

Flyme 推送 Android 客户端 SDK 文档

1 接入步骤

PushSDK 3.0 以后的版本使用了 aar 包方式，因此对于一些通用的权限配置，工程混淆，App 接入时不需要再进行配置了，只需要按以下步骤简单接入即可。

1.1 配置依赖

我们已经将 PushSDK 发布到 jcenter，您只需要在工程 gradle 文件中进行如下依赖配置：

```
dependencies {  
    compile 'com.meizu.flyme.internet:push-internal:3.8.3'  
}
```

注意：如果由于各种原因不能使用 jcenter 依赖，可以直接下载 AAR 包进行手动集成：[点击下载](#)。

1.2 声明权限

在您工程 `AndroidManifest.xml` 中进行以下权限的声明：

```
<!-- 兼容 Flyme5 以下版本，魅族内部接入 PushSDK 必填，不然无法收到消息-->  
<uses-permission android:name="com.meizu.flyme.push.permission.RECEIVE" />  
<permission android:name="【替换您的包名】.push.permission.MESSAGE"  
android:protectionLevel="signature"/>  
<uses-permission android:name="【替换您的包名】.push.permission.MESSAGE" />  
<!-- 兼容 Flyme3 配置权限-->  
<uses-permission android:name="com.meizu.c2dm.permission.RECEIVE" />  
<permission android:name="【替换您的包名】.permission.C2D_MESSAGE" android:protectionLevel="signature"  
/>  
<uses-permission android:name="【替换您的包名】.permission.C2D_MESSAGE"/>
```

注意：请将以上【替换您的包名】替换成您自己 App 的包名。

1.3 注册消息接收的广播

在您工程 `AndroidManifest.xml` 中进行消息接收广播的声明：

```
<!-- push 应用定义消息 receiver 声明 -->  
<receiver android:name="【替换您的包名】.MyPushMsgReceiver">  
    <intent-filter>  
        <!-- 接收 push 消息 -->  
        <action android:name="com.meizu.flyme.push.intent.MESSAGE" />  
        <!-- 接收 register 消息 -->  
        <action android:name="com.meizu.flyme.push.intent.REGISTER.FEEDBACK" />  
        <!-- 接收 unregister 消息-->  
        <action android:name="com.meizu.flyme.push.intent.UNREGISTER.FEEDBACK"/>  
        <!-- 兼容低版本 Flyme3 推送服务配置 -->  
        <action android:name="com.meizu.c2dm.intent.REGISTRATION" />
```

```

        <action android:name="com.meizu.c2dm.intent.RECEIVE" />
        <category android:name="【替换您的包名】" />
    </intent-filter>
</receiver>

```

注意：请将以上【替换您的包名】替换成您自己 App 的包名。

1.4 实现消息接收的广播

创建消息接收的广播类，名称如：MyPushMsgReceiver，实现如下：

```

public class MyPushMsgReceiver extends MzPushMessageReceiver {

    @Override
    @Deprecated
    public void onRegister(Context context, String s) {
        // 调用旧版的订阅 PushManager.register(context) 方法后，
        // 会在此回调订阅状态（已废弃）
    }

    @Override
    @Deprecated
    public void onUnRegister(Context context, boolean b) {
        // 调用旧版的反订阅 PushManager.unregister(context) 方法后，
        // 会在此回调反订阅状态（已废弃）
    }

    @Override
    public void onRegisterStatus(Context context, RegisterStatus registerStatus) {
        // 调用新版的订阅 PushManager.register(context, appId, appKey) 方法后，
        // 会在此回调订阅状态
    }

    @Override
    public void onUnRegisterStatus(Context context, UnRegisterStatus unRegisterStatus) {
        // 调用新版的反订阅 PushManager.unregister(context, appId, appKey) 方法后，
        // 会在此回调订阅状态
    }

    @Override
    public void onPushStatus(Context context, PushSwitchStatus pushSwitchStatus) {
        // 调用 PushManager.switchPush/checkPush 方法后，
        // 会在此回调通知栏和透传消息的开关状态
    }

    @Override
    public void onSubTagsStatus(Context context, SubTagsStatus subTagsStatus) {
        // 调用 PushManager.subscribeTags/unSubscribeTags/unSubscribeAllTags
        // /checkSubscribeTags 方法后，会在此回调标签相关信息
    }

    @Override

