ChipletRing APP SDK 说明书

IOS

文档版本号：	1.0.4	文档编号：	
文档密级：		归属部门：	Chiplet 研发部
产品名：	BCL_603M	归属项目：	
编写人：	郑玉虎	编写日期：	2023.11.2

修订记录：

版本号	修订人	修订日期	修订描述
1.0.0	郑玉虎	2023.11.3	修改错别字，更新目录
1.0.2	郑玉虎	2023.11.23	汇总已解决的问题
1.0.3	郑玉虎	2023.11.29	新增问答 Q&A
1.0.4	郑玉虎	2023.12.20	新增故障排除

目 录

一、 文档简介 6

1、 文档目的 6

2、 适用范围 6

3、 简介说明 6

4、 功能介绍 6

二、 快速使用 7

1、 使用 CocoaPods 7

2、 导入 SDK 7

三、 API 功能说明 8

1、 设备连接相关 8

1.1 搜索蓝牙 8

1.2 停止搜索 8

1.3 连接指定蓝牙 8

1.4 断开蓝牙 9

1.5 当前连接蓝牙 9

1.6 蓝牙连接状态 9

1.7 监听连接状态 9

2、 指令功能 10

2.1 同步时间 10

2.2 读取时间 10

2.3 读取软件版本号 11

2.4 读取硬件版本号 11

2.5 读取电池电量 12

2.6 读取电池充电状态 12

2.7 实时测量心率值 13

2.8 心率变异性 13

2.9 实时测量血氧值 14

2.10 读取温度 15
2.11 读取当天实时步数 15
2.12 清除实时步数 15
2.13 读取本地数据 16
2.14 删除数据 18
2.15 设置采集周期 18
2.16 读取采集周期 19
2.17 恢复出厂设置 19
3、 固件升级（OTA） 20
4、 数据库 RingDBManager 20
4.1 获取指定时间数据 21
4.2 获取某天数据 21
4.3 获取最近一条数据 21
4.4 获取某天睡眠数据 21
4.5 删除所有数据 22
4.6 删除指定时间到现在的数据 22
5、 逻辑算法 22
5.1 计算步行距离 22
5.2 睡眠时间计算 23
5.3 获取睡眠时长 23
6、 日志配置 23
四、 其他 24
1、 可能会遇到的问题 24
1.1 使用 SDK 出错 24
1.2 读取软件版本时，会多返回两个空格 24
1.3 调用接口，返回超时 24
1.4 戒指的睡眠功能如何测量 24
1.5 睡眠数据读取含有前几日数据 24
1.6 云端 OTA 更新 25

1.7 数据库无数据或数据库始终只有 n 条数据 25
2、 补充资料 25
2.1 睡眠数据 25
2.1.1 遵循协议 25
2.1.2 数据解析 25
3、 Q&A 答疑 26
3.1 关于戒指 26
3.2 SDK 使用可能产生的疑问 26
3.3 故障排除 29
3.3.1 使用 SDK 搜索附近蓝牙设备失败 29
3.3.2 压缩包解压失败 29

一、 文档简介

1、 文档目的

为方便 IOS 端 APP 与戒指通讯进行二次开发 ，特对通讯协议进行封装 ，以达到简洁明了 ，让开发者不需要关注与戒指通讯层 ，专注业务逻辑交互层面开发。

2、 适用范围

支持 iOS 版本：iOS 13.0+

支持的语言版本：Swift 5.0+

开发工具: xcode 15+

3、 简介说明

这是智能戒指的蓝牙通信 SDK，主要提供与戒指通信相关的 API 管理类 RingManager，以及相关的数据存储管理类 RingDBManager，同时也提供部分复杂算法的实现，例如睡眠相关算法。

