Android SDK 集成指南

使用提示

本文是 JPush Android SDK 标准的集成指南文档。用以指导 SDK 的使用方法，默认读者已经熟悉 IDE（Eclipse 或者 Android Studio）的基本使用方法，以及具有一定的 Android 编程知识基础。

本篇指南匹配的 JPush Android SDK 版本为：v3.0.0 及以后版本。

- 3 分钟快速 Demo（Android）：如果您想要快速地测试、感受下极光推送的效果，请参考本文在几分钟内跑通Demo。
- 极光推送文档网站上，有极光推送相关的所有指南、API、教程等全部的文档。包括本文档的更新版本，都会及时地发布到该网站上。
- 极光社区网站：大家除了文档之外，还有问题与疑问，会到这里来提问题，以及时地得到解答。
- 如果您看到本文档，但还未下载Android SDK，请访问SDK下载页面下载。

产品功能说明

极光推送（JPush）是一个端到端的推送服务，使得服务器端消息能够及时地推送到终端用户手机上，让开发者积极地保持与用户的连接，从而提高用户活跃度、提高应用的留存率。极光推送客户端支持 Android, iOS 两个平台。

本 Android SDK 方便开发者基于 JPush 来快捷地为 Android App 增加推送功能。

主要功能

- 保持与服务器的长连接，以便消息能够即时推送到达客户端
- 接收通知与自定义消息，并向开发者App 传递相关信息

主要特点

- 客户端维持连接占用资源少、耗电低
- SDK丰富的接口，可定制通知栏提示样式
- 服务器大容量、稳定

jpush-android-release-3.x.y.zip 集成压缩包内容

- AndroidManifest.xml
  - 客户端嵌入SDK参考的配置文件

- libs/jcore-android.v1.x.y.jar
  - 极光开发者服务的核心包。
- libs/jpush-android_v3.x.y.jar
  - JPush SDK 开发包。
- libs/(cpu-type)/libjcore1xy.so
  - 各种CPU类型的native开发包。
- res
  - 集成SDK必须添加的资源文件
- example
  - 是一个完整的 Android 项目，通过这个演示了 JPush SDK 的基本用法，可以用来做参考。

Android SDK 版本

目前SDK只支持Android 2.3或以上版本的手机系统。富媒体信息流功能则需Android3.0或以上版本的系统。

jcenter 自动集成步骤

说明： 使用jcenter自动集成的开发者，不需要在项目中添加jar和so，jcenter会自动完成依赖；在AndroidManifest.xml中不需要添加任何JPush SDK 相关的配置，jcenter会自动导入。

- 如果开发者需要修改组件属性，可以在本地的 AndroidManifest 中定义同名的组件并配置想要的属性，然后用 xmlns:tools 来控制本地组件覆盖 jcenter 上的组件。示例：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.tests.flavorlib.app"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:icon="@drawable/icon"
        android:name="com.example.jpushdemo.ExampleApplication"
        android:label="@string/app_name" >

        <service android:name="cn.jpush.android.service.PushService"
            android:process=":multiprocess"
            tools:node="replace" >

            ……
        </service>

    ……
  </application>

  ……
</manifest>
```

- 确认android studio的 Project 根目录的主 gradle 中配置了jcenter支持。（新建project默认配置就支持）

```
buildscript {
    repositories {
        jcenter()
    }
    ......
}

allprojets {
    repositories {
        jcenter()
    }
}
```

- 在 module 的 gradle 中添加依赖和AndroidManifest的替换变量。

```
android {
    ......
    defaultConfig {
        applicationId "com.xxx.xxx" //JPush上注册的包名.
        ......

        ndk {
            //选择要添加的对应cpu类型的.so库。
            abiFilters 'armeabi', 'armeabi-v7a', 'armeabi-v8a'
            // 还可以添加 'x86', 'x86_64', 'mips', 'mips64'
        }

        manifestPlaceholders = [
            JPUSH_PKGNAME : applicationId,
            JPUSH_APPKEY : "你的appkey", //JPush上注册的包名对应的appkey.
            JPUSH_CHANNEL : "developer-default", //暂时填写默认值即可.
        ]
        ......
    }
    ......
}

dependencies {
    ......