```

```
public void onSubAliasStatus(Context context, SubAliasStatus subAliasStatus) {
    // 调用 PushManager.subscribeAlias/unSubscribeAlias/checkSubscribeAlias
    // /checkSubscribeTags 方法后，会在此回调别名相关信息
}

@Override
public void onUpdateNotificationBuilder(PushNotificationBuilder pushNotificationBuilder) {
    // 兼容旧版本 Flyme 系统中设置消息弹出后状态栏中的小图标
    // 同时请在相应的 drawable 不同分辨率文件夹下放置一张名称务必为
    // mz_push_notification_small_icon 的图片
    pushNotificationBuilder.setmStatusBarIcon(R.drawable.mz_push_notification_small_icon);
}

@Override
public void onNotificationClicked(Context context, MzPushMessage mzPushMessage) {
    // 当用户点击通知栏消息后会在此方法回调
}

@Override
public void onNotificationArrived(Context context, MzPushMessage mzPushMessage) {
    // 当推送的通知栏消息展示后且应用进程存在时会在此方法回调
}
}
```

至此 PushSDK 已经接入完毕，现在您还需要在您工程中去调用一下订阅方法（建议在 Application onCreate 中执行）：

```
PushManager.register(this, APP_ID, APP_KEY);
```

并确保您自定义的消息接收 Receiver 类中能在 onRegisterStatus(RegisterStatus registerStatus) 方法中正确回调，这样接入就成功了。现在就可以到 [Push 平台](#)找到您的应用进行消息推送了。

2 功能说明

2.1 PushManager 方法说明

| 方法名称 | 方法说明 | 使用建议 | 对应 MzPushReceiver 回调方法 |
|---|------|--------------------------------|--|
| register(Context context, String appId, String appKey) | 订阅 | 建议在 Application onCreate 中调用 | onRegisterStatus(Context context, RegisterStatus registerStatus) |
| unRegister(Context context, String appId, String appKey) | 反订阅 | 取消所有推送时使用，请慎用，若取消订阅，将有可能停止所有推送 | onUnRegisterStatus(Context context, UnRegisterStatus unRegisterStatus) |
| subscribeTags(Context context, String appId, String appKey, String pushId, String | 标签订阅 | 无 | onSubTagsStatus(Context context, SubTagsStatus subTagsStatus) |

| tags) | | | |
|--|---------------------------|-------------------------|---|
| unSubscribeTags (Context context, String appId, String appKey, String pushId, String tags) | 取消标签订阅 | 无 | onSubTagsStatus (Context context, SubTagsStatus subTagsStatus) |
| unSubscribeAllTags (Context context, String appId, String appKey, String pushId) | 取消所有标签订阅 | 无 | onSubTagsStatus (Context context, SubTagsStatus subTagsStatus) |
| checkSubScribeTags (Context context, String appId, String appKey, String pushId) | 获取标签列表 | 无 | onSubTagsStatus (Context context, SubTagsStatus subTagsStatus) |
| subScribeAlias (Context context, String appId, String appKey, String pushId, String alias) | 别名订阅 | 无 | onSubAliasStatus (Context context, SubAliasStatus subAliasStatus) |
| unSubScribeAlias (Context context, String appId, String appKey, String pushId, String alias) | 取消别名订阅 | 无 | onSubAliasStatus (Context context, SubAliasStatus subAliasStatus) |
| checkSubScribeAlias (Context context, String appId, String appKey, String pushId) | 获取别名 | 无 | onSubAliasStatus (Context context, SubAliasStatus subAliasStatus) |
| switchPush (Context context, String appId, String appKey, String pushId, boolean switcher) | 通知栏和透传开关同时转换 | 透传功能已停用 | onPushStatus (Context context, PushSwitchStatus pushSwitchStatus) |
| switchPush (Context context, String appId, String appKey, String pushId, int pushType, boolean switcher) | 通知栏或透传开关单独转换 | 透传功能已停用 | onPushStatus (Context context, PushSwitchStatus pushSwitchStatus) |
| checkPush (Context context, String appId, String appKey, String pushId) | 检查通知栏和透传开关状态 | 透传功能已停用，此方法在有无网络下都能成功返回 | onPushStatus (Context context, PushSwitchStatus pushSwitchStatus) |
| clearNotification (Context context) | 清除应用弹出的所有通知栏消息 | 无 | 无 |
| clearNotification (Context context, int notifyId) | 清除应用弹出的指定 notifyId 的通知栏消息 | 无 | 无 |