注：由于 sdk 需要用到蓝牙，仅支持真机调试，且注意在你项目的 info 中添加蓝牙权限说明

4、 功能介绍

功能模块	说明	相关文档
蓝牙基础模块	蓝牙的开关操作 蓝牙的搜索链接操作 蓝牙的数据写入监听操作	

通讯协议模块	时间管理 版本号管理 电池管理 心率测量 血氧测量 温度测量 计步管理 历史记录管理 系统设置 日志管理	

二、 快速使用

1、 使用 CocoaPods

首先在你的项目的 Podfile 文件中新增一个 Spec 源，即将下面的语句复制粘贴到项目中的 Podfile 最上方：

```
1. source 'https://github.com/wcb133/RingSpec.git'
```

2、 导入 SDK

在 pod 中导入该 SDK：

```
1. pod 'Rings-SDK'
```

然后安装：

```
1. pod install
```

在使用的地方引入 SDK 即可：

```
1. import RingsSDK
```

三、 API 功能说明

1、 设备连接相关

1.1 搜索蓝牙

接口说明： 搜索附近的蓝牙设备，开始搜索后可通过回调获取搜索到的设备列表，也可通过 RingManager.shared.devices 属性获取当前搜索到的设备

接口声明：

```
1. RingManager.shared.startScan { devices in
2.     print("搜索到的设备列表 ==========>\(String(describing: devices))")
3. }
```

注意事项：无

1.2 停止搜索

接口说明：停止搜索附近的蓝牙设备，停止搜索不会清空已搜索到的设备列表，即不会清空 RingManager.shared.devices

接口声明：

```
1. RingManager.shared.stopScan()
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

1.3 连接指定蓝牙

接口说明：通过设备的 uuid 来连接指定设备，uuid 由搜索到的设备模型 DeviceInfo 中获取

接口声明：

```
1. RingManager.shared.startConnect(deviceUUID: "uuidString", resultBlock: { res in
2.     switch res {
3.     case .success(let deviceInfo):
4.         print("已连接设备 ==========\(String(describing: deviceInfo.peripheralName))")
5.     case .failure(let error):
```

```
            print("连接失败 =========> \(error)")
        }
    })

```

注意事项：无

1.4 断开蓝牙

接口说明：断开当前设备的连接

接口声明：

```
RingManager.shared.disconnect()
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

1.5 当前连接蓝牙

接口说明：获取当前已连接的设备

接口声明：

```
let currentDevice = RingManager.shared.currentDevice
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

1.6 蓝牙连接状态

接口说明：获取设备连接状态

接口声明：

```
let isDidConnect = RingManager.shared.isDidConnect
```

注意事项：无

1.7 监听连接状态

接口说明：设备连接状态变化监听

接口声明：

```
RingManager.shared.connectStateChangeBlock = { isConnected in
        print("是否已连接 =========\(isConnected)")

    }
```

注意事项：无

2、 指令功能

2.1 同步时间

接口说明：

接口声明：

```
RingManager.shared.syncTime(date: Date()) { res in
        switch res {
        case .success(let isSuccess):
            print("同步时间结果======\(isSuccess)")
        case .failure(let error):
            print("同步失败======\(error)")
        }
    }
```

注意事项：调用此接口 ， 需保证与戒指处于连接状态

2.2 读取时间

接口说明：

接口声明：

```
RingManager.shared.readTime { res in
        switch res {
        case .success(let value):
            print("成功======>\(value)毫秒")
        case .failure(let error):
```

```
            print("失败=====>\(error)")
        }
    }
```

注意事项：调用此接口 ， 需保证与戒指处于连接状态

2.3 读取软件版本号

接口说明：

接口声明：

```
RingManager.shared.readAppVersion { res in
        switch res {
        case .success(let version):
            print("软件版本号=====>\(version)")
        case .failure(let failure):
            print("失败=====>\(failure)")
        }
    }
```

注意事项：调用此接口 ， 需保证与戒指处于连接状态

2.4 读取硬件版本号

接口说明：

接口声明：

```
RingManager.shared.readHardWareVersion { res in
        switch res {
        case .success(let version):
            print("硬件版本号=====>\(version)")
        case .failure(let failure):
            print("失败=====>\(failure)")
```

