Android_SDK_V1.0.4

1 搜索、连接、断开、回连设备

1.1 初始化设置

初始化 YCBTClient

```
/***
 * 初始化
 * @param context          上下文
 * @param isReconnect    是否需要重连
 * @param isDebug          是否是调试模式
 */
public static void initClient(Context context, boolean isReconnect, boolean isDebug)
YCBTClient.initClient(this,true, BuildConfig.DEBUG);
```

初始化时，调用 Reconnect 类

```
public void init(Context context, boolean isReconnect)
```

参数说明

- context 上下文
- isReconnect 设置是否自动回连，true 回连，false 不回连

示例代码

```
Reconnect.getInstance().init(getApplicationContext(),true);
```

1.2 搜索附近设备

```
YCBTClient.startScanBle(bleScanResponse);
```

蓝牙设备信息会逐个回调到 onScanResponse，可以用于展示蓝牙设备列表

```
YCBTClient.startScanBle(new BleScanResponse() {
    @Override
    public void onScanResponse(int i, ScanDeviceBean scanDeviceBean) {
        if (scanDeviceBean != null) {}
    }
}, 6);

public class ScanDeviceBean implements Comparable<ScanDeviceBean> {
    private String deviceMac;
    private String deviceName;
    private int deviceRssi;
    public BluetoothDevice device;
}
```

蓝牙设备信息包含：设备 mac，设备名称，rssi，BluetoothDevice 对象

```
YCBTClient.connectBle(mac, bleConnectResponse);
```

1.3 连接设备

通过设备 mac 连接设备

```
YCBTClient.connectBle(mac, bleConnectResponse);
```

1.4 全局监听蓝牙设备连接状态

```
//添加监听
YCBTClient.registerBleStateChange(bleConnectResponse);
//移除监听
YCBTClient.unRegisterBleStateChange(bleConnectResponse);
YCBTClient.registerBleStateChange(new BleConnectResponse() {
    @Override
    public void onConnectResponse(int code) {
        EventBusMessageEvent eventBusMessageEvent = new EventBusMessageEvent();
        if (code == Constants.BLEState.Disconnect) {
            //断开连接
            eventBusMessageEvent.belState = EventBusMessageEvent.DISCONNECT;
        } else if (code == Constants.BLEState.ReadWriteOK) {
            //连接成功
            eventBusMessageEvent.belState = EventBusMessageEvent.CONNECTED;
        } else if (code == Constants.BLEState.TimeOut) {
            //连接超时
            eventBusMessageEvent.belState = EventBusMessageEvent.TIMEOUT;
        } else if (code == Constants.BLEState.Disconnecting) {
            //正在断开
            eventBusMessageEvent.belState = EventBusMessageEvent.DISCONNECTING;
        } else {
            //正在连接
            eventBusMessageEvent.belState = EventBusMessageEvent.CONNECTING;
        }
        EventBus.getDefault().post(eventBusMessageEvent);
    }
});
```

1.5 断开蓝牙设备连接

```
YCBTClient.disconnectBle();
```

1.6 判断蓝牙连接状态

```
if(YCBTClient.connectState() == Constants.BLEState.ReadWriteOK){
   //已连接并读写成功
}

public static class BLEState {
    public static final int TimeOut = 0x01;          //超时
    public static final int NotOpen = 0x02;
    public static final int Disconnect = 0x03;       //未连接
    public static final int Disconnecting = 0x04; //断开连接中
    public static final int Connecting = 0x05;       //连接中
    public static final int Connected = 0x06;         //已连接
    public static final int ServicesDiscovered = 0x07; //发现服务
    public static final int CharacteristicDiscovered = 0x08; //发现特征
    public static final int CharacteristicNotification = 0x09; //使能通知
```

```
    public static final int ReadWriteOK = 0x0a;         //读写成功
}
```

1.7 设置是否重连

```
/**
 *  设置是否需要重连
 * @param isReconnect  是否需要重连
 */
public static void setReconnect(boolean isReconnect)
YCBTClient.setReconnect(true);
```

2 OTA 升级

2.1 连接设备

通过设备 mac 连接设备

```
YCBTClient.connectBle(mac, bleConnectResponse);
```

2.2 OTA 升级

调用更新固件方法，开始发送固件到设备，回调中可获取当前进度，发送固件到设备完成后，会断开蓝牙连接，需要重新连接上，再调用更新固件的方法，此时设备会显示固件升级的界面，进度条再次从 0 开始，固件升级完成后，设备自动重启
调用固件升级
杰理平台升级
开始发送固件，界面显示进度条，进度条满会自动断开连接，监听连接状态，如果断开需要回连，连接上之后再次调用固件升级方法，此时设备进入升级状态，进度条再次从 0 开始，升级完成后设备会自动重启

```
YCBTClient.upgradeFirmware(this, mac, name, filePath, dfuCallBack);
```

非杰理平台
发送固件和升级只需要调用一次即可完成

2.3 UI 升级

传递升级文件路径，在回调中获取当前进度

```
YCBTClient.watchUiUpgrade(path, new BleDataResponse() {
    @Override
    public void onDataResponse(int code, float ratio, HashMap resultMap) {
        Object progress1 = resultMap.get("progress");
        Object data = resultMap.get("data");
        if (progress1 != null) {
            //当前进度百分比
                progress = (float) progress1;
        }else if(data != null){
            if((int)data == 0){
            //升级成功
            }else{
            //升级失败
            }
        }
```

```
    }
}
```

3 同步、删除健康历史数据

获取设备的健康数据， 指的是查询设备通过主动检测或自动检测的心率，血压，睡眠，血氧等 数据。
设备中的健康数据类型都不相同，请依据设备的返回值与功能支持开关来进行调用。 设备支持的数据类型当中，前 5 个数据类型(运动, 睡眠, 心率, 血压, 组合数据)是大部分设备都支持，其余的则是定制版本才支持。 注意:通过组合数据获取到的心率，步数，睡眠，血压，不要使用，也就是说，通过此类型获取到的数据，只能使用血氧、呼吸率、体温、体脂等。而心率、步数、睡眠、血压这几 个只能通过单独的类型去获取。
设备中的数据 不会主动删除 ，所以 App 获取以后，应该主动删除，否则下次获取时，会获得相同的数据，且一旦设备中的数据超过存储大小将会被自动删除。
查询到应的数据后，结果会在 response 中返回，每个类型返回的结果也不相同，需要转换成对应的对象。具体的类型已经逐个列出，参考 3.1 的内容。

3.1 获取数据

```
public static void healthHistoryData(int dataType, BleDataResponse bleDataResponse)
```

参数说明：
dataType：类型
bleDataResponse：数据回调
类型：

运动，Constants.DATATYPE.Health_HistorySport
睡眠，Constants.DATATYPE.Health_HistorySleep
心率，Constants.DATATYPE.Health_HistoryHeart
血压，Constants.DATATYPE.Health_HistoryBlood
组合数据,Constants.DATATYPE.Health_HistoryAll
血脂、尿酸,Constants.DATATYPE.Health_HistoryComprehensiveMeasureData

---

```
YCBTClient.healthHistoryData(Constants.DATATYPE.Health_HistorySport,bleDataResponse);
```

数据回调 bleDataResponse：

```
@Override
public void onDataResponse(int code, float ratio, HashMap resultMap) {
    if (code == 0 && resultMap != null) {
        //获取成功
        BloodResponse mBloodResponse = new Gson().fromJson(String.valueOf(resultMap), BloodResponse.class);
        List<BloodResponse.DataBean> mBloodData = mBloodResponse.getData();
    } else {
        //获取失败
    }
}
public class BloodResponse {
    private int code;
    private List<DataBean> data;//根据不同的类型，返回的数据不同
    private int dataType;
}
```

健康数据类型与对应的返回值类型

```
public static class SportDataBean {
    private long sportStartTime;// 开始时间戳(秒)
    private long sportEndTime;// 结束时间戳(秒)
    private int sportStep;// 步数 (步)
    private int sportDistance;// 距离 (米)
```

```
    private int sportCalorie;// 卡路里(千卡)
}
public static class SleepDataBean {
    private int deepSleepCount; // 深睡次数
    private int lightSleepCount; // 浅睡次数
    private long startTime; // 开始睡眠时间
    private long endTime; // 结束睡眠时间
    private int deepSleepTotal; // 深睡总时间
    private int lightSleepTotal; // 浅睡总时间
    @SerializedName(value = "rapidEyeMovementTotal", alternate = {"remTimes"})
    public int rapidEyeMovementTotal; //眼动总时间
    public int wakeCount;//清醒次数
    public int wakeDuration;//清醒时长
    private List<SleepData> sleepData = new ArrayList<>(); // 睡眠数据
}

public static class SleepData {
    @SerializedName(value = "sleepStartTime", alternate = {"stime"})
    private long sleepStartTime; // 开始时间
    @SerializedName(value = "sleepLen", alternate = {"sleepLong"})
    private int sleepLen; // 睡眠长度
    private int sleepType; //深睡浅睡标志，标志类型为下面的 SleepType
}

public static class SleepType{
    public static final int unknow = -1;// 未知
    public static final int deepSleep = 0xF1;// 深睡
    public static final int lightSleep = 0xF2;// 浅睡
    public static final int rem = 0xF3;// 快速眼动
    public static final int awake = 0xF4;// 清醒
}
public static class DataBean {
    private int heartValue;// 心率值
    private long heartStartTime;// 开始时间戳(秒)
}
public static class DataBean {
    private long bloodStartTime;// 开始时间戳
    private int bloodDBP;// 舒张压
    private int bloodSBP;// 收缩压
}
public static class AllDataBean {
    public int heartValue;// 心率值
    public int hrvValue;// HRV
    public int cvrrValue;// CVRR
    public int OOValue;// 血氧值
    public int stepValue;// 步数 (步)
    public int DBPValue;// 舒张压
    public int tempIntValue;// 温度整数部分
    public int tempFloatValue;// 温度小数部分
    public long startTime;// 开始时间戳
    public int SBPValue;// 收缩压
    public int respiratoryRateValue;// 呼吸率值
```

```
    public int bodyFatIntValue;//体脂整数部分
    public int bodyFatFloatValue;//体脂小数部分
    public int bloodSugarValue;//血糖*10 的值
}
public static class ComprehensiveMeasureDataBean {
    //tc
    public int cholesterolInteger;//血脂整数部分
    public int cholesterolFloat;//血脂小数部分
    //tg
    public int triglycerideCholesterolInteger;//未使用
    public int triglycerideCholesterolFloat;//未使用
    //ldlc
    public int highLipoproteinCholesterolInteger;//未使用
    public int highLipoproteinCholesterolFloat;//未使用
    //hdlc
    public int lowLipoproteinCholesterolInteger;//未使用
    public int lowLipoproteinCholesterolFloat;//未使用
    public int bloodFatModel;//血脂测量模式

    public int uricAcid;//尿酸值
    public int uricAcidModel;//尿酸测量模式

    public int bloodKetoneInteger;//未使用
    public int bloodKetoneFloat;//未使用
    public int bloodKetoneModel;//未使用

    public int bloodSugarInteger;//未使用
    public int bloodSugarFloat;//未使用
    public int bloodSugarModel;//未使用

    public long time;// 开始时间戳
}
```

3.2 删除数据

```
YCBTClient.deleteHealthHistoryData(type,bleDataResponse);
```

参数说明：

类型：

运动，Constants.DATATYPE.Health_DeleteSport

睡眠，Constants.DATATYPE.Health_DeleteSleep

心率，Constants.DATATYPE.Health_DeleteHeart

血压，Constants.DATATYPE.Health_DeleteBlood

组合数据,Constants.DATATYPE.Health_DeleteAll

血脂、尿酸,Constants.DATATYPE.Health_DeleteComprehensiveMeasureData

bleDataResponse：数据回调

```
@Override
public void onDataResponse(int code, float ratio, HashMap resultMap) {
    if (code == 0) {
        //删除成功
    } else {
        //删除失败
```

```
    }
}
```

4 表盘

表盘下载包含查询设备中的表盘信息，App 切换表盘，删除表盘，设备操作表盘，App 自定义表盘。

4.1 查询设备表盘信息

```
//调用获取表盘接口，传入回调参数
YCBTClient.watchDialInfo(new BleDataResponse() {
    @Override
    public void onDataResponse(int i, float v, HashMap hashMap) {
        if (i == 0 && hashMap != null) {
            maxDials = (int) hashMap.get("maxDials");//最大表盘数量
            currDials = (int) hashMap.get("currDials");//当前表盘数量
            List<DialsBean> dials = (List<DialsBean>) hashMap.get("dials");//所有表盘的数据
            List<DialsBean> customDials = (List<DialsBean>) hashMap.get("customDials");//自定义表盘的数据，部分设备不支持此功能
        }
    }
});

//表盘信息
public class DialsBean {
    public int dialplateId;//表盘 Id
    public int blockNumber;//当前接收的 block 包数
    public boolean isCanDelete;//是否可删除
    public int dialVersion;//表盘版本
}
```

4.2 删除表盘

杰理平台和非杰理平台的调用方法不一样

```
//判断平台
if (YCBTClient.getChipScheme() == Constants.Platform.JieLi) {
    //杰理平台,根据表盘文件名进行删除
    String fileName = datas.get(position).fileName;
    //截取表盘名称，示例：/watch0
    YCBTClient.jlWatchDialDelete(fileName.substring(fileName.lastIndexOf("/")), new BleDataResponse() {
        @Override
        public void onDataResponse(int i, float v, HashMap hashMap) {
            if (i == 0) {
              //删除成功
            }else{
              //删除失败
            }
        }
    });
} else {
    //非杰理平台，根据表盘 dialplateId 进行删除
```

```
    DialResultBean.Data data = datas.get(position);
    data.state = 0;
    data.progress = 0;
    data.blockNumber = 0;
    YCBTClient.watchDialDelete(data.dialplateId, new BleDataResponse() {
        @Override
        public void onDataResponse(int i, float v, HashMap hashMap) {
            if (i == 0) {
                //删除成功
            }else{
                //删除失败
            }
        }
    });
}
```