    compile 'cn.jiguang.sdk:jpush:3.0.3'  // 此处以JPush 3.0.3 版本为例。
    compile 'cn.jiguang.sdk:jcore:1.1.1'  // 此处以JCore 1.1.1 版本为例。
    ......
}
```

注：如果在添加以上 abiFilter 配置之后android Studio出现以下提示：

```
    NDK integration is deprecated in the current plugin. Consider trying the new e
xperimental plugin.
```

则在 Project 根目录的gradle.properties文件中添加：

```
    android.useDeprecatedNdk=true。
```

说明：若没有res/drawable-xxxx/jpush*notification*icon这个资源默认使用应用图标作为通知icon，在5.0以上系统将应用图标作为statusbar icon可能显示不正常，用户可定义没有阴影和渐变色的icon替换这个文件，文件名不要变。

手动集成步骤

- 解压缩 jpush-android-release-3.x.y.zip 集成压缩包。
- 复制 libs/jcore-android_v1.x.y.jar 到工程 libs/ 目录下。
- 复制 libs/jpush-android_v3.x.y.jar 到工程 libs/ 目录下。
- 复制 libs/(cpu-type)/libjcore1xy.so 到你的工程中存放对应cpu类型的目录下。
- 复制 res/ 中drawable-hdpi, layout, values文件夹中的资源文件到你的工程中 res/ 对应同名的目录下。

说明 1：若没有res/drawable-xxxx/jpush*notification*icon这个资源默认使用应用图标作为通知icon，在5.0以上系统将应用图标作为statusbar icon可能显示不正常，用户可定义没有阴影和渐变色的icon替换这个文件，文件名不要变。

说明 2：使用android studio的开发者，如果使用jniLibs文件夹导入so文件，则仅需将所有cpu类型的文件夹拷进去；如果将so文件添加在module的libs文件夹下，注意在module的gradle配置中添加一下配置：

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

配置 AndroidManifest.xml

根据 SDK 压缩包里的 AndroidManifest.xml 样例文件，来配置应用程序项目的 AndroidManifest.xml 。

主要步骤为：

- 复制备注为 "Required" 的部分
- 将标注为“您应用的包名”的部分，替换为当前应用程序的包名
- 将标注为“您应用的Appkey”的部分，替换为在Portal上注册该应用的的Key,例如：9fed5bcb7b9b87413678c407

小帖士

如果使用android studio, 可在AndroidManifest中引用applicationId的值，在build.gradle配置中 defaultConfig 节点下配置，如：

