目录

[SDK包功能： 2](#_Toc30860)

[sdk代码 2](#_Toc32041)

[bsSdk 3](#_Toc28325)

[MessageControl包 3](#_Toc6251)

[MessageController 3](#_Toc27605)

[MessageProtocol 5](#_Toc23827)

[Module包 5](#_Toc28115)

[Bean 5](#_Toc22873)

[ArfcnConvert 5](#_Toc7144)

[AtCmd 6](#_Toc27015)

[BaseNRBean 6](#_Toc4792)

[BsBeam 6](#_Toc11055)

[DeviceWifi 7](#_Toc1821)

[InvalidLoc 7](#_Toc7945)

[LocBean 8](#_Toc16657)

[LteBand 8](#_Toc5826)

[NrBand 9](#_Toc14015)

[PwrCtl 9](#_Toc13894)

[WifiBean 9](#_Toc4395)

[Module 10](#_Toc19524)

[CdmaInfo 10](#_Toc28435)

[GsmInfo 11](#_Toc9221)

[LTE\_CT 12](#_Toc8192)

[LTE\_W 13](#_Toc17101)

[LteCInfo 14](#_Toc8342)

[LteMInfo 14](#_Toc21850)

[LteMInfoT 14](#_Toc10915)

[LteTInfo 14](#_Toc30267)

[LteTInfoT 14](#_Toc32391)

[LteUInfo 14](#_Toc31772)

[LteUInfoT 14](#_Toc30943)

[NR\_5G 14](#_Toc14140)

[NR\_Collecter 14](#_Toc24229)

[NR\_M 14](#_Toc32666)

[NR\_POINT 14](#_Toc23323)

[NR\_QS 15](#_Toc16809)

[NR\_T 15](#_Toc17978)

[NR\_U 15](#_Toc25994)

[NrCInfo 15](#_Toc16257)

[NrMInfo 15](#_Toc3350)

[NrTInfo 15](#_Toc18520)

[NrUInfo 15](#_Toc18242)

[Upgrade 15](#_Toc8800)

[WcdmaInfo 15](#_Toc2983)

[WifiModule 15](#_Toc6499)

[Util包 15](#_Toc10027)

[Util 16](#_Toc24014)

[BsSdk 16](#_Toc19237)

[BsService 16](#_Toc31695)

[Native 17](#_Toc14962)

[SerialPort 18](#_Toc7313)

[SerialPortController 18](#_Toc11344)

[SerialPortFinder 19](#_Toc27179)

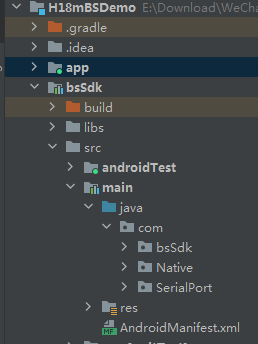
# SDK包功能：

获取数据解析数据

要看每个模块如何获取如何解析

# sdk代码

看代码主要看有什么工具，能做什么，怎么实现的

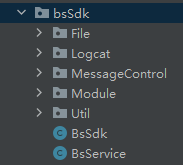


bsSdk 主包

Native 电源开启关闭，JNI对接

SerialPort 串口相关

# bsSdk



File 文件相关操作

Logcat 日志文件相关

MessageControl 信息接口和协议

Module 各种模块实现和实体类

Util 工具类

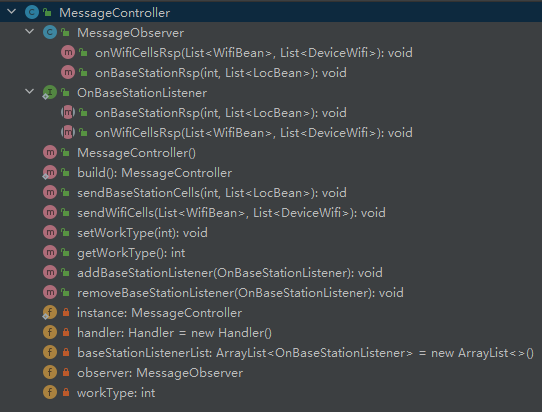
BsSdk 开启关闭底层服务

BsService 线程服务类

# MessageControl包

# MessageController

主要处理基站和Wifi数据的获取和发送



**属性：**

instance

当前类MessageController引用

handler

Handler的实例

baseStationListenerList

持有接口实例列表

observer

响应类引用

workType

工作模式引用

**方法：**

OnBaseStationListener

创建一个调用基站数据的接口

MessageObserver

响应类，调用接口的方法（回调的实现）  
MessageController

初始化响应类和工作模式（旧采集方式或强搜）

Build（）

创建MessageController实例（通过锁保证同步）

sendBaseStationCells（int， List<LocBean>）

同步开启线程调用响应类中的onBaseStationRsp（）并传入数据

sendWifiCells

同步调用响应类中的onWifiCellsRsp（）并传入数据

setWorkType

配置采集数据格式，当格式和当前的不相等时开启对应的模块

getWorkType

返回当前格式

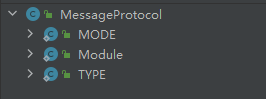
addBaseStationListener（OnBaseStationListener）