4.3 切换表盘

切换表盘与删除表盘的参数都是相同的

```
//判断平台
if (YCBTClient.getChipScheme() == Constants.Platform.JieLi) {
   //杰理平台,根据表盘文件名进行删除
   String fileName = datas.get(position).fileName;
   //截取表盘名称
   YCBTClient.jlWatchDialSetCurrent(fileName.substring(fileName.lastIndexOf("/")), new BleDataResponse() {
        @Override
        public void onDataResponse(int code, float v, HashMap hashMap) {
            if (code == 0) {
                //设置成功
            }else{
                //设置失败
            }
        }
   });
} else {
   //非杰理平台，根据表盘 dialplateId 进行删除
   YCBTClient.watchDialSetCurrent(datas.get(position).dialplateId, new BleDataResponse() {
        @Override
        public void onDataResponse(int code, float v, HashMap hashMap) {
            if (code == 0) {
                //设置成功
            }else{
                //设置失败
            }
        }
   });
}
```

4.4 下载表盘

下载新表盘之前，如果超过表盘最大数量，需要先删除一个旧表盘

4.4.1 杰理平台下载

```
/**
 * 表盘下载    ,    表盘版本 号 整个文件 CRC
 *
 * @param filePath          是表盘文件路径，必须存在
 * @param isNoNeedCheck ：是否跳过文件校验    false：  表盘文件需要文件校验     true :  自定义背景文件不需要文件校验，但需要转换工具进行算法转换
 * @param dataResponse    code 0x00：结束下载 0x01：开始下载     data dialState Ox00：接收成功 0x01：参数错误 0x02: 表盘个数超过最大存储数量
 */
public static void jlWatchDialDownload(String filePath, boolean isNoNeedCheck, BleDataResponse dataResponse)
YCBTClient.jlWatchDialDownload(path, false, new BleDataResponse() {
   @Override
   public void onDataResponse(int i, float v, HashMap hashMap) {
      if (code == 0) {//Ox00：接收成功 0x01：参数错误 0x02: 表盘个数超过最大存储数量
         //判断是否是返回进度
         if((int) hashMap.get("dataType") == Constants.DATATYPE.WatchDialProgress){
            int progress = (int) ((float) hashMap.get("progress"));//当前下载进度
         }else{
            //下载完成
         }
      }else{
         //下载失败
      }
   }
});
```

4.4.2 非杰理平台下载

```
/**
 * 表盘下载，表盘版本号，整个文件 CRC
 *
 * @param type              0x00：结束下载 0x01：开始下载
 * @param dialDatas       表盘的二进制数据
 * @param dialId            表盘 ID，ID 相同 则为更新，不同则 为新的文件
 * @param dialBlock       已接收 block 数. 没有断点信息，则为 0
 * @param dialVersion    表盘的版本号
 * @param dataResponse code 0x00：结束下载 0x01：开始下载     data dialState Ox00：接收成功 0x01：参数错误 0x02: 表盘个数超过最大存储数量
 */
public static void watchDialDownload(int type, byte[] dialDatas, int dialId, int dialBlock, int dialVersion, BleDataResponse dataResponse)
FileInputStream inputStream = new FileInputStream(new File(path));
byte[] buffer = new byte[1024];
int len = 0;
ByteArrayOutputStream bos = new ByteArrayOutputStream();
while ((len = inputStream.read(buffer)) != -1) {
      bos.write(buffer, 0, len);
}
bos.flush();
YCBTClient.watchDialDownload(0x01, bos.toByteArray(), id, datas.get(position).blockNumber, Integer.parseInt(datas.get(position).dialVersion), new BleDataResponse() {
      @Override
      public void onDataResponse(int i, float v, HashMap hashMap) {
         if (code == 0) {//Ox00：接收成功 0x01：参数错误 0x02: 表盘个数超过最大存储数量
            //判断是否是返回进度
            if((int) hashMap.get("dataType") == Constants.DATATYPE.WatchDialProgress){
```

```
                int progress = (int) ((float) hashMap.get("progress"));//当前下载进度
            }else{
                //下载完成
            }
        }else{
            //下载失败
        }
    }
});
```

4.5 自定义表盘

获取表盘参考 4.1 查询设备表盘信息，同时获取到系统表盘和自定义表盘
说明
自定义表盘是基于厂商提供的自定义表盘源文件，进行图片与文字颜色的修改，产生 一个新的表盘文件，下载到设备中。 如果不修改图片，可以传入 null，SDK 会保持原来的背景图片与缩略图片。 生成了新的表盘文件，直接调用表盘下载的方法，下载到设备中去即可。 文档中给出了一个表盘 BMP 查询信息的方法，可能在 App 开发界面,或生成缩略图片时 要用到相关的参数。

```
/***
 *  获取屏幕参数
 * @param dataResponse dataResponse hashmap.get("data");
 *                              screenType 0:圆屏    1:方屏
 *                              screenWidth    屏幕宽度（单位像素）
 *                              screenHeight  屏幕高度
 *                              screenCorner  屏幕圆角
 */
public static void getScreenParameters(BleDataResponse dataResponse)
YCBTClient.getScreenParameters(new BleDataResponse() {
  @Override
  public void onDataResponse(int i, float v, HashMap hashMap) {
      if (hashMap != null) {
          //屏幕圆角
          int screenCorner = (int) hashMap.get("screenCorner");
      }
  }
}
```

生成自定义表盘数据

```
/*
 *  修改自定义表盘文件
 *    @param dst              目标文件地址  byte[]
 *    @param size             bin 文件  byte[]
 *    @param bg_src           背景图  byte[]
 *    @param thumb_src    缩略图  byte[]
 *    @param x                日期显示的位置 x 坐标  short
 *    @param y                日期显示的位置 y 坐标  short
 *    @param r                日期字体颜色值
 *    @param g                日期字体颜色值
 *    @param b                日期字体颜色值
 *
 *    @return                 true：修改成功    false：修改失败
 *
 */
public boolean modifyBinFile(byte[] dst, byte[] src, byte[] bg_src, byte[] thumb_src, int x, int y, byte r, byte g, byte b)
```

4.5.1 杰理平台

```
/**
 * 杰理平台 bitmap 图像转换
 *
 * @param inPath            输入文件路径<p>例如：in.png ,in.jpg</p>
 * @param outPath           输出文件路径<p>例如:out.res, out</p>
 * @param dataResponse code 0x00：成功 0x01：失败
 */
public static void jlSaveCustomizeDialBg(String inPath, String outPath, BleDataResponse dataResponse)
/**
 * 杰理安装自定义表盘
 * @param bgFilePath 文件路径
 * @param dataResponse 回调
 */
public static void jlInstallCustomizeDial(String bgFilePath, BleDataResponse dataResponse)
/**
 * 设置杰理表盘文字的位置和颜色
 * @param position 9 个位置之一
 * 0x01:上
 * 0x02:下
 * 0x03:左
 * 0x04:右
 * 0x05:左上
 * 0x06:右上
 * 0x07:左下
 * 0x08:右下
 * 0x09:居中
 * @param color 颜色 rgb565 格式
 * @param dataResponse
 */
public static void jieliSetDialText(int position,int color,BleDataResponse dataResponse)
```

4.5.2 非杰理平台

获取、切换、删除和其它表盘一样。

```
/*
 * 获取自定义表盘背景图片大小
 *
 */
public ImageBean getBmpSize(byte[] binArray)
/*
 * 背景图 bmp888 转 bmp565
 *
 */
public byte[] toBmp565(byte[] binArray, int size, boolean isFlip)
/*
 * 修改自定义表盘文件
 *   @param dst             目标文件地址 byte[]
```

```
 *   @param size           bin 文件  byte[]
 *   @param bg_src         背景图  byte[]
 *   @param thumb_src      缩略图  byte[]
 *   @param x              日期显示的位置 x 坐标  short
 *   @param y              日期显示的位置 y 坐标  short
 *   @param r              日期字体颜色值
 *   @param g              日期字体颜色值
 *   @param b              日期字体颜色值
 *
 *   @return               true：修改成功   false：修改失败
 *
 */
public boolean modifyBinFile(byte[] dst, byte[] src, byte[] bg_src, byte[] thumb_src, int x, int y, byte r, byte g, byte b)
```

4.6 监听表盘切换

```
YCBTClient.deviceToApp(new BleDeviceToAppDataResponse() {
            @Override
            public void onDataResponse(int code, HashMap hashMap) {
            //监听设备操作，表盘切换或删除
            if (code == 0 && hashMap != null && hashMap.get("dataType") != null && (int) hashMap.get("dataType") == Constants.DATATYPE.DeviceSwitchDial) {
                if (YCBTClient.getChipScheme() == Constant.Platform.JieLi) {
                    //杰理平台
                    String deviceName = hashMap.get("datas").toString();
                    for (int i = 0; i < datas.size(); i++) {
                        DialResultBean.Data data = datas.get(i);
                        String fileName = data.fileName;
                        String substring = fileName.substring(fileName.lastIndexOf("/"));
                        //判断并修改表盘的当前状态
                        if (substring.equalsIgnoreCase(deviceName)) {
                            //当前使用的表盘
                            data.state = 4;
                            oldDialPosition = i;
                        } else {
                            if (data.state == 4) {
                                data.state = 3;
                            }
                        }
                    }
                } else {
                    //非杰理平台
                    byte[] ids = (byte[]) hashMap.get("datas");
                    int id = (ids[0] & 0xff) + ((ids[1] & 0xff) << 8) + ((ids[2] & 0xff) << 16) + ((ids[3] & 0xff) << 24);
                    for (int i = 0; i < datas.size(); i++) {
                       //判断并修改表盘的当前状态
                        DialResultBean.Data data = datas.get(i);
                        if (data.state == 4) {
                            data.state = 3;
                        }
                        if (data.dialplateId == id) {
                            //当前使用的表盘
                            data.state = 4;
                            oldDialPosition = i;
```

```
                    }
                }
            }
        }
        }
}
```

4.7 设置自定义表盘

封装了杰理和非杰理的设置自定义表盘

```
 /**
    *  设置自定义表盘
    * @param context  上下文
    * @param imgPath  图片路径
    * @param thumbnailPath  缩略图路径
    * @param customDialId  表盘 id
    * @param saveFileName  表盘名称   例如：WATCH900     ,bin 文件路径
    * @param pointX   文字 x 轴   杰理表盘没有 x 轴 y 轴参数，可传 0
    * @param pointY   文字 y 轴
    * @param position   文字位置    //上：1   下：2   左：3   右：4   左上：5   右上：6   左下：7   右下：8   中：9
    * @param parseColor   文字颜色  rgb565 格式
    * @param isCanDelete  是否可删除
    * @param dialProgressListener  回调,status:0 正常，1 安装个数达上限，2 失败,progress:进度
    */
   public static void setDialCustomize(Context context,String imgPath, String thumbnailPath,
                                        int customDialId, String saveFileName, int pointX, int pointY, int position,
                                        int parseColor, boolean isCanDelete, DialUtils.DialProgressListener dialProgressListener)
```

5 拍照

启动拍照有两种方式，一种是设备启动进入拍照模式，一种是 App 启动进入拍照模式。进入拍照模式后，执行拍照动作也有两种方式，一种是点击 App 进行拍照，一种是点击 设备进行拍照。真正的拍照都是在手机上完成，如果是设备点击执行拍照，则拍照完成后需要回复设备，拍照是否成功。

5.1 App 开启与关闭拍照模式

```
/**
 * APP 控制手环拍照界面
 *
 * @param type                0x00:  退出拍照模式  0x01:  进入拍照模式
 * @param dataResponse 0x00  成功    0x01  失败
 */
public static void appControlTakePhoto(int type, BleDataResponse dataResponse)
YCBTClient.appControlTakePhoto(1, dataResponse);
```

5.2 设备开启与关闭拍照模式

```
 YCBTClient.deviceToApp(new BleDeviceToAppDataResponse() {
                    @Override
                    public void onDataResponse(int i, HashMap hashMap) {
```

```
                    if (hashMap != null) {
                        if (i == 0) {//
                            int dataType = (int) hashMap.get("dataType");
                            int data = -1;
                            if (hashMap.get("data") != null)
                                data = (int) hashMap.get("data");
                            switch (dataType) {
                                case Constants.DATATYPE.DeviceTakePhoto://相机拍照控制
                                    if (PermissionUtil.openCameraPermission(context) && PermissionUtil.openSDCardPermission(context)) {
                                        //0x00:退出拍照模式  0x01:进入拍照模式  0x02:拍照
                                        if (data == 1) {

                                        } else {
                                            //发送事件通知，退出拍照模式
                                        }
                                    }
                                break;
                            }
                        }
                    }
                }
 });
```

当接收到状态为拍照时，拍照依然是在手机上操作，设备只负责上报当前的状态。

5.3 交叉操作

拍照的交互逻辑一旦出现交叉，比如手机启动，设备退出，或者设备启动，手机退出。要注意状态的变化，再调用对应的接口就可以了。

6 通讯录

通讯录功能，仅仅是将用户名称和号码，发送给设备进和保存，设备所能存储的最大数量为 30 个。传输通讯录的整个过程中，开启和退出同步只需要执行一次，而发送通讯数据，则需要重复执行，因为执行一次只能发送一条记录

6.1 获取通讯录

```
/**
 * 获取通讯录
 */
public static void getDeviceContacts(Context context, BleDataResponse bleDataResponse)
 YCBTClient.getDeviceContacts(ContactsActivity.this, new BleDataResponse() {
      @Override
      public void onDataResponse(int i, float v, HashMap hashMap) {
            if (hashMap != null && hashMap.get("data") != null) {
                List<ContactsBean> beans = (ArrayList) hashMap.get("data");
            }
      }
});

public class ContactsBean implements Serializable {
      public short id;//通讯录 id
      public String name;//名称
      public String number;//号码
```

```
}
```

6.1 进入同步通讯录

设备只有设置开始同步后，才能进入真正的同步数据。只需要执行一次。

```
/**
 * 开始/结束推送通讯录
 *
 * @param type                操作符  0x00:结束, 0x02:开始
 * @param dataResponse 0x00 同步成功    0x01 同步失败
 */
public static void appPushContactsSwitch(int type, BleDataResponse dataResponse)
```

