

**SDK需求说明书**

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

**北京勇芯科技有限公司 版权所有**

**内部资料 注意保密**

**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 5](#_Toc22617)

[1、 文档目的 5](#_Toc31993)

[2、 适用范围 5](#_Toc9622)

[3、 简介说明 5](#_Toc5238)

[4、 功能介绍 5](#_Toc18717)

[二、 快速使用 6](#_Toc13571)

[1、 使用CocoaPods 6](#_Toc27317)

[2、 导入SDK 6](#_Toc22686)

[三、 API功能说明 7](#_Toc12369)

[1、 设备连接相关 7](#_Toc6560)

[1.1 搜索蓝牙 7](#_Toc20077)

[1.2 停止搜索 7](#_Toc22538)

[1.3 连接指定蓝牙 7](#_Toc5267)

[1.4 断开蓝牙 8](#_Toc7818)

[1.5 当前连接蓝牙 8](#_Toc7903)

[1.6 蓝牙连接状态 8](#_Toc27748)

[1.7 监听连接状态 8](#_Toc23731)

[2、 指令功能 8](#_Toc29665)

[2.1 同步时间 8](#_Toc24059)

[2.2 读取时间 9](#_Toc6384)

[2.3 读取软件版本号 9](#_Toc9142)

[2.4 读取硬件版本号 9](#_Toc6688)

[2.5 读取电池电量 10](#_Toc21919)

[2.6 读取电池充电状态 10](#_Toc24093)

[2.7 实时测量心率值 11](#_Toc3617)

[2.8 心率变异性 11](#_Toc29264)

[2.9 实时测量血氧值 11](#_Toc14083)

[2.10 读取温度 12](#_Toc27482)

[2.11 读取当天实时步数 12](#_Toc1140)

[2.12 清除实时步数 12](#_Toc7803)

[2.13 读取本地数据 13](#_Toc18772)

[2.14 删除数据 14](#_Toc9002)

[2.15 设置采集周期 14](#_Toc30452)

[2.16 读取采集周期 15](#_Toc18020)

[2.17 恢复出厂设置 15](#_Toc27551)

[3、 固件升级（OTA） 15](#_Toc9595)

[4、 数据库RingDBManager 16](#_Toc3757)

[4.1 获取指定时间数据 16](#_Toc5237)

[4.2 获取某天数据 16](#_Toc28145)

[4.3 获取最近一条数据 16](#_Toc21556)

[4.4 获取某天睡眠数据 17](#_Toc28600)

[4.5 删除所有数据 17](#_Toc32505)

[4.6 删除指定时间到现在的数据 17](#_Toc1469)

[5、 逻辑算法 17](#_Toc1549)

[5.1 计算步行距离 17](#_Toc28858)

[5.2 睡眠时间计算 18](#_Toc12683)

[5.3 获取睡眠时长 18](#_Toc24957)

[6、 日志配置 18](#_Toc23459)

[四、 其他 18](#_Toc27565)

[1、 可能会遇到的问题 18](#_Toc10958)

[1.1 使用SDK出错 18](#_Toc8092)

[1.2 读取软件版本时，会多返回两个空格 19](#_Toc27340)

[1.3 调用接口，返回超时 19](#_Toc3043)

[1.4 戒指的睡眠功能如何测量 19](#_Toc25183)

[1.5 睡眠数据读取含有前几日数据 19](#_Toc11412)

[1.6 云端OTA更新 19](#_Toc13351)

[1.7 数据库无数据或数据库始终只有n条数据 19](#_Toc15687)

[2、 补充资料 20](#_Toc2824)

[2.1 睡眠数据 20](#_Toc19540)

[2.1.1 遵循协议 20](#_Toc5880)

[2.1.2 数据解析 20](#_Toc8423)

[3、 Q&A答疑 20](#_Toc17719)

[3.1 关于戒指 20](#_Toc21220)

[3.2 SDK使用可能产生的疑问 21](#_Toc14131)

[3.3 故障排除 23](#_Toc10927)

[3.3.1 使用SDK搜索附近蓝牙设备失败 23](#_Toc21934)

[3.3.2 压缩包解压失败 23](#_Toc20555)

