密级：	保密	版本号：	V1.0.6
使用范围：	内部使用	文档状态：	发行

NexRing JS SDK API 说明文档

文件编号：	LT_SW_2023042702	类别：	RD
拟制：	Chenzq	日期：	2023-4-27
审核：	Yangxd	日期：	2023-4-27
批准：	Tangzp	日期：	2023-4-27

LINKTOP凌拓

厦门市凌拓通信科技有限公司

■ 文档版本

版本	发布日期	描述
V1.0.0	2023.04.27	初版本
V1.0.1	2023.05.26	更新睡眠算法
V1.0.2	2023.6.30	增加 OEM 认证,对 HRV 和电量设备兼容问题的修改
V1.0.3	2023.7.07	增加 OTA 功能
V1.0.4	2023.7.19	修改睡眠算法实现
V1.0.5	2023.8.10	兼容新旧固件历史数据获取，计步增加获取计步算法类型
V1.0.6	2023.9.04	增加运动模式，增加卡路里计算，手动模式增加 hrv 输出，手动模式增加原始波形数据输出

目录

1.集成说明

1.1 工程开发说明

（1）将ringSDK.js文件导入开发项目工程src目录下的lib目录中。

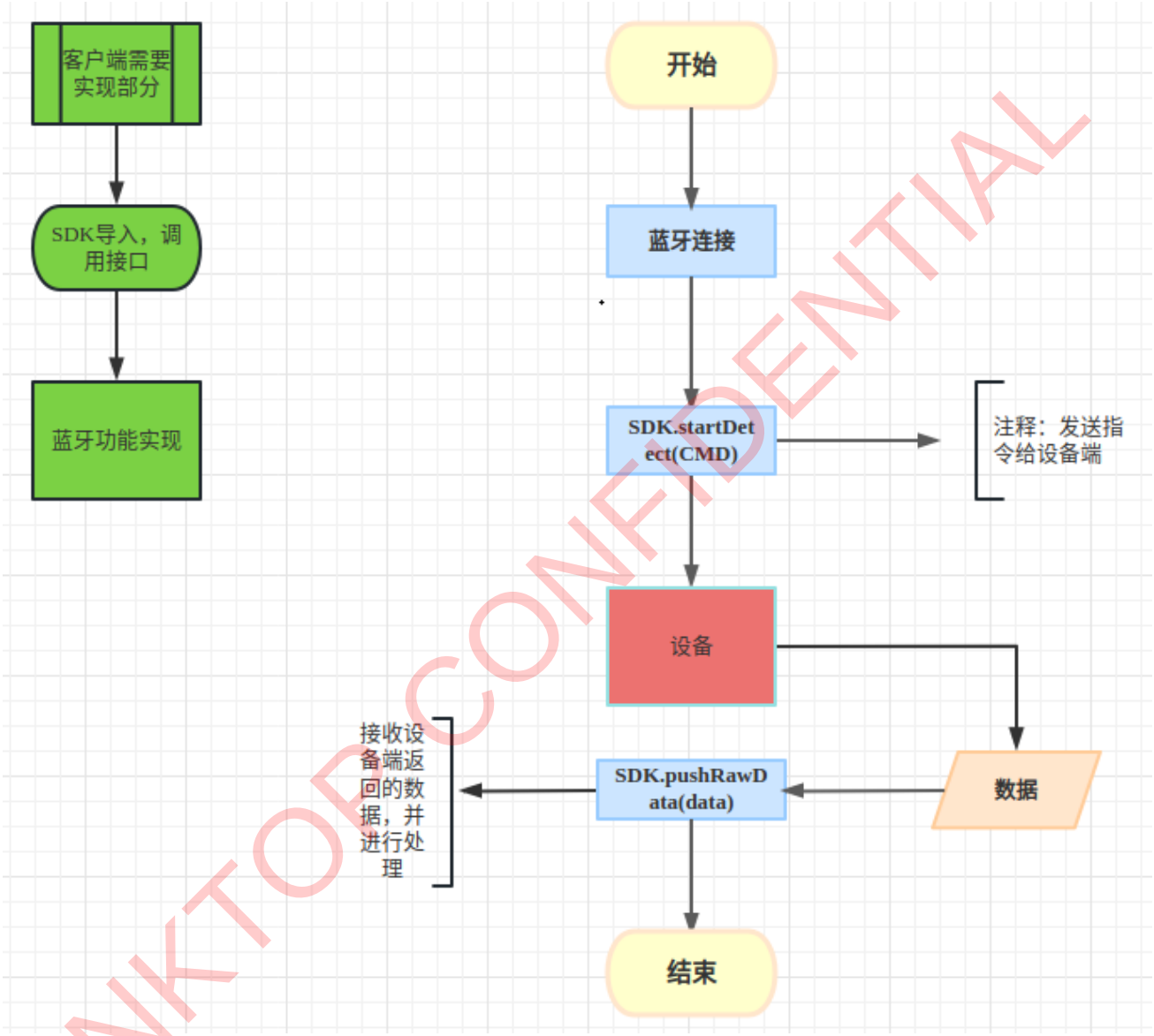
Demo 中采用的是 react-native-ble-manager 蓝牙库

2. API 说明

2.1 功能说明

SDK通过BLE与设备建立连接。您可以通过SDK轻松与设备进行通信，提取设备提供的心电，血氧，体温等数据服务。您可以使用本套SDK开发基于javascript开发react native应用和微信小程序。本SDK为仅提供心电，血氧，体温等数据输出，供用户开发测试使用。

2.2 流程图



2.3 对外接口详解

1．接口初始化

在你需要使用接口的地方进行初始化

ImportSDKfrom"../lib/ringSDK"

蓝牙连接参数：

```
export const FILTER_UUID="FEF5";
export const UUID_SERVICE="00001822-0000-1000-8000-00805F9B34FB";
export const UUID_SERVICE_IOS="1822";
export const WRITE_UUID="000066FE-0000-1000-8000-00805F9B34FB";
export const NOTIFY_UUID="000066FE-0000-1000-8000-00805F9B34FB";
```

FILTER_UUID:用于蓝牙扫描的时候过滤用的UUID

UUID_SERVICE:蓝牙连接成功后anroid端服务UUID

UUID_SERVICE_IOS:蓝牙连接成功后IOS端服务UUID

WRITE_UUID：写数据的UUID

NOTIFY_UUID: 接收数据的UUID

蓝牙实现部分：

蓝牙扫描

```
BleManager.scan([FILTER_UUID], 30, true)
    .then(() => {
        console.log('Scan started');
        resolve();
    })
    .catch(error => {
