# 翻滚吧插件库使用文档

- 在根目录的build.gradle的最末尾加上：

```
ext {
    minSdkVersion = 15
    targetSdkVersion = 23
    compileSdkVersion = 23
    buildToolsVersion = '23.0.2'

    gson = '2.6.1'
}
```

（数字可以根据自己的实际版本做修改。）
（如无特别说明，后面提到的所有『**调用**』都是通过`PluginEngine.getInstance()`进行）

- 依赖`PluginCommunicateEngine`工程
- 应用打开时调用`init()`初始化，推荐在`Application`的`onCreate()`里面进行
- 每次都调用`start()`开始插件通信，在`onEngineConnected()`回调里面，通过传入的参数`PluginExecutor`进行发送命令操作。
  发送的`code`（指令）从`PluginEngine`里面获取;
  发送的`message`通过`HostDataBuilder`构造，可参看末尾示例。
  （`start()`可以反复调用）
- 调用`registerReceiver()`接收核心层的广播数据
- 页面关闭时调用`unregisterReceiver()`取消接收数据
- 在程序退出时调用`quit()`反注册整个插件通信。

# 示例代码

## 1、打开某个插件

```
PluginEngine.getInstance().start(new PluginEngine.IPluginConnection() {
            @Override
            public void onEngineConnected(PluginEngine.PluginExecutor pluginExecutor) {
                try {
                    //Bundle可以不要
                    Bundle bundle = new Bundle();
                    bundle.putString("key_extra", "这是通过插件通信打开的插件");

                    String data = HostDataBuilder.startPluginBuilder()
                            .packageName(getPackageName()) //指定插件的packageName，getPackageName() 也就是打开自己本身，这里只是做个示范
                            .build();

                    pluginExecutor.send(PluginEngine.CODE_START_PLUGIN, data, bundle);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }

            @Override
            public void onEngineDisconnected() {

            }
        });
```

## 2、打开插件的指定页面

```
PluginEngine.getInstance().start(new PluginEngine.IPluginConnection() {
            @Override
            public void onEngineConnected(PluginEngine.PluginExecutor pluginExecutor) {
                try {
                    //Bundle可以不要
                    Bundle bundle = new Bundle();
                    bundle.putString("test", "这是给插件页面的额外数据");
```

```
                    String data = HostDataBuilder.startPluginBu
ilder()
                            .activityName("com.techjumper.plugi
nappdemo.SecondActivity") //页面完整的包名
                            .packageName(getPackageName()) //指
定插件的packageName
                            .build();

                    pluginExecutor.send(PluginEngine.CODE_START
_PLUGIN_ACTIVITY, data, bundle);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }

            @Override
            public void onEngineDisconnected() {

            }
        });
```

## 3、获得插件的信息（得到PackageInfo集合）

发送获取指令：

```
PluginEngine.getInstance().start(new PluginEngine.IPluginConnec
tion() {
                    @Override
                    public void onEngineConnected(PluginEngine.
PluginExecutor pluginExecutor) {
                        try {
                            pluginExecutor.send(PluginEngine.CO
DE_GET_PLUGIN_INFO);
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }

                    @Override
                    public void onEngineDisconnected() {
                    }
                });
```

接收：

```
@Override
public void onPluginMessageReceive(int code, String message, Bu
ndle extras) {

        StringBuilder sb = new StringBuilder();
        switch (code) {
            case PluginEngine.CODE_GET_PLUGIN_INFO:
                ArrayList<Parcelable> parcelableArrayList = ext
ras.getParcelableArrayList(PluginEngine.KEY_EXTRA);
                if (parcelableArrayList == null) {
                    //插件信息为null
                    return;
                }
                sb.append("插件个数:").append(parcelableArrayLis
t.size()).append("\n");
                for (Parcelable parcelable : parcelableArrayLis
t) {
                    if (parcelable instanceof PackageInfo) {
                        PackageInfo pluginInfo = (PackageInfo)
parcelable;
                        sb.append("插件包名:").append(pluginInfo
.packageName).append("\n");
                    }
                }
                break;
        }
    }
```