# 文档简介

## 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。

## 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+

## 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

## 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 |  |

# 快速使用

## 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' |

## 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 |

# API功能说明

## 设备连接相关

### 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

### 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

### 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): 6. print("连接失败 ========> \(error)") 7. } 8. }) |

注意事项：无

### 断开蓝牙

接口说明：断开当前设备的连接

接口声明：

|  |
| --- |
| 1. RingManager.shared.disconnect() |

注意事项：调用此接口 ，需保证与戒指处于连接状态

### 当前连接蓝牙

接口说明：获取当前已连接的设备

接口声明：

|  |
| --- |
| 1. 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

### 蓝牙连接状态

接口说明：获取设备连接状态

接口声明：

|  |
| --- |
| 1. let isDidConnect = RingManager.shared.isDidConnect |

注意事项：无

### 监听连接状态

接口说明：设备连接状态变化监听

接口声明：

|  |
| --- |
| 1. RingManager.shared.connectStateChangeBlock = { isConnected in 2. print("是否已连接 ========\(isConnected)") 4. } |

注意事项：无

## 指令功能

### 同步时间

接口说明：

接口声明：

|  |
| --- |
| 1. RingManager.shared.syncTime(date: Date()) { res in 2. switch res { 3. case .success(let isSuccess): 4. print("同步时间结果======\(isSuccess)") 5. case .failure(let error): 6. print("同步失败======\(error)") 7. } 8. } |

注意事项：调用此接口 ，需保证与戒指处于连接状态

### 读取时间

接口说明：

接口声明：

|  |
| --- |
| 1. RingManager.shared.readTime { res in 2. switch res { 3. case .success(let value): 4. print("成功=====>\(value)毫秒") 5. case .failure(let error): 6. print("失败=====>\(error)") 7. } 8. } |

注意事项：调用此接口 ，需保证与戒指处于连接状态

### 读取软件版本号

接口说明：

接口声明：

|  |
| --- |
| 1. RingManager.shared.readAppVersion { res in 2. switch res { 3. case .success(let version): 4. print("软件版本号=====>\(version)") 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

### 读取硬件版本号

接口说明：

接口声明：

|  |
| --- |
| 1. RingManager.shared.readHardWareVersion { res in 2. switch res { 3. case .success(let version): 4. print("硬件版本号=====>\(version)") 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

### 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：

|  |
| --- |
| 1. RingManager.shared.readBattery { res in 2. switch res { 3. case .success(let value): 4. print("电量=====>\(value)") 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

### 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：

|  |
| --- |
| 1. RingManager.shared.readChargeStatus { res in 2. switch res { 3. case .success(let state): 4. print("状态=====>\(state)") 5. case .failure(let error): 6. print("失败=====>\(error)") 7. } 8. } |

注意事项：调用此接口 ，需保证与戒指处于连接状态

### 实时测量心率值

接口说明：输出实时测量心率值，单位BPM。只有当设置isOpenWave为true时，才会读取波形数据，并且通过tableBlock闭包回调。

接口声明：

|  |
| --- |
| 1. RingManager.shared.readHeartRate(progressBlock: { progress in 2. print(" 进度 =====>\(progress)") 3. }, isOpenWave: true) { seq, num, datas in 4. print("序号 ====>\(seq)") 5. print("数据个数 ====>\(num)") 6. print("波形数据 ====>\(datas)") 7. } resultBlock: { res in 8. switch res { 9. case .success(let success): 10. print("成功=====>\(success)") 11. case .failure(let failure): 12. print("失败=====>\(failure)") 13. } 14. } |

注意事项：调用此接口 ，需保证与戒指处于连接状态

### 心率变异性

接口说明： 输出心率变异性，单位毫秒(ms)。只有当设置isOpenWave为true时，才会读取波形数据，并且通过tableBlock闭包回调。

接口声明：

|  |
| --- |
| 1. RingManager.shared.readHRV(progressBlock: { progress in 2. BDLogger.info(" 进度 =====>\(progress)") 3. }, isOpenWave: true) { seq, num, datas in 4. BDLogger.info("序号 ====>\(seq)") 5. BDLogger.info("数据个数 ====>\(num)") 6. BDLogger.info("波形数据 ====>\(datas)") 7. } resultBlock: { res in 8. switch res { 9. case .success(let success): 10. BDLogger.info("成功=====>\(success)") 11. case .failure(let failure): 12. BDLogger.info("失败=====>\(failure)") 13. } 14. } |

