



ChipletRing APP SDK 说明书

IOS

| | | | |
|--------|----------|-------|-------------|
| 文档版本号: | 1.0.10 | 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 |
| 1.0.5 | 郑玉虎 | 2024.07.09 | 新增睡眠逻辑流程图 |
| 1.0.6 | 郑玉虎 | 2024.09.12 | 修改部分描述, 适配 SDK |
| 1.0.10 | 郑玉虎 | 2024.11.21 | 添加心率停止接口、设置蓝牙名称、硬链接及重连、午睡、更新蓝牙断开状态、午睡示例 |
| | | | |

目 录

| | |
|--------------------------|----|
| 一、 文档简介 | 5 |
| 1、 文档目的 | 5 |
| 2、 适用范围 | 5 |
| 3、 简介说明 | 5 |
| 4、 功能介绍 | 5 |
| 二、 快速使用 | 6 |
| 1、 使用 CocoaPods | 6 |
| 2、 导入 SDK | 6 |
| 三、 API 功能说明 | 7 |
| 1、 设备连接相关 | 7 |
| 1.1 搜索蓝牙 | 7 |
| 1.2 停止搜索 | 7 |
| 1.3 连接指定蓝牙 | 7 |
| 1.4 通过 MAC 连接戒指的示例 | 8 |
| 1.5 断开蓝牙 | 9 |
| 1.6 当前连接蓝牙 | 9 |
| 1.7 蓝牙连接状态 | 9 |
| 1.8 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 | 14 |
| 2.11 读取当天实时步数 | 15 |
| 2.12 清除实时步数 | 16 |
| 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 |
| 2.18 设置蓝牙名称 | 20 |
| 2.19 获取蓝牙名称 | 20 |
| 2.20 心率停止测量 | 21 |
| 2.21 血氧停止测量 | 21 |

| | |
|----------------------------------|-----------|
| 2.22 可设置时间的心率及测量返回 RR 原始数据 | 22 |
| 3、固件升级 (OTA) | 22 |
| 4、数据库 RingDBManager | 23 |
| 4.1 获取指定时间数据 | 23 |
| 4.2 获取某天数据 | 24 |
| 4.3 获取最近一条数据 | 24 |
| 4.4 获取某天睡眠数据 | 24 |
| 4.5 删除所有数据 | 24 |
| 4.6 删除指定时间到现在的数据 | 25 |
| 5、逻辑算法 | 25 |
| 5.1 计算步行距离 | 25 |
| 5.2 睡眠时间计算 | 25 |
| 5.3 获取睡眠时长 | 26 |
| 6、日志配置 | 26 |
| 四、其他 | 27 |
| 1、可能会遇到的问题 | 27 |
| 1.1 使用 SDK 出错 | 27 |
| 1.2 读取软件版本时，会多返回两个空格 | 27 |
| 1.3 调用接口，返回超时 | 27 |
| 1.4 戒指的睡眠功能如何测量 | 27 |
| 1.5 睡眠数据读取含有前几日数据 | 28 |
| 1.6 云端 OTA 更新 | 28 |
| 1.7 数据库无数据或数据库始终只有 n 条数据 | 28 |
| 2、补充资料 | 28 |
| 2.1 睡眠数据 | 28 |
| 2.1.1 遵循协议 | 28 |
| 2.1.2 数据解析 | 29 |
| 2.2 睡眠逻辑图 | 29 |
| 2.3 午睡逻辑图 | 30 |
| 3、Q&A 答疑 | 30 |
| 3.1 关于戒指 | 30 |
| 3.2 SDK 使用可能产生的疑问 | 31 |
| 3.3 故障排除 | 34 |
| 3.3.1 使用 SDK 搜索附近蓝牙设备失败 | 34 |
| 3.3.2 压缩包解压失败 | 34 |

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/Z-y-hu/ChipletRing-SDK-iOS.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):
```

```
6.         print("连接失败 =====> \(error)")
7.     }
8. })
9.
```

注意事项：无

1.4 通过 MAC 连接戒指的示例

接口说明：通过设备的 mac 地址来连接指定设备

接口声明：

```
1. if sender.isSelected {
2.     RingManager.shared.startScan { [self] devices in
3.         BDLogger.info("设备列表 =====> \ (String(describing: devices))")
4.         if let devices = devices{
5.             let macTarget="B0:02:30:00:03:E1"
6.             for device in devices{
7.                 var macString = ""
8.                 if let macData = device.advertisementData["kCBAAdvDataManufacturerData"] as? Data, macData.count >= 8 {
9.                     macString = String(format: "%02X:%02X:%02X:%02X:%02X:%02X", macData[7], macData[6], macData[5], macData[4], macData[3], macData[2])
10. //                                     print("扫描到的蓝牙 mac:\ (macString) ")
11.                 }
12.                 if macString == macTarget{
13.                     self.connectDevice(device: device)
14.                     self.uuidS = device.uuidString
15.                     print("公版连接指定 mac, 连接状态 \ (self.isDidConnect) ")
16.                 }
17.             }
18.         }
19.     }
20. } else {
```



```
21.         RingManager.shared.stopScan ()
22.     }
```

注意事项：无

1.5 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

1.6 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

1.7 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

1.8 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)")
3.     }
```

4. }

注意事项：无

2、 指令功能

2.1 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

2.2 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

2.3 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

2.4 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

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：

```
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

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：

```
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

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：

```
1. RingManager.shared.readHeartRate(progressBlock: { progress in
2.     print(" 测量进度 =====> \(progress)")
3. }) { res in
4.     switch res {
5.     case .success(let value):
6.         print("心率=====> \(value)")
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

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：