        console.log('Scan started fail', error);
        reject(error);
    });
```

蓝牙连接

```
BleManager.connect(id)
    .then(() => {
        console.log('Connected success');
        // 获取已连接蓝牙设备的服务和特征
        return BleManager.retrieveServices(id);
    })
    .then(peripheralInfo => {
        console.log('Connected peripheralInfo', peripheralInfo);
        this.peripheralId = peripheralInfo.id;
        this.getUUID(peripheralInfo);
        resolve(peripheralInfo);
    })
    .catch(error => {
        console.log('Connected fail', error);
        reject(error);
    });
```

蓝牙发送数据:

```
function sendData(cmd, data) {
    var result = SDK.startDetect(cmd, data);
    console.log(`sendData result=${result}`);
    bleModule.write(Array.from(new Uint8Array(result)));
}
```

```
write(data, index = 0) {
    return new Promise((resolve, reject) => {
        BleManager.write(
            this.peripheralId,
            this.writeWithResponseServiceUUID[index],
            this.writeWithResponseCharacteristicUUID[index],
            data,
        )
            .then(() => {
                console.log('Write success', data.toString());
                resolve();
            })
            .catch(error => {
                console.log('Write failed', data);
                reject(error);
            });
    });
}
```

蓝牙接收数据:

```
function handleUpdateValue(data) {
    let value = data.value;
    SDK.pushRawData(value);
}
```

2．启动接口

startDetect(type);