```
defaultConfig {
      applicationId "cn.jpush.example" // <--您应用的包名
      ……
 }
```

在AndroidManifest中使用 ${applicationId} 引用gradle中定义的包名

AndroidManifest 示例

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="您应用的包名"
    android:versionCode="303"
    android:versionName="3.0.3"
    >
    <uses-sdk android:minSdkVersion="9" android:targetSdkVersion="23" />

    <!-- Required -->
    <permission
        android:name="您应用的包名.permission.JPUSH_MESSAGE"
        android:protectionLevel="signature" />

    <!-- Required -->
    <uses-permission android:name="您应用的包名.permission.JPUSH_MESSAGE" />
    <uses-permission android:name="android.permission.RECEIVE_USER_PRESENT" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" /
>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_SETTINGS" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

    <!-- Optional. Required for location feature -->
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" /> <!--
 用于开启 debug 版本的应用在6.0 系统上 层叠窗口权限 -->
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMAN
DS" />
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
    <uses-permission android:name="android.permission.GET_TASKS" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:name="Your Application Name">
```

```
        <!-- Required SDK 核心功能-->
        <!-- 可配置android:process参数将PushService放在其他进程中 -->
        <service
            android:name="cn.jpush.android.service.PushService"
            android:enabled="true"
            android:exported="false" >
            <intent-filter>
                <action android:name="cn.jpush.android.intent.REGISTER" />
                <action android:name="cn.jpush.android.intent.REPORT" />
                <action android:name="cn.jpush.android.intent.PushService" />
                <action android:name="cn.jpush.android.intent.PUSH_TIME" />
            </intent-filter>
        </service>

        <!-- since 1.8.0 option 可选项。用于同一设备中不同应用的JPush服务相互拉起的功能。 -
->
        <!-- 若不启用该功能可删除该组件，将不拉起其他应用也不能被其他应用拉起 -->
         <service
             android:name="cn.jpush.android.service.DaemonService"
             android:enabled="true"
             android:exported="true">
             <intent-filter >
                 <action android:name="cn.jpush.android.intent.DaemonService" />
                 <category android:name="您应用的包名"/>
             </intent-filter>
         </service>

        <!-- Required SDK核心功能-->
        <receiver
            android:name="cn.jpush.android.service.PushReceiver"
            android:enabled="true" >
          <intent-filter android:priority="1000">
                <action android:name="cn.jpush.android.intent.NOTIFICATION_RECEIVE
D_PROXY" />
                <category android:name="您应用的包名"/>
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.USER_PRESENT" />
                <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
            </intent-filter>
            <!-- Optional -->
            <intent-filter>
                <action android:name="android.intent.action.PACKAGE_ADDED" />
                <action android:name="android.intent.action.PACKAGE_REMOVED" />
                <data android:scheme="package" />
            </intent-filter>
        </receiver>
```

```
        <!-- Required SDK核心功能-->
        <activity
            android:name="cn.jpush.android.ui.PushActivity"
            android:configChanges="orientation|keyboardHidden"
            android:theme="@android:style/Theme.NoTitleBar"
            android:exported="false" >
            <intent-filter>
                <action android:name="cn.jpush.android.ui.PushActivity" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="您应用的包名" />
            </intent-filter>
        </activity>
        <!-- SDK核心功能-->
        <activity
            android:name="cn.jpush.android.ui.PopWinActivity"
            android:configChanges="orientation|keyboardHidden"
            android:exported="false"
            android:theme="@style/MyDialogStyle">
            <intent-filter>
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="您应用的包名" />
            </intent-filter>
        </activity>

        <!-- Required SDK核心功能-->
        <service
            android:name="cn.jpush.android.service.DownloadService"
            android:enabled="true"
            android:exported="false" >
        </service>

        <!-- Required SDK核心功能-->
        <receiver android:name="cn.jpush.android.service.AlarmReceiver" />

        <!-- User defined. 用户自定义的广播接收器-->
         <receiver
             android:name="您自己定义的Receiver"
             android:enabled="true">
             <intent-filter>
                 <!--Required 用户注册SDK的intent-->
                 <action android:name="cn.jpush.android.intent.REGISTRATION" />
                 <!--Required 用户接收SDK消息的intent-->
                 <action android:name="cn.jpush.android.intent.MESSAGE_RECEIVED" /
>
                 <!--Required 用户接收SDK通知栏信息的intent-->
                 <action android:name="cn.jpush.android.intent.NOTIFICATION_RECEIV
ED" />
```

```
                <!--Required 用户打开自定义通知栏的intent-->
                <action android:name="cn.jpush.android.intent.NOTIFICATION_OPENED
" />
                <!-- 接收网络变化 连接/断开 since 1.6.3 -->
                <action android:name="cn.jpush.android.intent.CONNECTION" />
                <category android:name="您应用的包名" />
            </intent-filter>
         </receiver>

        <!-- Required. For publish channel feature -->
        <!-- JPUSH_CHANNEL 是为了方便开发者统计APK分发渠道。-->
        <!-- 例如: -->
        <!-- 发到 Google Play 的APK可以设置为 google-play; -->
        <!-- 发到其他市场的 APK 可以设置为 xxx-market。 -->
        <!-- 目前这个渠道统计功能的报表还未开放。-->
        <meta-data android:name="JPUSH_CHANNEL" android:value="developer-default"/
>
        <!-- Required. AppKey copied from Portal -->
        <meta-data android:name="JPUSH_APPKEY" android:value="您应用的Appkey"/>
    </application>
</manifest>
```

