XPush项目

**消息推送客户端用户手册**

版本<1.3>

修订文档历史记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | 日期 | 文档版本 | 作者 | 编写/修订说明 |
| 1 | 2014.12.26 | V1.0 | 魏巍 | 初稿 |
| 2 | 2015.2.5 | V1.1 | 丁锐 | 加入集成说明 |
| 3 | 2015.6.8 | V1.2 | 丁锐 | 删除冗余的接口 |
| 4 | 2017.2.15 | V1.3 | 余慧君 | 新增接口 |

目录

1. 简介 4
   1. 目的 4
   2. 范围 4
   3. 定义、首字母缩写词和缩略语 4
   4. 参考资料 4
   5. 主要需求 4
2. SDK功能说明 4
   1. 框架设计 4
   2. 主要功能 5
3. 开发前准备 5
4. 客户端集成说明 5
   1. 使用 SDK开发应用 5
      1. 添加 SDK 到APP工程 5
      2. 调用API 8
5. 客户端API接口说明 11
   1. 类说明 11
   2. 接口说明 11
      1. Push服务初始化 12
      2. Push服务停止 12
      3. 判断Push服务是否运行 12
      4. 设置默认推送通知栏样式 12
      5. 统计分析api 13
      6. PushReceiver注册推送服务回调 13
      7. PushReceiver取消注册推送服务回调 13
      8. PushReceiver获取推送消息回调 13
      9. PushReceiver点击消息通知栏回调 14
      10. 客制化心跳频率和push服务重连次数 14

消息推送客户端用户手册

# 简介

## 目的

XPush SDK是科大讯飞官方推出的推送服务的Android平台开发SDK，提供给Android开发者简单的接口，轻松集成讯飞推送服务。

XPush服务以后台service方式运行。如果一款手机安装了多个集成了XPush SDK的应用，不会每个应用都开启一个后台 service，而是只有一个service实例运行，采用多个应用共享一个Push通道的方式。这样的设计能够减少手机系统运行的进程数，减少内存使用，降低功耗，同时一个通道能减少网络流量开销。

Push Service运行于一个独立进程，不和主进程运行于同一进程，主程序不需要常驻内存，当Push service接受到Push消息后，会通过Intent接口发送给主程序处理。

Push Android SDK的完整下载包为ifly\_push\_sdk\_VERSION.jar，VERSION是版本号，如1.0.1015。

## 范围

本文档面向所有使用该 SDK的Android开发人员、测试人员、合作伙伴以及对此感兴趣的其他用户。

## 定义、首字母缩写词和缩略语

* **XPuchclient**  讯飞推送sdk
* **Android** 谷歌公司开发的移动终端操作系统
* **SDK** 软件开发包 (Software Development Kit)
* **API** 应用编程接口 (Application Programming Interface)

## 参考资料

## 主要需求

1. app登录
2. 客户端与服务端保持长连接
3. 获取服务端推送消息
4. 客户端多进程共享一个推送长连接
5. 客户端支持消息类型通知栏显示和透传
6. 记录并上传消息推送日志

# SDK功能说明

## 框架设计

XPush Android SDK是开发者与Push服务器之间的桥梁。可以让用户越过复杂的Push HTTP/HTTPS API，直接和Push 服务器进行交互来使用Push服务。（框架设计如图 1 所示）

用户应用

Push SDK

Push HTTP/HTTPS API

Push服务器

图 1 Push SDK框架图

## 主要功能

本SDK主要提供以下功能的接口：

1. Push服务

- Push 服务初始化及绑定

- Push 服务停止和恢复

1. 通知推送

接收和展现通知，还提供自定义通知栏样式的功能，包括：

- 设置自定义通知的Builder

- 设置默认的通知的Builder

1. 推送效果反馈

- 通知的点击或删除情况统计

- 应用的使用情况统计

# 开发前准备

开发者需到语音云服务器注册开发者账户，获取应用appid和token

# 客户端集成说明

## 使用 SDK开发应用

### 添加 SDK 到APP工程

1. 创建一个Android Project
2. 在该工程下创建一个libs文件夹
3. 将ifly\_push\_sdk\_VERSION.jar拷贝至刚刚创建的libs目录下
4. 将上述jar包添加到工程的java build path
5. AndroidManifest.xml声明permission

