



百度云推送 **Android SDK**

升级指南

(Android 版)

发布日期： 2014年12月12日

百度开发者中心

(版权所有，翻版必究)

目录

第 1 章 SDK 当前版本信息 3

第 2 章 CHANGE LOG 3

第 3 章 升级提示 3

第 4 章 升级指南 3

 4.1 更新 PUSHSERVICE 声明 3

 4.2 初始化 PUSH 4

 4.3 使用新的 MYPUSHMESSAGE RECEIVER(可选) 5

 4.4 删除功能特性(可选) 12

第 5 章 联系我们 12

第1章 SDK 当前版本信息

此次发布的 SDK 的版本信息如下：

版本号	发布日期	版本名称	描述
4.3	2014 年 12 月 12 日 星期五	Baidu-Push-SDK-Android-L2-4.3.0	优化推送到达策略，显著提升到达率和及时性。 更新 so 库文件。 增加四个可选的唤醒事件，提升 Service 存活率。 修复极端情况下推送到达后分发不正常的问题。 优化异常处理，修复大量可能引发 Service 工作不正常的潜在逻辑 bug。 修复若干情况下长连接不必要的断开重连的问题。

第2章 Change Log

- 1. 优化推送到达策略
- 2. 增加四个可选的唤醒事件
- 3. 修复若干分发 bug

第3章 升级提示

修复了若干 bug，提高了 Service 存活率，显著提升到达率和及时性。强烈建议开发者升级。

第4章 升级指南

4.1 更新 AndroidManifest 声明

4.0 版本的 AndroidManifest.xml 中 PushService 声明新增一个 intent-filter，如下：

```
<service android:name="com.baidu.android.pushservice.PushService"
    android:exported="true"  android:process=" bdservice_v1">
    <intent-filter >
    <action android:name="com.baidu.android.pushservice.action.PUSH_SERVICE"/>
```

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

```
</service>
```

4.3 版本的 AndroidManifest.xml 中 PushServiceReceiver 中增加四个可选 Action，如下：

```
<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>
```

4.2 更新 so 库

请将 libs 目录下 armeabi/mips 目录下原有的 libbdpush_V*_* .so，替换为最新的 libbdpush_V2_2.so。（从 4.1 版本起不再单独提供 x86 下 so，目前 x86 机型均支持 arm 指令集兼容，push 功能运行正常，请删除原有 x86 目录下的 so 文件）。

如果开发者从 3.0 及以后版本升级到最新版本，则无需修改代码；从 3.0 之前版本升级时请关注以下内容。

4.3 初始化 Push

必须在自定义 Application 中进行初始化调用，有三种方法：

- a. 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，比如：

```
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();
    }
}
```

如果之前没用用到自定义 Application 类，还需要在 AndroidManifest.xml 文件中的 Application 标签中指定 android:name 属性值。如：

```
<application android:name="com.baidu.push.example.DemoApplication"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name">
    <activity
        android:name="com.baidu.push.example.PushDemoActivity"
        android:launchMode="singleTask"
        android:screenOrientation="portrait"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

- b. 如果应用不想实现自定义的 Application 类，也可以直接在 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>
```

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

FrontiaApplication.initFrontiaApplication(Context context)

4.4 使用新的 MyPushMessageReceiver(可选)

新版提供更友好的接收消息的回调方法(在 MessageReceiver 中)，但兼容原来的接收方式。

继承新的 FrontiaPushMessageReceiver 后，需要把原来的父类 BroadcastReceiver 替换掉则可以在 onBind, onUnbind, onMessage, onNotificationClicked, onDeleteTags, onListTags, onSetTags 等回调方法中取得所有需要的字段。代码示例如下：

```
package com.baidu.push.example;
```

```
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

import org.json.JSONException;
import org.json.JSONObject;

import android.content.Context;
import android.content.Intent;
import android.util.Log;

import com.baidu.frontia.api.FrontiaPushMessageReceiver;

/**
 * Push 消息处理 receiver。请编写您需要的回调函数，
 * 一般来说：
 * onBind 是必须的，用来处理 startWork 返回值；
 * onMessage 用来接收透传消息；
 * onSetTags、onDelTags、onListTags 是 tag 相关操作的回调；
 * onNotificationClicked 在通知被点击时回调；
 * onUnbind 是 stopWork 接口的返回值回调*
 *
 */
public class MyPushMessageReceiver extends FrontiaPushMessageReceiver extends
BroadcastReceiver{
    /** TAG to Log */
    public static final String TAG = MyPushMessageReceiver.class.getSimpleName();

    @Override
    public void onReceive(final Context context, Intent intent) {
        //原来的处理逻辑
    }
}

/**
 * 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。
 *

