SDK 用户手册

（Android 版）

发布日期： 2014年12月12日

百度开发者中心

目录

第1章 简介

百度 Push 服务 Android SDK 是百度官方推出的 Push 服务的 Android 平台开发 SDK，提供给 Android 开发者简单的接口，轻松集成百度 Push 推送服务。

Android Push 服务以后台 service 方式运行。如果一款手机安装了多个集成了 Push 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 Baidu-Push-SDK-Android-L2-VERSION.zip，VERSION 是版本号，如 2.1.0。下载解压后的目录结构如下所示：

- demo

  存放一个 Android 示例工程，可以快速帮助用户了解如何使用 SDK
- libs

  pushservice-VERSION.jar：push SDK 以 jar 的方式提供；

  libbdpush_V2_2.so： Push 服务需要用到的 jni 资源。请将您要支持的对应体系的 so 文件夹拷贝到您的工程 libs 目录下(从 4.1 版本开始，push 不再单独提供 x86 平台下 so，该平台下市场出售机器均可兼容支持 arm 平台 so，push 功能在该平台机器上完全正常，请开发者只需要集成 arm 平台和 mips 平台 so 即可)。

  请将工程的 Application 类继承 FrontiaApplication 类，在 onCreate 函数中加上:

  super.onCreate()，否则会崩溃;

  另外一种方法是：在自定义 Application 的 onCreate 方法中调用 Push 的接口：

  FrontiaApplication.initFrontia(Context context)，否则 push 的接口无法使用。
- SDK 富媒体功能用到的资源文件 res 文件夹
- 用户手册
- 版本说明

第2章 阅读对象

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。

第3章 SDK 功能说明

3.1 框架设计

Push 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）

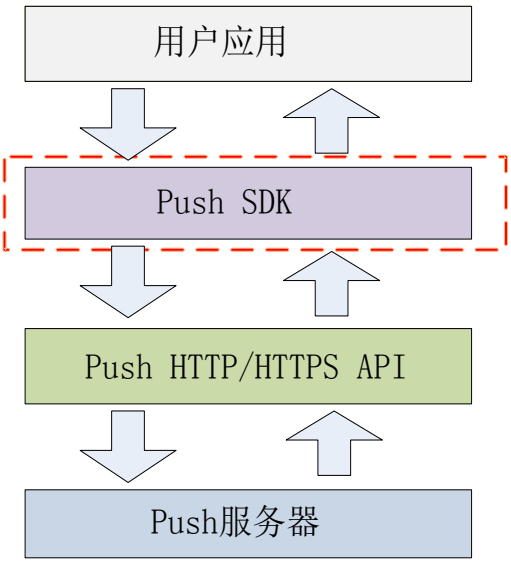

图 1 Push SDK 框架图

3.2 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
   - Push 免打扰时段设置
2. Tag 管理

   创建、删除、列出标签
   - 创建 Tag
   - 删除 Tag
   - 列出 Tag
3. 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
   - 设置富媒体通知的 Builder

4. LBS 推送
   - 打开精确 LBS 推送模式 enableLbs
   - 关闭精确 LBS 推送模式 disableLbs
5. 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
6. 富媒体推送
7. 基于精确地理位置的推送开关
8. 轻应用推送
9. 服务设置
   - 开启调试模式

第4章 开发前准备

4.1 运行环境

可运行于 Android 1.6（API Level 4）及以上版本。

4.2 参数申请及权限开通

4.2.1 获取应用 ID 及 API Key

开发者需要使用百度账号登录百度开发者中心注册成为百度开发者并创建应用，方可获取应用 ID、对应的 API Key 及 Secret Key 等信息。具体信息，请参考百度开发者中心上的“创建应用”的相关介绍。

其中，应用 ID（即：APP ID）用于标识开发者创建的应用程序；API Key（即：Client_id）是开发者创建的应用程序的唯一标识，开发者在调用百度 API 时必须传入此参数。

4.3 账户支持

4.3.1 百度账户

开发者可选择使用 oauth2.0 协议接入百度开放平台，所有用户标识使用百度的 userid 作为唯一标识，使用 AccessToken 作为验证凭证。

Android 端使用 PushManager.startWork(context, PushConstants.LOGIN_TYPE_ACCESS_TOKEN, UserAccessToken)启动基于百度账户的 Push 服务，当然要获得 AccessToken 需要登陆界面的辅助，在 Android demo 里有相关示例代码。