蓝牙连接后调用startDetect启动测试接口，参数type =>

```
const step = "step";
const timeSyn = "timeSyn";
const openSingleHealth = "openSingleHealth";
const closeSingleHealth = "closeSingleHealth";
const openHealth = "openHealth";
const closeHealth = "closeHealth";
const temperature = "temperature";
const shutDown = "shutDown";
const restart = "restart";
const restoreFactorySettings = "restoreFactorySettings";
const historicalNum = "historicalNum";
const historicalData = "historicalData";
const cleanHistoricalData = "cleanHistoricalData";
const deviceInfo1 = "deviceInfo1";
const deviceInfo2 = "deviceInfo2";
const batteryDataAndState = "batteryDataAndState"
const deviceBind = "deviceBind"
const deviceUnBind = "deviceUnBind"
```

例：startDetect(step) 开启计步测量,建议每次连接戒指调用startDetect(timeSyn)同步时间

2．回调接口

1) SDK.registerHealthListener(healthListener);

心率血氧回调函数注册

```
const healthListener = {
    onResult: (data) => {
        if (data && data.status == 2) {
            if (data.oxValue == 0) {
                setHeartData({
                    heartValue: data.heartValue,
                    hrvValue: data.hrvValue,
                })
            } else {
                if (data.oxValue >= 95) {
                    var ox = data.oxValue >= 100 ? 99 : data.oxValue;
                    setHealthData({
                        oxValue: ox,
                        heartValue: data.heartValue,
                    })
                }
            }
        }
    }
}
```

回调函数

onResult:(data)

data.status =>0:未启动测量；1：测量中；2：测量数据有效

data.heartValue=>心率数据

data.oxValue=>血氧数据

data.hrvValue=>hrv数据

2) SDK.registerHealthListener(batteryDataAndStateListener);

电量回调函数注册

```
const batteryDataAndStateListener = {
    onResult: (data) => {
        if (data) {
            var isWireless = false;
            if (bleName) {
                isWireless = bleName.toUpperCase().indexOf('W') == -1 ? false : true;
            }
            var charging = data.status == 1
            var result = charging ? "充电中" : "未充电";
            var batteryPer = SDK.calcBattery(data.batteryValue, charging, isWireless);
            setBattery({
                batteryValue: data.batteryValue,
                status: result,
                batteryPer
            })
        }
    }
}
```

onResult:(data)

isWireless:通过判断蓝牙名是否包含W来判断是无线充电还是有线充电

data.status=>0:未充电;1:充电中

data.batteryValue=>电池电压

batteryPer=>电池电量百分比（SDK.calcBattery计算电池电量百分比）

3) SDK.registerDeviceInfo1Listener(deviceInfo1Listener)

设备信息1回调函数注册

```
const deviceInfo1Listener = {
    onResult: (data) => {
        if (data) {
            var color = ""
            if (data.color == 0) {
                color = "Deep Black"
            } else if (data.color == 1) {
                color = "Silver"
            }
            setDevice1Value({
                color,
                size: data.size,
                bleAddress: data.bleAddress,
                deviceVer: data.deviceVer
            })
        }

    }
}
```

onResult:(data)

data.color=>戒指的颜色0:Deep Black;1:Silver

data.size=>戒指的尺寸

data.bleAddress=>戒指的蓝牙mac地址

data.deviceVer=>戒指的设备版本号

data.switchOem=>0:OEM认证开关关闭 1:OEM认证开关开启

data.chargingMode=>0:带电池充电仓磁吸充电 1：无线充电 2：NFC无线充电 3：不带充电座磁吸充电 4：USB线磁吸充电

data.mainChipModel=>主芯片型号

data.productIteration=>产品迭代

data.hasSportsMode=>是否支持运动模式

4) SDK.registerDeviceInfo2Listener(deviceInfo2Listener);

设备信息2回调函数注册

```
const deviceInfo2Listener = {
    onResult: (data) => {
        setDevice2Value({
            sn: data.sn,
            bindStatus: data.bindStatus,
            samplingRate: data.samplingRate,
        })
    }
}
```

onResult:(data)

data.sn=>设备sn号

data.bindStatus=>绑定状态

data.samplingRate=>血氧IR波形数据采样率

5) SDK.registerHistoricalDataListener(historicalDataListener)