```
            }
        }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

2.5 读取电池电量

接口说明： 若返回的电池电量为 101，则表示正在充电中

接口声明：

```
RingManager.shared.readBattery { res in
            switch res {
            case .success(let value):
                print("电量======>\(value)")
            case .failure(let failure):
                print("失败======>\(failure)")
            }
        }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

2.6 读取电池充电状态

接口说明： 读取电池充电状态，返回结果为 ChargeStatus 枚举类型，值有:

- full：充满
- charging：充电中
- normal：正常未充电状态

接口声明：

```
RingManager.shared.readChargeStatus { res in
        switch res {
        case .success(let state):
            print("状态======>\(state)")
        case .failure(let error):
            print("失败======>\(error)")
        }
    }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

2.7 实时测量心率值

接口说明： 输出实时测量心率值，单位 BPM

接口声明：

```
RingManager.shared.readHeartRate(progressBlock: { progress in
        print(" 测量进度 ======>\(progress)")
    }) { res in
        switch res {
        case .success(let value):
            print("心率======>\(value)")
        case .failure(let error):
            print("失败======>\(error)")
        }
    }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

2.8 心率变异性

接口说明： 输出心率变异性，单位毫秒(ms)

接口声明：

```
RingManager.shared.readHRV(progressBlock: { progress in
            print(" 测量进度 ======>\(progress)")
        }) { res in
            switch res {
            case .success(let value):
                print("心率变异性======>\(value)")
            case .failure(let error):
                print("失败======>\(error)")
            }
        }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

2.9 实时测量血氧值

接口说明： 输出实时血氧值

接口声明：

```
RingManager.shared.readO2(progressBlock: { progress in
            print(" 测量进度 ======>\(progress)")
        }) { res in
            switch res {
            case .success(let value):
                print("血氧值======>\(value)")
            case .failure(let error):
                print("失败======>\(error)")
            }
        }

```

注意事项：调用此接口 ，需保证与戒指处于连接状态

2.10 读取温度

接口说明： 读取当前温度，返回结果单位为摄氏度(℃)

接口声明：

```
RingManager.shared.readTemperature { res in
        switch res {
        case .success(let value):
            BDLogger.info("温度=====>\(value)")
        case .failure(let error):
            BDLogger.info("失败=====>\(error)")
        }
    }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

2.11 读取当天实时步数

接口说明：

接口声明：

```
RingManager.shared.readSteps { res in
        switch res {
        case .success(let value):
            print("步数=====>\(value)步")
        case .failure(let error):
            print("失败=====>\(error)")
        }
    }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

2.12 清除实时步数

接口说明：

接口声明：

```
RingManager.shared.clearSteps { res in
        switch res {
        case .success(let isSuccess):
            print("结果=====>\(isSuccess)")
        case .failure(let error):
            print("失败=====>\(error)")
        }
    }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

2.13 读取本地数据

接口说明： 读取本地历史数据，并且内部会将每一条数据存入到数据库中。关于如何从数据库中获取数据，见数据库管理类 RingDBManager 部分说明

接口声明：

```
RingManager.shared.readDatas { progress, dataModel in
        print(" 进度 =====>\(progress)==\(dataModel)")
    } resultBlock: { res in
        switch res {
        case .success(let state):
            print("结果=====>\(state)")
        case .failure(let error):
            print("失败=====>\(error)")
        }
    }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