4.3.2 无账户登录体系

开发者无需接入百度账户体系，每个终端直接通过 API Key 向 Server 请求用户标识 userid，此 id 是根据端上的属性生成，具备唯一性，开发者可通过此 id 对应到自己的账户系统，此方式方便灵活，但需要开发者自己设计账户体系和登录界面。无账户登录体系启动 Android 端 Push 服务的方法：

PushManager.startWork(context, PushConstants.LOGIN_TYPE_API_KEY, apiKey);

第5章 使用 SDK 开发应用

5.1 添加 SDK 到 APP 工程 （具体操作可参考 PushDemo 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 pushservice-VERSION.jar 拷贝到刚刚创建的 libs 文件夹中，把 SDK 压缩包中的 libs/armeabi 目录下的 libbdpush_V2_1.so 拷贝到工程对应的 libs/armeabi 文件夹下。
4. 将上述 jar 包添加到工程的 Java Build Path
5. 在自定义 Application 中进行初始化调用，有三种方法：

a. (推荐使用方法，最简单) 直接在 AndroidManifest.xml 中指定 Application 的 android:name 属性值为 FrontiaApplication 类。

```
<application android:name="com.baidu.frontia.FrontiaApplication">
        <!--  其它的略去-- >
</application>
```

b.（PushDemo 所示方法）请将工程的 Application 类继承 FrontiaApplication 类，在 onCreate 函数中加上: super.onCreate()，否则会崩溃，示例如下:

```
import com.baidu.frontia.FrontiaApplication;
public class DemoApplication extends FrontiaApplication {
    @Override
    public void onCreate() {
        //必须加上这一句，否则会崩溃
        super.onCreate();
    }
}
```

同时还需要在 AndroidManifest 文件中的 Application 标签中指定 android:name 属性值为该 Application。如下：

```
<application android:name="com.baidu.push.example.DemoApplication"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"、>
  <!-- ……..  其它的略去-- >
</application>
```

c. 在自定义 Application 的 onCreate 方法中调用 Push 的接口：

```
FrontiaApplication.initFrontiaApplication(Context context)
```

6. 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.DISABLE_KEYGUARD" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

7. 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。

```
<!-- push service client -->
<receiver android:name="your.package.MyPushMessageReceiver">
<intent-filter>
    <!-- 接收 push 消息 -->
    <action android:name="com.baidu.android.pushservice.action.MESSAGE" />
    <!-- 接收 bind、setTags 等 method 的返回结果 -->
    <action android:name="com.baidu.android.pushservice.action.RECEIVE" />
    <!-- 可选。接受通知点击事件，和通知自定义内容 -->
    <action android:name=" com.baidu.android.pushservice.action.notification.CLICK”/>
</intent-filter>
</receiver>
```

8. AndroidManifest.xml 增加 pushservice 配置

注意：在 4.0 (包含)之后的版本，PushService 增加上了一个 intent-filter action，如下所示。

```
<!-- push service start -->
<!-- 用于接收系统消息以保证 PushService 正常运行 -->
<receiver android:name="com.baidu.android.pushservice.PushServiceReceiver"
android:process=": bdservice_v1">
<intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED" />
    <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    <action android:name="com.baidu.android.pushservice.action.notification.SHOW" />
    <action android:name="com.baidu.android.pushservice.action.media.CLICK" />
    <!-- 以下四项为可选的 action 声明，可大大提高 service 存活率和消息到达速度 -->
    <action android:name="android.intent.action.MEDIA_MOUNTED" />
    <action android:name="android.intent.action.USER_PRESENT" />
    <action android:name="android.intent.action.ACTION_POWER_CONNECTED" />
    <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED" />
</intent-filter>
</receiver>
<!-- Push 服务接收客户端发送的各种请求-->
<!-- 注意：RegistrationReceiver 在 2.1.1 及之前版本有拼写失误，为 RegistratonReceiver ，用新版本 SDK 时请更改为如下代码-->
<receiver android:name="com.baidu.android.pushservice.RegistrationReceiver"
android:process=": bdservice_v1">
<intent-filter>
    <action android:name="com.baidu.android.pushservice.action.METHOD " />
    <action android:name="com.baidu.android.pushservice.action.BIND_SYNC " />
</intent-filter>
<intent-filter>
    <action android:name="android.intent.action.PACKAGE_REMOVED"/>
    <data android:scheme="package" />
</intent-filter>
</receiver>
<!-- Push 服务 -->
<!-- 注意：在 4.0 (包含)之后的版本需加上如下所示的 intent-filter action -->
<service android:name="com.baidu.android.pushservice.PushService"
android:exported="true"  android:process=":bdservice_v1">
```