历史数据回调函数注册

```
const historicalDataListener = {
    onResult: (data) => {
        if (data.hrv > 0) {
            var wearStatus=data.wearStatus==1?wear:noWear;
            var chargeStatus=data.chargeStatus==1?charging:uncharged;
            var detectionModeStatus=data.detectionMode==1?BloodOxygenMode:HeartRateMode
            setHistory({
                timeStamp: data.timeStamp,
                heartRate: data.heartRate,
                motionDetectionCount: data.motionDetectionCount,
                detectionMode: detectionModeStatus,
                wearStatus: wearStatus,
                chargeStatus: chargeStatus,
                uuid: data.uuid,
                hrv: data.hrv,
                temperature: data.temperature,
                step: data.step,
                ox: data.ox
            })
        }

        if (data.heartRate >= 50 && data.heartRate <= 175 && data.wearStatus == 1 && data.chargeStatus == 0) {
            mArray.current.push({
                ts: data.timeStamp,
                hr: data.heartRate,
                hrv: data.hrv,
                motion: data.motionDetectionCount,
                steps: data.step,
                ox: data.ox
            })
        }
        if (data.heartRate >= 60 && data.heartRate <= 175 && data.wearStatus == 1 && data.chargeStatus == 0) {
            mHrArray.current.push({
                ts: data.timeStamp,
                hr: data.heartRate,
            })
        }

    }
}
```

onResult:(data)

data.hrv=>hrv值

data.timeStamp=>时间戳

data.heartRate=>心率

data.motionDetectionCount=>运动检测计数

data.detectionMode=>测量模式0:心率测量模式；1:血氧测量模式

data.wearStatus=>佩戴状态0:未佩戴；1:佩戴

data.chargeStatus=>充电状态0:未充电；1:充电中

data.uuid=>uuid值，从1开始累计

data.temperature=>手指温度

data.step=>步数

data.ox=>血氧

建议可以读取历史记录后，将数据保存进数据库，然后清除历史记录

6) SDK.registerHistoricalNumListener(historicalNumListener);

历史数据个数回调函数注册

```
const historicalNumListener = {
    onResult: (data) => {
        console.log(`历史数据个数 data=${JSON.stringify(data)}`)
    }
}
```

onResult:(data)

data.num=>历史数据数量

data.minUUID=>uuid最小值

data.maxUUID=>uuid最大值

注意要请求历史数据之前先要请求历史数据个数

7) SDK.registerStepListener(stepListener);

步数回调函数注册

```
const stepListener = {
    onResult: (data) => {
        if (data) {
            setStepValue(data);
        }
    }
}
```

onResult:(data)

data.stepCount=>步数

data.stepAlgorithm=>使用的步数算法，返回LIS2DS12:使用设备的原始步数数据，如果没有返回数据则使用APP的计步算法

8) SDK.registerTemperatureListener(temperatureListener)

手指温度回调函数注册

```
const temperatureListener = {
    onResult: (data) => {
        if (data) {
            setTemperatureValue(data);
        }
    }
}
```

onResult:(data)

data=>手指温度

9) SDK.registerRePackageListener(rePackageListener);

回包回调函数注册

```
const rePackageListener = {
    onResult: (data) => {
        if (data) {
            setRePackage(
                {
                    cmd: data.cmd,
                    result: data.result,
                    reason:data.reason
                }
            )
            dealStatus(data);
        }
    }
}
```

onResult:(data)

data.cmd=>下发的指令

data.result=>返回下发命令的结果

data.reason=>返回下发命令失败原因

10) NativeSleepModule.getSleepData(Array,function)