注意事项：调用此接口 ，需保证与戒指处于连接状态

### 实时测量血氧值

接口说明： 输出实时血氧值。只有当设置isOpenWave为true时，才会读取波形数据，并且通过tableBlock闭包回调。

接口声明：

|  |
| --- |
| 1. RingManager.shared.readO2(progressBlock: { progress in 2. BDLogger.info(" 进度 =====>\(progress)") 3. }, isOpenWave: true) { seq, num, datas in 4. BDLogger.info("序号 ====>\(seq)") 5. BDLogger.info("数据个数 ====>\(num)") 6. BDLogger.info("波形数据 ====>\(datas)") 7. } resultBlock: { res in 8. switch res { 9. case .success(let success): 10. BDLogger.info("成功=====>\(success)") 11. case .failure(let failure): 12. BDLogger.info("失败=====>\(failure)") 13. } 14. } |

注意事项：调用此接口 ，需保证与戒指处于连接状态

### 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：

|  |
| --- |
| 1. RingManager.shared.readTemperature { res in 2. switch res { 3. case .success(let value): 4. BDLogger.info("温度=====>\(value)") 5. case .failure(let error): 6. BDLogger.info("失败=====>\(error)") 7. } 8. } |

注意事项：调用此接口 ，需保证与戒指处于连接状态

### 读取当天实时步数

接口说明：

接口声明：

|  |
| --- |
| 1. RingManager.shared.readSteps { res in 2. switch res { 3. case .success(let value): 4. print("步数=====>\(value)步") 5. case .failure(let error): 6. print("失败=====>\(error)") 7. } 8. } |

注意事项：调用此接口 ，需保证与戒指处于连接状态

### 清除实时步数

接口说明：

接口声明：

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

注意事项：调用此接口 ，需保证与戒指处于连接状态

### 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：

|  |
| --- |
| 1. RingManager.shared.readDatas { progress, dataModel in 2. print(" 进度 =====>\(progress)==\(dataModel)") 3. } resultBlock: { res in 4. switch res { 5. case .success(let state): 6. print("结果=====>\(state)") 7. case .failure(let error): 8. print("失败=====>\(error)") 9. } 10. } |

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：

|  |
| --- |
| 1. public final class RingDataModel: NSObject,TableCodable { 2. *// 总个数* 3. public var total:UInt32 = 0 4. *// 序号，1开始* 5. public var serialNum:UInt32 = 0 6. *// 时间戳,秒* 7. public var timestamp:UInt32 = 0 8. *// 当天截止当前累计步数* 9. public var stepsOfTheDay:UInt16 = 0 10. *// 心率,0无效* 11. public var rate = 0 12. *// 血氧,0无效* 13. public var O2 = 0 14. *// 心率变异性,0无效* 15. public var hrv = 0 16. *// 精神压力指数,0无效* 17. public var mentalStress = 0 18. *// 温度，有符号的* 19. public var temp:Float = 0 20. *// 运动激烈程度 0: 静止  0x65:运动* 21. public var isRunning = false 22. *// 睡眠类型 0：无效 1：清醒 2：浅睡 3：深睡 4.眼动期* 23. public var sleepType = 0 24. *// RR 期间个数* 25. public var RRNums = 0 26. *// RR数组* 27. public var rrs:[Int] = [] 28. } |

返回值：state为ReadDataResult枚举类型，值有:

|  |
| --- |
| 1. public enum ReadDataResult { 2. case empty *// 无数据* 3. case complete *// 完成* 4. } |

### 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

### 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

### 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：

|  |
| --- |
| 1. RingManager.shared.readFrequency { res in 2. switch res { 3. case .success(let value): 4. print("成功=====>\(value)秒") 5. case .failure(let error): 6. print("失败=====>\(error)") 7. } 8. } |

注意事项：调用此接口 ，需保证与戒指处于连接状态

### 恢复出厂设置

接口说明：

接口声明：

|  |
| --- |
| 1. RingManager.shared.reset { res in 2. switch res { 3. case .success(let isSuccess): 4. print("成功=====>\(isSuccess)") 5. case .failure(let error): 6. print("失败=====>\(error)") 7. } 8. } |