|  |
| --- |
| <!-- Push service 运行需要的权限 --/>  <uses-permission android:name="android.permission.INTERNET" />  <uses-permission android:name="android.permission.READ\_PHONE\_STATE" />  <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />  <uses-permission android:name="android.permission.RECEIVE\_BOOT\_COMPLETED" />  <uses-permission android:name="android.permission.WRITE\_SETTINGS" />  <uses-permission android:name="android.permission.VIBRATE" />  <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />  <uses-permission android:name="android.permission.ACCESS\_DOWNLOAD\_MANAGER" />  <uses-permission android:name="android.permission.DOWNLOAD\_WITHOUT\_NOTIFICATION" />  <uses-permission android:name="android.permission.DISABLE\_KEYGUARD" />  <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />  <uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" /> |

1. AndroidManifest.xml增加 pushservice配置

|  |
| --- |
| <!-- push service start -->  <!-- Push 服务 -->  <service  android:name="com.iflytek.pushclient.manager.PushService"  android:exported="true"  android:process=":xpush"></service>  <!-- 用于接收系统消息以保证PushService正常运行 -->  <receiver  android:name="com.iflytek.pushclient.manager.PushServiceReceiver"  android:process=":xpush">  <intent-filter>  <action android:name="android.intent.action.BOOT\_COMPLETED" />  <action android:name="android.net.conn.CONNECTIVITY\_CHANGE" />  <action android:name="com.iflytek.pushclient.action.notification.SHOW" />  </intent-filter>  </receiver>  <!-- Push服务接收客户端发送的各种请求-->  <receiver  android:name="com.iflytek.pushclient.manager.RegistrationReceiver"  android:process=":xpush">  <intent-filter>  <action android:name="com.iflytek.pushclient.action.METHOD" />  <action android:name="com.iflytek.pushclient.action.BIND\_SYNC" />  </intent-filter>  <intent-filter>  <action android:name="android.intent.action.PACKAGE\_REMOVED" />  <data android:scheme="package" />  </intent-filter>  </receiver>  <!--您在讯飞语音云平台申请的appkey-->  <meta-data android:name="IFLYTEK\_APPKEY" android:value="您应用的Appkey"/>  <!--app对应的渠道号(非必填)-->  <meta-data android:name="IFLYTEK\_CHANNEL" android:value="Android\_Demo"/>  <!-- push service end --> |

1. AndroidManifest.xml注册消息接收 receiver

客户端需实现自己的 MyPushMessageReceiver，接收Push服务的消息，并实现对消息的处理

以下是AndroidManifest.xml中的配置代码。

|  |
| --- |
| <receiver android:name="com.iflytek.pushclient.test.MyPushReceiver">  <intent-filter>  <!-- 接收 push消息 -->  <action android:name="com.iflytek.pushclient.action.MESSAGE" />  <!-- 接收 bind、unbind等method 的返回结果 -->  <action android:name="com.iflytek.pushclient.action.RECEIVE" />  <!-- 可选。接受通知点击事件,删除事件和通知自定义内容 -->  <action android:name="com.iflytek.pushclient.action.notification.CLICK" />  <action android:name="com.iflytek.pushclient.action.notification.DELETE" />  </intent-filter>  </receiver> |

