -->
                <action android:name="应用包名.XGVIP_PUSH_ACTION" />
            </intent-filter>
        </service>

    <!-- 【必须】【注意】authorities 修改为包名.XGVIP_PUSH_AUTH -->
    <provider
        android:name="com.tencent.android.tpush.XGPushProvider"
        android:authorities="应用包名.XGVIP_PUSH_AUTH" />

    <!-- 【必须】【注意】authorities 修改为包名.TPUSH_PROVIDER -->
    <provider
        android:name="com.tencent.android.tpush.SettingsContentProvider"
        android:authorities="应用包名.TPUSH_PROVIDER" />

    <!-- 【可选】用于增强保活能力 -->
    <provider
        android:name="com.tencent.android.tpush.XGVipPushKAProvider"
        android:authorities="应用包名.AUTH_XGPUSH_KEEPALIVE"
        android:exported="true" />

    <!-- 【可选】APP实现的Receiver，用于接收消息透传和操作结果的回调，请根据需要添加 -->
    <!-- YOUR_PACKAGE_PATH.CustomPushReceiver需要改为自己的Receiver： -->
    <receiver android:name="应用包名.MessageReceiver"
        android:exported="false">
        <intent-filter>
            <!-- 接收消息透传 -->
            <action android:name="com.tencent.android.xg.vip.action.PUSH_MESSAGE" /
            <!-- 监听注册、反注册、设置/删除标签、通知被点击等处理结果 -->
            <action android:name="com.tencent.android.xg.vip.action.FEEDBACK" />
```

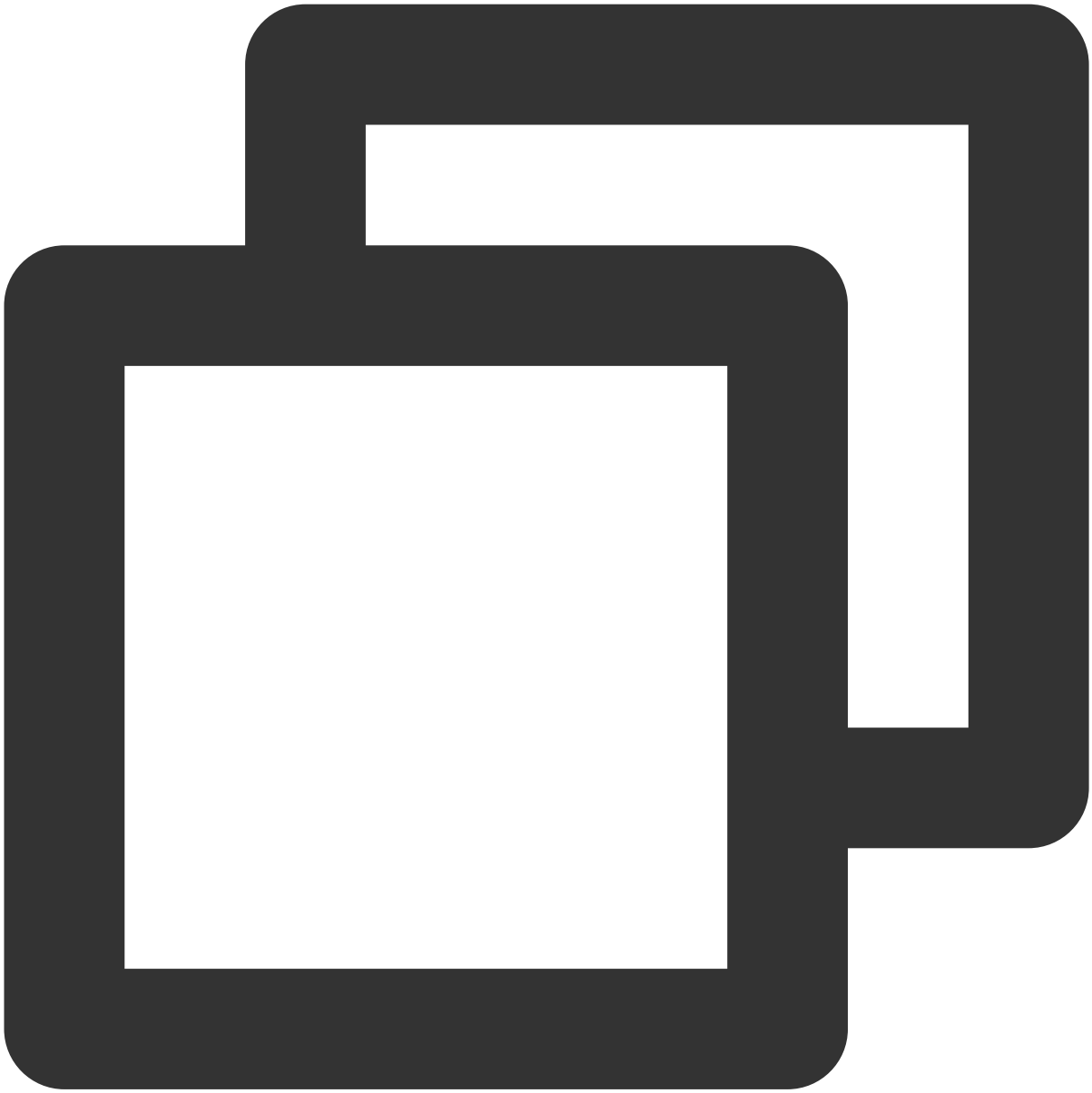
```
        </intent-filter>
    </receiver>

    <!-- MQTT START -->
    <service android:exported="false"
            android:process=":xg_vip_service"
            android:name="com.tencent.tpns.mqttchannel.services.MqttService" />

    <provider
            android:exported="false"
            android:name="com.tencent.tpns.baseapi.base.SettingsContentProvider"
            android:authorities="应用包名.XG_SETTINGS_PROVIDER" />

    <!-- MQTT END-->

    <!-- 【必须】 请修改为 APP 的 AccessId，“15”开头的10位数字，中间没空格 -->
    <meta-data
        android:name="XG_V2_ACCESS_ID"
        android:value="APP的AccessId" />
    <!-- 【必须】 请修改为APP的AccessKey，“A”开头的12位字符串，中间没空格 -->
    <meta-data
        android:name="XG_V2_ACCESS_KEY"
        android:value="APP的AccessKey" />

</application>

<!-- 【必须】 移动推送 TPNS SDK 5.0版本所需权限 -->
<permission
    android:name="应用包名.permission.XGPUSH_RECEIVE"
    android:protectionLevel="signature" />
<uses-permission android:name="应用包名.permission.XGPUSH_RECEIVE" />

<!-- 【必须】 移动推送 TPNS SDK所需权限 -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>

<!-- 【常用】 移动推送 TPNS SDK所需权限 -->
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.RECEIVE_USER_PRESENT" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

注意：

如果您的应用服务接入点为广州，SDK 默认实现该配置。

如果您的应用服务接入点为上海、新加坡或中国香港，请按照下文步骤完成其他服务接入点域名配置，否则会导致注册推送服务失败并返回-502或1008003错误码。

在 AndroidManifest 文件 application 标签内添加以下元数据：



```
  <application>
 // 其他安卓组件
 <meta-data
        android:name="XG_SERVER_SUFFIX"
        android:value="其他服务接入点域名" />
</application>
```

说明：

其他服务接入点域名如下：

上海：`tpns.sh.tencent.com`

新加坡：`tpns.sgp.tencent.com`

中国香港：`tpns.hk.tencent.com`

调试及设备注册

开启 Debug 日志数据

注意：

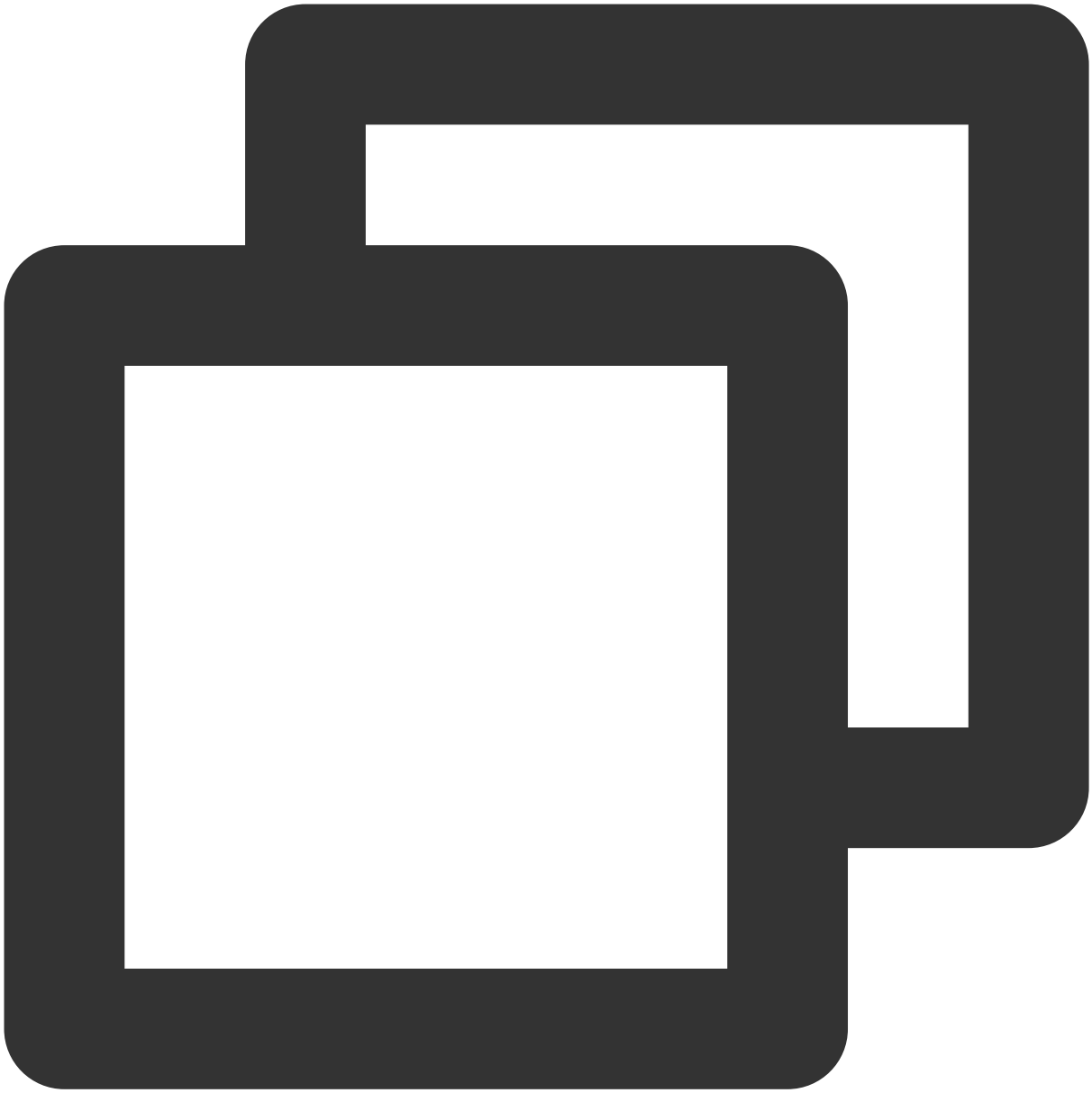
上线时请设置为 false。

Token 注册

在需要启动推送服务的地方调用推送服务注册接口：

注意：

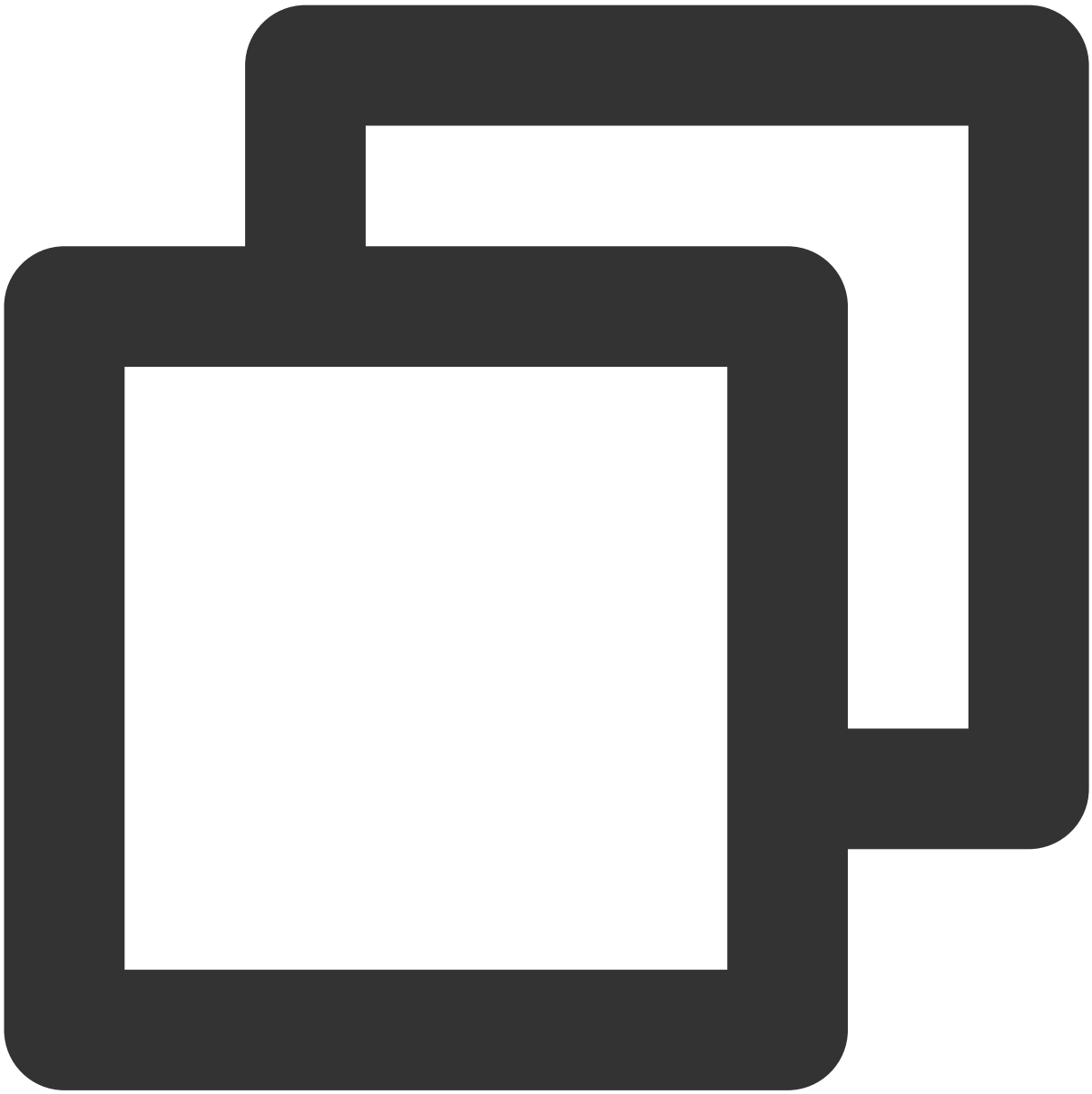
建议仅在 App 的主进程内调用注册接口。



```
XGPushManager.registerPush(this, new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        //token在设备卸载重装的时候有可能会变
        Log.d("TPush", "注册成功，设备token为：" + data);
    }

    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.d("TPush", "注册失败，错误码：" + errCode + ",错误信息：" + msg);
    }
```

```
});
```

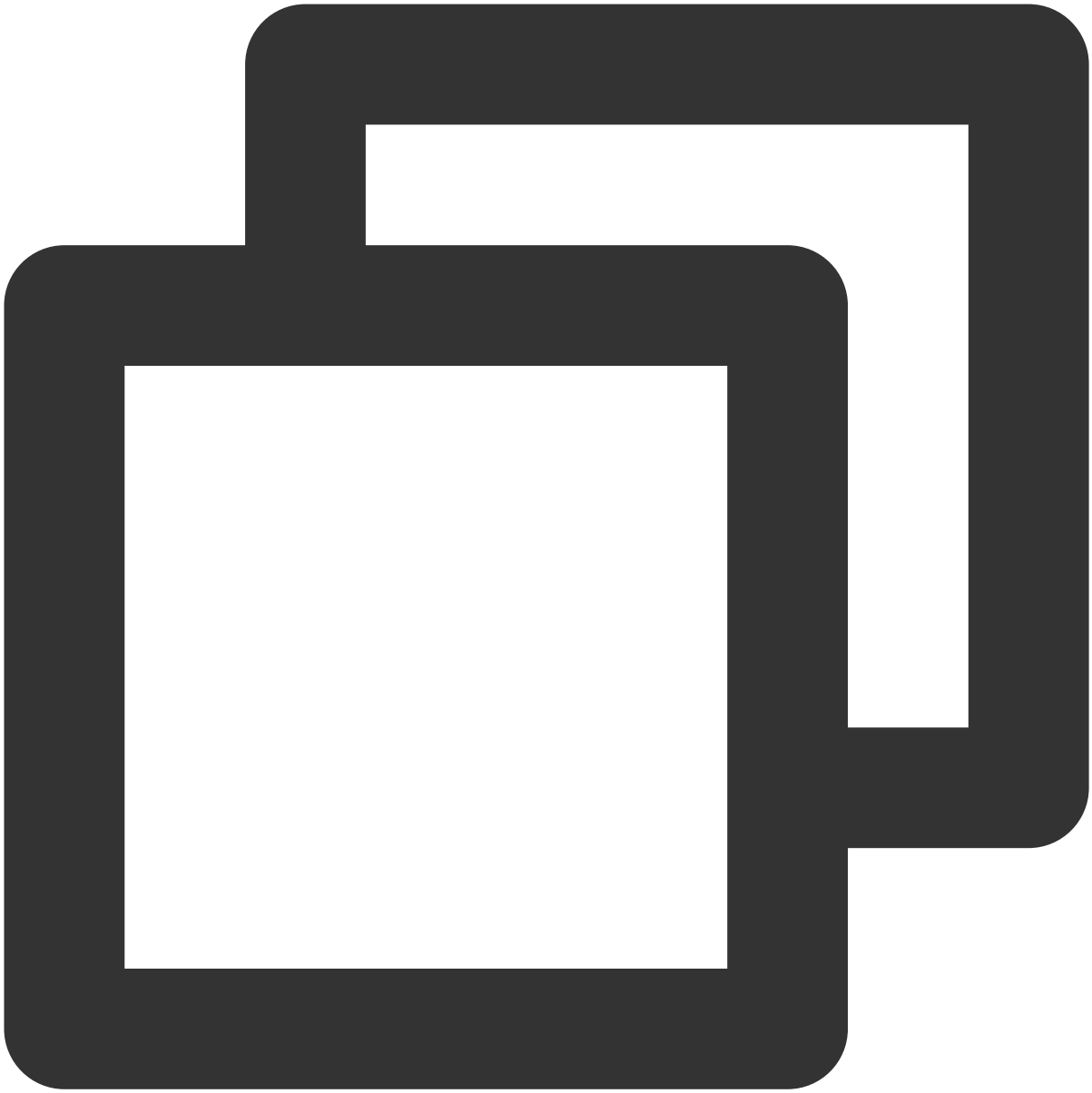
过滤 "TPush" 注册成功的日志如下：



```
TPNS register push success with token : 6ed8af8d7b18049d9fed116a9db9c71ab44d5565
```

关闭日志打印

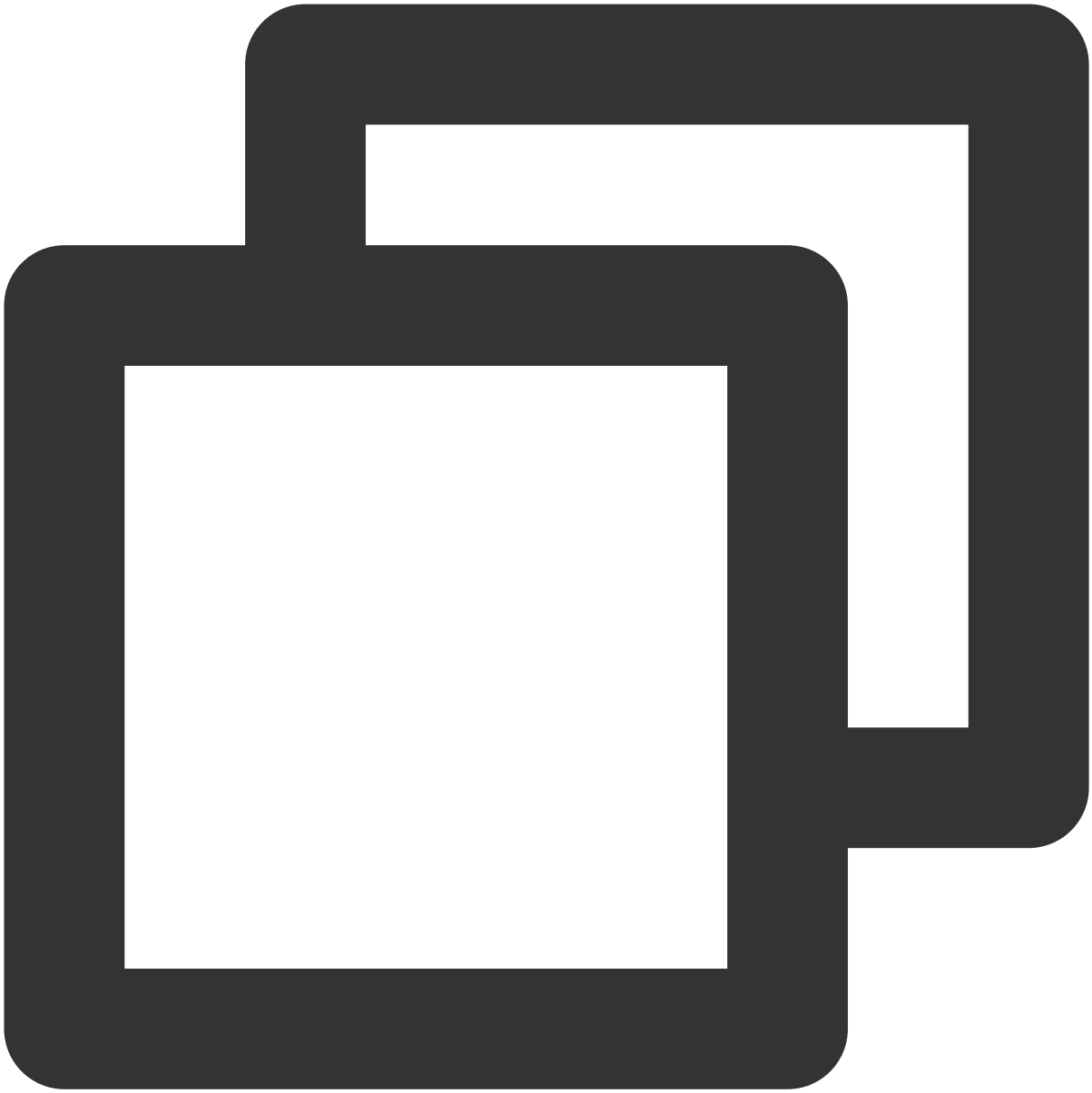
调用 XGPushConfig.enableDebug(context, false) 关闭 SDK debug 日志开关时，SDK 默认仍会打印部分日常运行日志（包含移动推送 Token）。
您可以通过在 Application.onCreate 内调用如下方法，来关闭这些日常运行日志在控制台的输出打印：



```
new XGPushConfig.Build(context).setLogLevel(Log.ERROR);
```

代码混淆

如果您的项目中使用 proguard 等工具，已做代码混淆，请保留以下选项，否则将导致移动推送服务不可用：



```
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep class com.tencent.android.tpush.** {*;}
-keep class com.tencent.tpns.baseapi.** {*;}
-keep class com.tencent.tpns.mqttchannel.** {*;}
-keep class com.tencent.tpns.dataacquisition.** {*;}

-keep class com.tencent.bigdata.baseapi.** {*;}    // 1.2.0.1 及以上版本不需要此条配置
-keep class com.tencent.bigdata.mqttchannel.** {*;}  // 1.2.0.1 及以上版本不需要此条配
```

注意：

如果移动推送 SDK 被包含在 App 的公共 SDK 里，即使公共 SDK 有增加配置混淆规则，主工程 App 也必须要同时增加配置混淆规则。

高级配置（可选）

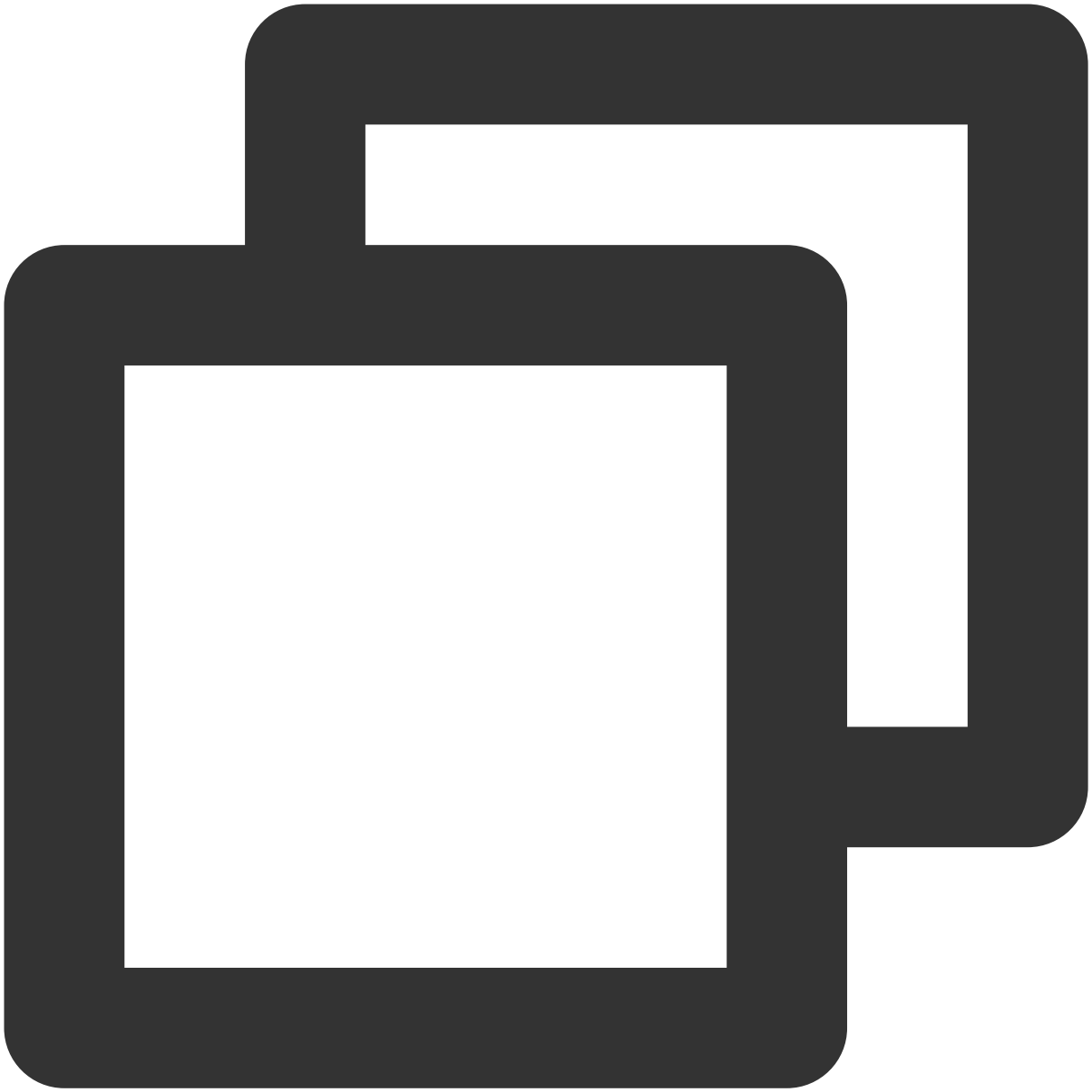
关闭联合保活

如需关闭联合保活功能，请在应用初始化的时候，例如 Application 或 LauncherActivity 的 onCreate 中调用如下接口，并传递 false 值：

注意：

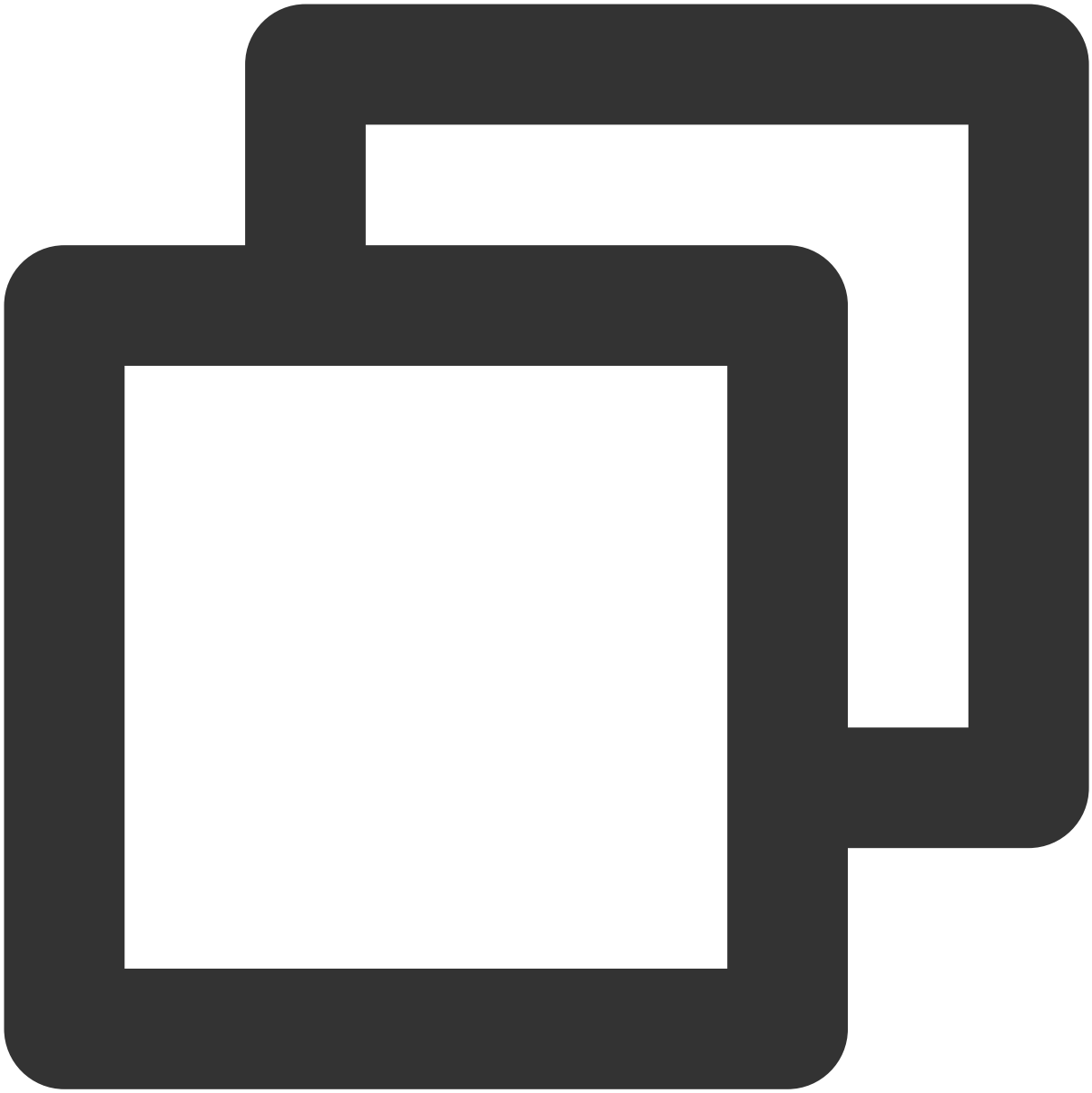
仅 1.1.6.0 之后版本支持关闭联合保活功能，1.1.6.0之前版本移动推送默认开启联合保活能力，且不可关闭。

1.2.6.0 起默认关闭联合保活功能，可不再调用此接口。



```
XGPushConfig.enablePullUpOtherApp(Context context, boolean pullUp);
```

若您使用 gradle 自动集成方式，请在自身应用的 AndroidManifest.xml 文件 <application> 标签下配置如下结点，其中 `xxx` 为任意自定义名称；如果使用手动集成方式，请修改如下节点属性：



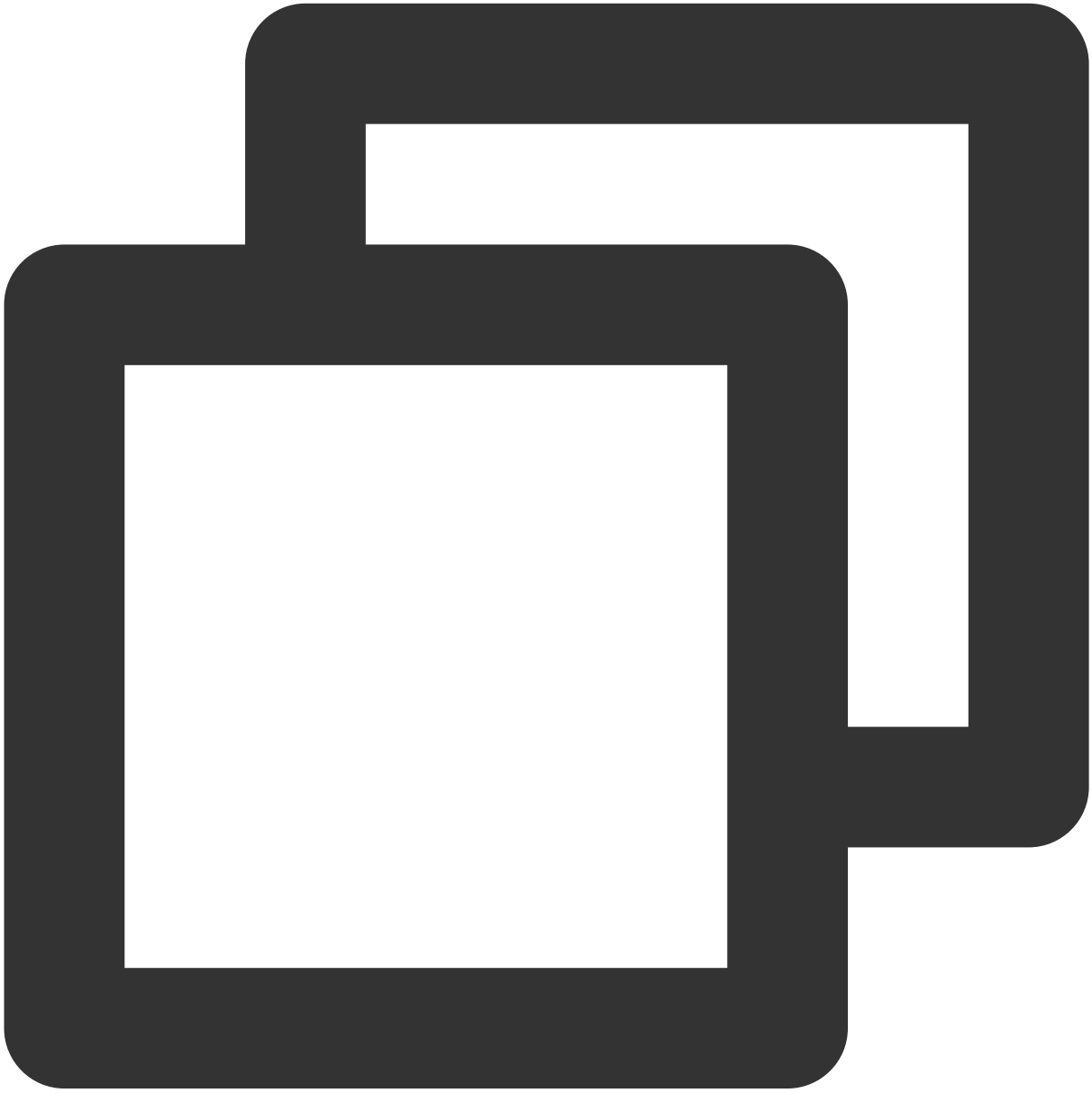
```
<!-- 在自身应用的AndroidManifest.xml文件中添加如下结点，其中 xxx 为任意自定义名称： -->
<!-- 关闭与 TPNS 应用的联合保活功能，请配置 -->
<provider
        android:name="com.tencent.android.tpush.XGPushProvider"
        tools:replace="android:authorities"
        android:authorities="应用包名.xxx.XGVIP_PUSH_AUTH"
        android:exported="false" />
```

若控制台有以下日志打印，则表明联合保活功能已经关闭：`I/TPush: [ServiceUtil] disable pull up other app` 。

获取移动推送 Token 交互建议

建议您完成 SDK 集成后，在 App 的【关于】、【意见反馈】等比较不常用的 UI 中，通过手势或者其他方式显示移动推送Token，控制台和 Restful API 推送需要根据移动推送Token 进行 Token 推送，后续问题排查也需要根据移动推送Token 进行定位。

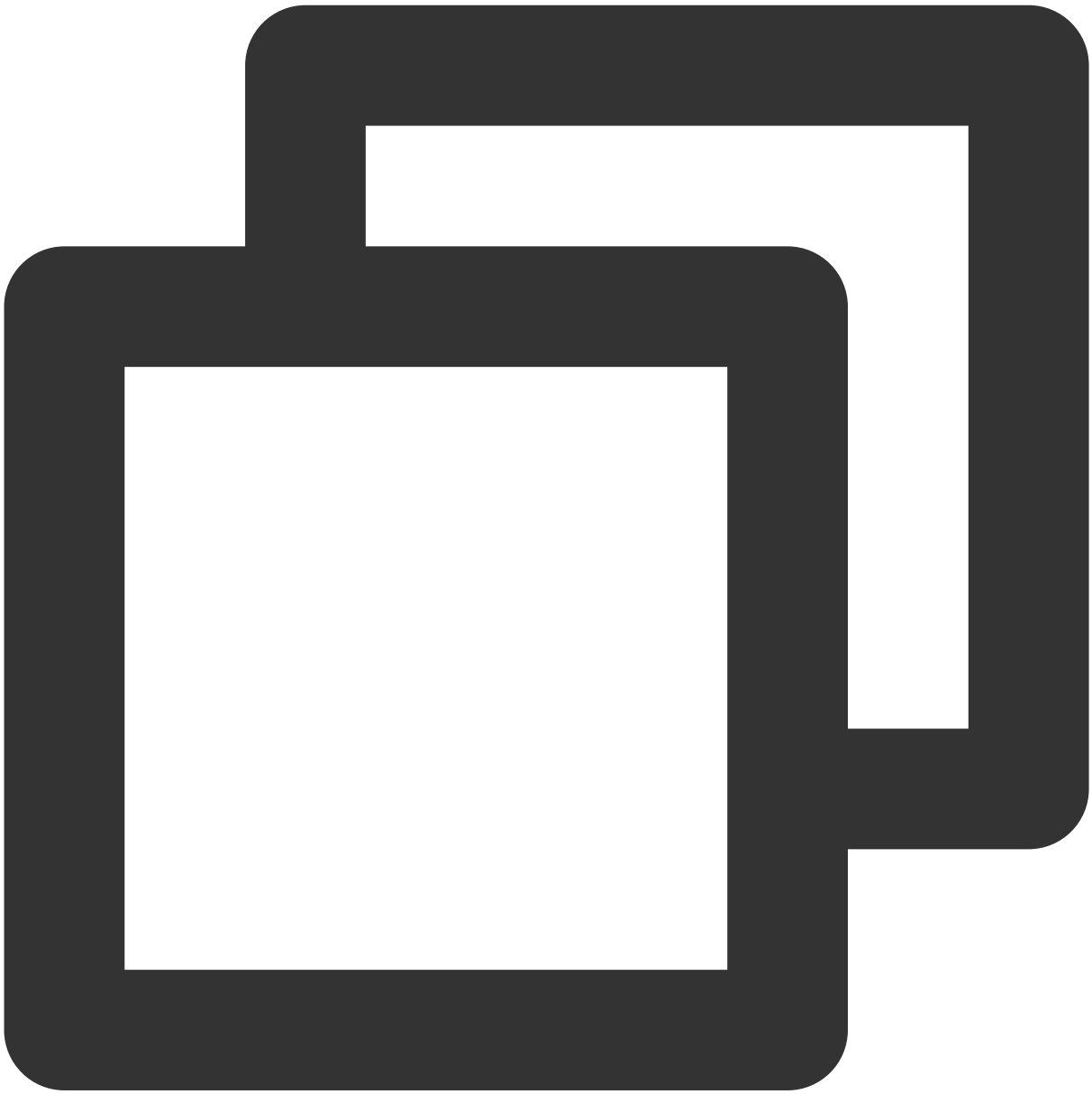
示例代码如下：



```
//获取 Token
XGPushConfig.getToken(getApplicationContext());
```

获取移动推送运行日志交互建议

SDK 提供日志上报接口。如用户在应用上线后遇到推送相关问题，可以通过引导用户操作触发此接口，上传 SDK 运行日志并获取回调返回的日志文件下载地址，方便问题排查。详情参考 日志上报接口。
示例代码如下：

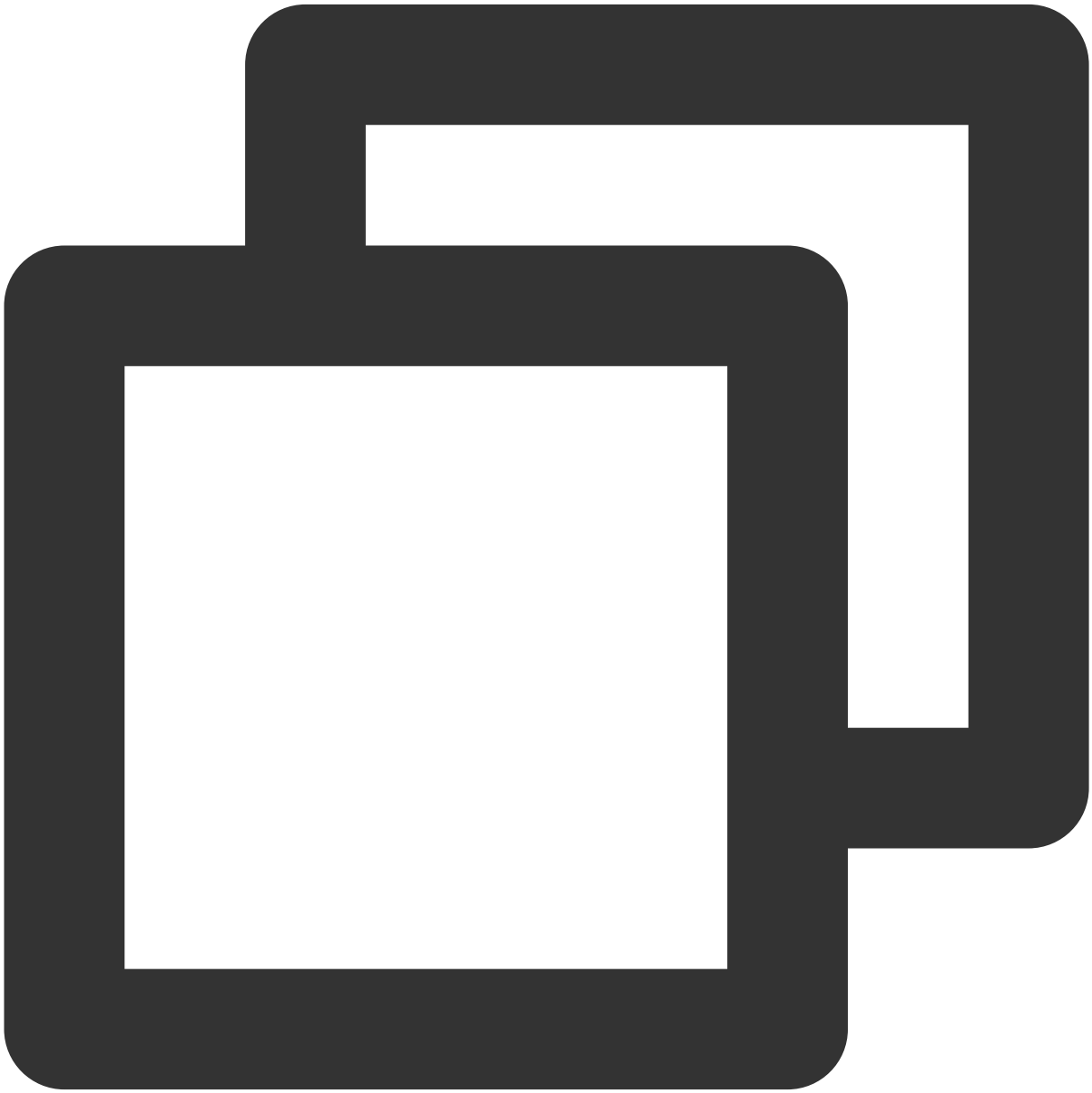


```
XGPushManager.uploadLogFile(context, new HttpRequestCallback() {
    @Override
    public void onSuccess(String result) {
        Log.d("TPush", "上传成功，文件地址：" + result);
    }
```

```
        @Override
    public void onFailure(int errCode, String errMsg) {
        Log.d("TPush", "上传失败，错误码：" + errCode + ",错误信息：" + errMsg);
    }
});
```

隐私协议声明建议

您可在申请 App 权限使用时，使用以下内容声明授权的用途：



我们使用 腾讯云移动推送 TPNS 用于实现产品信息的推送，在您授权我们“访问网络连接”和“访问网络状态”权

其中上述声明授权的两个链接如下：

腾讯云移动推送： `https://www.tencentcloud.com/products/tpns`

接口文档

最近更新时间：2024-01-16 17:39:39

说明

本文档中账号功能、删除标签功能适用于 SDK 1.2.3.0 或更高版本，1.2.3.0 及之前版本请参见 接口文档。
所有 API 接口的包名路径前缀都是：`com.tencent.android.tpush`，其中有以下几个重要的对外提供接口的类名，如下表所示：

类名	说明
XGPushManager	Push 服务推送
XGPushConfig	Push 服务配置项接口
XGPushBaseReceiver	接收消息和结果反馈的 Receiver，需要开发者在 AndroidManifest.xml 自主完成静态注册

启动与注册

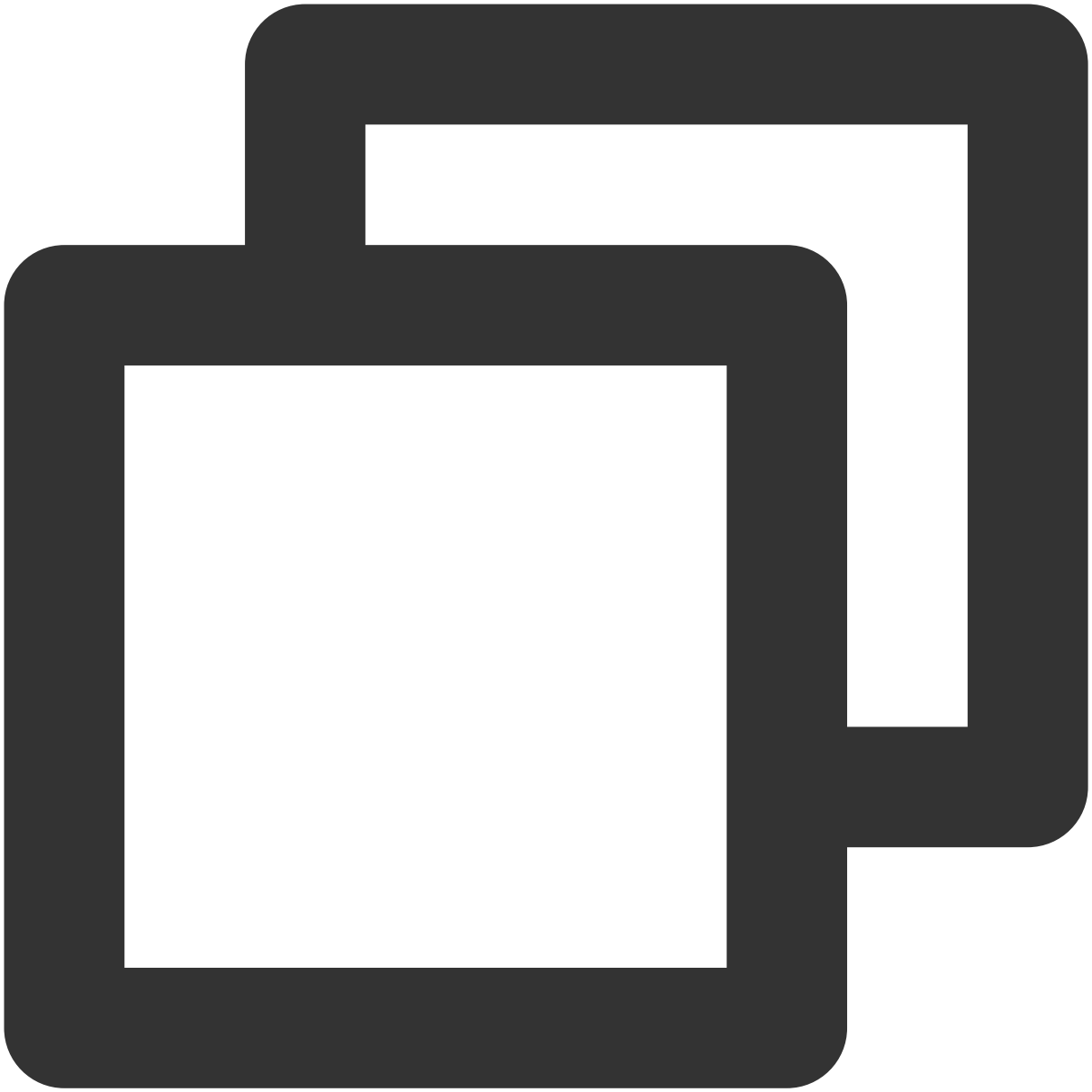
App 只有在完成移动推送的启动与注册后才可以移动推送 SDK 提供 Push 服务，在这之前请确保配置 AccessId 和 AccessKey。
新版的 SDK 已经将启动移动推送和 App 注册统一集成在注册接口中，即只需调用注册接口便默认完成启动和注册操作。
注册成功后，会返回设备 Token，Token 用于标识设备唯一性，同时也是移动推送维持与后台连接的唯一身份标识。关于如何获取 Token 请参考 获取 Token。
注册接口通常提供简版和带 callback 版本的接口，请根据业务需要决定选择接口。

设备注册

以下为设备注册相关接口方法，若需了解调用时机及调用原理，可查看 设备注册流程。

接口说明

普通注册只注册当前设备，后台能够针对不同的设备 Token 发送推送消息，以下有2个版本的 API 接口方法：

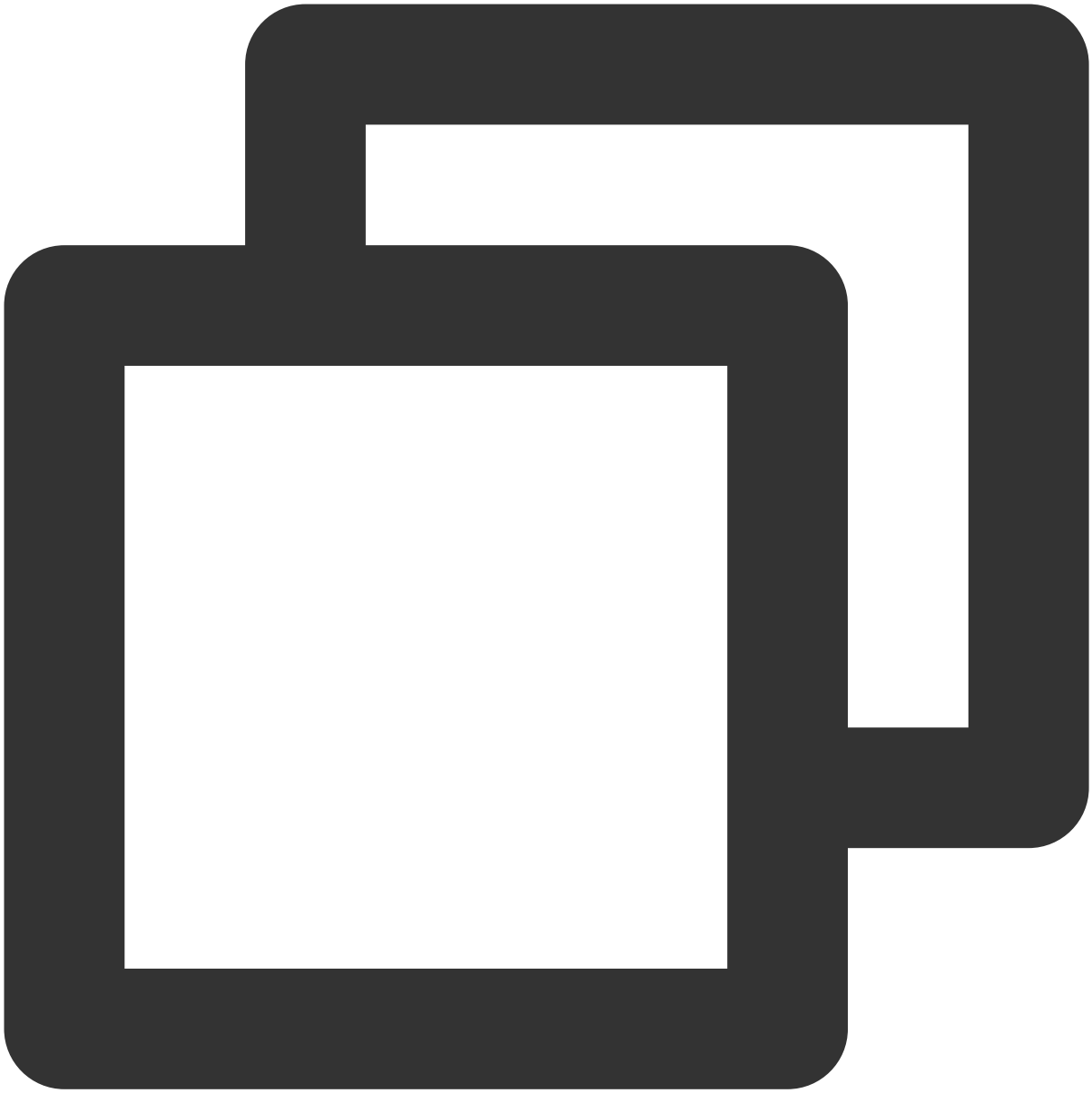


```
public static void registerPush(Context context)
```

参数说明

context：当前应用上下文对象，不能为 null。

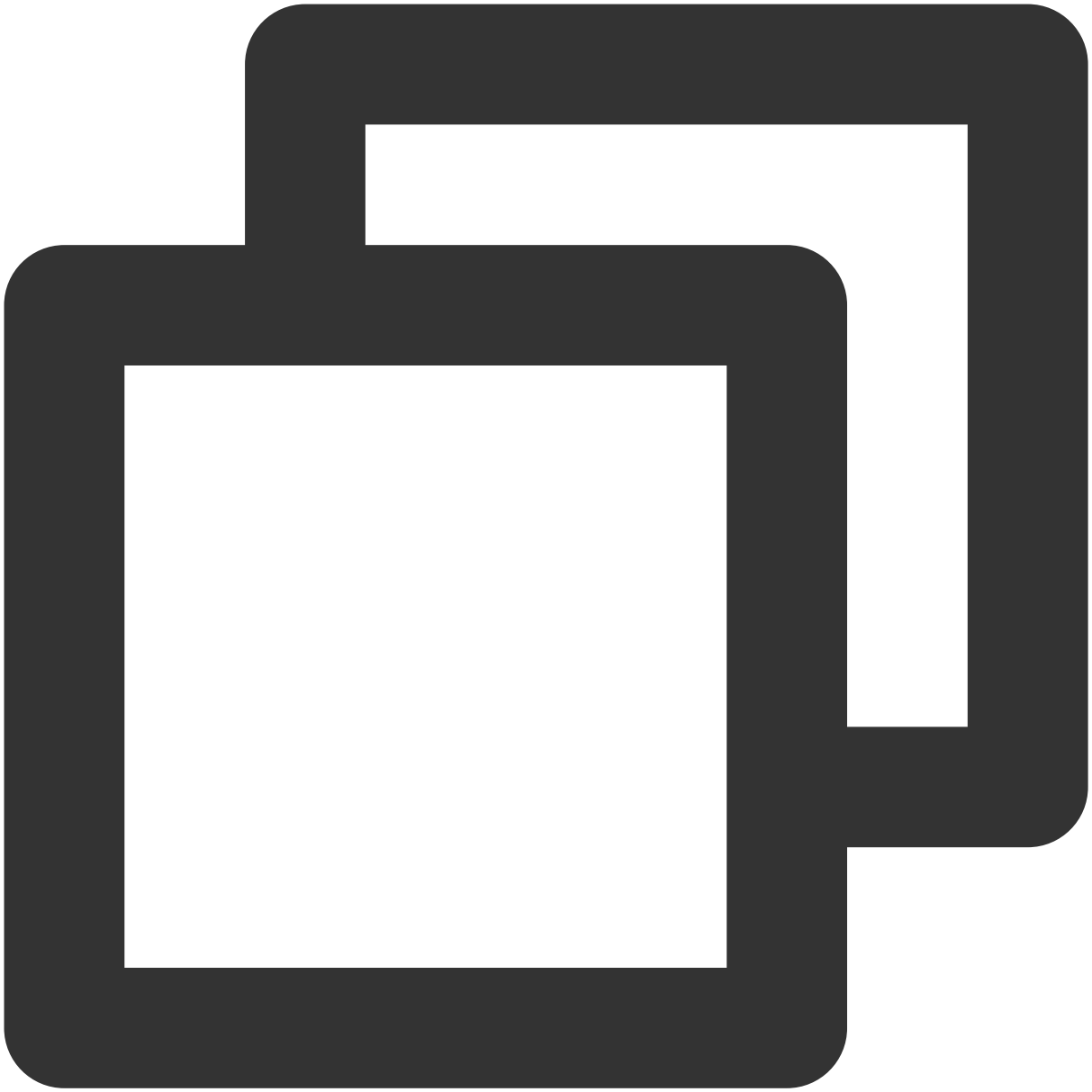
示例代码



```
XGPushManager.registerPush(getApplicationContext());
```

接口说明

为方便用户获取注册是否成功的状态，提供带 callback 的版本。



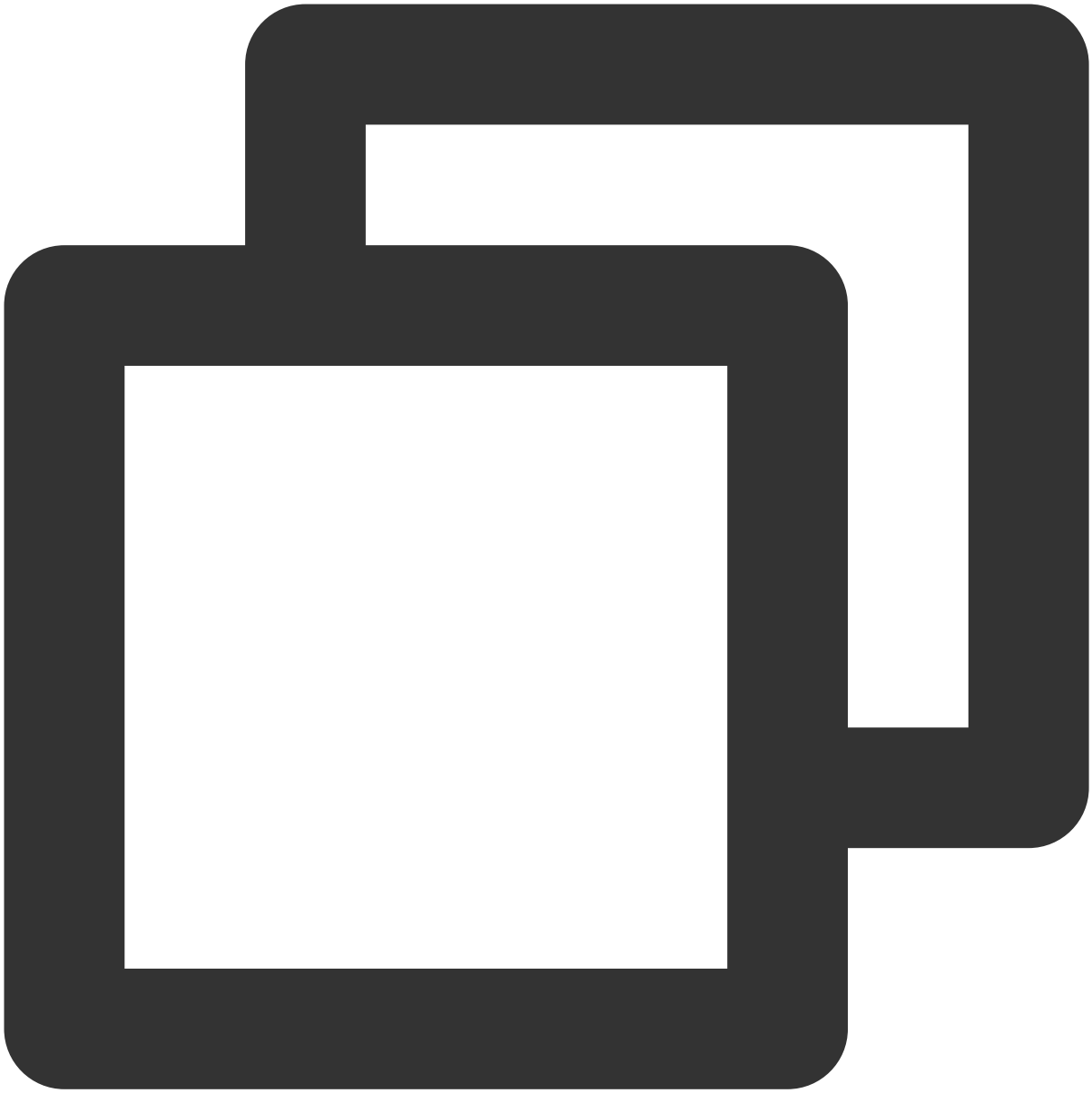
```
public static void registerPush(Context context,final XGIOperateCallback callback)
```

参数说明

context：当前应用上下文对象，不能为 null。

callback：callback 调用，主要包括操作成功和失败的回调，不能为 null。

示例代码



```
XGPushManager.registerPush(this, new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        Log.d("TPush", "注册成功，设备token为：" + data);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.d("TPush", "注册失败，错误码：" + errCode + ",错误信息：" + msg);
    }
})
```

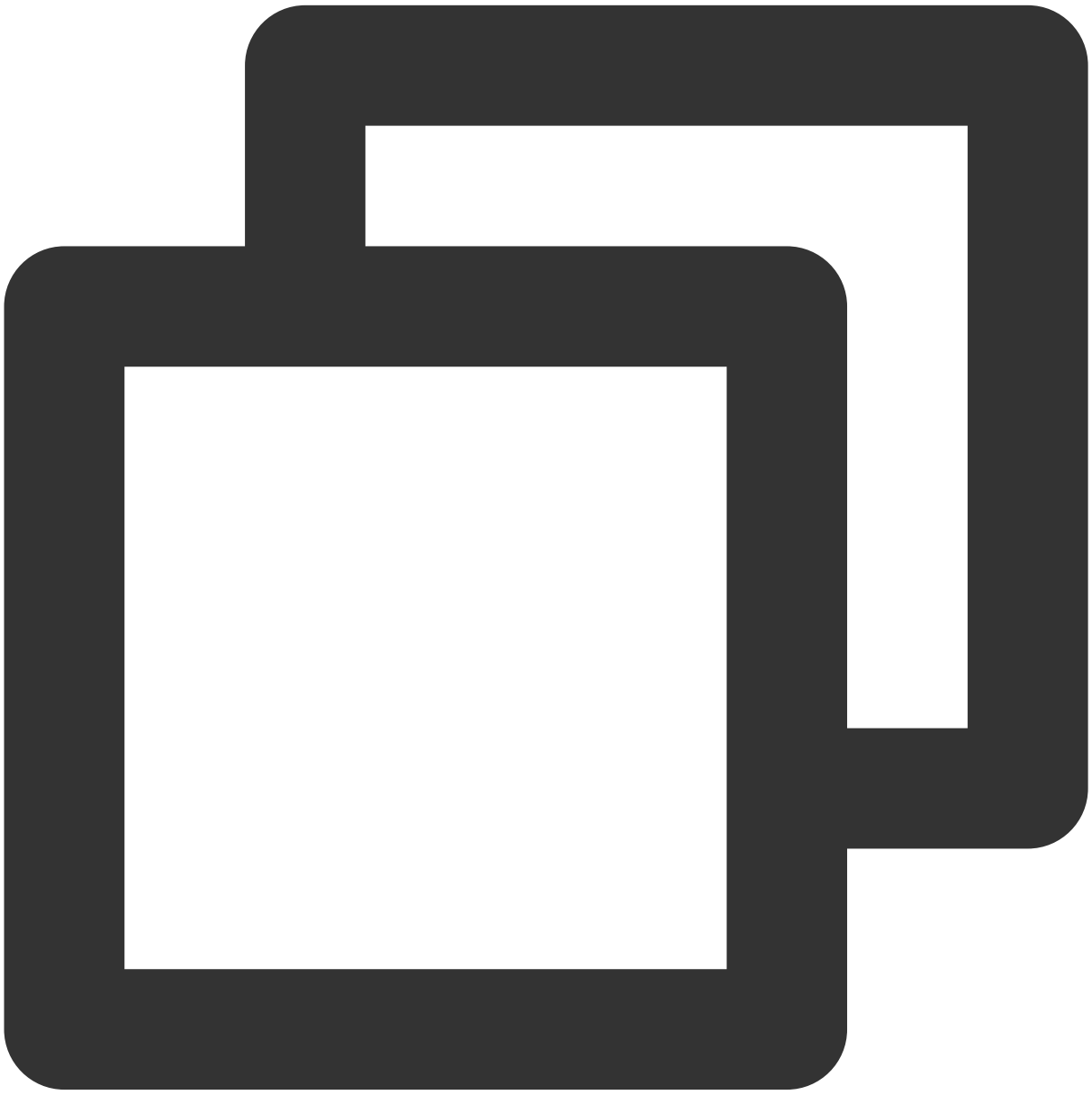
获取注册结果

有2种途径可以获取注册是否成功。

使用 Callback 版本的注册接口。

XGIOperateCallback 类提供注册成功或失败的处理接口，请参考注册接口里面的示例。

示例代码



```
/**
* 操作回调接口
*/
```

```
public interface XGIOperateCallback {
    /**
    * 操作成功时的回调。
    * @param data 操作成功的业务数据，如注册成功时的token信息等。
    * @param flag 标记码
    */
    public void onSuccess(Object data, int flag);
    /**
    * 操作失败时的回调
    * @param data 操作失败的业务数据
    * @param errCode 错误码
    * @param msg 错误信息
    */
    public void onFail(Object data, int errCode, String msg);
}
```

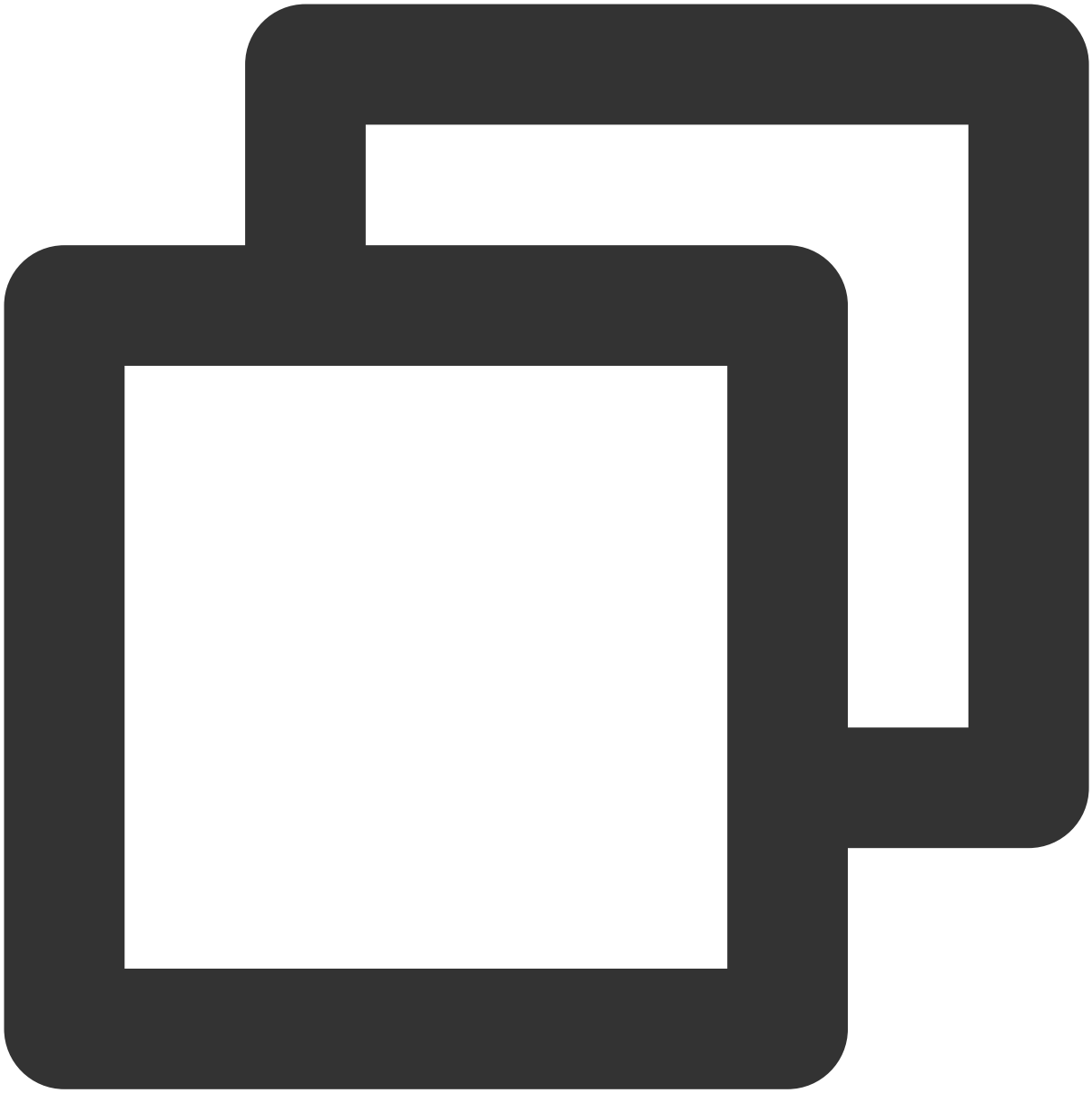
继承 XGPushBaseReceiver

可通过重写 XGPushBaseReceiver 的 onRegisterResult 方法获取。

说明：

继承 XGPushBaseReceiver 的子类 需要配置在 AndroidManifest.xml，请参考下文 消息配置。

示例代码



```
/**
*
* @param context 当前上下文
* @param errorCode 0 为成功，其它为错误码
* @param message 注册结果返回
*/
@Override
public void onRegisterResult(Context context, int errorCode, XGPushRegisterResult m
            if (context == null || message == null) {
                return;
            }
```

```
            String text = "";
            if (errorCode == XGPushBaseReceiver.SUCCESS) {        // 注册成功
                // 在这里拿token
                String token = message.getToken();
                text = "注册成功，token：" + token;
            } else {
                text = message + "注册失败，错误码：" + errorCode;
            }
            Log.d(LogTag, text);
}
```

类方法列表

方法名	返回值	默认值	描述
getToken()	String	无	设备的 Token，即设备唯一识别 ID
getAccessId()	long	0	获取注册的 AccessId
getAccount	String	无	获取注册绑定的账号
getTicket()	String	无	登录态票据
getTicketType()	short	0	票据类型

反注册

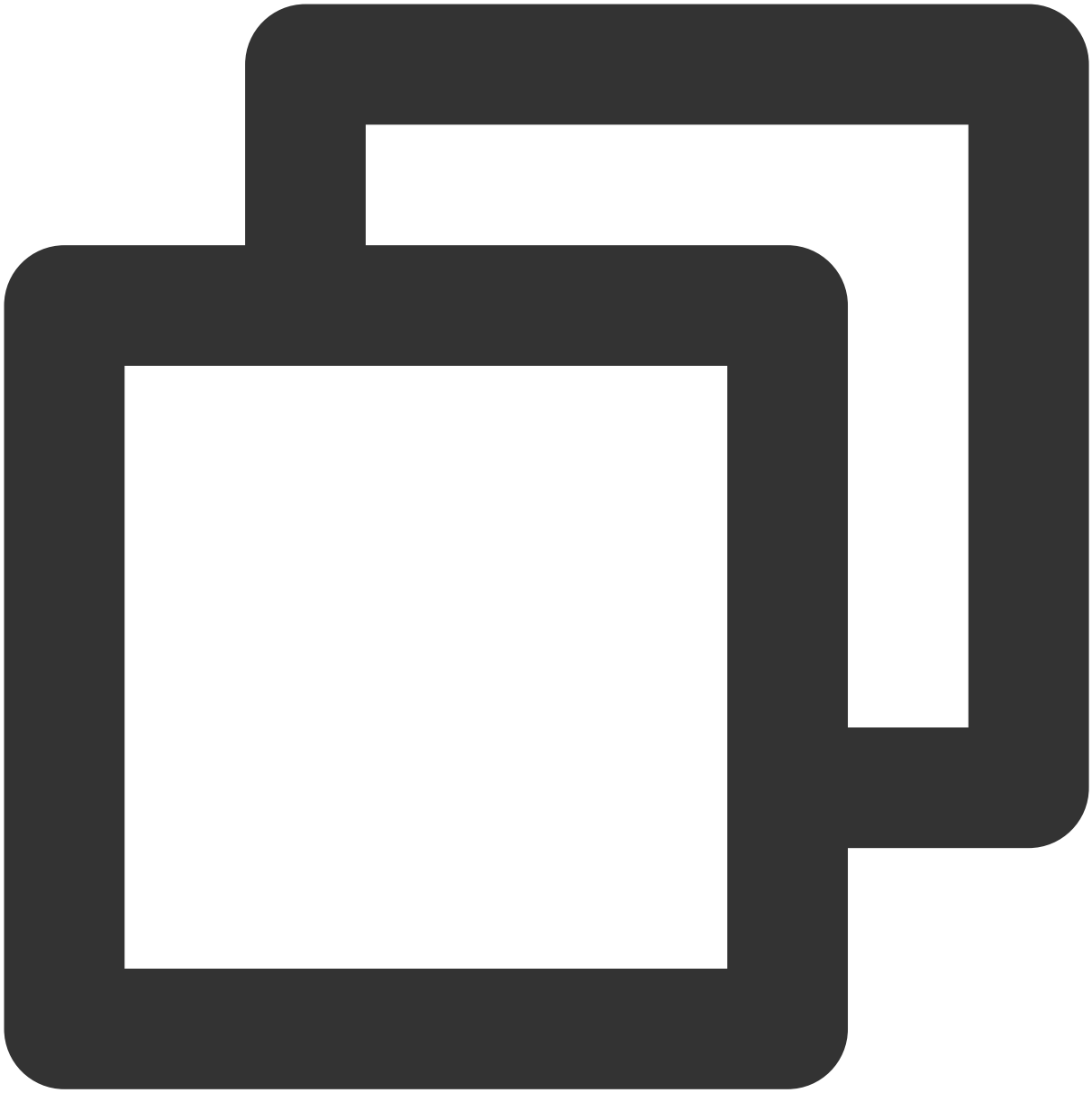
以下为反注册接口方法，若需了解调用时机及调用原理，可查看 设备反注册流程。

注意：

调用反注册接口后，需要重新调用注册接口才可接收到推送。

接口说明

当用户已退出或 App 被关闭，不再需要接收推送时，可以取消注册 App，即反注册（一旦设备反注册，直到这个设备重新注册成功期间内，下发的消息该设备都无法收到）。

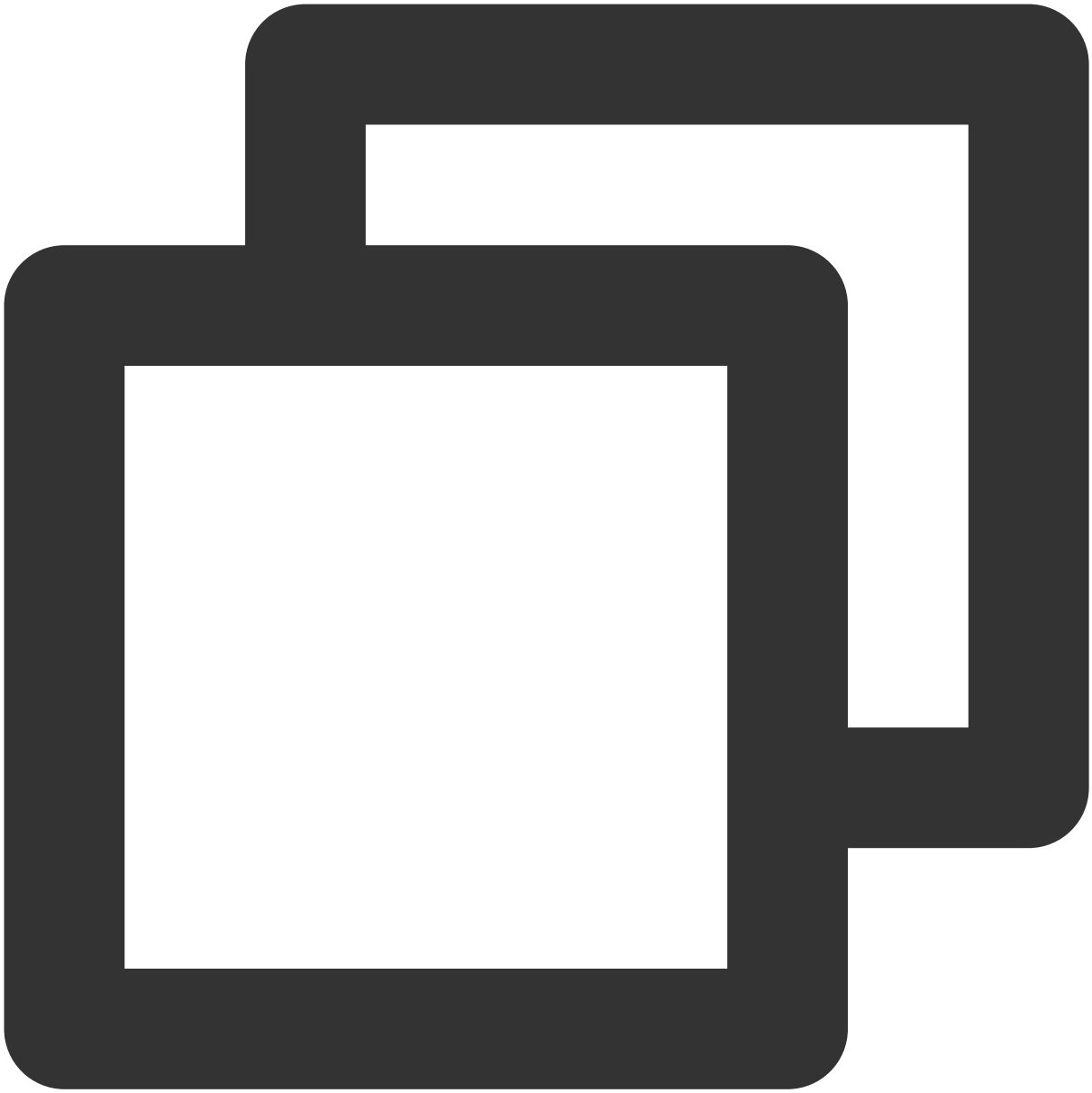


```
public static void unregisterPush(Context context)
```

参数说明

context： App 的上下文对象。

示例代码



```
XGPushManager.unregisterPush(getApplicationContext(), new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int i) {
        Log.d("TPush", "反注册成功");
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.d("TPush", "反注册失败，错误码：" + errCode + ",错误信息：" + msg);
    }
});
```

获取反注册结果

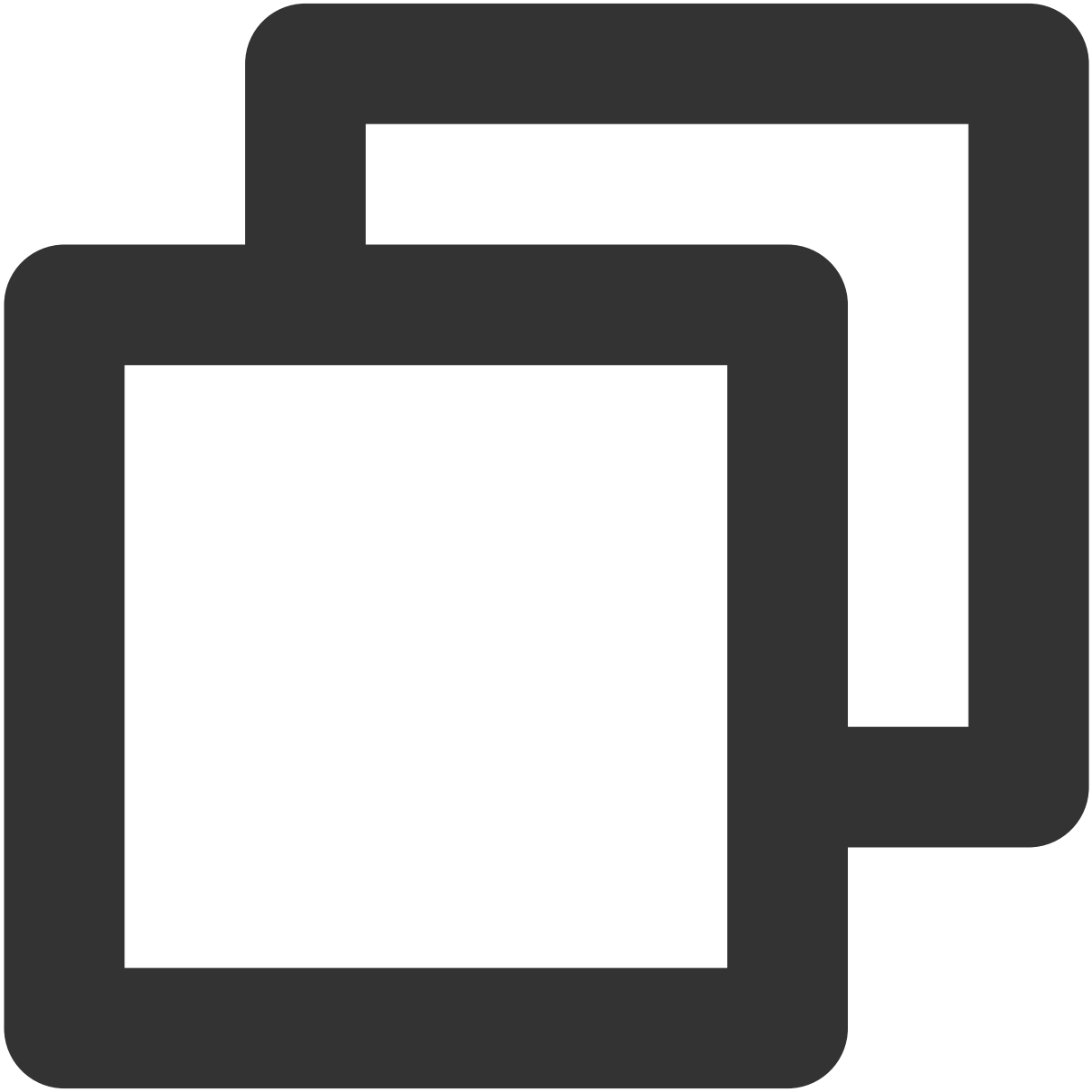
可通过重写 XGPushBaseReceiver的onUnregisterResult 方法获取。

说明：

反注册操作切勿过于频繁，可能会造成后台同步延时。

切换账号无需反注册，多次注册自动会以最后一次为准。

示例代码



```
/**
* 反注册结果
```

```
* @param context 当前上下文
* @param errorCode  为成功，其它为错误码
*/
@Override
public void onUnregisterResult(Context context, int errorCode) {
    if (context == null) {
            return;
     }
    String text = "";
    if (errorCode == XGPushBaseReceiver.SUCCESS) {
         text = "反注册成功";
    } else {
         text = "反注册失败" + errorCode;
    }
    Log.d(LogTag, text);
}
```

推送通知（展现在通知栏）

指的是在设备的通知栏展示的内容，由移动推送 SDK 完成所有的操作，App 可以监听通知被打开的行为，即在前台下发的通知，无需 App 做任何处理，默认会展示在通知栏。

说明：

成功注册移动推送服务后，通常不需要任何设置便可下发通知。

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，请考虑使用消息。

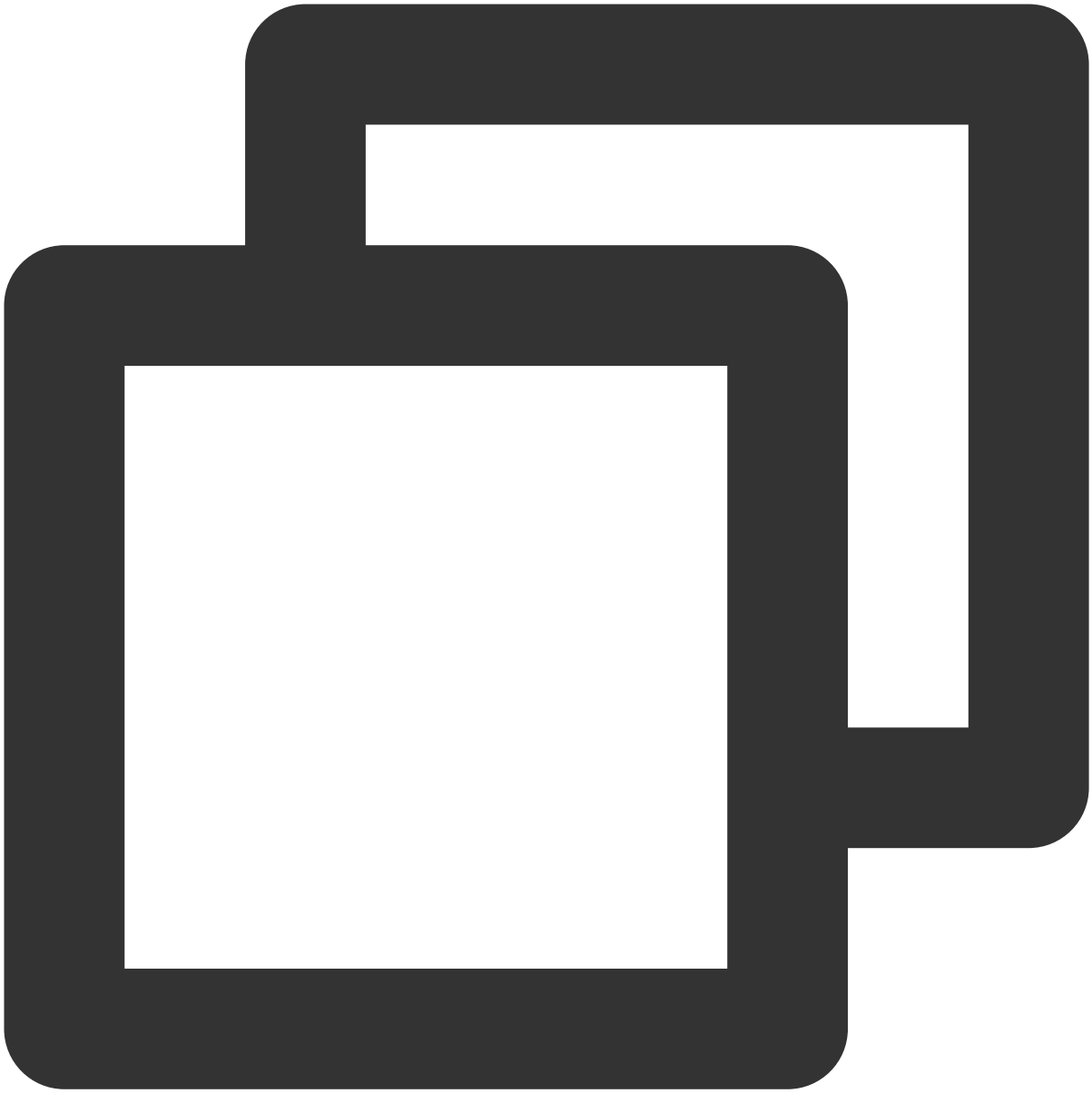
获取通知

接口说明

移动推送 SDK 提供回调接口供开发者获取抵达的通知内容，可以通过重写XGPushBaseReceiver 的 `onNotificationShowedResult(Context, XGPushShowedResult)` 方法实现。其中，XGPushShowedResult 对象提供读取通知内容的接口。

注意：

因部分厂商通道 SDK 未提供通知抵达回调方法，且当 App 进程未运行时，厂商通道的抵达回调方法无法触发。因此 SDK 内提供的回调接口 `onNotificationShowedResult` 仅支持移动推送自建通道下发通知抵达的监听，不支持厂商通道消息抵达的监听。



```
public abstract void onNotificationShowedResult(Context context,XGPushShowedResult
```

参数说明

context：当前应用上下文。

notifiShowedRlt： 抵达的通知对象。

获取通知点击结果

通知回调监听和自定义参数解析

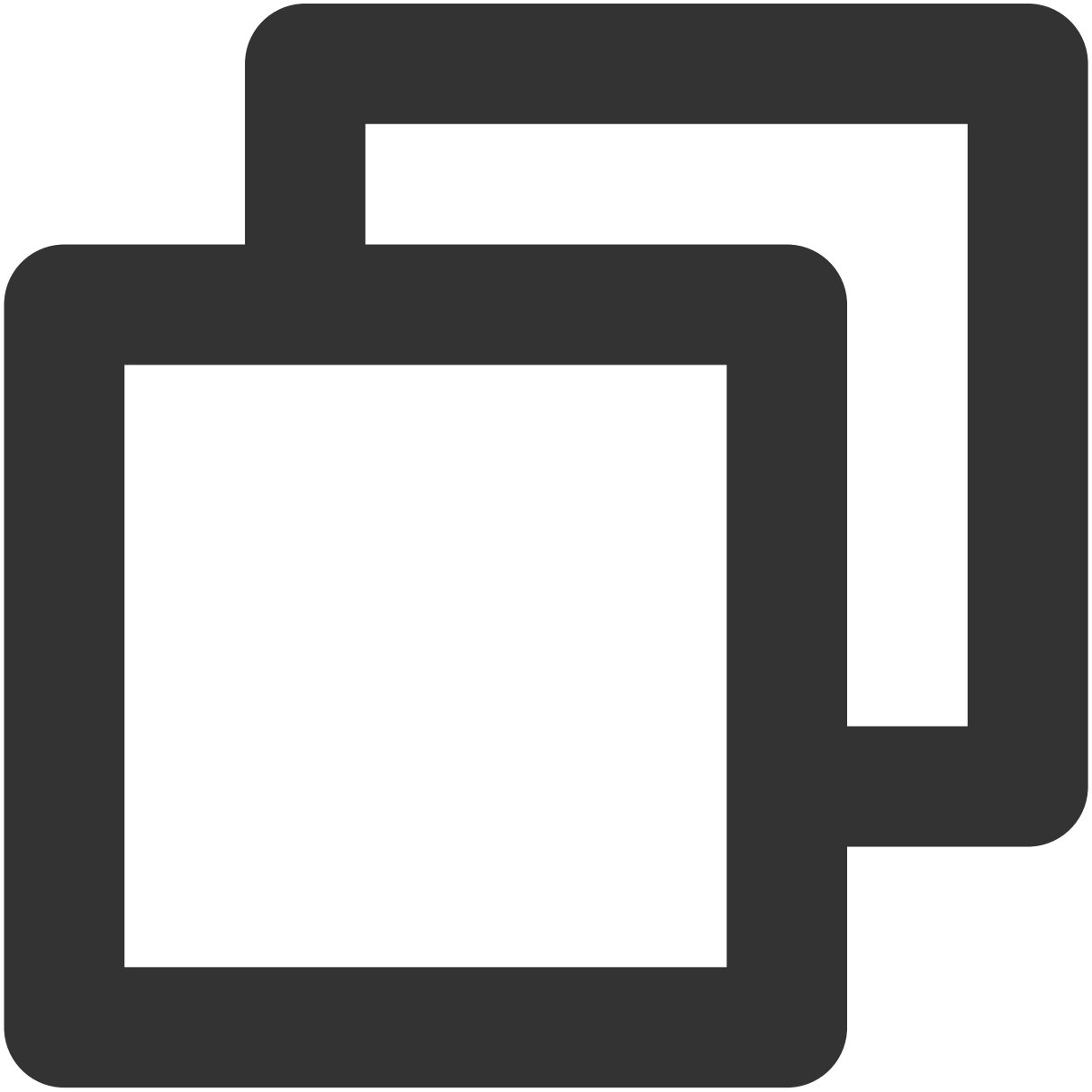
使用移动推送 SDK 默认已经统计通知/消息的抵达量、通知的点击和清除动作。SDK 提供回调接口供开发者监听通知点击事件，通过重写 XGPushBaseReceiver 的 `onNotificationClickedResult(Context, XGPushClickedResult)` 方法实现。

说明：

自 SDK 版本 v1.2.0.1 起，支持各厂商通道、移动推送自建通道下发的通知点击事件的监听。

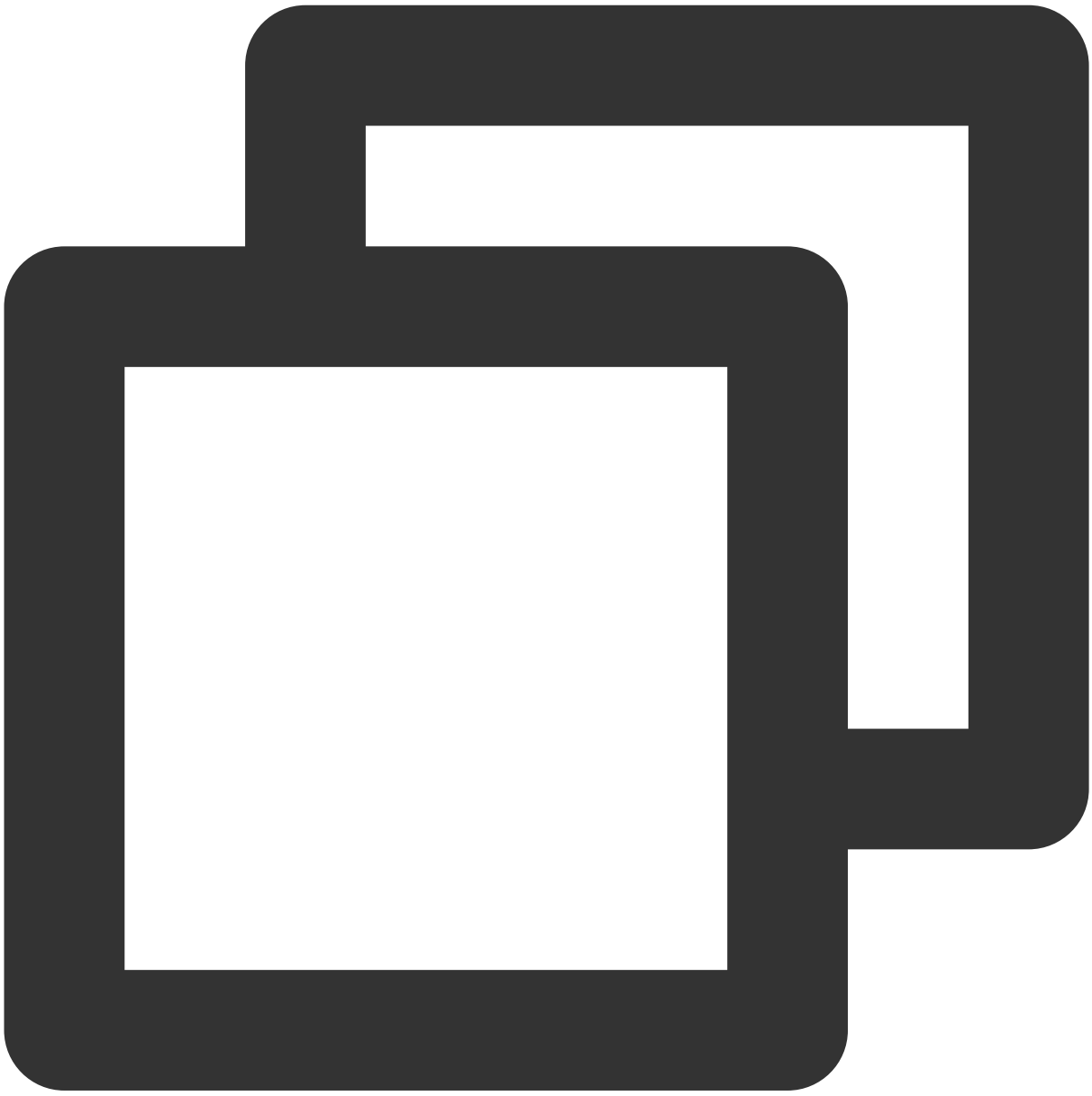
请不要此回调接口内另外做页面跳转动作，SDK 会自动按照推送任务配置的跳转动作进行通知点击跳转。如需下发并获取推送自定义参数，推荐使用 Intent 方式，请参考文档 通知点击跳转。

接口说明



```
public abstract void onNotificationClickedResult(Context context, XGPushClickedResu
```

示例代码



```
// 通知点击回调，actionType=0 为该消息被点击，actionType=2 为该消息被清除
@Override
public void onNotificationClickedResult(Context context, XGPushClickedResult messag
    if (context == null || message == null) {
        return;
    }
```

```
    String text = "";
    if (message.getActionType() == NotificationAction.clicked.getType()) {
        // 通知在通知栏被点击
        // APP自己处理点击的相关动作
        text = "通知被打开 :" + message;
    } else if (message.getActionType() == NotificationAction.delete.getType()) {
        // 通知被清除
        // APP自己处理通知被清除后的相关动作
        text = "通知被清除 :" + message;
    }

    // APP自主处理的过程。

    Log.d(LogTag, "广播接收到通知:" + text);
}
```

参数说明

context：当前应用上下文。

XGPushClickedResult：被打开的通知对象。

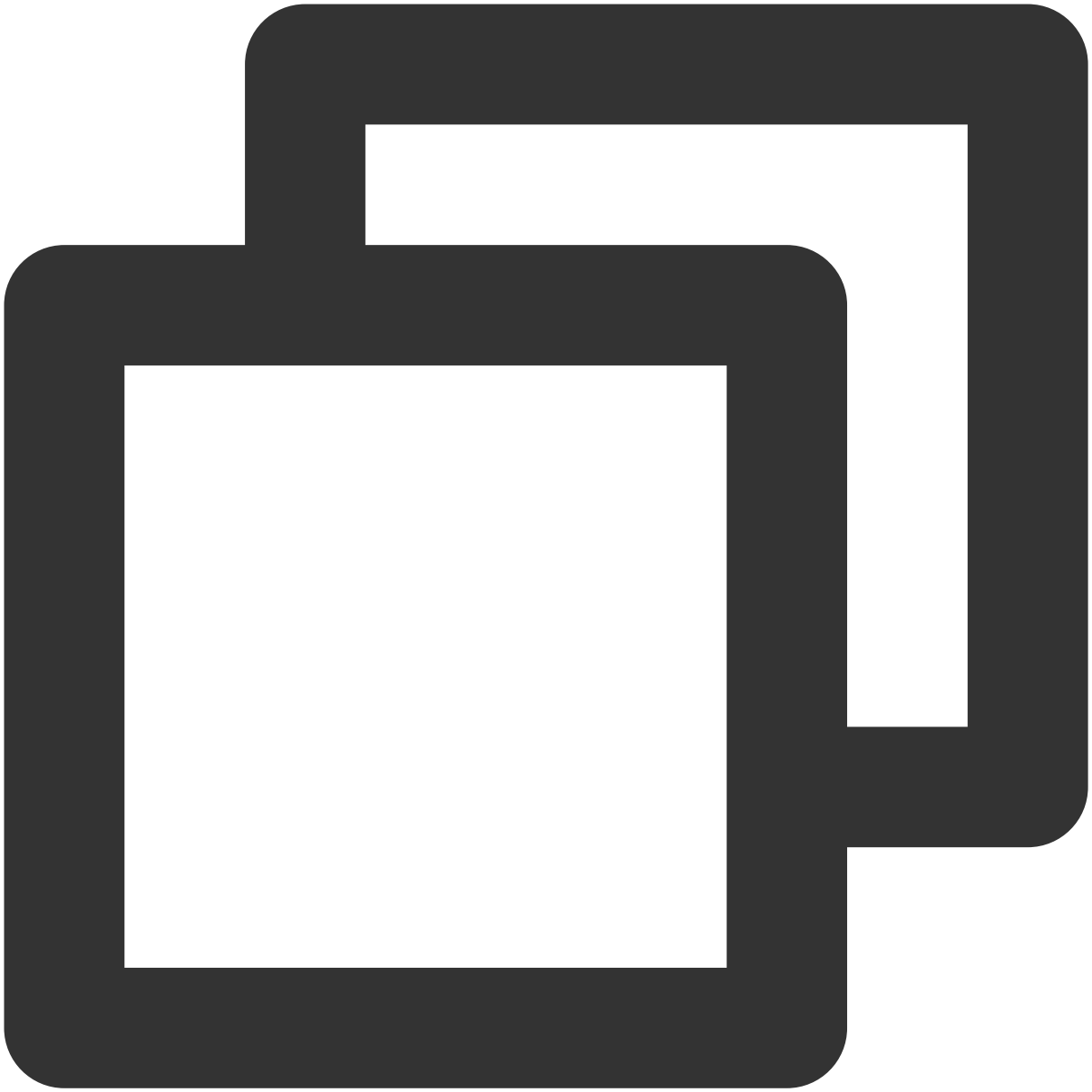
`XGPushClickedResult` 类成员方法列表：

方法名	返回值	默认值	描述
getMsgId()	long	0	消息 ID
getTitle()	String	无	通知标题
getContent()	String	无	通知正文内容
getActionType()	String	无	0 表示该通知被点击，2 表示该通知被清除
getPushChannel()	String	100	被点击通知的所下发通道标识。 100：移动推送自建通道。 101：FCM 通道。 102：华为通道。 103：小米通道。 104：vivo 通道。 105：OPPO 通道。 106：魅族通道。

清除所有通知

接口说明

清除本 App 在通知栏上的所有通知。

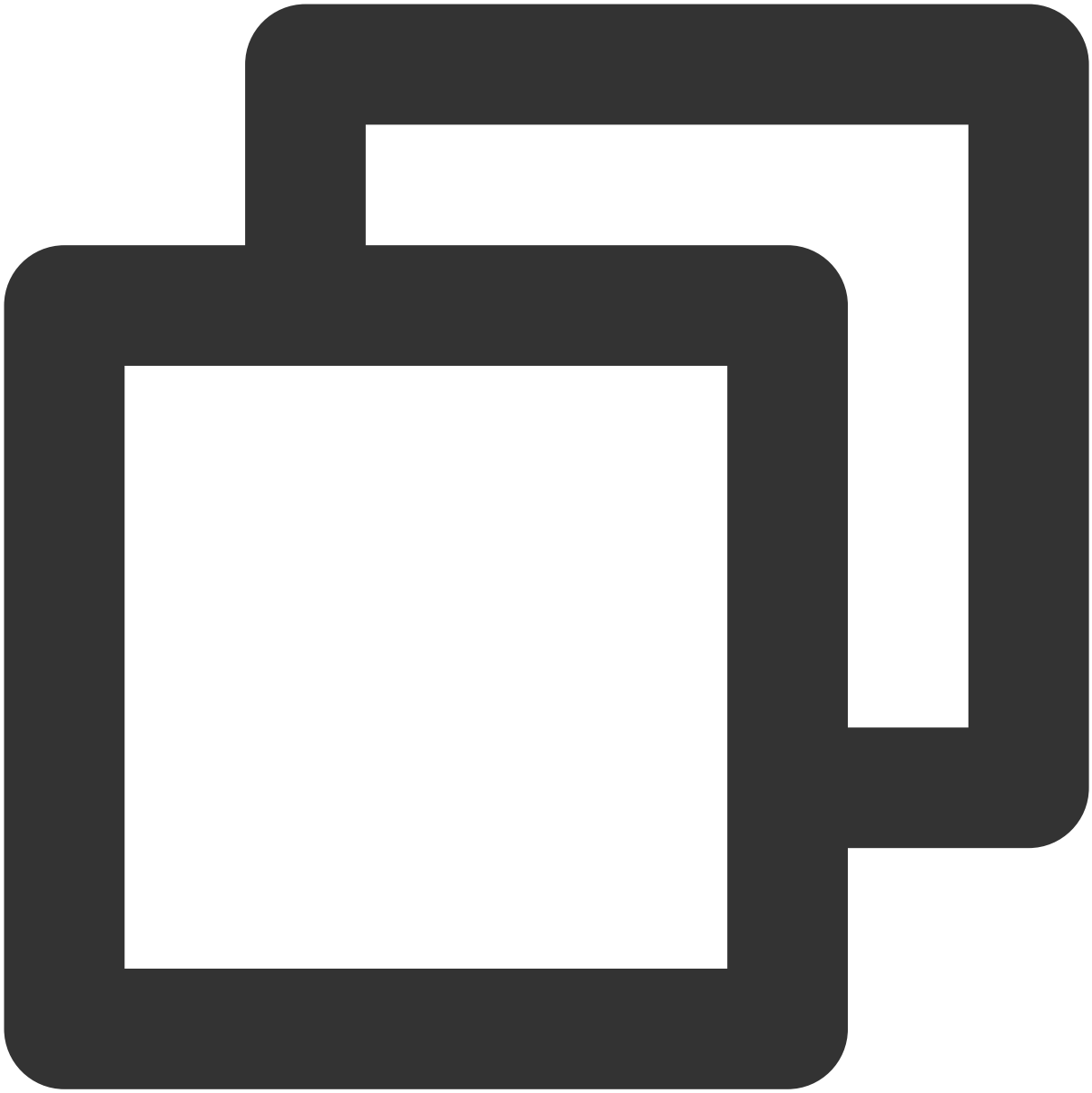


```
public static void cancelAllNotifaction(Context context)
```

参数说明

context：Context 对象。

示例代码

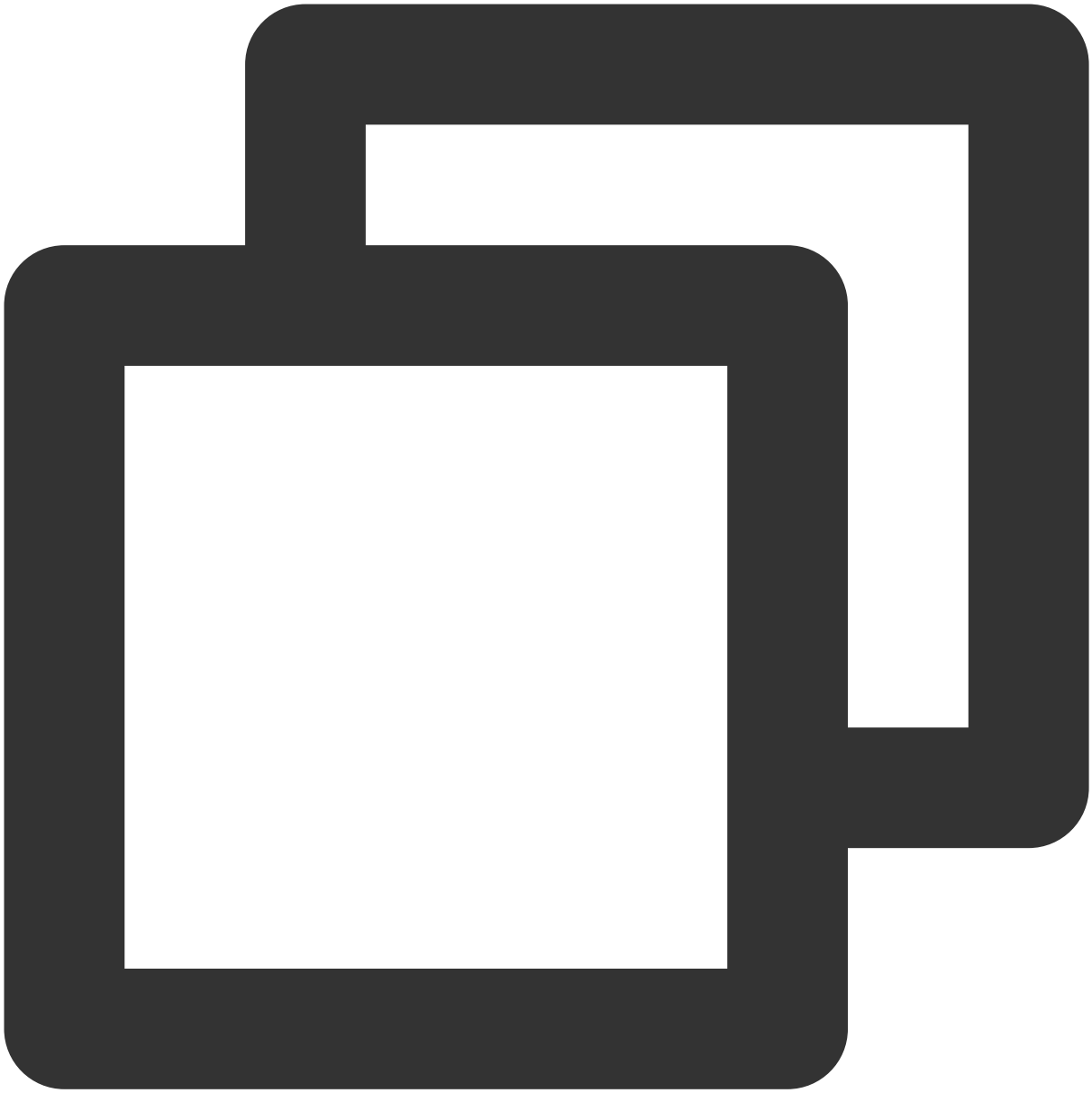


```
XGPushManager.cancelAllNotifaction(context);
```

创建通知渠道

接口说明

开发者可以创建通知 channel。



```
public static void createNotificationChannel(Context context, String channelId, Str
```

说明：

此接口仅适用于1.1.5.4及以上版本。

参数说明

context：当前应用上下文。

channelId：通知渠道 Id。

channelName：通知渠道名称。

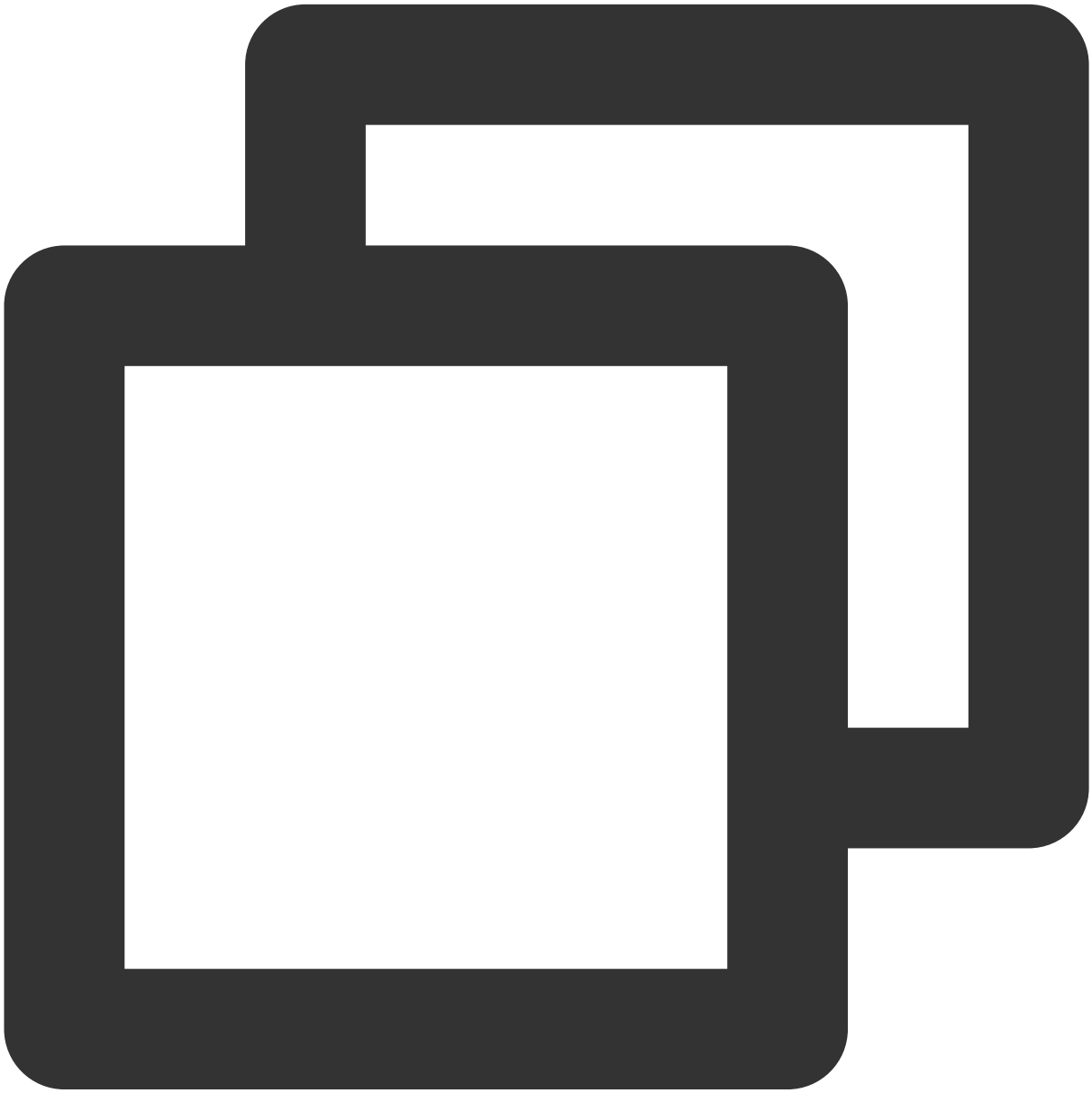
enableVibration：是否震动。

enableLights：是否有呼吸。

enableSound：是否有铃声。

soundUri ：铃声资源 Uri，enableSound 为 true 才有效，若使用系统默认铃声，则设置为 null。

示例代码



```
// 请将铃声文件放置在 Android 工程资源目录 raw 下，此处以文件 ring.mp3  为例
String uri = "android.resource://" + context.getPackageName() + "/" + R.raw.ring;
Uri soundUri = Uri.parse(uri);
```

```
XGPushManager.createNotificationChannel(context,"default_message", "默认通知", true,
```

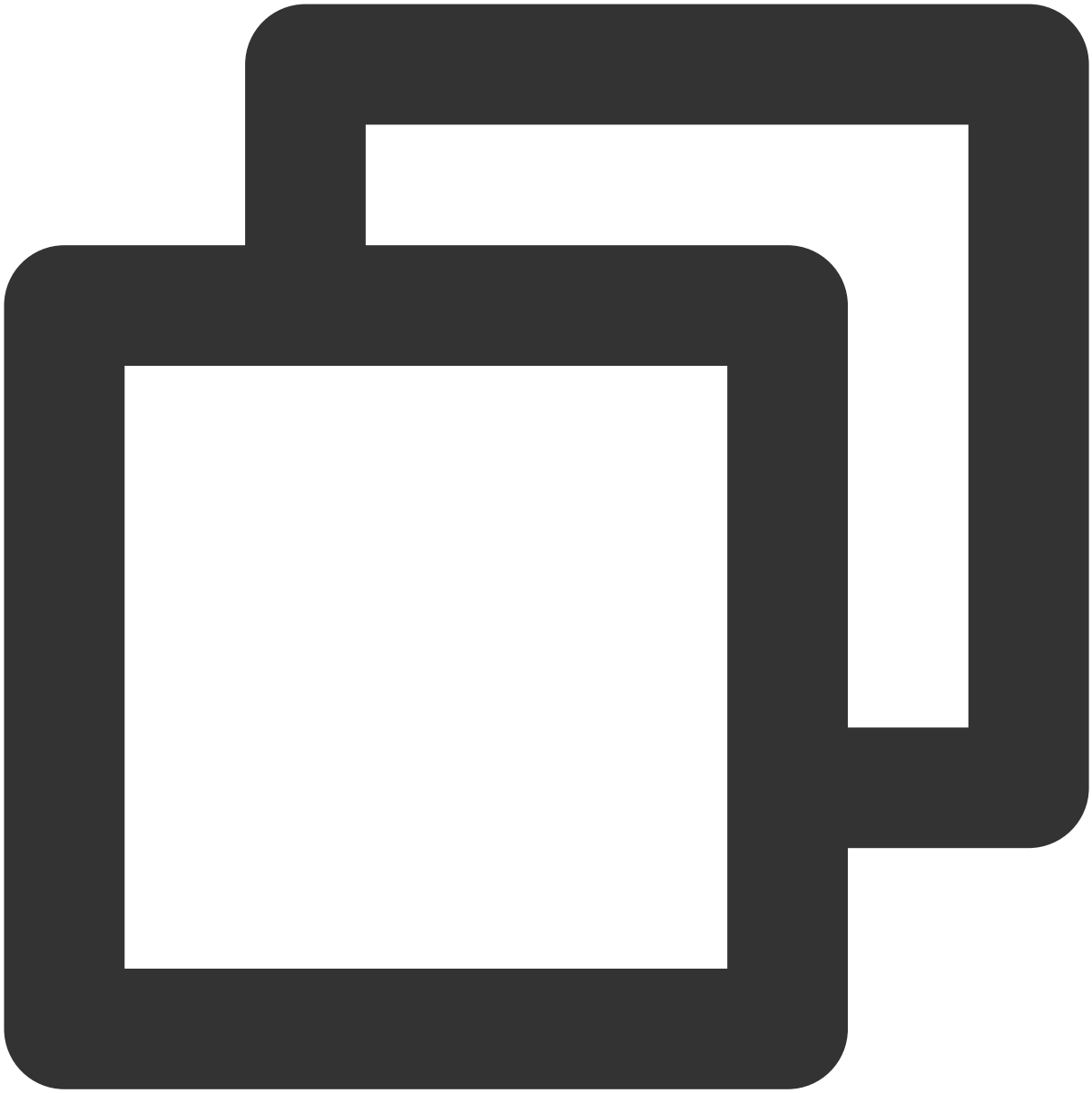
推送消息（消息不展示到通知栏）

指的是由移动推送下发给 App 的内容，需要 App 继承 XGPushBaseReceiver 接口实现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，移动推送只负责将消息从移动推送服务器下发到 App 这个过程，不负责消息的处理逻辑，需要 App 自己实现。
消息指的是由开发者通过前台或后台脚本下发的文本消息，移动推送只负责将消息传递给 App，App 完全自主负责消息体的处理。
消息具有灵活性强和高度定制性的特点，更适合 App 自主处理个性化业务需求，例如下发 App 配置信息、自定义处理消息的存储和展示等。

消息配置

请自行继承 XGPushBaseReceiver ，并且在配置文件中配置如下内容：

示例代码



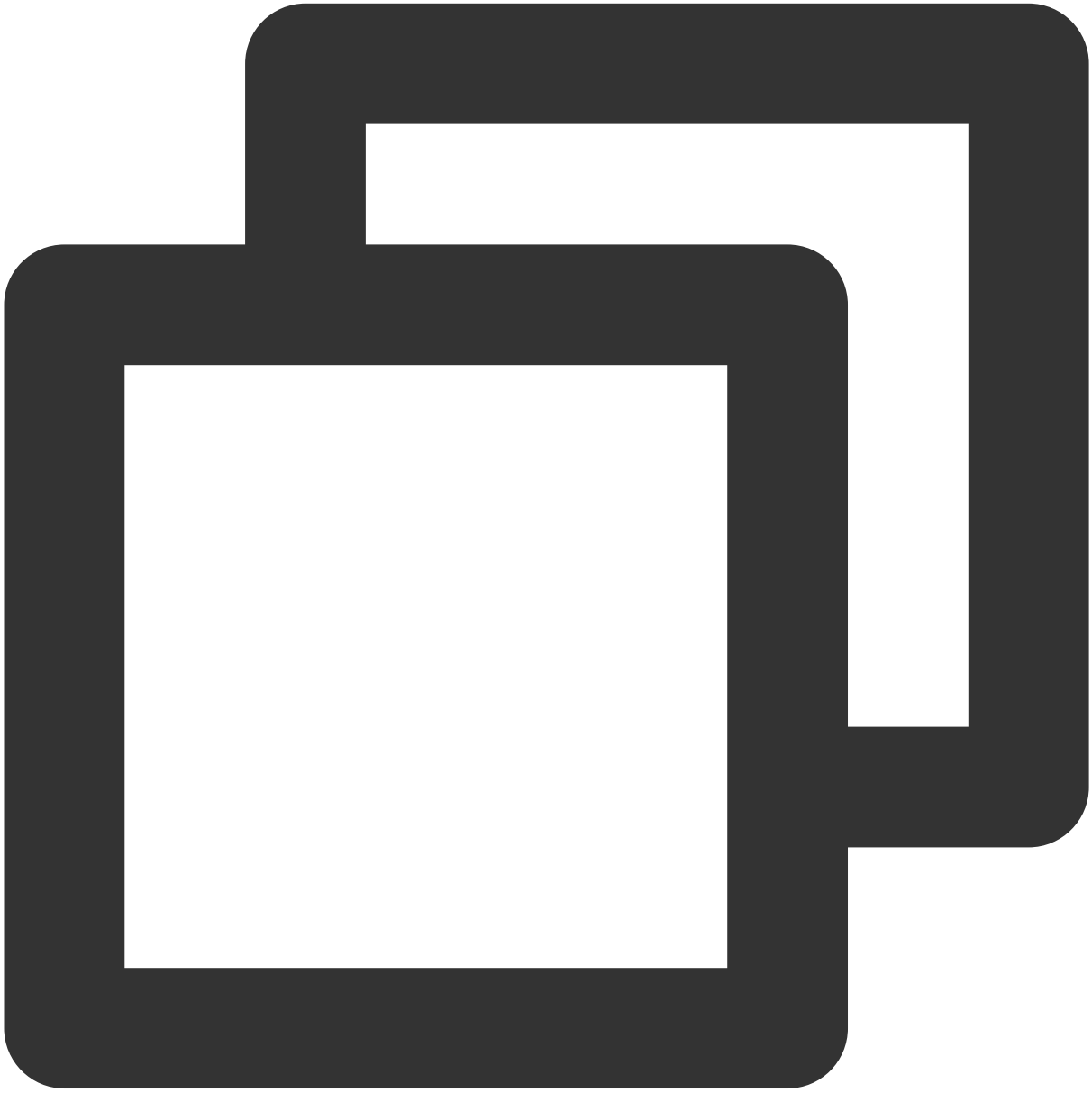
```
<receiver android:name="com.tencent.android.xg.cloud.demo.MessageReceiver">
    <intent-filter>
        <!-- 接收消息透传 -->
        <action android:name="com.tencent.android.xg.vip.action.PUSH_MESSAGE" />
        <!-- 监听注册、反注册、设置/删除标签、通知被点击等处理结果 -->
        <action android:name="com.tencent.android.xg.vip.action.FEEDBACK" />
    </intent-filter>
</receiver>
```

获取应用内消息

开发者在前台下发消息，需要 App 继承 XGPushBaseReceiver 重写 onTextMessage 方法接收，成功接收后，再根据特有业务场景进行处理。

说明：

请确保在 AndroidManifest.xml 已经注册过该 receiver，即设 YOUR_PACKAGE.XGPushBaseReceiver。



```
public void onTextMessage(Context context,XGPushTextMessage message)
```

参数说明

context：应用当前上下文。
message：接收到消息结构体。

类方法列表

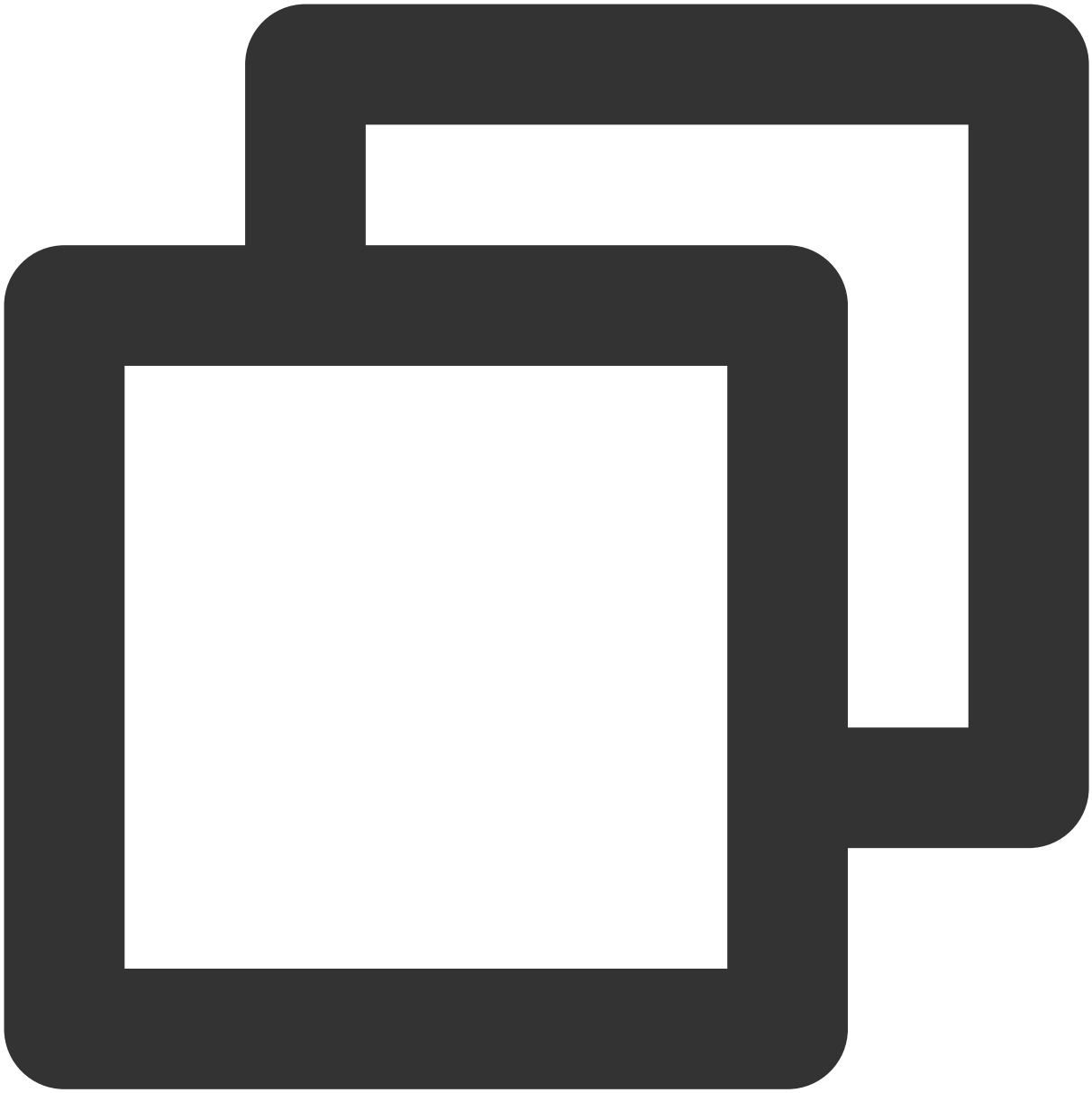
方法名	返回值	默认值	描述
getContent()	String	无	消息正文内容，通常只需要下发本字段即可
getCustomContent()	String	无	消息自定义 key-value
getTitle()	String	无	消息标题（从前台下发应用内消息字中的描述不属于标题）

应用内消息展示

SDK 1.2.7.0 新增，设置是否允许应用内消息窗口的展示，例如在允许展示应用内消息窗口的 Activity 页面设置开启，在不允许展示的 Activity 页面设置关闭。

注意：

应用内消息基于 Android WebView 框架进行展示，默认情况下，移动推送 SDK 提供的应用内消息展示 WebView 运行在 App 主进程中。**自 Android 9 起，应用无法再让多个进程共享一个 WebView 数据目录，如果您的 App 必须在多个进程中使用 WebView 实例，则您必须先使用 `WebView.setDataDirectorySuffix()` 方法为每个进程指定唯一的数据目录后缀，否则可能引起程序崩溃**。配置示例代码如下：



```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
    // 自 Android 9 起，未非 app 主进程的 WebView 实例设置不同的 WebView 数据目录
    String processName = getProcessName()
    if (processName != null
            && !processName.equals(context.getPackageName())) {
        WebView.setDataDirectorySuffix(processName)
    }
}
```

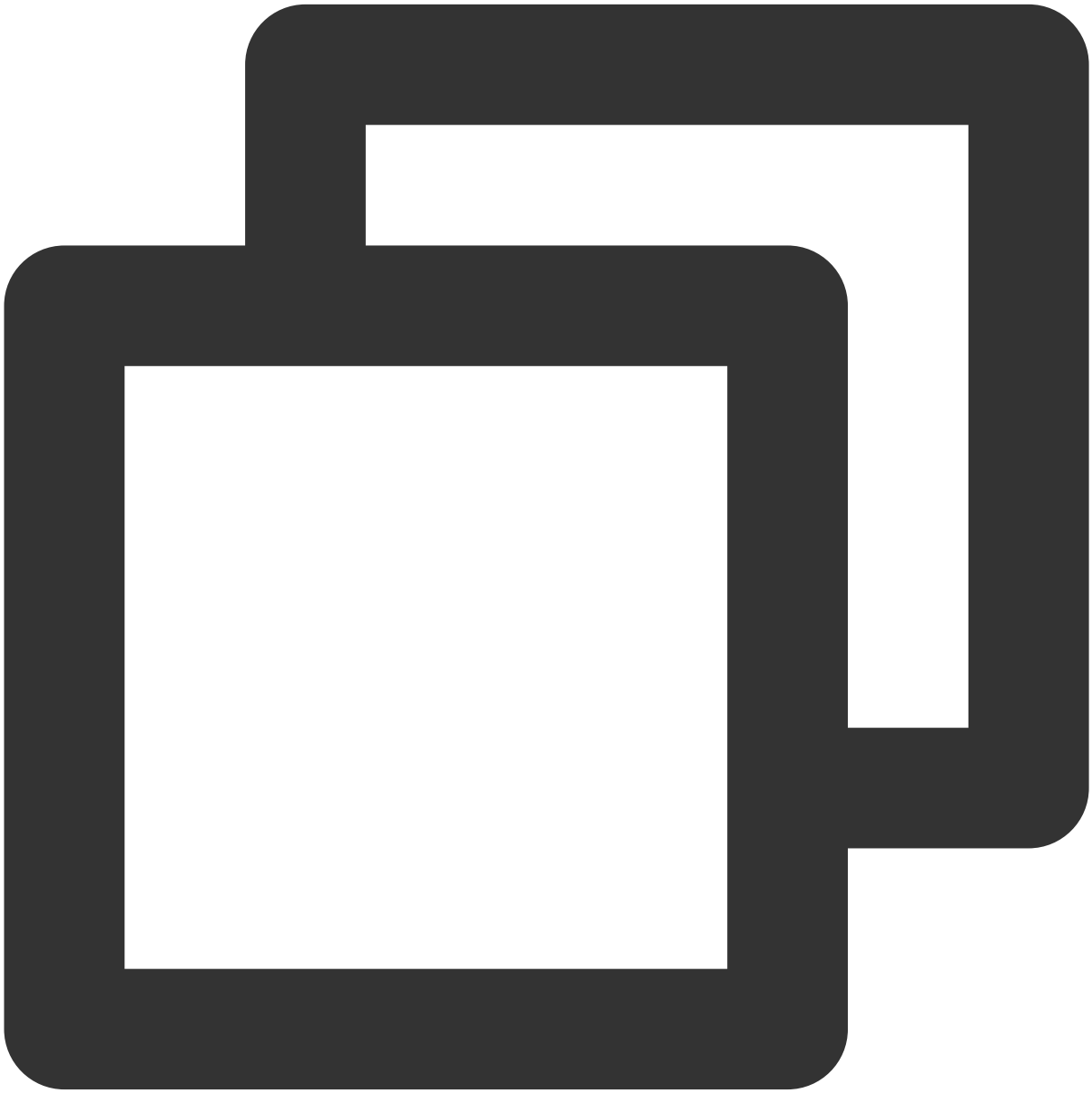
参考文档：谷歌开发者 按进程分设基于网络的数据目录。

设置是否允许展示应用内消息窗口

参数说明

context：Context 对象。
flag：是否允许应用内消息展示，true：允许，false：不允许；默认值 false。

示例代码



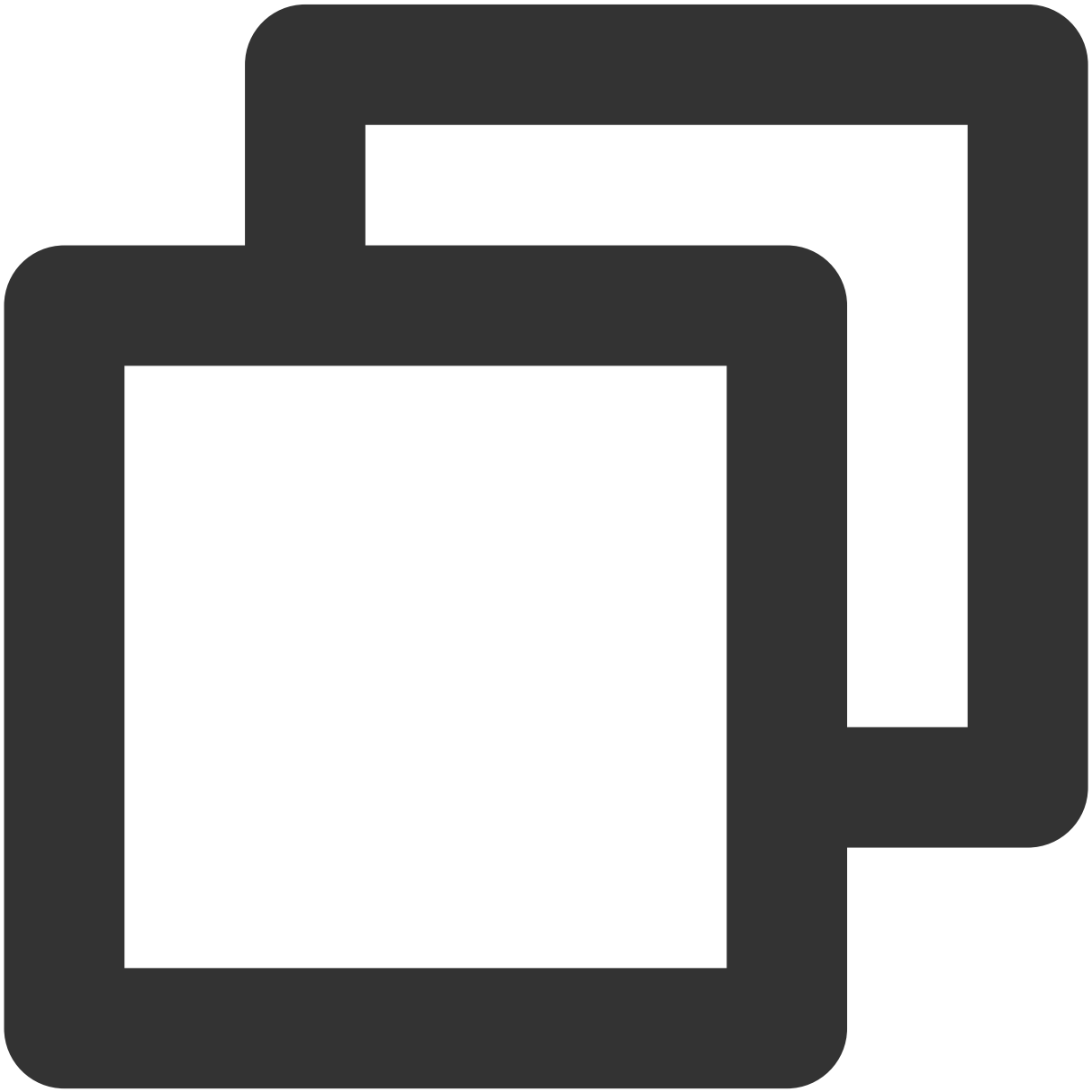
```
XGPushConfig.enableShowInMsg(context, true);
```

本地通知

增加本地通知

本地通知由用户自定义设置，保存在本地。当应用打开，移动推送 SDK Service 会根据网络心跳，判断当前是否有通知（5分钟一次）， 本地通知需要 Service 开启才能弹出，可能存在5分钟左右延时。（当设置的时间小于当前设备时间通知弹出）

示例代码

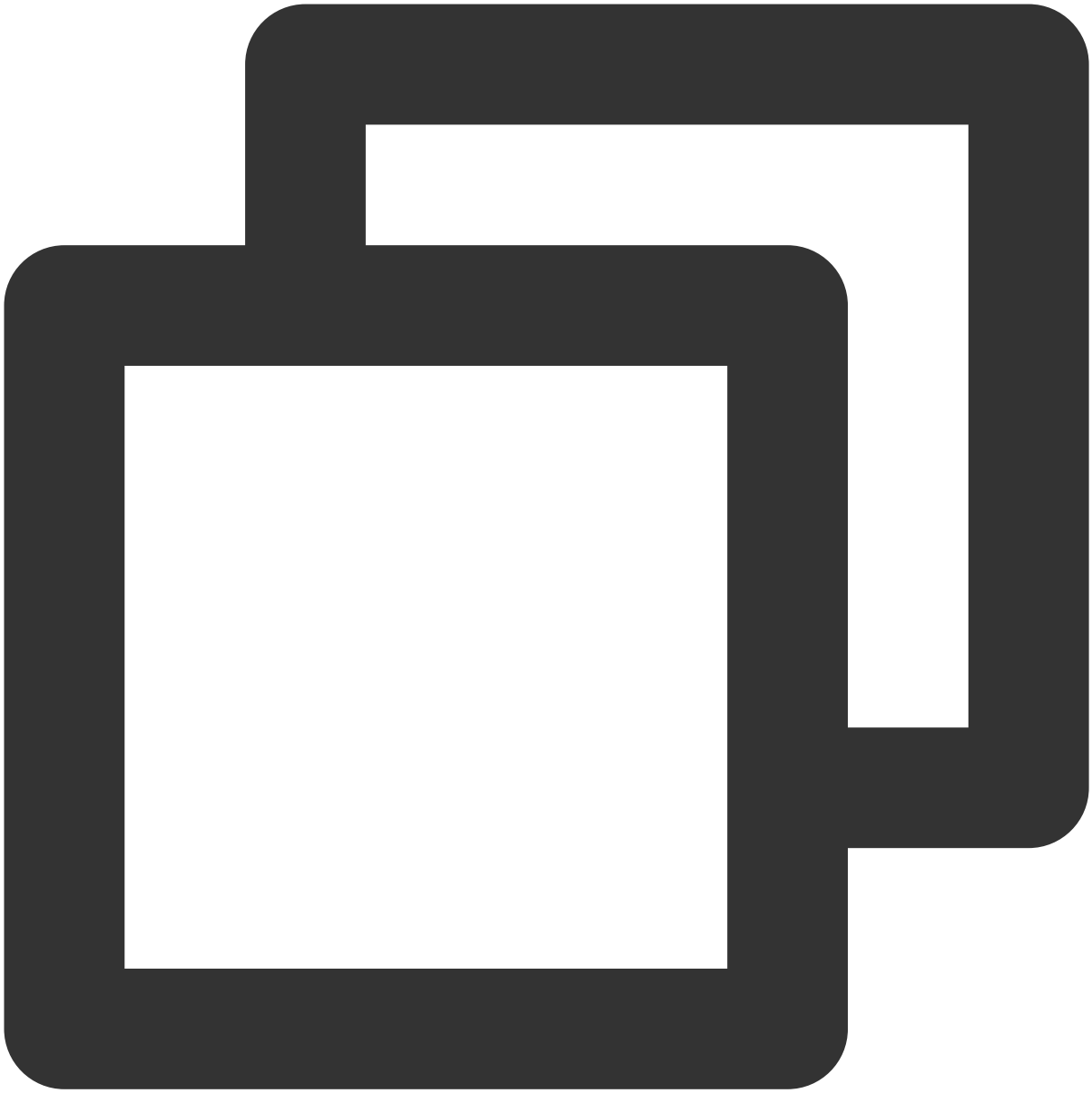


```
//新建本地通知
XGLocalMessage local_msg = new XGLocalMessage();
//设置本地消息类型，1:通知，2:消息
local_msg.setType(1);
// 设置消息标题
local_msg.setTitle("qq");
//设置消息内容
local_msg.setContent("ww");
//设置消息日期，格式为：20140502
local_msg.setDate("20140930");
//设置消息触发的小时(24小时制)，例如：22代表晚上10点
local_msg.setHour("19");
//获取消息触发的分钟，例如：05代表05分
local_msg.setMin("31");
//设置消息样式，默认为0或不设置
local_msg.setBuilderId(0);
//设置动作类型：1打开activity或App本身，2打开浏览器，3打开Intent ，4通过包名打开应用
local_msg.setAction_type(1);
//设置拉起应用页面
local_msg.setActivity("com.qq.xgdemo.SettingActivity");
// 设置URL
local_msg.setUrl("http://www.baidu.com");
// 设置Intent
local_msg.setIntent("intent:10086#Intent;scheme=tel;action=android.intent.action.DI
// 是否覆盖原先build_id的保存设置。1覆盖，0不覆盖
local_msg.setStyle_id(1);
// 设置音频资源
local_msg.setRing_raw("mm");
// 设置key,value
HashMap<String, Object> map = new HashMap<String, Object>();
map.put("key", "v1");
map.put("key2", "v2");
local_msg.setCustomContent(map);
//添加通知到本地
XGPushManager.addLocalNotification(context,local_msg);
```

清除本地通知

接口说明

清除本 App 已经创建但未弹出的本地通知。

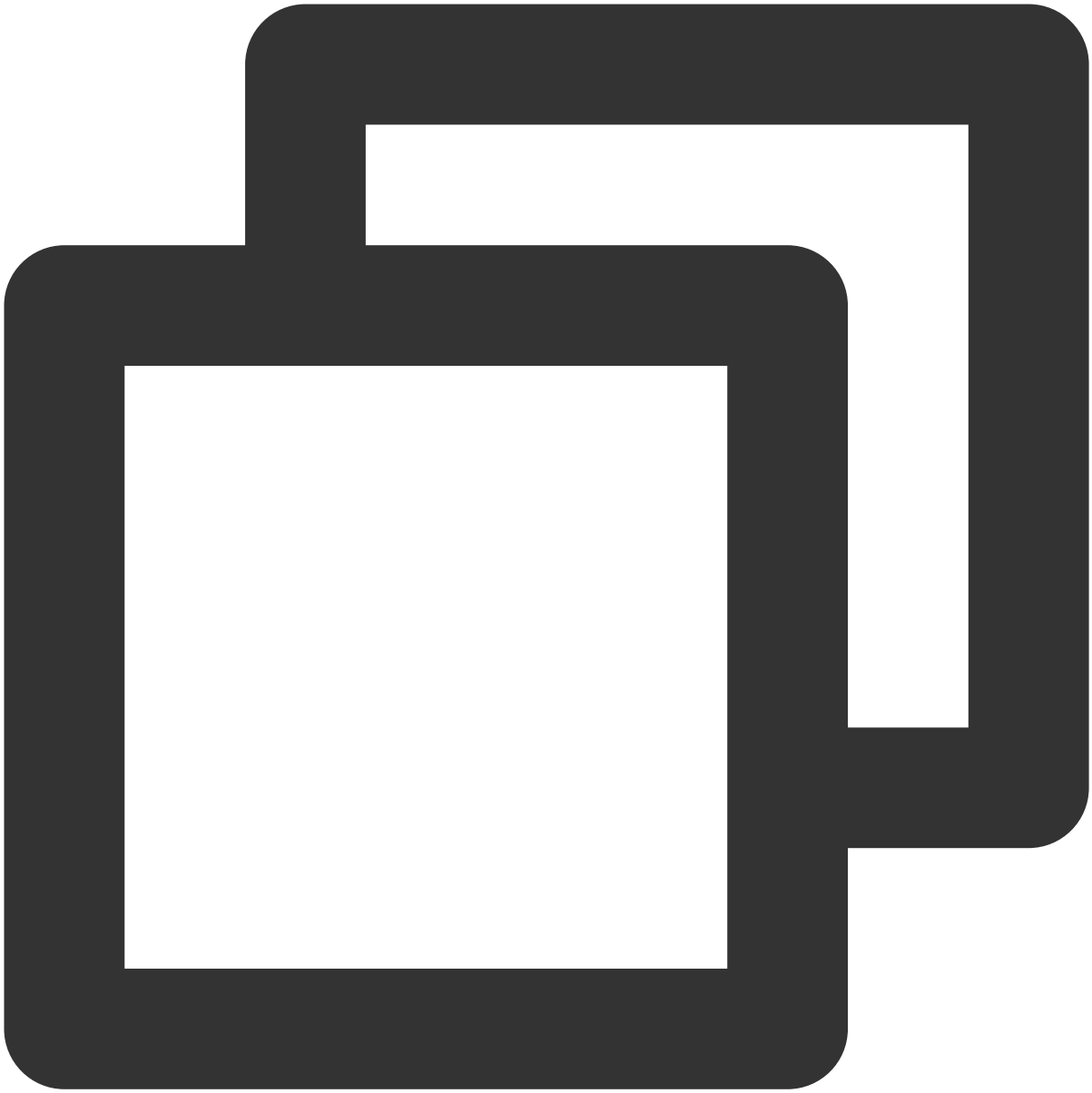


```
public static void clearLocalNotifications(Context context)
```

参数说明

context：Context 对象。

示例代码



```
XGPushManager.clearLocalNotifications(context);
```

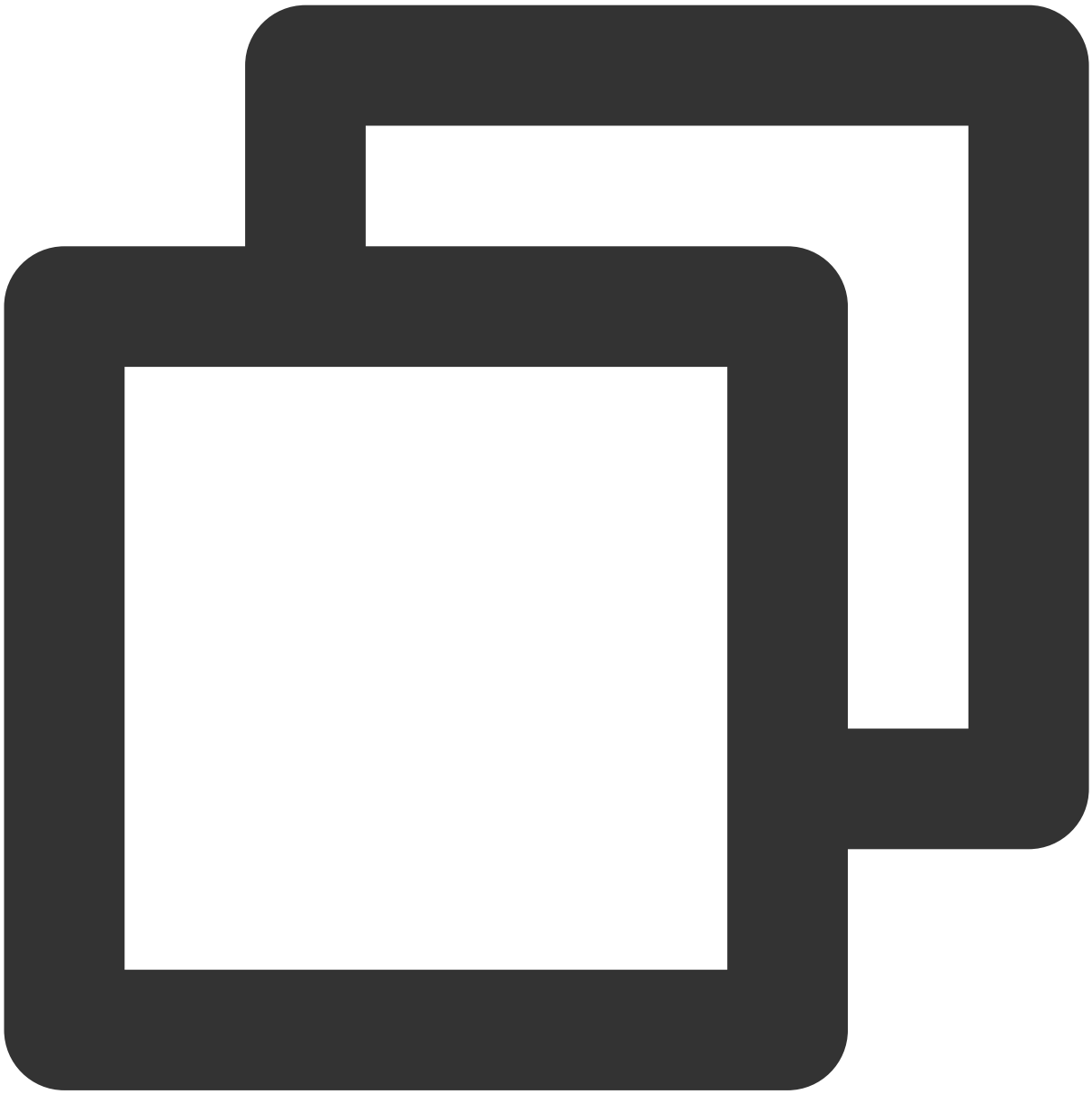
账号管理

以下为账号管理相关接口方法，若需了解调用时机及调用原理，可查看 账号相关流程。

添加账号

接口说明

添加或更新账号。若原来没有该类型账号，则添加；若原来有，则覆盖。可以同时添加多个账号，一个账号对应一个账号类型。



```
public static void upsertAccounts(Context context, List<AccountInfo> accountInfoLis
```

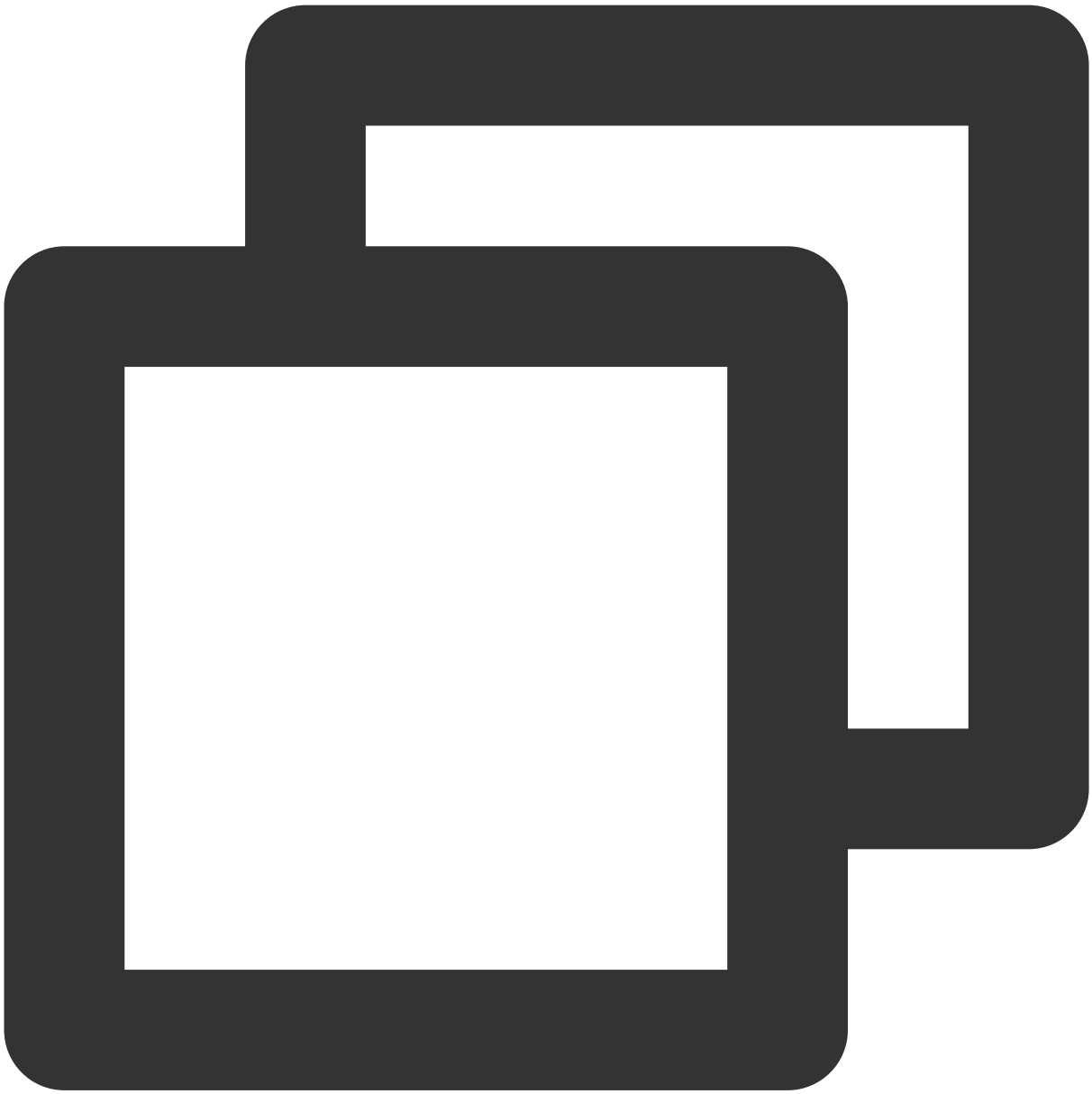
参数说明

context： Context 对象。

accountInfoList： 账号列表：账号信息包含一个账号类型和账号名称。

callback： 绑定账号操作的回调。

示例代码



```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
```

```
            Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + ms
        }
    };
    List<XGPushManager.AccountInfo> accountInfoList = new ArrayList<>();
    accountInfoList.add(new XGPushManager.AccountInfo(XGPushManager.AccountType.UNKNOWN
    XGPushManager.upsertAccounts(context, accountInfoList, xgiOperateCallback);
```

说明：

每个账号最多支持绑定100个 token。

账号可以是邮箱、手机号、用户名等任意类别的业务账号，账号类型取值可参考 账号类型取值表。

同一个账号绑定多个设备时，后台将默认推送消息到最后绑定的设备，如需推送所有绑定的设备可查看 Rest API 文档中 account_push_type 参数设置。

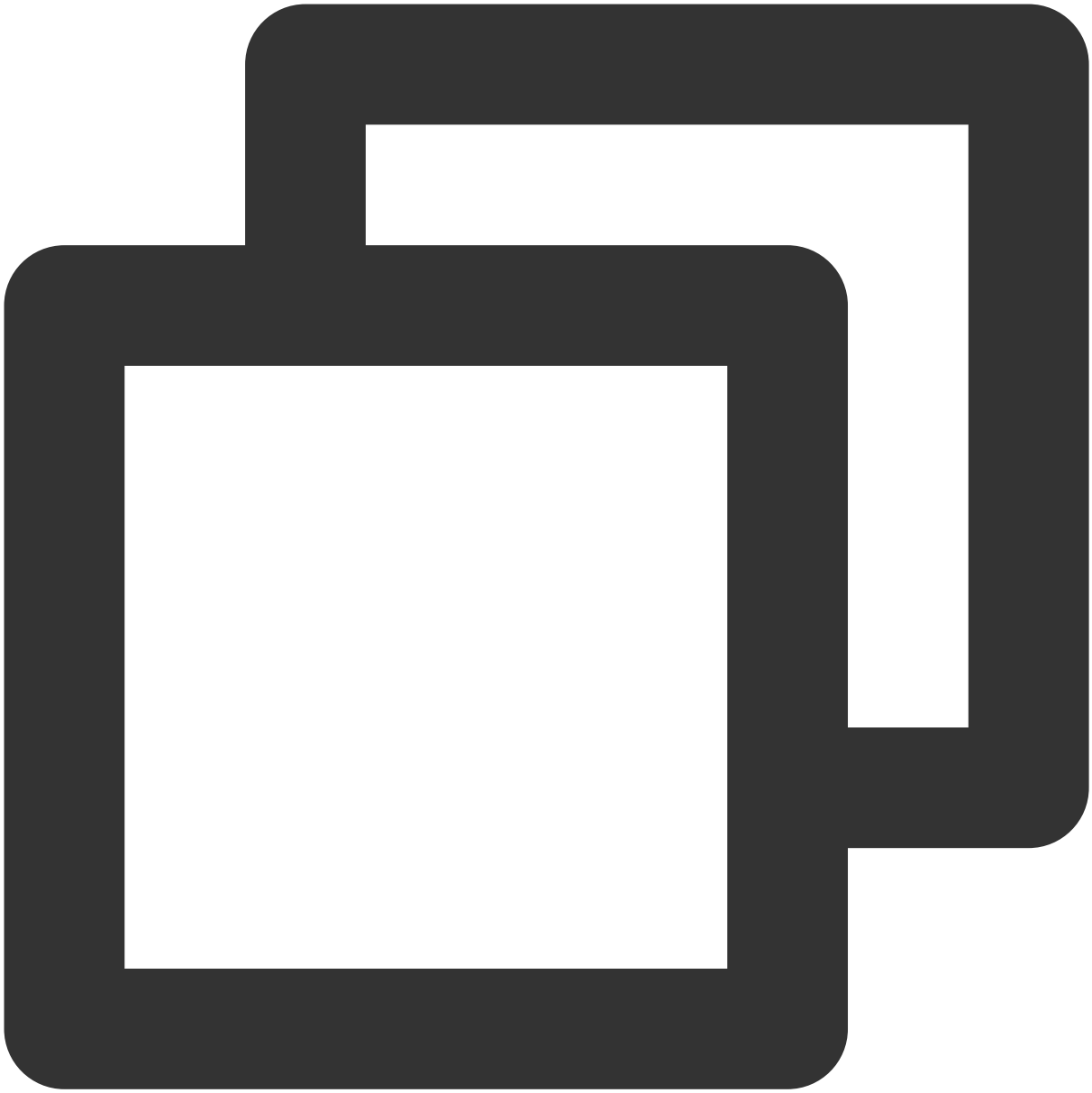
添加手机号

接口说明

添加或更新手机号码。若原来没有绑定手机号码，则绑定；若原来有，则覆盖（SDK 1.2.5.0+）

说明：

手机号格式为 `+[国家或地区码][手机号]` ，例如+8613711112222（其中前面有一个+号，86为国家或地区码，13711112222为手机号）。若绑定的手机号不带**国家或地区码**，则移动推送下发短信时自动增加+86的前缀；若带上**国家或地区码**，则按照指定的号码绑定。如需删除绑定的手机号，则需调用 `delAccountsByKeys` 接口并设置 `accountTypeSet 为 1002` 。



```
public static void upsertPhoneNumber(Context context, String phoneNumber, XGIOperat
```