6.2 发送通讯录数据

6.2.1 非杰理平台

```
/**
 * 推送单个通讯录
 *
 * @param phoneNumber    手机号码
 * @param name               名字（UTF-8 字符串）最大不能超过 8 个中文字符
 * @param dataResponse 0x00 同步成功    0x01 同步失败
 */
public static void appPushContacts(String phoneNumber, String name, BleDataResponse dataResponse)

/**
 * 推送一组通讯录，内部实现仍是循环读取数据，推送单个通讯录
 *
 * @param contactsBeans  联系人实体类集合
 * @param dataResponse    0x00 同步成功    0x01 同步失败
 */
public static void appNewPushContacts(Context context, List<ContactsBean> contactsBeans, BleDataResponse dataResponse)
YCBTClient.appPushContactsSwitch(2,dataResponse)//开始
for (ContactsBean bean : contactsBeans) {
      YCBTClient.appPushContacts(bean.number, bean.name, dataResponse);//推送单个
}
YCBTClient.appPushContactsSwitch(0, dataResponse);//结束
```

6.2.2 杰理平台

杰理平台的通讯录的信息发送是使用的平台本身的 API，是独立的 API。

```
List<DeviceContacts> contacts = new ArrayList<>();
for (ContactsBean bean : contactsBeans) {
      DeviceContacts deviceContacts = new DeviceContacts(bean.id, bean.name, bean.number);
      if (bean.name != null) {
            bean.name = ByteUtil.getData(bean.name, 20);
      }
      contacts.add(deviceContacts);
}
```

```
UpdateContactsTask task = new UpdateContactsTask(WatchManager.getInstance(), context, contacts);
task.setListener(new SimpleTaskListener() {
        @Override
        public void onBegin() {
        }

        @Override
        public void onError(int code, String msg) {
                if (dataResponse != null) {
                        HashMap<String, String> map = new HashMap<String, String>();
                        map.put("msg", msg);
                        dataResponse.onDataResponse(-1, 0, map);
                }
        }

        @Override
        public void onFinish() {
                if (dataResponse != null) {
                        dataResponse.onDataResponse(0, 0, null);
                }
        }
});
task.start();
```

7 ECG 检测

ECG 检测包含启与停止ECG，获取 ECG 的结果，关于绘制 ECG 的波形，请参考 Demo 的案例演示，文档会将相关的方法进行举例说明。ECG 检测的开启和关闭都由 App 来完成，建议测量时间为 60 ~ 90 秒。测试过程会获取到测量的数据。同样，设备本身也能启动 ECG 测量，App 可以获取到相关的信息。

7.1 设置穿戴位置

设备的佩戴位置与设置位置不匹配时，产生的波形是相反的。

```
/***
   *  左右手佩戴设置
   * @param leftOrRight 0x00:  左手  0x01:  右手
   * @param dataResponse 0x00 同步成功    0x01 同步失败
      */
public static void settingHandWear(int leftOrRight, BleDataResponse dataResponse)
YCBTClient.settingHandWear(Constants.HandWear.Left, new BleDataResponse() {
        @Override
        public void onDataResponse(int code, float ratio, HashMap resultMap) {
                if (code == 0) {
                        Log.i("EcgMeasurDialog", "你选择的是:左手");
                }
        }
});
```

7.2 开启与结束 ECG 测量

启动测量 ECG 后，需要将另一个手的手指放到与电极位置的地方，进行接触，测量开始后，设备会将上报测量的数据。

```
/****
 *  开始 ECG 实时测试
 * @param dataResponse
 * @param realDataResponse
 */
public static void appEcgTestStart(BleDataResponse dataResponse, BleRealDataResponse realDataResponse)

/***
 *  结束 ECG 实时测试
 * @param dataResponse
 */
public static void appEcgTestEnd(BleDataResponse dataResponse)
YCBTClient.appEcgTestStart(dataResponse, new BleRealDataResponse() {
      @Override
      public void onRealDataResponse(int i, HashMap hashMap) {
            if (hashMap != null) {
                  int dataType = (int) hashMap.get("dataType");
                  if (i == Constants.DATATYPE.Real_UploadECG) {
                        List<Integer> data = (List<Integer>) hashMap.get("data");
                        person(data);
                  } else if (i == Constants.DATATYPE.Real_UploadECGHrv) {
                        if (!hrv_is_from_device && hashMap.get("data") != null && ((float) hashMap.get("data")) != 0) {
                              mHRV = (int) ((float) hashMap.get("data"));
                        }
                  } else if (i == Constants.DATATYPE.Real_UploadECGRR) {
                        float param = (float) hashMap.get("data");
                  } else if (i == Constants.DATATYPE.Real_UploadBlood) {
                        mHeart = (int) hashMap.get("heartValue");//心率
                        mDBP = (int) hashMap.get("bloodDBP");//高压
                        mSBP = (int) hashMap.get("bloodSBP");//低压
                        if (hashMap.get("hrv") != null && ((int) hashMap.get("hrv")) != 0) {
                              hrv_is_from_device = true;
                              mHRV = (int) hashMap.get("hrv");
                        }
                  } else if (i == Constants.DATATYPE.AppECGPPGStatus) {
                     //部分定制设备会返回 PPG 数据
                  }
            }
      }
});

YCBTClient.appEcgTestEnd(new BleDataResponse() {
      @Override
      public void onDataResponse(int code, float ratio, HashMap resultMap) {
            if (code == 0) {
                  if (mEcgMeasureList.size() > 2800) {
                        //  心率分析
                  }
            }
      }
});

private void person(List<Integer> datas) {
      mEcgMeasureList.addAll(datas);
```

```
    int index = 0;
    for (Integer data : datas) {
        if (index % 3 == 0) {
            int value = data / 40;
            drawLists.add(value > 500 ? 500 : value);
        }
        index++;
    }
}
```

7.2 获取 ECG 结果

```
AITools.getInstance().getAIDiagnosisResult(new BleAIDiagnosisResponse() {
    @Override
    public void onAIDiagnosisResponse(AIDataBean aiDataBean) {
        if (aiDataBean != null) {
            short heart = aiDataBean.heart;//心率
            // 诊断类型  1 正常  4 ,5 ,9 异常
            mDiagnoseType = aiDataBean.qrstype;//类型 1 正常心拍 5 室早心拍 9 房早心拍  14 噪声
            // 是否房颤
            isAfib = aiDataBean.is_atrial_fibrillation;//是否心房颤动
            // 存储数据
            saveData();
        }
    }
});

public class AIDataBean implements Serializable {
    public short heart;//心率  heart<= 50 疑似心动过缓  heart>= 120  疑似心动过速  hrv >= 125 疑似窦性心律不齐
    public int qrstype;//类型  1 正常  2 心室扑动  3 交界性早搏  4  交界性逸搏  5 室性早搏  6 室性逸搏  7 左束支传导阻滞  8 右束支传导阻滞  9 房性早搏  10 房性逸搏  14 异常心电图(波形不正常)
    public boolean is_atrial_fibrillation;//是否心房颤动(房颤)
}
```

7.3 同步 ecg 数据和 PPG 数据

```
//同步 ECG
public static void collectEcgList(BleDataResponse dataResponse)
//同步 PPG 数据
public static void collectPpgList(BleDataResponse dataResponse)
YCBTClient.collectEcgList(new BleDataResponse() {
  @Override
  public void onDataResponse(int code, float v, HashMap resultMap) {
      if (code == 0) {
          if (resultMap == null) {
              return;
          }
          EcgSyncListResponse ecgSyncListResponse = new Gson().fromJson(String.valueOf(resultMap), EcgSyncListResponse.class);
          if (ecgSyncListResponse == null || ecgSyncListResponse.data == null) {
              return;
          }
          List<EcgSyncListResponse.DataBean> dataBeans = ecgSyncListResponse.data;
```

```
                for (EcgSyncListResponse.DataBean dataBean : dataBeans) {
                    List<Integer> lists = SharedPreferencesUtil.readEcgListMsg(
                            new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date(dataBean.collectStartTime)),
                            EcgActicvity.this);
                    if (lists != null && lists.size() > 0) {
                        //获取到数据后，删除手表中的数据
                        deleteEcgInfo(dataBean.collectSendTime);
                    } else {
                        datas.add(dataBean);
                    }
                }
            }
        }
});

public class EcgSyncListResponse {
    public int collectType;
    public int code;
    public List<DataBean> data;

    public class DataBean {
        public int collectDigits;
        public int collectType;
        public int collectSN; // 手环数据条数
        public int collectTotalLen;
        public long collectSendTime;
        public long collectStartTime;
        public int collectBlockNum;
    }
}
/***
 * 同步历史原始数据
 * @param type 0x00: 心电图数据（ECG） 0x01: PPG 数据 0x02: 三轴加速度数据 0x03: 六轴传感器数据（3 轴加速度+3 轴陀螺仪）
 *                   0x04: 九轴传感器数据（3 轴加速度+3 轴陀螺仪+3 轴磁力计） 0x05:三轴磁力计数据 0x06: 充气血压数据 0x07: PPI 数据
 * @param dataResponse
 */
public static void collectHistoryListData(int type, BleDataResponse dataResponse)

/***
 * 删除历史原始数据
 * @param type 0x00: 心电图数据（ECG） 0x01: PPG 数据 0x02: 三轴加速度数据 0x03: 六轴传感器数据（3 轴加速度+3 轴陀螺仪）
 *                   0x04: 九轴传感器数据（3 轴加速度+3 轴陀螺仪+3 轴磁力计） 0x05:三轴磁力计数据 0x06: 充气血压数据
 * @param    timestamp 时间戳，如果是 (0xFFFFFFFF) 代表全删除
 * @param dataResponse
 */
public static void deleteHistoryListData(int type, long timestamp, BleDataResponse dataResponse)
```

7.4 删除设备历史数据

```
/***
 * 删除历史原始数据
 * @param type 0x00: 心电图数据（ECG） 0x01: PPG 数据 0x02: 三轴加速度数据 0x03: 六轴传感器数据（3 轴加速度+3 轴陀螺仪）
```

```
 *                    0x04: 九轴传感器数据（3 轴加速度+3 轴陀螺仪+3 轴磁力计） 0x05:三轴磁力计数据 0x06: 充气血压数据
 * @param    timestamp 时间戳，如果是 (0xFFFFFFFF) 代表全删除
 * @param dataResponse
 */
public static void deleteHistoryListData(int type, long timestamp, BleDataResponse dataResponse)
YCBTClient.deleteHistoryListData(0, sendTime, new BleDataResponse() {
    @Override
    public void onDataResponse(int code, float ratio, HashMap resultMap) {
        if (code == 0) {
            //删除成功
        } else {
            //删除失败
        }
    }
});
```

7.5 获取设备启动测量的 ECG 和 PPG 数据

1. 如果是设备启动的 ECG 测量，只能获取到测量的 ECG 或 PPG 数据，其它数据都没有。
2. 获取数据的相关操作，请参照历史采集数据章节。

8 运动

```
public class SportType {
    public static final int RESERVED = 0;//预留
    public static final int RUN = 1;//跑步
    public static final int SWIMMING = 2;//游泳
    public static final int RIDE = 3;//骑行
    public static final int FITNESS = 4;//健身
    public static final int SCRAP = 5;//预留
    public static final int SKIPPING_ROPE = 6;//跳绳
    public static final int BASKETBALL = 7;//篮球
    public static final int WALKING = 8;//健走
    public static final int BADMINTON = 9;//羽毛球
    public static final int FOOTBALL = 10;//足球
    public static final int MOUNTAINEERING = 11;//登山
    public static final int PING_PONG = 12;//乒乓球
    public static final int FREE_MODE = 13;//自由模式
    public static final int RUN_INDOORS = 14;//室内跑步（或跑步机）
    public static final int RUN_OUTSIDE = 15;//户外跑步
    public static final int WALK_OUTDOOR = 16;//户外步行
    public static final int WALK_INDOOR = 17;//室内步行
    public static final int WALK_AND_RUN = 18;//走跑模式
    public static final int INDOOR_CYCLING = 19;//室内骑行（或动感单车）
    public static final int STEPPER = 20;//踏步机
    public static final int ROWING_MACHINE = 21;//划船机
    public static final int REAL_TIME_MODE = 22;//实时监护模式
    public static final int SIT_UPS = 23;//仰卧起坐
    public static final int LEAPING_MOTION = 24;//跳跃运动
    public static final int WEIGHT_TRAINING = 25;//重量训练
    public static final int YOGA = 26;//瑜伽
```

```
    public static final int ONFOOT = 0x1B;// 徒步
    public static final int VOLLEYBALL = 0x1C;// 排球
    public static final int KAYAK = 0x1D;// 皮划艇
    public static final int ROLLER_SKATING = 0x1E;// 轮滑
    public static final int TENNIS = 0x1F;// 网球
    public static final int GOLF = 0x20;// 高尔夫
    public static final int ELLIPTICAL_MACHINE = 0x21;// 椭圆机
    public static final int DANCE = 0x22;// 舞蹈
    public static final int ROCK_CLIMBING = 0x23;// 攀岩
    public static final int AEROBICS = 0x24;// 健身操
    public static final int OTHERSPORTS = 0x25;// 其他运动
}
```