参数说明

appId: [Push 平台](#) 申请的应用 id。

appKey: [Push 平台](#) 申请的应用 key。

pushId: 在订阅接口的回调方法 onRegisterStatus 中，通过：registerStatus.getPushId(); 获取。

tags: 标签名称，多个逗号隔离，每个标签不能超过 20 个字符，限 100 个。

alias: 别名名称，长度不能超过 20 个字符，每一个应用用户仅能设置一个别名。

pushType: 消息类型，0：通知栏消息 1：透传消息。

switcher: 开关状态

notifyId: 由服务端生成的通知 Id，在通知展示的回调方法 onNotificationArrived 中，通过：mzPushMessage.getNotifyId(); 获取。

2.2 MzPushReceiver 回调方法说明

| 方法名称 | 方法说明 | 使用建议 |
|---|----------------|---|
| onRegister(Context context, String pushId) | 旧版订阅回调 | 已废弃 |
| onUnRegister(Context context, boolean success) | 旧版反订阅回调 | 已废弃 |
| onRegisterStatus(Context context, RegisterStatus registerStatus) | 订阅回调 | 无 |
| onUnRegisterStatus(Context context, UnRegisterStatus unRegisterStatus) | 反订阅回调 | 无 |
| onPushStatus(Context context, PushSwitchStatus pushSwitchStatus) | 通知栏和透传消息开关状态回调 | 无 |
| onSubTagsStatus(Context context, SubTagsStatus subTagsStatus) | 标签状态回调 | 无 |
| onSubAliasStatus(Context context, SubAliasStatus subAliasStatus) | 别名状态回调 | 无 |
| onUpdateNotificationBuilder(PushNotification Builder pushNotificationBuilder) | 通知栏图标设置 | 兼容旧版本的 Flyme 系统中设置消息弹出后状态栏中的小图标，同时请在相应的 drawable 不同分辨率文件夹下放置一张名称务必为 mz_push_notification_small_icon 的图片 |
| onNotificationClicked(Context var1, MzPushMessage mzPushMessage) | 通知点击回调 | 无 |
| onNotificationArrived(Context context, MzPushMessage mzPushMessage) | 通知展示回调 | 只有在应用进程存在时才会在此方法回调 |
| onMessage(Context context, String message, String platformExtra) | 透传消息回调 | 透传功能已停用 |

3 通知栏消息推送原理

3.1 通知的生成

在您的应用正确接入 PushSDK 并在平台上进行消息推送后，所有这些推送消息都是通过 Flyme 系统应用“推送服务”进行处理和展示。“推送服务”会在云端与客户端之间建立一条稳定、可靠的长连接，从而实现客户端应用实时接收推送平台下发的消息，而且它还充当着通知栏消息创造角色，在您客户端应用进程不在的情况下也能为其创造出通知栏消息。使得能有效地帮助您应用拉动用户活跃度，改善产品体验。

