NativeJs使用Gatt方式连接蓝牙

一、蓝牙功能说明

蓝牙功能简单说来就是：1.蓝牙设备的连接。2.蓝牙设备的数据交换（这部分没有太多深入研究，直接使用之前的代码）。

> 目前项目中接触到的蓝牙连接分为三种类型：
>
> - 1.`Gatt方式连接`——代码中使用`gatt`方式直接连接
> - 2.`createBond-pin码方式连接`——现实场景中是连接蓝牙设备时，输入蓝牙设备的`pin码`进行连接。这个功能可以使用代码实现
> - 3.`createBond-setPairingConfirmation方式连接`——现实场景中是连接蓝牙设备时，弹窗确认蓝牙匹配进行连接。这个功能需要是系统应用才可以实现。

前面两种都是已知蓝牙设备的特定信息可以直接连接上。

后面这种【确认配对】的方式因为没有蓝牙的信息需要有Android系统权限【android.permission.BLUETOOTH_PRIVILEGED】但是这个权限是需要【系统应用】才可以拥有的，所以这种方式连接不上。

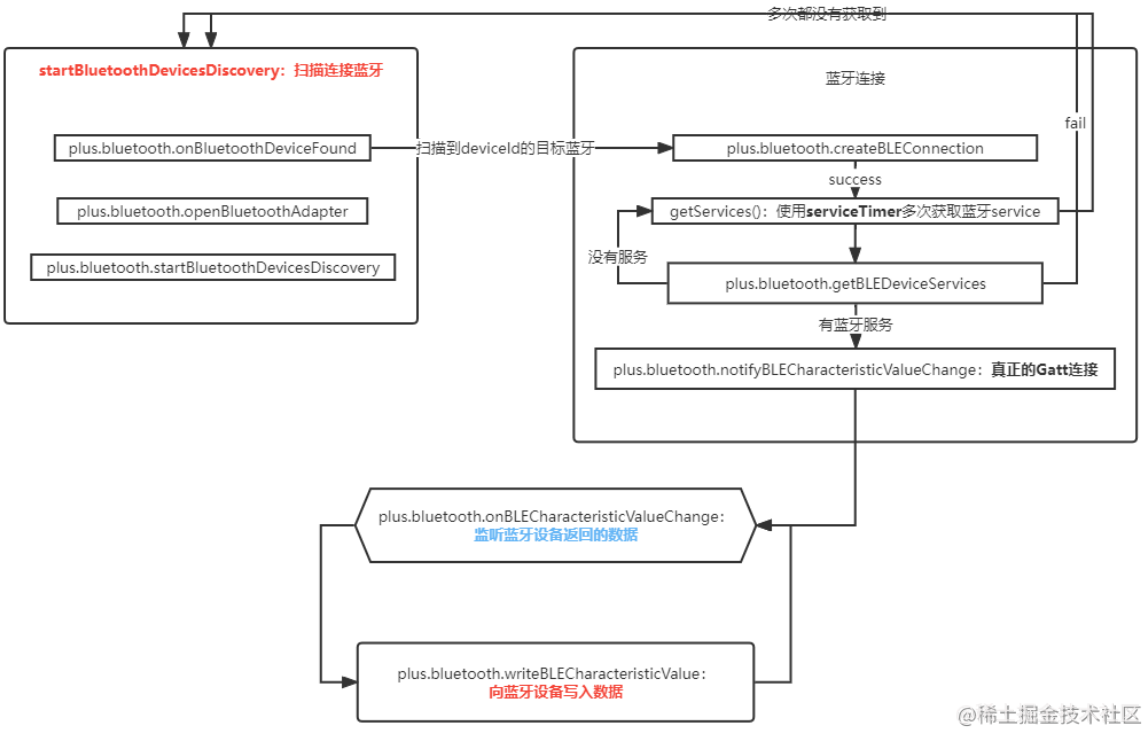
二、蓝牙Gatt方式连接过程

在连接前就知道了目标蓝牙设备的MAC地址

- 1.应用具有【定位权限】(并且需要Android设备打开系统定位功能)
- 2.判断Android蓝牙功能是否打开，没有打开则打开蓝牙
- 3.使用Android蓝牙扫描功能发现附近蓝牙设备
- 4.判断扫描到的蓝牙设备的MAC地址和目标蓝牙设备的MAC地址匹配，使用Gatt方式连接目标蓝牙设备
- 5.Android分段分批使用Gatt方式发送数据到蓝牙设备，每次发送后蓝牙设备也会通过Gatt方式返回数据，Android将返回的数据过滤拼接使用。
- 6.重复步骤`5`的操作3次左右，整个数据传输完成。