注意事项：调用此接口 ，需保证与戒指处于连接状态

## 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：

|  |
| --- |
| 1. if let path = Bundle.main.path(forResource: "otafileName", ofType: nil) { 2. let url = URL(fileURLWithPath: path) 3. RingManager.shared.startOTA(url) { res in 4. switch res { 5. case .start: 6. print("开始升级 =====>") 7. case .progress(let value): 8. print("升级进度 =====>\(value)") 9. case .success: 10. print("升级成功 =====>") 11. case .fail(let errorString): 12. print("升级失败 =====>\(errorString)") 13. } 14. } 15. } |

注意事项：调用此接口 ，需保证与戒指处于连接状态

## 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。

### 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

### 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

### 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

### 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

### 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

### 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

## 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

### 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

### 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

### 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

## 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

# 其他

## 可能会遇到的问题

### 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+

### 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

### 调用接口，返回超时

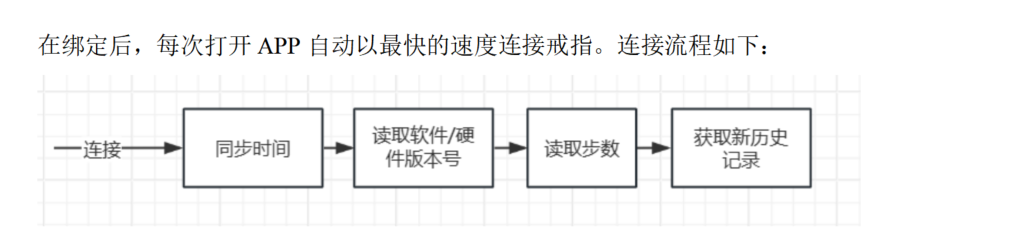
发送了一个指令，过了3秒，戒指没有响应，会返回超时

超时问题要具体分析

### 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。

### 睡眠数据读取含有前几日数据



可能因为时间戳未同步

### 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

### 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）

## 补充资料

### 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

#### 遵循协议

1变非1是入睡，再变1是清醒

#### 数据解析

睡眠类型：

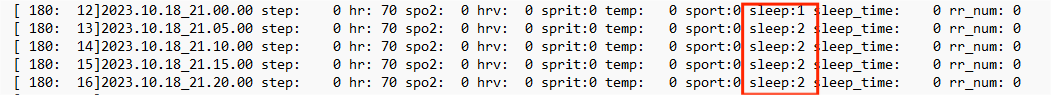
0：无效

1：清醒

2：浅睡

3：深睡

4：眼动期



## Q&A答疑

### 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

### 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))") 8. let sleepTimes = RingManager.shared.calculateSleepTimes(sleepDatas: datas) 9. print("睡眠时间 ======>\(sleepTimes)分钟")  12. var lastModel:RingDataModel? 13. *// 清醒时长，单位秒* 14. var wakeTimes:UInt32 = 0 15. *// 浅睡时长，单位秒* 16. var lightSleepTimes:UInt32 = 0 17. *// 深睡时长，单位秒* 18. var deepSLeepTimes:UInt32 = 0 19. *// 眼动期时长，单位秒* 20. var eyesTimes:UInt32 = 0 21. datas.forEach { model in 22. if let tempLastModel = lastModel { 23. switch model.sleepType { 24. case 1:*// 清醒* 25. wakeTimes += model.timestamp - tempLastModel.timestamp 26. case 2:*// 浅睡* 27. lightSleepTimes += model.timestamp - tempLastModel.timestamp 28. case 3:*// 深睡* 29. deepSLeepTimes += model.timestamp - tempLastModel.timestamp 30. case 4:*// 眼动* 31. eyesTimes += model.timestamp - tempLastModel.timestamp 32. default : 33. break 34. } 35. lastModel = model 36. }else{ 37. lastModel = model 38. } 39. } 41. print("清醒时长 ======>\(wakeTimes)") 42. print("浅睡时长 ======>\(lightSleepTimes)") 43. print("深睡时长 ======>\(deepSLeepTimes)") 44. print("眼动时长 ======>\(eyesTimes)") |

### 故障排除

#### 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。

#### 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