8.1 运动的启动与关闭

```
/***
 * 运动模式
 * @param sportSwitch 0x00: 停止 0x01: 开始 0x02: 暂停 0x03: 继续
 * @param sportType 类型选择 0x00: 预留 0x01: 跑步 0x02: 游泳 0x03: 骑行 0x0C: 乒乓球
 * 0x04: 健身 0x08: 健走 0x05: 预留 0x09: 羽毛球 0x06: 跳绳 0x0A: 足球 0x07: 篮球 0x0B: 登山
 * 0x0C: 乒乓球 0x0D：自由模式 0x0E:室内跑步 0x0F:户外跑步 0x10:户外步行 0x11：室内步行
 * 0x12：走跑模式 0x13:室内骑行 0x14:踏步机 0x15:划船机 0x16:实时监护模式  0x17:仰卧起坐  0x18:跳跃运动  0x19:重量训练 0x1A:瑜伽
 * @param dataResponse 0x00 同步成功  0x01 同步失败
 */
public static void appRunMode(int sportSwitch, int sportType, BleDataResponse dataResponse)
YCBTClient.appRunMode(Constants.SportState.Start,Constants.SportType.RUN,dataResponse);
/***
 * 注册监听实时数据
 */
public static void appRegisterRealDataCallBack(BleRealDataResponse realDataResponse)
YCBTClient.appRegisterRealDataCallBack(new BleRealDataResponse() {

    @Override
    public void onRealDataResponse(int dataType, HashMap dataMap) {
        if (dataMap == null) {
            return;
        }
        if (YCBTClient.isSupportFunction(Constants.FunctionConstant.ISHASREALEXERCISEDATA)) {
            if (dataType == Constants.DATATYPE.Real_UploadOGA) {
                if(mSportType != SportType.RIDE){
                    mRunInfo.distance = (int) dataMap.get("sportsRealDistance");// 总距离
                    mRunInfo.calorie = (int) dataMap.get("sportsRealCalories");// 卡路里
                }
                mRunInfo.runTime = (int) dataMap.get("recordTime");// 运动时长
                mRunInfo.heart = (int) dataMap.get("heartRate");// 心率
                sportStep = (int) dataMap.get("sportsRealSteps");// 总步数
            }
        } else {
            if (dataType == Constants.DATATYPE.Real_UploadSport) {
                if (isFirst) { // 初始距离 初始卡路里 初始步数
                    startDistance = (int) dataMap.get("sportDistance");
                    startCalorie = (int) dataMap.get("sportCalorie");
```

```
                        startSportStep = (int) dataMap.get("sportStep");
                        isFirst = false;
                    }
                    sportStep = (int) dataMap.get("sportStep");//步数
                    sportDistance = (int) dataMap.get("sportDistance");//距离
                    sportCalorie = (int) dataMap.get("sportCalorie");//卡路里
                    mRunInfo.distance = (sportDistance - startDistance);//总距离
                    mRunInfo.calorie = (sportCalorie - startCalorie);//总卡路里
                    sportStep = (sportStep - startSportStep);//总步数
                } else if (dataType == Constants.DATATYPE.Real_UploadHeart) {
                    heartValue = (int) dataMap.get("heartValue");//  心率
                    mRunInfo.heart = heartValue;
                }
            }
        }
});

public class RunInfo implements Serializable {
    public int type = -1;//运动类型：0：预留，1：跑步，2：游泳，3：骑行，4：健身，5：预留 6:跳绳 7: 篮球 8: 健走 9: 羽毛球 10: 足球 11: 登山 12: 乒乓球
    public long beginDate; // 开始时间
    public float distance; // 总距离 手环
    public float distances; // 总距离 手环 里程 distance / 1000
    public int calorie;// 卡路里
    public String minkm; //配速
    public int heart; // 心率
    public int runTime; // 运动时长
    public float kmh; //时速
    public String mapCoordinatesList; //地图开始结束坐标    轨迹
    public Boolean isUpload; // 是否上传
}
YCBTClient.deviceToApp(new BleDeviceToAppDataResponse() {
  @Override
  public void onDataResponse(int i, HashMap hashMap) {
      if (hashMap != null) {
          if (i == 0) {
              int dataType = (int) hashMap.get("dataType");
              int data = -1;
              if (hashMap.get("data") != null)
                  data = (int) hashMap.get("data");
              switch (dataType) {
                  case Constants.DATATYPE.DeviceSportModeControl:
                      if (hashMap.get("datas") != null) {
                          // 判断如果是开启运动的指令，打开运动界面
                          if (((byte[]) hashMap.get("datas")).length >= 2 && ((byte[]) hashMap.get("datas"))[0] == 1) {
                              Intent intent = new Intent(context, SportRunningActivity.class);
                              intent.putExtra("Title", ((byte[]) hashMap.get("datas"))[1] & 0xff);
                              startActivity(intent);
                          } else {
                              //更新运动数据
                              EventBus.getDefault().post(new EventBusExitExerciseEvent((byte[]) hashMap.get("datas")));
                          }
                      }
                      break;
```

```
                    }
                }
            }
        }
});
```

```
/**
 *  读取手环运动实时数据
 *
 * @param type                0x00:  关闭  0x01:开启
 * @param dataResponse
 */
public static void appRealSportFromDevice(int type, final BleDataResponse dataResponse)
YCBTClient.appRealSportFromDevice(type, new BleDataResponse() {
    @Override
    public void onDataResponse(int i, float v, HashMap hashMap) {
    }
});
if (type == 1) {
    YCBTClient.appRegisterRealDataCallBack(new BleRealDataResponse() {
        @Override
        public void onRealDataResponse(int i, HashMap hashMap) {
            if (i == Constants.DATATYPE.Real_UploadSport) {
                if (hashMap != null && hashMap.size() > 0) {
                    sportStep = (int) hashMap.get("sportStep");//步数
                    sportDistance = (int) hashMap.get("sportDistance");//距离
                    sportCalorie = (int) hashMap.get("sportCalorie");//卡路里
                }
            }
        }
    });
}
```

8.2 运动历史数据

绝大部分设备，由启动运行后，都不会记录运动的相关数据，需要 App自己处理。
对于某些定制设备，会将运动数据进行保存。

9 设置

9.1 关机、复位、重启

```
/**
 *  关机、重启、进入运输模式控制
 *  运输模式下，会非常省电，需要充电才能开机。
 * @param type              0x01:  关机  0x02:  进入运输模式  0x03:  重启     0x04:设备主动断开连接
 * @param dataResponse 0x00:  设置成功  0x01:  设置失败
 */
public static void appShutDown(int type, BleDataResponse dataResponse)
YCBTClient.appShutDown(1,dataResponse);
```

9.2 恢复出厂设置

```
/***
 * 恢复出厂设置
 * @param dataResponse 0x00: 设置成功 0x01: 设置失败
 */
public static void settingRestoreFactory(BleDataResponse dataResponse)
YCBTClient.settingRestoreFactory(dataResponse);
```

9.3 清空数据队列

sdk 指令以队列保存，先进先出，可以调用此方法清除所有指令

```
/***
 * 清空数据队列
 */
public static void resetQueue()
YCBTClient.resetQueue();
```

9.4 设置语言

```
/***
 * 语言设置
 * @param langType 0x00:英语 0x01: 中文 0x02: 俄语 0x03: 德语 0x04: 法语
 * 0x05: 日语 0x06: 西班牙语 0x07: 意大利语 0x08: 葡萄牙文
 * 0x09: 韩文 0x0A: 波兰文 0x0B: 马来文 0x0C: 繁体中文 0xFF:其它
 * @param dataResponse 0x00: 设置成功 0x01: 设置失败
 */
public static void settingLanguage(int langType, BleDataResponse dataResponse)
YCBTClient.settingRestoreFactory(1,dataResponse);
```

9.5 设置单位

```
/***
 * 单位设置 (不需要的设置项置 0)
 * @param distanceUnit 0x00:km 0x01:mile 距离
 * @param weightUnit 0x00:kg 0x01:lb 0x02:st 体重
 * @param temperatureUnit 0x00: °C 0x01: °F 温度
 * @param timeFormat 0x00:24 小时 0x01:12 小时 时间制式
 * @param dataResponse 0x00:成功 0x01:失败
 */
public static void settingUnit(int distanceUnit, int weightUnit, int temperatureUnit, int timeFormat, BleDataResponse dataResponse)
```

9.6 目标设置

如果参数赋值错误，或者是设备不支持目标设置，都会返回失败。

```
/***
 * 目标设置
 * @param goalType 0x00:步数  0x01:卡路里  0x02:距离  0x03:睡眠  0x04:运动时间  0x05:有效步数
 * @param target  目标值(类型为  睡眠时，此处填充  0x00)
 * @param sleepHour  睡眠目标:时  (类型为非睡眠时，此处填充  0x00)
 * @param sleepMinute  睡眠目标:分  (类型为非睡眠  时，此处填充  0x00)
 * @param dataResponse
 */
public static void settingGoal(int goalType, int target, int sleepHour, int sleepMinute, BleDataResponse dataResponse)
YCBTClient.settingGoal(0x00, 1000, 0x00, 0x00,dataResponse);
```

9.7 用户信息设置

```
/***
 * 用户信息设置
 * @param height  身高 cm 50 ~ 255
 * @param weight  体重(kg)
 * @param sex  性别  0 男  1 女
 * @param age  年龄  1~150
 * @param dataResponse
 */
public static void settingUserInfo(int height, int weight, int sex, int age, BleDataResponse dataResponse)
YCBTClient.settingUserInfo(170,50,1,30,dataResponse);
```

9.8 设置久坐提醒

```
/***
 * 设置久坐提醒
 * @param time1StartHour  时间段 1 开始小时
 * @param time1StartMin  时间段 1 开始分
 * @param time1EndHour  时间段 1 结束小时
 * @param time1EndMin  时间段 1 结束分
 * @param time2StartHour  时间段 2 开始小时
 * @param time2StartMin  时间段 2 开始分
 * @param time2EndHour  时间段 2 结束小时
 * @param time2EndMin  时间段 2 结束分
 * @param intervalTime    间隔时间(分钟)
 * @param repeat  周重复
 * bit7    bit6 bit5 bit4 bit3 bit2 bit1 bit0
 *  总开关  周日  周六    周五  周四    周三    周二    周一
 * 0 关 1 开
 * @param dataResponse
 */
public static void settingLongsite(int time1StartHour, int time1StartMin, int time1EndHour, int time1EndMin,
                                   int time2StartHour, int time2StartMin, int time2EndHour, int time2EndMin,
                                   int intervalTime, int repeat, BleDataResponse dataResponse)
YCBTClient.settingLongsite(mSedentaryAmStartHour, mSedentaryAmStartMin, mSedentaryAmEndHour, mSedentaryAmEndMin, mSedentaryPmStartHour, mSedentaryPmStartMin, mSedentaryPmEndHour, mSedentaryPmEndMin, mSedentaryIntervalTime,
mWeek,dataResponse);
```

9.9 通知提醒开关设置

```
/***
 * 通知提醒开关设置
 * @param on 通知提醒总开关 0x00: 关 0x01: 开
 * @param sub1 提醒项子开关 Bit7:来电 Bit6:信短 Bit5:邮件 Bit4:微信 Bit3:QQ Bit2:新浪微博 Bit1:facebook Bit0:twitter 0:关 1：开
 * @param sub2 提醒项子开关 Bit7:Messenger Bit6:WhatsAPP Bit5:Linked in Bit4:Instagram Bit3:Skype Bit2:Line Bit1:Snapchat Bit0:Telegram 0: 关 1：开
 * @param sub3 提醒项子开关 Bit7:预留 Bit6:预留 Bit5:预留 Bit4:预留 Bit3:预留 Bit2:预留 Bit1:Viber Bit0:其他 0: 关 1：开
 * @param dataResponse
 */
public static void settingNotify(int on, int sub1, int sub2, int sub3, BleDataResponse dataResponse)
YCBTClient.settingNotify(on, mMessageOne, mMessageTwo, mMessageThree,dataResponse);
```

9.10 防丢提醒

防丢指的是设备与手机连接信号变弱或断连后，手环会发生震动。防丢类型中的后三个取值效果是相同的。

```
/***
 * 防丢提醒设置
 * @param type 0x00: 不防丢 0x01: 近距离防丢 0x02: 中距离防丢(默认) 0x03: 远距离防丢
 * @param dataResponse 0x00: 成功 0x01: 失败
 */
public static void settingAntiLose(int type, BleDataResponse dataResponse)
YCBTClient.settingAntiLose(3, dataResponse);
```

9.11 健康监测

健康监测指的是设备会在固定时间测量对应的数据并保存。
健康监测中，心率监测和温度监测是通用的功能，也可以满足约大部分的场景。其它如血氧监测只有个别设备才可能用到。

9.11.1 心率监测

```
/****
 * 心率监测模式设置
 * @param mode 模式 0x00: 手动模式 0x01: 自动模式
 * @param intervalTime 自动模式下心率监测间隔(1-60 分)
 * @param dataResponse
 */
public static void settingHeartMonitor(int mode, int intervalTime, BleDataResponse dataResponse)
YCBTClient.settingHeartMonitor(0x01, 60, dataResponse);
```

9.11.2 温度监测

```
/**
 * 温度监测
 *
 * @param on_off 温度监测开关
 * @param interval 间隔时间 分钟
 * @param dataResponse
 */
public static void settingTemperatureMonitor(boolean on_off, int interval, BleDataResponse dataResponse)
```

```
YCBTClient.settingTemperatureMonitor(true, 60, dataResponse);
```

9.11.3 环境光检测

```
/**
 * 环境光检测设置
 *
 * @param on_off          环境光检测开关
 * @param interval       环境光监测间隔（分）
 * @param dataResponse
 */
public static void settingAmbientLight(boolean on_off, int interval, BleDataResponse dataResponse)
YCBTClient.settingAmbientLight(true, 60, dataResponse);
```

9.11.4 血氧监测

```
/**
 * 血氧监测模式设置
 *
 * @param on_off          血氧监测模式开关
 * @param interval       间隔时间  分钟
 * @param dataResponse
 */
public static void settingBloodOxygenModeMonitor(boolean on_off, int interval, BleDataResponse dataResponse)
YCBTClient.settingBloodOxygenModeMonitor(true, 60, dataResponse);
```

9.11.5 环境温湿度监测

```
/**
 * 环境温湿度检测模式设置
 *
 * @param on_off          环境温湿度检测模式开关
 * @param interval       间隔时间  分钟
 * @param dataResponse
 */
public static void settingAmbientTemperatureAndHumidity(boolean on_off, int interval, BleDataResponse dataResponse)
YCBTClient.settingAmbientTemperatureAndHumidity(true, 60, dataResponse);
```

