# OPPO三方SDK

此SDK，提供OPPO手表与手机通信的能力

# 一、功能

- 连接节点查询与监听
- 收发信息
  - 目前消息仅支持长度为 `100k` 以内
  - 信息体只支持 `byte[]` ，建议使用 `protobuf` 来创建消息体
- 手机控制能力，仅支持手表侧
  - 查询手机是否有安装 `meta-data` 配置的应用
  - 尝试打开手机应用市场安装 `meta-data` 配置的应用
  - 尝试打开手机应用
  - 尝试打开手机浏览器
  - 蓝牙网络查询与监听

# 二、怎么使用

1. 在AndroidManifest.xml里添加权限申请，**手机手表应用都需要添加**

```
<uses-permission android:name="com.heytap.wearable.oms.permission.TRACSPORT" />
```

2. 在AndroidManifest.xml里添加meta-data，**手机手表应用都需要添加**

```
<!--对端的签名MD5-->
<meta-data
    android:name="targetSignature"
    android:value="8ddb342f2da5408402d7568af21e2xxx" />
<!--对端的包名-->
<meta-data
    android:name="targetPackage"
    android:value="com.heytap.xxx.xxx" />
```

3. 获取连接节点

```
//获取节点Client
NodeClient nodeClient = Wearable.getNodeClient(context);
//获取结果回调器
PendingResult<NodeClient.NodeResult> pendingResult = nodeClient.getNode();
//通过阻塞方式取得回调结果，此方法不能在主线程调用
NodeClient.NodeResult nodeResult = nodeClient.getNode().await();
//如果成功即可取得节点
if(nodeResult.getStatus().isSuccess()){
    nodeId = nodeResult.getNode().getId();
    ...
}

//也可以通过监听来获取节点
NodeClient.OnNodeChangedListener onNodeChangedListener = new NodeClient.OnNodeChangedListener()
    @Override
    public void onPeerConnected(Node node) {
        nodeId = nodeResult.getNode().getId();
        ...
    }

    @Override
    public void onPeerDisconnected(Node node) {
        nodeId = null;
        ...
    }
};
nodeClient.addListener(onNodeChangedListener);
```

### 4. 发送消息

```
//获取消息Client
MessageClient messageClient = Wearable.getMessageClient(context);
//定义path
String path = "xxx";
//定义消息
byte[] data = "Hello world!".getBytes();
//发送消息
messageClient.sendMessage(nodeId, path, data);
```

### 5. 接收消息

```
//创建接收器
MessageClient.OnMessageReceivedListener listener = new MessageClient.OnMessageReceivedListener()
    void onMessageReceived(MessageEvent messageEvent){
        String path = messageEvent.getPath();
        String message = new String(messageEvent.getData());
        ...
    }
};
//添加接收器
messageClient.addListener(listener);
```

# 三、状态

| code | message | description |
| --- | --- | --- |
| 0 | SUCCESS | 成功 |
| 3 | SERVICE_DISABLED | SDK服务被禁用 |
| 4 | SERVICE_MISSING | SDK服务未安装 |
| 6 | NODE_NOT_CONNECTED | 蓝牙节点断开 |
| 8 | INTERNAL_ERROR | 内部错误 |
| 14 | INTERRUPTED | 中断 |
| 15 | TIMEOUT | 超时 |
| 20 | TARGET_MISSING | 对方不存 |
| 21 | TARGET_ALREADY_INSTALLED | 对方已安装 |
| 22 | REQUEST_PERMISSION | 缺少权限 |
| 23 | REQUEST_META | 缺少meta配置 |
| 25 | NODE_NOT_MATCH | 发送的节点与连接的节点不匹配 |
| 26 | MESSAGE_TOO_LARGE | 发送内容过长，超过100K |
| 2014 | API_INTERRUPTED | 蓝牙服务连接中断 |
| 2015 | API_TIMEOUT | 蓝牙服务连接超时 |
| 2020 | API_DISCONNECTED | 蓝牙服务断开 |

| code | message | description |
| --- | --- | --- |
| 3014 | OMS_INTERRUPTED | SDK服务连接中断 |
| 3015 | OMS_TIMEOUT | SDK服务连接超时 |
| 3020 | OMS_DISCONNECTED | SDK服务断开 |