9. 增加富媒体功能（可选）

富媒体是相对于一般消息而言，一种高级的通知，提供文本、视频、音乐等形式的消息。您需

要把我们提供的资源文件(包括 layout 文件、图片和字符资源分别添加到相应的资源目录)，同时在 AndroidManifest.xml 文件里声明用到的 activity。如果应用不支持富媒体推送功能，可以不添加相关的资源和声明。

- 资源
  - Layout:
    - bpush_download_progress.xml
    - bpush_media_list_item.xml
    - bpush_media_list.xml
  - drawable:
    - bpush_gray_logo.png
    - bpush_list_item_bg.9.png
    - bpush_return_btn.png
    - bpush_top_bg.9.png
- AndroidManifest.xml 声明 Activity

```
<!-- push service rich media display -->
<activity
    android:name="com.baidu.android.pushservice.richmedia.MediaViewActivity"
    android:configChanges="orientation|keyboardHidden"
    android:label="MediaViewActivity" >
</activity>
<activity
    android:name="com.baidu.android.pushservice.richmedia.MediaListActivity"
    android:configChanges="orientation|keyboardHidden"
    android:label="MediaListActivity"
    android:launchMode="singleTask" >
</activity>
```

10. 请确保 AndroidManifest.xml 中申明了必须的组件以及组件的属性，否则 Push 服务部分功能可能不能正常工作。

5.2 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，其中 loginValue 是百度账户的 accessToken 或者是 ApiKey，由 loginType 决定。注意：不要在 Application 的 onCreate 里去做 startWork 的操作，否则可能造成应用循环重启的问题，将严重影响应用的功能和性能。

```
PushManager.startWork(context, loginType, loginValue)
```

2. 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.setNotificationBuilder(this, notificationBuilderId, cBuilder);
```

3. 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

现在支持两种方式，开发者只需要选择其中一种实现：

1）继承 FrontiaPushMessageReceiver

```
public class MyPushMessageReceiver extends FrontiaPushMessageReceiver {
/** TAG to Log */
public static final String TAG = MyPushMessageReceiver.class.getSimpleName();

/**
 * 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
public void onBind(Context context, int errorCode, String appid,
                    String userId, String channelId, String requestId) {
    String responseString = "onBind errorCode=" + errorCode + " appid="
                    + appid + " userId=" + userId + " channelId=" + channelId
                    + " requestId=" + requestId;
}

/**
 * 接收透传消息的函数。
 */
@Override
public void onMessage(Context context, String message, String customContentString) {
    String messageString = "透传消息  message=" + message + " customContentString="
                    + customContentString;
    Log.d(TAG, messageString);

    // 自定义内容获取方式，mykey 和 myvalue 对应透传消息推送时自定义内容中设置的键和值
    if (customContentString != null & customContentString != "") {
        JSONObject customJson = null;
        try {
            customJson = new JSONObject(customContentString);
            String myvalue = null;
            if (customJson.isNull("mykey")) {
                myvalue = customJson.getString("mykey");
            }
        } catch (JSONException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

```
        /**
         * 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。
         */
        @Override
        public void onNotificationClicked(Context context, String title,
                        String description, String customContentString) {
                String notifyString = "通知点击  title=" + title + " description="
                        + description + " customContent=" + customContentString;
                Log.d(TAG, notifyString);

                // 自定义内容获取方式，mykey 和 myvalue 对应通知推送时自定义内容中设置的键和值
                if (customContentString != null & customContentString != "") {
                        JSONObject customJson = null;
                        try {
                                customJson = new JSONObject(customContentString);
                                String myvalue = null;
                                if (customJson.isNull("mykey")) {
                                        myvalue = customJson.getString("mykey");
                                }
                        } catch (JSONException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        }

        /**
         * setTags()  的回调函数。
         */
        @Override
        public void onSetTags(Context context, int errorCode,
                        List<String> sucessTags, List<String> failTags, String requestId) {
                String responseString = "onSetTags errorCode=" + errorCode + " sucessTags="
                        + sucessTags + " failTags=" + failTags + " requestId="
                        + requestId;
        }
```

```
        /**
         * delTags() 的回调函数。
         */
        @Override
        public void onDelTags(Context context, int errorCode,
                                List<String> sucessTags, List<String> failTags, String requestId) {
            String responseString = "onDelTags errorCode=" + errorCode + " sucessTags="
                                + sucessTags + " failTags=" + failTags + " requestId="
                                + requestId;
        }

        /**
         * listTags() 的回调函数。
         */
        @Override
        public void onListTags(Context context, int errorCode,
                                List<String> tags, String requestId) {
            String responseString = "onListTags errorCode=" + errorCode + " tags=" + tags;
        }

        /**
         * PushManager.stopWork() 的回调函数。
         */
        @Override
        public void onUnbind(Context context, int errorCode, String requestId) {
            String responseString = "onUnbind errorCode=" + errorCode
                                + " requestId = " + requestId;
        }

    }
```