## 4、保存数据到本地

```
PluginEngine.getInstance().start(new PluginEngine.IPluginConnec
tion() {
                    @Override
                    public void onEngineConnected(PluginEngine.
PluginExecutor pluginExecutor) {
                        try {
                            String data = HostDataBuilder.saveI
nfoBuilder()
                                    .name("test") //文件名
                                    .put("key1", Math.random()
* 100 + "") //key可写成自己的，value只能是String
                                    .put("key2", Math.random()
```

```
* 100 + "") //可连续调用 .put()
                                    .build();
                            pluginExecutor.send(PluginEngine.CO
DE_SAVE_INFO, data);
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }

                    @Override
                    public void onEngineDisconnected() {

                    }
                });
```

接收保存成功的消息：

```
@Override
public void onPluginMessageReceive(int code, String message, Bu
ndle extras) {
        switch (code) {
            case PluginEngine.CODE_SAVE_INFO:
            Log.d("TAG","保存成功, 文件名:" + message);
            break;
        }
}
```

## 5、获取本地数据

发送获取指令：

```
PluginEngine.getInstance().start(new PluginEngine.IPluginConnec
tion() {
                    @Override
                    public void onEngineConnected(PluginEngine.
PluginExecutor pluginExecutor) {
                        try {
                            String data = HostDataBuilder.saveI
nfoBuilder()
                                    .name("test") //文件名
                                    .build();
                            pluginExecutor.send(PluginEngine.CO
DE_GET_SAVE_INFO, data);
```

```
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }

                    @Override
                    public void onEngineDisconnected() {

                    }
                });
```

接收：

```
@Override
public void onPluginMessageReceive(int code, String message, Bu
ndle extras) {
        StringBuilder sb = new StringBuilder();
        switch (code) {
            case PluginEngine.CODE_GET_SAVE_INFO:
                SaveInfoEntity saveInfoEntity = GsonUtils.fromJ
son(message, SaveInfoEntity.class);
                if (saveInfoEntity == null || saveInfoEntity.ge
tData() == null)
                    return;
                String name = saveInfoEntity.getData().getName(
);
                sb.append("文件名:").append(name).append("\n");
                HashMap<String, String> values = saveInfoEntity
.getData().getValues();
                if (values == null || values.size() == 0)
                    sb.append("无内容");
                else {
                    for (Map.Entry<String, String> next : value
s.entrySet()) {
                        sb.append("key:").append(next.getKey())
.append(",value:").append(next.getValue()).append("\n");
                    }
                }
                break;
        }
    }
```

## 6、手动从服务器更新插件（就算不手动调用也会定时

## 主动去更新）

```
PluginEngine.getInstance().start(new PluginEngine.IPluginConnec
tion() {
                    @Override
                    public void onEngineConnected(PluginEngine.
PluginExecutor pluginExecutor) {
                        try {
                            pluginExecutor.send(PluginEngine.CO
DE_UPDATE_PLUGIN);
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }

                    @Override
                    public void onEngineDisconnected() {
                    }
                });
```

## 7、发送自定义消息（核心层不做任何处理，只是将消息转发）

发送：

```
PluginEngine.getInstance().start(new PluginEngine.IPluginConnec
tion() {
                    @Override
                    public void onEngineConnected(PluginEngine.
PluginExecutor pluginExecutor) {
                        try {
                            pluginExecutor.send(PluginEngine.CO
DE_CUSTOM, "{\"msg\":\"test\"}");
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }

                    @Override
                    public void onEngineDisconnected() {
                    }
                });
```

接收：

```
@Override
    public void onPluginMessageReceive(int code, String message
, Bundle extras) {
        switch (code) {
            case PluginEngine.CODE_CUSTOM:
                String msg = "收到自定义消息: " + message;
                break;
        }
    }
```

## 8、获取PUSH ID

发送：

```
PluginEngine.getInstance().start(new PluginEngine.IPluginConnec
tion() {
                    @Override
                    public void onEngineConnected(PluginEngine.
PluginExecutor pluginExecutor) {
                        try {
                            pluginExecutor.send(PluginEngine.CO
DE_GET_PUSH_ID);
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }

                    @Override
                    public void onEngineDisconnected() {

                    }
                });
```

接收 push id：

```
@Override
public void onPluginMessageReceive(int code, String message, Bu
ndle extras) {
        case PluginEngine.CODE_GET_PUSH_ID:
            String pushIdStr = "PUSH ID: " + extras.getString(P
```

```
luginEngine.KEY_MESSAGE);
        }
}
```