# 慧推 SDK 集成指南

目录

# 1、简介

慧推 SDK 是由百度安全实验室自主研发的，专门为产品业务方输出服务功能的协同解决方案。其宗旨是在全网移动应用间建立互信共赢的生态联盟，解决产品因被”强停”或其他主客观因素所导致的 app 沉默，从而影响留存用户价值最大化等问题。

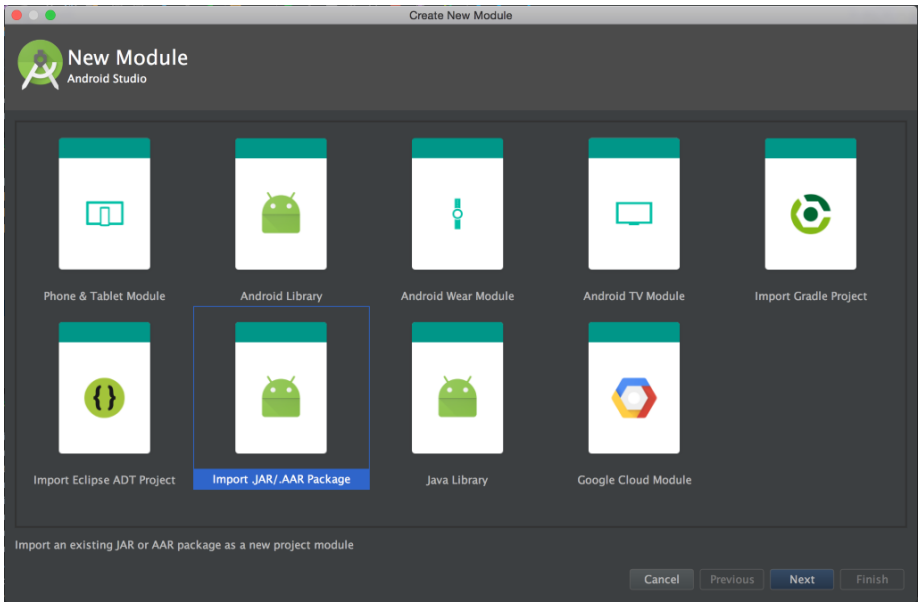
# 2、版本说明

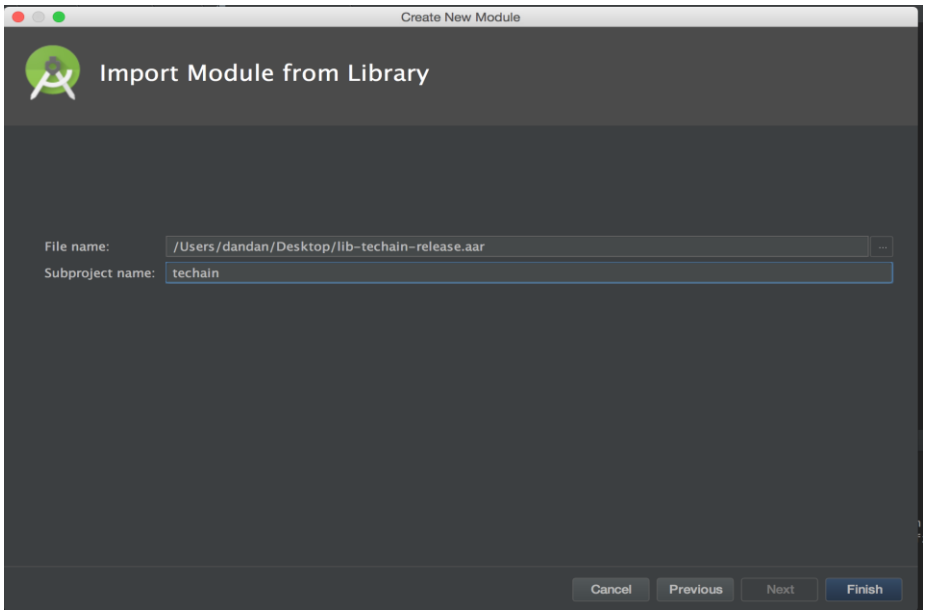
| | |
|---|---|
| 发布日期 | 2022/9/10 |
| 版本号 | 3.5.9.9 |
| 系统支持 | 支持 Android2.3 及以上版本 |
| 更新说明 | 3.3.9.8：<br>aar 中开始携带所有 abi 的 so 包，app 使用 abiFilter 自行过滤，以简化集成过程。<br>在多进程使用本 SDK 时，需要配置到独立进程中的组件增加了 com.baidu.techain.THProvider<br>3.3.9.8.2：<br>增加一个接口,由宿主设置用户是否同意了隐私协议。<br>3.4.2.0:<br>增加一个接口,由用户设置其自身定义的设备 Id。<br>3.5.5.0<br>同意隐私协议前,所有调用都不再生效。初始化过程会被中止，直到同意隐私协议后再恢复。<br>3.5.9.1.2<br>优化功能<br>增加通知权限引导、富媒体通知的功能<br>3.5.9.9<br>功能优化，bug 修改 |