2）直接继承 BroadcastReceiver

- Push 消息通过 action=com.baidu.android.pushservice.action.MESSAGE 的 Intent 把数据发送给客户端 your.package.MyPushMessageReceiver, 消息格式由应用自己决定，push service 只负责把服务器下发的消息以字符串格式透传给客户端。
- 接口调用回调通过 action = com.baidu.android.pushservice.action.RECEIVE Intent 返回给 your.package.MyPushMessageReceiver
- 通知的点击事件回调通过 action = com.baidu.android.pushservice.action.notification.CLICK

Intent 在 your.package.MyPushMessageReceiver 中接收处理。

your.package.MyPushMessageReceiver 代码示例如下：

```
public class MyPushMessageReceiver extends BroadcastReceiver {
    /** TAG to Log */
    public static final String TAG = PushMessageReceiver.class.getSimpleName();

    /**
     * @param context
     *                  Context
     * @param intent
     *                  接收的 intent
     */
    @Override
    public void onReceive(final Context context, Intent intent) {

            Log.d(TAG, ">>> Receive intent: \r\n" + intent);

            if (intent.getAction().equals(PushConstants.ACTION_MESSAGE)) {
                //获取消息内容
                String message = intent.getExtras().getString(
                                PushConstants.EXTRA_PUSH_MESSAGE_STRING);

                //消息的用户自定义内容读取方式
                Log.i(TAG, "onMessage: " + message);

                //自定义内容的 json 串
            Log.d(TAG, "EXTRA_EXTRA = " +
intent.getStringExtra(PushConstants.EXTRA_EXTRA));

            } else if (intent.getAction().equals(PushConstants.ACTION_RECEIVE)) {
                //处理绑定等方法的返回数据
                //PushManager.startWork()的返回值通过 PushConstants.METHOD_BIND
得到
                //获取方法
                final String method = intent
                                .getStringExtra(PushConstants.EXTRA_METHOD);
```

```
    //方法返回错误码。若绑定返回错误（非 0），则应用将不能正常接收消息。
                    //绑定失败的原因有多种，如网络原因，或 access token 过期。
                    //请不要在出错时进行简单的 startWork 调用，这有可能导致死循环。
                    //可以通过限制重试次数，或者在其他时机重新调用来解决。
                    int errorCode = intent
                                    .getIntExtra(PushConstants.EXTRA_ERROR_CODE,
                                                    PushConstants.ERROR_SUCCESS);
                    String content = "";
                    if (intent.getByteArrayExtra(PushConstants.EXTRA_CONTENT) != null) {
                            //返回内容
                            content = new String(

        intent.getByteArrayExtra(PushConstants.EXTRA_CONTENT));
                    }

                    //用户在此自定义处理消息,以下代码为 demo 界面展示用
                    Log.d(TAG, "onMessage: method : " + method);
                    Log.d(TAG, "onMessage: result : " + errorCode);
                    Log.d(TAG, "onMessage: content : " + content);


                //可选。通知用户点击事件处理
                } else if (intent.getAction().equals(
                                PushConstants.ACTION_RECEIVER_NOTIFICATION_CLICK)) {
                    Log.d(TAG, "intent=" + intent.toUri(0));

                    //自定义内容的 json 串
                Log.d(TAG, "EXTRA_EXTRA = " +
intent.getStringExtra(PushConstants.EXTRA_EXTRA));


                }
        }

    }
```

5.3 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，pushservice-VERSION.jar 为集成的 jar 包名字)：

```
-libraryjars libs/pushservice-VERSION.jar
-dontwarn com.baidu.**
```

第6章 API 说明

本文档提供开放接口的 API 说明。。

6.1 类

本 SDK 中有四个重要的开放类，分别为：PushManager、PushSettings、BasicPushNotificationBuilder 和 CustomPushNotificationBuilder，还有常量类 PushConstants。

类	描述
PushManager	PushManager 提供了所有使用 Push 服务的静态方法
BasicPushNotificationBuilder	用于定制 Android Notification 里的 defaults / flags / icon 等基础样式（行为）
CustomPushNotificationBuilder	用于定制 Android Notification 里的 defaults / flags / icon，以及通知栏的 layout、图标和状态栏图标
PushSettings	PushSettings 提供了端上 Push 服务的配置静态方法
PushConstants	SDK 对外的常量定义
FrontiaPushMessageReceiver	Push 消息处理 receiver