向接口列表内添加接口

removeBaseStationListener（OnBaseStationListener）

移除接口列表的某个接口

# MessageProtocol



实体类，定义模式数据和索引

MODE

模式：强搜和旧采集方式

Module

模块数据头

TYPE

模块数据处理索引：运营商和网路对应的数字

# Module包

模块

# Bean

实体类

# ArfcnConvert

频点转换类

parse5to4

5G频点转4G频点

calaMidFrq

根据5G频点算中心频率

calcGSCN

根据中心频点算GSCN

getGSCN

根据频点获取GSCN

# AtCmd

**属性：**

Instance

持有本类AtCmd引用

STM32\_VER

STM32 版本号

UART\_WIFI

串口？

**方法**

AtCmd

初始化

build

使用锁同步创建实例

sendAndReadRsp

使用串口发送并接收反回值

sendAtCmd

使用串口发送指令

**内部类**

PWR

标记一些设备对应的数字

NR

5G的一些功能对应的数字

LTE

4G的一些功能对应的数字

NR\_QS

强搜的功能？对应的数字

# BaseNRBean

5g的属性实体类

# BsBeam

基站实体类

beam\_num

ssb\_id

rsrp

参考信号接收功率

# DeviceWifi

Wifi设备实体类

探针数据默认格式：

A8:15:4D:5F:BD:36|00:08:22:29:11:4F|02|04|1|-94|simdo123|0

其中：

第一列：08:00:23:6F:58:CE

第二列：12:F8:56:FF:6F:56

第三列：00 代表Frame的大类

第四列：08 代表Frame的小类

第五列：3 代表Chanel

第六列：信号强度，据此可算出大概的距离

第七列：WIFI名称

第八列：手机状态，0代表正常，1代表黑屏

**属性**

wifiName

wifiMac

deviceMac

bigFrame Frame的大类

smallFrame Frame的小类

chanel

Rssi 信号强度

deviceWork 手机状态 0代表正常，1代表黑屏

distance

is5g

**方法**

calcDistance（Rssi）

距离公式

# InvalidLoc

**属性**

Instance 引用本类InvalidLoc实例

arfcnList

**内部类**

ArfcnPci实体类 持有type，Arfcn和Pci

**方法**

InvalidLoc

初始化

Build

同步创建实例

Init

清除arfcnList

addArfcn(int type, String arfcn, String pci)

当列表内没有相同时加入列表

isInvalid(int type, String arfcn, String pci)

判断列表内是否已存在

reInit(int type)

删除列表中type相同的所有项

# LocBean

**属性**

tac // 大区，CDMA:SID WCDMA|GSM|TD显示为：LAC

Eci // 小区，CDMA: NID NR[5G]: NCI, WCDMA|GSM|TD：CID

arfcn

pci // GSM:BSIC CDMA: PN WCDMA: PSC TD: CPI

rx

pri // LTE|NR 频点优先级 // CDMA: BID

simCard // 是否插卡

type // 值为：MessageProtocol.iModule

main // 是否为主区

lock

alarm

isRadix

beamList List<BsBeam>

ecio

**方法**

setBeamList

# LteBand

**属性**

NMM\_NUM\_RADIO\_BAND

nmm\_earfcn\_to\_freq\_lut

**方法**

earfcn2band 好像是选出在这个范围内的标签？

# NrBand

**属性**

NMM\_NUM\_RADIO\_BAND

nmm\_earfcn\_to\_freq\_lut

**方法**

earfcn2band 同4g

# PwrCtl

电源管理控制类

**属性**

DELAY\_TIMER 开机之后启动的延迟时间

Instance 持有本类PwrCtl的引用

**方法**

build

同步创建实例

GsmPwrCtl（boolean）

移动2G模块电源控制，调用本地方法然后启动延迟，传入true或false以下方法同本方法（reset就是方法改成reset重启）

WcdmaPwrCtl

CdmaPwrCtl

LteMpwrCtl

LteUpwrCtl

LteTpwrCtl

NrMpwrCtl

NrMpwrReset

NrUpwrCtl

NrUpwrReset

NrTpwrCtl

NrTpwrReset

WifiPwrCtl

WIFI模块电源开，无延迟

modulePwrCtl

模块电源控制，所有模块电源开或关

# WifiBean

**属性**

ssid

mac

chanel

rssi

capabilities // 安全类型 模块无此数据[6]，本机有[0~5]