3.2 通知的点击

通知栏消息支持四种点击动作，分别是：打开应用主页、打开应用内页面、打开 URI 页面 以及 应用客户端自定义，如图。

Flyme 推送平台

首页 创建推送 数据统计 配置管理

推送通知 透传消息 分组推送 定时任务 桌面通知 快应用推送通知 快应用透传消息

① 任务备注

请选择

通知栏样式

安卓原生

图片

标题

请输入标题

内容

请输入内容

展开方式

标准

文本

大图

ACT文件

① 点击动作

请输入调起的应用包名

打开应用

打开应用页面

打开URI页面

应用客户端自定义

输入参数

输入值

—

添加参数

存为参数模板

使用参数模板

其中前三种打开方式都是通过构建 Intent 的方式，利用 startActivity 方法调起，代码如下：

```
Intent privateIntent = buildIntent(context(), message);
if(privateIntent != null){
    privateIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    try {
        context().startActivity(privateIntent);
    } catch (Exception e){
        DebugLogger.e(TAG, "Click message StartActivity error "+e.getMessage());
    }
}
```

3.2.1 打开应用主页 和 打开应用内页面

上述代码会在您接入的 PushSDK 内部，所以它还支持打开 exported 为 false 的内部 Activity，只要在平台上正确配置完整的 Activity 名称即可，比如：com.meizu.pushdemo.TestActivity。

参数传递

打开应用主页 和 打开应用内页面 都支持附加参数。上述代码中，buildIntent 方法内部解析了平台传递的消息打开类型和参数列表，通过 intent.putString("key", "value") 的方式在构建 Intent 时进行添加。

参数获取

当跳转到目标 Activity 时，可以通过如下方式获取在平台填写的参数值：

```
String value = getIntent().getStringExtra("key")
```

注意：点击通知栏的时候，除了获取用户自定义的参数，还可以获取平台 taskid 等参数，此参数为 SDK 传递的默认参数，不需在平台配置，如果需要 taskId 相关参数可以通过如下方式获取：

```
String platfromExtra = getIntent().getStringExtra("platform_extra");
```

该参数的格式如：{"task_id":"123456"}

3.2.2 打开 URI

打开 URI 可支持配置 https/http 的网页地址，也可支持您应用内部自定义的 URI。使用时要特别注意您应用是否全版本支持该 URI，以免造成在旧版本中点击通知栏无效的情况。

3.2.3 客户端自定义

该方式不会触发 startActivity 的逻辑，而是会在 MzPushReceiver 的 onNotificationClicked 回调方法的 MzPushMessage 参数中附带平台上配置的自定义内容，代码如下：

```
@Override
public void onNotificationClicked(Context context, MzPushMessage mzPushMessage) {
    String selfDefineContentString = mzPushMessage.getSelfDefineContentString();
}
```

4 历史变更和兼容

魅族推送服务经历几次大的变更，从之前的 C2DM，到现在可以完全脱离 Flyme 平台作为一种完全开放给第三方应用的 SDK，在这个阶段出现多种接入方式，给以后的应用接入带来极大的困扰，魅族 PushSDK 极力在减少 Flyme 版本迭代给应用接入带来的麻烦，但应用还是需要做细小的更改才能做到与低版本 Flyme 的兼容。

4.1 订阅和反订阅接口变更

过去版本中，订阅和反订阅接口是使用了 **PushManager** 的以下方法：

```
register(Context context);
unRegister(Context context);
```

对应的回调方法是 **MzPushReceiver** 的以下方法：

```
onRegister(Context context, String pushId)
onUnRegister(Context context, boolean success)
```

而目前版本中，订阅和反订阅接口已经换成了 **PushManager** 的以下方法：

```
register(Context context, String appId, String appKey)
unRegister(Context context, String appId, String appKey)
```

对应的回调方法是 **MzPushReceiver** 的以下方法：

```
onRegisterStatus(Context context, RegisterStatus registerStatus)
onUnRegisterStatus(Context context, UnRegisterStatus unRegisterStatus)
```