9.12 健康值预警

9.12.1 心率告警

```
/***
 * 心率告警设置
 * @param on  报警开关  0x00:  关
 * @param highHeart  最高心率告警阈值  100 – 240
 * @param lowHeart  最低心率告警阈值  30 - 60
 * @param dataResponse
 */
public static void settingHeartAlarm(int on, int highHeart, int lowHeart, BleDataResponse dataResponse)
YCBTClient.settingHeartAlarm(1, 100，30, dataResponse);
```

9.12.2 温度报警

```
/**
 *  温度报警
 *
 * @param on_off           温度报警开关
 * @param temp              温度报警上限值(-127 - 127)
 * @param dataResponse
 */
public static void settingTemperatureAlarm(boolean on_off, int temp, BleDataResponse dataResponse)
YCBTClient.settingTemperatureAlarm(true, 38，dataResponse);
```

9.12.3 血压预警

```
/**
 *  血压报警设置
 *
 * @param on_off           报警开关  0x00:关    0x01:开
 * @param maxSBP            最高收缩压报警阈值  140 – 250 mmHg
 * @param maxDBP            最高舒张压报警阈值  90 - 160 mmHg
 * @param minSBP           最低收缩压报警阈值  50-90 mmHg
 * @param minDBP            最低舒张压报警阈值  40-60 mmHg
 * @param dataResponse
 */
public static void settingBloodAlarm(int on_off, int maxSBP, int maxDBP, int minSBP, int minDBP, BleDataResponse dataResponse)
YCBTClient.settingBloodAlarm(1, 140,90,50,40,dataResponse);
```

9.12.4 血氧预警

```
/**
 *  血氧报警设置
 *
 * @param on_off              0x00：关闭     0x01:打开
 * @param minBloodOxygen  最低血氧报警阈值  80- 95
 * @param dataResponse
 */
public static void settingBloodOxygenAlarm(int on_off, int minBloodOxygen, BleDataResponse dataResponse)
YCBTClient.settingBloodOxygenAlarm(1, 80,dataResponse);
```

9.12.5 呼吸率预警

```
/**
 *  呼吸率报警
 *
 * @param onOff             类型  0x00：关    0x01：开
 * @param high              最高呼吸率报警阀值
 * @param low               最低呼吸率报警阀值
 * @param dataResponse code 0x00:  设置成功    0x01:  设置失败
 */
public static void settingRespiratoryRateAlarm(int onOff, int high, int low, BleDataResponse dataResponse)
YCBTClient.settingRespiratoryRateAlarm(1, 30，6,dataResponse);
```

9.13 勿扰设置

手环进入勿扰模式后，所有的提醒类功能都不工作。

```
/**
 * 勿扰模式设置
 *
 * @param on                    0x00:关  0x01:开
 * @param startTimeHour  开始时间：时
 * @param startTimeMin    开始时间：分
 * @param endTimeHour      结束时间：时
 * @param endTimeMin       结束时间：分
 * @param dataResponse    code 0x00:  设置成功    0x01:  设置失败
 */
public static void settingNotDisturb(int on,
                                                    int startTimeHour, int startTimeMin,
                                                    int endTimeHour, int endTimeMin, BleDataResponse dataResponse)
YCBTClient.settingNotDisturb(1, 6, 30, 16, 30, dataResponse);
```

9.14 抬腕亮屏开关

```
/***
 * 抬腕亮屏开关设置
 * @param on 0x00:关闭  0x01:  打开
 * @param dataResponse    code 0x00:  设置成功    0x01:  设置失败
 */
public static void settingRaiseScreen(int on, BleDataResponse dataResponse)
YCBTClient.settingRaiseScreen(1,dataResponse);
```

9.15 屏幕设置

9.15.1 屏幕亮度

```
/***
 * 显示屏亮度设置
 * @param level 0x00:低    0x01:  中    0x02:  高    0x03：自动    0x04:  较低    0x05：较高
 * @param dataResponse code 0x00:  设置成功    0x01:  设置失败
 */
public static void settingDisplayBrightness(int level, BleDataResponse dataResponse)

public class DisplayBrightness{
        public static final int Low = 0;
        public static final int Middle = 1;
        public static final int High = 2;
        public static final int Auto = 3;
        public static final int Lower = 4;
        public static final int Higher = 5;
}
YCBTClient.settingDisplayBrightness(Constants.DisplayBrightness.Middle,dataResponse);
```

9.15.2 息屏时间

注意时间间隔不是具体的数值 ，而是 PauseTime 。

```
/**
 * 息屏时间设置
 *
 * @param level              0x00：5s 0x01: 10s 0x02: 15s 0x03：30s
 * @param dataResponse
 */
public static void settingScreenTime(int level, BleDataResponse dataResponse)

public class PauseTime{
      public static final int Time5s = 0;
      public static final int Time10s = 1;
      public static final int Time15s = 2;
      public static final int Time30s = 3;
}
YCBTClient.settingScreenTime(Constants.PauseTime.Time5s,dataResponse);
```

9.16 肤色设置

肤色设置会影响到设备检测健康数据与 ECG 测试等，一般皮肤越黑，毛发越多的用户，取值越大。

```
/***
 * 肤色设置
 * @param skinColor   0x00:白 0x01: 白间黄
 * 0x02: 黄 0x03: 棕色 0x04: 褐色 0x05: 黑 0x07: 其它
 * @param dataResponse
 */
public static void settingSkin(int skinColor, BleDataResponse dataResponse)

public class SkinColor {
      /**
       * 白
       */
      public static final int SKIN_WHITE = 0;
      /**
       * 白间黄
       */
      public static final int SKIN_WHITE_BETWEEN_YELLOW = 1;
      /**
       * 黄色
       */
      public static final int SKIN_YELLOW = 2;
      /**
       * 棕色
       */
      public static final int SKIN_BROWN = 3;
      /**
       * 褐色
       */
      public static final int SKIN_BROWNNESS = 4;
      /**
       * 黑色
```

```
     */
    public static final int SKIN_BLACK = 5;
}
YCBTClient.settingSkin(Constants.SkinColor.SKIN_BROWNNESS,dataResponse);
```

9.17 血压范围设置

当测量到的光电血压与实际血压偏差较大时，可以设置设备的血压等级值来进行校
正。
注意：如果设备有血压校准功能，此不需要使用此功能，直接调用血压校准

```
/***
 * 血压范围设置
 * @param level 0x00:偏低 0x01: 正常 0x02: 轻微偏高 0x03: 中度偏高 0x04: 重度高
 * @param dataResponse
 */
public static void settingBloodRange(int level, BleDataResponse dataResponse)
YCBTClient.settingBloodRange(1,dataResponse);
```

9.18 设备名称设置

设置名称不允许超过 12 字节，不建议使用特殊字符。
此方法用于工厂生产使用，开发普通应用程序不需要使用

```
/***
 * 设备名称设置
 *
 * @param dataResponse
 */
public static void settingDeviceName(String name, BleDataResponse dataResponse)
YCBTClient.settingDeviceName(replace,dataResponse);
```

9.19 主题设置

获取主题

```
/***
 * 获取设备主界面样式配置
 * @param dataResponse BleDataResponse
 */
public static void getThemeInfo(BleDataResponse dataResponse)
 YCBTClient.getThemeInfo(new BleDataResponse() {
     @Override
     public void onDataResponse(int code, float ratio, HashMap resultMap) {
     }
});
```

设置主题

```
/***
 * 设备主界面样式配置
 * @param themeType  样式(0-(N-1)), N 代表设备支持的主界面数量，由获取指令查询
 * @param dataResponse
 */
public static void settingMainTheme(int themeType, BleDataResponse dataResponse)
YCBTClient.settingMainTheme(0,dataResponse);
```

9.20 提醒设置

9.20.1 设置睡眠提醒时间

设置成功后，当前时间进入提醒时间时，设备将震动显示睡眠提醒画面

```
/***
 * 设置睡眠提醒时间
 * @param startHour 时
 * @param startMin 分
 * @param repeat 周重复
 * bit7    bit6 bit5 bit4 bit3 bit2 bit1 bit0
 * 总开关 周日 周六   周五 周四   周三   周二   周一
 * @param dataResponse
 */
public static void settingSleepRemind(int startHour, int startMin, int repeat, BleDataResponse dataResponse)
YCBTClient.settingSleepRemind(1，1，0xff,dataResponse);
```

9.20.2 断开或运动达标提醒设置

当设备达到设置的提醒条件时，将产生震动

```
/**
 * 手环状态提醒设置
 *
 * @param on_off          手环状态提醒开关
 * @param type            提醒类型 0x00: 蓝牙断开提醒 0x01：运动达标提醒
 * @param dataResponse
 */
public static void settingBraceletStatusAlert(boolean on_off, int type, BleDataResponse dataResponse)
YCBTClient.settingBraceletStatusAlert(1，1,dataResponse);
```

9.20.3 上传提醒

用于表示设备存储数据达到指定比例时，产生上传提醒。

```
/**
 * 上传提醒设置
 *
 * @param on_off          上传提醒开关
 * @param threshold       存储阈值
 * @param dataResponse
 */
public static void settingUploadReminder(boolean on_off, int threshold, BleDataResponse dataResponse)
YCBTClient.settingSleepRemind(true，80,dataResponse);
```

9.21 数据采集配置

不是所有的设备都支持文档中列举的类型
注意方法中的最后两个参数的单位和取值
文档中列出了两个方法，其中第二个方法是给某些定制设备使用的，两个方法的部分
参数是不相同的。

```
/***
 * 数据采集配置
```

```
 * @param on 0x01: 开启 0x00: 关闭
 * @param type 0x00: PPG 0x01: 加速度数据 0x02：ECG 0x03：温湿度 0x04：环境光 0x05：体温
 * @param collectLong 每次采集时长(单位:秒) (关闭时填 0)
 * @param collectInterval 采集间隔(单位:分钟) (关闭时填 0)
 * @param dataResponse
 */
public static void settingDataCollect(int on, int type, int collectLong, int collectInterval, BleDataResponse dataResponse)
/**
 * 不同工作模式下数据采集配置
 *
 * @param mode              模式  0x00：设置为正常工作模式采集配置  0x01:设置为关怀工作模式采集配置  0x02：设置为省电工作模式采集配置  0x03:设置运动模式下采集配置
 * @param type              数据类型 0x00:PPG,  0x01:六轴,  0x02：ECG,  0x03：温湿度,  0x04：环境光,  0x05：体温,  0x06：心率
 * @param secondTime     每次采集时长（单位：  秒）(关闭时填 0）
 * @param interval       采集间隔(单位：分钟) （关闭时填 0）
 * @param dataResponse
 */
public static void settingConfigInDifWorkingModes(int mode, int type, int secondTime, int interval, BleDataResponse dataResponse)
YCBTClient.settingDataCollect(1, 0x00, 90, 60, dataResponse);

YCBTClient.settingConfigInDifWorkingModes(0,1, 0x00, 90, 60, dataResponse);
```

9.22 工作模式切换

设备进入不同的工作模式后，设备的监测时间或采样频率等都会发生变化

```
/**
 * 工作模式切换设置
 *
 * @param level            0x00：设置为正常工作模式 0x01: 设置为关怀工作模式 0x02：设置为省电工作模式 0x03: 设置为自定义工作模式
 * @param dataResponse
 */
public static void settingWorkingMode(int level, BleDataResponse dataResponse)
YCBTClient.settingWorkingMode(3,dataResponse);
```

9.23 意外监测开关(保留)

保留接口，暂无设备支持。

```
/**
 * 意外监测模式设置
 *
 * @param on_off          意外监测模式开关
 * @param dataResponse
 */
public static void settingAccidentMode(boolean on_off, BleDataResponse dataResponse)
YCBTClient.settingAccidentMode(true,dataResponse);
```

9.24 计步时间设置

设置设备的计步频率，固定取值为 10， 5， 1， 单位是分钟。

```
/**
 * 计步状态时间设置
 *
 * @param interval       计步状态时间 分钟
```

```
 * @param dataResponse
 */
public static void settingStepCountingStateTime(int interval, BleDataResponse dataResponse)
YCBTClient.settingStepCountingStateTime(10,dataResponse);
```

9.25 蓝牙广播间隔设置

注意，参数中的时间间隔的单位和取值，取值必须是 0.625ms 的整位倍。

```
/**
 *  设置蓝牙广播间隔
 *
 * @param time              广播间隔时间    单位 ms    20 ~ 10240ms
 * @param dataResponse
 */
public static void settingBluetoothBroadcastInterval(int time, BleDataResponse dataResponse)
YCBTClient.settingBluetoothBroadcastInterval(2000,dataResponse);
```

9.26 蓝牙发射功率设置

发射功率最好不要为负数，需要大于等于 0DBM。

```
/**
 *  设置蓝牙发射功率
 *
 * @param power             发射功率  >= 0 dbm
 * @param dataResponse
 */
public static void settingBluetoothTransmittingPower(int power, BleDataResponse dataResponse)
YCBTClient.settingBluetoothTransmittingPower(0,dataResponse);
```

9.27 运动心率区间设置

依据不同的运动类型，设置不同的心率范围。

```
/**
 *  运动心率区间设置
 *
 * @param type              0:休养静歇，  1:休闲热身，  2:心肺强化，  3:减脂塑形，  4:运动极限，  5:空状态
 * @param minHeart          该模式下最小心率
 * @param maxHeart          该模式下最大心率
 * @param dataResponse
 */
public static void settingExerciseHeartRateZone(int type, int minHeart, int maxHeart, BleDataResponse dataResponse)
YCBTClient.settingExerciseHeartRateZone(0,60,100,dataResponse);
```

9.28 保险界面开关

仅仅控制设备的保险功能界面是否显示

```
/**
 *  保险界面显示开关控制
 *
 * @param on_off            0x00：关闭保险界面显示，  0x01:打开保险界面显示
 * @param dataResponse
```

```
 */
public static void settingInsuranceSwitch(int on_off, BleDataResponse dataResponse)
YCBTClient.settingInsuranceSwitch(1,dataResponse);
```