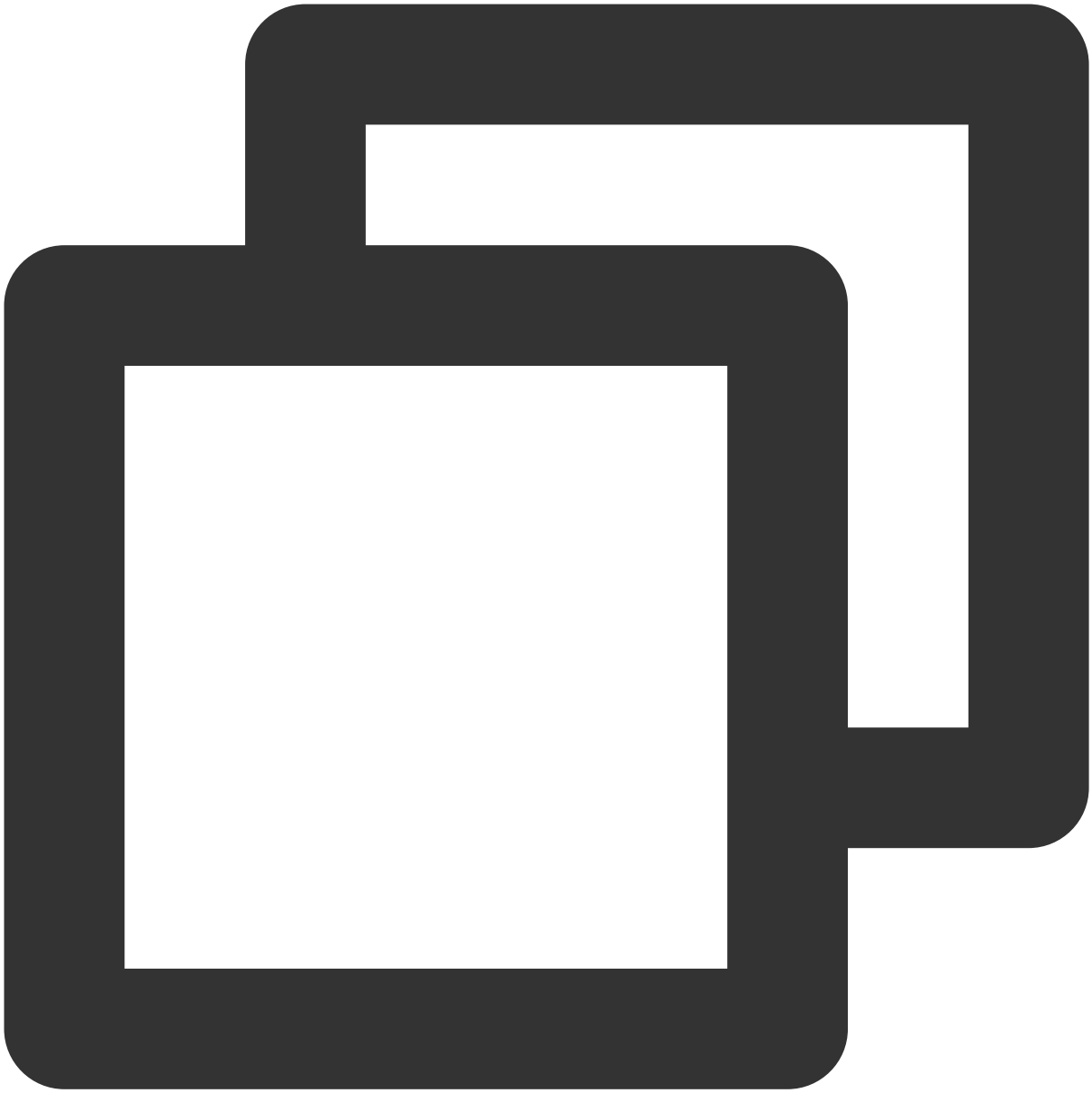
参数说明

context： Context 对象。

phoneNumber：phoneNumber E.164标准，格式为+[国家或地区码][手机号]，例如+8613711112222。SDK 内部加密传输。

callback： 绑定手机号操作的回调。

示例代码

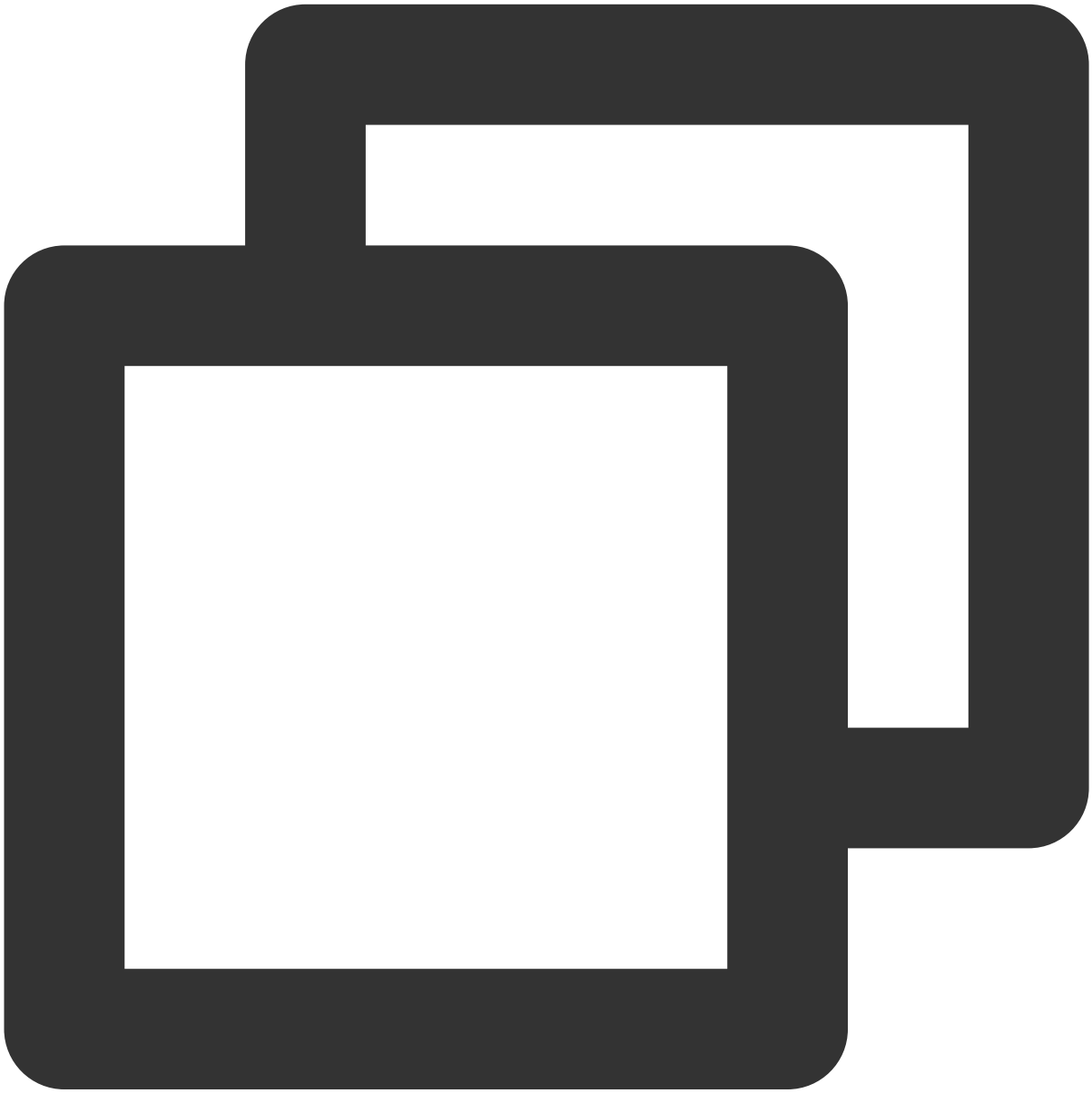


```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
  }
   @Override
  public void onFail(Object data, int errCode, String msg) {
      Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg)
  }
};
XGPushManager.upsertPhoneNumber(context, phoneNumber, xgiOperateCallback);
```

账号解绑

接口说明

对已绑定的账号进行解绑。



```
//解绑指定账号（有注册回调）
void delAccount(Context context, final String account, XGIOperateCallback callback)
//解绑指定账号（无注册回调）
void delAccount(Context context, final String account )
```

说明：

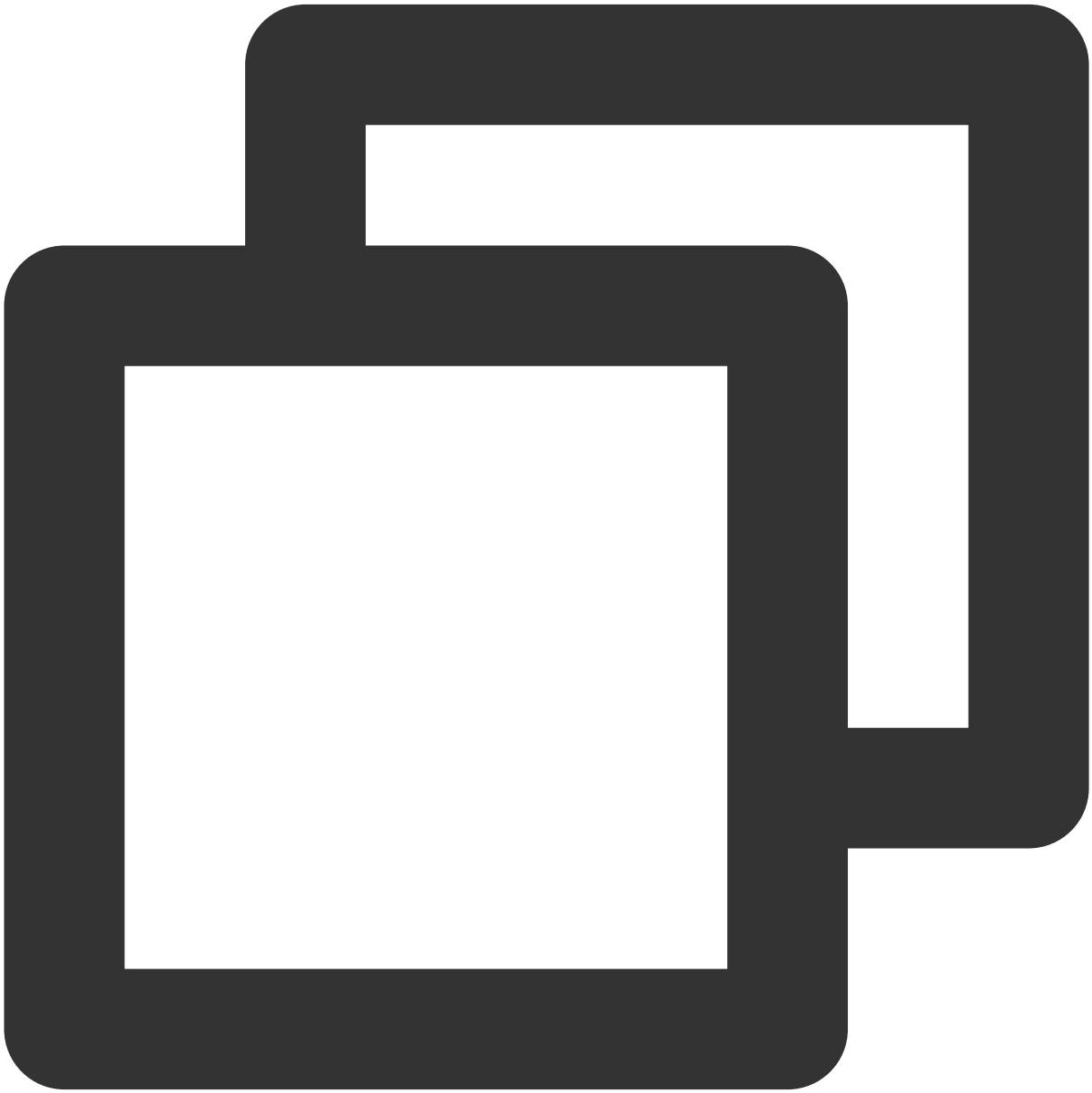
账号解绑只是解除 Token 与 App 账号的关联，若使用全量/标签/Token 推送，仍然能收到通知/消息。

参数说明

context：当前应用上下文对象，不能为 null。

account：账号。

示例代码

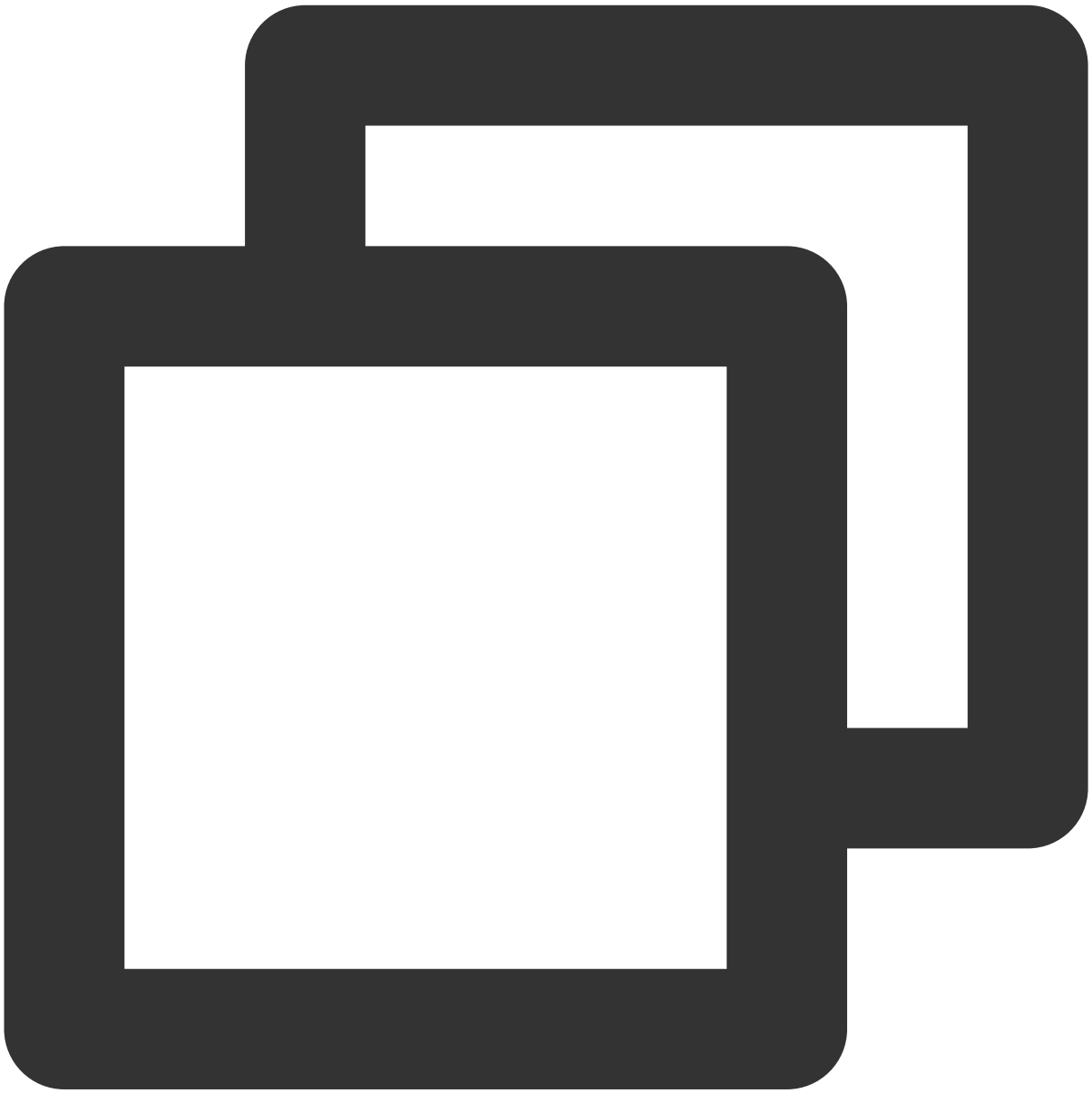


```
XGPushManager.delAccount(getApplicationContext(),"test");
```

账号类型解绑

接口说明

对一个或多个账号类型的账号进行解绑。（SDK 1.2.3.0+）



```
public static void delAccounts(Context context, final Set<Integer> accountTypeSet,
```

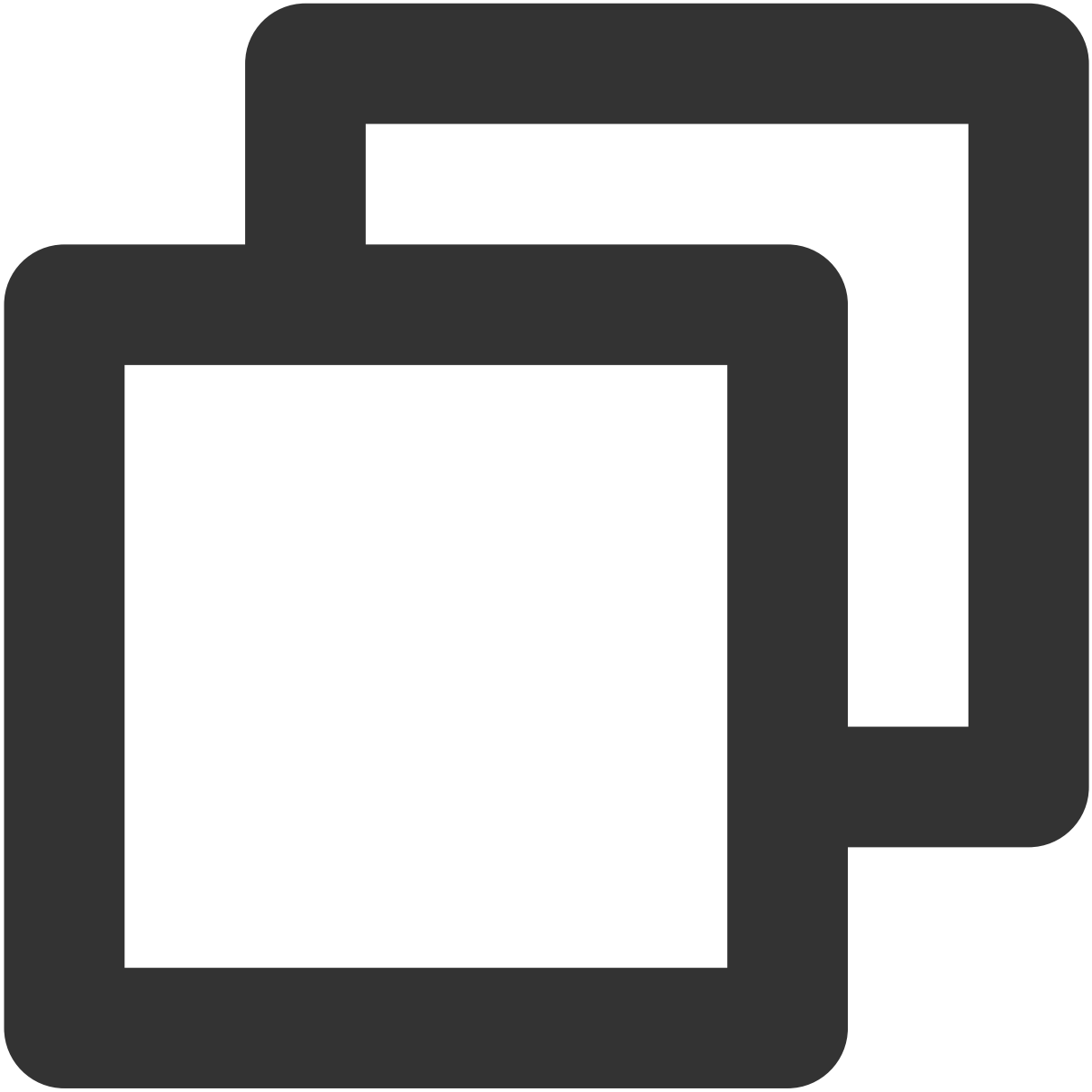
参数说明

context： Context 对象。

accountTypeSet： 需解绑账号的账号类型。

callback： 账号解绑操作的回调。

示例代码



```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
  }
  @Override
```

```
    public void onFail(Object data, int errCode, String msg) {
        Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg)
    }
};

Set<Integer> accountTypeSet = new HashSet<>();
accountTypeSet.add(XGPushManager.AccountType.CUSTOM.getValue());
accountTypeSet.add(XGPushManager.AccountType.IMEI.getValue());
XGPushManager.delAccounts(context, accountTypeSet, xgiOperateCallback);
```

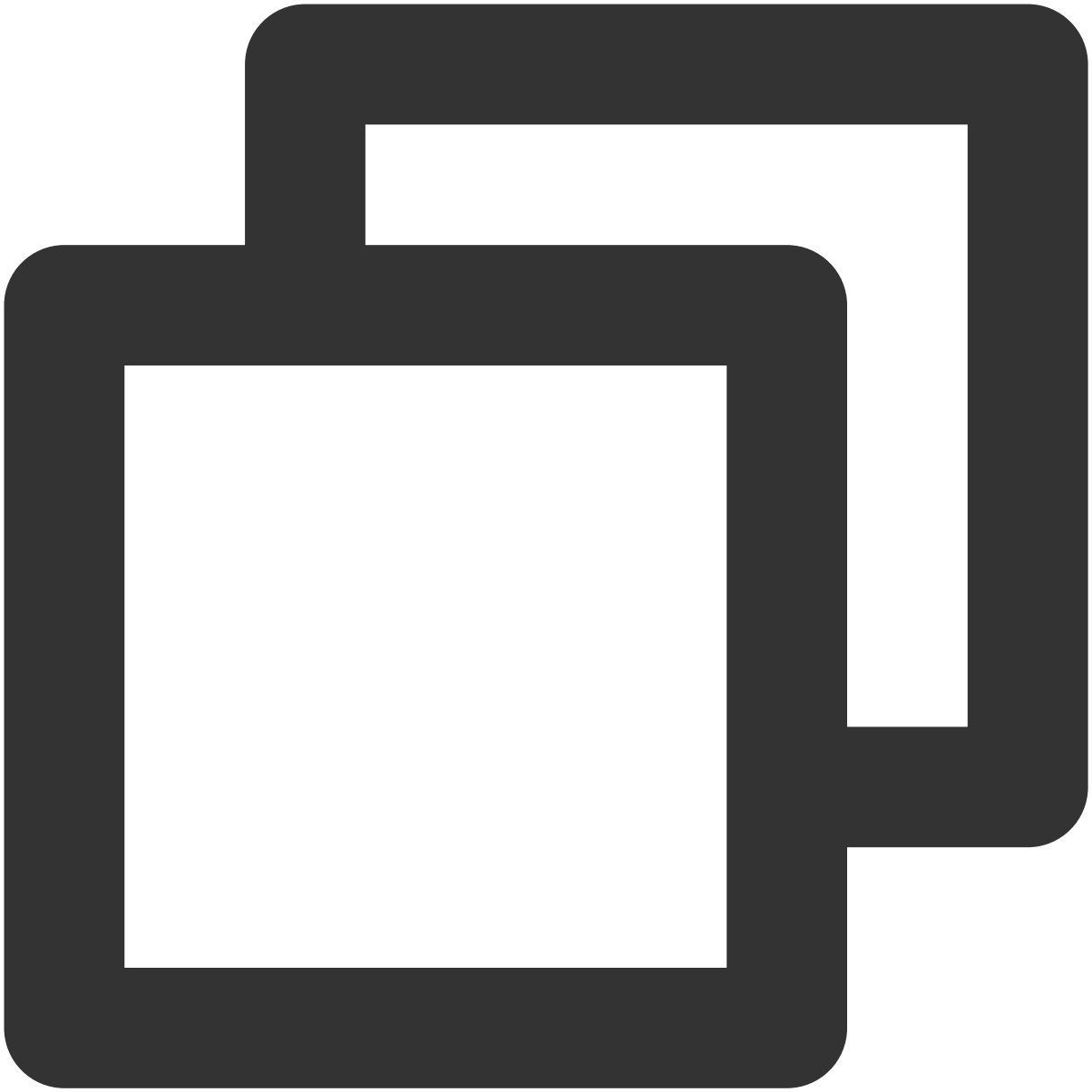
清空所有账号

说明：

SDK 1.2.2.0 版本废弃 delAllAccount 接口，推荐使用 clearAccounts 接口。

接口说明

对的所有已绑定账号进行解绑。



```
//解绑所有的账号信息（有注册回调）
void clearAccounts(Context context, XGIOperateCallback callback)
//解绑所有的账号信息（无注册回调）
void clearAccounts(Context context)
```

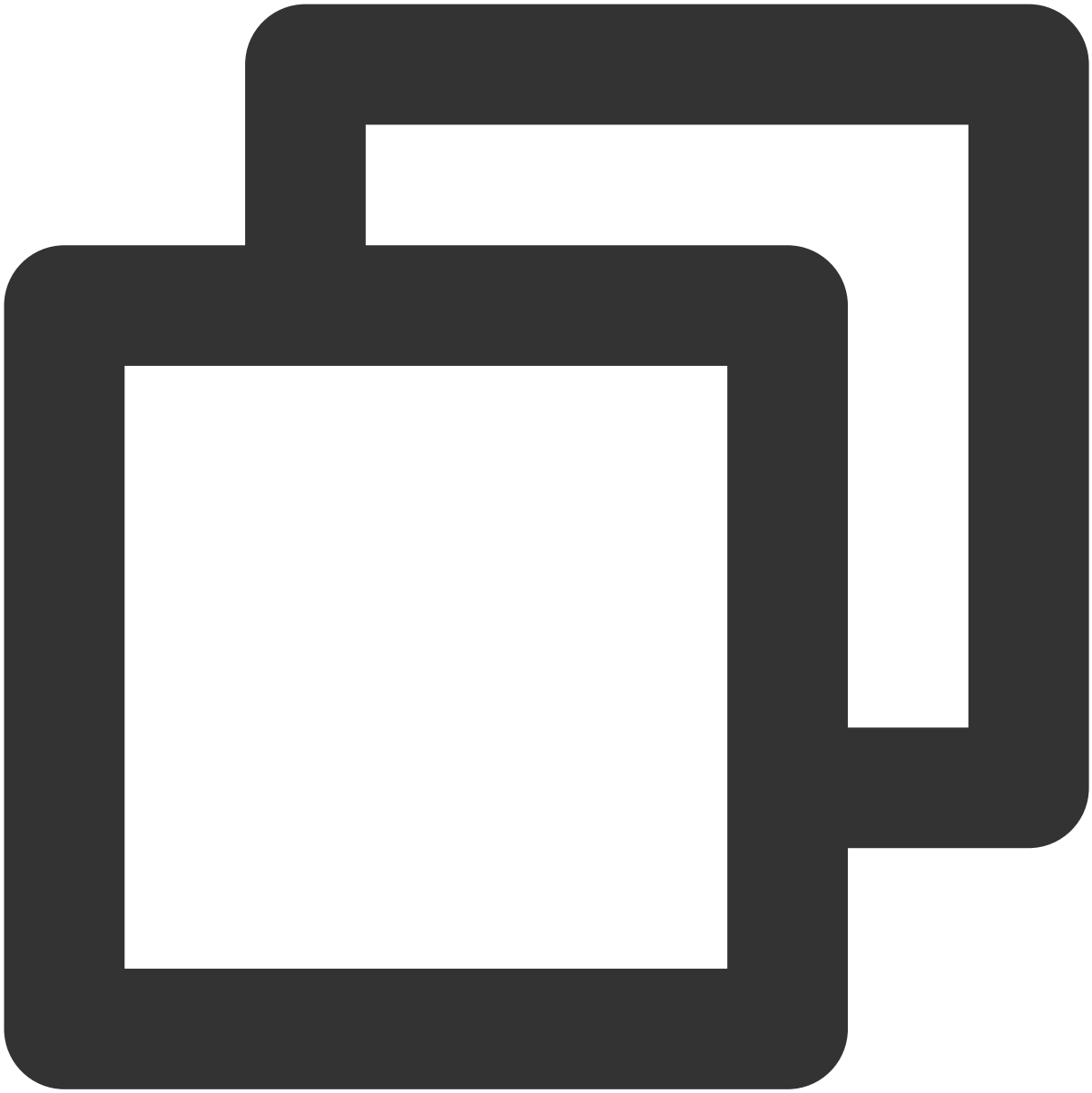
说明：

账号解绑只是解除 Token 与 App 账号的关联，若使用全量/标签/Token 推送，仍然能收到通知/消息。

参数说明

context：当前应用上下文对象，不能为 null。

示例代码



```
XGPushManager.clearAccounts(getApplicationContext());
```

标签管理

以下为标签管理相关接口方法，若需了解调用时机及调用原理，可查看 标签相关流程。

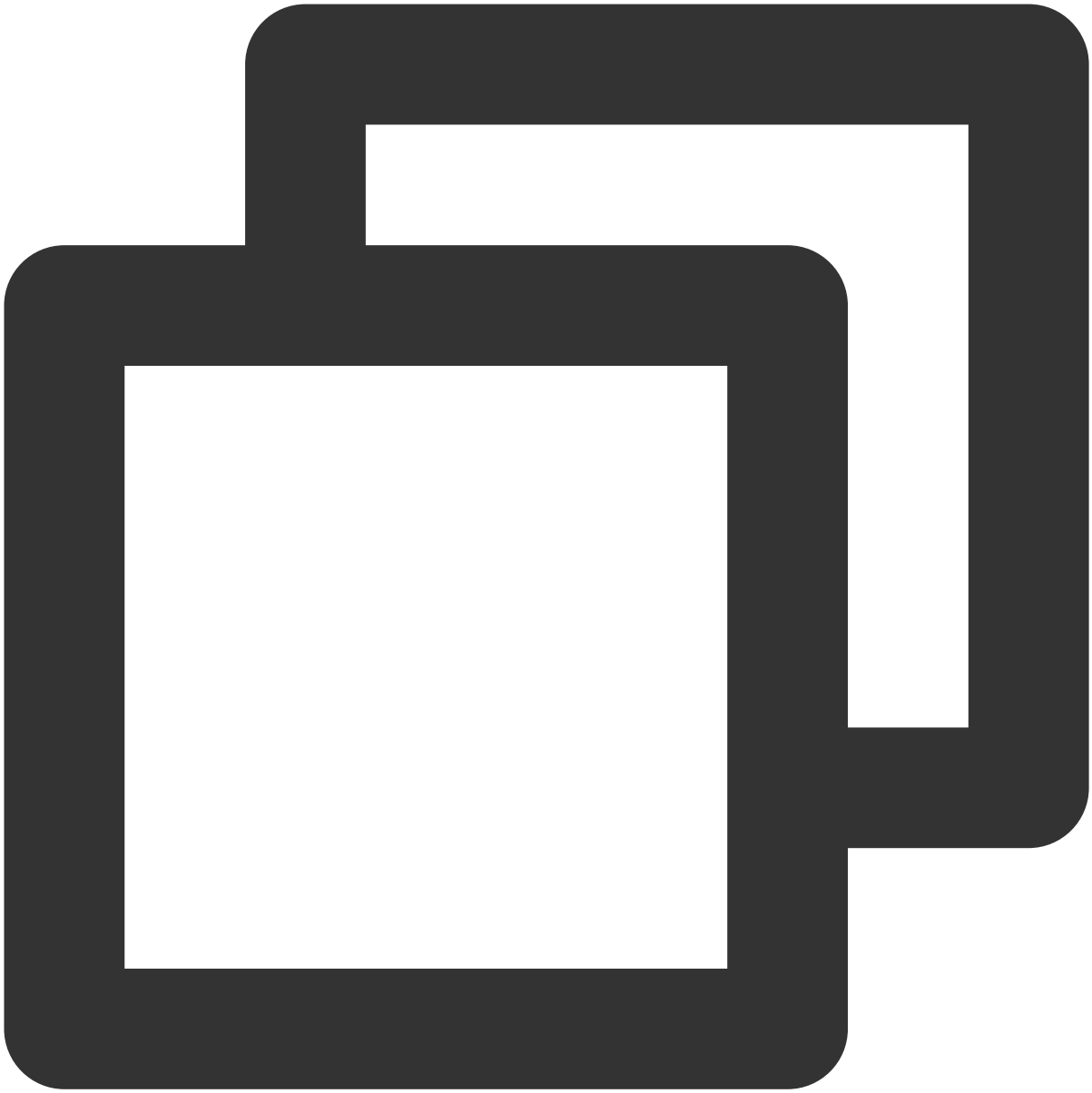
预设标签

目前移动推送平台提供的预设标签包括：App 版本，系统版本，省份，活跃信息，系统语言，SDK 版本，国家&地区，手机品牌，手机机型。预设标签会在 SDK 内部自动上报。

覆盖多个标签

接口说明

一次设置多个标签，会覆盖这个设备之前设置的标签。
开发者可以针对不同的用户设置标签，然后根据标签名群发通知。 一个应用最多有10000个 tag， 每个 Token 在一个应用下最多100个 tag，如需提高该限制，请联系 在线客服。每个自定义 tag 可绑定的设备 Token 数量无限制，tag 中不准包含空格。



```
public static void clearAndAppendTags(Context context, String operateName, Set<Stri
```

参数说明

context：Context 对象。

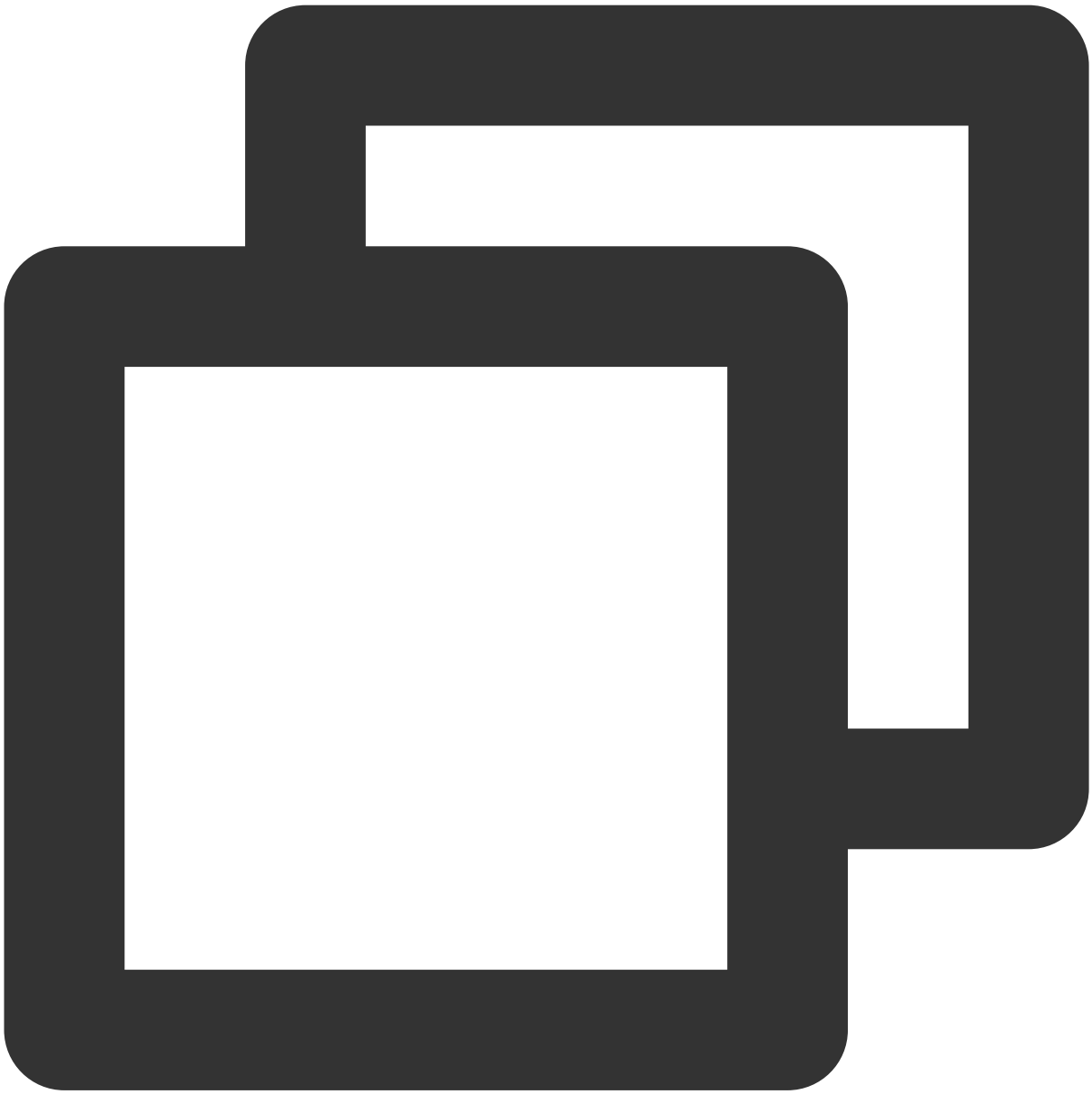
operateName：用户定义的操作名称，回调结果会原样返回，用于标识回调属于哪次操作。

tags：标签名集合，每个标签是一个 String。限制：每个 tag 不能超过50字节（超过会抛弃），不能包含空格（含有空格会删除空格）。最多设置100个 tag，超过部分会抛弃。

处理结果

可通过重写 XGPushBaseReceiver的onSetTagResult 方法获取。

示例代码



```
String[] tags = "tag1 tag2".split(" ");
Set<String> tagsSet = new HashSet<>(Arrays.asList(tags));
XGPushManager.clearAndAppendTags(getApplicationContext(), "clearAndAppendTags :" +
```

新增多个标签

说明：

SDK 1.2.2.0 版本废弃 addTags 接口，推荐使用 appendTags 接口。

接口说明

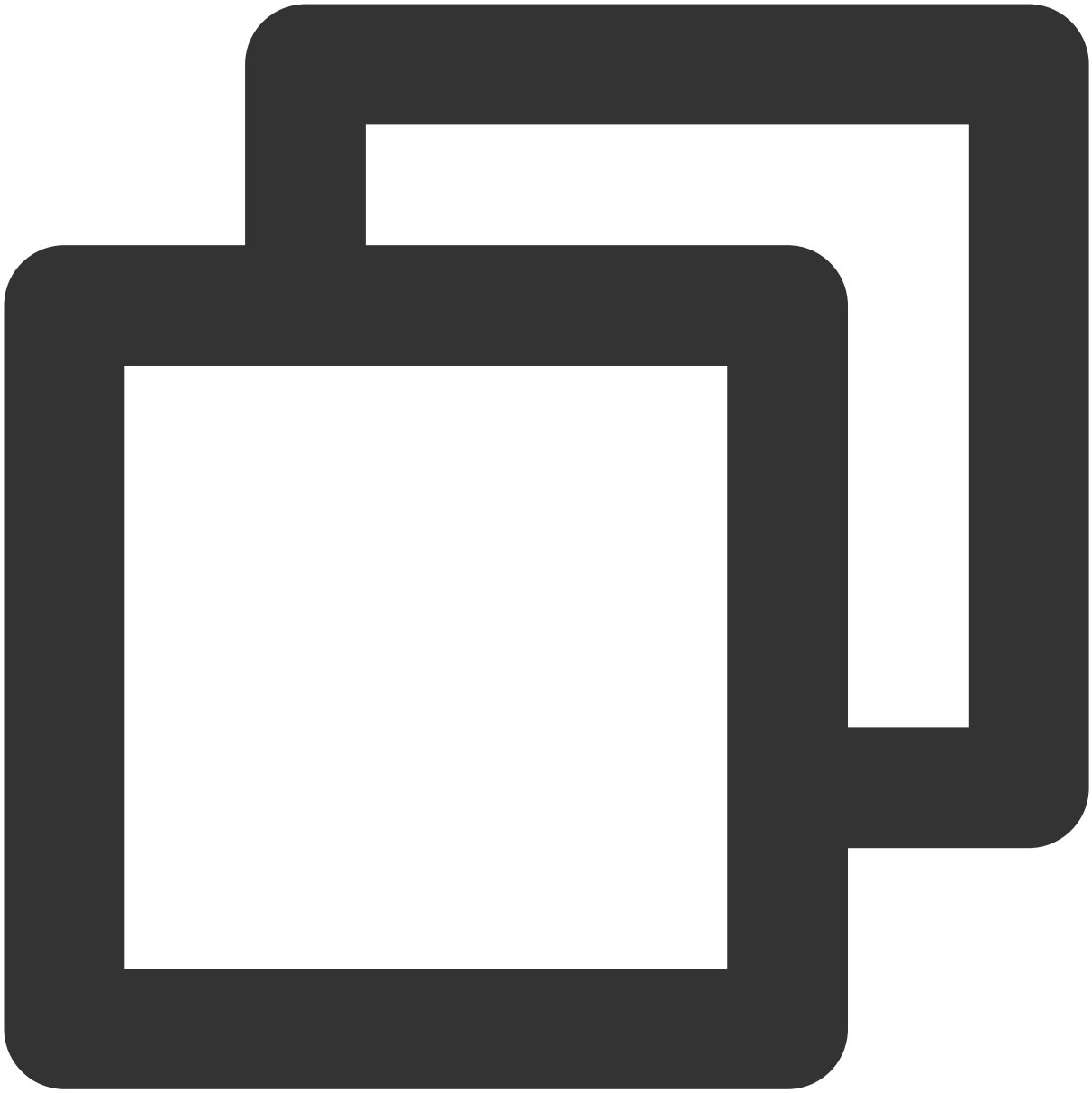
如果新覆盖的标签都带有 `:` 号，例如 `test:2, level:2`，则会删除这个设备已绑定的所有 `test:*` 和 `level:*` 标签，再新增 `test:2` 和 `level:2`。

如果新增的标签有部分不带 `:` 号，例如 `test:2  level`，则会删除这个设备的全部历史标签，再新增 `test:2` 和 `level` 标签。

说明：

新增的 tags 中，`:` 号为后台关键字，请根据具体的业务场景使用。

此接口调用的时候需要间隔一段时间（建议大于5s），否则可能造成更新失败。



```
public static void appendTags(Context context, String operateName, Set<String> tags
```

参数说明

context：Context 对象。

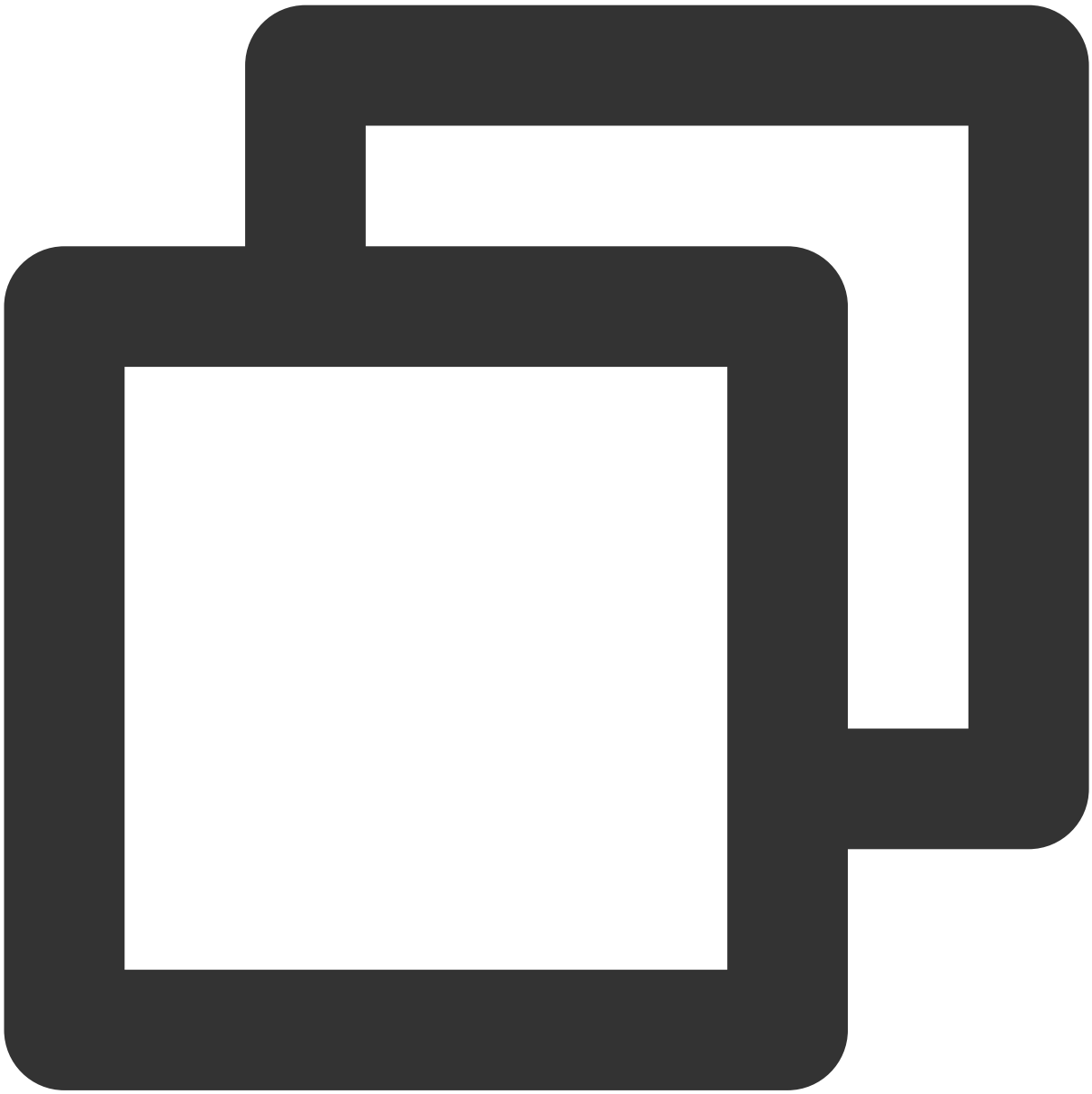
operateName：用户定义的操作名称，回调结果会原样返回，用于标识回调属于哪次操作。

tags：标签名集合，每个标签是一个 String。限制：每个 tag 不能超过50字节（超过会抛弃），不能包含空格（含有空格会删除空格）。最多设置100个 tag，超过部分会抛弃。

处理结果

可通过重写 XGPushBaseReceiver的onSetTagResult 方法获取。

示例代码



```
String[] tags = "tag1 tag2".split(" ");
Set<String> tagsSet = new HashSet<>(Arrays.asList(tags));
XGPushManager.appendTags(getApplicationContext(), "appendTags:" + System.currentTim
```

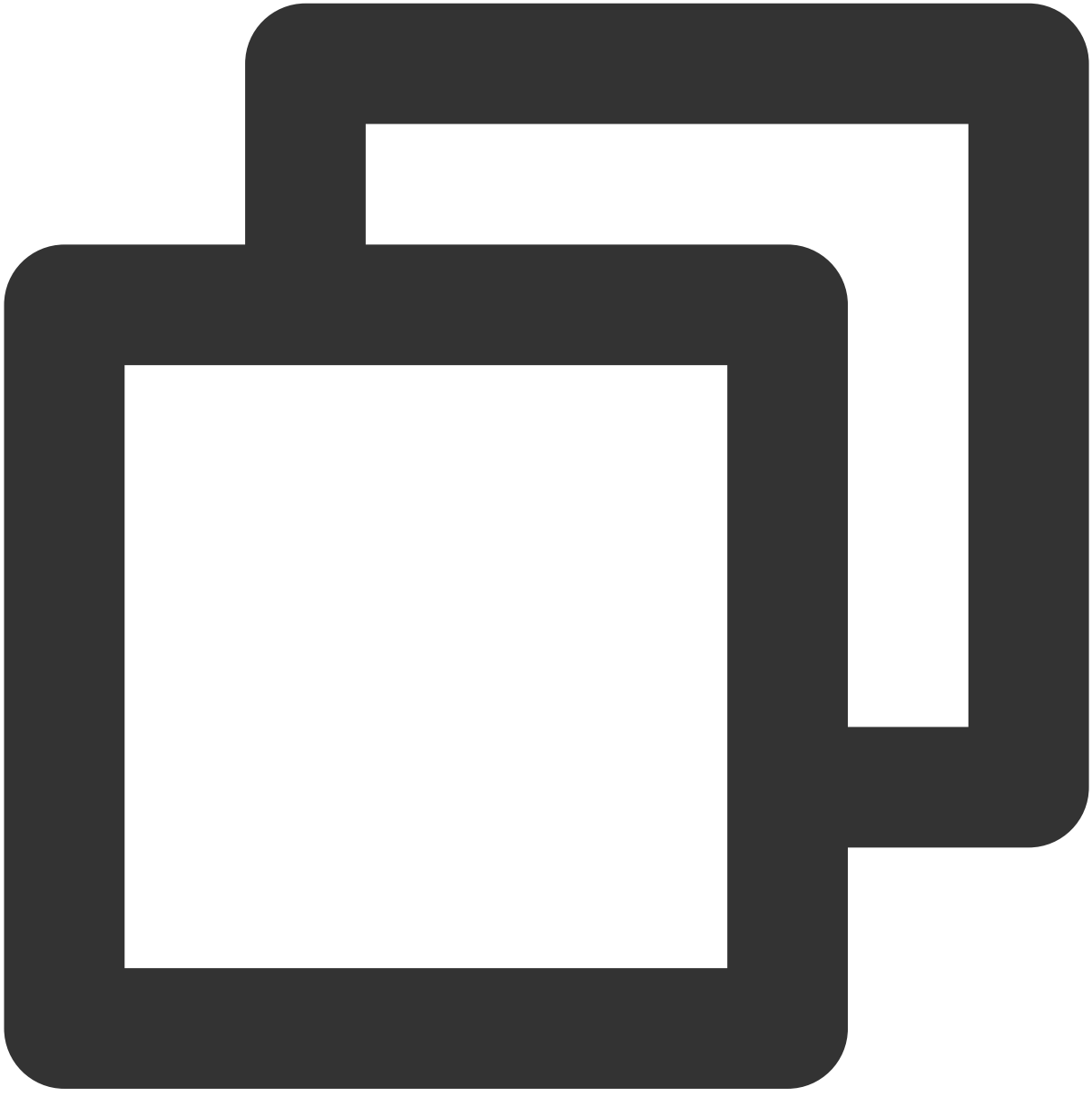
删除多个标签

说明：

SDK 1.2.2.0 版本废弃 deleteTags 接口，推荐使用 delTags 接口。

接口说明

一次删除多个标签。



```
public static void delTags(Context context, String operateName, Set<String> tags, X
```

参数说明

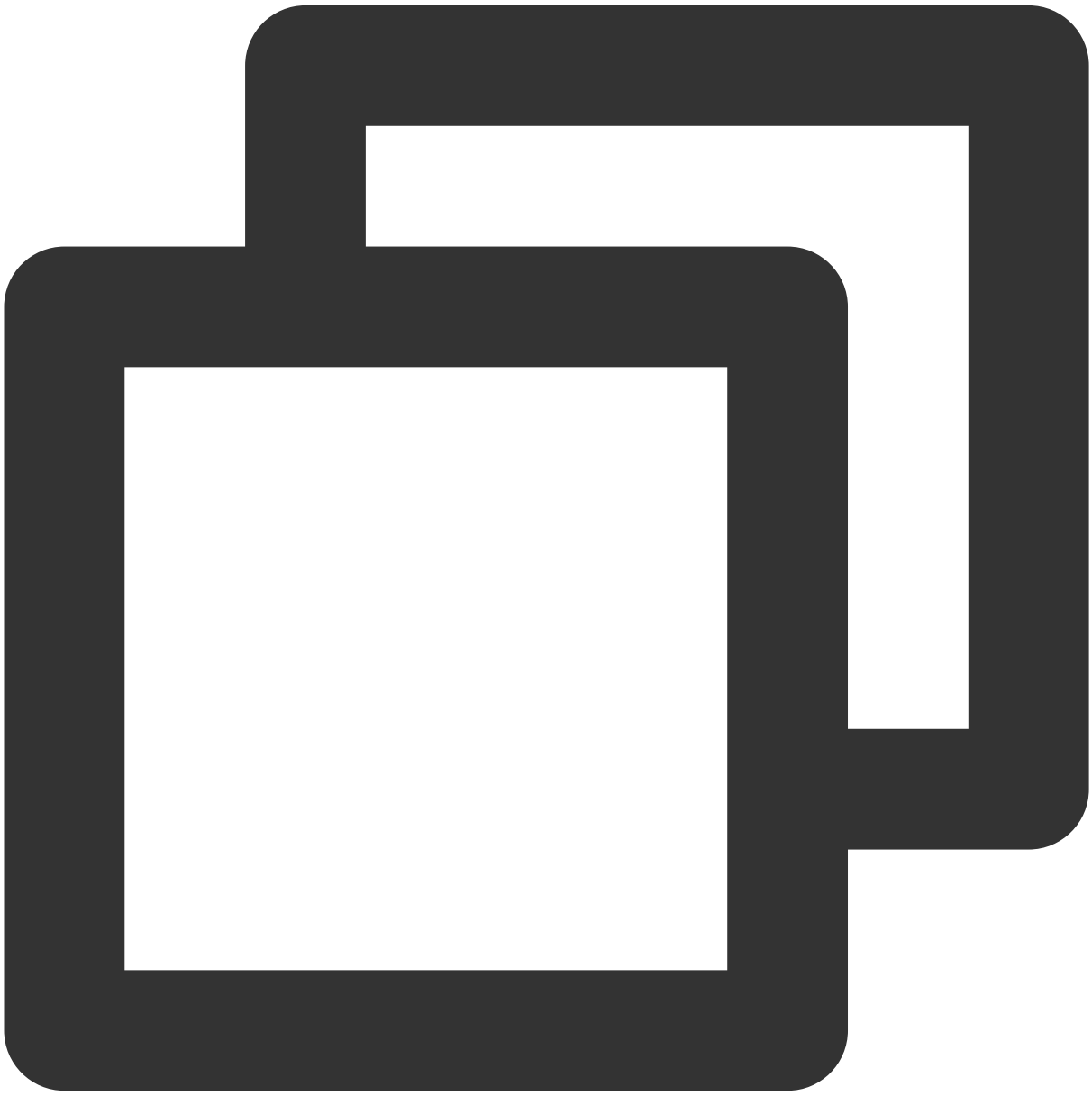
context：Context 对象。

operateName：用户定义的操作名称，回调结果会原样返回，用于标识回调属于哪次操作。
tags：标签名集合，每个标签是一个 String。限制：每个 tag 不能超过50字节（超过会抛弃），不能包含空格（含有空格会删除空格）。最多设置100个tag，超过部分会抛弃。
callback： 删除标签操作的回调

处理结果

可通过重写 XGPushBaseReceiver 的 onSetTagResult 方法获取。

示例代码



```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
  }
  @Override
  public void onFail(Object data, int errCode, String msg) {
      Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg)
  }
};

Set<String> tagSet = new HashSet<>();
tagSet.add("tag1");
tagSet.add("tag2");
XGPushManager.delTags(context, "delTags", tagSet, xgiOperateCallback);
```

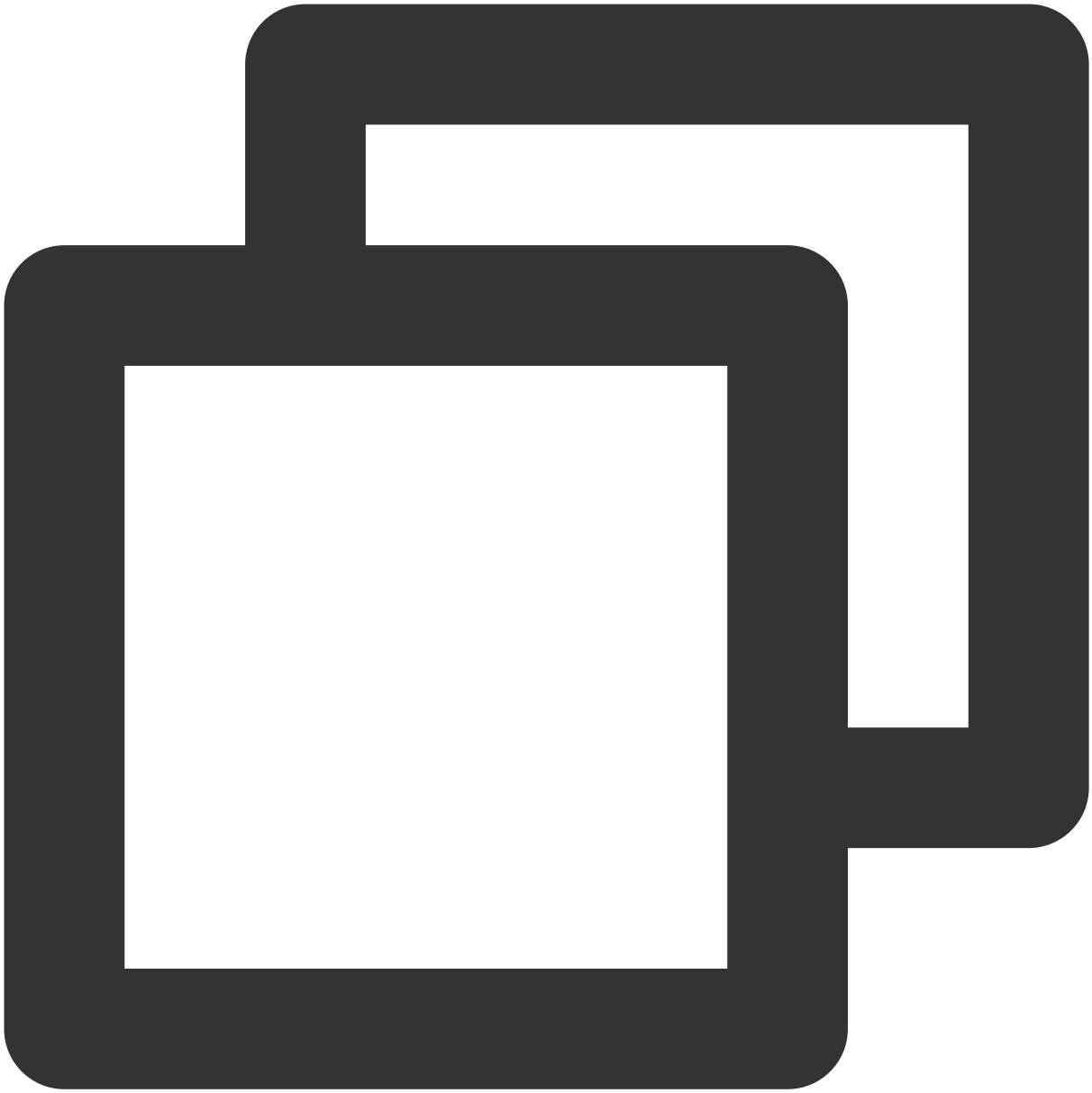
清除所有标签

说明：

SDK 1.2.2.0 版本开始废弃 cleanTags 接口，推荐使用 clearTags 接口。

接口说明

清除这个设备的所有标签。



```
public static void clearTags(Context context, String operateName, XGIOperateCallbac
```

参数说明

context：Context 对象。

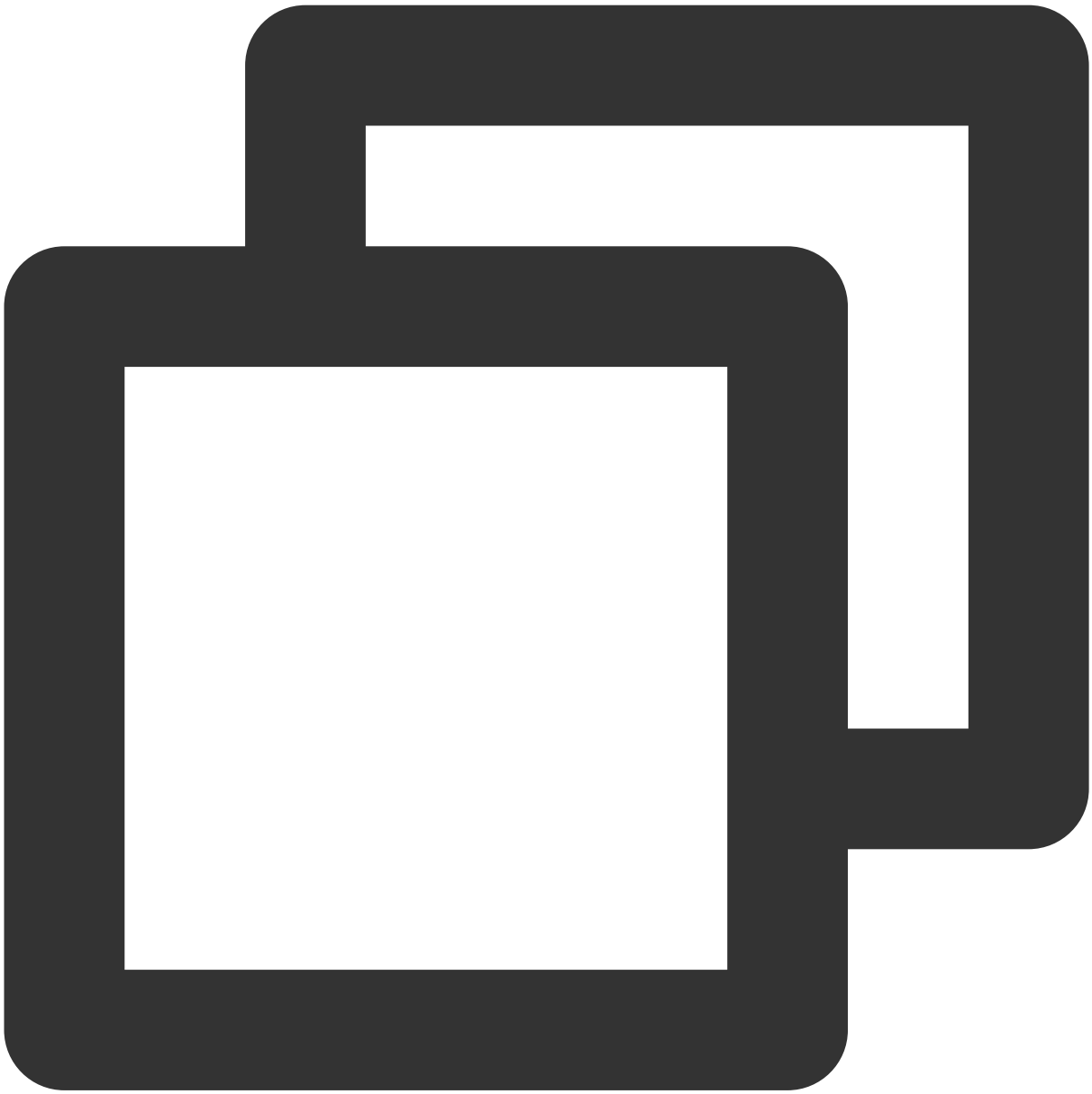
operateName：用户定义的操作名称，回调结果会原样返回，用于标识回调属于哪次操作。

callback： 清理所有标签操作的回调。

处理结果

可通过重写 XGPushBaseReceiver 的 onSetTagResult 方法获取。

示例代码



```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
  }
  @Override
  public void onFail(Object data, int errCode, String msg) {
```

```
        Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg)
    }
};

XGPushManager.clearTags(context, "clearTags", xgiOperateCallback);
```

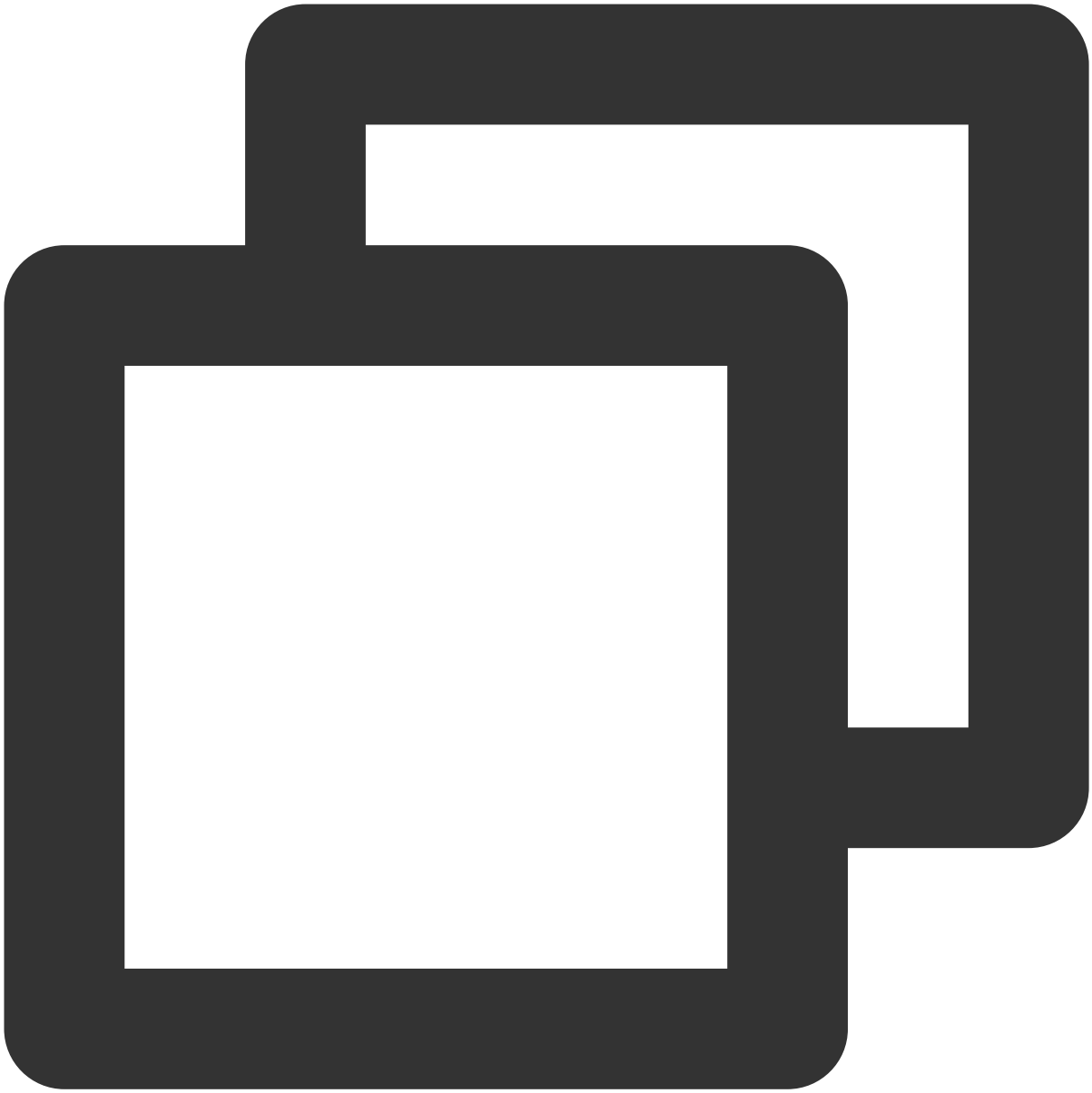
查询标签

说明：

查询设备关联的标签（此接口仅适用于1.2.5.0及以上版本）。

接口说明

获取这个设备的标签。



```
public static void queryTags(final Context context, final String operateName, fin
```

参数说明

context：Context 对象

operateName：用户定义的操作名称，回调结果会原样返回，用于给用户区分是哪个操作

offset：开始的位置

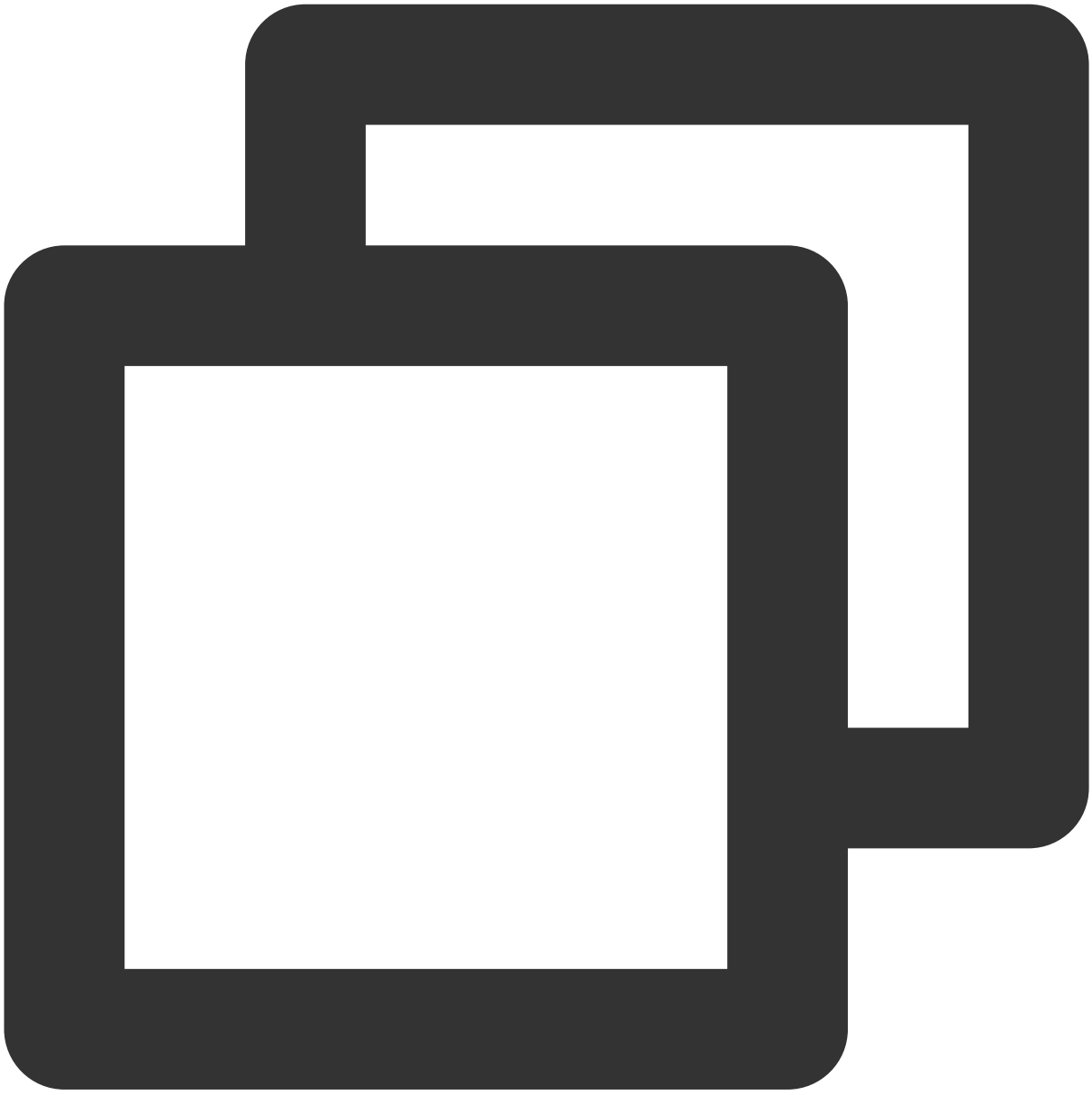
limit：获取标签的数量，最多为100个

callback：获取标签操作的回调

处理结果

可通过重写 XGPushBaseReceiver 的 onQueryTagsResult 方法获取。

示例代码



```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
  }
  @Override
```

```
    public void onFail(Object data, int errCode, String msg) {
        Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg)
    }
};
XGPushManager.queryTags(context, 0, 100, xgiOperateCallback);
```

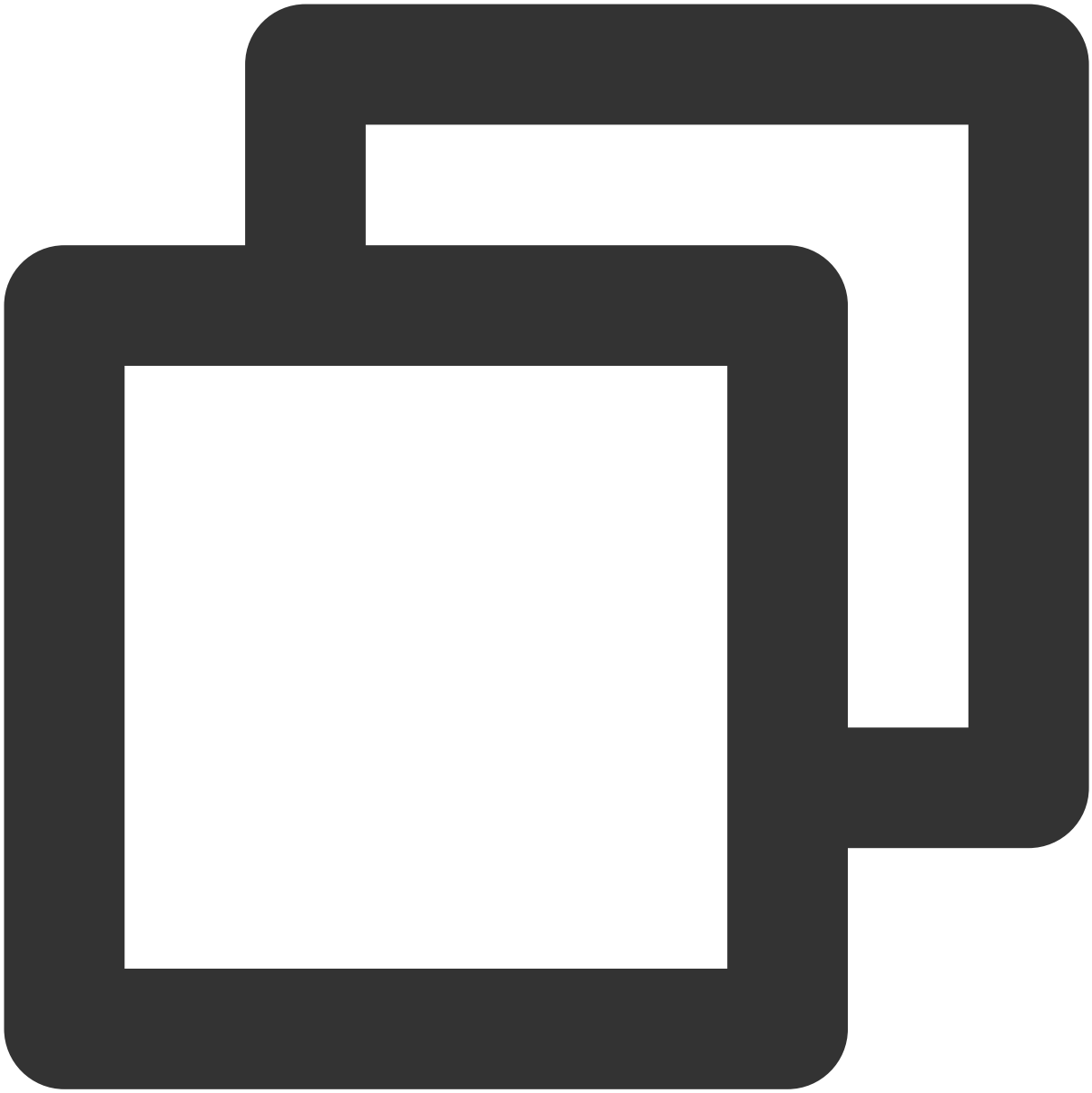
用户属性管理

开发者可以针对不同的用户设置属性，然后在管理平台推送的时候进行个性化推送。以下为用户属性相关接口方法，若需了解调用时机及调用原理，可查看 用户属性相关流程。

新增用户属性

接口说明

添加属性（带回调）：有则覆盖，无则添加。



```
public static void upsertAttributes(Context context, String operateName, Map<String
```

参数说明

context：Context 对象。

operateName：用户定义的操作名称，回调结果会原样返回，用于给用户区分是哪个操作。

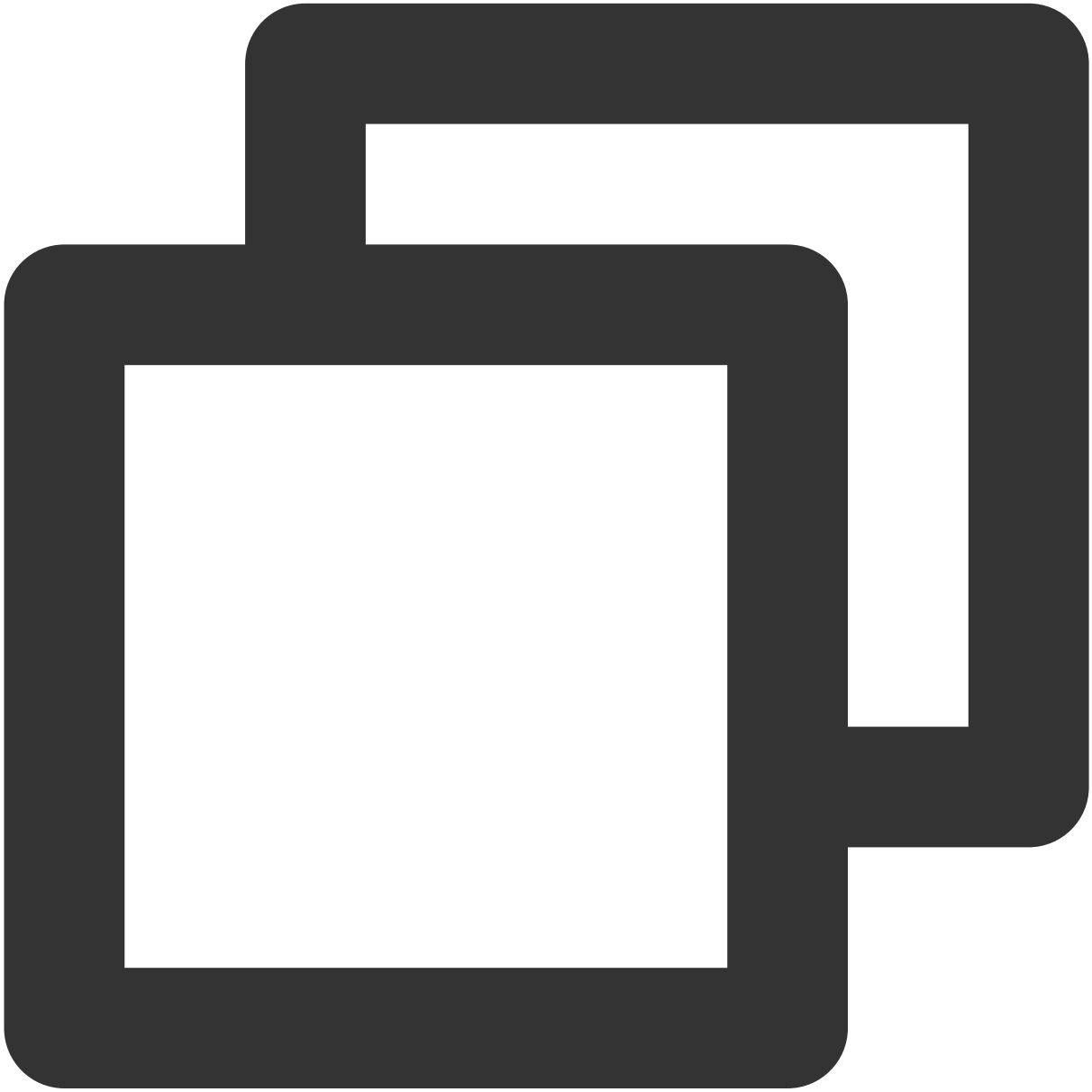
attributes：属性集合，每个属性通过 key-value 标识。

callback：添加属性操作的回调。

注意：

1. 属性使用键值对传输，都只接受 string 字符串类型，非空串。
2. 属性个数限制50个。
3. 属性 key，value 长度都限制50个字符以内。

示例代码



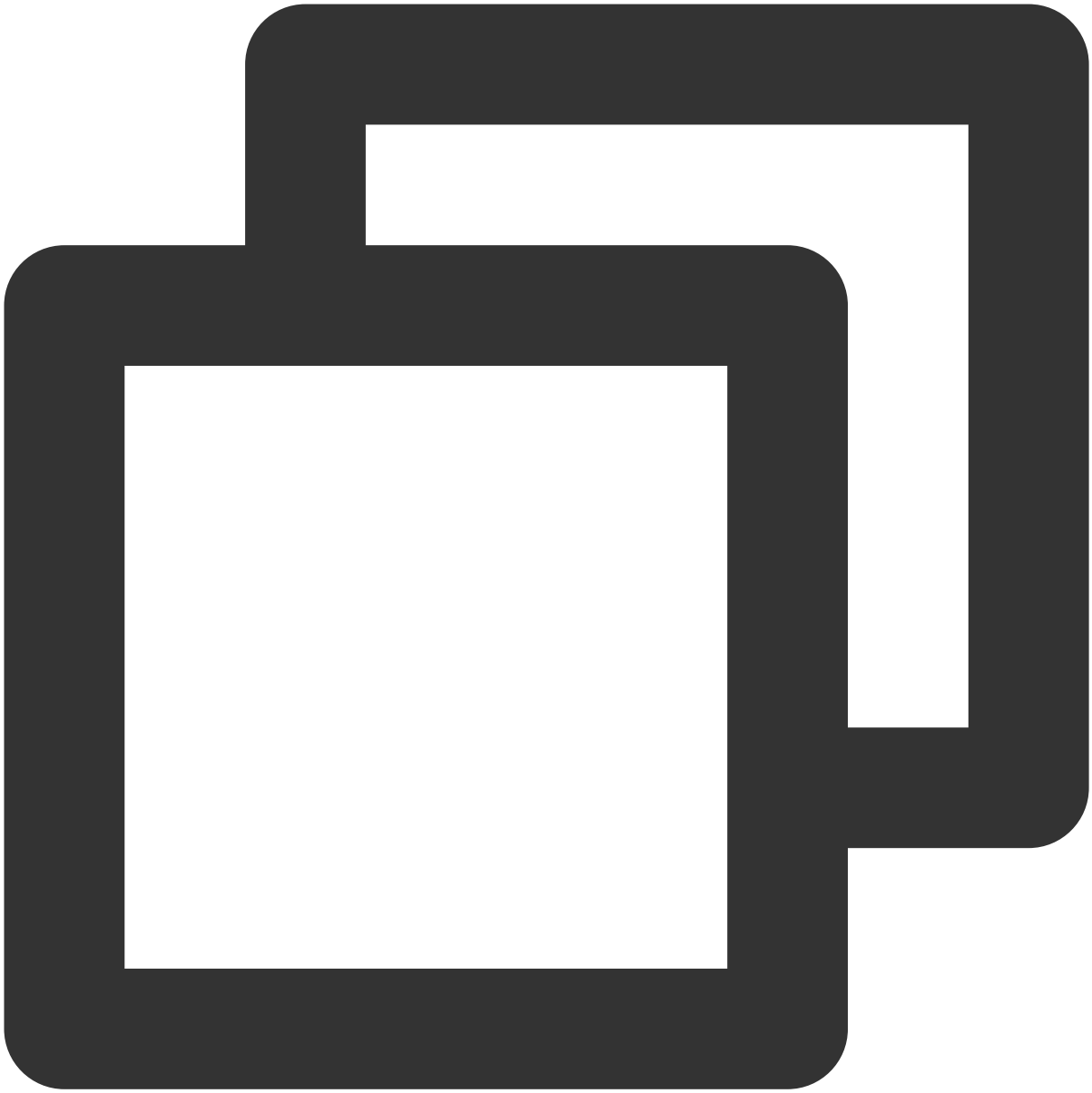
```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      log("action - onSuccess, data:" + data + ", flag:" + flag);
  }
```

```
    @Override
    public void onFail(Object data, int errCode, String msg) {
        log("action - onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};

Map<String,String> attr = new HashMap<>();
attr.put("name", "coding-test");
attr.put("gender", "male");
attr.put("age", "100");
XGPushManager.upsertAttributes(context, "addAttributes-test", attr, xgiOperateCallb
```

删除用户属性

接口说明

删除指定的属性。



```
public static void delAttributes(Context context, String operateName, Set<String> a
```

参数说明

context：Context 对象。

operateName：用户定义的操作名称，回调结果会原样返回，用于给用户区分是哪个操作。

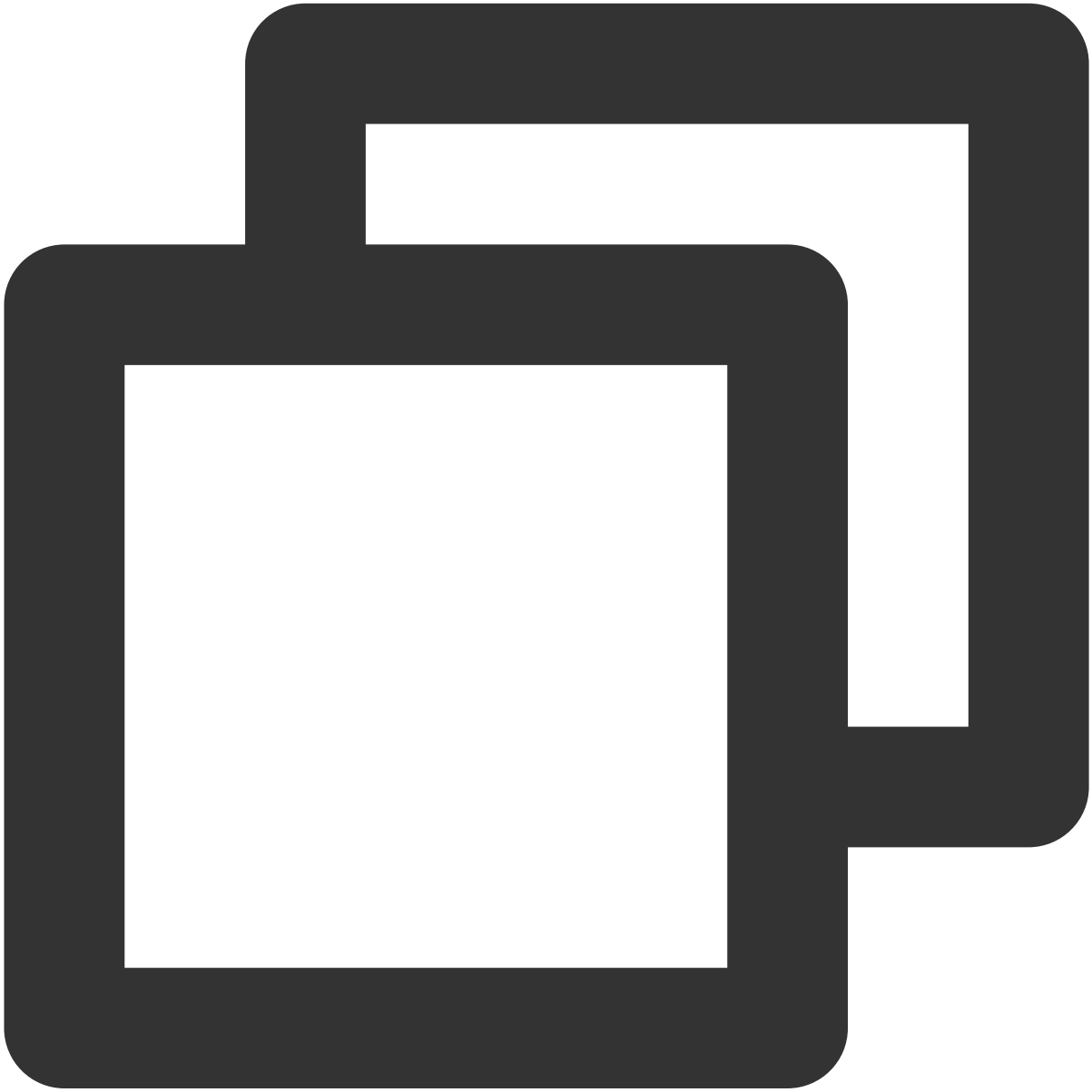
attributes：属性集合，每个属性通过 key-value 标识。

callback：删除属性操作的回调。

注意：

1. 属性使用键值对传输，都只接受 string 字符串类型，非空串。
2. 属性个数限制50个。
3. 属性 key，value 长度都限制50个字符以内。

示例代码



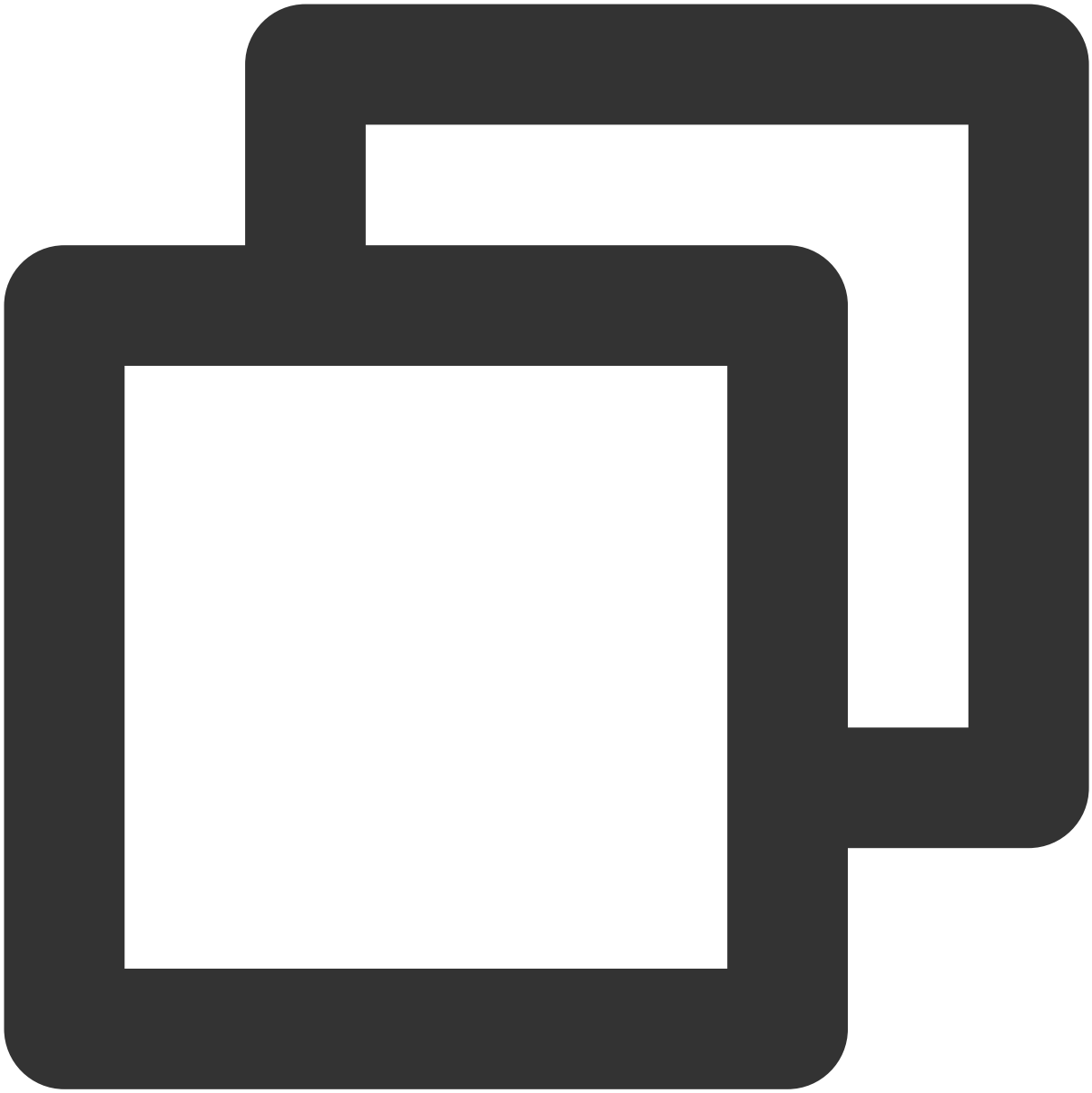
```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      log("action - onSuccess, data:" + data + ", flag:" + flag);
  }
```

```
    @Override
    public void onFail(Object data, int errCode, String msg) {
        log("action - onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};
Set<String> stringSet = new HashSet<>();
stringSet.add("name");
stringSet.add("gender");

XGPushManager.delAttributes(context, "delAttributes-test", stringSet, xgiOperateCal
```

清空已有用户属性

接口说明

删除已设置的所有属性。



```
public static void clearAttributes(Context context, String operateName, XGIOperateC
```

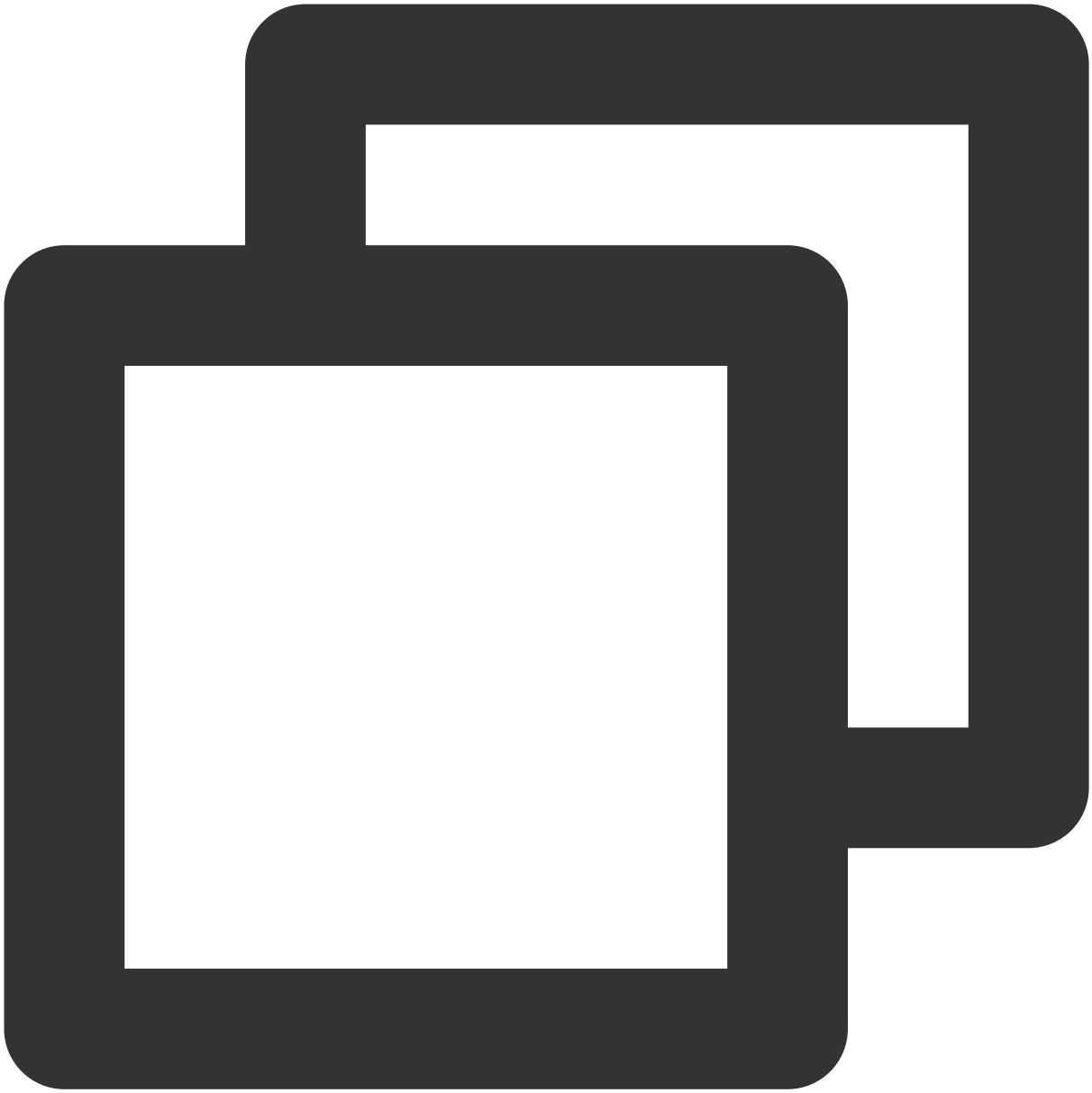
参数说明

context：Context 对象。

operateName：用户定义的操作名称，回调结果会原样返回，用于给用户区分是哪个操作。

callback：清理所有属性操作的回调。

示例代码



```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      log("action - onSuccess, data:" + data + ", flag:" + flag);
  }
  @Override
  public void onFail(Object data, int errCode, String msg) {
      log("action - onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
  }
};
```

```
XGPushManager.clearAttributes(context, "cleanAttributes-test", xgiOperateCallback);
```

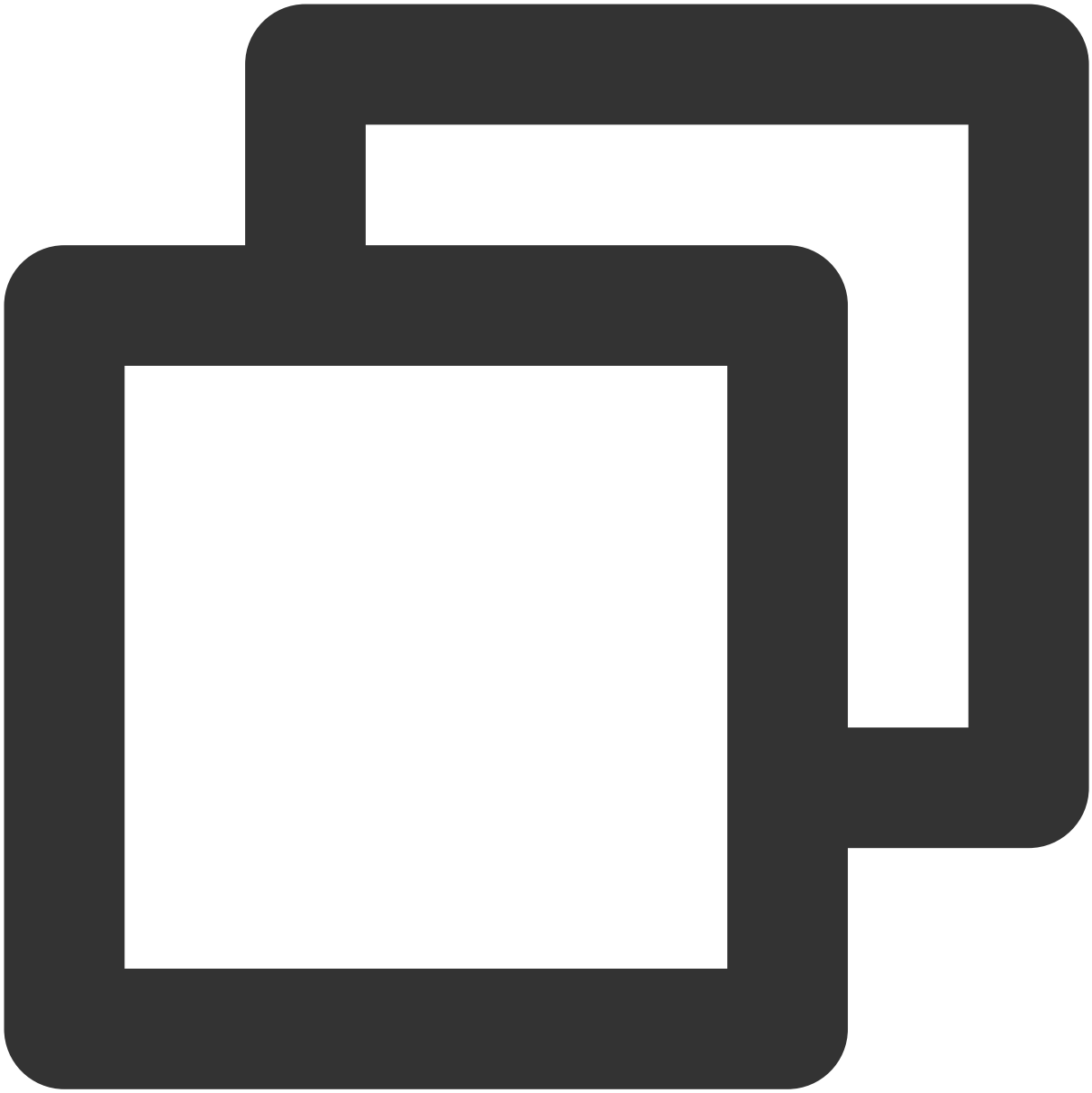
更新用户属性

接口说明

设置属性（带回调），会覆盖这个设备之前设置的所有属性（即清理并设置）。

注意：

1. 属性使用键值对传输，都只接受 string 字符串类型，非空串。
2. 属性个数限制50个。
3. 属性 key，value 长度都限制50个字符以内。



```
public static void clearAndAppendAttributes(Context context, String operateName, Ma
```

参数说明

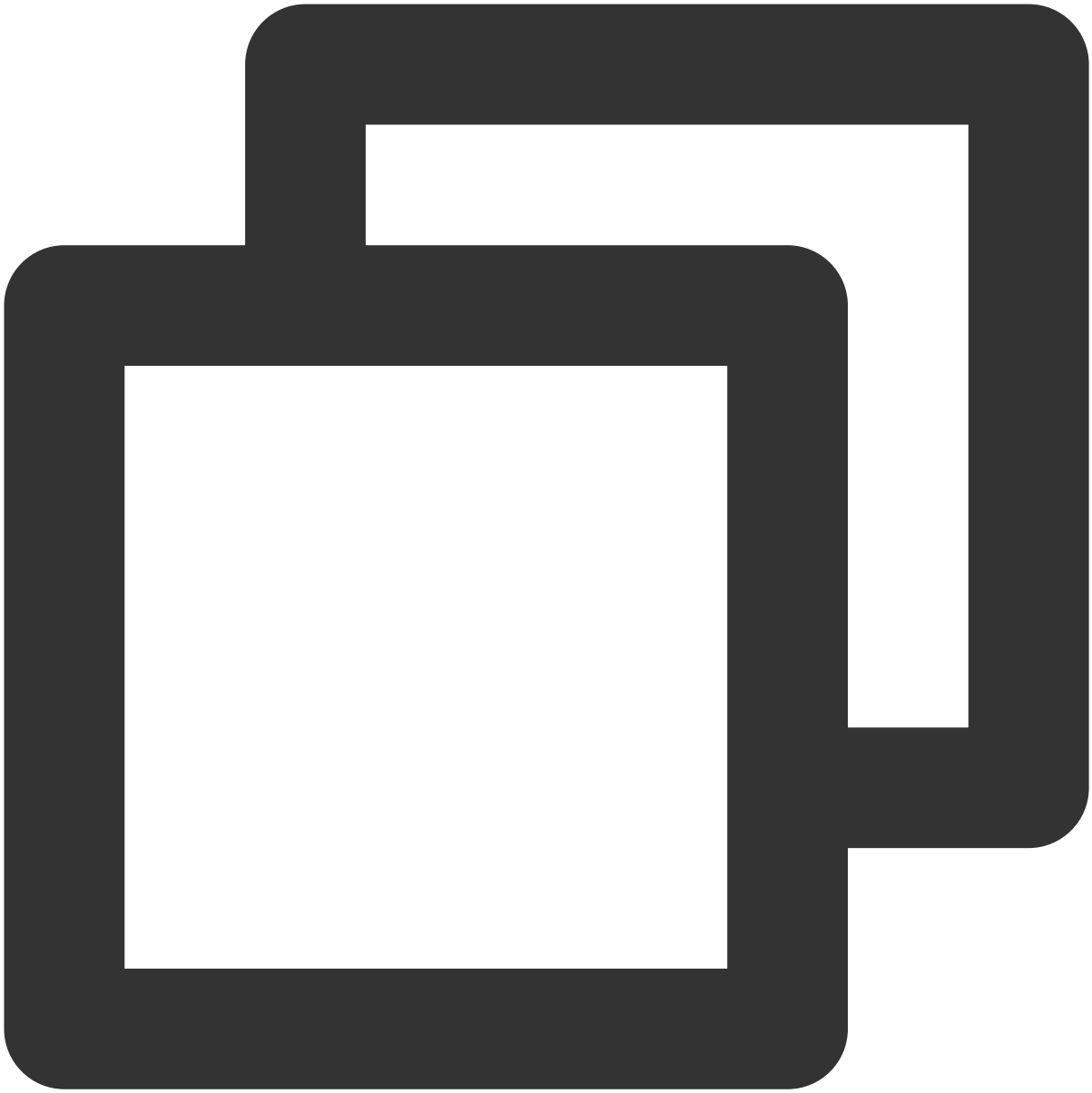
context：Context 对象。

operateName：用户定义的操作名称，回调结果会原样返回，用于给用户区分是哪个操作。

attributes：属性集合，每个属性通过 key-value 标识。

callback：设置属性操作的回调。

示例代码



```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      log("action - onSuccess, data:" + data + ", flag:" + flag);
  }
  @Override
  public void onFail(Object data, int errCode, String msg) {
      log("action - onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
  }
};
```

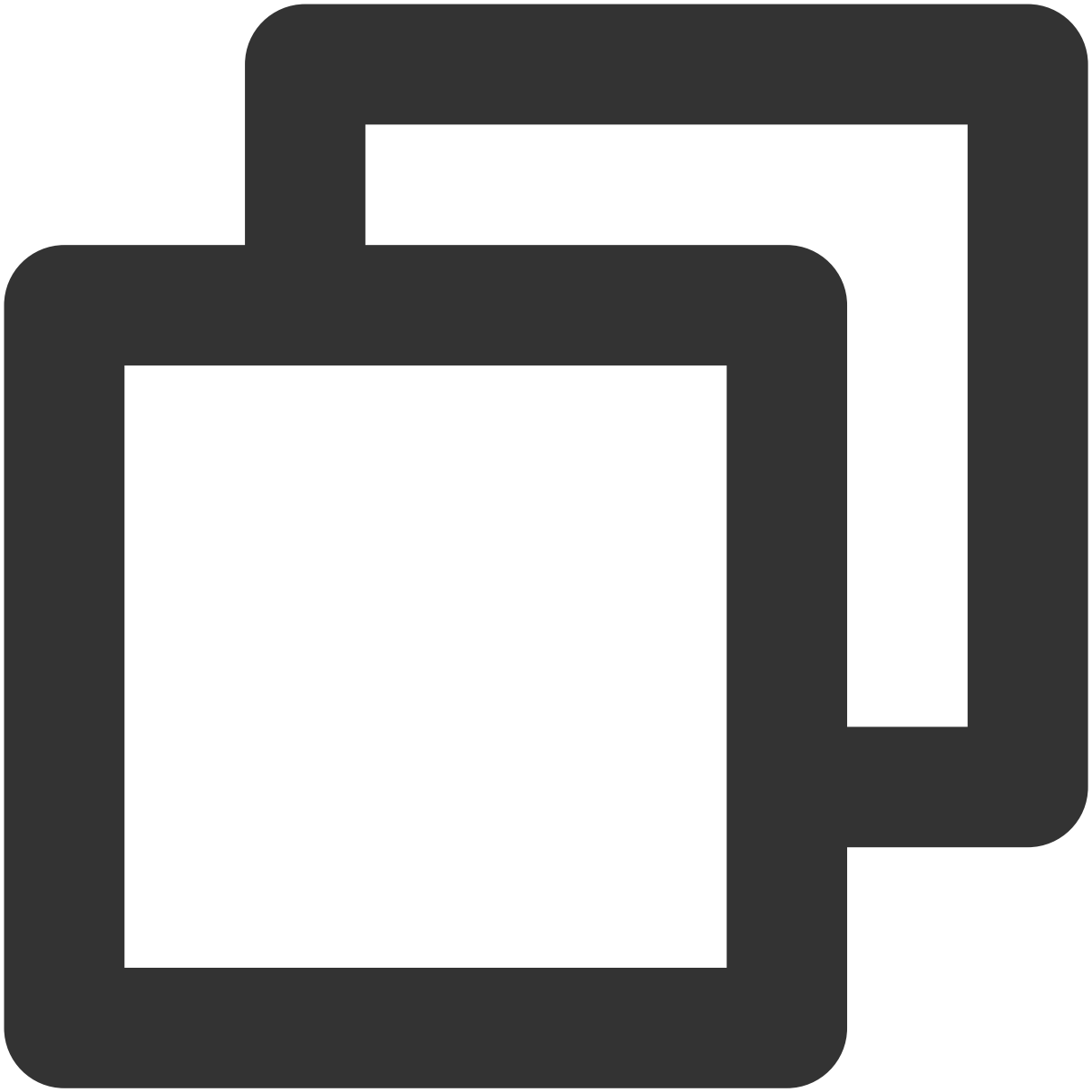
```
Map<String,String> attr = new HashMap<>();
attr.put("name", "coding-test");
attr.put("gender", "male");
attr.put("age", "100");
XGPushManager.clearAndAppendAttributes(context, "setAttributes-test", attr, xgiOper
```

配置接口

所有的配置相关接口在 XGPushConfig 类中，为了使配置及时生效，开发者需要保证配置接口在启动或注册移动推送之前被调用。

关闭联合保活能力(1.1.6.1+)

移动推送默认开启联合保活能力，若需要关闭联合保活能力，请在应用初始化的时候，例如 Application 或 LauncherActivity 的 onCreate 中调用如下接口，并传递 false。



```
XGPushConfig.enablePullUpOtherApp(Context context, boolean pullUp);
```

说明：

1.2.6.0 起默认关闭联合保活功能，可不再调用此接口。

参数说明

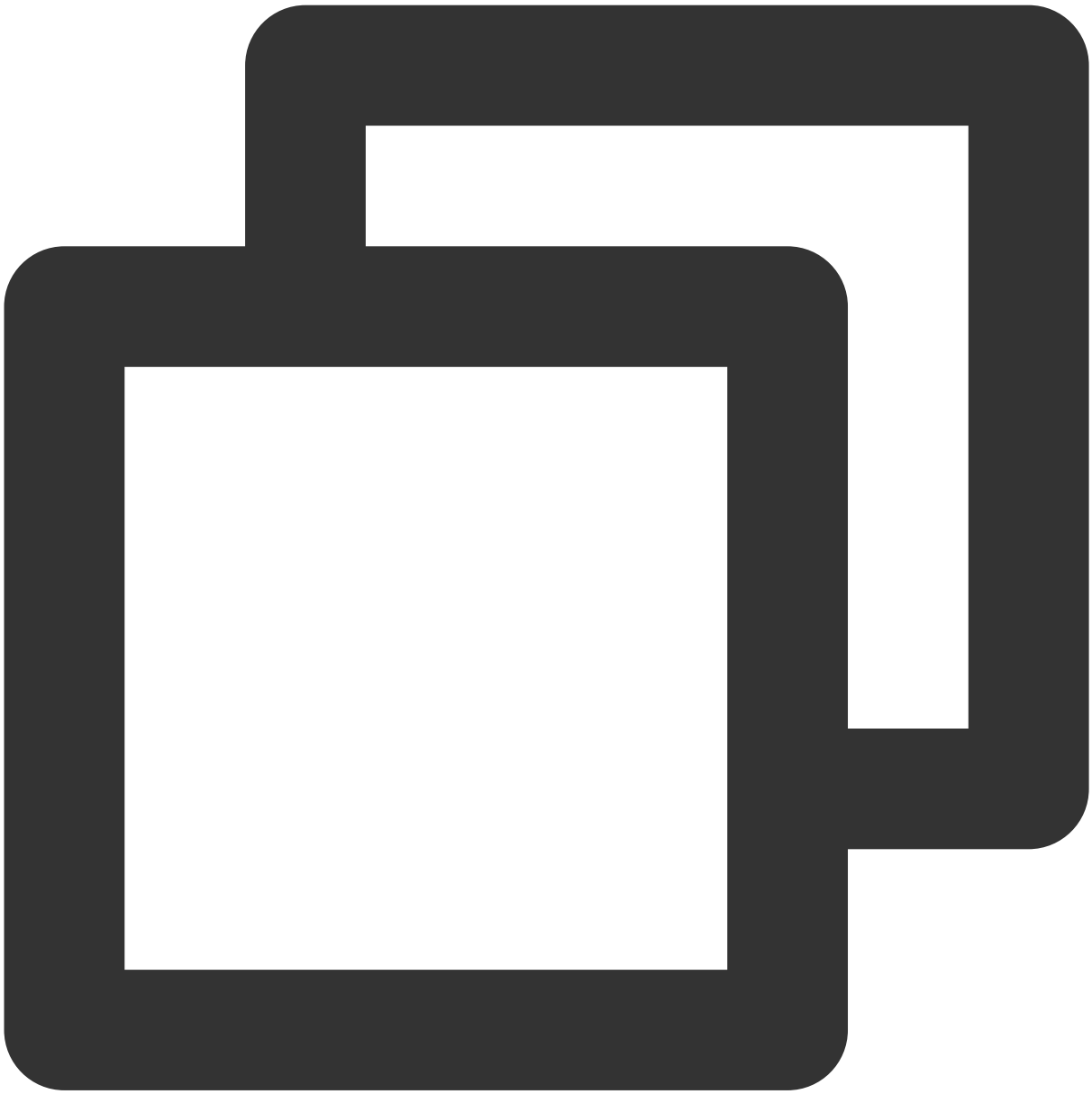
context：应用上下文

pullUp：true（开启联合保活）；false（关闭联合保活）

说明：

若有以下日志打印，则表明联合保活功能已经关闭：I/TPNS: [ServiceUtil] disable pull up other app。

示例代码

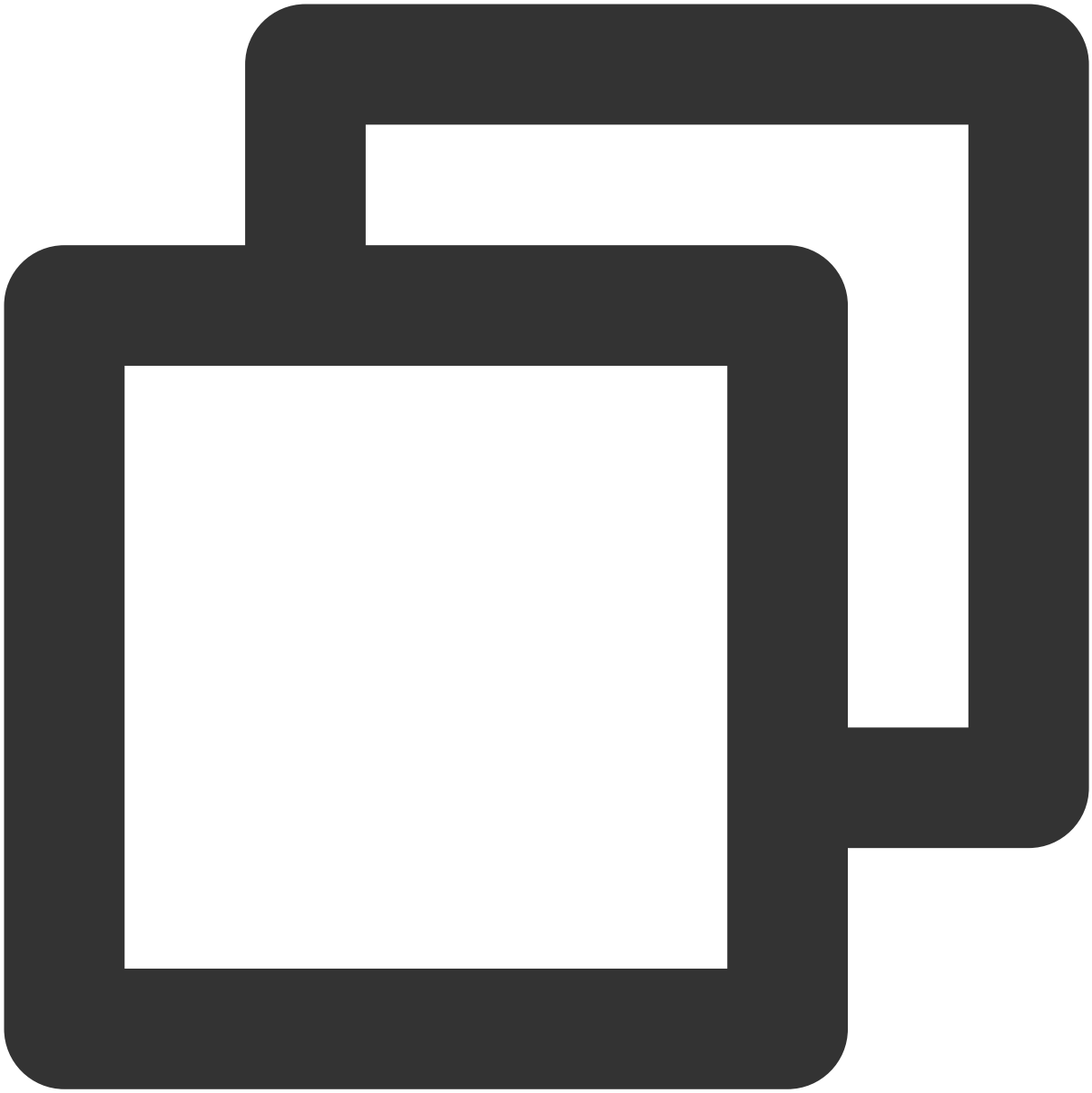


```
XGPushConfig.enablePullUpOtherApp(context, false); // 默认为 true: 开启保活
```

Debug 模式

接口说明

为保证数据的安全性，请在发布时确保已关闭 Debug 模式。



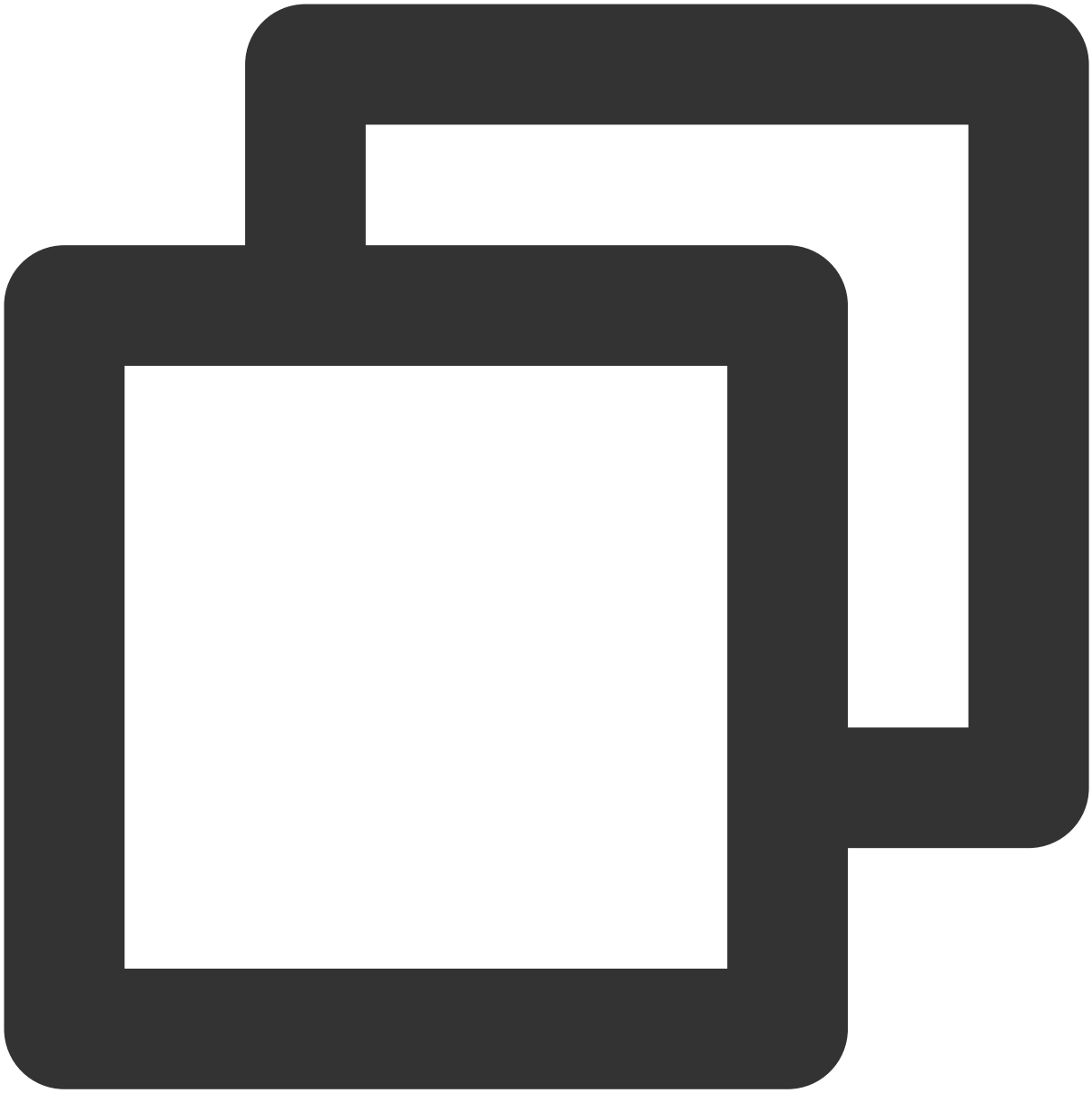
```
public static void enableDebug(Context context, boolean debugMode)
```

参数说明

context：App 上下文对象。

debugMode：默认为 false。如果要开启 Debug 日志，设为 true。

示例代码



```
XGPushConfig.enableDebug(context, true); // 默认为 false: 不打开
```

获取设备 Token

接口说明

Token 是移动推送保持与后台长连接的唯一身份标识，是 App 接收消息的唯一 ID，只有设备注册成功后才能获取 Token，获取方法如下。（移动推送的 Token 在应用卸载重新安装的时候有可能会变。）

1. 通过带 callback 的注册接口获取

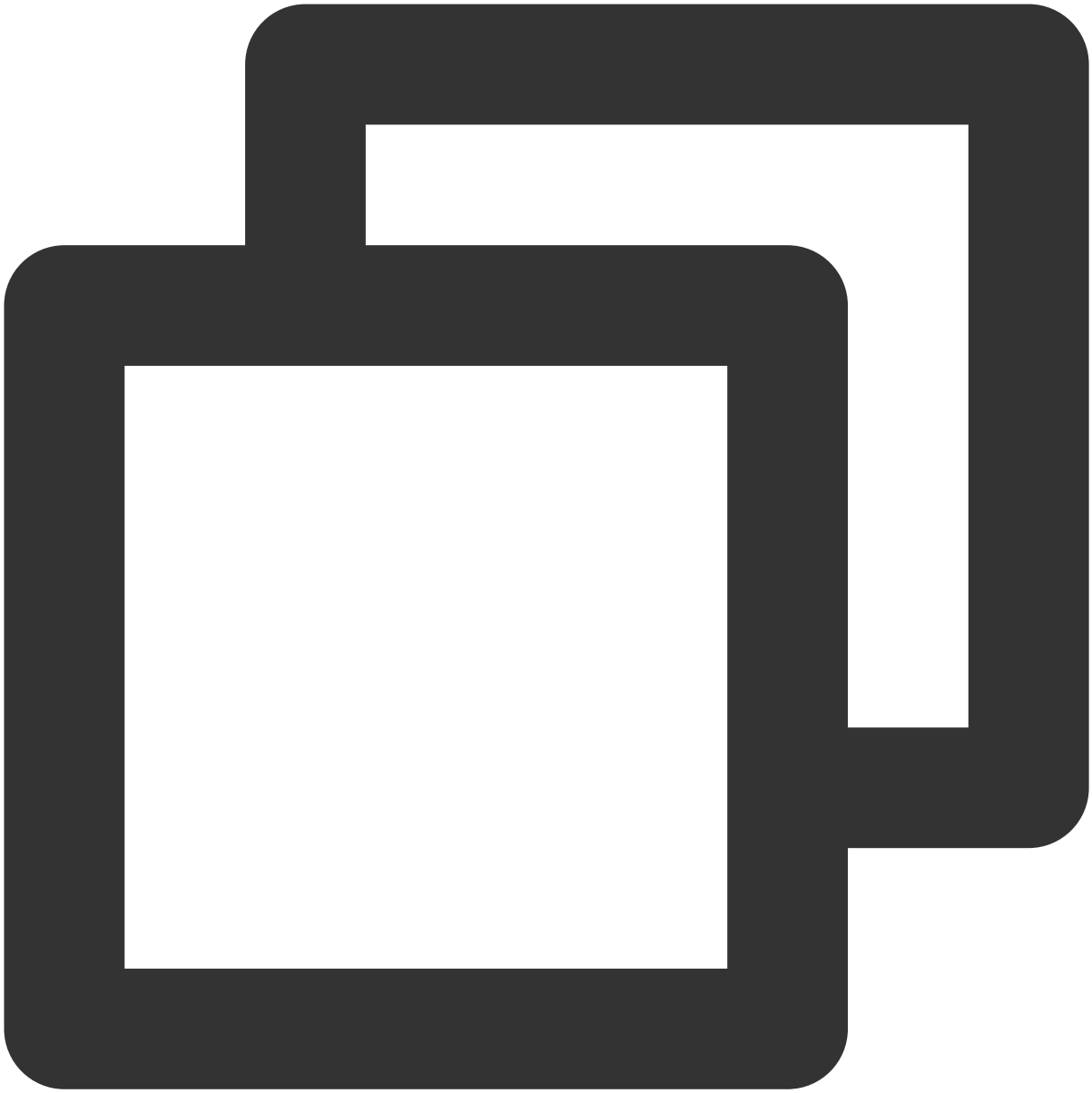
带 XGIOperateCallback 的注册接口的 onSuccess(Object data, int flag) 方法中，参数 data 便是 Token，具体可参考注册接口的相关示例。

2. 继承 XGPushBaseReceiver

重写 XGPushBaseReceiver 的 onRegisterResult (Context context, int errorCode,XGPushRegisterResult registerMessage) 方法，通过参数 registerMessage 提供的 getToken 接口获取，具体请参见 获取注册结果 章节。

3. XGPushConfig.getToken(context)

当设备一旦注册成功后，便会将 Token 存储在本地，之后可通过 XGPushConfig.getToken(context) 接口获取。Token 是一个设备的身份识别 ID，由服务器根据设备属性随机产生并下发到本地，同一 App 在不同设备上的 Token 不同。



```
public static String getToken(Context context)
```

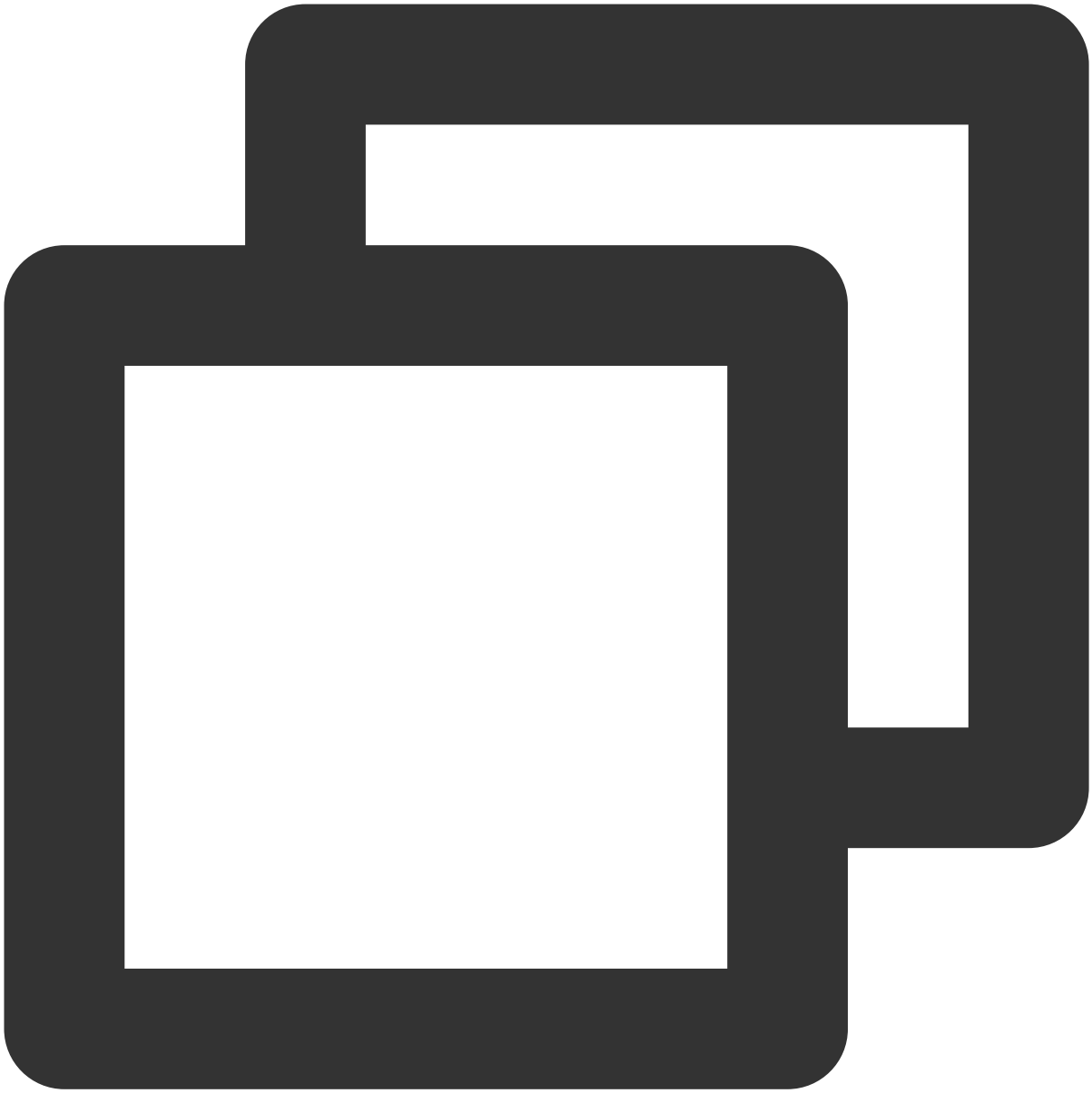
说明：

App 第一次注册会产生 Token，之后一直存储在手机上，不论之后是否进行注销注册操作，该 Token 一直存在。当 App 完全卸载重装后，Token 会发生变化。不同 App 之间的 Token 不同。

参数说明

context：App 上下文对象。

示例代码



```
XGPushConfig.getToken(context);
```

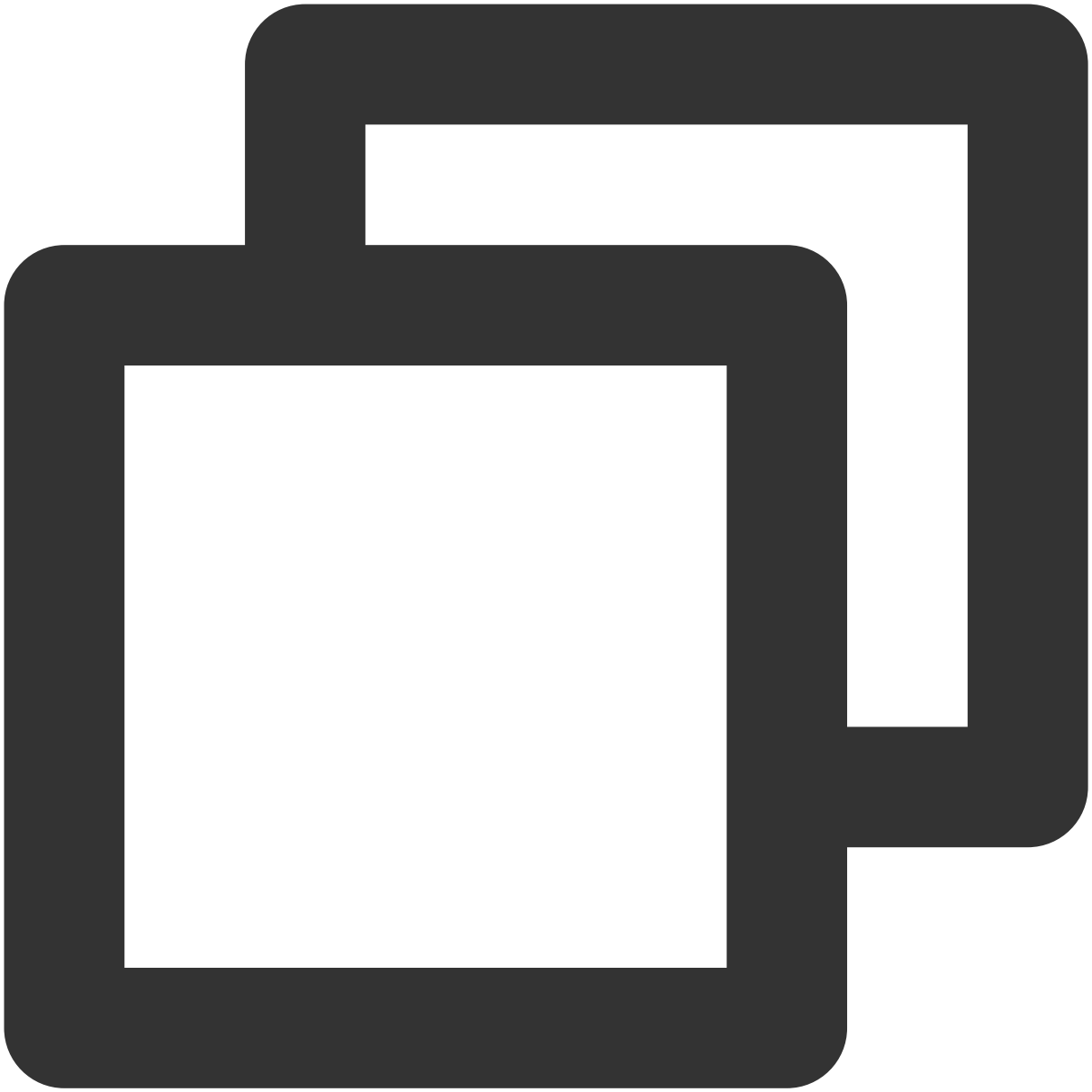
返回值

成功时返回正常的 Token；失败时返回 null 或0。

获取第三方厂商 Token

接口说明

第三方厂商 Token 是厂商设备的身份识别 ID，由厂商下发到本地，同一 App 在不同设备上的 Token 不同。



```
public static String getOtherPushToken(Context context)
```

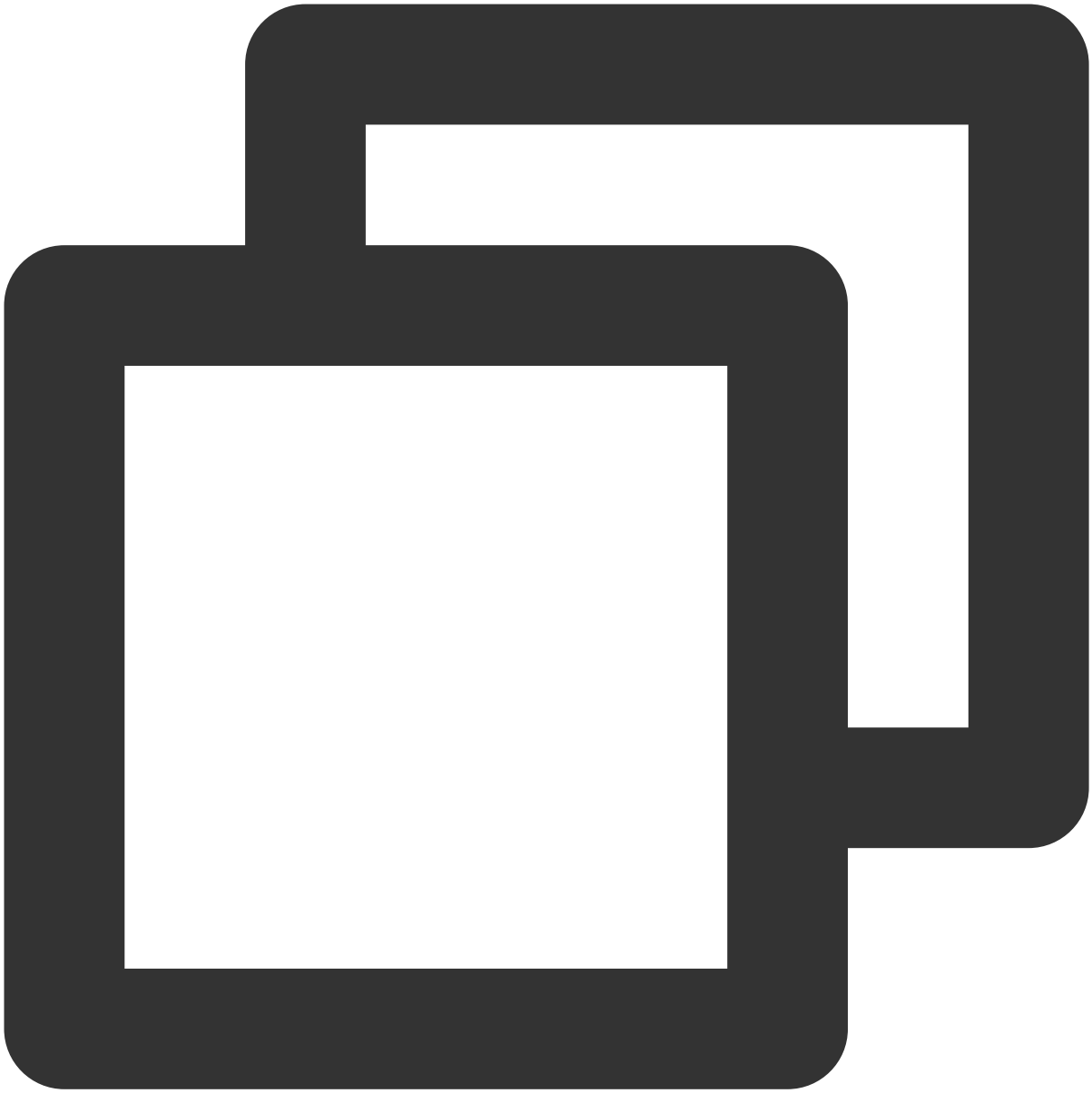
说明：

需要注册成功之后才能调用，不然返回为 NULL。

参数说明

context：App 上下文对象。

示例代码



```
XGPushConfig.getOtherPushToken(context);
```

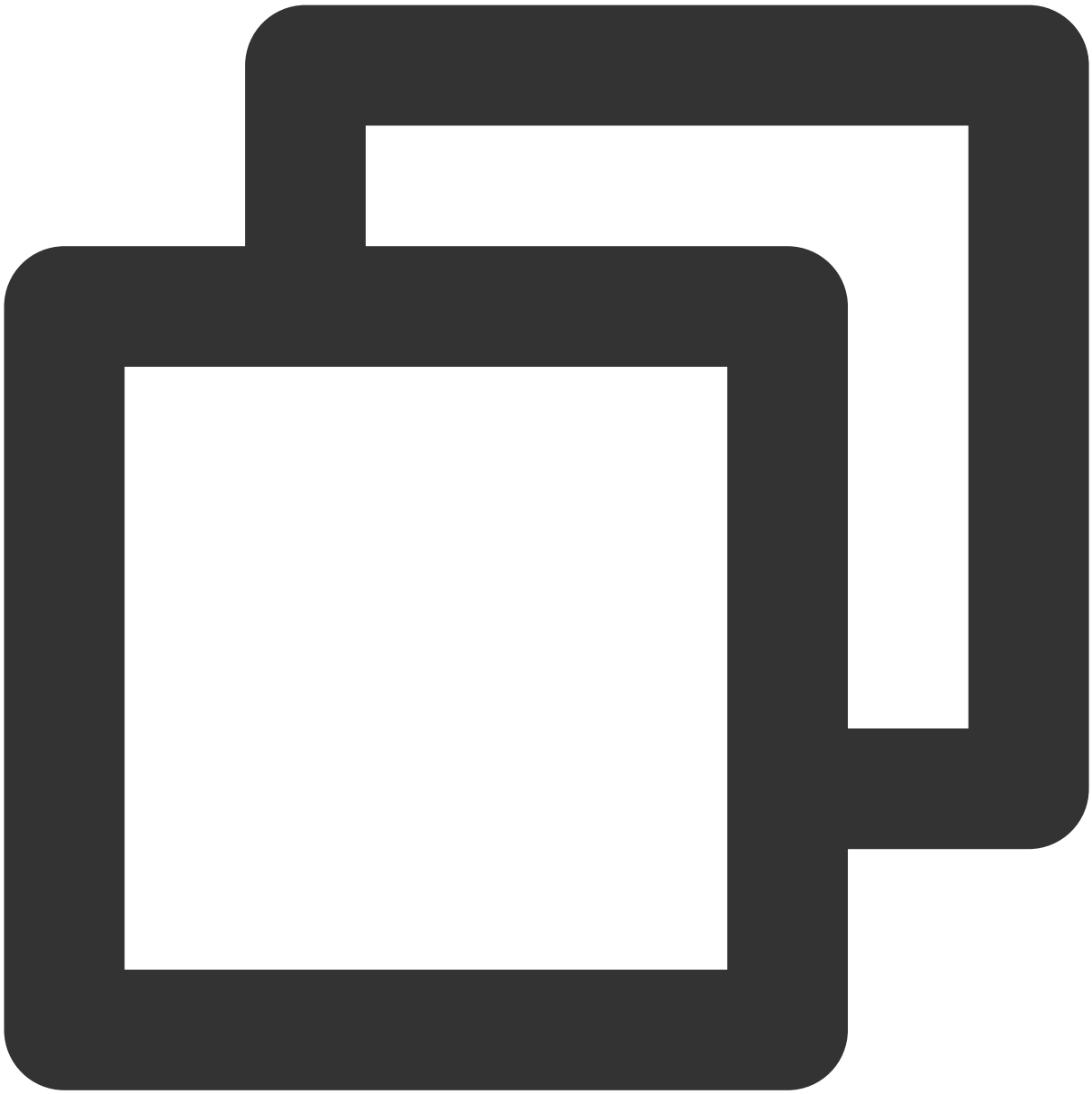
返回值

成功时返回正常的 Token；失败时返回 null 或0。

在通知点击目标页面内获取随通知下发的自定义参数（custom_content）内容

SDK 1.3.2.0 新增，当通知被点击打开时，可以在通知设置的目标页面内，通过此接口直接获取创建推送任务时配置的自定义参数（custom_content）内容；

详细使用方式参见 通知点击跳转。



```
public static String getCustomContentFromIntent(Context context, Intent intent)
```

返回值

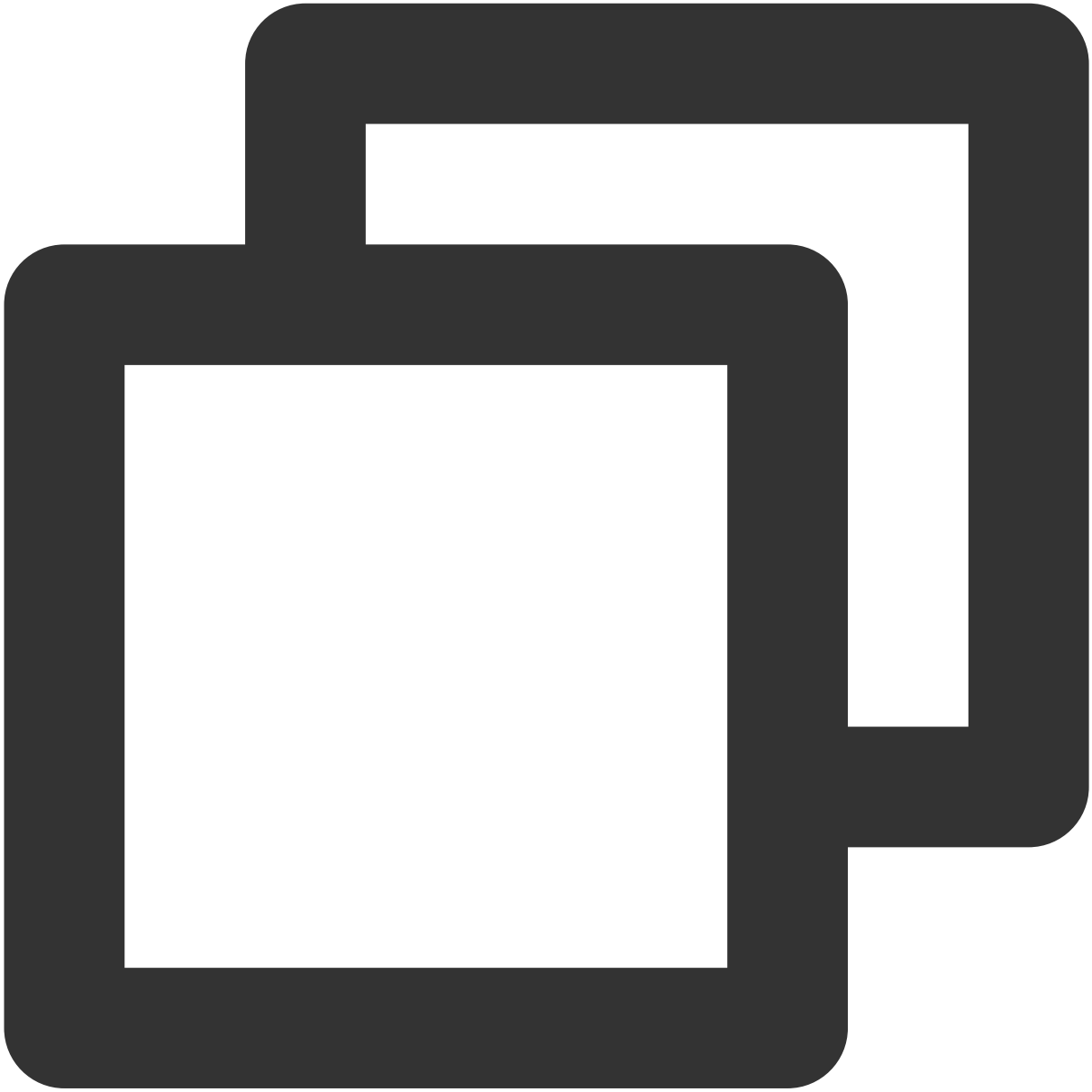
随推送下发的自定义参数（custom_content）字符串

参数说明

context：Context 对象， 不能为 null。

intent：activity intent；在 onCreate 内请直接传入 this.getIntent()；在 onNewIntent 内请直接传入回调的 intent。

示例代码

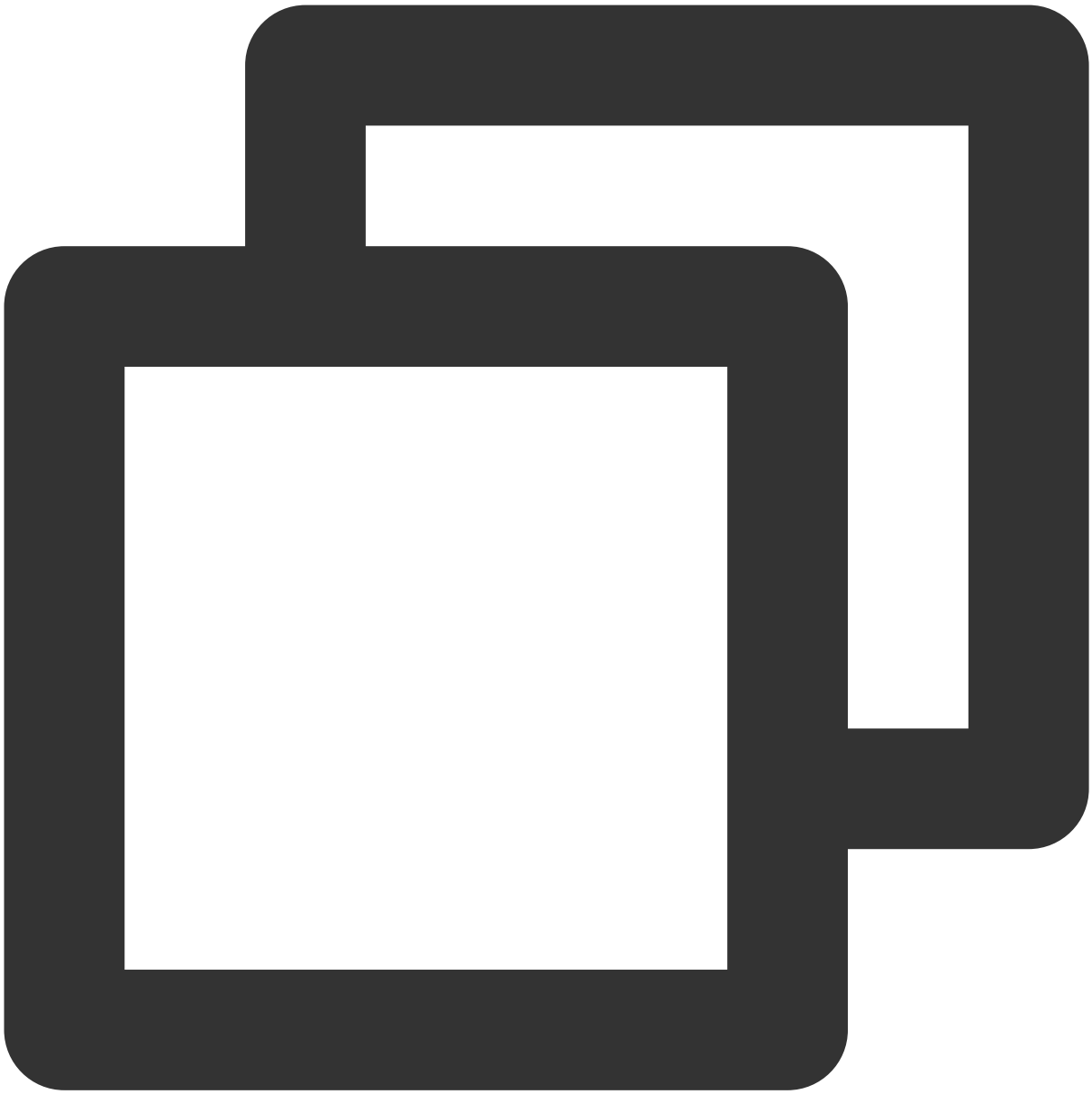


```
String customContent = XGPushManager.getCustomContentFromIntent(this, intent);
```

设置 AccessID

接口说明

如果已在 AndroidManifest.xml 配置过，无需再次调用；如果二者都存在，则以本接口为准。



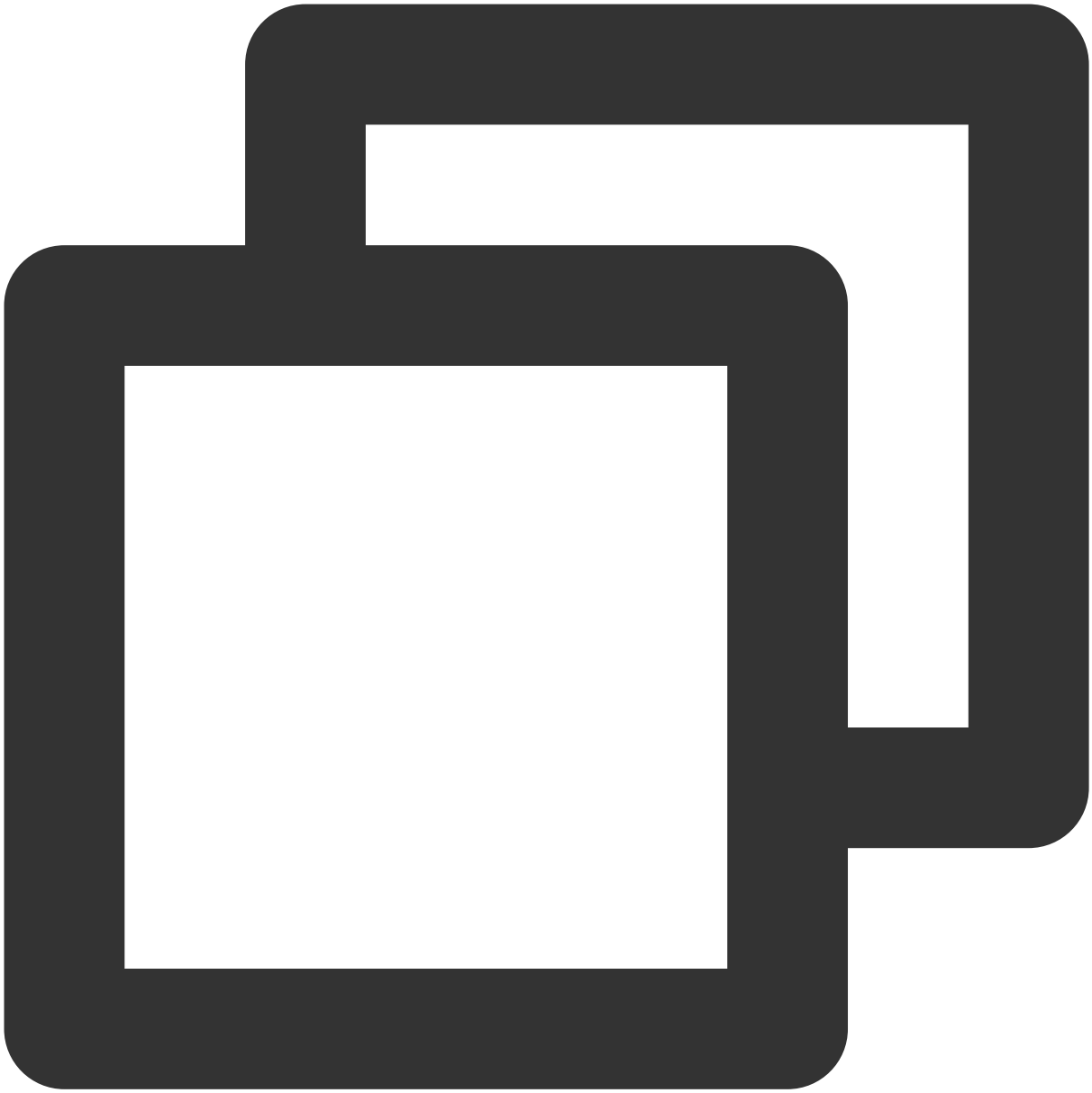
```
public static boolean setAccessId(Context context, long accessId)
```

参数说明

Context：对象。

accessId：前台注册得到的 accessId。

示例代码



```
long accessId = 0L; // 当前应用的 accessId
XGPushConfig.setAccessId(context, accessId);
```

返回值

true：成功。

false：失败。

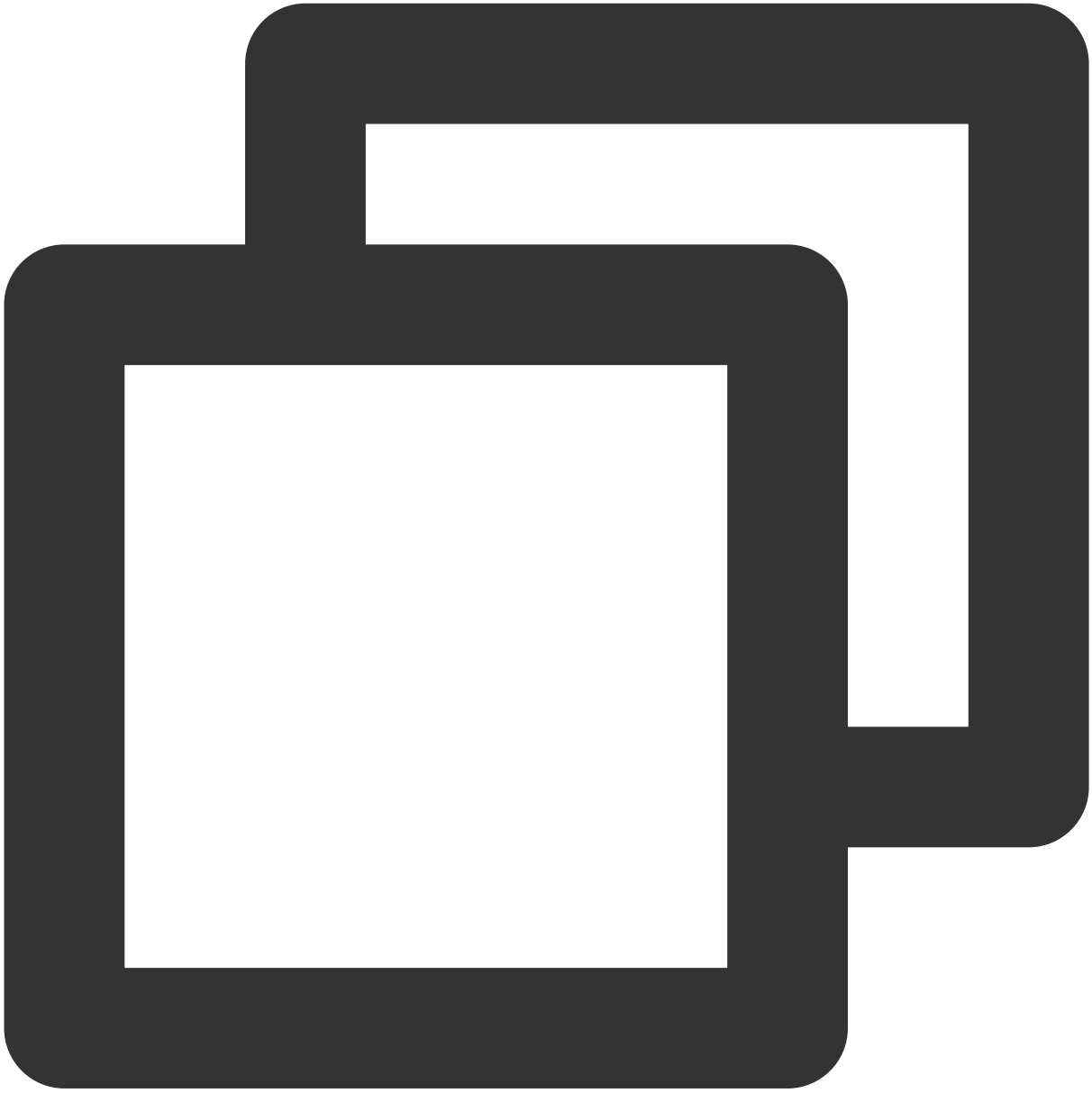
说明：

通过本接口设置的 accessId 会同时存储在文件中。

设置 AccessKey

接口说明

如果已在 AndroidManifest.xml 配置过，无需再次调用；如果二者都存在，则以本接口为准。



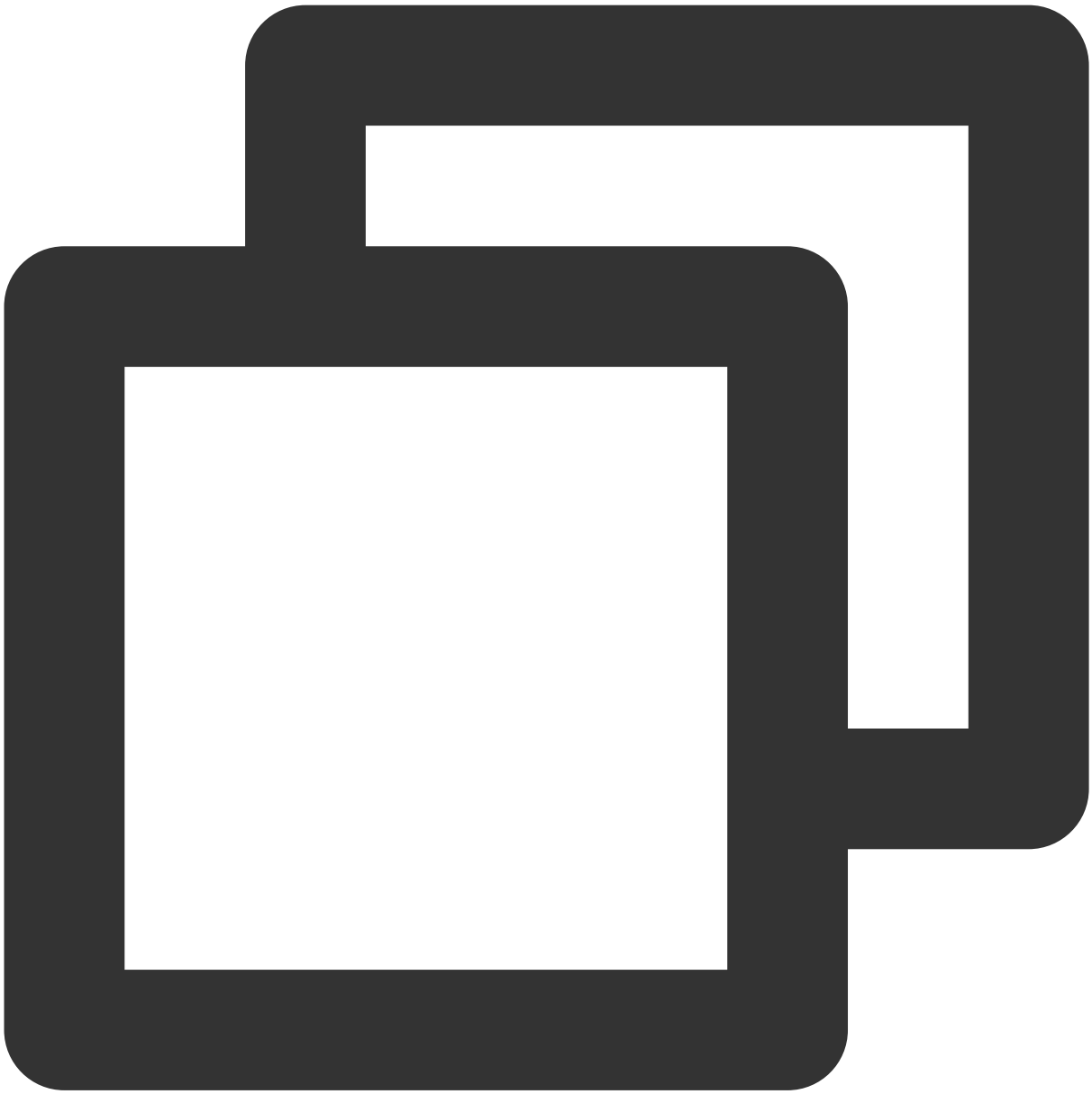
```
public static boolean setAccessKey(Context context, String accessKey)
```