注意：旧接口代码依然保留在 SDK 中，但已经不再建议使用，如果您还使用着旧接口，建议尽快进行更新。

4.2 设置通知的状态栏小图标

在 **MzPushReceiver** 中存在 **onUpdateNotificationBuilder(PushNotificationBuilder pushNotificationBuilder)** 方法。在目前较新的 Flyme 系统已经不再需要专门进行状态栏图标的设置，此方法是作用于兼容旧版本 Flyme 系统中设置消息弹出后状态栏中的小图标。设置小图标代码如下：

```
@Override
public void onUpdateNotificationBuilder(PushNotificationBuilder pushNotificationBuilder) {
    pushNotificationBuilder.setmStatusBarIcon(R.drawable.mz_push_notification_small_icon);
}
```

注意：请在相应的 drawable 不同分辨率文件夹下放置一张名称**务必**为 `mz_push_notification_small_icon` 的图片。

4.3 透传功能回调

在 **MzPushReceiver** 中存在三个透传功能的回调方法：

```
onMessage(Context context, Intent intent)
onMessage(Context context, String message)
onMessage(Context context, String message, String platformExtra)
```

第一个带 `Intent` 参数的方法，是在 Flyme3 系统中处理透传消息使用。

第二个带一个 `String` 方法和 第三个带两个 `String` 方法会同时回调透传消息，带三个参数的方法中额外增加一个平台参数，格式如：`{"task_id":"1232"}`

注意：目前如果要处理透传消息的回调，只使用 `onMessage(Context context, String message, String platformExtra)` 方法即可。

4.4 透传功能停用

透传功能已经停止使用，文档中所有关于接入和介绍透传功能的地方，开发者接入时可以直接忽略。

4.5 声明权限

“接入步骤”的“声明权限”中可见，权限声明分 Flyme3 和 Flyme5 两个部分，按提示接入即可。

4.6 MzPushMessageReceiver action 声明

“接入步骤”的“注册消息接收的广播”中可见，广播注册中兼容 Flyme3 的两个 action 声明，按提示接入即可。

4.7 通知栏消息点击回调方法

过去版本中，`MzPushReceiver` 的 `onNotificationClicked` 方法是：

```
onNotificationClicked(Context context, String title, String content, String selfDefineContentString)
```

而目前版本中，方法是：

```
onNotificationClicked(Context context, MzPushMessage mzPushMessage)
```

4.8 通知栏消息展示回调方法

过去版本中，`MzPushReceiver` 的 `onNotificationArrived` 方法是：

```
onNotificationArrived(Context context, String title, String content, String selfDefineContentString)
```

而目前版本中，方法是：

```
onNotificationArrived(Context context, MzPushMessage mzPushMessage)
```

5 常见问题

问题 1：为什么订阅回调提示 AppID 不合法？

请先确定您接入的 SDK 是 Flyme 推送还是集成推送，以及使用的 AppID 是 Flyme 推送的还是集成推送的。
Flyme 推送：

- 后台：<http://push.meizu.com/>
- SDK：<http://open-wiki.flyme.cn/doc-wiki/index#id?74>

- 文档: <http://open-wiki.flyme.cn/doc-wiki/index?id?129>
- Demo: <https://github.com/MEIZUPUSH/PushDemo>

集成推送（集成了：魅族、华为、小米、OPPO 等多平台推送）：

- 后台: <http://mzups.meizu.com/>
- SDK: <http://open-wiki.flyme.cn/doc-wiki/index?id?58>
- 文档: <http://open-wiki.flyme.cn/doc-wiki/index?id?57>
- Demo: https://github.com/comsince/ups_meizu_pushsdk

问题 2：为什么执行了订阅后一直没有收到广播回调？

1. 请检查您手机网络是否设置了代理、是否稳定畅通，尝试切换网络后重试。
2. MzPushMessageReceiver 广播中的回调方法里 onRegister 方法已经废除，正常情况下会在 onRegisterStatus 方法中回调，请检查是否使用错误。
3. 请检查接入 PushSDK 的过程是否存在错误（可参考文档中“接入步骤”），包括：AndroidManifest.xml 中权限的声明、广播的定义，广播必须继承 MzPushMessageReceiver，如下图。