睡眠数据算法（android）

```
function getSleepData() {
    if (mArray.current.length != 0) {
        if (Platform.OS == "android") {
            NativeSleepModule.getSleepData(mArray.current, (result) => {
                var sleepTimeArray = [];
                for (let index = 0; index < result.length; index++) {
                    const data = result[index];
                    var lightTime = 0;
                    var deepTime = 0;
                    var remTime = 0;
                    var wakeTime = 0;
                    var napTime = 0;
                    for (let index = 0; index < data.stagingList.length; index++) {
                        const element = data.stagingList[index];
                        switch (element.stagingType) {
                            case "NREM1":
                                lightTime += element.endTime - element.startTime
                                break
                            case "NREM3":
                                deepTime += element.endTime - element.startTime
                                break;
                            case "REM":
                                remTime += element.endTime - element.startTime
                                break;
                            case "WAKE":
                                wakeTime += element.endTime - element.startTime
                                break;
                            case "NAP":
                                napTime += element.endTime - element.startTime
                                break;
                        }
                    }
                    sleepTimeArray.push({
                        deepSleep: `deepTime= ${Math.floor(deepTime / (1000 * 60 * 60))}h${Math.floor((deepTime % (1000 * 60
                        lightTime: `lightTime= ${Math.floor(lightTime / (1000 * 60 * 60))}h${Math.floor((lightTime % (1000 *
                        remTime: `remTime= ${Math.floor(remTime / (1000 * 60 * 60))}h${Math.floor((remTime % (1000 * 60 * 60)
                        wakeTime: `wakeTime= ${Math.floor(wakeTime / (1000 * 60 * 60))}h${Math.floor((wakeTime % (1000 * 60 *
                        napTime: `napTime= ${Math.floor(napTime / (1000 * 60 * 60))}h${Math.floor((napTime % (1000 * 60 * 60)
                        startTime: formatDateTime(data.startTime),
                        endTime: formatDateTime(data.endTime)
                    })
                }
                setSleepTime(sleepTimeArray);
            })
```

参数:

Array:历史数据数组

返回值:

deepSleep:深度睡眠值

lightTime:轻度睡眠值

remTime：快速动眼睡眠值

wakeTime: 苏醒时间值

naptime:零星小睡

sleepTimePeriod.startTime:睡眠开始时间

sleepTimePeriod.endTime:睡眠结束时间

deepList:深度睡眠时间段数组

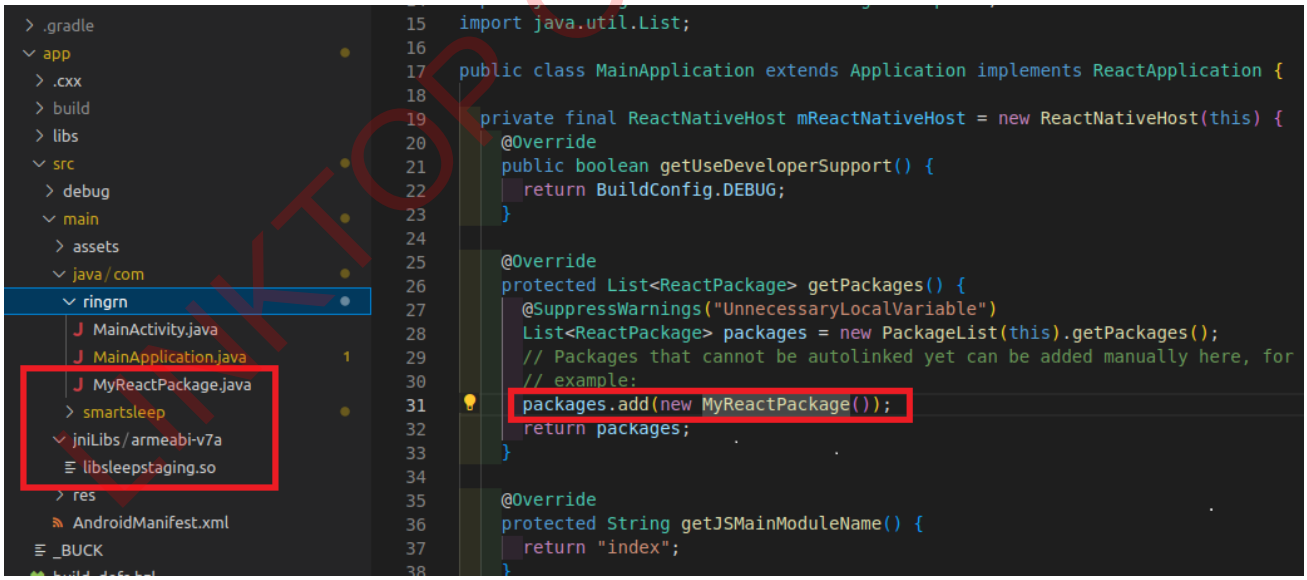
lightList:轻度睡眠时间段数组

remList:快速动眼睡眠时间段数组

wakeList:苏醒时间段数组

napList:零星小睡时间段数组

android目录需要添加的代码

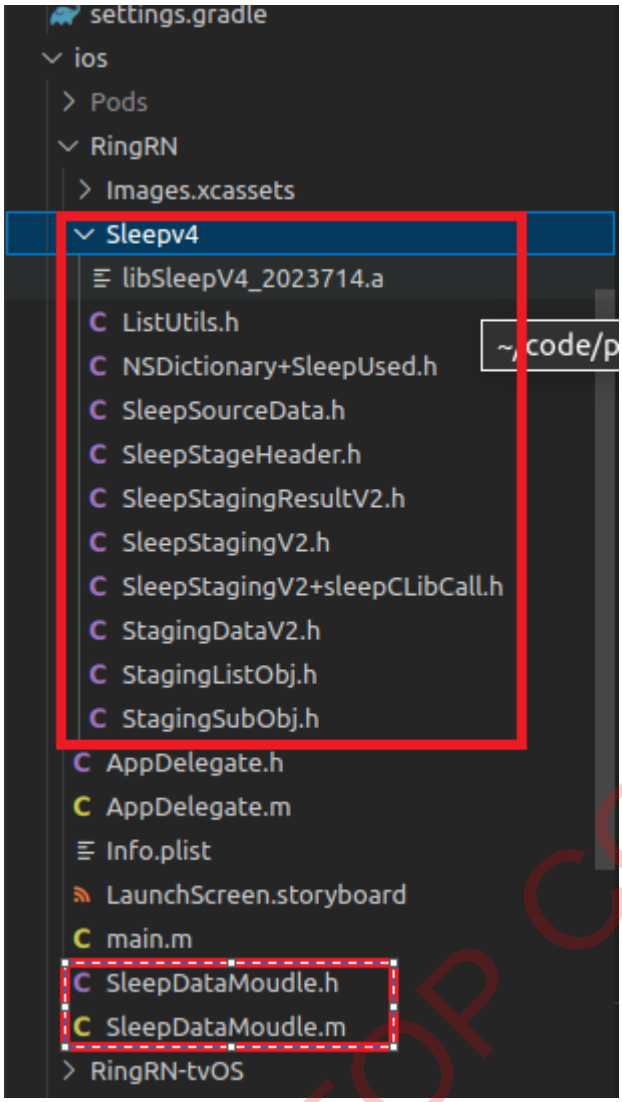


SleepDataModule.getIOSSleepData(Array,function)

睡眠数据算法（IOS）

```
SleepDataMoudle.getIOSSleepData(mArray.current, (error, result) => {
    var sleepTimeArray = [];
    for (let index = 0; index < result.length; index++) {
        const data = result[index];
        var lightTime = 0;
        var deepTime = 0;
        var remTime = 0;
        var wakeTime = 0;
        var napTime = 0;
        for (let index = 0; index < data.stagingList.length; index++) {
            const element = data.stagingList[index];
            switch (element.stagingType) {
                case "NREM1":
                    lightTime += element.endTime - element.startTime
                    break
                case "NREM3":
                    deepTime += element.endTime - element.startTime
                    break;
                case "REM":
                    remTime += element.endTime - element.startTime
                    break;
                case "WAKE":
                    wakeTime += element.endTime - element.startTime
                    break;
                case "NAP":
                    napTime += element.endTime - element.startTime
                    break;
            }
        }
        sleepTimeArray.push({
            deepSleep: `deepTime= ${Math.floor(deepTime / (1000 * 60 * 60))}h${Math.floor((deepTime % (100
            lightTime: `lightTime= ${Math.floor(lightTime / (1000 * 60 * 60))}h${Math.floor((lightTime %
            remTime: `remTime= ${Math.floor(remTime / (1000 * 60 * 60))}h${Math.floor((remTime % (1000 *
            wakeTime: `wakeTime= ${Math.floor(wakeTime / (1000 * 60 * 60))}h${Math.floor((wakeTime % (100
            napTime: `napTime= ${Math.floor(napTime / (1000 * 60 * 60))}h${Math.floor((napTime % (1000 *
            startTime: formatDateTime(data.startTime),
            endTime: formatDateTime(data.endTime)
        })
    }
    setSleepTime(sleepTimeArray);
})
```