# 3、客户端集成步骤(AndroidStudio)

## 集成协同 sdk

创建一个新的 Module，导入我们提供的 AAR 文件，这里以 techain*.aar 为例，module 取名：techain（名称任意，这里只是举例）,如下图所示：




在自己项目的 build.gradle 中添加依赖，引入 techain 依赖，然后 sync。

```
dependencies {
      ...
            compile project(':techain')
               ...
      }
```

so 包相关：aar 包中包含了 armeabi，armeabi-v7a,arm64-v8a,x86 的 so 包，如果集成的 App 自身兼容的 abi 种类少于本 aar，应当在 build.gradle 文件中，加入 ndk 的 abiFilter 配置，选定本 app 支持的 abi，以防止引入全部 4 种 abi 的 so 导致在指定 abi 上缺失其他 so 文件。

```
    defaultConfig {

      ...
      ndk {
        abiFilter "armeabi"
      }
      ...
    }
```

# 4、API 接口使用说明

以下方法调用必须在同一进程中，否则会发生错误。请注意在多进程情况下确保只在主进程初始化本 SDK.

## 通用结果码

部分接口的结果码包括以下通用部分，在使用这些通用结果码的接口中，标注了“其他结果码见通用结果码。”：
-5：未绑定的设备。
-4：内部 Json 格式错误。
-3：网络异常导致的失败。
-2：内部逻辑错误。
-1：其他错误。

## 通用错误码

部分接口传入的 Callback 的 onError 方法会回调错误码，在这些接口标注了“错误码含义见通用错误码。”：
1：接口方法名为空
2：未找到对应的接口方法
3：内部错误
4：慧推 SDK 初始化失败
12：因未设置同意隐私协议，导致的调用失败。

## 初始化

程序启动后需要在在应用的 Application 类的 onCreate 中调用慧推 SDK 初始化代码:

```
TH.init(Context context, String  techain_appkey, String  techain_seckey, 100028, 100019);
```

如果在同意隐私协议前调用初始化接口，初始化过程会被中止，并在调用同意隐私协议方法后恢复。

参数说明：
String techain_appkey 和 String techain_seckey 用于服务器的访问校验，这是分配给渠道集成我们 SDK 的凭证。这两个值在测试环境和正式环境是不一样的。集成之前请向我们申请。这两个 key 是和集成宿主的包名和签名唯一关联，测试接口时请确保宿主的签名信息和申请这两个 key 时的签名一致。

注：key 测试环境和正式环境是不一样的，集成之前需申请正式 key。

## 延迟初始化

SDK 支持在程序启动后在应用的 Application 类的 onCreate 中调用延迟初始化代码，根据传入的参数使初始化行为延迟一段时间开始,注意该接口与上面的初始化接口只能调用一个。

```
TH.initDelay(Context context, int delaySeconds,String techain_appkey,
String techain_seckey, 100028,100019)
```

参数说明：
int delaySeconds 用于延迟初始化的时间，单位是秒，初始化行为将于 delaySeconds 秒后开始执行。
String techain_appkey 和 String techain_seckey 用于服务器的访问校验，这是分配给渠道集成我们 SDK 的凭证。