3.2 声明权限

在您工程AndroidManifest.xml中进行以下权限的声明：

```
<!-- 兼容Flyme5以下版本，魅族内部接入PushSDK必填，不然无法收到消息-->
<uses-permission android:name="com.meizu.flyme.push.permission.RECEIVE" />
<permission android:name="【替换您的包名】.push.permission.MESSAGE" android:protectionLevel="signature"/>
<uses-permission android:name="【替换您的包名】.push.permission.MESSAGE" />
<!-- 兼容Flyme3配置权限-->
<uses-permission android:name="com.meizu.c2dm.permission.RECEIVE" />
<permission android:name="【替换您的包名】.permission.C2D_MESSAGE" android:protectionLevel="signature" />
<uses-permission android:name="【替换您的包名】.permission.C2D_MESSAGE"/>
```

注意：请将以上【替换您的包名】替换成您自己App的包名。

3.3 注册消息接收的广播

在您工程AndroidManifest.xml中进行消息接收广播的声明：

```
<!-- push应用定义消息receiver声明 -->
<receiver android:name="【替换您的包名】.MyPushMsgReceiver">
    <intent-filter>
        <!-- 接收push消息 -->
        <action android:name="com.meizu.flyme.push.intent.MESSAGE" />
        <!-- 接收register消息 -->
        <action android:name="com.meizu.flyme.push.intent.REGISTER.FEEDBACK" />
        <!-- 接收unregister消息-->
        <action android:name="com.meizu.flyme.push.intent.UNREGISTER.FEEDBACK"/>
        <!-- 兼容低版本Flyme3推送服务配置 -->
        <action android:name="com.meizu.c2dm.intent.REGISTRATION" />
        <action android:name="com.meizu.c2dm.intent.RECEIVE" />
        <category android:name="【替换您的包名】" />
    </intent-filter>
</receiver>
```

注意：请将以上【替换您的包名】替换成您自己App的包名。

3.4 实现消息接收的广播

创建消息接收的广播类，名称如：MyPushMsgReceiver，实现如下：

```
public class MyPushMsgReceiver extends MzPushMessageReceiver {
```

- 4. 不要在您的 App 中去实现多个 MzPushMessageReceiver，因为只会回调其中一个。
- 5. 手机【系统设置】-【应用管理】-【所有应用】点击右上角【显示系统服务应用】找到【推送服务】和【您自己的 App】，如下图，分别进行“清除数据”，然后重启手机，待手机启动后再次执行一次订阅操作。



问题 3：为什么一直无法收到消息，该如何定位？

- 1. 先检查消息是否被放进了通知栏右上角收纳盒子里，如下图红圈位置。一般地当 App 多次消息到达到都没有对其进行点击，消息就会自动收进收纳盒里。若要恢复，可在收纳盒里长按消息选择“不再收纳”。



2. 在 [Flyme 推送平台后台](#) 【配置管理】- 【问题排查】中按从上往下步骤进行相应的状态查询，如下图。

- 【设备对应关系查询】输入手机的 IMEI 点击“查询”按钮获得相应的 PushId，若提示“PushId 未注册”，请执行 PushSDK 中 `PushManager.register(Context context, String appId, String appKey)` 进行推送订阅；
 - 【设备是否在线查询】输入刚才获得的 PushId，点击“查询”按钮查询状态，若手机处理离线状态，解决方法请见“问题 4”；
 - 【设备是否订阅查询】输入刚才获得的 PushId，点击“查询”按钮查询订阅信息：
“是否已经订阅”，若未订阅请执行 PushSDK 中 `PushManager.register(Context context, String appId, String appKey)` 方法进行推送订阅；
“通知栏开关”，若为关，请执行 PushSDK 中 `PushManager.switchPush(Context context, String appId, String appKey, String pushId, int pushType, boolean switcher)` 方法进行打开；
“系统通知栏开关”，若为关，解决方法请见“问题 5”；
 - 【推送测试】中输入刚才获得的 PushId，并点击“推送”按钮，下方显示“已推送 msgId: xxx”代表已经成功发送测试通知。
- 弹出消息时是否存在 `Invalid notification (no valid small icon)` 异常日志输出？这是设置通知栏图标异常，如果开启了像 AndResGuard 之类的资源路径混淆，尝试在 `whiteList` 中添加：
`R.drawable.stat_sys_third_app_notify`。
 - 在较老的 Flyme 系统也出现 `Invalid notification (no valid small icon)` 异常的话，还可以在 `drawable` 不同分辨率文件夹下放置一张名为 `mz_push_notification_small_icon` 的图片，并在 `onUpdateNotificationBuilder` 回调方法中按文档说明进行设置通知栏小。