ios目录需要添加的代码



注意.a静态库需要在xcode的build Phases=>Link Binary With Libraries中添加

11) SDK.calcOxygenSaturation(SleepTimeArray, Array)

血氧饱和度算法

```
function getOxygenSaturation() {
    if (mArray.current.length != 0) {
        if (mSleepTimeArray.current.length == 0) {
            mSleepTimeArray.current = SDK.calcSleepTime(mArray.current);
        }
        let array = SDK.calcOxygenSaturation(mSleepTimeArray.current, mArray.current);
        setOxygenSaturation(array)
    } else {
        showDialog();
    }
}
```

参数:

SleepTimeArray:睡眠数据

Array:历史数据

返回值:

startTime:睡眠开始时间

endTime:睡眠结束时间

oxygen:血氧饱和度值

12）呼吸率计算

```
function calcRespiratoryRate(timeStamp = -1) {
    initSleepData();
    var result=SDK.calcRespiratoryRate(mSleepTimeArray.current, mArray.current,timeStamp);
    if(result?.type == 'number'){
        setRespiratoryRate(result.respiratoryRate+" BPM")
    }else if(result?.type=='Array'){
        var arr=result?.result;
        var result = ""
        for (let index = 0; index < arr.length; index++) {
            const element = arr[index];
            result += (index + 1) + "group startTime" + formatDateTime(element.timeSlot.startTime) + "-> endTime" +
        }
        setRespiratoryRate(result);
    }
}
```

13) SDK.calcRestingHeartRate(HrArray, timeStamp);

静息心率算法

```
function calcRestingHeartRate(timeStamp = -1, refresh = true) {
    if (mHrArray.current.length != 0) {
        var restingHeartRate = SDK.calcRestingHeartRate(mHrArray.current, timeStamp);
        if (typeof restingHeartRate == 'number') {
            if (refresh) {
                setRestingHeartRate(Math.floor(restingHeartRate) + " BPM");
            }
            return restingHeartRate;
        } else if (restingHeartRate instanceof Array) {
            let result = ""
            for (let index = 0; index < restingHeartRate.length; index++) {
                const element = restingHeartRate[index];
                var ts = formatDateTime(element.ts, false);
                var data = Math.floor(element.data);
                result += ts + " restingHeartRate" + data + " BPM "
            }
            if (refresh) {
                setRestingHeartRate(result);
            }

            return restingHeartRate;
        }
    } else {
        showDialog();
    }

}
```

参数:

HrArray:历史心率和时间戳的数组

timestamp:时间戳（值为-1的时候计算，所有睡眠时间段的平均心率，值当在某个睡眠时间段区间，计算当前睡眠段的静息心率）

14）心率沉浸计算

```
function getHeartRateImmersion(timeStamp = -1) {
    initSleepData();
    var result=SDK.calcHeartRateImmersion(mSleepTimeArray.current,mArray.current, mHrArray.current, timeStamp)
    if(result.type=='number'){
        setHeartRateImmersion(result+"%");
    }else if(result.type=='Array'){
        var data="";
        var arr=result.result
        for (let index = 0; index < arr.length; index++) {
            const element = arr[index];
            data+=element.time+" heartRateImmersion:" +element.restingHeartRate+"% "
        }
        setHeartRateImmersion(data);
    }
}
```