注：key 测试环境和正式环境是不一样的，集成之前需申请正式 key。

## 绑定设备 ID

通过调用此接口绑定宿主应用定义的设备ID,用于ID映射。

```
TH.setDid(Context context, String did)
```

**参数说明：**

- String did 为宿主应用用于标识本设备的唯一 Id 值。请尽早绑定设备 ID 以方便进行 ID 映射（可以在调用初始化接口前）。

## 设置用户同意隐私协议接口

通过调用此接口告知慧推SDK用户是否同意了隐私协议。

在同意隐私协议前，所有对本SDK的调用将不会生效，具有返回值的方法会获取到空的返回值，具有错误码的接口会接收到错误码12（见通用错误码），初始化过程会在delay结束后被中止，直到同意了隐私协议。

```
TH.setAgreePolicy(Context context, boolean agree)
```

**参数说明：**

- boolean agree 为用户是否同意了隐私协议，true 为同意，false 为不同意。

## 设置通知状态栏小图标

```
TH.tinvoke(100019, "ssi",new Class[] { int.class }, R.drawable.b_b);
```

该方法用于设置通知状态栏小图标。
参数说明：
R.drawable.b_b：需要设置的图标文件资源 id。

## 设置静默时间段

```
TH.tinvoke(100019, "setSilentTime",new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        return null;
    }

    @Override
    public Object onError(Object...arg0) {
        return null;
    }
},new Class[] { int.class,int.class }, int startHour,int durationHour);
```

该方法用于设置静默时间。请于 init 方法后调用。若不设置静默时间，默认静默时间为晚 11 点到次日 7 点。

参数说明：

"setSilentTime" 是慧推 SDK 的接口名称。

```
new Callback() {
  @Override
```

```
    public Object onEnd(Object...arg0) {
        return null;
    }

    @Override
    public Object onError(Object...arg0) {
        return null;
    }
}
```

是 com.baidu.techain.ac.Callback 的一个实例对象，用于接收该方法的执行结果。
CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 boolean 型的结果，true 表示设置成功，false 表示设置失败。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

new Class[] { int.class,int.class} 是慧推 SDK 接口的参数列表。

int startHour 是静默时间的开始时间值，24 小时制，如传入 13 表示静默时间从下午 1 时开始。
int durationHour 是静默时间的持续时间，单位为小时，如静默 2 小时则传入 2。

## 检查 Push 在线状态

```
TH.tinvoke(100019, "isPushEnabled",new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        return null;
    }

    @Override
    public Object onError(Object...arg0) {
        return null;
    }
});
```

该方法用于检查当前 Push 在线状态。请于 init 方法后调用。

参数说明：

"isPushEnabled " 是慧推 SDK 的接口名称。

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 boolean 型的结果，true 表示当前 Push 服务在线，false 表示当前 Push 服务不在线。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

## 获取 Push Uid

```
TH.tinvoke(100019, "getPushUid",new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        return null;
    }

    @Override
    public Object onError(Object...arg0) {
        return null;
    }
});
```

该方法用于获取当前设备的 Push Uid，该 Id 用于对设备进行唯一映射。

参数说明：
"getPushUid" 是慧推 SDK 的接口名称。

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 String 型的结果，值为当前设备的 Push Uid。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

## 开启/关闭 Push 服务

```
TH.tinvoke(100019, "setPushActive", new Callback() {

    @Override
```

```
    public Object onEnd(Object...arg0) {
        System.out.println("setPushActive onEnd:" + arg0[0]);
        return super.onEnd(arg0);
    }

    @Override
    public Object onError(Object...arg0) {
        System.out.println("setPushActive onError:" +arg0[0]);
        return super.onError(arg0);
    }

}, new Class[] { boolean.class }, boolean isActive);
```