6.2 API

本 SDK 目前支持以下接口：

分类	功能	API 函数原型
Push 服务接口	提供 Push 服务	startWork, stopWork, resumeWork
Tag 管理接口	Tag 的创建与删除	setTags, delTags, listTags
通知管理接口	自定义通知样式	CustomPushNotificationBuilder, BasicPushNotificationBuilder setNotificationFlags, setNotificationDefaults, setStatusbarIcon, setLayoutDrawable, setNotificationBuilder
推送效果反馈	反馈推送通知的效果	activityStarted, activityStoped
设置接口	Push 服务设置	enableDebugMode
LBS 推送接口	打开或关闭精确 LBS 推送	enableLbs, disableLbs
异步消息处理接口	Push 消息处理 receiver	onBind, onMessage, onNotificationClicked, onSetTags, onDelTags, onListTags, onUnbind
统计渠道号设置	在统计中查看渠道分布	在 AndroidManifest.xml 中添加 meta：BaiduPush_CHANNEL

6.2.1 Push 服务初始化及绑定-- startWork

- 函数原型

```
public static void startWork(Context context, int loginType, String loginValue)
```

- 功能

PushManager 类定义的静态方法，完成 Push 服务的初始化， 并且自动完成 bind 工作。

- 参数

context：当前执行 Context

loginType：绑定认证方式

无账号认证方式用 PushConstants.LOGIN_TYPE_API_KEY；百度 Auth2.0 认证方式用 PushConstants.LOGIN_TYPE_ACCESS_TOKEN

loginValue：和 loginType 对应，分别是应用的 API Key，或者百度 Auth2.0 Access Token

- 返回结果

通过 com.baidu.android.pushservice.action.RECEIVE Intent 发送给客户端 receiver，Intent extra PushConstants.EXTRA_CONTENT 包含返回字符串，为 json 格式，如：

```
{
        “request_id”:12394838223,
        “response_params”:
        {
            “appid”: “0696110321”,
            “user_id”:”123456789012345678”,
            “channel_id”:”4923859573096872165”,
        }
}
```

参数说明：

user_id 系统返回的绑定 Baidu Channel 的用户标识

appid 系统返回的应用标识

channel_id 系统返回的设备连接的通道标识

如果您通过服务端接口（REST 或 SDK）向百度 Push 服务推送消息，bind 成功后需要把 channel_id 和 access token（或者 appid + user_id）传给应用的服务器，以便您的服务器向百度 Push 服务推送消息。如果只使用管理控制台来发送消息，可以不用关心这些返回信息。

注意：不要在 Application 的 onCreate 里去做 startWork 的操作。

6.2.2 停止和恢复 Push 服务-- stopWork、resumeWork

- 函数原型

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

- 功能

PushManager 类定义的静态方法，停止本应用 Push 服务进程， 并且完成 unbind 工作。startWork 和 resumeWork 都会重新开启本应用 Push 功能。

- 参数

  context：当前执行 Context
- 返回结果

  无

- 函数原型

  public static void resumeWork(Context context)
- 功能

  PushManager 类定义的静态方法，恢复本应用 Push 服务， 并且再次完成 bind 工作。
- 参数

  context：当前执行 Context
- 返回结果

  无

6.2.3 查询 push 是否被停止的接口-- isPushEnabled

- 函数原型

  public static boolean isPushEnabled(Context context)
- 功能

  PushManager 类定义的静态方法，查询 push 是否已经被停止。
- 参数

  context：当前执行 Context
- 返回结果

  无

6.2.4 设置免打扰时段

- 函数原型

  PushManager.setNoDisturbMode(Context context, int startHour, int startMinute, int endHour, int endMinute)
- 功能

  PushManager 类定义的静态方法，设置免打扰模式的具体时段，该时间内处于免打扰模式，通知到达时去除通知的提示音、振动以及提示灯闪烁。
- 参数

  context：当前执行 Context

startHour，startMinute：起始时间，24 小时制，取值范围 0~23，0~59

endHour，endMinute：结束时间，24 小时制，取值范围 0~23，0~59

- 返回结果

无

6.2.5 设置 Tag-- setTags

- 函数原型

```
public static void setTags(Context  context,  List<String>  tags)
```