### 调用API

下面介绍如何调用 SDK中已封装的API完成各项操作：

1. 在主Activiy的OnCreate方法中，调用接口 startWork

|  |
| --- |
| PushManager.startWork(context, appId) |

1. 自定义通知样式（可选）

这是通知推送的高级功能，对于很多应用来说，使用系统默认的通知栏样式就足够了。

SDK 提供了2个用于定制通知栏样式的构建类（PushNotificationBuilder是两者的基类）：

■ BasicPushNotificationBuilder

用于定制 Android Notification 里的 defaults / flags / icon 等基础样式（行为）

■ CustomPushNotificationBuilder

除了让开发者定制BasicPushNotificationBuilder中的基础样式， 让开发者进一步定制

Notification Layout

当开发者需要为不同的通知指定不同的通知栏样式（行为）时，则需要调用

PushManager.setNotificationBuilder设置多个通知栏构建类。

设置时，开发者自己维护 notificationBuilderId 这个编号，下发通知时使用

notification\_builder\_id 指定该编号，从而 SDK 会调用开发者应用程序里设置过的指定编号的通

知栏构建类，来定制通知栏样式。如果通过管理控制台来推送通知，请在 高级设置 的 自定义

目前不支持notificationBuilderId，请将notificationBuilderId设为0

样式 栏中指定编号。

这里以CustomPushNotificationBuilder 为例，代码如下

|  |
| --- |
| CustomPushNotificationBuilder cBuilder = new CustomPushNotificationBuilder(layoutId,  layoutIconId, layoutTitleId, layoutTextId);  cBuilder.setNotificationFlags(Notification.FLAG\_AUTO\_CANCEL);  cBuilder.setNotificationDefaults(Notification.DEFAULT\_SOUND  |Notification.DEFAULT\_VIBRATE);  cBuilder.setStatusbarIcon(statusbarIconId);  cBuilder.setLayoutDrawable(notificationIconId);  PushManager.setDefaultNotificationBuilder(this, cBuilder); |

1. 消息接收器

客户端程序需要自己实现一个 BroadcastReceiver 来接收 Push 消息、接口调用回调以及通知点击

事件也就是上文提到的AndrodManifest.xml中注册的receiver：

your.package.MyPushMessageReceiver

现在支持一种方式，开发者只需要选继承PushReceiver，实现其中的抽象方法即可

|  |
| --- |
| public class MyPushReceiver extends PushReceiver {  private static final String TAG = "PushClient\_MyPushReceiver";  /\*\*  \* 调用PushManager.startWork后，sdk将对 push server发起绑定请求，这个过程是异步  \* 的。绑定请求的结果通过onBind返回。  \*/  @Override  protected void onBind(Context context, String did, String appId, int errorCode) {  Log.d(TAG, "onBind | did = " + did + ",appId = " + appId + ", errorCode = " + errorCode);  }  /\*\*  \* 调用PushManager.stopWork解绑回调，sdk将对该应用进行解绑，该应用的消息将被丢弃  \* @param context  \* @param did  \* @param appId  \* @param errorCode  \*/  @Override  protected void onUnBind(Context context, String did, String appId, int errorCode) {  Log.d(TAG, "onUnBind | did = " + did + ",appId = " + appId + ", errorCode = " + errorCode);  }  /\*\*  \* 接收透传消息的函数。  \*  \* @param context  \* @param msgId 消息的id  \* @param content 透传消息的内容，由各应用自己解析  \*/  @Override  protected void onMessage(Context context, String msgId, byte[] content) {  Log.d(TAG, "onMessage | msgId = " + msgId + ", content = " + new String(content));  }  /\*\*  \* 接收通知点击的函数。注：推送通知被用户点击前，应用无法通过接口获取通知的内容。  \*  \* @param context  \* @param messageId  \* @param title  \* @param content  \* @param extraContent  \*/  @Override  protected void onClickNotification(Context context, String messageId, String title, String content, String extraContent) {  }  } |

1. 混淆打包说明

如果开发者需要混淆自己的 APK，请在混淆文件（一般默认为Android工程下proguard-project.txt或

者proguard.cfg）中添加如下说明(VERSION 为版本名称，ifly\_push\_sdk\_VERSION.jar 为集成的jar 包名

字)：

|  |
| --- |
| -libraryjars libs/ifly\_push\_sdk\_VERSION.jar  -dontwarn com.iflytek.pushclient.\*\*  -dontwarn com.iflytek.psunflower.\*\*  -dontwarn com.google.protobuf.\*\* |

# 客户端API接口说明

## 类说明

客户端SDK对外提供的类如下：

|  |  |
| --- | --- |
| 类名 | 描述 |
| PushManager | 调用消息推送服务的封装类，提供静态方法供外部调用 |
| PushReceiver | SDK广播消息接收器，外部可以继承该类，接收推送服务返回的消息 |
| PushMessage | 推送消息数据结构封装 |
| PushNotificationBuilder | SDK通知栏管理，可以定制通知icon/flags等 |
| PushConstants | 消息推送SDK对外提供常量定义 |