该方法用开启或关闭当前的 Push 服务状态，默认为开启。
如果不确认当前的服务开启或关闭状态，请调用该接口将状态设置为希望的值。

参数说明：
" setPushActive " 是慧推 SDK 的接口名称。

new Class[] { boolean.class }是慧推 SDK 接口的参数列表。

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 String 型的结果，值为 Integer 类型的结果码。结果码含义如下：
2：Push 服务被设置为开启。
1：Push 服务被设置为关闭。
其他结果码见通用结果码。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

isActive 是要设置的 Push 服务开启/关闭状态的值。

## 开启/关闭调试信息

```
TH.tinvoke(100019, "setDebug", new Class[] { boolean.class }, boolean debug);
```

该方法用开启或关闭调试信息输出。如果不调用该方法，默认值为 false。
将 debug 开关设为 true 之后,将会在 logcat 中输出 TAG 为“PUSH_SDK”的调试信息。

参数说明：

"setDebug" 是慧推 SDK 的接口名称。

new Class[] { boolean.class }是慧推 SDK 接口的参数列表。

debug 是要设置的调试信息开关的值。

## 设置别名

```
TH.tinvoke(100019, "setAlias", new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        Log.d("PUSH_SDK", "onEnd:" + arg0[0]);
        return super.onEnd(arg0);
    }

    @Override
    public Object onError(Object...arg0) {
        Log.d("PUSH_SDK", "onError:" + arg0[0]);
        return super.onError(arg0);
    }
}, newClass[] { String.class },String alias);
```

该方法用设置别名，一台设备只能设置一个别名

参数说明：
" setAlias " 是慧推 SDK 的接口名称。

new Class[] { String.class }是慧推 SDK 接口的参数列表。

alias 是要设置的别名。

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 Integer 类型的结果码。结果码含义如下：
0 - 成功
10000 - 失败：设备未注册
100001 - 绑定失败：别名格式不符合规范

其他结果码见通用结果码。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

## 获取别名

```
TH.tinvoke(100019, "getAlias", new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        Log.d("PUSH_SDK", "onEnd：" + arg0[0]);
        return super.onEnd(arg0);
    }

    @Override
    Public Object onError(Object...arg0) {
        Log.d("PUSH_SDK", "onError：" + arg0[0]);
        return super.onError(arg0);
    }
}, newClass[] {});
```

该方法用获取别名。

参数说明：
"getAlias" 是慧推 SDK 的接口名称。

new Class[] {  }是慧推 SDK 接口的参数列表。

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 Pair<Integer, String>型的结果，Integer 为结果码,String 为别名。
结果码含义如下：
0 - 成功
10000 - 失败：设备未注册
110001 - 查询别名出错，原因：设备尚未绑定别名
其他结果码见通用结果码。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

## 删除别名

```
TH.tinvoke(100019, "deleteAlias", new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        Log.d("PUSH_SDK", "onEnd：" + arg0[0]);
```

```
        return super.onEnd(arg0);
    }

    @Override
    public Object onError(Object...arg0) {
        Log.d("PUSH_SDK", "onError：" + arg0[0]);
        return super.onError(arg0);
    }
}, newClass[] {});
```

该方法用删除别名。

参数说明：
"deleteAlias" 是慧推 SDK 的接口名称。

new Class[] { }是慧推 SDK 接口的参数列表。

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 Integer 型的数字结果码。
结果码含义如下：
0 - 成功
10000 - 失败：设备未注册
100001 - 绑定失败：别名格式不符合规范
其他结果码见通用结果码。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

## 添加标签

```
TH.tinvoke(100019, "addTags", new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        Log.d("PUSH_SDK", "onEnd：" + arg0[0]);
        returnsuper.onEnd(arg0);
    }

    @Override
    Public Object onError(Object...arg0) {
        Log.d("PUSH_SDK", "onError：" + arg0[0]);
        return super.onError(arg0);
    }
}, newClass[] {String[].class},new Object[]{String[] tags});
```

该方法用添加标签，可以添加多个标签。

参数说明：
"addTags" 是慧推 SDK 的接口名称。

new Class[] { String[].class}是慧推 SDK 接口的参数列表。
tags 是一个 String[]，存放要添加的标签。

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 Integer 型的数字结果码。
结果码含义如下：
0 - 成功
10000 - 失败：设备未注册
200001 - 失败，标签不符合规范
200002 - 失败，此设备标签数量超过上限
200003 - 失败，增加或更新的标签为空
200004 - 失败，每次最多设置和更新 100 个标签