该 API 获取设备历史数据，每获取到一条历史信息，都会通过 progressBlock 闭包回调，并在获取全部结果之后，通过 resultBlock 回调最终结果。其中 dataModel 为设备数据模型，相关属性如下：

```
public final class RingDataModel: NSObject,TableCodable {
```

```
    // 总个数
    public var total:UInt32 = 0
    // 序号，1 开始
    public var serialNum:UInt32 = 0
    // 时间戳,秒
    public var timestamp:UInt32 = 0
    // 当天截止当前累计步数
    public var stepsOfTheDay:UInt16 = 0
    // 心率,0 无效
    public var rate = 0
    // 血氧,0 无效
    public var O2 = 0
    // 心率变异性,0 无效
    public var hrv = 0
    // 精神压力指数,0 无效
    public var mentalStress = 0
    // 温度，有符号的
    public var temp:Float = 0
    // 运动激烈程度 0: 静止 0x65:运动
    public var isRunning = false
    // 睡眠类型 0：无效 1：清醒 2：浅睡 3：深睡 4.眼动期
    public var sleepType = 0
    // RR 期间个数
    public var RRNums = 0
    // RR 数组
    public var rrs:[Int] = []
}
```

返回值：state 为 ReadDataResult 枚举类型，值有:

```
public enum ReadDataResult {
```

```
2.      case empty // 无数据
3.      case complete // 完成
4.  }
```

2.14 删除数据

接口说明： 删除全部本地数据历史记录

接口声明：

```
1.  RingManager.shared.clearRingData { res in
2.              switch res {
3.              case .success(let isSuccess):
4.                  print("结果======>\(isSuccess)")
5.              case .failure(let failure):
6.                  print("失败======>\(failure)")
7.              }
8.          }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

2.15 设置采集周期

接口说明： 采集周期设置，采集周期设置单位为秒(s)

接口声明：

```
1.  RingManager.shared.setFrequency(time: time) { res in
2.              switch res {
3.              case .success(let isSuccess):
4.                  print("结果======>\(isSuccess)")
5.              case .failure(let failure):
6.                  print("失败======>\(failure)")
7.              }
8.          }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

2.16 读取采集周期

接口说明： 采集周期读取，读取到的采集周期为秒(s)

接口声明：

```
RingManager.shared.readFrequency { res in
        switch res {
        case .success(let value):
            print("成功=====>\(value)秒")
        case .failure(let error):
            print("失败=====>\(error)")
        }
    }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

2.17 恢复出厂设置

接口说明：

接口声明：

```
RingManager.shared.reset { res in
        switch res {
        case .success(let isSuccess):
            print("成功=====>\(isSuccess)")
        case .failure(let error):
            print("失败=====>\(error)")
        }
    }
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

3、 固件升级（OTA）

接口说明： 固件升级，参数 fileUrl 为固件文件所在的本地路径，升级进度以及结果通过 handle 进行回调

接口声明：

```
if let path = Bundle.main.path(forResource: "otafileName", ofType: nil) {
    let url = URL(fileURLWithPath: path)
    RingManager.shared.startOTA(url) { res in
        switch res {
        case .start:
            print("开始升级 =====>")
        case .progress(let value):
            print("升级进度 =====>\(value)")
        case .success:
            print("升级成功 =====>")
        case .fail(let errorString):
            print("升级失败 =====>\(errorString)")
        }
    }
}
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

4、 数据库 RingDBManager

使用 func readDatas(progressBlock: @escaping (Double, RingDataModel)->Void, resultBlock: @escaping (Result<ReadDataResult, ReadError>)->Void)从设备中获取到的本地历史数据，都会被保存到本地数据库中，访问数据库需要使用 RingDBManager 类，内部提供了一个单例对象用来操作相关数据，单例对象可通过 RingDBManager.shared 来获取，以下是相关的 API 说明。

4.1 获取指定时间数据

接口说明： 从数据库中获取指定时间到目前为止的所有历史数据，timestamp 参数为 10 位的时间戳

接口声明：

```
1.  public func getObjects(from timestamp:TimeInterval) -> [RingDataModel]
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

4.2 获取某天数据

接口说明： 从数据库中获取某一天的历史数据（获取到的是该日期当天 0 时到 24 时的数据）

接口声明：

```
1.  public func getObjects(of date:Date) -> [RingDataModel]
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

4.3 获取最近一条数据

接口说明： 从数据库中获取距离当前时间最近的一条历史数据

接口声明：

```
1.  public func getLatestObject() -> RingDataModel?
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