# 四、API

## Wearable

Client获取入口

### Method

- **getCapabilityClient**
  - 获取CapabilityClient实例
  - **Parameters**
    - `context`（`Context`）: 上下文
    - `options`（`WearableOptions`）: 可选，指定回调结果的 `Looper` ，如果没有默认为 `MainLooper`
- **getMessageClient**
  - 获取MessageClient实例
  - **Parameters**
    - `context`（`Context`）: 上下文
    - `options`（`WearableOptions`）: 可选，指定回调结果的 `Looper` ，如果没有默认为 `MainLooper`
- **getNodeClient**
  - 获取NodeClient实例
  - **Parameters**
    - `context`（`Context`）: 上下文
    - `options`（`WearableOptions`）: 可选，指定回调结果的 `Looper` ，如果没有默认为 `MainLooper`

## WearableOptions

`Wearable` 的可选配置，指定回调结果的 `Looper` ，只能通过 `Builder` 创建

## WearableOptions.Builder

`WearableOptions` 的创建者

## Methods

- **setLooper**
  - 指定回调结果的 `Looper`
  - **Parameters**
    - `looper` （ `Looper` ): 回调结果的 `Looper`
- **build**
  - 创建 `WearableOptions`

# NodeClient

节点Client

## Method

- **getNode**
  - 获取当前连接的节点
  - **Result**
    - `PendingResult<NodeResult>` ：结果为阻塞回调或异步回调，具体查看 `NodeResult` 和 `PendingResult`
- **addListener**
  - 添加节点监听器
  - **Parameters**
    - `listener` （ `OnNodeChangedListener` ): 节点监听器
- **removeListener**
  - 移除节点监听器
  - **Parameters**
    - `listener` （ `OnNodeChangedListener` ): 节点监听器

# OnNodeChangedListener

节点监听器

## Method

- **onPeerConnected**
  - 节点连接上时触发
  - **Parameters**
    - `node` （ `Node` ): 节点
- **onPeerDisconnected**

    - 节点断开时触发
    - **Parameters**
        - `node` （ `Node` ): 节点

# NodeResult

获取当前连接节点的结果

## Method

- **getNode**
    - 当前连接节点的结果，只有在 `Status` 为 `isSuccess` 可用
    - **Result**
        - `Node`：当前连接节点具体查看 `Node`
- **getStatus**
    - 获取结果状态
    - **Result**
        - `Status`：具体查看 `Status`

# Node

节点信息

## Method

- **getId**
    - 节点id
    - **Result**
        - `String`：节点id

# MessageClient

消息Client，当前仅支持100K以内的消息收发，消息体只支持 `byte[]` ，建议使用 `protobuf` 来创建消息体

## Method

- **sendMessage**
    - 发送消息
    - **Parameters**
        - `nodeId` （ `String` ): 连接的节点
        - `path` （ `String` ): 消息体标识

        - `data`（`byte[]`）: 消息体
    - **Result**
        - `PendingResult<SendMessageResult>` ：结果为阻塞回调或异步回调，具体查看 `SendMessageResult` 和 `PendingResult`
- **addListener**
    - 添加消息接收器
    - **Parameters**
        - `listener`（`OnMessageReceivedListener`）: 消息接收器
- **removeListener**
    - 移除消息接收器
    - **Parameters**
        - `listener`（`OnMessageReceivedListener`）: 消息接收器

# OnMessageReceivedListener

消息接收器

## Method

- **onMessageReceived**
    - 接收到消息时触发
    - **Parameters**
        - `messageEvent`（`MessageEvent`）: 接收到的消息，具体查看 `MessageEvent`

# MessageEvent

接收到的消息

## Method

- **getRequestId**
    - 请求id
- **getPath**
    - 消息体标识
- **getData**
    - 消息体
- **getSourceNodeId**
    - 消息来源节点

# SendMessageResult

蓝牙网络查询结果

## Method

- **getRequestId**
  - 请求id，只有在 `Status` 为 `isSuccess` 可用
  - **Result**
    - `int`：请求id
- **getStatus**
  - 获取结果状态
  - **Result**
    - `Status`：具体查看 `Status`