其他结果码见通用结果码。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

## 删除标签

```
TH.tinvoke(100019, "deleteTags", new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        Log.d("PUSH_SDK", "onEnd：" + arg0[0]);
        return super.onEnd(arg0);
    }

    @Override
    public Object onError(Object...arg0) {
    Log.d("PUSH_SDK", "onError：" + arg0[0]);
    return super.onError(arg0);
    }
}, newClass[] {String[].class},new Object[]{String[] tags});
```

该方法用删除标签，可以删除多个标签。

参数说明：
"deleteTags" 是慧推 SDK 的接口名称。

new Class[] { String[].class }是慧推 SDK 接口的参数列表。
tags 是一个 String[]，存放要删除的标签。

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 Integer 型的数字结果码。
结果码含义如下：
0 - 成功
10000 - 失败：设备未注册
200001 - 失败，标签不符合规范
200002 - 失败，此设备标签数量超过上限
200003 - 失败，增加或更新的标签为空
200004 - 失败，每次最多设置和更新 100 个标签

其他结果码见通用结果码。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

## 更新标签

```
TH.tinvoke(100019, "updateTags", new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        Log.d("PUSH_SDK", "onEnd：" + arg0[0]);
        return super.onEnd(arg0);
    }

    @Override
    Public Object onError(Object...arg0) {
        Log.d("PUSH_SDK", "onError：" + arg0[0]);
        return super.onError(arg0);
    }
}, newClass[] {String[].class},new Object[]{String[] tags});
```

该方法用更新标签，更新标签会清空原有所有标签，并更换为新设的标签列表。

参数说明：
"updateTags" 是慧推 SDK 的接口名称。

new Class[] { String[].class }是慧推 SDK 接口的参数列表。
tags 是一个 String[]，存放要更新的标签。

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 Integer 型的数字结果码。
结果码含义如下：
0 - 成功
10000 - 失败：设备未注册
200001 - 失败，标签不符合规范
200002 - 失败，此设备标签数量超过上限
200003 - 失败，增加或更新的标签为空
200004 - 失败，每次最多设置和更新 100 个标签

其他结果码见通用结果码。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

## 清空标签

```
TH.tinvoke(100019, "cleanTags", new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        Log.d("PUSH_SDK", "onEnd：" + arg0[0]);
        return super.onEnd(arg0);
    }

    @Override
    public Object onError(Object...arg0) {
        Log.d("PUSH_SDK", "onError：" + arg0[0]);
        return super.onError(arg0);
    }
});
```

该方法用清空标签列表

参数说明：
"cleanTags " 是慧推 SDK 的接口名称。

new Class[] { }是慧推 SDK 接口的参数列表。

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 Integer 型的数字结果码。
结果码含义如下：
0 - 成功
10000 - 失败：设备未注册

其他结果码见通用结果码。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

## 查询单个标签是否绑定

```
TH.tinvoke(100019, "isTagBinding", new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        Log.d("PUSH_SDK", "onEnd：" + arg0[0]);
        return super.onEnd(arg0);
    }

    @Override
    public Object onError(Object...arg0) {
        Log.d("PUSH_SDK", "onError：" + arg0[0]);
        return super.onError(arg0);
    }
}, newClass[] {String.class},String tag });
```

该方法用来查询单个标签是否已经绑定

参数说明：
"isTagBinding " 是慧推 SDK 的接口名称。

new Class[] { String.class}是慧推 SDK 接口的参数列表。
tag 表示要查询的标签
CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 Integer 型的数字结果码。
结果码含义如下：
0 - 成功 & 设备绑定所查询的标签
10000 - 失败：设备未注册
200001 - 失败，标签不符合规范

210001 - 查询单个标签，设备未绑定所查询的标签
其他结果码见通用结果码。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

## 查询所有标签

```
TH.tinvoke(100019, "getAllTags", new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        Log.d("PUSH_SDK", "onEnd：" + arg0[0]);
        return super.onEnd(arg0);
    }

    @Override
    public Object onError(Object...arg0) {
        Log.d("PUSH_SDK", "onError：" + arg0[0]);
        return super.onError(arg0);
    }
});
```