三、项目中的蓝牙连接代码分析：

核心功能分析：



> `BlueUtils`(blue-utils.js)：主要作用是Android蓝牙打开、蓝牙设备连接、蓝牙数据传输

> `Request`(request.js)：主要作用是请求后台接口获取`bwnr`报文内容字符串及发送`bwnr`

> `BluePlus`(blue-plus.js)：主要作用是提供方法供页面使用及数据回调到页面。提供的方法包括：使用`BlueUtils`和`Request`进行蓝牙连接，蓝牙数据传输等。将`bwnr`分段处理及回来的数据处理。

1.`BlueUtils`代码分析：

0.对外提供的方法有：

- 0.`constructor(deviceId)`
  - 蓝牙设备的`MAC地址`需要通过参数传入（本项目中这里是写死的）
- 1.`startBluetoothDevicesDiscovery({readyBack,valueBack})`
  - `readyBack`是连接到目标蓝牙后的回调
  - `valueBack`是接收到蓝牙设备返回的数据回调
- 2.`writeValue(value)`
  - 向目标蓝牙设备写入数据，需要在上面的`readyBack`之后使用。

1.部分代码介绍：

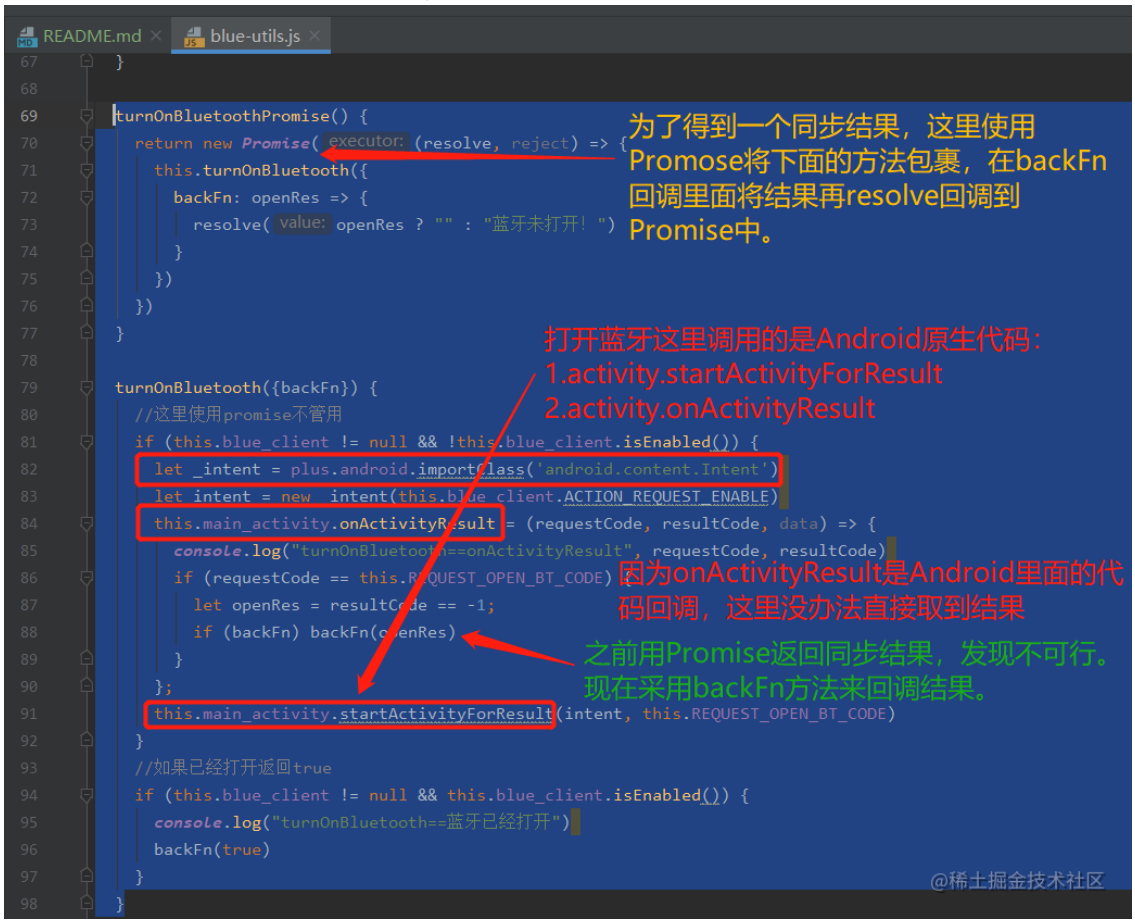
```
class BlueUtils {
    constructor() {...}

    canBlueToothUse(){//蓝牙硬件是否可用
        1.判断当前设备是否支持蓝牙，支持蓝牙则走下面的逻辑!
        this.blueToothError = await this.openLocationPermission()
        这里就可以使用await接收到【申请权限方法】的同步结果。
        2.判断定位权限是否存在，不存在则去请求权限，存在则走下面的逻辑!
        3.判断蓝牙功能是否打开，没打开则去申请打开，打开则走下面的逻辑!
        4.上面的功能在不满足的情况下就提示错误信息，并返回false结果。
    }
```

```
    //这个方法的作用就是统一日志输出。正常情况下只需要传入fn方法名及e有用信息两个参数即可
    //如果在【Promise】对象中使用时，先输出console日志，然后再通过【resolve】方法将【e有用
信息】回调到【Promise】对象中。
    promiseBackSuccess(fn, e, back = null) {
      console.log(fn + "---success", e)
      if (back) back(e)
    }

    //上面的方法在这里有使用，目的就是在【Promise】中先输出日志，然后再将结果返回去。
    openLocationPermission() {//申请定位权限
      return new Promise((resolve, reject) => {

plus.android.requestPermissions(['android.permission.ACCESS_FINE_LOCATION'],
            e => {
              if (e.deniedAlways.length > 0)
this.promiseBackFail("requestPermissions_ACCESS_FINE_LOCATION", "定位权限被拒绝！",
reject)
              if (e.deniedPresent.length > 0)
this.promiseBackFail("requestPermissions_ACCESS_FINE_LOCATION", "定位权限被拒绝！",