distance

is5g

**方法**

WifiBean

初始化，is5g是通过chanel来判断的，distance是通过rssi来计算的

calcDistance

根据公式计算距离

# Module

模块

# CdmaInfo

**内部类**

T内部有个AT\_CCINFO属性

**属性**

Instance

本类引用

curList List<LocBean>

rstModule

atCnt

refreshCnt

cmdType

enableAt

AT\_CCINFO

AT+xxxx的一般是指令

**方法**

CdmaInfo

build 同步创建实例

Init 初始化

initInfo 清空列表

sendAtCmd

是否向串口发送指令、开启电池、向基站发送指令

parseData

解析数据、判断是否向基站发送指令

parseFirstLocInfo

解析第一次的数据，在里面进行各种属性的解析工作，最后添加进列表

parseLocInfo

解析其他的数据，比第一次的少几个

getRandomRxStr

得到随机的Rx串

getList

返回列表

# GsmInfo

**内部类**

T内部有两个属性

AT\_GSM\_NO\_CARD\_INIT

AT\_GSM\_NO\_CARD

**属性**

Instance

unicomTmp List<LocBean>

moblieTmp List<LocBean>

moblieInfo List<LocBean>

unicomInfo List<LocBean>

errorCnt

refreshCnt

refreshCntM

rstModule

atCnt

enableAt

ugsmIsInVirtualCnt

mgsmIsInVirtualCnt

lastUgsmMsg

lastMgsmMsg

cmdType

// A4:GSM 模块初始化【插卡与未插卡】

AT\_GSM\_PWF\_OFF

AT\_GSM\_NO\_CARD

AT\_GSM\_NO\_CARD\_INIT

**方法**

GsmInfo

build

init

initVirtualPara

创建lastUgsmMsg和lastMgsmMsg的空实例

initGsmInfo

根据mobileInfo创建lastMgsmMsg和lastMgsmMsg实例

sendAtCmd

根据cmdType判断需要向串口发送什么指令，如果电池没开启要开启电池

parseData

解析数据、判断是否向基站发送指令

parseSim800AInfo

解析数据加入unicomTmp和moblieTmp列表

sortLocationData

无卡时对采集出来的数据做RX排序，信号最强的放最前面

checkIsInVirtualMode

检查是否在虚拟模式

getListM

getListU

下面只解释看得懂或重要的

# LTE\_CT

**属性**

instance

curList

rstModule

lcData

cmdType

inc

cpsiOk

smemiOk

resetOk

AT\_CNLSA // 允许锁定运营商

AT\_CNMP // 查询及锁定工作模式

AT\_CPSI

AT\_CCINFO

AT\_RTRECF\_Q //查询系统当前的RTRECF [重启生效]

AT\_RTRECF\_S

AT\_CRESET // 重启

AT\_SMEID\_Q // 查询系统当前的MEID,若正确则不需要设置

AT\_SMEID\_S

AT\_ESN\_Q //查询系统当前的ESN

AT\_ESN\_S

AT\_AKEY\_Q //查询系统当前的AKEY

AT\_AKEY\_S

// 以下AT命令设置必须要求AT+RTRECF？返回值是1.

AT\_MCC\_Q //查询MCC

AT\_MCC\_S

AT\_MNC\_Q //查询MNC

AT\_MNC\_S

AT\_IMSI\_Q //查询IMSI

AT\_IMSI\_S

AT\_CFSERRFATAL // 恢复出出厂设置

AT\_CFUN0 // 飞行

AT\_CFUN1

AT\_SIMCOMATI

AT\_CNBP // 查询当前模块配置参数

AT\_CNBP\_S

**内部类**

TYPE里面定义了对GSM的属性

# LTE\_W

handleNeiberData（）

如果curList为空，将saveData的tac和eci赋给curList

AT\_CCINFO

// 实时读出GSM数据

AT\_RST\_ARFCN

// 配置自动模式【锁频后】

AT\_RST\_ARFCN\_PSC

// 配置自动模式 【ARFCN & BSIC后】

AT\_CNBP

// 查询当前模块配置参数

AT\_CFSERRFATAL

// 恢复出出厂设置

AT\_CPOF

// 关机

# LteCInfo

handleSimdoRecordData

修改mStrHnxaLc的值

# LteMInfo

# LteMInfoT

# LteTInfo

# LteTInfoT

# LteUInfo

# LteUInfoT

# NR\_5G

rstModule// 重启模块标志

opType; // 制式类型

opCount, changeCount; // 同一个制式采集数据计数

// 锁频段后第一次回来的数据, 1：联通， 2：电信， 0\3\4：其它

private int dataType = 0;