该方法用获取标签列表

参数说明：
"getAllTags " 是慧推 SDK 的接口名称。

new Class[] { }是慧推 SDK 接口的参数列表。

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 Pair<Integer,ArrayList<String>>型,其中 Integer 代表结果码，ArrayList<String>代表标签列表。
结果码含义如下：
0 - 成功
10000 - 失败：设备未注册
其他结果码见通用结果码。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

## 判断当前应用是否拥有通知栏权限

```
TH.tinvoke(100019, "areNotificationsEnabled", new Callback() {
    @Override
    public Object onEnd(Object...arg0) {
        Log.d("PUSH_SDK", "onEnd: " + arg0[0]);
        return super.onEnd(arg0);
    }

    @Override
    public Object onError(Object...arg0) {
        Log.d("PUSH_SDK", "onError: " + arg0[0]);
        return super.onError(arg0);
    }
});
```

该方法用获取是否有通知栏

参数说明：
"areNotificationsEnabled" 是慧推 SDK 的接口名称。

new Class[] { }是慧推 SDK 接口的参数列表。

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 Boolean 类型结果码
结果码含义如下：
true - 成功。
其他结果码见通用结果码。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

## 设置应用处于前台状态下，能否显示通知栏

```
 TH.tinvoke(100019, "setNotificationEnableInForeground",new Class[]
{ boolean.class }, false);
```

参数说明：
"areNotificationsEnabled" 是慧推 SDK 的接口名称。
new Class[] { boolean.class } 参数列表
false 代表最终设置的参数

## 设置通知栏最大显示的条数

```
TH.tinvoke(100019, "setMaxNotificationCount", new Class[] { int.class }, 3);
```

参数说明：

" setMaxNotificationCount " 是慧推 SDK 的接口名称。

new Class[] { int.class } 参数列表

3 代表最终设置的参数

## 设置自定义 DeviceId

```
TH.tinvoke(100019, "setHostDeviceId", new Callback() {
@Override
public Object onEnd(Object...arg0) {
Log.d("PUSH_SDK", "onEnd：" + arg0[0]);
return super.onEnd(arg0);
}
@Override
public Object onError(Object...arg0) {
Log.d("PUSH_SDK", "onError：" + arg0[0]);
return super.onError(arg0);
}
}, newClass[] {String.class},String deviceId});
```

该接口是用于业务方自己上传自定义设备 id 来进行指定自定义设备 id 推送

参数说明：

" setHostDeviceId" 是慧推 SDK 的接口名称。

new Class[] { String.class } 参数列表

deviceId 表示要设置的 ID

CallBack 的 onEnd 方法会在执行结束后在其 arg0[0]位置上返回 Integer 型的数字结果码。

结果码含义如下：

0 - 成功

其他结果码见通用结果码。

CallBack 对象的 onError 方法会在执行异常时收到 Integer 型的错误码，错误码为其参数 arg0[0]。错误码含义见通用错误码。

# 用户自定义 Receiver

用户需要自定义一个 BroadcastReceiver 接收指定的 Intent，来进行推送相关的事件处理：

请在 AndroidManifest 文件中配置该 Receiver 如下，并填充 Receiver 名称和 category 标签：

```
<receiver
        android:name=" YourReceiver Name "
        android:exported="false" >
<intent-filter>
<action android:name="com.baidu.techain.push.action.PUSH_EVENT" />
<category android:name=" Your package Name " />
</intent-filter>

</receiver>
```

该 BroadcastReceiver 接收到此 action 的 Intent 后：
调用 intent.getExtras()获取 Bundle，从 Bundle 中获取内容字段。其中”event_type”字段的 int 值表示本条事件的类型，该值含义和该含义下其他内容字段如下：