```

```

* @param context
*         BroadcastReceiver 的执行 Context
* @param errorCode
*         绑定接口返回值, 0 - 成功
* @param appid
*         应用 id。errorCode 非 0 时为 null
* @param userId
*         应用 user id。errorCode 非 0 时为 null
* @param channelId
*         应用 channel id。errorCode 非 0 时为 null
* @param requestId
*         向服务端发起的请求 id。在追查问题时有用;
* @return
*         none
*/
@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;
    Log.d(TAG, responseString);

    // Demo 更新界面展示代码, 用户请在这里加入自己的处理逻辑
    updateContent(context, responseString);
}

/**
* 接收透传消息的函数。
*
* @param context 上下文
* @param message 推送的消息
* @param customContentString 自定义内容, 为空或者 json 字符串
*/
@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();
        }
    }

    // Demo 更新界面展示代码，用户请在这里加入自己的处理逻辑
    updateContent(context, messageString);
}

/**
 * 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 上下文
 * @param title 推送的通知的标题
 * @param description 推送的通知的描述
 * @param customContentString 自定义内容，为空或者 json 字符串
 */
@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);

```


和值

// 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

```
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();
    }
}

// Demo 更新界面展示代码，用户请在这里加入自己的处理逻辑
updateContent(context, notifyString);
}

/**
 * setTags() 的回调函数。
 *
 * @param context 上下文
 * @param 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。
 * @param successTags 设置成功的 tag
 * @param failTags 设置失败的 tag
 * @param requestId 分配给对云推送的请求 id
 */
@Override
public void onSetTags(Context context, int errorCode,
    List<String> successTags, List<String> failTags, String requestId) {
    String responseString = "onSetTags errorCode=" + errorCode + " successTags="
        + successTags + " failTags=" + failTags + " requestId="
        + requestId;
    Log.d(TAG, responseString);
}
```

```

        // Demo 更新界面展示代码，用户请在这里加入自己的处理逻辑
        updateContent(context, responseString);
    }

    /**
     * deleteTags() 的回调函数。
     *
     * @param context 上下文
     * @param 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。
     * @param successTags 成功删除的 tag
     * @param failTags 删除失败的 tag
     * @param requestId 分配给对云推送的请求 id
     */
    @Override
    public void onDelTags(Context context, int errorCode,
        List<String> successTags, List<String> failTags, String requestId) {
        String responseString = "onDelTags errorCode=" + errorCode + " successTags="
            + successTags + " failTags=" + failTags + " requestId="
            + requestId;
        Log.d(TAG, responseString);

        // Demo 更新界面展示代码，用户请在这里加入自己的处理逻辑
        updateContent(context, responseString);
    }

    /**
     * listTags() 的回调函数。
     *
     * @param context 上下文
     * @param errorCode 错误码。0 表示列举 tag 成功；非 0 表示失败。
     * @param tags 当前应用设置的所有 tag。
     * @param requestId 分配给对云推送的请求 id
     */
    @Override
    public void onListTags(Context context, int errorCode,

```

```
        List<String> tags, String requestId) {
    String responseString = "onListTags errorCode=" + errorCode + " tags=" + tags;
    Log.d(TAG, responseString);

    // Demo 更新界面展示代码，用户请在这里加入自己的处理逻辑
    updateContent(context, responseString);
}

/**
 * PushManager.stopWork() 的回调函数。
 *
 * @param context 上下文
 * @param errorCode 错误码。0 表示从云推送解绑定成功；非 0 表示失败。
 * @param requestId 分配给对云推送的请求 id
 */
@Override
public void onUnbind(Context context, int errorCode, String requestId) {
    String responseString = "onUnbind errorCode=" + errorCode
        + " requestId = " + requestId;
    Log.d(TAG, responseString);

    // Demo 更新界面展示代码，用户请在这里加入自己的处理逻辑
    updateContent(context, responseString);
}

private void updateContent(Context context, String content) {
    Log.d(TAG, "updateContent");
    String logText = "" + Utils.logStringCache;

    if (!logText.equals("")) {
        logText += "\n";
    }

    SimpleDateFormat sDateFormat = new SimpleDateFormat("HH-mm-ss");
    logText += sDateFormat.format(new Date()) + ": ";
    logText += content;

    Utils.logStringCache = logText;
}
```

```
        Intent intent = new Intent();
        intent.setClass(context.getApplicationContext(), PushDemoActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        context.getApplicationContext().startActivity(intent);
    }
}
```

4.5 删除了功能特性(可选)

如果仅使用基础功能，开发者可以在 `AndroidManifest.xml` 文件中去除 `android.permission.SYSTEM_ALERT_WINDOW` 的权限要求。

第5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

QQ 群： 324533810

百度工程师会在第一时间回复您。