参数说明

Context：对象。

accessKey：前台注册得到的 accesskey。

示例代码



```
String accessKey = ""; // 您应用的 accessKey
XGPushConfig.setAccessKey(context, accessKey);
```

返回值

true：成功。

false：失败。

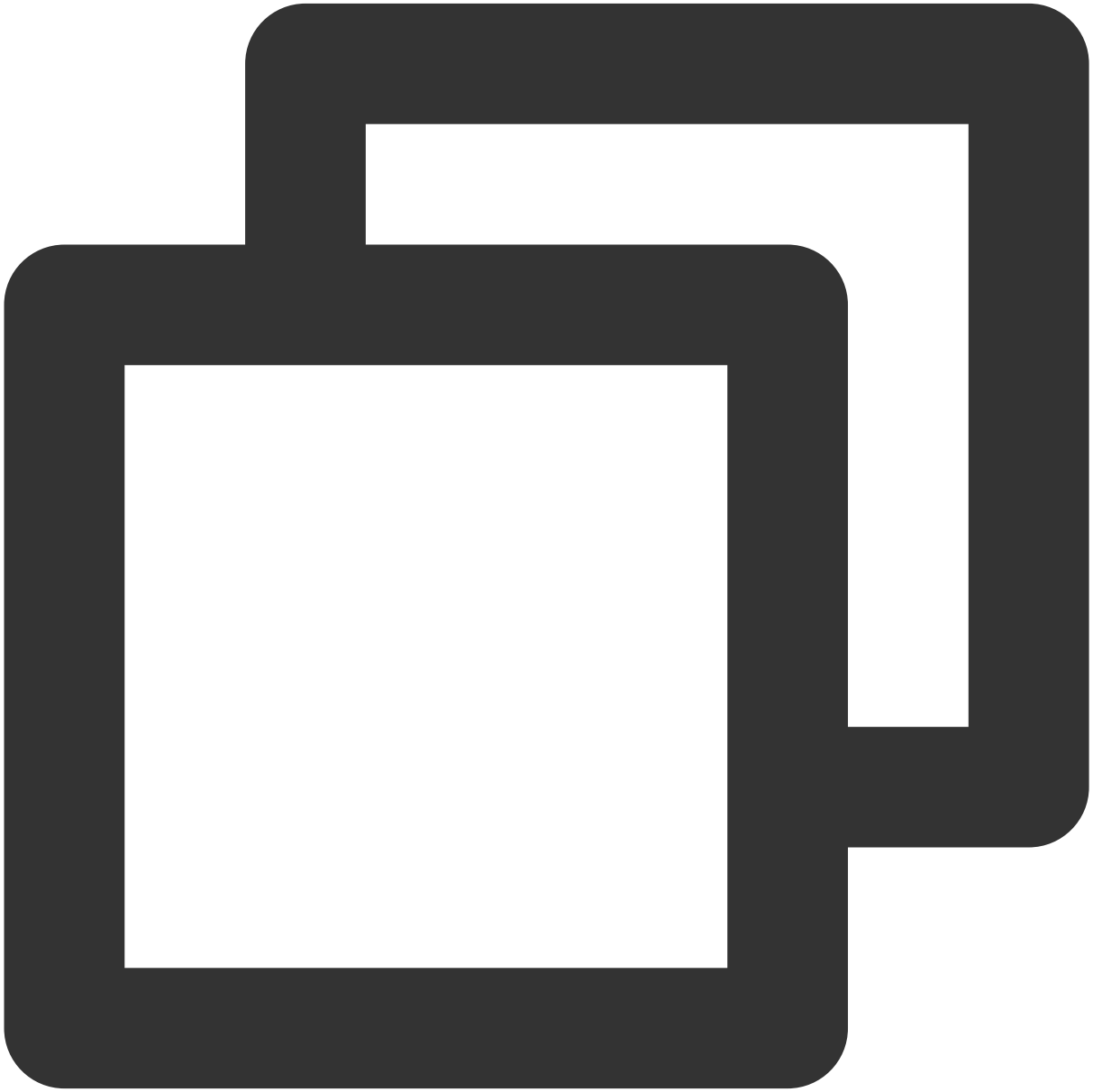
说明：

通过本接口设置的 accessKey 会同时存储在文件中。

新增日志上报接口

接口说明

开发者如果发现 TPush 相关功能异常，可以调用该接口，触发本地 Push 日志的上报，反馈问题时，请联系 在线客服 将文件地址给到我们，便于我们排查问题。



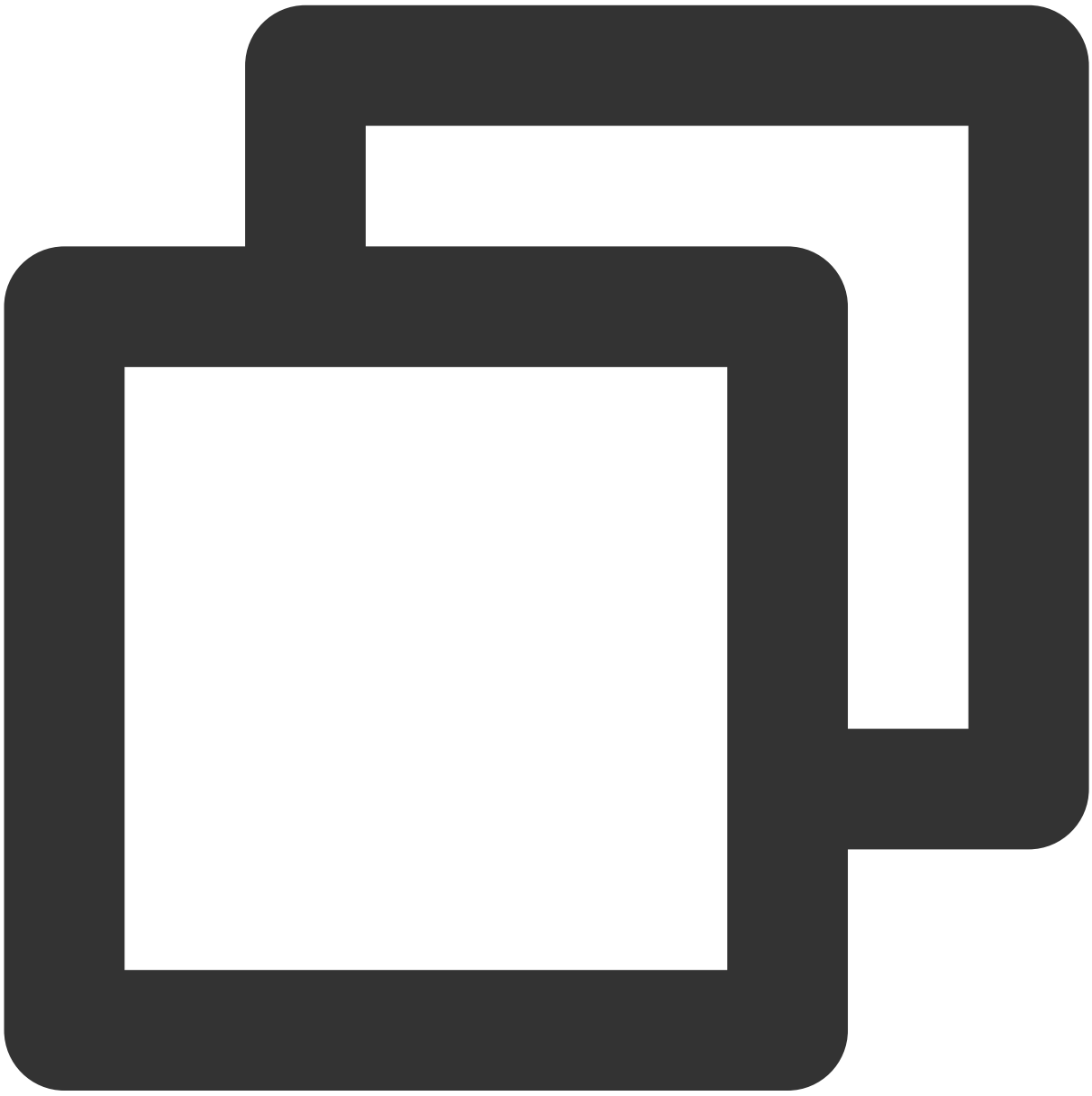
```
public static void uploadLogFile(Context context, HttpRequestCallback httpRequestCa
```

参数说明

context：Context 对象， 不能为 null。

httpRequestCallback： 上传日志结果回调，主要包括操作成功和失败，不能为 null。

示例代码



```
XGPushManager.uploadLogFile(context, new HttpRequestCallback() {
  @Override
  public void onSuccess(String result) {
           Log.d("TPush", "上传成功，文件地址：" + result);
```

```
    }
    @Override
    public void onFailure(int errCode, String errMsg) {
            Log.d("TPush", "上传失败, 错误码：" + errCode + ",错误信息：" + errMsg);
    }
});
```

说明：

首先需要开启 `XGPushConfig.enableDebug(this, true);` 。

厂商通道接入指南
华为通道 V5 接入

最近更新时间：2024-01-16 17:39:39

操作场景

移动推送跟进各厂商通道推送服务的更新进度，提供集成华为推送 HMS Core Push SDK 的插件依赖包供用户选择使用。

注意：

华为推送只有在签名发布包环境下，才可注册厂商通道成功并通过厂商通道进行推送。

华为通道不支持抵达回调，支持点击回调。

在华为推送平台配置应用

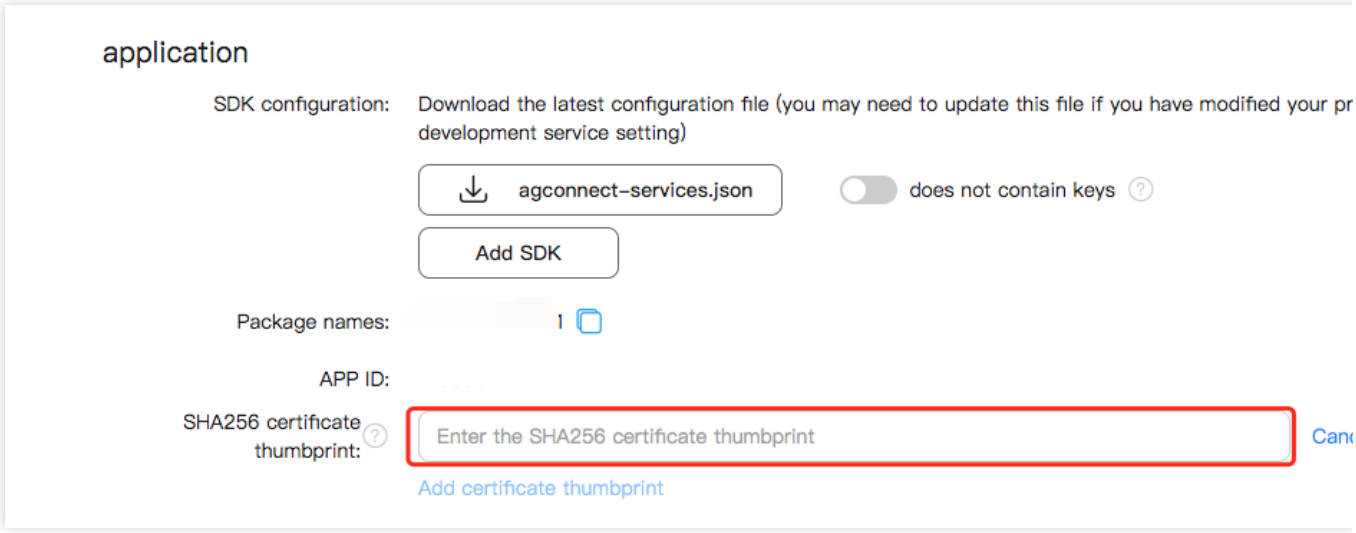
获取密钥

1. 进入 华为开放平台。
2. 注册和登录开发者账号，详情参见 账号注册认证（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，详情参见 创建应用（应用包名需跟您在移动推送平台填写的一致）。
4. 进入**我的项目** > **项目设置** > **常规**中的应用获取并复制APPID和Client Secret，填入 **移动推送控制台**>**配置管理**>**基础配置**>**华为官方推送通道**栏目中。

配置 SHA256 证书指纹

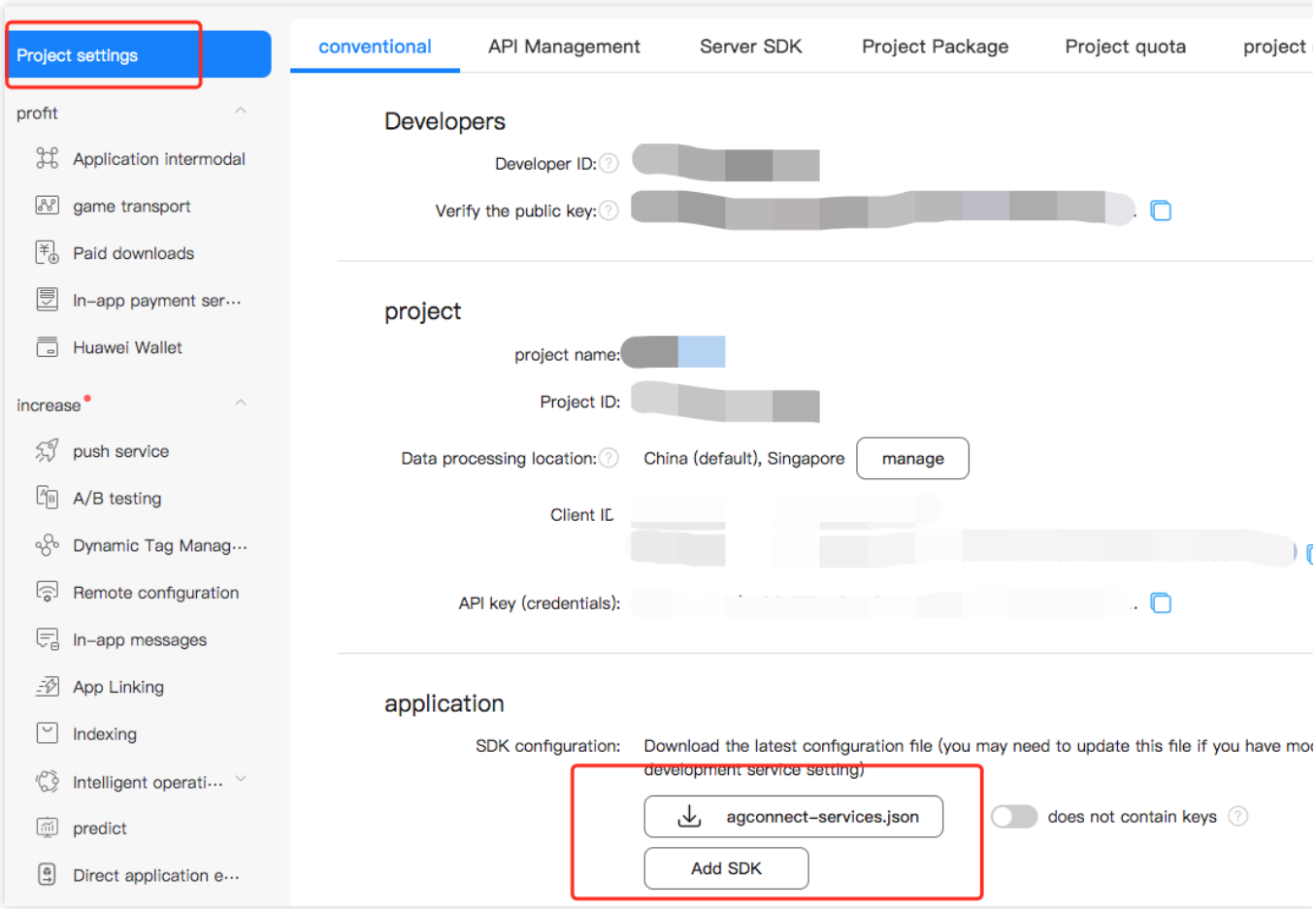
获取 SHA256 证书指纹，并在华为推送平台中配置证书指纹，**单击**

保存。证书指纹获取可参见 生成签名证书指纹。



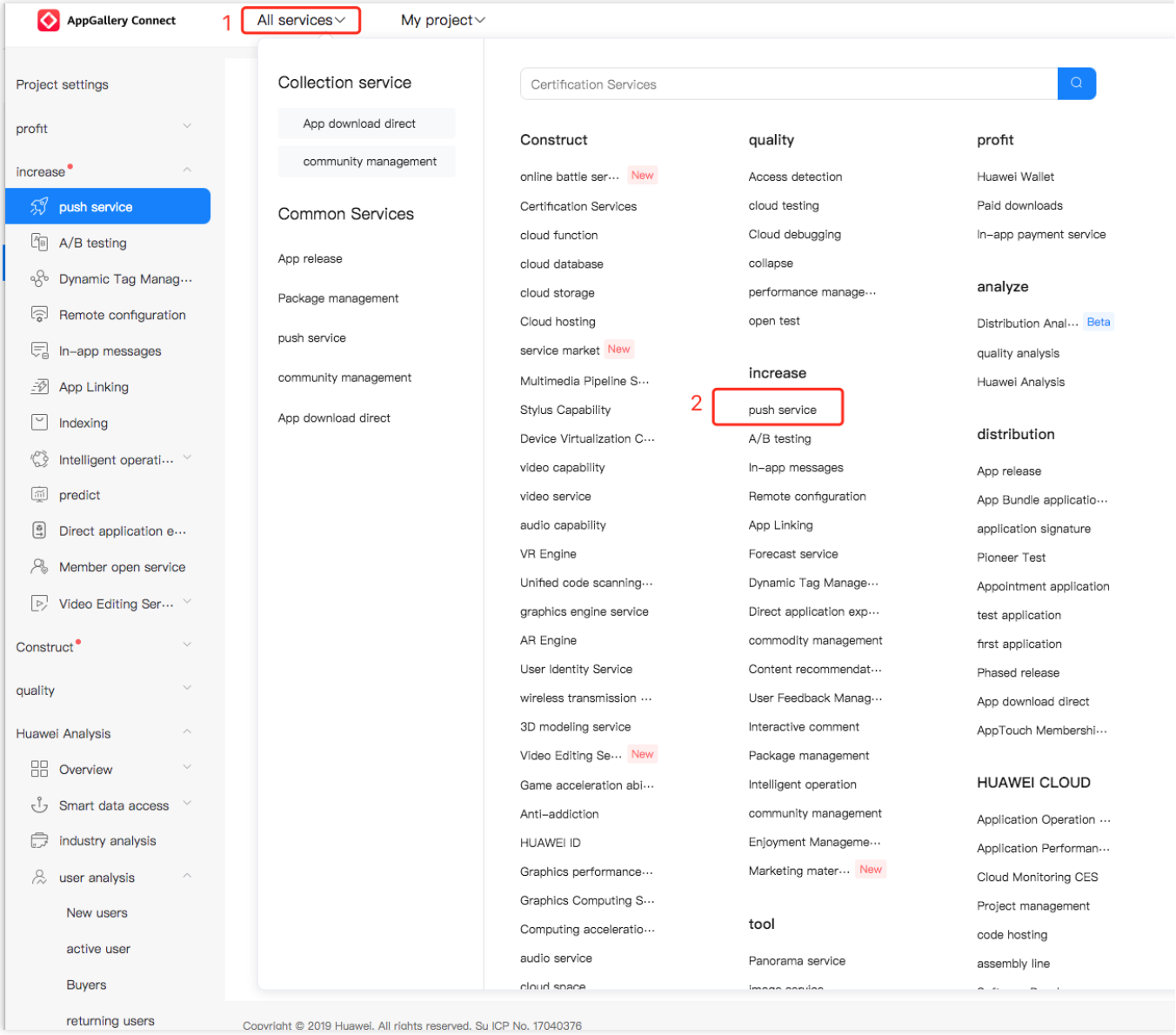
获取华为推送配置文件

登录华为开放平台，进入**我的项目**> 选择项目 > **项目设置**，下载华为应用最新配置文件 agconnect-services.json。

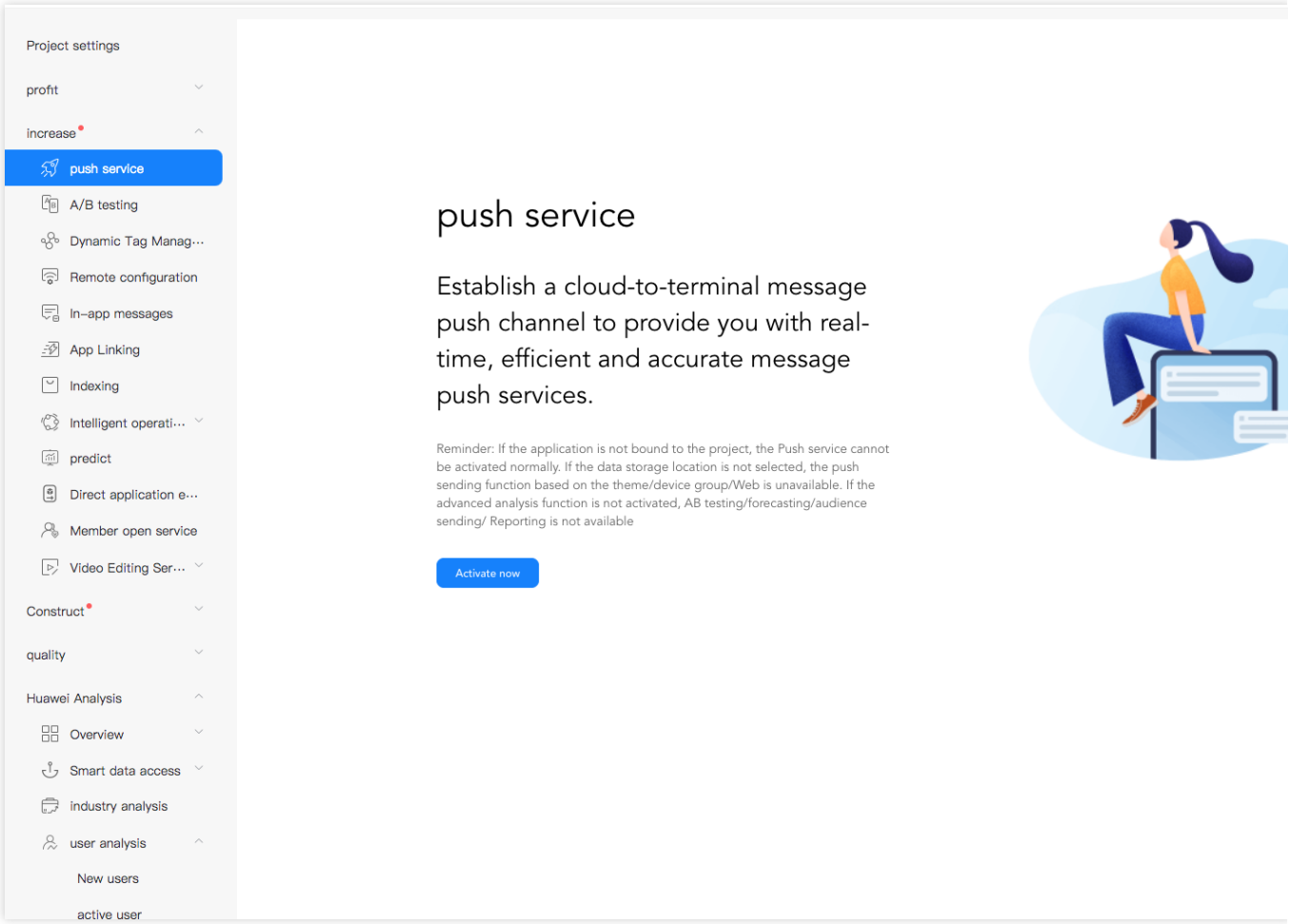


打开推送服务开关

1. 在华为推送平台，单击**全部服务>推送服务**，进入推送服务页面。



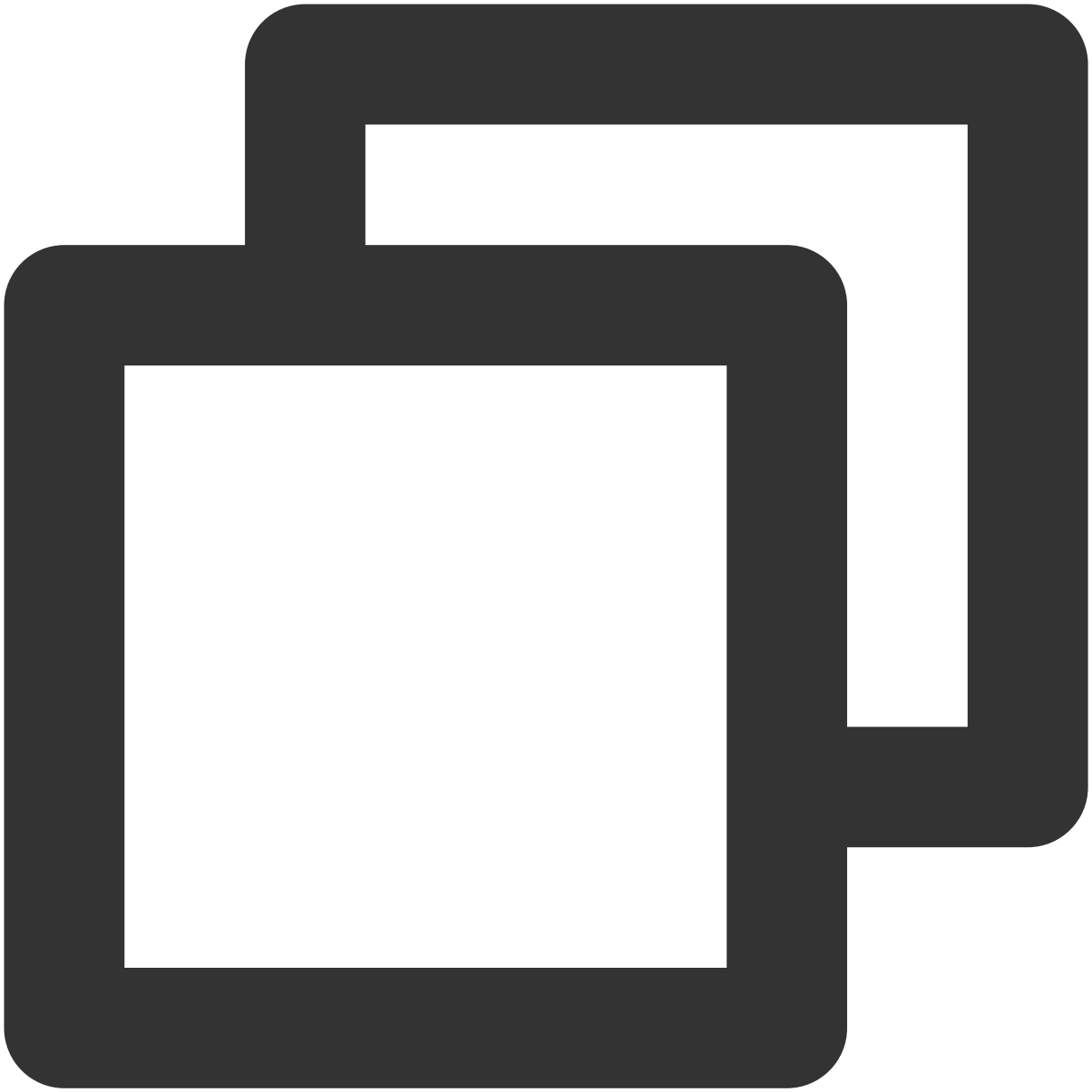
2. 在“推送服务”页面，单击**立即开通**，详情请参见 打开推送服务开关。



SDK 集成（二选一）

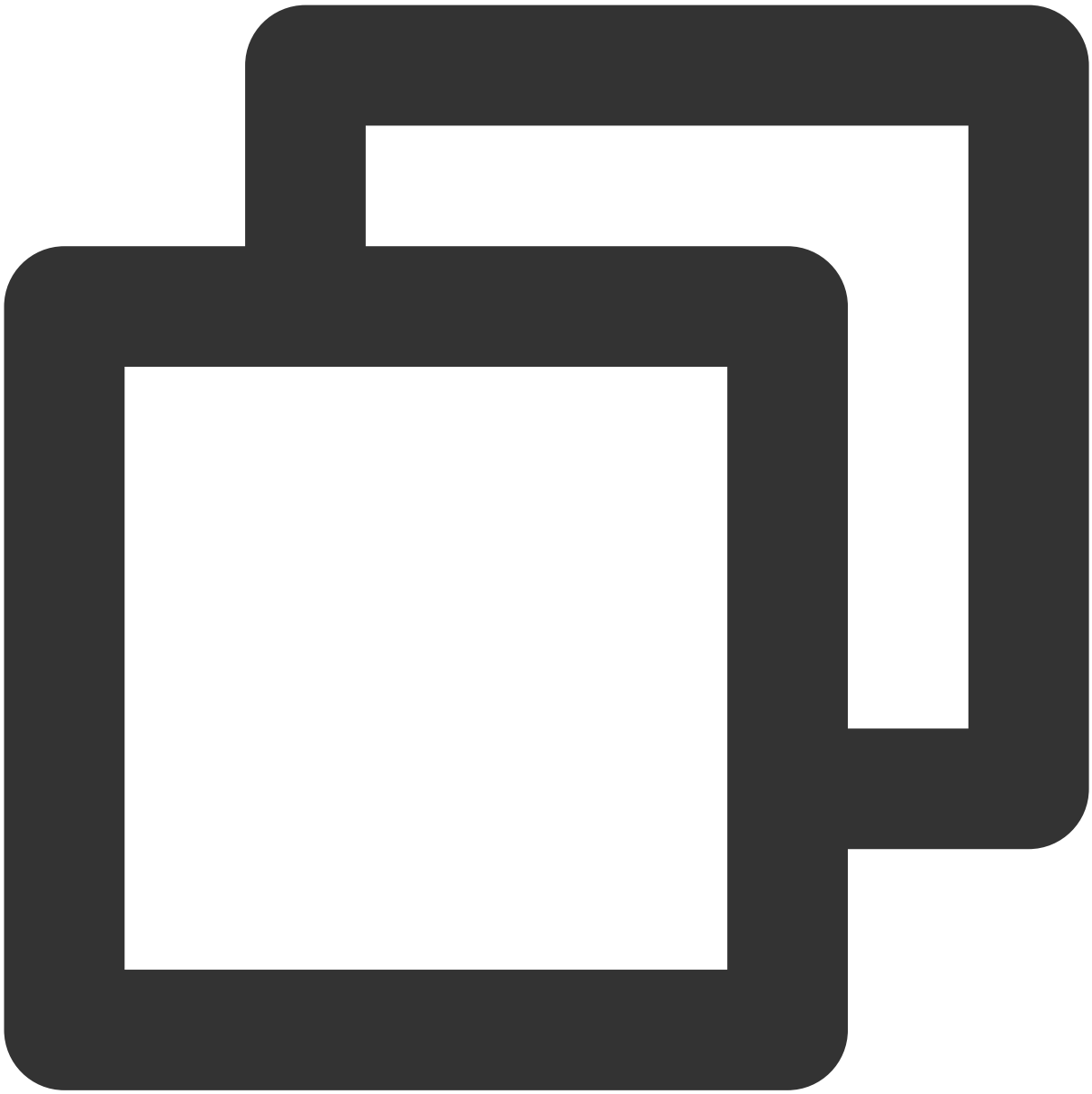
Android Studio Gradle 自动集成

1. 在 Android 项目级目录 build.gradle 文件，**buildscript > repositories & dependencies** 下分别添加华为仓库地址和 HMS gradle 插件依赖：



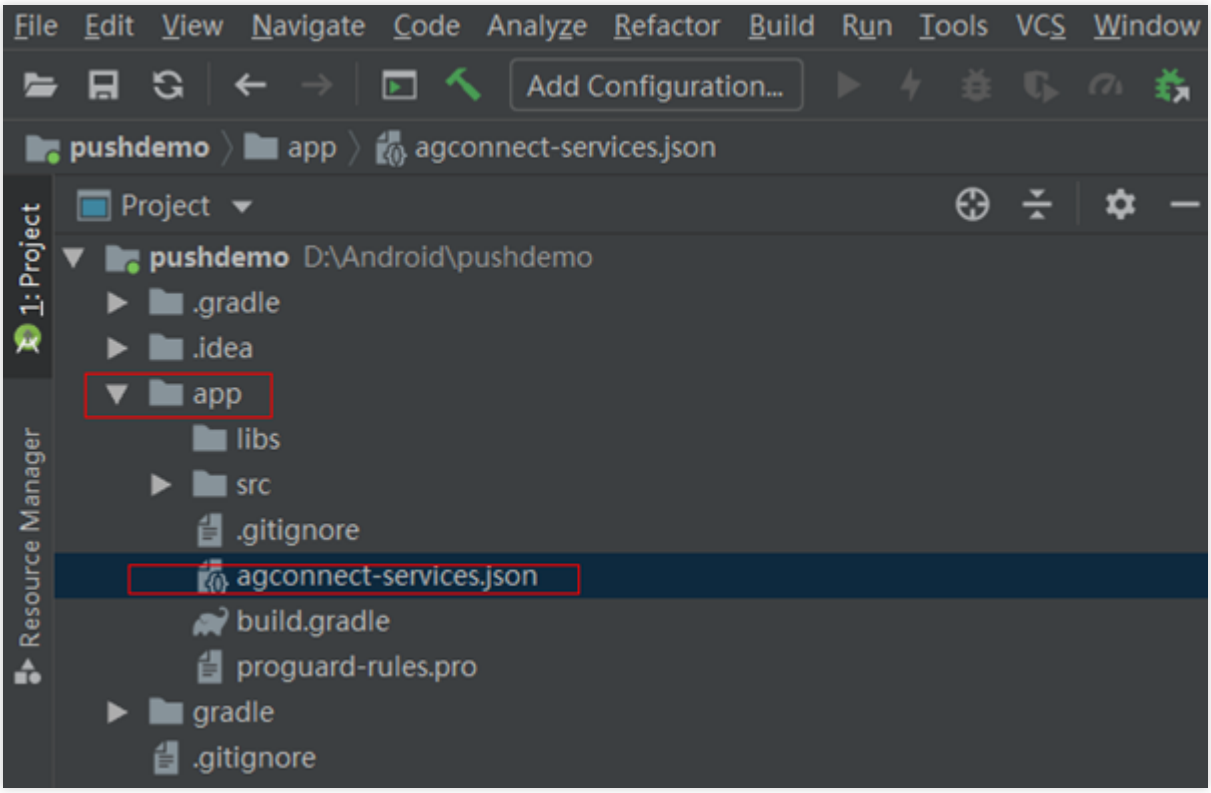
```
buildscript {
 repositories {
     google()
     maven {url 'https://developer.huawei.com/repo/'}      // 华为 maven 仓库地址
 }
 dependencies {
     // 其他classpath配置
     classpath 'com.huawei.agconnect:agcp:1.6.0.300'      // 华为推送 gradle 插件依赖
 }
}
```

2. 在 Android 项目级目录 build.gradle 文件，**allprojects**>**repositories**下添加华为依赖仓库地址：

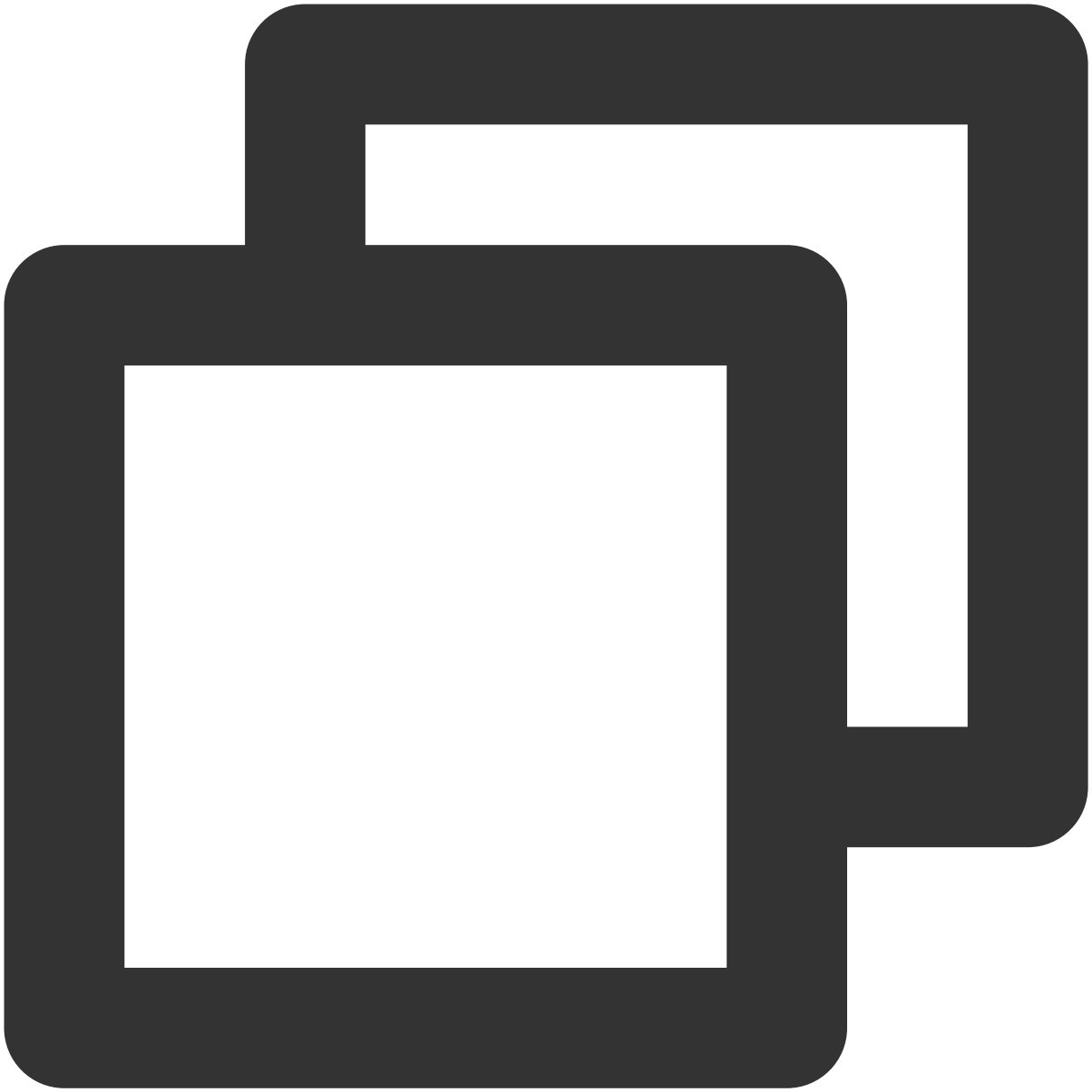


```
allprojects {
 repositories {
     google()
     maven {url 'https://developer.huawei.com/repo/'}      // 华为 maven 仓库地址
 }
}
```

3. 将从华为推送平台获取的应用配置文件 agconnect-services.json 拷贝到 app 模块目录下（请勿放在子模块下）。

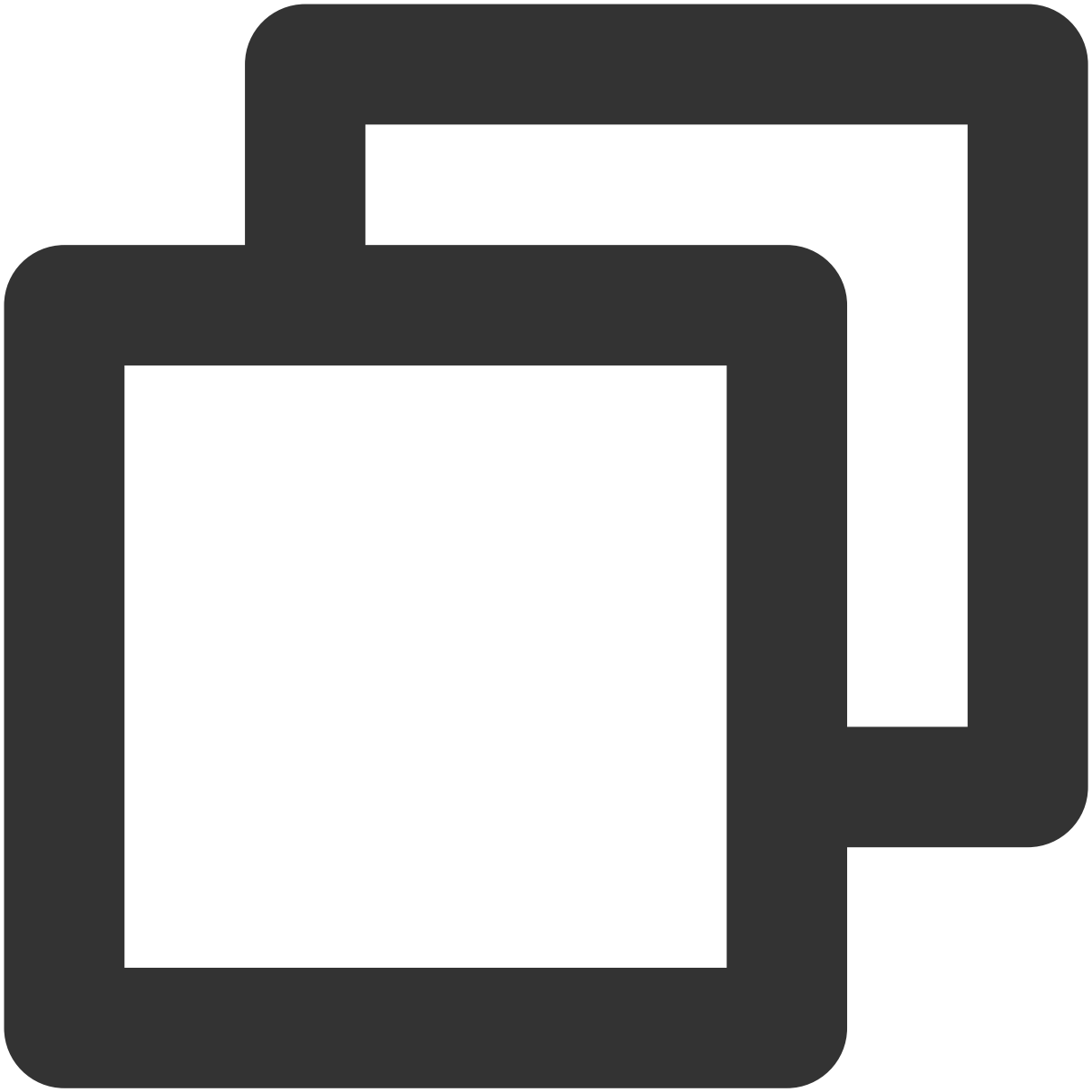


4. 在 app 模块下 build.gradle 文件头部添加以下配置（请勿放在子模块 build.gradle 下）：



```
// app 其他 gradle 插件
apply plugin: 'com.huawei.agconnect'      // HMS SDK gradle 插件
android {
 // app 配置内容
}
```

5. 在 app 模块下 build.gradle 文件内导入华为推送相关依赖：



```
dependencies {
    // ... 程序其他依赖
    implementation 'com.tencent.tpns:huawei:[VERSION]-release'      //  华为推送 [VI
    implementation 'com.huawei.hms:push:6.5.0.300'       // HMS Core Push 模块依赖包
    }
```

说明：

华为推送 hms:push 依赖自6.1.300起适配 Android 11预置 `<queries>` 标签，请注意升级 Android Studio 至3.6.1或更高版本、Android Gradle 插件至 3.5.4或更高版本，否则可能导致工程构建出错。

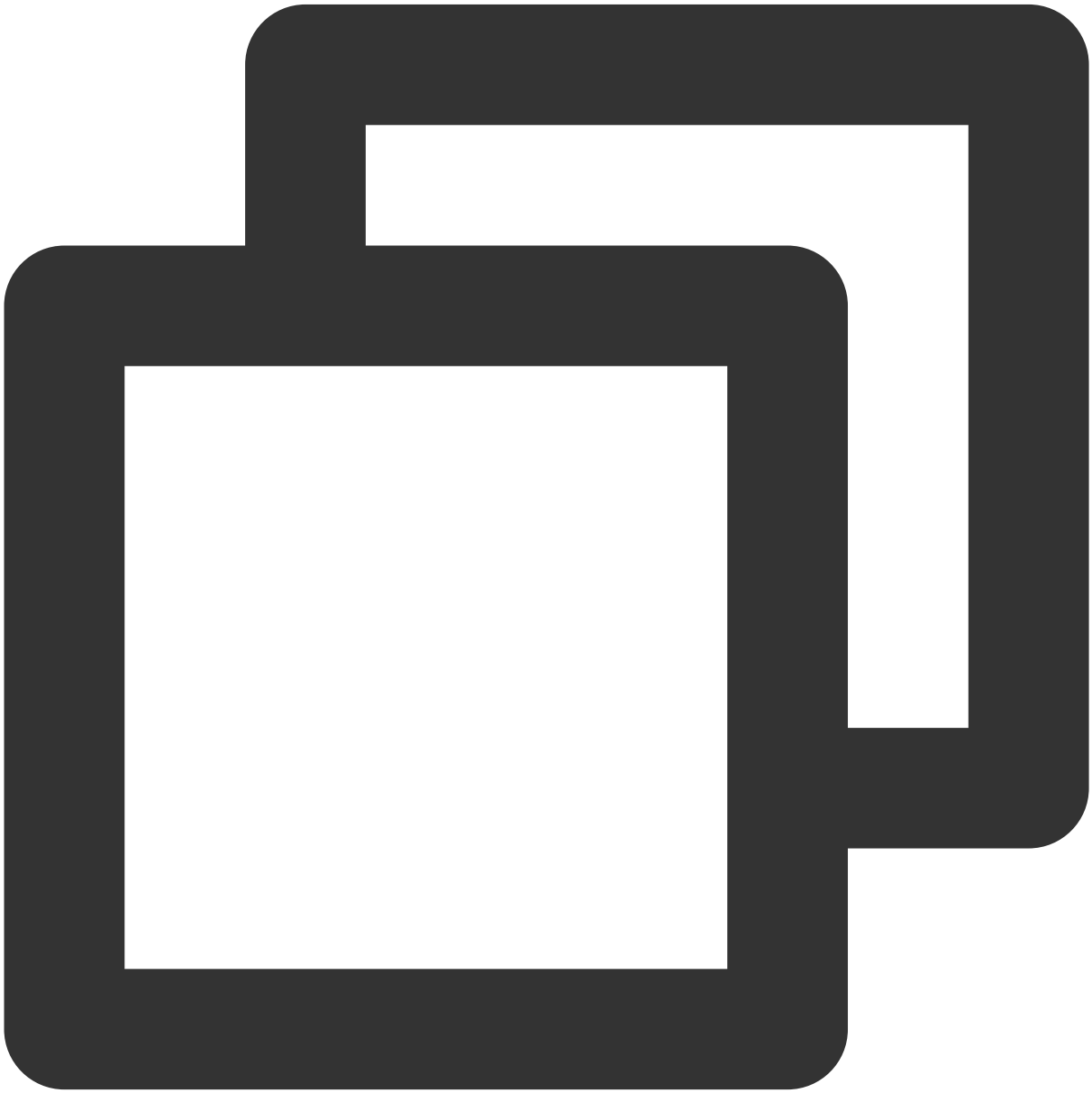
华为推送 [VERSION] 为当前最新 SDK 版本号，版本号可在 Android SDK 发布动态 查看。

移动推送 Android SDK 自1.2.1.3版本起正式支持华为推送 V5 版本，请使用1.2.1.3及以上版本的移动推送华为依赖以避免集成冲突问题。

Android Studio 手动集成

针对开发者内部构建环境无法访问华为 maven 仓库的情况，提供以下手动集成方法。

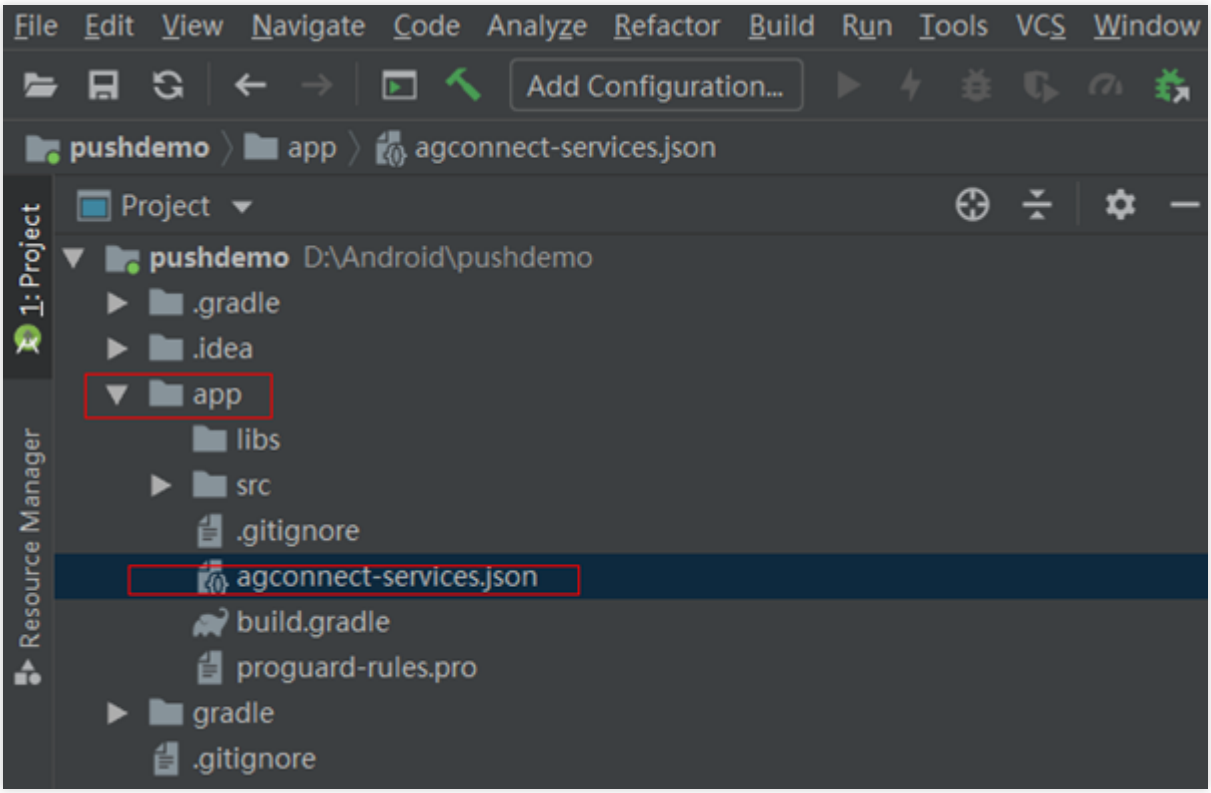
1. 下载 SDK 安装包。
2. 打开 Other-Push-jar 文件夹， 导入 huaweiv5 推送相关依赖包，将全部 jar、aar 包复制到项目工程中。
3. 在 Android 项目级目录 build.gradle 文件，**buildscript** > **dependencies**下添加 HMS gradle 插件的依赖：



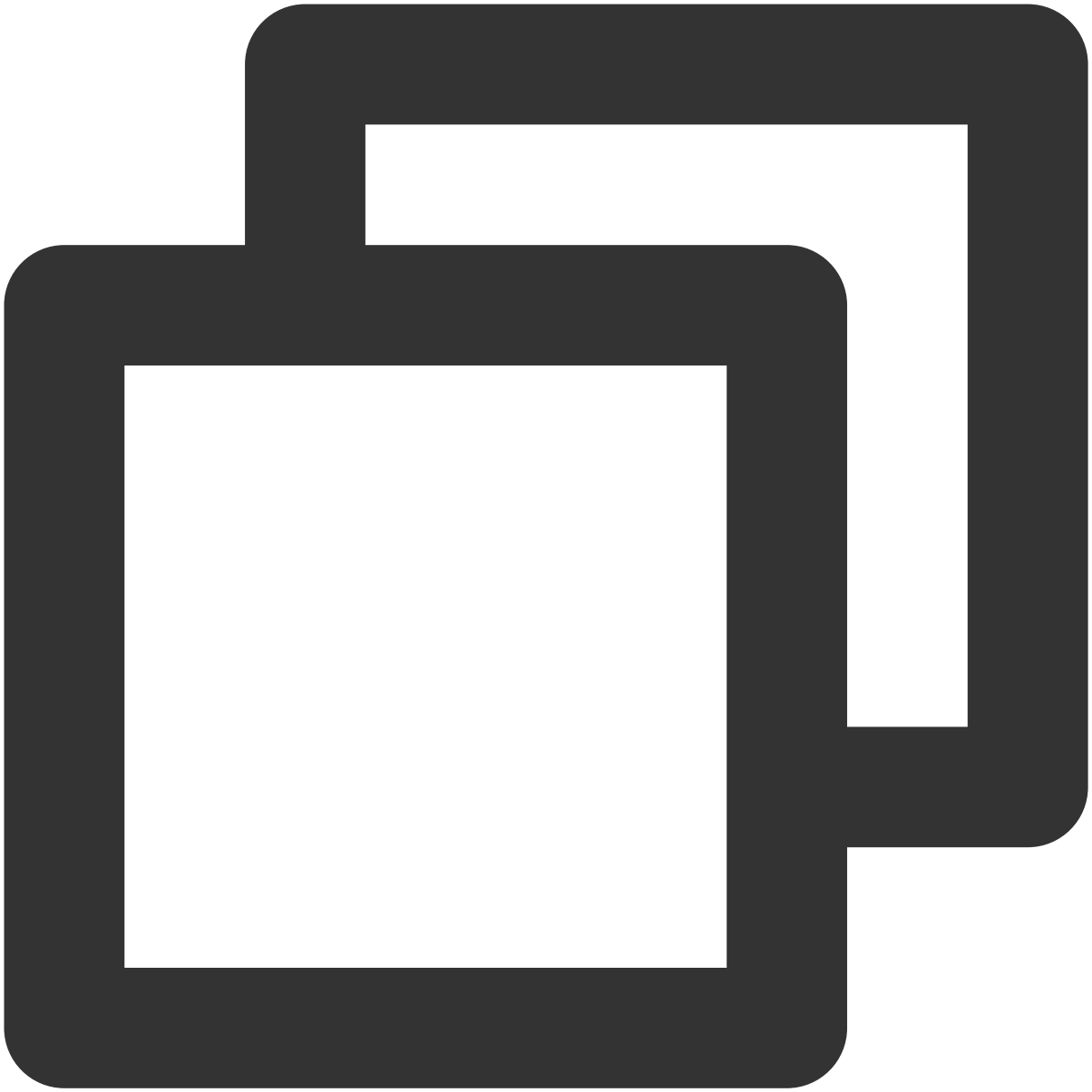
```
buildscript {
```

```
    repositories {
        google()
        jcenter()
    }
    dependencies {
        // 其他 classpath 配置
        classpath files('app/libs/agcp-1.4.1.300.jar')      // 华为推送 gradle 插件依赖
    }
}
```

4. 将从华为推送平台获取的应用配置文件 agconnect-services.json 拷贝到 app 模块目录下。

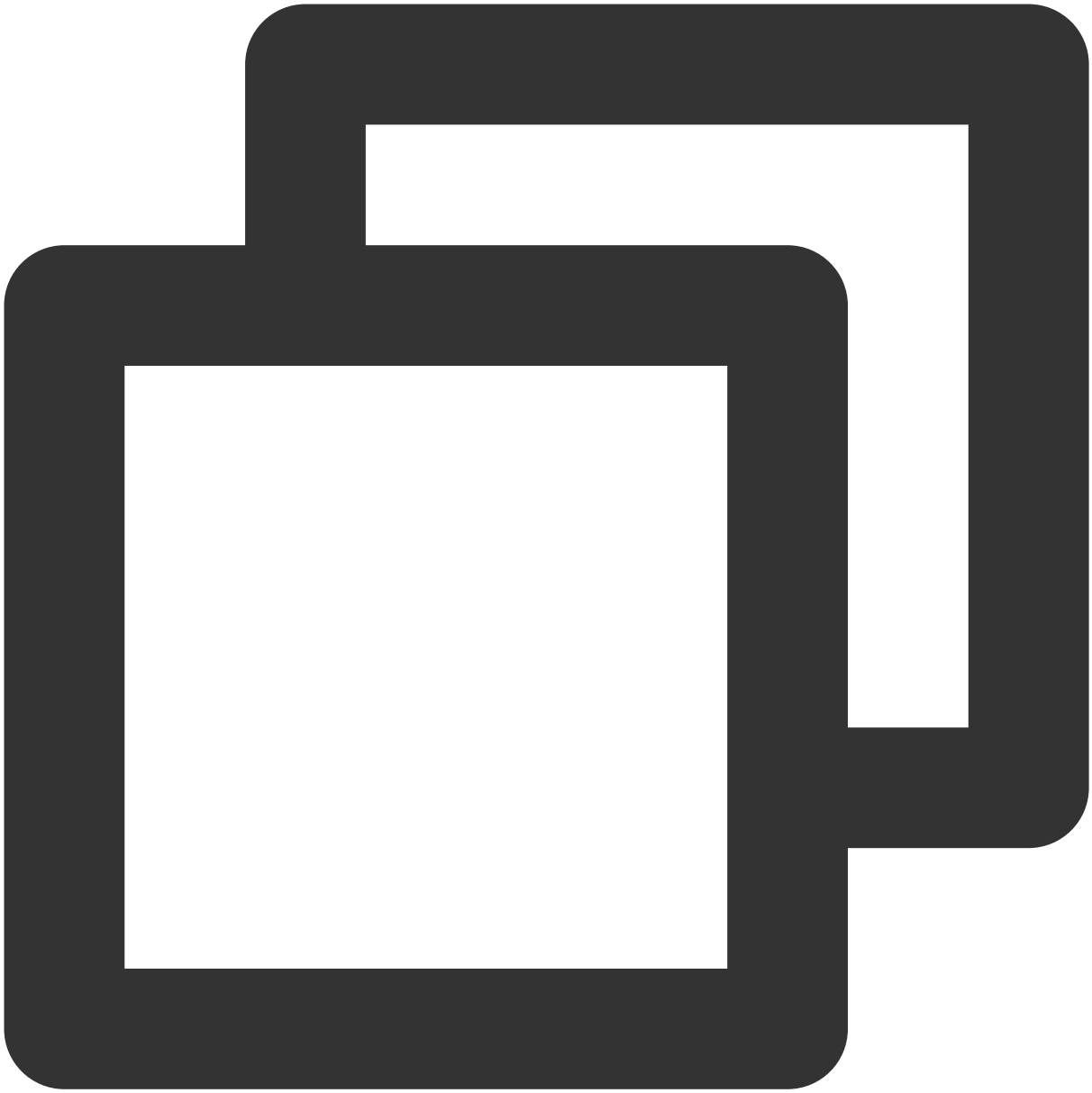


5. 在 app 模块下 build.gradle 文件头部添加以下配置：



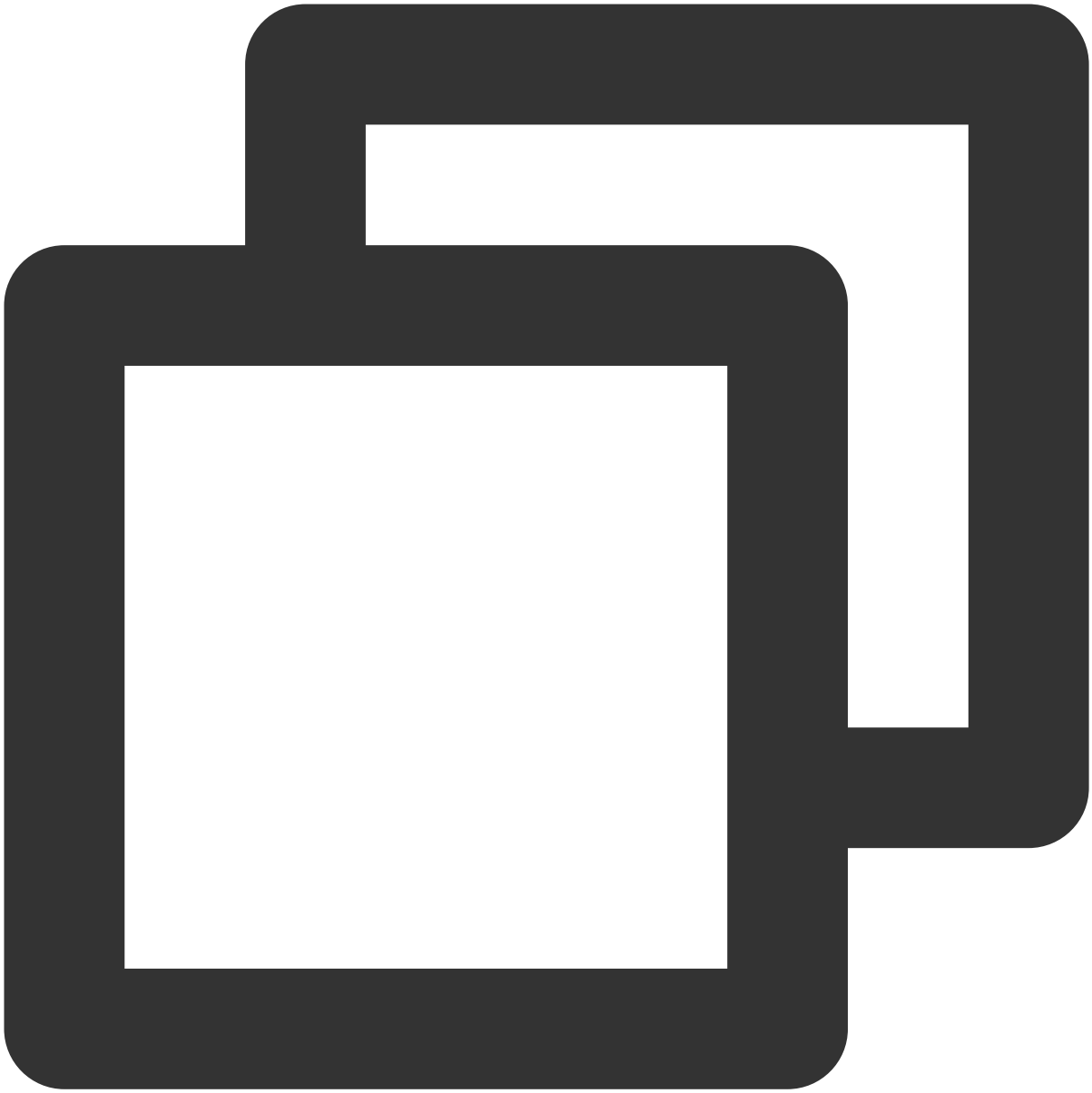
```
// app 其他 gradle 插件
apply plugin: 'com.huawei.agconnect'      // HMS SDK V4 gradle 插件
android {
 // app 配置内容
}
```

6. 在 app 模块下 build.gradle 文件内导入华为推送相关依赖：



```
dependencies {
      // ... 程序其他依赖
      implementation files('libs/tpns-huaweiv5-1.2.1.1.jar')      // 适用于 HMS Core
      implementation fileTree(include: ['*.aar'], dir: 'libs')    // HMS Core Push 模
 }
```

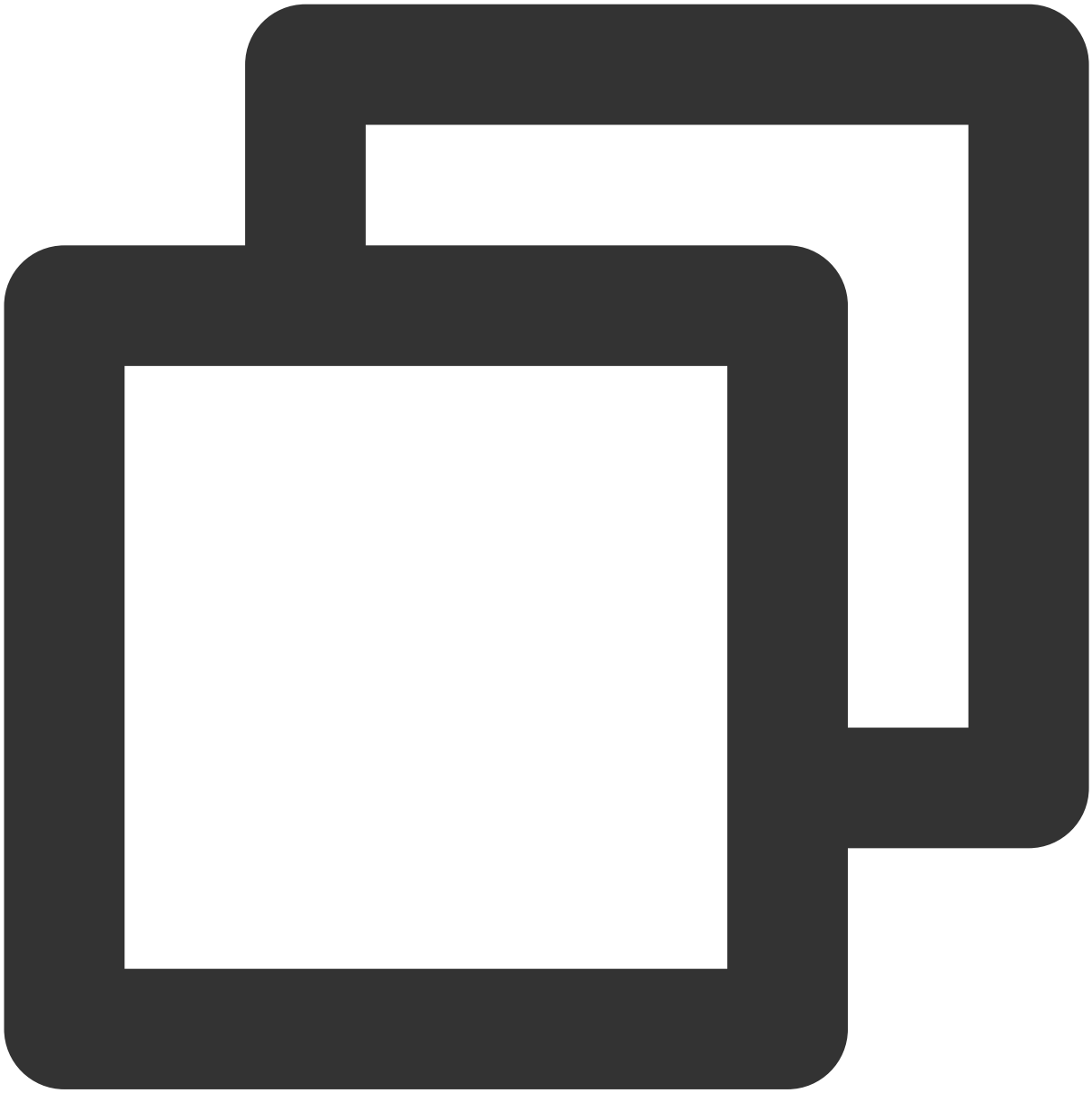
7. 在 manifest 文件 `<application> </application>` 标签内添加以下组件：



```
<application>
     <service
         android:name="com.huawei.android.hms.tpns.HWHmsMessageService"
         android:exported="false">
         <intent-filter>
             <action android:name="com.huawei.push.action.MESSAGING_EVENT" />
         </intent-filter>
     </service>
</application>
```

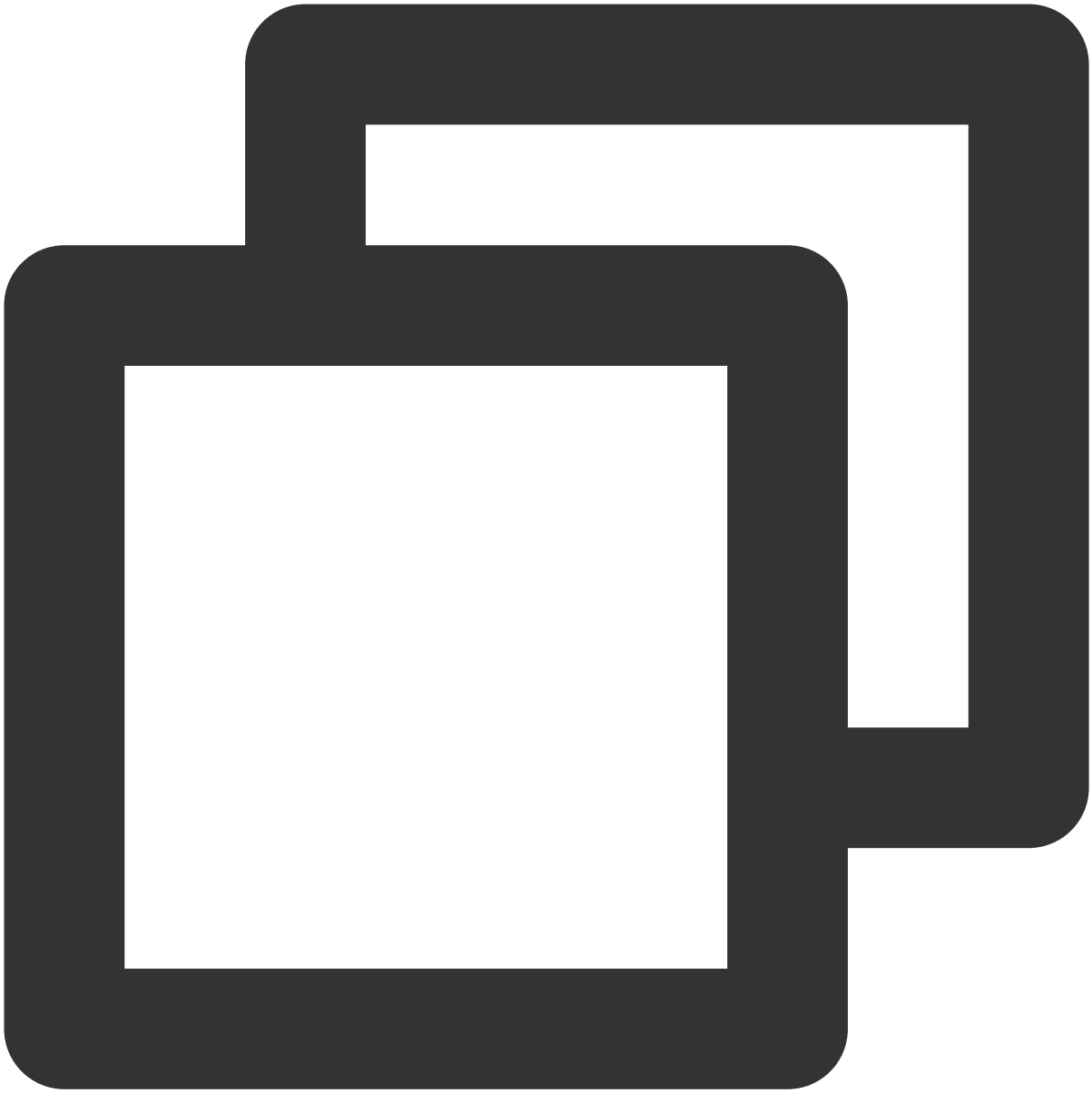
启动华为推送

1. 在调用移动推送注册接口 `XGPushManager.registerPush` 前，开启第三方推送接口：



```
//打开第三方推送
XGPushConfig.enableOtherPush(getApplicationContext(), true);
```

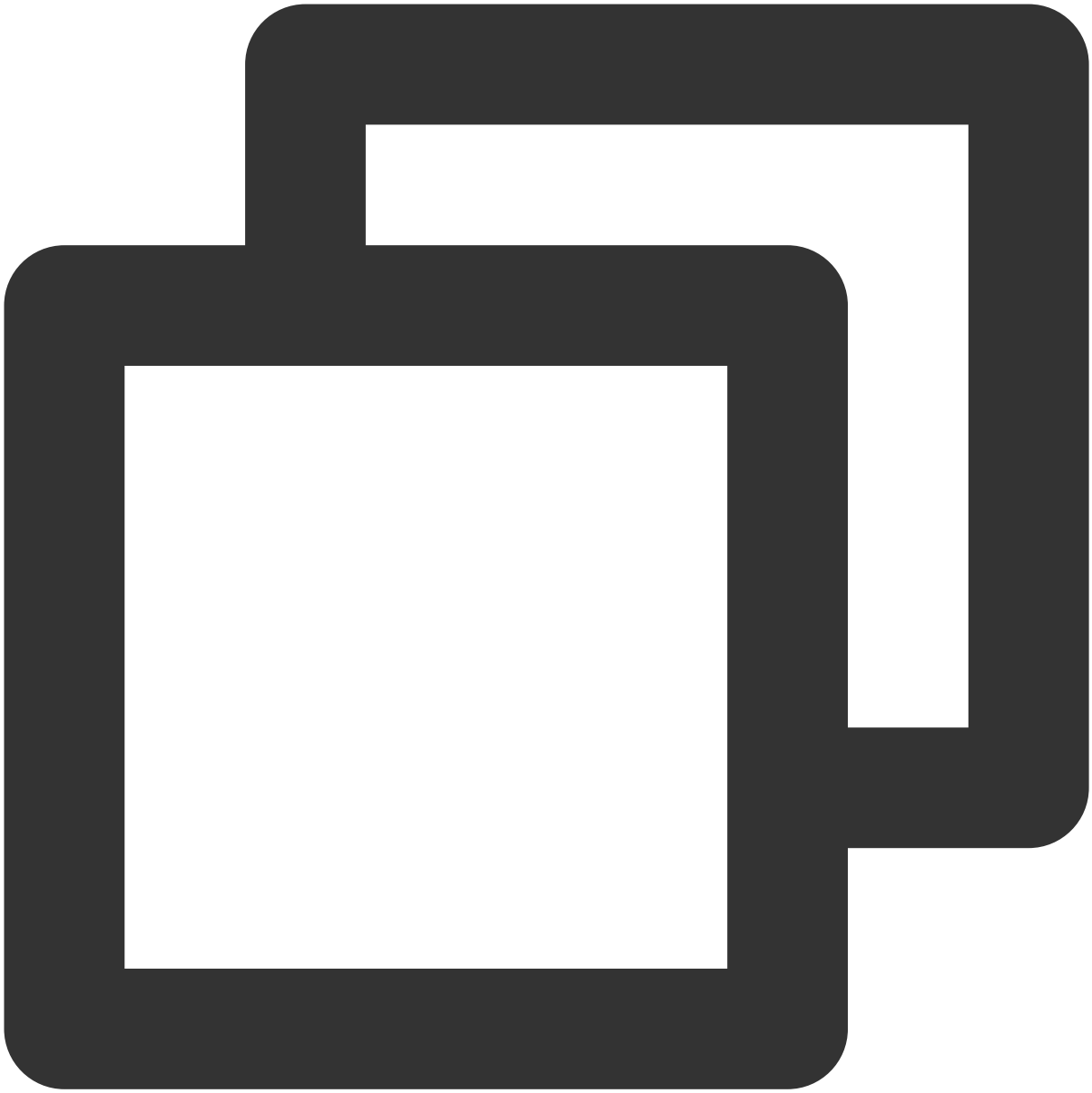
2. 注册成功的日志如下：



```
V/TPush: [XGPushConfig] isUsedOtherPush:true
E/xg.vip: get otherpush errcode: errCode : 0 , errMsg : success
V/TPush: [XGPushConfig] isUsedOtherPush:true
I/TPush: [OtherPushClient] handleUpdateToken other push token is : IQAAAACy0PsqAADx
```

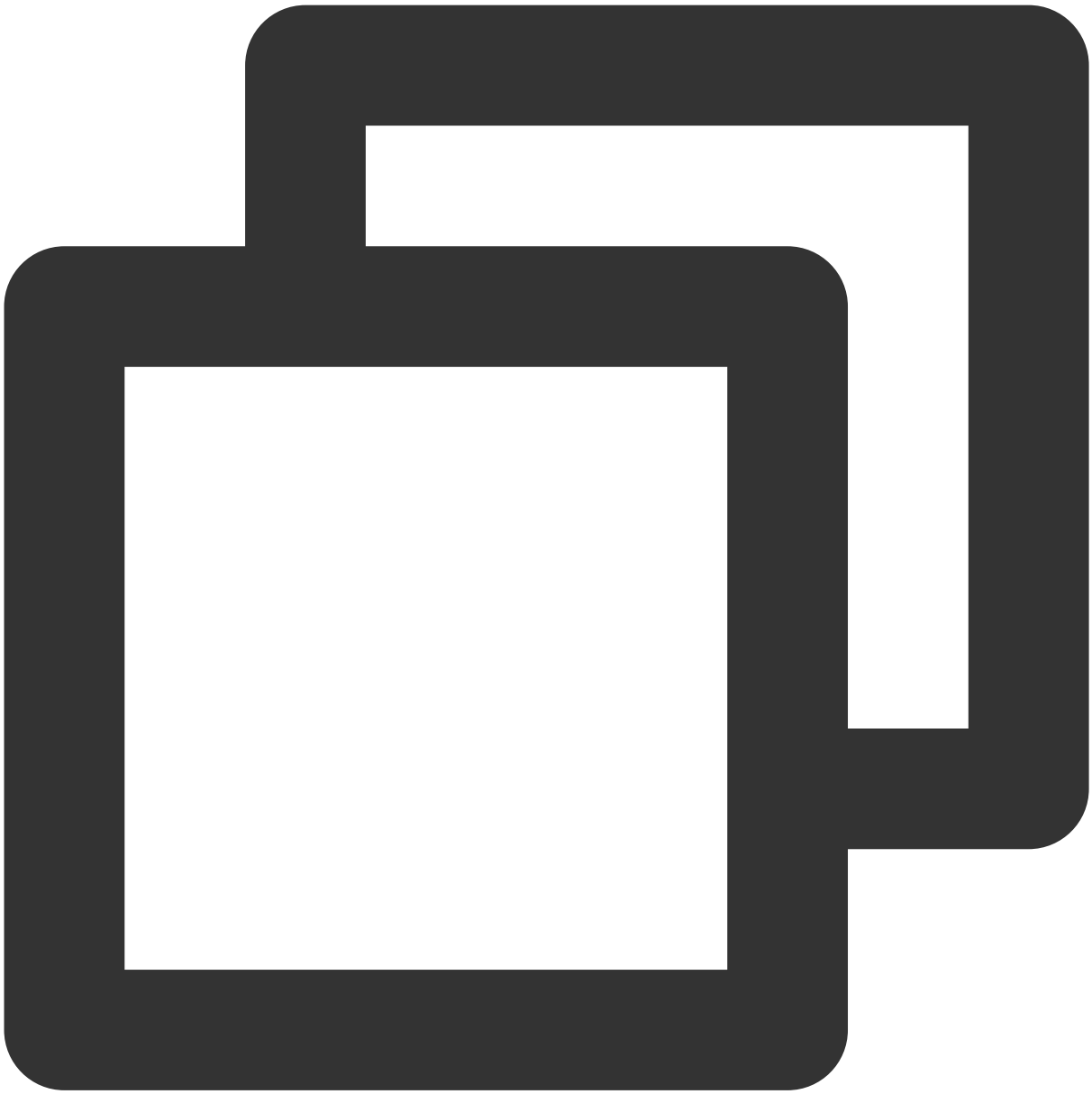
代码混淆

1. 将以下混淆规则添加在 App 项目级别的 proguard-rules.pro 文件中。



```
-ignorewarnings
-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.agconnect.**{*;}
```

2. 如应用使用了 AndResGuard 插件，请在 AndResGuard 配置白名单中添加以下内容。如果未使用 AndResGuard 插件，则请忽略该步骤。



```
whiteList = [
 "R.string.hms*",
 "R.string.connect_server_fail_prompt_toast",
 "R.string.getting_message_fail_prompt_toast",
 "R.string.no_available_network_prompt_toast",
 "R.string.third_app_*",
 "R.string.upsdk_*",
 "R.layout.hms*",
```

```
  "R.layout.upsdk_*",
  "R.drawable.upsdk*",
  "R.color.upsdk*",
  "R.dimen.upsdk*",
  "R.style.upsdk*",
  "R.string.agc*"
 ]
```

说明：

更多混淆配置请参见 华为推送配置混淆脚本。

高级配置（可选）

华为通道抵达回执配置

华为通道抵达回执需要开发者自行配置，您可参见 华为厂商通道回执配置指引 进行配置。完成后，可在推送记录中查看华为推送通道的抵达数据。

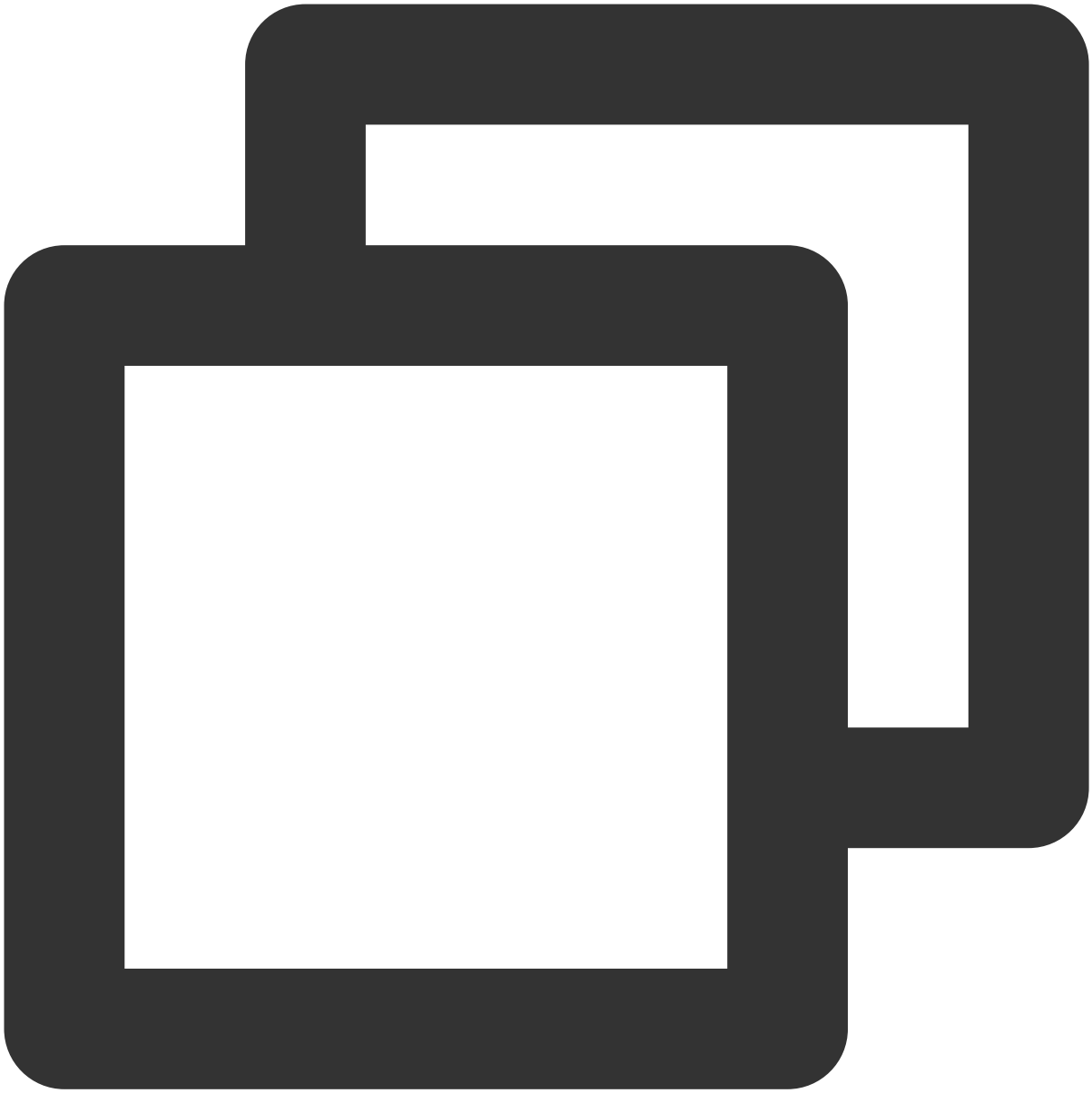
华为设备角标适配

华为设备支持设置应用角标，需要开发者申请应用内角标设置和设置应用启动类权限，详情请参见 角标适配指南 文档。

常见问题排查

华为推送注册错误码查询方法

华为推送服务接入过程配置要求较严格，若观察到如下类似日志则说明华为厂商通道注册失败，开发者可以通过以下方式获取华为推送注册错误码：



```
[OtherPushClient] handleUpdateToken other push token is :  other push type: huawei
```

推送服务 debug 模式下，过滤关键字“OtherPush”或“HMSSDK”，查看相关返回码日志（例如 `[OtherPushHuaWeiImpl] other push huawei onConnect code:907135702`），并前往 厂商通道注册失败排查指南 查找对应原因，获取解决办法。

通过华为通道下发的通知，为什么没有通知提醒？

华为推送从 EMUI 10.0版本开始将通知消息智能分成两个级别：一般与重要。EMUI 10.0之前的版本没有对通知消息进行分类，只有一个级别，消息全部通过“默认通知”渠道展示，等价于 EMUI 10.0的重要级别消息。若通知被归类

为“一般”级别，则没有震动、响铃、和状态栏图标提示，目前可通过自定义通知渠道将消息级别设为“重要”；但遵照华为推送相关规则，最终展示效果仍将与华为推送智能分类计算出的级别共同决定，两者取低，例如重要与一般取一般。详情请参见 华为消息分类使用指南。

FCM 通道接入

最近更新时间：2024-01-16 17:39:39

操作场景

FCM 通道是谷歌推出的系统级推送通道，在国外具备谷歌 Service 框架的手机上，鉴于其较宽松的后台进程管理方式，在应用进程未被强制停止的情况下，可以收到推送消息。FCM 通道不支持国内集群。

操作步骤

获取密钥

FCM 推送支持两种密钥配置，以下方式二选一，推荐使用“服务器私钥”的新协议方式。

1. 服务器私钥 （推荐）

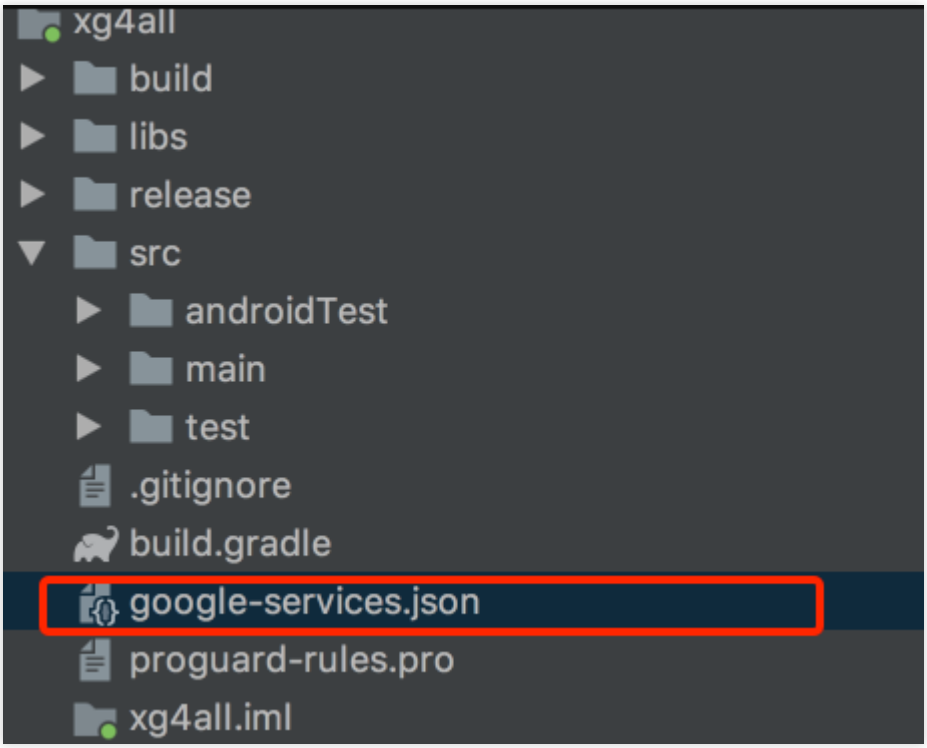
进入 FireBase 官网，注册应用信息。在 **Firebase 项目** > **选择具体的项目应用** > **设置** > **服务账号** > **Firebase Admin SDK**，单击**生成新的私钥**，获取到 Firebase 服务器私钥 json 文件。然后进入 **移动推送控制台** > **配置管理** > **基础配置** > **FCM 官方推送通道** 栏目中，选中“（推荐）服务器私钥”，单击**点击上传**，上传获取到的 json 文件。

2. 旧版服务器密钥

进入 FireBase 官网，注册应用信息。在**Firebase 项目** > **选择具体的项目应用** > **设置** > **云消息传递** 获取到的 FCM 应用推送 **服务器密钥**，并配置到 **移动推送控制台** > **配置管理** > **基础配置** > **FCM 官方推送通道** 栏目中。

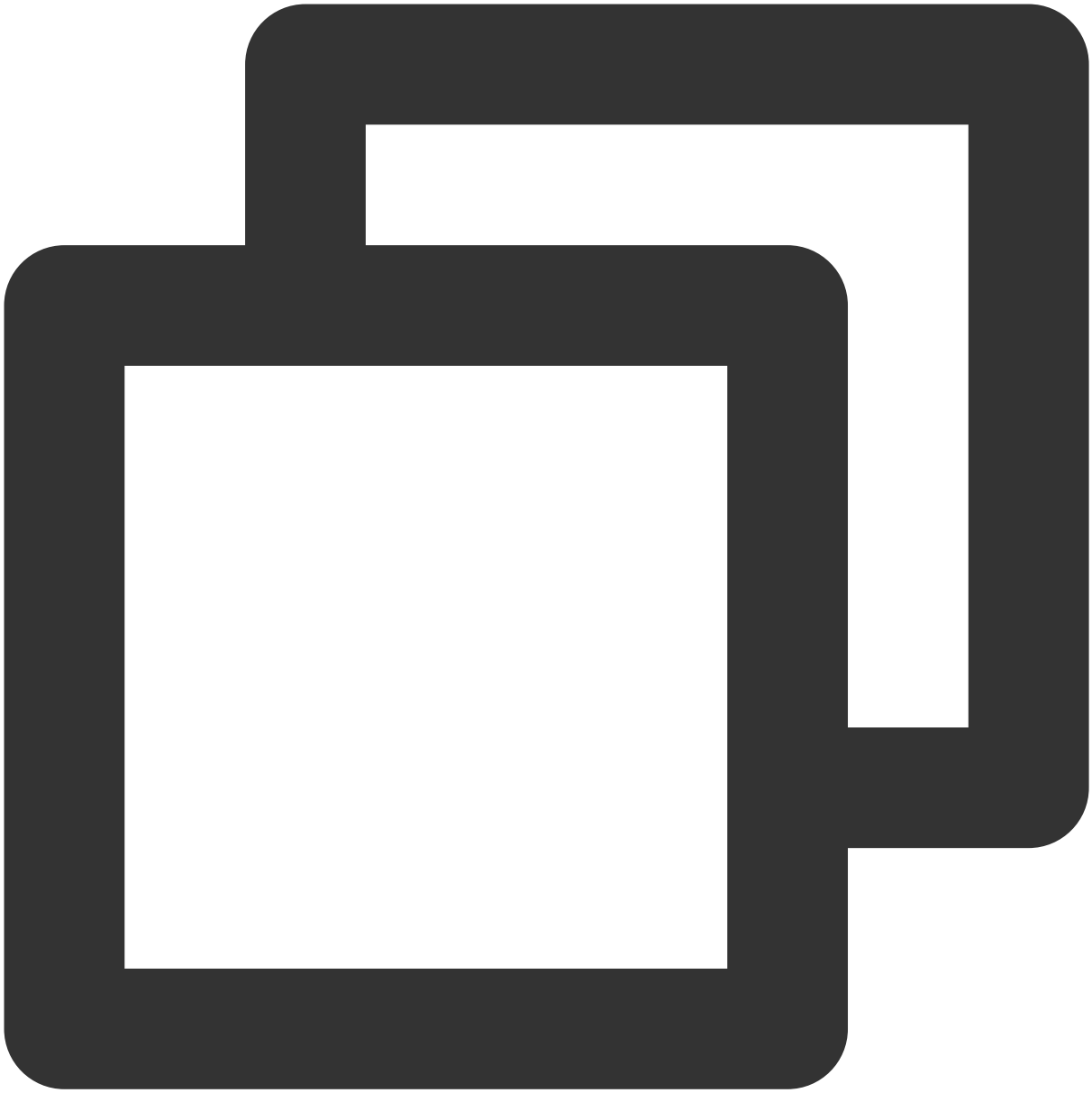
配置内容

1. 配置 google-services.json 文件。如图所示：



2. 配置 gradle，集成谷歌 service。

1. 在项目级的 build.gradle 文件中的 dependencies 节点中添加下面代码：

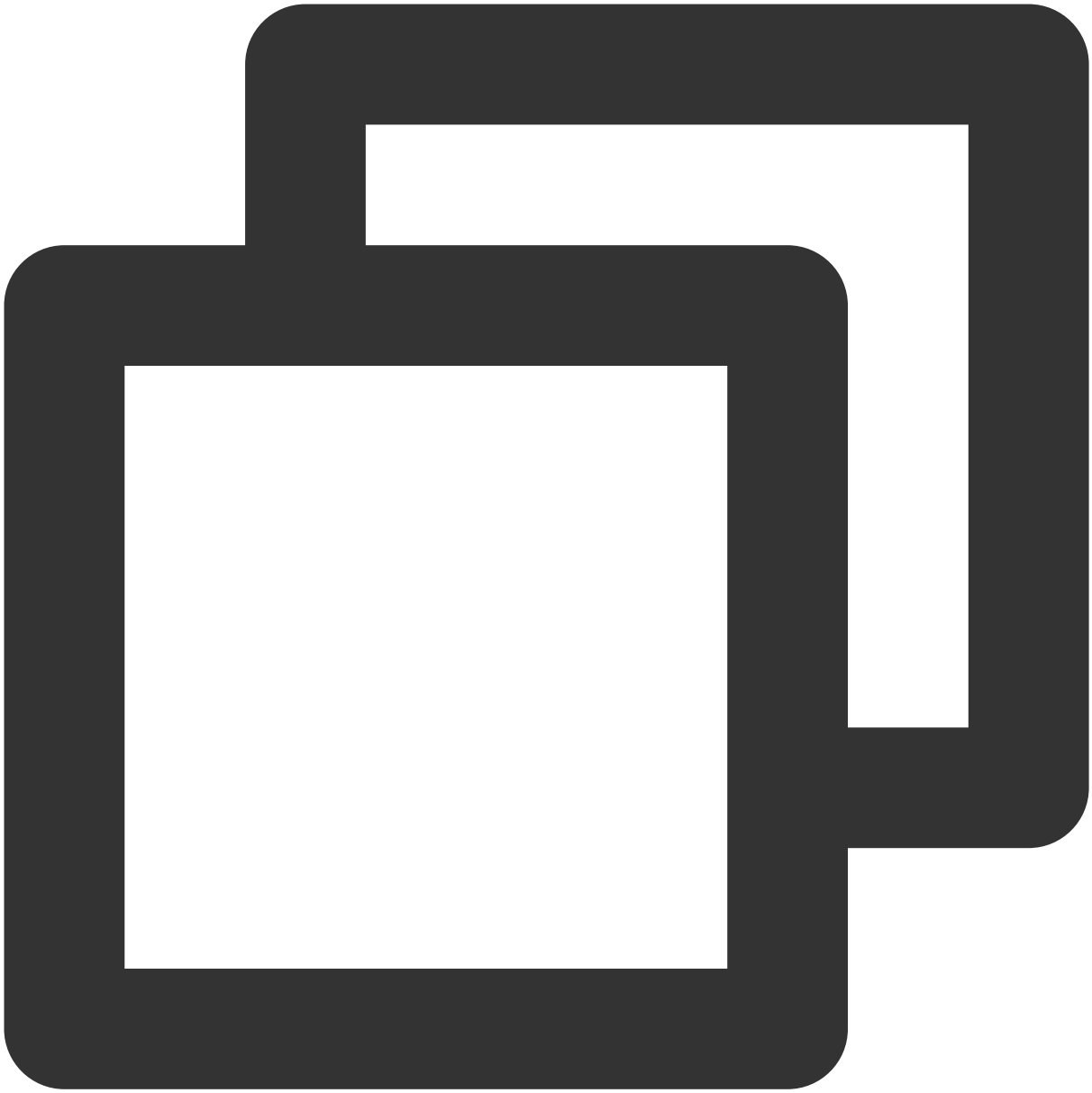


```
classpath 'com.google.gms:google-services:4.2.0'
```

注意：

如果使用低于4.2.0版本出现 `FCM Register error! java.lang.IllegalStateException: Default FirebaseApp is not initialized in this process com.qq.xg4all. Make sure to call FirebaseApp.initializeApp(Context) first.` ，建议在 res/values 文件夹下的 string.xml， 加上 YOUR_GOOGLE_APP_ID。

2. 在应用级的 build.gradle 文件中，添加依赖：



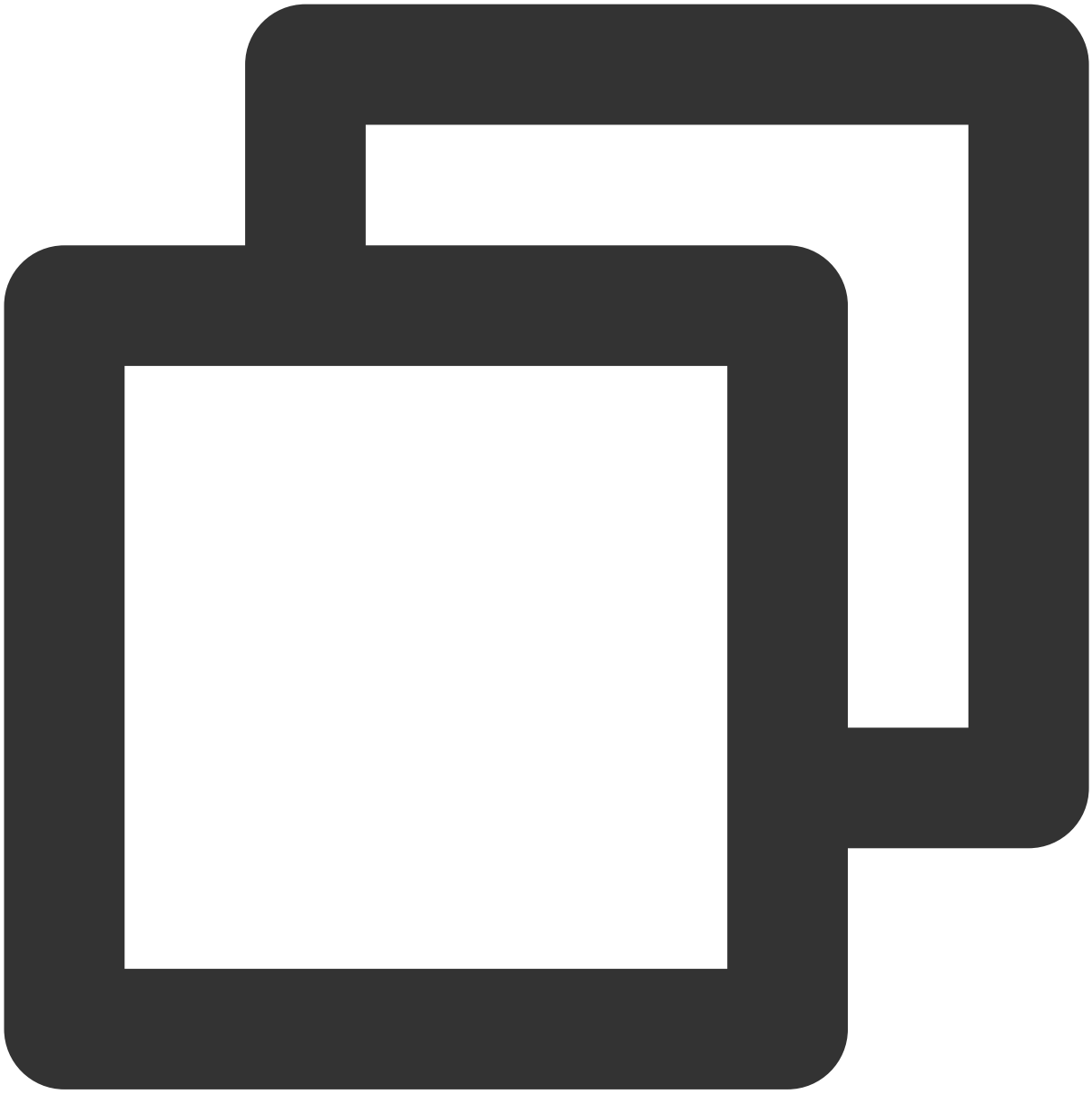
```
        implementation 'com.tencent.tpns:fcm:[VERSION]-release' // FCM 推送 [VERSION
 implementation  'com.google.firebase:firebase-messaging:17.6.0'

//在应用级的 gradle 文件的最后一行代码中新增并将 google-services.json 放进您应用 model 的根
apply plugin: 'com.google.gms.google-services'
```

注意：

1. FCM 推送 [VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态 查看。
2. Google 配置 google-play-services（建议版本 17.0.0+，较低版本有可能出现无法注册 FCM 风险）。

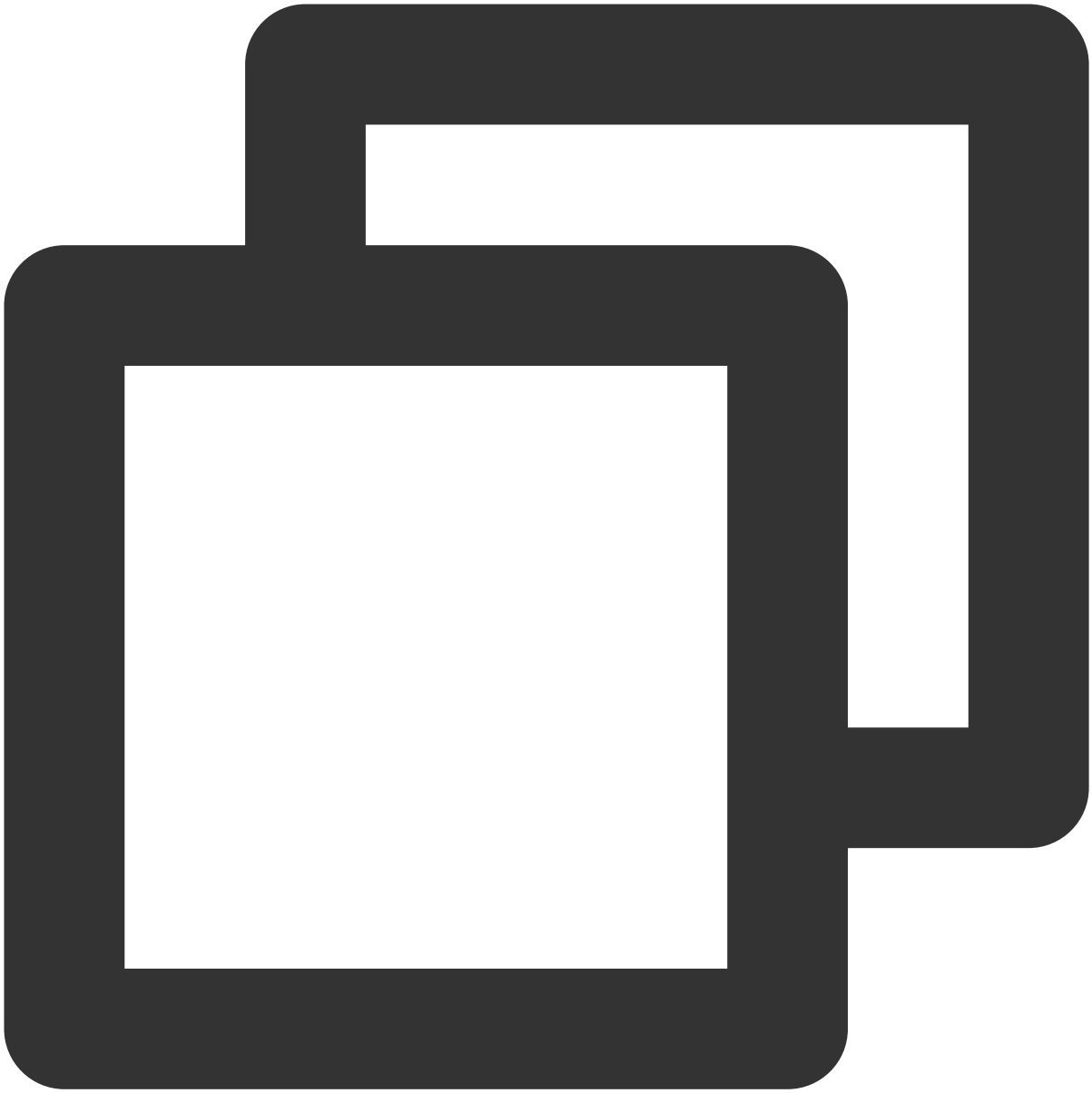
启用 FCM 推送

在调用移动推送注册代码 XGPushManager.registerPush 前，添加以下代码设置：



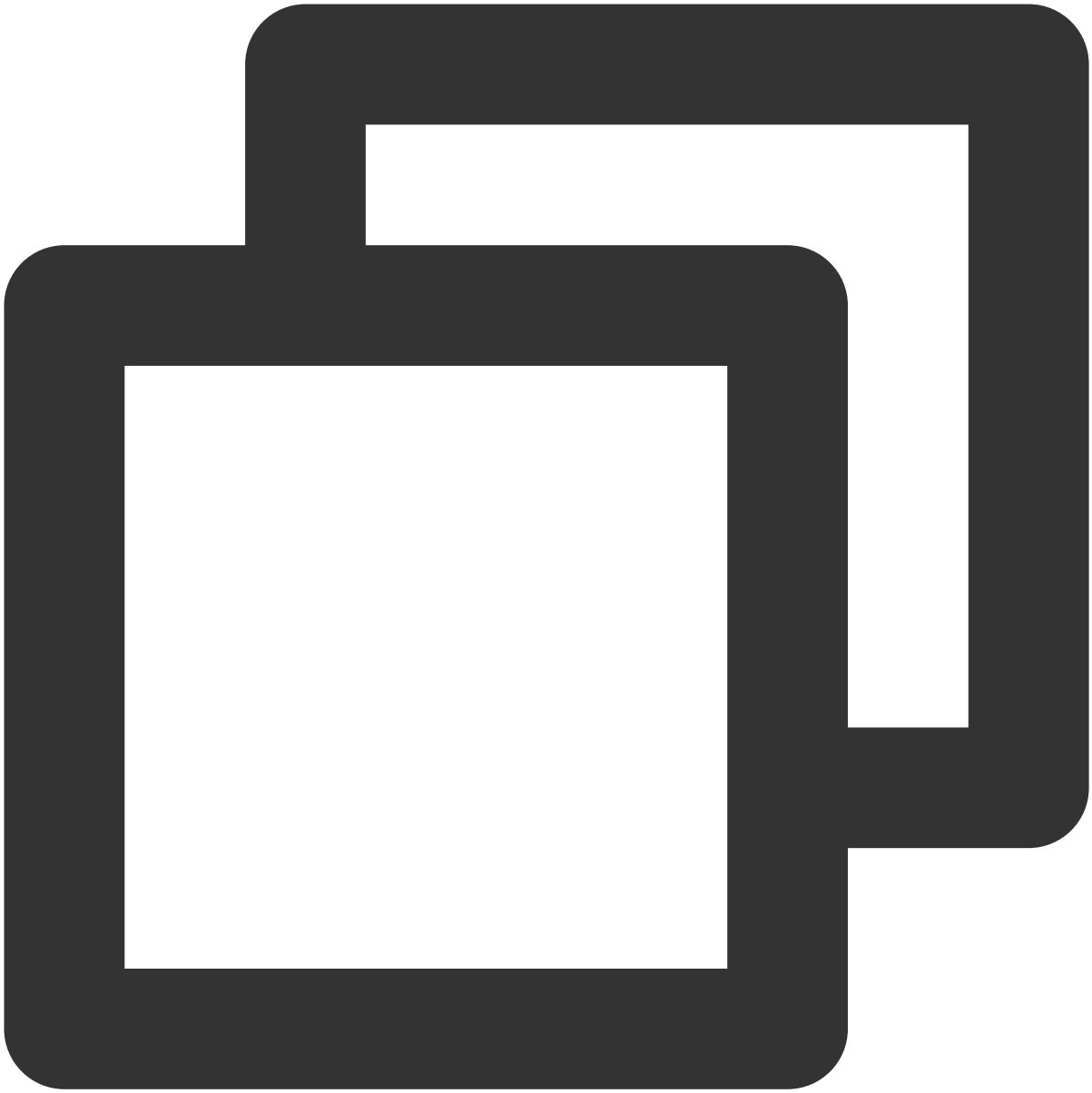
```
XGPushConfig.enableOtherPush(this, true);
```

注册 FCM 成功的日志如下：



```
V/TPush: [XGPushConfig] isUsedOtherPush:true
I/TPush: [OtherPush] checkDevice pushClassNamecom.tencent.android.tpush.otherpush.f
I/TPush: [XGPushManager] other push token is : dSJA5n4fSZ27YeDf2rFg1A:APA91bGiqSPCM
```

代码混淆



```
-keep class com.google.firebase.** {*;}
```

说明：

混淆规则需要放在 App 项目级别的 proguard-rules.pro 文件中。

常见问题排查

推送 FCM 推送收不到，是什么原因？

1. 在境外具备谷歌 Service 框架的手机上，鉴于其较宽松的后台进程管理方式，在应用进程未被强制停止的情况下，FCM 消息抵达较为稳定。
2. 在大陆发行的国内品牌手机，其后台进程管理普遍较为严格，谷歌 service 后台服务同样也会受到限制，这些手机上 FCM 消息抵达可能会受到影响，FCM 无法进行下发和接收，建议保持 App 在前台接收。

什么是强制停止应用进程？

在手机设置-应用管理-具体应用-点击“结束运行”/“强制停止”等按钮停止了应用。大部分国内品牌手机，在多任务页面划掉应用进程，也可认为是强制停止了应用进程（境外手机不会）。

OPPO 通道接入

最近更新时间：2024-01-16 17:39:39

操作场景

OPPO 通道是由 OPPO 官方提供的系统级推送通道。在 OPPO 手机上，推送消息能够通过 OPPO 的系统通道抵达终端，无需打开应用就能够收到推送。详情请参见 OPPO 推送官网。

说明：

OPPO 通道暂不支持应用内消息的发送，此类型的消息会通过移动推送自建通道进行下发。

OPPO 通道对应用的每日推送量有额度限制，详情请参见 厂商通道限额说明，超过限制部分将走移动推送自建通道进行补推发送。

OPPO 通道需要 OPPO 手机系统 ColorOS V3.1 及以上支持。

操作步骤

开通权限

使用 OPPO 企业开发者帐号，登录 OPPO 开发平台，在**管理中心** > **应用服务平台** > **移动应用列表** > **选择应用** > **开发服务** > **推送服务** 中完成 OPPO PUSH 权限申请。

说明：

通知栏推送权限申请需要应用在 OPPO 软件商店上架才可通过，且主营业务不为借贷类的应用。

获取密钥

说明：

仅开发者帐号（主帐号）可查看。

1. Opush 申请开通成功后，您可在 **OPPO 推送平台** > **配置管理** > **应用配置页面**，查看 AppKey、AppSecret 和 MasterSecret。

2. 复制应用的 AppKey、AppSecret 和 MasterSecret 参数填入 **移动推送控制台** > **配置管理** > **基础配置** > **OPPO 官方推送通道** 栏目中。

配置推送通道

为兼容安卓8.0及以上版本的 OPPO 手机的通道配置，用户需在 OPPO 管理台上，创建一个移动推送推送的默认通道。详情请参见 OPPO 官方文档。

具体内容为：

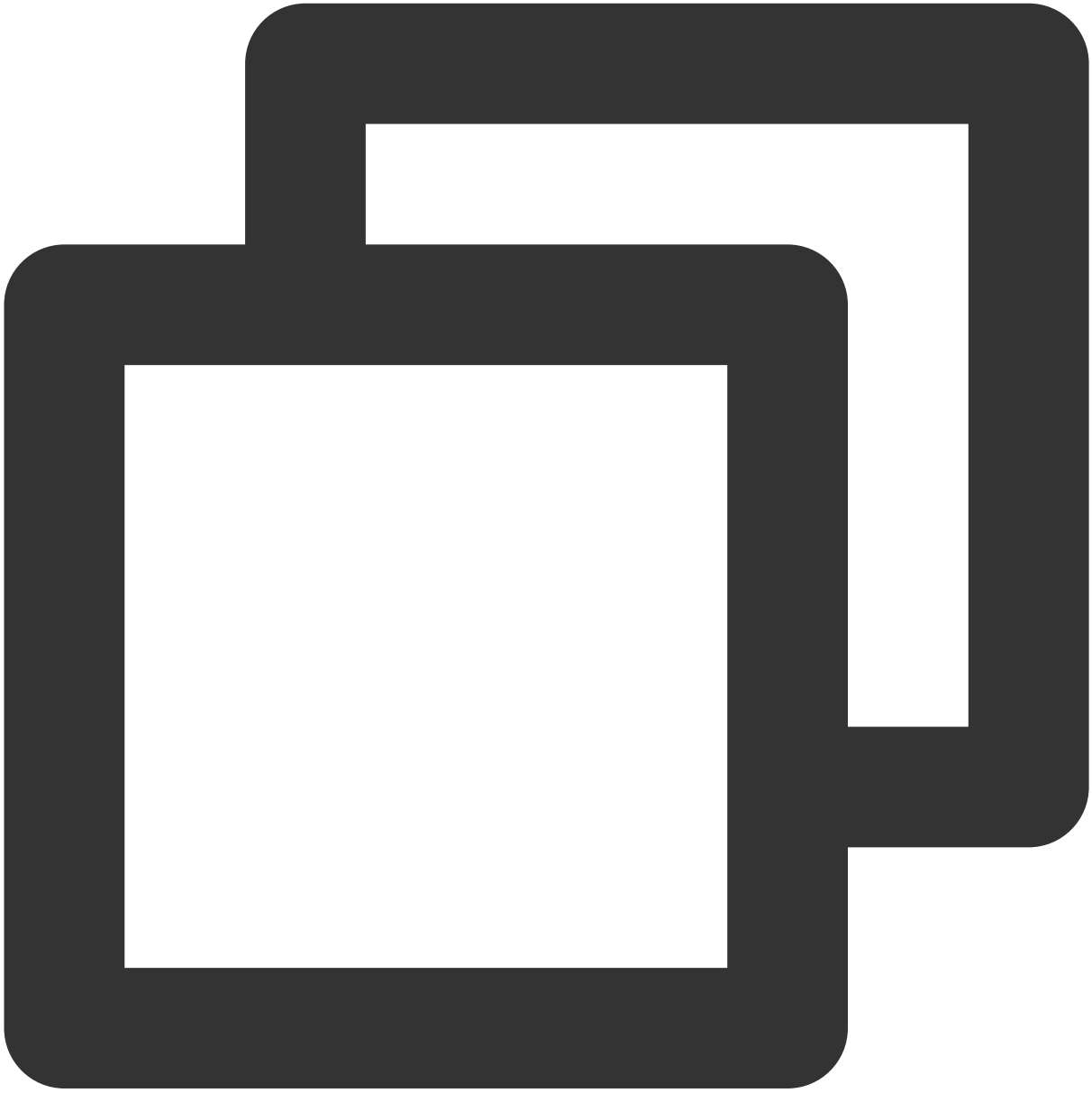
“通道 ID”：“default_message”

“通道名称”：“默认通知”

配置内容

AndroidStudio 集成方法

导入 OPPO 推送相关依赖。示例代码如下：



```
//  OPPO 推送 SDK，[VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态查看
implementation 'com.tencent.tpns:oppo:[VERSION]-release'

// 自 SDK 1.3.2.0 起，需一并加入以下依赖语句，否则可能导致 OPPO 推送注册失败
implementation 'com.google.code.gson:gson:2.6.2'
implementation 'commons-codec:commons-codec:1.15'
```

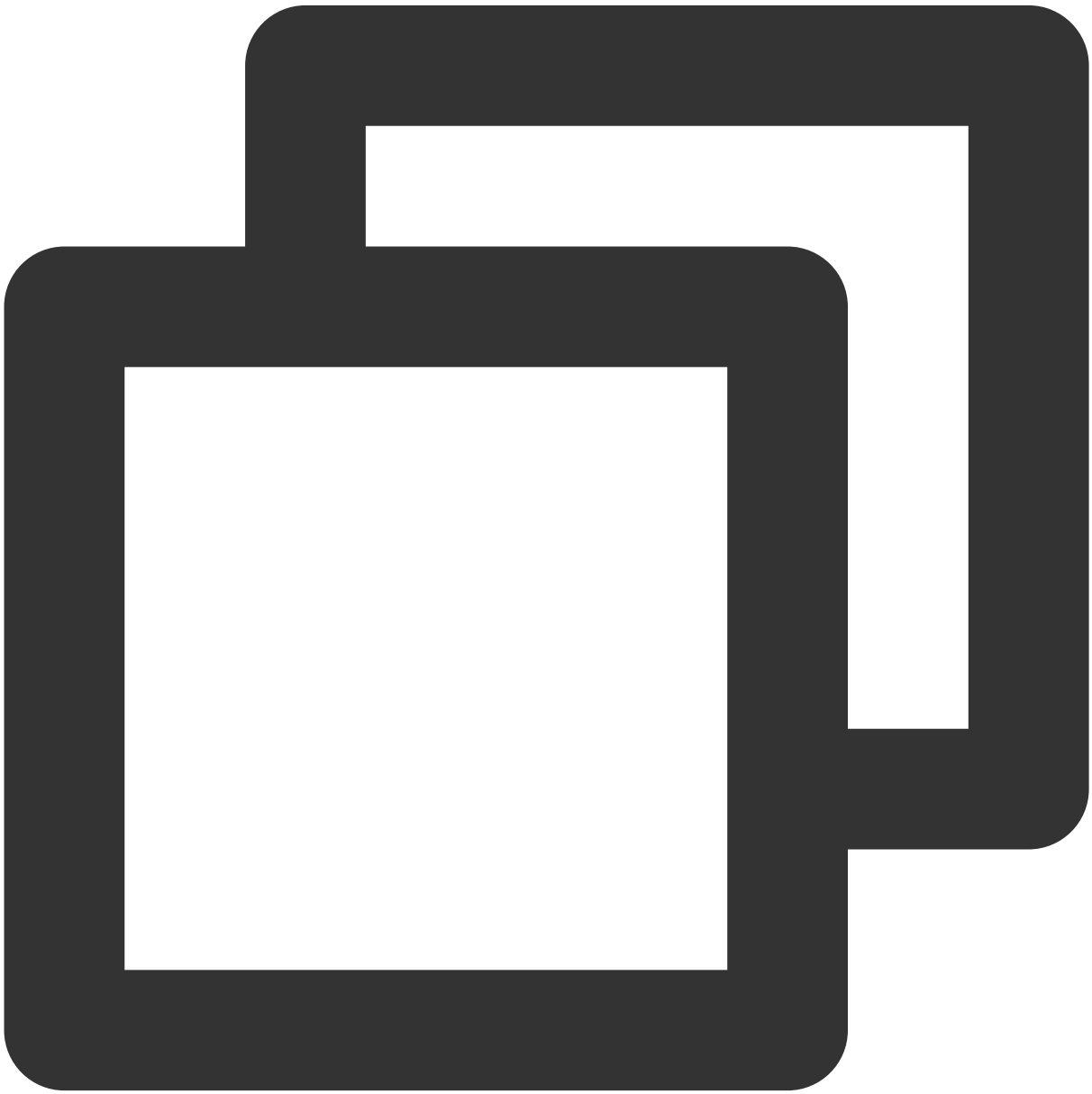
说明：

OPPO 推送 [VERSION] 为当前 SDK 版本号，版本号可在 Android SDK 发布动态 查看。

Eclipes 集成方法

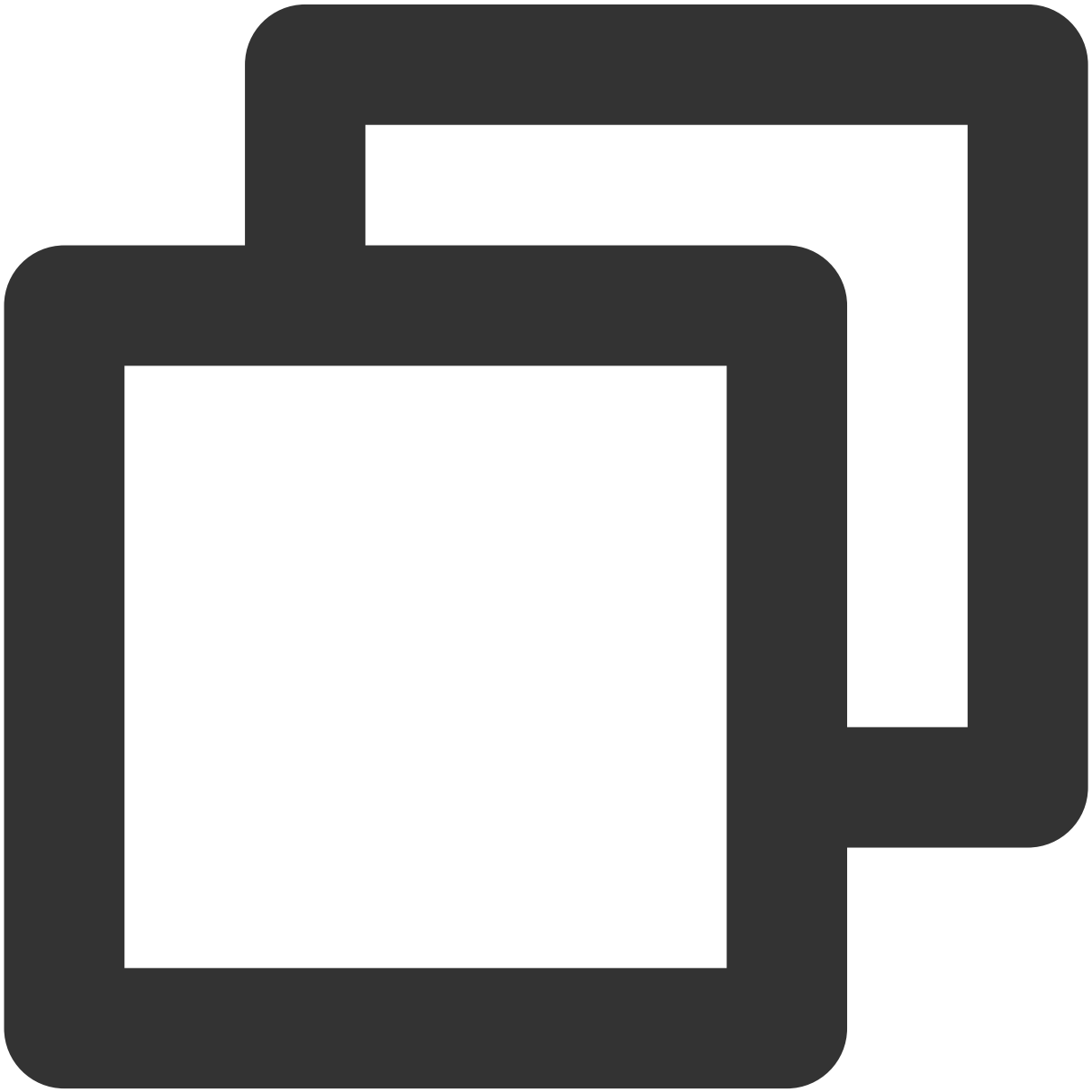
获取移动推送 OPPO 通道 SDK 包后，按照移动推送官网手动集成方法，在配置好移动推送主版本的基础下，进行以下设置。

1. 打开 Other-push-jar 文件夹，将 OPPO 推送相关 jar 导入项目工程中。
2. 在主工程添加类资源文件，代码如下：



```
package com.pushsdk;

class R {
 public static final class string {
 public final static int system_default_channel = com.tencent.android.tpns.demo.R.s
 }
}
```

3. 在 `Androidmanifest.xml` 文件中新增如下配置：

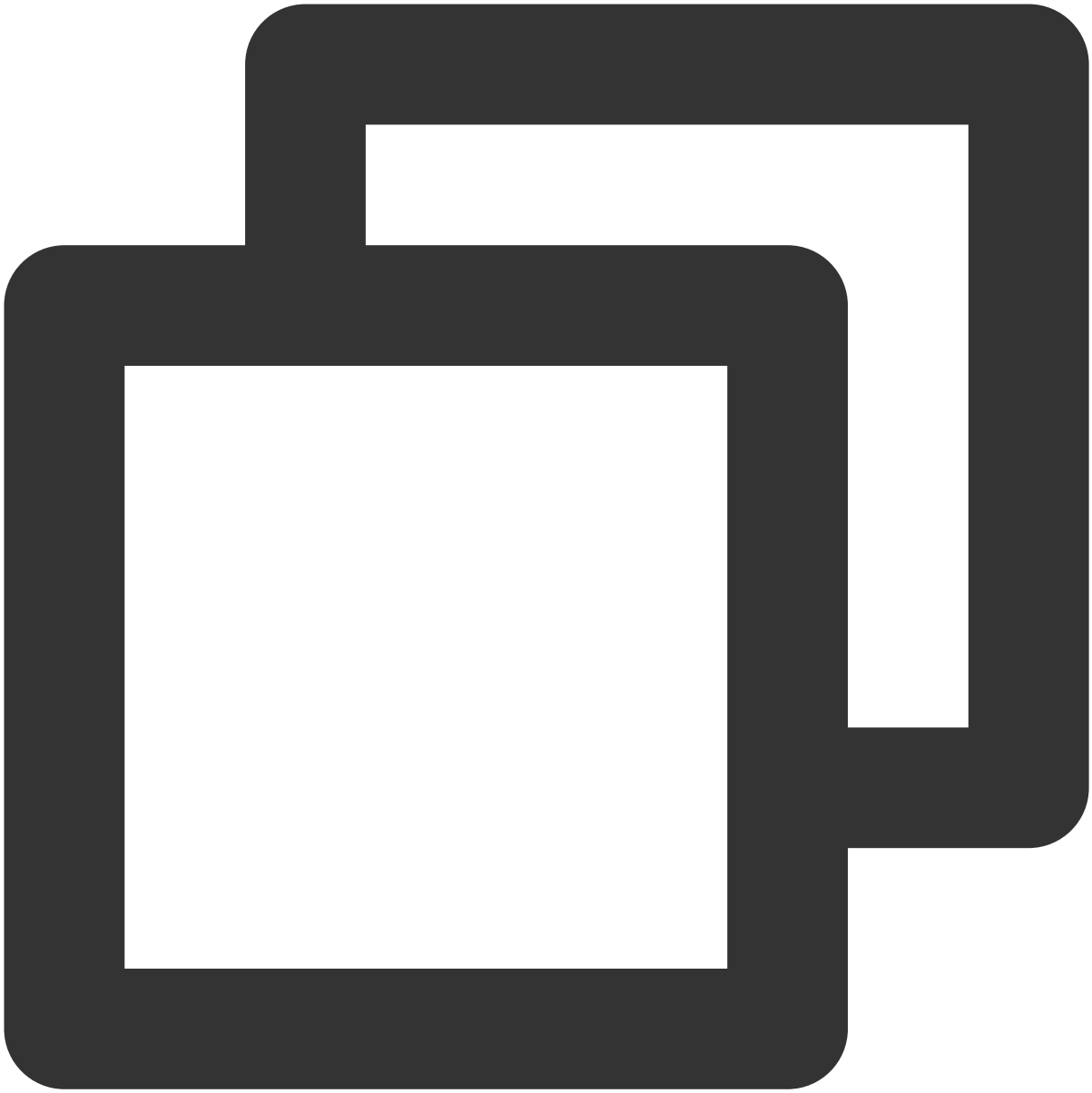


```
<!--OPPO 推送服务必须权限-->
```

```
<uses-permission android:name="com.coloros.mcs.permission.RECIEVE_MCS_MESSAGE"/>
<uses-permission android:name="com.heytap.mcs.permission.RECIEVE_MCS_MESSAGE"/>
<application>
        <service
            android:name="com.heytap.msp.push.service.CompatibleDataMessageCallback
            android:permission="com.coloros.mcs.permission.SEND_MCS_MESSAGE"
            android:exported="true">
            <intent-filter>
            <action android:name="com.coloros.mcs.action.RECEIVE_MCS_MESSAGE" />
            </intent-filter>
        </service>

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

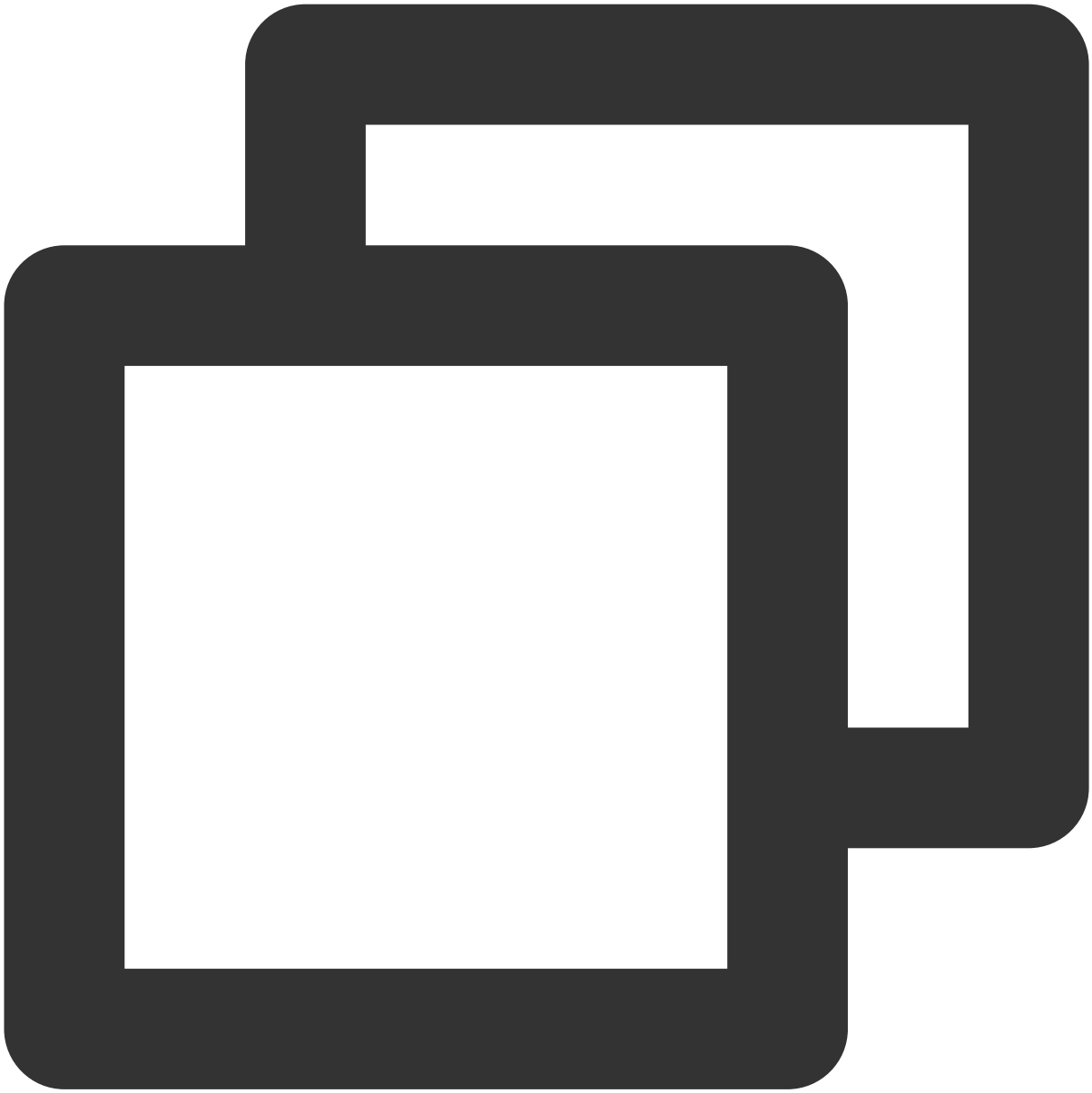
开启 OPPO 推送

在调用移动推送 `XGPushManager.registerPush` 之前，调用以下代码：



```
// 注意这里填入的是 Oppo 的 AppKey，不是AppId
XGPushConfig.setOppoPushAppId(getApplicationContext(), "Oppo的AppKey");
// 注意这里填入的是 Oppo 的 AppSecret，不是 AppKey
XGPushConfig.setOppoPushAppKey(getApplicationContext(), "Oppo的AppSecret");
//打开第三方推送
XGPushConfig.enableOtherPush(getApplicationContext(), true);

//注册成功的日志如下
I/TPush: [RegisterReservedInfo] Reservert info: other push token is : CN_fc0f0b3822
I/TPush: [PushServiceBroadcastHandler] >> bind OtherPushToken success ack with [acc
```

代码混淆



```
-keep public class * extends android.app.Service
-keep class com.heytap.mcssdk.** {*;}
-keep class com.heytap.msp.push.** { *;}
```

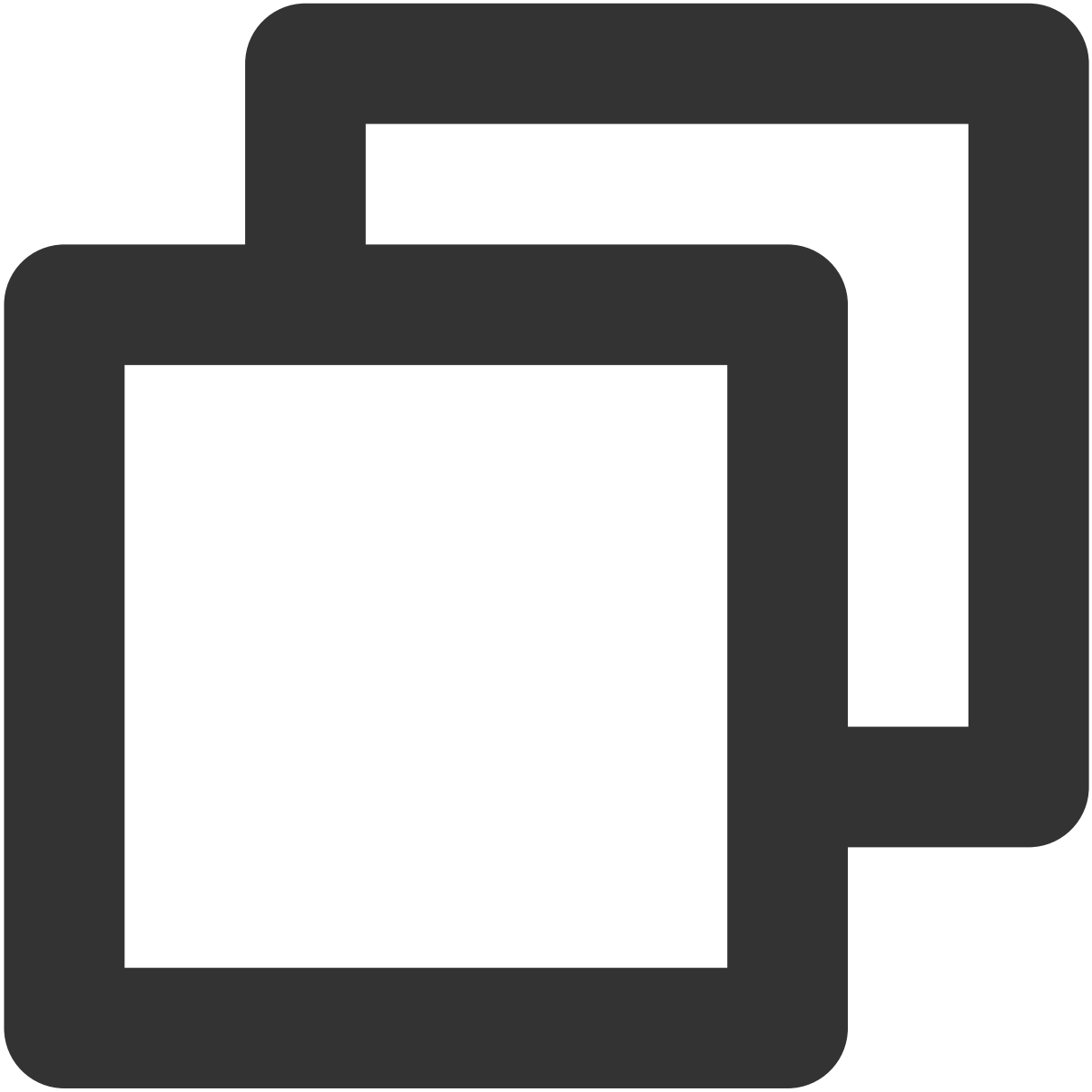
说明：

混淆规则需要放在 App 项目级别的 proguard-rules.pro 文件中。

常见问题排查

OPPO 推送注册错误码查询方法

若观察到如下类似日志则说明 OPPO 厂商通道注册失败，开发者可以通过以下方式获取 OPPO 推送注册错误码：



```
[OtherPushClient] handleUpdateToken other push token is :  other push type: OPPO
```

推送服务 debug 模式下，过滤关键字 “OtherPush”，查看相关返回码日志（例如 `[OtherPushOppoImpl] OppoPush Register failed, code=14, msg=INVALID_APP_KEY` ），并前往 厂商通道注册失败排查指南

查找对应原因，获取解决办法。

推送 OPPO 响应失败 code:30，是什么原因？

应用审核中不可发送正式消息，请前往 OPPO 平台确认推送权限审核进度。

厂商通道高级功能
角标适配指南

最近更新时间：2024-01-16 17:39:39

Android 阵营各厂商机型角标开放能力不同，移动推送对推送角标的支持程度做以下说明，供开发者参考使用。

概览

厂商	是否支持角标/红点显示	是否需要配置	适配说明
华为/荣耀	支持角标	是	请参考下文 华为手机角标适配说明
小米	支持角标	否	遵从系统默认逻辑，感应通知栏通知数目，按 1 自动增减
魅族	支持红点	否	遵从系统默认逻辑，仅支持红点展示，有通知则展示，无则不展示
OPPO	支持红点	否	圆点展示需由用户在通知设置中手动开启，遵从系统默认逻辑，有通知则展示，无则不展示； 数值展示只对指定应用开启，例如 QQ、微信，需向官方进行权限申请，暂无明确适配说明
vivo	支持角标	是	请参考下文 vivo 手机角标适配说明

服务端下发角标设置

您可以通过移动推送控制台或 Push API 设置服务端下发角标：

方式1：通过控制台推送页面设置

方式2：通过 Push API 设置

1. 登录 移动推送控制台。
2. 找到您需要配置的 Android 产品，在其右侧【操作】项下单击【推送管理】，进入推送管理页面。

3. 单击您需要配置的推送，进入推送配置页面。
4. 在【高级设置】配置项中，开启角标数字。

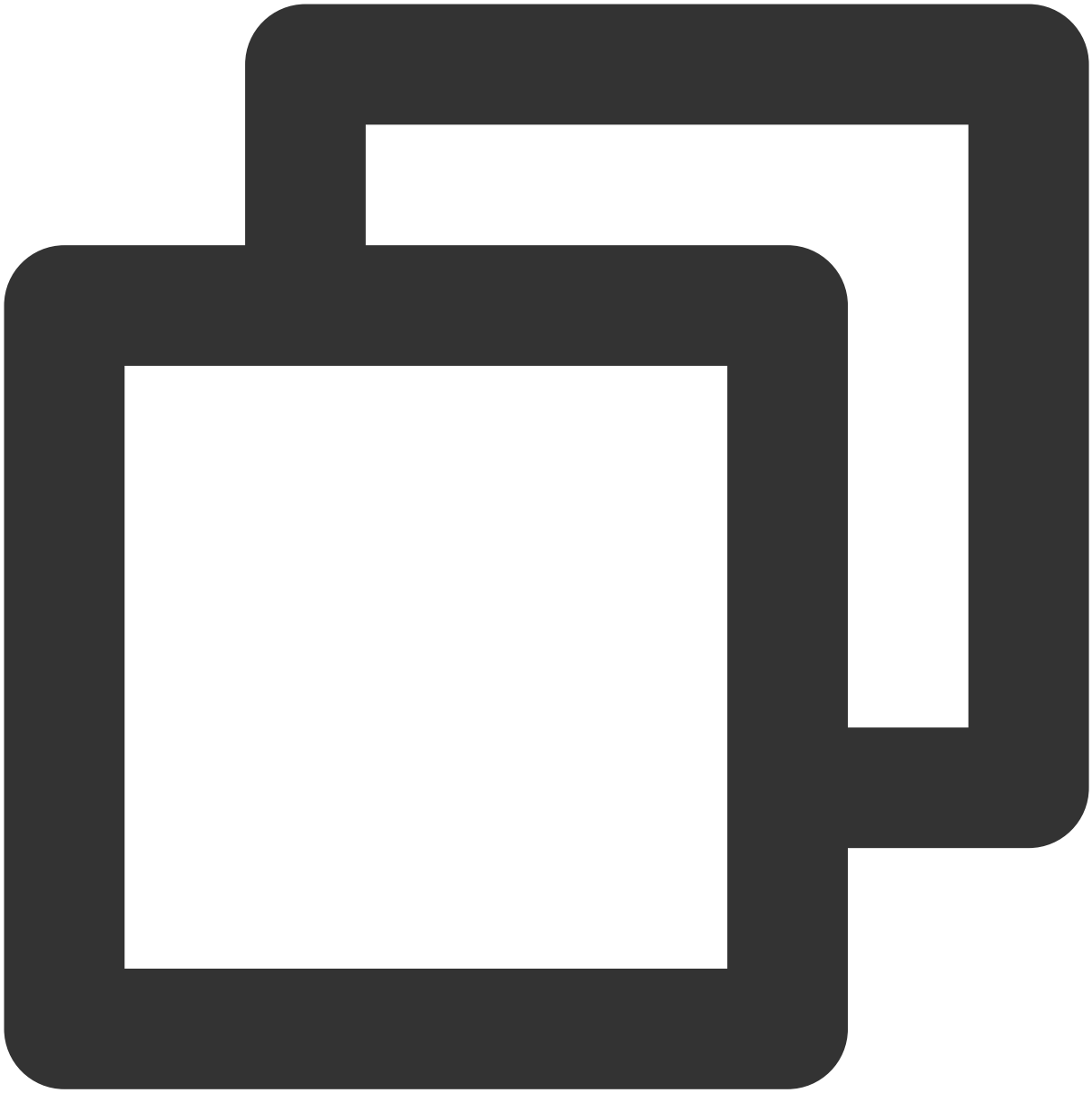
在推送消息体 `body.message.android` 下添加字段 "badge_type"，属性如下：

参数名	类型	父项目	是否必需	默认值	描述
badge_type	int	android	否	-1	通知角标： -2：自动增加1，支持华为设备 -1：不变，支持华为、vivo 设备 [0, 100)：直接设置，支持华为、vivo 设备

说明：

不同厂商设备的角标适配能力不同，详情参考下方各厂商的角标适配说明。

消息体示例：



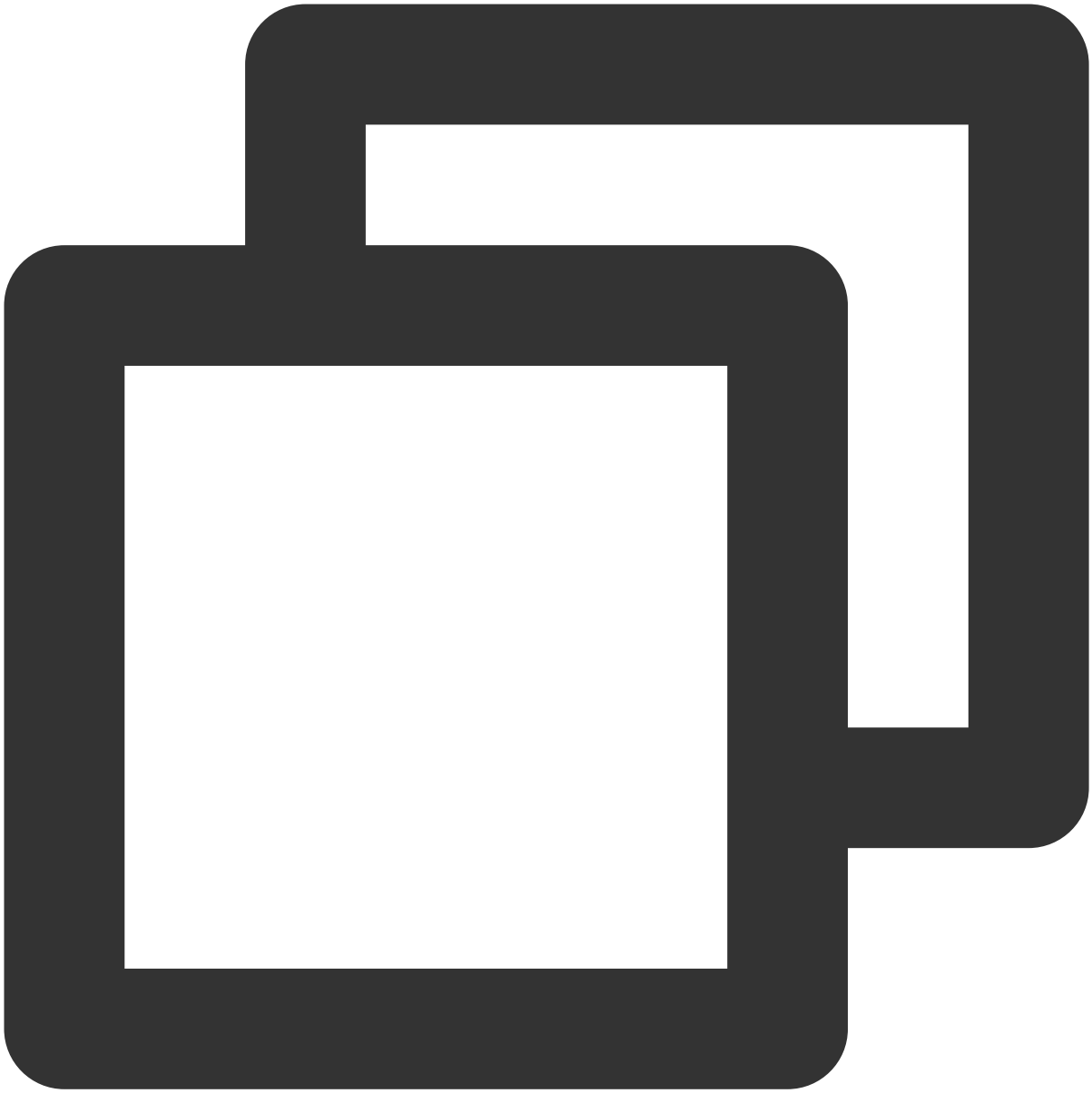
```
{
    "audience_type": "token",
    "expire_time": 3600,
    "message_type": "notify",
    "message": {
            "android": {
                    "badge_type": -2,
                    "clearable": 1,
                    "ring": 1,
                    "ring_raw": "xtcallmusic",
                    "vibrate": 1,
```

```
                    "lights": 1,
                    "action": {
                            "action_type": 1,
                            "activity": "com.qq.xg4all.JumpActivity",
                            "aty_attr": {
                            "if": 0,
                            "pf": 0
                    }
                    }
            },
            "title": "android test",
            "content": "android test 21"
    },
    "token_list": [
        "01f6ac091755a79015b4a30c9c4c7ddba1ea"
    ],
    "multi_pkg": true,
    "platform": "android",
}
```

终端通用 API

直接设置角标数值（SDK v1.2.0.1 起）

直接设置角标数值；当前支持华为、OPPO、vivo 手机角标展示，其中 OPPO 需另外向厂商申请角标展示权限。

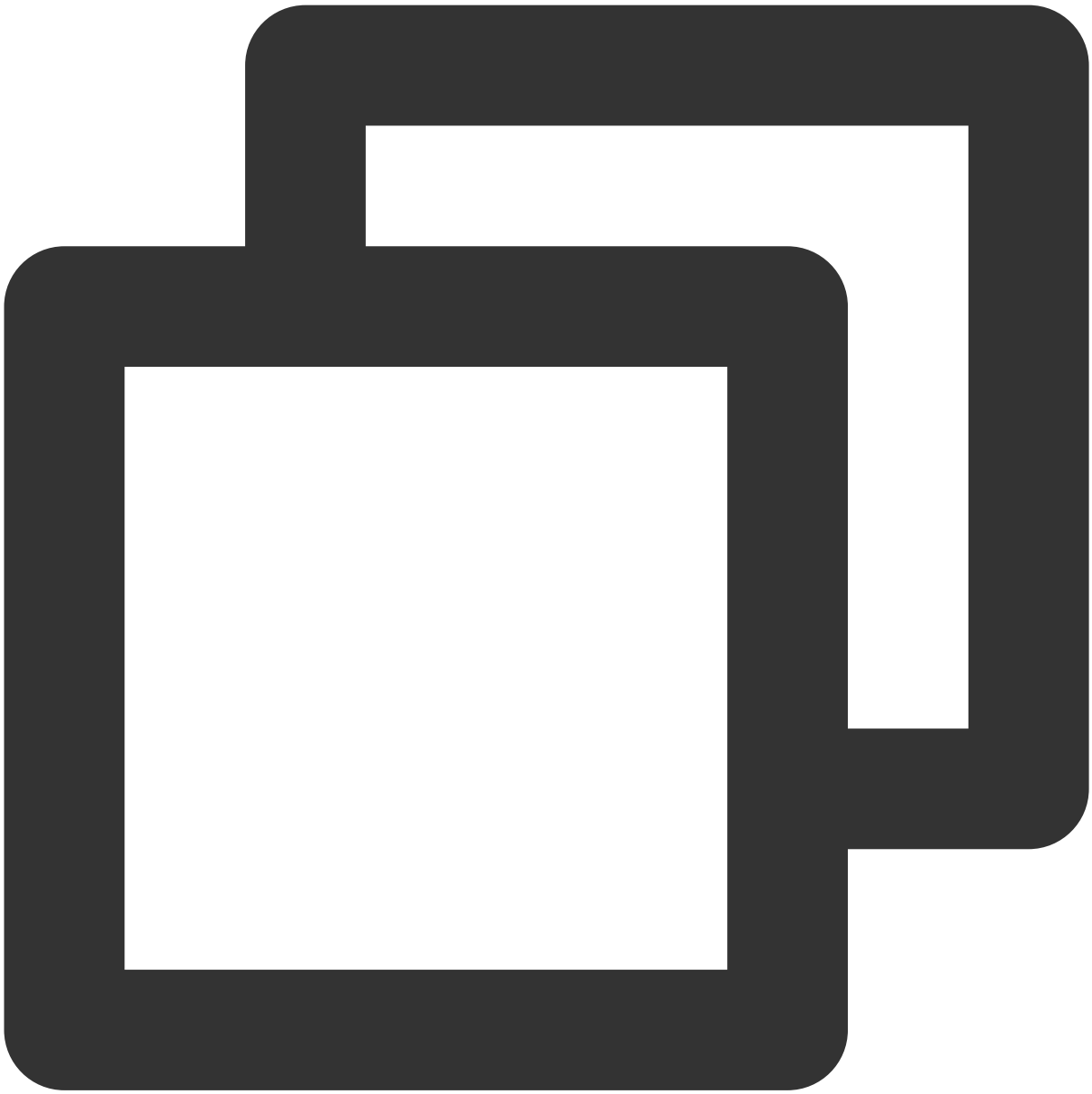


```
/**
 * @param context 应用上下文
 * @param setNum  修改角标值
 * @since v1.2.0.1
 */
XGPushConfig.setBadgeNum(Context context, int setNum);
```

示例：在收到透传消息时，调用 `XGPushConfig.setBadgeNum(context, 8)` 设置角标数值为 8。

清除角标数值（SDK v1.2.0.1 起）

设置手机应用角标归零；当前支持华为、OPPO、vivo 手机角标展示，其中 OPPO 需另外向厂商申请角标展示权限。