1：Push 服务启动成功。每次连接成功时会接收到此事件。
　　其他字段：
　　”push_uid” ：String 型，本设备推送识别 Id，请自行保存，用于关联到该设备。
2：推送服务在线状态改变。每次在线状态改变时会接受到此事件。
　　其他字段：
　　“conn_status”：boolean 型，当前连接状态，true 为在线，false 为不在线。
3：接收到透传消息。
　　其他字段：
　　“id” ：String 型，消息 id。
　　“description” ：String 型，描述。
　　“content” ：String 型，消息内容。
4：接收到一个通知栏消息。
　　其他字段：
　　“id” ： String 型，消息 id。
　　“title” ：String 型，通知标题
　　“content” ：String 型，通知内容
　　“extra” ：String 型 Json 数据，自定义参数键值对。
5：通知栏被点击。
　　其他字段：

“id” ： String 型，被点击通知栏对应的消息 id。
“item” ：int 型，被点击的物件：
0：通知栏
“title” ：String 型，通知标题
“content” ：String 型，通知内容
“extra” ：String 型 Json 数据，自定义参数键值对。

## 通知点击--自定义参数获取方式

以下提供通知点击获取自定义参数方式，详情如下：
用户可在目标 Activity#onCreate 方法中获取 Intent，通过 Intent 来获取平台设置的相关自定义参数

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Intent intent = getIntent();
    if (intent != null){
        String value = intent.getStringExtra(“自定义参数的 key”);
    }

}
```

# 5、多进程支持

协同 SDK 支持同一宿主下多进程运行，默认是运行在主进程中。
a) 如果需要配置协同 SDK 运行在其它进程，请自行在自身的 AndroidManifext 配置 SDK 所需要的组件。如下组件都需要通过 android:process 属性配置在同一个进程中。

```
<activity
    android:name="com.baidu.techain.TechainActivity "
    android:exported="true"
    android:theme="@style/BD_TranslucentTheme"
    android:excludeFromRecents="true"
    android:launchMode="standard">
    <intent-filter>
        <action android:name="com.baidu.action.Techain.VIEW" />
        <category android:name="com.baidu.category.techain"/>
        <category android:name="android.intent.category.DEFAULT"/>
```

```
      </intent-filter>
      <meta-data android:name="acsr"
          android:value="t">
    </meta-data>
</activity>

<provider android:authorities="应用包名+.techain.ac.provider"
android:name="com.baidu.techain.TechainProvider" android:exported="false"/>

<service
      android:name="com.baidu.techain.TechainService"
      android:exported="false">
      <intent-filter>
          <action android:name="com.baidu.action.Techain.VIEW" />
           <category android:name="com.baidu.category.techain"/>
           <category android:name="android.intent.category.DEFAULT"/>
      </intent-filter>
</service>
<receiver
   android:name="com.baidu.techain.THReceiver"
     android:exported="true"/>
<service
   android:name="com.baidu.techain.THService"
   android:exported="true"/>
<provider
   android:name="com.baidu.techain.THProvider"
   android:authorities="应用包名+.techain.th.provider"
   android:exported="true" />
```

# 6、资源混淆说明

集成方工程如存在资源混淆情况，请根据自身配置对协同 SDK 内资源进行 keep，具体如下：

1. 需要 keep 的资源文件：

```
file n_b_s0.xml
file n_b_s1.xml
file n_b_s2.xml
file n_b_s3.xml
file a_a.xml
file b_b.png
```

2.需要 keep 的 a_a.xml 内的 ID 值

```
id BD_TranslucentTheme
id b_b
id a_a
id n_b_s0
id n_b_s1
id n_b_s2
id n_b_s3
id push_custom_notification_layout
id style_0
id notification_pic_0
id notification_timer_0
id notification_title_0
id notification_text_0
id style_1
id notification_pic_1
id notification_timer_1
id notification_title_1
id notification_text_1
id style_2
id notification_pic_2
id style_3
id notification_pic_3
id notification_btn_3
id notification_title_3
id notification_text_3
id style_4
id notification_pic_4
id notification_timer_4
id notification_title_4
id notification_text_4
id notification_lbtn_4
id notification_rbtn_4
id style_5
id notification_pic_5
id notification_title_5
id notification_progress_5
```

# 7、集成成功检查

调用初始化接口，数秒后在用户自定义 Receiver 中接收到 event_type 为 1 的广播事件，说明集成成功。
若无法接收到该事件，请检查网络是否连通。并可以调用检查 Push 在线状态接口，根据其 onError 的返回值排查原因。