reject)
              if (e.granted.length > 0)
this.promiseBackSuccess("requestPermissions_ACCESS_FINE_LOCATION", "", resolve)
            },
            err => {
              this.showToast("申请权限出错！")
              this.promiseBackFail("requestPermissions_ACCESS_FINE_LOCATION",
JSON.stringify(err), reject)
            });
      })
    }

    //开启蓝牙扫描-连接方法
    startBluetoothDevicesDiscovery(){
        //1.判断蓝牙功能是否可用，不可用去申请相关权限或提示错误信息
        let blueToothUse = await this.canBlueToothUse()
        if (!blueToothUse) return
        //2.以下都是调用【plus的蓝牙扫描、扫描监听、连接方法】的完整流程
        this.listenBlueToothAdapter()【扫描监听放在扫描的前面】
        let openRes = await this.openBlueTooth()
        let startDisRes = await this.startDiscovery()
    }

    listenBlueToothAdapter(){
        plus.bluetooth.onBluetoothDeviceFound(e => {
            if (deviceObj.deviceId == this.deviceId) {
                this.connectDevice(success=> //扫描到目标蓝牙，进行连接操作
                    this.getServices(success=> //sccess：连接成功，获取蓝牙服务
                        if (services.length > 0) {//success:获取服务成功，并且服务存
在：
                          this.listenValueChange(success=>
                            //到这里才是真正意义上的连接上蓝牙
                            this.readValueListener()//监听蓝牙返回的数据
                          )
                          this.clearServiceTimer()
```

```
                        }
                    )
                )
            }
        })
    }

}
```

在`canBlueToothUse`方法中的几个判断，都是使用`await`得到`Promoise`的同步结果，这样看起来更清晰。各个独立的方法只处理自己的业务逻辑即可，防止了多层嵌套的代码逻辑。

- 1.打开定位权限可以直接使用plus提供的方法`plus.android.requestPermissions`
- 2.打开蓝牙需要调用Android原生的api代码：