9.29 马达振动时长设置

可以修改马达振动时长，注意单位是毫秒。

```
/**
 *  振动时间设置
 *
 * @param type              类型  0x00:闹钟
 * @param time              振动时间，单位为毫秒
 * @param dataResponse
 */
public static void settingVibrationTime(int type, int time, BleDataResponse dataResponse)
YCBTClient.settingVibrationTime(0,1000,dataResponse);
```

9.30 事件

事件的基本处理与功能与闹钟比较类似，事件与闹钟的最大区别是可以增加备注名称，事件是某些定制设备才支持。

9.30.1 事件提醒设置

事件名称要注意长度, 事件间隔的时间使用枚举值而非具体值 ，它的单位是分钟。
设置成功后，会返回对应的事件 id, 1 ~ 10

```
/**
 *  事件提醒设置
 *
 * @param code             0x01：添加事件提醒   0x02：删除事件提醒   0x03：修改事件提醒   最多 10 个事件
 * @param index            时间 id   1-10
 * @param on_off           0x00：关   0x01：开
 * @param type             0x00：闹钟   0x01：自定义事件
 * @param hour             响应的小时
 * @param min              响应的分钟
 * @param weekRepeat       重复
 *                         bit7   bit6 bit5 bit4 bit3 bit2 bit1 bit0
 *                         保留    周日 周六   周五 周四   周三   周二   周一
 * @param interval         间隔时间  0x00:单次   0x01:10min 0x02:20min 0x03:30min
 * @param name             事件的名字  最大 4 个中文
 * @param dataResponse
 */
public static void settingEventReminder(int code, int index, int on_off, int type, int hour, int min, int weekRepeat, int interval, String name, BleDataResponse dataResponse)
//添加事件
YCBTClient.settingEventReminder(1,1,1,1,1,1,0,0,"name",dataResponse);

//删除事件，通过指定事件编号就可以直接删除
YCBTClient.settingEventReminder(2,1,1,1,1,1,0,0,"name",dataResponse);
```

9.30.2 事件提醒开关控制

设置设备事件是否生效，如果为 false，所有的事件都无效。

```
/**
 * 事件提醒开关控制
 *
 * @param on_off            0x00：关闭     0x01:打开
 * @param dataResponse
 */
public static void settingEventReminderSwitch(int on_off, BleDataResponse dataResponse)
YCBTClient.settingEventReminderSwitch(1,dataResponse);
```

9.30.3 获取事件提醒信息

```
/***
 * 获取当前手环事件提醒信息
 * @param dataResponse
 */
public static void getEventReminderInfo(BleDataResponse dataResponse)
YCBTClient.getEventReminderInfo(dataResponse);
```

9.31 时间

```
/**
 * 同步时间
 *
 * @param dataResponse
 */
public static void settingTime(BleDataResponse dataResponse)
YCBTClient.settingTime(dataResponse);
```

9.32 设置手机系统

```
/***
 * 手机操作系统设置
 * @param os 0x00: Android 0x01: IOS
 * @param devVersion 版本号
 * @param dataResponse
 */
public static void settingAppSystem(int os,String devVersion, BleDataResponse dataResponse)
YCBTClient.settingAppSystem(0, Build.VERSION.RELEASE,dataResponse);
```

9.33 设置设备 mac

```
/**
 * APP 设置设备 MAC
 *
 * @param mac               设备 Mac 地址   格式为 12:31:0A:AB:AF:9F   或 12310AABAF9F
 * @param dataResponse
 */
public static void settingDeviceMac(String mac, BleDataResponse dataResponse)
```

10 获取设备数据

10.1 获取基本信息

```
/***
 * 获取设备基本信息
 * @param dataResponse    BleDataResponse 设备 ID, 固件版本号,电池状态,电池电量,绑定状态
 */
public static void getDeviceInfo(BleDataResponse dataResponse)
YCBTClient.getDeviceInfo(new BleDataResponse() {
      @Override
      public void onDataResponse(int code, float v, HashMap resultMap) {
         if (code == 0) {
            if (resultMap != null) {
               String backVal = resultMap.toString();
               Gson gson = new Gson();

               //产品类型，0x00:手表/手环     0x01:戒指
               String type = (String) YCBTClient.readDeviceInfo(Constants.FunctionConstant.DEVICETYPE);

               BandBaseInfo bandBaseInfo = gson.fromJson(backVal, BandBaseInfo.class);
      }
});

public class BandBaseInfo {
      private int dataType;//数据类型
      private int code;//返回码
      private BandBaseInfoModel data;

      public static class BandBaseInfoModel{
            private String deviceBatteryValue;//电池电量
            private String deviceVersion;//设备固件版本
      }

}
```

10.2 获取闹钟信息

```
/**
 * 查询闹钟
 *
 * @param dataResponse
 */
public static void settingGetAllAlarm(BleDataResponse dataResponse)
YCBTClient.settingGetAllAlarm(new BleDataResponse() {
      @Override
      public void onDataResponse(int code, float ratio, HashMap resultMap) {
            if (code == 0) {
                  List data = (ArrayList) resultMap.get("data");
                  if (mMeAlarmClock != null) {
                        mMeAlarmClock.clear();
                  }
```

```
                if (data.size() != 0) {
                    for (int i = 0; i < data.size(); i++) {
                        HashMap dataMap = (HashMap) data.get(i);
                        int alarmHour = (int) dataMap.get("alarmHour"); //  时
                        int alarmMin = (int) dataMap.get("alarmMin"); //  分
                        int alarmDelayTime = (int) dataMap.get("alarmDelayTime"); //  贪睡时间
                        int alarmType = (int) dataMap.get("alarmType"); //  类型
                        int alarmRepeat = (int) dataMap.get("alarmRepeat"); //  闹钟重复
                        String mAlarmRepeat = clockRepeatToValueArray(alarmRepeat);
                        char[] ar = mAlarmRepeat.toCharArray();//char 数组
                        mMeAlarmClock.add(new MeAlarmClock(alarmHour, alarmMin, WeekUtil.getLable(ar, context), mAlarmRepeat, ar[ar.length - 1] + ""));
                    }
                }
            }
        }
});
```

10.2.1 添加闹钟

```
/***
 *  添加闹钟
 * @param type 0x00:起床  0x01:睡觉  0x02:锻炼  0x03:吃药  0x04:约会  0x05:聚会  0x06:会议  0x07:自定义
 * @param startHour  开始时间小时
 * @param startMin    开始时间分钟
 * @param weekRepeat  重复&开关
 * bit7    bit6 bit5 bit4 bit3 bit2 bit1 bit0
 *  总开关  周日  周六    周五  周四    周三    周二    周一
 * 0 关 1 开
 * @param delayTime  贪睡时长(单位:分钟) 0-59
 * @param dataResponse  数据返回结果
 */

public static void settingAddAlarm(int type, int startHour, int startMin, int weekRepeat, int delayTime, BleDataResponse dataResponse)
```

10.2.2 修改闹钟

```
/***
 *  修改闹钟
 * @param startHour  以前闹钟小时
 * @param startMin    以前闹钟分钟
 * @param newType    0x00:起床  0x01:睡觉  0x02:锻炼  0x03:吃药  0x04:约会  0x05:聚会  0x06:会议  0x07:自定义
 * @param newStartHour  开始时间小时
 * @param newStartMin  开始时间分钟
 * @param newWeekRepeat  重复&开关
 * bit7    bit6 bit5 bit4 bit3 bit2 bit1 bit0
 *  总开关  周日  周六    周五  周四    周三    周二    周一
 * 0 关 1 开
 * @param newDelayTime  贪睡时长(单位:分钟) 0-59
 * @param dataResponse 0x00:  设置成功  0x01:  设置失败
 */
public static void settingModfiyAlarm(int startHour, int startMin,
```

```
                                        int newType,
                                        int newStartHour, int newStartMin,
                                        int newWeekRepeat, int newDelayTime, BleDataResponse dataResponse)
```

10.2.3 删除闹钟

```
/***
 * 删除闹钟
 * @param startHour  闹钟开始小时
 * @param startMin  闹钟开始分钟
 * @param dataResponse
 */
public static void settingDeleteAlarm(int startHour, int startMin, BleDataResponse dataResponse)
```

10.3 获取芯片信息

```
/***
 * 获取当前手环芯片方案
 * @param dataResponse 0x00：NRF52832    0x01: RTK8762C      0x02: RTK8762D
 */
public static void getChipScheme(BleDataResponse dataResponse)
YCBTClient.getChipScheme(new BleDataResponse() {
    @Override
    public void onDataResponse(int code, float ratio, HashMap resultMap) {
        if (code == 0 && resultMap != null) {
                chipScheme = (int) resultMap.get("chipScheme");
                if(chipScheme == Platform.Nordic){

                }
        }
    }
});

public class Platform {
        public static final int Nordic = 0;//nordic  平台     NRF52832
        public static final int Realtek = 1;//瑞昱平台      RTK8762C
        public static final int Realtek2 = 2;//瑞昱平台      RTK8762D
        public static final int JieLi = 3;//杰理平台      jl701n
}
```

10.4 设备型号

```
/***
 * 获取设备名称或型号信息
 * @param dataResponse    BleDataResponse  设备名称或型号信息
 */
public static void getDeviceName(BleDataResponse dataResponse)
 YCBTClient.getDeviceName(new BleDataResponse() {
        @Override
        public void onDataResponse(int i, float v, HashMap hashMap) {
            if (i == 0 && hashMap != null) {
```

```
                String deviceName = (String) hashMap.get("data");
            }
        }
});
```

11 app 发送通知到设备

消息推送，通知设备

```
/**
 *  消息推送  通知手环
 *
 * @param type               0x00:  来电  0x01:短信  0x02:邮件  0x03:其它  0x04:QQ 0x05:微信  0x06:新浪微博  0x07:Twitter 0x08:Facebook 0x09:Messenger
 *                           0x0A:Whatsapp 0x0B:LinkedIn 0x0C:Instagram 0x0D:Skype 0x0E:Line 0x0F:Snapchat    0x10  拒接电话
 * @param title             消息的标题
 * @param content           消息的内容  消息内容多少由屏幕大小和字体大小决定
 * @param dataResponse 0x00:  成功  0x01:  失败
 */
public static void appSengMessageToDevice(final int type, final String title, final String content, final BleDataResponse dataResponse)
YCBTClient.appSengMessageToDevice(0x01, "title", "content",dataResponse);
```

12 控制设备

12.1 找设备

调用方法后，手环会产生震动。

```
/**
 *  寻找设备
 *
 * @param mode                  0x01:  开始寻找设备  0x00:  结束寻找设备
 * @param remindNum             设备提醒次数
 * @param remindInterval  提醒间隔秒
 * @param dataResponse 0x00:  成功  0x01:  失败
 */
public static void appFindDevice(int mode, int remindNum, int remindInterval, BleDataResponse dataResponse)
//开启，提醒次数 5 次，间隔 2 秒
YCBTClient.appFindDevice(0x01, 5, 2, bleDataResponse);
```

12.2 血压校准

血压校准是指校准光电血压，执行了血压校准后，不需要使用血压等级设置，即二者只用其中一个，且血压校准优先级高。

```
/****
 *  血压校准
 * @param sbp  收缩压      dbp  舒张压
 * @param dataResponse 0x00:  校准成功  0x01:  校准失败-参数错误  0x02:  校准失败-设备未正在测试血压或者血压还没有出值
 */
public static void appBloodCalibration(int sbp, int dbp, BleDataResponse dataResponse)
YCBTClient.appBloodCalibration(SBP, DBP,bleDataResponse);
```

12.3 温度校准

温度校准是用于生产设备时校准温度传感器，使温度测量更加准确，开发应用程序一般不需要使用此方法。

```
/**
 *  温度校准
 *
 * @param tempInt          温度整数部分(-127 - 127)
 * @param tempFloat        温度小数部分(0-99)
 * @param dataResponse
 */
public static void appTemperatureCorrect(int tempInt, int tempFloat, BleDataResponse dataResponse)
YCBTClient.appTemperatureCorrect(40, 5,bleDataResponse);
```

12.4 修改体温二维码的颜色

对于某些个别定制设备可以修改体温二维码的颜色

```
/**
 *  体温二维码控制
 *
 * @param code               温度颜色  0x00:  绿色    0x01:  红色    0x02:  橙色
 * @param dataResponse
 */
public static void appTemperatureCode(int code, BleDataResponse dataResponse)
YCBTClient.appTemperatureCorrect(0,bleDataResponse);
```

12.5 天气数据

对于手环是否支持明天天气设置，可以依据功能属性或返回值来判断。
天气中的温度都是摄氏温度
天气类型只有前 6 个值是通用的，后续列举的取值是某些特殊定制的设备才能使用。
剩余可选参数都是定制设备才能使用，其它设备一律使用null

12.5.1 今日天气

```
/***
 *  今日天气信息数据传送
 * @param lowTemp  最底温
 * @param highTemp  最高温
 * @param curTemp  当前温度
 * @param type  天气类型  1(sunny),   2(cloudy),   3(wind),   4(rain),   5(Snow),   6(foggy),   0(unknown)
 * @param dataResponse
 */
public static void appTodayWeather(String lowTemp, String highTemp, String curTemp, int type, BleDataResponse dataResponse)
YCBTClient.appTodayWeather(min, max, temp, cont,bleDataResponse);
```

12.5.2 明日天气

```
/***
 *  明日天气信息数据传送
 * @param lowTemp  最底温
 * @param highTemp  最高温
```

```
 * @param curTemp  当前温度
 * @param type  天气类型  1(sunny),   2(cloudy),   3(wind),   4(rain),   5(Snow),   6(foggy),   0(unknown)
 * @param dataResponse
 */
public static void appTomorrowWeather(String lowTemp, String highTemp, String curTemp, int type, BleDataResponse dataResponse)
YCBTClient.appTomorrowWeather(tomorrow_min, tomorrow_max, "20", tomorrow_cont,bleDataResponse);
```