```
/**
 * @param context 应用上下文
 * @since v1.2.0.1
 */
XGPushConfig.resetBadgeNum(Context context);
```

示例：在透传消息已阅读或打开应用时，调用 `XGPushConfig.resetBadgeNum(context)` 清除角标数值。

注意：

因厂商通道抵达的通知不支持通知清除时角标数值自动减1，建议您在恰当时机调用此接口来清除角标数值，如重新从桌面打开应用时。

华为手机角标适配说明

使用限制

华为手机角标展示支持 EMUI 8.0 及以上手机。

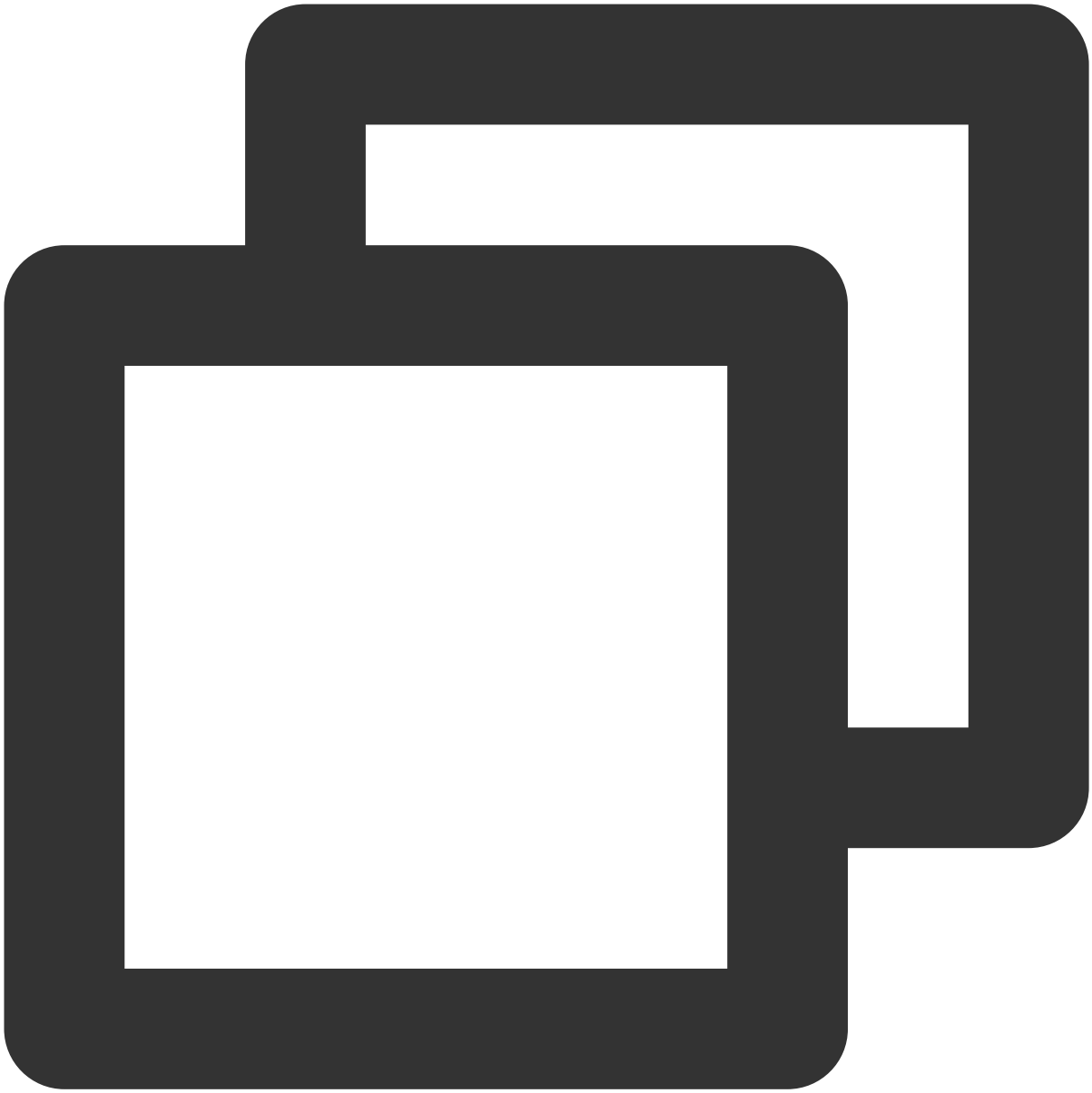
受限于华为手机角标能力的开放程度，在不同的推送场景下角标功能有所不同，详见下表。请按照指导方式使用华为手机角标功能。

推送形式	角标能力	实现方式
华为通道通知	支持角标自动加1、直接设置或不变，支持通知点击的自动减1，不支持通知清除的自动减1	通过管理台或 Push API 关键字设置
移动推送自建通道通知	支持角标自动加1、直接设置或不变，支持通知点击/清除的自动减1	通过管理台或 Push API 关键字设置
透传消息	开发者自行处理设置、加减逻辑	调用移动推送SDK 开放接口

配置内容

应用内角标设置权限申请

为能实现角标修改的正确效果，请首先为应用添加华为手机上的角标读写权限，具体实现为在应用 AndroidManifest.xml 文件的 manifest 标签下添加以下权限配置：



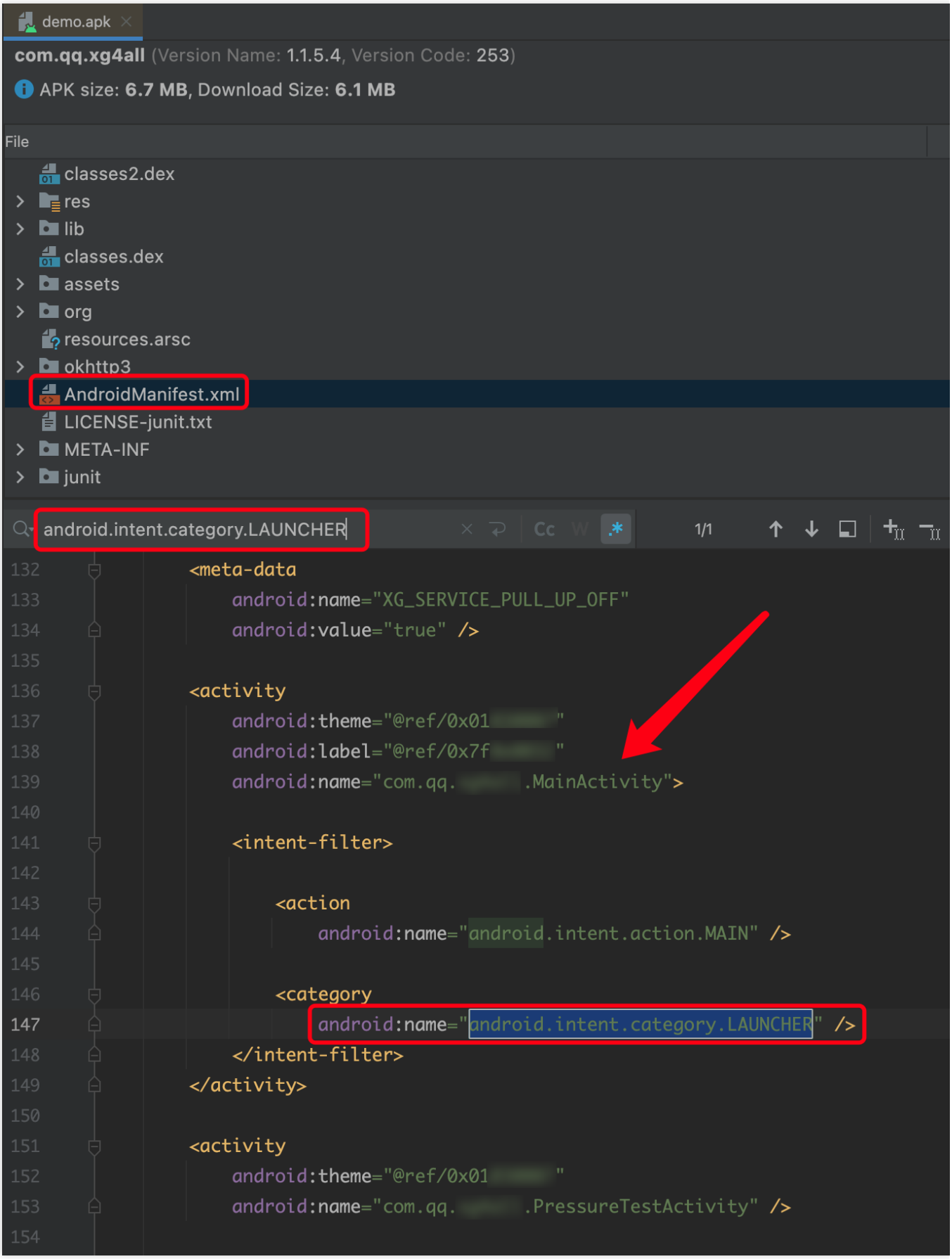
```
<uses-permission android:name="com.huawei.android.launcher.permission.CHANGE_BADGE"
<!-- 兼容荣耀手机 -->
<uses-permission android:name="com.hihonor.android.launcher.permission.CHANGE_BADGE
```

通知下发角标设置

请一定先在管理台华为通道开启及参数配置处填写桌面图标对应的应用入口 Activity 类，如“com.test.badge.MainActivity”，否则华为通道下发通知的角标设置将不生效。

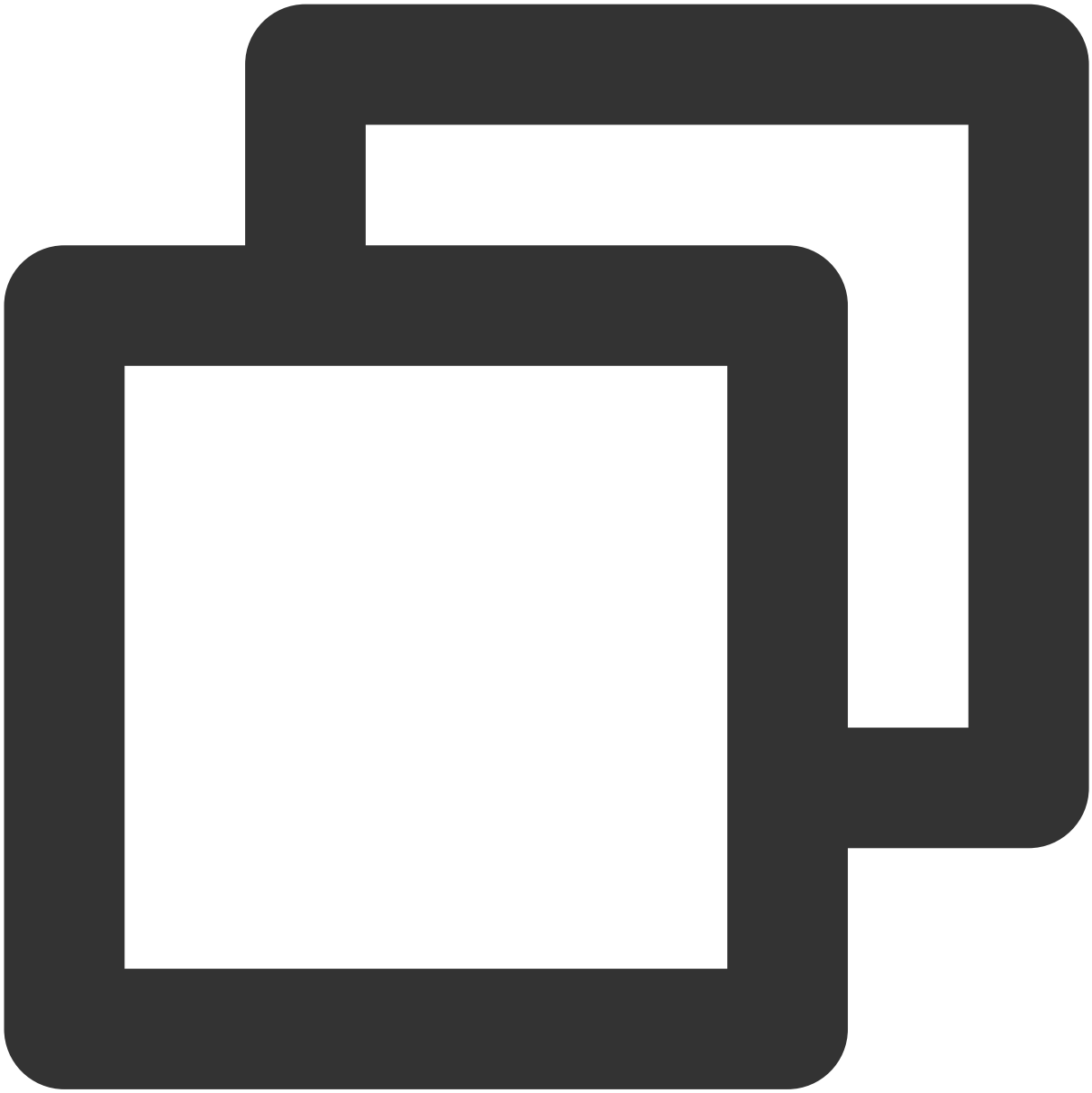
启动类名称获取指引：

请将打包完成的 apk 文件拖入 AndroidStudio，进入其中的 AndroidManifest 文件，搜索关键字“android.intent.category.LAUNCHER”，所属的 activity.name 属性即为启动类名称。


demo.apk
com.qq.xg4all (Version Name: 1.1.5.4, Version Code: 253)
APK size: 6.7 MB, Download Size: 6.1 MB
File
classes2.dex
res
lib
classes.dex
assets
org
resources.arsc
okhttp3
AndroidManifest.xml
LICENSE-junit.txt
META-INF
junit
android.intent.category.LAUNCHER
1/1
<meta-data
android:name="XG_SERVICE_PULL_UP_OFF"
android:value="true" />
<activity
android:theme="@ref/0x01
android:label="@ref/0x7f
android:name="com.qq. .MainActivity">
<intent-filter>
<action
android:name="android.intent.action.MAIN" />
<category
android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<activity
android:theme="@ref/0x01
android:name="com.qq. .PressureTestActivity" />
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154

华为手机终端设置角标自增减

华为手机支持角标自动增减1，接口如下：



```
/**
 * 华为手机角标修改接口
 *
 * @param context    应用上下文
 * @param changeNum 改变的数字，修改效果为累加；例如先前角标为5，入参为1，则角标被设置为6。
 *           当前支持 1：角标加1；-1：角标-1
 */
XGPushConfig.changeHuaweiBadgeNum(Context context, int changeNum);
```

示例：在收到透传消息时，调用 `XGPushConfig.changeHuaweiBadgeNum(context, 1)` 实现角标加1；在需要清除该消息的角标时调用 `XGPushConfig.changeHuaweiBadgeNum(context, -1)` 实现角标减1。

vivo 手机角标适配说明

使用限制

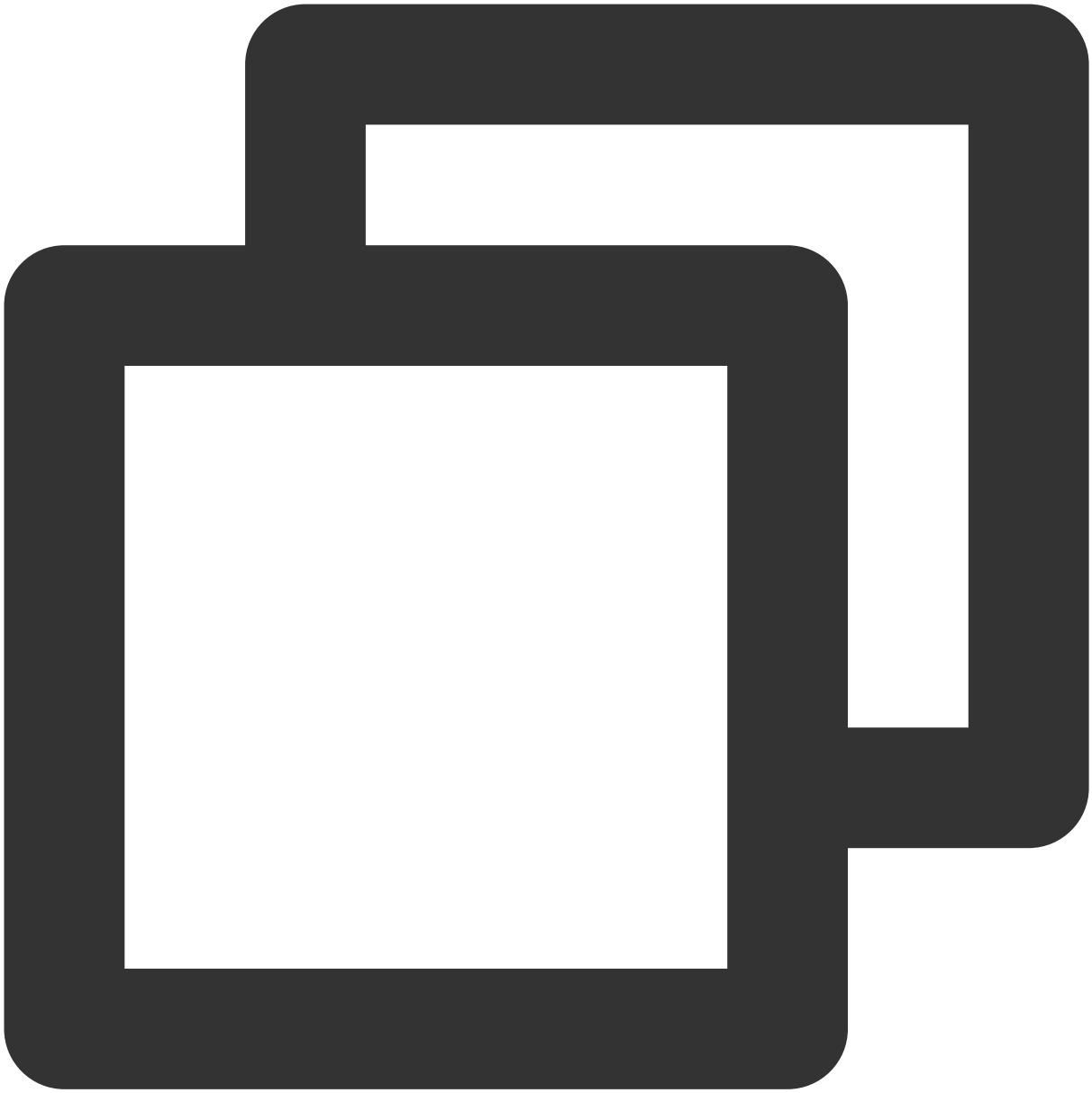
受限于 vivo 手机角标能力的开放程度，当前 vivo 手机角标仅支持直接设置角标数值，不支持自动增减；仅支持移动推送自建通道下发的通知。

推送形式	角标能力	实现方式
vivo 通道通知	不支持	不支持
移动推送自建通道通知	支持角标直接设置或不变，不支持自动增减	通过管理台或 Push API 关键字设置
透传消息	开发者自行处理设置逻辑	调用移动推送 SDK 开放接口

配置内容

应用内角标设置权限申请

为能实现角标修改的正确效果，请首先为应用添加 vivo 手机上的角标读写权限，具体实现为在应用 AndroidManifest.xml 文件的 manifest 标签下添加以下权限配置：



```
<uses-permission android:name="com.vivo.notification.permission.BADGE_ICON" />
```

手机设置内开启“桌面应用图标”

接入成功后，“桌面图标角标”默认关闭，需要用户手动开启。

开启路径：**设置 > 通知与状态栏 > 应用通知管理 > 应用名称 > 桌面图标角标**。

未成功接入“桌面图标角标”的应用，无“桌面图标角标”选项。

说明：

视 OS 版本差异，“桌面图标角标”名称可能为“应用图标标记”或“桌面角标”。

厂商通道测试方法

最近更新时间：2024-01-16 17:39:39

说明：

此测试方法适用所有版本的手机厂商通道。

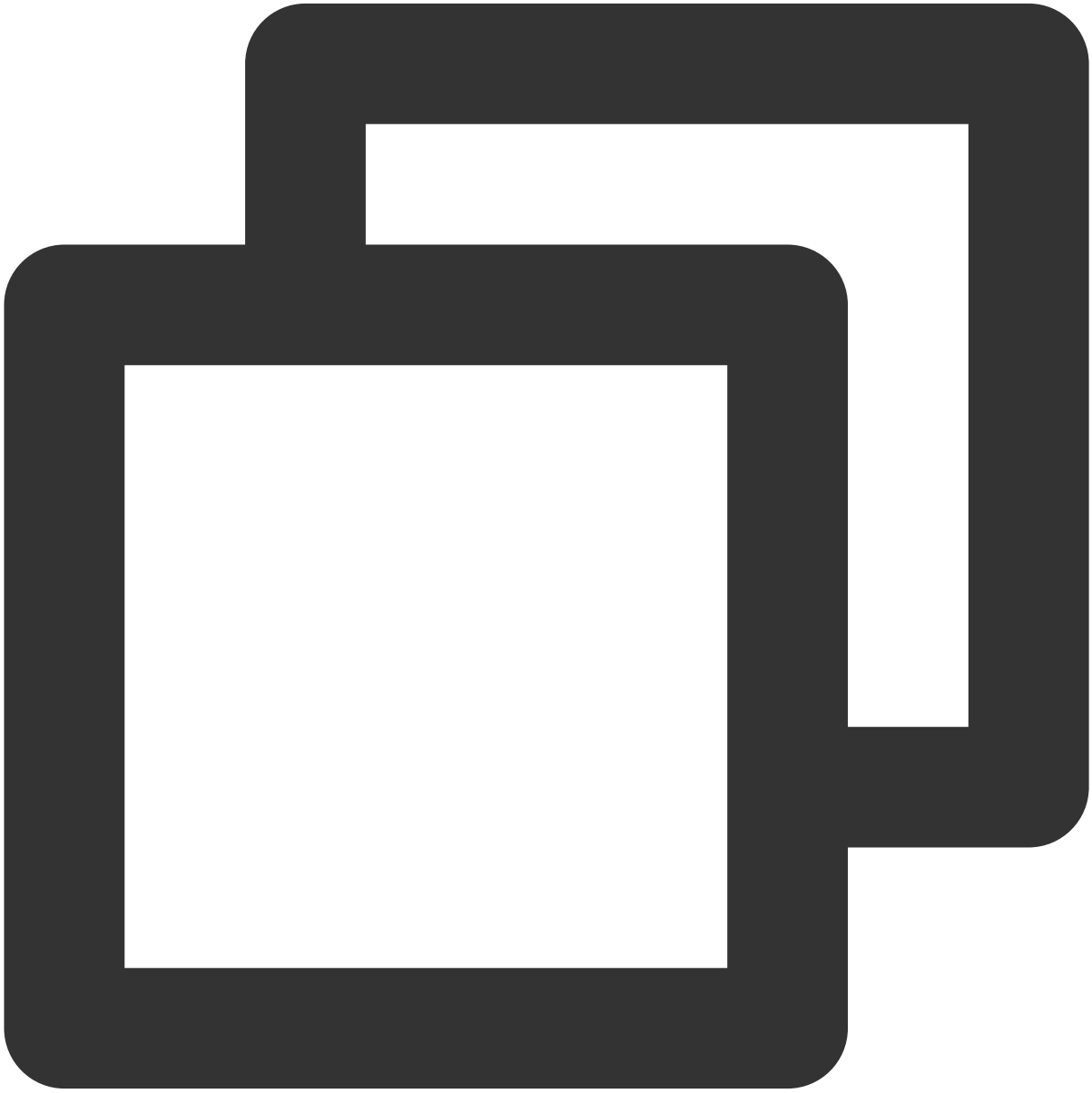
1. 在您的 App 中集成移动推送V1.0.9以上版本的 SDK，并且按照厂商通道集成指南，集成所需的厂商 SDK。
2. 确认已在 移动推送控制台 > **配置管理** > **基础配置** > **厂商通道**中填写相关的应用信息。通常相关配置将在1个小时后生效，请您耐心等待，在生效后再进行下一个步骤。
3. 将集成好的 App（测试版本）安装在测试机上，并且运行 App。
4. 保持 App 在前台运行，尝试对设备进行单推/全推。
5. 如果应用收到消息，将 App 退到后台，并且停止所有 App 进程。
6. 再次进行单推/全推，如果能够收到推送，则表明厂商通道集成成功

厂商通道注册失败排查指南

最近更新时间：2024-01-16 17:39:39

问题描述

如您的应用接入了厂商通道，但在应用运行日志中观察到如下类似日志：



```
[OtherPushClient] handleUpdateToken other push token is :  other push type: huawei
```

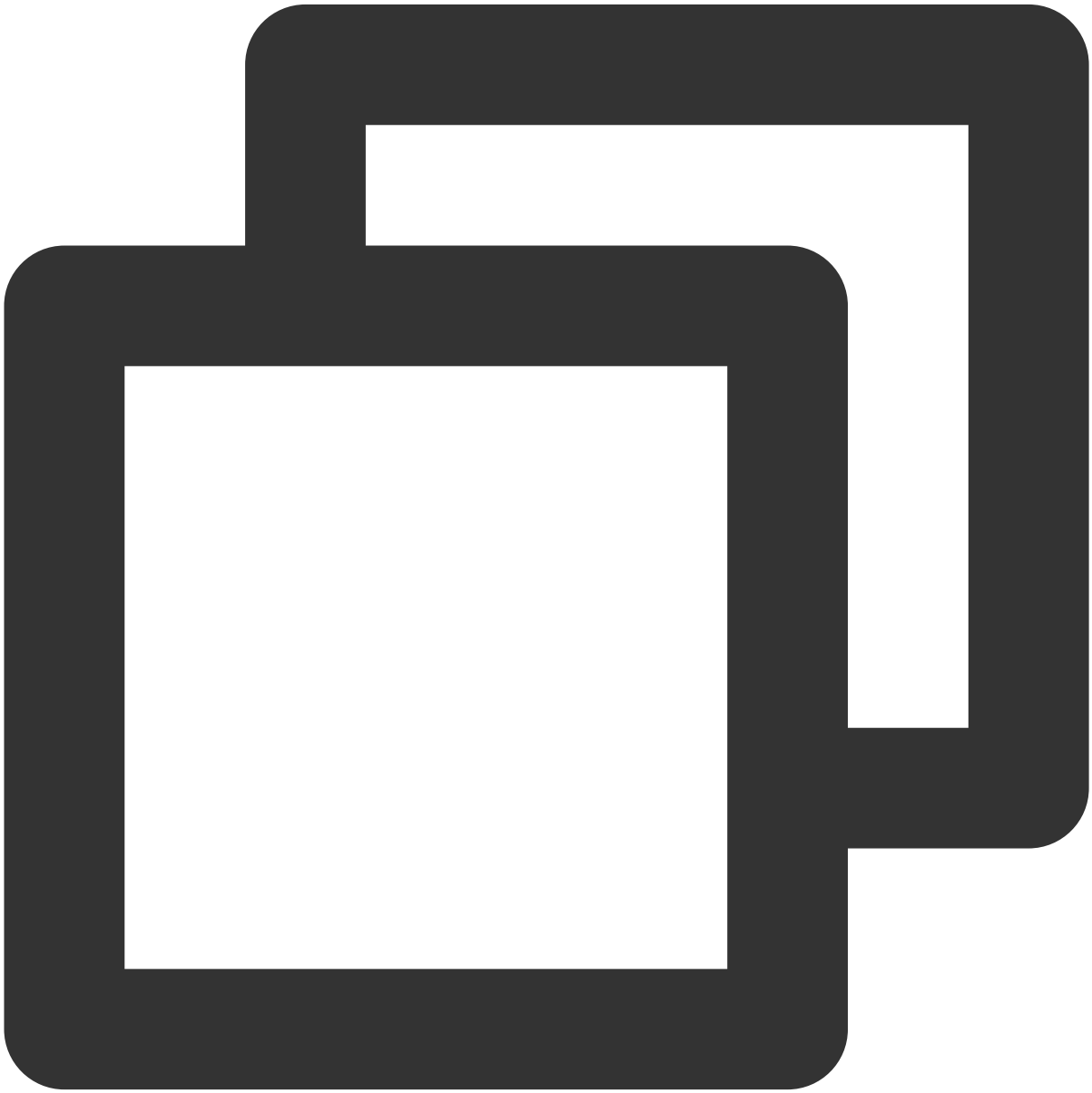
表示您的应用注册该厂商通道失败。您可以通过获取厂商通道注册失败的返回码来进行问题定位和排查。

排查步骤

获取厂商通道注册返回码

移动推送Android SDK 提供以下方式获取厂商通道注册返回码：

在应用运行日志中通过过滤关键字 `OtherPush`，找到如下类似日志来定位厂商通道注册返回码：



```
// 华为通道
// 如果过滤关键字 `OtherPush` 找不到返回码，可以过滤关键字 `HMSSDK`，并注意查看 onResult 或 c
[OtherPushHuaWeiImpl] other push huawei onConnect code:907135702

// 小米通道
[OtherPush_XG_MI] register failed, errorCode: 22022, reason: Invalid package name:

// 魅族通道
[OtherPush_XG_MZ] onRegisterStatus BasicPushStatus{code='110000', message='appId不合

// OPPO 通道
[OtherPushOppoImpl] OppoPush Register failed, code=14, msg=INVALID_APP_KEY

// vivo 通道
[OtherPushVivoImpl] vivoPush Register or UnRegister fail, code = 10003
```

返回码问题排查

您可以前往各厂商推送的官方文档获取返回码具体含义并进行问题排查。部分常见错误码可参考下表：

厂商通道	返回码	含义	解决建议	厂商通道官方返回码地址
华为	1001	请确认手机中安装有应用 “华为移动服务” 或 “HMS-Core”，华为推送必须	前往华为应用商店下载安装应用 “HMS-Core”	华为返回码参考
	6003	应用 APK 未打签名或与华为开放平台登记签名信息不一致，华为推送必须	为 APK 文件打上签名或检查签名信息配置是否一致： 华为通道配置 App 签名证书指纹	
	907135000	appId 不合法	检查华为推送配置文件 agconnect-services.json 文件内 appId 字段内容是否和应用包名匹配 检查配置文件位置是否在工程 app 模块的根目录下（和 app build.gradle 文件同级）	
	907135702	签名文件的 SHA256 值与在华为推送平台上配置的不一致	前往华为推送平台检查填写的签名文件 SHA256 值是否配置一致（华为支持添加多个）	

	907135003	apiclient对象无效	请检查手机网络是否可以正常访问互联网，或重新进行网络连接 此问题多和华为手机系统应用华为移动服务 HMS-Core 版本不兼容有关，您可以尝试进入华为应用商店搜索 HMS-Core 或华为移动服务，检查是否最新版本并进行升级	
荣耀	8001000	设备不支持荣耀推送	请使用荣耀设备测试 请更新包含推送服务Magic UI的版本	荣耀返回码参考
	8001003	应用证书指纹获取失败	请配置好您的证书指纹	
小米	22006	应用程序 ID 不合法	前往小米推送平台检查应用的包名、appId、appKey 是否匹配	小米返回码参考
	22007	应用程序 Key 不合法	前往小米推送平台检查应用的包名、appId、appKey 是否匹配	
	22022	应用程序 package name 不合法	前往小米推送平台检查应用的包名、appId、appKey 是否匹配	
魅族	110000	appId 不合法	前往魅族推送平台检查应用的包名、appId、appKey 是否匹配，是 Flyme 推送平台 的应用信息	魅族返回码参考
	110001	appKey 不合法	前往魅族推送平台检查应用的包名、appId、appKey 是否匹配	
OPPO	14	无效的 AppKey 参数	请注意 setOppoPushAppId 填入的是 OPPO 的 AppKey，不是 AppId；setOppoPushAppKey 填入的是 Oppo 的 AppSecret，不是 AppKey	OPPO 返回码参考
	15	缺少 AppKey 参数	补充 AppKey 参数	
vivo	10003	App 包名与配置不匹	前往 vivo 推送平台检查应用的包	vivo 返回码参考

		配	名、appId、appKey 是否匹配	
	10004	appkey 不匹配	前往 vivo 推送平台检查应用的包名、appId、appKey 是否匹配	
	10005	appid 传入错误	前往 vivo 推送平台检查应用的包名、appId、appKey 是否匹配	

其他排查

华为推送需要在华为推送平台开启推送服务

如您在华为设备上无法获取华为 Token，但获取到厂商推送注册返回码为0，请前往 华为推送平台，进入**开发 > 推送服务**页面，确认应用的推送开关是否开启；进入**开发 > 项目设置 > API 管理**页面，确认 `Push Kit、App Messaging` 开关是否开启。

小米推送需要在小米推送平台开启推送服务

如您未找到小米通道注册返回码，请前往小米开放平台 > **推送运营平台**，确认应用的消息推送服务是否启用。

OPPO 推送需要申请推送功能开通后才能进行正式推送

在 OPPO 开放平台 的推送服务界面可以看到已开启服务应用和未开启服务应用，在未开启服务中单击需要申请 Push 权限的应用，进入 Push 服务并单击申请开通。

vivo 推送需要申请推送功能开通后才能进行正式推送

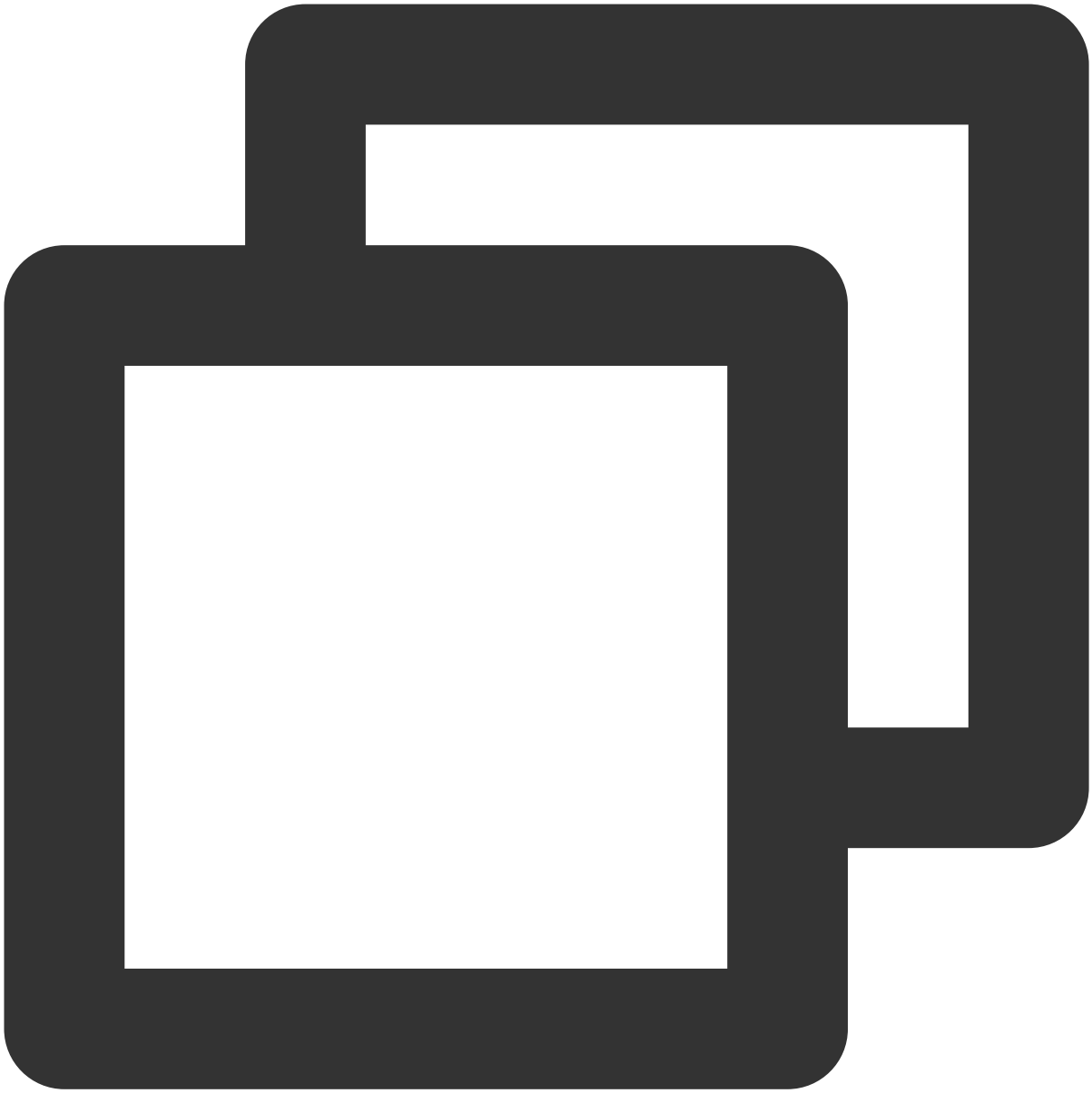
进入vivo 开放平台 > **推送运营平台】**，在**消息推送 > 全部应用**中，所创建应用将会列入在应用名称里，单击**应用名称**选择要申请的应用后单击**提交申请**。

说明：

部分厂商推送开关开启生效有约5分钟延迟，若开启开关后仍遇到注册失败，可以稍等片刻再进行尝试。

华为移动服务版本过低

搜索日志关键字“HMSSDK”，如观察如下类似日志，即 `connect versionCode` 小于 `connect minVersion`，表示系统应用“华为移动服务”或“HMS_Core”版本较低，请做升级后尝试重新注册。



```
I/HMSSDK_HuaweiApiClientImpl: ====== HMSSDK version: 20601301 ======
I/HMSSDK_HuaweiApiClientImpl: Enter connect, Connection Status: 1
E/HMSSDK_Util: In getHmsVersion, Failed to read meta data for the       HMS VERSIO
I/HMSSDK_HuaweiApiClientImpl: connect minVersion:20600000
I/HMSSDK_HuaweiMobileServicesUtil: connect versionCode:20301306
D/HMSAgent: connect end:-1001
```

vivo 部分机型不支持推送服务

vivo 推送仅支持部分较新的机型和对应的系统及以上系统，详情请参见 vivo推送常见问题汇总。

厂商消息分类功能使用说明

最近更新时间：2024-01-16 17:39:39

简介

目前，厂商会逐步对 App 开发者的通知消息根据分类进行限额限频，以此保证终端用户不被过度骚扰，不同的消息分类主要通过渠道 ID（ChannelID）进行区分。移动推送综合各厂商的分类能力，支持将消息分为两类：
普通消息（默认）：适用于推送全员公告、运营活动、热点新闻等，多为用户普适性的内容。
通知消息/个人私信：适用于推送聊天消息、个人订单变化、交易提醒等与私人通知相关的内容，通知消息的推送数量不受限制。
消息分类支持在调用推送 API 时指定。
魅族暂不支持消息分类，且不限额。

使用步骤

1. 若需要使用厂商通知消息，按以下说明申请或创建通知消息的 Channel ID：
OPPO 申请指南
小米申请指南
vivo 申请指南
华为使用指南
荣耀使用指南

说明：
从 Android 8.0（API 级别 26）开始，弹出通知栏通知必须先为应用创建通知渠道，并为要弹出的通知分配渠道，否则通知将不会显示。通过将通知分配给特定的通知渠道，则该通知将以该通知渠道已被开启的行为功能展示在通知栏中。用户可以为应用的每个通知渠道进行个性化控制，而非直接管理应用的所有通知，例如控制每个渠道的开闭、视觉和听觉选项等。
一个应用可以有多个通知渠道，建议设置不超过7个通知渠道。应用的每个通知渠道按照通知渠道id（channel_id）区分，通知渠道以通知渠道名称（channel_name）定义的文本展示在应用的通知设置中。
通知渠道一旦创建，设备用户拥有完全控制权，开发者便无法更改通知行为。对同一个通知渠道id（channel_id）进行重复创建的代码调用，仅不同的通知渠道名称（channel_name）和渠道描述参数会生效，其他的视觉、听觉、重要性等选项无法改变。
2. 若需要使用厂商做渠道分类管理，自定义 Channel ID，从而做到根据 App 自身的业务消息类别进行消息分类，可根据不同厂商对应进行配置：

推送通道	配置说明
移动推送自建通道	App 端，调用 Android SDK 创建 Channel ID 接口创建 Channel ID。 调用移动推送 服务端 API 时，指定对应的 Channel ID（不限额度）。

华为	在华为管理台 申请自分类权益，自分类权益生效后，应用的推送消息将根据hw_category字段进行归类。。 调用移动推送 服务端 API 时，指定 hw_category 参数。 华为的 ChannelID 作为自定义的渠道策略展示消息提醒方式，不作消息分类。
小米	在小米开放平台管理台上创建 Channel ID 或通过小米服务端 API 创建。 调用移动推送 服务端 API 时，指定对应的 Channel ID。
OPPO	App 端，调用 Android SDK 创建 Channel ID。 在 OPPO 管理台登记该 Channel ID，保持一致性. 调用移动推送 服务端 API 时，指定对应的 Channel ID。
魅族	无 Channel 相关说明。
vivo	支持配置使用 vivo 系统消息/运营消息，不支持自定义通知渠道 Channel 。 调用移动推送 服务端 API 时，指定vivo_ch_id参数传值”
荣耀	支持配置使用 荣耀 服务通讯/资讯营销消息，不支持自定义通知渠道 Channel 。 调用移动推送 服务端 API 时，指定hw_importance参数传值”

3. 若既不需要使用厂商通知消息，也不需要自定义 Channel ID，则无需做任何处理，移动推送会为 App 的所有消息指定一个默认的 Channel ID，消息归到默认类别中。

OPPO 通知渠道申请指南

OPPO 通知渠道介绍

OPush 平台上默认的是公信通道，目前在原有基础上新增“私信”通道，对单个用户推送个性化信息时，不再受推送数量限制。以下是“公信”和“私信”的对比：

类型		公信	私信
推送内容		热点新闻、新品推广、平台公告、社区话题、有奖活动等，多用户普适性的内容。	个人订单变化、快递通知、订阅内容更新、评论互动、会员积分变动等，与单个用户信息强相关的内容。
单用户推送限制（条/日）	新闻类（三级分类为新闻类）	5条	不限量。
	其他应用类型	2条	不限量。
推送数量限制		所有公信类通道共享推送次数，当日达到次数限制后，所	不限量。

	有公信类通道将不能再推送消息，目前单日推送数量为：累计注册用户数 * 2 。	
配置方式	默认。	需要在 OPPO PUSH 运营平台上登记该通道，并将通道对应属性设置为“私信” 。

注意：

OPPO 官方提醒：切记！一定不要利用私信通道用于普适性消息推送（如热点新闻、新品推广等），后台会实时监控，如违反运营规则，OPush 有权关闭您的私信通道权限。由此产生的后果，如调用接口异常，或使用私信通道发送的消息没到达用户等，由业务方自行承担。

OPPO 私信通道申请

1. 进入 OPPO 开放平台，在**应用配置>新建通道**中新建通道，通道 ID 与通道名称必填且需要与应用客户端保持一致，其他选项可不填。

注意：

通道 ID 一旦确定下来不能随意变更或删除。

2. 目前 OPPO 私信通道需要邮件申请后才能生效，请按照以下要求发送邮件给 OPush 平台，详情请参见 OPPO PUSH 通道升级公测邀请。

OPPO 私信通道使用

1. 客户端创建通知渠道，请选择以下任意一种方式创建：

1.1 使用 Android API 创建通知渠道，详见 Android 官方文档 创建和管理通知渠道。

1.2 使用移动推送SDK（1.1.5.4及以上的版本）创建通知渠道，详见文档 创建通知渠道。

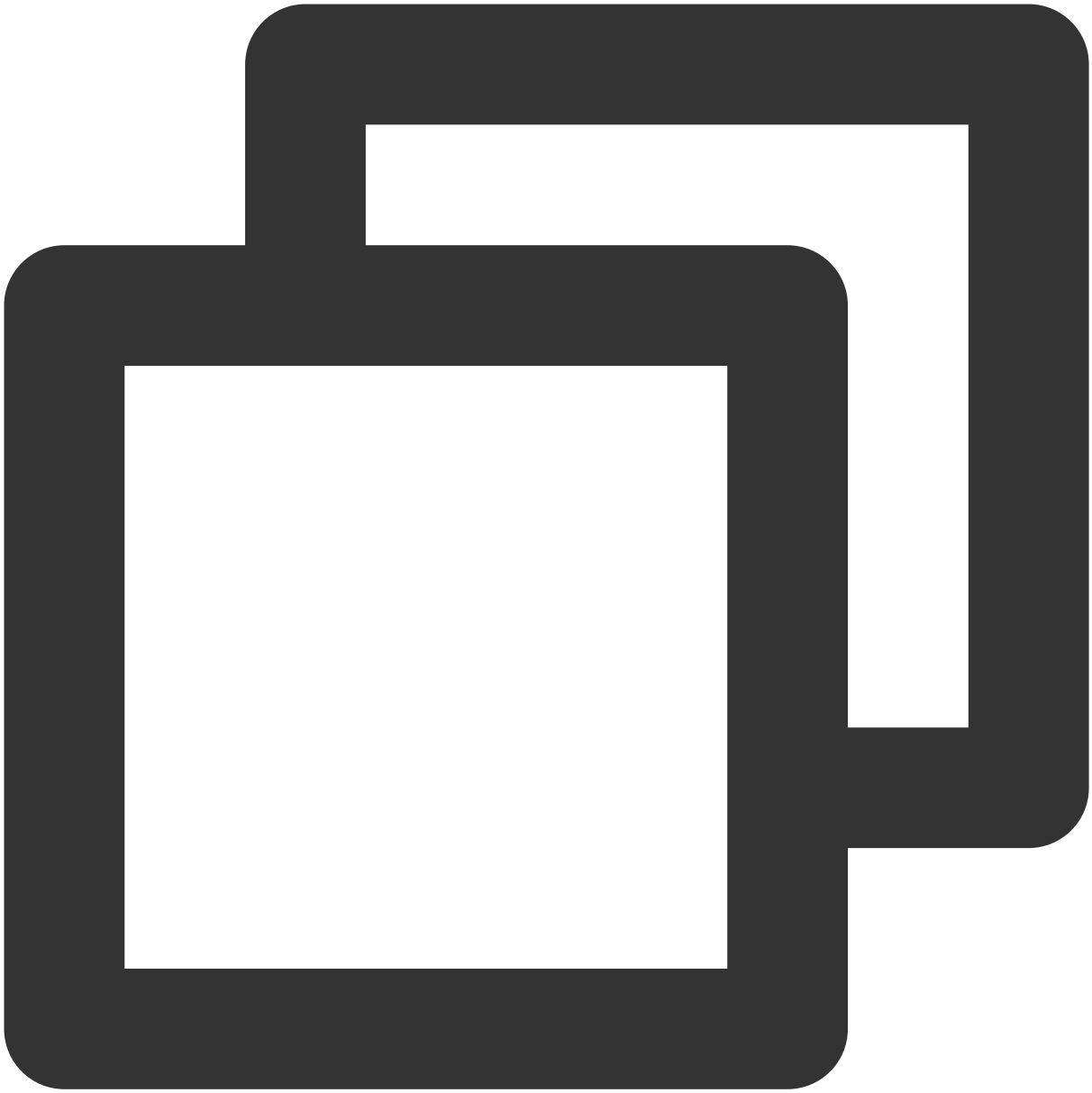
2. Rest API 推送。

在 Rest API 请求参数 Android 结构体中设置 `oppo_ch_id` 参数，可实现根据通知渠道下发，具体参见 PushAPI 参数说明。

注意：

目前 OPPO 私信通道通知只能通过 Rest API 进行下发，控制台暂不支持。

推送示例如下：



```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a1c2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
    "title": "测试标题",
    "content": "测试内容",
    "android": {
        "oppo_ch_id": "私信通道id"}
  }
}
```

小米通知渠道申请指南

小米通知渠道介绍

小米推送（Mipush）的通知渠道分为“私信消息”和“公信消息”两类，不同类别对应不同的权限，详情请参见 小米推送消息分类新规。

公信消息适用于推送热点新闻、新品推广、平台公告、社区话题、有奖活动等，多为用户普适性的内容。

私信消息适用于推送聊天消息、个人订单变化、快递通知、交易提醒、IOT 系统通知等与私人通知相关的内容，通知消息的推送数量不受限制。

小米推送对推送消息数量、推送速率 QPS 进行了统一管理，详情请参见 小米推送消息限制说明。

公信消息与私信消息限制说明：

消息类型	消息内容	用户接收数量限制	申请方式
默认	可按照小米的 公信场景说明	单个应用单个设备单日一条	无需申请
公信消息	热点新闻、新品推广、平台公告、社区话题、有奖活动等，多用户普适性的内容。	单个应用单个设备单日5-8条。	需在小米推送平台申请，详情请参见 channel 申请及接入方式。
私信消息	聊天消息、个人订单变化、快递通知、交易提醒、IoT系统通知等与私人通知相关的内容。	不限量	

小米额外提升推送量级申请

目前小米通知消息需要开发者登录小米推送运营平台，在**应用管理>通知类别**菜单页面进行申请操作，详情请参见 小米推送通知消息。

注意：

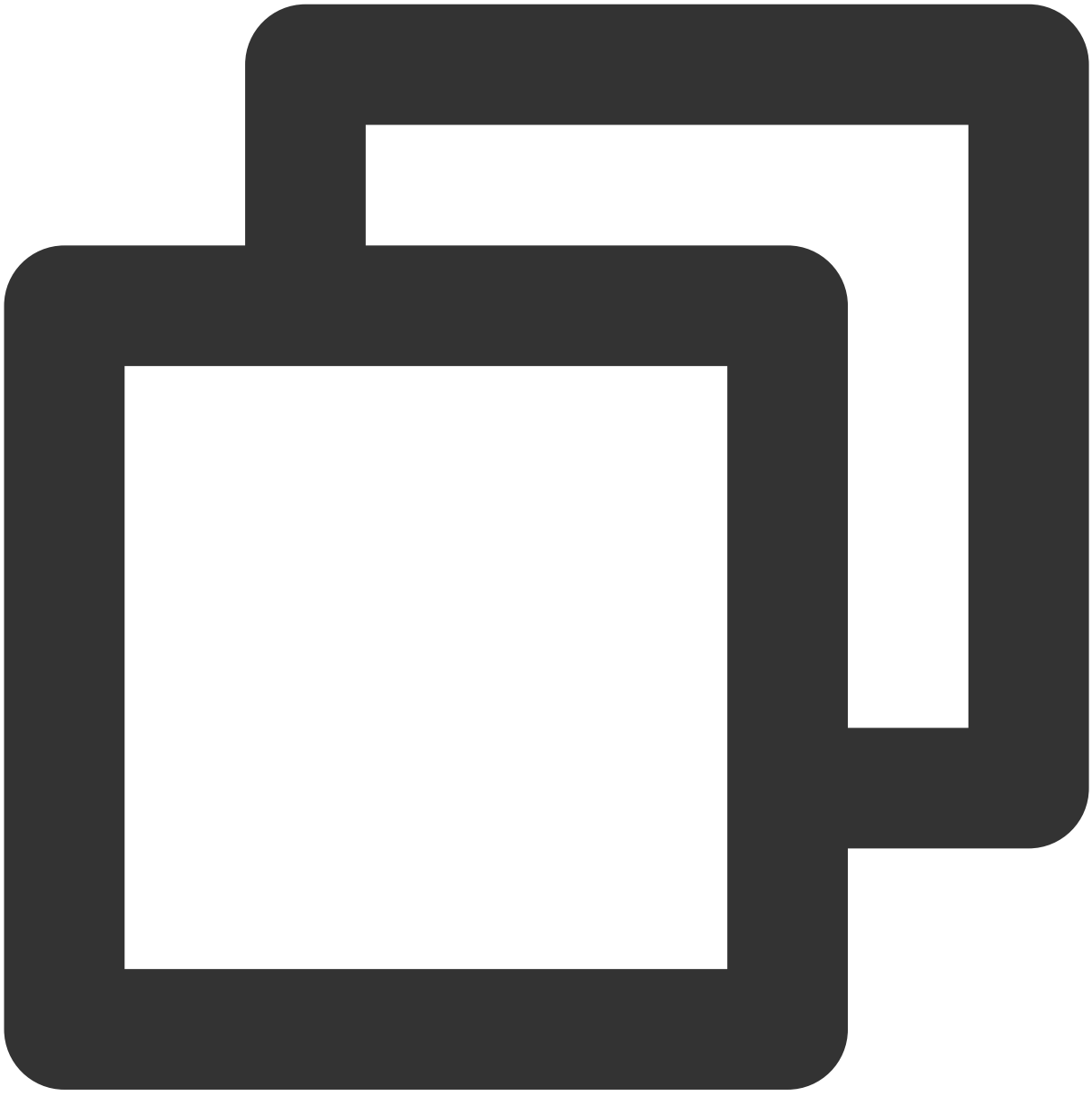
为了规范使用小米推送，请准守小米的 推送营运规则。

小米通知消息使用

目前小米通知消息只能通过 Rest API 进行下发，控制台暂不支持。

在 Rest API 请求参数 `Android` 结构体中设置 `xm_ch_id` 参数，可实现小米通知渠道下发，详情请参见 PushAPI 参数说明。

推送示例如下：



```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a***2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
     "title": "小米通知消息",
```

```
        "content": "测试内容",
        "android": {
            "xm_ch_id": "小米通知消息的channel_id"
        }
          }
}
```

vivo 系统消息申请指南

vivo 消息分类介绍

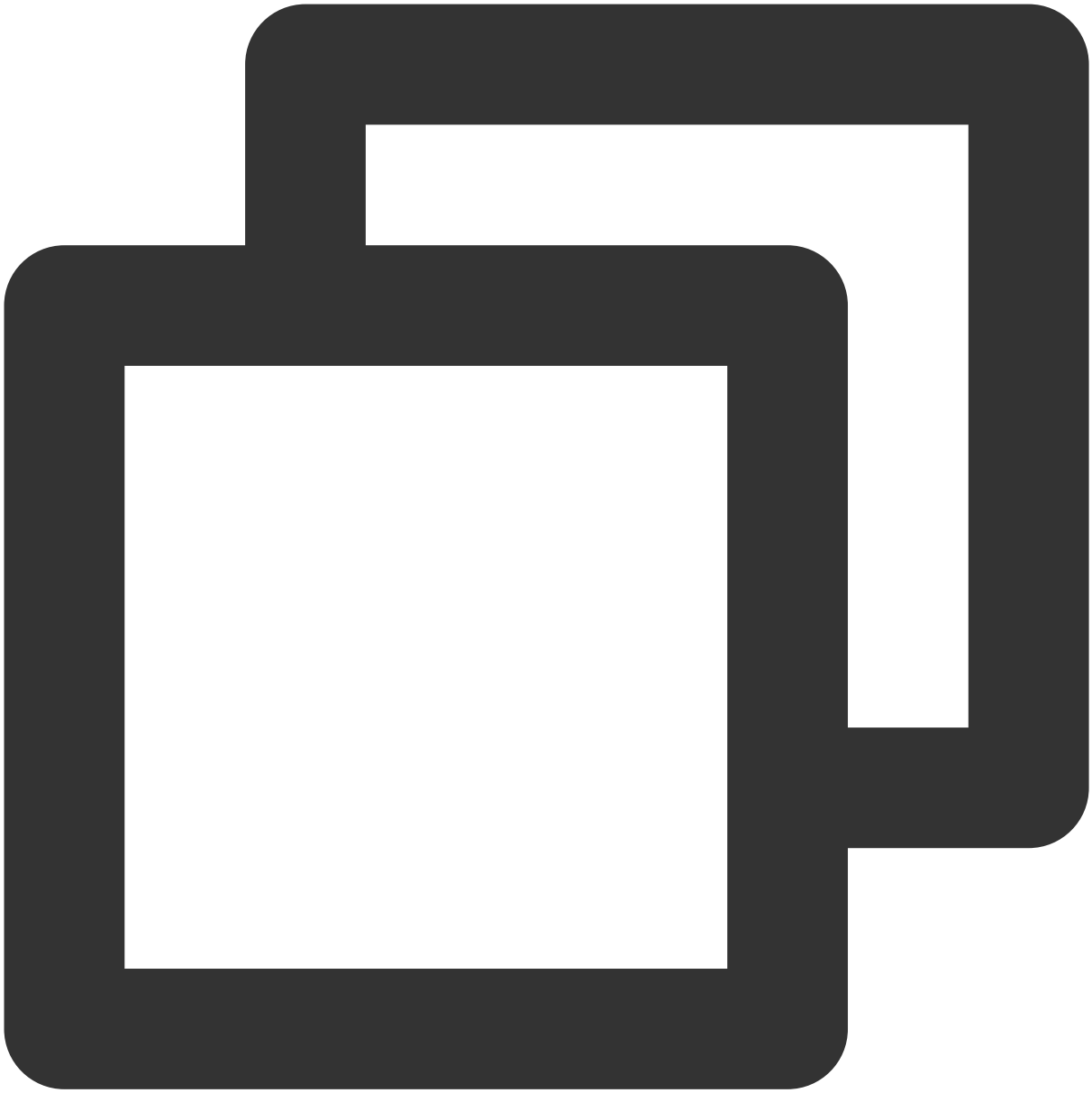
vivo 消息分类功能将推送消息类型分为运营消息和系统消息。
为提升用户消息通知体验，营造良好推送生态，vivo 推送服务于2023年4月3日起，针对不同应用类别的消息进行统一管理，详情请参见 vivo 的 推送消息限制说明。

消息分类	应用类别	推送数量限制	用户接收数量限制
系统消息	/	3倍通知开启有效用户数（可邮件申请消息不限量权限）	无限制
运营消息	新闻类（具备《互联网新闻信息服务许可证》）	3倍通知开启有效用户数	5条
	其他类	2倍通知开启有效用户数	2条

推送数量超过当日限制时，返回错误码10070或10073，10070：运营消息发送量总量超出限制；10073：系统消息发送量总量超出限制。
说明：
1. 2020年6月1日前，无论是否接入消息分类，频控规则不变，均按每个应用单用户“公共类消息（全推，群推，标签推）”每天接收上限为5条，不限制单推条数。2020年6月1日起，频控规则变更为按每个应用单用户“运营消息”接收条数上限5条进行频控，若出现用户体验类投诉，将会根据实际情况调整条数。
2. Funtouch OS_10及以上版本没有消息盒子，应用不存活时窄条展示，具体样式以实际为准。
3. 系统消息量级和运营消息量级均会随着 SDK 订阅数变化而自动变化，如特殊情况需额外提升系统消息量级，申请方法详见下方 **vivo 系统消息申请**。
4. 若某 vivo 用户当前接收运营消息超过5条，则当天触发限额后的运营消息均会通过移动推送自建通道下发，不再通过 vivo 通道下发。
5. 若您已向 vivo 申请提升运营消息限额，请联系我们进行后台配置，否则限额变更无法生效。

vivo 系统消息申请

系统消息量默认是3倍的 SDK 订阅数，当系统消息不够时，可以邮件按实际需求 申请系统消息 量级，按照下面的邮件模板填写信息后发送至邮箱：push@vivo.com。



主题：xxx应用申请增加im消息/系统消息发送量级
正文：……
应用名称：……
包名：……
应用简介：……
IM消息/系统消息需求量级（万）：……
具体推送场景说明：例如，用户个人聊天、商家聊天

vivo 系统消息使用

目前 vivo 通知消息只能通过 Rest API 进行下发，控制台暂不支持。

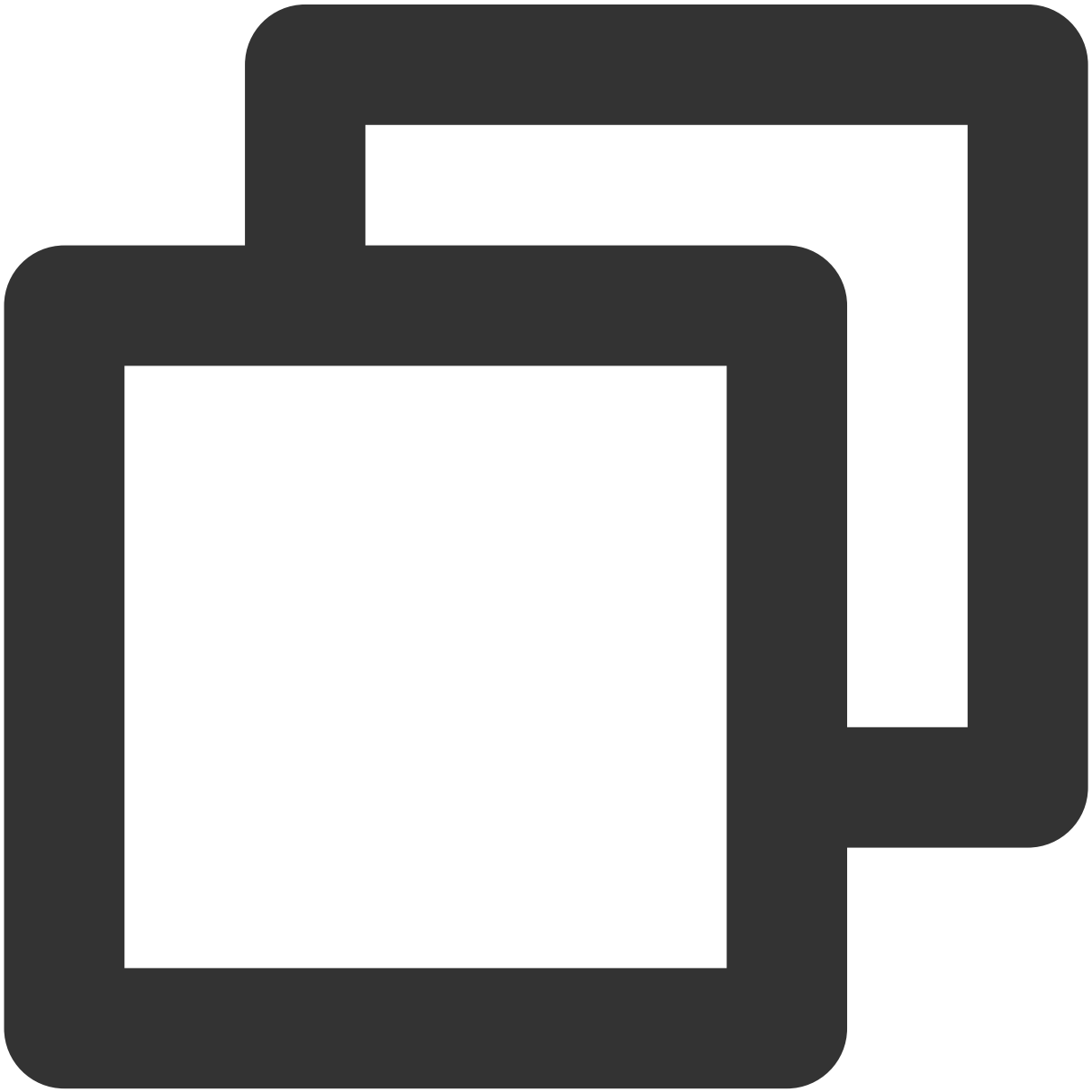
在 Rest API 请求参数 `Android` 结构体中设置 `vivo_ch_id` 参数为"1"，可实现 vivo 系统消息下发，详情请参见 PushAPI 参数说明。

说明：

vivo 推送平台将按消息分类标准，对系统消息进行每日巡检，巡查开发者以系统消息渠道发送运营消息的情况，并按违规程度及频次进行相应处罚，最高将关闭消息推送功能。

请开发者参见 消息分类功能 文档，严格按照平台消息分类运营。具体处罚规则：消息分类功能说明>五.运营监管及处罚>3.处罚标准。

推送示例如下：



```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a***2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
     "title": "vivo系统通知",
     "content": "测试内容",
     "android": {
        "vivo_ch_id": "1"
     }
   }
}
```

华为消息分类使用指南

华为消息分类介绍

华为推送从 EMUI 10.0版本开始将通知消息智能分成两个级别：「服务与通讯」和「资讯营销」。EMUI 10.0之前的版本没有对通知消息进行分类，只有一个级别，消息全部通过“默认通知”渠道展示，等价于 EMUI 10.0的服务与通讯。资讯营销类消息的每日推送数量自2023年01月05日起根据应用类型对推送数量进行上限管理，服务与通讯类消息每日推送数量不受限。

华为回执 256 表示当日的发送量超出资讯营销类消息的限制，请您调整发送策略，各消息类型详情请参见华为的 推送数量关系细则。如图所示：

不同消息级别呈现样式对比：

消息级别	在通知中心的显示	在状态栏上显示	锁屏通知	响铃	振动
服务与通讯	正常显示	支持	支持	支持	支持
资讯营销	正常显示	否	否	否	否

若您希望服务与通讯类消息也按照静默的方式发送，可以添加hw_importance字段且传值为“1”，详情请参见 PushAPI 参数说明。

分类规则：

消息智能分类

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

消息自分类

2021年7月1日起，华为推送服务开始接收开发者自分类权益的申请。申请成功后，允许开发者根据华为推送分类规范，自行对消息进行分类。

自分类消息权益申请

华为通知消息自分类权限需要申请后才能生效，详情请参见 消息分类方式。

注意：

若应用没有自分类权益，则应用的推送消息将通过智能分类进行自动归类。

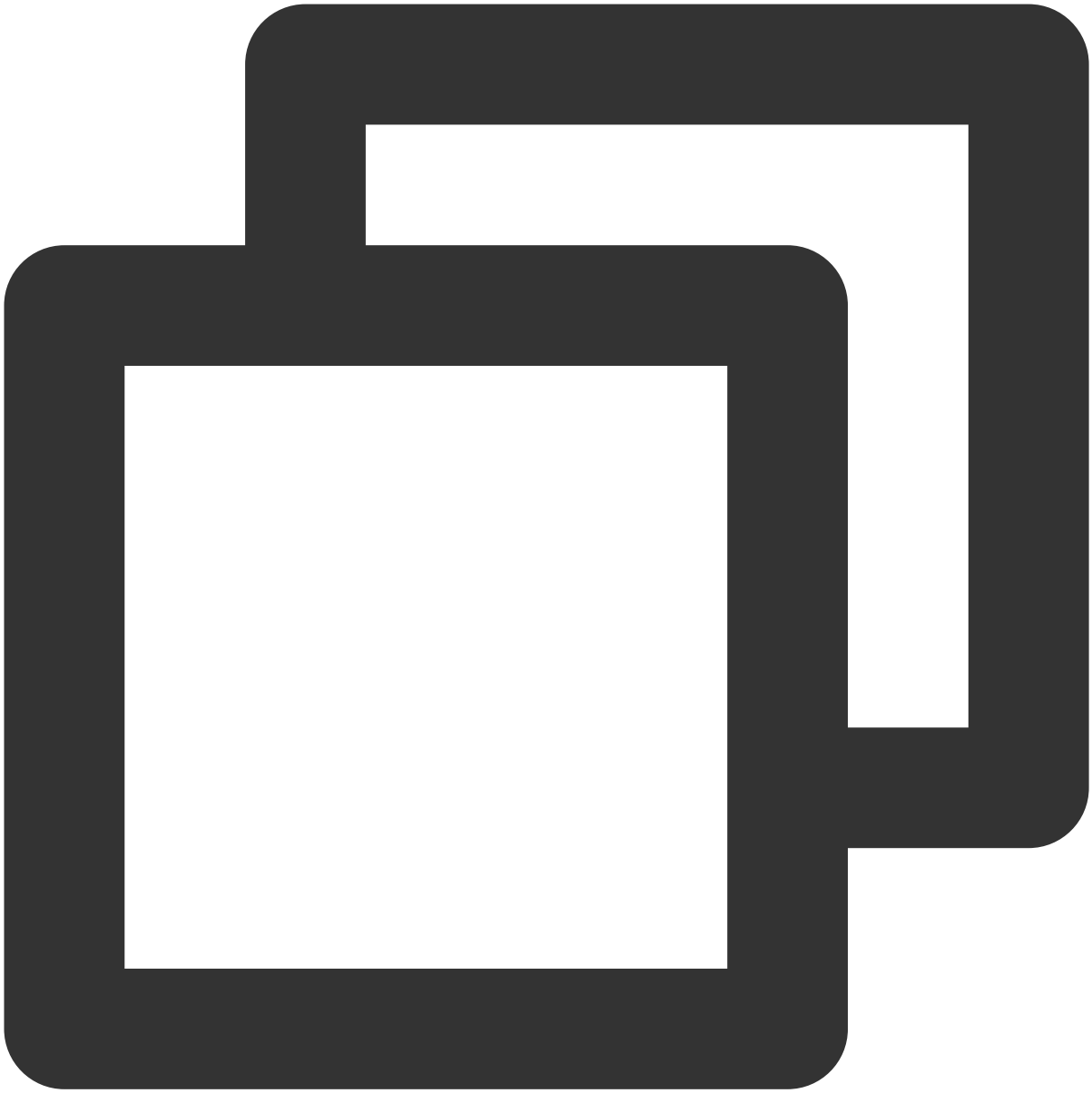
若应用有自分类权益，将信任开发者提供的分类信息，消息不经过智能分类。

自分类消息使用

自分类消息仅支持 API 进行下发，控制台暂不支持，在您自分类消息申请后，可通过以下方式使用：

在 Rest API 请求参数 `Android` 结构体中设置 `hw_category` 参数，可实现华为自分类消息下发，详情请参见 PushAPI 参数说明。

推送示例如下：



```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a***2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
        "title": "帐号下线：",
        "content": "您的帐号出现异地登录，已经下线。",
        "android": {
            "hw_category":"VOIP"
        }
    }
```

```
}
```

华为通知渠道创建

华为推送支持应用自定义通知渠道，客户端创建通知渠道，请选择以下任意一种方式创建：

1. 使用 Android API 创建通知渠道，详情请参见 Android 官方文档 创建和管理通知渠道。
2. 使用移动推送SDK（1.1.5.4及以上的版本）创建通知渠道，详情请参见文档 创建通知渠道。

华为通知渠道使用

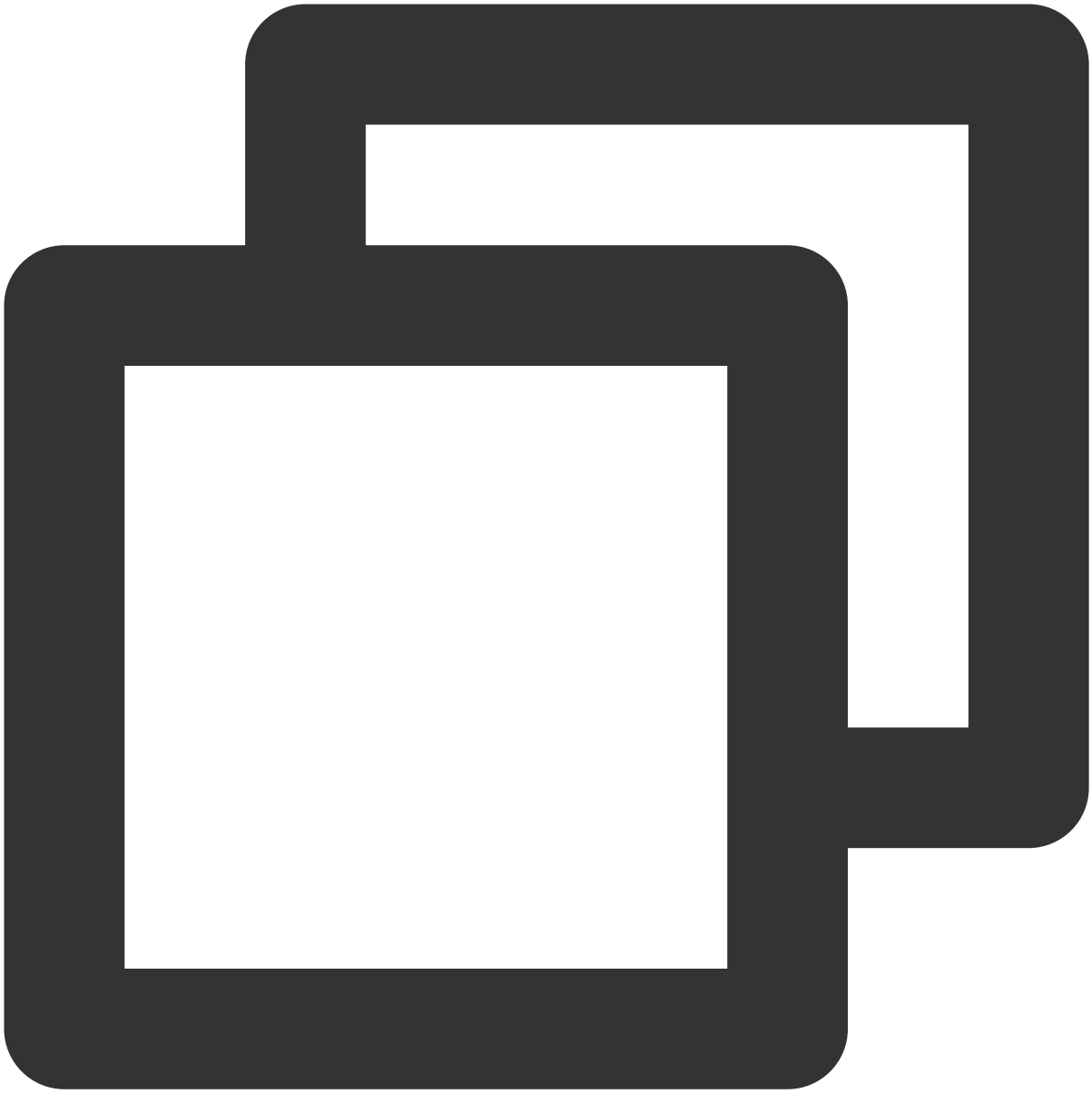
目前自定义渠道只能通过 Rest API 进行下发，控制台暂不支持，在您创建完成通知渠道后，可通过以下方式使用：在 Rest API 请求参数 `Android` 结构体中设置 `hw_ch_id` 参数，可实现华为通知渠道下发，详情请参见 PushAPI 参数说明。

注意：

如果您的应用在华为推送控制台申请开通华为推送服务时，选择的数据处理位置为中国区，自定义渠道功能将不再适用于您的应用。您的推送消息将按照智能分类系统或消息自分类权益确认的消息级别，归类为服务与通讯类或资讯营销类消息。详见 自定义通知渠道。

自定义渠道功能需要您的应用具有消息自分类权益，请参见上文进行申请。

推送示例如下：



```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a***2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
        "title": "华为通知消息",
        "content": "测试内容",
        "android": {
            "hw_ch_id": "华为通知消息的channel_id"
        }
    }
```

```
}
```

荣耀消息分类使用指南

荣耀消息分类介绍

荣耀推送服务将根据应用类型、消息内容和消息发送场景，将推送消息分成服务通讯和资讯营销两大类别，资讯营销类消息的每日推送数量根据应用类型对推送数量进行上限管理，服务与通讯类消息每日推送数量不受限，详细说明请参见 荣耀推送数量管理细则。

根据消息分类，对不同类别消息的默认展示方式、消息样式进行差异化管理，具体如下：

消息级别	通知栏下拉的显示	图标显示	锁屏通知	响铃	振动
服务与通讯	正常显示	支持	支持	支持	支持
资讯营销	正常显示	否	否	否	否

importance 字段值为“1”时：表示消息为资讯营销类，默认展示方式为静默通知，仅在下拉通知栏展示。

importance 字段值为“2”时：表示消息为服务通讯类，默认展示方式为锁屏展示+下拉通知栏展示。

分类规则：

消息智能分类

智能算法将根据APP类型和消息内容等维度，自动将您的消息按照分类标准进行归类。

消息自分类

允许开发者根据消息分类规范，自行对消息进行分类。

目前，所有消息默认通过消息自分类方式进行分类处理，荣耀推送服务将充分信任开发者提供的分类结果，并展示对应信息。随着荣耀推送服务能力的不断补充和演进，分类方式也会逐渐更新与升级，请参见 荣耀最新分类说明。

注意：

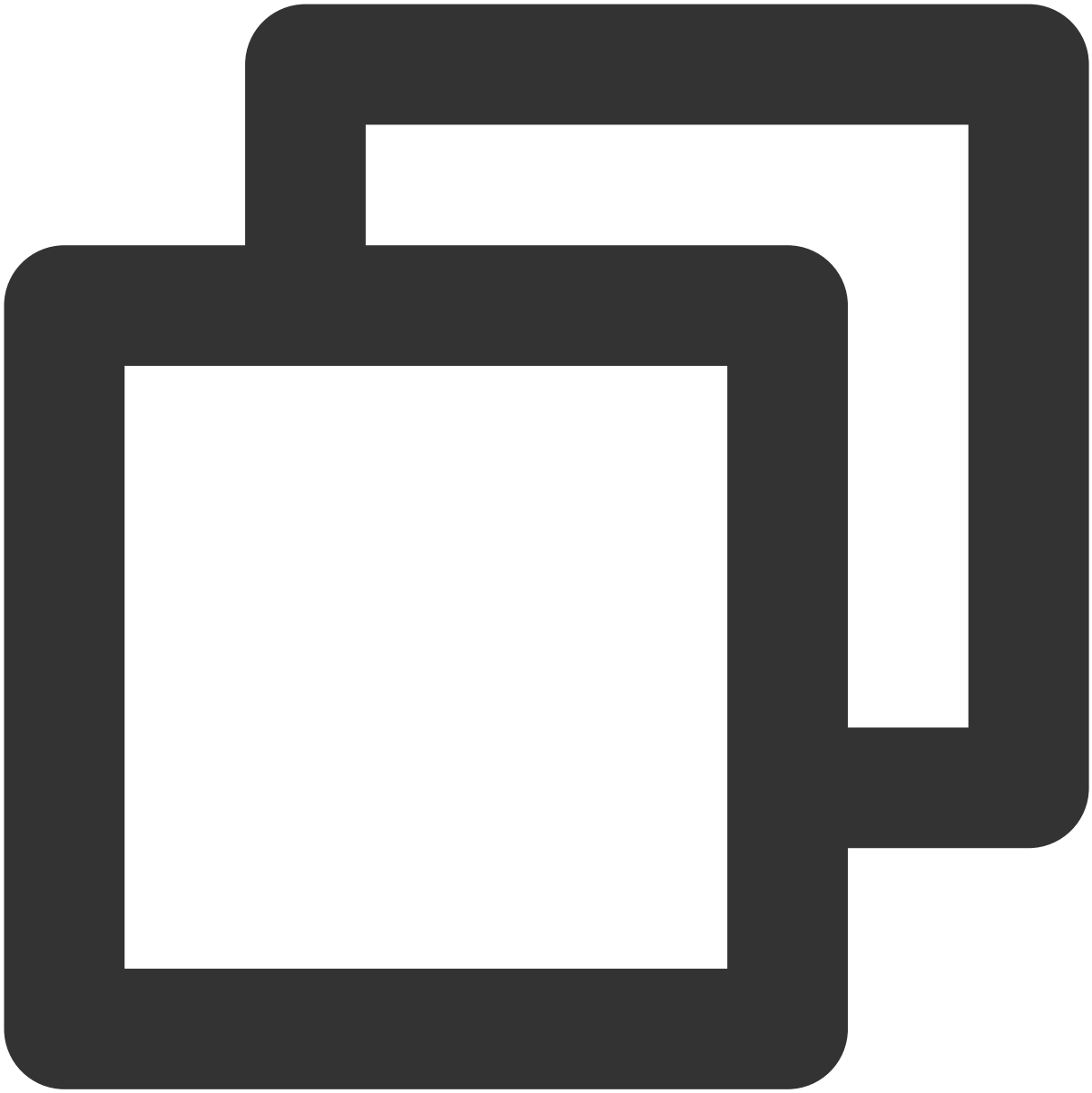
请遵守荣耀的推送分类规则，违规者将受到荣耀对应的处罚，违规行为及相应的处罚措施请参见 消息违规处罚标准。

自分类消息使用

自分类消息仅支持 API 进行下发，控制台暂不支持，可通过以下方式使用：

在 Rest API 请求参数 `Android` 结构体中设置 `hw_importance` 参数，可实现荣耀自分类消息下发，详情请参见 PushAPI 参数说明。

推送示例如下：



```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a***2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
        "title": "荣耀：",
        "content": "自分类推送。",
        "android": {
            "hw_importance":2
        }
    }
```

```
}
```

厂商通道抵达回执获取指南

最近更新时间：2024-01-16 17:39:39

为了帮助客户全链路分析推送效果，移动推送提供厂商通道抵达数据展示。国内不同厂商通道对抵达数据回执的支持程度不同，部分通道抵达数据不能直接获取，需要开发者进行相应配置。
配置成功后，开发者可在移动推送控制台推送详情中查看推送转化数据，也可通过移动推送Rest API 获取推送转化数据。

概览

厂商通道	是否支持抵达	是否需要配置
华为通道	是	**是**
魅族通道	是	**是**
小米通道	是	否
OPPO 通道	是	否
vivo 通道	是	否
FCM 通道	是	否

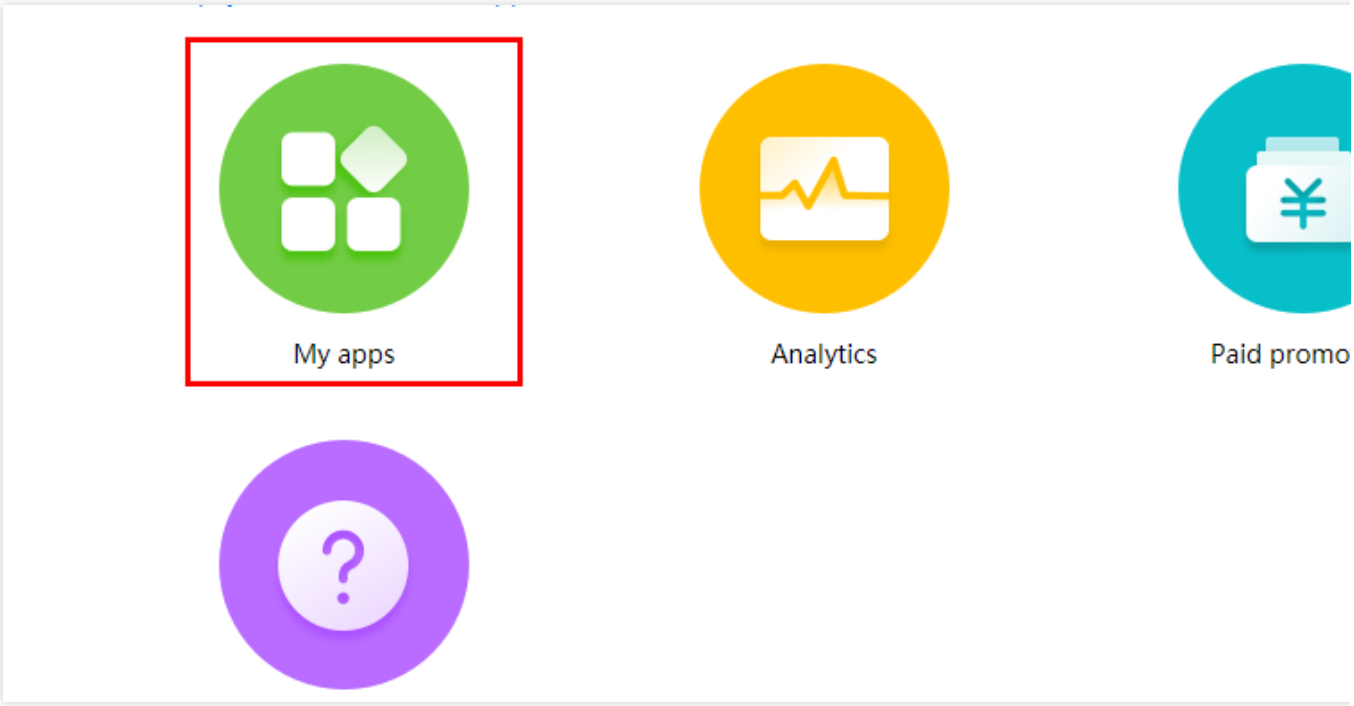
说明：
厂商通道抵达回执数据仅供参考。

华为厂商通道回执配置指引

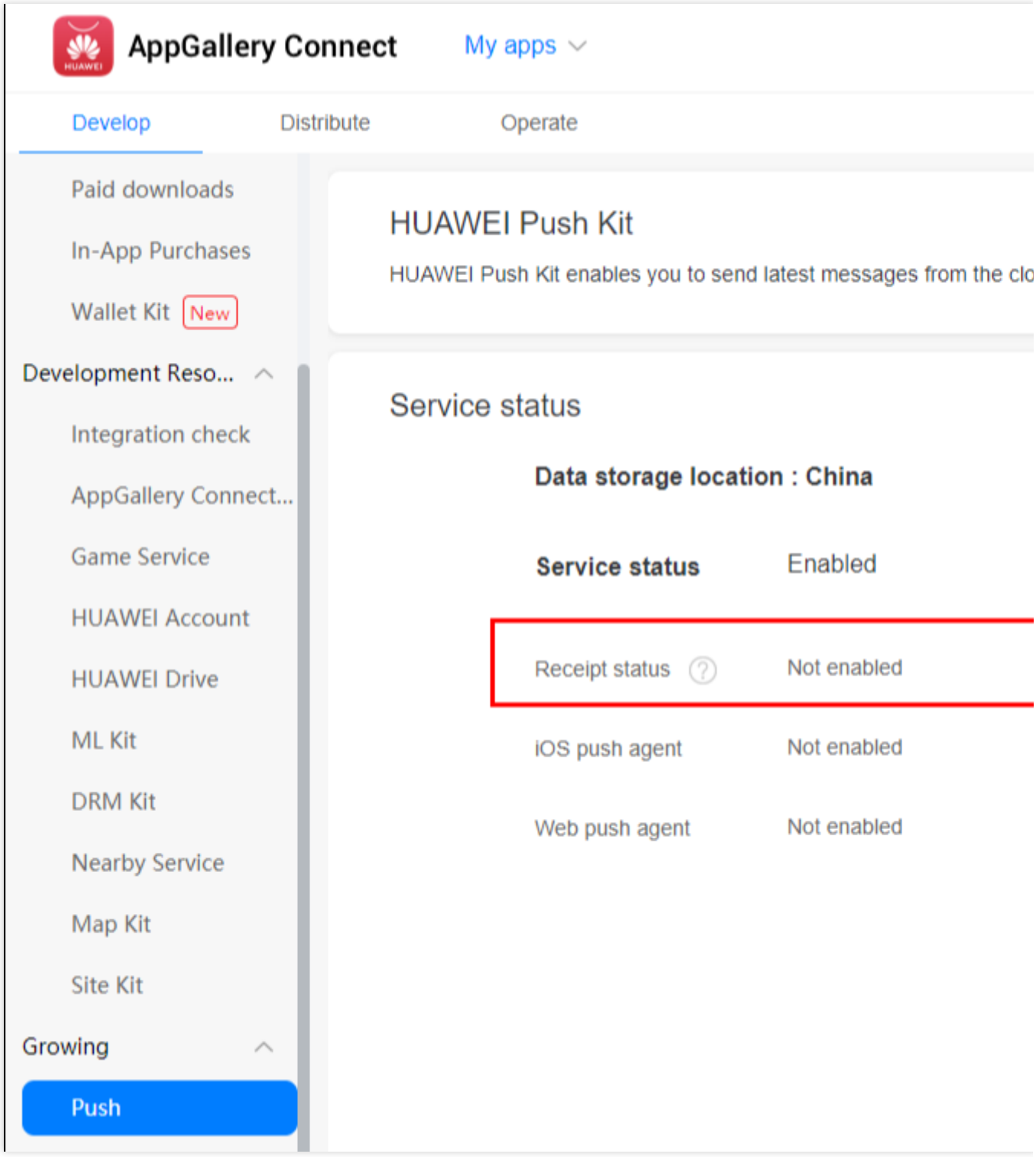
完成华为厂商通道 SDK 集成后，需要开发者在华为开放平台开通并配置消息回执，才能获取到华为通道抵达数据。具体配置方法可参考华为开放平台 消息回执，配置流程如下：

开通回执权益

1. 登录 AppGallery Connect 网站，选择**我的应用**。



2. 选择需要开通服务的应用所属产品的名称，进入应用信息页面。
3. 在“应用信息”页面，选择**全部服务>推送服务**。
4. 在“推送服务”页面，找到**应用回执状态**，单击**开通**。



回执参数配置

1. 配置消息回执地址。请在 移动推送控制台 查看您的应用的服务接入点，并复制该应用的 AccessID，选择对应服务接入点的回执地址进行配置：

服务接入点	回执地址
广州服务接入点	https://stat.tpns.tencent.com/log/statistics/hw/AccessID

中国香港服务接入点	https://stat.tpns.hk.tencent.com/log/statistics/hw/AccessID
新加坡服务接入点	https://stat.tpns.sgp.tencent.com/log/statistics/hw/AccessID
上海服务接入点	https://stat.tpns.sh.tencent.com/log/statistics/hw/AccessID

说明：

AccessID 替换为您应用的 AccessID。例如应用为广州服务接入点，则回执地址配置为 `https://stat.tpns.tencent.com/log/statistics/hw/1500016691`

2. 根据您的应用服务接入点，下载对应的 HTTPS 证书，下载完成后，用文本打开该证书，将内容复制到文本框中。

服务接入点	下载地址
广州服务接入点	点击下载
中国香港服务接入点	点击下载
新加坡服务接入点	点击下载
上海服务接入点	点击下载

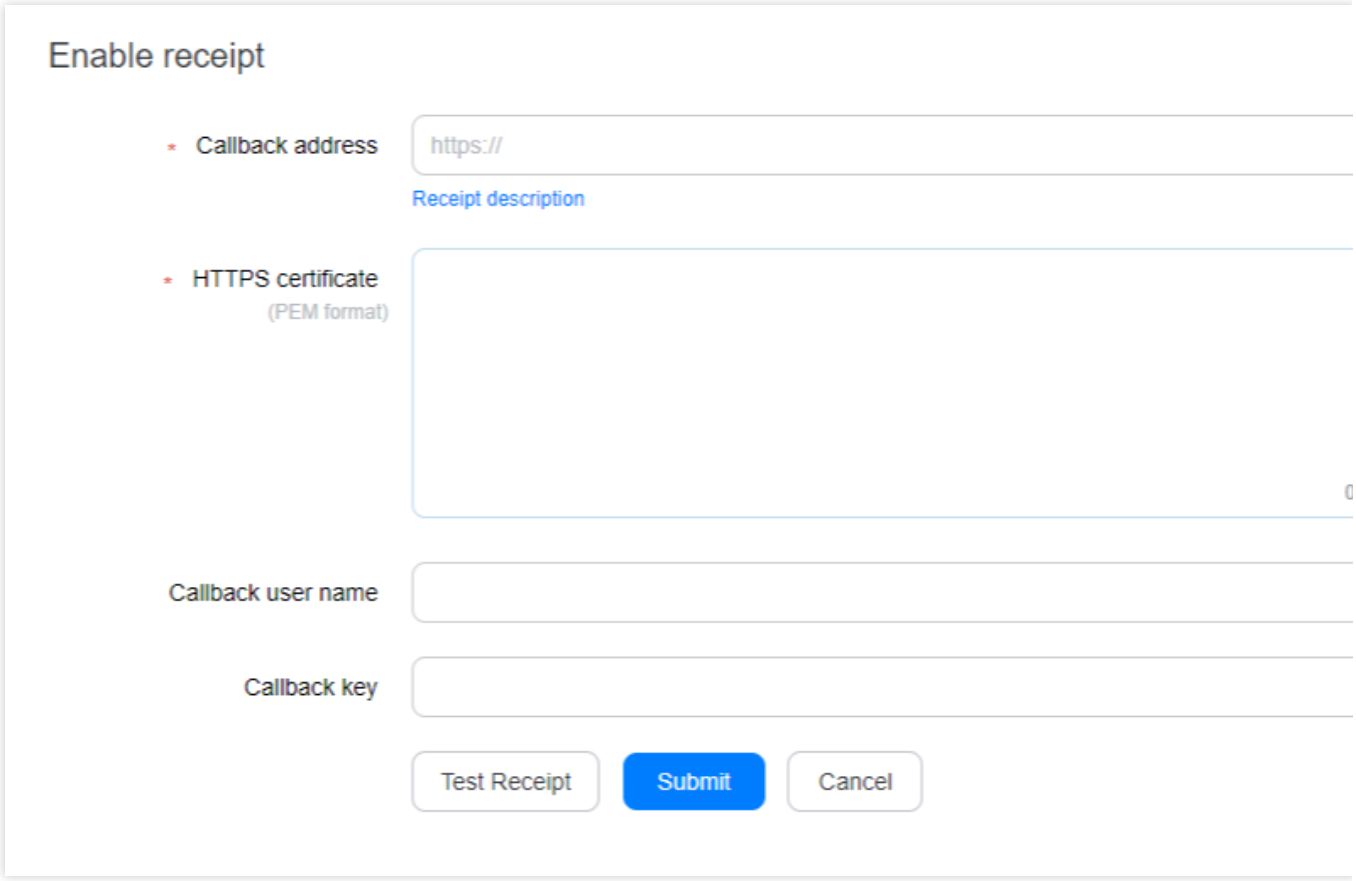
3. 配置用户名和密钥（非必填）进行身份验证。

4. 单击**测试回执**，可以对回执地址进行功能测试。

注意：

目前单击**测试回执**，会提示“测试回调地址失败”，请忽略并直接单击**提交**。

5. 单击**提交**，即可完成服务的开通。



魅族厂商通道回执配置指引

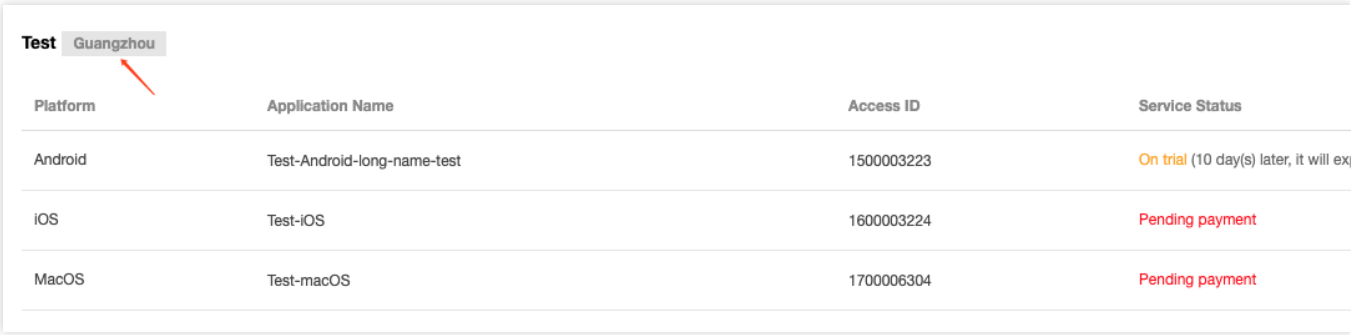
完成魅族厂商通道 SDK 集成后，需要开发者在 Flyme 推送平台中新建回执，并在移动推送控制台完成激活后，才能获取到魅族通道抵达数据，配置方法如下：

注意：

以下两步缺一不可，否则可能导致魅族通道下发失败。

配置回执

1. 登录 Flyme 推送平台 ，选择需要配置回执的应用，单击**打开应用**。
2. 在推送通知页面，点击**配置管理>回执管理**。
3. 在**新建回执**中填写回执地址。请在 移动推送控制台 查看您的应用的服务接入点，并选择对应服务接入点的回执地址进行配置：



服务接入点	回执地址
广州服务接入点	https://api.tpns.tencent.com/log/statistics/mz
	https://stat.tpns.tencent.com/log/statistics/mz
中国香港服务接入点	https://stat.tpns.hk.tencent.com/log/statistics/mz
新加坡服务接入点	https://stat.tpns.sgp.tencent.com/log/statistics/mz
上海服务接入点	https://stat.tpns.sh.tencent.com/log/statistics/mz

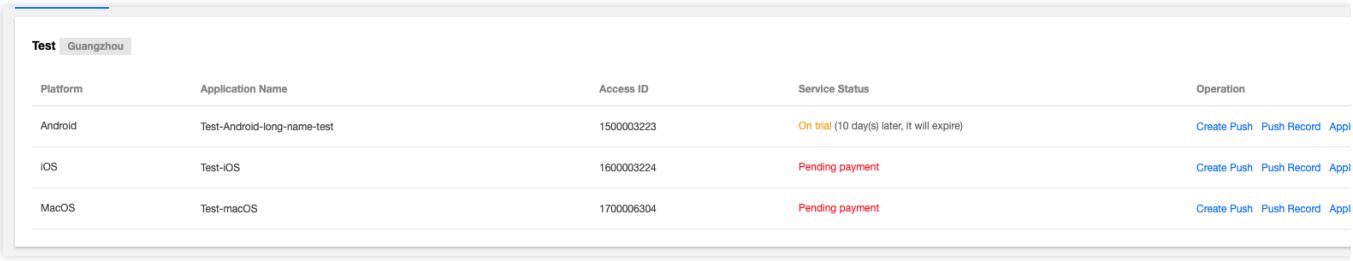
注意：

广州服务接入点的两个回执地址都需要填写。

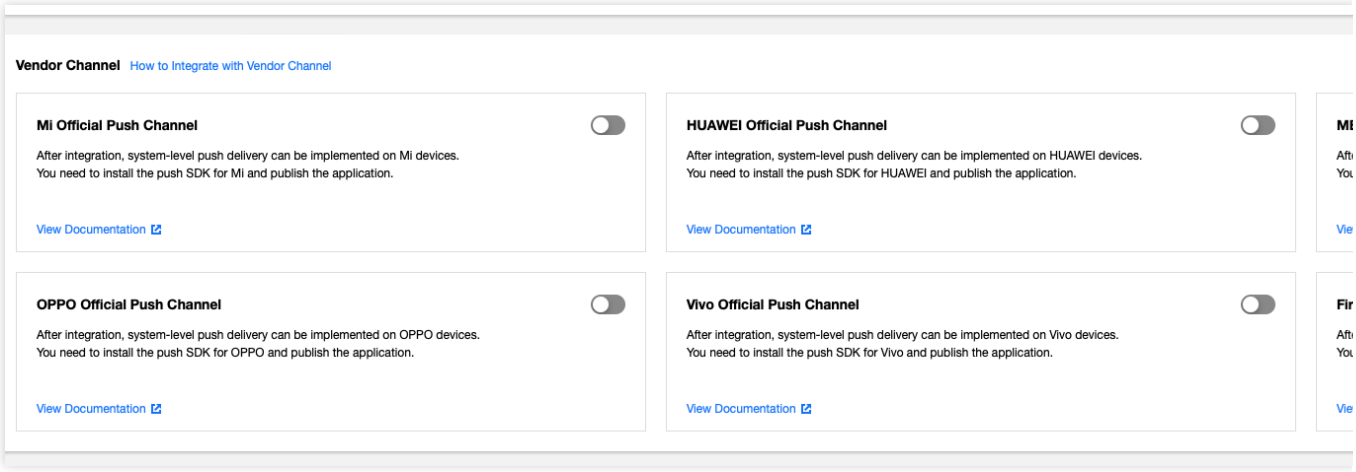
4. 填写回执地址后，单击右侧**新增**，**回执列表**中正确显示新建的回执即完成配置。

激活回执

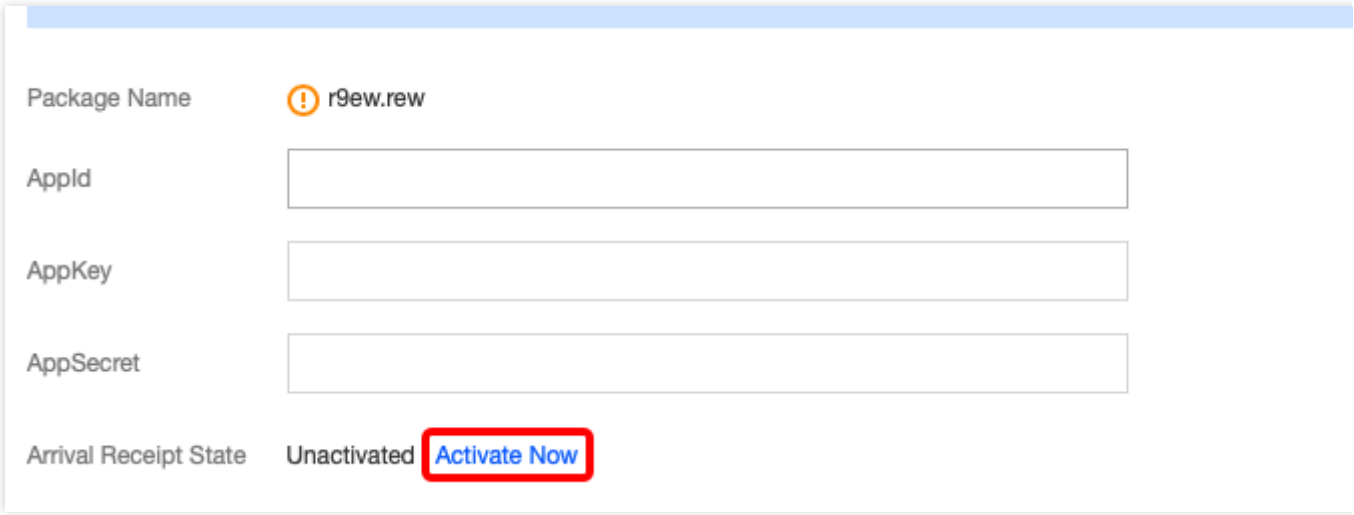
1. 进入 产品管理 页面，选择需要激活的应用，单击**配置管理**。



2. 在配置管理页面，**厂商通道>魅族官方推送通道>** 单击编辑图标。



3. 在编辑魅族官方推送通道页面，按照提示单击**点击激活**。



荣耀厂商通道回执配置指引

完成荣耀厂商通道 SDK 集成后，如需开通荣耀通道抵达回执，当前需要开发者联系荣耀 Push 团队，向其提供如下回调地址进行配置，请参见 荣耀推送回执模块。

回执参数配置

请在 移动推送 控制台 查看您的应用的服务接入点，并复制该应用的 AccessID，选择对应服务接入点的回执地址进行配置。

服务接入点	回执地址
广州服务接入点	https://stat.tpns.tencent.com/log/statistics/honor/AccessID
中国香港服务接入点	https://stat.tpns.hk.tencent.com/log/statistics/honor/AccessID

新加坡服务接入点	https://stat.tpns.sgp.tencent.com/log/statistics/honor/AccessID
上海服务接入点	https://stat.tpns.sh.tencent.com/log/statistics/honor/AccessID

说明：

AccessID 替换为您应用的 AccessID。例如应用为广州服务接入点，则回执地址配置为：`https://stat.tpns.tencent.com/log/statistics/honor/1500016691`

厂商通道限额说明

最近更新时间：2024-01-16 17:39:39

vivo 平台限制

限额说明

推送消息限额说明

正式消息分为系统消息和运营消息，两者每日限制发送量均根据 SDK 订阅数推算，SDK 订阅数小于10000，按10000计数；大于10000，则等于 SDK 订阅数。如特殊情况需额外提升系统消息量级，申请方法详见 vivo 系统消息申请。
系统消息：包括邮件、用户设置的提醒、物流、订单、待办待阅读、财务、功能提醒、即时消息8类消息。
运营消息：包括但不限于广告、推荐、推广、活动等对用户有主动运营作用的推送，或者其他非用户主动触发的信息；未订阅的影音视听内容、商品推广、宣传，或者折扣、红包、领券优惠信息等。
通过 API 发送的测试消息每日限制发送量为运营消息100条，系统消息10000条，测试设备设置上限20台。
目前不限制单推和群推的比例，可发送的单推和群推消息指定的用户量不得超过每日限制的推送总量。

接收消息限额说明

用户单应用每日运营消息接收条数上限5条，系统消息无限制。后续会结合平台对用户体验的要求和用户反馈情况对条数进行调整，具体以 vivo 官方公告为准。
用户单应用接收条数限制以“到达量”是否超过5条为准，在发送时校验单用户是否到达5条，超限则计入管控量。

额度查询指引

在vivo 开放平台 > **推送统计** > **推送数据**中可以查看 SDK 订阅数和可发送的消息总量，详情请参见 vivo 推送平台使用指南。

OPPO 平台限制

限额说明

公信通道（适用于默认的多用户普适性消息推送）中累计用户数 < 50000，可推送量为100000，累计用户数 ≥ 50000，可推送总数量为累计用户数 × 2。
私信通道（适用于单个用户的私人消息推送）的推送数量不受限制，详情请参见 OPPO 通知渠道介绍。
用户接收数量限制：通过 OPPO 推送通道下发的消息（包含公私信），单用户接收上限2000条/日。
单设备推送条数限制，详情请参见 OPPO 推送服务受限说明。

类型	公信	私信

单用户推送限制（条/日）	新闻类（三级分类为新闻类）	5条	不限量。
	其他应用类型	2条	不限量。
推送数量限制		所有公信类通道共享推送次数，当日达到次数限制后，所有公信类通道将不能再推送消息，目前单日推送数量为：累计注册用户数 * 2 。	不限量。

额度查询指引

管理台查询：累计用户数在 OPPO 推送运营平台 可查询，数据每天刷新。

API 查询：请参见 OPPO PUSH 服务端 API。

小米平台限制

限额说明

公信消息（默认的多用户普适性消息）的单日可推送数量总量：为应用在MIUI上安装且通知开启数x倍数。默认倍数为2倍，具备《互联网新闻信息服务许可证》的应用为3倍，具体如表1所示。通知开启数小于10000的按10000计数。

私信消息（单个用户的私人消息）的推送数量不受限制，详情请参见 小米通知渠道介绍。

公信消息限制倍数,于2023年2月1日生效，详情请参见 小米推送消息限制说明。

是否具备《互联网新闻信息服务许可证》	单个应用单日单设备通知推送数量限制倍数（单位：倍）	单个设备单日单应用接收通知数量（单位：条）
有	3	8
无	2	5

说明：

若厂商通道推送量超过当日限制，超量后的推送任务将会通过移动推送自建通道补发。

额度查询指引

管理台查询：在小米开放平台 > **推送运营平台** > **推送统计** > **用户数据** > **数据详情**，可查询 MIUI 日联网设备数。

API 查询：请参见 小米推送消息限制说明 查询当日可下发总量和当日已送达数。

魅族平台限制

限额说明

单个业务的推送有速率限制，默认 App 为500条/秒。
单个业务每天的推送有次数限制，默认为1000次/天。
单个业务订阅标签的个数不超过100个。
单个设备单个业务推送消息 ≥ 4条会被折叠展示，消息多次不点击后有可能会被收纳于右上角消息收纳盒。
单个设备1个月内不活跃，将取消订阅。
一个 IP 地址每小时请求 API 接口有次数限制，具体次数魅族官方未给出说明。
单个业务每天累计请求 API 接口有次数限制，具体次数魅族官方未给出说明。
单个业务每天推送的消息总量有限制，具体次数魅族官方未给出说明。

华为平台限制

限额说明

发送条数限制：资讯营销类消息的每日推送数量自2023年01月05日起根据应用类型对推送数量进行上限管理，服务与通讯类消息每日推送数量不受限，详情请参见华为的 推送数量管理细则。
发送速率限制：华为推送对推送速度的分配，主要依据 App 在华为渠道的月活、App 在华为应用市场上架时的应用类型这两个要素进行计算分配。

荣耀平台限制

限额说明

发送条数限制：根据消息分类标准，荣耀推送服务将通知消息分为资讯营销、服务与通讯两大类别。资讯营销类消息的每日推送数量根据应用类型对推送数量进行上限管理，服务与通讯类消息每日推送数量不受限。不同应用类别的推送数量上限要求，详情请参见 荣耀的推送数量管理细则。

厂商通道QPS限制说明

最近更新时间：2024-01-16 17:39:39

各个手机厂商的推送通道在发送速率上有一定的限制，如目前速率无法满足您的运营需求，可参考以下说明和申请流程向厂商申请调整 QPS。(仅部分厂商可申请)

华为推送

QPS 限制规则

发送速率（QPS）= App 在华为渠道 MAU x 应用类别权重系数 x 0.00072

名词解释

MAU：在华为渠道推送应用的每月最后一个自然日的值作为当月的 MAU。
分类规则：依据在华为应用市场上架的应用分类。

分组名称	应用分类	权重系数
IM类	社交通讯	5
金融类	金融理财	5
新闻类	新闻阅读；资讯生活	4
内容类	图书阅读；影音娱乐；拍摄美化	3
电商类	购物比价	3
衣食住行类	便捷生活；出行导航；美食；旅游住宿	3
商务类	商务	3
游戏类	网络游戏；休闲益智；经营策略；棋牌	2
工具类	实用工具	1
运动健康类	医疗健康；运动健康	1
其他类	儿童；教育；主题个性；汽车	1
Default	Default	1

说明：

如果没在华为应用市场发布，按 Default 分类。
如您的应用通过 QPS 公式计算所得的值不足6000，将执行默认6000的 QPS。
如全网流量较高时，也会出现系统级流控。
另外，无论何种类别的应用，针对单个设备每天不能推送超过10万条/天，否则将进行推送权益限制，需要整改并申报整改方案重新申请 push 权益。

QPS 提升申请

如您对产品有任何相关疑问和建议，请反馈至 `hwpush@huawei.com`，格式如下：**标题：【QPS】咨询及建议+应用名称**

华为消息推送 QPS 咨询及建议表	
应用名称：	-
企业名称：	-
应用包名：	-
在应用市场应用分类：	-
问题类型：	申请 QPS 提升、QPS 具体值咨询描述等。
需求背景：	-
具体诉求：	-
联系方式（邮箱）	-

小米推送

QPS 限制规则

小米推送对推送速率（QPS）的分配主要依据 App 的 MIUI 日联网设备数进行分级计算。
QPS：表示1秒可调用的请求数。1个请求里最多可以携带1000个目标设备。例如：3000 QPS时，1秒内最多可推送300万设备。
说明：
MIUI 日联网设备数的查询路径：推送运营平台 - 推送统计 - 用户数据 - 数据详情。
不同量级的 MIUI 日联网设备数分配不同的QPS：

MIUI 日联网设备数	QPS
≥1000万	3000
≥500万且<1000万	2500

≥100万且<500万	2000
≥10万且<100万	1000
<10万	500

QPS 提升申请

暂不开放申请。

OPPO 推送

QPS 限制规则

OPPO 推送对 QPS 的分配主要依据应用的累计用户数，应用类别权重和平台推送系数三个值进行加权计算，其中累计用户数在 OPUSH 平台上的查询路径是 OPPO PUSH 推送运营平台—我要推送消息—应用列表中的累计用户数。

计算公式

应用 QPS = 推送 QPS 参考值 × 应用类别权重 × 平台推送系数

累计用户数	QPS	应用类别权重	平台推送系数(默认 = 1)
≥10000万	30000	1	1
≥5000万且<10000万	20000	1	1
≥1000万且<5000万	10000	1	1
<1000万	5000	1	1

QPS 提升申请

暂不开放申请。

vivo 推送

QPS 限制规则

QPS：推送 QPS 根据 SDK 订阅数自动调整，默认最低值为500条/秒。

QPS 提升申请

暂不开放申请。

魅族推送

QPS 限制规则

QPS：默认 App 是 500 条/秒，如果超过此速率可以联系魅族进行调整。

QPS 提升申请

可以联系魅族进行调整 QPS。
推送服务咨询邮箱： `push_support@meizu.com` 。

错误码

最近更新时间：2024-01-16 17:39:39

客户端返回码

错误码	原因以及解决办法
0	调用成功
2	参数错误，例如，已绑定单字符的别名，或是 iOS 的 Token 长度不正确，应为64个字符
-1	SDK 内部错误，请保存日志 联系我们
-2	等待服务器回包超时
-3	资源已经被销毁，需要重新购买服务，并且需要升级 App 版本（即：App 版本号需要比之前的版本号大）才可恢复
-4	连接超出套餐限制，连接会随机拒绝，需要升级服务即可恢复
-5	获取 Guid 出错，请检查网络，以及 AccessId 和 AccessKey 是否正确，资源是否已经购买
-7	发送请求包出错，请检查网络，或保存日志 联系我们
-11	连接 MQTT 服务器失败，请检查网络
-101	SDK 出现某些内容 JSON 格式错误，请保存日志 联系我们
-102	获取不到 Token，检查网络和 App 配置
-502	获取 Guid 出错，请检查网络，以及服务接入点域名配置是否正确，详情参见 SDK 集成文档
-701	发送请求包时出现网络异常
-702	发送请求包超时
-1101	连接 MQTT 服务器出现网络异常，请检查网络
-1102	连接 MQTT 服务器出现异常，请检查网络或 联系我们
-1103	连接 MQTT 服务器超时，请检查网络
-10005	AIDL 配置出错
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，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	没有配置正确的 ContentProvider，请检查 AndroidManifest 文件
10009	jce JAR 错误或者缺少 jce JAR（检查是否已将 wup 包编译进去了，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，请记录错误日志并 联系我们
10006	AccessKey 或者 AccessID 错误
10007	初始化移动推送 Service 错误
10008	AccessKey 或者 AccessID 错误
10110	认证过程错误
10115	短时间内重复注册

10300	跑马策略相关返回码
10400	SDK 参数错误
20002	无有效的网络连接，请确认该应用是否付费
10030009	应用不存在，SDK 集成时需要根据接入的服务接入点配置域名，详情参见 SDK 集成文档

服务端返回码

错误码	含义
1010001	资源未部署，请确认应用是否已购买推送资源
1008001	参数解析错误
1008002	必填参数缺失
1008003	认证失败
1008004	调用服务失败
1008006	Token 无效，请检查设备 Token 是否注册成功
1008007	参数校验失败
1008011	文件上传失败
1008012	上传文件为空
1008013	证书解析失败
1008015	推送任务 ID 不存在
1008016	日期时间参数格式不对
1008019	被内容安全服务判定不和谐
1008020	证书包名校验失败
1008021	p12 证书格式内容校验失败
1008022	p12 证书密码不对
1008025	创建应用失败，产品下已存在该平台的应用
1008026	批量操作，部分失败

1008027	批量操作，全部失败
1008028	超出限频
1008029	Token 校验非法
1008030	App 未付费
1008031	App 资源已销毁
10110008	查询的 Token ， 账号不存在
10010005	推送目标不存在
10010012	非法的推送时间，请更改推送时间。 定时推送时 send_time 传入的值如果是过去的时间， 具体规则如下： 如果 | send_time - 当前时间 | <= 10min， 推送任务会创建，接收任务后会立即调度 如果 | send_time - 当前时间 | > 10min， 推送任务会被拒绝， 接口返回失败
10010018	重复推送
10030002	AccessID 和 AccessKey 不匹配

iOS 接入指南

简介

最近更新时间：2024-01-16 17:42:20

iOS 端实现推送消息的服务涉及到三个角色：终端应用（Client App），APNs（Apple Push Notification service），移动推送 服务器（移动推送Provider）。在使用移动推送 服务实现给客户端推送消息，需要这三个角色在整个流程中相互配合，任何一个角色出现异常，都可能会导致消息推送无法收到。

SDK 说明

文件组成

`XGPush.h`，`XGPushPrivate.h`（SDK 提供接口的头文件）
`libXG-SDK-Cloud.a`（主 SDK 文件）
`libXGExtension.a`、`XGExtension.h`（“抵达和富媒体”扩展插件库及接口头文件）
`XGMTACloud.framework`（“点击上报”组件）
`XGInAppMessage.framework`（应用内消息）

版本说明

支持 iOS 8.0+。
针对 iOS 10.0+ 以上版本。
需要额外引入 UserNotification.framework。
建议使用 Xcode 8.0 +。
如果使用 Xcode7 及其以下的版本，需要自行配置 iOS SDK 来支持 UserNotification 框架的编译。

功能说明

iOS SDK 是移动推送 服务为客户端实现消息推送而提供给开发者的接口，主要负责完成：
设备 Token 的自动化获取和注册，降低接入门槛。
账号、标签与设备的绑定接口，以便开发者实现特定群组的消息推送，丰富推送方式。
点击量上报，统计消息被用户点击的次数。

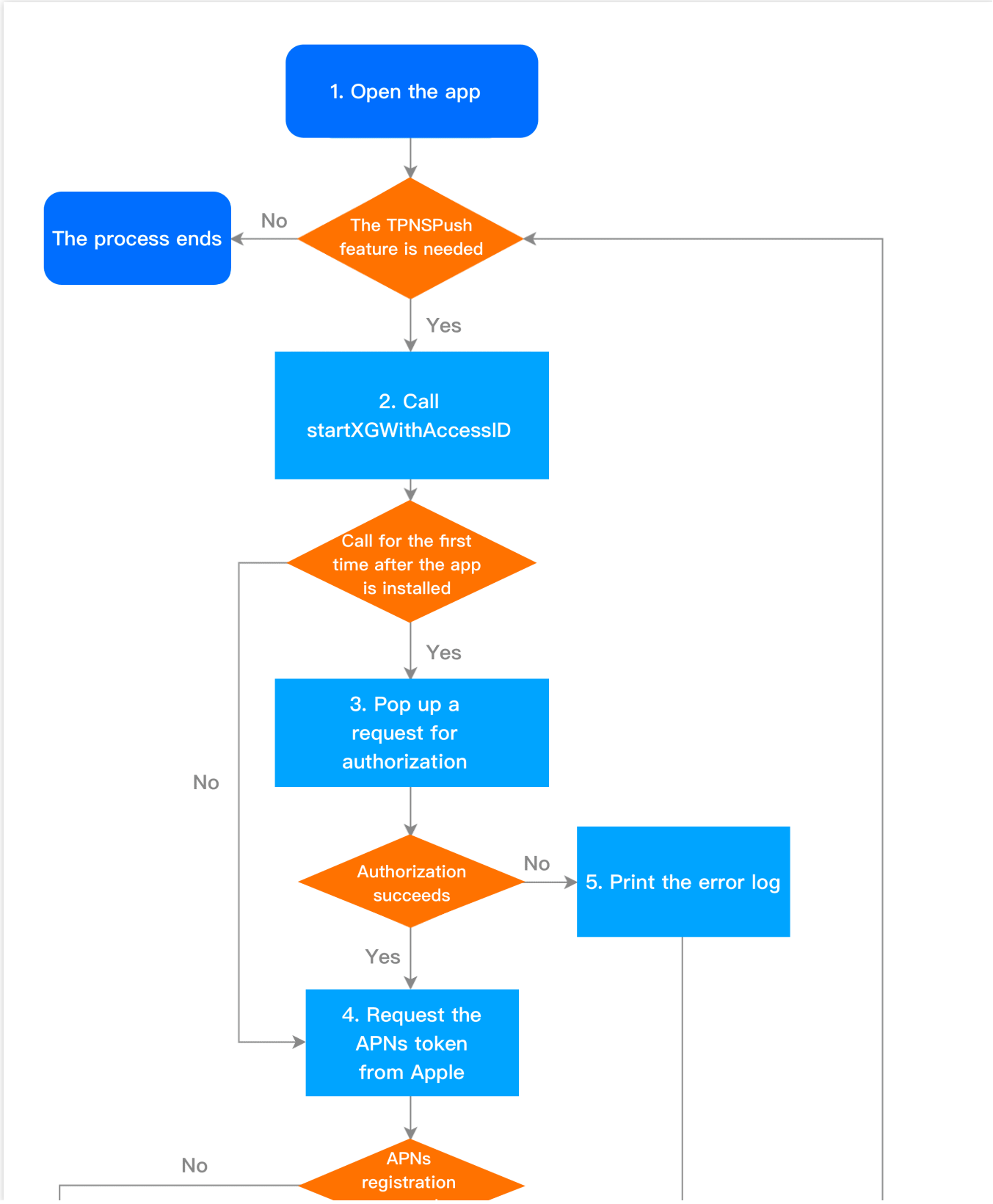
推送通道

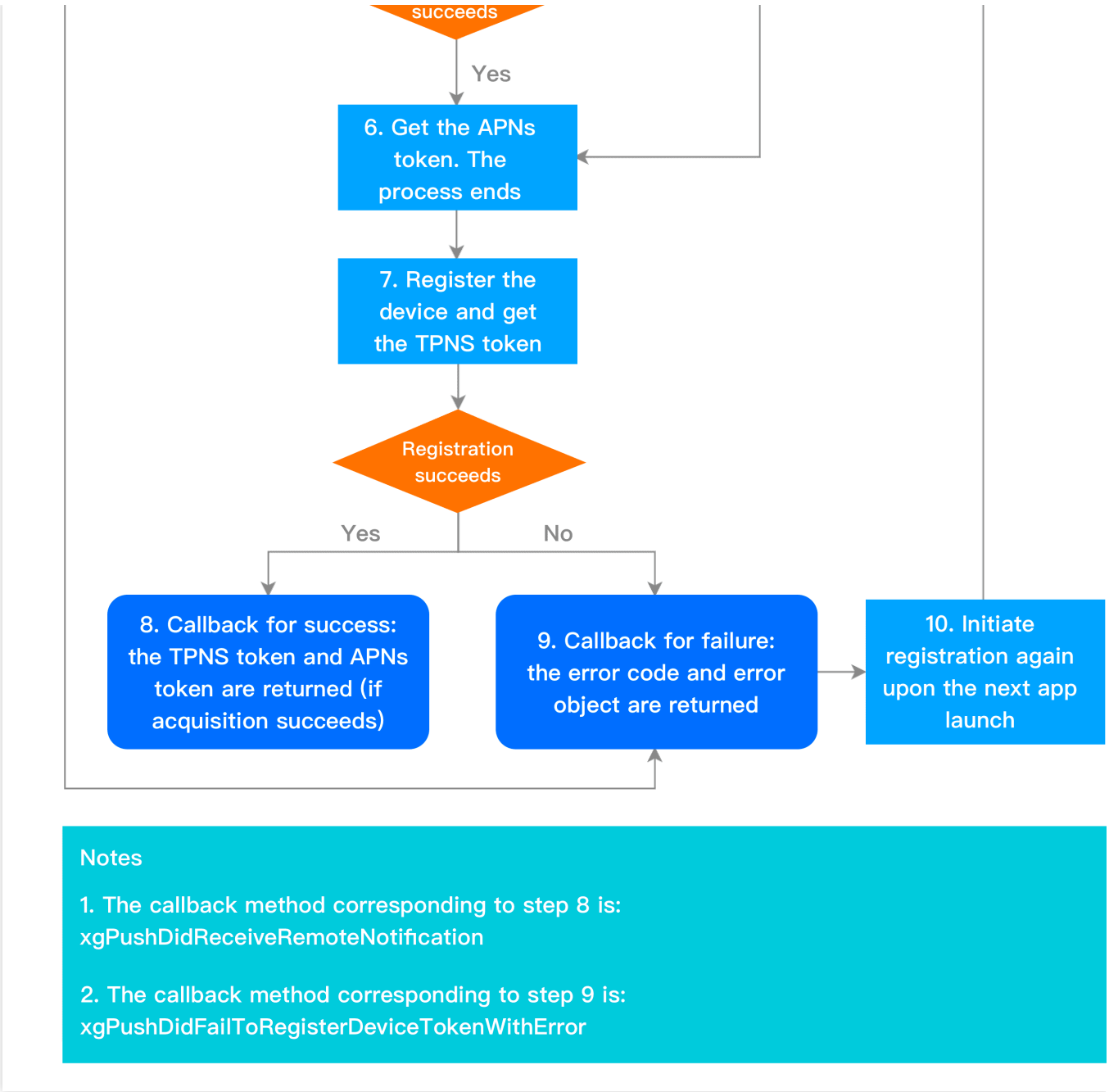
移动推送 使用的消息下发通道：
移动推送在线通道：移动推送在线通道是移动推送 的自建通道，依赖移动推送 Service 在线（与移动推送 后台服务器保持长连接）才能下发消息；需要 SDK 1.2.8.0 及以上版本支持。
APNs 通道：苹果官方提供的消息推送服务，详情请参见 APNs。

常用场景流程说明

设备注册流程

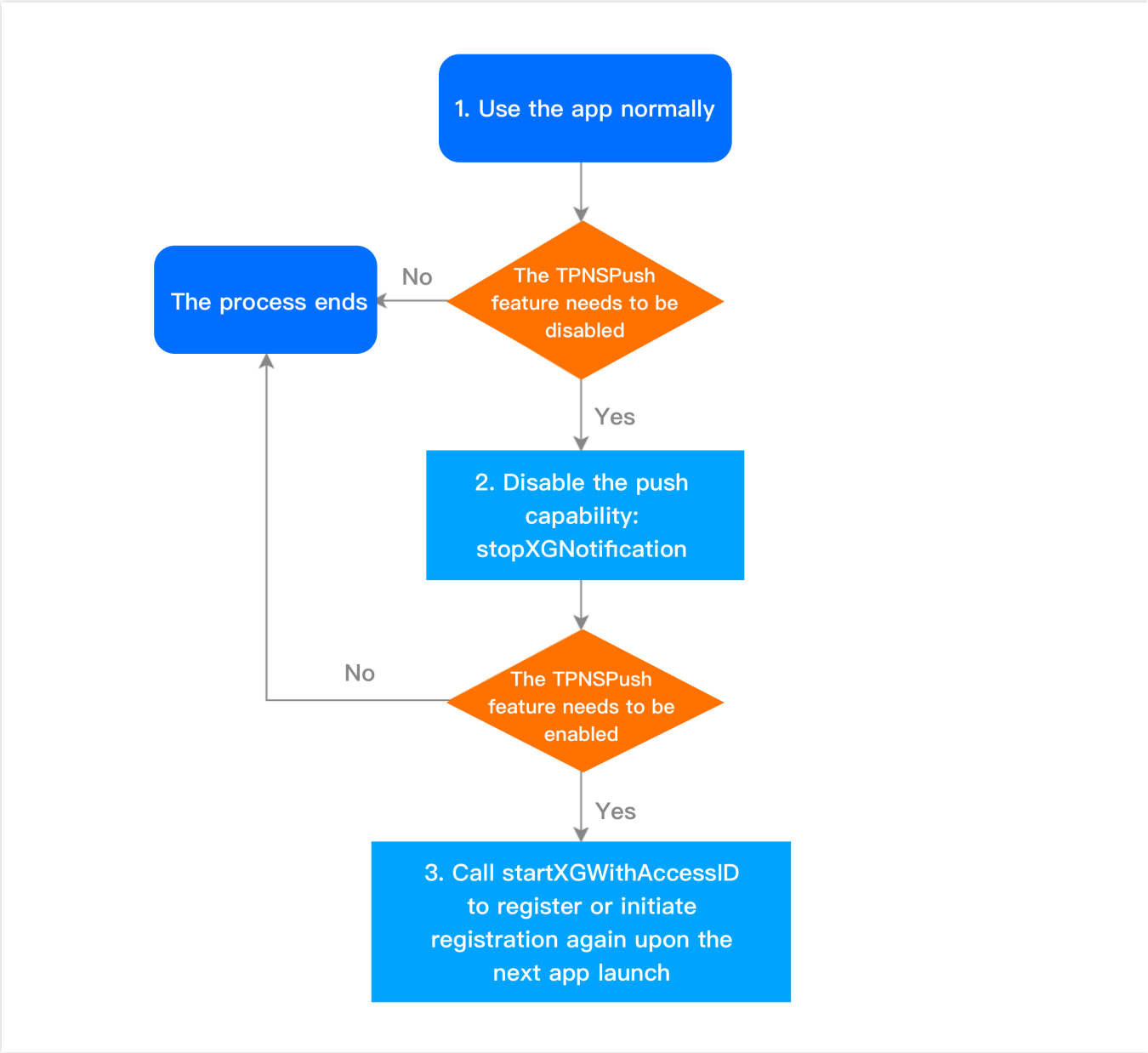
下图为设备注册相关流程，具体接口方法请查看 启动腾讯移动推送服务。





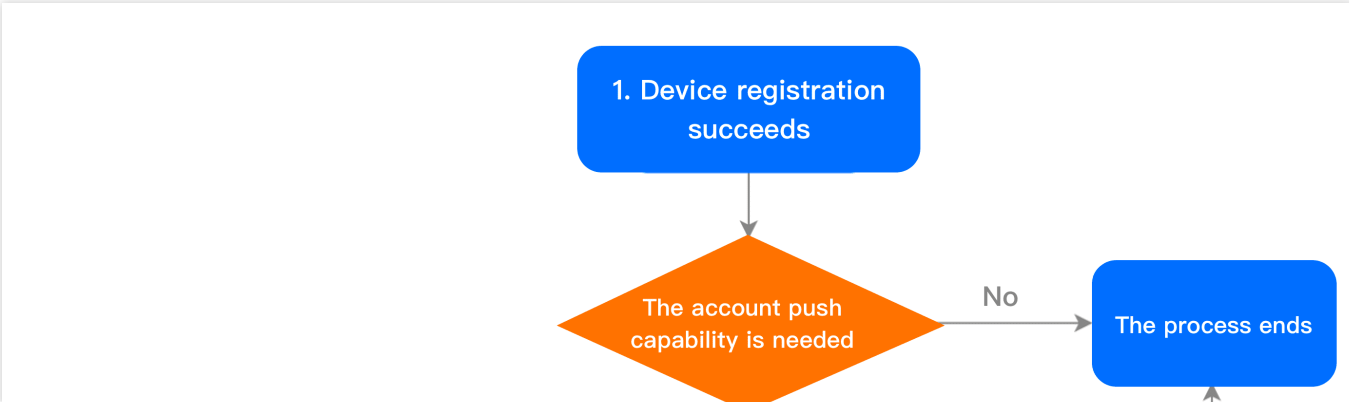
设备反注册流程

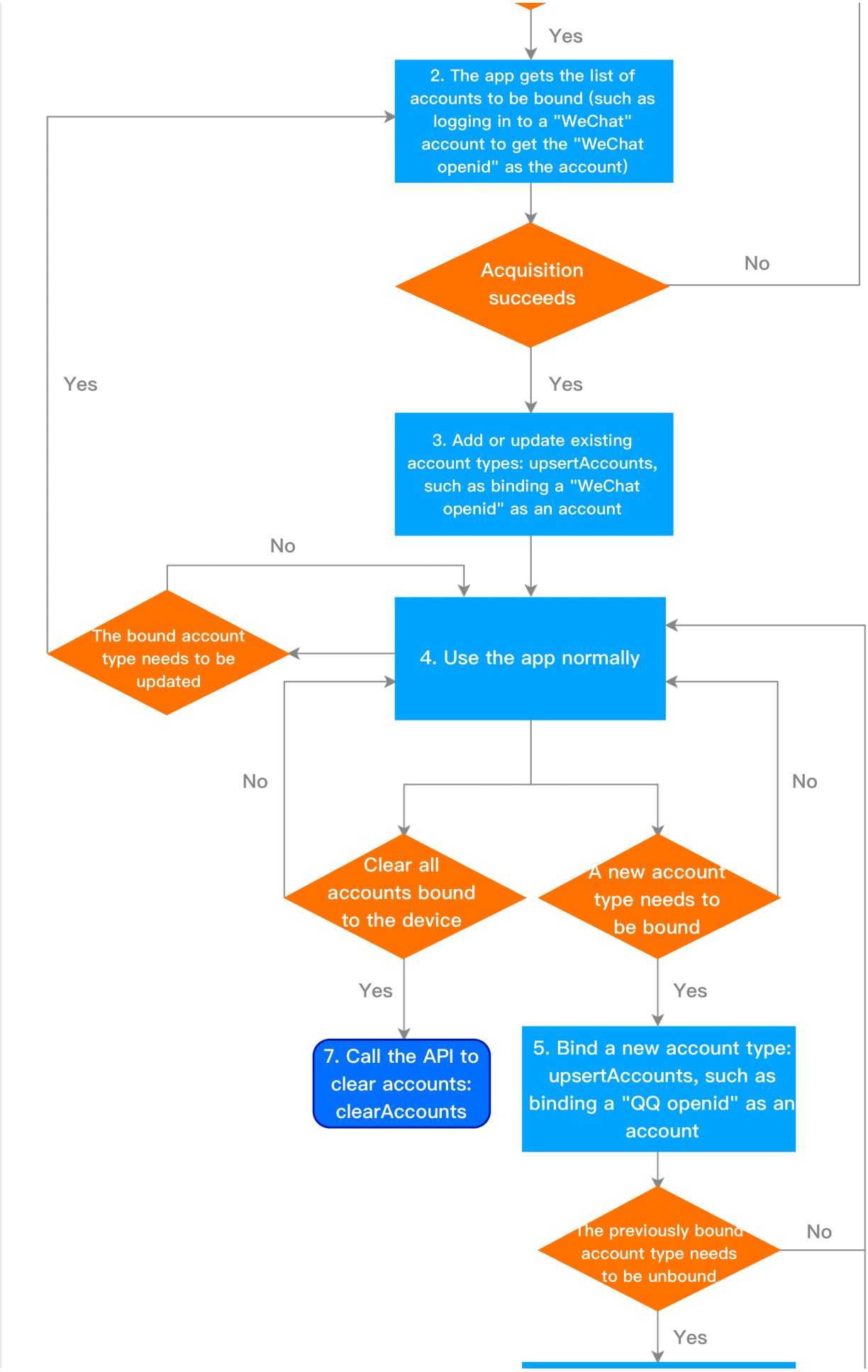
下图为设备反注册相关流程，具体接口方法请查看 终止腾讯移动推送服务。

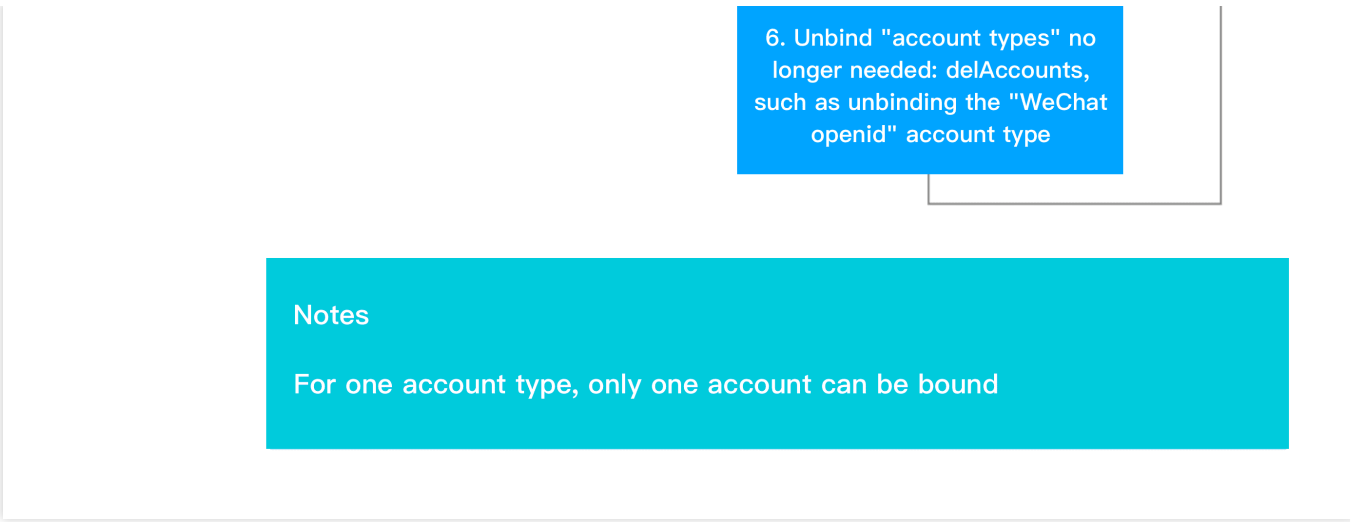


账号相关流程

下图为账号相关流程，具体接口方法请查看 账号管理。

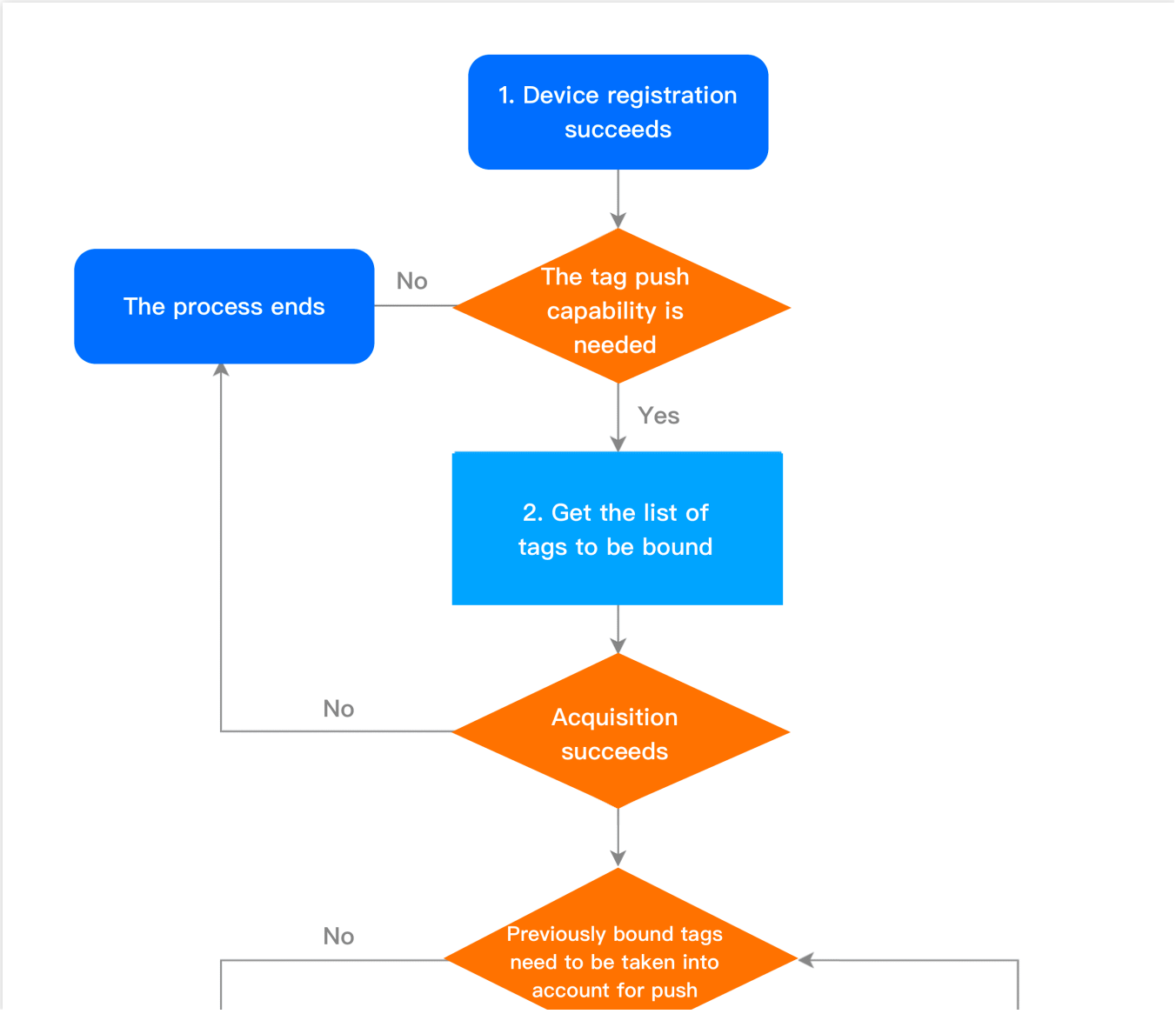


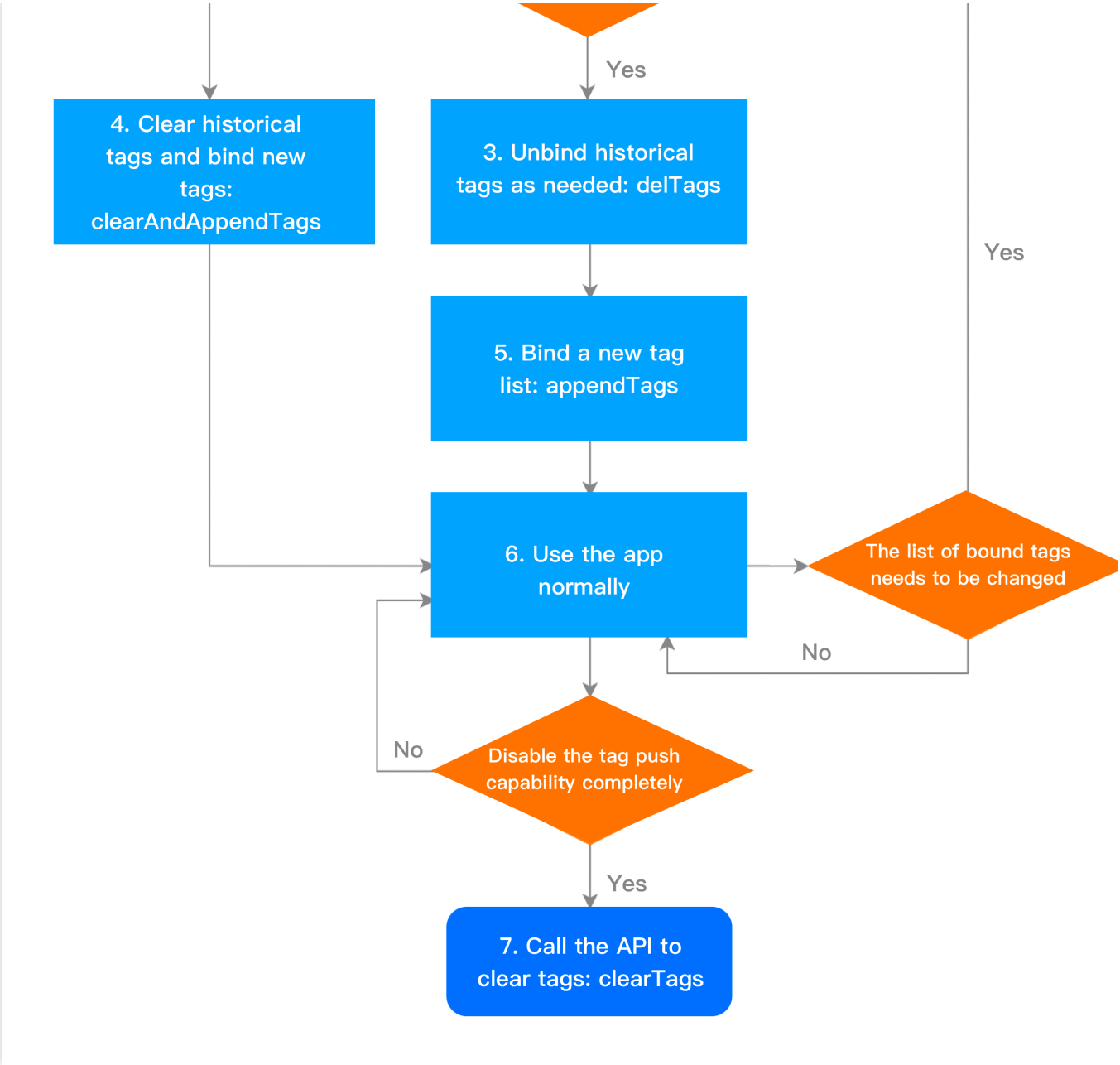

Yes
2. The app gets the list of accounts to be bound (such as logging in to a "WeChat" account to get the "WeChat openid" as the account)
Acquisition succeeds
No
Yes
Yes
3. Add or update existing account types: upsertAccounts, such as binding a "WeChat openid" as an account
No
The bound account type needs to be updated
4. Use the app normally
No
No
Clear all accounts bound to the device
A new account type needs to be bound
Yes
Yes
7. Call the API to clear accounts: clearAccounts
5. Bind a new account type: upsertAccounts, such as binding a "QQ openid" as an account
The previously bound account type needs to be unbound
No
Yes



标签相关流程

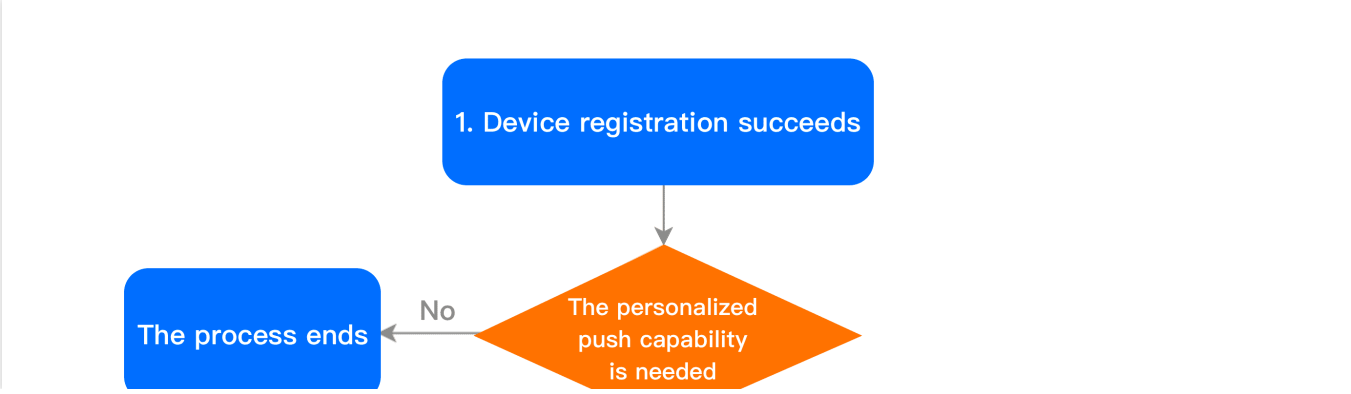
下图为标签相关流程，具体接口方法请查看 标签管理。

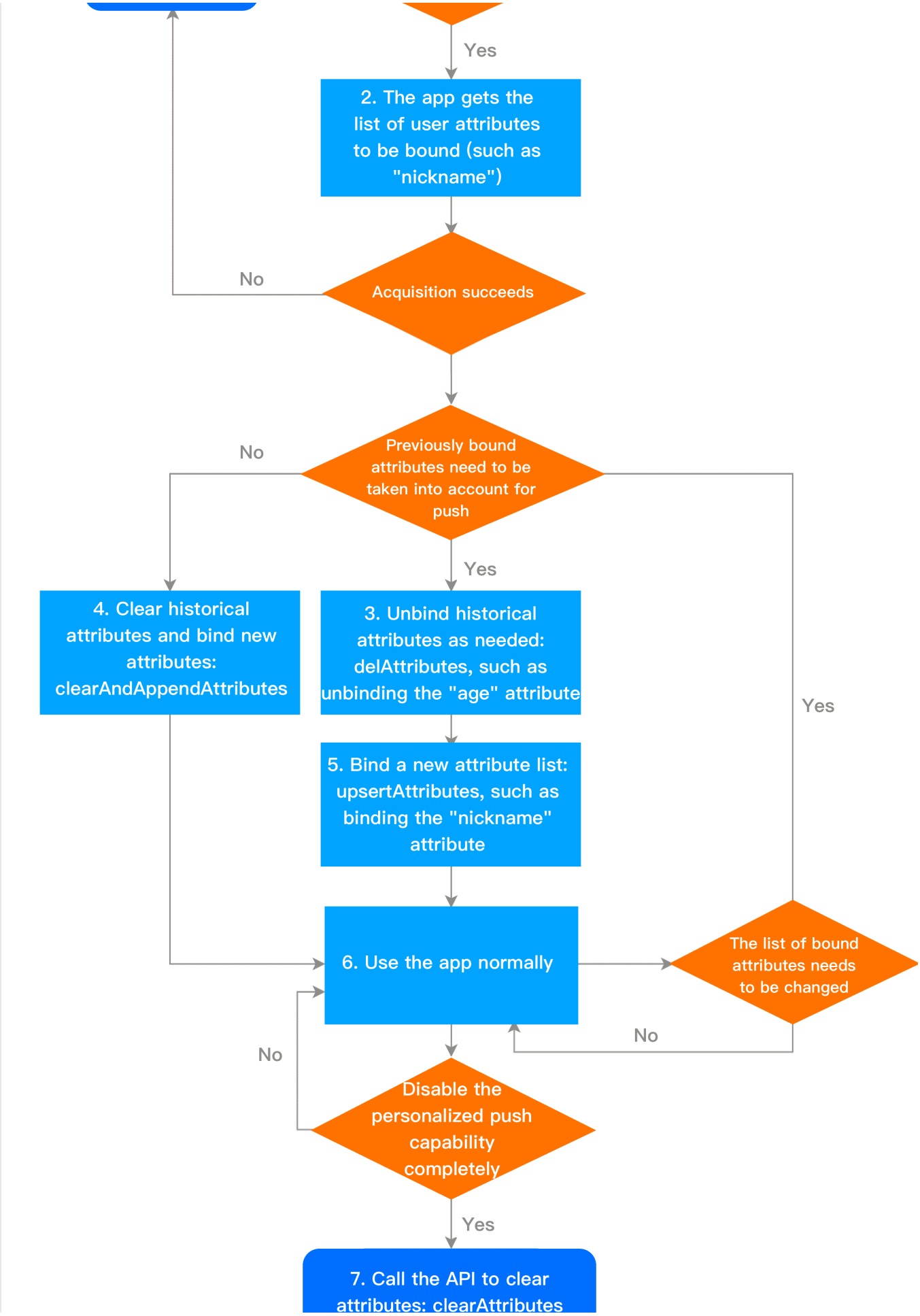




用户属性相关流程

下图为用户属性相关流程，具体接口方法请查看 用户属性管理。




Yes
2. The app gets the list of user attributes to be bound (such as "nickname")
No
Acquisition succeeds
No
Previously bound attributes need to be taken into account for push
Yes
4. Clear historical attributes and bind new attributes: clearAndAppendAttributes
3. Unbind historical attributes as needed: delAttributes, such as unbinding the "age" attribute
Yes
5. Bind a new attribute list: upsertAttributes, such as binding the "nickname" attribute
6. Use the app normally
The list of bound attributes needs to be changed
No
No
Disable the personalized push capability completely
Yes
7. Call the API to clear attributes: clearAttributes

Notes

Supported user attributes need to be configured in the web console or through RESTful APIs first

SDK 集成

最近更新时间：2024-01-16 17:42:20

简介

本文档提供关于 SDK 接入以及开启推送服务的示例代码（SDK 版本：V1.0+ 版本）。

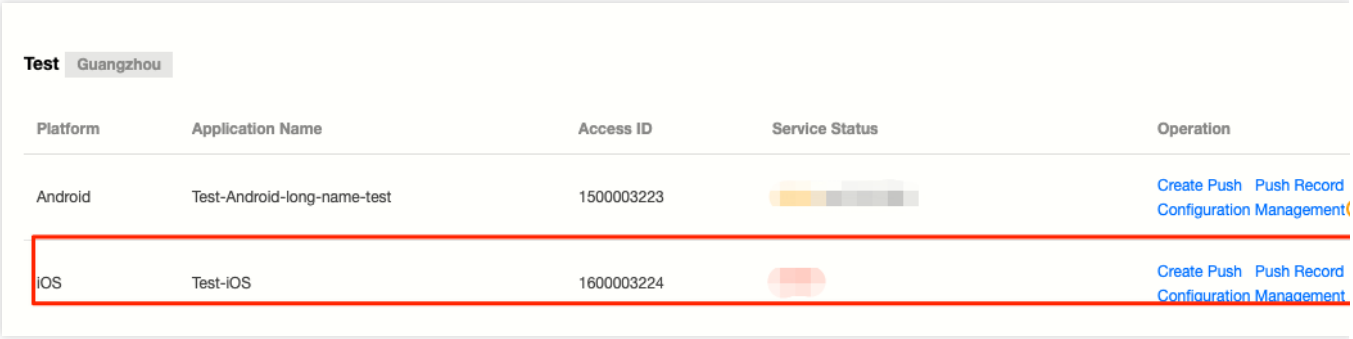
SDK 组成

doc 文件夹：腾讯移动推送 iOS SDK 开发指南。
demo 文件夹：包含样例工程，腾讯移动推送 SDK（仅包含 OC demo，Swift Demo 请前往 腾讯工蜂 进行下载）。

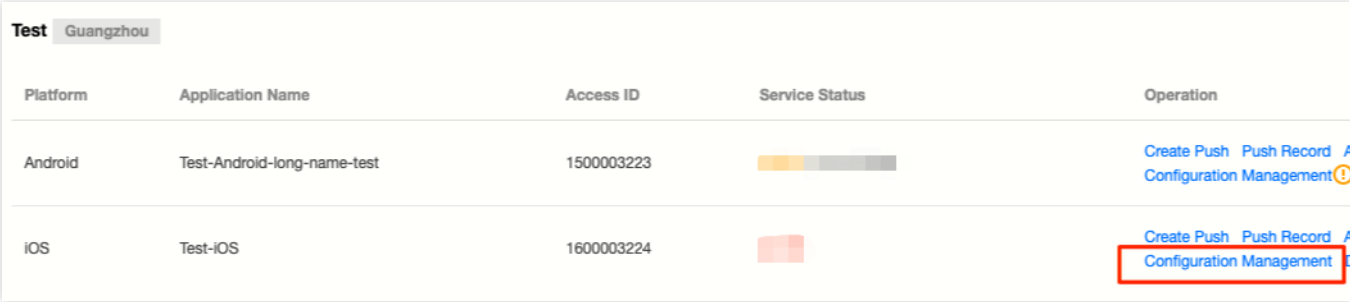
SDK 集成

接入前准备

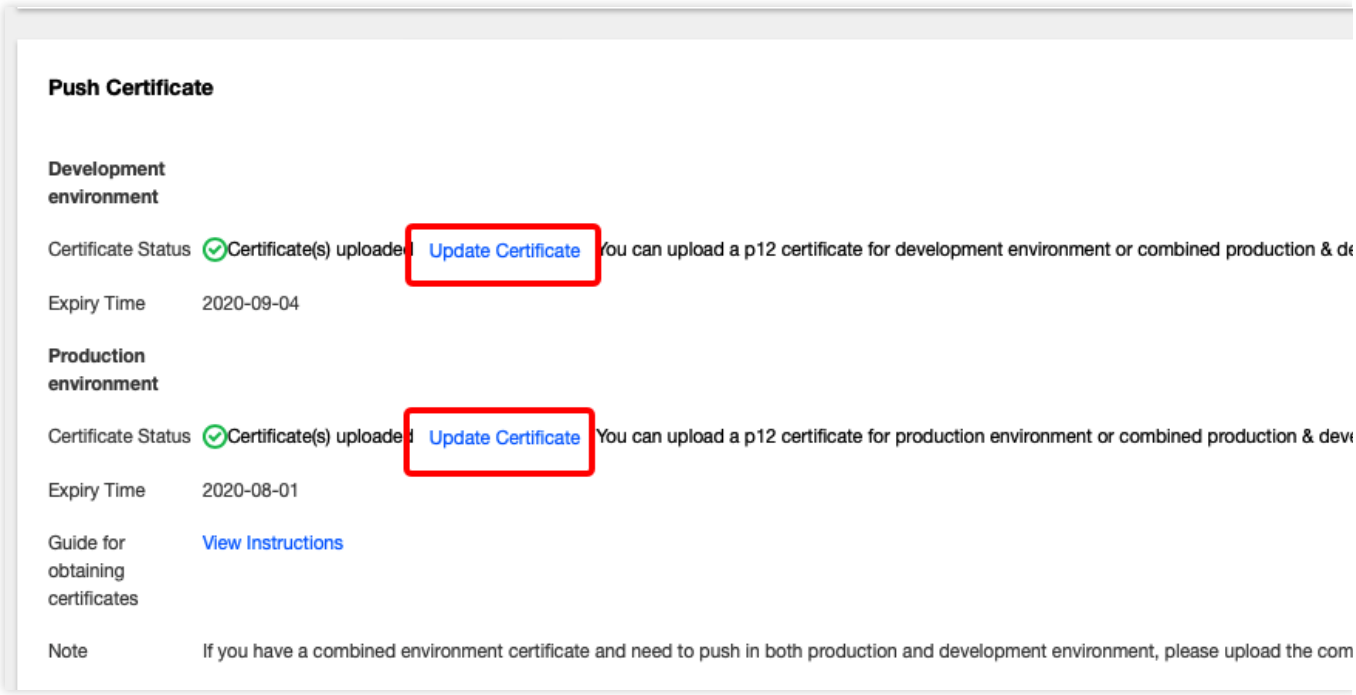
1. 接入 SDK 之前，请前往移动推送控制台 创建产品和 iOS 应用，详细操作可参考 创建产品和应用。



2. 单击**配置管理**，进入管理页面。



3. 单击**上传证书**，完成上传操作。推送证书获取详情请参考 证书获取指引。

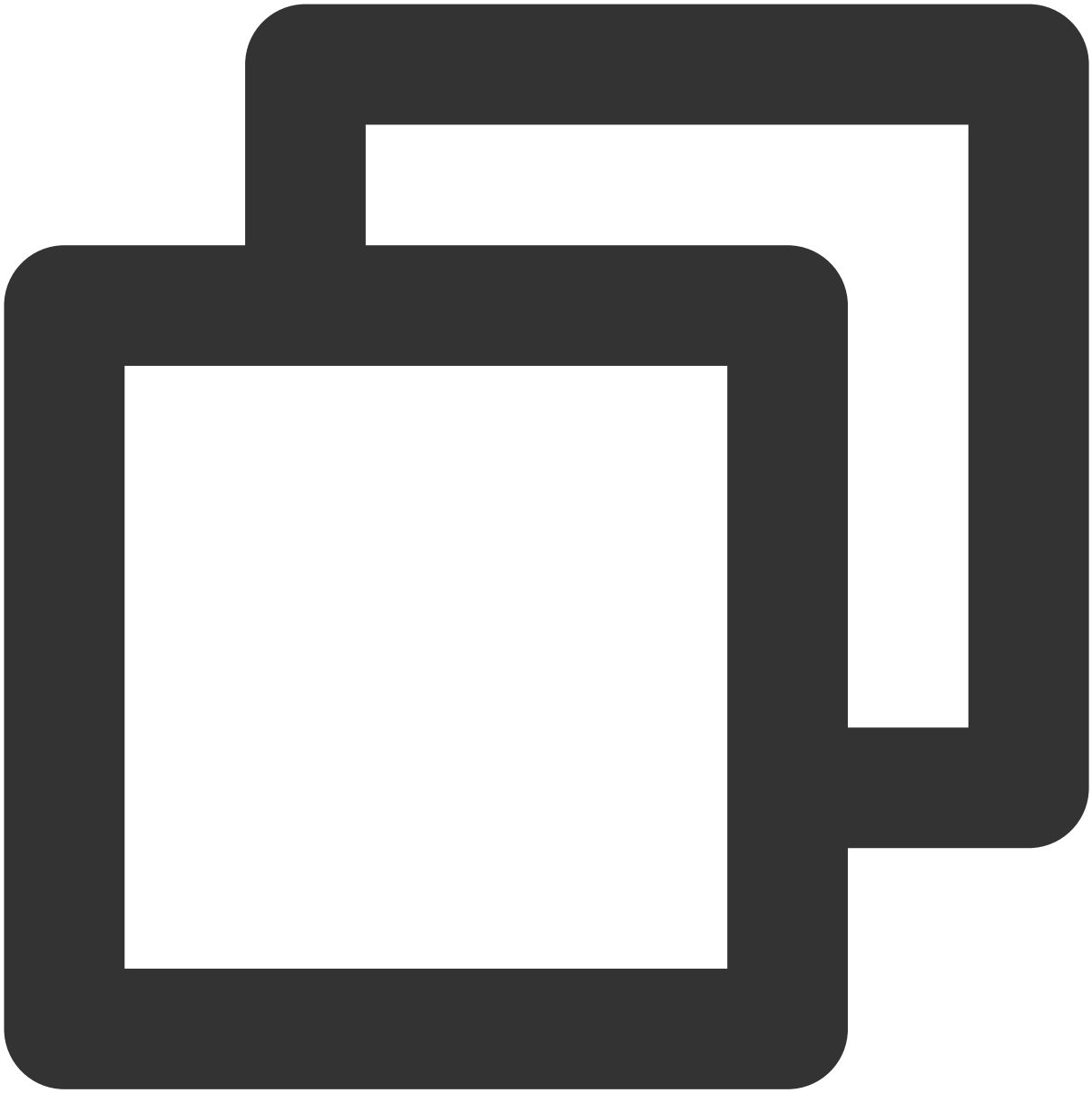


4. 证书上传成功后，在应用信息栏中，获取应用 Access ID 和 Access KEY。

导入 SDK（二选一）

方式一：Cocoapods 导入

通过 Cocoapods 下载地址：



```
pod 'TPNS-iOS', '~> 版本'  // 如果不指定版本则默认为本地 pod TPNS-iOS 最新版本
```

说明：

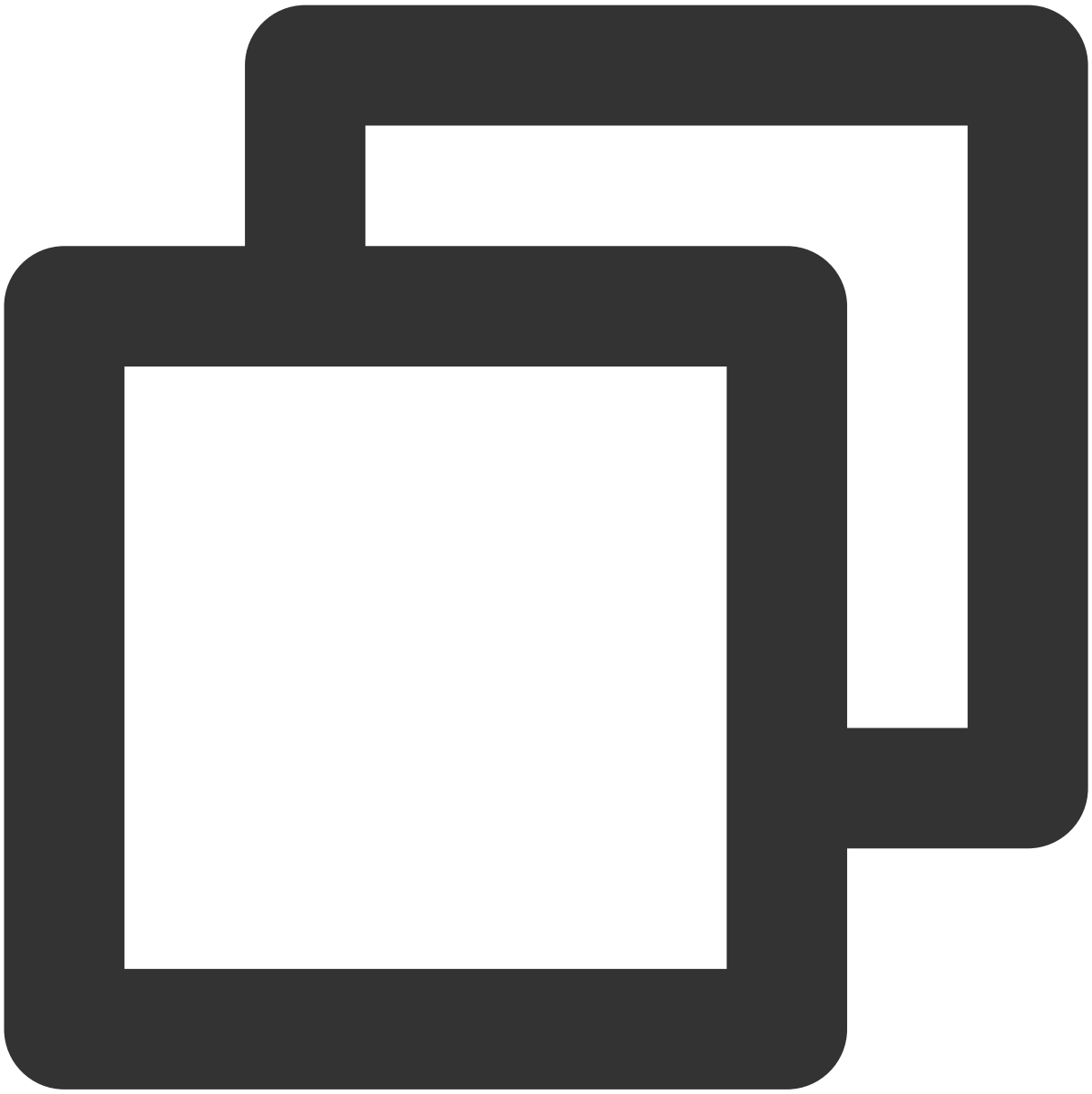
首次下载需要登录 仓库地址，并在【账户】菜单栏中 设置用户名和密码。设置成功后，在 Terminal 输入对应的用户名和密码，后续即可正常使用，当前 PC 不需要再次登录。

由于仓库地址变更，如果 pod 提示 `Unable to find a specification for 'TPNS-iOS'`，那么需要执行以下命令，并更新仓库确认版本：