# CapabilityClient

能力Client，仅支持手表侧

## Method

- **checkInstalled**
  - 查询手机是否有安装meta-data配置的应用
  - **Parameters**
    - `nodeId`（`String`）: 连接的节点
  - **Result**
    - `PendingResult<Status>`：结果为阻塞回调或异步回调，具体查看 `Status` 和 `PendingResult`
- **tryInstall**
  - 尝试打开手机应用市场安装meta-data配置的应用
  - **Parameters**
    - `nodeId`（`String`）: 连接的节点
  - **Result**
    - `PendingResult<Status>`：结果为阻塞回调或异步回调，具体查看 `Status` 和 `PendingResult`
- **tryAwaken**
  - 尝试打开手机应用
  - **Parameters**
    - `nodeId`（`String`）: 连接的节点
    - `action`（`String`）: 手机端应用的 `action`
    - `data`（`byte[]`）: 需要传递的参数，在对端应用可以通过 `intent.getByteArrayExtra("heytap_data")` 获取
  - **Result**

        - `PendingResult<Status>` ：结果为阻塞回调或异步回调，具体查看 `Status` 和 `PendingResult`
- **tryOpenUrl**
    - 尝试打开手机浏览器
    - **Parameters**
        - `nodeId` （`String`）: 连接的节点
        - `url` （`String`）: 需要打开的地址
    - **Result**
        - `PendingResult<Status>` ：结果为阻塞回调或异步回调，具体查看 `Status` 和 `PendingResult`
- **isBluetoothNetProxy**
    - 查询当前连接网络是否为蓝牙网络，蓝牙网络仅支持 `http` 、 `https` 和 `websocket` ，其中 `http` 仅支持 `80` 端口和 `https` 仅支持 `443` 端口
    - **Parameters**
        - `nodeId` （`String`）: 连接的节点
        - `url` （`String`）: 需要打开的地址
    - **Result**
        - `PendingResult<BluetoothNetProxyResult>` ：结果为阻塞回调或异步回调，具体查看 `BluetoothNetProxyResult` 和 `PendingResult`
- **addBluetoothNetProxyChangedListener**
    - 添加蓝牙网络监听器
    - **Parameters**
        - `listener` （`OnBluetoothNetProxyChangedListener`）: 蓝牙网络监听器
- **removeBluetoothNetProxyChangedListener**
    - 移除蓝牙网络监听器
    - **Parameters**
        - `listener` （`OnBluetoothNetProxyChangedListener`）: 蓝牙网络监听器

# OnBluetoothNetProxyChangedListener

蓝牙网络监听器

## Method

- **onChanged**
    - 蓝牙网络发生变更时触发
    - **Parameters**
        - `enable` （`boolean`）: 当前是否为蓝牙网络

# BluetoothNetProxyResult

蓝牙网络查询结果

### Method

- **enable**
  - 蓝牙网络查询结果，只有在 `Status` 为 `isSuccess` 可用
  - **Result**
    - `boolean`：当前是否为蓝牙网络
- **getStatus**
  - 获取结果状态
  - **Result**
    - `Status`：具体查看 `Status`

## Status

状态

### Method

- **constructor**
  - 初始化
  - **Parameters**
    - `statusCode`（`int`）: 状态码
    - `statusMessage`（`String`）: 可选，状态描述
- **getStatusCode**
  - 返回状态码
- **getStatusMessage**
  - 返回状态描述
- **isSuccess**
  - 是否成功， `statusCode` 等于 `0`

## PendingResult

查询结果获取器

### Method

- **await**
  - 阻塞获取结果，此方法只能在非 `MainThread` 中使用，与 `setResultCallback` 互斥
- **await**
  - 阻塞获取结果，或到一定时间后返回超时，此方法只能在非 `MainThread` 中使用，与 `setResultCallback` 互斥

    - **Parameters**
        - `timeout`（`long`）: 等待时间
        - `unit`（`TimeUnit`）: 时间单位
- **setResultCallback**
    - 通过回调方法获取结果，回调线程在创建 `client` 时指定，与 `await` 互斥，只能设置一个回调
    - **Parameters**
        - `resultCallback`（`ResultCallback`）: 结果回调器
- **setResultCallback**
    - 通过回调方法获取结果，或到一定时间后返回超时，回调线程在创建 `client` 时指定，与 `await` 互斥，只能设置一个回调
    - **Parameters**
        - `resultCallback`（`ResultCallback`）: 结果回调器
        - `timeout`（`long`）: 等待时间
        - `unit`（`TimeUnit`）: 时间单位
- **addStatusListener**
    - 添加状态变更回调，可以设置多个
    - **Parameters**
        - `statusListener`（`StatusListener`）: 状态回调器，具体查看 `StatusListener`