12.6 获取实时数据

获取设备的实时数据比较特殊，开启与接收是在两个方法，而为了区分，返回数据类型与控制类型设置不完全一样，Demo 会列出获取数据的过程与步骤，对于没有列举的内容，可能会在其它使用场景出现，如果整篇文档都没有出现则可能不支持或不需要此功能。

12.6.1 开启获取实时数据

```
/**
 *  读取手环运动实时数据
 *
 * @param type               0x00:  关闭  0x01:开启
 * @param mode               0x00:步数、里程、卡路里    0x01:心率    0x07:综合数据
 *                           （0x02:血氧    0x03:血压    0x04:HRV    0x05:呼吸率    0x06:运动模式实时数据  暂时没用到）
 * @param dataResponse
 */
public static void appRealDataFromDevice(int type, int mode, final BleDataResponse dataResponse)
YCBTClient.appRealDataFromDevice(1, 0x00, dataResponse);
```

12.6.2 接收设备的上报的实时数据

```
/***
 *  注册监听实时数据
 */
public static void appRegisterRealDataCallBack(BleRealDataResponse realDataResponse)
YCBTClient.appRegisterRealDataCallBack(new BleRealDataResponse() {
      @Override
      public void onRealDataResponse(int dataType, HashMap hashMap) {
            if(dataType == Constants.DATATYPE.Real_UploadECG){
                  List<Integer> data = (List<Integer>) hashMap.get("data");
            }
      }
});
```

12.7 波形上传

此方法是 SDK 内部使用的方法，可能某些定制设备的特殊场景需要使用。
开启功能后，设备会将波形上报，SDK 内部接收后会以通知的形式发送，需要监听通知。接收数据可以参考上一节的内容。

```
/****
 *  波形上传控制
 * @param enable 0x00:  停止上传  0x01:  开始上传(上传包不带序号) 0x02:  开始上传（上传包带  8  位序号）
 * @param type      0x00:  光电波形  0x01:  心电波形  0x02:  多轴传感器波形  0x03:  环境光
 * @param dataResponse 0x00:  请求成功  0x01:  类型不支持
 */
public static void appControlWave(int enable, int type, BleDataResponse dataResponse)
YCBTClient.appControlWave(1, 0x00, dataResponse);
```

12.8 健康数据测量

App 启动测量由启动测试和接收测量数据两部分完成。

12.8.1 开启与关闭测量

设备启动测量后，测量过程中的值会主动上报，对于解析接收到的数据，请查看 12.6.2 中的代码演示。测量开始后，用户退出了测量界面或者测量结束了，设备都会上报状态，SDK 会将状态发送出来。

```
/**
 * APP 开启实时测量
 *
 * @param onOff            开关    0: 关闭      1: 单次测量      2：多次测量（预留）
 * @param type             类型    0x00:心率    0x01:血压    0x02:血氧    0x03:呼吸率    0x04:体温    0x05：血糖    0x06:尿酸    0x07:血酮
 * @param dataResponse 0x00：设置成功 0x01: 设置失败（参数错误）
 */
public static void appStartMeasurement(int onOff, int type, BleDataResponse dataResponse)
YCBTClient.appStartMeasurement(1, 5,dataResponse);
/***
 * 注册监听实时数据
 */
public static void appRegisterRealDataCallBack(BleRealDataResponse realDataResponse)

YCBTClient.appRegisterRealDataCallBack(new BleRealDataResponse() {
      @Override
      public void onRealDataResponse(int dataType, HashMap dataMap) {
         if(hashMap != null){
            if (dataType == Constants.DATATYPE.Real_UploadHeart) {
               Integer heartValue = (Integer)dataMap.get("heartValue");//心率值
            }
            if (dataType == Constants.DATATYPE.Real_UploadBlood) {
               //血压
               String bloodSBP = hashMap.get("bloodSBP").toString();//收缩压
               String bloodDBP = hashMap.get("bloodDBP").toString();//舒张压
            }
            if (dataType == Constants.DATATYPE.Real_UploadBloodOxygen){
                Integer bloodOxygenValue = (Integer) hashMap.get("bloodOxygenValue");//血氧
            }
            if (i == Constants.DATATYPE.Real_UploadComprehensive) {
                //腋下温度测量
                int tempInteger = (int) hashMap.get("tempInteger");
                int tempFloat = (int) hashMap.get("tempFloat");
                double temp = Double.parseDouble(tempInteger + "." + tempFloat);//体温
            }

         }
      }
});

/**
 * 获取单次体温数据，开启测量后，循环调用此方法获取体温
 */
private void readRealTemp() {
      YCBTClient.getRealTemp(new BleDataResponse() {
```

```
            @Override
            public void onDataResponse(int i, float v, HashMap hashMap) {
                if (i == 0) {
                    String temp = (String) hashMap.get("tempValue");
                }
            }
        });
}

YCBTClient.deviceToApp(new BleDeviceToAppDataResponse() {
    @Override
    public void onDataResponse(int i, HashMap hashMap) {
        if (i == 0 && hashMap != null) {
            switch ((int) hashMap.get("dataType")) {
                case Constants.DATATYPE.DeviceMeasurementResult:
                if (hashMap.get("datas") != null) {
                    byte[] data = (byte[]) hashMap.get("datas");
                    //data[0] 0x00:心率, 0x01:血压, 0x02:血氧, 0x03:呼吸率, 0x04:体温
                    //data[1] 0x00:用户退出测量, 0x01:测量成功, 0x02:测量失败
                    if (data[1] == 1) {
                        //测量结束
                    }else if (data[1] == 2) {
                        //测量失败
                    }else{
                        //测量取消
                    }
                }
                    break;
            }
        }
    }
}
```

12.9 健康参数、预警信息

发送预警信息后，设备达到指定条件后将发生震动。

```
/***
 *  健康参数、预警信息发送
 * @param warnState  预警状态  0x00:  无预警  0x01:  预警生效中
 * @param healthState  健康状态  0x00:未知  0x01:优秀  0x02:良好  0x03:一般  0x04:较差  0x05:生病
 * @param healthIndex  健康指数  0~120
 * @param friendWarn  亲友预警  0x00:无预警  0x01:预警生效  中
 * @param dataResponse
 */
public static void appHealthArg(int warnState, int healthState, int healthIndex, int friendWarn, BleDataResponse dataResponse)
YCBTClient.appHealthArg(1, 1, 60,1,dataResponse);
```

12.10 亲友消息

此方法是某个定制设备才支持

```
/**
 *  亲友消息
 *
 * @param index             表情包的下标  0-4
 * @param hour              时
 * @param min               分
 * @param name              亲友昵称
 * @param dataResponse 0x00:  手环显示震动成功  0x01:  手环未佩戴  0x02：其他错误
 */
public static void appEmoticonIndex(int index, int hour, int min, String name, BleDataResponse dataResponse)
YCBTClient.appEmoticonIndex(4, 9, 40, "俺是大宝 1",dataResponse);
```

12.11 数据回写

此部分只针对某些定制设备的特有功能使用，其它设备可以忽略。

12.11.1 健康值回写

```
/**
 *  健康值回写到手环
 *
 * @param healthValue    健康值
 * @param healthState    健康状态
 * @param dataResponse
 */
public static void appHealthWriteBack(int healthValue, String healthState, BleDataResponse dataResponse)
YCBTClient.appHealthWriteBack(1, 1, dataResponse);
```

12.11.2 睡眠数据回写

```
/**
 * APP  睡眠数据回写到手环
 *
 * @param deepSleepTimeHour    深睡(小时)
 * @param deepSleepTimeMin     深睡(分钟)
 * @param lightSleepTimeHour  浅睡(小时)
 * @param lightSleepTimeMin    浅睡(分钟)
 * @param totalSleepTimeHour  总睡眠(小时)
 * @param totalSleepTimeMin    总睡眠(分钟)
 * @param dataResponse            0x00:  手环接收成功  0x01:  设置失败-参数错误
 */
public static void appSleepWriteBack(int deepSleepTimeHour, int deepSleepTimeMin, int lightSleepTimeHour, int lightSleepTimeMin, int totalSleepTimeHour, int totalSleepTimeMin, BleDataResponse dataResponse)
YCBTClient.appSleepWriteBack(3, 40, 2, 20, 6, 0,dataResponse);
```

12.11.3 个人信息回写

```
/**
 * APP  用户个人信息回写到手环
 *
 * @param type              0x00:  用户保险类信息，其后内容为字符串，UTF8  编码  0x01:  用户会员状态信息，其后内容为字符串，UTF8  编码
 * @param content           信息内容
 * @param dataResponse 0x00:  用户保险类信息，其后内容为字符串，UTF8  编码  0x01:  用户会员状态信息，其后内容为字符串，UTF8  编码
```

```
 */
public static void appUserInfoWriteBack(int type, String content, BleDataResponse dataResponse)
YCBTClient.appUserInfoWriteBack(0, "content", dataResponse);
```

12.11.4 升级进度回写

```
/**
 * 升级进度回写
 *
 * @param on_off          0x00: 关闭提醒 0x01: 打开提醒
 * @param percent         百分比
 * @param dataResponse 0x00: 手环接收成功 0x01: 设置失败-参数错误
 */
public static void appUpgradeReminder(int on_off, int percent, BleDataResponse dataResponse)
YCBTClient.appUpgradeReminder(1, 50, dataResponse);
```

12.11.5 运动数据回写

```
/**
 * 运动数据回写
 *
 * @param step            有效步数
 * @param type            运动类型 0x00:休养静歇 0x01:休闲热身 0x02:心肺强化 0x03:减脂塑性 0x04:运动极限 0x05:空状态
 * @param dataResponse 0x00: 手环同步成功 0x01：手环同步失败
 */
public static void appEffectiveStep(int step, int type, BleDataResponse dataResponse)
YCBTClient.appEffectiveStep(10, 0, dataResponse);
```

12.11.6 计算心率同步

```
/**
 * APP 计算心率同步
 *
 * @param heart           有效心率
 * @param dataResponse 0x00: 手环同步成功 0x01：手环同步失败
 */
public static void appEffectiveHeart(int heart, BleDataResponse dataResponse)
YCBTClient.appEffectiveHeart(78, dataResponse);
```

12.11.7 测量数据回写

注意除血压值外，其它都是一个值，血压是将收缩压写在前面，舒张压写在后面，不管是几个值都必须是数组的形式传递。
如果传值有小数的话，则整部分在前，小数部分在后

```
/**
 * APP 测量数据回写到手环
 *
 * @param type            0x00:心率 0x01: 血压 0x02: 血氧 0x04: 呼吸率 0x05: HRV
 * @param value1          测量的类型返回值只 有 1byte 的，本字节填 测量值（例如心率、呼吸率等）， 返回两个字节的（如血压，本字节填收缩压）
 * @param value2          测量的类型返回值只 有 1byte 的，本字节默 认为 0（例如心率、呼吸率等）， 返回两个字节的（如血压，本字节填舒张压）
 * @param dataResponse 2bytes 0x00: 心率 0x01: 血压 0x02: 血氧 0x04: 呼吸率 0x05: HRV 0x00 成功 0x01 失败
 */
public static void appWritebackData(int type, int value1, int value2, BleDataResponse dataResponse)
```

```
YCBTClient.appWritebackData(1,135,94, dataResponse);
```

12.12 传感器数据存储开关控制

控制设备是否记录相关的数据，只针对某个特殊的设备有效。

```
/**
 * 传感器数据存储开关控制
 *
 * @param type              0x00: PPG 0x01: 加速度数据 0x02：ECG 0x03：温湿度 0x04：环境光 0x05：体温
 * @param on_off            0x00：关闭 0x01：打开
 * @param dataResponse
 */
public static void appSensorSwitchControl(int type, int on_off, BleDataResponse dataResponse)
YCBTClient.appSensorSwitchControl(0,1, dataResponse);
```

12.13 发送手机型号

```
/**
 * 发送当前手机型号
 *
 * @param model             手机型号
 * @param dataResponse 0x00: 推送成功 0x01: 推送失败-参数错误
 */
public static void appMobileModel(String model, BleDataResponse dataResponse)
YCBTClient.appMobileModel(”model“, dataResponse);
```

12.14 消息发送

发送消息有固定取值，只有 index 为 6 时，content 传值，其它情况一律传 null。

```
/**
 * APP 信息推送
 *
 * @param type              类型      0x00 有新的周报生成，请到 APP 上查看。
 *                          0x01 有新的月报生成，请到 APP 上查看。
 *                          0x02 收到亲友信息，请到 APP 上查看。
 *                          0x03 很久没测量了，测量一下吧。
 *                          0x04 您已成功预约咨询。
 *                          0x05 您预约的咨询，将在一小时后开始。
 *                          0x06 自定义内容
 * @param message           信息内容   类型为 0x06 时,才有信息内容
 * @param dataResponse 0x00: 手环同步成功 0x01：手环同步失败
 */
public static void appPushMessage(int type, String message, BleDataResponse dataResponse)
YCBTClient.appPushMessage(0,null, dataResponse);
```

12.15 环境温湿度校准 (保留)

校准相关的传感器，使测量更加精确

```
/**
 * 温湿度校准
 *
```

```
 * @param tempInt          温度整数部分(-127 - 127)
 * @param tempFloat        温度小数部分(0-9)
 * @param humidInt         湿度整数部分(0-99)
 * @param humidFloat       湿度小数部分(0-9)
 * @param dataResponse 0x00  成功    0x01  失败
 */
public static void appPushTempAndHumidCalibration(int tempInt, int tempFloat, int humidInt, int humidFloat, BleDataResponse dataResponse)
YCBTClient.appPushTempAndHumidCalibration(37,8,30,0, dataResponse);
```

12.16 血糖标定

血糖检定是用于校准血糖测量的数据，执行此方法后，测试血糖结果会更加准确。

```
/****
 *  血糖校准
 * @param bloodSugarInteger  血糖整数部分
 * @param bloodSugarFloat      血糖小数部分
 * @param type      0:空腹    1:早餐后      2:午餐前      3:午餐后    4:晚餐前      5:晚餐后
 * @param dataResponse 0x00:  校准成功  0x01:  校准失败-参数错误  0x02:  校准失败-设备未正在测试血压或者血压还没有出值
 */
public static void appBloodSugarCalibration(int bloodSugarInteger, int bloodSugarFloat, int type, BleDataResponse dataResponse)
YCBTClient.appBloodSugarCalibration(4,6,0, dataResponse);
```