```
1. RingManager.shared.readHRV(progressBlock: { progress in
2.     print(" 测量进度 =====> \(progress)")
3. }) { res in
4.     switch res {
5.     case .success(let value):
6.         print("心率变异性=====> \(value)")
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

追加：返回值的第 5-10 字节含义（1-4 字节可以视为头文件，不需要解析）

[5]:佩戴状态 0 未佩戴, 1 佩戴, 2 充电中, 3 采集中, 4 繁忙

[6]:

[7]:心率变异性 0 无效, 单位 ms

[8]:

[9:10]:

例，取最后一个上报的 10 字节数据 0x00093100035c4c061c0e，第 7 个字节就是心率变异性,4c=76

2.9 实时测量血氧值

接口说明：输出实时血氧值

接口声明：

```
1. RingManager.shared.readO2(progressBlock: { progress in
2.     print(" 测量进度 =====>\(progress)")
3. }) { res in
4.     switch res {
5.     case .success(let value):
6.         print("血氧值=====>\(value)")
7.     case .failure(let error):
8.         print("失败=====>\(error)")
9.     }
10.    }
11.
```

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：

```
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

追加：返回值的第 5-7 字节含义（1-4 字节可以视为头文件，不需要解析）

[5]:佩戴状态 0 测量中，1 测量完成，2 未佩戴，3 繁忙

[6:7]: 温度(精度 0.01)，单位℃ 类型：有符号短整形

例，取最后一个上报的 7 字节数据 0x 000d3400016f0d, 第 6, 7 个字节就是温度,0d6f=3439, 即 34.39℃

2.11 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

2.12 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

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:

```
1. RingManager.shared.readDats { 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. }
```

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：

```
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

2.17 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.      }
```

注意事项：调用此接口，需保证与戒指处于连接状态

2.18 设置蓝牙名称

接口说明：设置戒指蓝牙名称

参数说明：name:蓝牙名称，不超过 12 个字节，可以为中文、英文、数字，即 4 个汉字或者 12 个英文

接口声明：

```
1.         manager.setBluetoothName(name: "你真的 5H"){res in
2.             switch res{
3.                 case .success(let success):
4.                     BDLogger.info("成功====>\(success)")
5.                 case .failure(let failure):
6.                     BDLogger.info("失败====>\(failure)")
7.             }
8.         }
```

注意事项：1.调用此接口，需保证与戒指处于连接状态

2. 设置蓝牙名称后，广播不会立即改变，需要等待一段时间

2.19 获取蓝牙名称

接口说明：获取戒指蓝牙名称

接口声明：

```
1.manager.readBluetoothName(){res in
2.     switch res{
3.         case .success(let success):
4.             BDLogger.info("成功====>\(success)")
5.         case .failure(let failure):
6.             BDLogger.info("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

2.20 心率停止测量

接口说明：可以在测量心率时调用来停止测量

接口声明：

```
1.manager.stopHeartRate{res in
2.    switch res{
3.        case .success(let success):
4.            BDLogger.info("成功====>\(success)")
5.        case .failure(let failure):
6.            BDLogger.info("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

2.21 血氧停止测量

接口说明：可以在测量血氧时调用来停止测量

接口声明：

```
1.manager.stopOxygenTest{res in
2.    switch res{
3.        case .success(let success):
4.            BDLogger.info("成功====>\(success)")
5.        case .failure(let failure):
6.            BDLogger.info("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

2.22 可设置时间的心率及测量返回 RR 原始数据

接口说明： 读取设定时间的心率及 RR

接口声明：

参数声明： rrTime:心率测量时间 单位： S

```
1.manager.readHeartRateWithRR(rrTime: 50, progressBlock: {progress in
2.         BDLogger.info(" 进度 =====>\(progress) ")
3.     }, inHeartRRBlock: {seq, num, datas in
4.         BDLogger.info("序号 =====>\(seq) ")
5.         BDLogger.info("数据个数 =====>\(num) ")
6.         BDLogger.info("RR 数据 =====>\(datas) ")
7.     }, inHeartBlock: {heartValue in
8.         BDLogger.info("测量中的心率\ \(heartValue) ")
9.     }, resultBlock: { res in
10.         switch res {
11.             case .success(let success) :
12.                 BDLogger.info("成功=====>\(success) ")
13.             case .failure(let failure) :
14.                 BDLogger.info("失败=====>\(failure) ")
15.         }
16.     })
```

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：

```
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

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]])
```

使用示例：

```
1. let dateString = "2024-11-14 10:30:33"
2.         let dateFormatter = DateFormatter()
3.         dateFormatter.dateFormat = "yyyy-MM-dd HH:mm:ss"
4.         dateFormatter.timeZone = TimeZone(abbreviation: "UTC") /
/ 设置时区为 UTC
5.         guard let targetDate = dateFormatter.date(from: dateStri
ng) else { return }
6.//         let targetDate = Date() - 1.days
7.         let (sleepData,sporadicSleepData) = manager.caculateSlee
pData(targetDate: targetDate)
8.         BDLogger.info("睡眠数据\(sleepData)=====零星睡眠数据
\ (sporadicSleepData)")
9.         break
```

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

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 readDats(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 getObject(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.getObject(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.
8. let sleepTimes = RingManager.shared.calculateSleepTimes(sleepDatas: datas)
9. print("睡眠时间 =====>\(sleepTimes)分钟")
10.
11.
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.     }
40.
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)")
```

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/Z-y-hu/ChipletRing-SDK-iOS>。可自行下载