15）OEM认证

```
if(data.switchOem&&startOem.current){
    startOem.current=false
    //Start oem certification
    SDK.startOEMVerify((cmd,data)=>{
        sendData(cmd,data)
    })
}
```

先调用获取设备信息1的接口后获取设备信息1的OEM开关状态，当OEM开关状态为1开启时，调用startOEMVerify开始OEM认证，如果戒指有打开OEM认证，必须先进行OEM认证后，其他功能才能正常运行。

16）心率测量时间设置

```
sendData(setHrTime, heartRateTime < 10 ? 10 : heartRateTime > 180 ? 180 : heartRateTime)
```

心率测量时间范围为10-180秒之间

17）OTA

```
const startUpdate = () => {
    console.log(`startUpdate= memoryType=${memoryType} `)
    SDK.SuotaManager.setMemoryType(memoryType);
    SDK.SuotaManager.setType(SDK.SuotaManager.TYPE);
    if (memoryType === OtaUtil.MEMORY_TYPE_I2C) {
        try {
            if (i2cAddr == 0) {
                i2cAddr = parseInt(OtaUtil.DEFAULT_I2C_DEVICE_ADDRESS, 10);
            }
            SDK.SuotaManager.setI2CDeviceAddress(i2cAddr);
        } catch (nfe) {
            showDialog("I2C Parameter Error,Invalid I2C device address.");
            return;
        }
    }
    let fileBlockSize = 1;
    if (SDK.SuotaManager.getType() === OtaUtil.TYPE) {
        try {
            fileBlockSize = Math.abs(parseInt(blockSize.toString(), 10));
        } catch (nfe) {
            fileBlockSize = 0;
        }
        if (fileBlockSize === 0) {
            showDialog("Invalid block size,The block size cannot be zero.");
            return;
        }
    }
    console.log(` fileBlockSize=${fileBlockSize} `)
    SDK.SuotaManager.fileSetType(SDK.SuotaManager.TYPE, bytes.current);
    SDK.SuotaManager.setFileBlockSize(fileBlockSize, OtaUtil.getFileChunkSize());
    var intent = {
        action: OtaUtil.ACTION_BLUETOOTH_GATT_UPDATE,
        step: 1
    }
    OtaUtil.otaStep(intent)
}
```

戒指项目默认选中SPI进行OTA升级，参数默认配置不需要更改

18）运动模式

```
sendData(SDK.SendCmd.SetSportModeParameters, {
    switch: sportModeSwitch,
    timeInterval: sportModeTimeInterval,
    duration: sportModeTimeDuration
})
```

switch:1：开启运动模式 ，0：关闭运动模式

timeInterval:数据采集时间间隔（心率，步数等）单位秒（10-180s）

duration:运动模式持续时间单位分钟(5-180min)

运动模式的数据可以通过调用获取历史记录接口获取

19）卡路里

```
const caloriesCalculation = () => {
    if (personalHeight) {
        let step = endStep - startStep
        if(step>0){
            let calories = SDK.caloriesCalculation(personalHeight, step, strengthGrade);
            setCalorie(calories)
        }
    }
}
```

personalHeight:戒指佩戴者的身高，单位cm

step:戒指佩戴者运动的步数

strengthGrade:运动强度分为3个级别：高强度运动为0.1，中等强度运动为0.08，低强度运动为0.05

20）红外源数据

```
sendData(SDK.SendCmd.setHealthPara, {
    samplingRate: device2Value.samplingRate, switch: 1
})}
```

```
const irResouceListener = {
    onResult: (data) => {
        waveList.current.push(...data);
        if (waveList.current.length >= 600 && drawWaveStart.current) {
            drawWaveStart.current = false
            startDraw()
        }
    }
}
```

samplingRate:采样率 通过设备信息2接口获取

switch:1:打开 ， 0:关闭

irResouceListener:波形数据接收

红外源数据打开后，打开血氧测量就能获取血氧 IR 的源数据