12.17 发送测量值

注意除血压值外，其它都是一个值，血压是将收缩压写在前面，舒张压写在后面，不管是几个值都必须是数组的形式传递。
如果传值有小数的话，则整部分在前，小数部分在后。注意：本命令，小数部分取值范围为 0-9。

```
/****
 *  下发测量数据
 * @param type                类型    0x00:心率    0x01:血压    0x02:血氧    0x03:呼吸率    0x04:HRV    0x05:血糖    0x06:温度
 * @param timestamp          时间戳
 * @param valueInteger    值的整数部分      血压:收缩压 SBP
 * @param valueFloat        值的小数部分      默认为 0      血压:舒张压 DBP    温度:小数部分    血糖:小数部分
 * @param dataResponse    code 0x00:下发指令成功      0x01:下发指令失败
 *                                byte[] data = hashMap.get("datas")    data[0]表示型号和 type 对应    data[1]    0 表示下发成功    1 表示下发失败
 */
public static void appSendMeasureNumber(int type, long timestamp, int valueInteger, int valueFloat, BleDataResponse dataResponse)

/****
 *  下发测量数据
 * @param type                类型    0x00:心率    0x01:血压    0x02:血氧    0x03:呼吸率    0x04:HRV    0x05:血糖    0x06:温度
 * @param timestamp          时间戳
 * @param valueInteger    值的整数部分
 * @param valueFloat        值的小数部分      默认为 0      血压:舒张压 DBP    温度:小数部分    血糖:小数部分
 * @param maxInteger        最大值的整数部分
 * @param maxFloat          最大值的小数部分      默认为 0      血压:舒张压 DBP    温度:小数部分    血糖:小数部分
 * @param minInteger       最小值的整数部分
 * @param minFloat          最小值的小数部分      默认为 0      血压:舒张压 DBP    温度:小数部分    血糖:小数部分
 * @param dataResponse    code 0x00:下发指令成功      0x01:下发指令失败
 *                                byte[] data = hashMap.get("datas")    data[0]表示型号和 type 对应    data[1]    0 表示下发成功    1 表示下发失败
 */
public static void appSendMeasureNumber(int type, long timestamp, int valueInteger, int valueFloat, int maxInteger, int maxFloat, int minInteger, int minFloat, BleDataResponse dataResponse)
```

```
//心率 80
YCBTClient.appSendMeasureNumber(0,1670938122,80,0, dataResponse);

//血压 135/94
YCBTClient.appSendMeasureNumber(1,1670938122,135,94, dataResponse);

// 血糖 当前值 4.6, 最大值 5.3, 最小值 3.7
YCBTClient.appSendMeasureNumber(5,1670938122,4,6,5,3,3,7, dataResponse);
```

12.18 血脂标定

血脂标定是用于校准血脂测量的数据，执行此方法后，测试血脂结果会更加准确。

```
/**
 * 血脂校准，血脂标定
 *
 * @param model                          校准模式（预留）
 * @param totalCholesterolInteger  总胆固醇整数部分 2-10 单位：mmol/L
 * @param totalCholesterolFloat     总胆固醇小数部分 0-99（小数点保留两位） 单位：mmol/L
 * @param dataResponse                  0x00: 校准成功   0x01: 校准失败-参数错误
 */
public static void appLipidCalibration(int model, int totalCholesterolInteger, int totalCholesterolFloat, BleDataResponse dataResponse)
int value1 = (int)(5.55f * 100);
YCBTClient.appLipidCalibration(0, value1 / 100, value1 % 100,   bleDataResponse);
```

12.19 尿酸标定

尿酸标定是用于校准尿酸测量的数据，执行此方法后，测试尿酸结果会更加准确。

```
/**
 * 尿酸校准，尿酸标定
 *
 * @param model            校准模式（预留）
 * @param range            范围 179-1190 单位：μmol/L
 * @param dataResponse 0x00: 校准成功   0x01: 校准失败-参数错误
 */
public static void appUricAcidCalibration(int model, int range, BleDataResponse dataResponse)
 YCBTClient.appUricAcidCalibration(0, (int) umolParseFloat, bleDataResponse);
```

13 接收设备响应

这部分指的是用户操作了设备或者设备监测到了某种信息，同时设备会将操作信息上报。SDK 将统一监听设备的响应，依据不同的类型进行处理。SDK 会以通知的形式发送设备的操作状态。应用程序需要进行监听并依据不同的类型进行解析。此外，有些内容在其它地方已经列举，这里将不再出现。如果没有接收到对应的数据，可能是设备不支持此功能。

```
/***
 * 监听设备发送数据到 App
 */
public static void deviceToApp(BleDeviceToAppDataResponse bleRealTypeResponse)
YCBTClient.deviceToApp(new BleDeviceToAppDataResponse() {
   @Override
   public void onDataResponse(int i, HashMap hashMap) {
         if (hashMap != null) {
               if (i == 0) {//
```

```
int dataType = (int) hashMap.get("dataType");
int data = -1;
if (hashMap.get("data") != null)
    data = (int) hashMap.get("data");
switch (dataType) {
    case Constants.DATATYPE.AppECGPPGStatus://设备实时状态上传
        int EcgStatus = (int) hashMap.get("EcgStatus");
        int PPGStatus = (int) hashMap.get("PPGStatus");
        if (PPGStatus == 0) {//0 :wear    1: no wear 2: error
        }
        break;
    case Constants.DATATYPE.DeviceFindMobile://寻找手机
        if (data == 0) {//0x00:  结束
            handler.removeCallbacks(runnable);
            SoundPoolUtil.getInstance(MainActivity.this).stop();
        } else if (data == 1) {//0x01:  开始
            SoundPoolUtil.getInstance(MainActivity.this).play(100);
            handler.removeCallbacks(runnable);
            handler.postDelayed(runnable, 15000);
        }
        break;
    case Constants.DATATYPE.DeviceLostReminder://防丢提醒
        if (data == 0) {//0:  结束  1:  开始
        }
        break;
    case Constants.DATATYPE.DeviceAnswerAndClosePhone://接听/拒接电话
        if (data == 0) {//0:  接听  1:  拒接
            answerCall();
        } else {
            rejectCall();
        }
        break;
case Constants.DATATYPE.DeviceTakePhoto://相机拍照控制
    if (PermissionUtil.openCameraPermission(context) && PermissionUtil.openSDCardPermission(context)) {
        if (data == 1) {//0x00:退出拍照模式  0x01:进入拍照模式  0x02:拍照
            startActivity(new Intent(MainActivity.this, CameraActivity.class));
        } else {
            EventBus.getDefault().post(new EventBusTakePhotoEvent(data));
        }
    }
    break;
case Constants.DATATYPE.DeviceStartMusic://音乐控制
    HealthApplication.getInstance().musicCon(data);
    break;
case Constants.DATATYPE.DeviceSos://开启一键呼救控制命令
    break;
case Constants.DATATYPE.DeviceDrinkingPatterns://开启饮酒模式控制命令
    break;
case Constants.DATATYPE.DeviceSportModeControl://手环运动模式控制
    if (hashMap.get("datas") != null) {
        if (((byte[]) hashMap.get("datas")).length >= 2 && ((byte[]) hashMap.get("datas"))[0] == 1) {
            Intent intent = new Intent(context, SportRunningActivity.class);
            intent.putExtra("Title", ((byte[]) hashMap.get("datas"))[1] & 0xff);
```

```
                                    startActivity(intent);
                                } else {
                                    EventBus.getDefault().post(new EventBusExitExerciseEvent((byte[]) hashMap.get("datas")));
                                }
                            }
                            break;
                    }
                }
            }
    }
});

public class DATATYPE{
    public static final int DeviceFindMobile = 0x0400;//寻找手机
    public static final int DeviceLostReminder = 0x0401;//防丢提醒
    public static final int DeviceAnswerAndClosePhone = 0x0402;//接听/拒接电话
    public static final int DeviceTakePhoto = 0x0403;//相机拍照控制
    public static final int DeviceStartMusic = 0x0404;//音乐控制
    public static final int DeviceSos = 0x0405;//一键呼救控制命令
    public static final int DeviceDrinkingPatterns = 0x0406;//饮酒模式控制命令
    public static final int DeviceConnectOrDisconnect = 0x0407;//手环蓝牙　连接/拒连
    public static final int DeviceSportMode = 0x0408;//手环运动模式控制命令
    public static final int DeviceSyncContacts = 0x0409;//请求同步手机通讯录
    public static final int DeviceRest = 0x040A;//通知手机当前手环恢复出厂设置
    public static final int DeviceEndECG = 0x040B;//通知 APP 手环已经结束实时 ECG 测试
    public static final int DeviceSportModeControl = 0x040C;//手环运动模式控制
    public static final int DeviceSwitchDial = 0x040D;//手环通知 APP 切换表盘
    public static final int DeviceMeasurementResult = 0x040E;//手环返回 APP 启动单次测量结果
    public static final int DeviceAlarmData = 0x040F;//手环上报预警数据
    public static final int DeviceInflatedBloodMeasureResult = 0x0410;//手环通知 APP 精准血压测量结果
    public static final int DeviceUpgradeResult = 0x0411;//手环返回设备升级结果
    public static final int DevicePPIData = 0x0412;//PPI 数据
    public static final int DeviceMeasurStatusAndResults = 0x0413;//手表返回有创测量状态及结果
}
```

14 定制相关

14.1 获取 CGM 设备信息

适用：H01,此方法可以获取到设备的启动时间和序列号，其它类型暂不支持。

```
/**
 * 获取 CGM 的启动时间及发射器序列号
 * 例：YCBTClient.customizeCGM(new BleDataResponse())
 *
 * @param dataResponse code: 0x00:成功,0x01:失败
 *                     serial: 发射器序列号,如：(String) hashMap.get("serial")
 *                     startTime: 启动时间时间戳,如：(String) hashMap.get("startTime")
 */
public static void getCustomizeCGM(BleDataResponse dataResponse)
YCBTClient.getCustomizeCGM(dataResponse);
```

14.2 查询与删除设备历史数据

适用：H01

14.2.1 查询设备数据

此方法通过不同的类型来获取不同的历史数据。

```
/**
 * 定制项目数据同步：1:注册实时数据监听，YCBTClient.appRegisterRealDataCallBack();2:开始数据同步，YCBTClient.startCustomizeDataSync()
 *
 *
 * @param type              类型：  0x01:CGM  血糖,0x02:理疗
 * @param dataResponse code:0x00:成功,0x01:失败
 *                              data:[{"offset":0,"cgm":2,"ele":0},{"offset":0,"cgm":3,"ele":0},...{"offset":0,"cgm":3,"ele":0}]
 */
public static void startCustomizeDataSync(int type, BleDataResponse dataResponse)
 YCBTClient.startCustomizeDataSync(1, dataResponse);
```

14.2.2 删除设备数据

此方法可以通过指定的类型来删除对应的数据。

```
/**
 * 删除定制项目数据
 *
 * @param type              类型：  0x01:CGM  血糖，0x02:理疗
 * @param key               删除 key 0x00:删除当天数据,0x01:删除历史数据,0x02:删除所有数据，0x03:删除原始数据
 * @param dataResponse code:0x00:成功,0x01:失败-删除失败,0x02:无记录数据,0x03:  失败-参数错误
 *                              state:  是否成功,如：(int) hashMap.get("state")
 *                              type:  类型,如：(String) hashMap.get("type")
 */
public static void deleteCustomizeData(int type, int key, BleDataResponse dataResponse)
YCBTClient.deleteCustomizeData(0x01, 0x02,dataResponse);
```

15 名片

15.1 下发名片

下发名片到设备

```
/**
   * 下发名片
   * @param type          类型   0x00:微信   0x01:QQ   0x02:Facebook   0x03:Twitter   0x04:Whatsapp   0x05:Instgram 0xFO:SN 码  0xF1:静态码  0xF2:动态码
   * @param sqrString     二维码的 UTF-8 的字节码  以空字节结尾
   * @param dataResponse 0x00:接收成功   0x01:接收失败-参数错误
   */
public static void appSendCardNumber(int type, String sqrString, BleDataResponse dataResponse)
YCBTClient.appSendCardNumber(0xF0, "5678",dataResponse);
```

15.2 获取名片信息

获取名片信息，0x00:微信，0x01:QQ ， 0x02:Facebook ， 0x03:Twitter ， 0x04:Whatsapp，0x05:Instgram，0xFO:SN 码，0xF1:静态码，0xF2:动态码

```
/**
 *  获取名片信息
 * @param type        类型   0x00:微信   0x01:QQ   0x02:Facebook   0x03:Twitter   0x04:Whatsapp   0x05:Instgram 0xFO:SN 码  0xF1:静态码  0xF2:动态码
 * @param dataResponse
 */
public static void getCardInfo(int type, BleDataResponse dataResponse)
YCBTClient.getCardInfo(0xF0,dataResponse);
```

15.3 监听动态码

```
YCBTClient.deviceToApp(new BleDeviceToAppDataResponse() {
                @Override
                public void onDataResponse(int i, HashMap hashMap) {
                    if (i == 0&&hashMap != null) {//
                            Log.i(TAG,new Gson().toJson(dataMap))
                    }
                }
});
```

16 智趣

16.1 智趣开启和关闭

```
/**
 *  智趣  打开与关闭
 *  例：YCBTClient.setWitOnOff(1,1,new BleDataResponse())
 *
 * @param onOff             开关  0x00 关闭，0x01 打开
 * @param type              0x01:短视频,0x02:音乐,0x03:阅读,0x04:拍照/录像,0x05:SOS,0x06:幻灯片
 * @param dataResponse code: 0x00:成功,0x01:失败
 */
public static void setWitOnOff(int onOff, int type, BleDataResponse dataResponse)
YCBTClient.setWitOnOff(isOpen ? 1 : 0, protocolIndex, bleDataResponse);
```