- 功能

PushManager 类定义的静态方法，用于设置标签；成功设置后，可以从管理控制台或您的服务后台，向指定的设置了该 tag 的一群用户进行推送。

注意：tag 设置的前提是已绑定的端，也就是应用有运行过 startWork 或 bind，且在 onBind 回调中返回成功。

- 返回结果

通过 com.baidu.android.pushservice.action.RECEIVE Intent 发送给客户端 receiver，Intent extra PushConstants.EXTRA_CONTENT 包含返回字符串，json 格式：

```
{
  “request_id”:12394838223,
  “response_params”:
   {
        success_amount:1,
        details:
        {
            “tag”:”abc”,
            “result”:0,//成功
        }
        {
            “tag”:”def”,
            “result”:1, //失败
        }
   }
}
```

参数说明：

tags 包含多个标签字符串的 List

6.2.6 删除 Tag-- delTags

- 函数原型

```
public static void delTags(Context  context,  List<String>  tags)
```

- 功能

PushManager 类定义的静态方法，用于删除标签。

- 返回结果

通过 com.baidu.android.pushservice.action.RECEIVE Intent 发送给客户端 receiver，Intent extra PushConstants.EXTRA_CONTENT 包含返回字符串，json 格式：

```
{
    “request_id”:12394838223,
    “response_params”:
    {
        success_amount:1,
        details:
        {
            “tag”:”abc”,
            “result”:0,//成功
        }
        {
            “tag”:”def”,
            “result”:1,//失败
        }
    }
}
```

参数说明：

tags 包含多个标签字符串的 List

6.2.7 列出 Tag-- listTags

- 函数原型

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

- 功能

PushManager 类定义的静态方法，用于列出本机绑定的标签。

- 返回结果

通过 com.baidu.android.pushservice.action.RECEIVE Intent 发送给客户端 receiver，Intent extra PushConstants.EXTRA_CONTENT 包含返回字符串，json 格式：

```
{
    “request_id”:12394838223,
    “response_params”:
    {
        success_amount:1,
        details:
        {
            “tag”:”abc”,
        }
        {
            “tag”:”def”,
        }
    }
}
```

参数说明：

tags 包含多个标签字符串的 List

6.2.8 统计 activity onStart-- activityStarted

- 函数原型

```
public static void activityStarted(Activity    activity)
```

- 功能

PushManager 类定义的静态方法，在您的 Activity 的 onStart 方法里面加上 PushManager.onStart()方法，会统计应用程序的包名、时间戳、用户是直接打开应用还是通过点击推送的通知打开的应用、messageId 和 Activity 的 hash code。

6.2.9 统计 activity onStop-- activityStoped

- 函数原型

```
public static void activityStoped(Activity    activity)
```

- 功能

PushManager 类定义的静态方法，在您的 Activity 的 onStop 方法里面加上 PushManager.onStop()方法，会统计应用程序的包名、时间戳和 Activity 的 hash code。

6.2.10 设置通知的 Builder -- setNotificationBuilder

- 函数原型

```
public static void setNotificationBuilder(Context    context,    int    id,
```

```
                                                    PushNotificationBuilder    notificationBuilder)
```

- 功能

PushManager 类定义的静态方法，设置通知栏样式，并为样式指定编号。在管理控制台或您的服务后台中，您可以指定相应的编号，让客户端显示预先设定好的样式。

- 参数

context ： android app 运行上下文

id ： notificationBuilder 编号，开发者自己维护

notificationBuilde ： 通知栏构建类

6.2.11 设置默认的通知 Builder-- setDefaultNotificationBuilder

- 函数原型

```
public static void setDefaultNotificationBuilder(Context    context,
                                                  PushNotificationBuilder
notificationBuilder)
```

- 功能

PushManager 类定义的静态方法，设置默认的通知栏样式；如果推送通知时不指定指定 id 的样式，都将显示该默认样式。

- 参数

Context ： android app 运行上下文
notificationBuilder ： 通知栏构建类

6.2.12 设置富媒体通知的 Builder-- setMediaNotificationBuilder

- 函数原型

```
public static void setMediaNotificationBuilder(Context    context,
                                                PushNotificationBuilder
notificationBuilder)
```

- 功能

PushManager 类定义的静态方法，为富媒体通知设置样式；用法和自定义通知样式相似。

6.2.13 自定义通知 Builder-- BasicPushNotificationBuilder

- 函数原型

```
BasicPushNotificationBuilder ()
```

- 功能

自定义通知状态栏构建类构造函数(定制通知栏基础样式) 。

6.2.14 自定义通知 Builder-- CustomPushNotificationBuilder

- 函数原型

```
CustomPushNotificationBuilder(layoutId,  layoutIconId,  layoutTitleId,  layoutTextId)
```