2.核心代码介绍：

```
class BlueUtils {

    //这里实际上是真正的Gatt连接
    listenValueChange() {
        plus.bluetooth.notifyBLECharacteristicValueChange({
          deviceId: this.deviceId,
          serviceId: '6E400001-B5A3-F393-E0A9-E50E24DCCA9E',
          characteristicId: '6E400003-B5A3-F393-E0A9-E50E24DCCA9E',
          success: e => {},
          fail: e => {}
        })
    }

    //Gatt方式监听蓝牙返回的数据
```

```
    readValueListener() {
      plus.bluetooth.onBLECharacteristicValueChange(e => {
        console.error("2>>>onBLECharacteristicValueChange >>>>>",
JSON.stringify(e))
        // console.error("2.1>>>onBLECharacteristicValueChange >>>>>", e.value)
        if (this.blueToothBack) this.blueToothBack(e)
      })
    }

    //Gatt方式向蓝牙设备写入数据
    writeValue(value) {
      return new Promise((resolve, reject) => {
        plus.bluetooth.writeBLECharacteristicValue({
          deviceId: this.deviceId,
          serviceId: "6E400001-B5A3-F393-E0A9-E50E24DCCA9E",
          characteristicId: "6E400002-B5A3-F393-E0A9-E50E24DCCA9E",
          value: value,
          success: e =>
this.promiseBackWarning("1>>>writeBLECharacteristicValue", "success", e,
resolve),
          fail: e => this.promiseBackWarning("writeBLECharacteristicValue",
"fail", e, reject)
        })
      })
    }
}
```

上面的`plus.bluetooth.notifyBLECharacteristicValueChange`和`plus.bluetooth.writeBLECharacteristicValue`方法都需要以下参数:

- 1.【deviceId】
- 2.【serviceId】
- 3.【characteristicId】这个值是不同的

> 这些特征值需要硬件方提供，或者使用bluetooth-android这个Android项目获取到。

2.`BluePlus`代码分析:

1.代码分析:

```
class BlueUtils {
    constructor(state) {
      this.blueUtils = new BlueUtils()
      this.dataUtils = new DataUtils()
      this.request = new Request()
    }

    async startBluetoothDiscovery() {
      await this.blueUtils.startBluetoothDevicesDiscovery({
        readyBack: async () => {//蓝牙准备就绪，请求接口，发送第一条bwnr
          console.log("startBluetoothDevicesDiscovery---readyBack")
          let bwnr = await this.request.firstRequest()
          console.log("<<<<<第1次请求报文内容", bwnr)
          if (bwnr) this.firstSendData(bwnr)
        },
```

```
        valueBack: async (e) => {
          console.error("3>>>startBluetoothDevicesDiscovery---valueBack",
this.dataUtils.buffer2hex(e.value))
          if (e.value != undefined) this.receiveBlueValue(e)
        }
      })
    }

    queryData = res => {
      //todo 处理页面逻辑
      if (res.ywbsm === '000104') {
        console.log('queryData---000104')
        this.firstSendData(res.bwnr)
      } else if (res.ywbsm === '500601') {
        console.log('queryData---500601')
        this.showToast("流程结束！")
      } else {
        console.log("?????????????queryData------------writeValue?????????????")
        this.blueUtils.writeValue(res.bwnr)
      }
    }
}
```

在调用`blueUtils.startBluetoothDevicesDiscovery`之后，

- 1.监听`readyBack`蓝牙连接成功后，请求后台接口获取`bwnr`字符串
- 2.执行`firstSendData`方法将`bwnr`分割后保存到`sendMsgArr`数组中分批次发送数据到蓝牙设备
    - 2.1每次发送数据到蓝牙设备后，会通过`valueBack`监听到蓝牙设备回传回来的数据，将回传的数据保存
- 3.判断`endFlag == 'end'`表示分批次传输`bwnr`报文内容数据完成。
- 4.进入下一次的`secondSendData`逻辑。
    - 4.1将`2.1`接收到的`receivedData`拼接当作参数传给接口再次获取`bwnr`数据
    - 4.2获取到接口数据后，执行`queryData`方法
        - 4.2.1判断`res.ywbsm === '000104'`继续执行`firstSendData`方法，进入到上面的`2-3-4`逻辑
        - 4.2.2判断`res.ywbsm === '500601'`整个数据交换逻辑完成，得到`bwnrJq = res.bwnr`充值结果。

这里的数据处理及交互逻辑都是复用之前的逻辑，所以代码感觉有点乱。
大概看一下即可，每个项目可能这部分的代码不太一样。