```
pod repo update
```

pod search TPNS-iOS

pod install //安装 SDK



```
#### 方式二：手动导入
1. 进入腾讯移动推送 [控制台](https://console.tencentcloud.com/tpns)，单击左侧菜单栏【[SDK
2. 打开 demo 目录下的 SDK 文件夹，将 XGPush.h 及 libXG-SDK-Cloud.a 添加到工程，打开 XGPush
3. 将 InAppMessage 文件夹导入到工程并在【Build Setting】>【Framework Search Paths】 添加至
4. 在 Build Phases 下，添加以下 Framework：
```

XGInAppMessage.framework

XGMTACloud.framework

CoreTelephony.framework

SystemConfiguration.framework

UserNotifications.framework

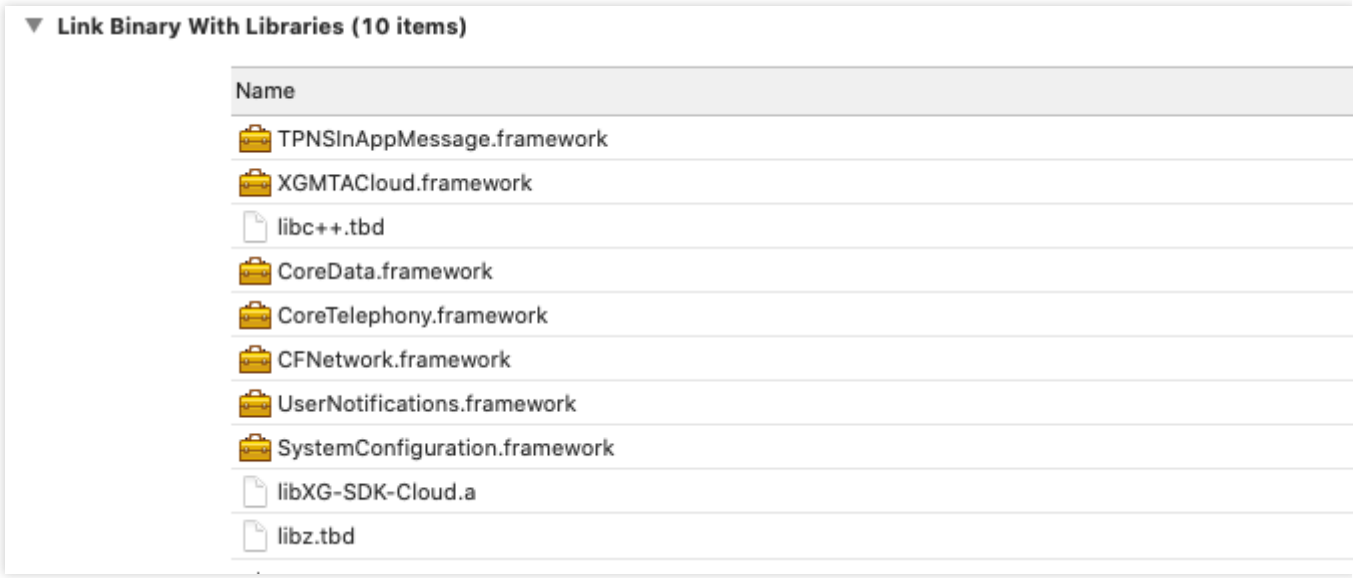
libXG-SDK-Cloud.a

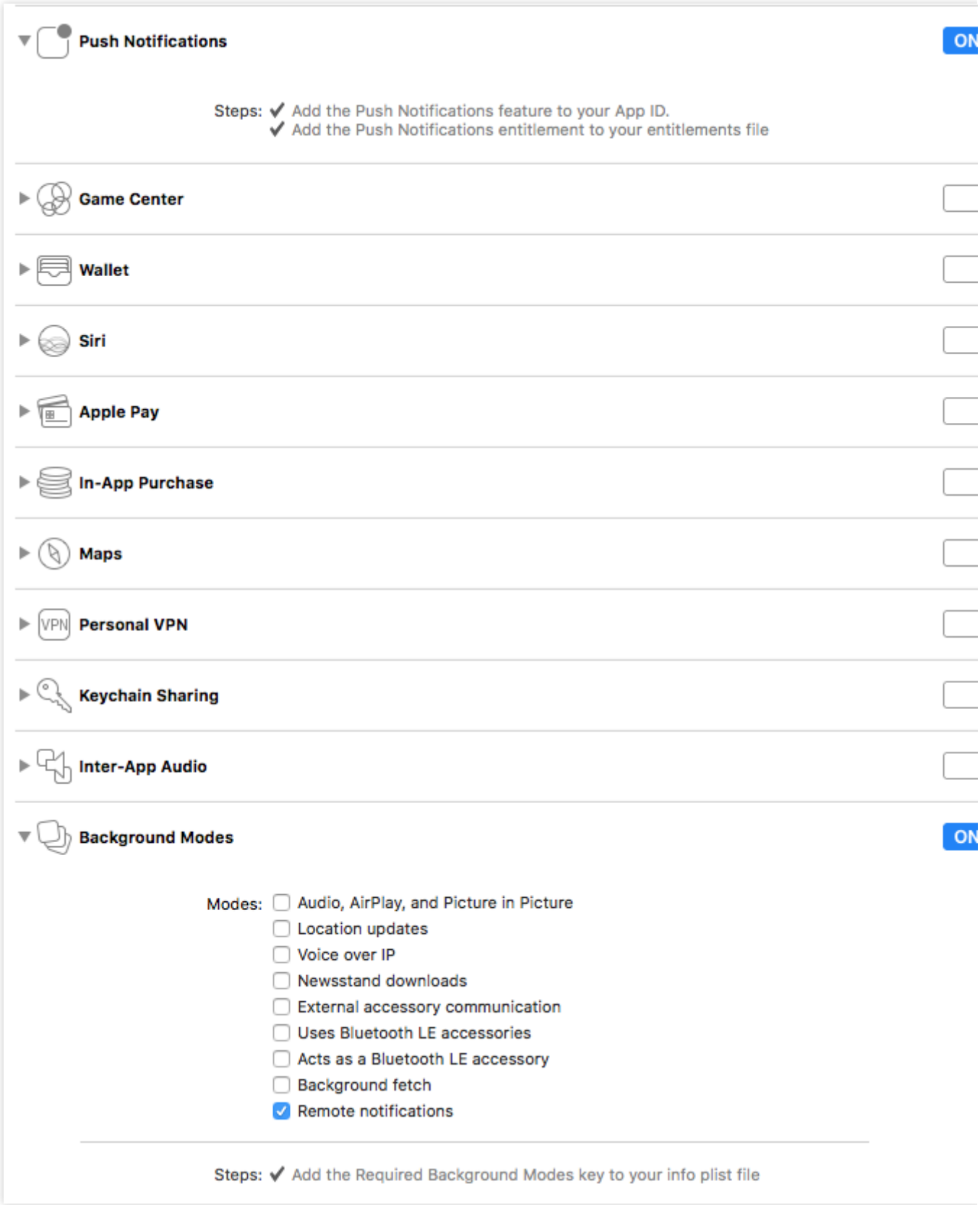
libz.tbd

CoreData.framework

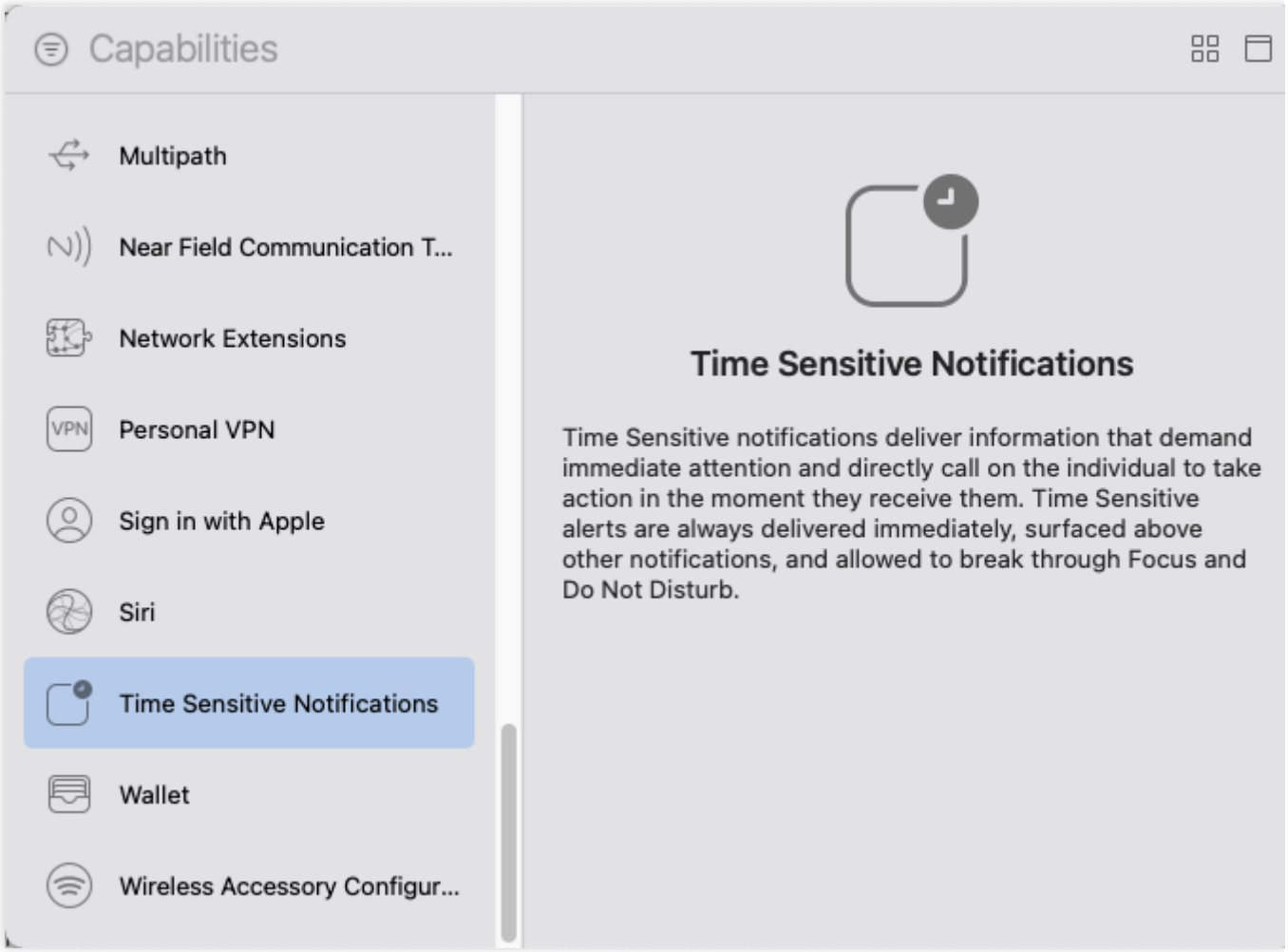
CFNetwork.framework

libc++.tbd

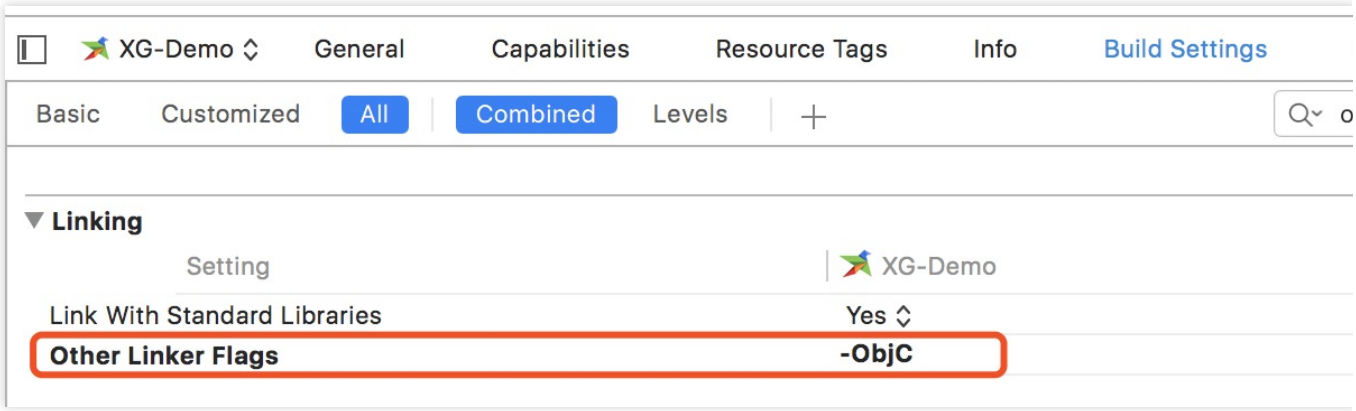
```

1. 添加完成后，库的引用如下：




### 工程配置

1. 在工程配置和后台模式中打开推送，如下图所示：
```



1.1 如需使用 iOS15 新增的"时效性通知功能"，请在 `Capabilities` 中开启 `Time Sensitive Notifications`



2. 添加编译参数 `-ObjC` 。



如 checkTargetOtherLinkFlagForObjc 报错，是因为 build setting 中，Other link flags 未添加 -ObjC。

注意：

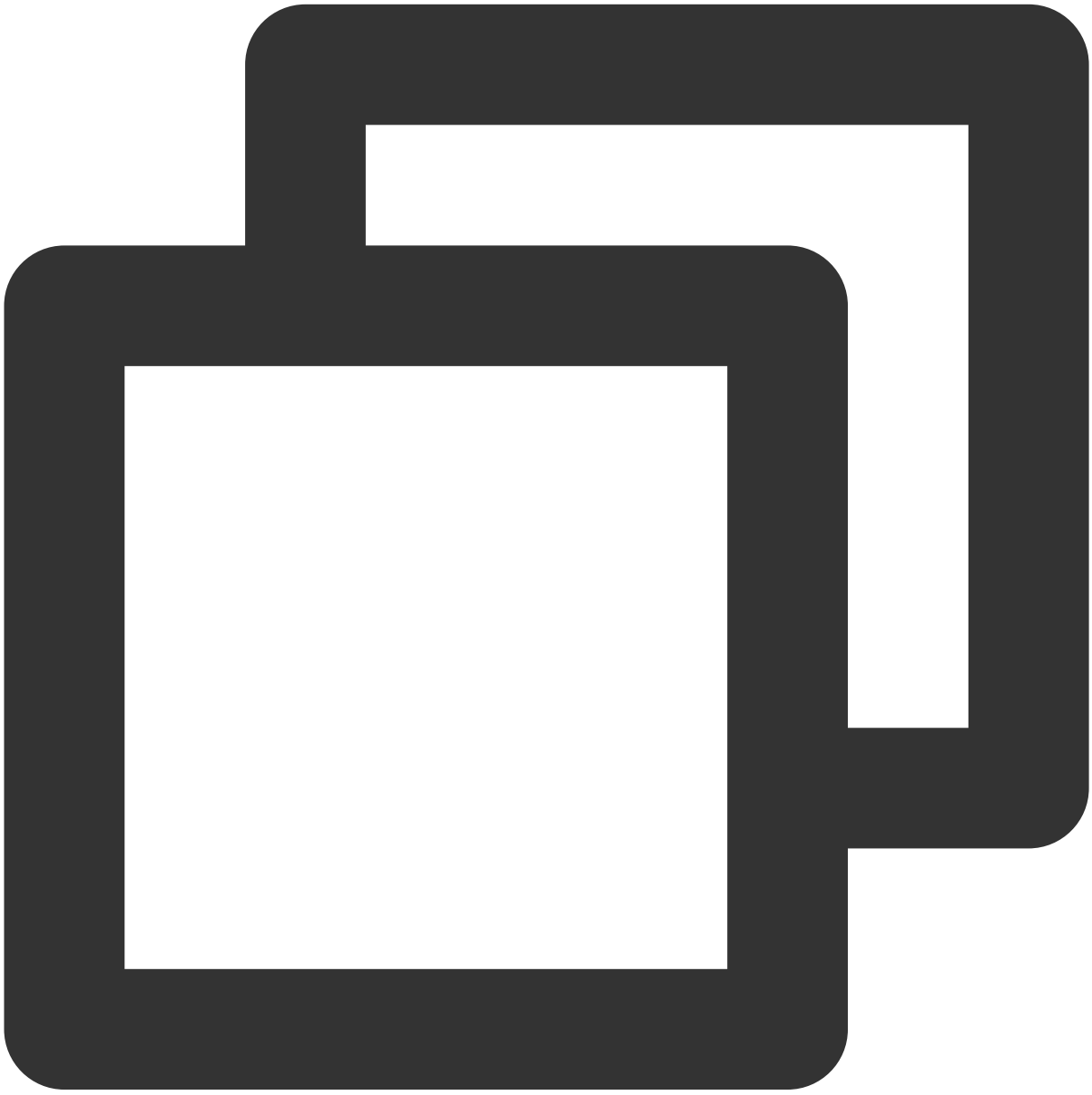
如果您的应用服务接入点为广州，SDK 默认实现该配置，广州域名为 `tpns.tencent.com` 。

如果您的应用服务接入点为上海、新加坡或者中国香港，请按照下文步骤完成其他服务接入点域名配置。

1. 解压 SDK 文件包，将 SDK 目录下的 XGPushPrivate.h 文件添加到工程中并在需要配置域名的类中引用(#import "XGPushPrivate.h")。
2. 在 `startXGWithAccessID:accessKey:delegate:` 方法之前调用头文件中的配置 `域名` 接口：

如需接入上海服务接入点，则将域名设置为 `tpns.sh.tencent.com` 。

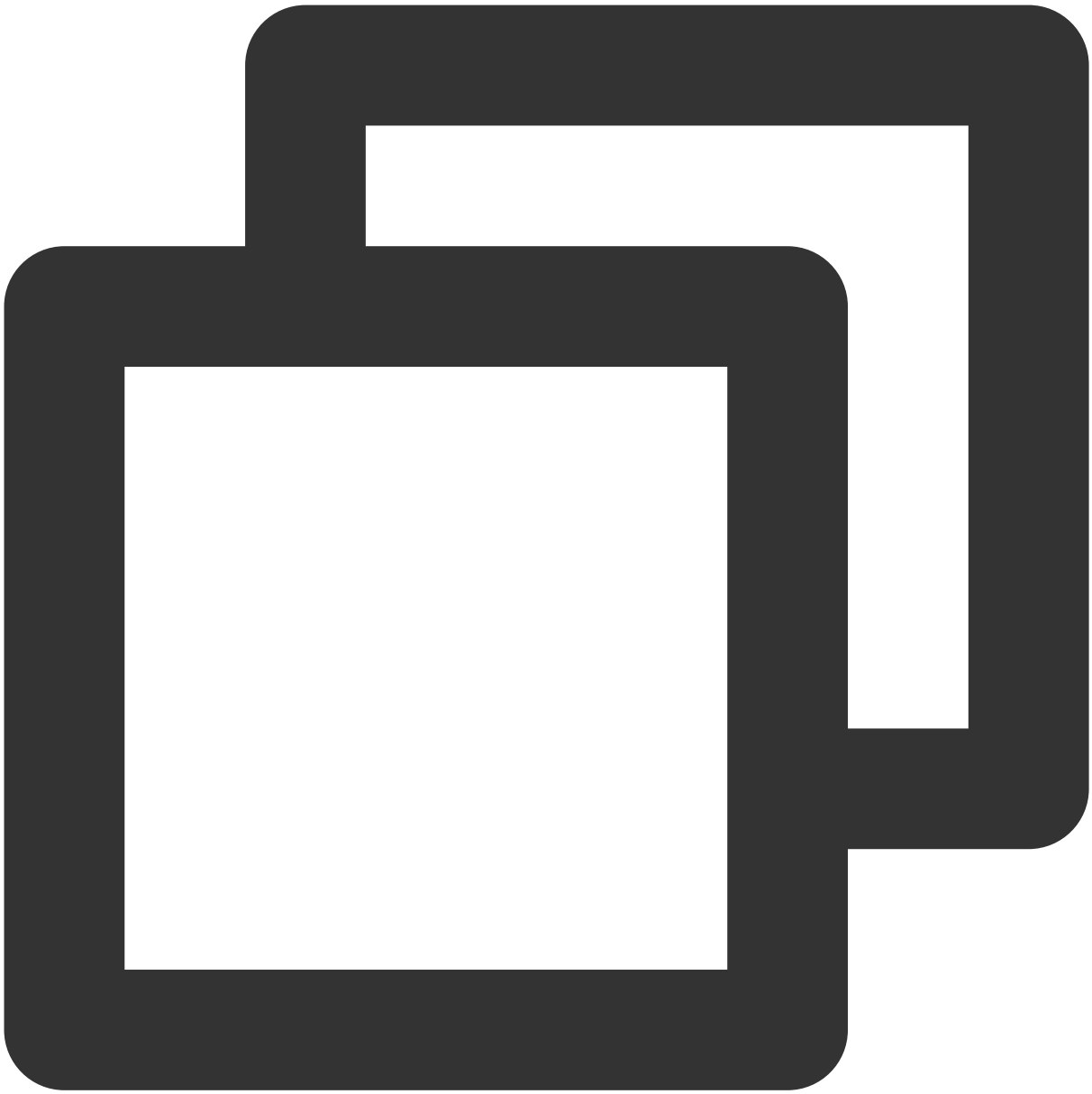
示例



```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.sh.tencent.com"];
```

如需接入新加坡服务接入点，则将域名设置为 `tpns.sgp.tencent.com` 。

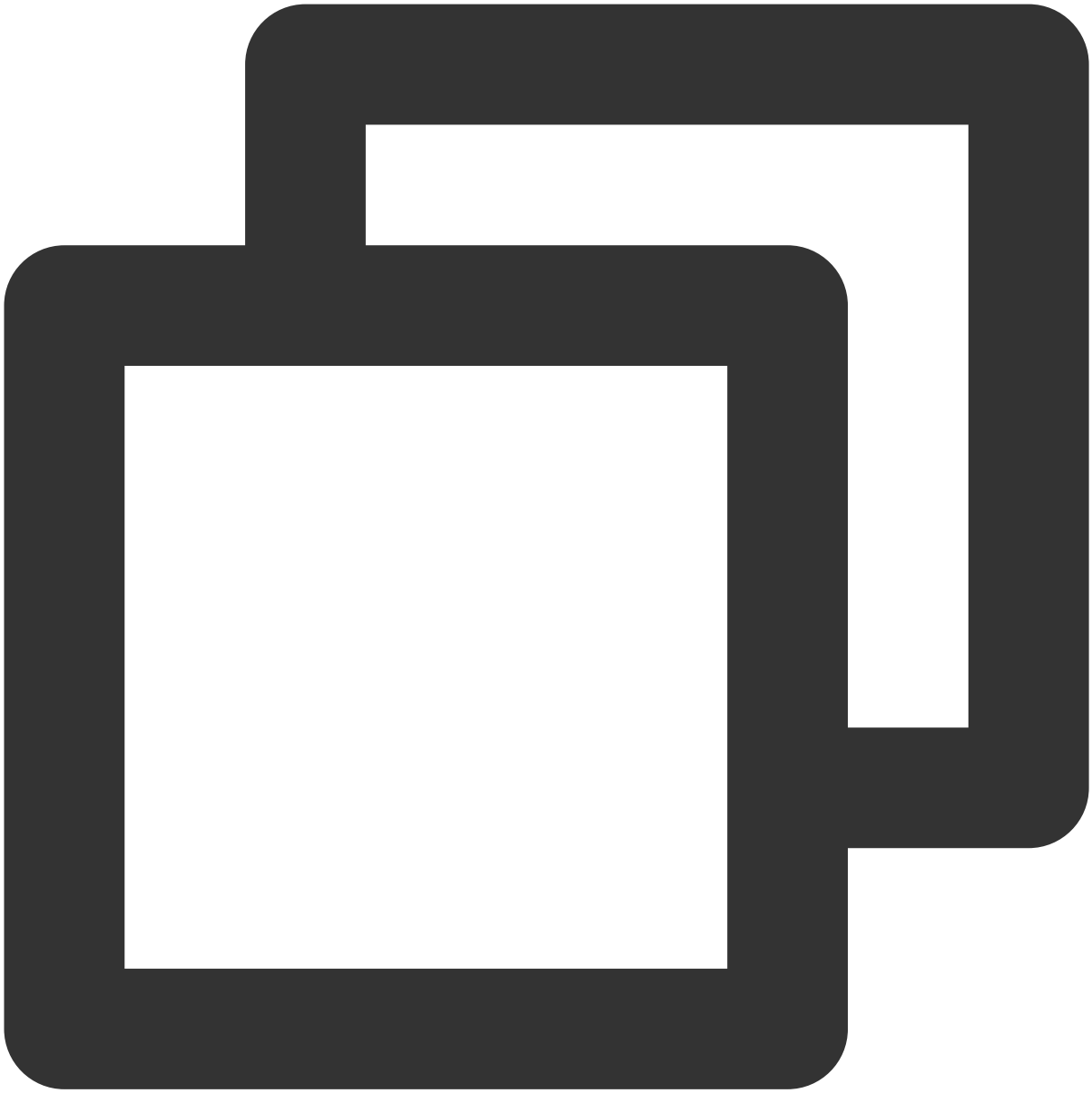
示例



```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.sgp.tencent.com"];
```

如需接入中国香港服务接入点，则将域名设置为 `tpns.hk.tencent.com` 。

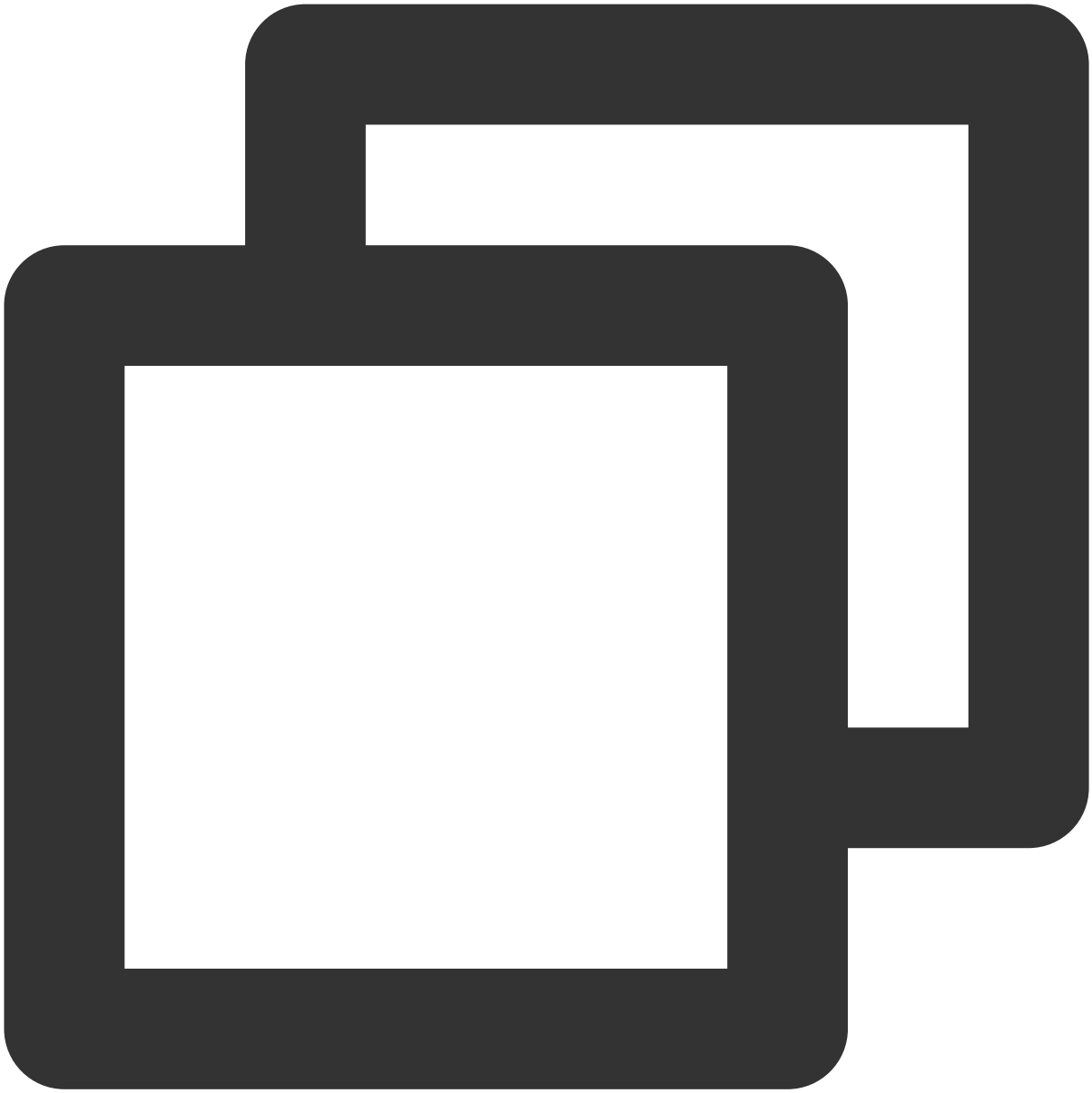
示例



```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.hk.tencent.com"];
```

如需接入中国广州服务接入点，则将域名设置为 `tpns.tencent.com` 。

示例

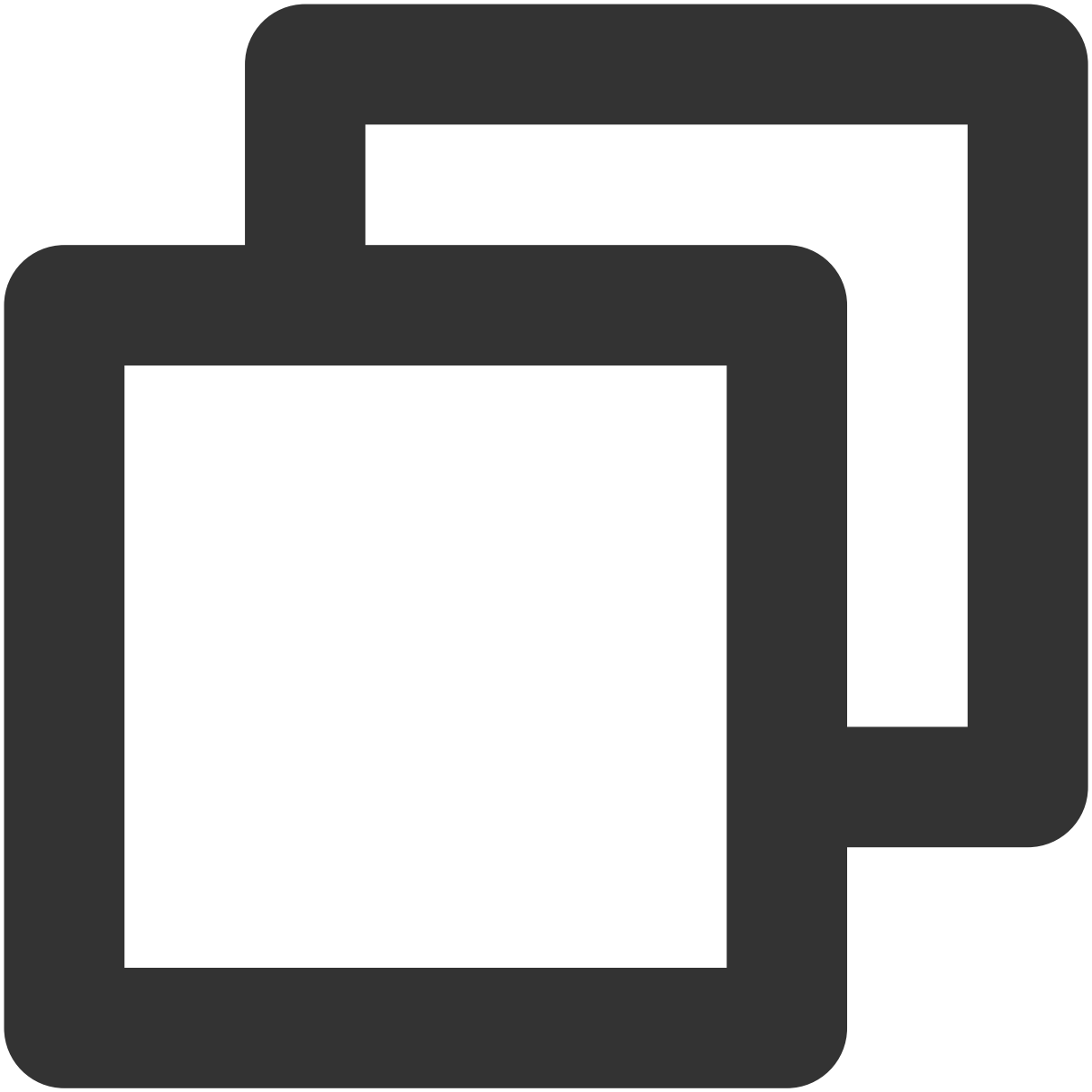


```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.tencent.com"];
```

接入样例

调用启动腾讯移动推送的 API，并根据需要实现 `XGPushDelegate` 协议中的方法，开启推送服务。

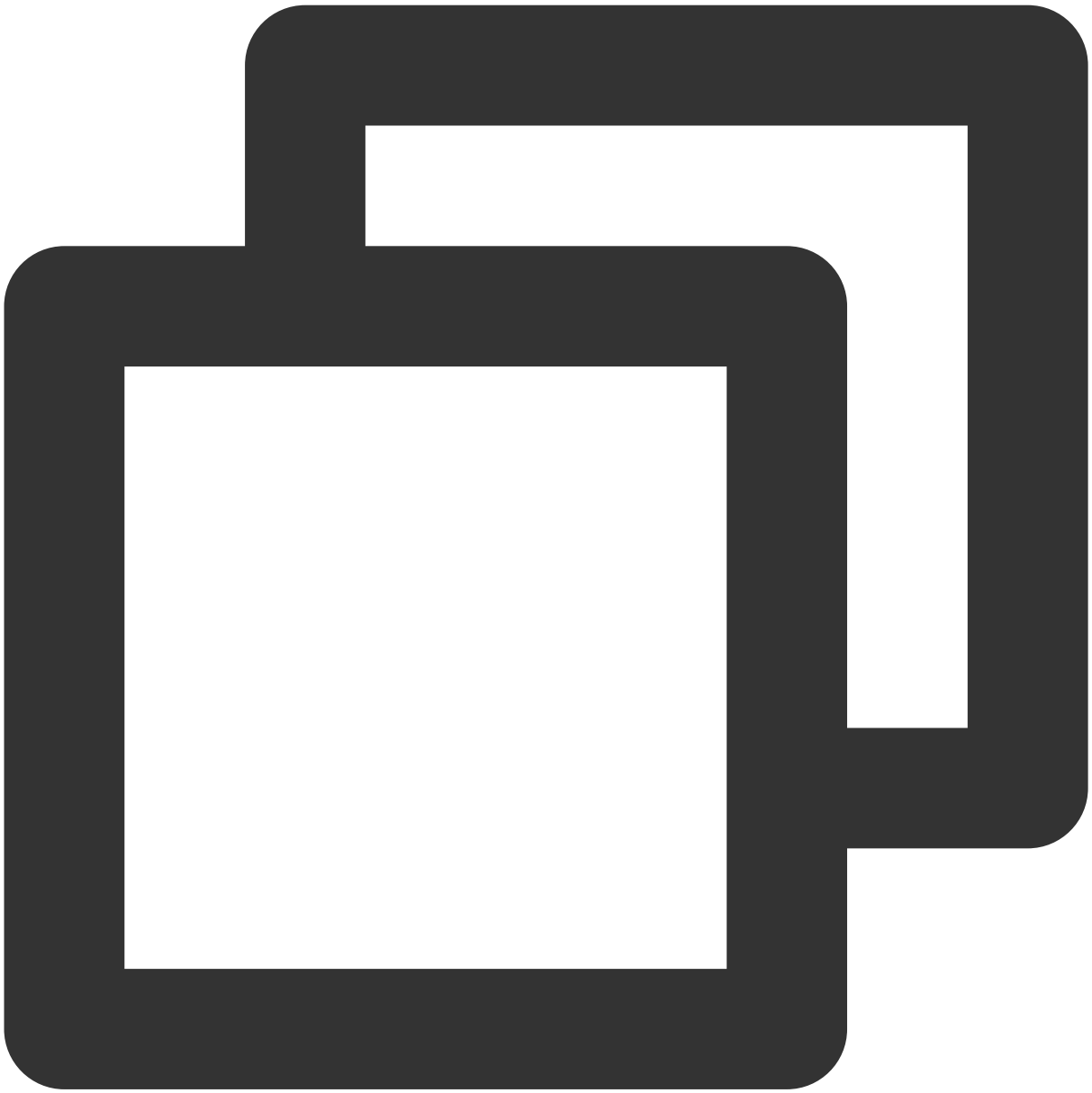
1. 启动腾讯移动推送服务， `AppDelegate` 示例如下：



```
@interface AppDelegate () <XGPushDelegate>
@end
/**
@param AccessID  通过 TPNS 管理台申请的 AccessID
@param AccessKey  通过 TPNS 管理台申请的 AccessKey
@param delegate 回调对象
**/
-(BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDi
{
  [[XGPush defaultManager] startXGWithAccessID:<your AccessID> accessKey:<your Acce
return YES;
```

```
}
```

2. 在 `AppDelegate` 中，选择实现 `XGPushDelegate` 协议中的方法：



```
/// 统一接收消息的回调
/// @param notification 消息对象(有2种类型NSDictionary和UNNotification具体解析参考示例代码
/// @note 此回调为前台收到通知消息及所有状态下收到静默消息的回调（消息点击需使用统一点击回调）
/// 区分消息类型说明：xg字段里的msgtype为1则代表通知消息msgtype为2则代表静默消息
- (void)xgPushDidReceiveRemoteNotification:(nonnull id)notification withCompletionH
 /// code
}
```

```
 /// 统一点击回调
/// @param response 如果iOS 10+/macOS 10.14+则为UNNotificationResponse，低于目标版本则为
- (void)xgPushDidReceiveNotificationResponse:(nonnull id)response withCompletionHan
  /// code
}
```

通知服务扩展插件集成

SDK 提供了 Service Extension 接口，可供客户端调用，从而可以使用以下扩展功能：

精准统计 APNs 通道消息抵达。

接收 APNs 通道图片、音视频富媒体消息。

接入步骤请参考文档 通知服务扩展的使用说明。

注意：

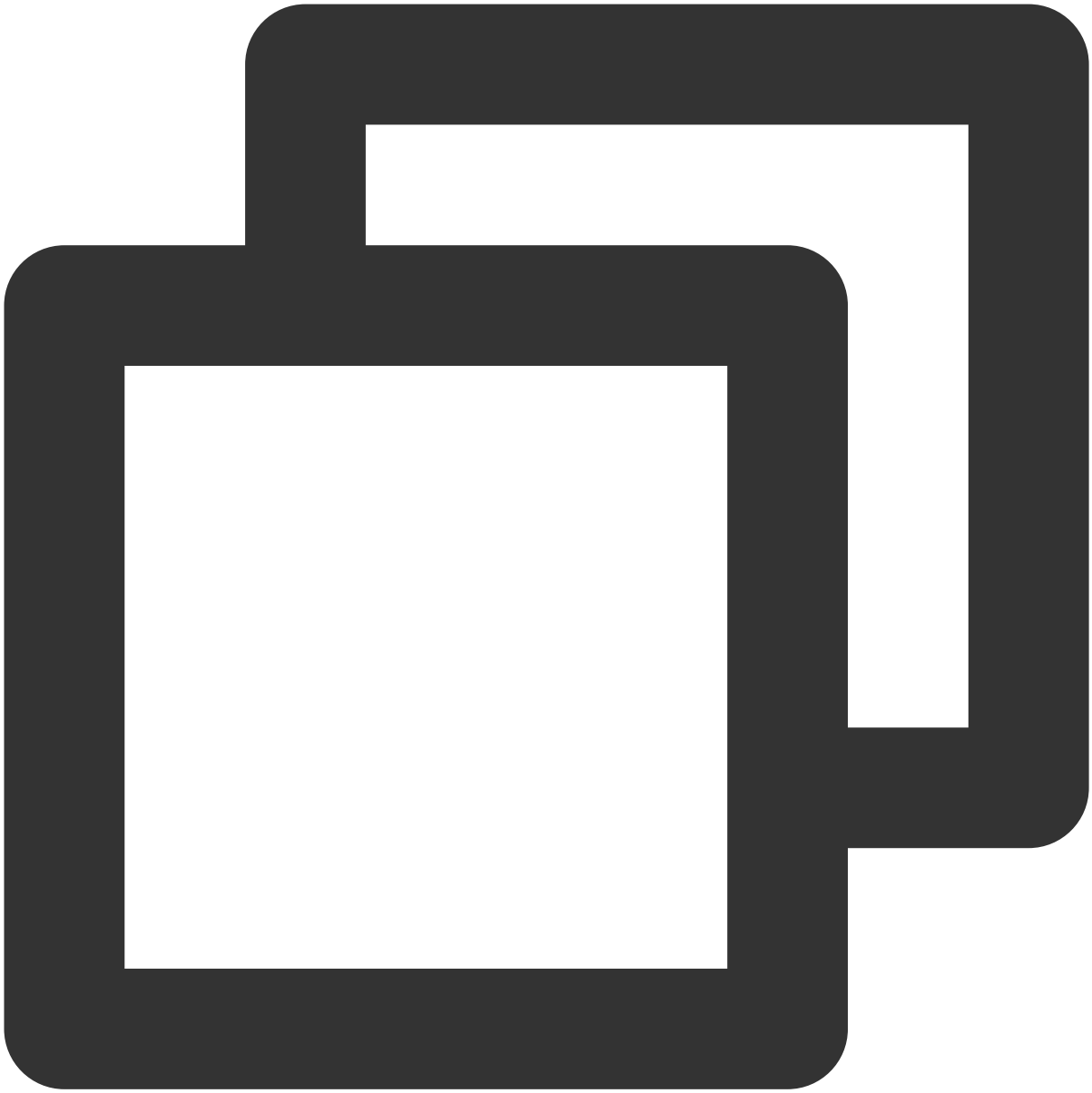
如果未集成此接口，则无法统计 APNs 通道“抵达数”。

调试方法

开启 Debug 模式

打开 Debug 模式，即可在终端查看详细的腾讯移动推送 Debug 信息，方便定位问题。

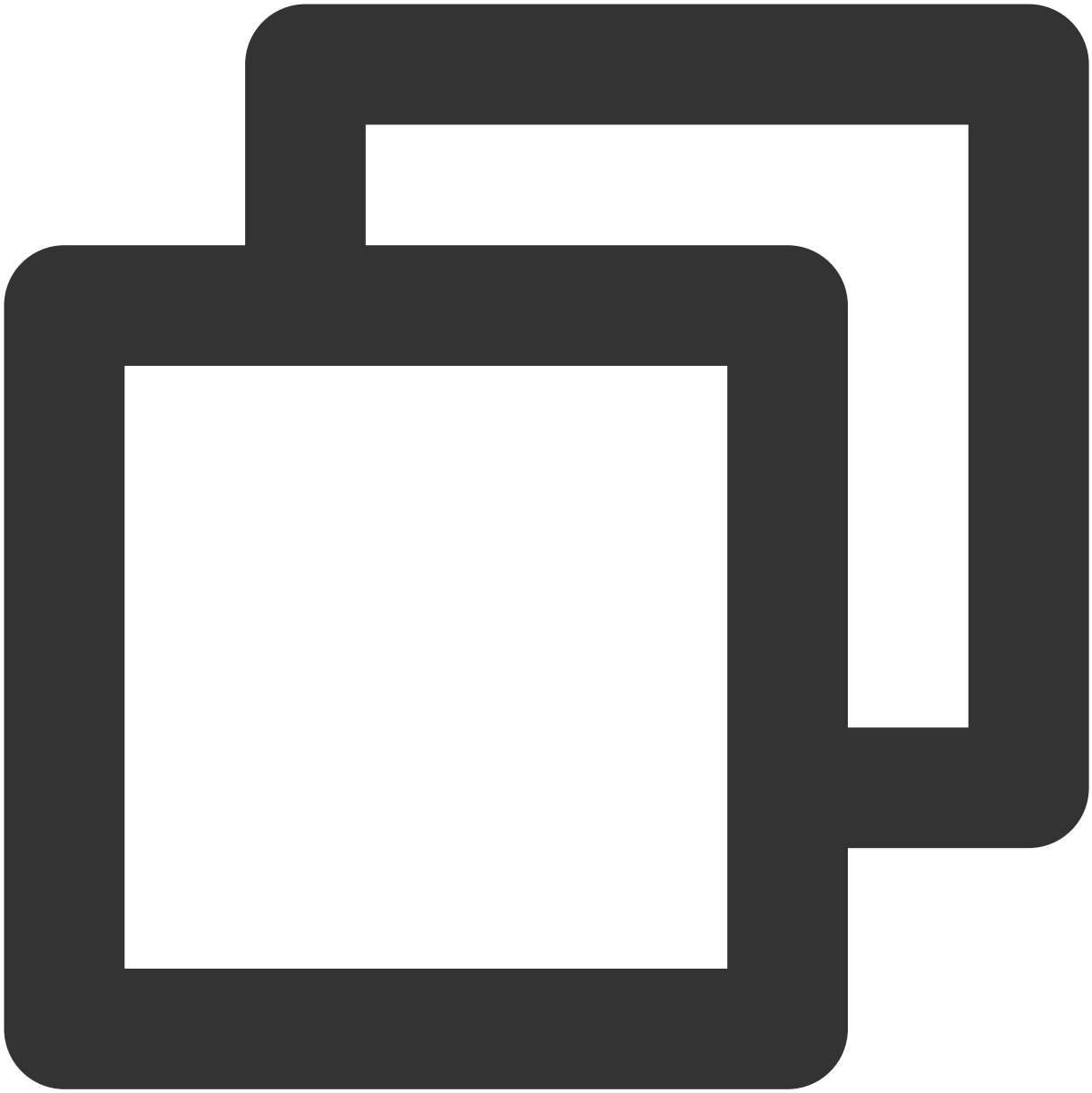
示例代码



```
//打开 debug 开关
[[XGPush defaultManager] setEnableDebug:YES];
```

实现 `XGPushDelegate` 协议

在调试阶段建议实现协议中的此方法，即可获取更详细的调试信息：

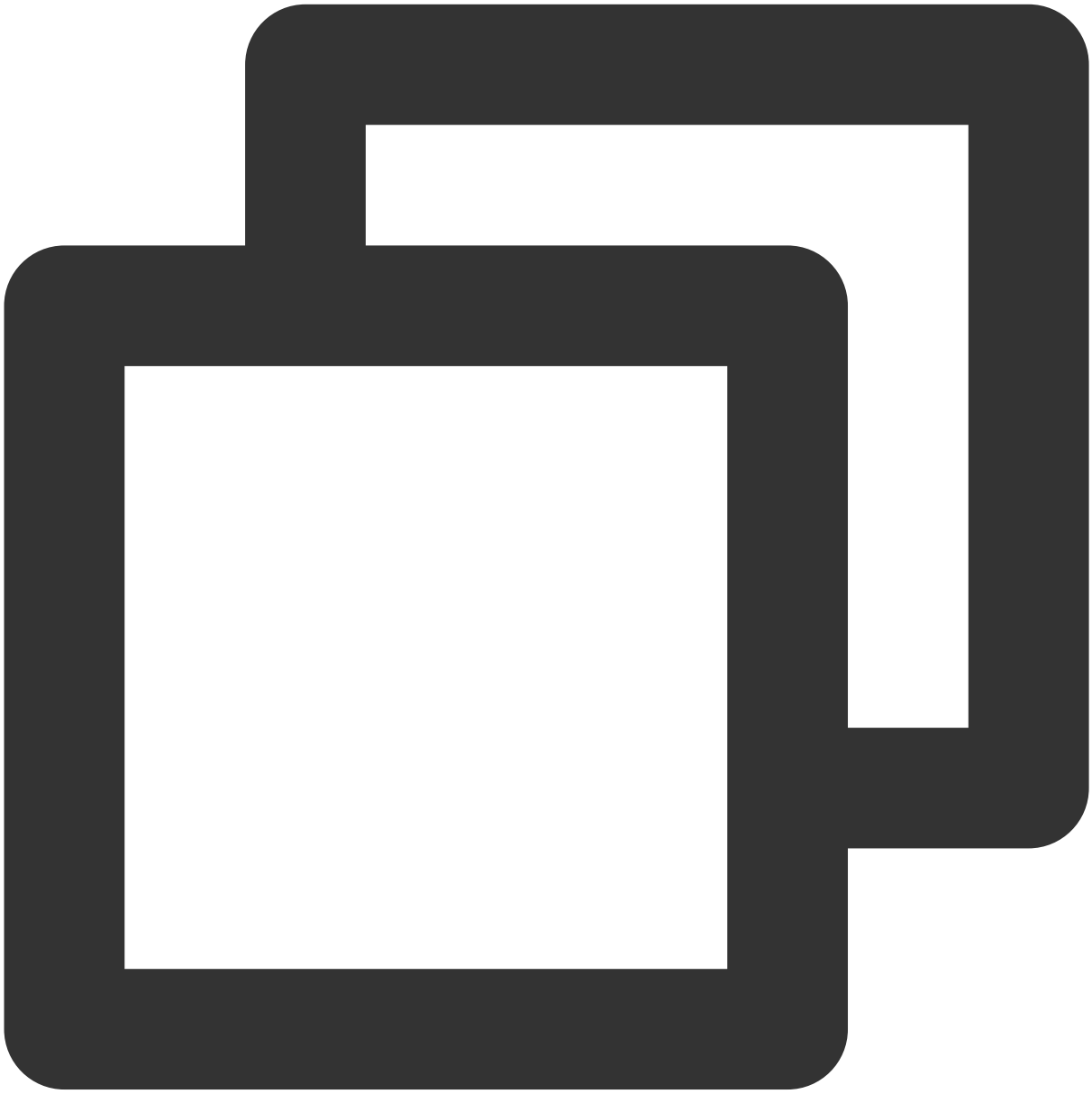


```
/**
@brief 注册推送服务回调
@param deviceToken APNs 生成的 Device Token
@param xgToken TPNS 生成的 Token，推送消息时需要使用此值。TPNS 维护此值与 APNs 的 Device To
@param error 错误信息，若 error 为 nil 则注册推送服务成功
@note TPNS SDK1.2.6.0+
*/
- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nu

/// 注册推送服务失败回调
/// @param error 注册失败错误信息
```

```
/// @note TPNS SDK1.2.7.1+
- (void)xgPushDidFailToRegisterDeviceTokenWithError:(nullable NSError *)error {
}
```

观察日志

如果 Xcode 控制台，显示如下相似日志，表明客户端已经正确集成 SDK。



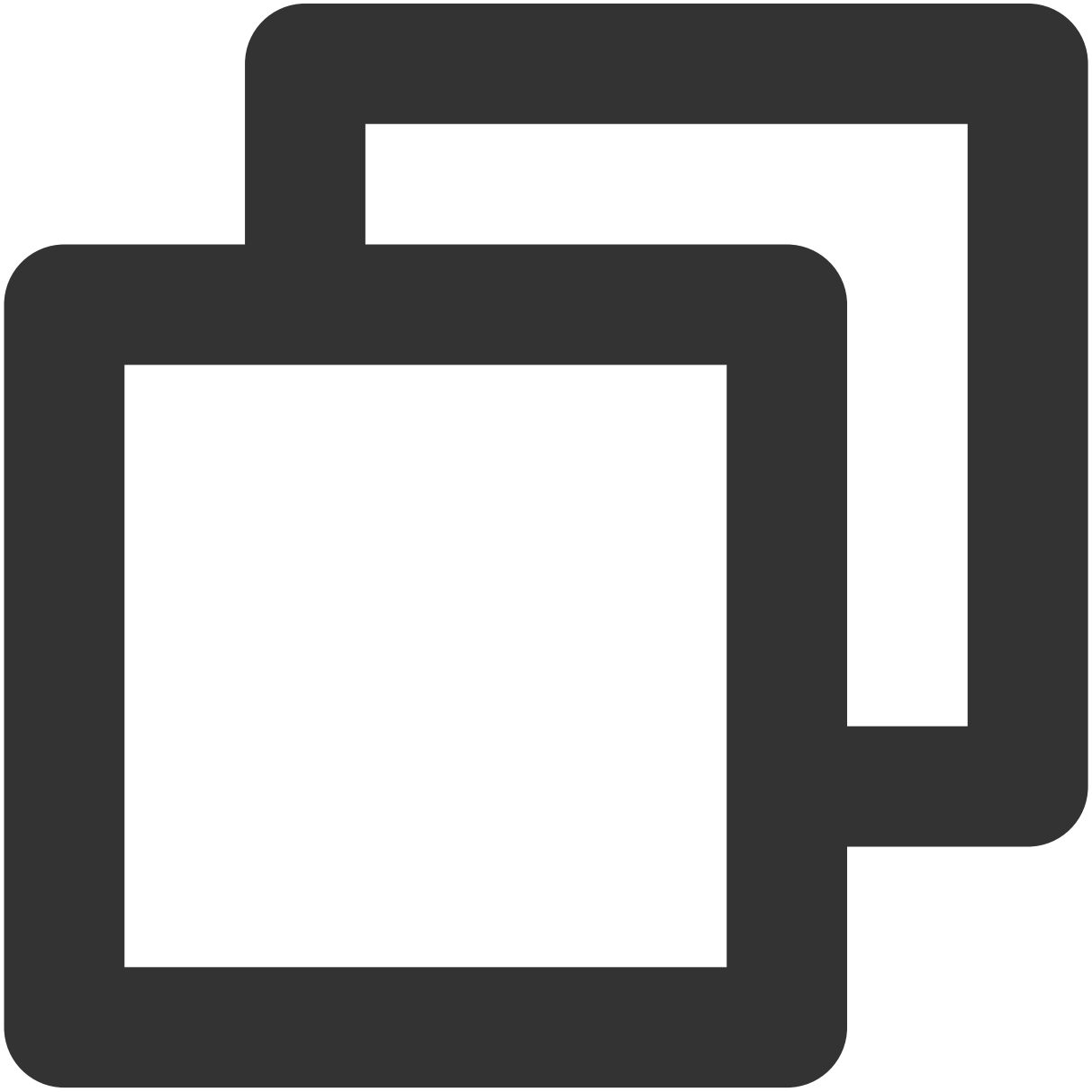
```
[TPNS] Current device token is 9298da5605c3b242261b57****376e409f826c2caf87aa0e6112
[TPNS] Current TPNS token is 00c30e0aeddff1270d8****dc594606dc184
```

注意：

在推送单个目标设备时请使用 TPNS 36位的 Token。

统一接收消息及点击消息回调说明

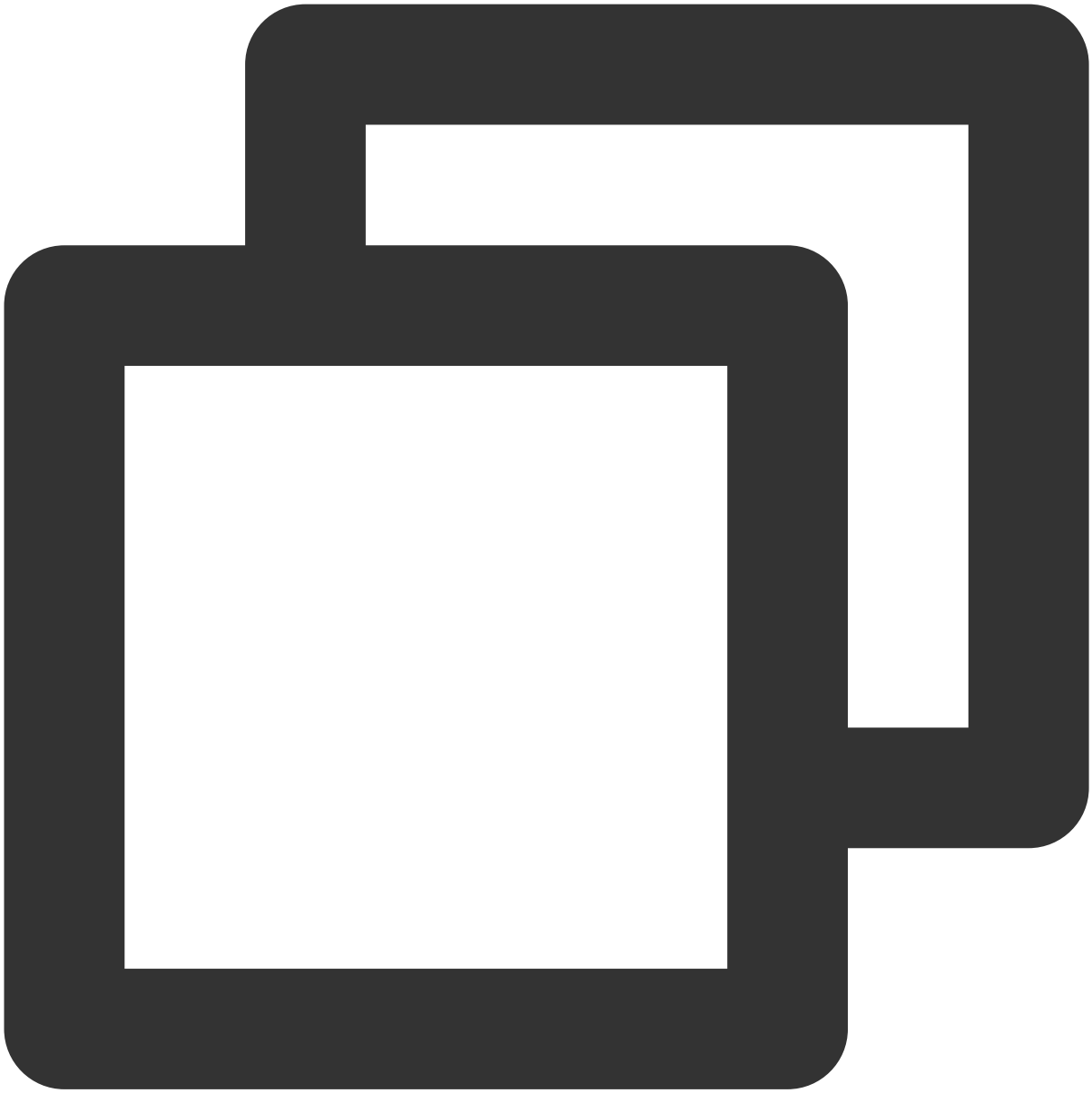
移动推送自建通道及 APNs 通道统一接收消息回调，当应用在前台收到通知消息，以及所有状态（前台、后台、关闭）下收到静默消息会走此回调。



```
- (void)xgPushDidReceiveRemoteNotification:(nonnull id)notification withCompletionH
```

说明：

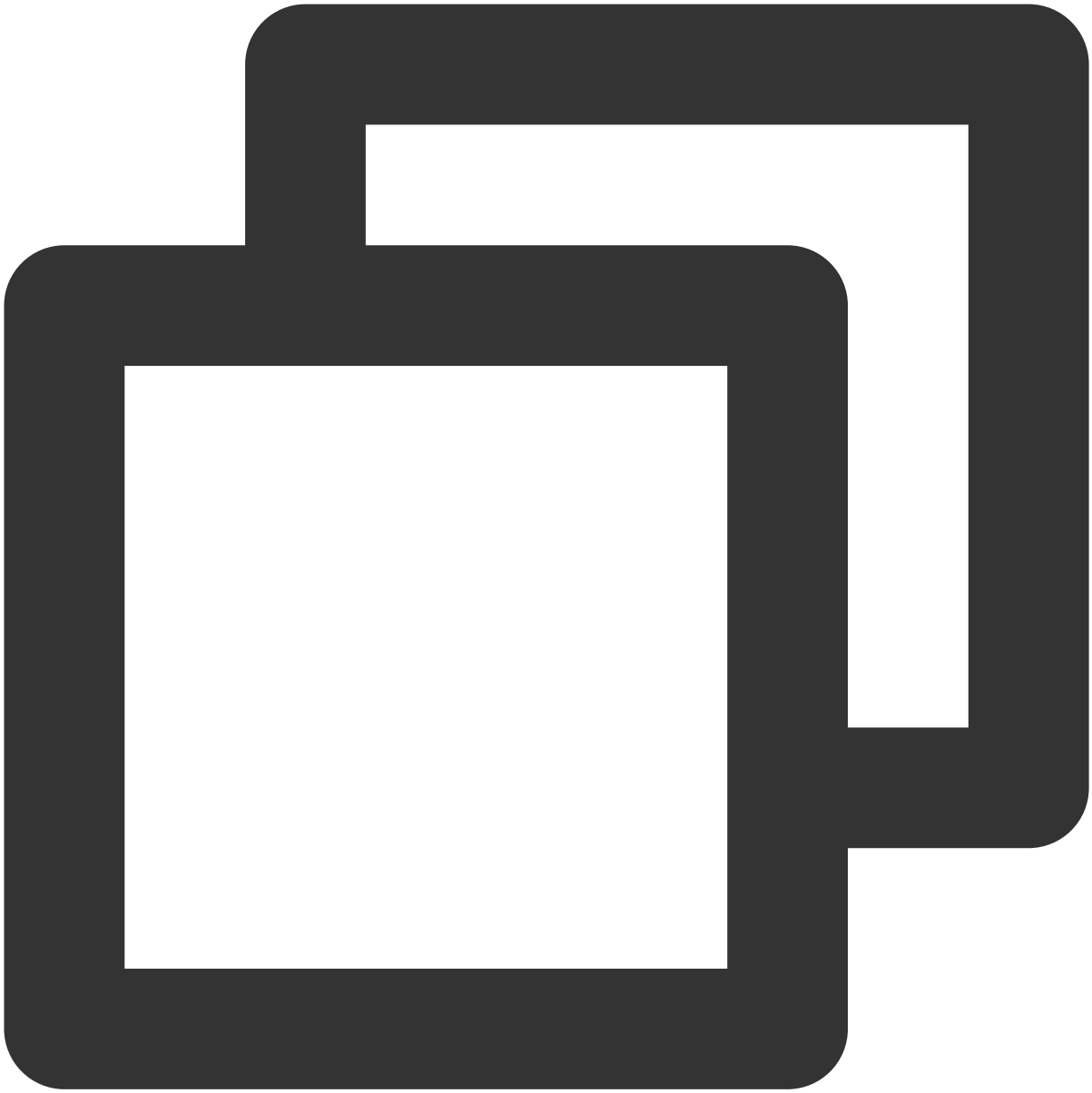
当您前台收到通知时默认是不弹横幅的，如需展示请参考如下代码：



```
if ([notification isKindOfClass:[UNNotification class]]) {
completionHandler(UNNotificationPresentationOptionBadge | UNNotificationPresentatio
}
```

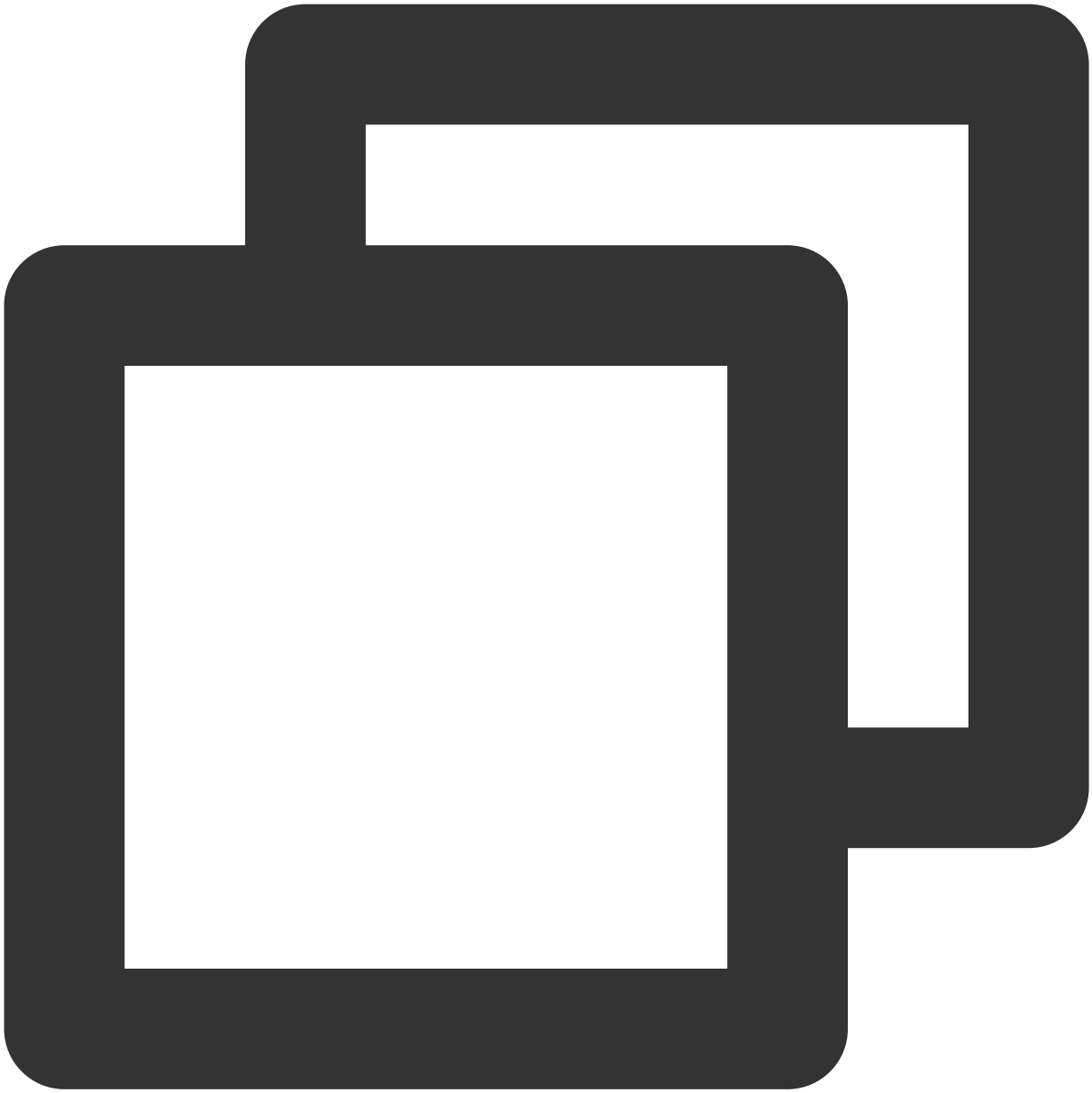
当应用在前台收到通知消息以及所有状态下收到静默消息时，会触发统一接收消息回调 xgPushDidReceiveRemoteNotification。

区分前台收到通知消息和静默消息示例代码如下：



```
NSDictionary *tpnsInfo = notificationDic[@"xg"];
NSNumber *msgType = tpnsInfo[@"msgtype"];
if (msgType.integerValue == 1) {
       /// 前台收到通知消息
  } else if (msgType.integerValue == 2) {
       /// 收到静默消息
  } else if (msgType.integerValue == 9) {
       /// 收到本地通知（TPNS本地通知）
  }
```

统一点击消息回调，此回调方法为应用所有状态（前台、后台、关闭）下的通知消息点击回调。



```
/// 统一点击回调
/// @param response 如果 iOS 10+/macOS 10.14+ 则为 UNNotificationResponse，低于目标版本
/// @note TPNS SDK1.2.7.1+
- (void)xgPushDidReceiveNotificationResponse:(nonnull id)response withCompletionHan
```

注意：

移动推送统一消息回调 `xgPushDidReceiveRemoteNotification` 会处理消息接收，并自动后续调用 `application:didReceiveRemoteNotification:fetchCompletionHandler` 方法。然而，该方法也可能被其他 SDK 也进行 hook 调用。

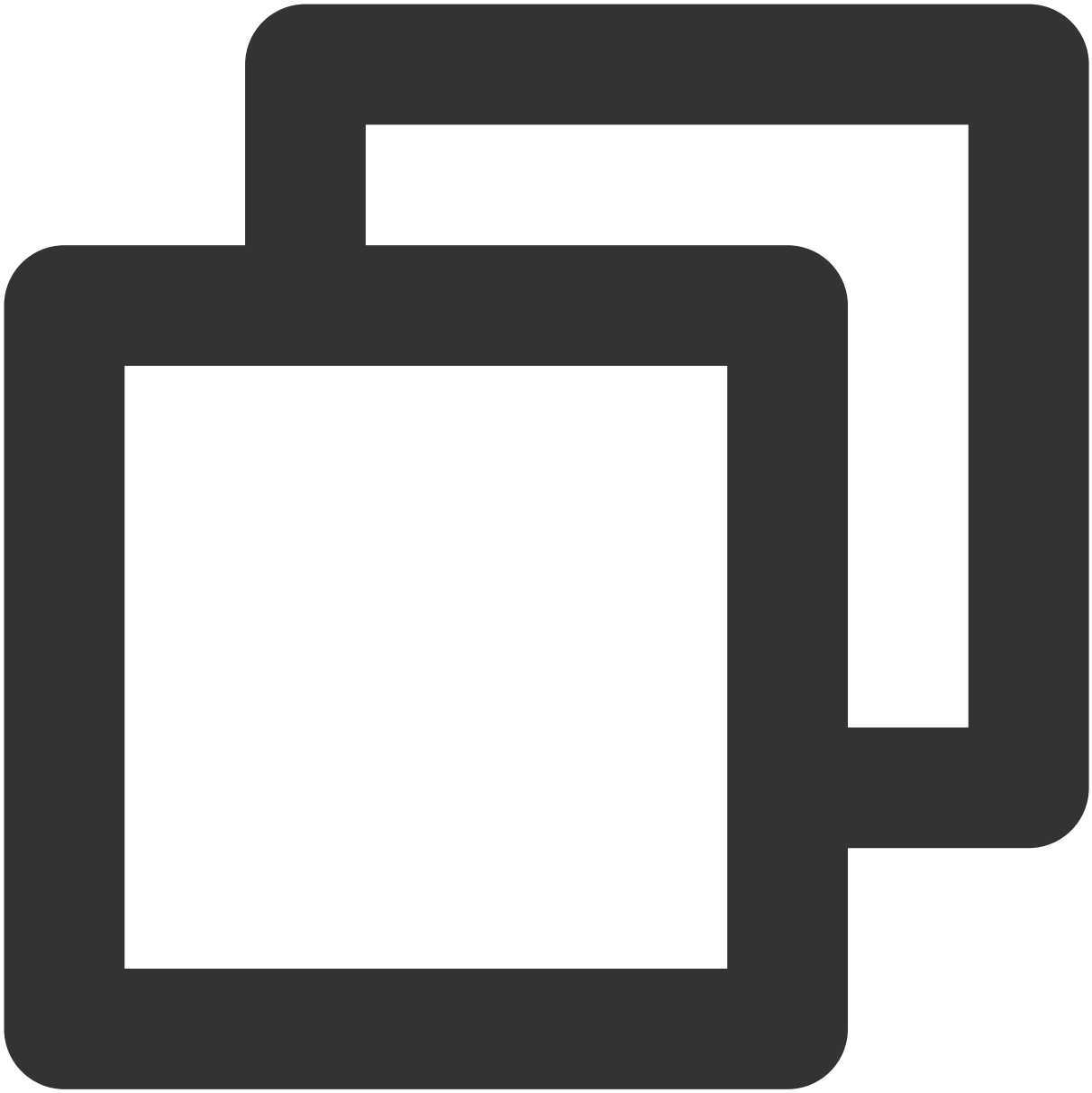
如果您只集成了移动推送平台，我们不推荐再去实现系统通知回调方法，请统一在移动推送通知回调中进行处理。
如果您集成了多推送平台，并且需要在 `application:didReceiveRemoteNotification:fetchCompletionHandler` 方法处理其他推送平台的业务，请参照如下指引，避免业务重复：
您需要区分平台消息，在两个消息回调方法中分别拿到消息字典后通过“xg”字段来区分是否是移动推送平台的消息，如果是移动推送的消息则在 `xgPushDidReceiveRemoteNotification` 方法进行处理，非 移动推送消息请统一在 `application:didReceiveRemoteNotification:fetchCompletionHandler` 方法处理 `xgPushDidReceiveRemoteNotification` 和 `application:didReceiveRemoteNotification:fetchCompletionHandler` 如果都执行，总共只需要调用一次 `completionHandler` 。如果其他 SDK 也调用 `completionHandler` ，确保整体的 `completionHandler` 只调用一次。这样可以防止由于多次 `completionHandler` 而引起的 crash。

高级配置（可选）

获取移动推送Token 交互建议

建议您完成 SDK 集成后，在 App 的【关于】、【意见反馈】等比较不常用的 UI 中，通过手势或者其他方式显示移动推送Token，控制台和 Restful API 推送需要根据移动推送Token 进行 Token 推送，后续问题排查也需要根据移动推送Token 进行定位。

示例代码

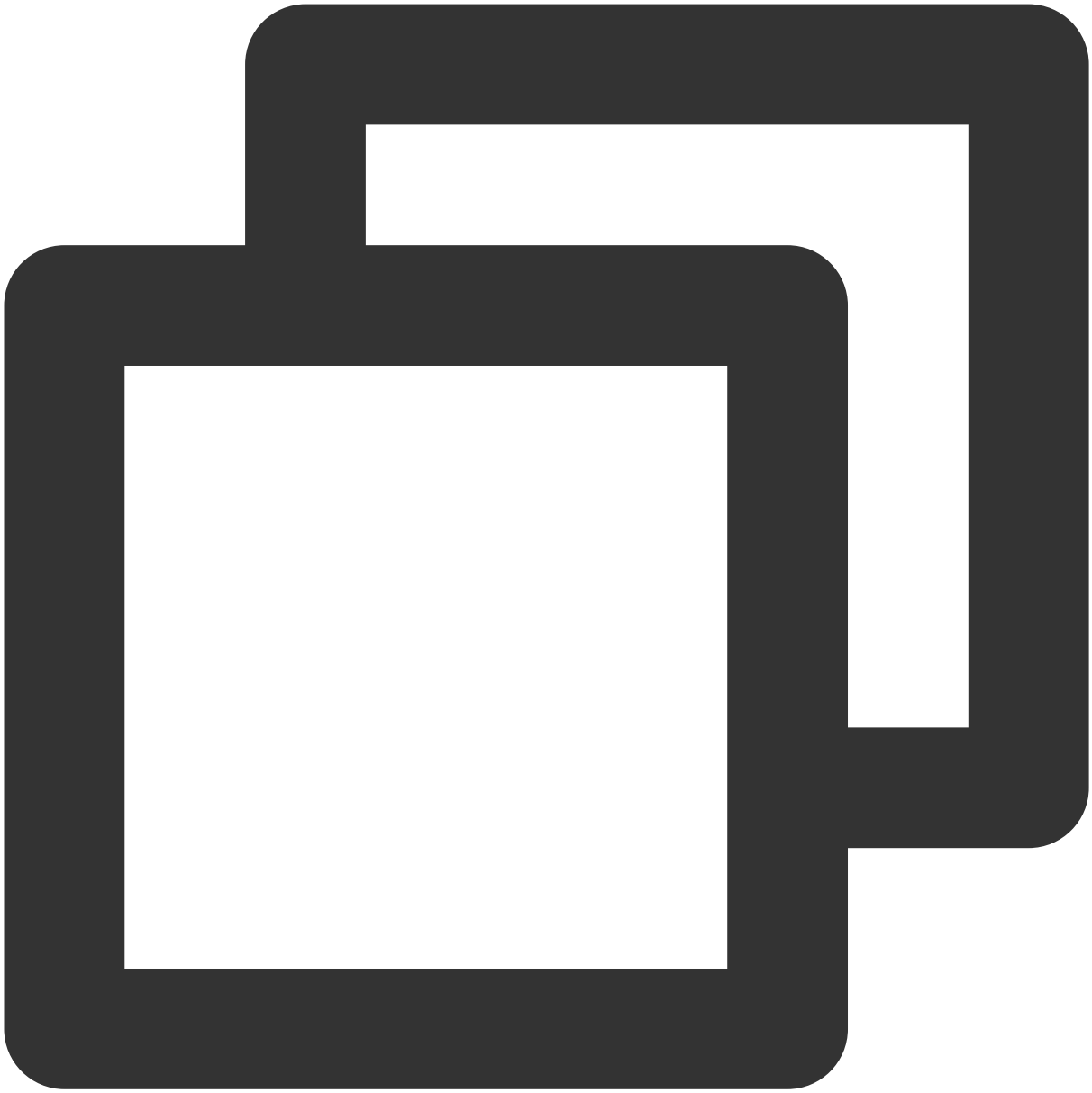


```
//获取 TPNS 生成的 Token
[[XGPushTokenManager defaultTokenManager] xgTokenString];
```

获取 "移动推送运行日志" 交互建议

建议您完成 SDK 集成后，在 App 的【关于】、【意见反馈】等比较不常用的 UI 中，通过手势或者其他方式显示"移动推送运行日志"，方便后续问题排查。

示例代码



```
[[XGPush defaultManager] uploadLogCompletionHandler:^(BOOL result, NSString * _Null
NSString *title = result ? NSLocalizedString(@"report_log_info", nil) : NSLocalized
if (result && errorMessage.length>0) {
UIPasteboard *pasteboard = [UIPasteboardgeneralPasteboard];
pasteboard.string = errorMessage;
}
[TPNSCommonMethodshowAlert:title message:errorMessage viewController:selfcompletion
}];
```

接口文档

最近更新时间：2024-01-16 17:42:20

说明

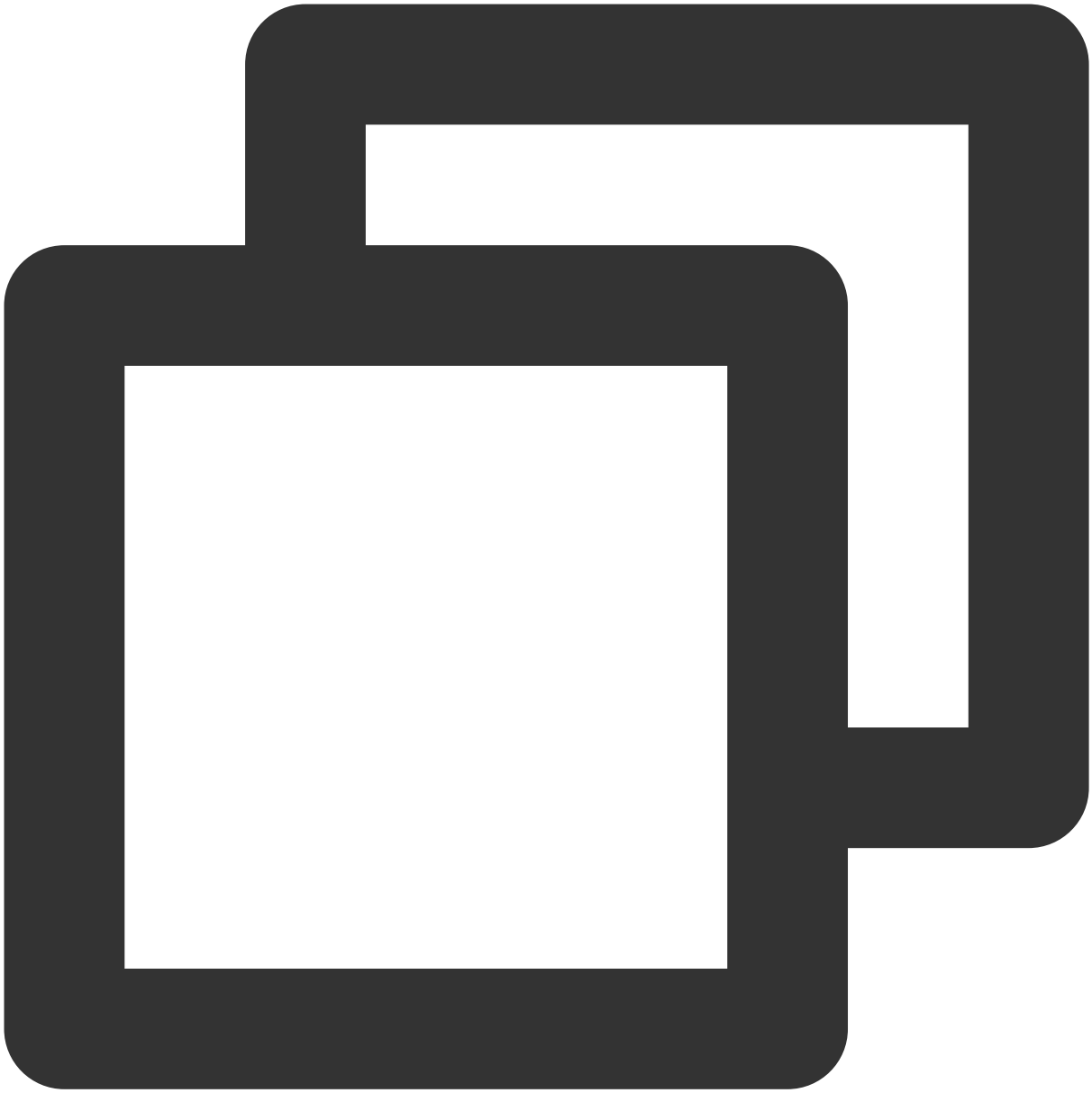
本文档中账号功能、标签功能及用户属性功能适用于 **SDK 1.2.9.0或更高版本**，**1.2.7.2**及之前版本请参见 接口文档。

启动腾讯移动推送服务

以下为设备注册相关接口方法，若需了解调用时机及调用原理，可查看 设备注册流程。

接口说明

通过使用在腾讯移动推送官网注册的应用信息，启动腾讯移动推送服务。
（此接口为 SDK 1.2.7.2版本新增，1.2.7.1及之前版本请参考 SDK 包内 XGPush.h 文件 `startXGWithAppID` 接口）。



```
/// @note TPNS SDK1.2.7.2+
- (void)startXGWithAccessID:(uint32_t)accessID accessKey:(nonnull NSString *)access
```

参数说明

accessID：通过前台申请的 AccessID。

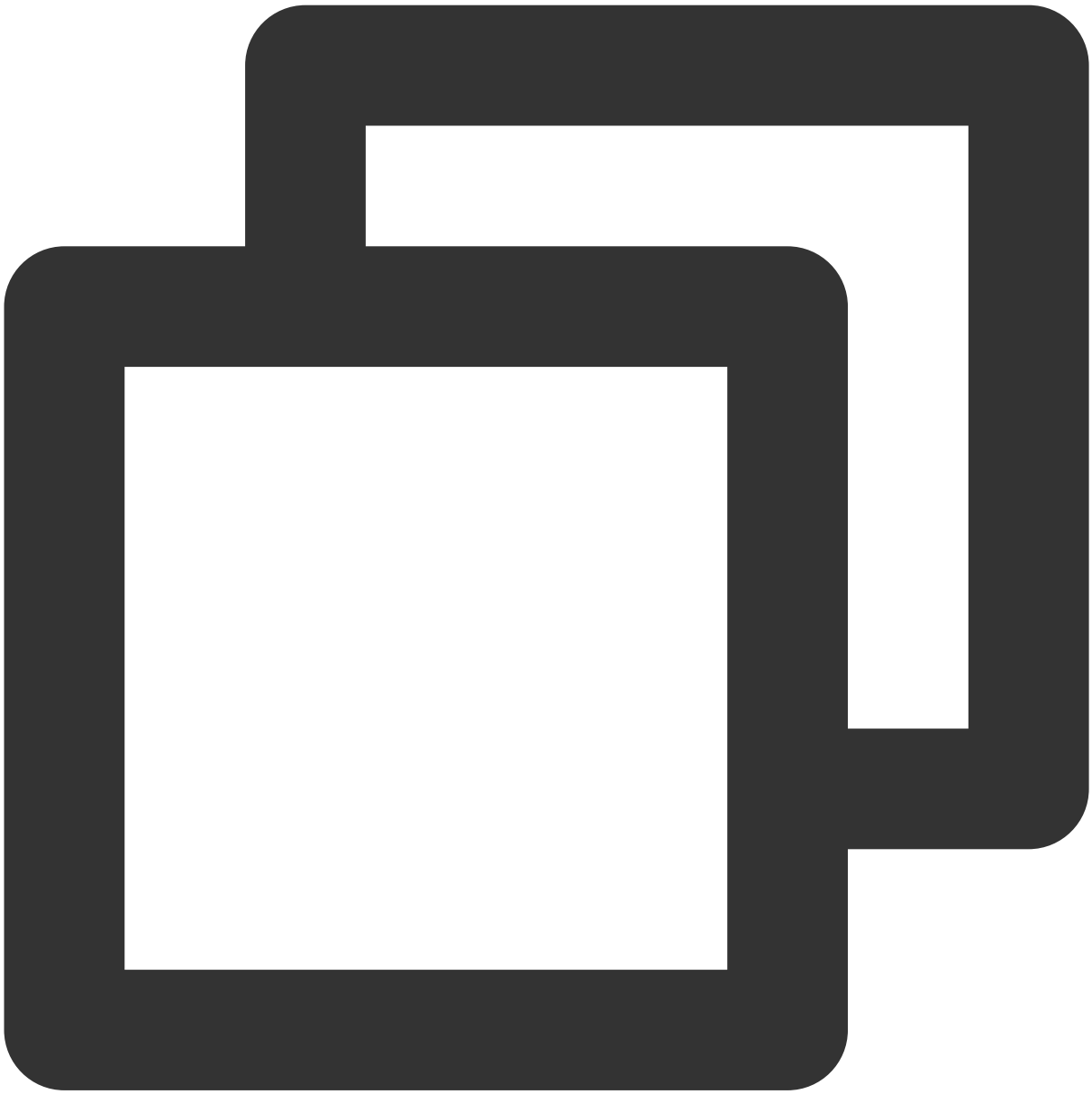
accessKey：通过前台申请的 AccessKey。

Delegate：回调对象。

注意：

接口所需参数必须要正确填写，否则腾讯移动推送服务将不能正确为应用推送消息。

示例代码



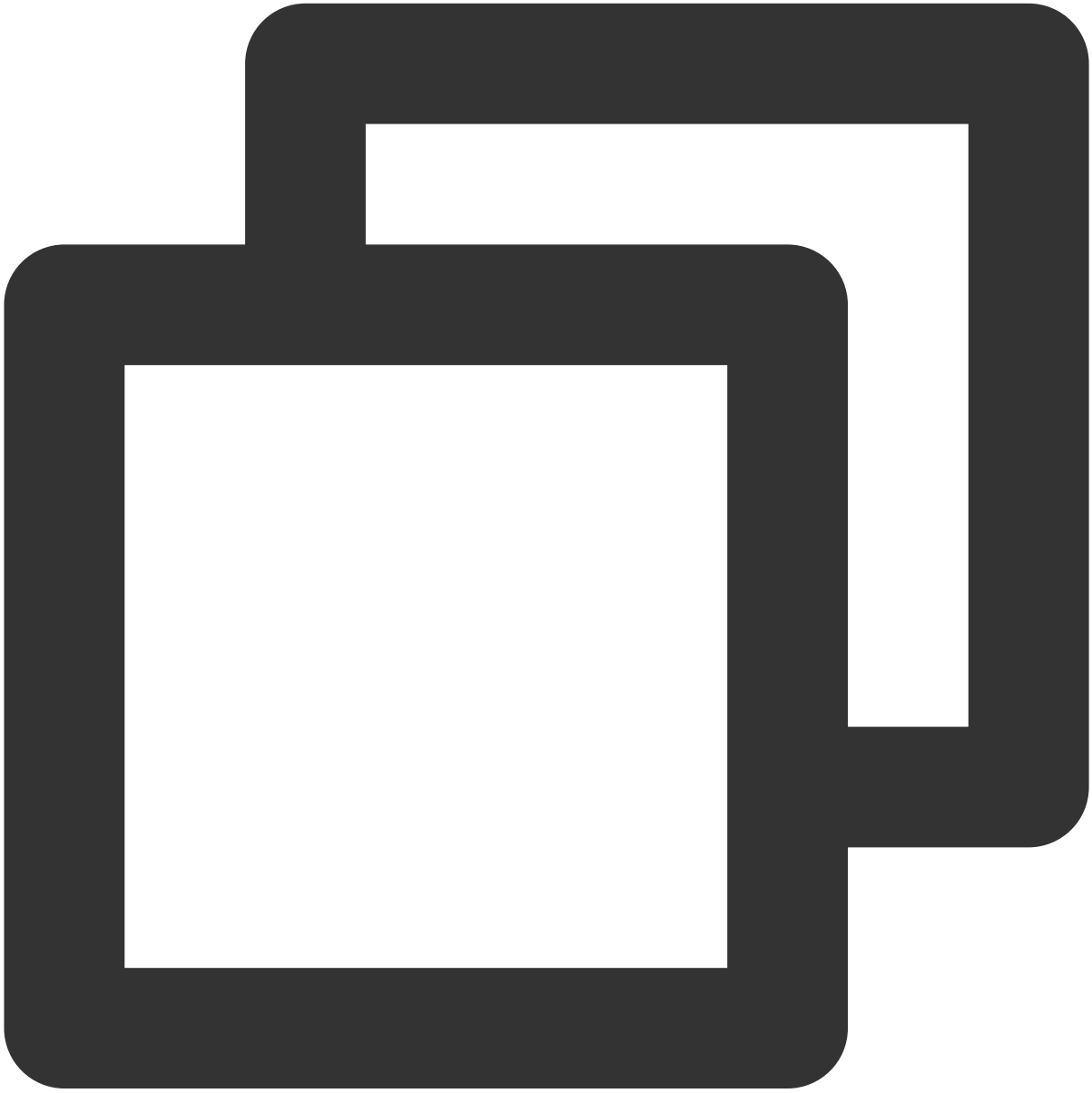
```
[[XGPush defaultManager] startXGWithAccessID:<your AccessID> accessKey:<your Acces
```

终止腾讯移动推送服务

以下为设备注册相关接口方法，若需了解调用时机及调用原理，可查看 设备反注册流程。

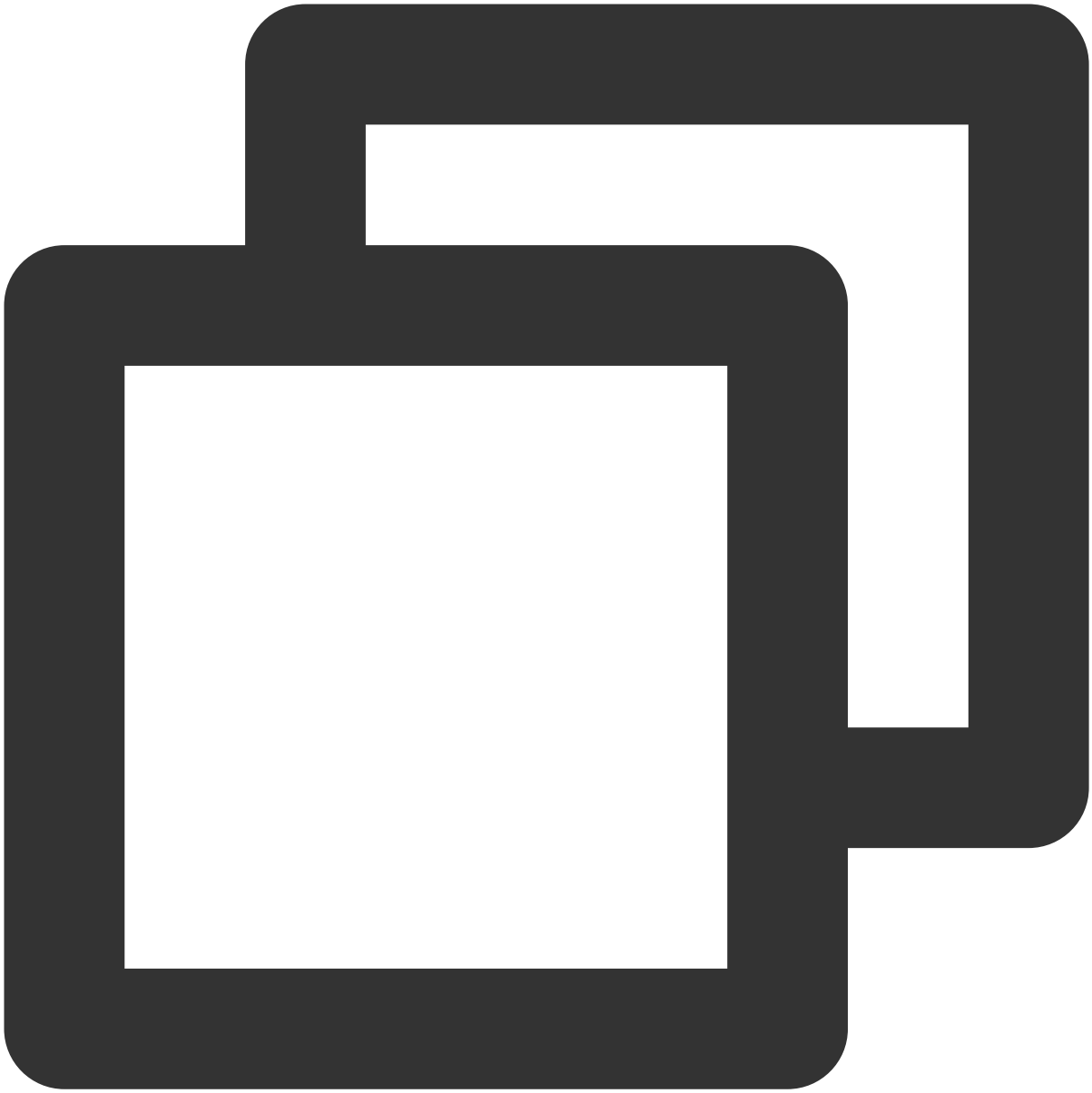
接口说明

终止腾讯移动推送服务后，将无法通过腾讯移动推送服务向设备推送消息，如再次需要接收腾讯移动推送服务的消息推送，则必须再次调用 `startXGWithAccessID:accessKey:delegate:` 方法重启腾讯移动推送服务。



```
- (void)stopXGNotification;
```

示例代码



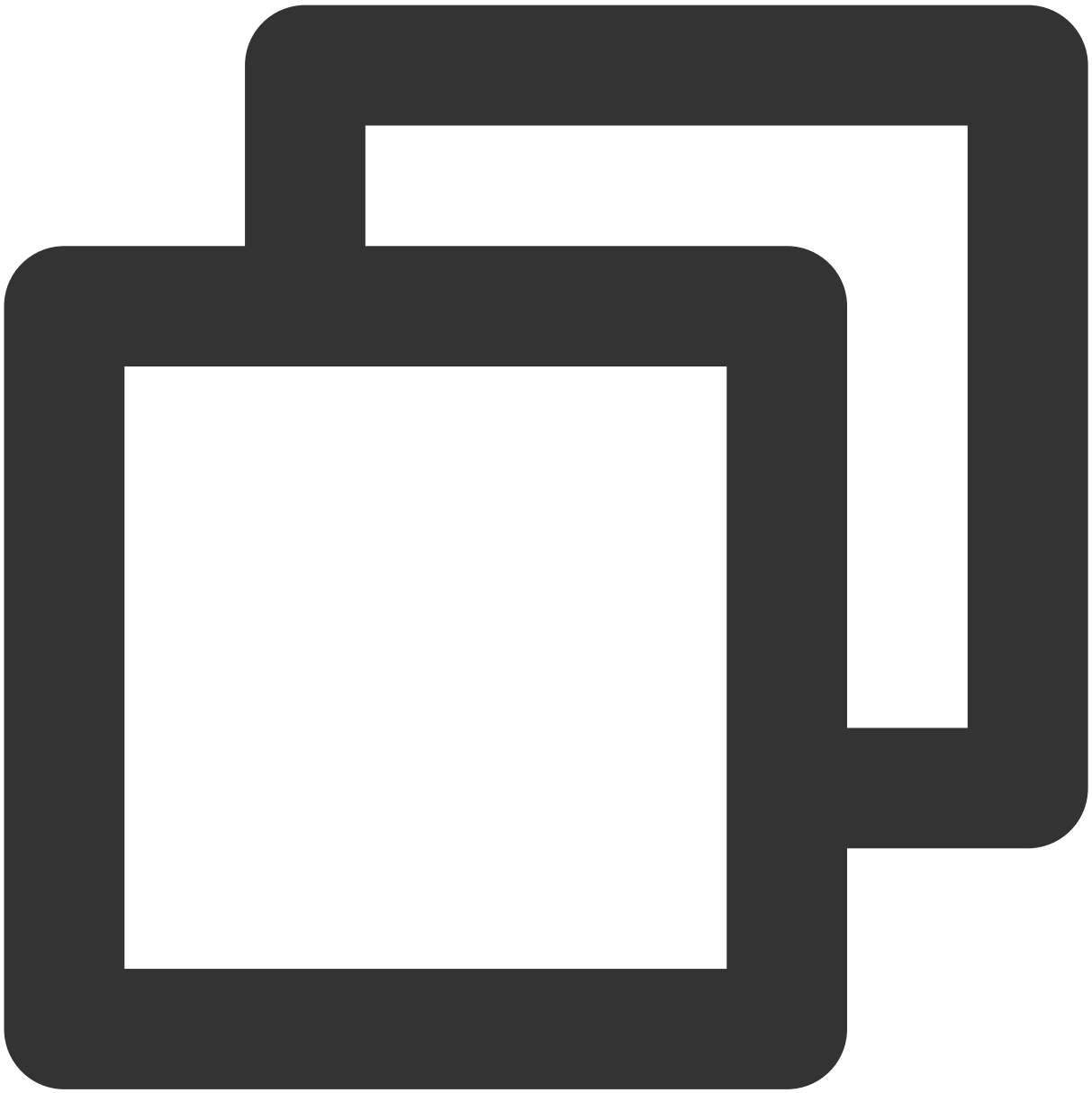
```
[[XGPush defaultManager] stopXGNotification];
```

移动推送 Token 及注册结果

查询移动推送 Token

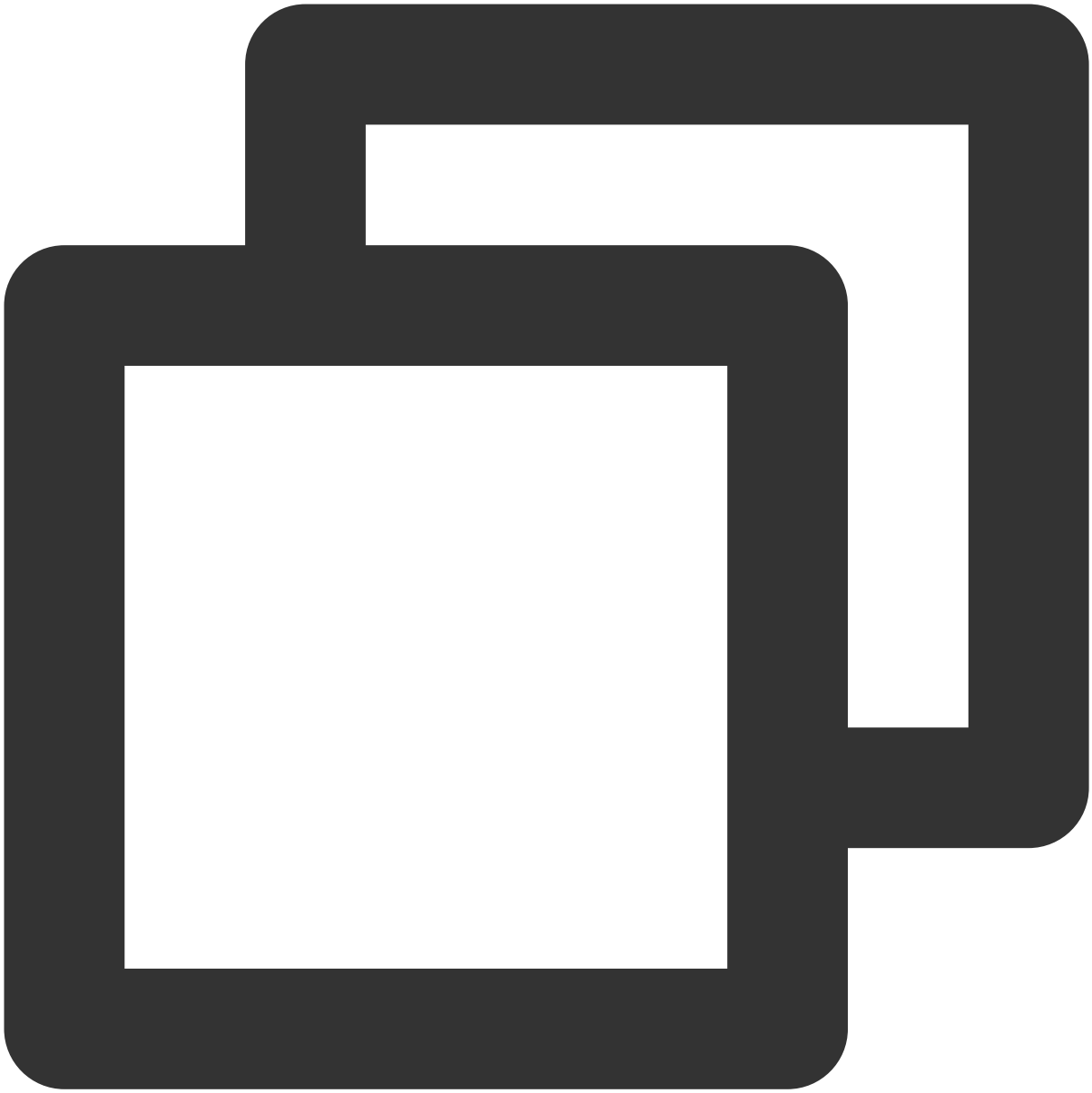
接口说明

查询当前应用从腾讯移动推送服务器生成的 Token 字符串。



```
@property (copy, nonatomic, nullable, readonly) NSString *xgTokenString;
```

示例代码



```
NSString *token = [[XGPushTokenManager defaultTokenManager] xgTokenString];
```

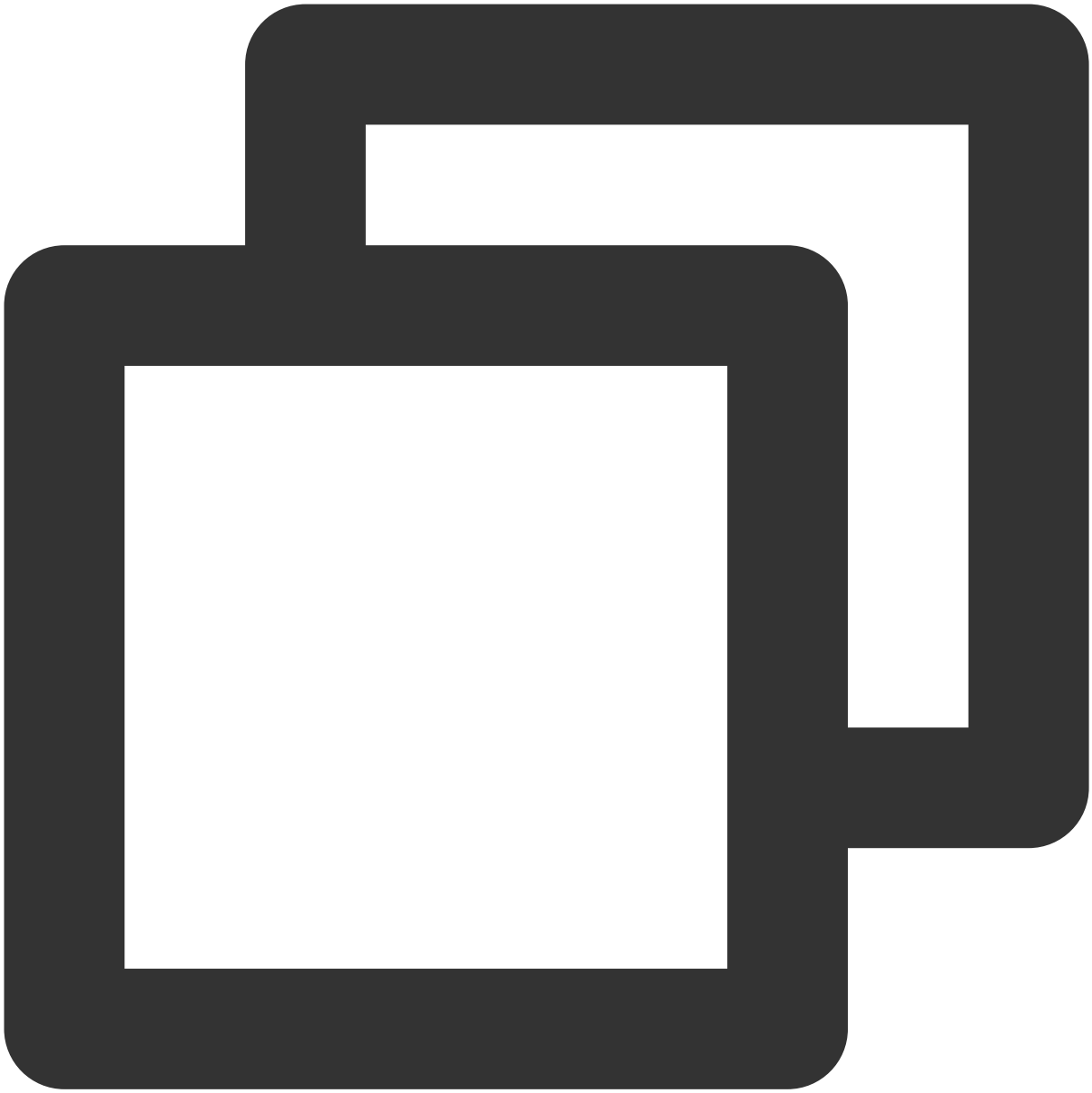
注意：

token 的获取应该在 xgPushDidRegisteredDeviceToken:error: 返回正确之后被调用。

注册结果回调

接口说明

SDK 启动之后，通过此方法回调来返回注册结果及 Token。



```
- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nu
```

返回参数说明

deviceToken：APNs 生成的 Device Token。

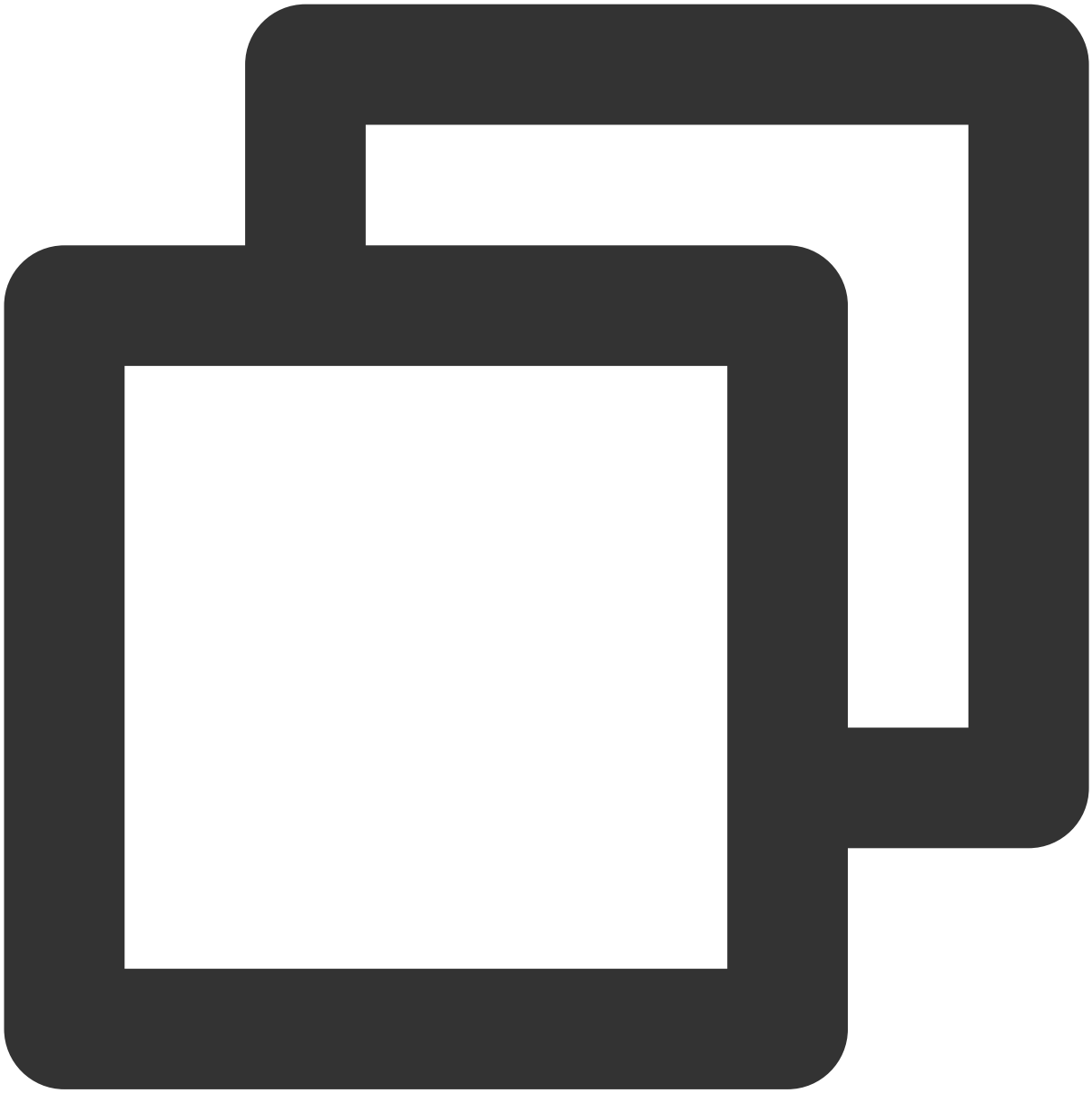
xgToken：移动推送生成的 Token，推送消息时需要使用此值。移动推送维护此值与 APNs 的 Device Token 的映射关系。

error：错误信息，若 error 为 nil，则注册推送服务成功。

注册失败回调

接口说明

SDK 1.2.7.2 新增，当注册推送服务失败会走此回调。

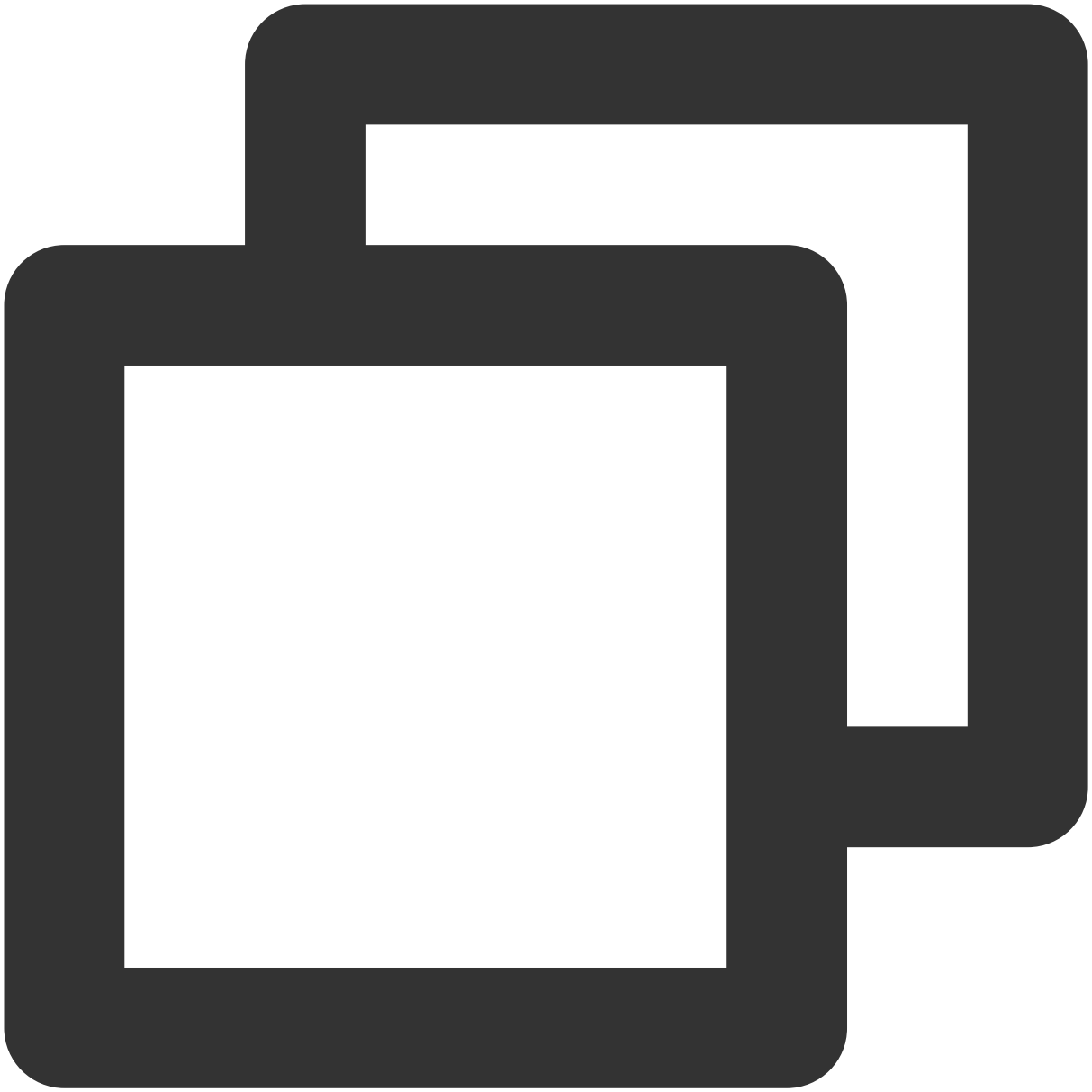


```
/// @note TPNS SDK1.2.7.2+
- (void)xgPushDidFailToRegisterDeviceTokenWithError:(nullable NSError *)error
```

通知授权弹窗的回调

接口说明

SDK 1.3.1.0 新增，通知弹窗授权的结果会走此回调。



```
- (void)xgPushDidRequestNotificationPermission:(bool)isEnable error:(nullable NSErr
```

返回参数说明

isEnable：是否同意授权。

error：错误信息，若 error 为 nil，则获取弹窗结果成功。

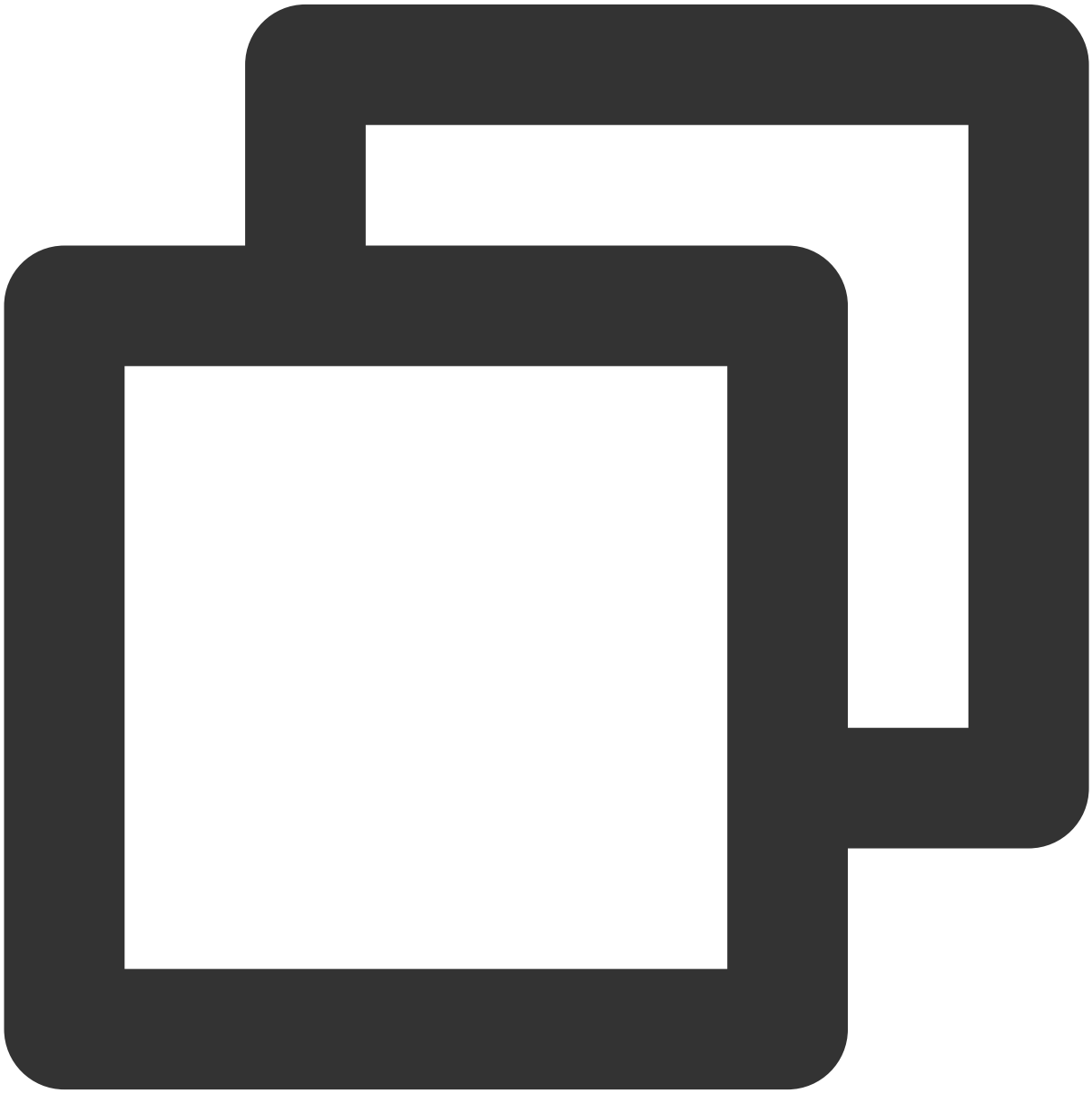
账号功能

以下为账号相关接口方法，若需了解调用时机及调用原理，可查看 账号相关流程。

添加账号

接口说明

若原来没有该类型账号，则添加；若原来有，则覆盖。（移动推送 SDK1.2.9.0+ 新增）



```
- (void)upsertAccountsByDict:(nonnull NSDictionary<NSNumber *, NSString *> *)accoun
```

说明：

此接口应该在 xgPushDidRegisteredDeviceToken:error: 返回正确之后被调用。

参数说明

accountsDict：账号字典。

说明：

账号类型和账号名称一起作为联合主键。

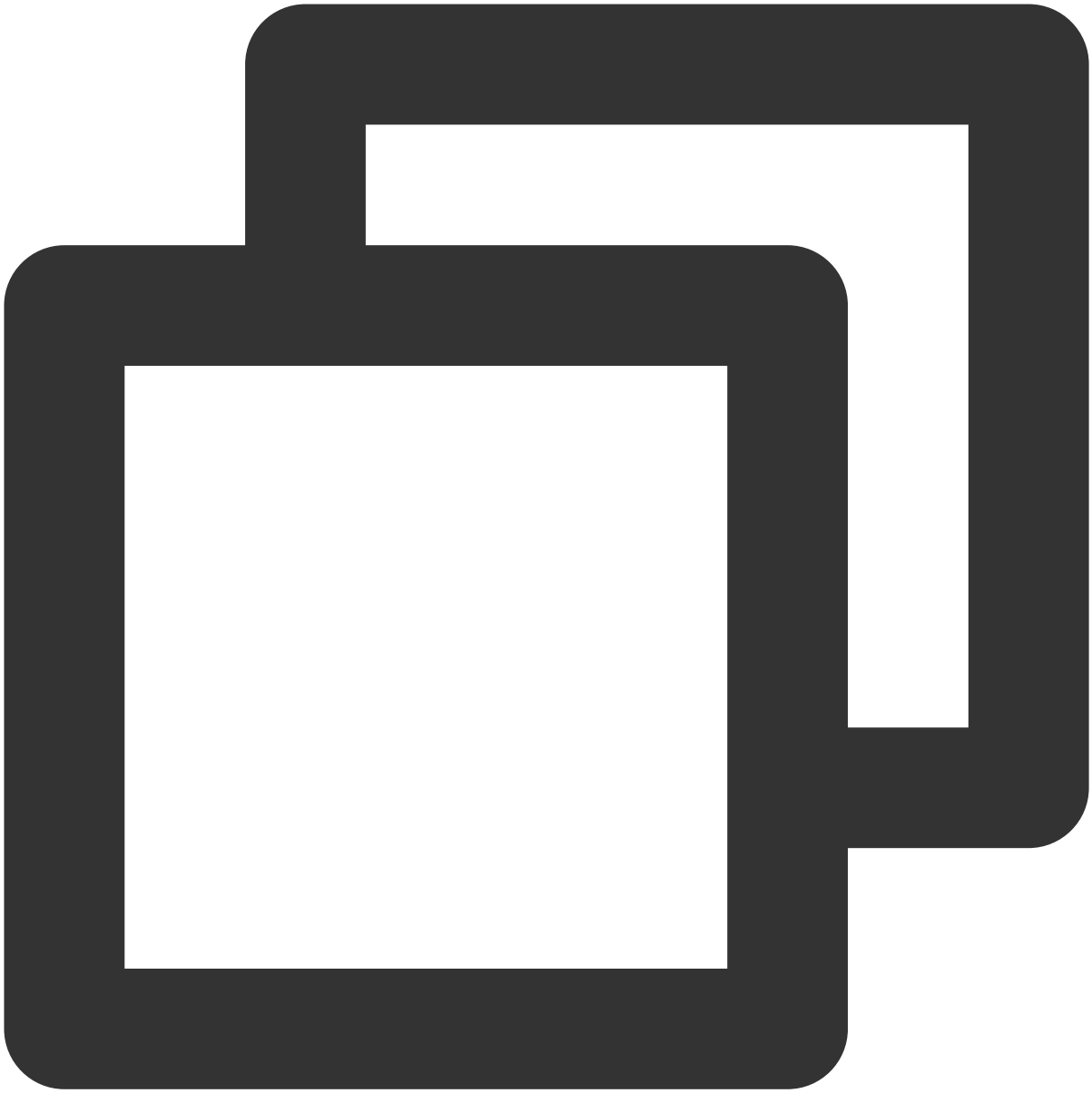
需要使用字典类型，key 为账号类型，value 为账号，示例：@{@(accountType):@"account"}。

Objective-C的写法 : @{@(0):@"account0",@(1):@"account1"}；Swift的写法：

[NSNumber(0):@"account0",NSNumber(1):@"account1"]。

更多 accountType 请参照 SDK Demo 包内的 XGPushTokenAccountType 枚举或 账号类型取值表。

示例代码

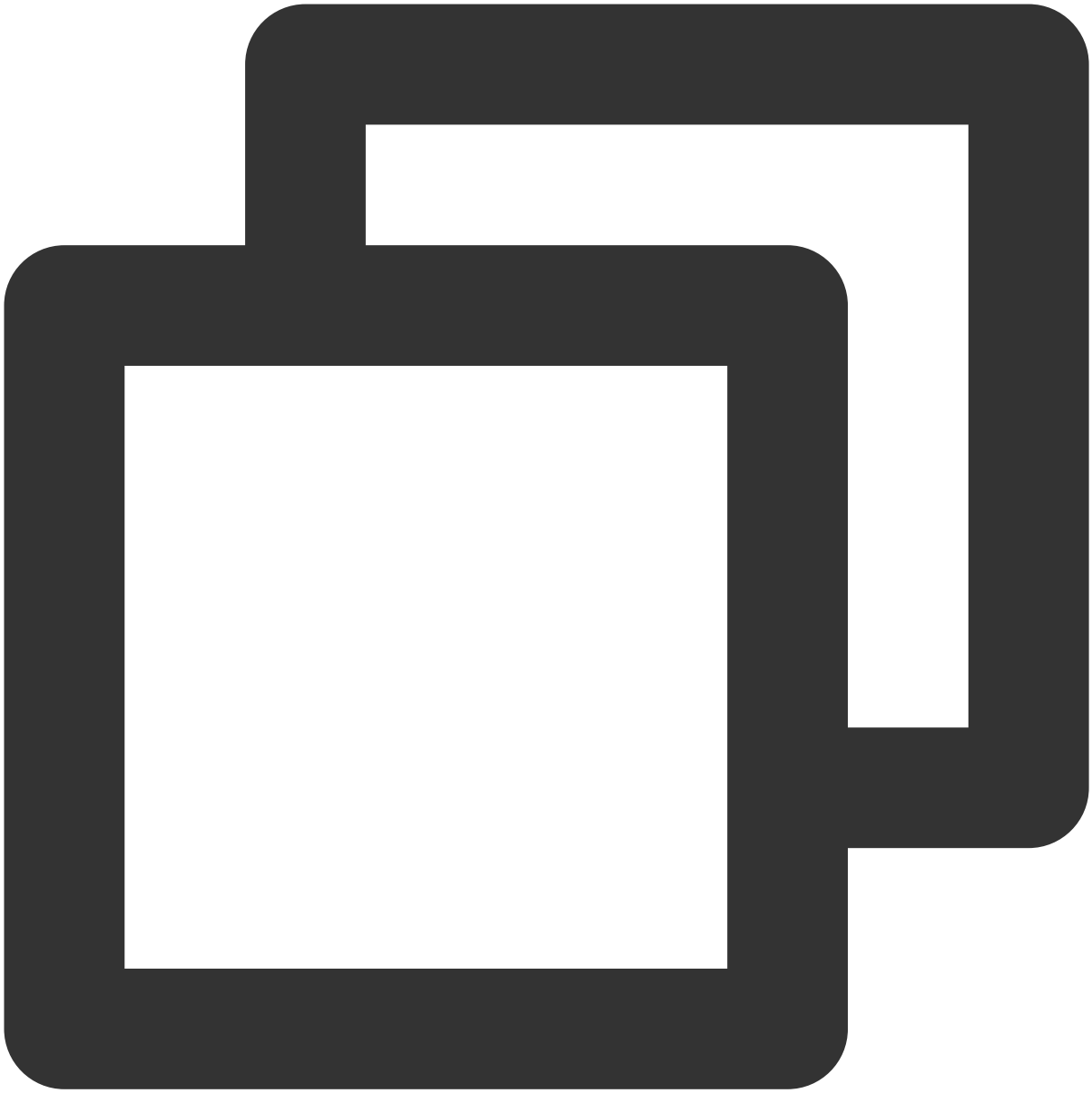


```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;
NSString *account = @"account";
[[XGPushTokenManager defaultTokenManager] upsertAccountsByDict:@{ @(accountType):ac
```

添加手机号

接口说明

添加或更新用户手机号，等于调用 `upsertAccountsByDict:@{@(1002):@"具体手机号"}` 。

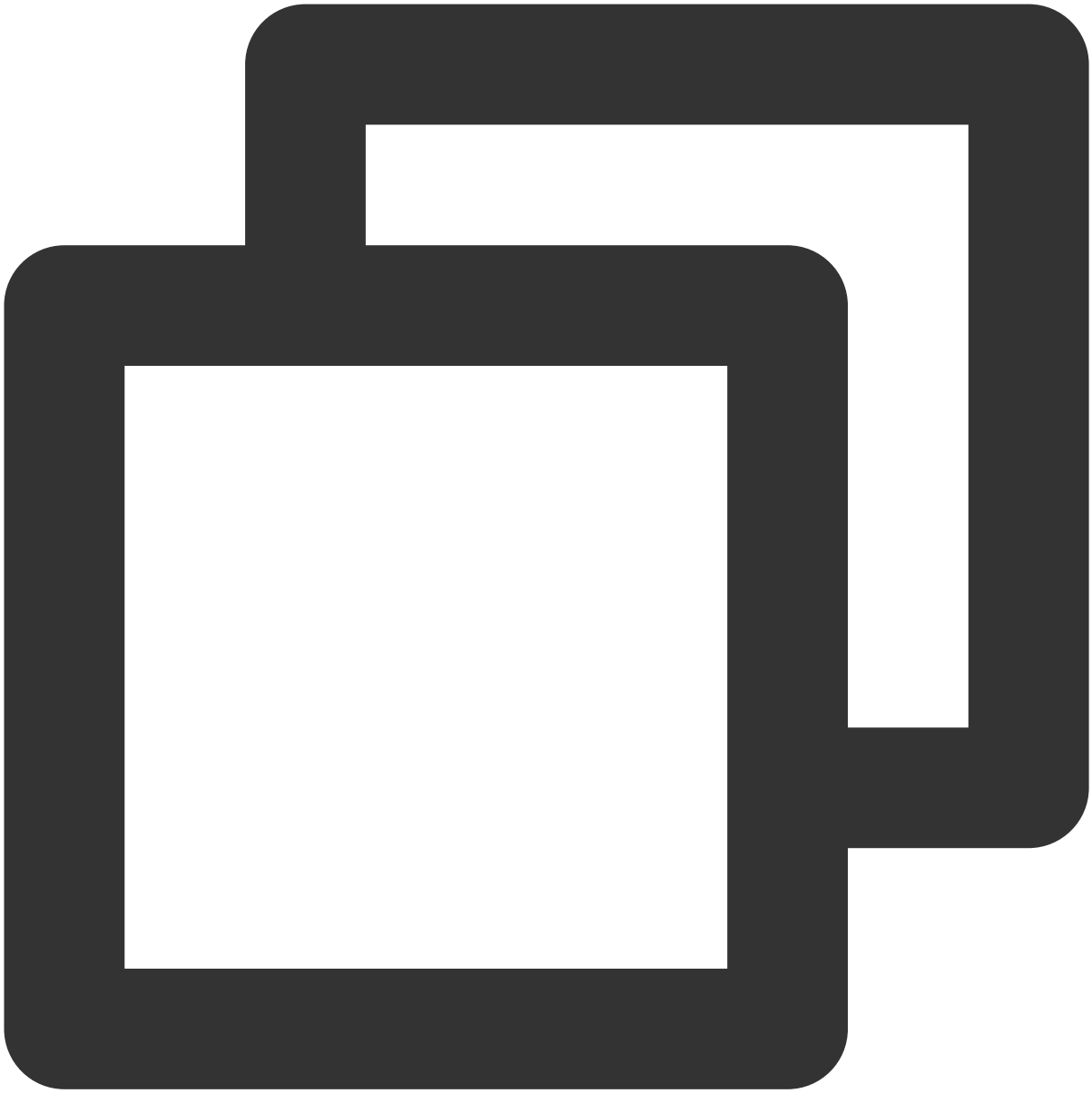


```
/// @note TPNS SDK1.3.2.0+
- (void)upsertPhoneNumber:(nonnull NSString *)phoneNumber;
```

参数说明

phoneNumber：E.164标准，格式为+[国家或地区码][手机号],例如+8613711112222。SDK内部加密传输。

示例代码



```
[[XGPushTokenManager defaultTokenManager] upsertPhoneNumber:@"+8613712345678"];;
```

注意：

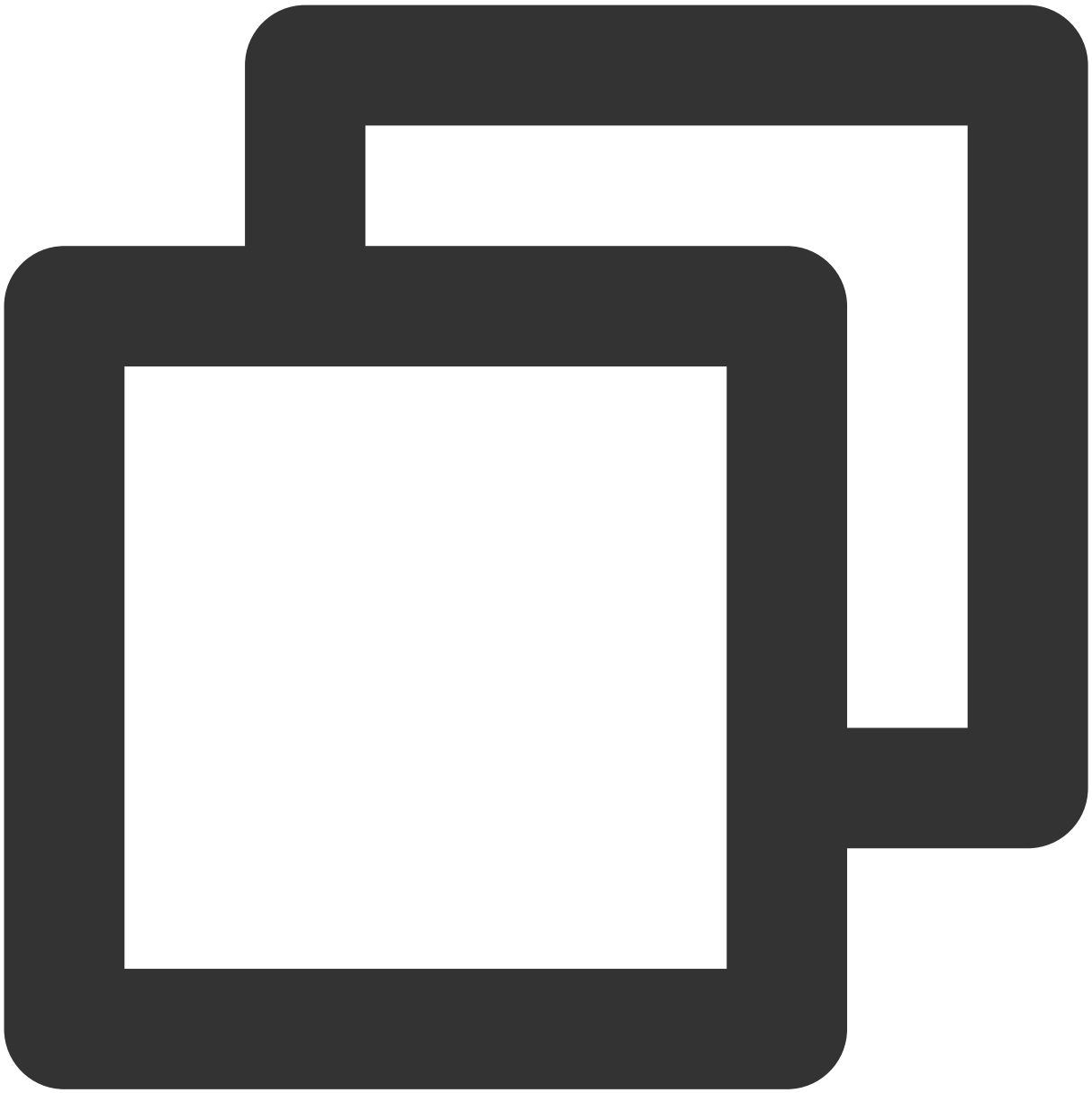
1.此接口应该在xgPushDidRegisteredDeviceToken:error:返回正确之后被调用

2.如需要删除手机号，调用 `delAccountsByKeys:[[NSSet alloc] initWithObjects:@(1002), nil]`

删除账号

接口说明

接口说明：删除指定账号类型下的所有账号。（移动推送 SDK1.2.9.0+ 新增）



```
- (void)delAccountsByKeys:(nonnull NSSet<NSNumber *> *)accountsKeys;
```

说明：

此接口应该在 xgPushDidRegisteredDeviceToken:error: 返回正确之后被调用。

参数说明

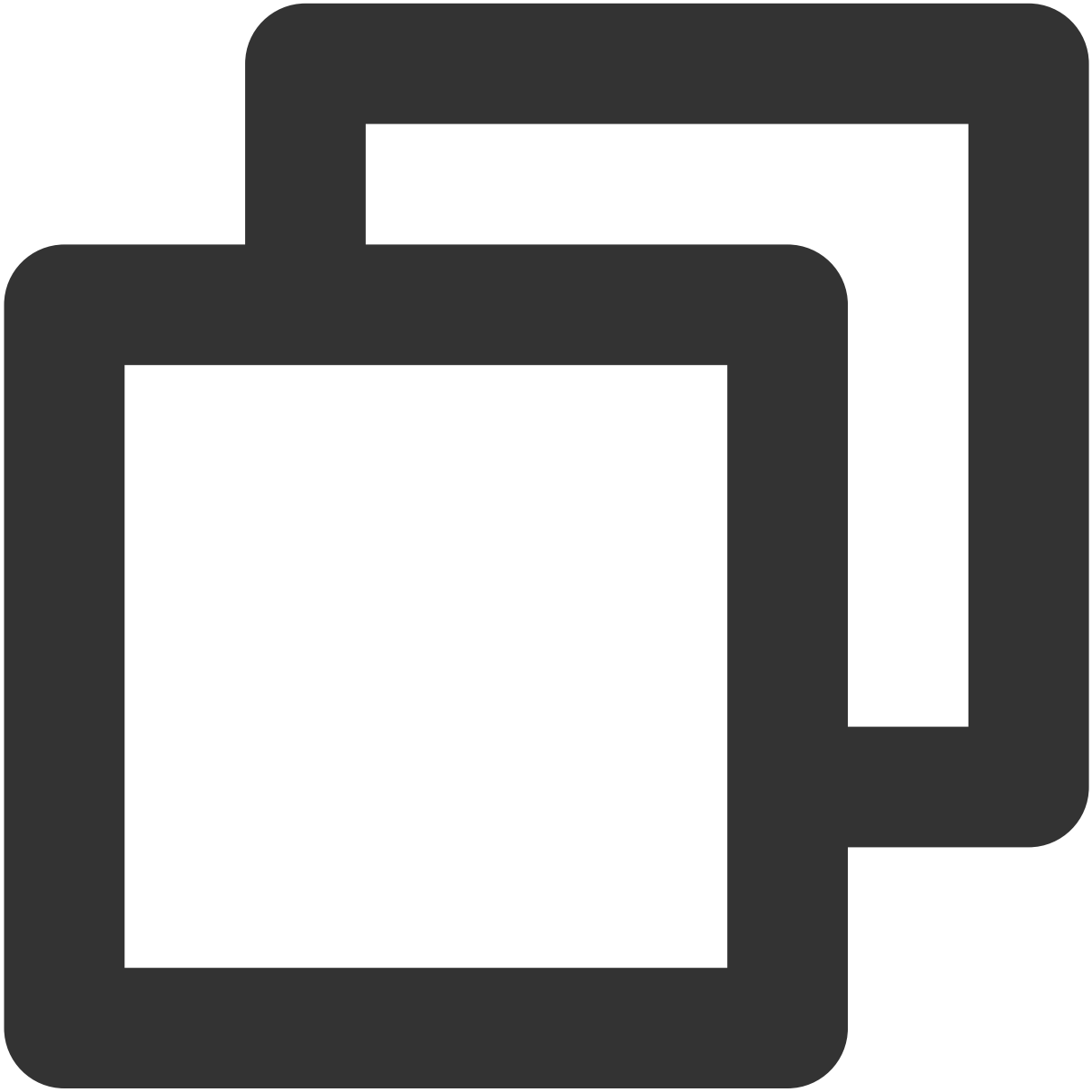
accountsKeys：账号类型组成的集合。

说明：

使用集合且 key 是固定要求。

更多 accountType 请参照 SDK 包内 XGPush.h 文件中的 XGPushTokenAccountType 枚举。

示例代码

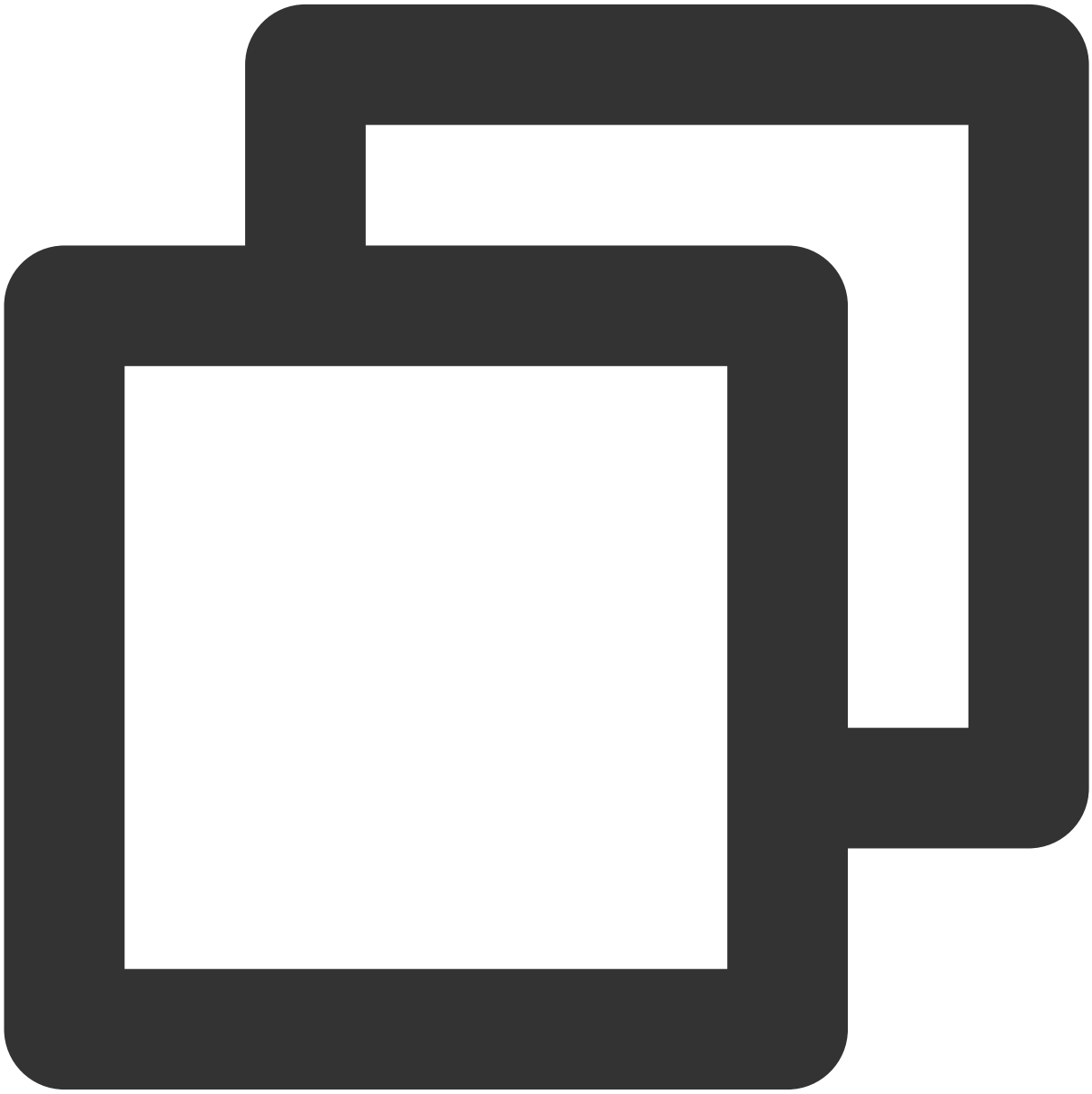


```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;

NSSet *accountsKeys = [[NSSet alloc] initWithObjects:@(accountType), nil];

[[XGPushTokenManager defaultTokenManager] delAccountsByKeys:accountsKeys];
```

清除账号

接口说明

清除所有设置的账号。

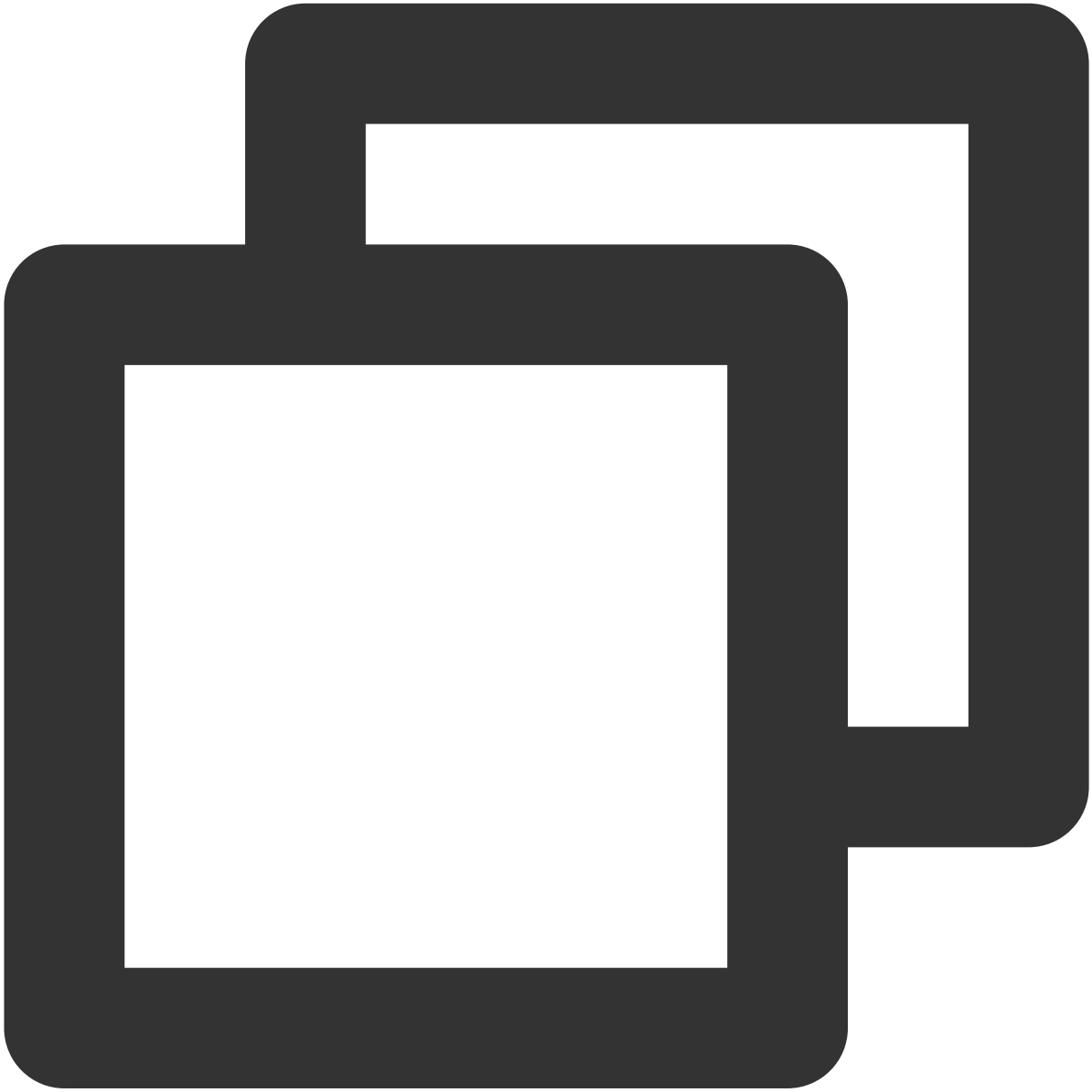


```
- (void)clearAccounts;
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码



```
[[XGPushTokenManager defaultTokenManager] clearAccounts];
```

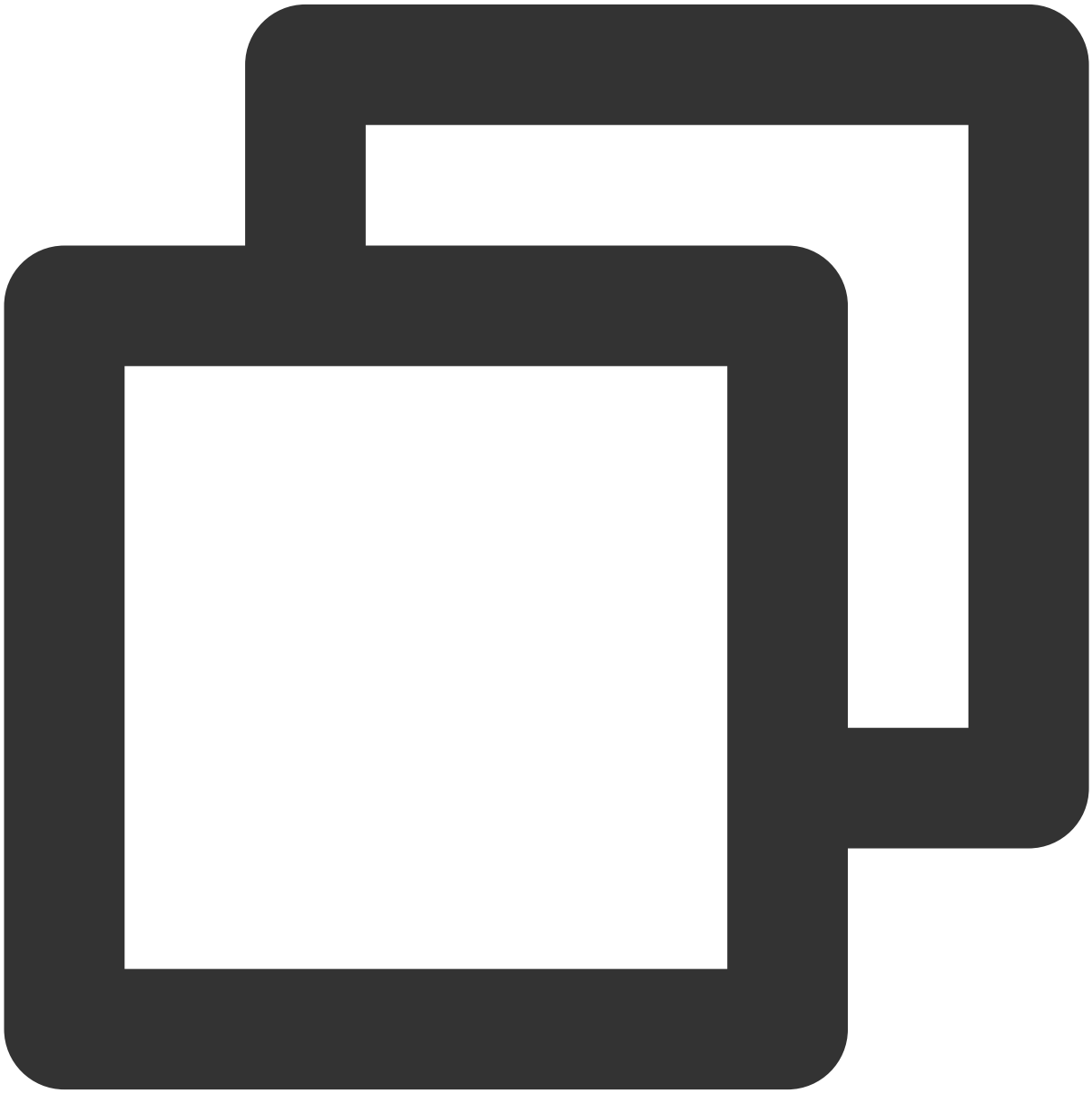
标签功能

以下为标签相关接口方法，若需了解调用时机及调用原理，可查看 标签相关流程。

绑定/解绑标签

接口说明

开发者可以针对不同的用户绑定标签，然后对该标签进行推送。



```
- (void)appendTags:(nonnull NSArray<NSString *> *)tags
```

```
- (void)delTags:(nonnull NSArray<NSString *> *)tags
```

说明：

此接口为追加方式。

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

单个应用最多可以有10000个自定义 tag， 每个设备 Token 最多可绑定100个自定义 tag，如需提高该限制，请 提交工单，每个自定义 tag 可绑定的设备 Token 数量无限制。

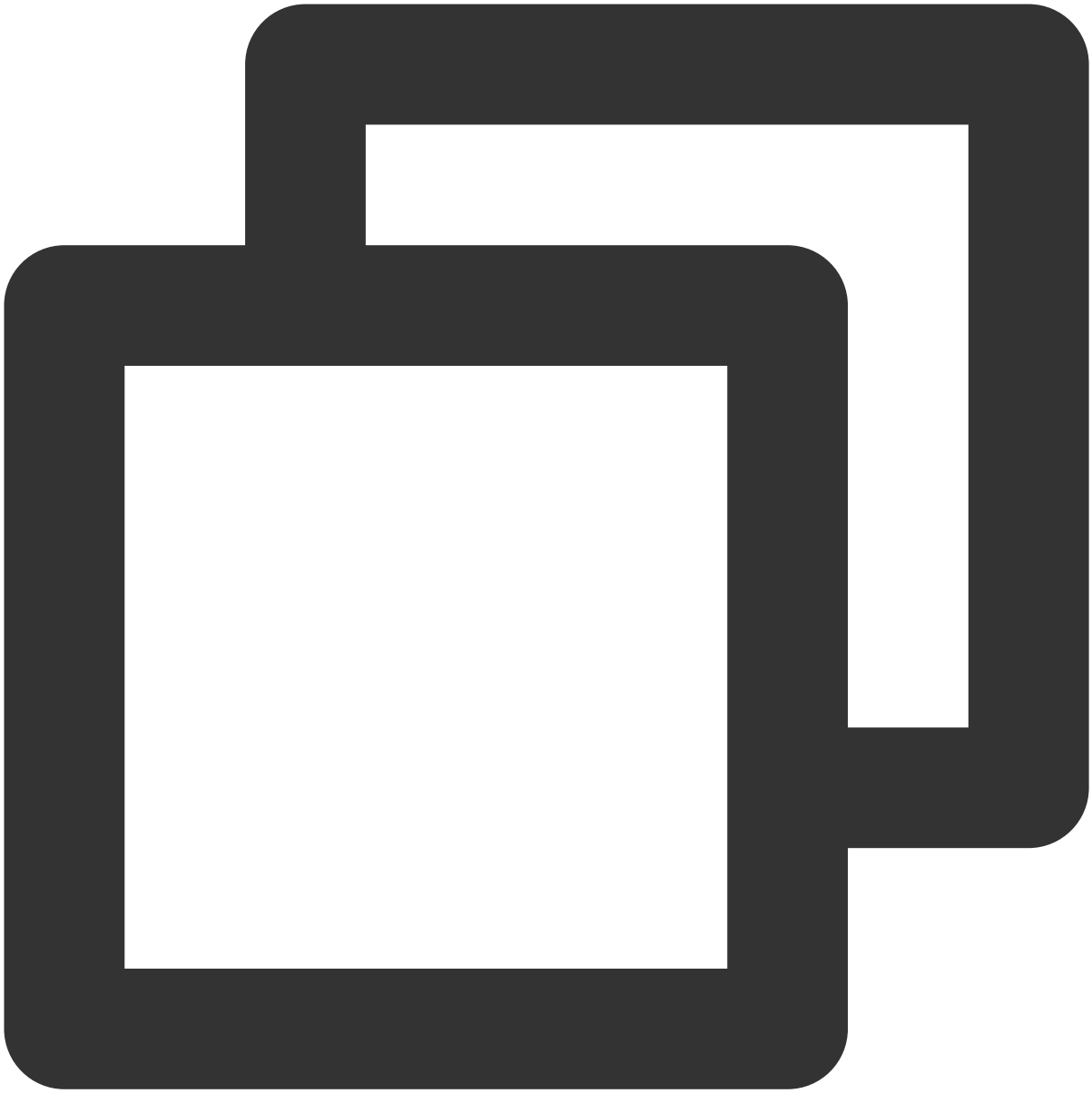
参数说明

tags：标签数组。

说明：

标签操作 tags 为标签字符串数组（标签字符串不允许有空格或者是 tab 字符）。

示例代码

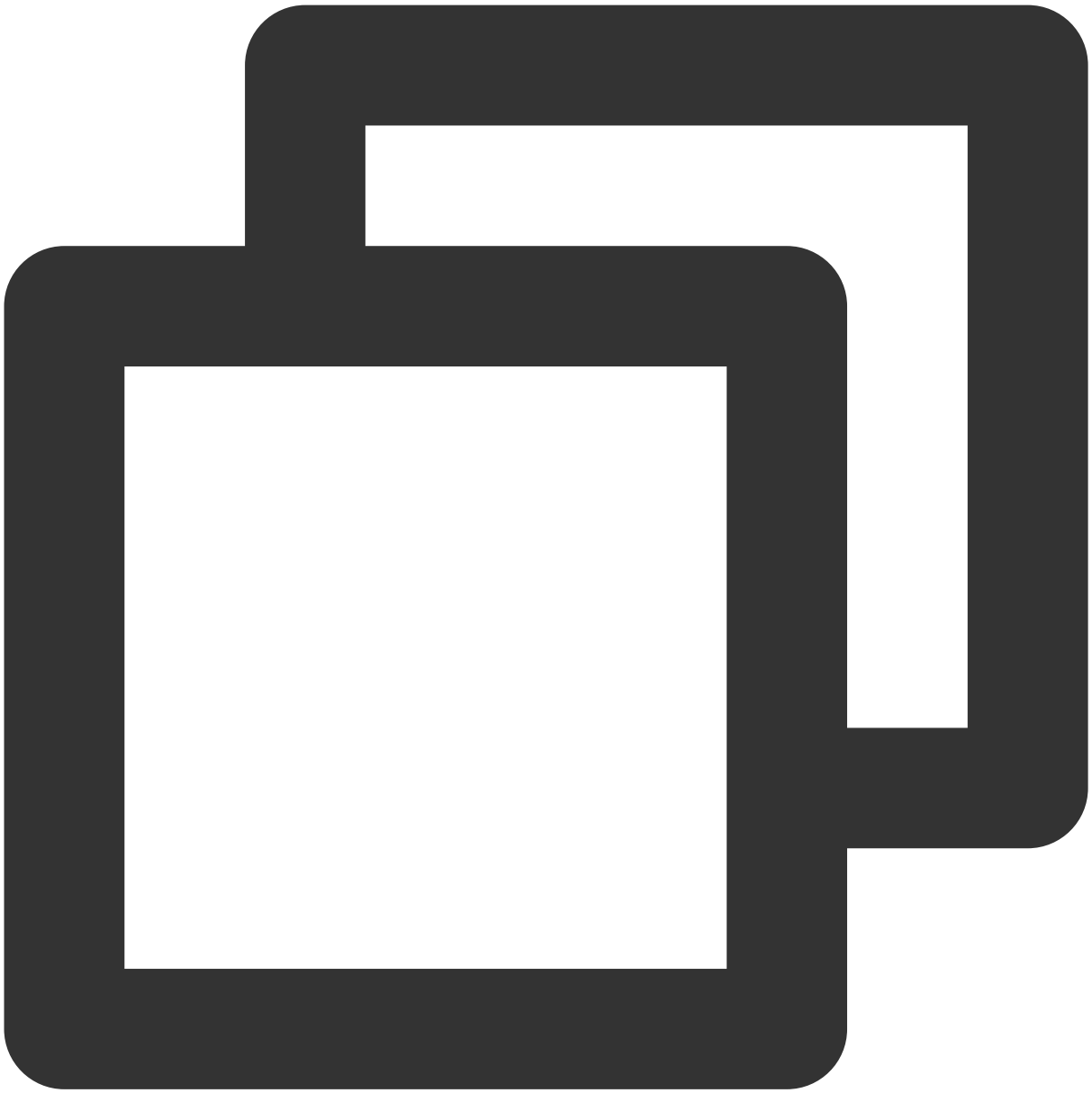


```
//绑定标签：
[[XGPushTokenManager defaultTokenManager] appendTags:@[ tagStr ]];

//解绑标签
[[XGPushTokenManager defaultTokenManager] delTags:@[ tagStr ]];
```