4.4 获取某天睡眠数据

接口说明： 从数据库中获取某一天的睡眠数据(获取到的是该日期前一天 18 时到该日期 18 时的数据)，该部分接口得到的数据主要用于睡眠的计算

接口声明：

```
1.  public func getSleepObjects(of date:Date) -> [RingDataModel]
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

4.5 删除所有数据

接口说明： 删除本地数据库所有历史数据

接口声明：

```
1.  public func deleteAll() -> Bool
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

4.6 删除指定时间到现在的数据

接口说明： 删除从指定时间到目前为止的所有历史数据，timestamp 参数为 10 位的时间戳

接口声明：

```
1.  public func deleteAllBeforeTimestamp(timestamp:TimeInterval) -> Bool
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

5、 逻辑算法

该部分代码在 RingManager 类中，使用 RingManager.shared 获取单例，然后调用相关 API 即可

5.1 计算步行距离

接口说明： 步行距离计算，steps 参数为步数，stepSize 参数为步长，单位为厘米(cm)，返回结果为距离，单位为米(m)

接口声明：

```
1.  func calculateDistance(steps:Int,stepSize:Int) -> Float
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

5.2 睡眠时间计算

接口说明： 睡眠时间计算，获取指定日期的睡眠数据及零星睡眠数据。返回值是一个元组，元组的第一个元素($0.0)是睡眠数据集合，第二个元素($0.1)是一个二维数组，是多个零星睡眠段的集合。$0.0 数组中的第一个数据点的时间为入睡时间，最后一个数据点的时间为醒来时间，中间的各个数据点时间差累加即为睡眠时间。零星睡眠时长计算同理，使用$0.1 数组中的数据分段计算即可。

接口声明：

```
1.  func caculateSleepData(targetDate: Date) -> ([RingDataModel], [[RingDataModel]])
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

5.3 获取睡眠时长

接口说明： 获取睡眠时长，传入睡眠时间数据点集合，即可得出睡眠时长，返回睡眠时长单位为分钟(min)

接口声明：

```
1.  func calculateSleepTimes(sleepDatas:[RingDataModel]) -> Int
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

6、 日志配置

接口说明： 设置日志保存路径，默认保存在沙盒的 Document 中。可通过以下 API 修改默认保存路径

接口声明：

```
1.  func configLogPath(directoryPath: String = defaultLogDirectoryPath)
```

