uniapp 安卓串口操作原生插件

目　录

使用说明 1
- 使用方法 1.1
- 更新日志 1.2

插件方法 2
- 获取所有设备列表 2.1
- 获取所有设备路径 2.2
- 设置串口 2.3
- 设置波特率 2.4
- 设置停止位 2.5
- 设置数据位 2.6
- 设置校验位 2.7
- 设置流控 2.8
- 设置延迟 2.9
- 开启串口 2.10
- 一键开启串口 v1.1.2 ..... 2.11
- 关闭串口 2.12
- 监听字节串口消息 2.13
- 发送字节数据 2.14
- 监听十六进制串口消息 2.15
- 发送16进制消息 2.16
- 监听ASCII消息串口消息 2.17
- 发送ASCII消息 2.18
- 串口是否开启 2.19
- 当前串口属性 2.20
- 取消监听 v1.2.0 ..... 2.21

粘包处理 3
- 固定长度粘包处理 3.1
- 可变长度粘包处理 3.2
- 首尾特殊字符粘包处理 3.3

使用方法

介绍

安卓串口操作原生插件，支持设置串口，波特率，停止位，数据位，校验位，流控以及延迟，支持粘包处理解决分包问题，支持多串口操作，支持无root操作

联系作者

关注微信公众号可联系作者

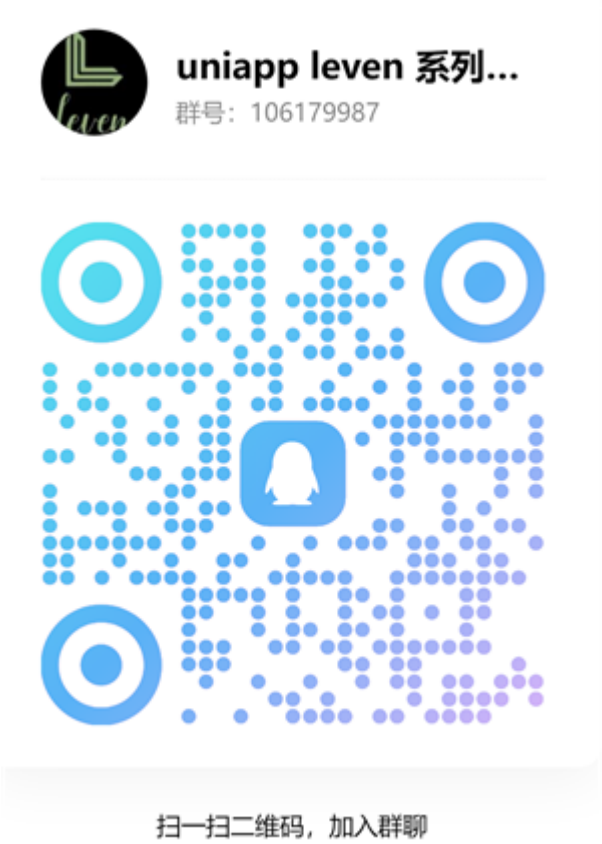
插件地址

https://ext.dcloud.net.cn/plugin?id=14244

示例文件和演示程序下载

使用方法

加QQ群下载，群号：106179987



用法

在需要使用插件的页面加载以下代码

```
const module = uni.requireNativePlugin("leven-serial-SerialModule");
```

多串口操作

如果您的项目涉及到多串口的话，可以多引入一些module，目前插件提供了以下几个module

```
const module = uni.requireNativePlugin("leven-serial-SerialModule");
const module2 = uni.requireNativePlugin("leven-serial-SerialTwoModule");
const module3 = uni.requireNativePlugin("leven-serial-SerialThreeModule");
const module4 = uni.requireNativePlugin("leven-serial-SerialFourModule");
const module5 = uni.requireNativePlugin("leven-serial-SerialFiveModule");
```

> 如果您不想这样使用，多串口可以直接使用 `const module = uni.requireNativePlugin("leven-serial-SerialModule");` 然后在使用

使用方法

> 时打开不用时关闭即可（即按需使用）
> 如果多串口一直保持打开状态的话就需要引入上面的module

页面内容

```
<template>
  <view>
    <view>
      <uni-card title="安卓串口操作原生插件">
        <view>
          <button type="primary" @click="logStr = ''">清空日志</button>
          <button type="primary" @click="getAllDevices">获取所有的设备列表</button>
          <button type="primary" @click="getAllDevicesPath">获取所有的设备路径</butto
          <button type="primary" @click="setPort">设置串口</button>
          <button type="primary" @click="setBaudRate">设置波特率</button>
          <button type="primary" @click="setStopBits">设置停止位</button>
          <button type="primary" @click="setDataBits">设置数据位</button>
          <button type="primary" @click="setParity">设置校验位</button>
          <button type="primary" @click="setFlowCon">设置流控</button>
          <button type="primary" @click="setDelay">设置延迟</button>
          <button type="primary" @click="open">开启串口</button>
          <button type="primary" @click="close">关闭串口</button>
          <button type="primary" @click="onListenerBytes">监听字节串口消息</button>
          <button type="primary" @click="sendBytes">发送字节数据</button>
          <button type="primary" @click="onListenerHex">监听十六进制串口消息</button>
          <button type="primary" @click="sendHex">发送16进制数据</button>
          <button type="primary" @click="onListenerText">监听ASCII串口消息</button>
          <button type="primary" @click="sendText">发送ASCII数据</button>
          <button type="primary" @click="isOpen">串口是否开启</button>
          <button type="primary" @click="getSerialProperty">当前串口属性</button>
        </view>
      </uni-card>
      <uni-card title="粘包处理">
        <view>
          <button type="primary" @click="setStaticLenStickPackage">固定长度粘包处理<
          <button type="primary" @click="setVariableLenStickPackage">可变长度粘包处理
          <button type="primary" @click="setSpecifiedStickPackage">首尾特殊字符粘包处
        </view>
      </uni-card>
    </view>
    <view>
      <uni-card class="uni-card-box" title="日志">
        <view><text style="font-size: 14px; flex-wrap: wrap;">{{logStr}}</text></v
      </uni-card>
    </view>
  </view>
</template>
```