更新标签

接口说明

清空已有标签，然后批量添加标签。



```
- (void)clearAndAppendTags:(nonnull NSArray<NSString *> *)tags
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

此接口会将当前 Token 对应的旧有的标签全部替换为当前的标签。

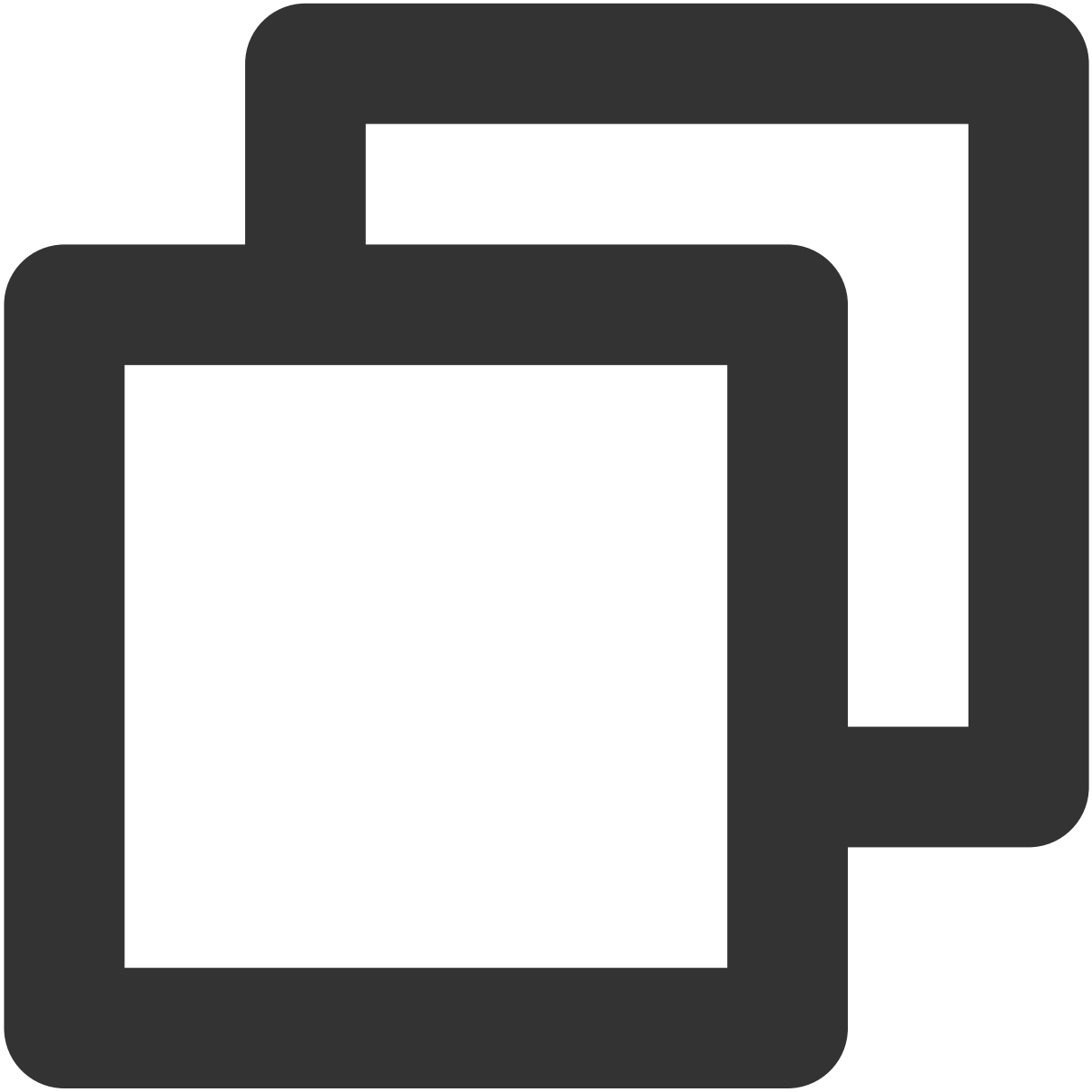
参数说明

tags：标签数组。

说明：

标签操作 tags 为标签字符串数组（标签字符串不允许有空格或者是 tab 字符）。

示例代码

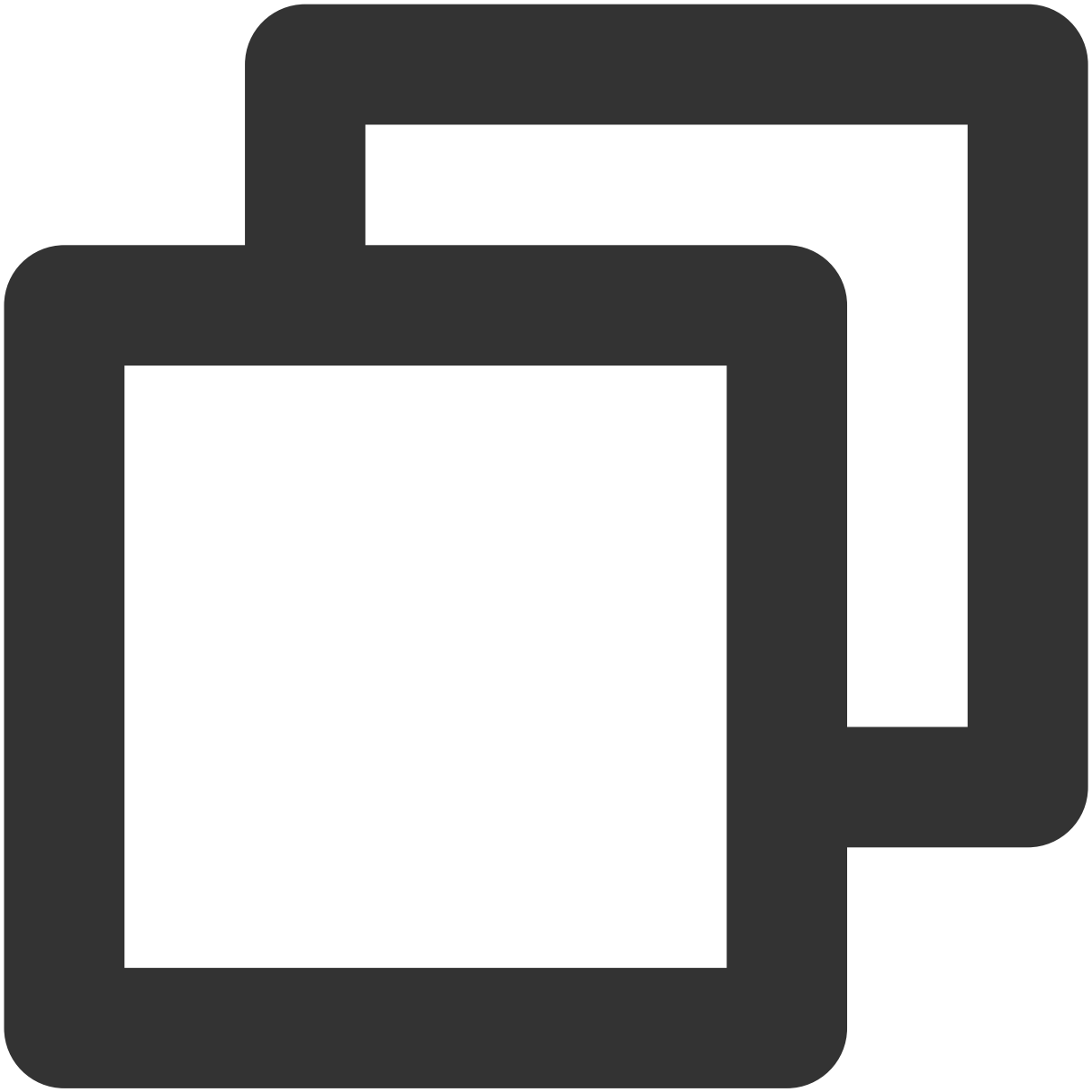


```
[[XGPushTokenManager defaultTokenManager] clearAndAppendTags:@[ tagStr ]];
```

清除全部标签

接口说明

清除所有设置的标签。

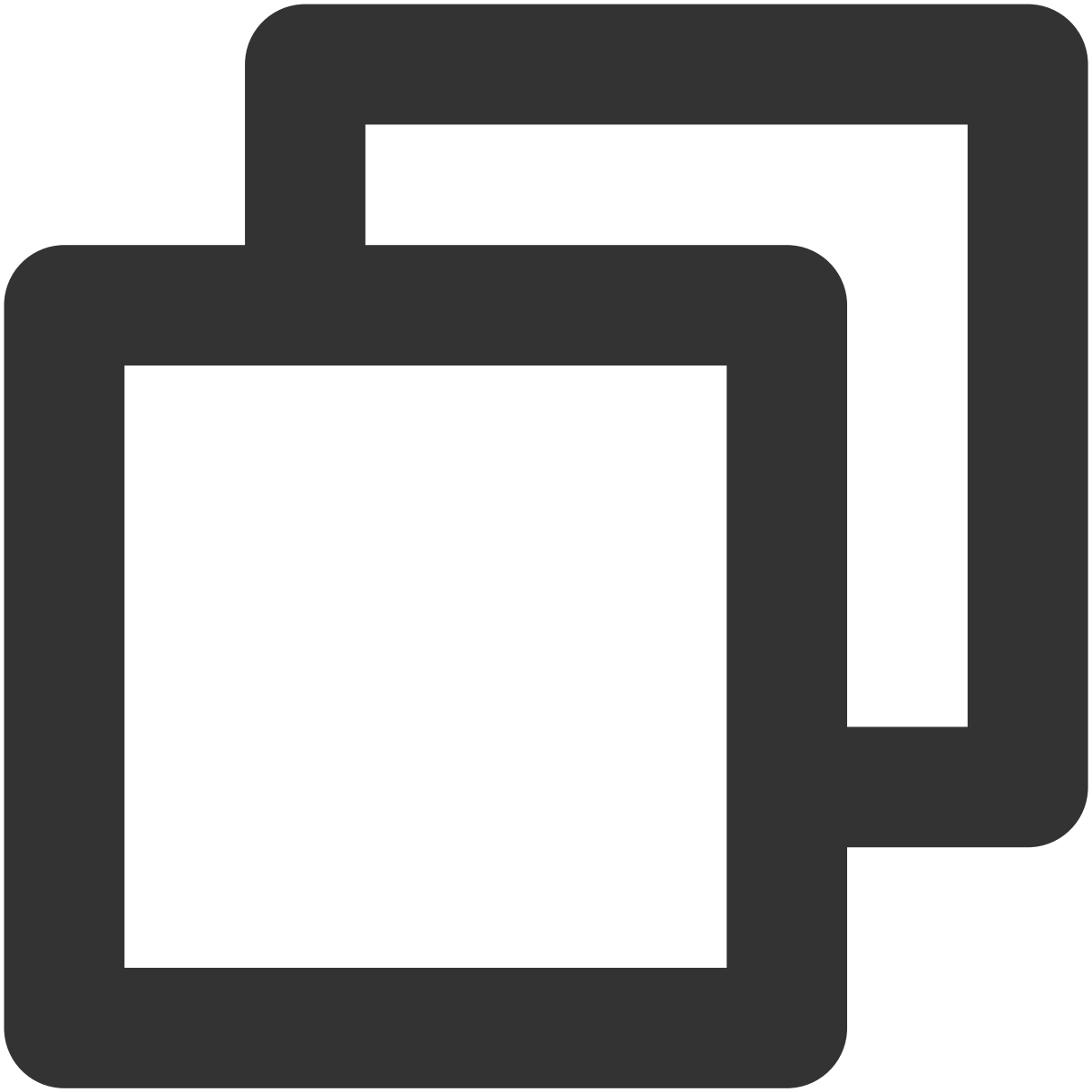


```
- (void)clearTags
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码

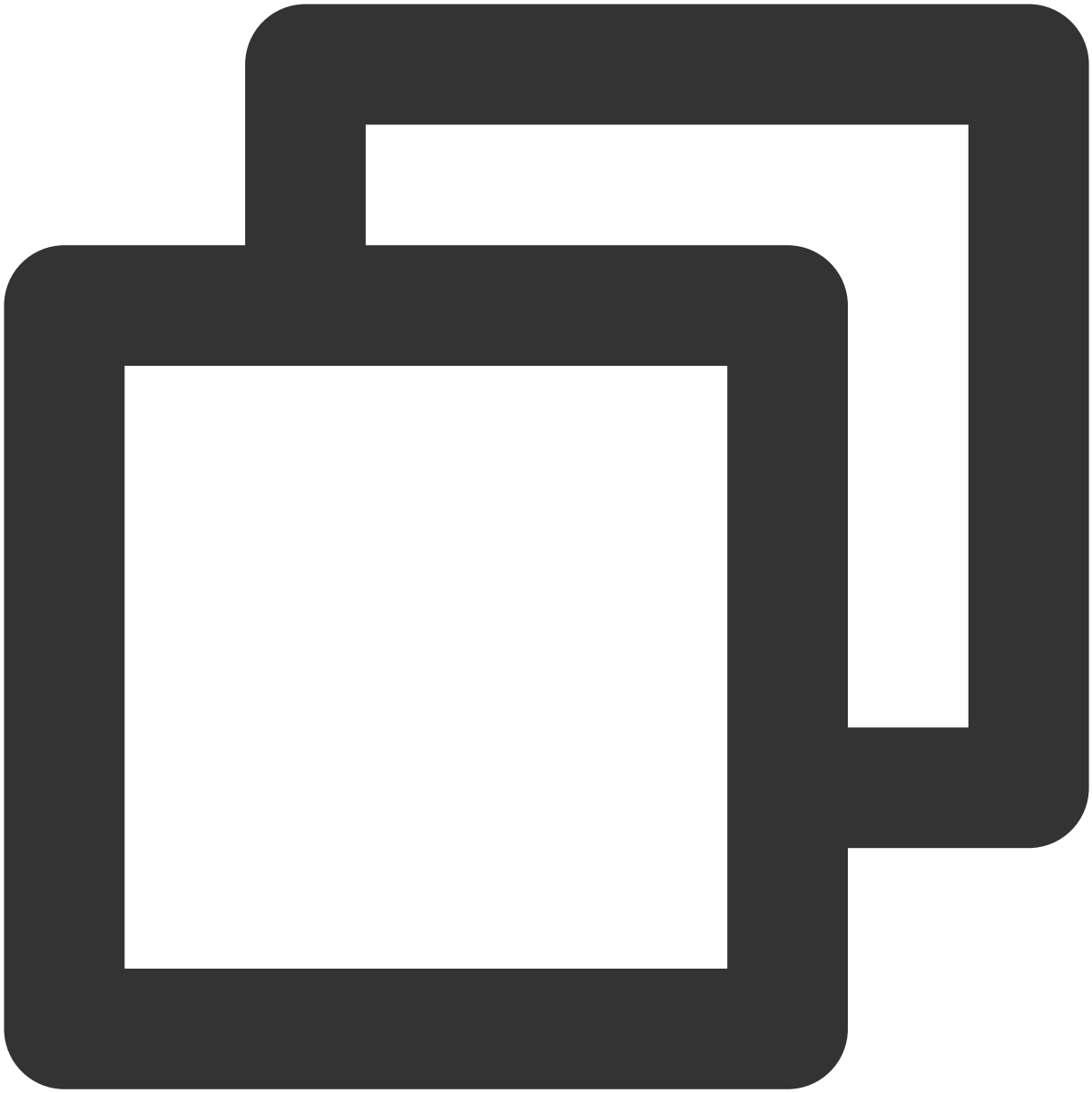


```
[[XGPushTokenManager defaultTokenManager] clearTags];
```

查询标签

接口说明

SDK 1.3.1.0 新增，查询设备绑定的标签。



```
- (void)queryTags:(NSUInteger)offset limit:(NSUInteger)limit;
```

说明：

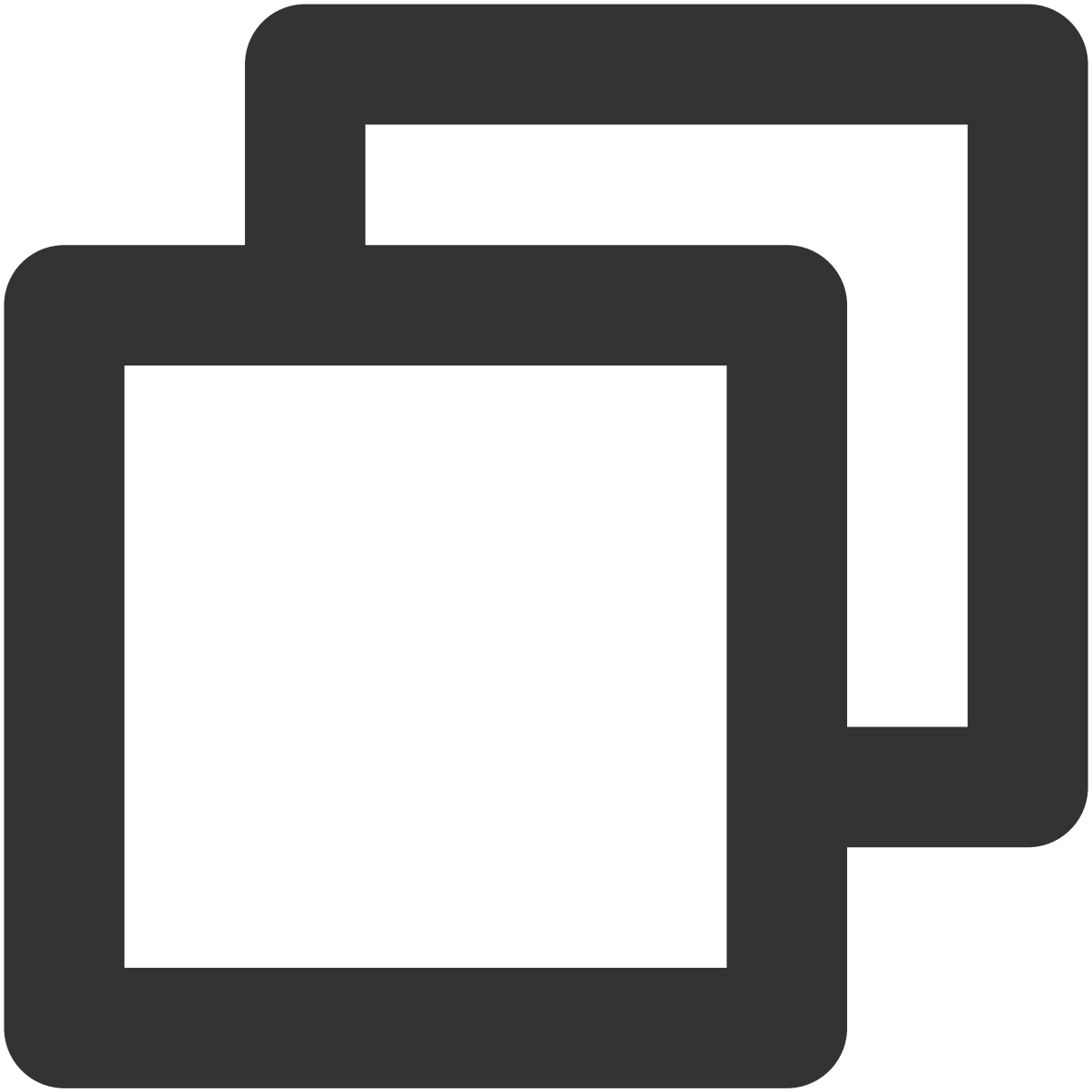
此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

参数说明

offset：此次查询的偏移大小。

offset：limit 此次查询的分页大小, 最大200。

示例代码

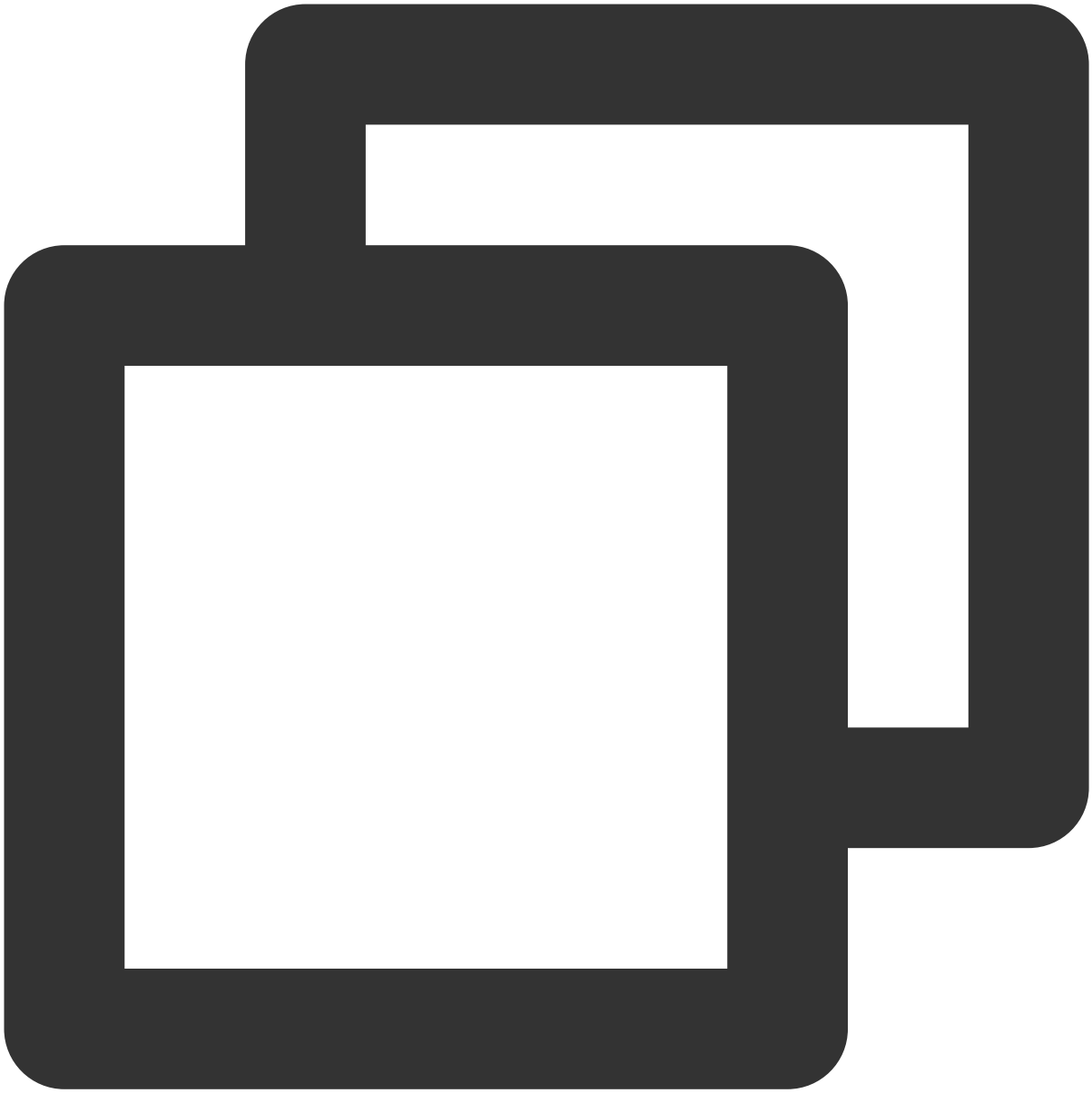


```
[[XGPushTokenManager defaultTokenManager] queryTags:0 limit:100];
```

查询标签的回调

接口说明

SDK 1.3.1.0 新增，查询标签的结果会走此回调。



```
- (void)xgPushDidQueryTags:(nullable NSArray<NSString *> *)tags totalCount:(NSUInte
```

返回参数说明

tags：查询条件返回的标签。

totalCount：设备绑定的总标签数量。

error：错误信息，若 error 为 nil，则查询成功。

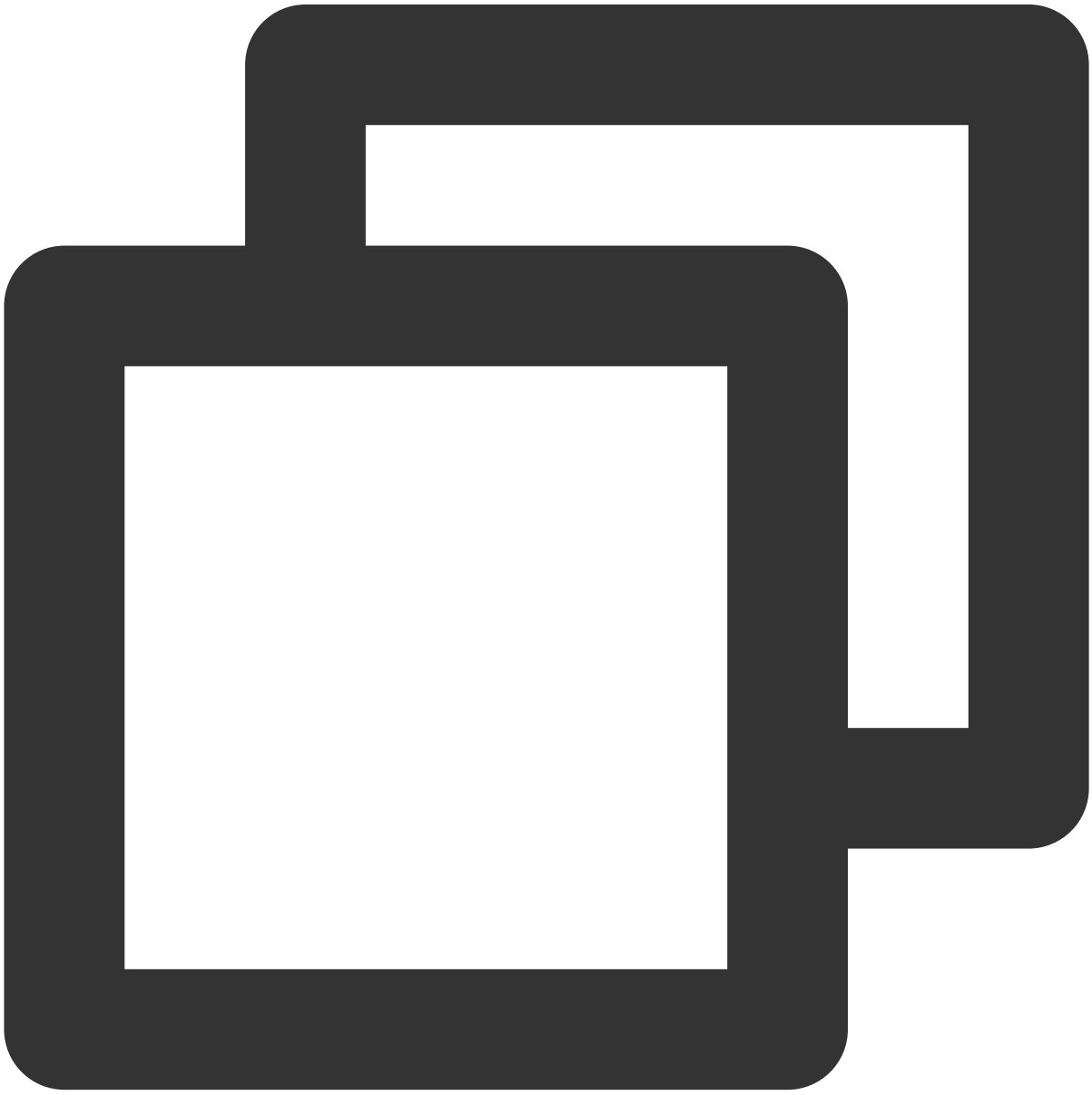
用户属性功能

以下为用户属性相关接口方法，若需了解调用时机及调用原理，可查看 用户属性相关流程。

新增用户属性

接口说明

添加或更新用户属性（key-value 结构，若原来没有该 key 的用户属性 value，则新增；若原来有该 key 的用户属性 value，则更新该 value）。



```
- (void)upsertAttributes:(nonnull NSDictionary<NSString *,NSString *> *)attributes
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

参数说明

attributes：用户属性字符串字典，字符串不允许有空格或者是 tab 字符。

说明：

需要先在管理台配置用户属性的键，才能操作成功。

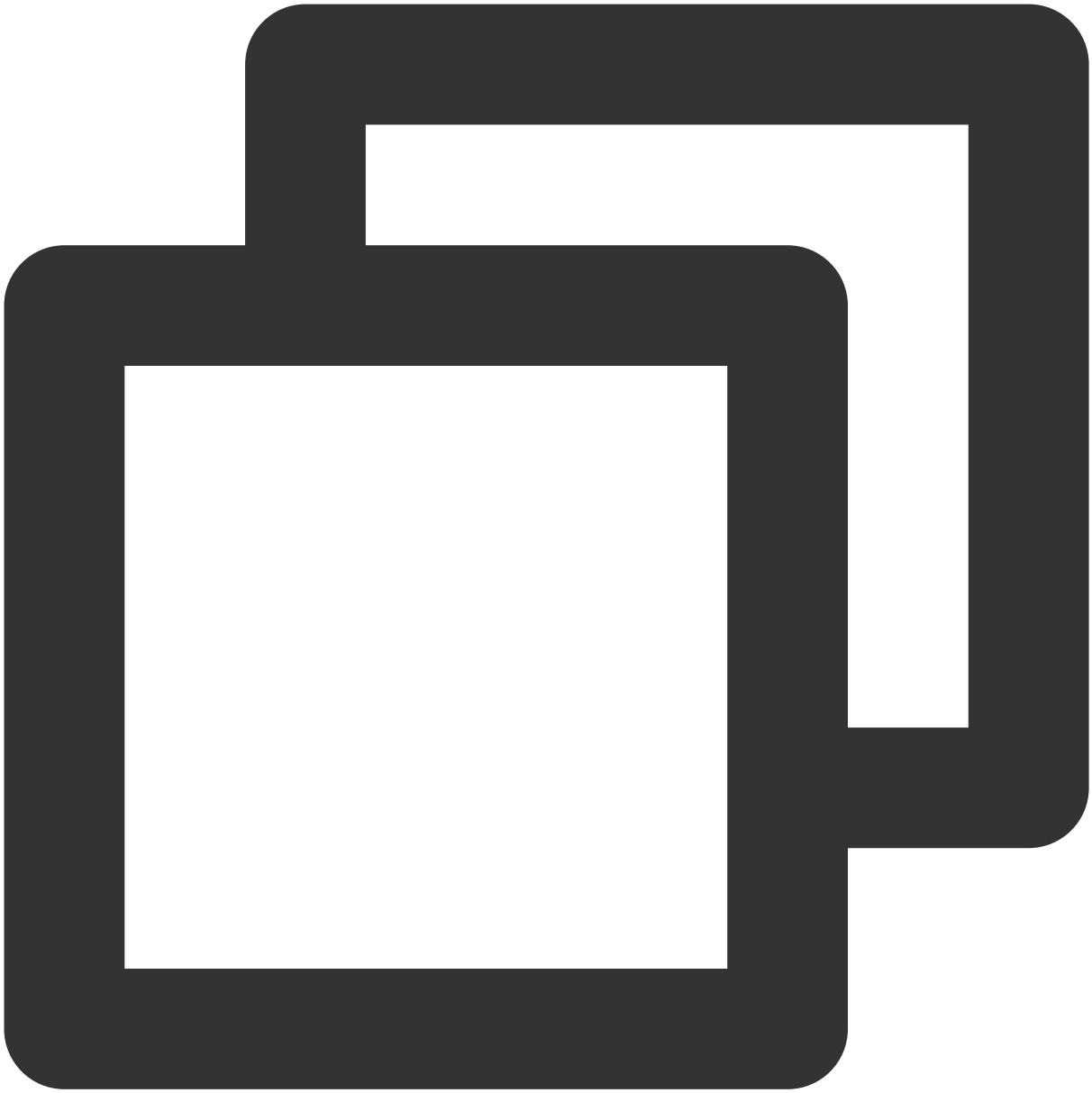
key，value 长度都限制50个字符以内。

需要使用字典且 key 是固定要求。

Objective-C 的写法 : @{@"gender": @"Female", @"age": @"29"}；

Swift 的写法：["gender":"Female", "age": "29"]

示例代码

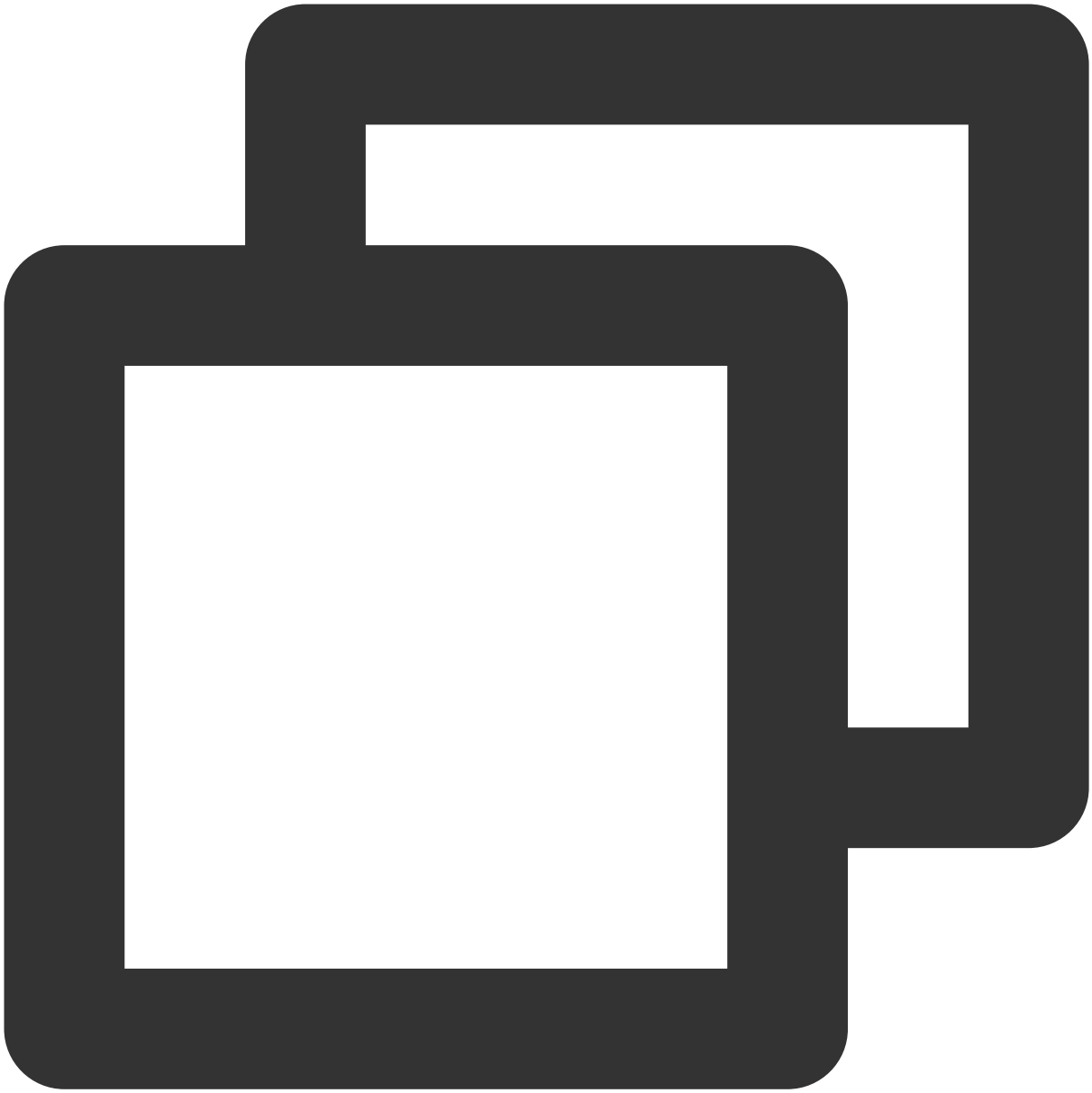


```
[[XGPushTokenManager defaultTokenManager] upsertAttributes:attributes];
```

删除用户属性

接口说明

删除用户已有的属性。



```
- (void)delAttributes:(nonnull NSSet<NSString *> *)attributeKeys
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

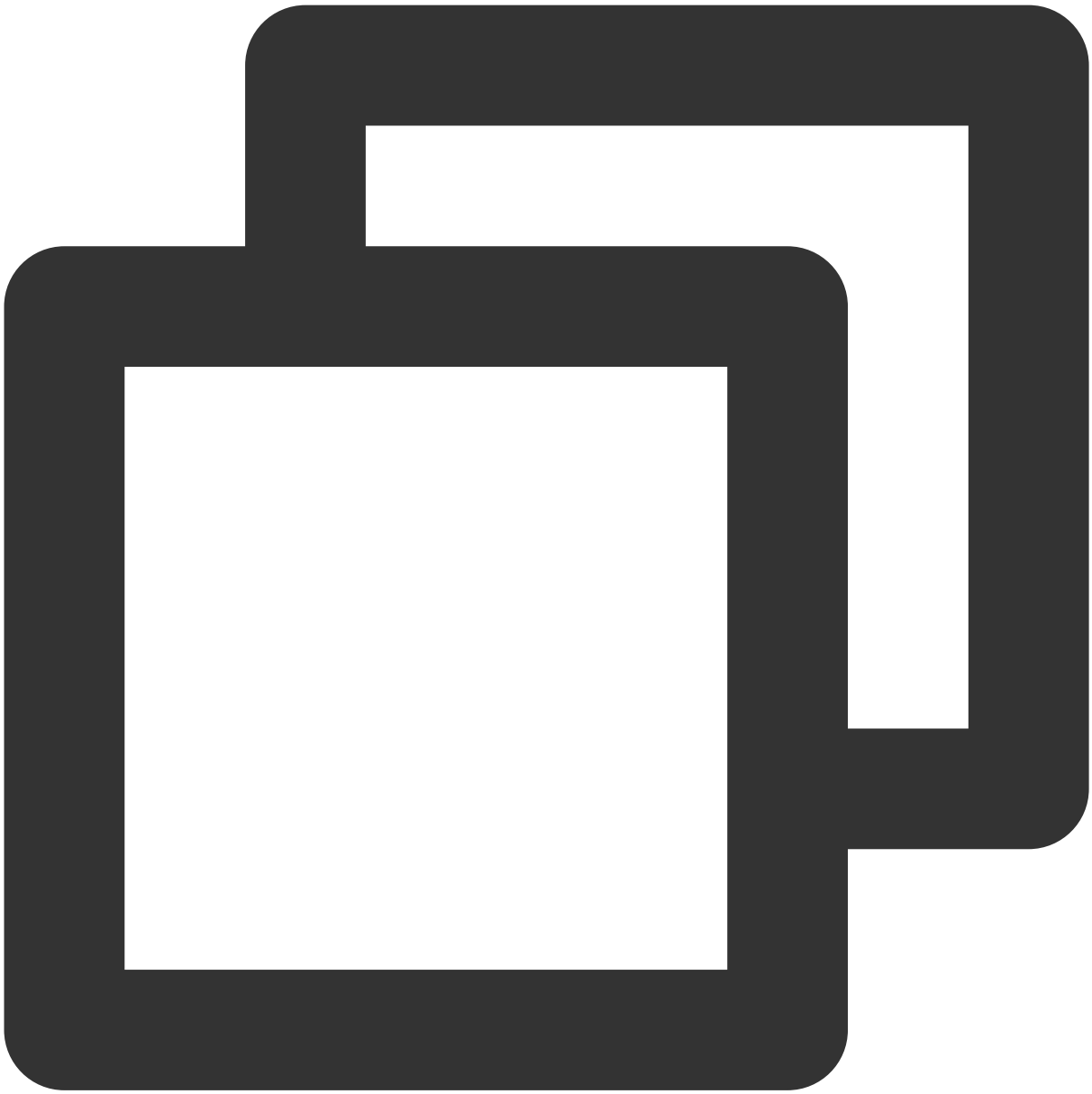
参数说明

attributeKeys：用户属性 key 组成的集合，字符串不允许有空格或者是 tab 字符。

说明：

使用集合且key是固定要求。

示例代码

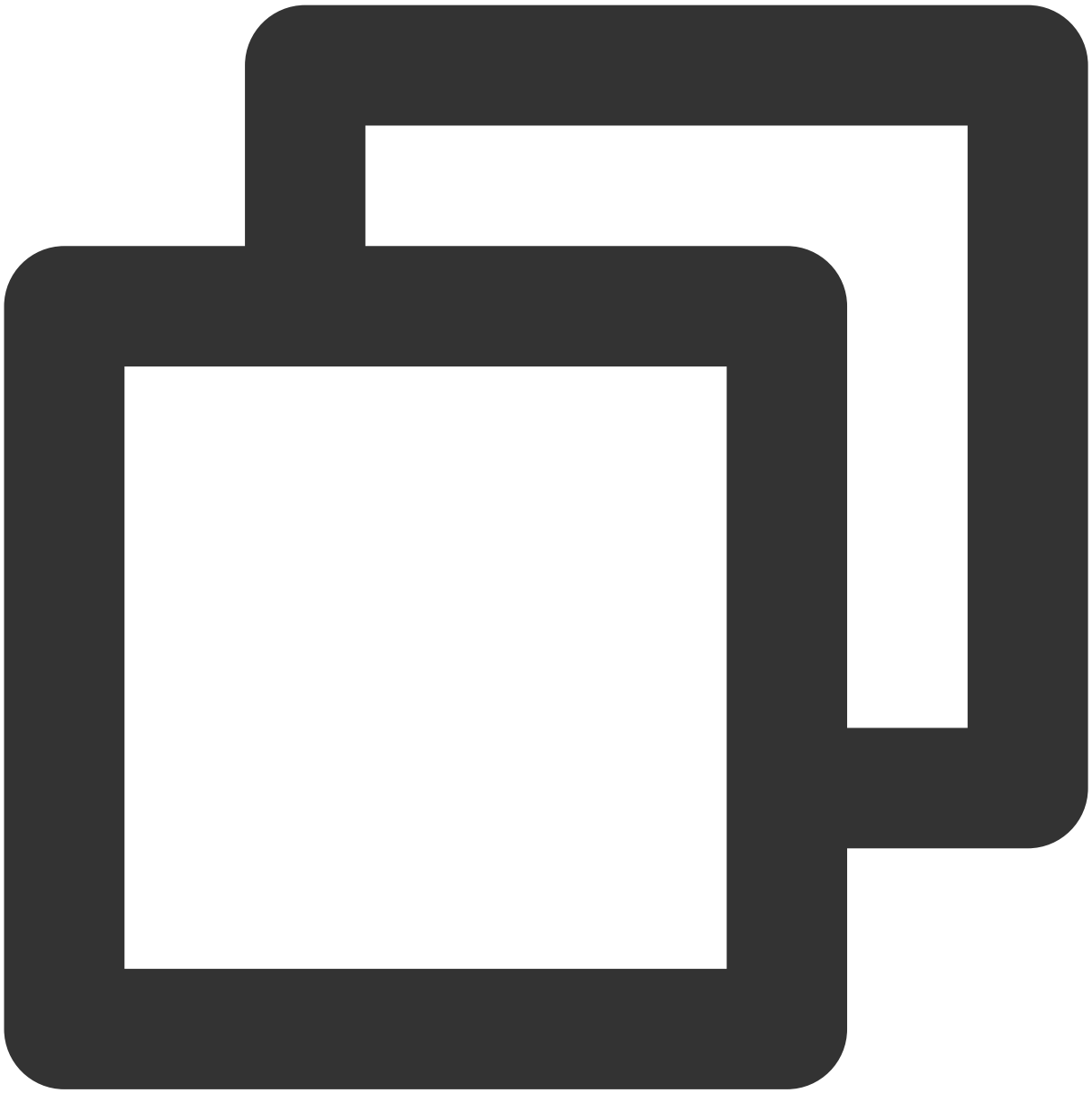


```
[[XGPushTokenManager defaultTokenManager] delAttributes:attributeKeys];
```

清空已有用户属性

接口说明

清空已有用户属性。

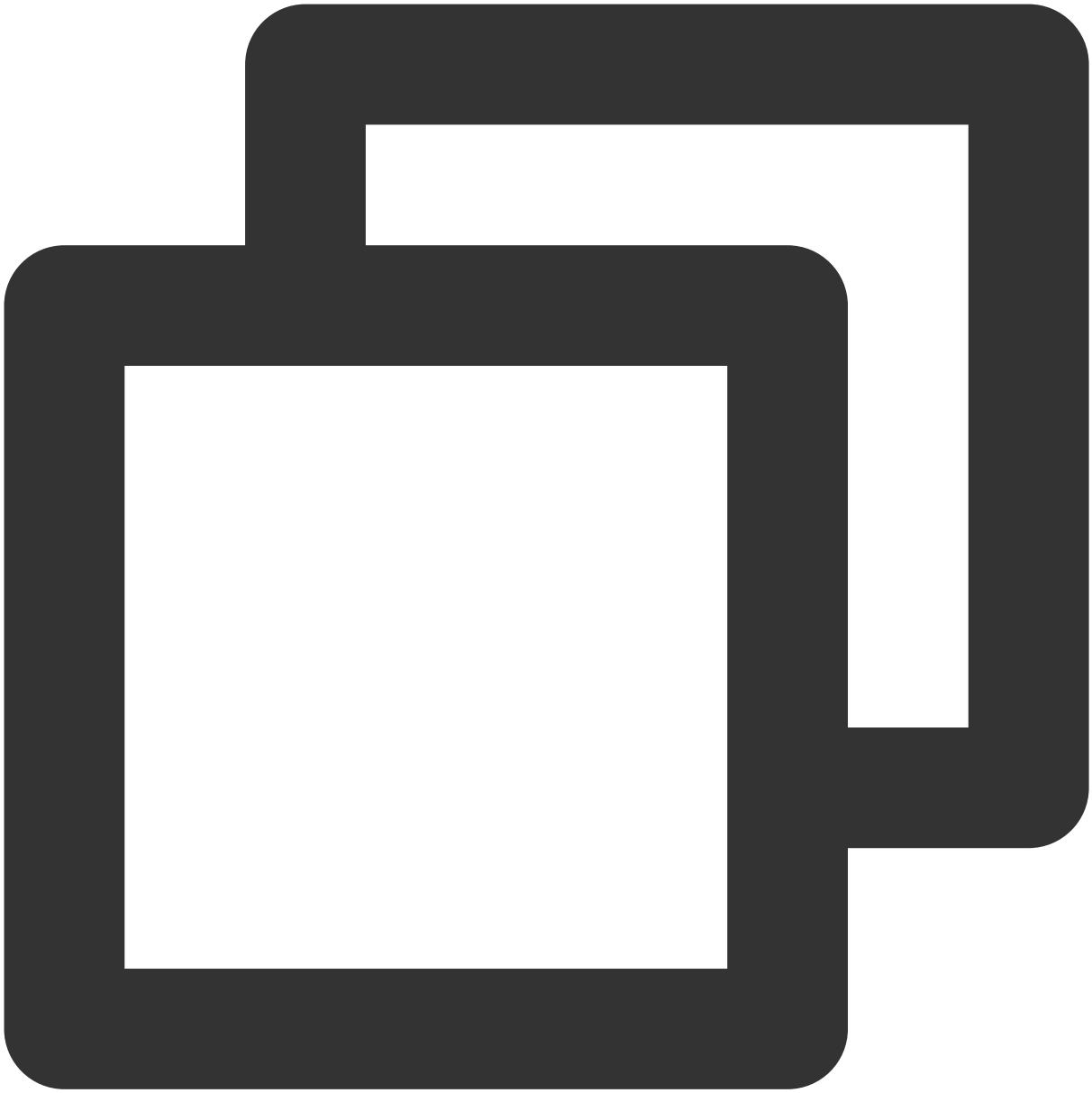


```
- (void)clearAttributes;
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码

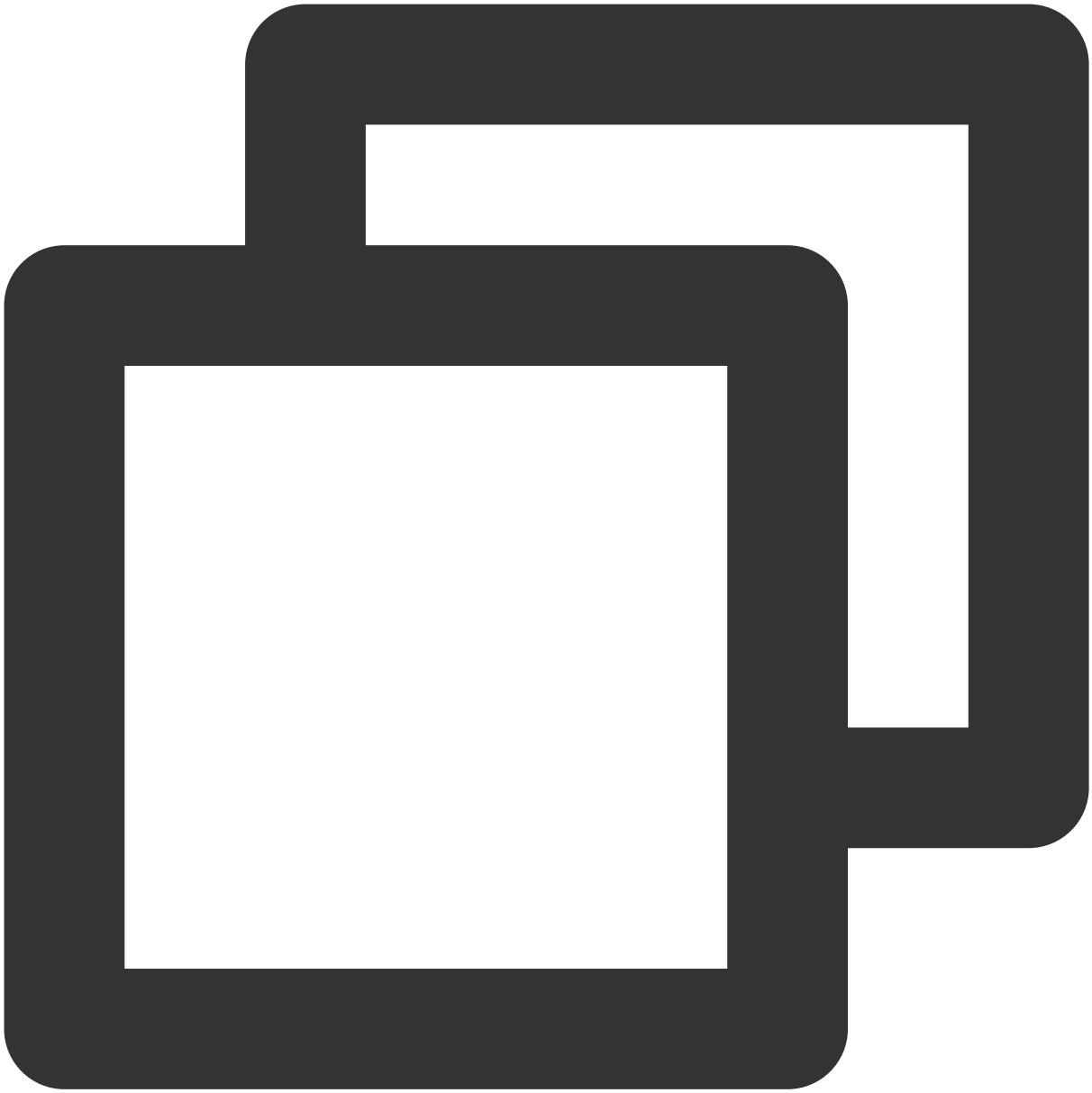


```
[[XGPushTokenManager defaultTokenManager] clearAttributes];
```

更新用户属性

接口说明

清空已有用户属性，然后批量添加用户属性。

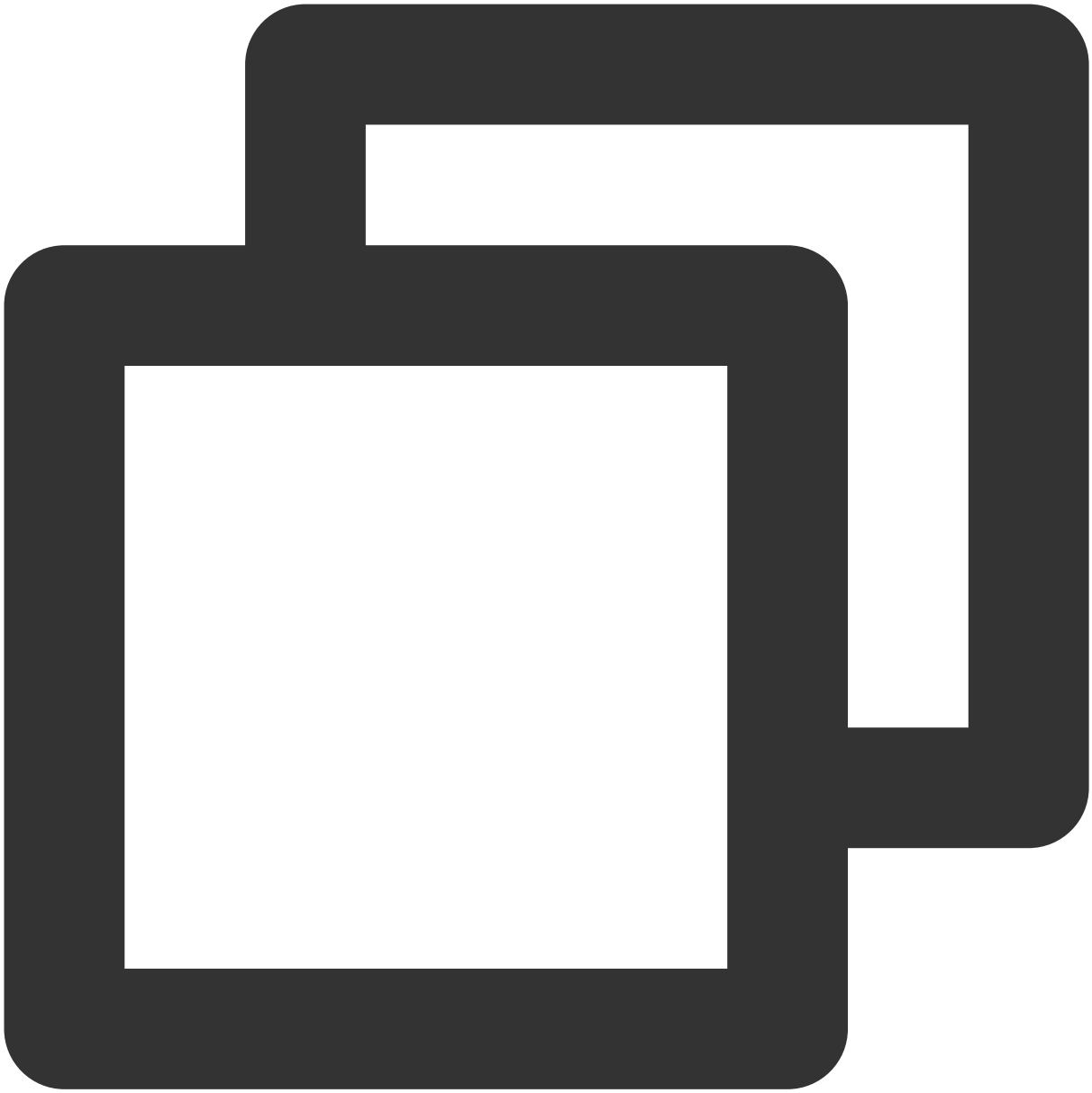


```
- (void)clearAndAppendAttributes:(nonnull NSDictionary<NSString *,NSString *> *)att
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码



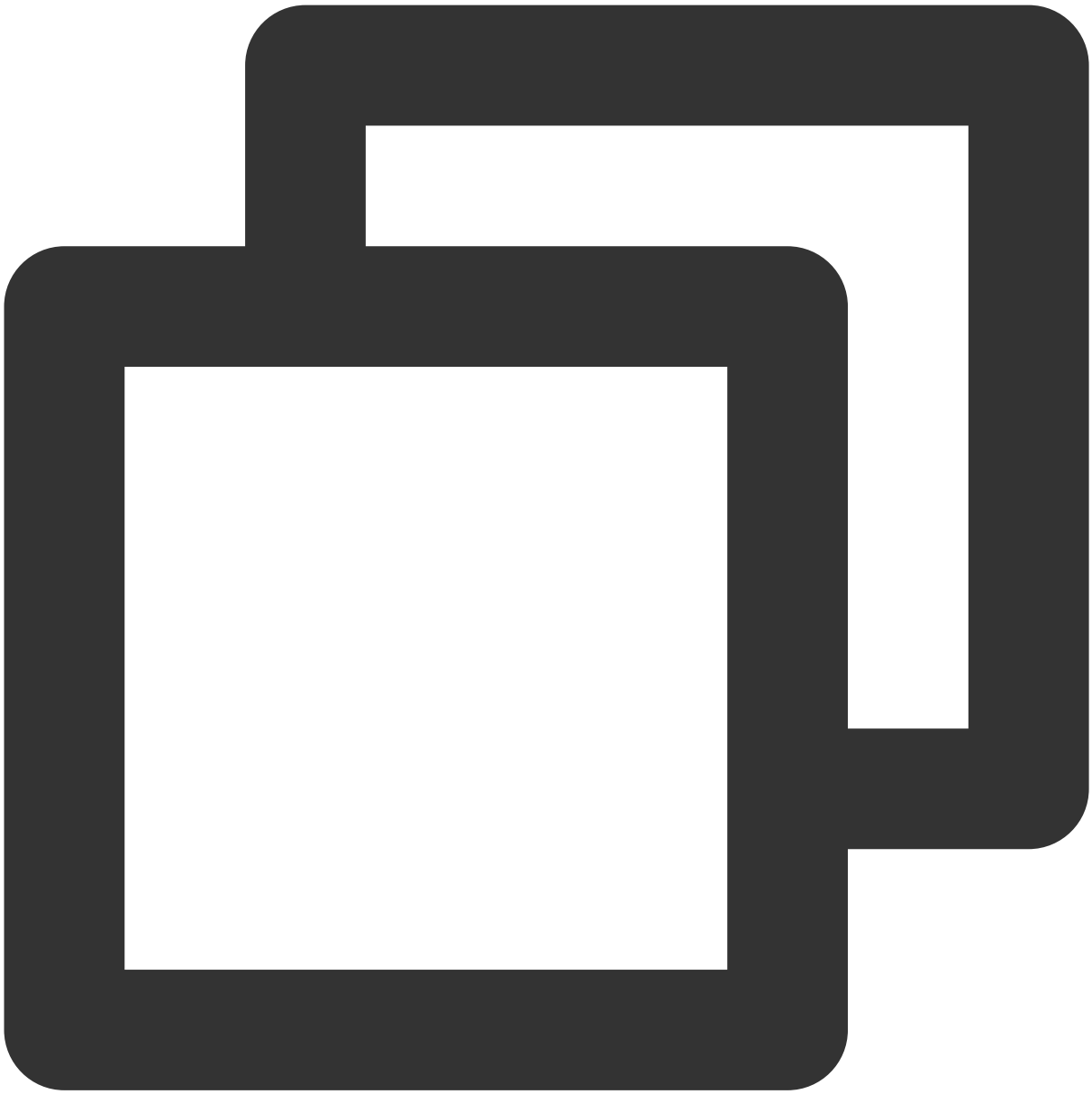
```
[[XGPushTokenManager defaultTokenManager] clearAndAppendAttributes:attributes];
```

角标功能

同步角标

接口说明

当应用本地角标值更改后，需调用此接口将角标值同步到移动推送服务器，下次推送时以此值为基准，此功能在管理台位置（【新建推送】>【高级设置】>【角标数字】）。



```
- (void)setBadge:(NSInteger)badgeNumber;
```

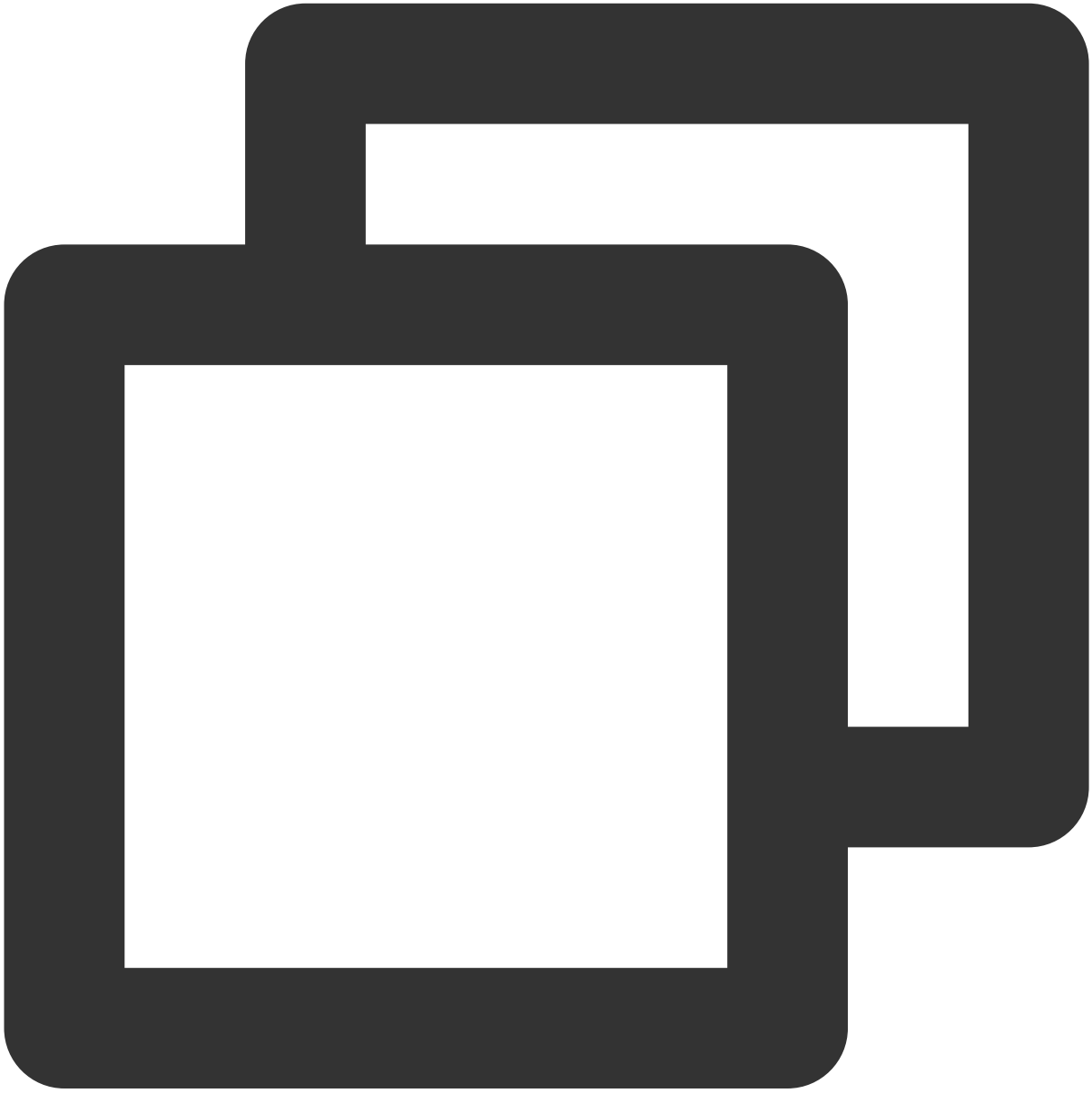
参数说明

badgeNumber：应用的角标数。

注意：

当本地应用角标设置后需调用此接口同步角标值到移动推送服务器，并在下次推送时生效，此接口必须在移动推送长链接建立后调用（xgPushNetworkConnected）。

示例代码



```
/// 冷启动调用时机
- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nu
    /// 在注册完成后上报角标数目
    if (!error) {
     /// 重置应用角标，-1不清空通知栏，0清空通知栏
     [XGPush defaultManager].xgApplicationBadgeNumber = -1;
```

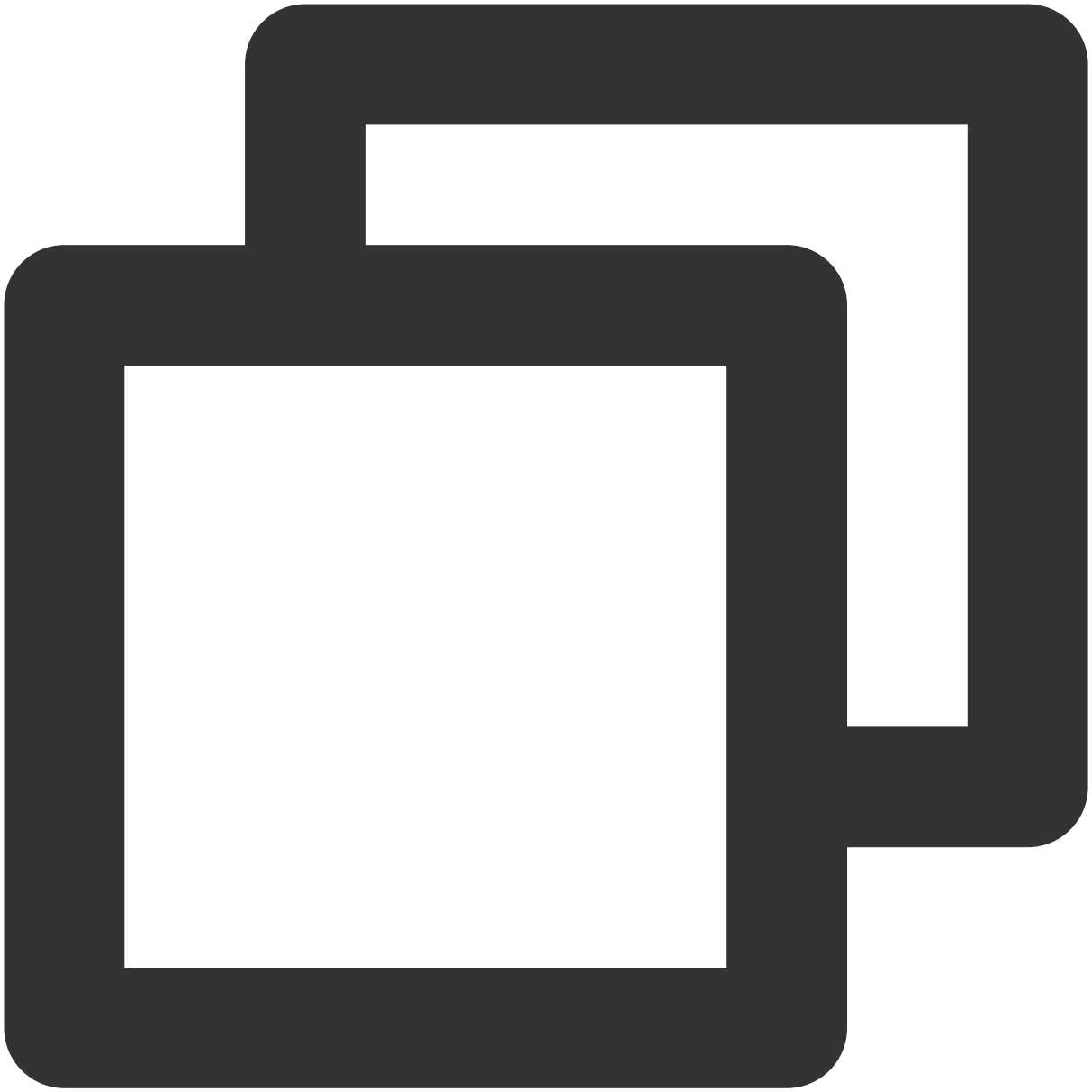
```
        /// 重置服务端自动+1基数
        [[XGPush defaultManager] setBadge:0];
    }
}

/// 热启动调用时机
/// _launchTag热启动标识，业务自行管理
- (void)xgPushNetworkConnected {
    if (_launchTag) {
        /// 重置应用角标，-1不清空通知栏，0清空通知栏
        [XGPush defaultManager].xgApplicationBadgeNumber = -1;
        /// 重置服务端自动+1基数
        [[XGPush defaultManager] setBadge:0];
        _launchTag = NO;
    }
}
```

应用内消息展示

轮询时间设置

接口说明

此接口可以设置应用内消息的轮询时间，最小为10s，默认为258s。



```
/// 设置消息轮询时间间隔，最小值为10s，此方法需要在单例初始化之前调用
- (void)setMessageTimerInterval:(NSTimeInterval)interval;
```

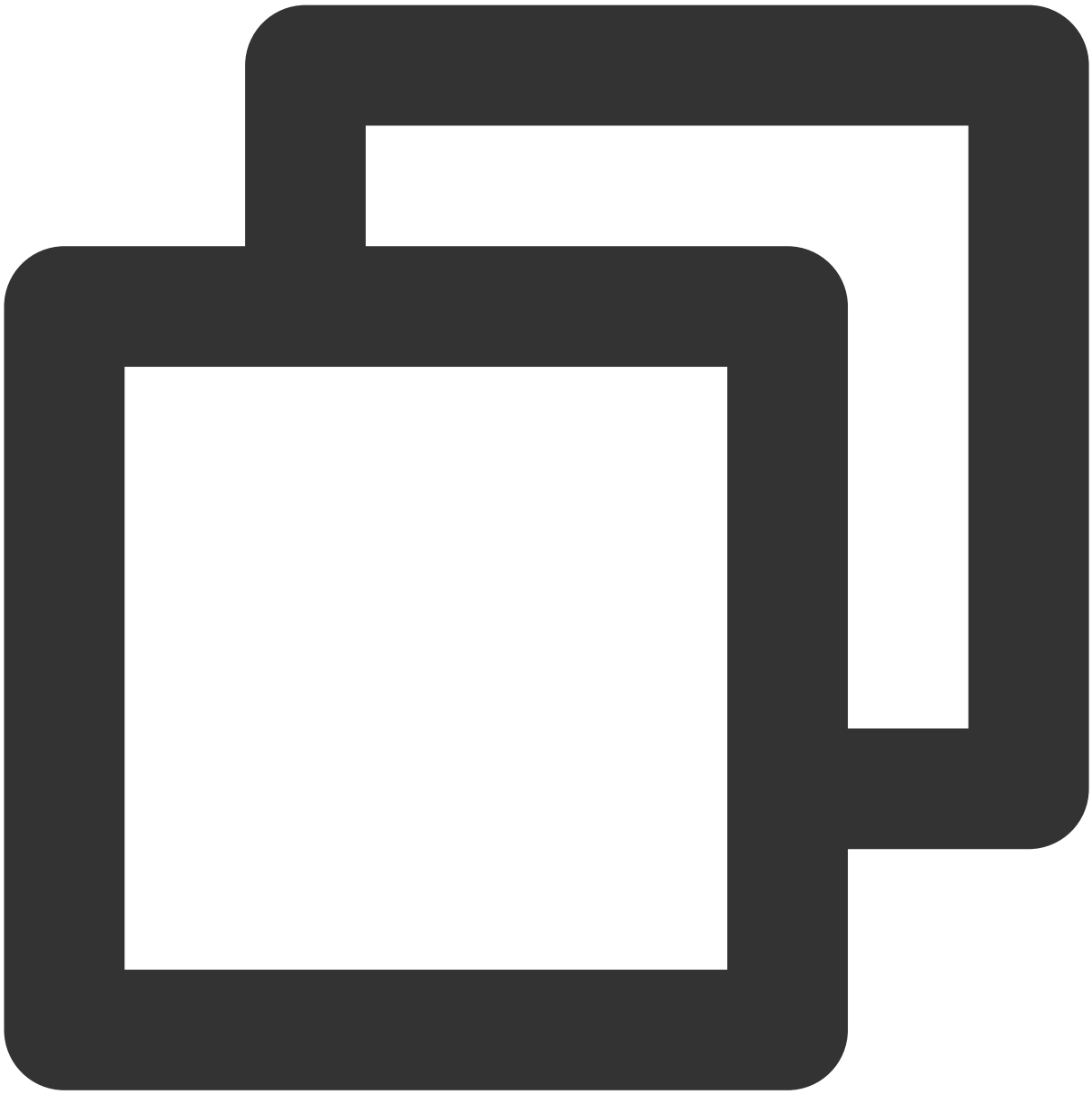
参数说明

NSTimeInterval：NSTimeInterval类型，应用内消息轮询时间间隔。

自定义事件处理

XGInAppMessageActionDelegate 代理说明

用户通过代理方法 `onClickWithCustomAction` 获取自定义事件参数来处理相关业务。

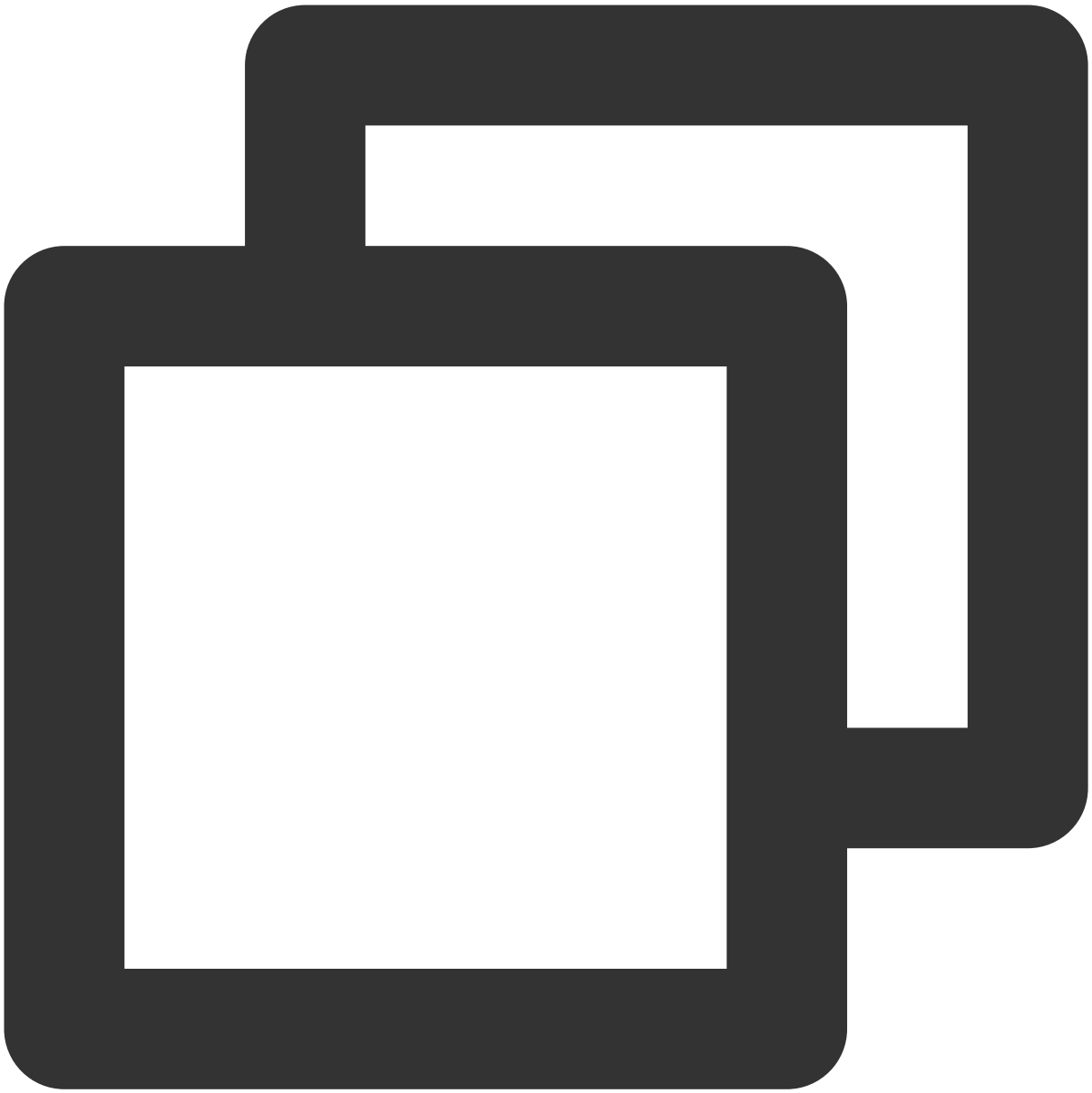


```
/// 按钮事件响应代理
@property (weak, nonatomic, nullable) id<XGInAppMessageActionDelegate> actionDelega
```

查询设备通知权限

接口说明

查询设备通知权限是否被用户允许。

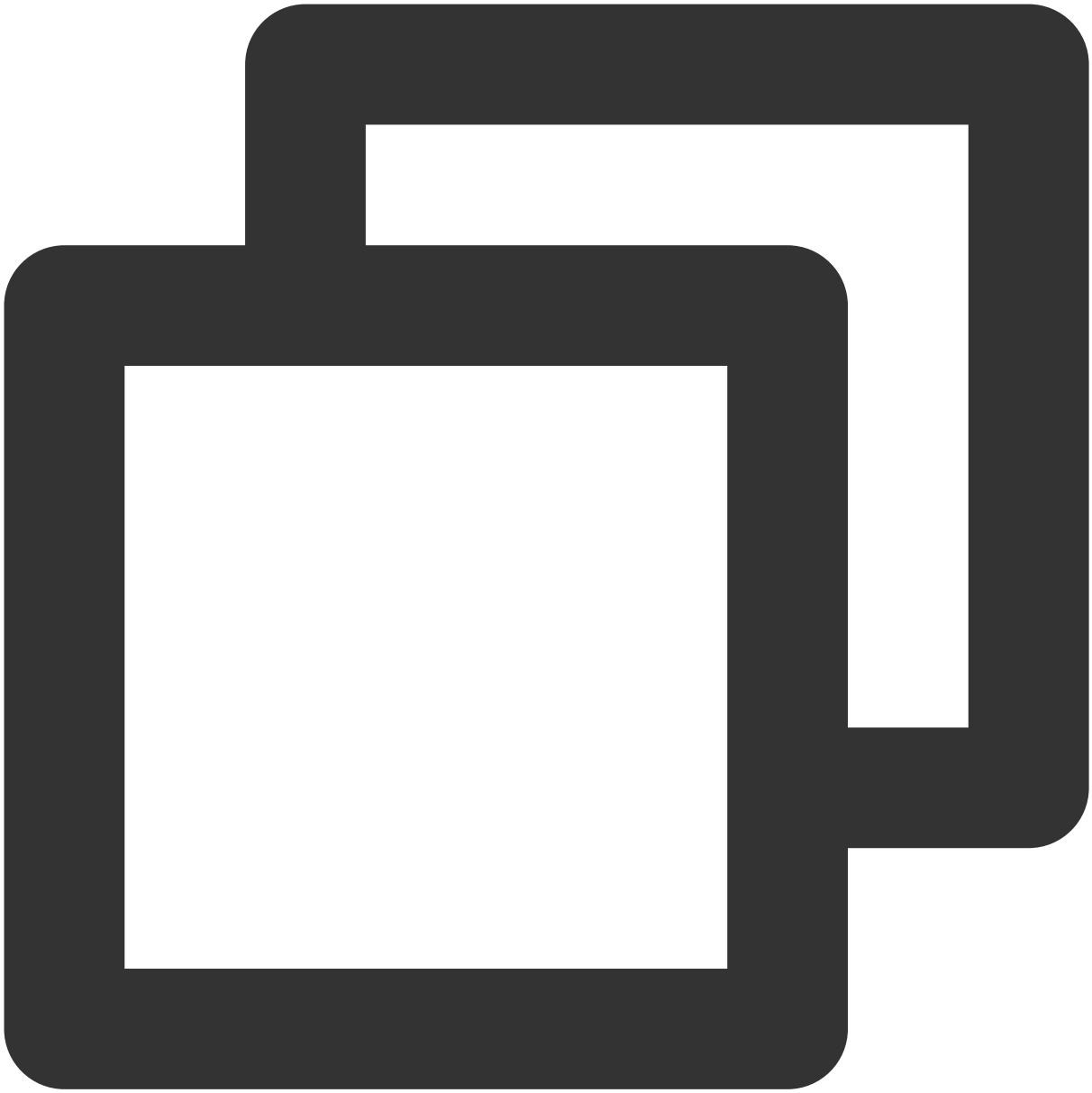


```
- (void)deviceNotificationIsAllowed:(nonnull void (^)(BOOL isAllowed))handler;
```

参数说明

handler：查询结果的返回方法。

示例代码

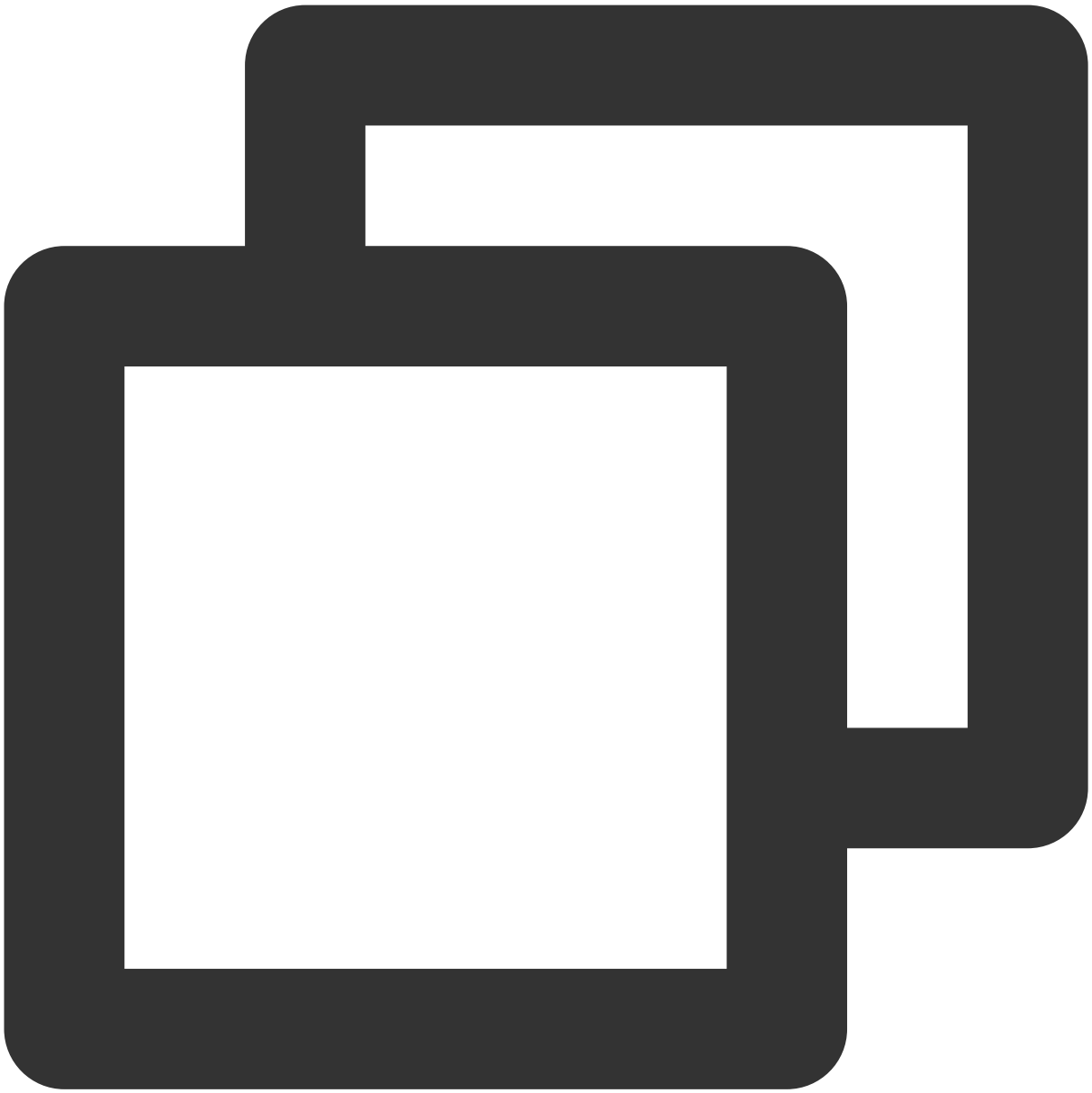


```
[[XGPush defaultManager] deviceNotificationIsAllowed:^(BOOL isAllowed) {
        <#code#>
    }];
```

查询 SDK 版本

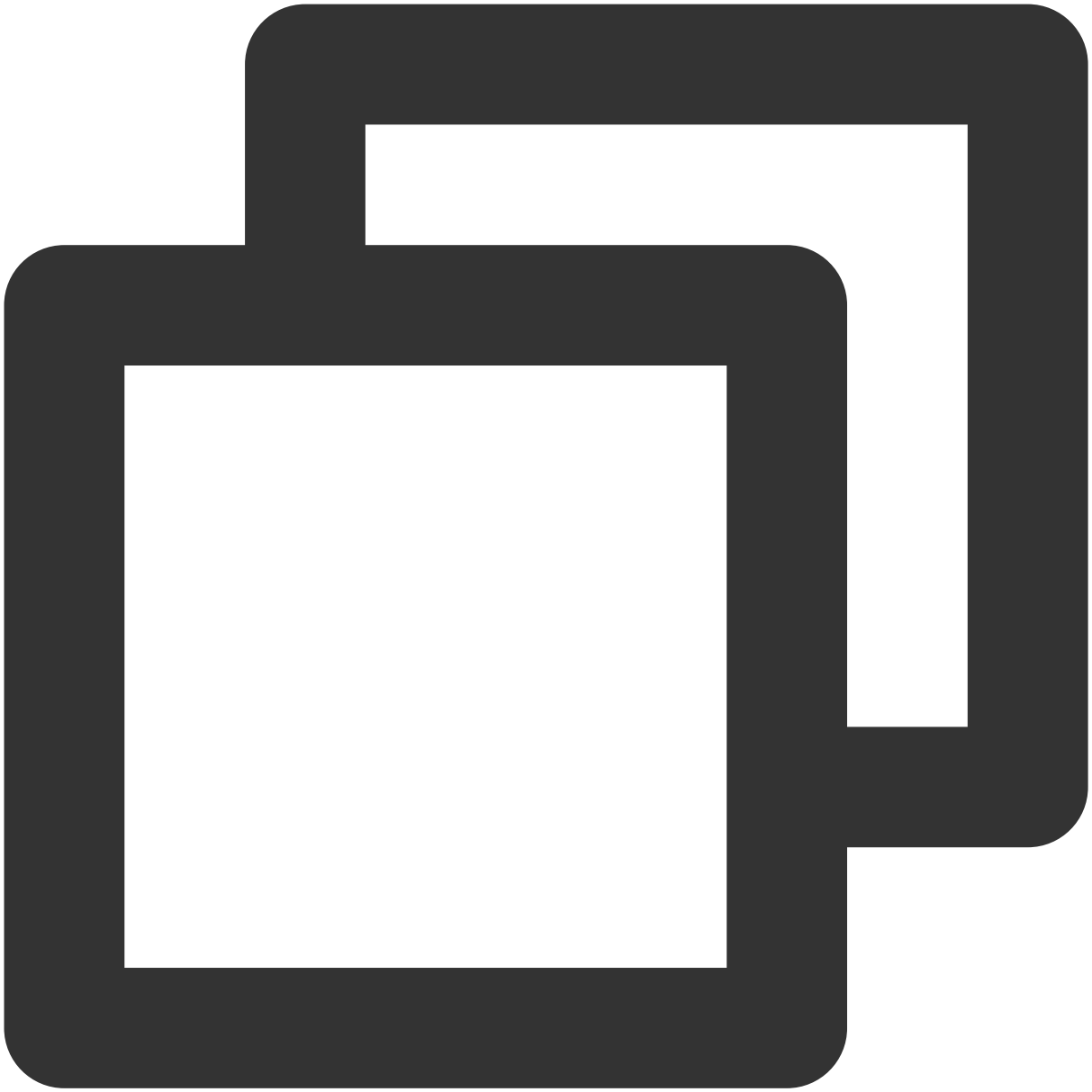
接口说明

查询当前 SDK 的版本。



```
- (nonnull NSString *)sdkVersion;
```

示例代码

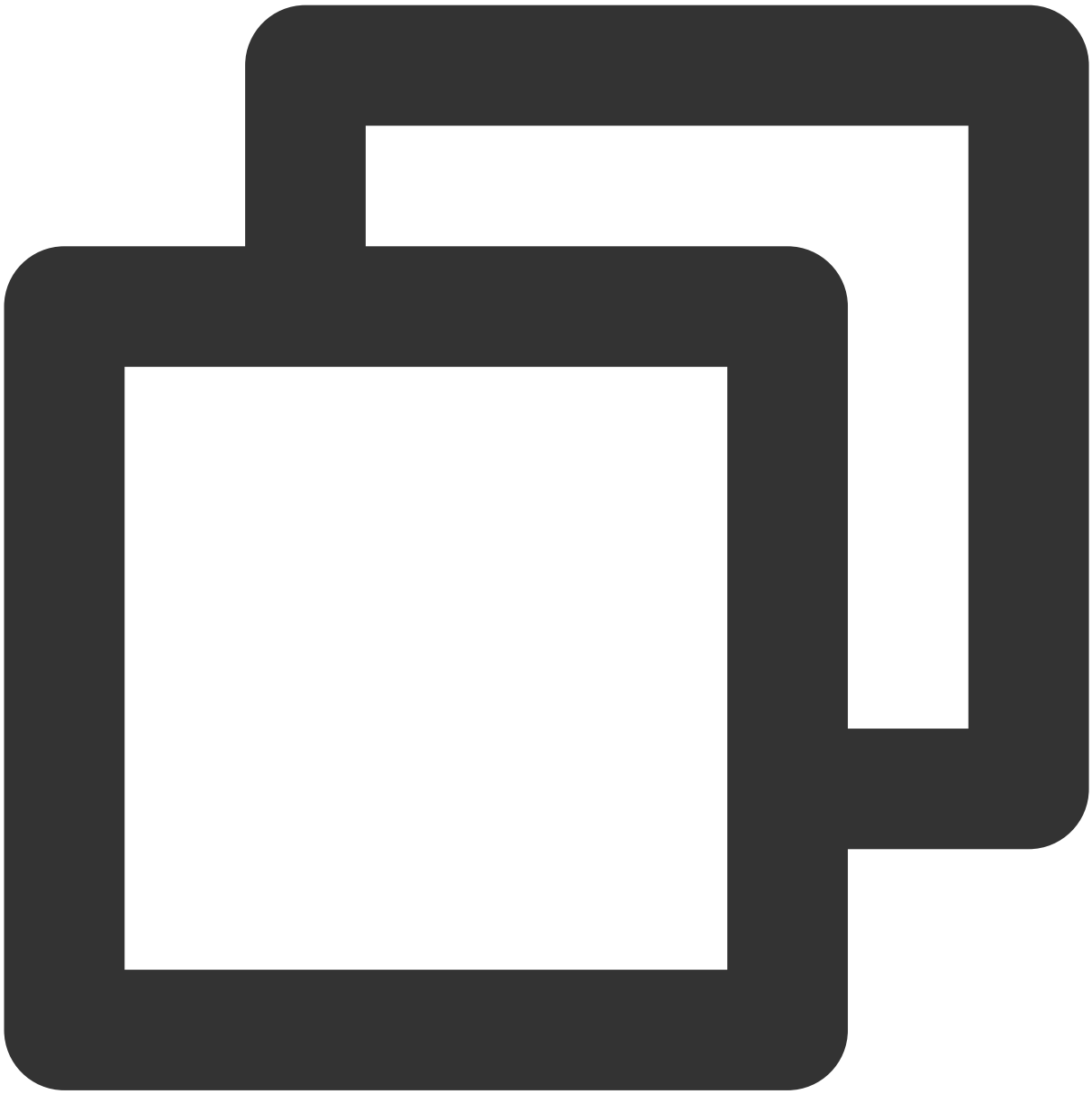


```
[[XGPush defaultManager] sdkVersion];
```

日志上报接口

接口说明

开发者如果发现推送相关功能异常，可以调用该接口，触发本地 push 日志的上报，通过联系 提交工单 反馈问题时，请将文件地址提供给我们，便于排查问题。



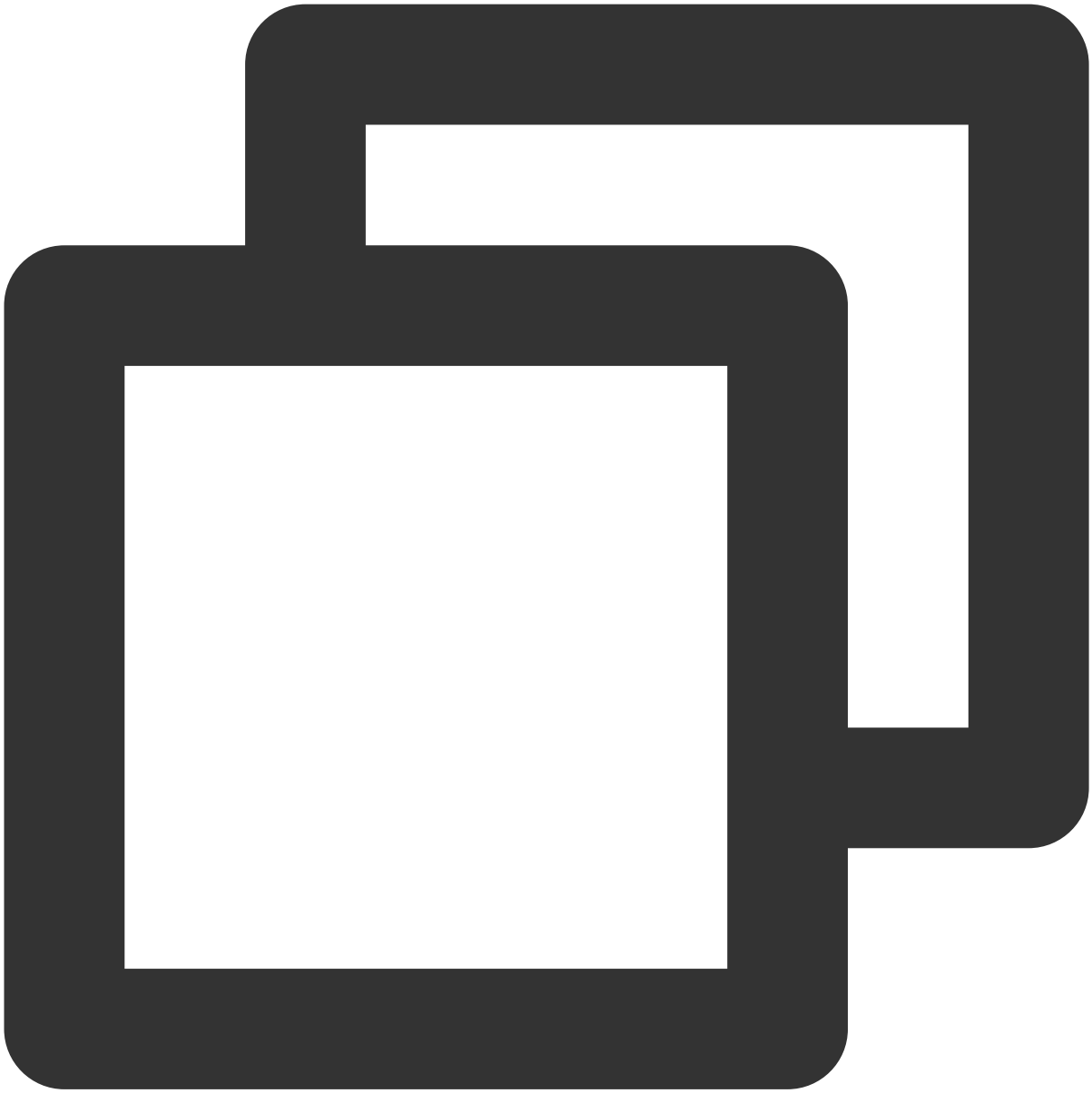
```
/// @note TPNS SDK1.2.4.1+
- (void)uploadLogCompletionHandler:(nullable void(^)(BOOL result,  NSString * _Null
```

参数说明

@brief：上报日志信息 （SDK1.2.4.1+）。

@param handler：上报回调。

示例代码



```
[[XGPush defaultManager] uploadLogCompletionHandler:nil];
```

移动推送日志托管

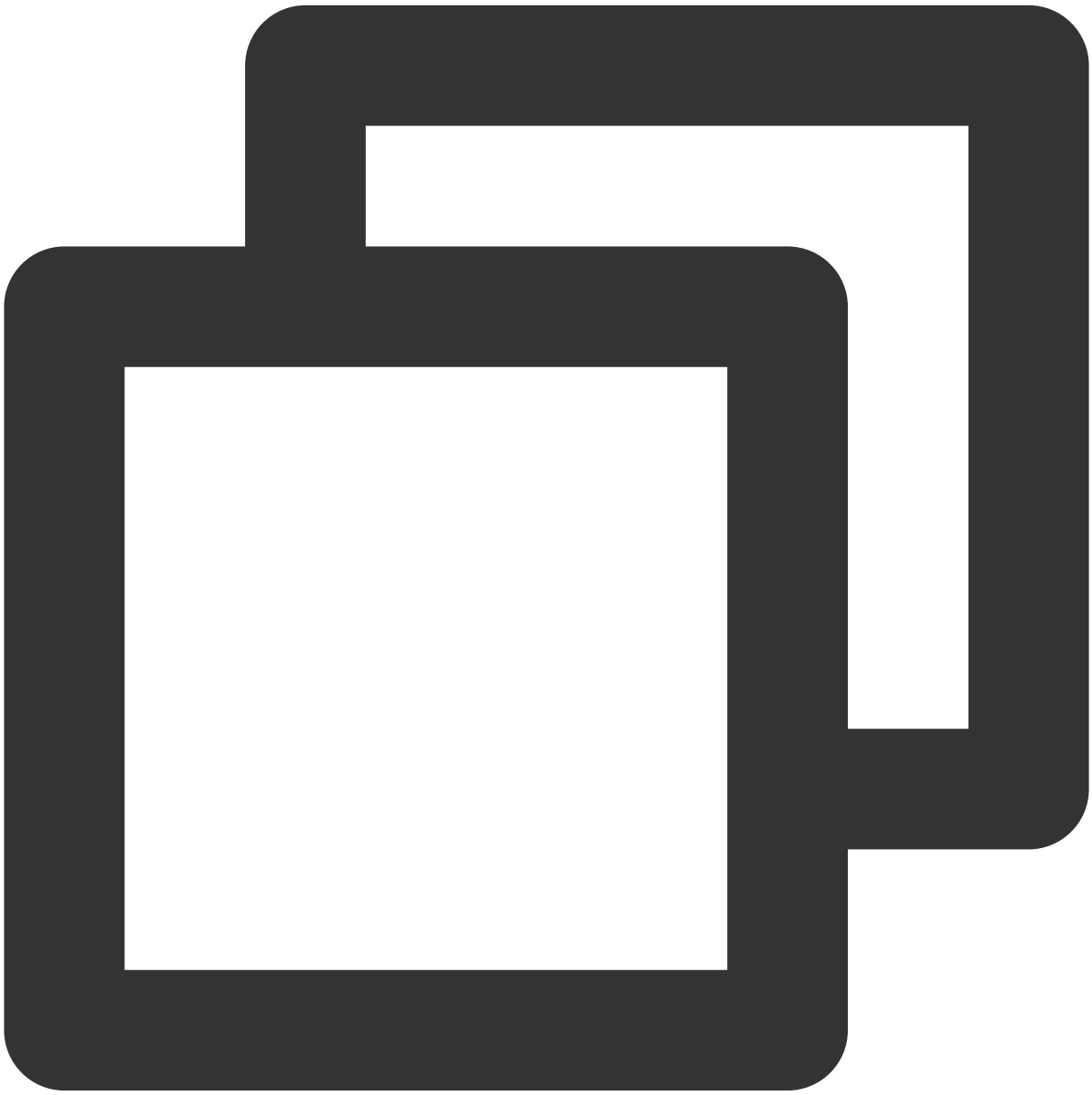
接口说明

可以在此方法获取移动推送的 log 日志。此方法和 XGPush > enableDebug 无关。

参数说明

logInfo：日志信息。

示例代码



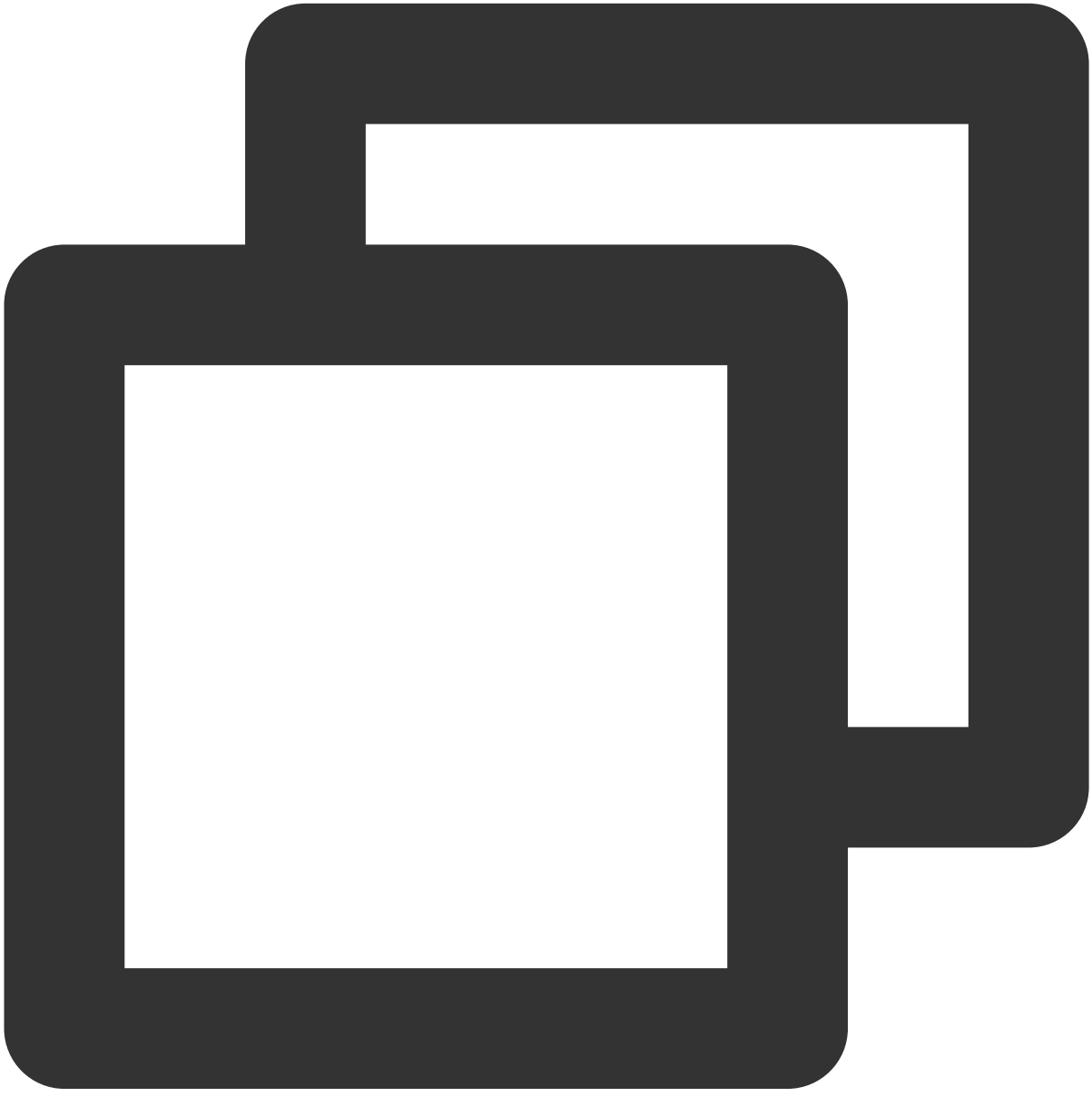
```
- (void)xgPushLog:(nullable NSString *)logInfo;
```

自定义通知栏消息行为

创建消息支持的行为

接口说明

在通知消息中创建一个可以点击的事件行为。



```
+ (nullable id)actionWithIdentifier:(nonnull NSString *)identifier title:(nonnull N
```

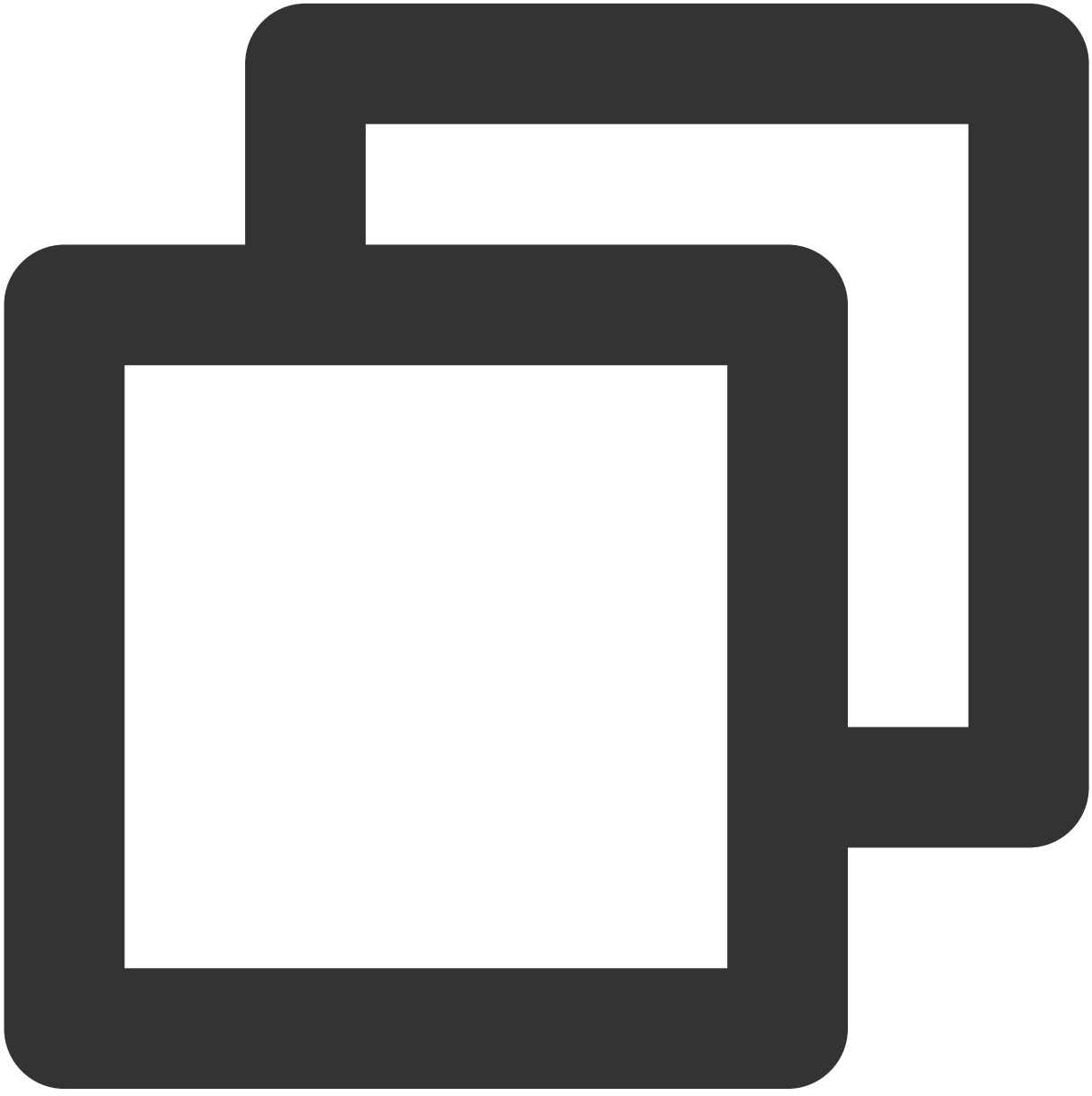
参数说明

identifier：行为唯一标识。

title：行为名称。

options：行为支持的选项。

示例代码



```
XGNotificationAction *action1 = [XGNotificationAction actionWithIdentifier:@"xgacti
```

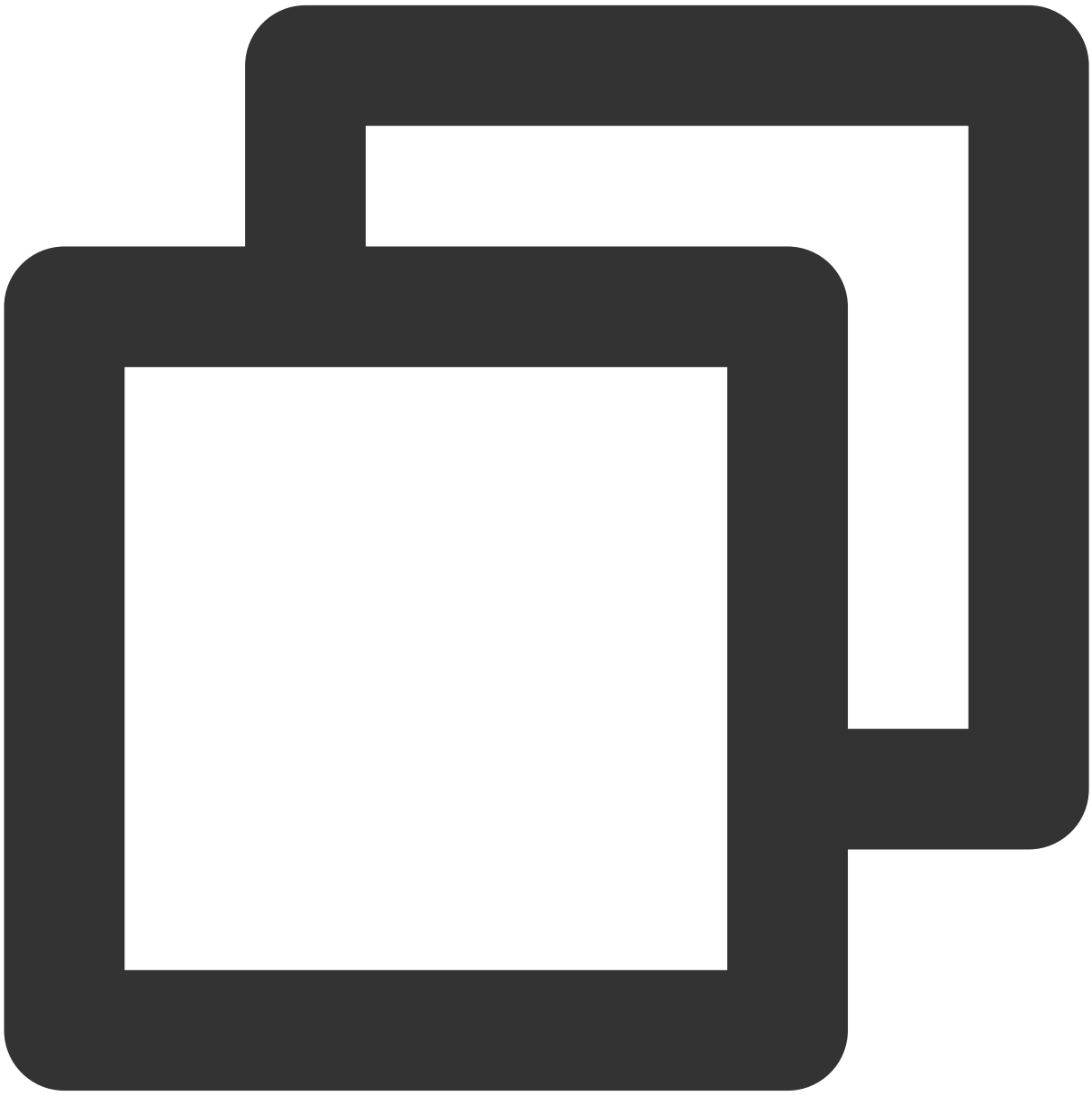
注意：

通知栏带有点击事件的特性，只有在 iOS8.0+ 以上支持，iOS 7.x or earlier 的版本，此方法返回空。

创建分类对象

接口说明

创建分类对象，用以管理通知栏的 Action 对象。



```
+ (nullable id)categoryWithIdentifier:(nonnull NSString *)identifier actions:(nulla
```

参数说明

identifier：分类对象的标识。

actions：当前分类拥有的行为对象组。

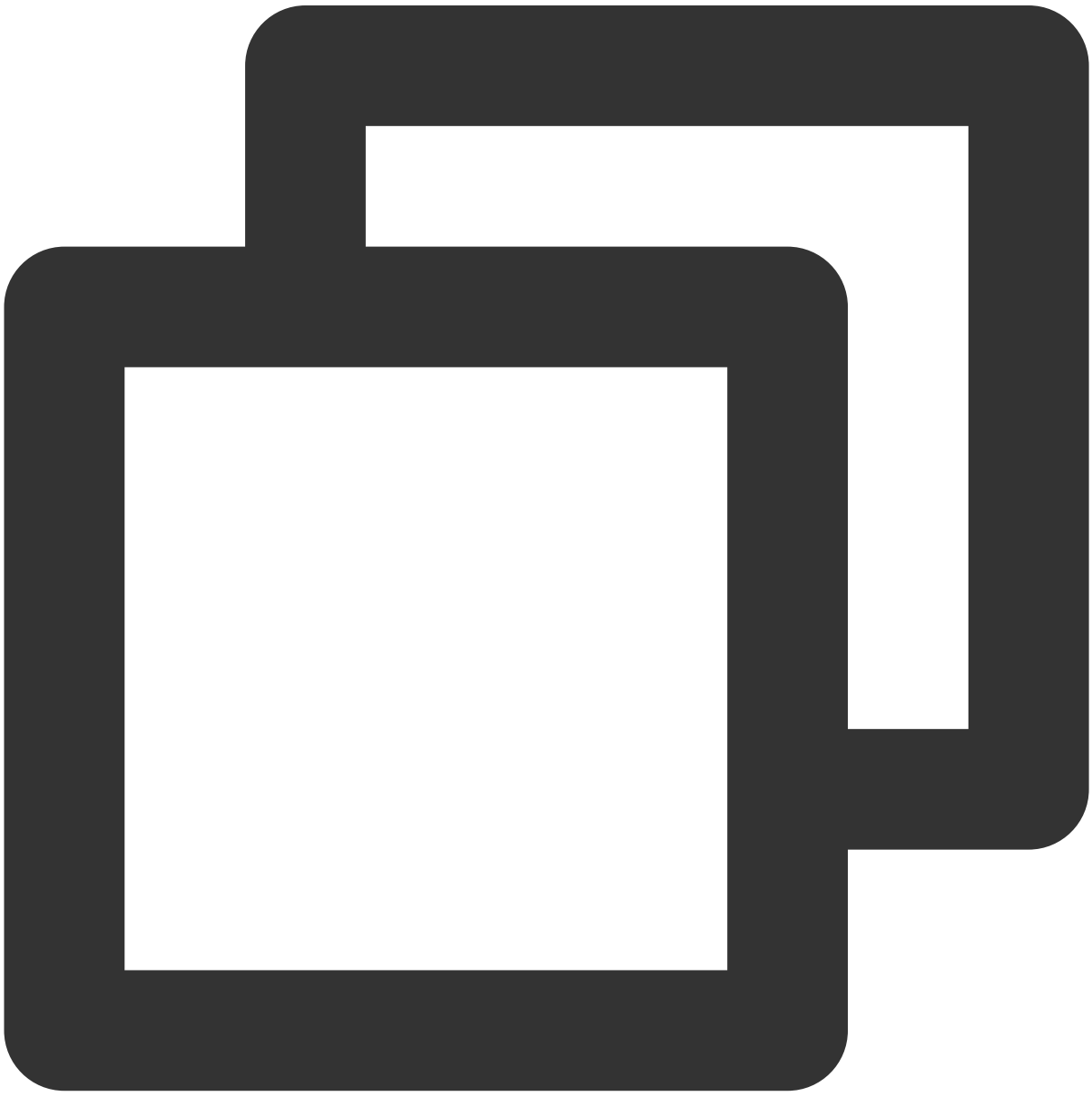
intentIdentifiers：用以表明可以通过 Siri 识别的标识。

options：分类的特性。

注意：

通知栏带有点击事件的特性，只有在 iOS8+ 以上支持，iOS 8 or earlier的版本，此方法返回空。

示例代码

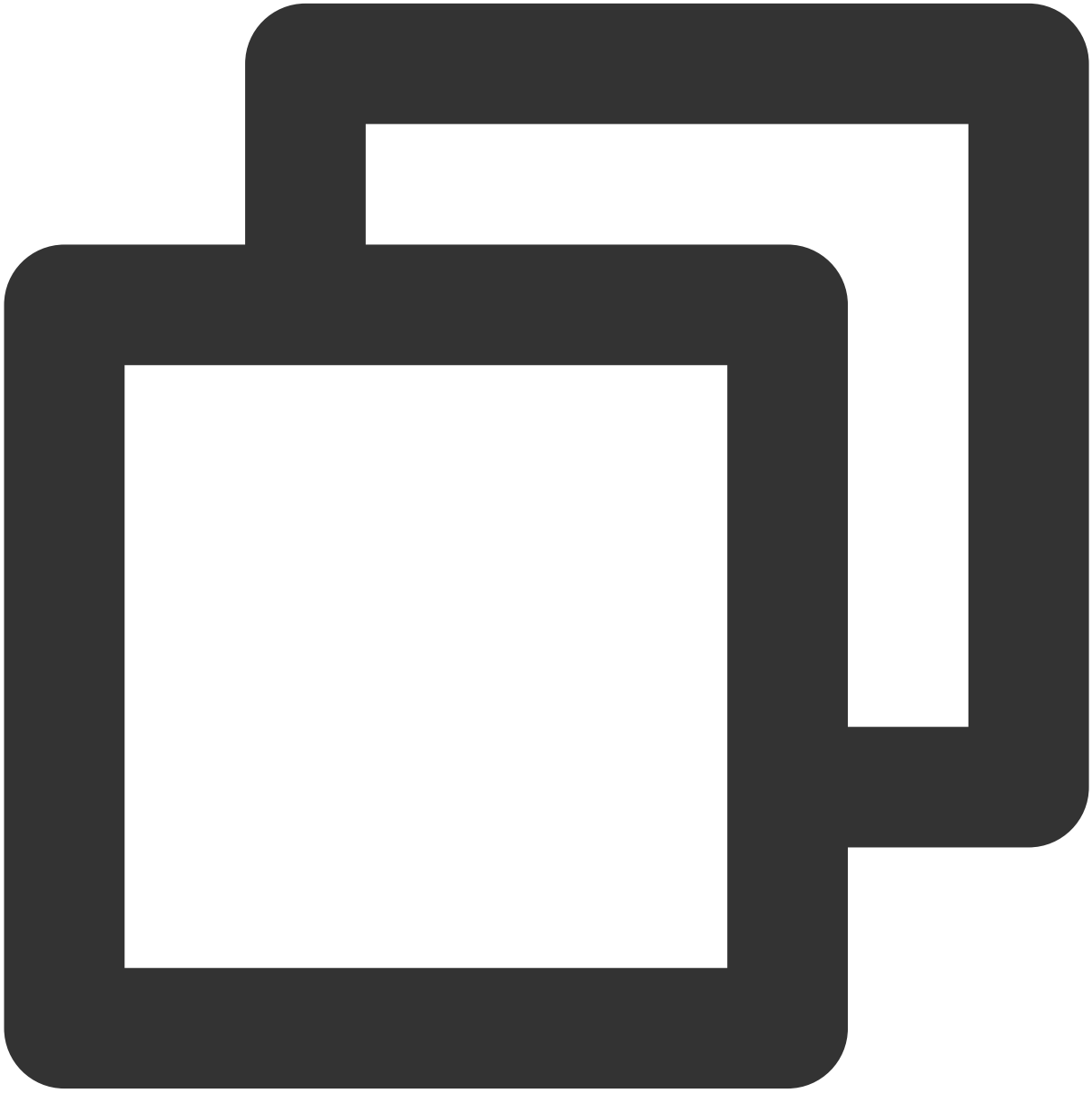


```
XGNotificationCategory *category = [XGNotificationCategory categoryWithIdentifier:@
```

创建配置类

接口说明

管理推送消息通知栏的样式和特性。



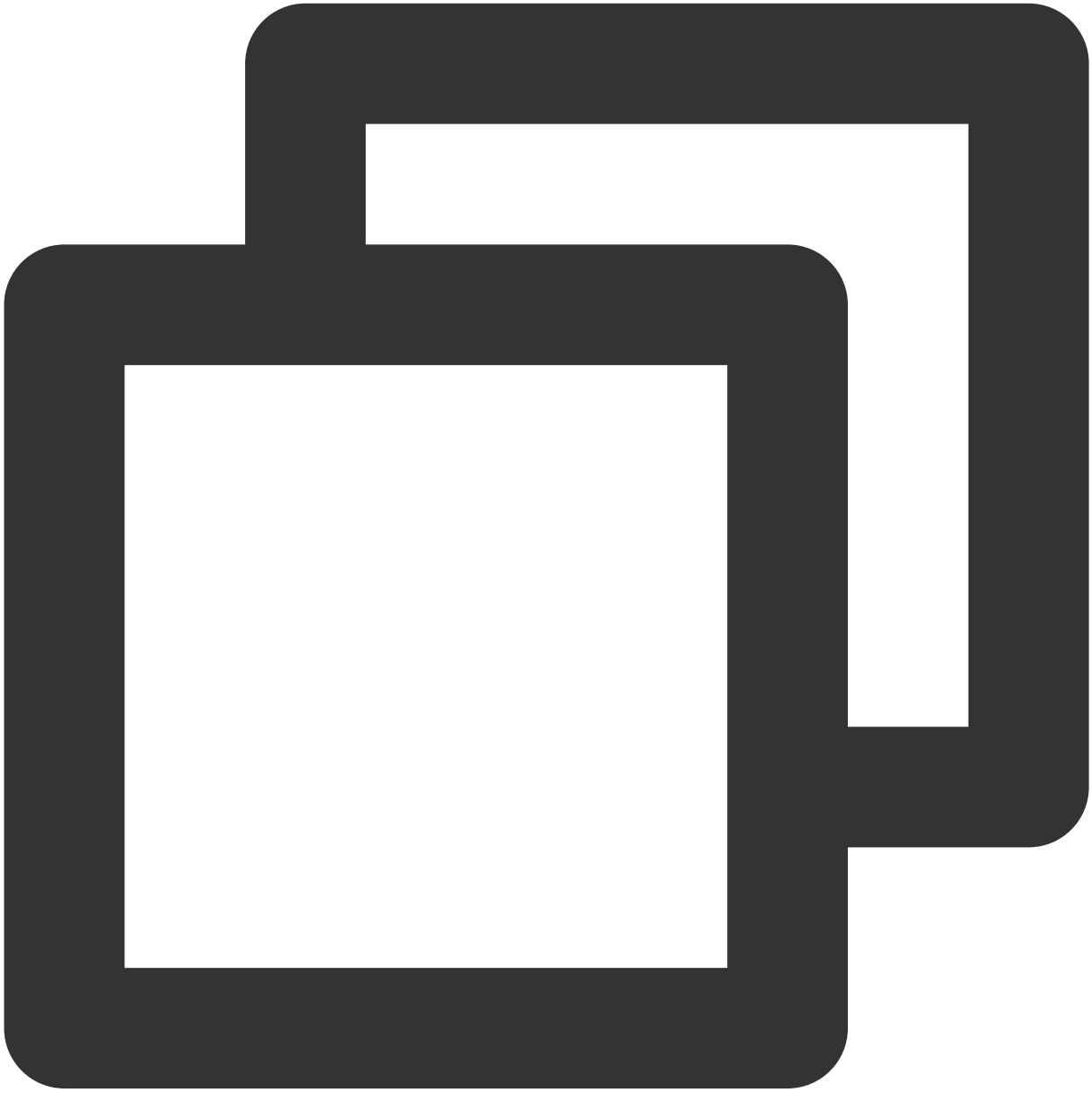
```
+ (nullable instancetype)configureNotificationWithCategories:(nullable NSSet<id> *)
```

参数说明

categories：通知栏中支持的分类集合。

types：注册通知的样式。

示例代码



```
XGNotificationConfigure *configure = [XGNotificationConfigure configureNotification
```

本地推送

本地推送相关功能请参见 苹果开发者文档。

推送证书获取指引

最近更新时间：2024-01-16 17:42:20

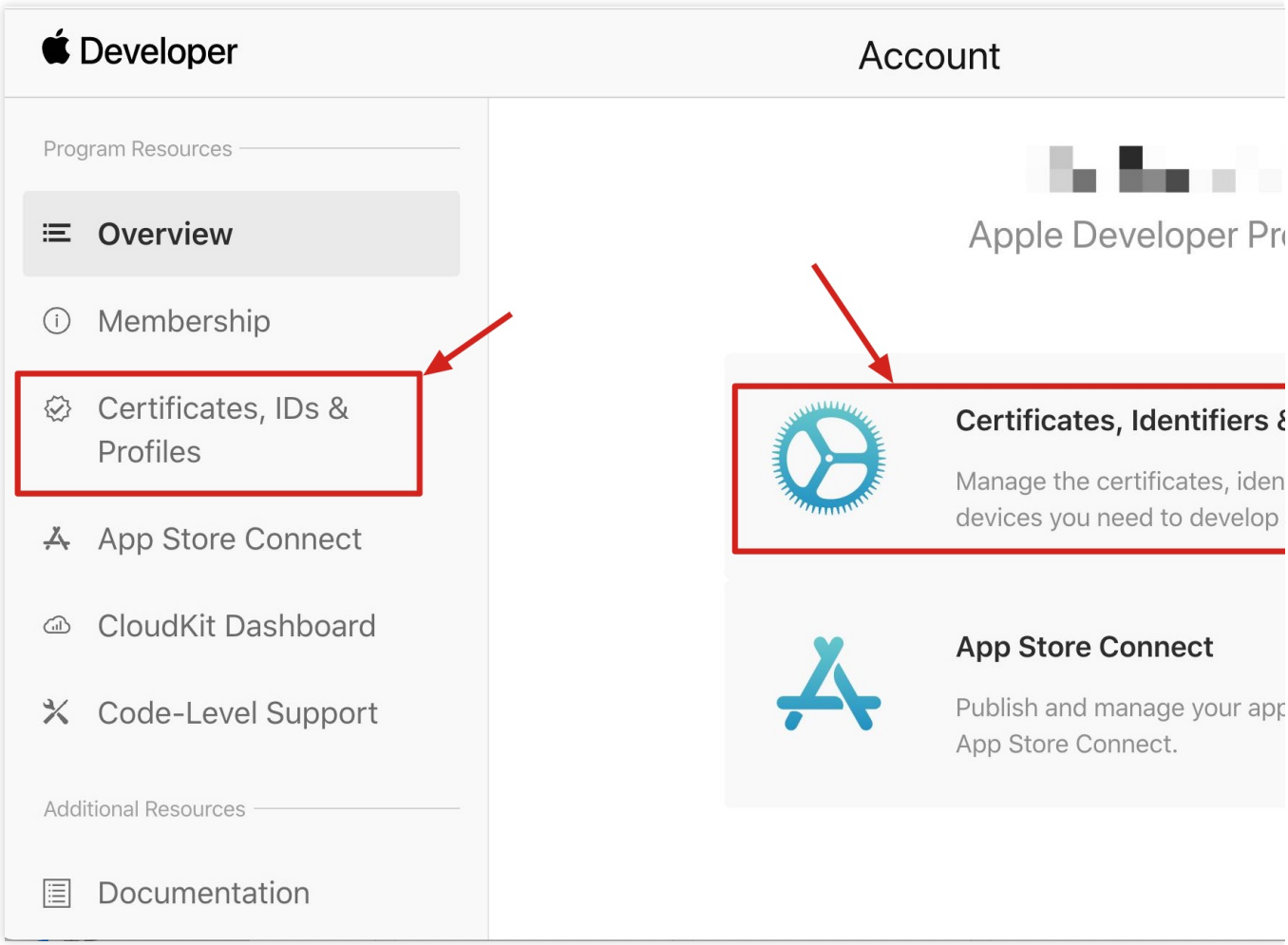
本文档主要指导您如何生成 iOS 消息推送证书及上传证书。

说明：

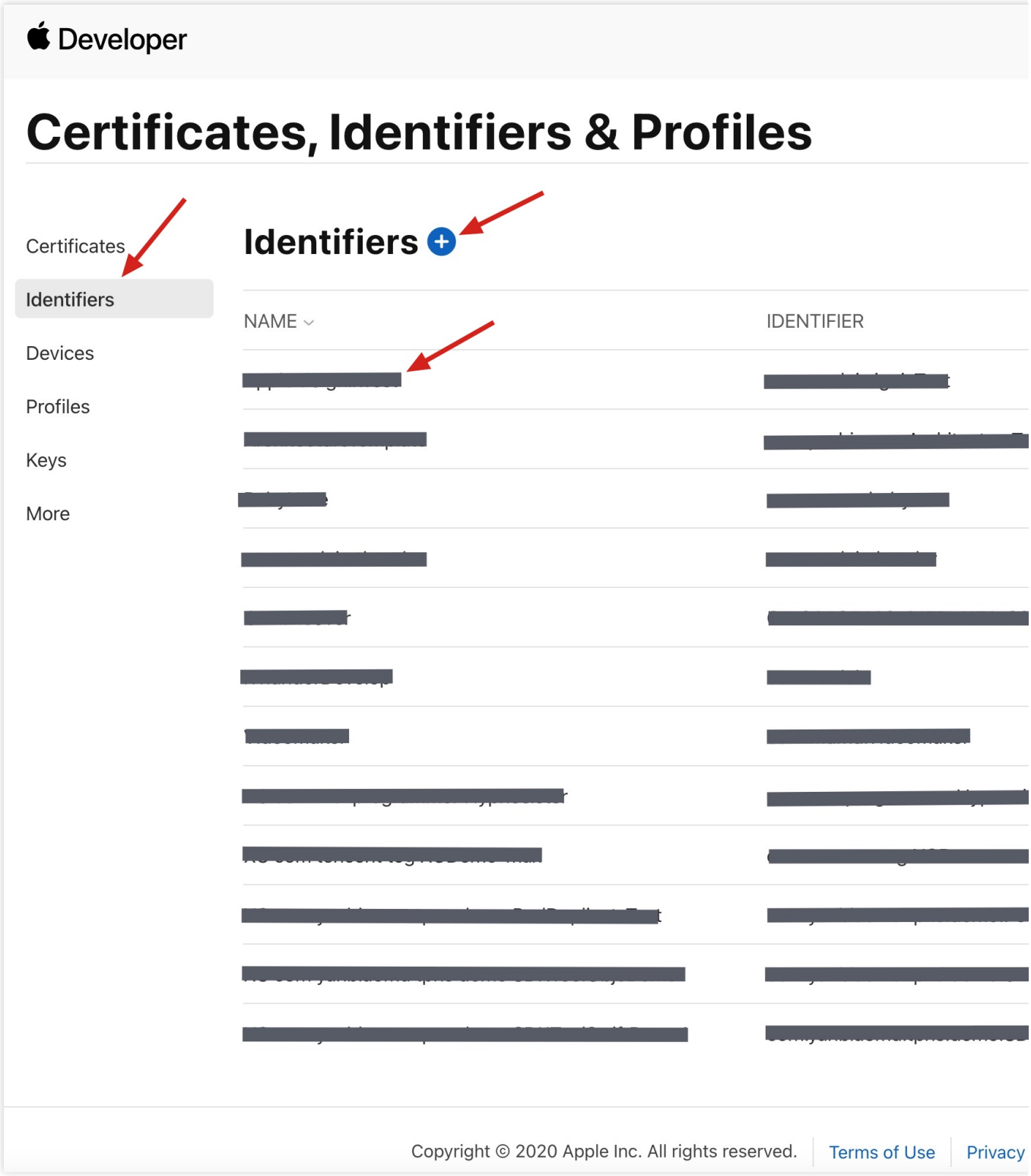
移动推送更推荐您使用 p12 来分别管理您的应用的推送服务；虽然 p8 比 p12 有更长的有效期，但是同时也有更大的推送权限和范围，若泄露，可能会造成更加严重的影响。

步骤1：为 App 开启远程推送服务

1. 登录 苹果开发者中心 网站，单击 **Certificates,Identifiers & Profiles** 或者侧栏的 **Certificates, IDS & Profiles**，进入 Certificates, IDS & Profiles 页面。

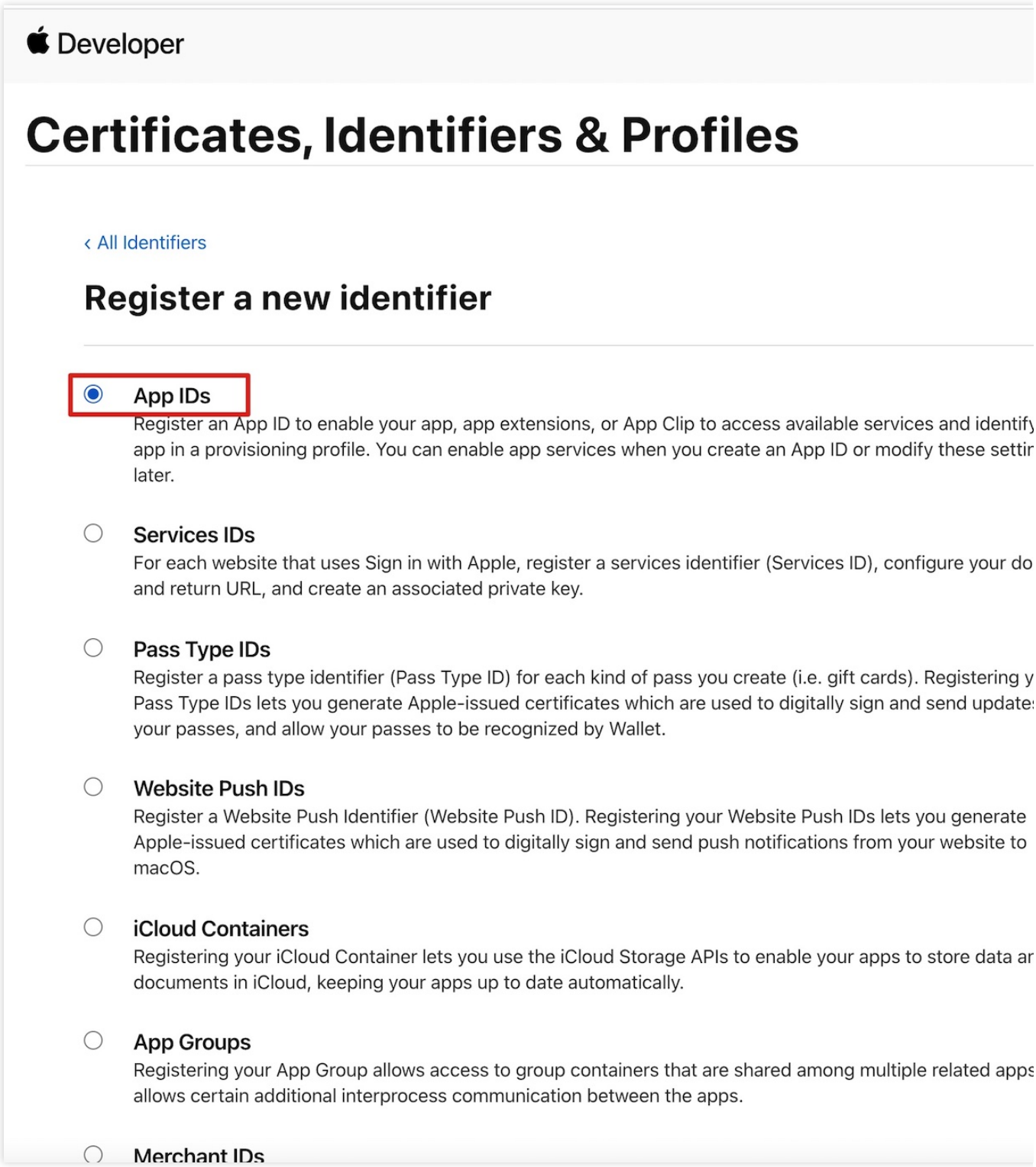


2. 单击 Identifiers 右侧的 **+**。

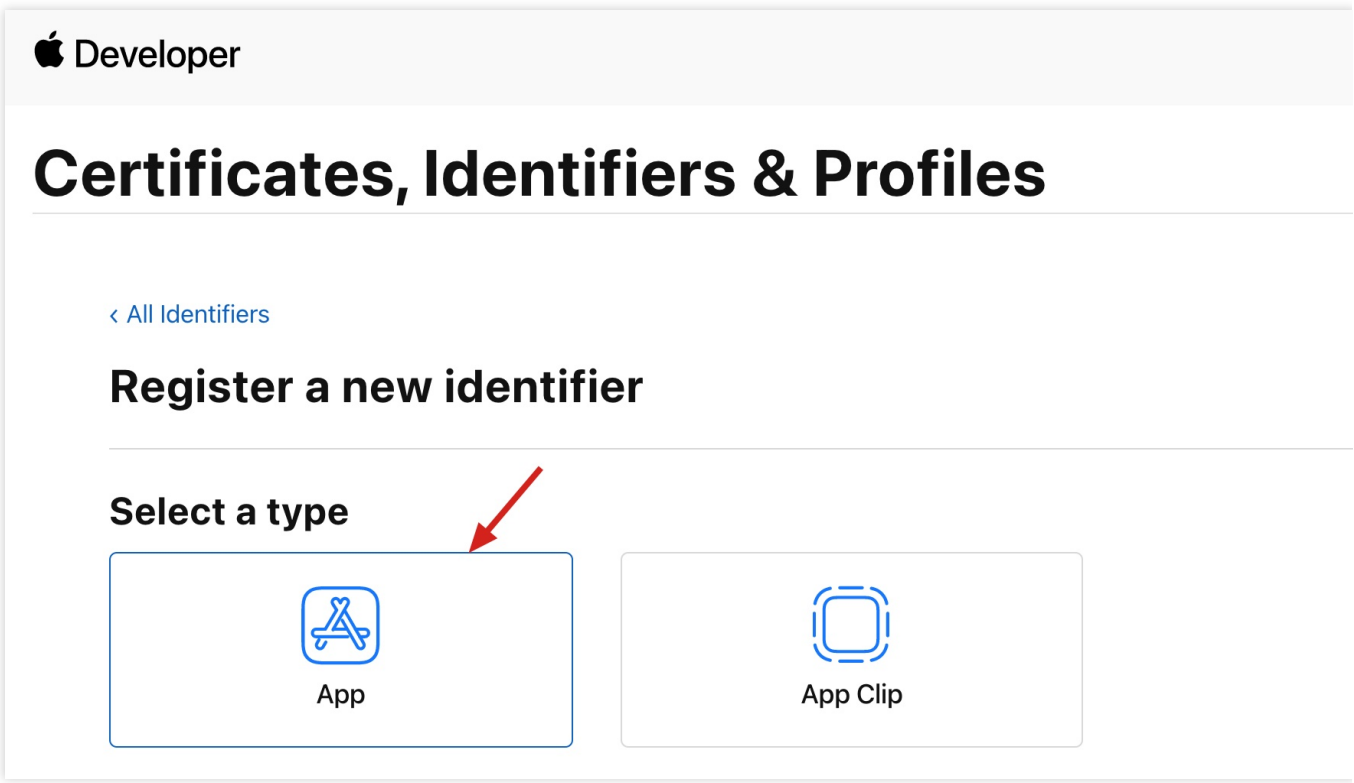


3. 您可以参考如下步骤新建一个 AppID，或者在您原有的 AppID 上增加 `Push Notification` 的 `Service`。需要注意的是，您 App 的 `Bundle ID` 不能使用通配符 `*`，否则将无法使用远程推送服务。

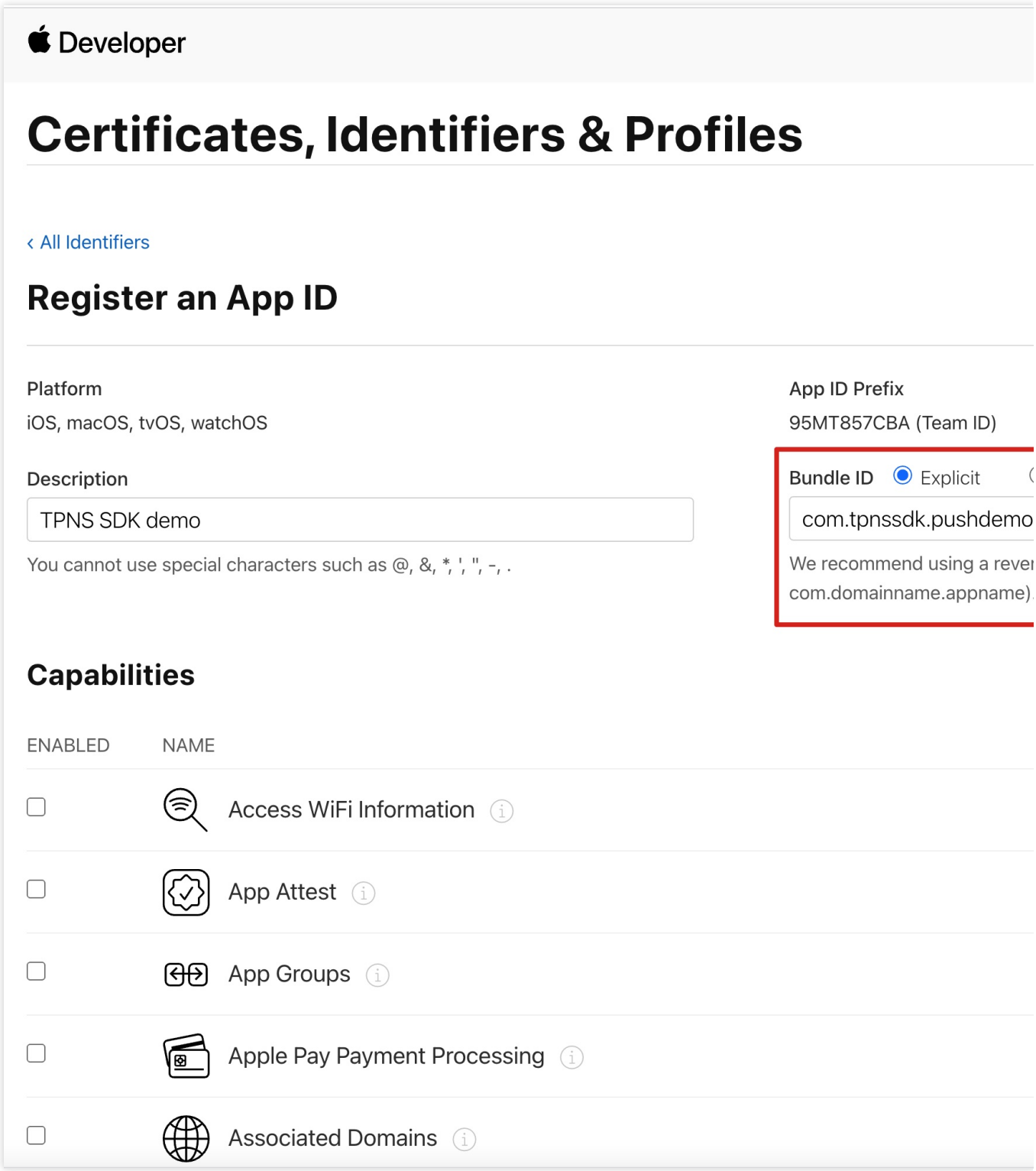
1. 勾选 **App IDs**，单击 **Continue** 进行下一步。



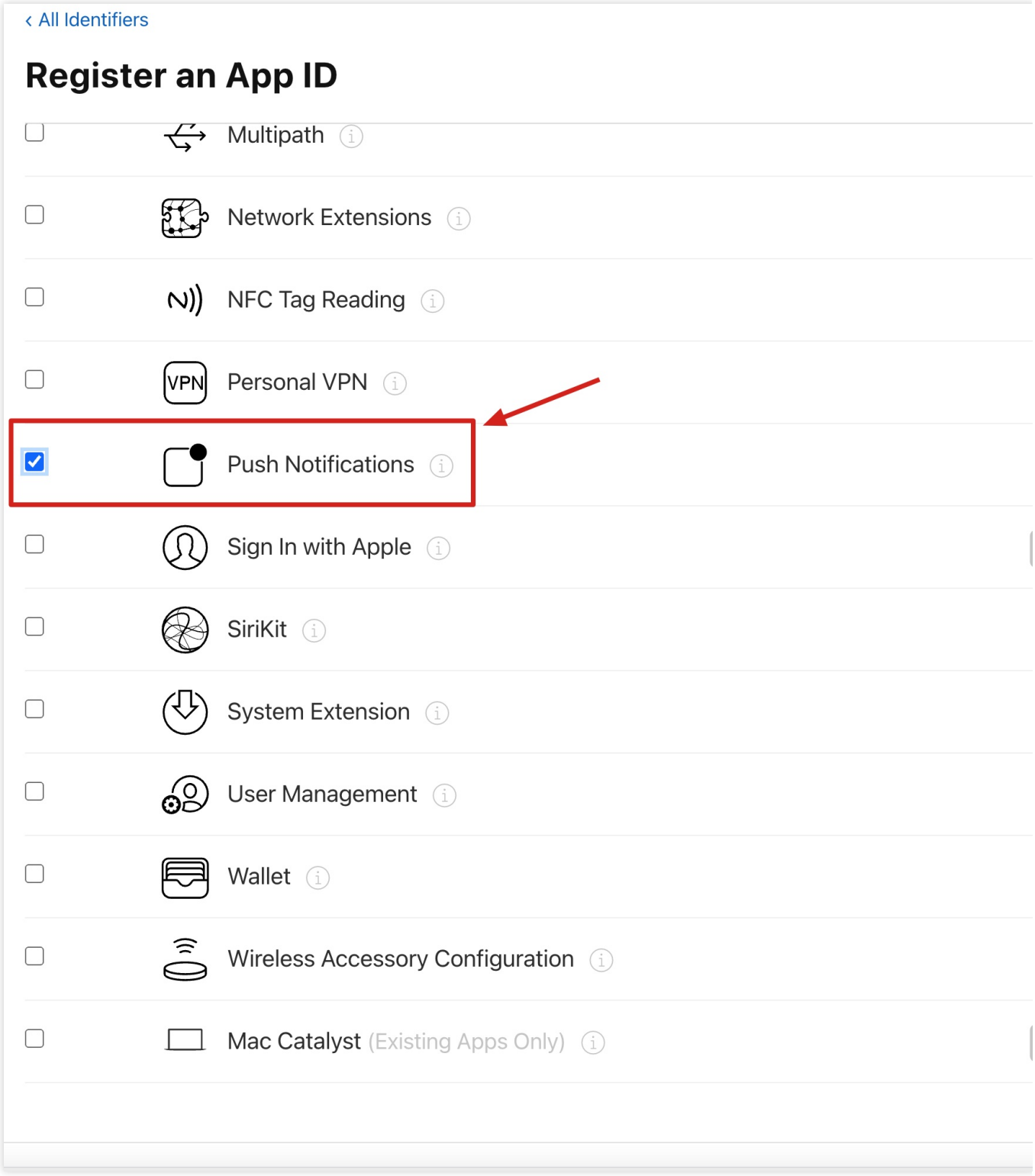
2. 选择 **App**，单击 **Continue** 进行下一步。



3. 配置 `Bundle ID` 等其他信息，单击 **Continue** 进行下一步。

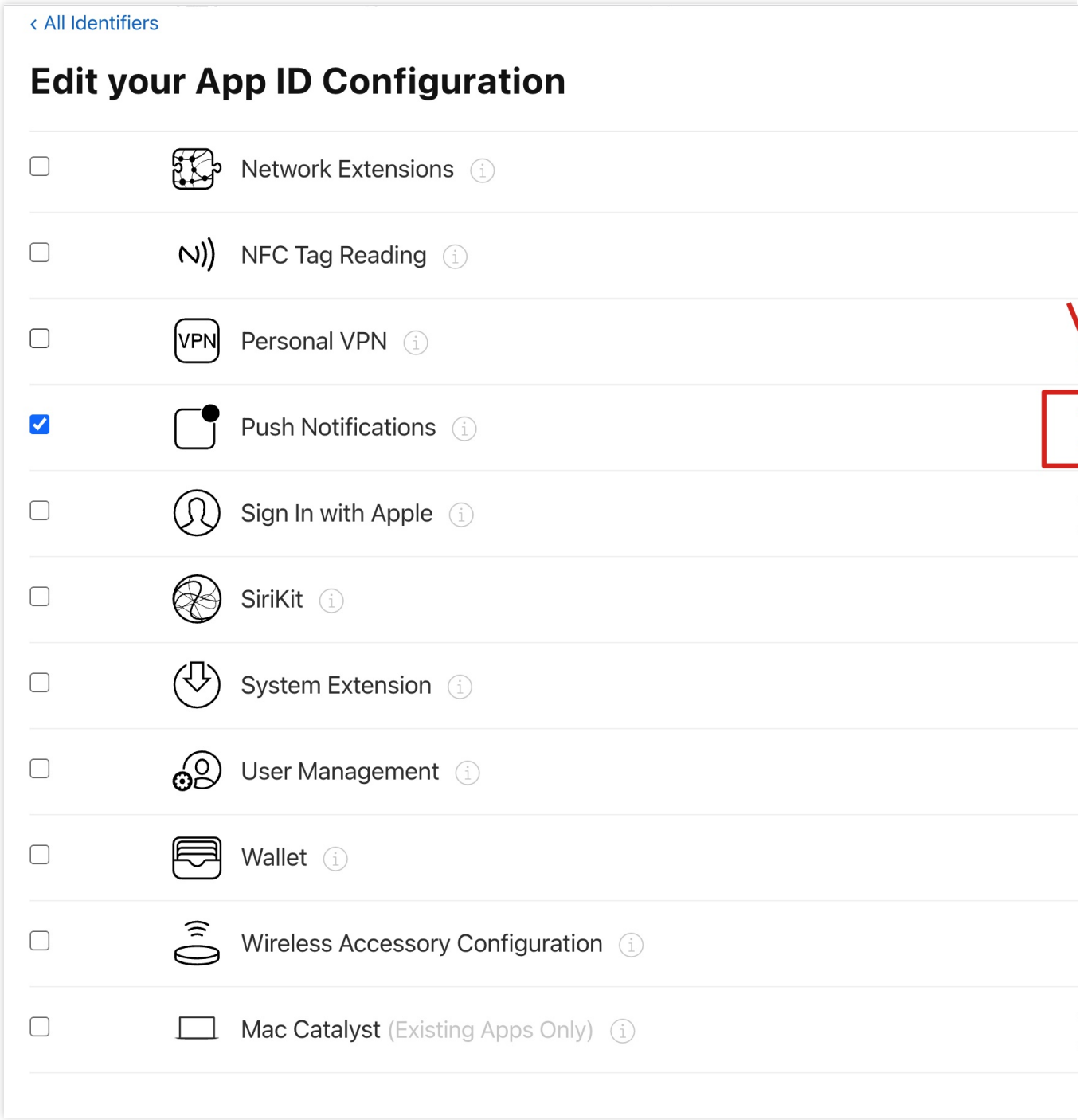


4. 勾选 **Push Notifications**，开启远程推送服务。

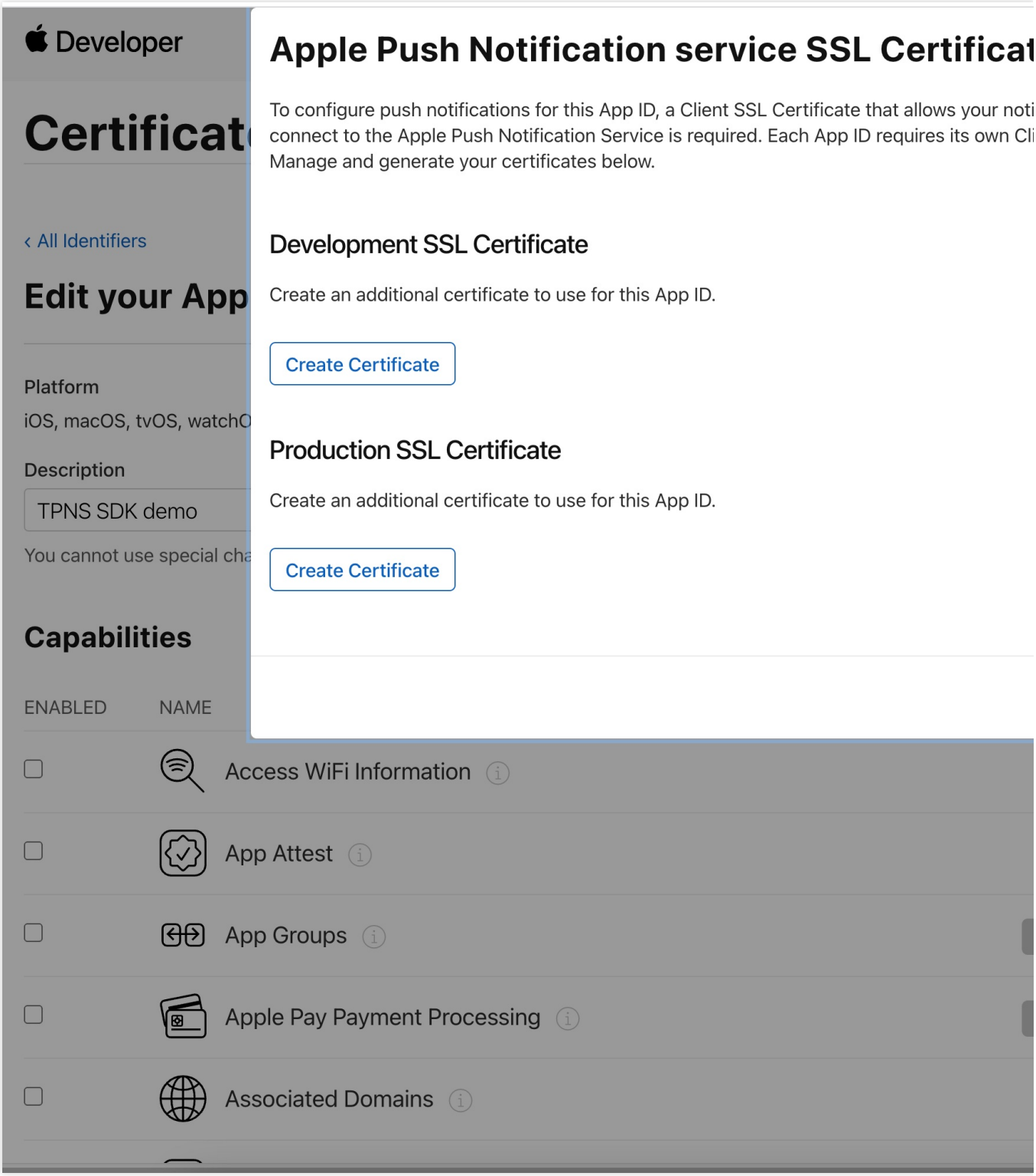


步骤2：生成并上传 P12 证书

1. 选中您的 AppID，选择 **Configure**。

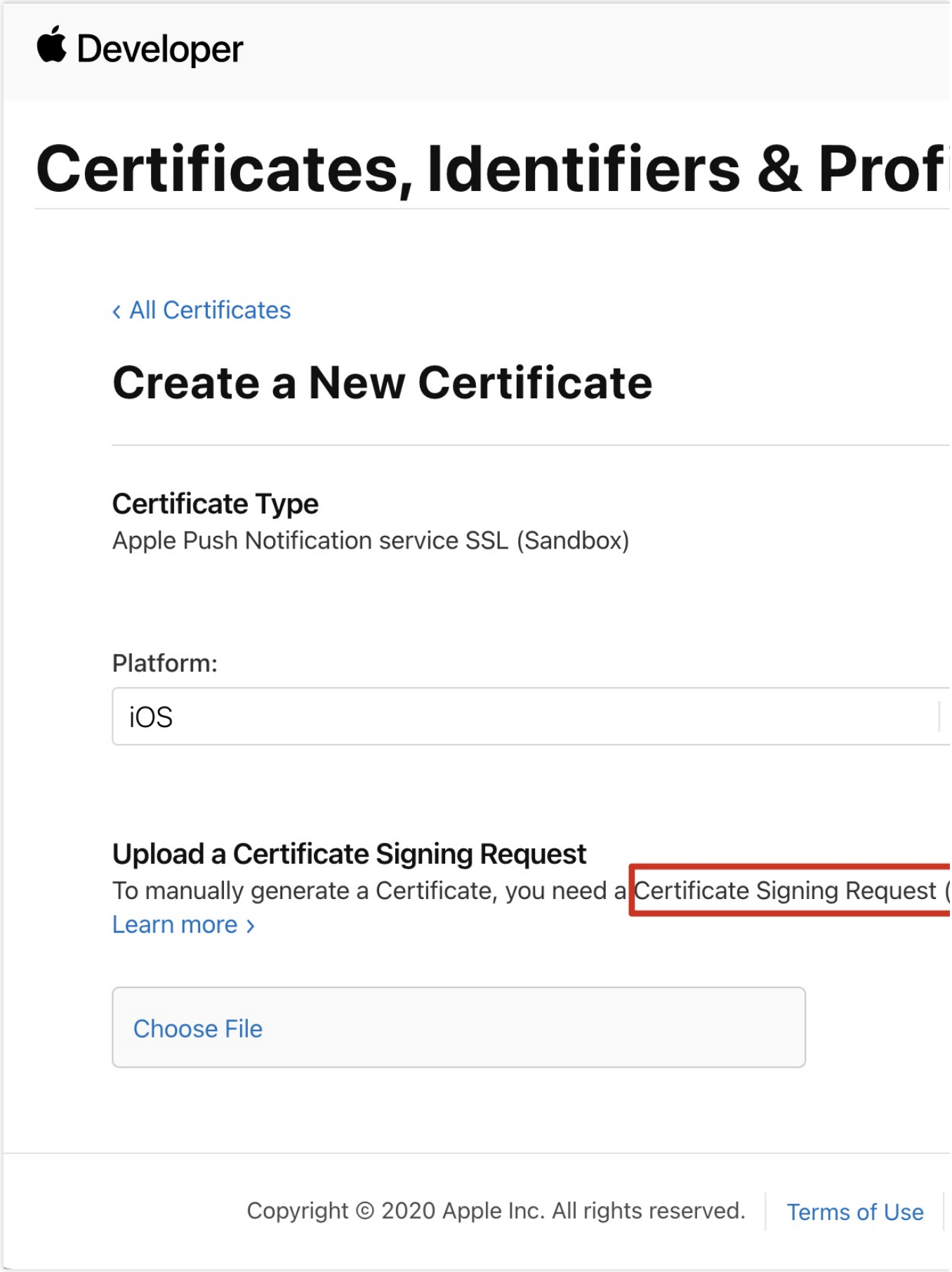


2. 可以看到在 **Apple Push Notification service SSL Certificates** 窗口中有两个 `SSL Certificate` ，分别用于开发环境（ `Development` ）和生产环境（ `Production` ）的远程推送证书，如下图所示：

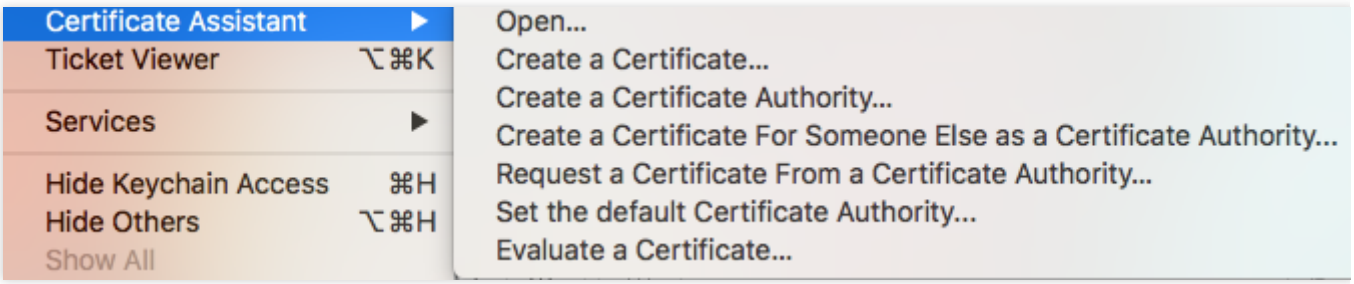


3.