注意事项：调用此接口 ，需保证与戒指处于连接状态

四、 其他

1、 可能会遇到的问题

1.1 使用 SDK 出错

将开发工具升级至适用范围

支持 iOS 版本：iOS 13.0+

支持的语言版本：Swift 5.0+

开发工具: xcode 15+

1.2 读取软件版本时，会多返回两个空格

版本号 10 个字节，空格补齐。

补充：固件版本一般是数字，末尾有可能加上字母 S

1.3 调用接口，返回超时

发送了一个指令，过了 3 秒，戒指没有响应，会返回超时

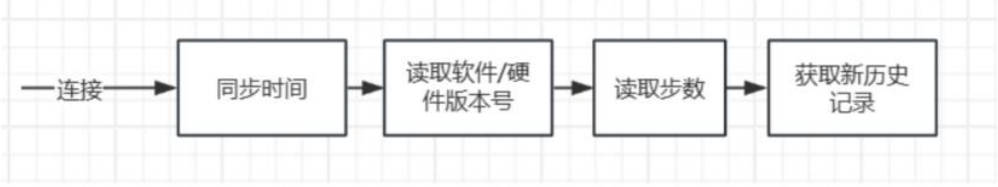
超时问题要具体分析

1.4 戒指的睡眠功能如何测量

戴着戒指睡觉即可。如果睡眠过程中，APP 读取 readDatas 接口，也能获取到当前的睡眠数据。

1.5 睡眠数据读取含有前几日数据

在绑定后，每次打开 APP 自动以最快的速度连接戒指。连接流程如下：



可能因为时间戳未同步

1.6 云端 OTA 更新

本 SDK 提供路径更新方式，OTA 固件包放云端，下载后将固件包的路径给 SDK 即可

注：OTA 升级新版本会清除数据

1.7 数据库无数据或数据库始终只有 n 条数据

问题：调用了 readData，但是在 db 里获取不到数据

具体：RingManager.shared.readHeartRate 有回调， 之后 readDatas， getObjects 数据库没数据

解决：把接入 SDK 的 APP 删除，重新安装（原因在于数据读取判定为非首次读取）

2、 补充资料

2.1 睡眠数据

提供睡眠用例测试（联系我们，以压缩包的形式发送），包含唤醒、基准、入睡、睡眠中醒来、组合测试，每项下都有.dat 文件即睡眠数据，数据分析方法如下

2.1.1 遵循协议

1 变非 1 是入睡，再变 1 是清醒

2.1.2 数据解析

睡眠类型：

0：无效

1：清醒

2：浅睡

3：深睡

4：眼动期

```
[ 180:  12]2023.10.18_21.00.00 step:    0 hr: 70 spo2:  0 hrv:  0 sprit:0 temp:   0 sport:0 sleep:1 sleep_time:    0 rr_num: 0
[ 180:  13]2023.10.18_21.05.00 step:    0 hr: 70 spo2:  0 hrv:  0 sprit:0 temp:   0 sport:0 sleep:2 sleep_time:    0 rr_num: 0
[ 180:  14]2023.10.18_21.10.00 step:    0 hr: 70 spo2:  0 hrv:  0 sprit:0 temp:   0 sport:0 sleep:2 sleep_time:    0 rr_num: 0
[ 180:  15]2023.10.18_21.15.00 step:    0 hr: 70 spo2:  0 hrv:  0 sprit:0 temp:   0 sport:0 sleep:2 sleep_time:    0 rr_num: 0
[ 180:  16]2023.10.18_21.20.00 step:    0 hr: 70 spo2:  0 hrv:  0 sprit:0 temp:   0 sport:0 sleep:2 sleep_time:    0 rr_num: 0
```

3、 Q&A 答疑

3.1 关于戒指

Q：手动测量的数据会保存到数据库么？心率血氧这些

A：会

Q：默认采集周期是多久

A：1200s/20min

Q：戒指的数据是如何产生的，产生的数据结构是怎么样的？

A：戒指每隔一定时间（默认是 5 分钟）就会进行一次自动测试，测试的内容包括心率、血氧、心率变异性、睡眠状态等，从而产生一个采样数据点，一个数据点对应一个 RingDataModel 模型对象，该采样数据会存储在戒指当中，随着时间的推移，戒指中存储的数据点会越来越多，但是最多只会保存最近 7 天的数据。

Q:使用 ChipletRing 测试时日志在哪

A:在 APP 页面上找到我的→日志

3.2 SDK 使用可能产生的疑问

Q：如何使用 SDK 从戒指中读取数据？

A：连接上戒指后，戒指中的数据不会主动上报，需要自己调用相应的 SDK 方法去获取，可使用以下方法去读取戒指中的数据

```
1.  func readDatas(progressBlock: @escaping (Double, RingDataModel)->Void, resultBlock
: @escaping (Result<ReadDataResult, ReadError>)->Void)
```

注：首次调用该方法，将会读取戒指中存储的全部历史数据点，成功获取全部历史数据之后，再调用该方法，只会读取戒指中未被读取过的新的数据点，若戒指无新数据产生或者戒指中数据已被读取过，调用该方法获取到的数据将会为空。该方法内部会将获取到的所有数据点都存入到 SDK 内置的本地数据库中。

另：SDK 如何知道数据有没有读取过呢？

在偏好设置里，本地有保存记录有没有读取过

Q：如何从 SDK 内置的数据库中获取指定日期的数据？

A：当我们将戒指中的数据读取上来之后，可使用 RingDBManager 管理类获取相关数据，可使用 getObjects(of date:Date)获取指定日期数据。如获取今天的数据

```
1.  // 从数据库中获取今日数据
2.  let date = Date()
3.  let datasOfToday = RingDBManager.shared.getObjects(of: date)
```