- 功能

自定义通知状态栏构建类构造函数(定制通知栏基础样式及 layout)。

- 参数

layoutId ： 自定义 layout 资源 id

layoutIconId ： 自定义 layout 中显示 icon 的 view id

layoutTitleId ： 自定义 layout 中显示标题的 view id

layoutTextId ： 自定义 layout 中显示内容的 view id

6.2.15 设置通知 flags-- setNotificationFlags

- 函数原型

```
public void setNotificationFlags (int  flags)
```

- 功能

基类 PushNotificationBuilder 定义的方法，定制 Android Notification 里的 flags。

- 参数

flags ： Android Notification flags

6.2.16 设置通知 defaults-- setNotificationDefaults

- 函数原型

```
public void setNotificationDefaults (int  defaults)
```

- 功能

基类 PushNotificationBuilder 定义的方法，定制 Android Notification 里的 defaults。

- 参数

defaults ： Android Notification defaults

6.2.17 设置通知状态栏 icon-- setStatusbarIcon

- 函数原型

```
public void setStatusbarIcon (int  icon)
```

- 功能

基类 PushNotificationBuilder 类定义的方法，定制 Android Notification 通知状态栏的 icon 图标。

参数

icon：图标资源 id

6.2.18 设置通知样式图片-- setLayoutDrawable

- 函数原型

```
public void setLayoutDrawable(int drawableId)
```

- 功能

CustomPushNotificationBuilder 类定义的方法，定制自定义 layout 中显示的图片。

参数

drawableId：图标资源 id

6.2.19 开启调试模式-- enableDebugMode

- 函数原型

```
public static void enableDebugMode(boolean debugEnabled)
```

- 功能

PushSettings 类定义的方法，开启调试模式，会输出调试 Log。注：发布应用时，请去除开启调试模式的调用，以免降低 Push 的性能。

6.2.20 开启精确 LBS 推送模式-- enableLbs

- 函数原型

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

- 功能

PushManager 类定义的方法，开启精确 LBS 推送模式，覆盖最近半小时内在指定区域出现过的终端。

6.2.21 关闭精确 LBS 推送模式-- disableLbs

- 函数原型

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

- 功能

PushManager 类定义的方法，关闭精确 LBS 推送模式，关闭后，服务端将无法有效发送基于地理位置的定向推送。

6.2.22 获取绑定请求的结果-- onBind

- 函数原型

```
public void onBind(Context context, int errorCode, String appid,
                   String userId, String channelId, String requestId)
```

- 功能

FrontiaPushMessageReceiver 的抽象方法，把 receiver 类继承 FrontiaPushMessageReceiver 可以使用。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。

如果您需要用单播推送，需要把这里获取的 channel id 和 user id 上传到应用 server 中，再调用 server 接口，用 channel id 和 user id 给单个手机或者用户推送。

- 参数

context　BroadcastReceiver 的执行 Context

errorCode 绑定接口返回值，0 - 成功

appid　应用 id。errorCode 非 0 时为 null

userId　应用 user id。errorCode 非 0 时为 null

channelId　应用 channel id。errorCode 非 0 时为 null

requestId 向服务端发起的请求 id。在追查问题时有用；

6.2.23 接收透传消息的函数-- onMessage

- 函数原型

```
public void onMessage(Context context, String message, String customContentString)
```

- 功能

FrontiaPushMessageReceiver 的抽象方法，把 receiver 类继承 FrontiaPushMessageReceiver 可以使用。接收透传消息。

- 参数

context 上下文

message 推送的消息

customContentString 自定义内容,为空或者 json 字符串

6.2.24 接收通知点击的函数-- onNotificationClicked

- 函数原型

```
public void onNotificationClicked(Context context, String title,
                   String description, String customContentString)
```

- 功能

FrontiaPushMessageReceiver 的抽象方法，把 receiver 类继承 FrontiaPushMessageReceiver 可

以使用。接收通知点击的函数。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。

- 参数

context 上下文

title 推送的通知的标题

description 推送的通知的描述

customContentString 自定义内容，为空或者 json 字符串

6.2.25 setTags 的回调函数-- onSetTags

- 函数原型

```
public void onSetTags(Context context, int errorCode,
                      List<String> sucessTags, List<String> failTags, String requestId)
```

- 功能

FrontiaPushMessageReceiver 的抽象方法，把 receiver 类继承 FrontiaPushMessageReceiver 可以使用。setTags() 的回调函数。