# StatusListener

状态回调器

## Method

- **onComplete**
    - 当结果状态发生变更时触发
    - **Parameters**
        - `status`（`Status`）: 具体查看 `Status`

# ResultCallback

结果回调器

## Method

- **onResult**
    - 当结果状态发生变更时触发
    - **Parameters**
        - `result`：无论成功与否，都会触发，只有在 `result.getStatus` 为 `isSuccess` 可用，具体查看 `Status`

# ResultCallbacks

结果回调器，继承 `ResultCallback` ，区分了成功与失败

## Method

- **onSuccess**
  - 只有成功时才触发
  - **Parameters**
    - `result`：结果
- **onFailure**
  - 只有失败时才触发
  - **Parameters**
    - `Status`：具体查看 `Status`

# SportClient

运动健康Client，仅支持手表侧

## Method

- (**已过时**)**addHeartRateListener**
  - 添加心率监听器
  - **Parameters**
    - `listener`（`OnHeartRateChangedListener`）: 心率监听器
    - `interval`（`int`）: 回调间隔，单位秒，多个调用者，以最小为准
- (**已过时**)**removeHeartRateListener**
  - 移除心率监听器
  - **Parameters**
    - `listener`（`OnHeartRateChangedListener`）: 心率监听器
- **addHeartRateListener**
  - 添加心率监听器
  - **Parameters**
    - `listener`（`OnHeartRateChangedListener2`）: 心率监听器
    - `interval`（`int`）: 回调间隔，单位秒，多个调用者，以最小为准
- **removeHeartRateListener**
  - 移除心率监听器
  - **Parameters**
    - `listener`（`OnHeartRateChangedListener2`）: 心率监听器
- **addDailyActivityListener**
  - 添加日活动监听器

    - **Parameters**
        - `listener`（`OnDailyActivityChangedListener`): 日活动监听器
- **removeDailyActivityListener**
    - 移除日活动监听器
    - **Parameters**
        - `listener`（`OnDailyActivityChangedListener`): 日活动监听器

# OnHeartRateChangedListener

(**已过时**)心率监听器

## Method

- **onHeartRateChanged**
    - 实时心率回调
    - **Parameters**
        - `value`（`float`): 心率值（单位：BPM）, 有效范围：1~255
        - `accuracy`（`float`): 可信度， 可信度（单位：%）有效范围：0~100
        - `state`（`int`): 手表佩戴状态, 0正常 1未静止 2未佩戴 3正在测量
        - `interval`（`int`): 回调间隔，单位秒，多个调用者时，以最小为准

# OnHeartRateChangedListener2

心率监听器

## Method

- **onHeartRateEventChanged**
    - 实时心率回调
    - **Parameters**
        - `event`（`SportHearRateEvent`): 心率事件

# SportHearRateEvent

心率事件

## Method

- **value**
    - 心率值（单位：BPM）, 有效范围：1~255
    - **Result**
        - `float`

- **accuracy**
  - 可信度， 可信度（单位：%）有效范围：0~100
  - **Result**
    - `float`
- **state**
  - 手表佩戴状态, 0正常 1未静止 2未佩戴 3正在测量
  - **Result**
    - `int`
- **interval**
  - 回调间隔，单位秒，多个调用者时，以最小为准
  - **Result**
    - `int`

# OnDailyActivityChangedListener

日活动监听器

## Method

- **onDailyActivityChanged**
  - 实时日活动回调
  - **Parameters**
    - `event`（`SportDailyActivityEvent`）: 日活动事件

# SportDailyActivityEvent

日活动事件

## Method

- **calorie**
  - 总计运动卡路里消耗值，单位：千卡
  - **Result**
    - `int`
- **step**
  - 总运动步数，最小计步单位为1步
  - **Result**
    - `int`
- **distance**
  - 总运动距离，单位：米
  - **Result**

        - `int`
- **floor**
    - 总爬楼数，单位：楼
    - **Result**
        - `int`
- **exercise**
    - 总锻炼时间，单位：分钟
    - **Result**
        - `int`