# NR\_Collecter

rstModule = 0// 重启模块标志

opType; // 制式类型

opCount; // 同一个制式采集数据计数

# NR\_M

# NR\_POINT

# NR\_QS

# NR\_T

# NR\_U

# NrCInfo

# NrMInfo

cardStatus // 无卡 = 0 ，有卡 = 1， 未开机 = 2

# NrTInfo

# NrUInfo

# Upgrade

# WcdmaInfo

# WifiModule

OnWifiScanListener//小区扫描相关接口

# Util包

BandUtil

DataUtil 数据转换工具类

DateUtil 日期工具类

FakeUtil 检测伪基站工具类

IOUtil 读写工具类

MD5

hexDigits 16进制摘要

MessageDigest 信息摘要

Util

# Util

Delay 延迟

isNumeric 字串是否为数字

onlyNumeric

randomData 产生随机数

handleRsrp

randomUInt

randomInt

getSwVersion 获取软件版本信息

getHwVersion 获取硬件版本信息

checkData

sortList\_op

isContainUpperCase

randomRsrp

removeDuplicate

# BsSdk

Build

同步创建实例

Init

打开各模块电源及初始化

同步创建实例

开启服务

延迟执行

initNative

设备初始化、串口初始化

initPara

初始化所有模块

startModule

开启电池、5g移动、5g电信、5g强搜

onDestory

关闭资源

openBsService

启动与结束底层服务

getContext

# BsService

**内部类**

LocThread

开启线程、发送指令

**方法**

onBind

onCreate

startThead

stopThead

onDestroy

# Native

jniUartClose

关闭串口通讯

jniUartInit(int baudrate)

串口初始化，波特率：115200

jniUartDataWrite(int uart\_port, int[] commond, int length)(int uart\_port, int[] commond)