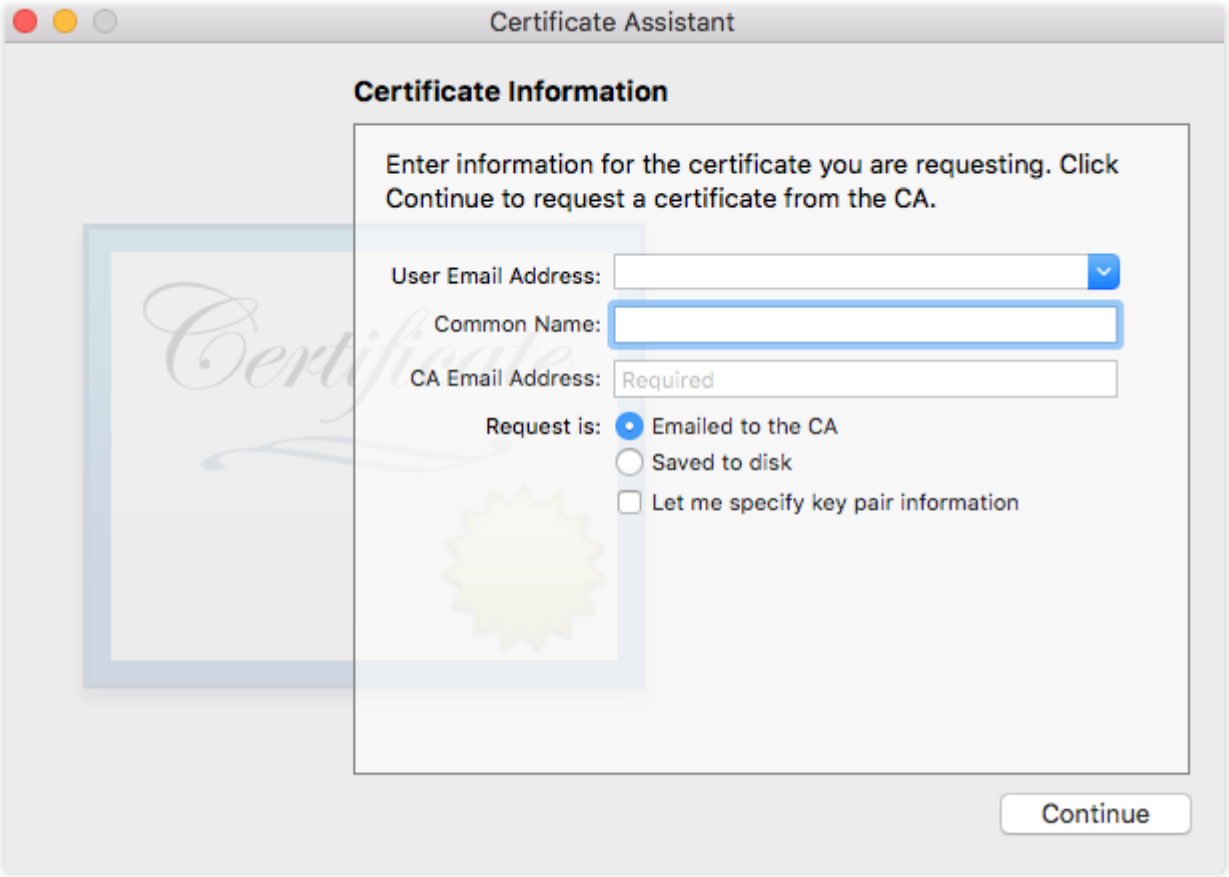
我们先选择开发环境（ `Development` ）的 **Create Certificate**，系统将提示我们需要一个 `Certificate Signing Request (CSR)` 。


Developer
Certificates, Identifiers & Prof
‹ All Certificates
Create a New Certificate
Certificate Type
Apple Push Notification service SSL (Sandbox)
Platform:
iOS
Upload a Certificate Signing Request
To manually generate a Certificate, you need a Certificate Signing Request (
Learn more ›
Choose File
Copyright © 2020 Apple Inc. All rights reserved.
Terms of Use

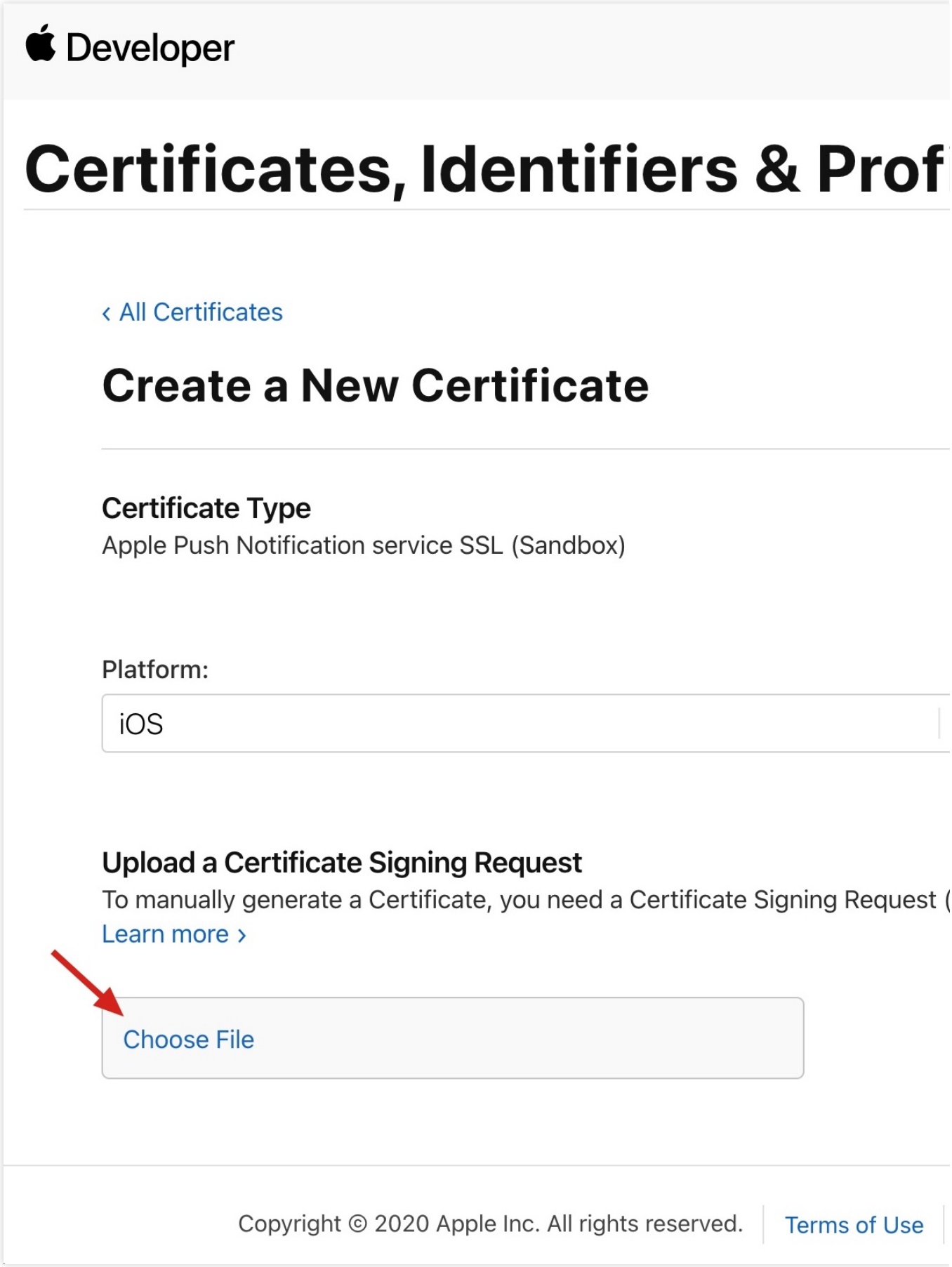
4. 在 Mac 上打开**钥匙串访问工具（Keychain Access）**，在菜单中选择**钥匙串访问 > 证书助理 > 从证书颁发机构请求证书**（`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`）。



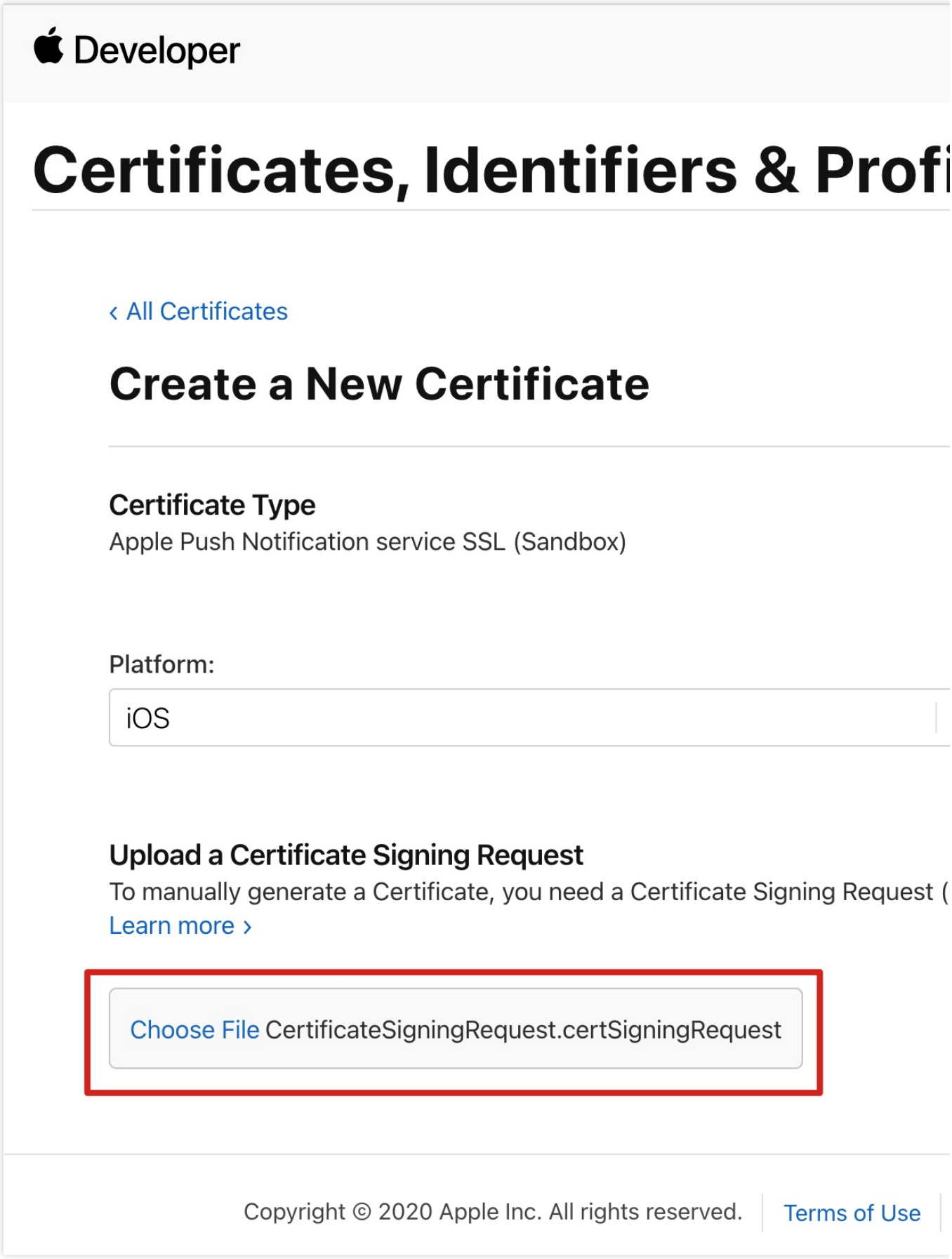
5. 输入用户电子邮件地址（您的邮箱）、常用名称（您的名称或公司名），选择**存储到磁盘**，单击继续，系统将生成一个 `*.certSigningRequest` 文件。



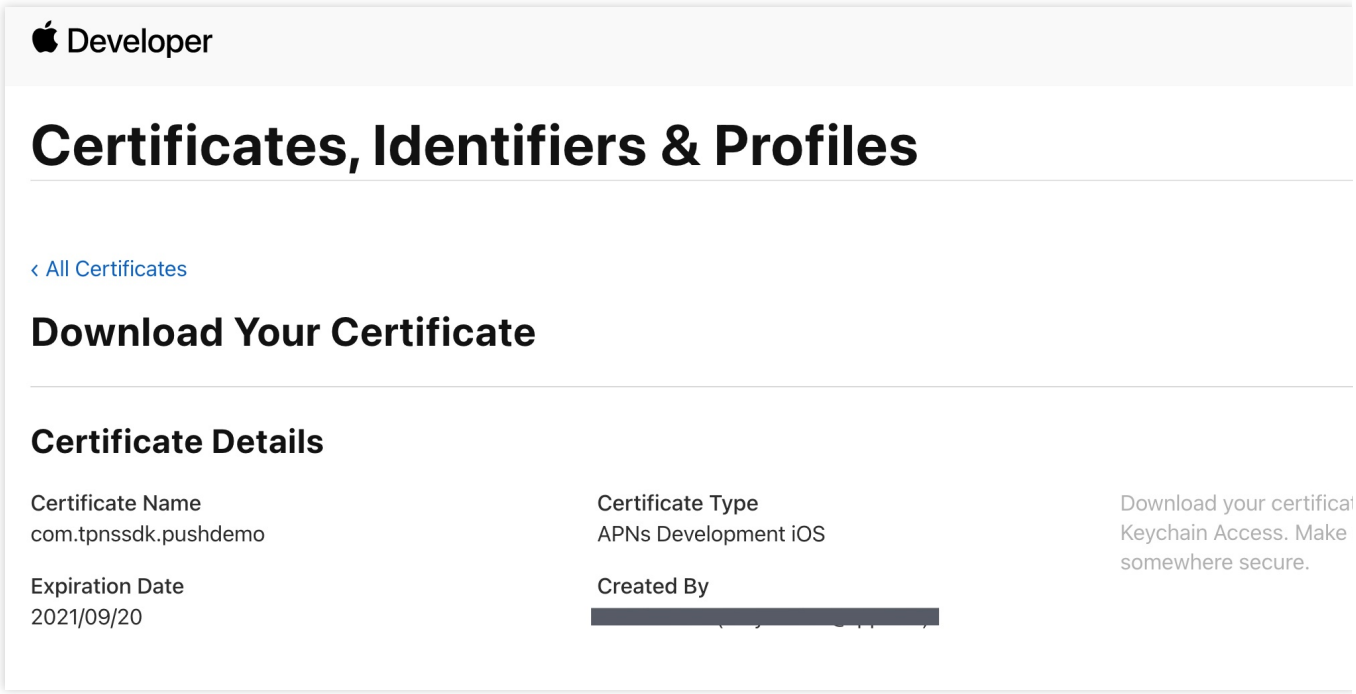
6. 返回上述 第3步骤 中 `Apple Developer` 网站刚才的页面，单击 **Choose File**上传生成的 `*.certSigningRequest` 文件。



7. 单击 **Continue**，即可生成推送证书。



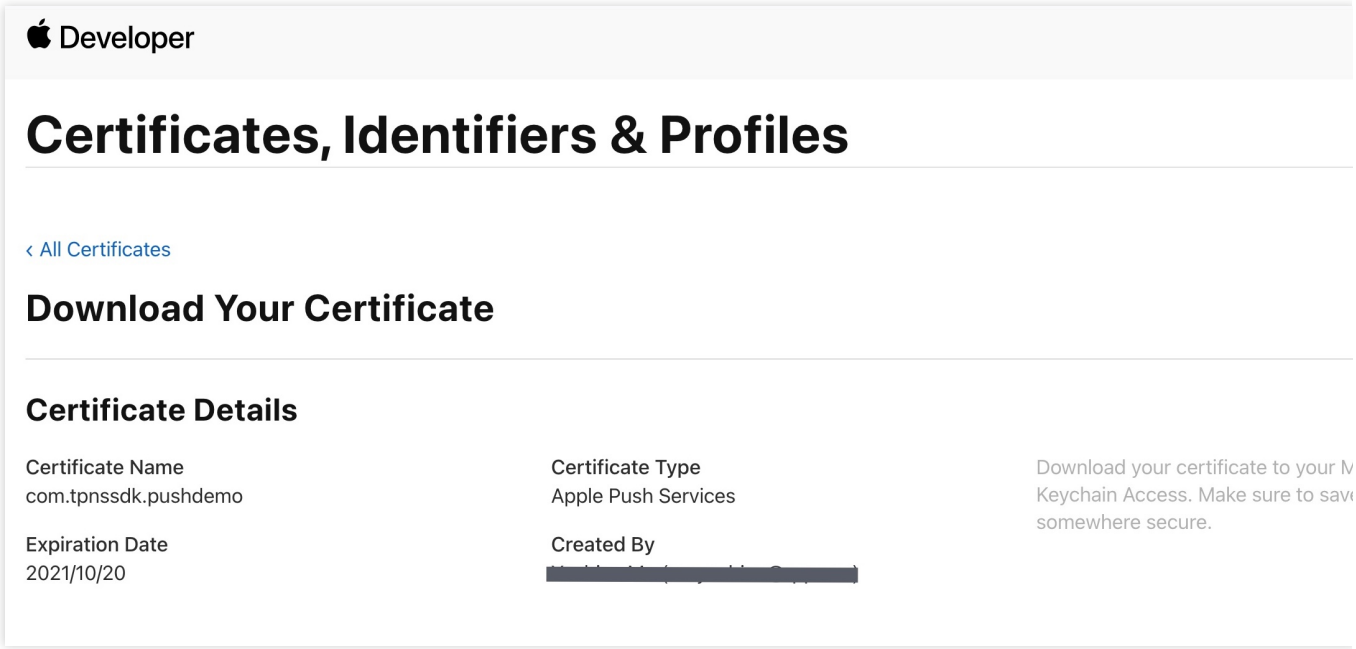
8. 单击 **Download** 下载开发环境的 `Development SSL Certificate` 到本地。



9. 按照上述步骤1 - 8，将生成生产环境的 `Production SSL Certificate` 并下载到本地。

说明：

生产环境的证书实际是开发(`Sandbox`)+生产(`Production`)的合并证书，可以同时作为开发环境和生产环境的证书使用。



10. 双击打开下载的开发环境和生产环境的 `SSL Certificate` ，系统会将其导入钥匙串中。

11. 打开钥匙串应用，在**登录 > 我的证书**，右键分别导出开发环境（ `Apple Development IOS Push Service: com.tpnssdk.pushdemo` ）和生产环境（ `Apple Push Services: com.tpnssdk.pushdemo` ）的 `P12` 文件。

注意：

保存 `P12` 文件时，请务必要为其设置密码。

步骤3：上传证书到移动推送控制台

1. 登录 移动推送控制台，选择**产品管理 > 配置管理**。
2. 进入证书上传页面，单击**上传证书**，分别上传开发环境和生产环境的证书。
3. 输入对应的证书密码，单击**点击选择**，选择您的证书。
4. 单击**上传**即可完成上传。

推送环境选择说明

最近更新时间：2024-01-16 17:42:20

管理台推送消息时，测试推送消息有两种环境可以选择。
开发环境：需要确定 App 已使用开发环境的签名证书打包，使用 `Xcode` 直接编译安装到设备。
生产环境：需要确定 App 已使用生产环境的签名证书打包，生产环境的 App 有以下3种打包方式：`Ad-Hoc` `TestFlight` `AppStore` 。

服务端指定推送环境

当您使用 `RESTAPI` 推送消息时，需要在 `PushAPI` 中指定 `environment` 字段，此字段中有两个可选值 `product` 和 `dev` 。
开发环境：需要指定 `environment` 为 `dev` 。
生产环境：需要指定 `environment` 为 `product` 。

推送证书说明

在管理台中需要上传开发环境和生产环境的二合一推送证书（Apple Push Notification service SSL (Sandbox & Production)）。此证书可以推送生产环境和开发环境，根据 App 实际使用的签名证书来进行选择，选择的方式参照上文。
说明：
App 签名证书，分为开发环境（对应 `xxx Developer:xxx` 字样的签名证书）和生产环境（对应 `xxx Distribution:xxx` 字样的签名证书），请根据实际情况选择。
App 推送证书为合并证书，已兼容开发环境与生产环境。

错误码

最近更新时间：2024-01-16 17:42:20

客户端返回码

错误码	含义
101	Guid 请求超时
701	SDK 异常
801	长链接超时
901	长链接错误
1001	deviceToken 为空，厂商 Token 未拿到
1101	设备网络错误
1102	未注册
1103	App 信息或路由配置错误
1104	业务接口操作类型传入错误
1105	业务接口参数传入错误
1106	业务接口参数为空
1107	系统不支持
1110	启动失败
1111	内存不足
1501	建立长链接失败
1502	建立长链接失败，应用非前台

服务端返回码

错误码	含义

1010001	资源未部署，请确认应用是否已购买推送资源
1008001	参数解析错误
1008002	必填参数缺失
1008003	认证失败
1008004	调用服务失败
1008006	Token 无效，请检查设备 Token 是否注册成功
1008007	参数校验失败
1008011	文件上传失败
1008012	上传文件为空
1008013	证书解析失败
1008015	推送任务 ID 不存在
1008016	日期时间参数格式不对
1008019	被内容安全服务判定不和谐
1008020	证书包名校验失败
1008021	p12 证书格式内容校验失败
1008022	p12 证书密码不对
1008025	创建应用失败，产品下已存在该平台的应用
1008026	批量操作，部分失败
1008027	批量操作，全部失败
1008028	超出限频
1008029	Token 校验非法
1008030	App 未付费
1008031	App 资源已销毁
10110008	查询的 Token， 账号不存在
10010005	推送目标不存在

10010012	非法的推送时间，请更改推送时间。 定时推送时 send_time 传入的值如果是过去的时间，具体规则如下： 如果 | send_time - 当前时间 | <= 10min， 推送任务会创建，接收任务后会立即调度 如果 | send_time - 当前时间 | > 10min， 推送任务会被拒绝， 接口返回失败
10010018	重复推送
10030002	AccessID 和 AccessKey 不匹配

拓展功能
通知服务扩展的使用说明

最近更新时间：2024-01-16 17:42:20

简介

为了**精准统计消息抵达率和接收富媒体消息**，SDK 提供了 Service Extension 接口，可供客户端调用，从而可以监听消息的到达和接收富媒体消息，您可以按以下指引使用此功能。

创建通知拓展 Target

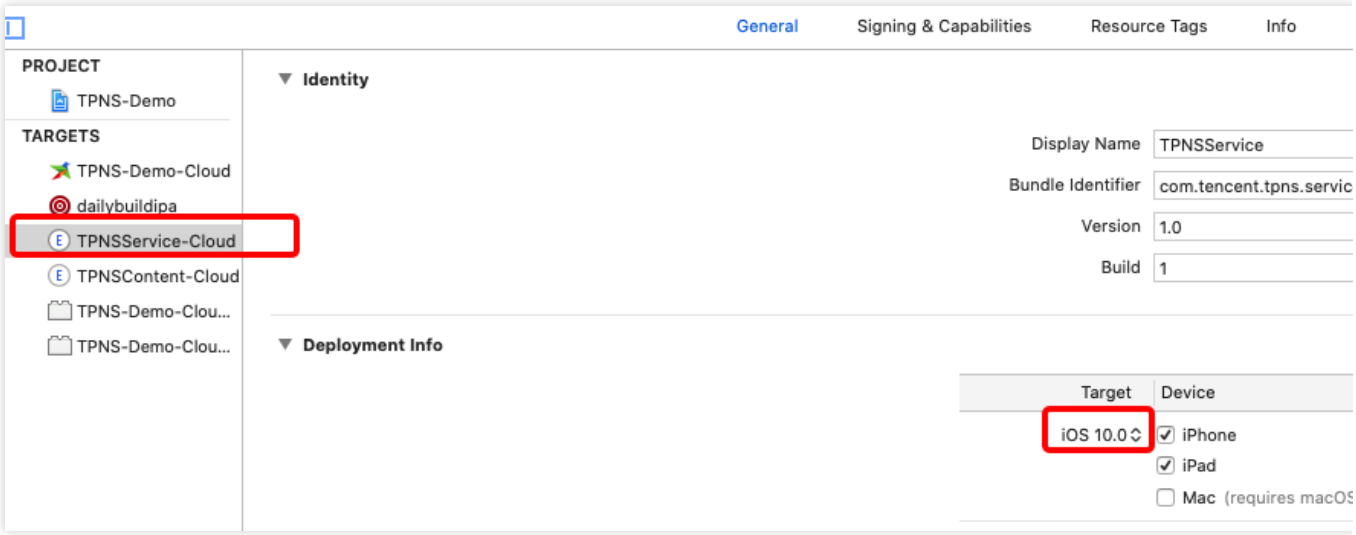
1. 在 xcode 菜单栏，选择 **File** > **New** > **Target**。

说明：

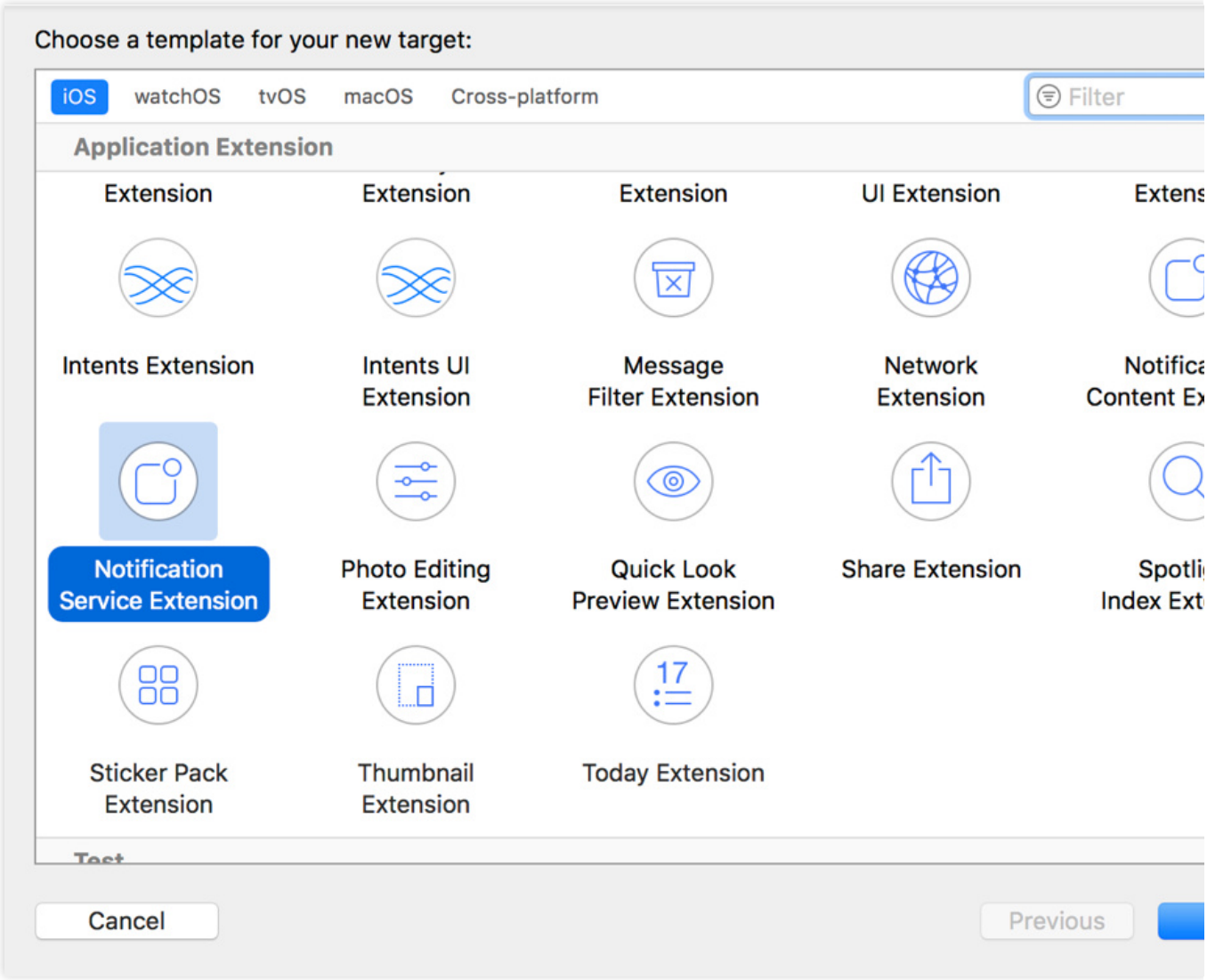
主工程的 Bundle Id 和 Service 的 Bundle Id 必须不同，且 Service 的 Bundle Id 必须以主工程的 Bundle Id 为前缀（例如，主工程的 Bundle Id：com.tencent.tpns，Service 的 Bundle Id：com.tencent.tpns.service）。

若主工程 Target 最低支持版本小于10.0，扩展 Target 系统版本设置为10.0。

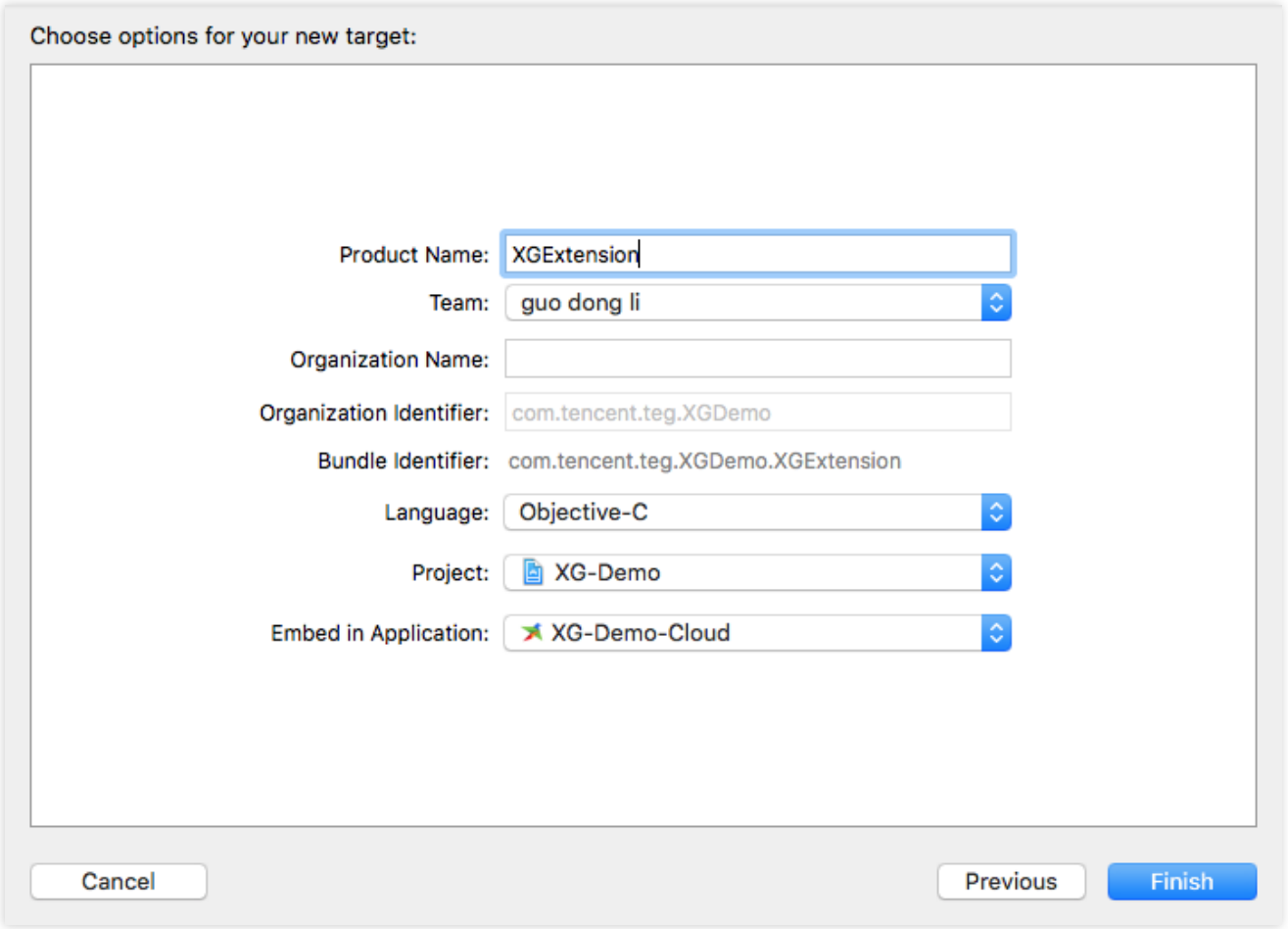
若主工程 Target 最低支持版本大于10.0，则扩展 Target 系统版本与主工程 Target 版本一致。



2. 进入 Target 页面，选择 **Notification Service Extension**， 单击 **Next**。



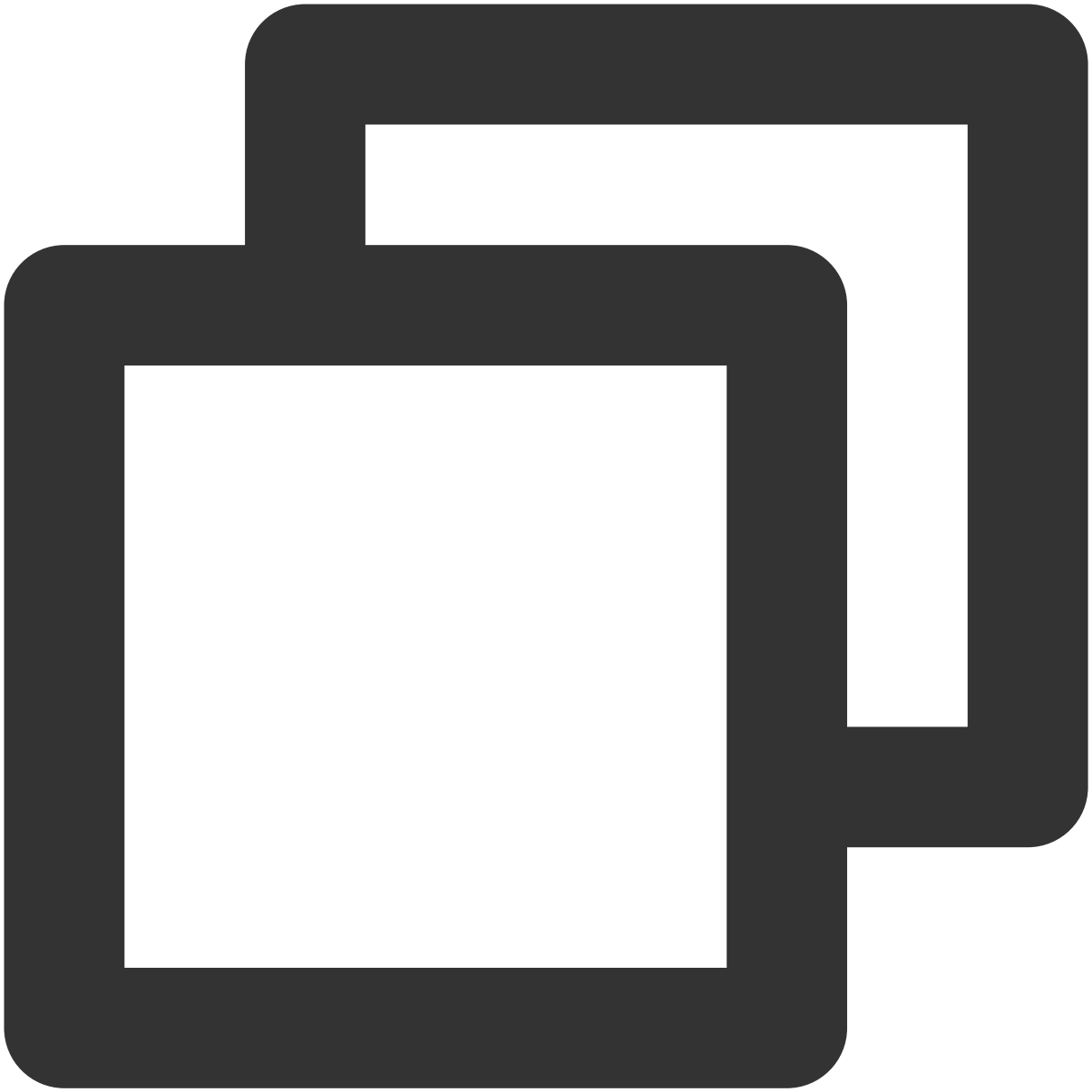
3. 输入 Product Name，单击 **Finish**。



添加移动推送扩展库（三选一）

方式一：Cocoapods 集成

通过 Cocoapods 下载地址：

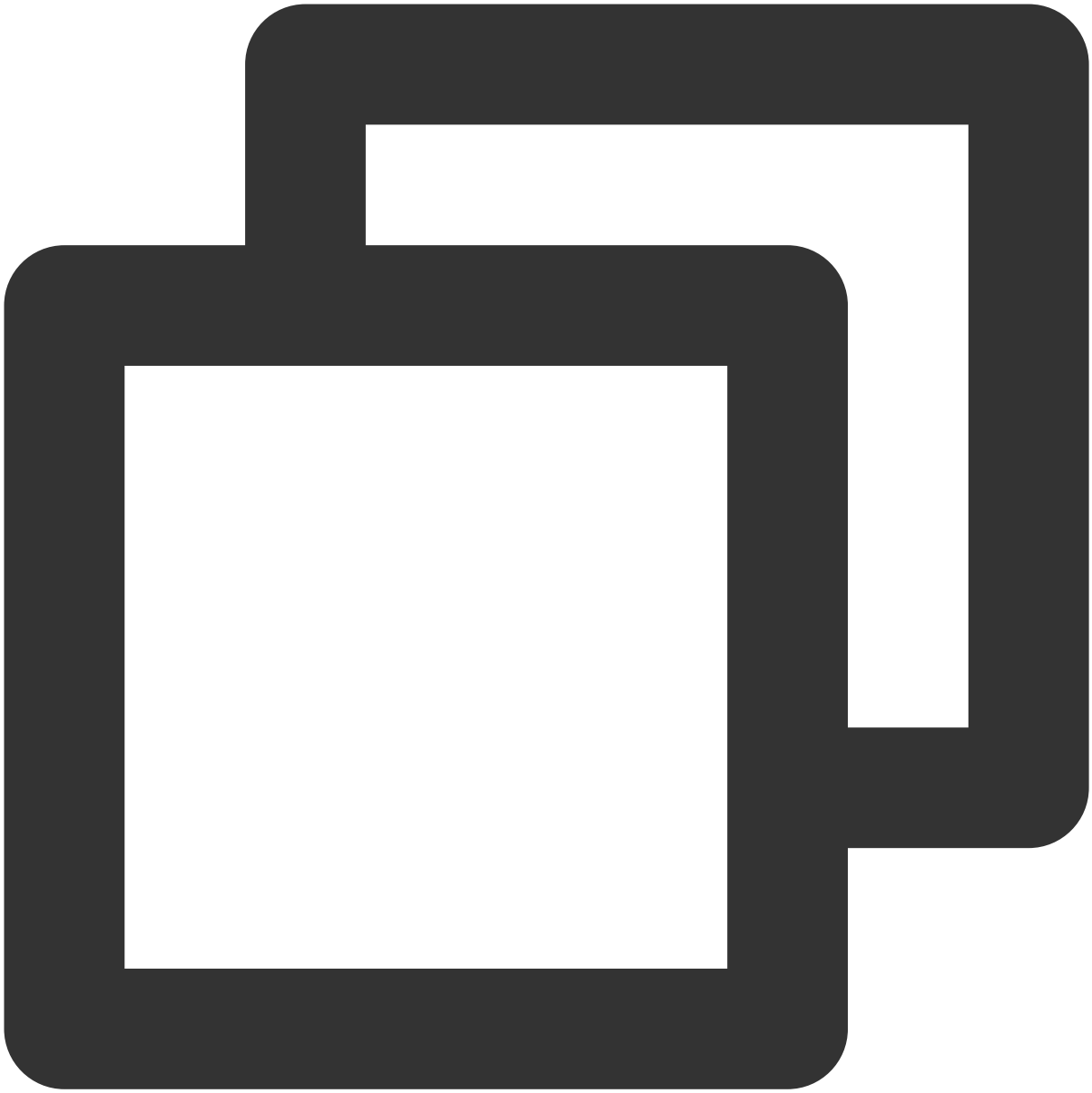


```
pod 'TPNS-iOS-Extension', '~> 版本'  // 如果不指定版本，则默认为本地 pod TPNS-iOS-Extens
```

使用说明：

1. 创建类型为 `Application Extension` 的 `Notification Service Extension` TARGET，例如 `XXServiceExtension` 。
2. 在 Podfile 新增 XXServiceExtension 的配置栏目。

Podfile 中增加配置项目后展示效果，示例如下：



```
target ‘XXServiceExtension'do
  platform:ios,'10.0'
  pod 'TPNS-iOS-Extension' , '~> 版本' // 需要与主SDK（TPNS-iOS）版本保持一致
end
```

方式二：手动集成

1. 登录 腾讯移动推送控制台。
2. 在左侧导航栏中，单击**工具箱** > **SDK 下载**，进入 SDK 下载管理页面。
3. 在 SDK 下载管理页面，选择 iOS 平台，单击**下载**。

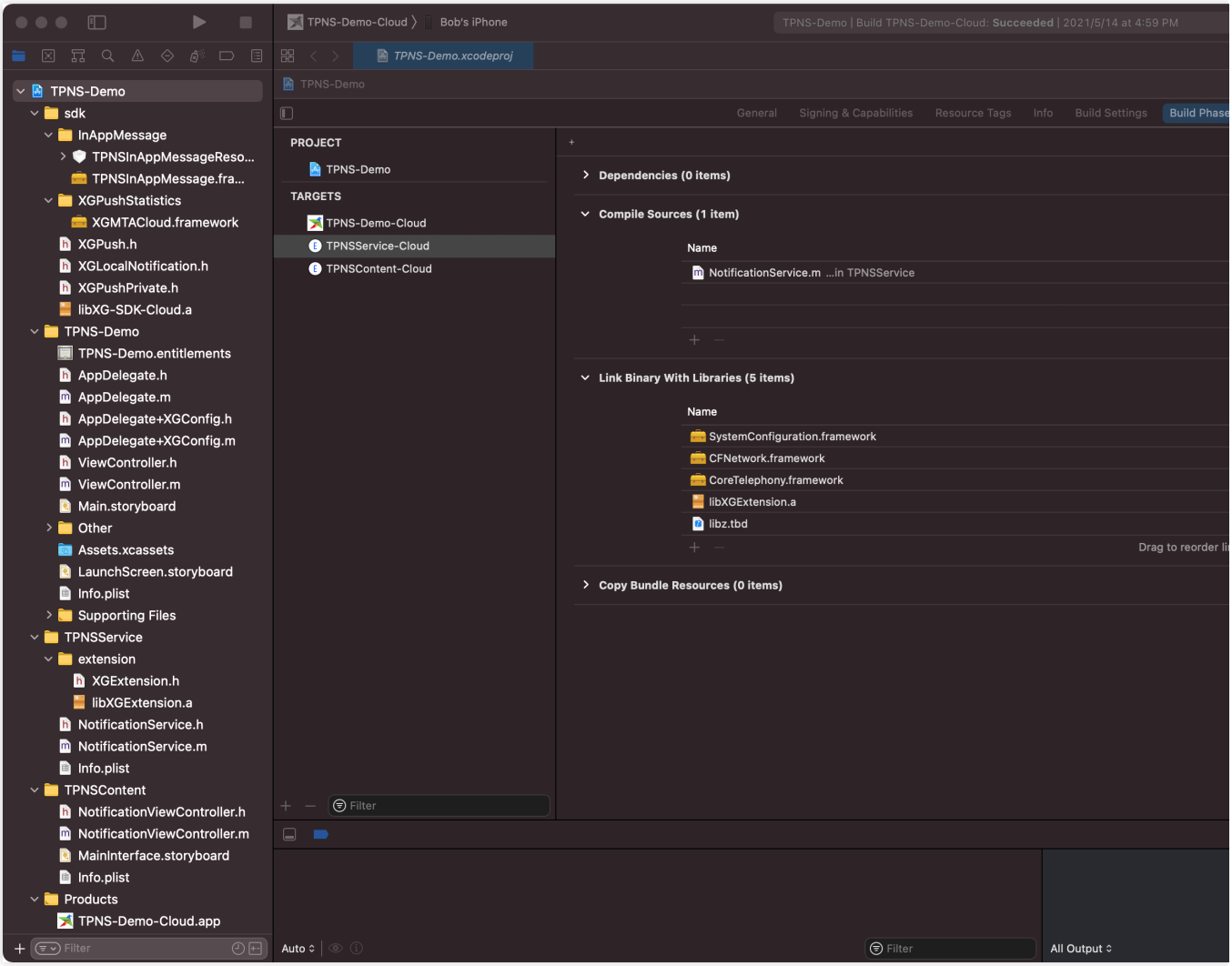
4. 解压缩 SDK 包，并依次打开 demo > sdk > XGPushStatistics > extension 目录，获取 `XGExtension.h` 及 `libXGExtension.a` 文件。

5. 将获取到的 `XGExtension.h` 及 `libXGExtension.a` 文件添加至通知扩展 Target：

添加系统库：libz.tbd

移动推送扩展库：libXGExtension.a

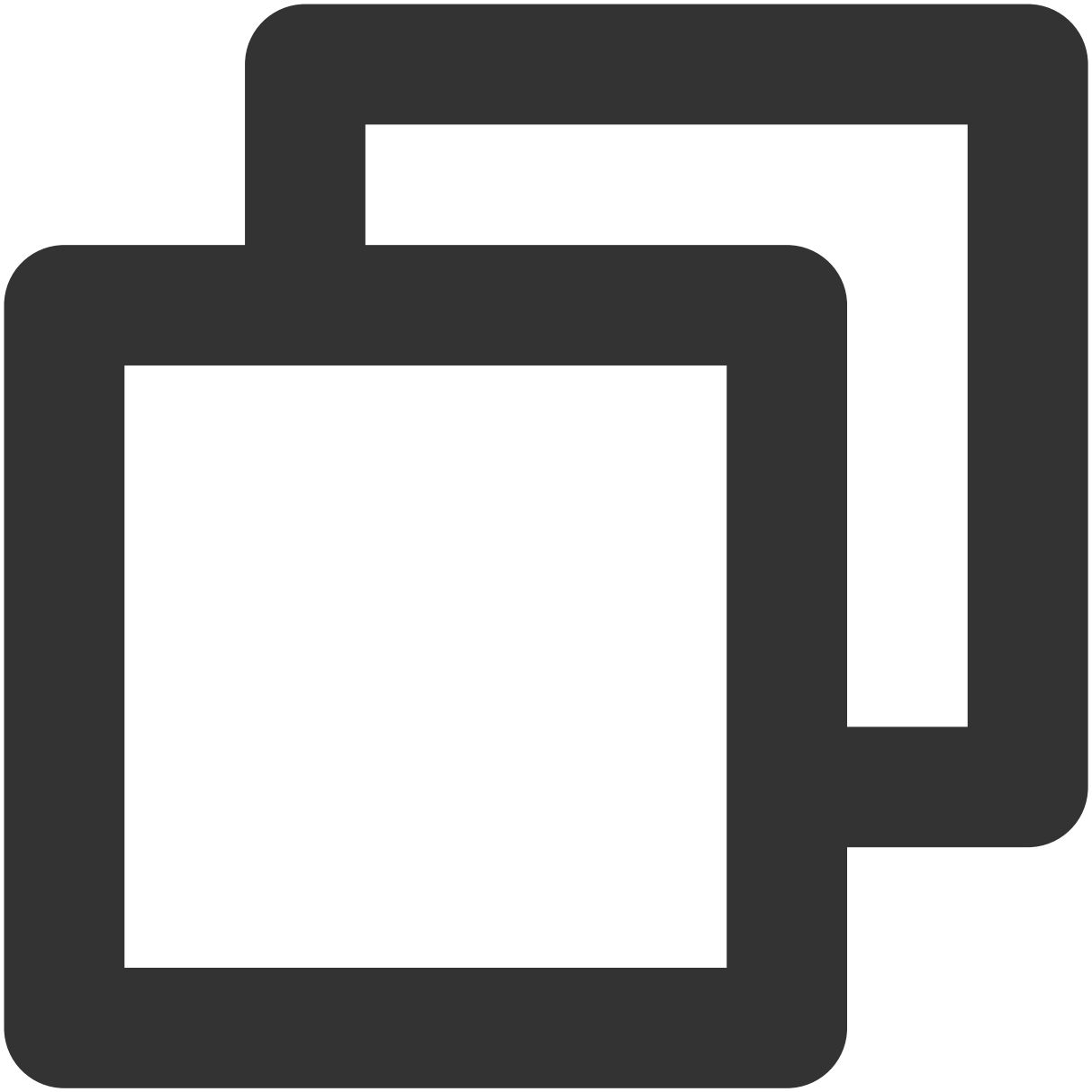
集成后的目录结构如下：



方式三：HomeBrew 集成

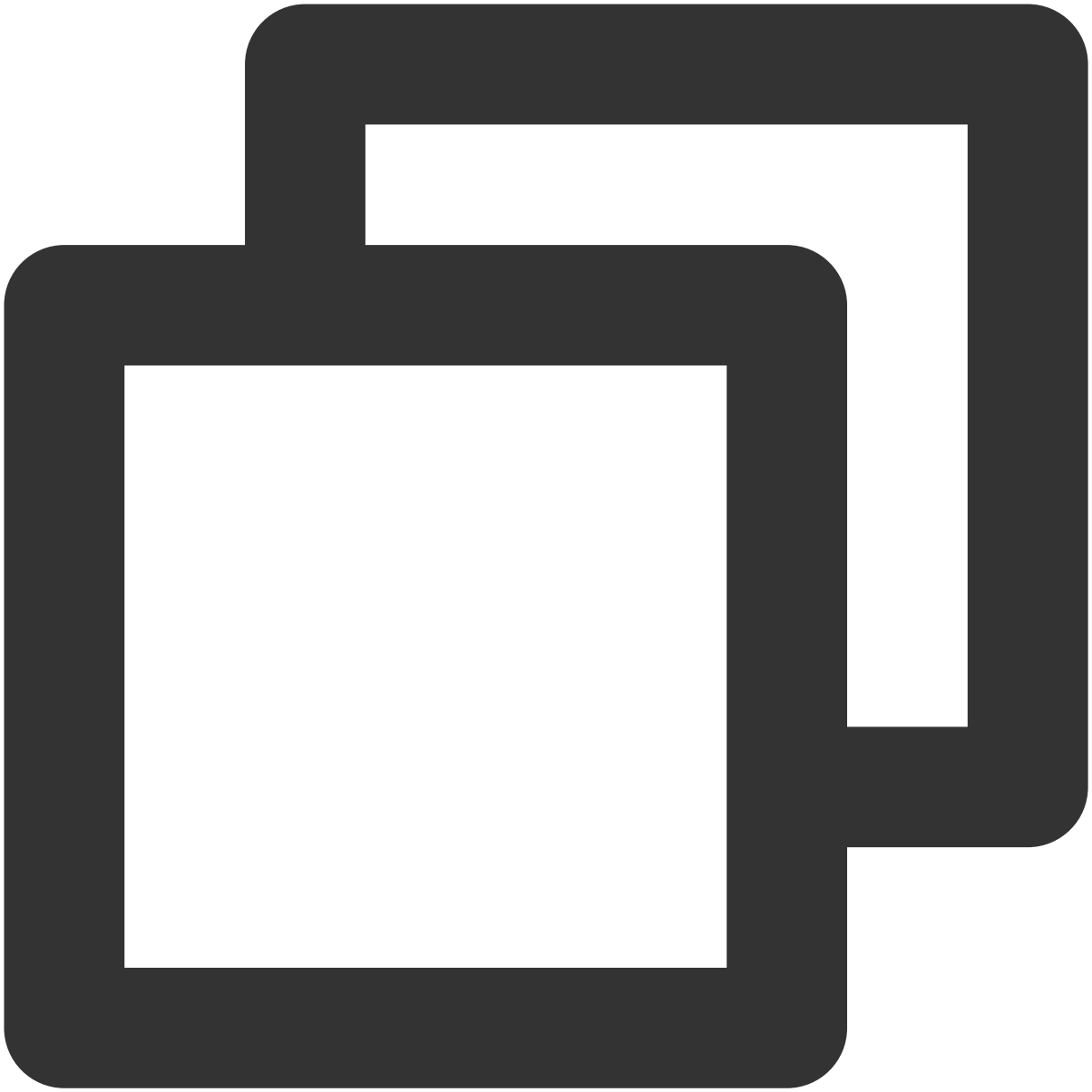
首次安装 `new_tpns_svc_ext`，请在终端执行下面的命令：

1. 关联 `移动推送 homebrew` 仓库。



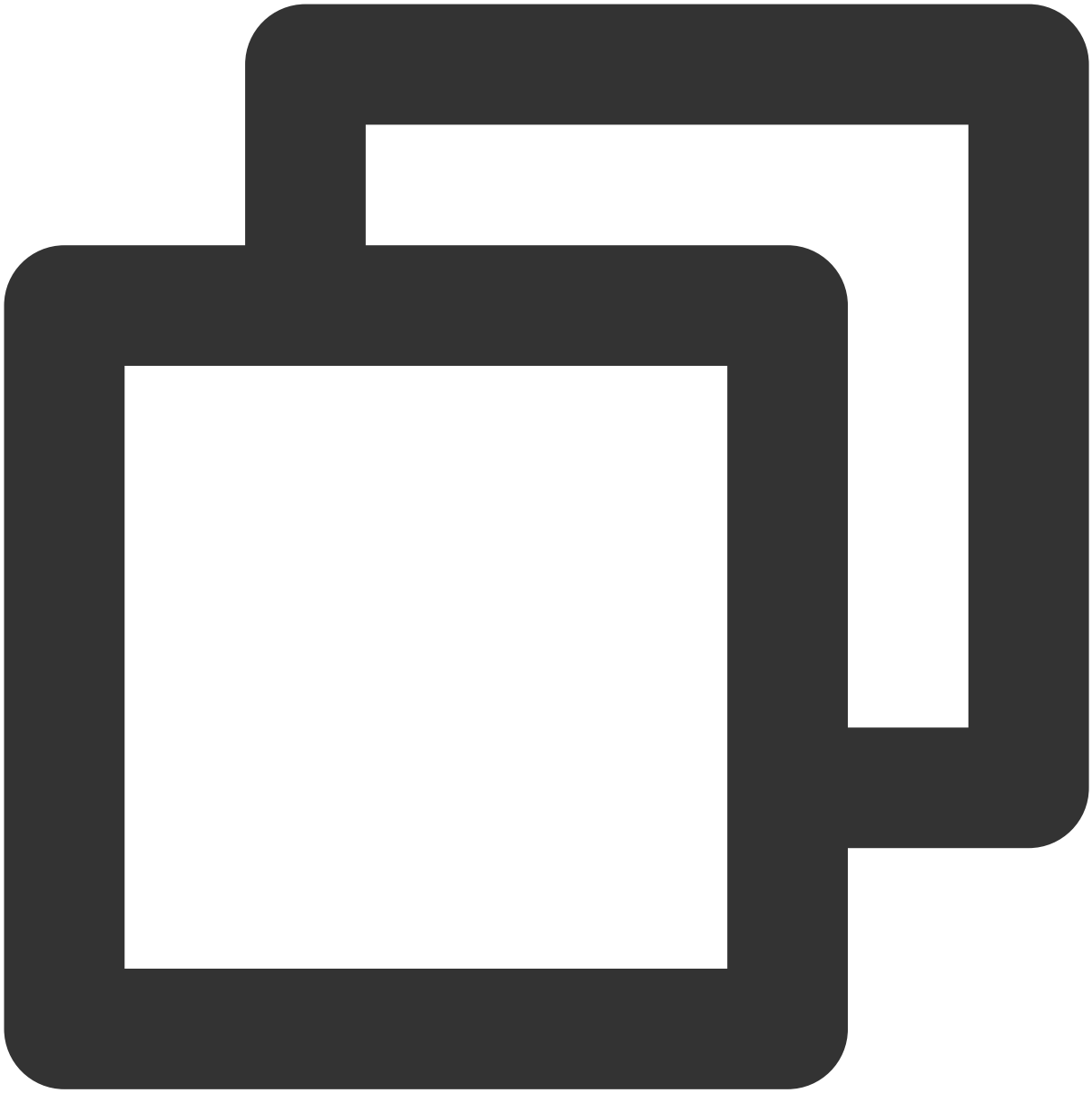
```
brew tap tpns/serviceExtension https://github.com/TencentCloud/homebrew-tpnsService
```

2. 安装 `new_tpns_svc_ext` 命令行。



```
brew install new_tpns_svc_ext
```

3. 安装移动推送通知扩展插件。



```
new_tpns_svc_ext  "AccessID" "AccessKey" "xxx.xcodeproj"
```

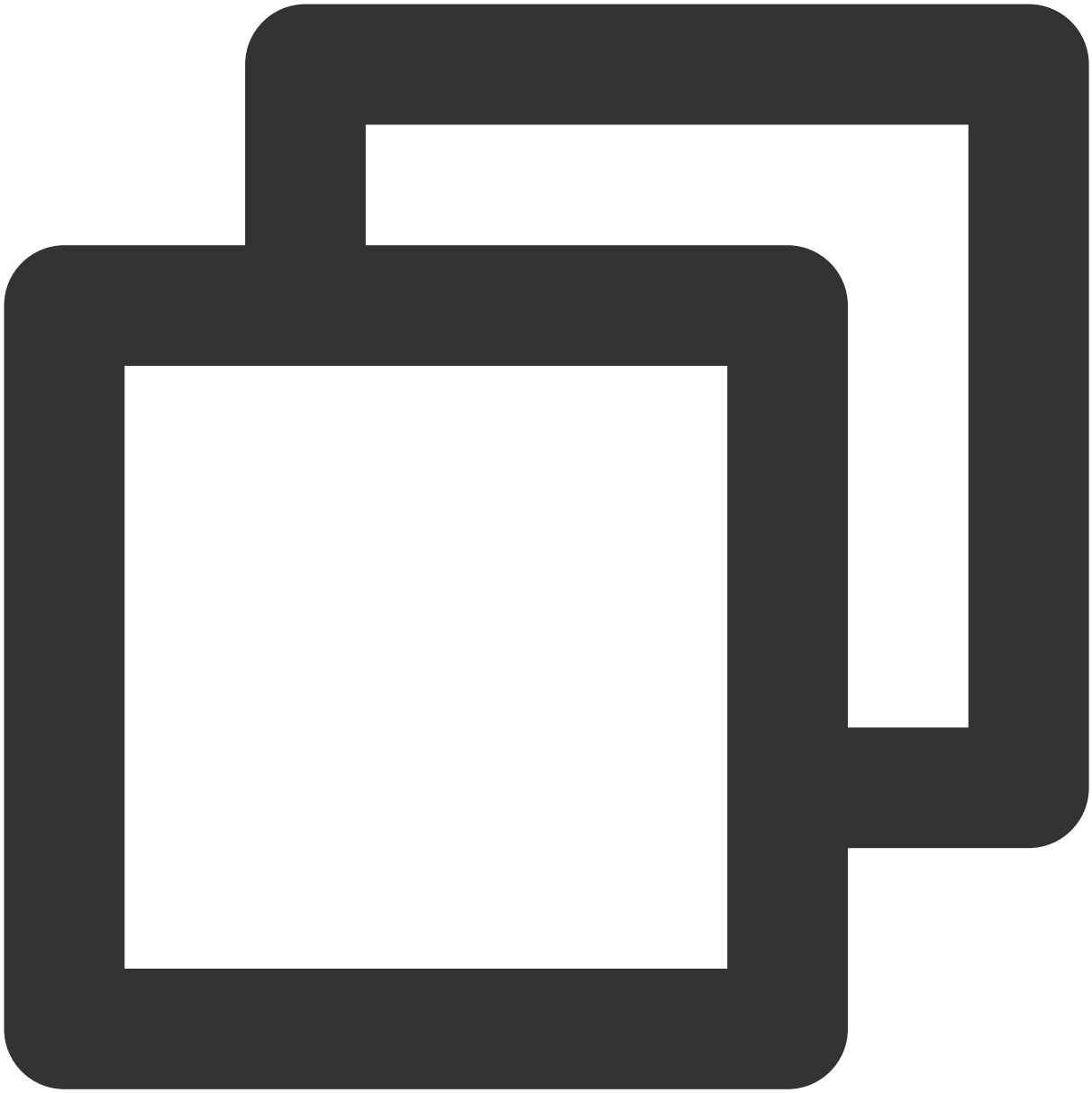
参数说明：

参数1：腾讯云-腾讯移动推送-您产品的 `AccessID`

参数2：腾讯云-腾讯移动推送-您产品的 `AccessKey`

参数3： `.xcodeproj` 的完整路径

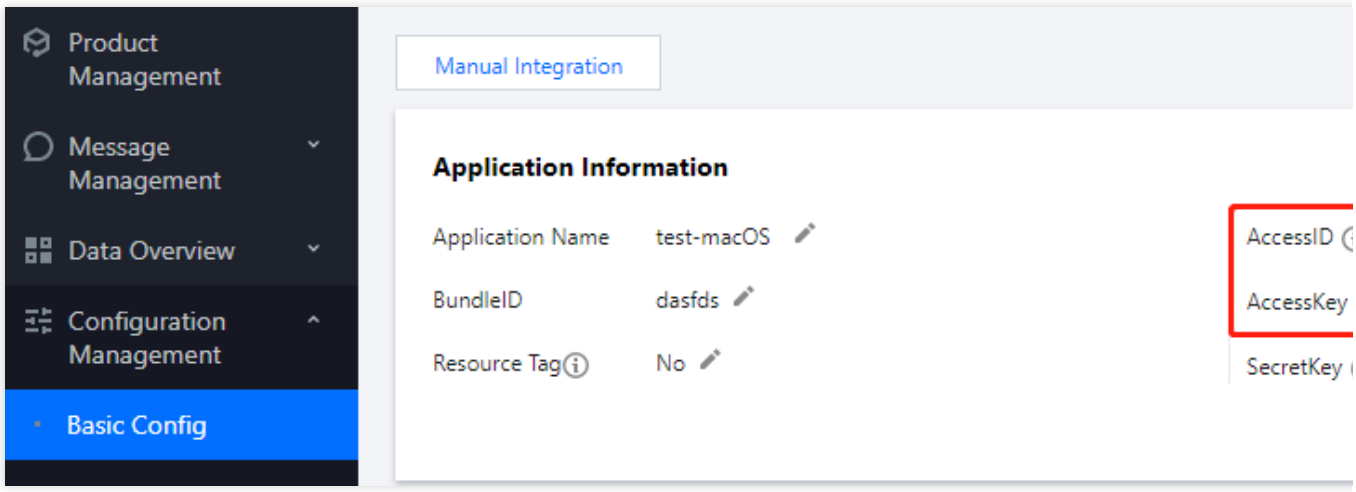
使用示例：



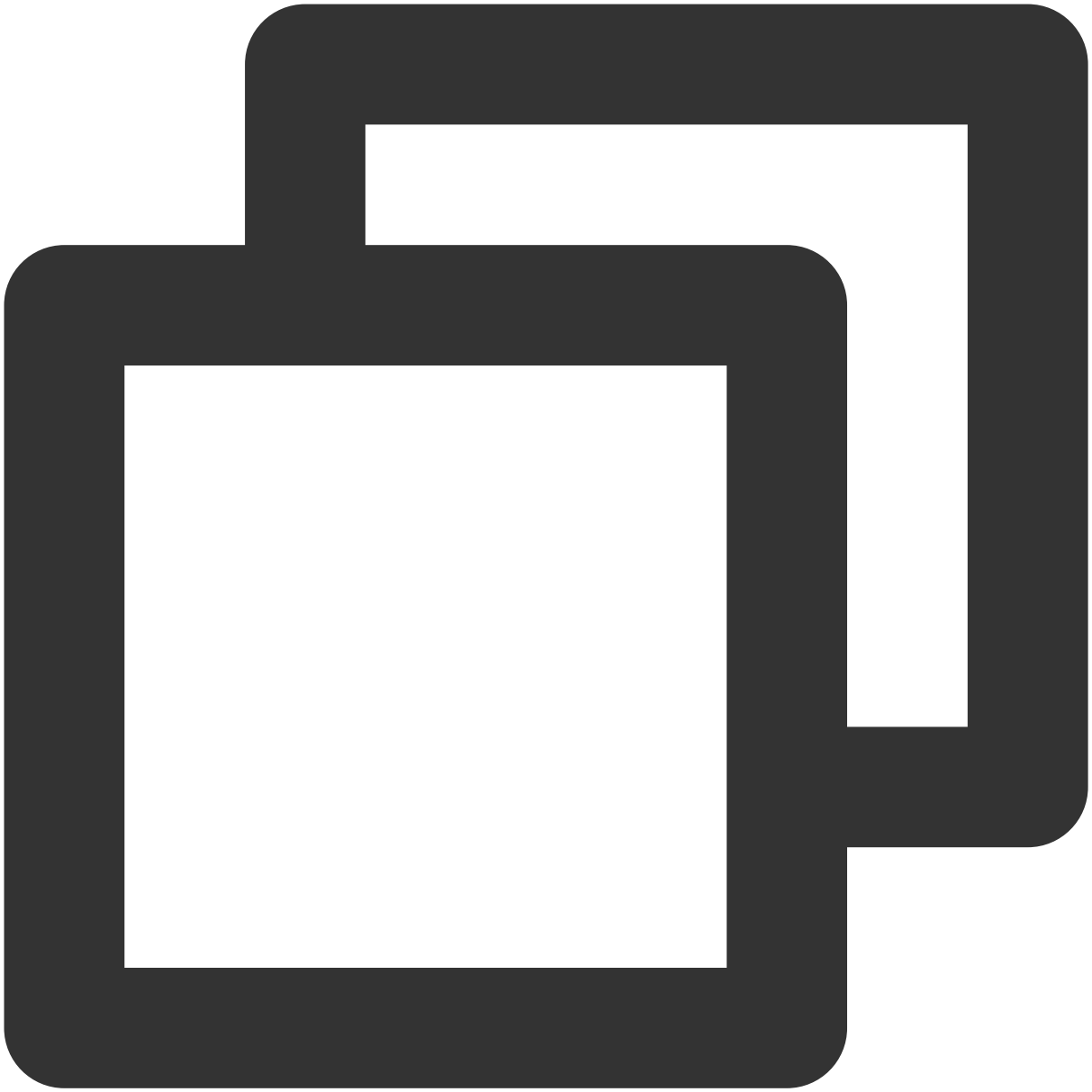
```
new_tpns_svc_ext "1600013400" "IWRNAHX6XXK6" "/Users/yanbiaomu/Developer/tencent/de
```

说明：

参数1、2的获取方式：请在 **移动推送控制台**>**产品管理**>**选中您要配置推送能力的产品**>**配置管理**中，粘贴复制 `AccessID` 和 `AccessKey` 到命令行 `new_tpns_svc_ext` 的参数1、2中。



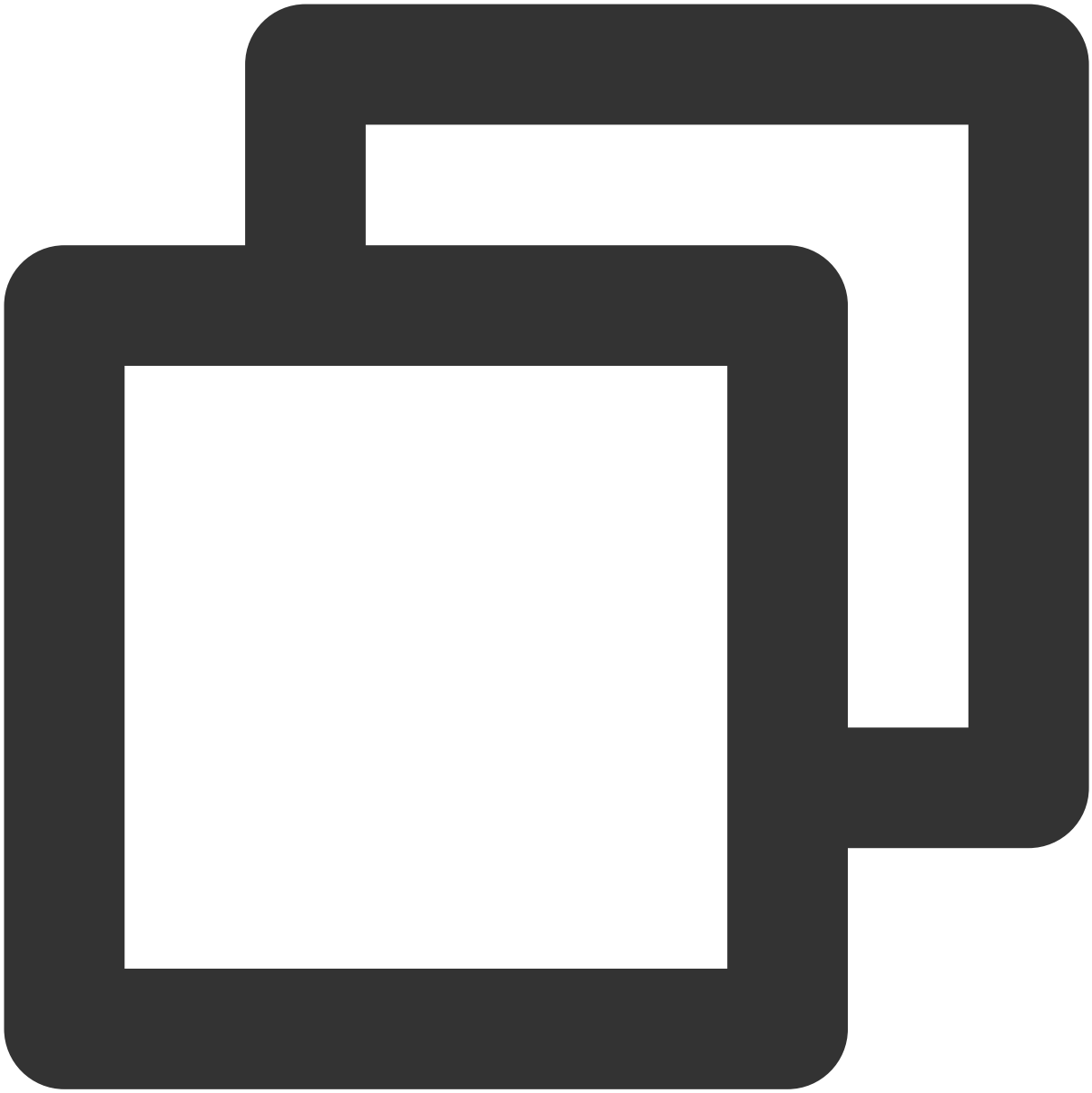
1. 执行 `new_tpns_svc_ext` 命令，进行结果验证。在终端执行 `new_tpns_svc_ext` 命令之后，如果输出如下结果，即说明集成通知扩展插件成功。



```
TPNS service auto coding done!
New TPNSService Extension Success
```

升级 `new_tpns_svc_ext` 版本

当有新版本 SDK 通知扩展插件发布，可在终端执行如下命令进行升级：



```
brew update && brew reinstall new_tpns_svc_ext
```

说明：

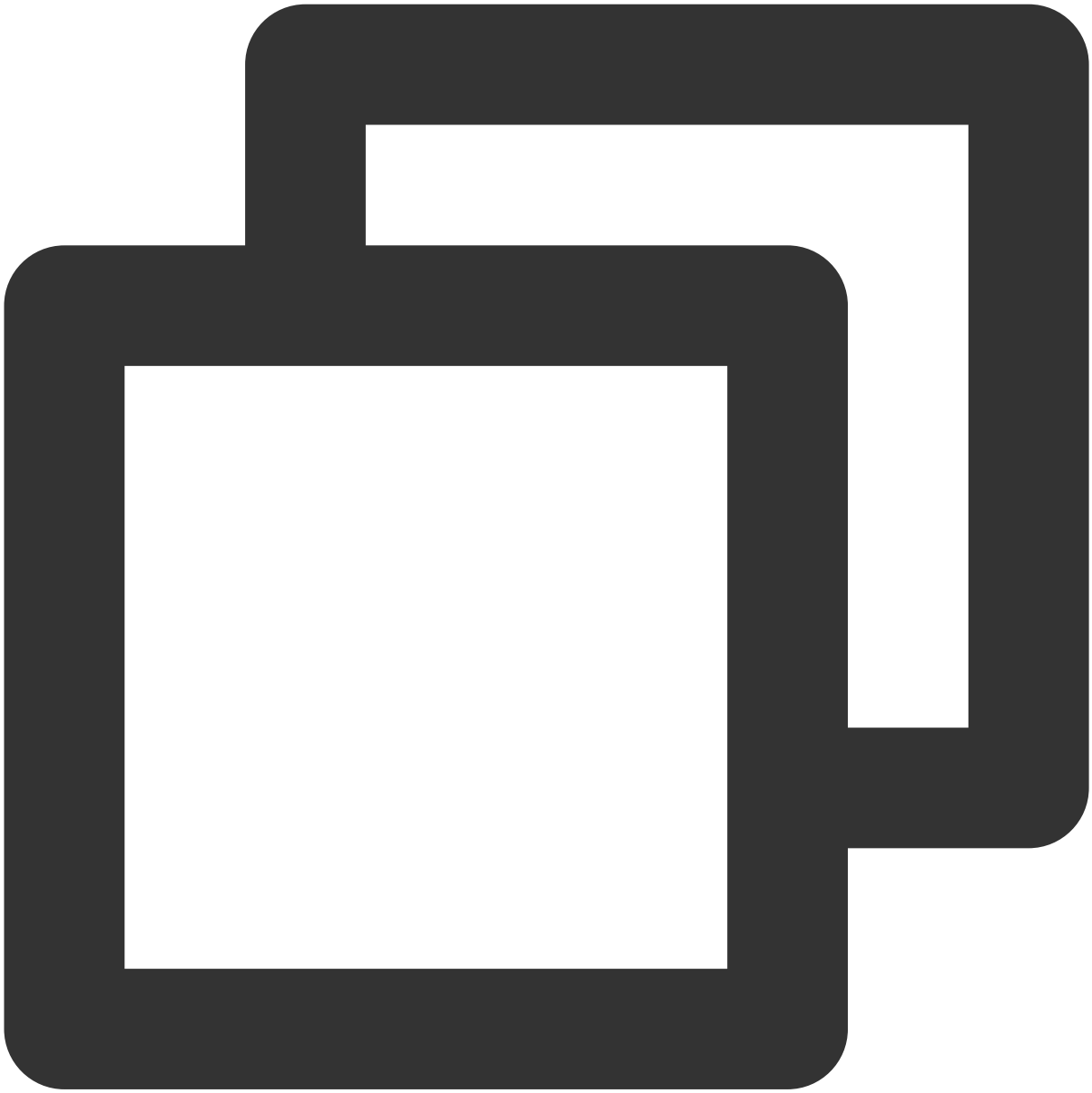
您可以在 iOS 发布动态页 查看最新版本更新详情。

当前 `homebrew` 命令 `new_tpns_svc_ext` 只支持集成通知扩展插件 `TPNSService`，暂不支持基础推送能力的集成。

使用方式

调用 SDK 统计上报接口

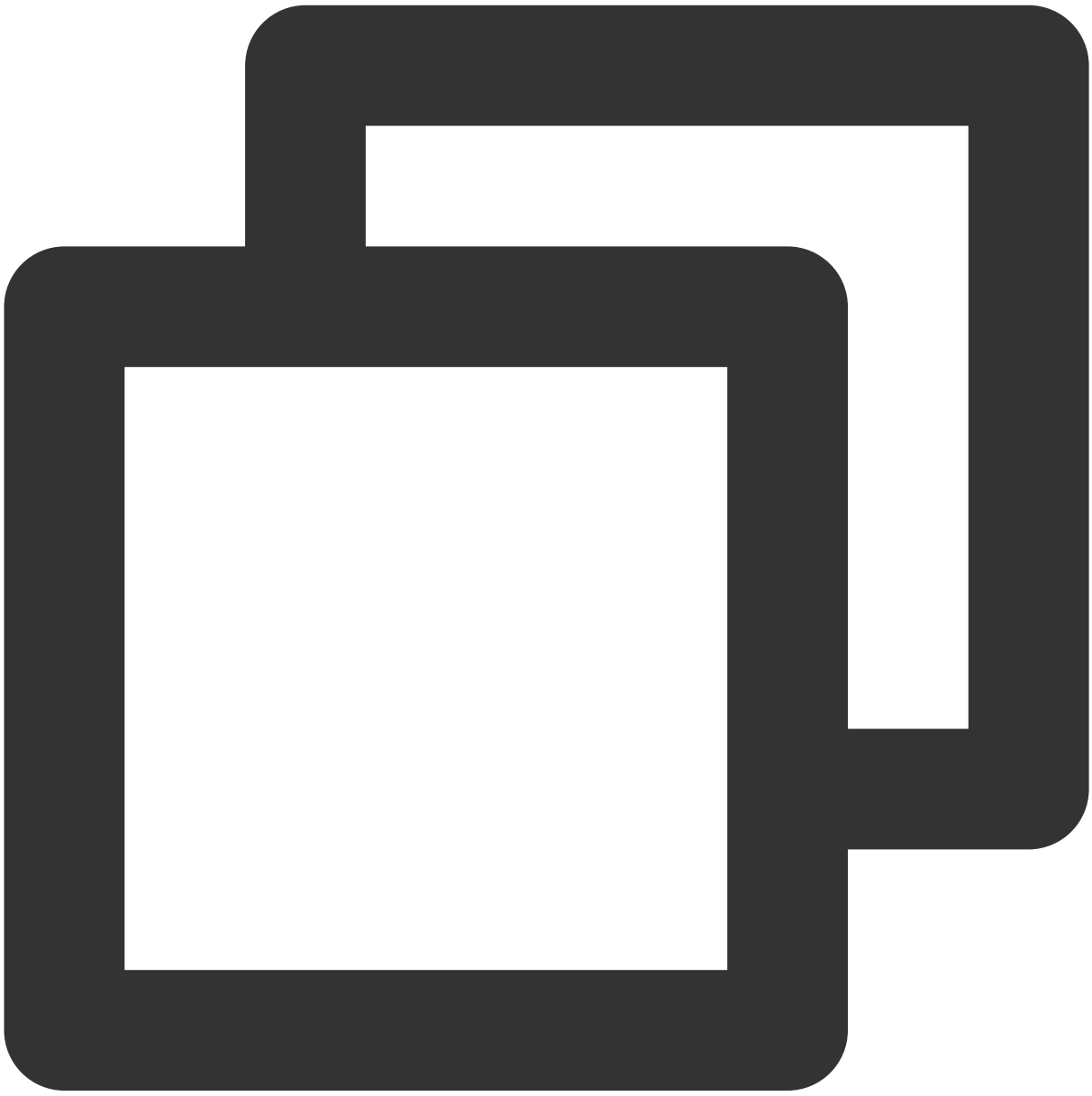
1. 在通知扩展类 NotificationService 引入头文件 XGExtension.h。
2. 在回调方法 didReceiveNotificationRequest:withContentHandler: 中调用如下示例代码：



```
  /**
@brief TPNS处理富媒体通知和抵达到终端的消息，即消息回执
@param request 推送请求
@param accessID TPNS应用 accessId
@param accessKey TPNS应用 accessKey
@param handler 处理消息的回调，回调方法中处理关联的富媒体文件
*/
```

```
(void)handleNotificationRequest:(nonnull UNNotificationRequest *)request
                        accessID:(uint32_t)accessID
                       accessKey:(nonnull NSString *)accessKey
                  contentHandler:(nullable <span class="hljs-keyword">void</span> (^)(NSAr
```

示例代码



```
- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentH
    self.contentHandler = contentHandler;
    self.bestAttemptContent = [request.content mutableCopy];
     /// 非广州集群，请开启对应集群配置（广州集群无需使用）
```

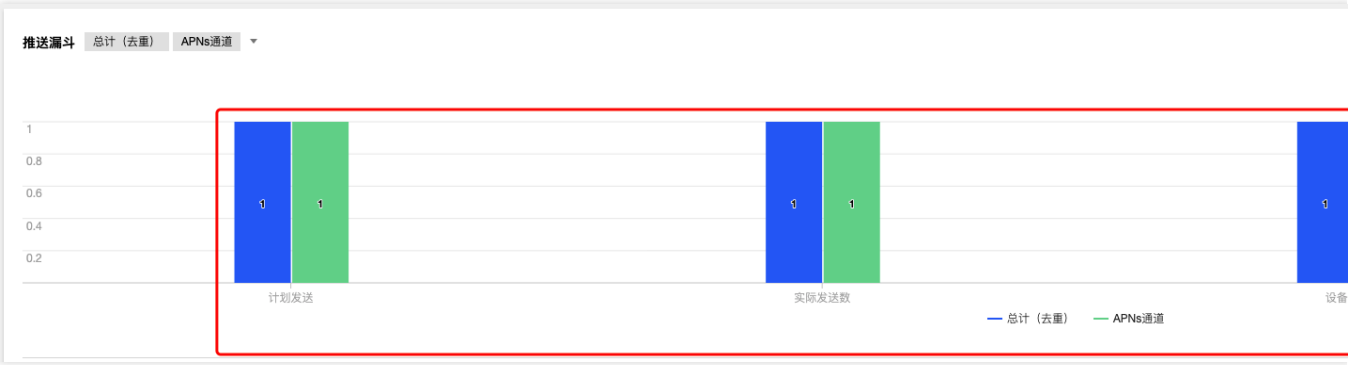
```
    //  [XGExtension defaultManager].reportDomainName = @"tpns.hk.tencent.com"; ///
    //  [XGExtension defaultManager].reportDomainName = @"tpns.sgp.tencent.com";  /
    //  [XGExtension defaultManager].reportDomainName = @"tpns.sh.tencent.com";  //
    [[XGExtension defaultManager] handleNotificationRequest:request accessID:<your
        > contentHandler:^(NSArray<UNNotificationAttachment *> * _Nullable attachme
        self.bestAttemptContent.attachments = attachments;
        self.contentHandler(self.bestAttemptContent);  // 如果需要在弹出通知前增加业务逻
    }];
}
```

接入验证

在您按以上流程完成接入后，可按如下步骤验证插件接入是否成功：

1. 关闭应用，给手机推送一条通知消息。
2. 在不点击的情况下，管理台查看消息抵达情况。

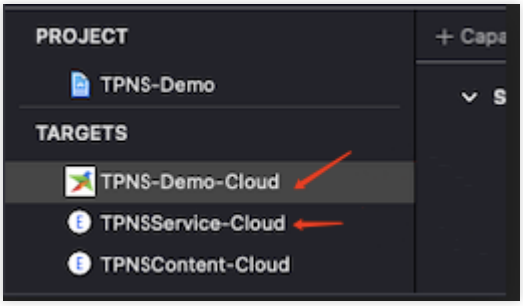
若有抵达数据，表示集成成功。



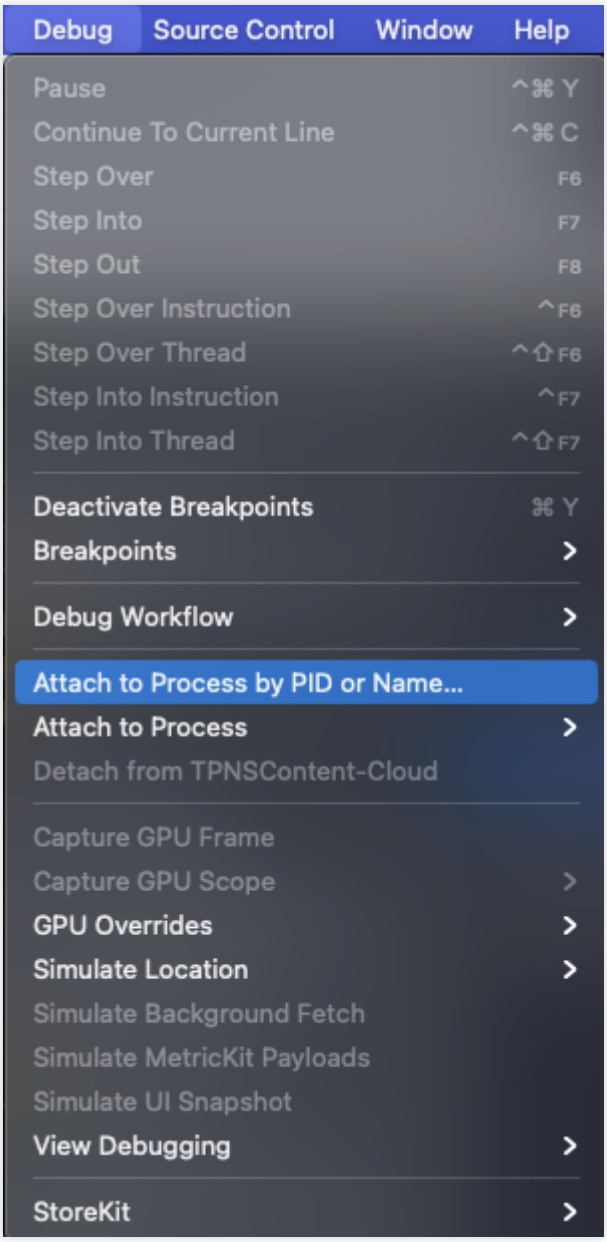
调试方式

若设备收到推送，但无抵达数据，可按照以下步骤进行调试：

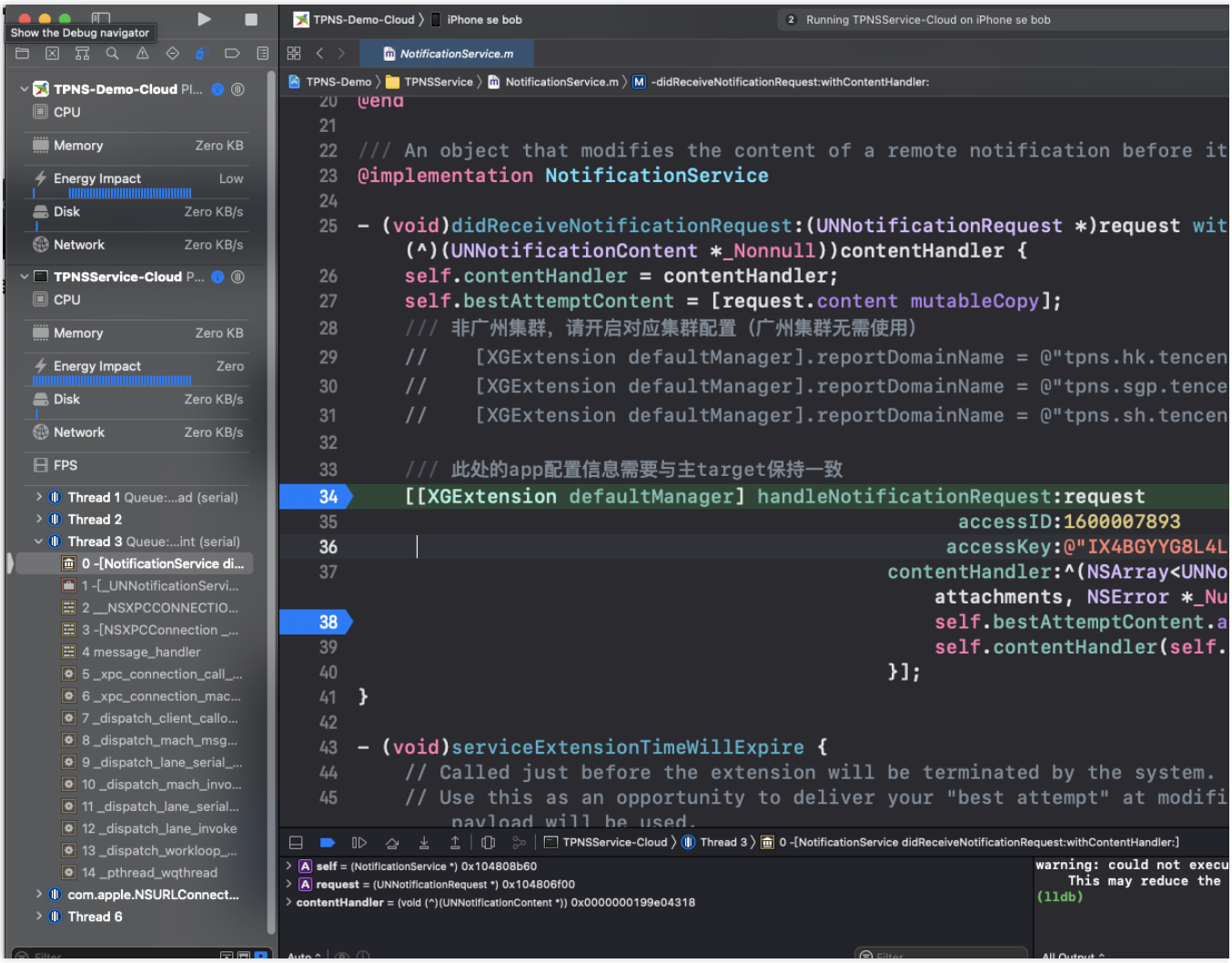
1. 运行主 Target（图中为 Demo 示例）



2. 通过 PID 或者 Name 将 UNNotificationServiceExtension 的实现 Target（Demo 中抵达插件为 TPNSService-Cloud）连接到主 Target。



3. 在如图代码34行、38行的位置加断点，发送一条通知配合调试，注意需要使用 APNs (Apple Push Notification service) 通道，通知的内容中 mutable-content 字段必须为1（移动推送SDK 1.2.8.0开始，后台默认使用 APNs 通道，前台使用移动推送自建通道，调试抵达插件需要将 App 退到后台后）。如果看到断点被执行了，说明调试成功，否则停止所有 Target，重新重第1步开始。



常见问题

为何发送通知后，却没有看到抵达上报？

客户端抵达上报需要集成**通知扩展服务插件**。若集成后，还没有抵达数据上报，则需要排查主工程与抵达插件的 ACCESS ID 和 ACCESS KEY 是否一致。 且需要排查 Web 管理台或者 RestAPI 通知的内容中 mutable-content 字段是否为1。

只有主工程与抵达插件的 ACCESS ID 和 ACCESS KEY 一致， 且 Web 管理台或者 RestAPI 通知的内容中 mutable-content 字段为1，才会运行客户端抵达插件，从而有抵达数据上报。

客户端集成插件

最近更新时间：2024-01-16 17:42:20

除了原生的 Android SDK 与 iOS SDK 之外，移动推送 TPNS 还提供主流的开发工具集成插件。

官方维护

官方维护的版本放在 **腾讯工蜂-tpns** 上以开源的形式发布。如果需要下载打包版本，请单击相应项目页面中的 **下载**，下载您需要的插件包。
官方插件地址包含：安装方法、demo（example 文件夹内）、API 说明，请开发者参阅。

项目	地址
Unity	官方地址
Flutter	官方地址
React-Native	官方地址
Swift Demo	官方地址
Cordova	官方地址

macOS接入指南

简介

最近更新时间：2024-01-16 17:42:20

macOS 端实现推送消息的服务涉及三个角色：终端应用（Client App），APNs（Apple Push Notification service），移动推送服务器（移动推送Provider）。在使用移动推送服务实现给客户端推送消息，需要这三个角色在整个流程中相互配合，任何一个角色出现异常都可能会导致消息无法推送。

文件组成

XG_SDK_Cloud_macOS.framework (主SDK文件)
XGMTACloud_macOS.framework （“点击上报”组件）

版本说明

支持 macOS 10.8+。
针对 macOS10.14+ 以上版本。
需要额外引入 UserNotification.framework。
建议使用 Xcode 10.0+。

主要功能

macOS SDK 是移动推送服务为客户端实现消息推送而提供给开发者的接口，主要负责完成：
设备 Token 的自动化获取和注册，降低接入门槛。
账号、标签与设备的绑定接口，以便开发者实现特定群组的消息推送，丰富推送方式。
点击量上报，统计消息被用户点击的次数。

移动推送 macOS 与 iOS 的差异

功能差异

注意：

以下特性由于苹果官方不支持，因此 macOS SDK 未提供。

功能描述	iOS	macOS	说明
通知扩展插件	✓	×	macOS 不支持通知扩展插件，不支持富媒体通知，不支持离线抵达统

			计
自定义通知声音	✓	×	macOS 不支持自定义通知声音
静默消息	✓	×	macOS 不支持静默消息
通知分组	✓	×	macOS 不支持通知分组

使用 Mac Catalyst 构建的 App 推荐使用移动推送 iOS SDK

Big Sur 系统（11.3及以下）开发环境无法获取 DeviceToken

此为苹果官方 Bug，已在11.4修复。

SDK 集成

最近更新时间：2024-01-16 17:42:20

操作场景

本文档提供关于 SDK 接入以及开启推送服务的示例代码。

SDK 组成

doc 文件夹：移动推送 macOS SDK 开发指南。
demo 文件夹：主要包含样例工程，移动推送 SDK。

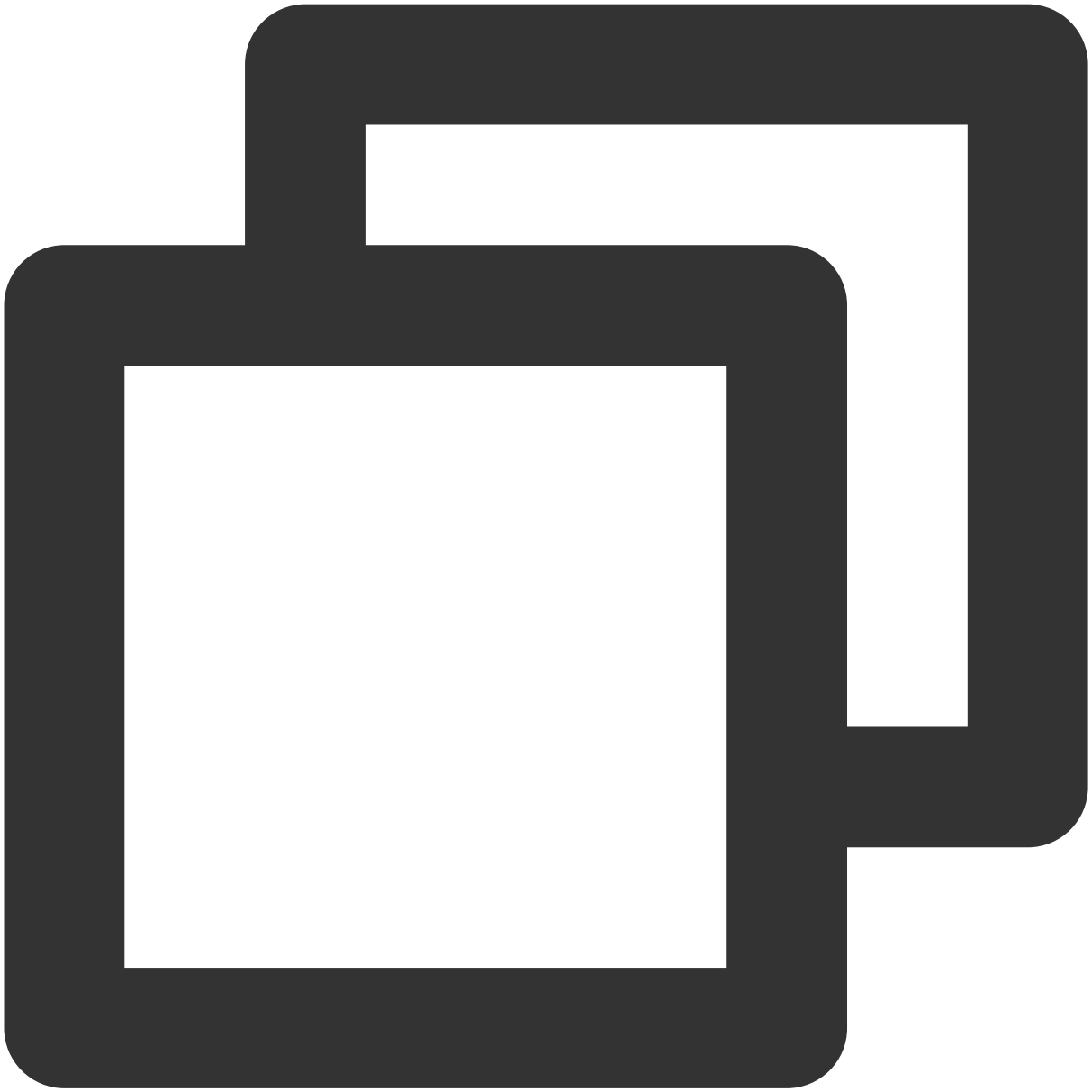
集成步骤

接入前准备

1. 登录 移动推送控制台，单击左侧菜单栏**产品管理**。
2. 进入产品管理页面，单击**新增产品**。
3. 进入新增产品页面，填写产品名称、产品详情，选择产品分类，单击**确定**，即可完成产品新增。
4. 产品创建完成后，选择左侧菜单**配置管理 > 基础配置**，在应用信息一栏中获取应用 `AccessID` 和 `AccessKEY` 。

导入 SDK（二选一）

方式一：Cocoapods 导入
通过 Cocoapods 下载地址：



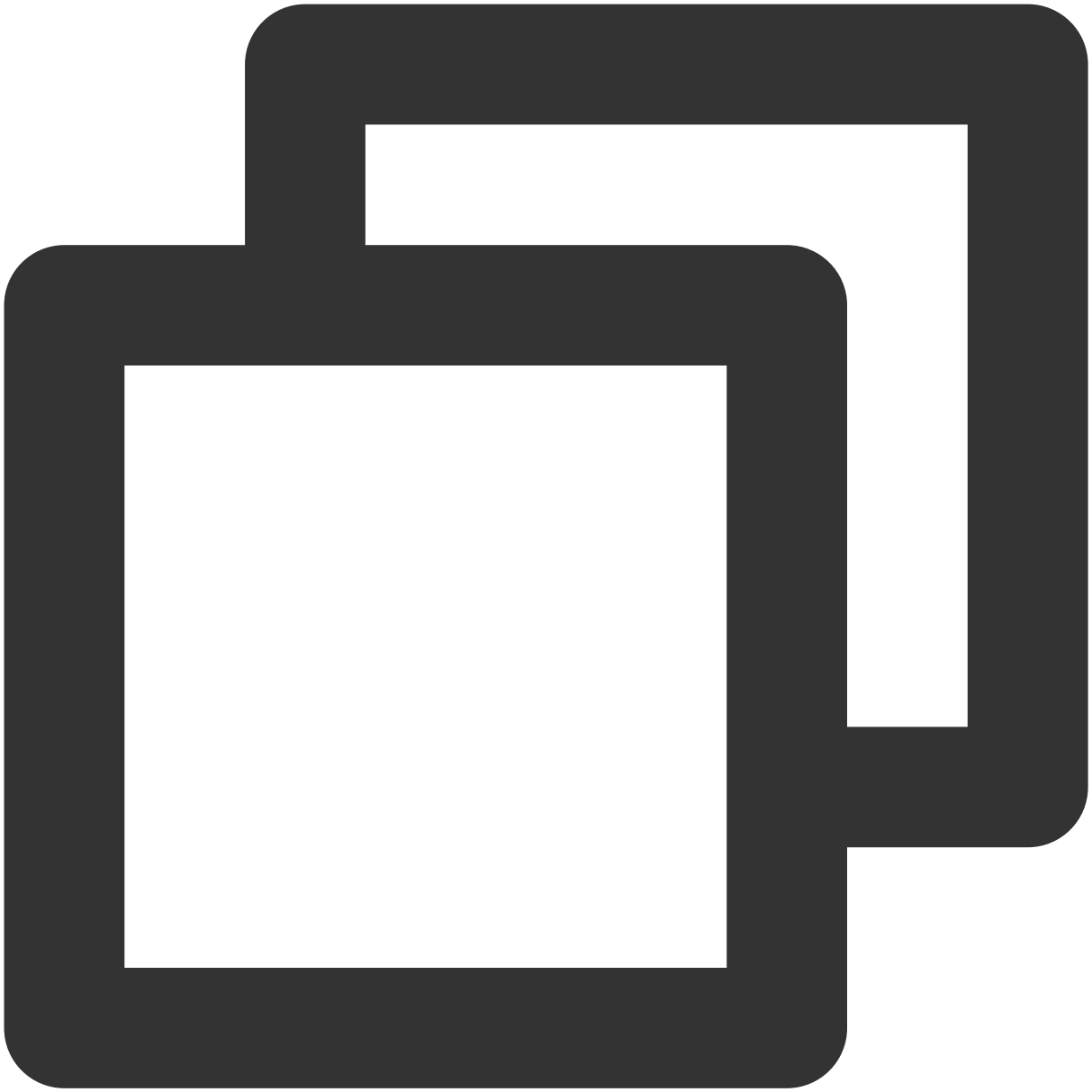
```
pod 'TPNS-macOS'
```

方式二：手动导入

进入控制台，单击左侧菜单栏【SDK 下载】，进入下载页面，选择 macOS 平台，在其操作栏下单击【下载】即可导入。

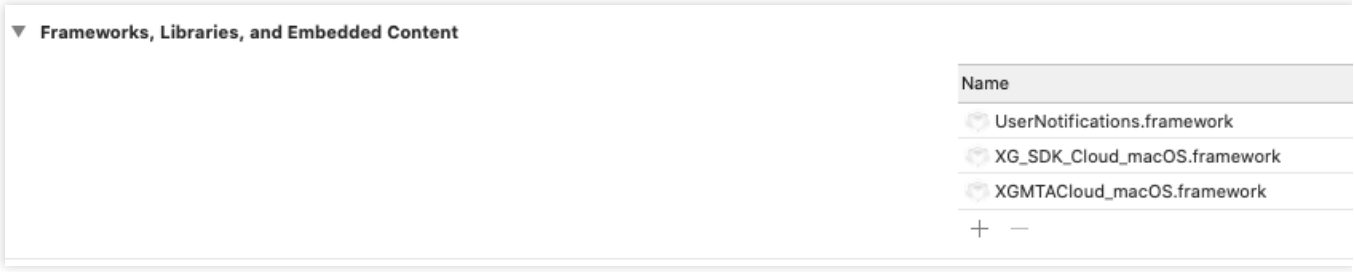
1.1 打开 demo 目录下的 XG-Demo-macOS 文件夹，将 XG_SDK_Cloud_macOS.framework 及 XGMTACloud_macOS.framework 添加到工程。

1.2 在 Build Phases 下添加以下 Framework：



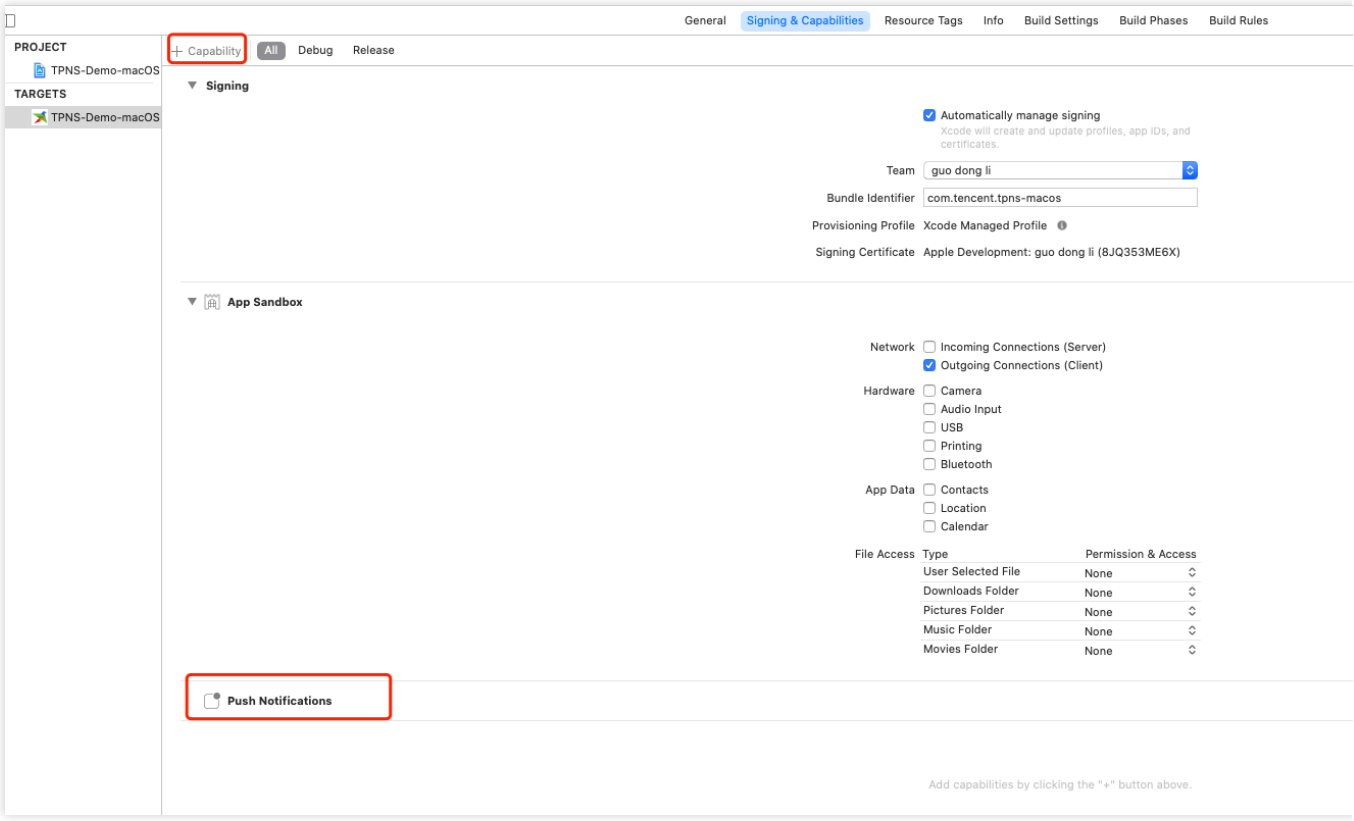
```
    * XG_SDK_Cloud_macOS.framework
* XGMTACloud_macOS.framework
  * UserNotifications.framework(10.14+)
```

1.3 添加完成以后， `TARGETS->General->Frameworks,Libraries,and Embedded Content` 选项下Embed选择Embed&Sign 如下图：

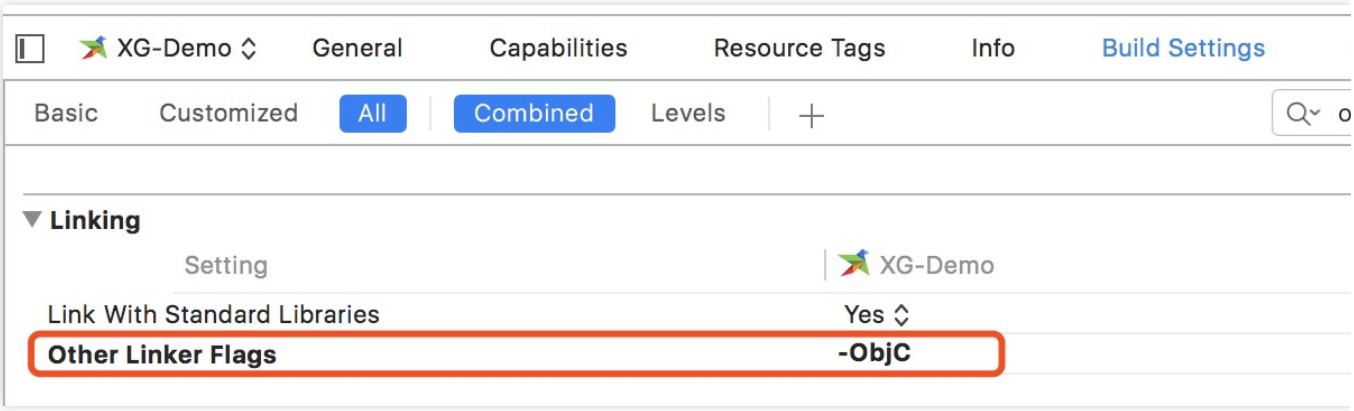


工程配置

1. 在工程配置中打开推送，如下图：



2. 在 `Build Settings->Other Linker Flags` 添加编译参数 `-ObjC` 。



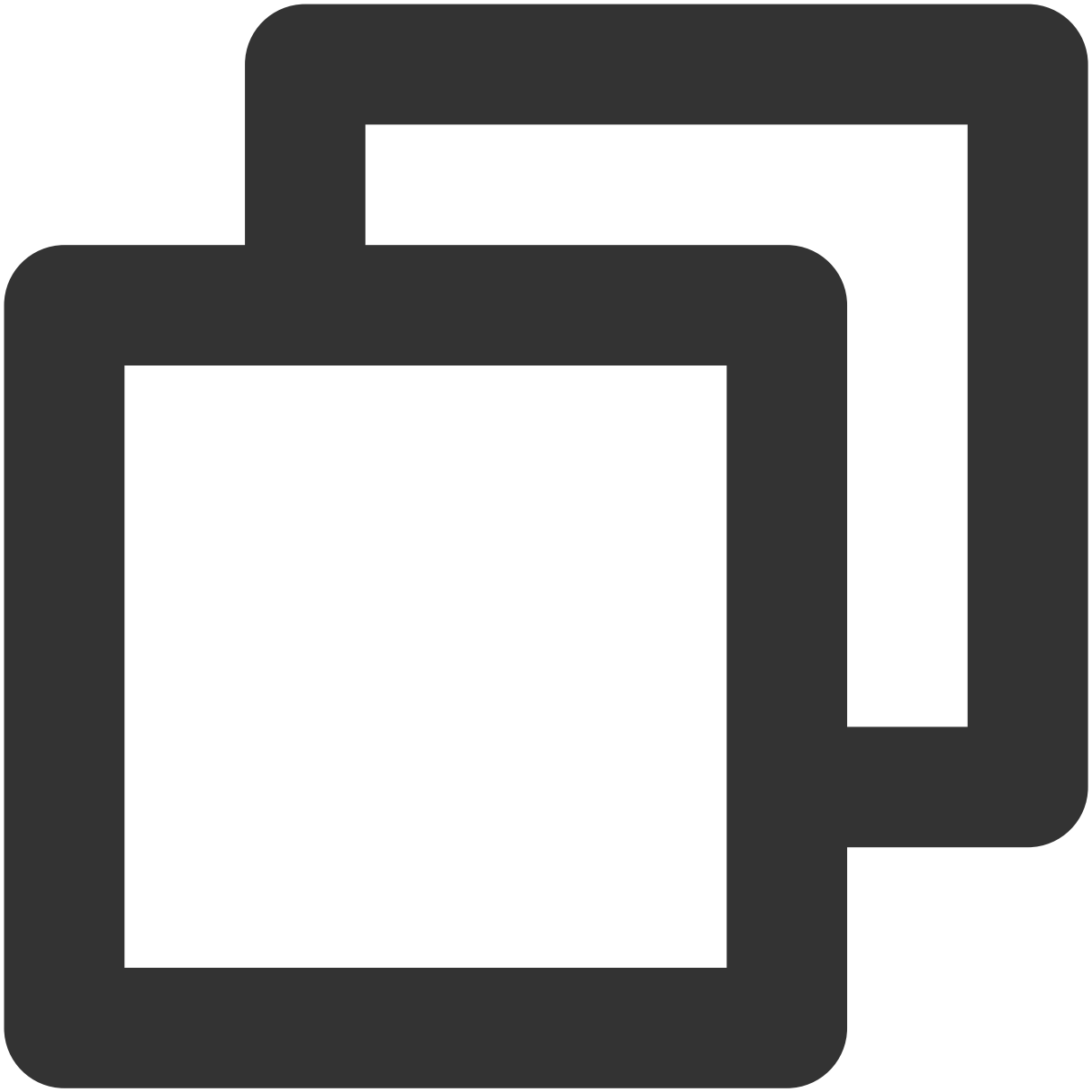
注意：

如 checkTargetOtherLinkFlagForObjc 报错，是因为 build setting 中，Other link flags 未添加 -ObjC。

接入样例

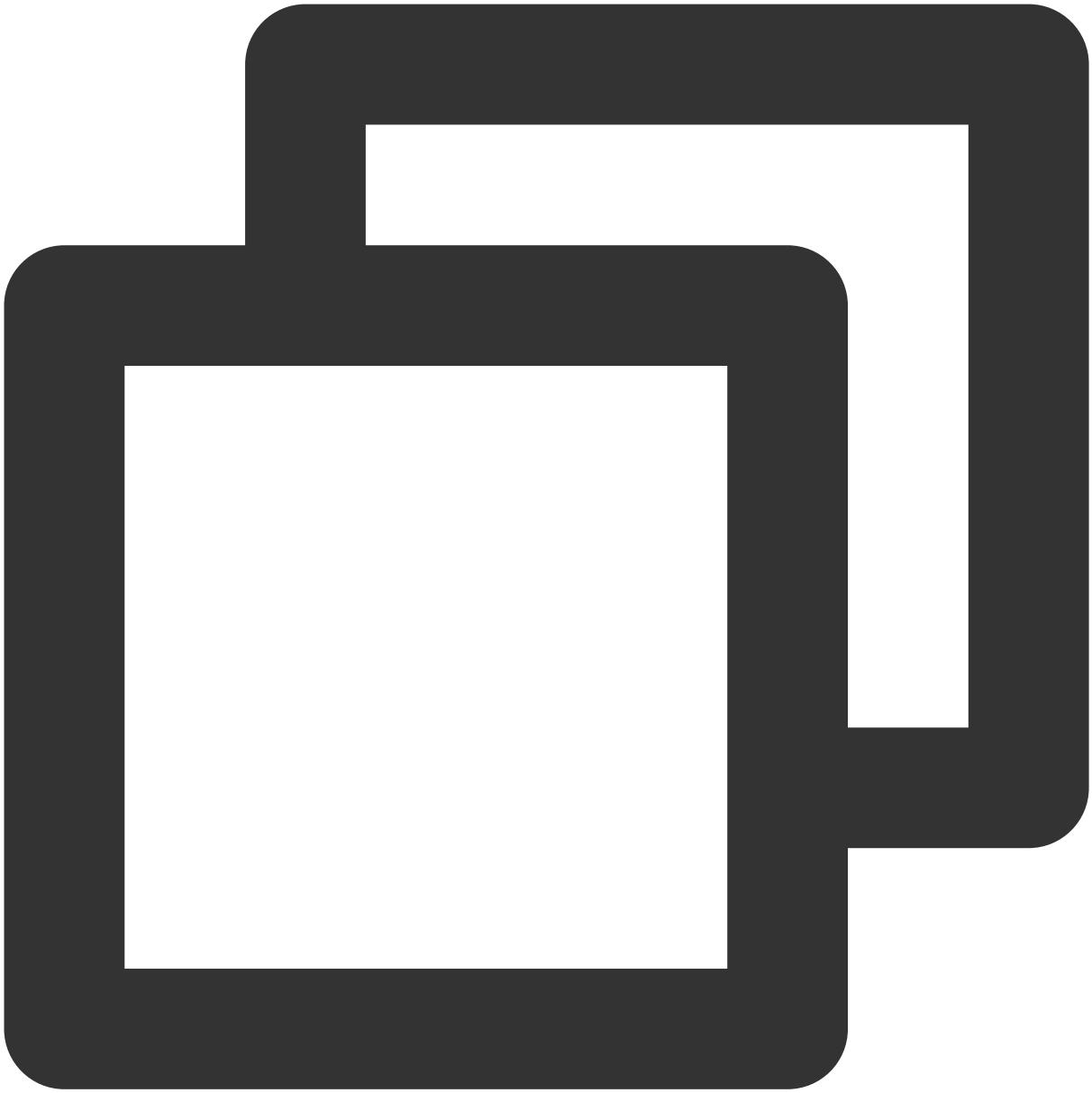
调用启动移动推送 的 API，并根据需要实现 XGPushDelegate 协议中的方法，开启推送服务。

1. 启动移动推送服务，以下是在 AppDelegate 中做演示：



```
  (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
  /// 打开 Debug 模式，即可在终端查看详细的移动推送 TPNS Debug 信息，方便定位问题。
//    [[XGPush defaultManager] setEnableDebug:YES];
  [XGPush defaultManager].launchOptions = [[aNotification userInfo] mutableCopy];
  [[XGPush defaultManager] startXGWithAccessID:TPNS_ACCESS_ID accessKey:TPNS_ACCESS
}
```

2. 在 `AppDelegate` 中选择实现 `XGPushDelegate` 协议中的方法：

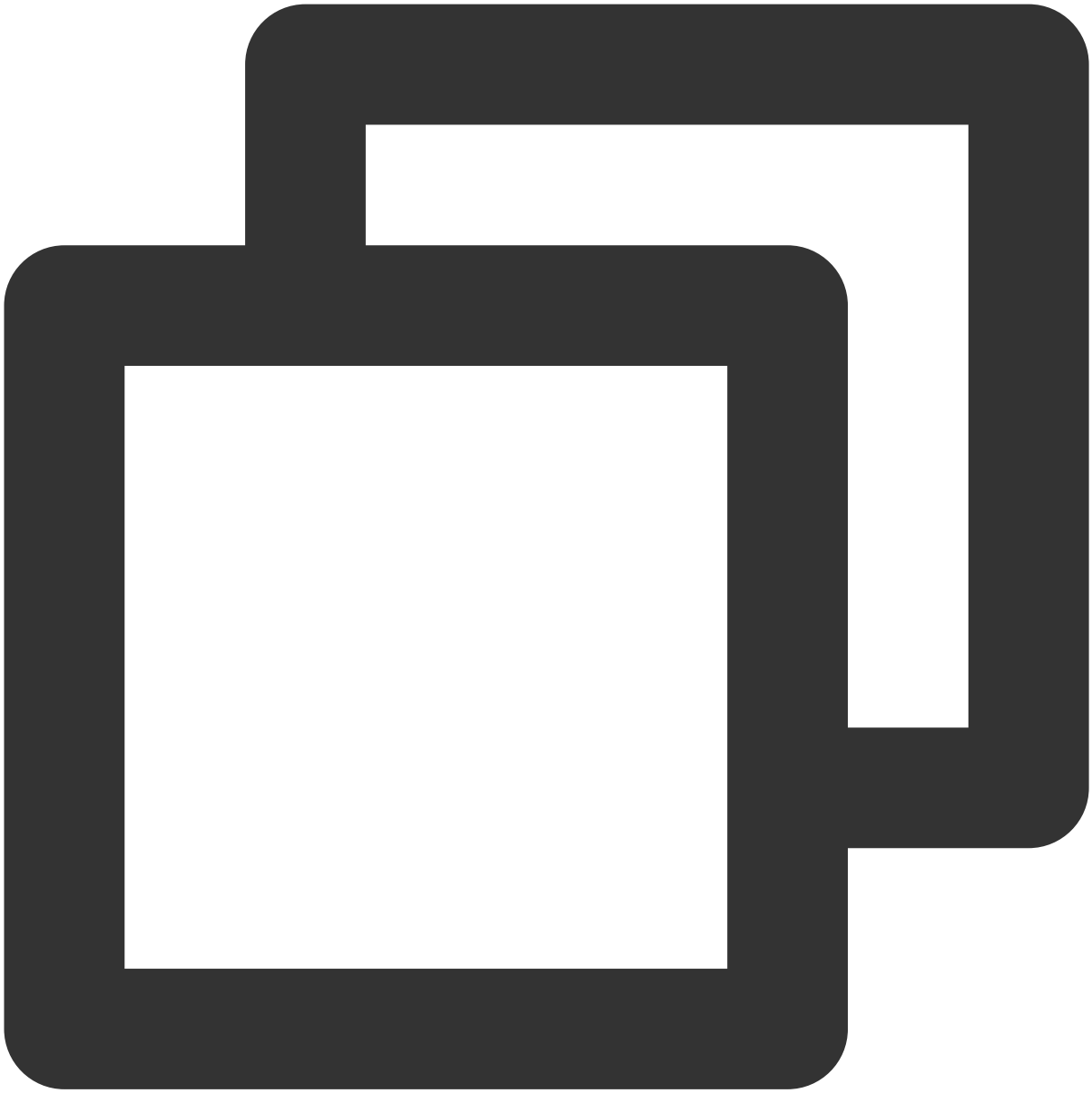


```
/// 注册推送服务成功回调
/// @param deviceToken APNs 生成的Device Token
/// @param xgToken TPNS 生成的 Token，推送消息时需要使用此值。TPNS 维护此值与APNs 的 Device
/// @param error 错误信息，若error为nil则注册推送服务成功
  (void)xgPushDidRegisteredDeviceToken:(NSString *)deviceToken xgToken:(NSString *)
 if (!error) {
   NSLog(@"%s, register success, deviceToken:%@, xgToken:%@", FUNCTION, deviceToken
 } else {
   NSLog(@"%s, register failed:%@, deviceToken:%@, xgToken:%@", FUNCTION,error.desc
}
}
```

```
// 统一收到通知消息的回调
/// @param notification 消息对象
/// @param completionHandler 完成回调
/// 区分消息类型说明：xg字段里的msgtype为1则代表通知消息msgtype为2则代表静默消息
/// notification消息对象说明：有2种类型NSDictionary和UNNotification具体解析参考示例代码
(void)xgPushDidReceiveRemoteNotification:(id)notification withCompletionHandler:(vo
NSLog(@"[TPNS Demo] receive notification: %@", notification);
}
/// 统一点击回调
/// @param response 如果iOS 10+/macOS 10.14+则为UNNotificationResponse，低于目标版本则为
/// 区分消息类型说明：xg字段里的msgtype为1则代表通知消息,msgtype为9则代表本地通知
(void)xgPushDidReceiveNotificationResponse:(nonnull id)response withCompletionHandl
if ([response isKindOfClass:[UNNotificationResponse class]]) {
NSLog(@"[TPNS Demo] click notification: %@", ((UNNotificationResponse *)response).n
} else if ([response isKindOfClass:[NSDictionary class]]) {
NSLog(@"[TPNS Demo] click notification: %@", response);
    }
  completionHandler();
}
```

观察日志

如果 Xcode 控制台显示如下相似日志，表明客户端已经正确集成 SDK。



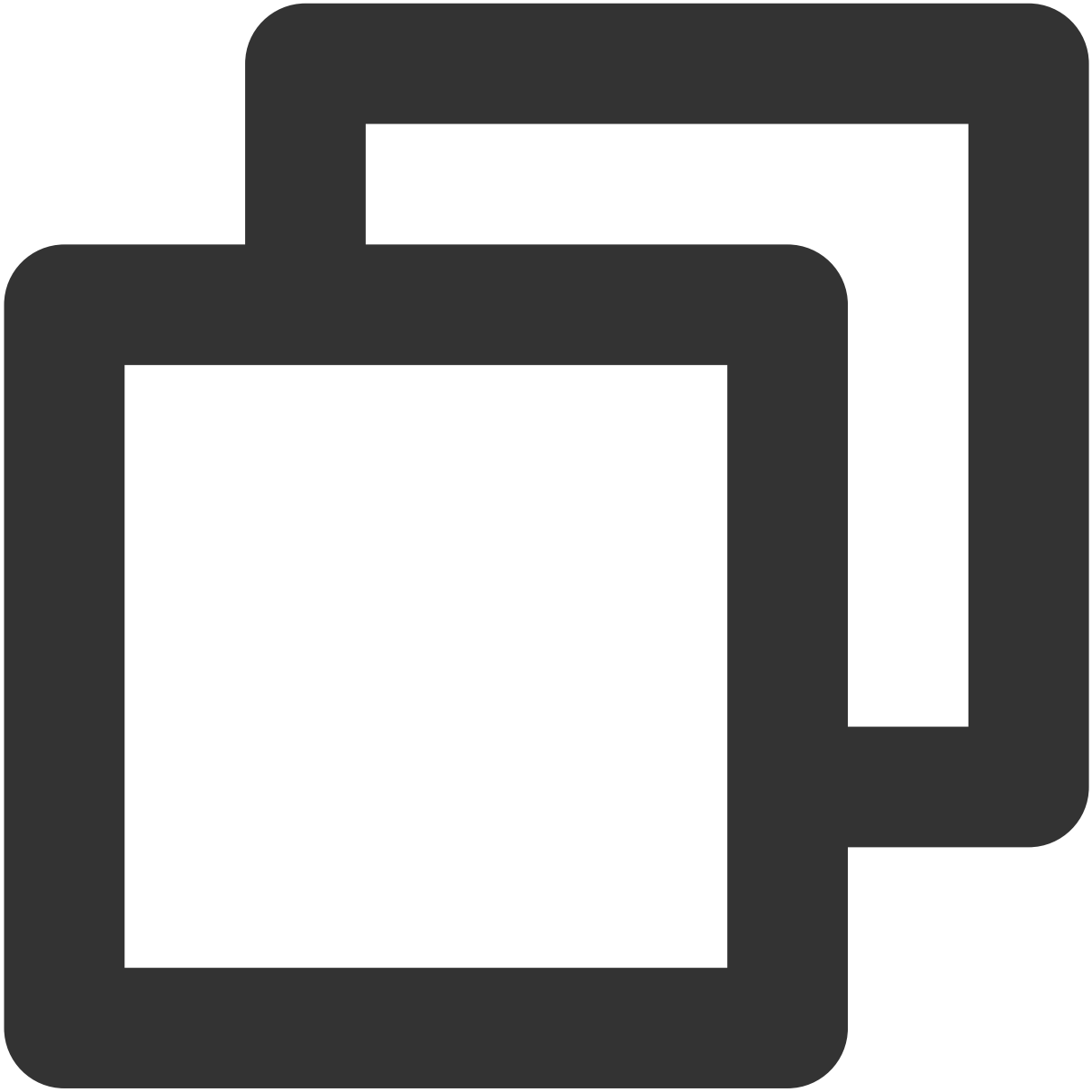
```
javascript
[TPNS]  Current device token is 2117b45c7e32bcdae2939f******57e420a376bdd44cf6f5861
[TPNS]  Current TPNS token is 0304b8f5d4e*****0af06b37d8b850d95606
[TPNS]  The server responds correctly, registering device successfully
```

集成建议

获取 Token （非必选）

建议您完成 SDK 集成后，在 App 的【关于】、【意见反馈】等比较不常用的 UI 中，通过手势或者其他方式显示 Token，该操作便于我们后续进行问题排查。

示例代码



```
objective-c
//获取 TPNS 生成的 Token
[[XGPushTokenManager defaultTokenManager] xgTokenString];
//获取 APNs 生成的 DeviceToken
[[XGPushTokenManager defaultTokenManager] deviceTokenString];
```

接口文档

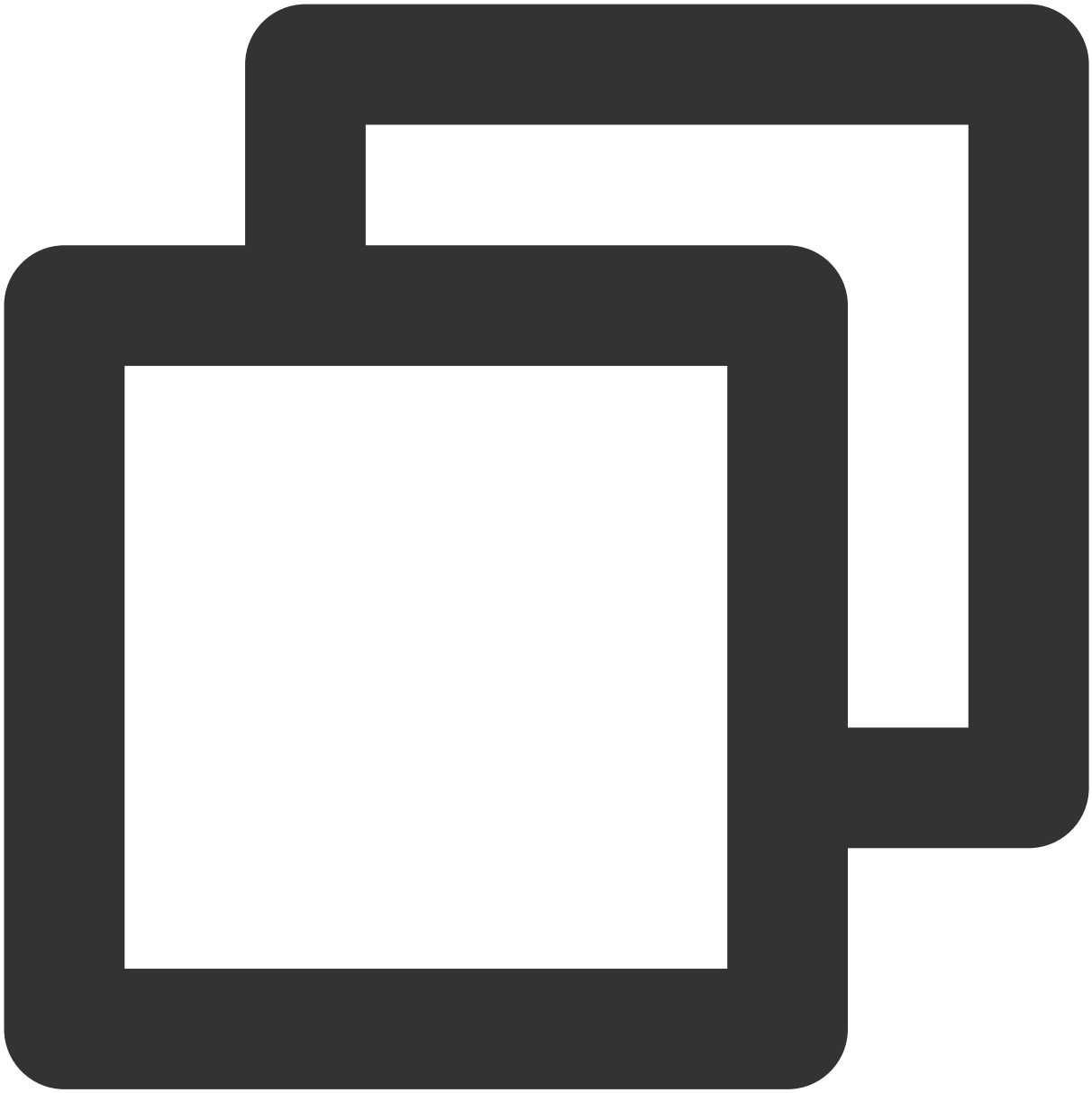
最近更新时间：2024-01-16 17:42:20

启动腾讯移动推送服务

以下为设备注册相关接口方法，若需了解调用时机及调用原理，可查看 设备注册流程。

接口说明

通过使用在腾讯移动推送官网注册的应用信息，启动腾讯移动推送服务。



```
- (void)startXGWithAccessID:(uint32_t)accessID accessKey:(nonnull NSString *)access
```

参数说明

accessID：通过前台申请的 AccessID。

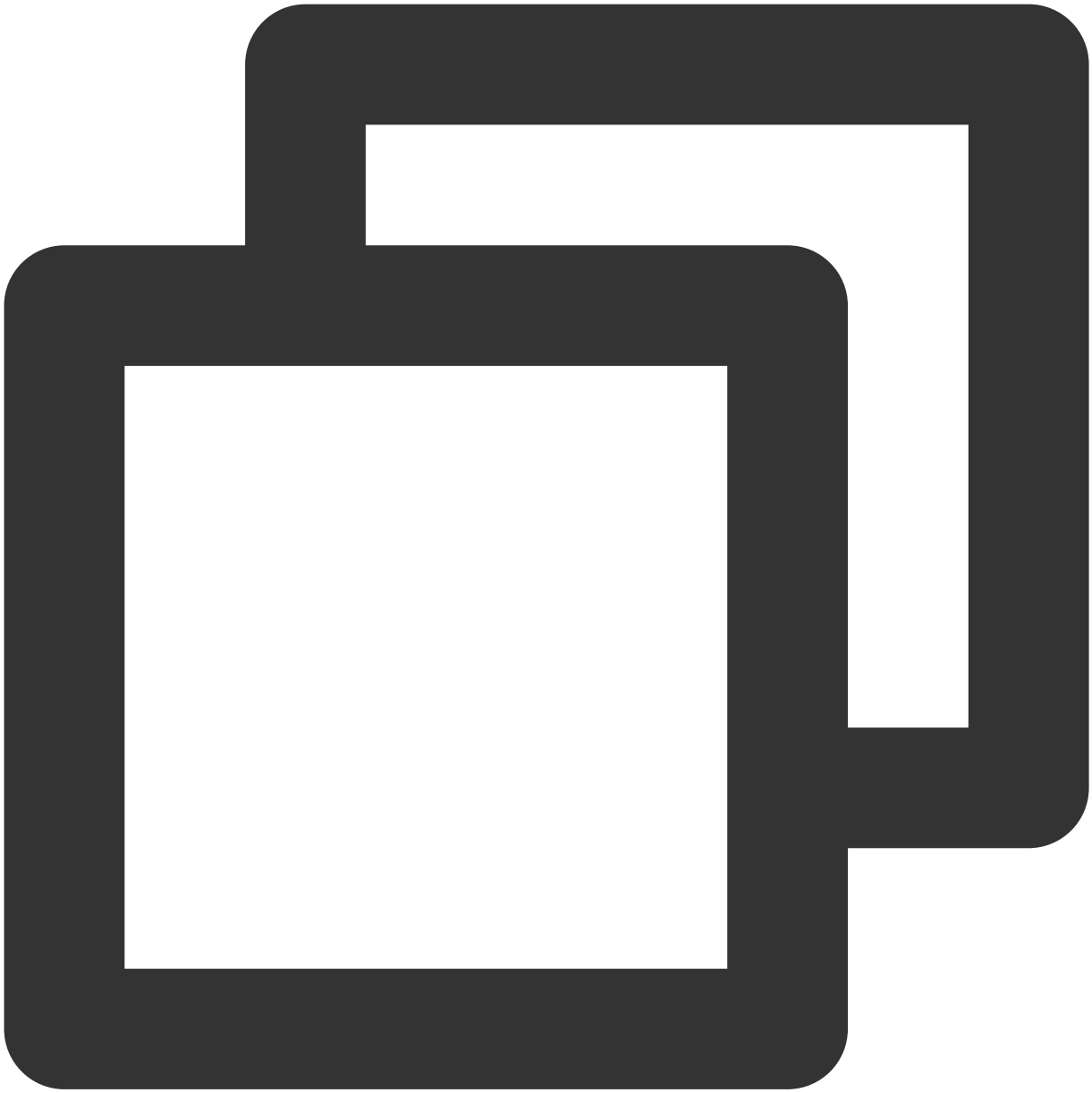
accessKey：通过前台申请的 AccessKey。

Delegate：回调对象。

注意：

接口所需参数必须要正确填写，否则腾讯移动推送服务将不能正确为应用推送消息。

示例代码



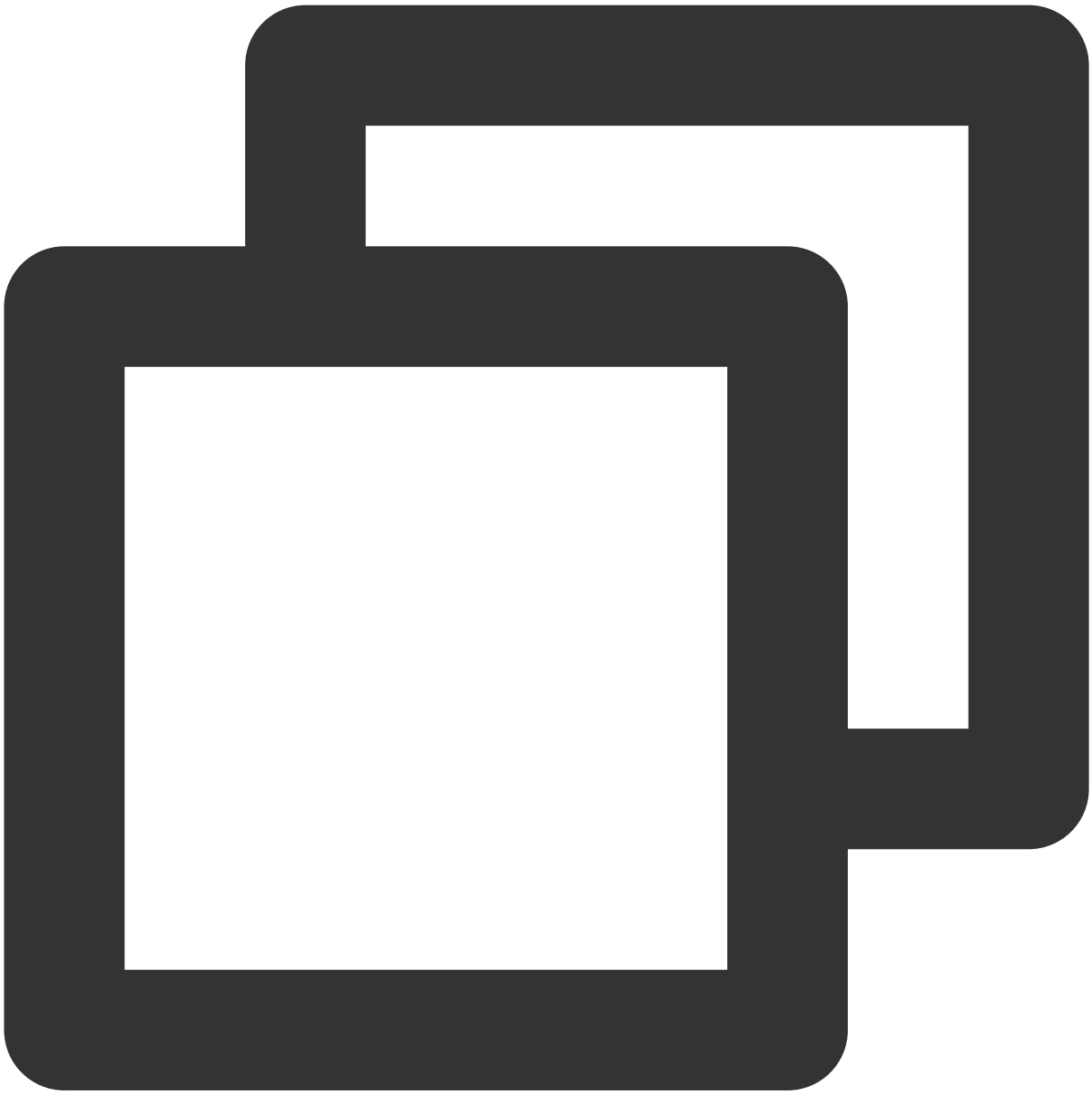
```
[[XGPush defaultManager] startXGWithAccessID:<your AccessID> accessKey:<your Acces
```

终止腾讯移动推送服务

以下为设备注册相关接口方法，若需了解调用时机及调用原理，可查看 设备反注册流程。

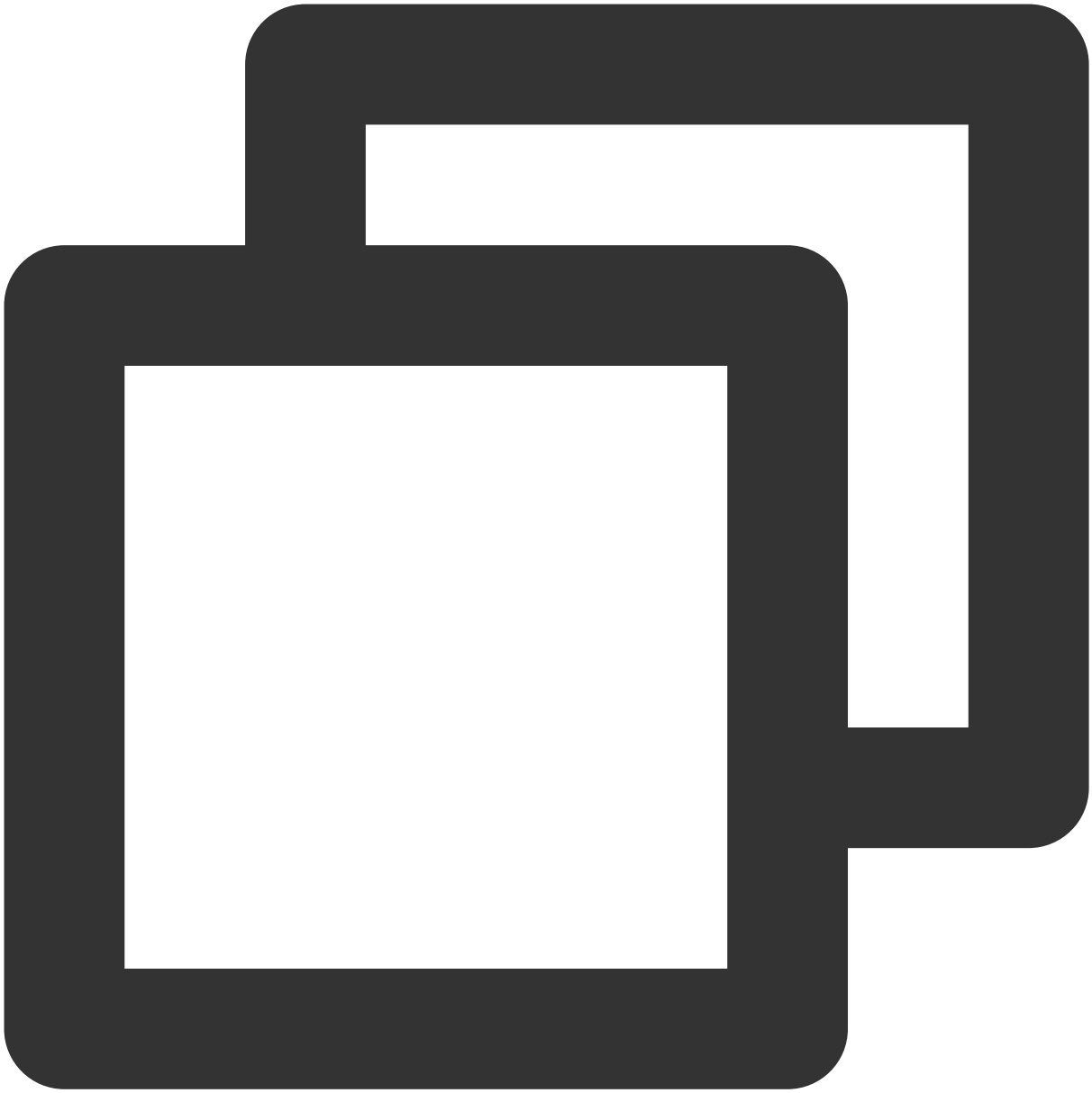
接口说明

终止腾讯移动推送服务后，将无法通过腾讯移动推送服务向设备推送消息，如再次需要接收腾讯移动推送服务的消息推送，则必须再次调用 `startXGWithAccessID:accessKey:delegate:` 方法重启腾讯移动推送服务。



```
- (void)stopXGNotification;
```

示例代码



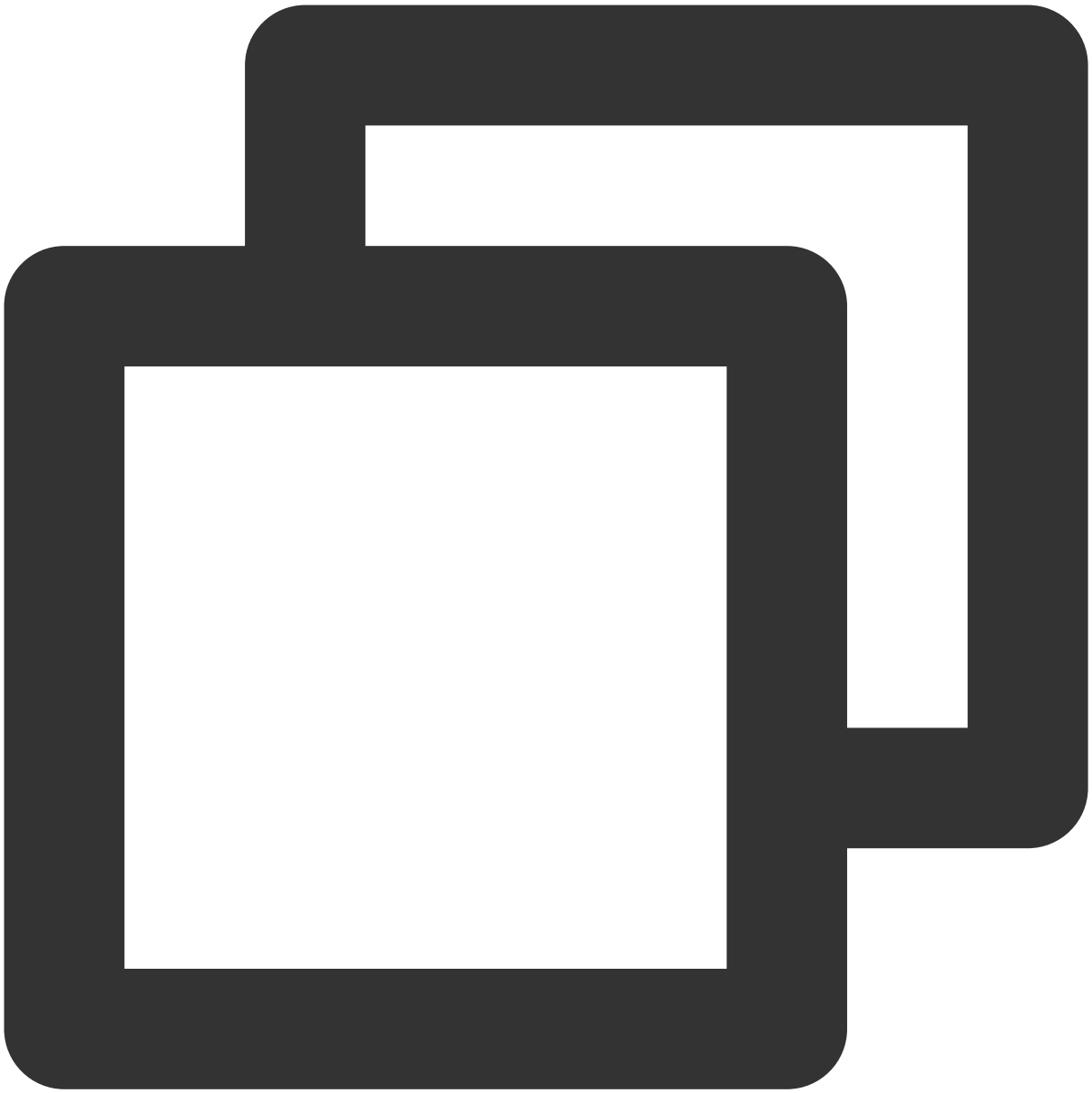
```
[[XGPush defaultManager] stopXGNotification];
```

移动推送 Token 及注册结果

查询移动推送 Token

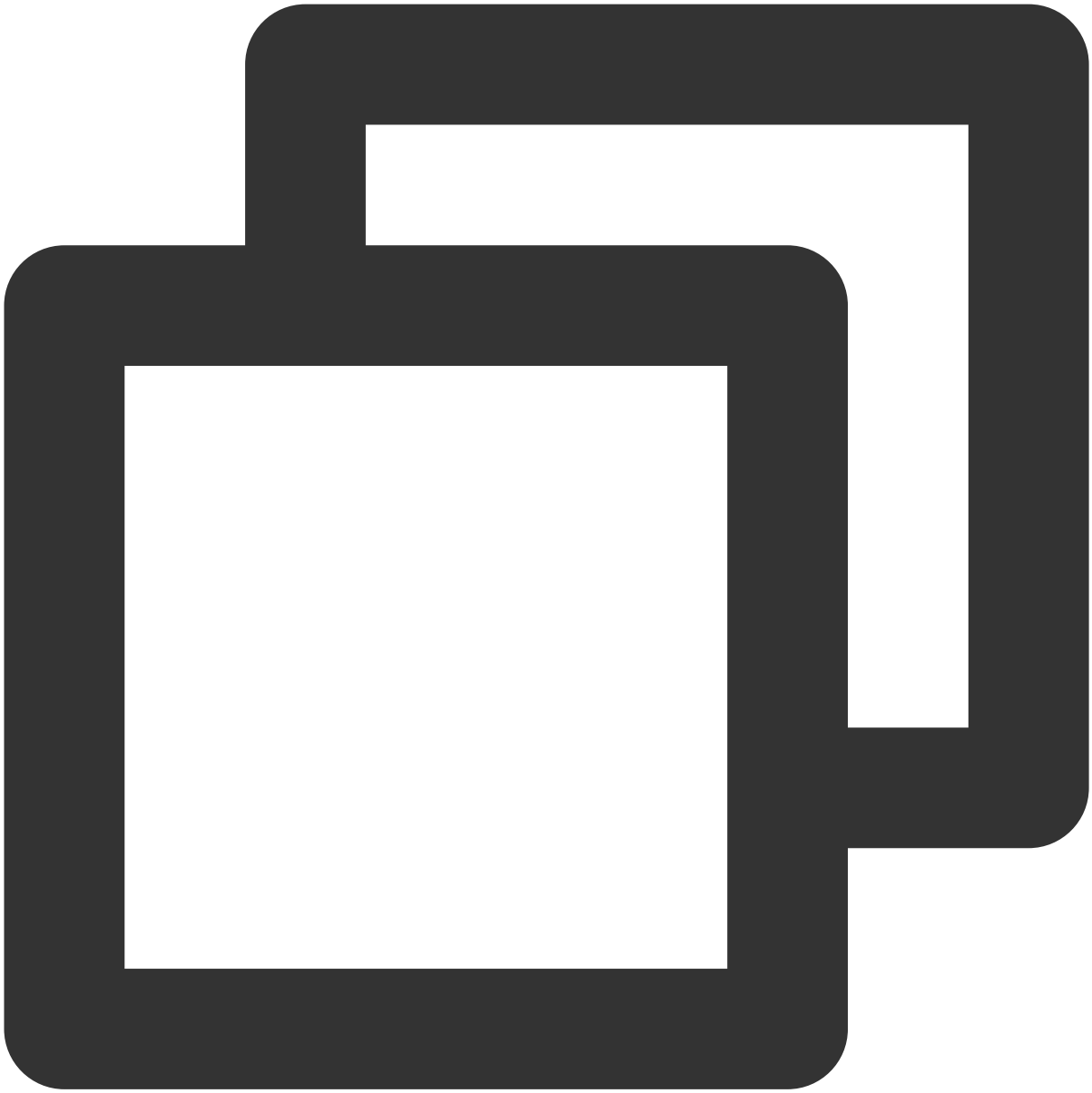
接口说明

查询当前应用从腾讯移动推送服务器生成的 Token 字符串。



```
@property (copy, nonatomic, nullable, readonly) NSString *xgTokenString;
```

示例代码

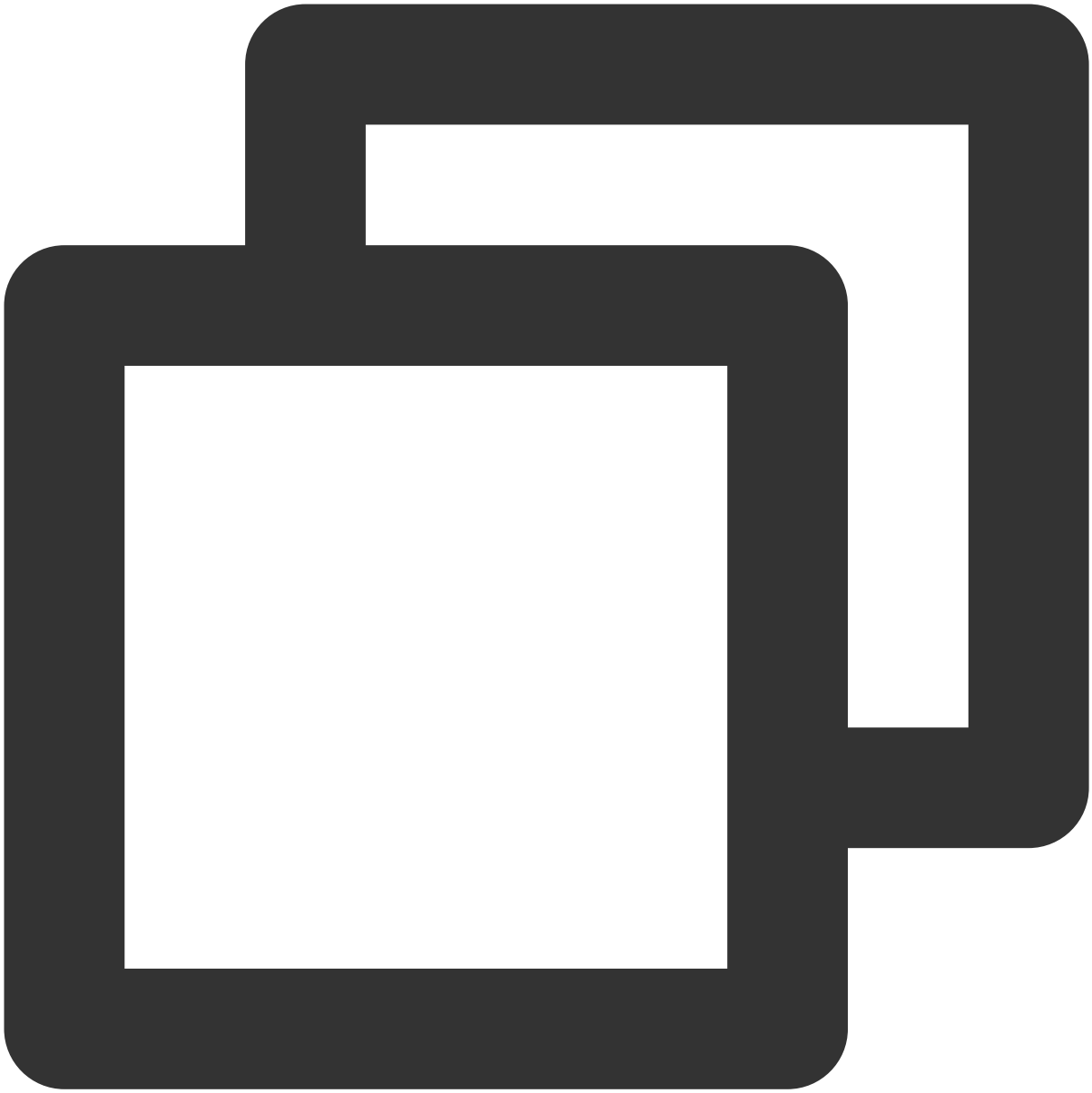


```
NSString *token = [[XGPushTokenManager defaultTokenManager] xgTokenString];
```

注册结果回调

接口说明

SDK 启动之后，通过此方法回调来返回注册结果及 Token。



```
- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nu
```

返回参数说明

deviceToken：APNs 生成的 Device Token。

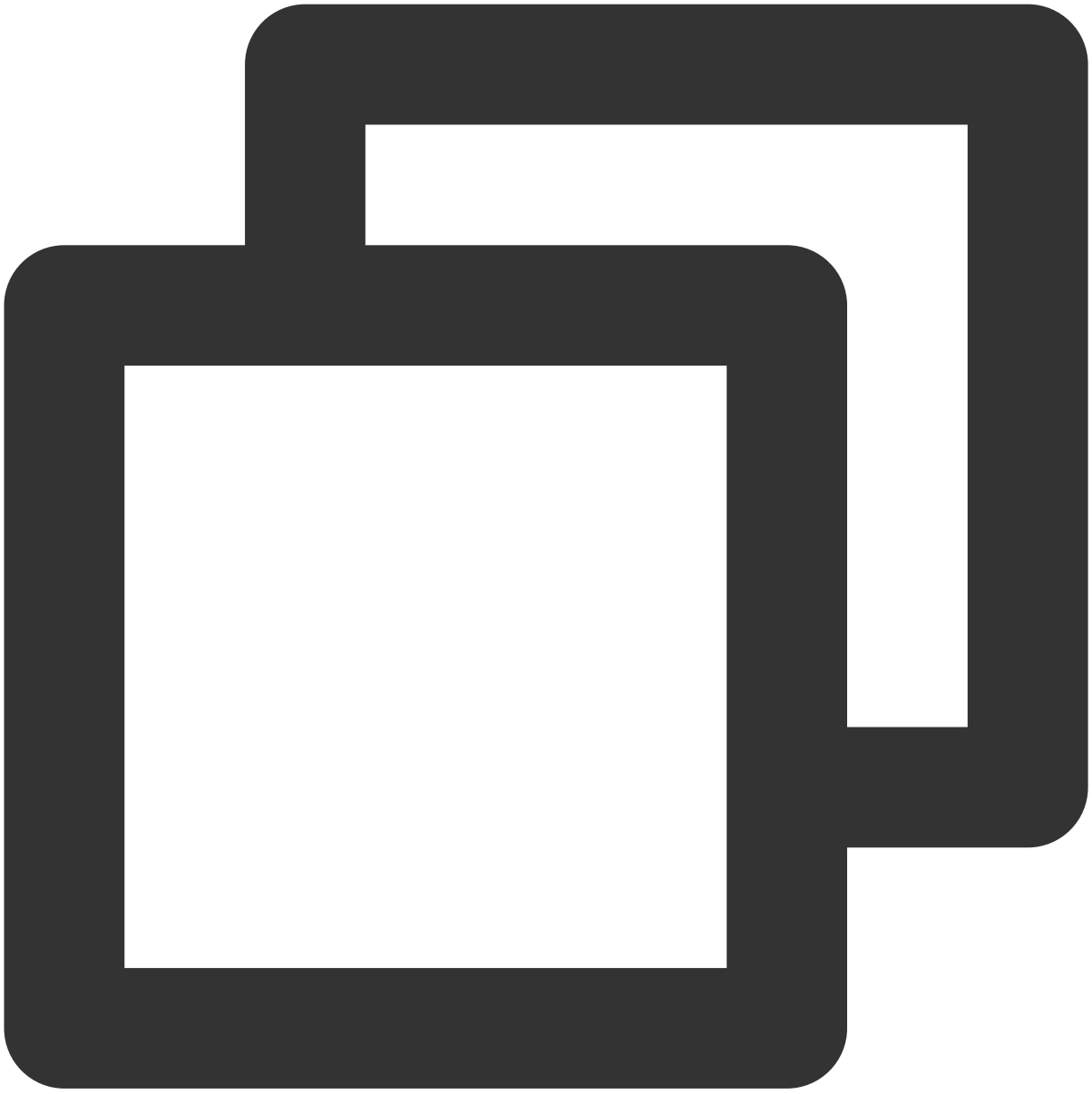
xgToken：移动推送生成的 Token，推送消息时需要使用此值。移动推送维护此值与 APNs 的 Device Token 的映射关系。

error：错误信息，若 error 为 nil，则注册推送服务成功。

注册失败回调

接口说明

当注册推送服务失败会走此回调。

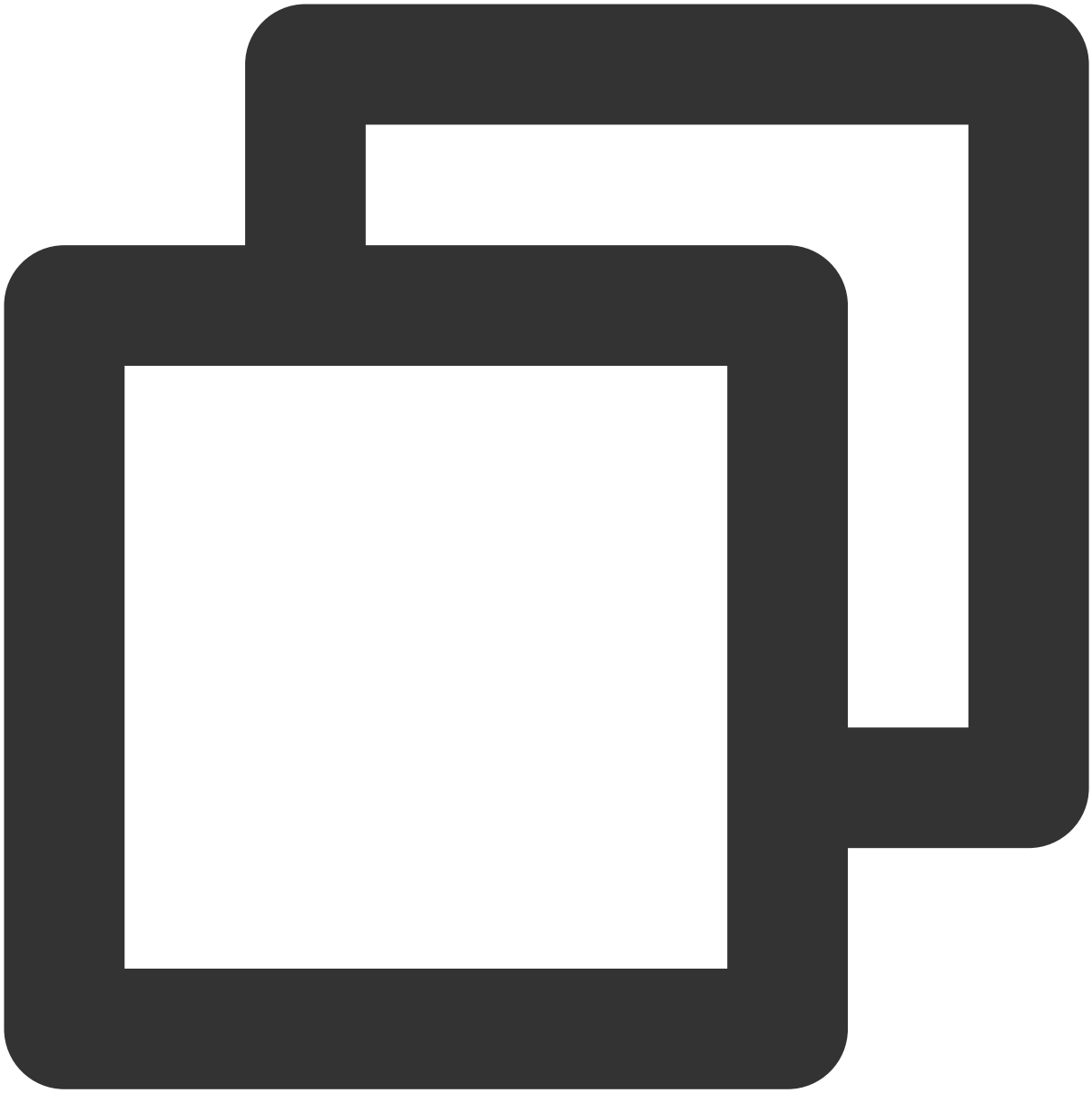


```
- (void)xgPushDidFailToRegisterDeviceTokenWithError:(nullable NSError *)error
```

