

百度云推送Android SDK

升级指南

（Android版）

发布日期： 2014年5月14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](#_Toc386625967)

[第2章 Change Log 3](#_Toc386625968)

[第3章 升级提示 3](#_Toc386625969)

[第4章 升级指南 3](#_Toc386625970)

[4.1 更新PushService声明 3](#_Toc386625971)

[4.2 初始化Push 3](#_Toc386625972)

[4.3 使用新的MyPushMessageReceiver(可选) 4](#_Toc386625973)

[4.4 删除功能特性(可选) 11](#_Toc386625974)

[第5章 联系我们 11](#_Toc386625975)

# 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** |
| 4.0 | 2014年5月14日 星期三 | Baidu-Push-SDK-Android-L2-4.0.0 | 增加轻应用推送支持  修复MIUI V5 rom有时无法启动Service的问题  修复部分手机出现的ClassNotFound异常 |

# Change Log

1. 增加轻应用推送支持
2. 修复MIUI V5 rom有时无法启动Service的问题
3. 修复部分手机出现的ClassNotFound异常

# 升级提示

提升了Service的稳定性，修复了MIUI V5 rom下有时无法启动Service的问题。建议开发者升级。

# 升级指南

## 更新PushService声明

4.0版本的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>

## 更新so库

请替换libs目录有三个文件夹armeabi/x86/mips下原有的libbdpush\_V1\_0.so，使用最新的libbdpush\_V2\_0.so。

开发者如果从3.0及以后版本升级到4.0，则无需进行代码修改；否则请关注以下内容

## 初始化Push

必须在自定义Application中进行初始化调用，有三种方法：

1. 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manifest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>

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

1. 在自定义Application的onCreate方法中调用Push的接口：**FrontiaApplication.initFrontiaApplication**(Context context)

## 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MesssageReceiver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BroadcastRecevier替换掉则可以在onBind，onUnbind，onMessage，onNotificationClicked，onDelTags，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，应用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，应用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);

// 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();

}

}

// Demo更新界面展示代码，应用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的id

\*/

@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;

Log.d(TAG, responseString);

// Demo更新界面展示代码，应用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的id

\*/

@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;

Log.d(TAG, responseString);

// Demo更新界面展示代码，应用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的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，应用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的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，应用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);

}

}

## 删除了功能特性(可选)

如果仅使用基础功能，开发者可以在manifest文件中去除android.permission.SYSTEM\_ALERT\_WINDOW的权限要求。

# 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](mailto:dev_support@baidu.com)

QQ群：324533810

百度工程师会在第一时间回复您。