## 接口说明

本SDK支持如下类型接口：

|  |  |  |
| --- | --- | --- |
| 分类 | 功能 | API函数原型 |
| 推送服务接口 | 提供推送服务能力 | startWork,stopWork,isPushEnabled |
| 通知管理接口 | 自定义通知图标、标题等样式 | setNotificationBuilder |
| 推送消息接收接口 | 接收推送消息和各种状态数据 | onBind,onUnBind,onMessage, onClickNotification |

接口具体描述：

### Push服务初始化

|  |
| --- |
| public static void startWork(Context context) |

功能：完成push服务初始化，sdk内部会和后端交互实现应用信息注册操作,appId通过读取AndroidManifest中获取

参数：context，应用的app context上下文

结果返回：异步返回数据，通过广播方式返回服务注册结果

### Push服务停止

|  |
| --- |
| public static void stopWork(Context context) |

功能：停止push服务，sdk内部会停止推送服务，取消应用信息注册

参数：context，应用的app context上下文

结果返回：异步返回数据，通过广播方式返回服务注册结果

### 判断Push服务是否运行

|  |
| --- |
| public static boolean isPushEnabled(Context context) |

功能： 判断当前push服务是否正在运行

参数：context，应用的app context上下文

结果返回：返回true，表示当前推送服务正在运行，返回false，表示当前推送服务停止

### 设置默认推送通知栏样式

|  |
| --- |
| public static void setDefaultNotificationBuilder(Context context, PushNotificationBuilder builder) |

功能： 设置默认推送通知栏参数，包括设置通知栏图标，通知样式，通知标志等

参数：context，应用的app context上下文

PushNotificationBuilder，默认通知参数管理类

结果返回：无

### 统计分析api

|  |
| --- |
| public static void onResume(Context context)  public static void onPause(Context context) |

功能：本 API 用于“用户使用时长”，“活跃用户”，“用户打开次数”的统计，并上报到服务器，在 Portal 上展示给开发者，建议开发者在activity的onResume和onPause时调用

参数：context，Activity activity 当前所在的Activity

结果返回：无

### PushReceiver注册推送服务回调

|  |
| --- |
| public void onBind(Context context, String did, String appId, int errorCode) |

功能： 注册推送服务接口回调，在调用startWork方法后返回

参数：context，应用的app context上下文

did, 设备id

appId，应用注册推送服务的appId

errorCode，请求推送服务错误码

结果返回：无

### PushReceiver取消注册推送服务回调

|  |
| --- |
| public void onUnBind(Context context, String did, String appId, int errorCode) |

功能： 取消注册推送服务接口回调，在调用stopWork方法后返回

参数：context，应用的app context上下文

did, 设备id

appId，应用注册推送服务的appId

errorCode，请求推送服务错误码

结果返回：无

### PushReceiver获取推送消息回调

|  |
| --- |
| public void onMessage(Context context, String msgId, byte[] content) |

功能：获取消息推送数据回调

参数：context，应用的app context上下文

msgId， 消息的id

content， 透传消息的内容

结果返回：无

### PushReceiver点击消息通知栏回调

|  |
| --- |
| public void onClickNotification(Context context, String messageId, String title, String content， String extraContent) |

功能：通知栏消息点击后回调

参数：context，应用的app context上下文

messageId，推送消息id，和返回的消息Id是一一对应的

title，通知栏消息标题title

content，通知栏消息内容content

extraContent, 用户自定义扩展字段,如json格式字符串

结果返回：无

* + 1. PushManager控制sdk中log输出接口

|  |
| --- |
| public static void setDebugEnable(boolean enable) |

功能：控制sdk debug信息输出

参数：enable，是否打印log

结果返回：无

### 客制化心跳频率和push服务重连次数

|  |
| --- |
| public static void setHeartRate(int heartRate,int reconnectTime,Context context) |

功能：客制化心跳频率

参数：heartRate，心跳频率，即两次心跳间隔时间

reconnecttime：重连次数，push服务断开后，主动重连的次数

context：应用的app context 上下文

结果返回：无