配置和代码说明

必须权限说明

权限	用途
You Package.permission.JPUSH_MESSAGE	官方定义的权限，允许应用接收JPUSH内部代码发送的广播消息。
RECEIVE_USER_PRESENT	允许应用可以接收点亮屏幕或解锁广播。
INTERNET	允许应用可以访问网络。
WAKE_LOCK	允许应用在手机屏幕关闭后后台进程仍然运行
READ_PHONE_STATE	允许应用访问手机状态。
WRITE_EXTERNAL_STORAGE	允许应用写入外部存储。
READ_EXTERNAL_STORAGE	允许应用读取外部存储。
WRITE_SETTINGS	允许应用读写系统设置项。
VIBRATE	允许应用震动。
MOUNT_UNMOUNT_FILESYSTEMS	允许应用挂载/卸载 外部文件系统。
ACCESS_NETWORK_STATE	允许应用获取网络信息状态，如当前的网络连接是否有效。

集成 JPush Android SDK 的混淆

- 请下载4.x及以上版本的proguard.jar， 并替换你Android Sdk "tools\proguard\lib\proguard.jar"
- 请在工程的混淆文件中添加以下配置：

```
-dontoptimize
-dontpreverify

-dontwarn cn.jpush.**
-keep class cn.jpush.** { *; }

-dontwarn cn.jiguang.**
-keep class cn.jiguang.** { *; }
```

- v2.0.5 ~ v2.1.7 版本有引入 gson 和 protobuf ， 增加排除混淆的配置。(2.1.8版本不需配置)

```
#==================gson && protobuf==========================
-dontwarn com.google.**
-keep class com.google.gson.** {*;}
-keep class com.google.protobuf.** {*;}
```

添加代码

JPush SDK 提供的 API 接口，都主要集中在 cn.jpush.android.api.JPushInterface 类里。

基础API

- init 初始化SDK

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

- setDebugMode 设置调试模式

  注：该接口需在init接口之前调用，避免出现部分日志没打印的情况。多进程情况下建议在自定义的Application中onCreate中调用。

```
// You can enable debug mode in developing state. You should close debug mode
when release.
public static void setDebugMode(boolean debugEnalbed)
```

添加统计代码

- 参考文档： 统计分析 API

调用示例代码（参考 **example** 项目）

- init 只需要在应用程序启动时调用一次该 API 即可。
- 以下代码定制一个本应用程序 Application 类。需要在 AndoridManifest.xml 里配置。请参考上面 AndroidManifest.xml 片断，或者 example 项目。

```
public class ExampleApplication extends Application {
@Override
    public void onCreate() {
        super.onCreate();
        JPushInterface.setDebugMode(true);
        JPushInterface.init(this);
    }
}
```

测试确认

- 确认所需的权限都已经添加。如果必须的权限未添加，日志会提示错误。
- 确认 AppKey（在Portal上生成的）已经正确的写入 Androidmanifest.xml 。
- 确认在程序启动时候调用了init(context) 接口

- 确认测试手机（或者模拟器）已成功连入网络 + 客户端调用 init 后不久，如果一切正常，应有登录成功的日志信息
- 启动应用程序，在 Portal 上向应用程序发送自定义消息或者通知栏提示。详情请参考管理Portal。
    - 在几秒内，客户端应可收到下发的通知或者正定义消息，如果 SDK 工作正常，则日志信息会如下：

```
[JPushInterface] action:init

.......

[PushService] Login succeed!
```

如图所示，客户端启动分为 4 步：

- 检查 metadata 的 appKey 和 channel ，如果不存在，则启动失败
- 初始化 JPush SDK，检查 JNI 等库文件的有效性，如果库文件无效，则启动失败
- 检查 Androidmanifest.xml，如果有 Required 的权限不存在，则启动失败
- 连接服务器登录，如果存在网络问题，则登陆失败,或者前面三步有问题，不会启动JPush SDK

高级功能

请参考：

API: Android

技术支持

邮件联系：support@jpush.cn

问答社区：极光社区