Q：使用该 SDK 开发 App，获取数据的一般流程

A：在连接设备之后，使用以下方法读取戒指中的数据

```
1.  func readDatas(progressBlock: @escaping (Double, RingDataModel)->Void, resultBlock: @escaping (Result<ReadDataResult, ReadError>)->Void)
```

获取成功之后，再调用以下方法从数据库中获取指定日期的数据，从而得到最新数据。

```
1.  func caculateSleepData(targetDate: Date) -> ([RingDataModel], [[RingDataModel]])
```

返回值是一个元组，元组的第一个元素($0.0)是睡眠数据集合，其中第一个数据点即为睡眠入睡点的数据，最后一个数据点即为睡眠醒来点的数据。 元组的第二个元素($0.1)是一个二维数组，是多个零星睡眠段的集合。其中的每个数组的含义与上面所述相同。

Q：获取到睡眠数据之后如何获得入睡时间、醒来时间、睡眠时间，如何统计清醒时长、浅睡时长、深睡时长、眼动期时长？

A：如下例子所示

```
1.      // 获取昨晚的睡眠数据
2.      let date = Date()
3.      let allDatasOfSleeps = RingManager.shared.caculateSleepData(targetDate: date)
4.      let datas = allDatasOfSleeps.0
5.      print("入睡时间 =======>\(String(describing: datas.first?.timestamp))")
6.      print("醒来时间 =======>\(String(describing: datas.last?.timestamp))")
7.
```

```
let sleepTimes = RingManager.shared.calculateSleepTimes(sleepDatas: datas)
print("睡眠时间 =======>\(sleepTimes)分钟")


var lastModel:RingDataModel?
// 清醒时长，单位秒
var wakeTimes:UInt32 = 0
// 浅睡时长，单位秒
var lightSleepTimes:UInt32 = 0
// 深睡时长，单位秒
var deepSLeepTimes:UInt32 = 0
// 眼动期时长，单位秒
var eyesTimes:UInt32 = 0
datas.forEach { model in
    if let tempLastModel = lastModel {
        switch model.sleepType {
        case 1:// 清醒
            wakeTimes += model.timestamp - tempLastModel.timestamp
        case 2:// 浅睡
            lightSleepTimes += model.timestamp - tempLastModel.timestamp
        case 3:// 深睡
            deepSLeepTimes += model.timestamp - tempLastModel.timestamp
        case 4:// 眼动
            eyesTimes += model.timestamp - tempLastModel.timestamp
        default :
            break
        }
        lastModel = model
    }else{
        lastModel = model
```

```
        }
    }

    print("清醒时长 =======>\(wakeTimes)")
    print("浅睡时长 =======>\(lightSleepTimes)")
    print("深睡时长 =======>\(deepSLeepTimes)")
    print("眼动时长 =======>\(eyesTimes)")
```

3.3 故障排除

3.3.1 使用 SDK 搜索附近蓝牙设备失败

1. iOS demo 运行 RingManager.shared.startScan 方法一直报错 Rings-SDK_Example[44209:4371892] [CoreBluetooth] API MISUSE: <CBCentralManager: 0x2816cb680> can only accept this command while in the powered on state

Situation:使用 SDK 过程中，调用 RingManager.shared.startScan 程序没有进行下一步，卡在了这里，调试里显示以上 message

排除方法：使用 ChipletRingAPP 测试后发现，可以发现蓝牙，但是连接不上：rssi：-74， 连接失败

找到原因：戒指没电了/电量太低，充电后问题解决。

注：RingManager.shared.startScan 调用后不用在意 message，有搜索到设备就会自动连接。

3.3.2 压缩包解压失败

这里提供 git 网址：https://github.com/wcb133/Rings-SDK。可自行下载