通知授权弹窗的回调

接口说明

通知弹窗授权的结果会走此回调。



```
- (void)xgPushDidRequestNotificationPermission:(bool)isEnable error:(nullable NSErr
```

返回参数说明

isEnable：是否同意授权。

error：错误信息，若 error 为 nil，则获取弹窗结果成功。

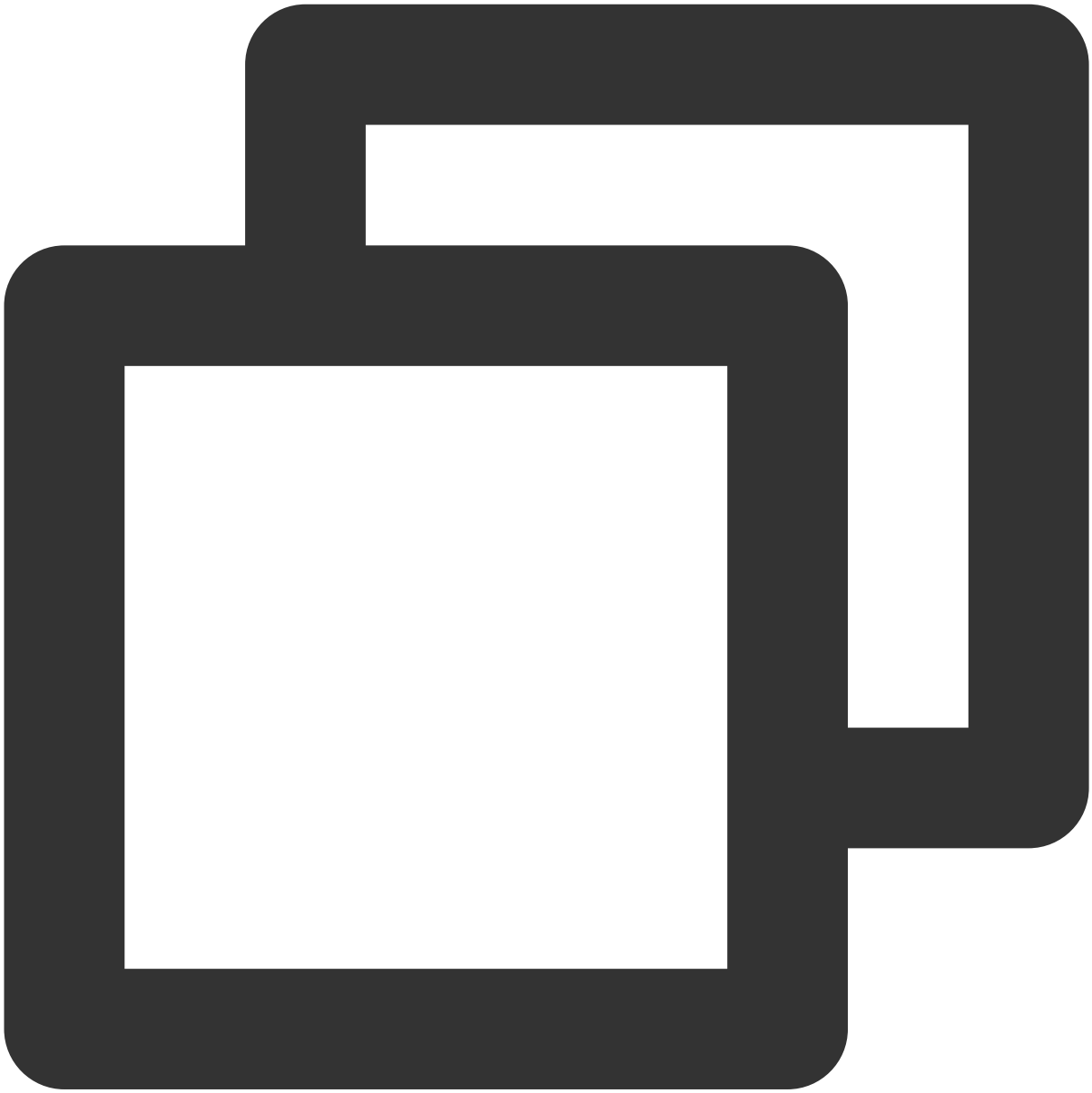
账号功能

以下为账号相关接口方法，若需了解调用时机及调用原理，可查看 账号相关流程。

添加账号

接口说明

若原来没有该类型账号，则添加；若原来有，则覆盖。



```
- (void)upsertAccountsByDict:(nonnull NSDictionary<NSNumber *, NSString *> *)accoun
```

说明：

此接口应该在 xgPushDidRegisteredDeviceToken:error: 返回正确之后被调用。

参数说明

accountsDict：账号字典。

说明：

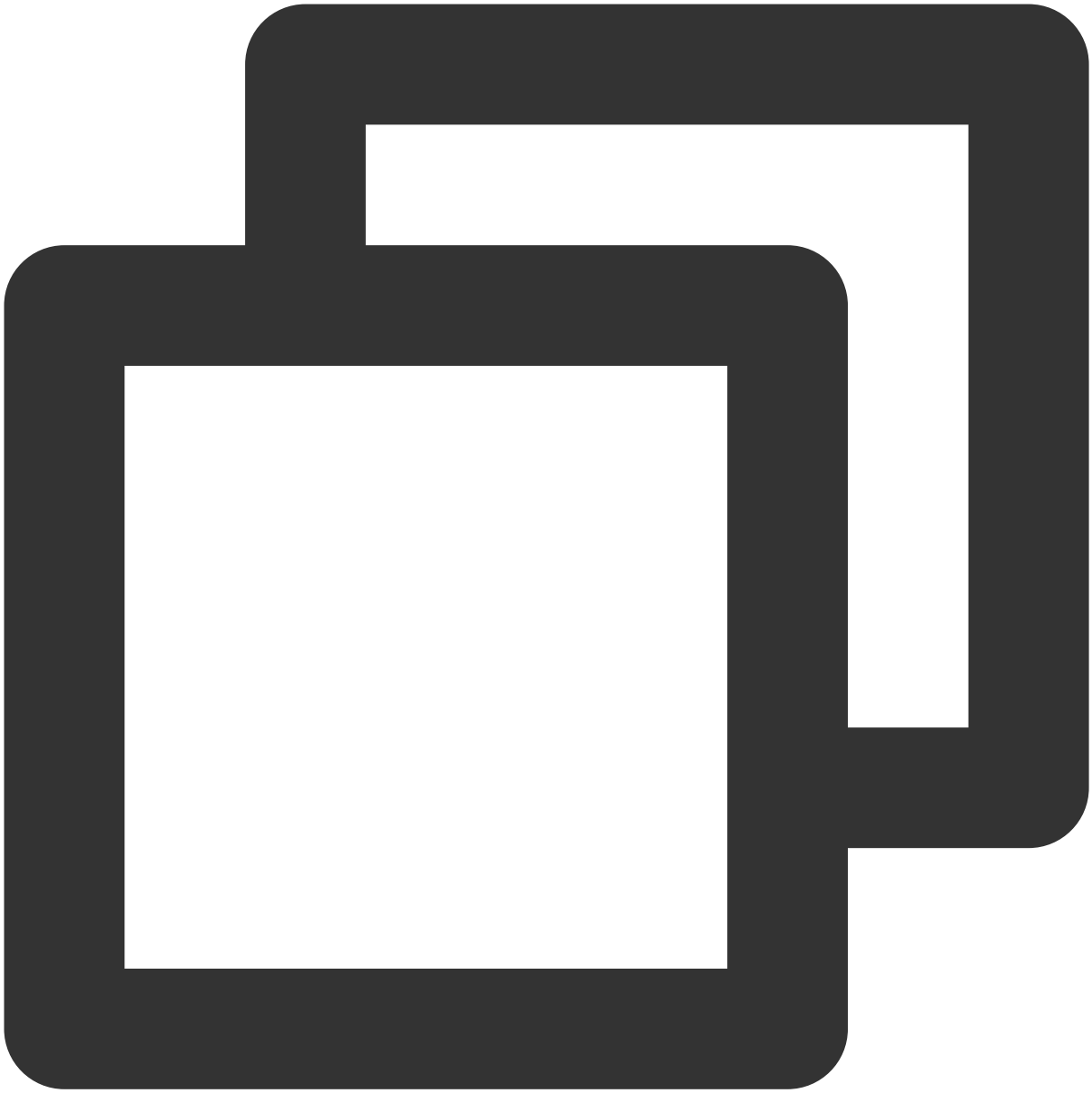
账号类型和账号名称一起作为联合主键。

需要使用字典类型，key 为账号类型，value 为账号，示例：@{@(accountType):@"account"}。

Objective-C的写法 : @{@(0):@"account0",@(1):@"account1"}；Swift的写法：[NSNumber(0):@"account0",NSNumber(1):@"account1"]。

更多 accountType 请参照 SDK Demo 包内的 XGPushTokenAccountType 枚举或 账号类型取值表。

示例代码

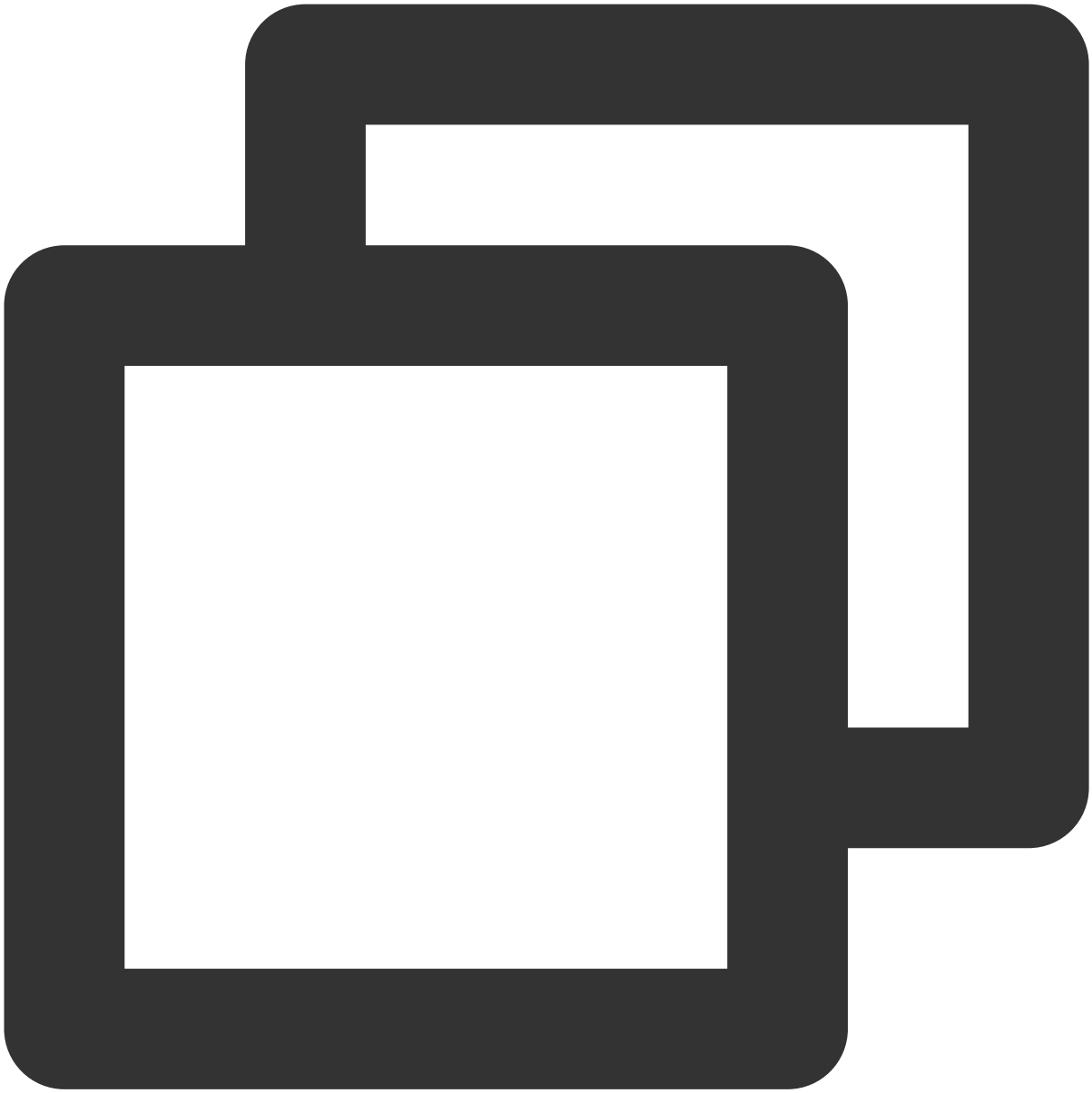


```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;
NSString *account = @"account";
[[XGPushTokenManager defaultTokenManager] upsertAccountsByDict:@{ @(accountType):ac
```

添加手机号

接口说明

添加或更新用户手机号，等于调用 `upsertAccountsByDict:@{@(1002):@"具体手机号"}` 。

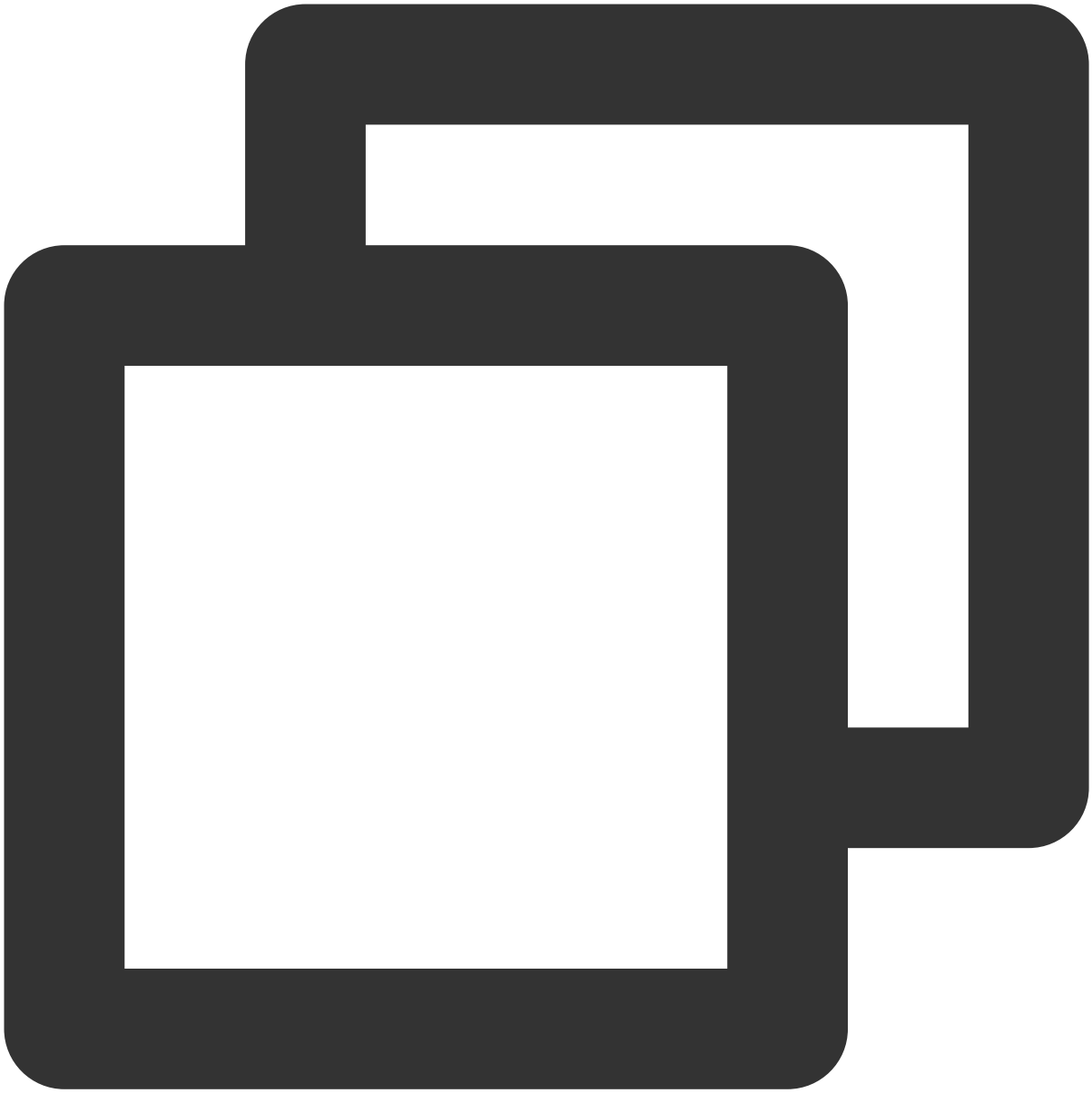


```
- (void)upsertPhoneNumber:(nonnull NSString *)phoneNumber;
```

参数说明

phoneNumber：E.164标准，格式为+[国家或地区码][手机号],例如+8613711112222。SDK内部加密传输。

示例代码



```
[[XGPushTokenManager defaultTokenManager] upsertPhoneNumber:@"13712345678"];;
```

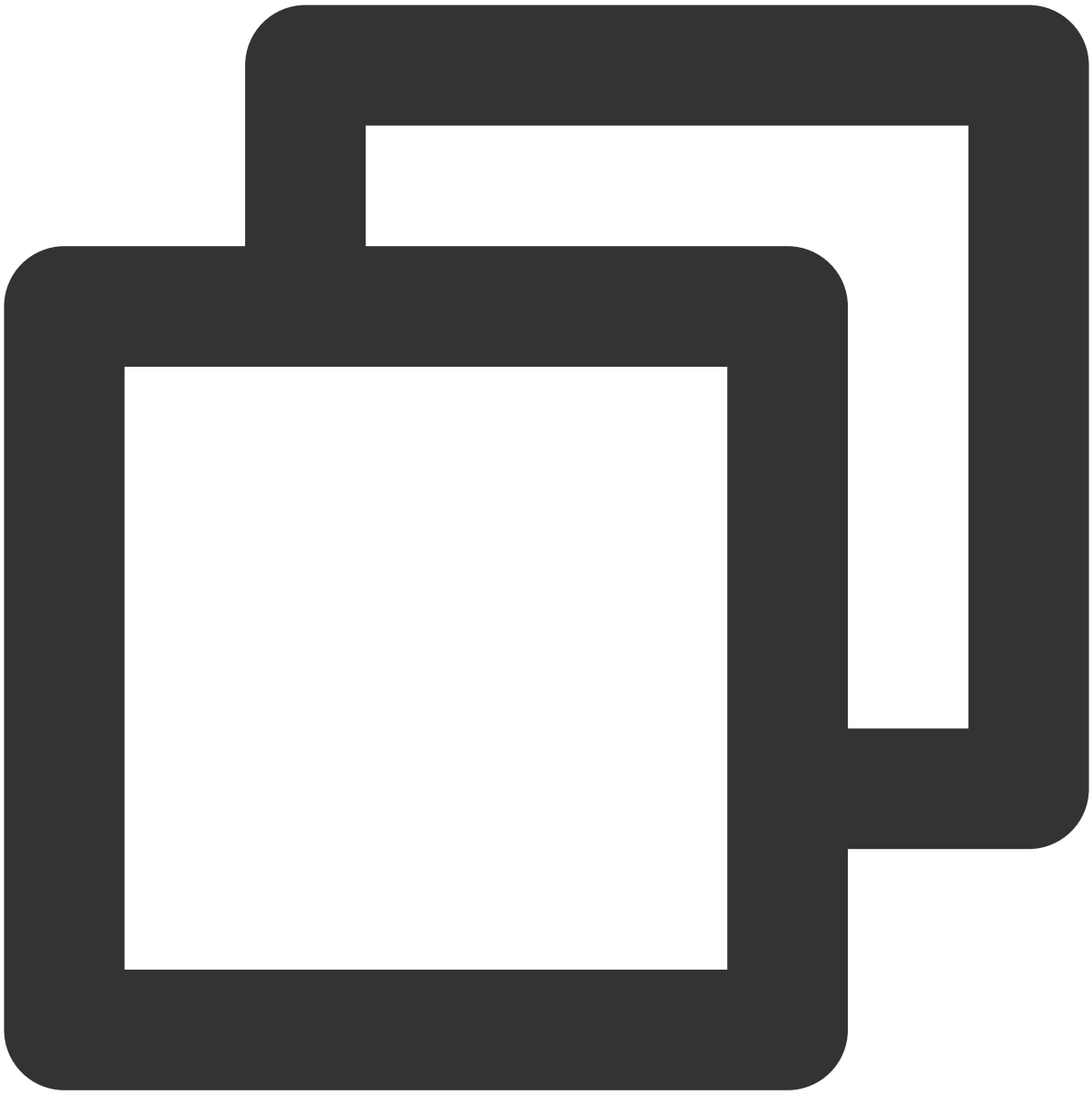
注意：

1. 此接口应该在xgPushDidRegisteredDeviceToken:error:返回正确之后被调用
2. 如需要删除手机号，调用 `delAccountsByKeys:[[NSSet alloc] initWithObjects:@(1002), nil]`

删除账号

接口说明

接口说明：删除指定账号类型下的所有账号。



```
- (void)delAccountsByKeys:(nonnull NSSet<NSNumber *> *)accountsKeys;
```

说明：

此接口应该在 xgPushDidRegisteredDeviceToken:error: 返回正确之后被调用。

参数说明

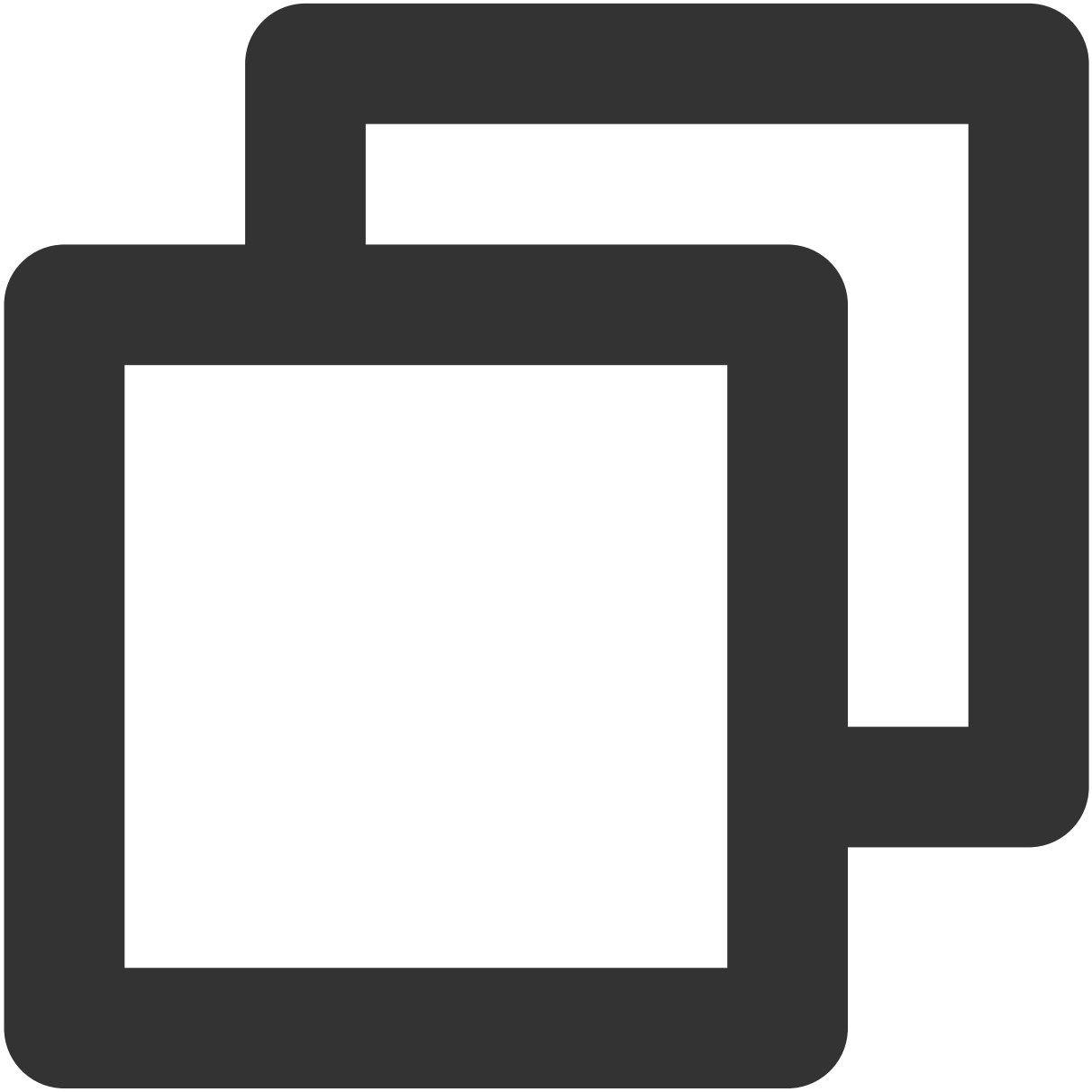
accountsKeys：账号类型组成的集合。

说明：

使用集合且 key 是固定要求。

更多 accountType 请参照 SDK 包内 XGPush.h 文件中的 XGPushTokenAccountType 枚举。

示例代码

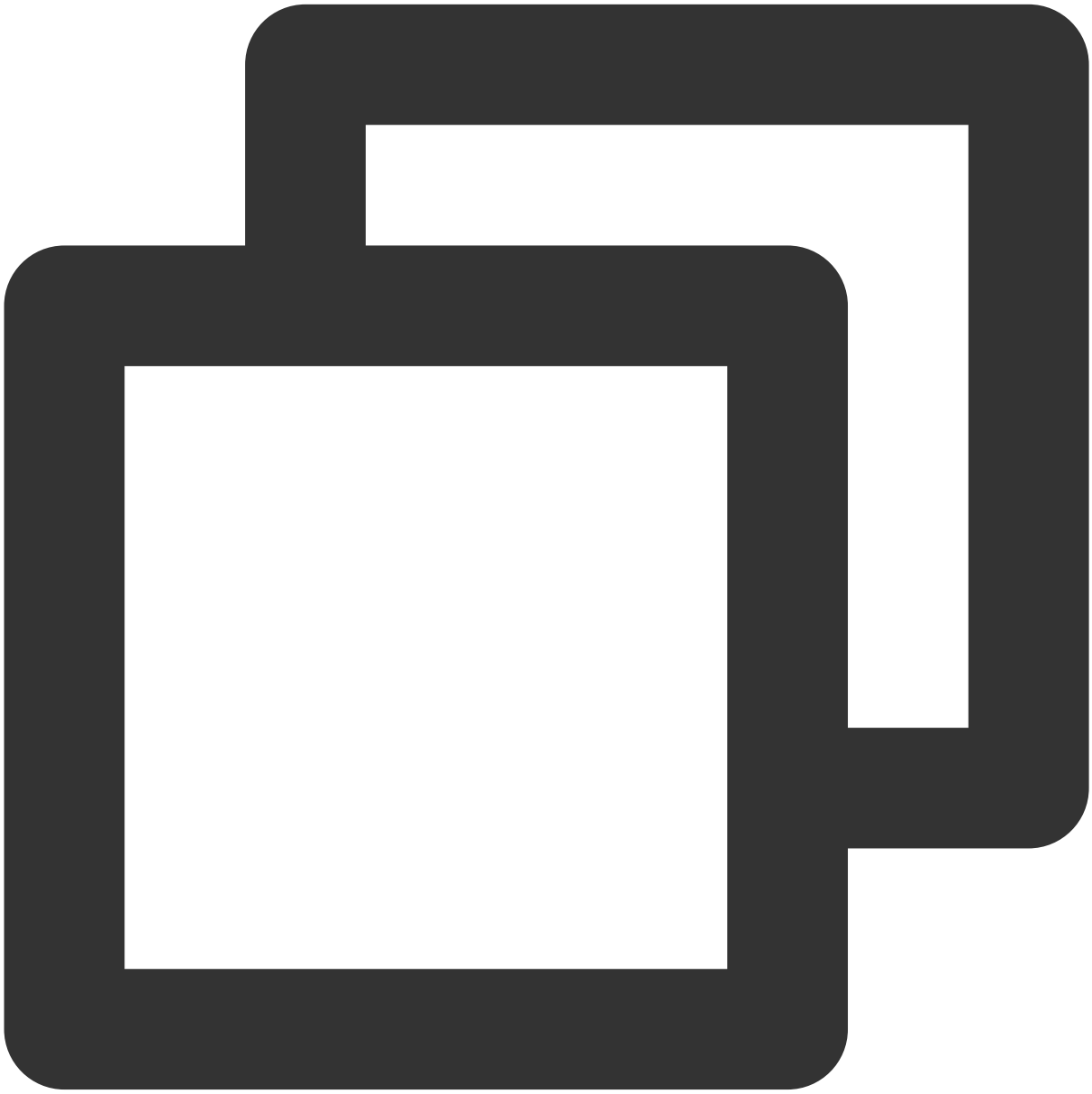


```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;

NSSet *accountsKeys = [[NSSet alloc] initWithObjects:@(accountType), nil];

[[XGPushTokenManager defaultTokenManager] delAccountsByKeys:accountsKeys];
```

清除账号

接口说明

清除所有设置的账号。

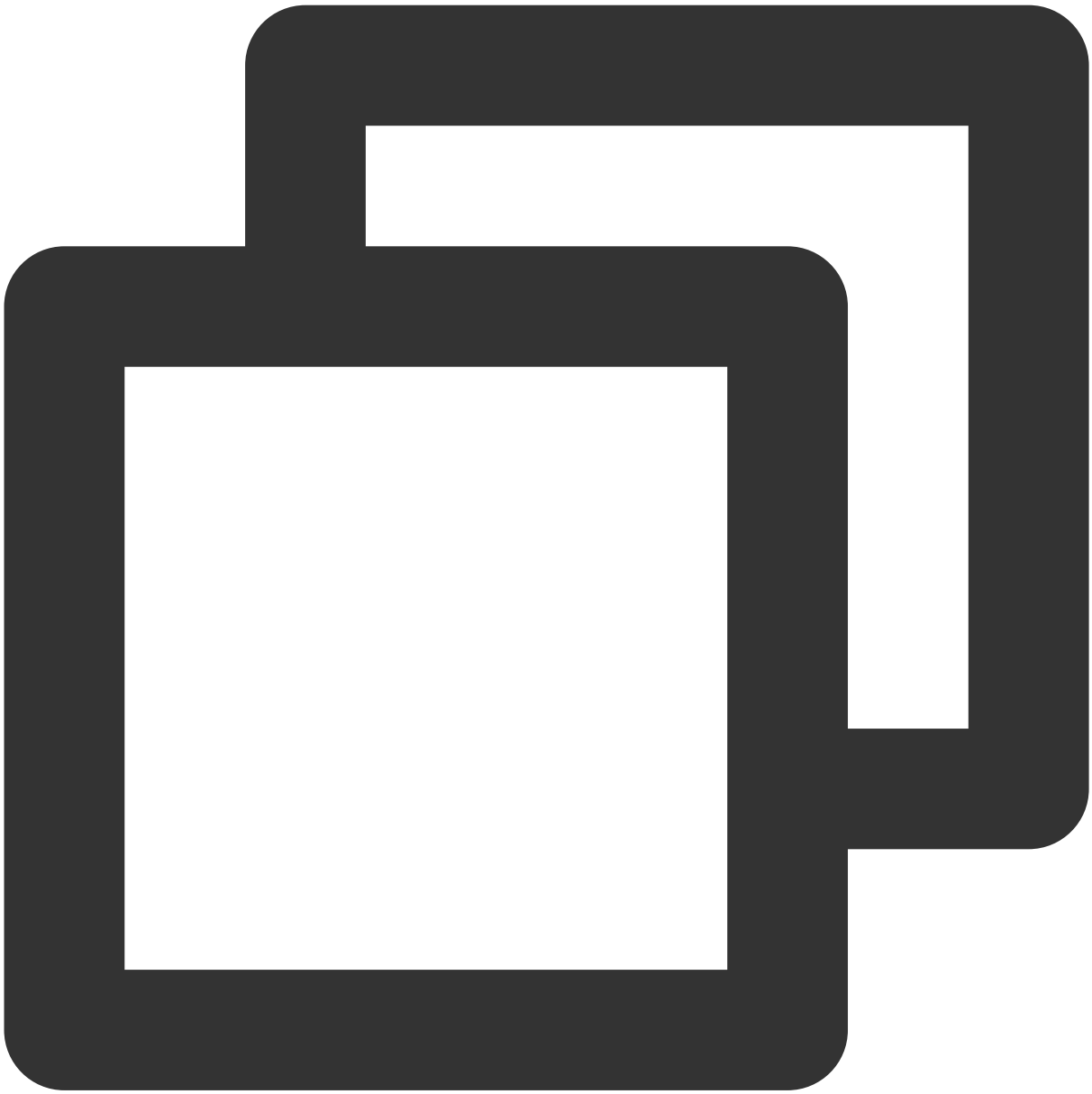


```
- (void)clearAccounts;
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码



```
[[XGPushTokenManager defaultTokenManager] clearAccounts];
```

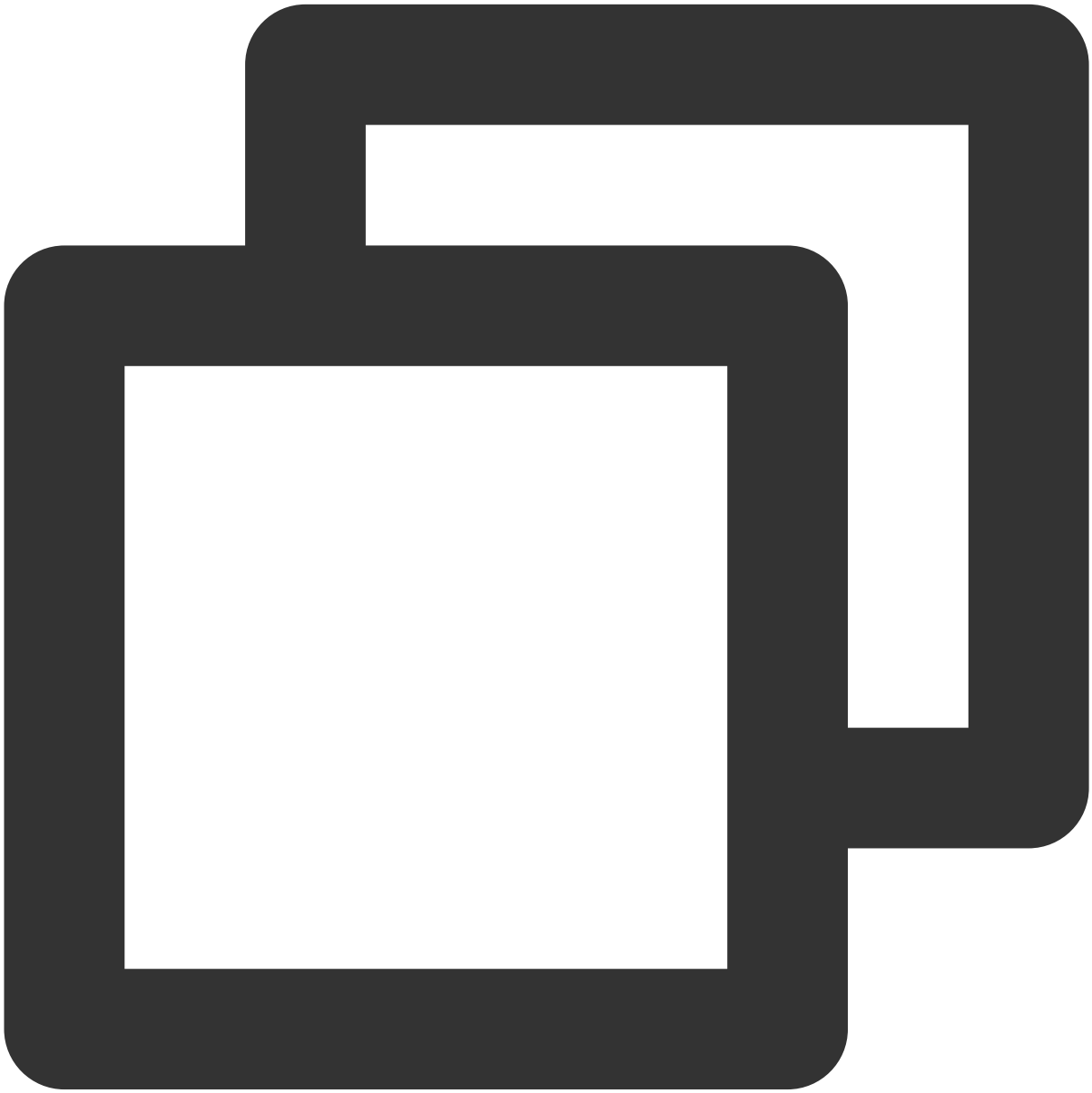
标签功能

以下为标签相关接口方法，若需了解调用时机及调用原理，可查看 标签相关流程。

绑定/解绑标签

接口说明

开发者可以针对不同的用户绑定标签，然后对该标签进行推送。



```
- (void)appendTags:(nonnull NSArray<NSString *> *)tags
- (void)delTags:(nonnull NSArray<NSString *> *)tags
```

说明：

此接口为追加方式。

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用

单个应用最多可以有10000个自定义 tag， 每个设备 Token 最多可绑定100个自定义 tag，如需提高该限制，请 提交工单 联系我们，每个自定义 tag 可绑定的设备 Token 数量无限制。

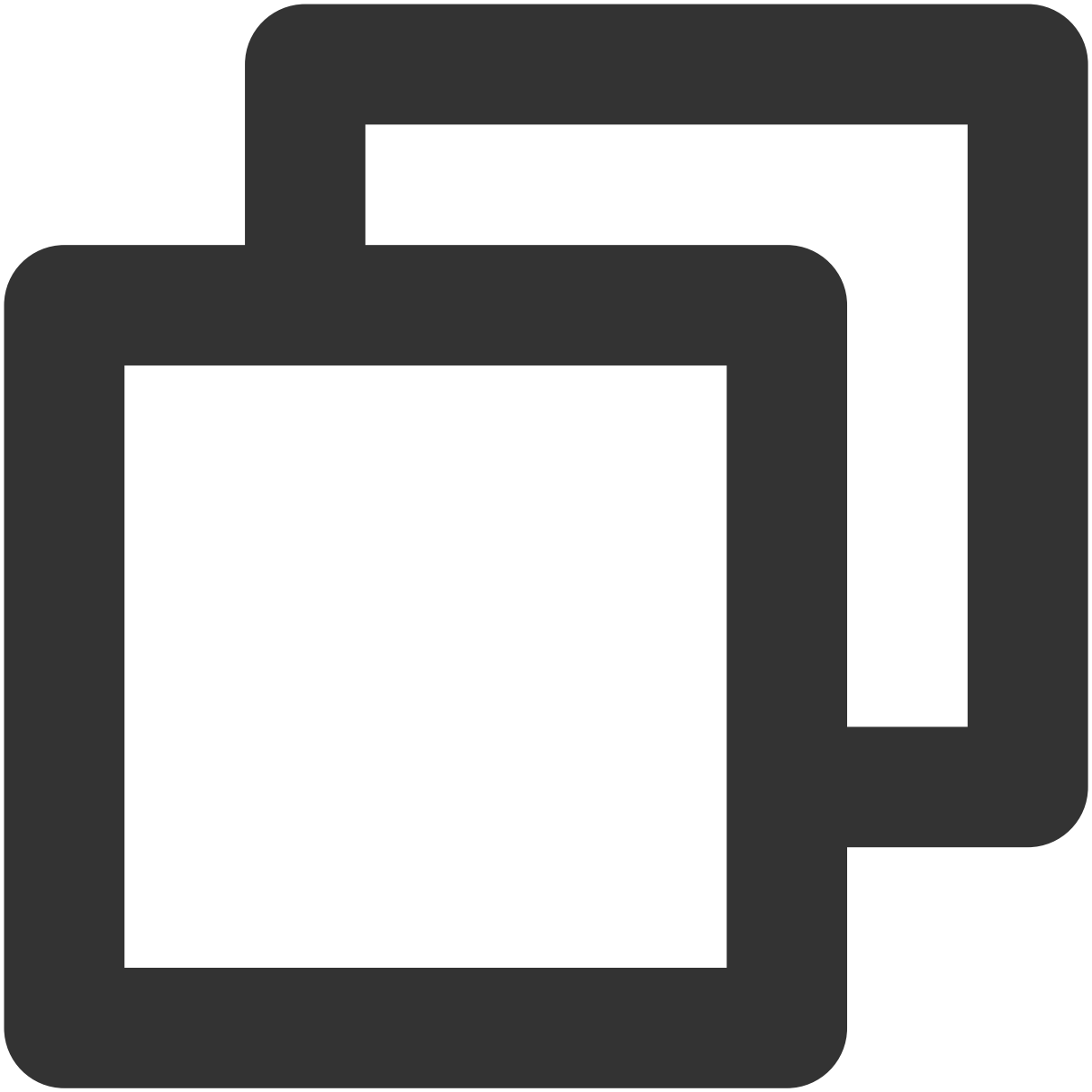
参数说明

tags：标签数组。

说明：

标签操作 tags 为标签字符串数组（标签字符串不允许有空格或者是 tab 字符）。

示例代码

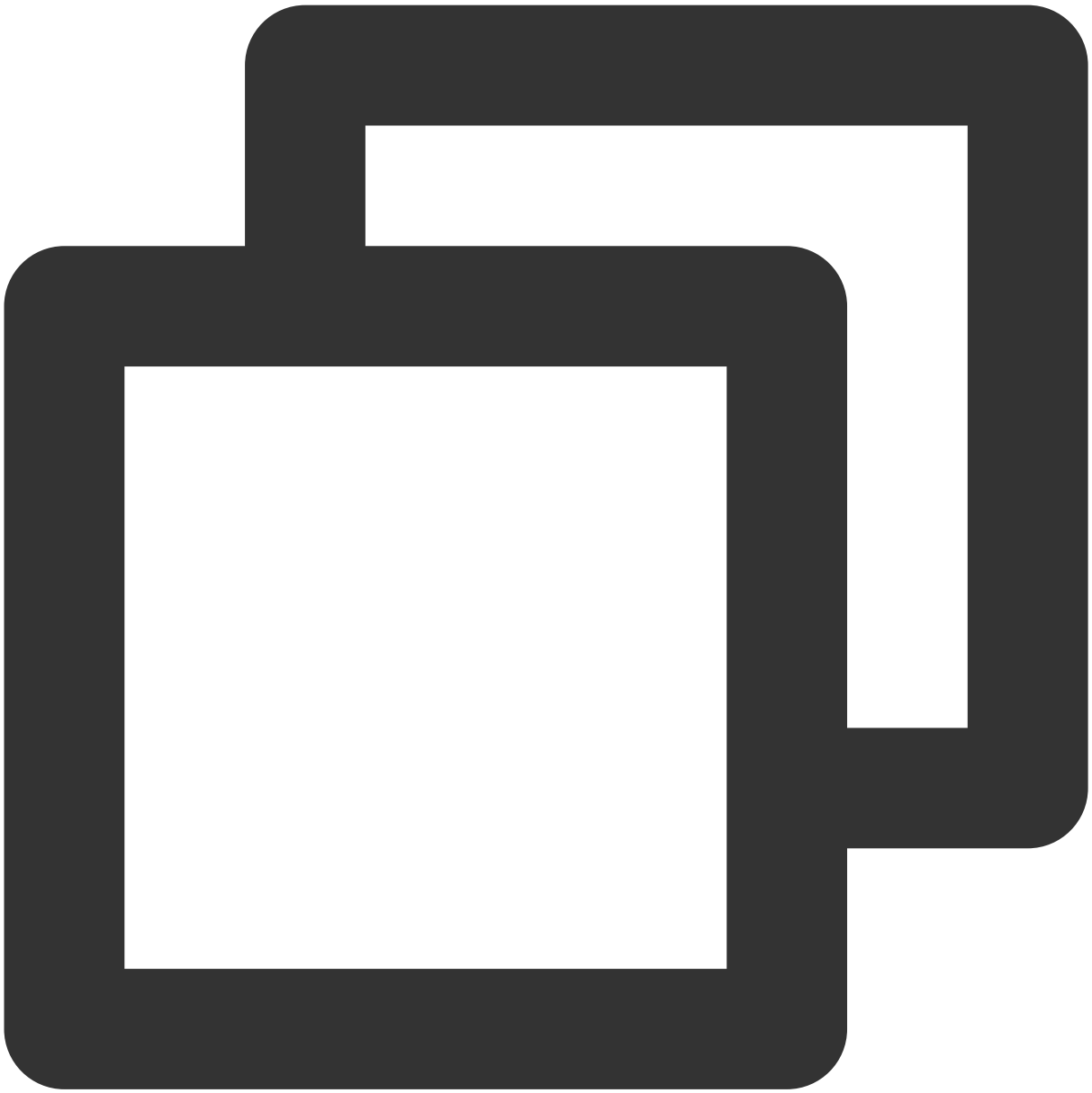


```
//绑定标签：
[[XGPushTokenManager defaultTokenManager] appendTags:@[ tagStr ]];

//解绑标签
[[XGPushTokenManager defaultTokenManager] delTags:@[ tagStr ]];
```

更新标签

接口说明

清空已有标签，然后批量添加标签。



```
- (void)clearAndAppendTags:(nonnull NSArray<NSString *> *)tags
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

此接口会将当前 Token 对应的旧有的标签全部替换为当前的标签。

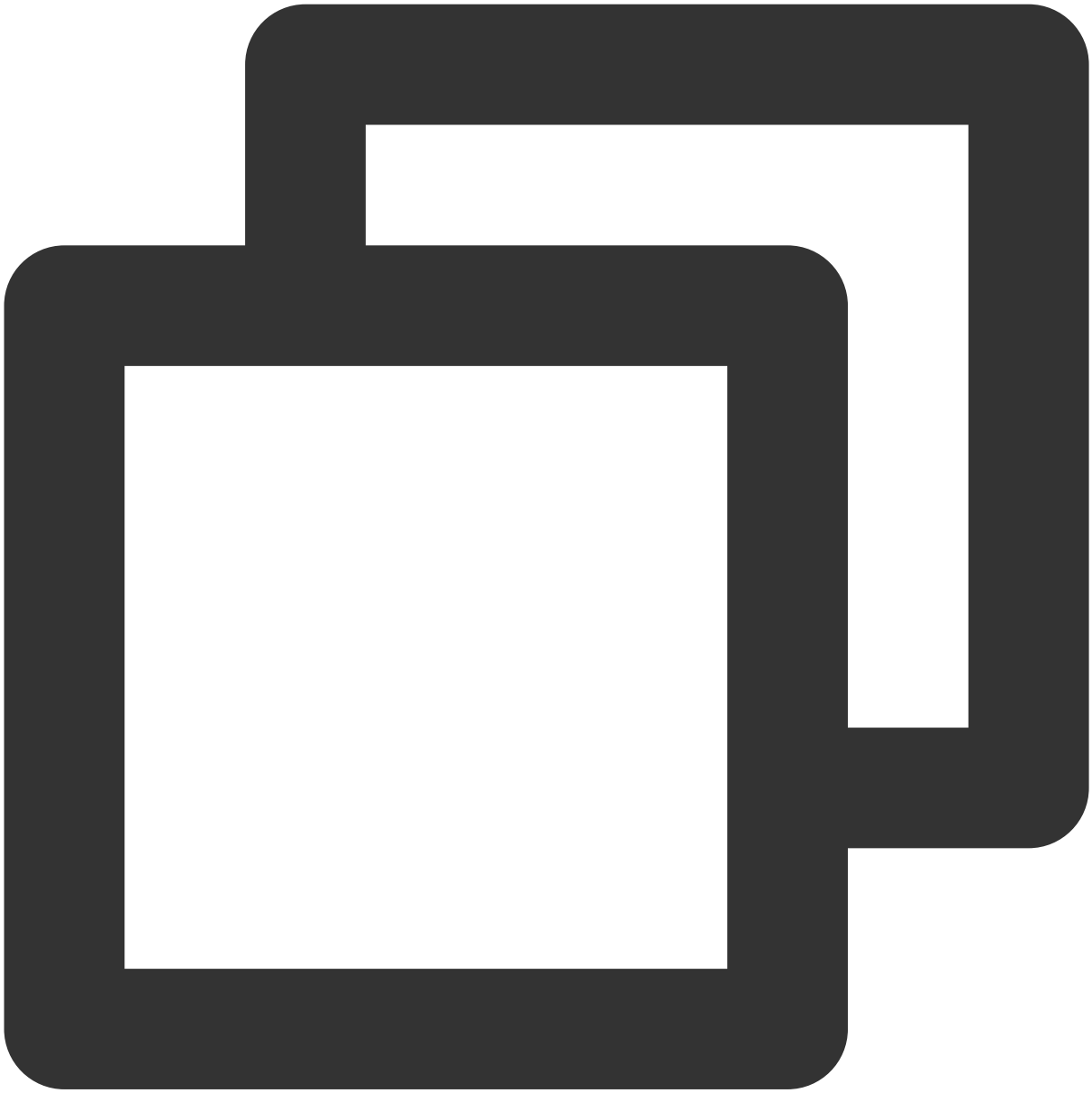
参数说明

tags：标签数组。

说明：

标签操作 tags 为标签字符串数组（标签字符串不允许有空格或者是 tab 字符）。

示例代码

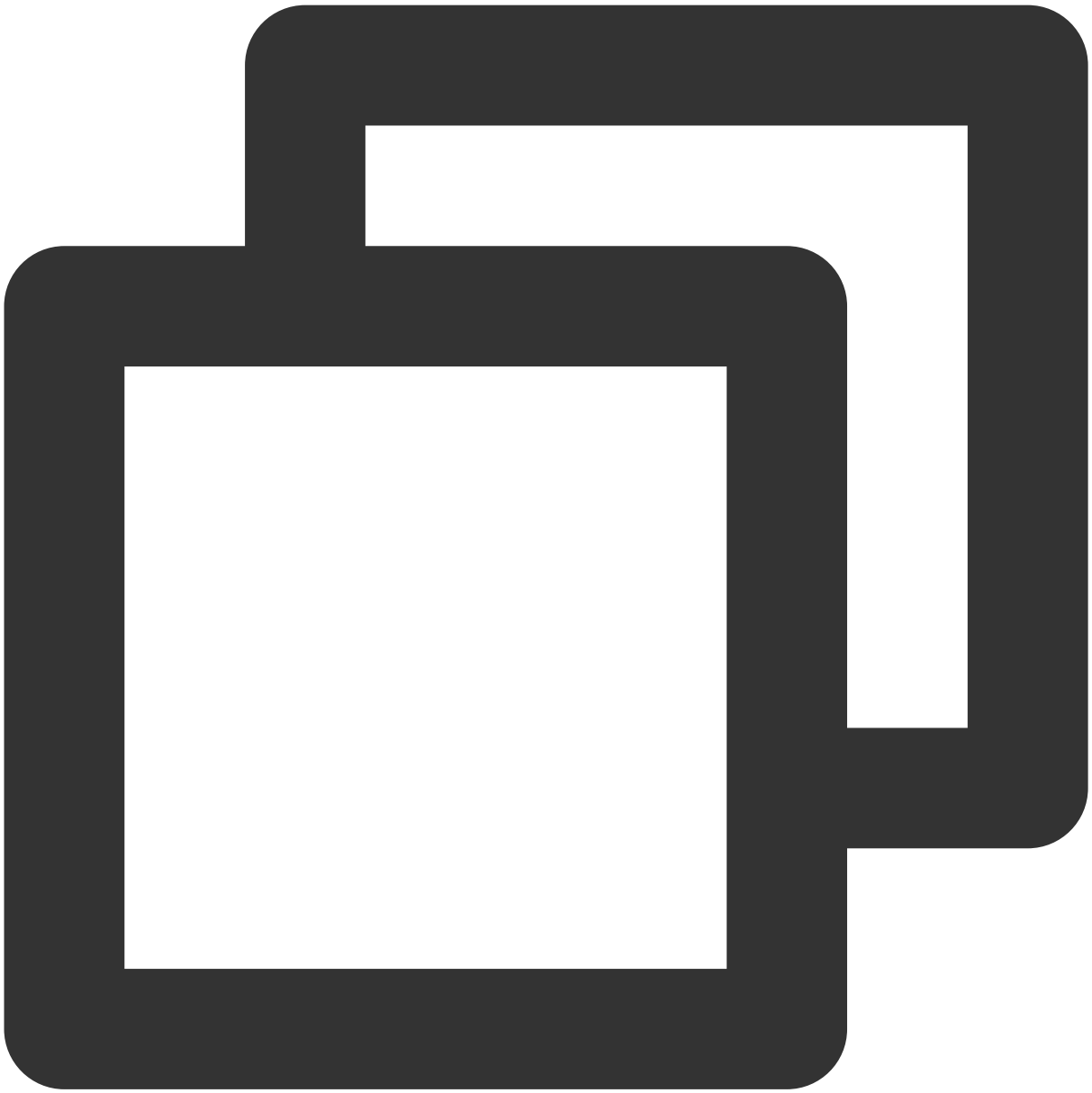


```
[[XGPushTokenManager defaultTokenManager] clearAndAppendTags:@[ tagStr ]];
```

清除全部标签

接口说明

清除所有设置的标签。

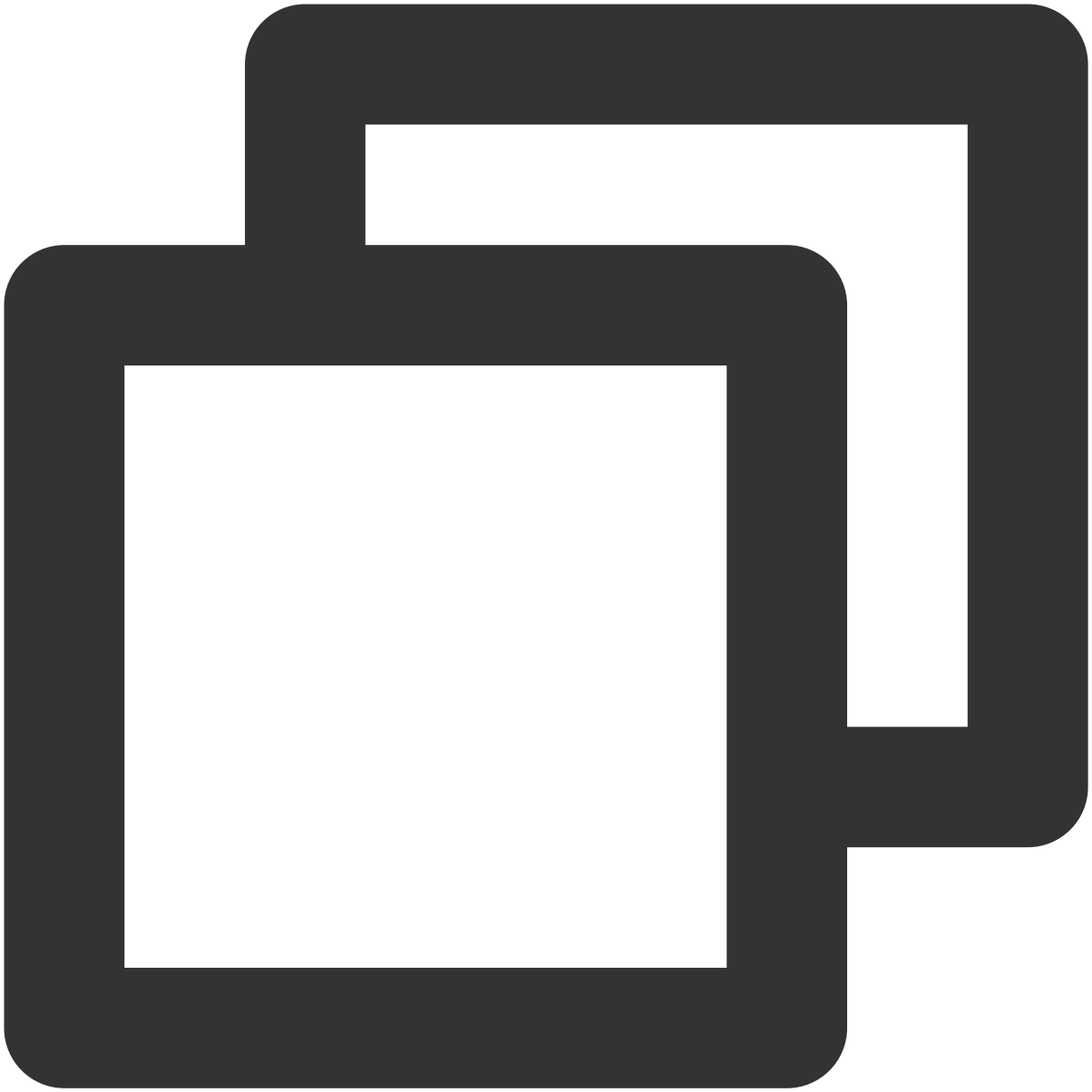


```
- (void)clearTags
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码

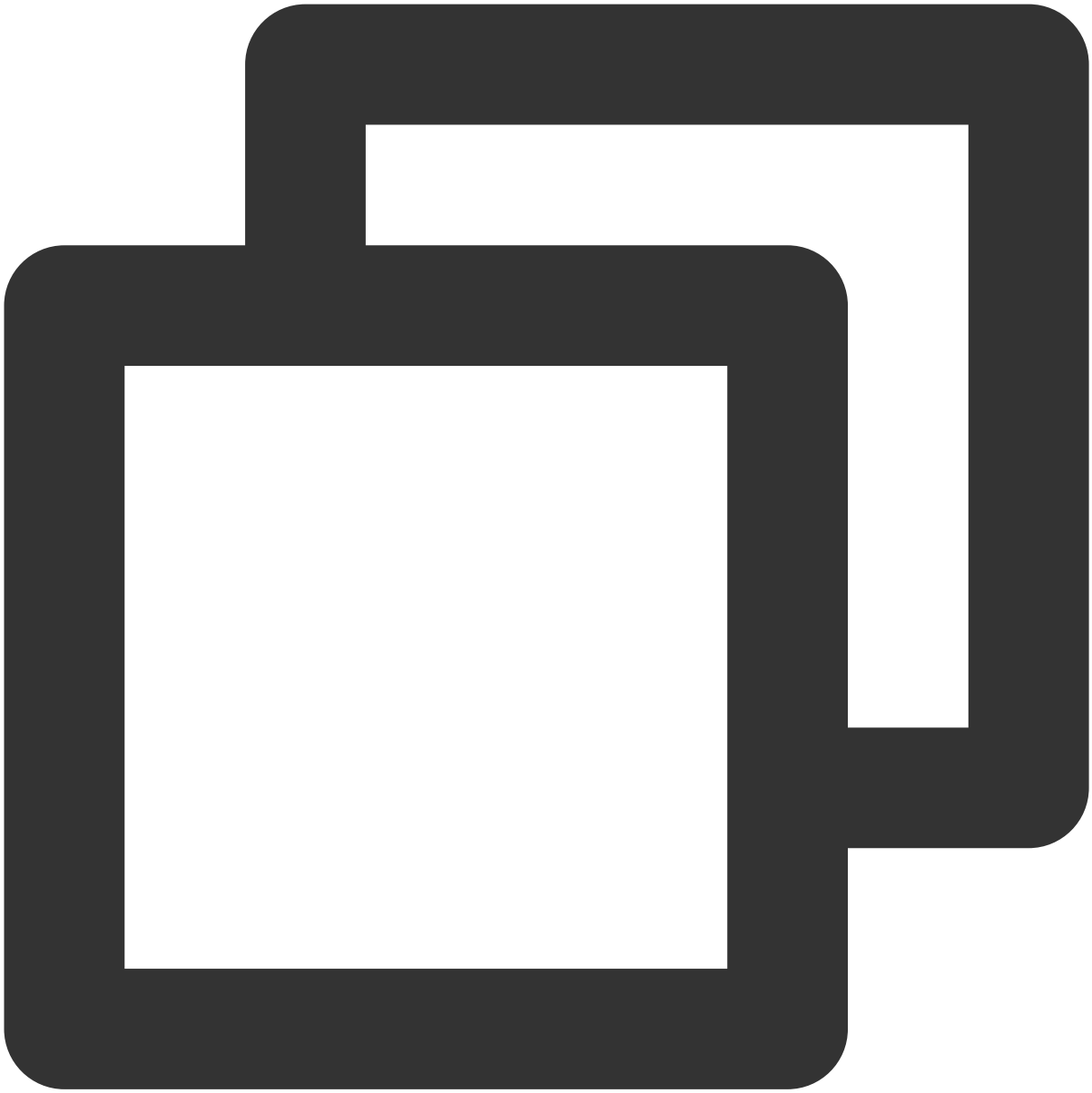


```
[[XGPushTokenManager defaultTokenManager] clearTags];
```

查询标签

接口说明

查询设备绑定的标签。



```
- (void)queryTags:(NSUInteger)offset limit:(NSUInteger)limit;
```

说明：

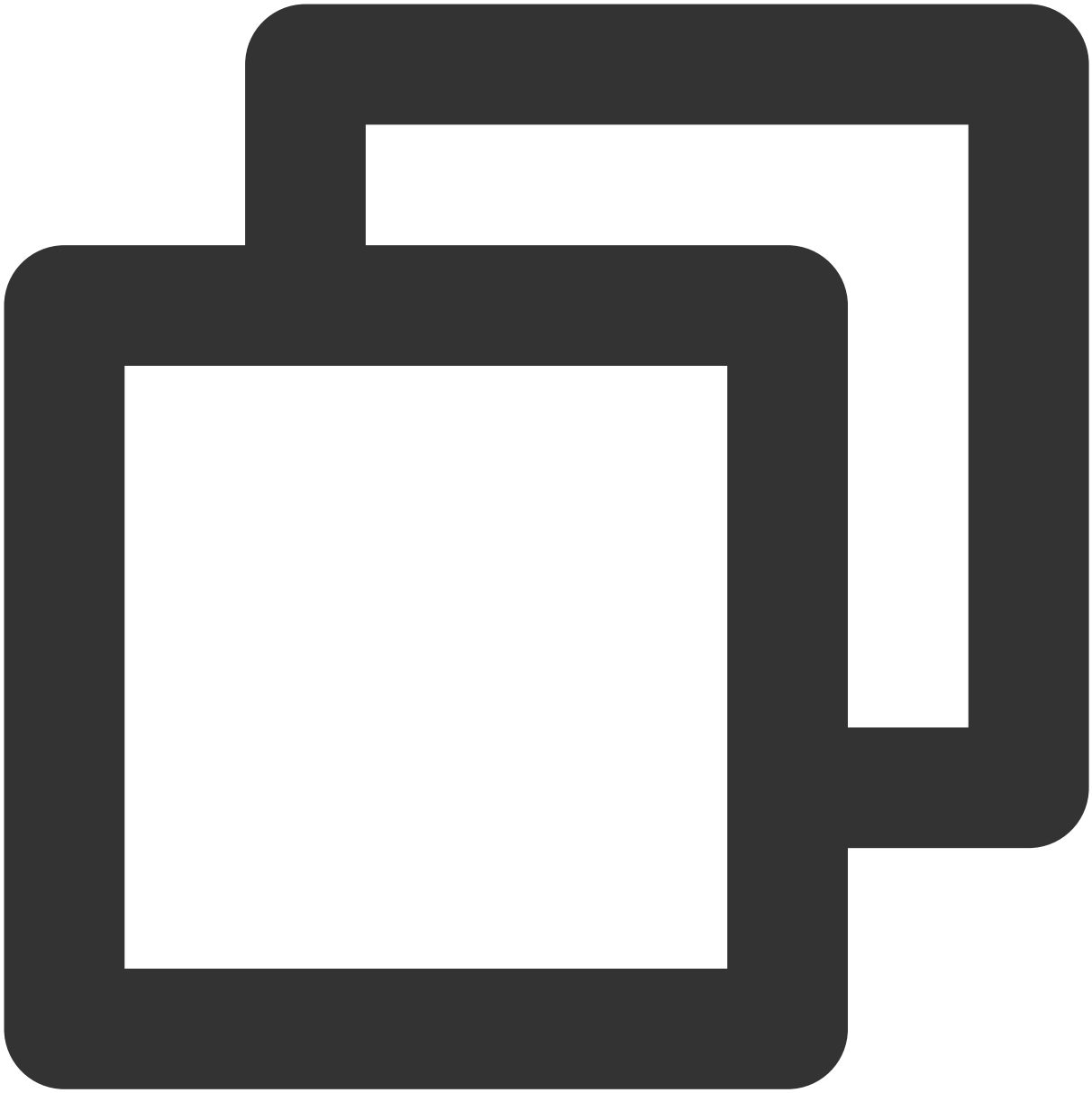
此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

参数说明

offset：此次查询的偏移大小。

offset：limit 此次查询的分页大小, 最大200。

示例代码

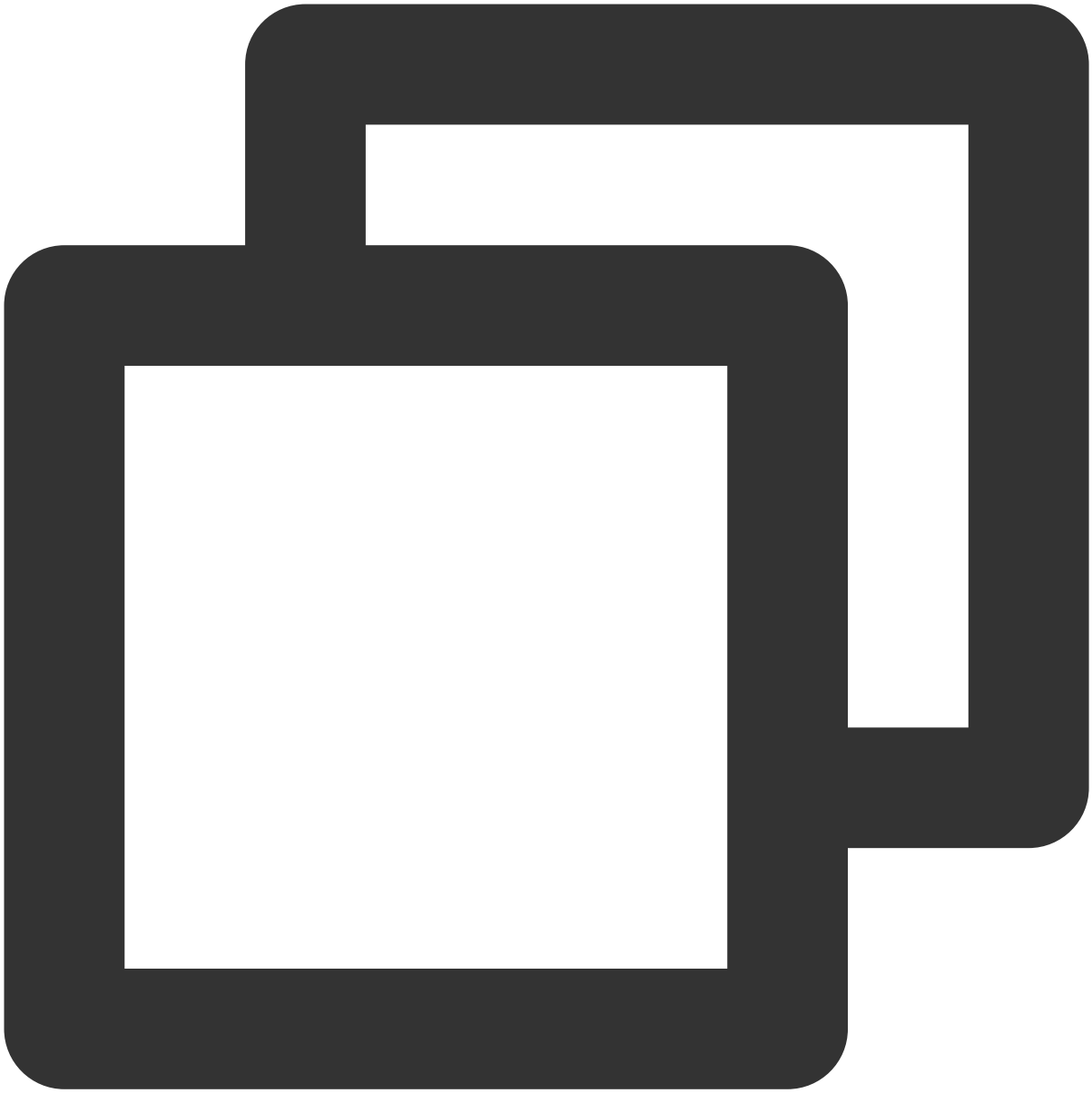


```
[[XGPushTokenManager defaultTokenManager] queryTags:0 limit:100];
```

查询标签的回调

接口说明

查询标签的结果会走此回调。



```
- (void)xgPushDidQueryTags:(nullable NSArray<NSString *> *)tags totalCount:(NSUInte
```

返回参数说明

tags：查询条件返回的标签。

totalCount：设备绑定的总标签数量。

error：错误信息，若 error 为 nil，则查询成功。

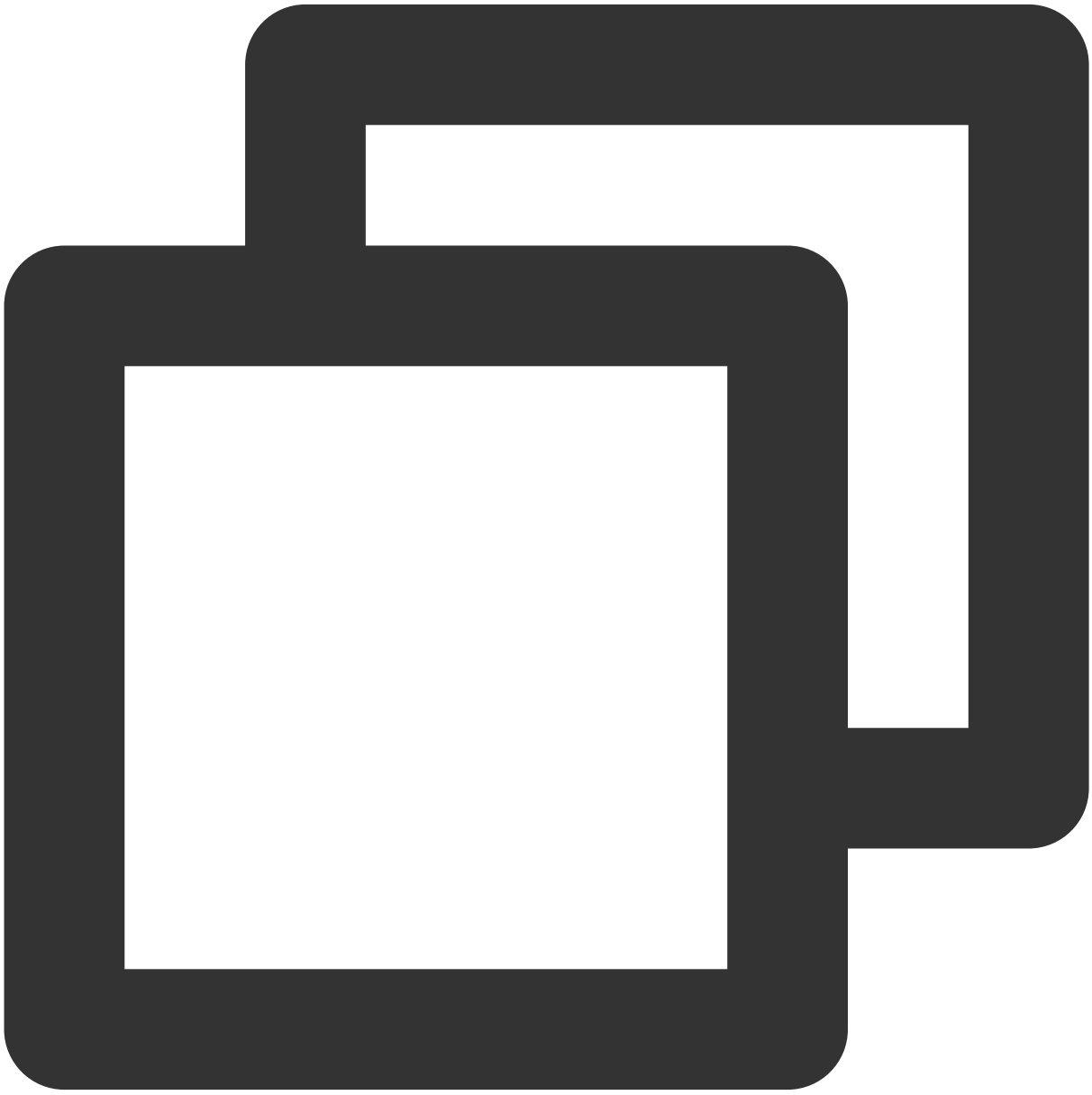
用户属性功能

以下为用户属性相关接口方法，若需了解调用时机及调用原理，可查看 用户属性相关流程。

新增用户属性

接口说明

添加或更新用户属性（key-value 结构，若原来没有该 key 的用户属性 value，则新增；若原来有该 key 的用户属性 value，则更新该 value）。



```
- (void)upsertAttributes:(nonnull NSDictionary<NSString *,NSString *> *)attributes
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

参数说明

attributes：用户属性字符串字典，字符串不允许有空格或者是 tab 字符。

说明：

需要先在管理台配置用户属性的键，才能操作成功。

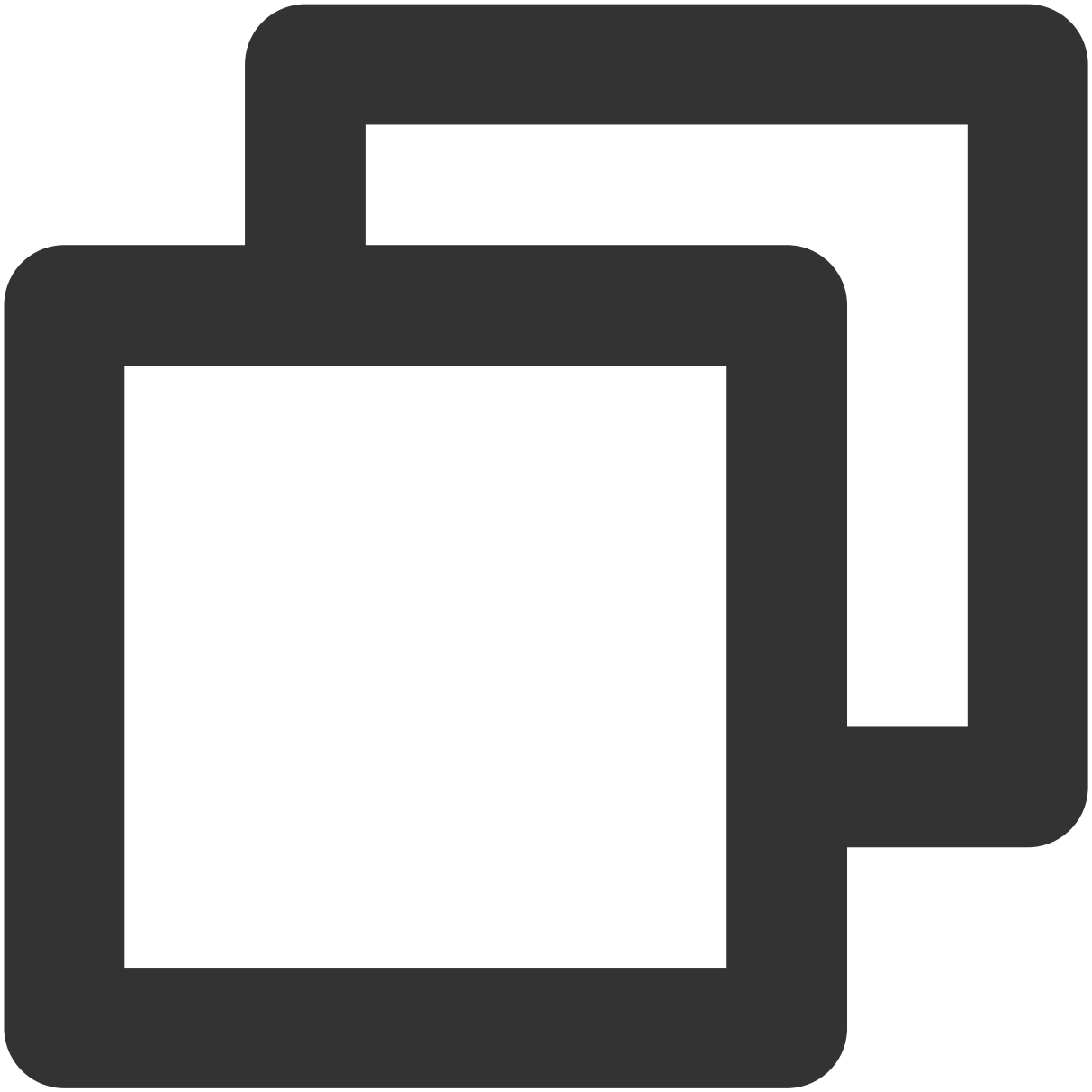
key，value 长度都限制50个字符以内。

需要使用字典且 key 是固定要求。

Objective-C 的写法 : @{@"gender": @"Female", @"age": @"29"}

Swift 的写法：["gender":"Female", "age": "29"]

示例代码

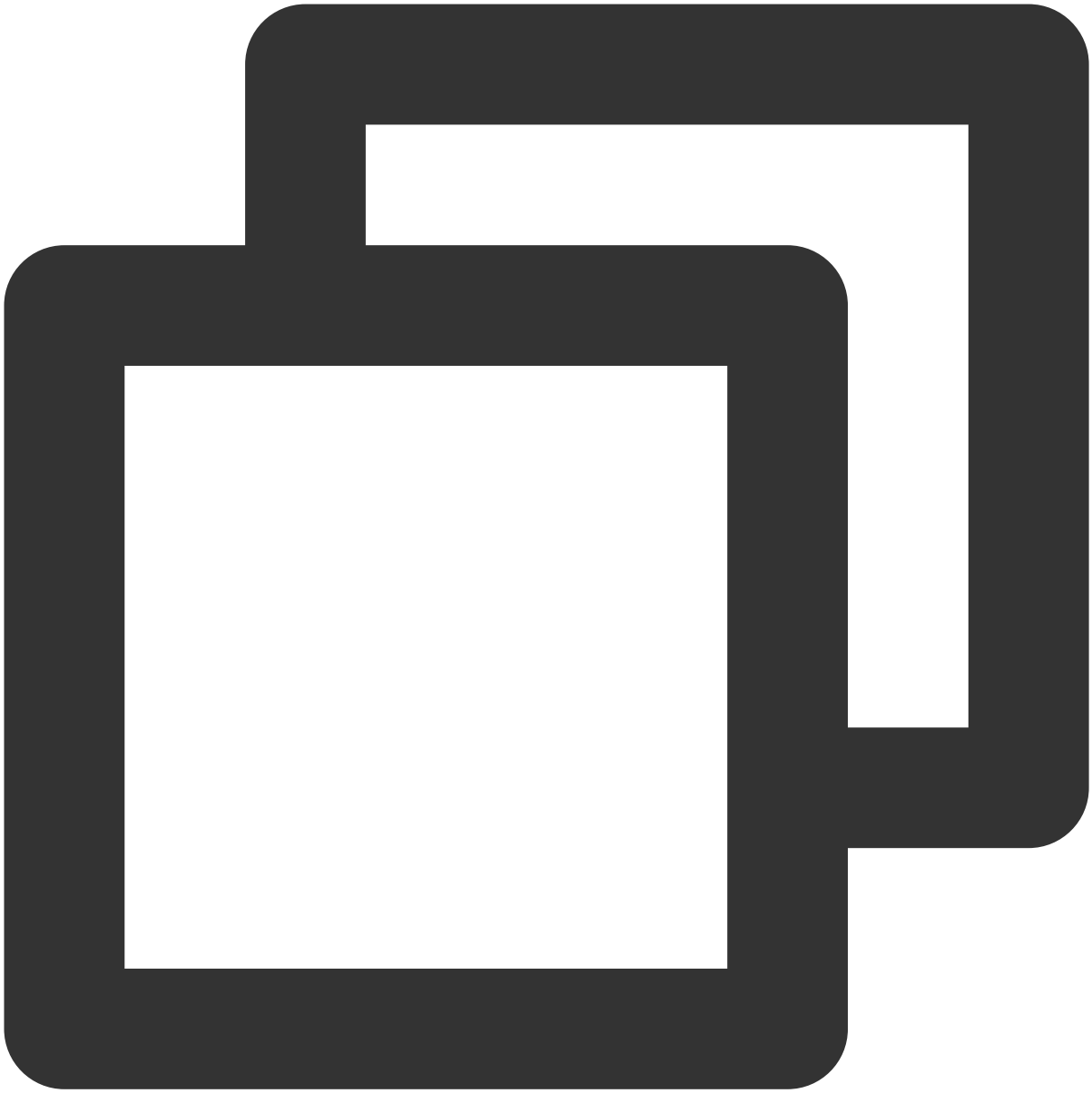


```
[[XGPushTokenManager defaultTokenManager] upsertAttributes:attributes];
```

删除用户属性

接口说明

删除用户已有的属性。



```
- (void)delAttributes:(nonnull NSSet<NSString *> *)attributeKeys
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

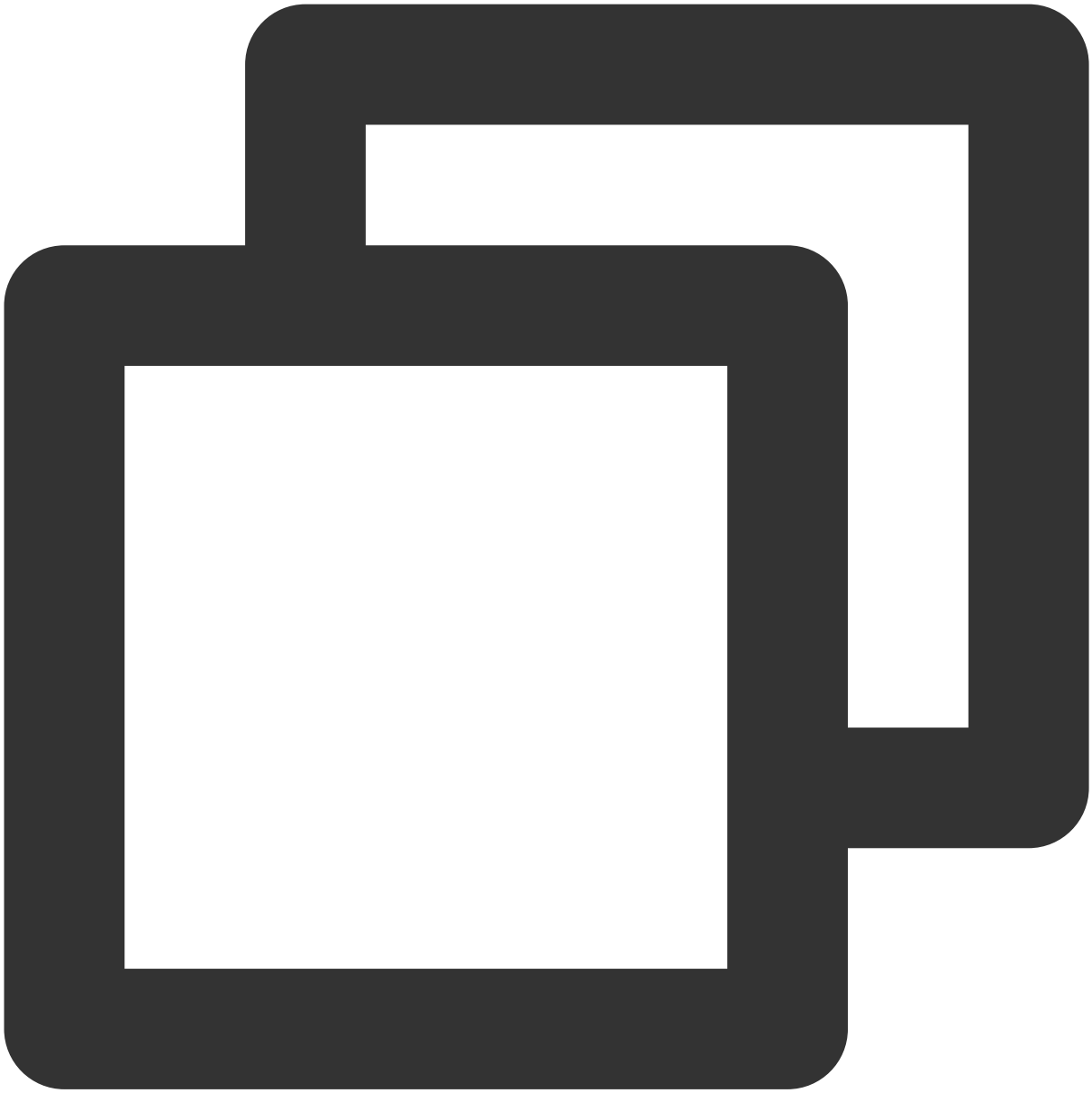
参数说明

attributeKeys：用户属性 key 组成的集合，字符串不允许有空格或者是 tab 字符。

说明：

使用集合且 key 是固定要求。

示例代码

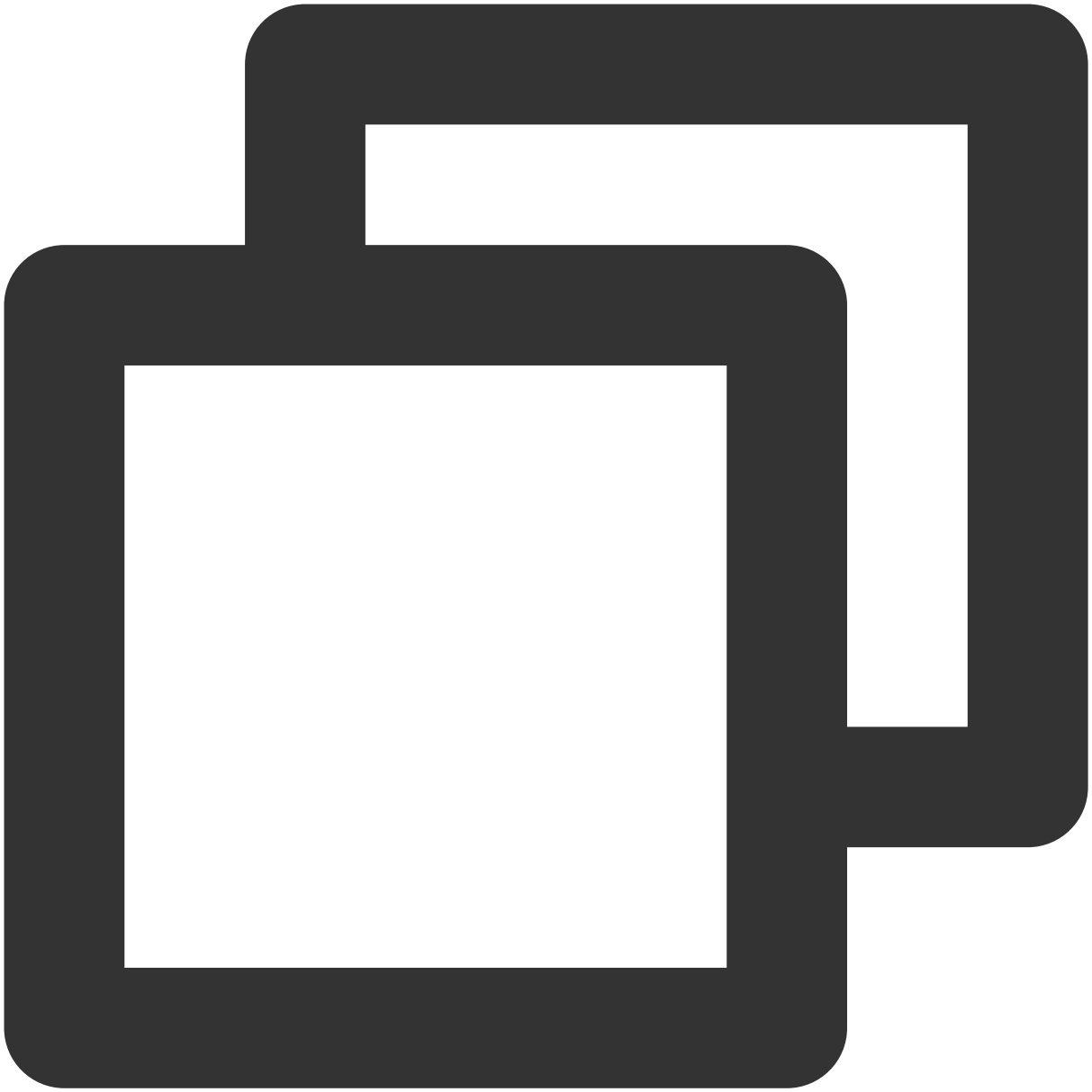


```
[[XGPushTokenManager defaultTokenManager] delAttributes:attributeKeys];
```

清空已有用户属性

接口说明

清空已有用户属性。

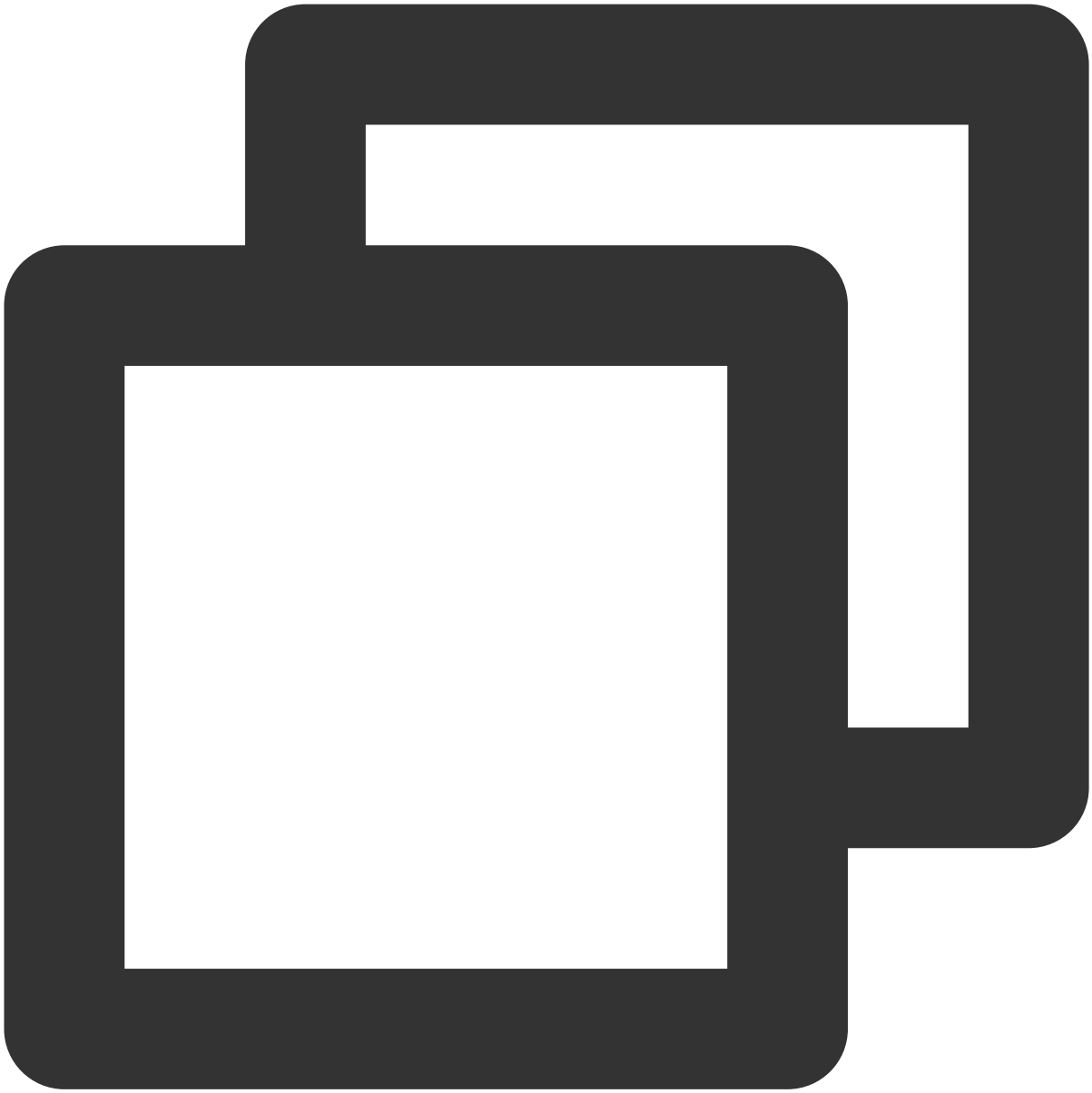


```
- (void)clearAttributes;
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码

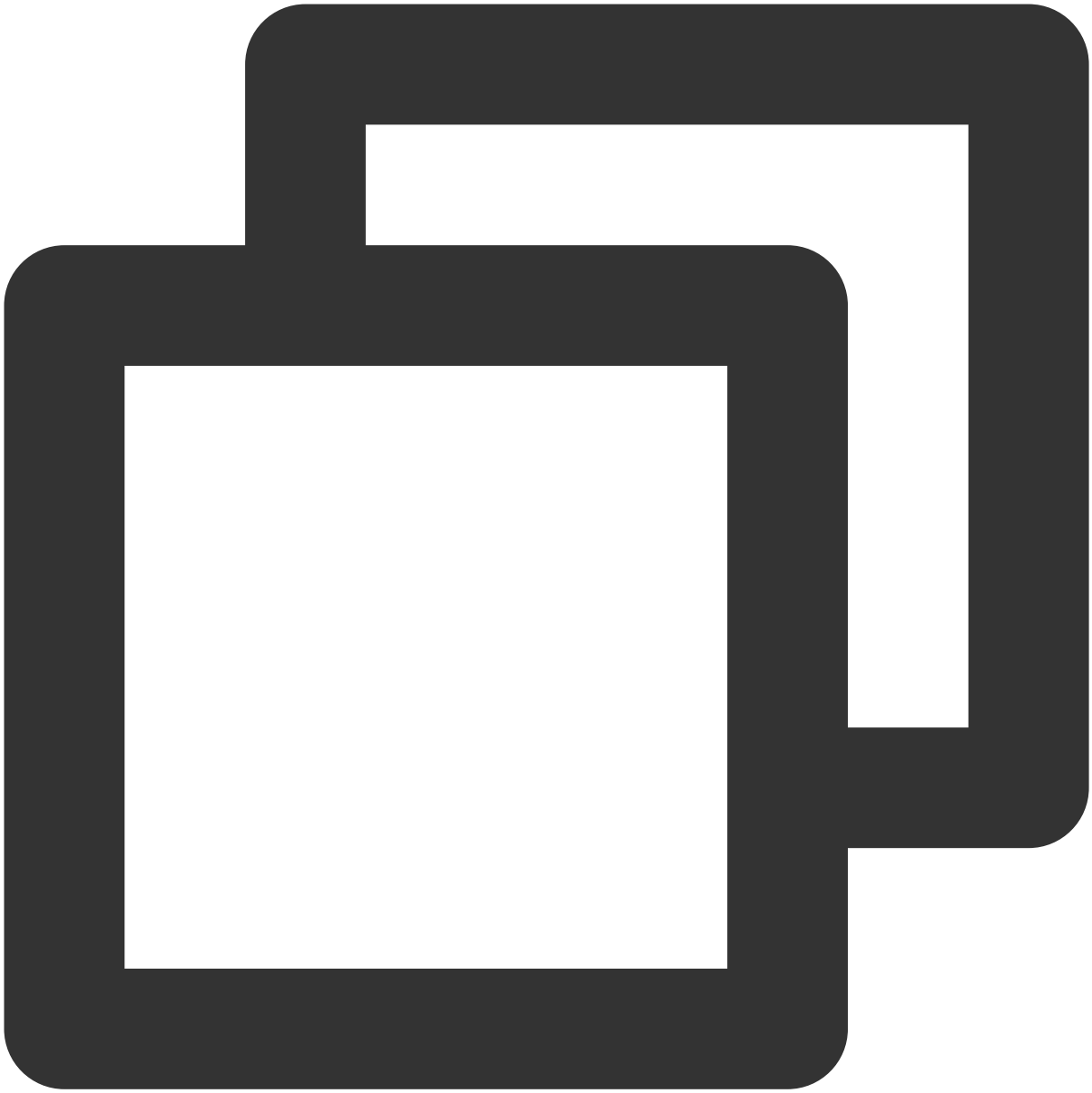


```
[[XGPushTokenManager defaultTokenManager] clearAttributes];
```

更新用户属性

接口说明

清空已有用户属性，然后批量添加用户属性。

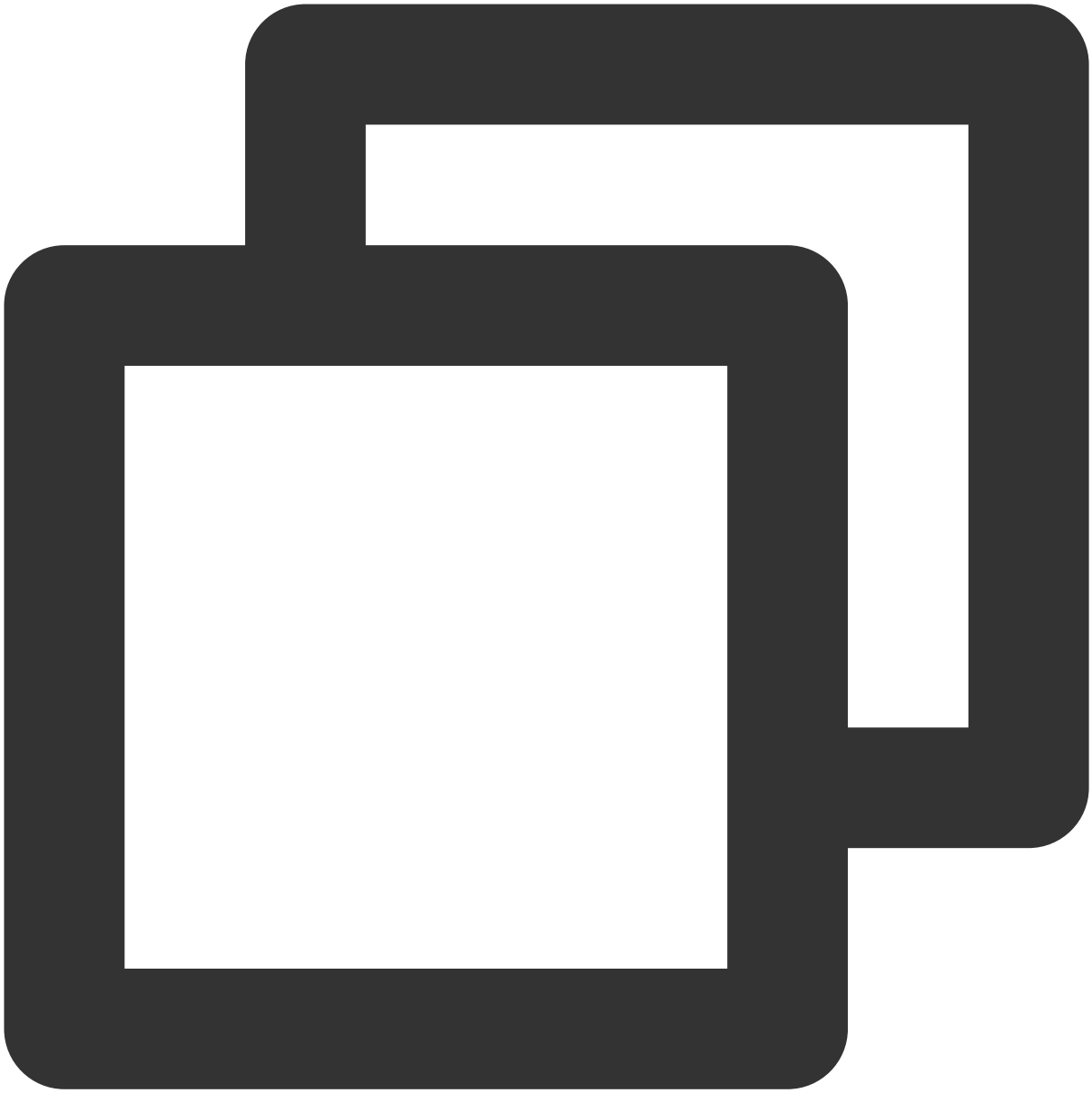


```
- (void)clearAndAppendAttributes:(nonnull NSDictionary<NSString *,NSString *> *)att
```

说明：

此接口应在 xgPushDidRegisteredDeviceToken:error: 返回正确后被调用。

示例代码



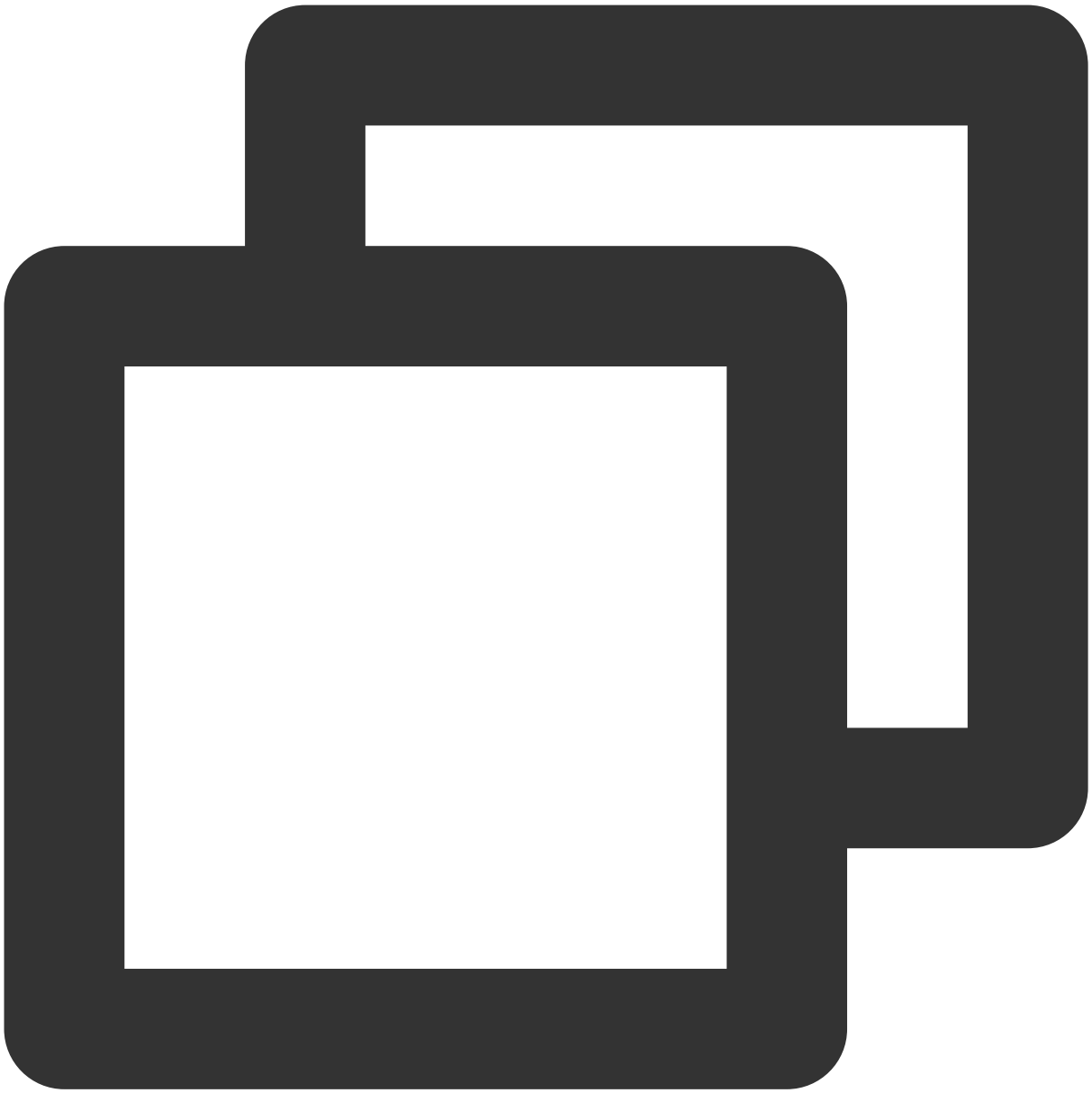
```
[[XGPushTokenManager defaultTokenManager] clearAndAppendAttributes:attributes];
```

角标功能

同步角标

接口说明

当应用本地角标值更改后，需调用此接口将角标值同步到移动推送服务器，下次推送时以此值为基准，此功能在管理台位置（【新建推送】>【高级设置】>【角标数字】）。



```
- (void)setBadge:(NSInteger)badgeNumber;
```

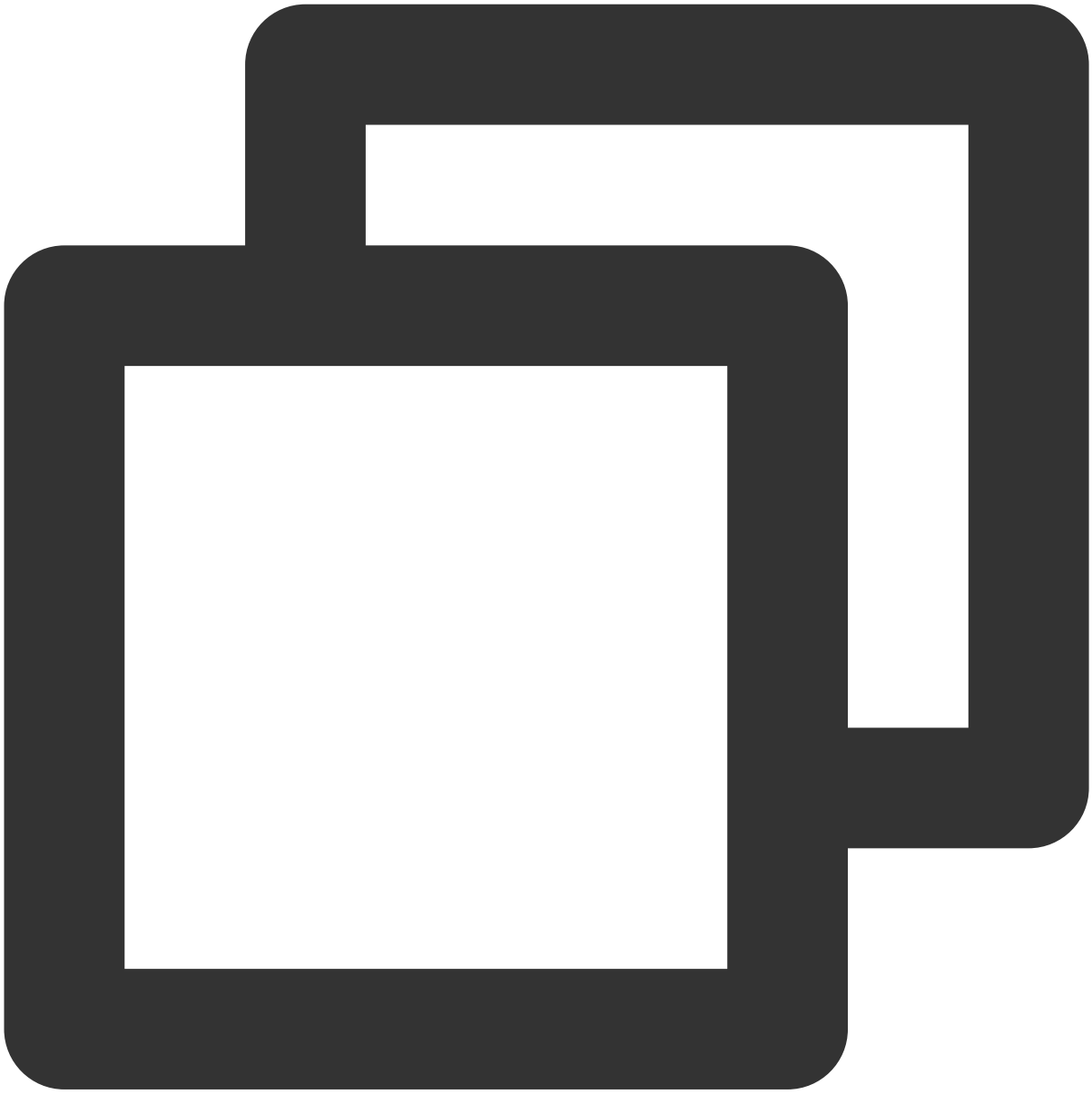
参数说明

badgeNumber：应用的角标数。

注意：

当本地应用角标设置后需调用此接口同步角标值到移动推送服务器，并在下次推送时生效，此接口必须在移动推送注册成功后调用（xgPushDidRegisteredDeviceToken）。

示例代码



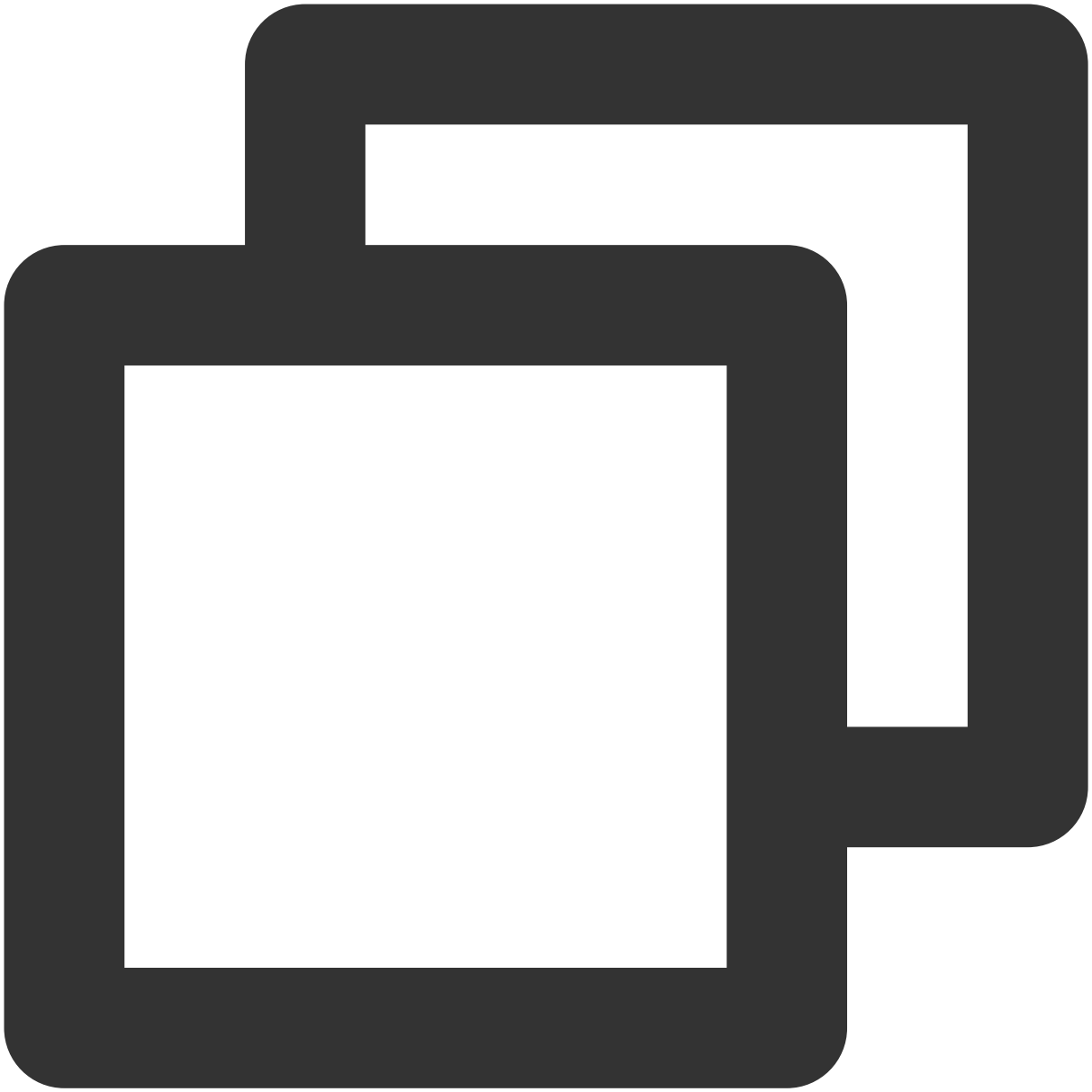
```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    /// 每次启动 App 应用角标清零（本地应用角标设置需要在主线程执行）
    if ([XGPush defaultManager].xgApplicationBadgeNumber > 0) {
        [XGPush defaultManager].xgApplicationBadgeNumber = 0;
    }
    return YES;
```

```
}

- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nu
    /// 在注册完成后同步角标数到 TPNS
    if (!error) {
        [[XGPush defaultManager] setBadge:0];
    }
}
```

查询设备通知权限

接口说明

查询设备通知权限是否被用户允许。

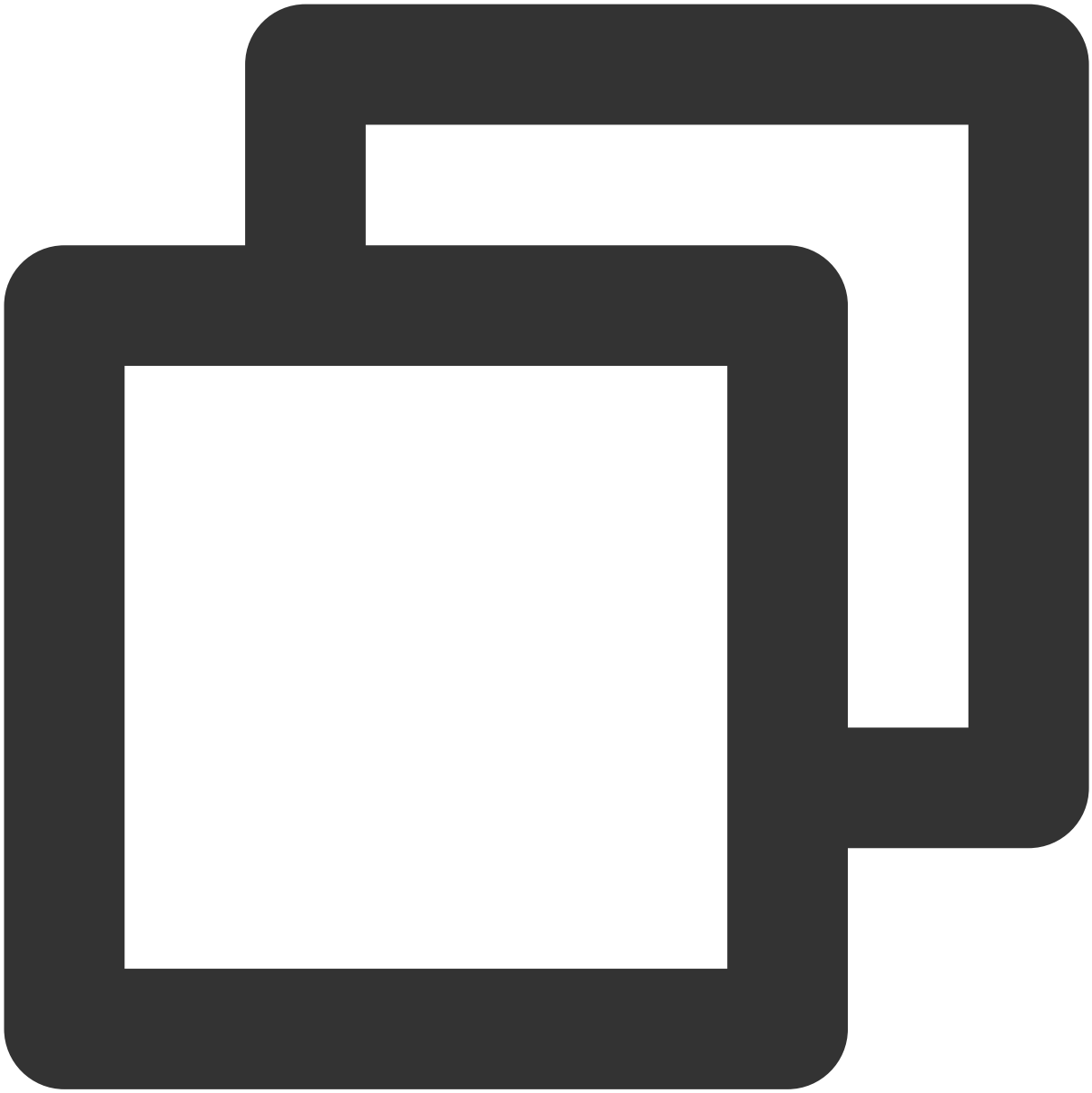


```
- (void)deviceNotificationIsAllowed:(nonnull void (^)(BOOL isAllowed))handler;
```

参数说明

handler：查询结果的返回方法。

示例代码

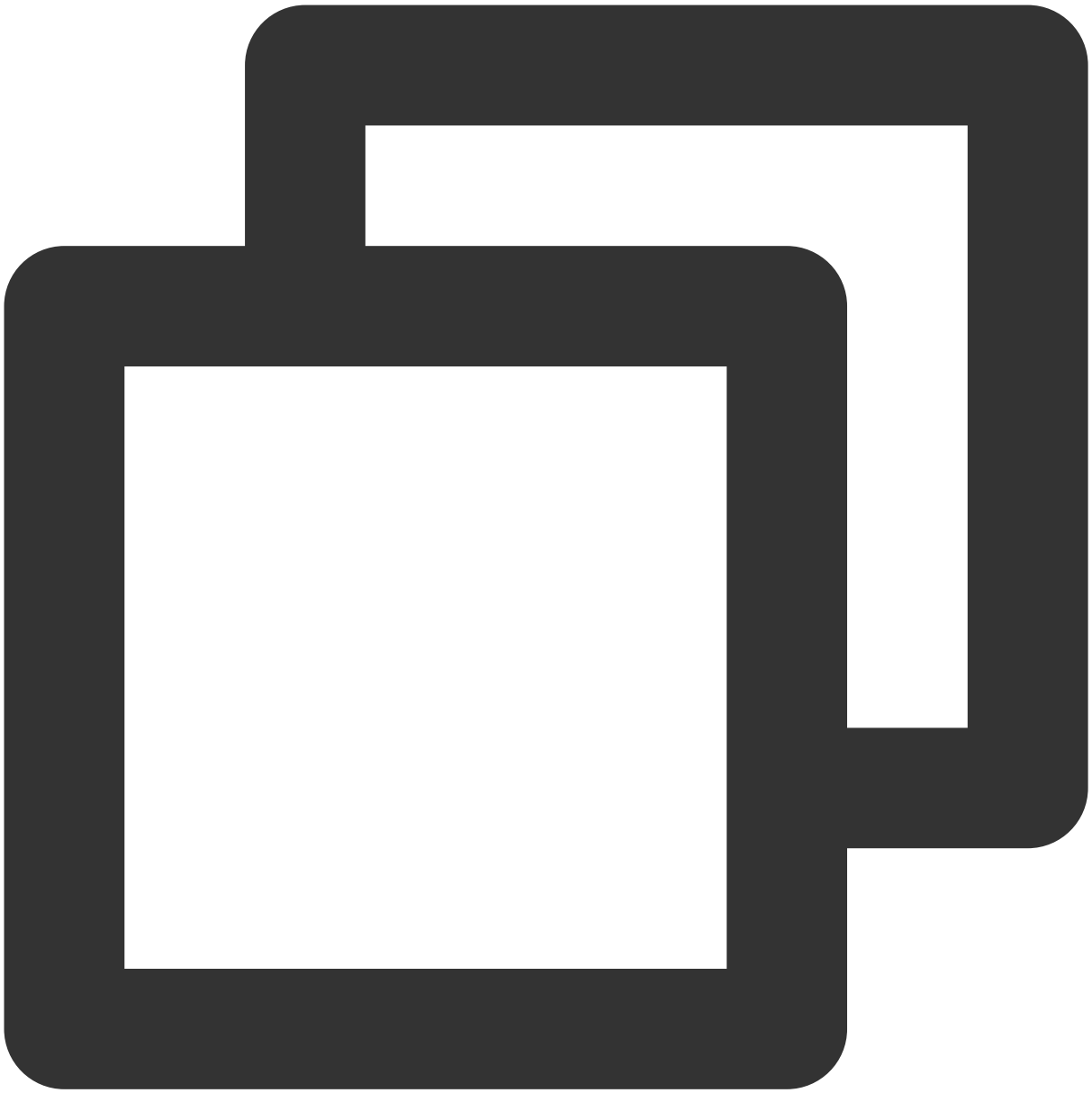


```
[[XGPush defaultManager] deviceNotificationIsAllowed:^(BOOL isAllowed) {
        <#code#>
    }];
```

查询 SDK 版本

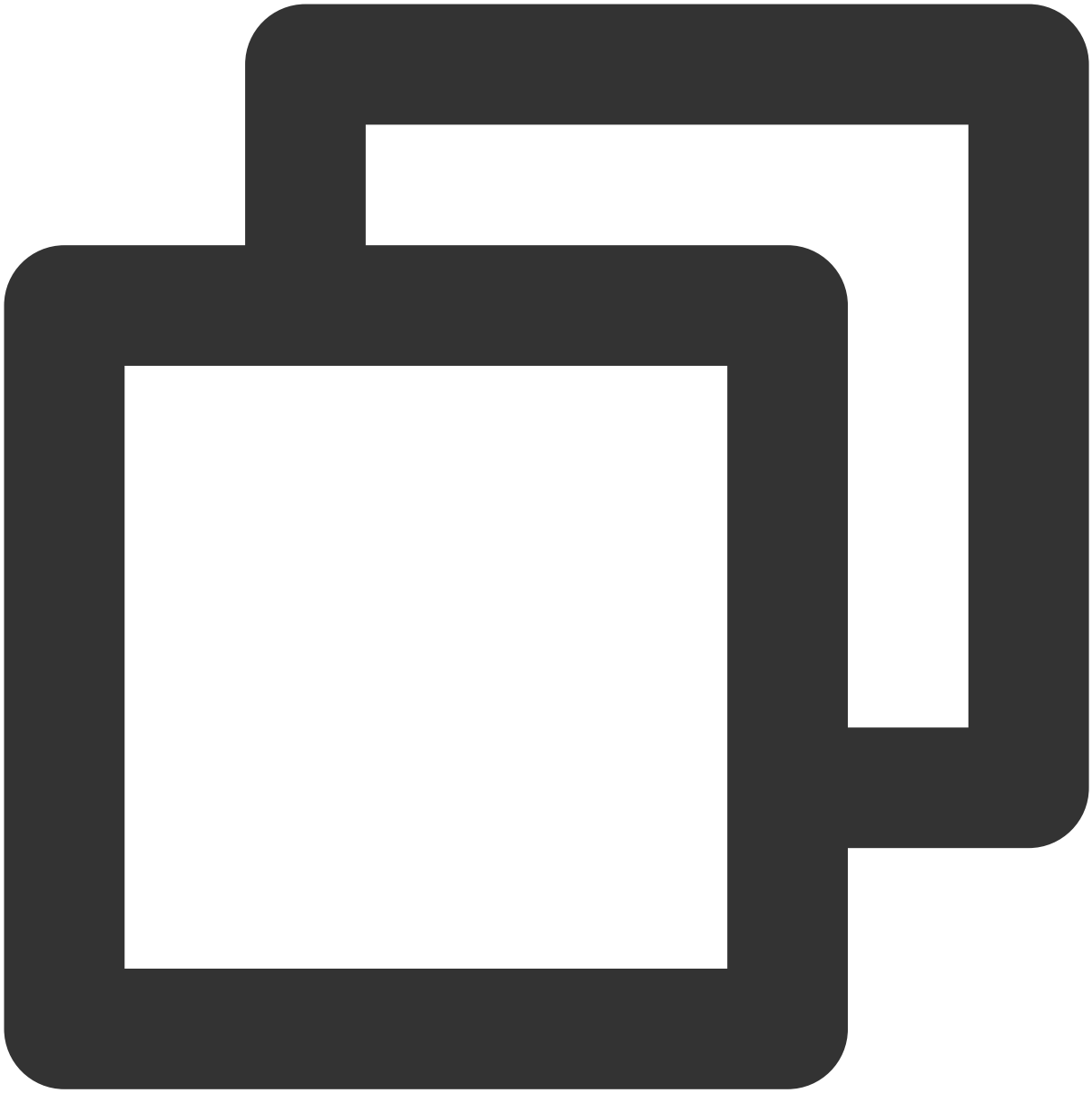
接口说明

查询当前 SDK 的版本。



```
- (nonnull NSString *)sdkVersion;
```

示例代码

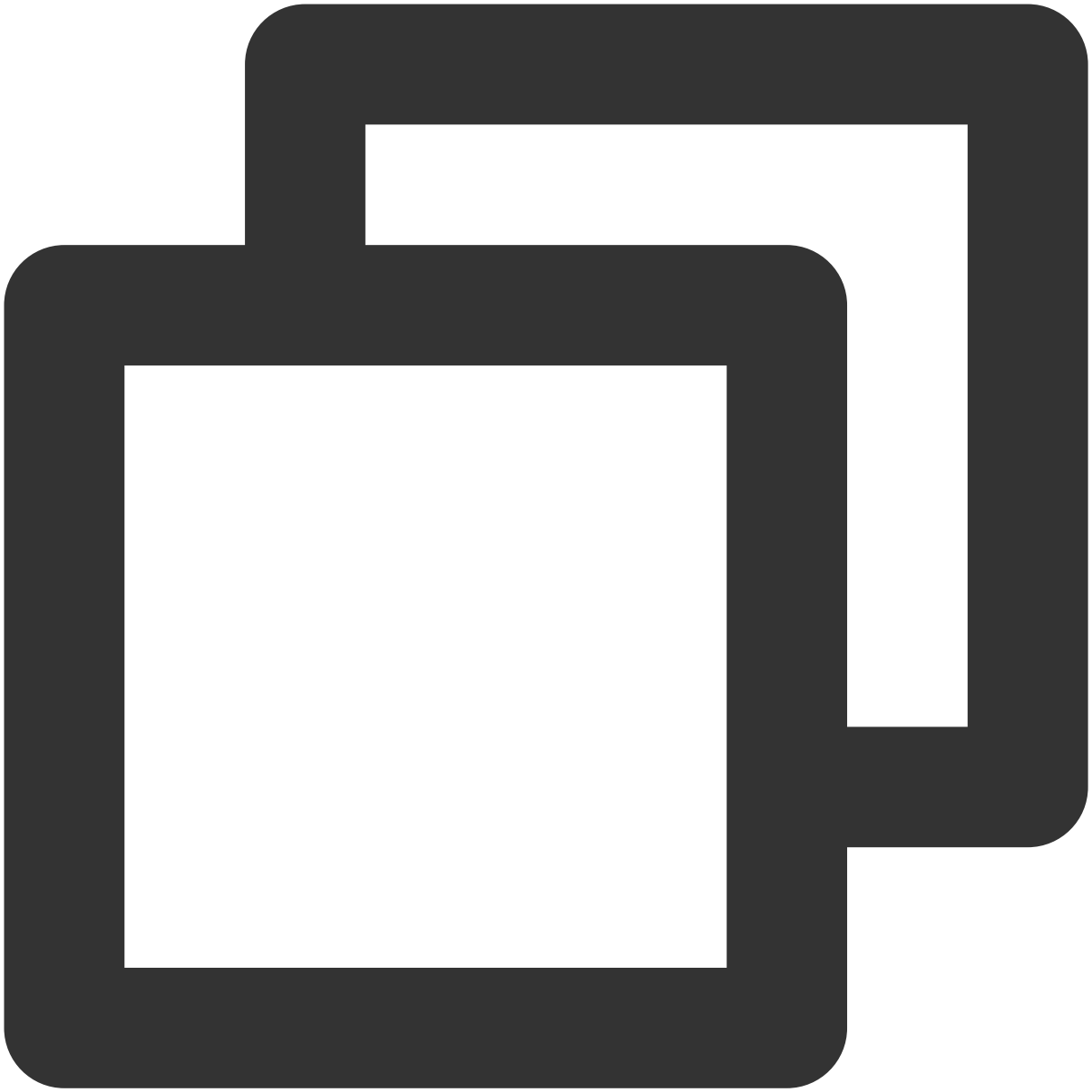


```
[[XGPush defaultManager] sdkVersion];
```

日志上报接口

接口说明

开发者如果发现推送相关功能异常，可以调用该接口，触发本地 push 日志的上报，反馈问题 提交工单 时，请将文件地址提供给我们，便于排查问题。



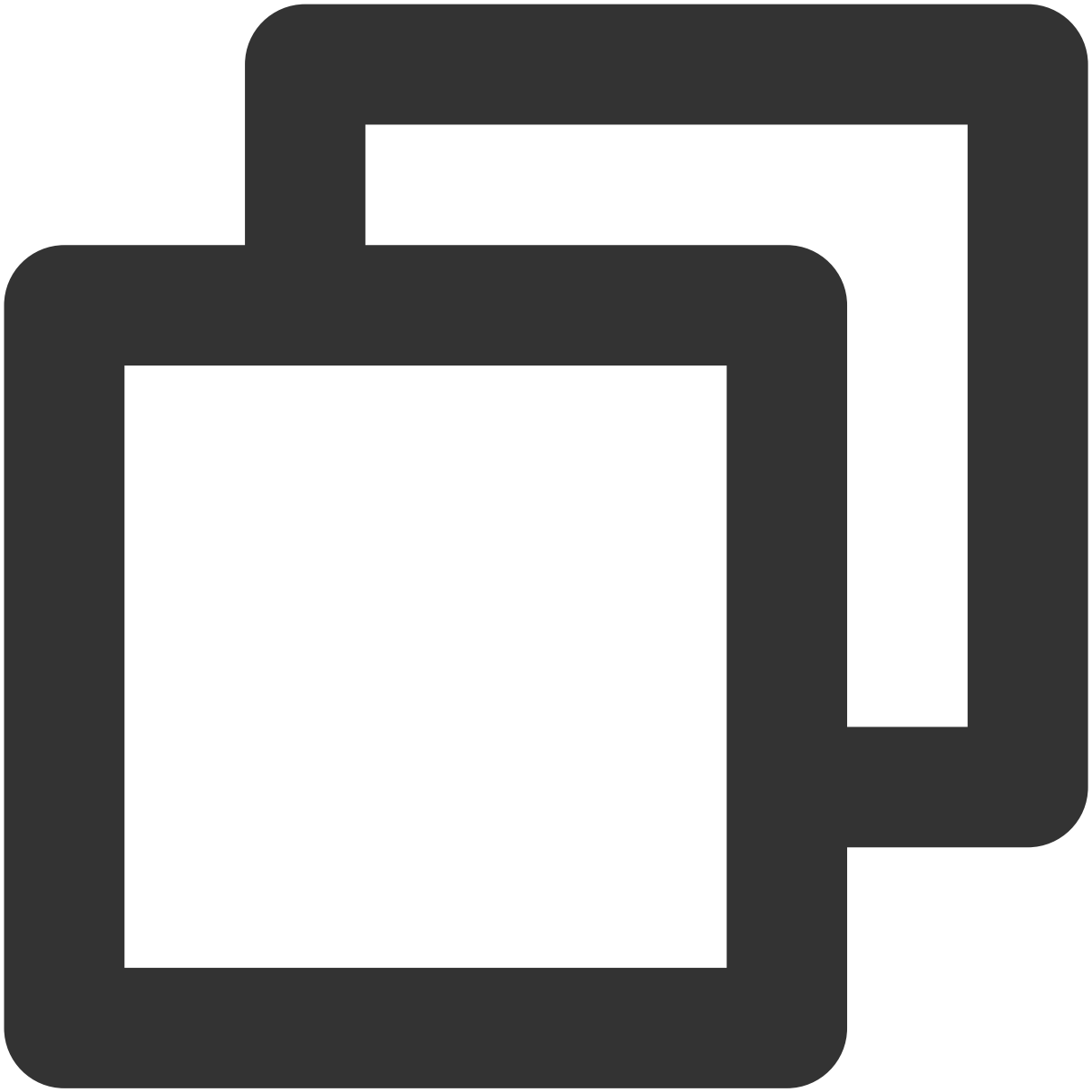
```
- (void)uploadLogCompletionHandler:(nullable void(^)(BOOL result,  NSString * _Null
```

参数说明

@brief：上报日志信息 。

@param handler：上报回调。

示例代码



```
[[XGPush defaultManager] uploadLogCompletionHandler:nil];
```

移动推送日志托管

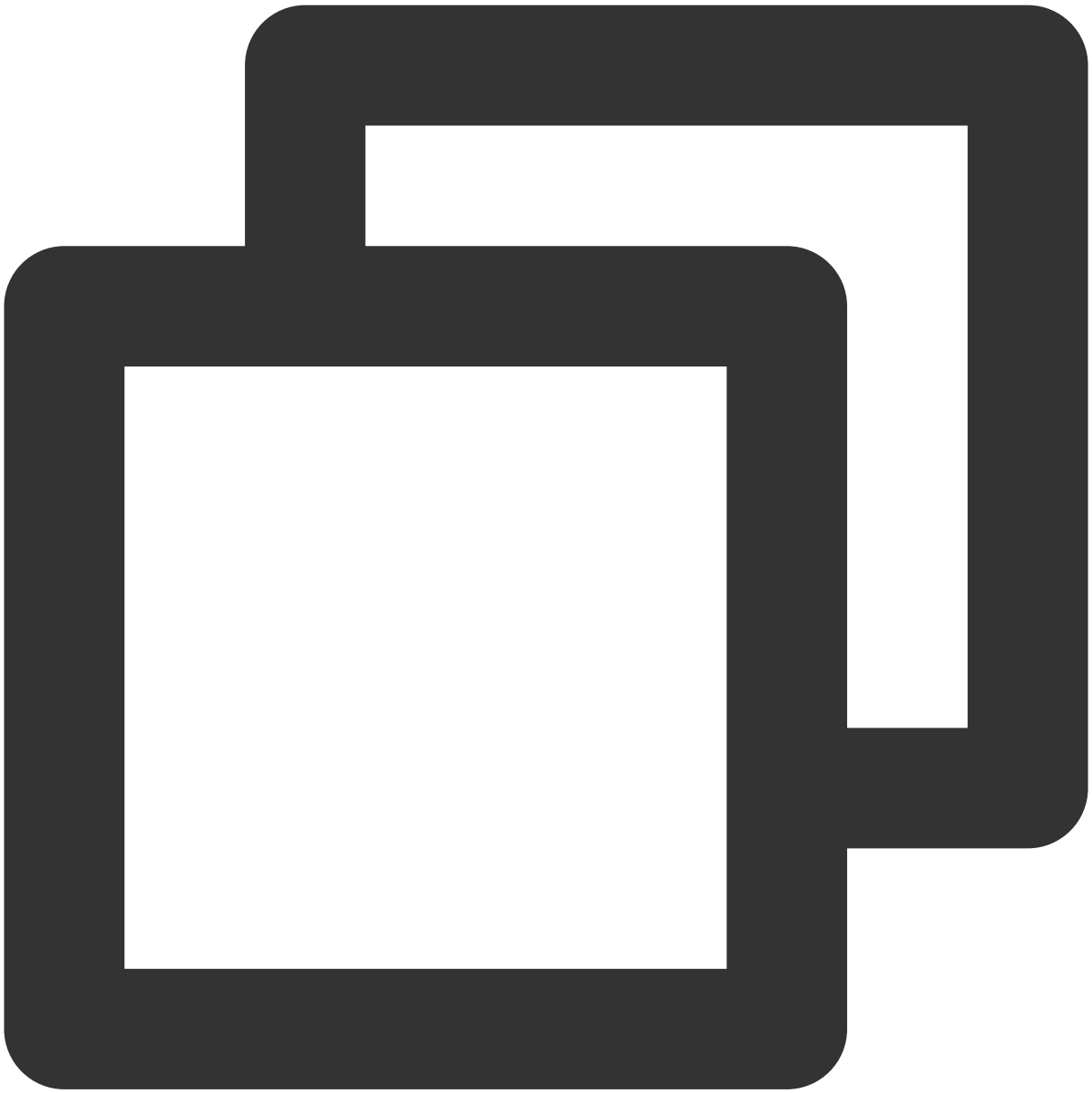
接口说明

可以在此方法获取移动推送的 log 日志。此方法和 XGPush > enableDebug 无关。

参数说明

logInfo：日志信息。

示例代码



```
- (void)xgPushLog:(nullable NSString *)logInfo;
```

本地推送

本地推送相关功能请参见 苹果开发者文档。

推送证书说明

最近更新时间：2024-01-16 17:42:20

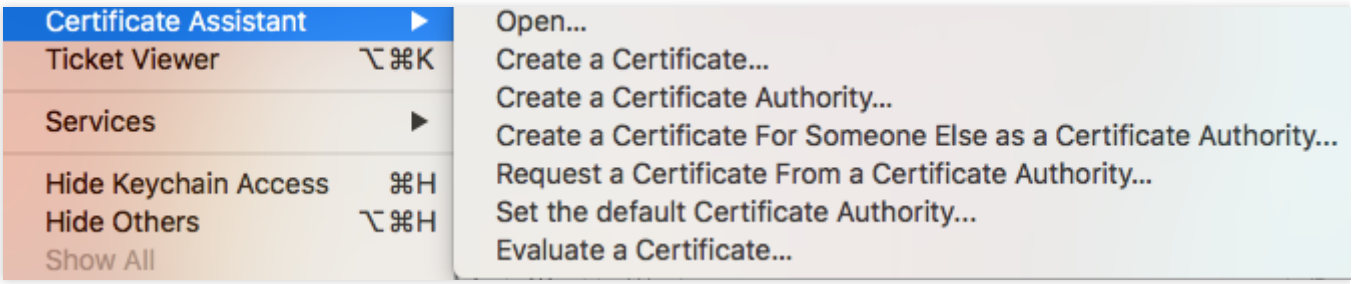
操作场景

macOS 推送证书分为开发环境的推送证书、发布环境的推送证书。本文档指导您如何制作开发环境的推送证书，发布环境操作步骤一致，在第5步选择 production 即可。

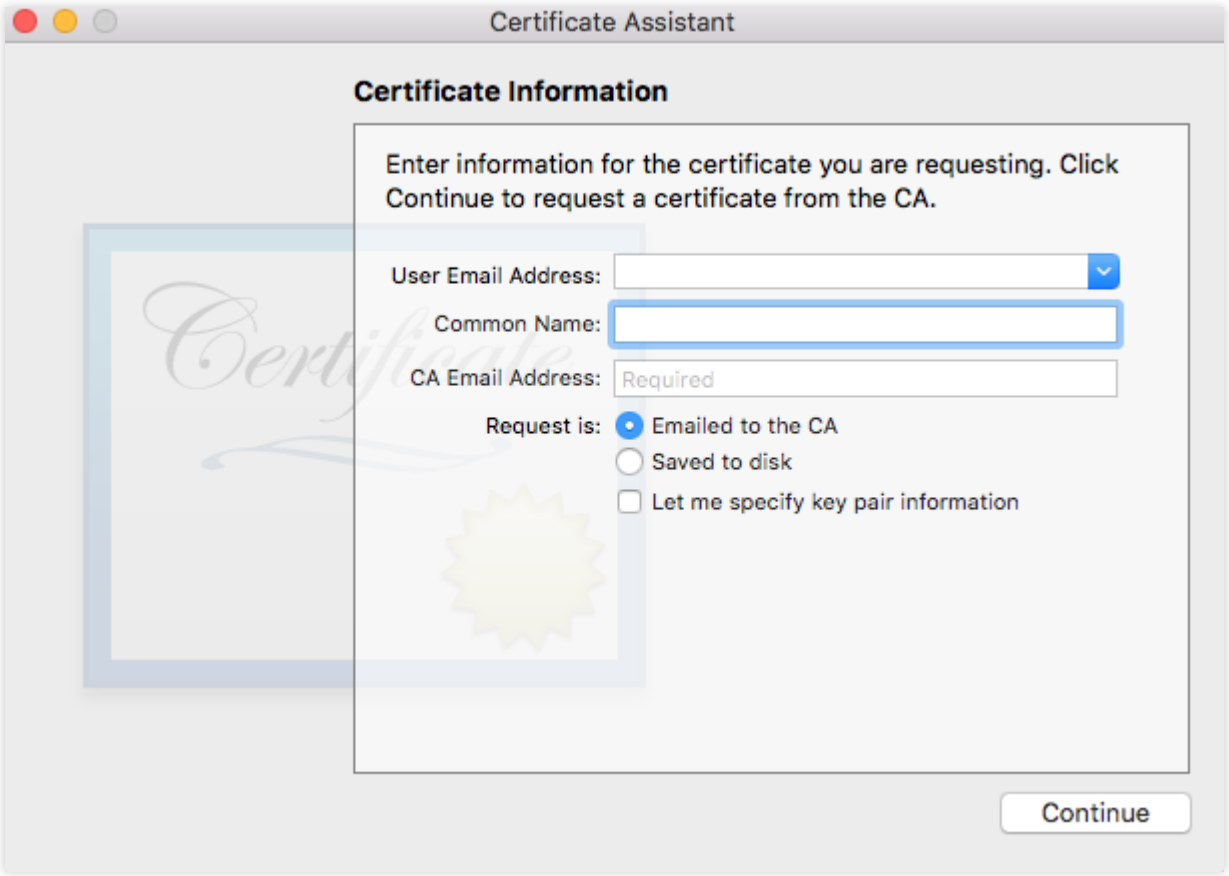
操作步骤

生成证书

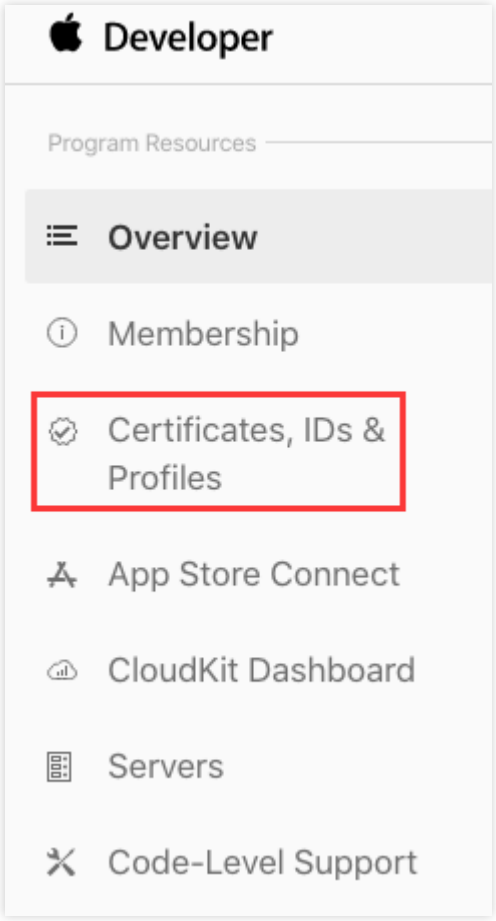
1. 在您的电脑中，打开 Keychain Access 工具，选择 **Request a Certificate From a Certificate Authority**。



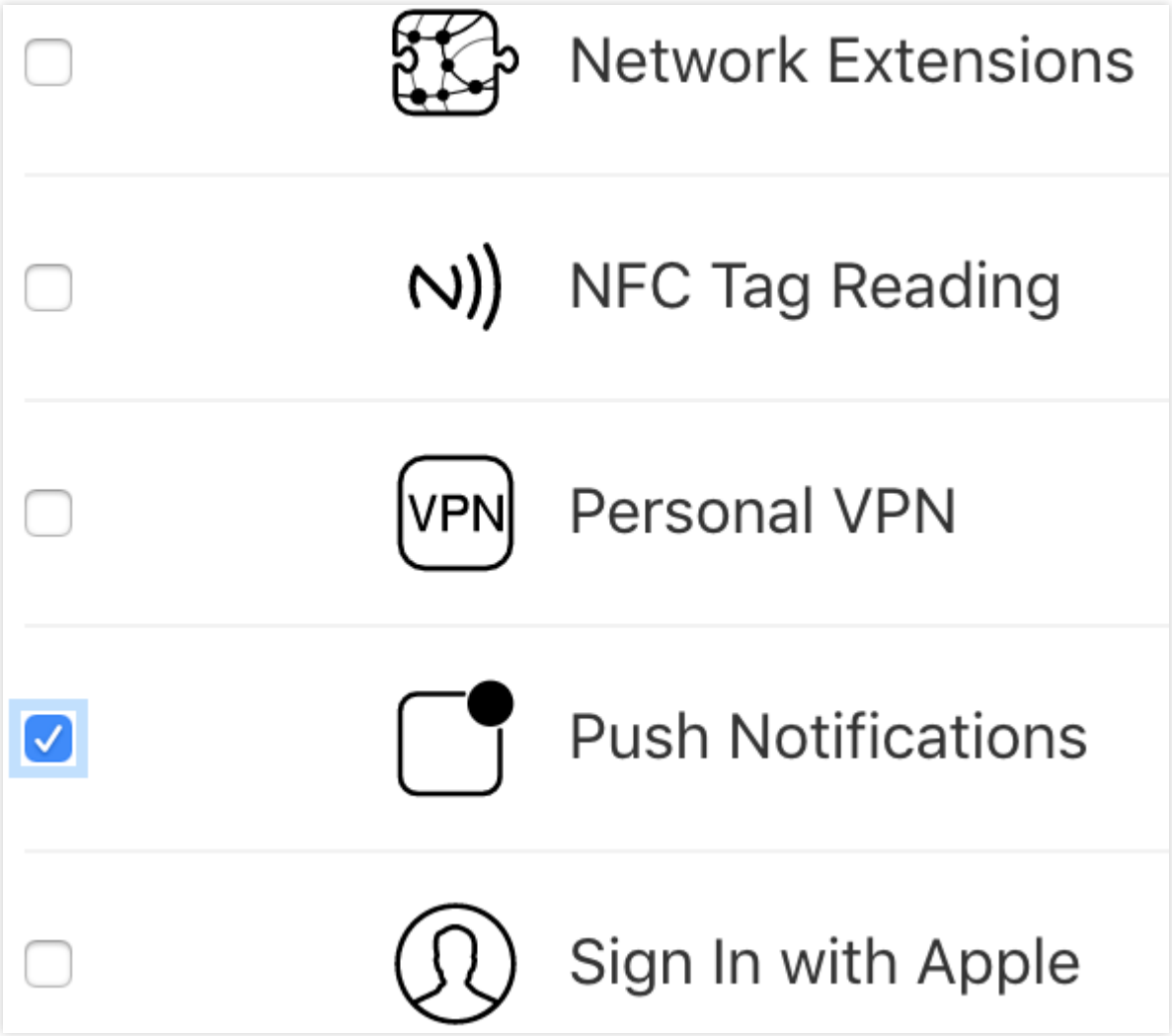
2. 填写邮件地址，其它留空，单击 **continue**，将证书保存到本地。



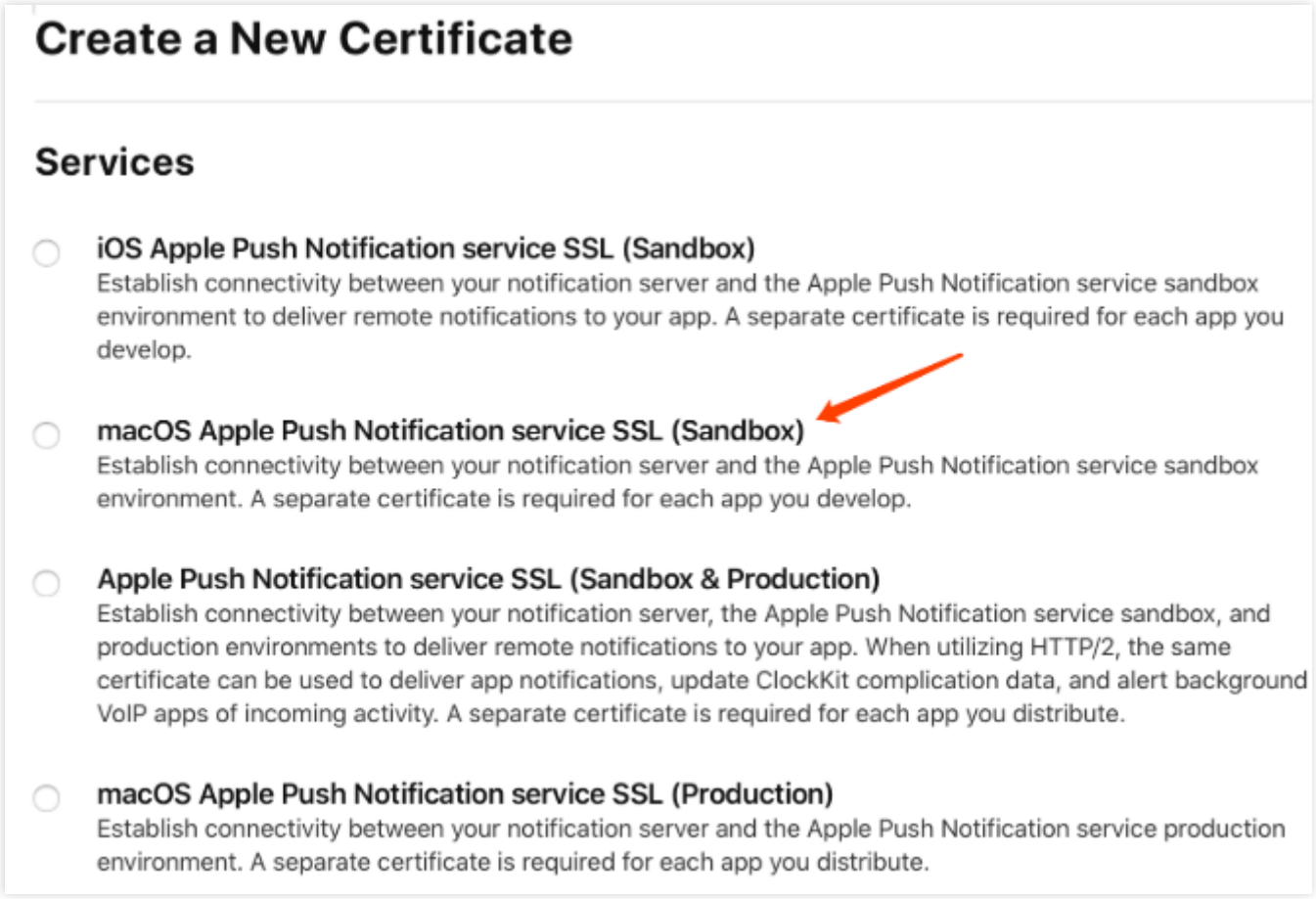
3. 登录苹果开发者中心网站，单击 **Certificates,Identifiers & Profiles**。



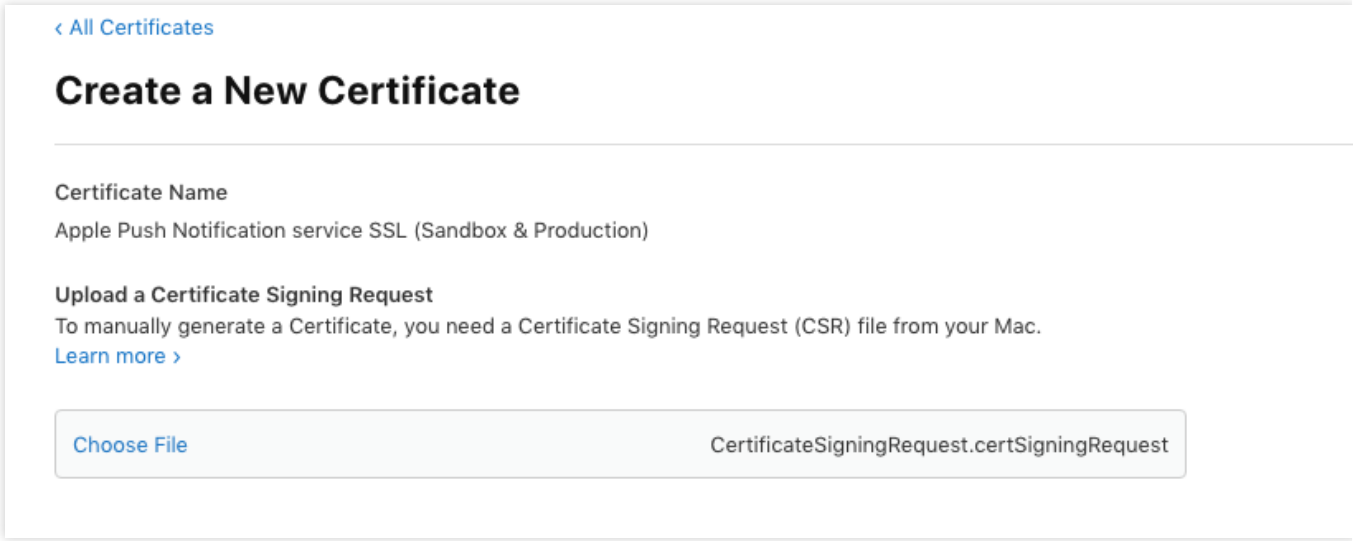
4. 选中需要制作消息推送证书的应用，勾选消息推送服务 **Push Notifications**。



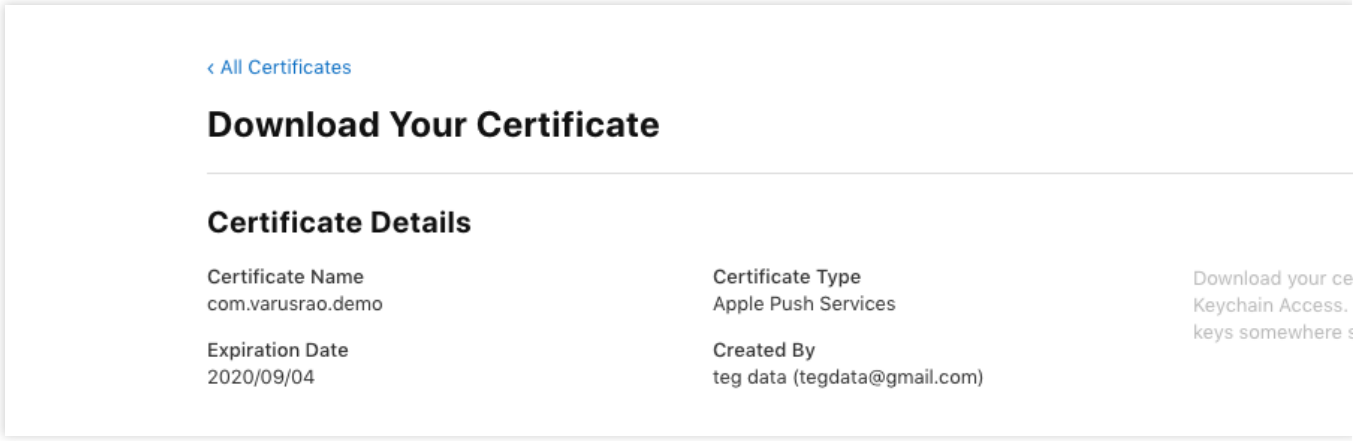
5. 勾选 **Create Certificate**，选择开发环境的推送证书。



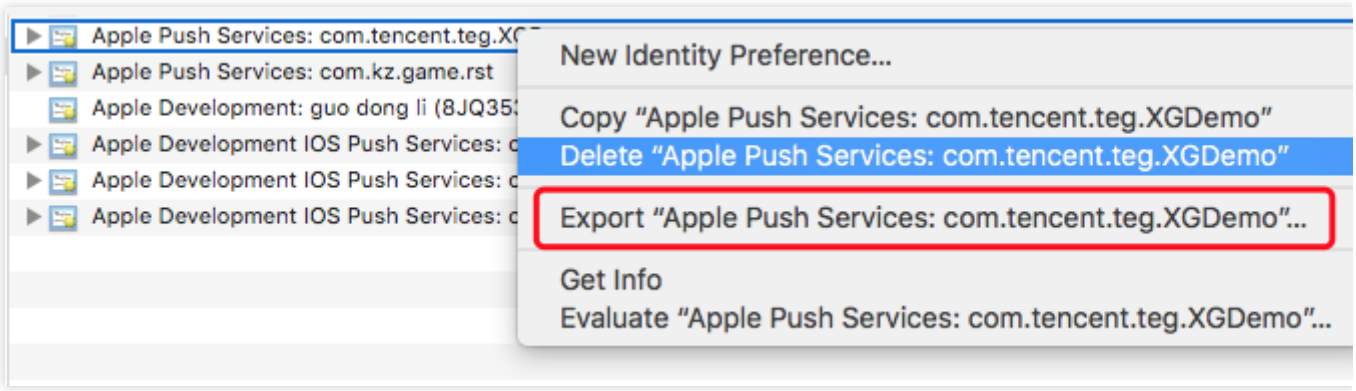
6. 选择第2步中创建的消息推送证书请求文件，上传完毕之后，单击 **continue**。



7. 将生成的消息推送证书下载到本地。



8. 双击上一步中下载的证书，将自动将消息推送证书安装到 Keychain 应用中。
9. 打开 Keychain Access， 选中需要导出的消息推送证书，右键选择导出证书，导出的格式为 P12，并设置密码。



上传证书

1. 登录 移动推送控制台，选择左侧菜单**配置管理**。
2. 进入配置管理页面，选择需要上传推送证书的应用。
3. 输入证书密码，单击上传证书】，选择您的证书即可完成上传。