- 参数

context 上下文

errorCode 错误码。0 表示某些 tag 已经设置成功；非 0 表示所有 tag 的设置均失败。

successTags 设置成功的 tag

failTags 设置失败的 tag

requestId 分配给对云推送的请求的 id

6.2.26 delTags 的回调函数-- onDelTags

- 函数原型

```
public void onDelTags(Context context, int errorCode,
                      List<String> sucessTags, List<String> failTags, String requestId)
```

- 功能

FrontiaPushMessageReceiver 的抽象方法，把 receiver 类继承 FrontiaPushMessageReceiver 可以使用。delTags() 的回调函数。

- 参数

context 上下文

errorCode 错误码。0 表示某些 tag 已经删除成功；非 0 表示所有 tag 均删除失败。

successTags 成功删除的 tag

failTags 删除失败的 tag

requestId 分配给对云推送的请求的 id

6.2.27 listTags 的回调函数-- onListTags

- 函数原型

```
public void onListTags(Context context, int errorCode,
                       List<String> tags, String requestId)
```

- 功能

FrontiaPushMessageReceiver 的抽象方法，把 receiver 类继承 FrontiaPushMessageReceiver 可以使用。listTags() 的回调函数。

- 参数

context 上下文

errorCode 错误码。0 表示列举 tag 成功；非 0 表示失败。

tags 当前应用设置的所有 tag。

requestId 分配给对云推送的请求的 id

6.2.28 stopWork 的回调函数-- onUnbind

- 函数原型

```
public void onUnbind(Context context, int errorCode, String requestId)
```

- 功能

FrontiaPushMessageReceiver 的抽象方法，把 receiver 类继承 FrontiaPushMessageReceiver 可以使用。PushManager.stopWork() 的回调函数。

- 参数

context 上下文

errorCode 错误码。0 表示从云推送解绑定成功；非 0 表示失败。

requestId 分配给对云推送的请求的 id

6.2.29 渠道号设置

通过 AndroidManifest.xml 中静态设置渠道号，meta-data：BaiduPush_CHANNEL，如：

<meta-data android:name="*BaiduPush_CHANNEL*" android:value="*91 *" />

6.3 常量说明

Android SDK 的常量定义都在 PushConstants 类中。如下：

✧ Push 发送给应用的 Action 的常量，都是 String 类型

ACTION_MESSAGE

接收消息时使用。

ACTION_RECEIVE

获取方法调用的返回值，包括绑定、设置 Tag、删除 Tag 等方法。

ACTION_RECEIVER_NOTIFICATION_CLICK

通知点击事件的截获。注意，通知发出时应用。

✧ 从 Intent 中获取 Extra 的常量，都是 String 类型

EXTRA_PUSH_MESSAGE_STRING

消息内容，在 ACTION_MESSAGE 中使用。Extra 获取方法 intent.getStringExtra(PushConstants.EXTRA_PUSH_MESSAGE_STRING)。

EXTRA_METHOD

方法名，在 ACTION_RECEIVE 中使用。Extra 获取方法 intent.getStringExtra(PushConstants. EXTRA_METHOD)。

EXTRA_ERROR_CODE

方法错误码，在 ACTION_RECEIVE 中使用。Extra 获取方法 intent.getIntExtra(PushConstants. EXTRA_ERROR_CODE)。

EXTRA_CONTENT

方法返回内容，在 ACTION_RECEIVE 中使用。Extra 获取方法 intent.getByteArrayExtra(PushConstants.*EXTRA_CONTENT*)。

EXTRA_NOTIFICATION_TITLE

通知标题，在 ACTION_RECEIVER_NOTIFICATION_CLICK 中使用。Extra 获取方法 intent.getStringExtra(PushConstants. EXTRA_NOTIFICATION_TITLE)。

EXTRA_NOTIFICATION_CONTENT

通知内容，在 ACTION_RECEIVER_NOTIFICATION_CLICK 中使用。Extra 获取方法 intent.getStringExtra(PushConstants. ACTION_RECEIVER_NOTIFICATION_CLICK)。

✧ 方法名常量，都是 String 类型

METHOD_BIND, METHOD_SET_TAGS, METHOD_DEL_TAGS

绑定方法名，在 EXTRA_METHOD 中取得的值是这三者之一。

第7章 联系我们

如果以上信息无法帮助您解决在开发中遇到的具体问题，请通过以下方式联系我们：

邮箱: dev_support@baidu.com

百度工程师会在第一时间回复您。

第8章 缩略语

缩略语	英文全称	说明
SDK	Software Development Kit	软件开发工具包。