问题 4：为什么手机一直连着网络，但还是显示处于离线状态？

手机离线状态并不是指没连网络，而是推送服务的客户端跟推送服务器无法建立长连接，常见于网络不稳动或者开发过程中，可按以下每个步骤进行修复。

1. 尝试断开网络再进行重连或者移动网络和 Wi-Fi 网络互相切换一下，再重试。
2. 再次执行一次订阅操作，再重试。
3. 重启手机，再重试。
4. 手机【系统设置】-【应用管理】-【所有应用】点击右上角【显示系统服务应用】找到【推送服务】和【您自己的 App】（操作如“问题 2”中第 5 点插图），分别进行“清除数据”，然后重启手机，待手机启动后再次执行一次订阅操作，再重试。
5. 查看您手机 Flyme 版本，对于 Flyme5 或以下较老的系统若进行以上操作后还是处于离线状态，那么等待几分钟后重试，同时建议对手机系统进行升级。

问题 5：问题排查中，“系统通知栏开关”是关闭状态，该如何打开？

打开您手机中【手机管家】-【权限管理】-【通知管理】，找到您的 App，把“通知消息”勾上，然后在手机【系统设置】-【应用管理】-【所有应用】点击右上角【显示系统服务应用】找到【推送服务】和【您自己的 App】（操作如“问题 2”中第 5 点插图），进行“清除数据”，然后重启手机，待手机启动后再次执行一次订阅操作，这样便会触发系统通知栏开关状态的上传，完成操作后再到问题排查中查看状态是否发生变化。

问题 6：为什么点击消息后，不能打开应用页面？

先查看输出日志中是否存在：`Click message StartActivity error` 或者 `android.content.ActivityNotFoundException` 异常。Flyme 推送支持打开内部非对外的 Activity，只要在平台上配置好完整的名称即可，同时请确保名称拼写正确以及名称前后不能含有空格等特殊字符。

问题 7：Debug 版本正常但在 Release 版本中报异常是什么原因？

常见于数据反序列化过程中异常，请检查 Release 中是否对 APK 进行了加固或者字符串的混淆之类的操作，如果是请对 `com.meizu.cloud` 进行过滤。

问题 8：接入 PushSDK 后编译不过什么原因？/ PushSDK 有离线包吗？

如果由于各种原因不能使用 jcenter 依赖，还可以直接下载 AAR 包进行手动集成：[点击下载](#)。

问题 9：PushID 会在什么场景下发生变化？

手机卸载了 App 或者 App 不活跃一个月。

问题 10：收到的推送消息可以对其进行删除吗？

不支持使用常规通知栏方法 `NotificationManager.cancel()` 删除，但可以使用 `PushManager.clearNotification(Context context, int notifyId)` 进行删除。

问题 11：通知到达后，会收到回调吗？

不一定，当通知到达后且 App 进程存在的情况下才会收到 `MzPushMessageReceiver` 的 `onNotificationArrived` 回调。

问题 12：接入 PushSDK 后，APK 会增大多少？

APK 会比原来增加一百多 K。

问题 13：使用透传功能提示超出了限制是什么原因？

已不再支持透传功能。

问题 14：推送支持角标设置吗？

不支持。

问题 15：推送消息支持自定义提示音吗？

不支持。

问题 16：为什么会出现熄屏状态下无法收到通知栏消息？

该异常会在较老的 Flyme 系统中发生，建议对手机系统进行升级。