串口数据传送：第三方客户可不关注 uart\_port : 串口

jniUartDataRead

串口数据读取 uart\_port : 串口

LcInit

设备初始化

LcClose

关闭设备

openNrPwr

开5G模块电源

setLcPwr

模块电源控制开关统一处理

setWifiMode

2.4 |5G WIFI模块选择 【UART: 0=2.4G;1=5G】

setModulePwr

/\*\*

\*\* GSM = 0

\*\* CDMA = 1

\*\* WCDMA = 2

\*\* LTE-M = 3

\*\* LTE-U = 4

\*\* LTE-T = 5

\*\* NR-M = 6

\*\* NR-U = 7

\*\* NR-T = 8

\*\* WIFI\_ONLY = 9

\*\*/

setModuleRst

/\*\*

\*\* GSM = 0

\*\* CDMA = 1

\*\* WCDMA = 2

\*\* LTE-M = 3

\*\* LTE-U = 4

\*\* LTE-T = 5

\*\* NR-M = 6

\*\* NR-U = 7

\*\* NR-T = 8

\*\*/

BarometerOpen

BMP280以下都是

BarometerClose

BarometerGetPress

BarometerGetTemp

# SerialPort

SerialPort

开启底层文件

getInputStream

getOutputStream

open

Close

# SerialPortController

内部类

ReadThread 线程类

调用UartReadMsg方法读取数据

SerialPortController

创建线程

build

getSerialPort

创建一个SerialPort

closeSerialPort

onStop

sendAtCmd

发送指令到串口

sendUpdata

UartReadMsg

根据信息读取数据

# SerialPortFinder

**内部类**

**Driver**

类似于驱动

Driver

getDevices

获取设备信息

getName

**方法**

getDrivers

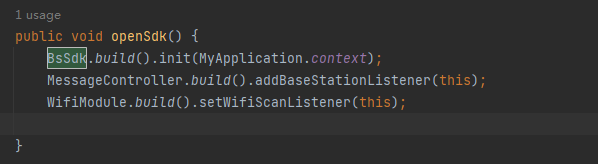
获取驱动

getAllDevices

getAllDevicesPath

# 操作关系

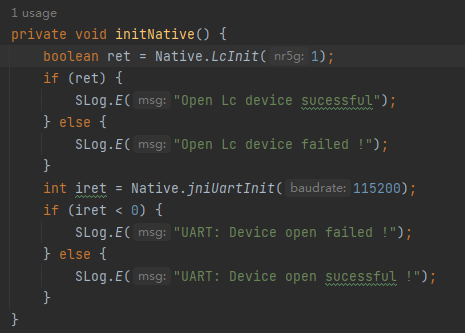
1、使用：调用BsSdk初始化方法、设置基站监听器、设置wifi监听器。



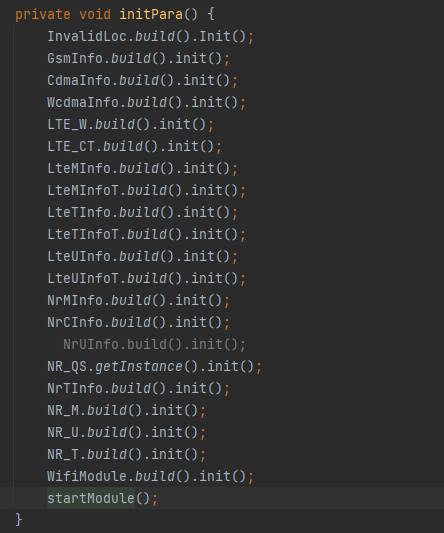
1. 在BsSdk的init中开启本地服务、串口控制器、频点采集底层服务、初始化模块



1. 在initNative中初始化设备和串口



1. 在initPara中初始化模块开启电池



# NR\_M和NrMInfo的区别：

NR\_M像是实体类、NrMInfo像是实现类

# SerialPortController

sendAtCmd负责发送指令到串口

其他实现类在频点采集底层服务BsService中统一调用实现类来发送指令，然后实现类再通过串口控制器的sendAtCmd发送指令到串口。

# 回调接收数据

1. 通过MessageController的内部类MessageObserver
2. 回调的实现是通过观察者模式来实现的。
3. MessageObserver类充当了一个观察者的角色，负责观察和接收消息。而MessageController作为观察者的一部分，负责处理和响应特定类型的消息。
4. 具体而言，MessageObserver类内部有一个存储OnBaseStationListener监听器的列表baseStationListenerList。当MessageController注册为OnBaseStationListener监听器时，会将其添加到该列表中。
5. 当有基站数据可用时，MessageObserver中的onBaseStationRsp方法会被调用。在该方法中，会遍历baseStationListenerList列表，并依次调用每个OnBaseStationListener监听器的onBaseStationRsp方法，将基站数据传递给它们。
6. 这样，通过观察者模式，MessageController作为观察者通过注册到MessageObserver中，可以被动接收并处理来自MessageObserver观察者的基站数据回调。当基站数据可用时，MessageObserver会主动调用MessageController中的对应方法，实现回调的效果。
7. 总结起来，回调的实现是通过观察者模式，观察者(MessageObserver)监测到特定事件 (基站数据可用)，然后主动调用注册的观察者(MessageController)的回调方法(onBaseStationRsp)，将数据传递给观察者进行处理。这种方式实现了事件的触发和消息的传递，使得MessageController能够及时获取到基站数据。

总结：

MessageController中有发送数据的方法，在方法内部调用observer.onBaseStationRsp将发送的数据传输给observer。发送数据的方法则在实现类中调用。

MessageController内部持有一个监听器列表baseStationListenerList，每当MessageObserver接受到基站的数据时就会遍历监听器列表并调用listener.onBaseStationRsp将数据传给监听器实现回调。

1. listener中定义回调方法
2. Observer中定义方法接收数据并调用listener的方法传递下去
3. Controller中定义方法发送数据在内部调用observer的方法将数据传递给observer
4. 实现listener并重写回调方法则可以接收到数据

回调实现：

1. 定义一个接口
2. 创建一个观察者类，在内部调用接口方法传递数据
3. 实现接口，重写方法获取数据
4. （还可以在其他地方创建观察者对象和实现接口的类对象，然后注册到观察者对象内，之后使用观察者触发事件）

例：

MyObserver observer = new MyObserver();

MyCallback callback = new MyCallbackImpl();

observer.registerCallback(callback);

// 示例：触发事件1

observer.performEvent1();

// 示例：触发事件2，传递数据

int data = 123;

observer.performEvent2(data);

// 注意：在使用完回调后，记得取消注册

observer.unregisterCallback(callback);