使用方法

```
<script>
  const module = uni.requireNativePlugin("leven-serial-SerialModule");
  export default {
    data() {
      return {
        logStr: ""
      }
    },
    mounted() {
      // 动态开启应用权限
      // this.openAndroidPermission();
    },
    methods: {
      // 获取所有设备列表
      getAllDevices() {
        let that = this;
        module.getAllDevices(res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 获取所有设备路径
      getAllDevicesPath() {
        let that = this;
        module.getAllDevicesPath(res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 设置串口
      setPort() {
        let that = this;
        module.setPort("/dev/ttyS1", res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 设置波特率
      setBaudRate() {
        let that = this;
        module.setBaudRate(9600, res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 设置停止位
      setStopBits() {
        let that = this;
        module.setStopBits(1, res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 设置数据位
```

```
    setDataBits() {
      let that = this;
      module.setDataBits(1, res => {
        that.writeLog(JSON.stringify(res));
      })
    },
    // 设置校验位
    setParity() {
      let that = this;
      module.setParity(1, res => {
        that.writeLog(JSON.stringify(res));
      })
    },
    // 设置流控
    setFlowCon() {
      let that = this;
      module.setFlowCon(1, res => {
        that.writeLog(JSON.stringify(res));
      })
    },
    // 设置延迟
    setDelay() {
      let that = this;
      module.setDelay(500, res => {
        that.writeLog(JSON.stringify(res));
      })
    },
    // 开启串口
    open() {
      let that = this;
      module.open(res => {
        that.writeLog(JSON.stringify(res));
      })
    },
    // 关闭串口
    close() {
      let that = this;
      module.close(res => {
        that.writeLog(JSON.stringify(res));
      })
    },
    // 监听字节串口消息
    onListenerBytes() {
      let that = this;
      module.onListenerBytes(res => {
        that.writeLog(JSON.stringify(res));
      })
    },
    // 发送字节数据
    sendBytes() {
```

```
        let that = this;
        module.sendBytes("ab", res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 监听十六进制串口消息
      onListenerHex() {
        let that = this;
        module.onListenerHex(res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 发送16进制数据
      sendHex() {
        let that = this;
        module.sendHex("F001020F", res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 监听ASCII消息串口消息
      onListenerText() {
        let that = this;
        module.onListenerText(res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 发送ASCII数据
      sendText() {
        let that = this;
        module.sendText("ABCDEFG", res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 串口是否开启
      isOpen() {
        let that = this;
        module.isOpen(res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 当前串口属性
      getSerialProperty() {
        let that = this;
        module.getSerialProperty(res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 固定长度粘包处理
      setStaticLenStickPackage() {
        let that = this;
```

```
        module.setStaticLenStickPackage({
          length: 16
        }, res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 固定长度粘包处理
      setVariableLenStickPackage() {
        let that = this;
        module.setVariableLenStickPackage({
          byteOrder: 1,
          lenSize: 2,
          lenIndex: 0,
          offset: 0
        }, res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 首尾特殊字符粘包处理
      setSpecifiedStickPackage() {
        let that = this;
        module.setSpecifiedStickPackage({
          head: "^",
          tail: "$"
        }, res => {
          that.writeLog(JSON.stringify(res));
        })
      },
      // 写日志
      writeLog(str) {
        console.log(str)
        let logStr = uni.$lv.date.format(null, "yyyy-mm-dd hh:MM:ss") + " " + str
        this.logStr = logStr + this.logStr;
      }
    }
  }
</script>

<style>

</style>
```

更新日志

2024-02-04 `v1.2.0`

- [新增] 新增接口【取消监听】

2024-01-22 `v1.1.4`

- [优化] 优化时间戳返回错误问题

2024-01-22 `v1.1.3`

- [优化] 监听串口消息返回新增当前时间戳字段

2023-12-28 v1.1.2

[新增]一键开启串口方法【详情】

2023-10-06 v1.1.1

1. [修复]修复设置首尾特殊字符不生效问题
2. [新增]设置首尾特殊字符新增参数isHex【转到】

2023-09-03 v1.1.0

1. [优化]设置波特率，设置停止位，设置数据位，设置校验位，设置流控，设置延迟更改入参方式
2. [新增]十六进制串口消息监听【转到】
3. [新增]字节串口消息监听【转到】
4. [新增]ASCII串口消息监听【转到】
5. [新增]多串口模式【转到】
6. [删除]删除整体串口监听

2023-08-24 v1.0.0

首次发布

获取所有设备列表

方法名

`getAllDevices`

用法

- 用法如下：

```
module.getAllDevices(res => {
 console.log(res)
})
```

- 参数说明

  无

回调

- 示例

```
{
  "message": "",
  "code": 0,
  "data": {
    "devices": [
      "ttyGS3 (g_serial)",
      "ttyGS2 (g_serial)",
      "ttyGS1 (g_serial)",
      "ttyGS0 (g_serial)",
      "ttysWK3 (ttySWK)",
      "ttysWK2 (ttySWK)",
      "ttysWK1 (ttySWK)",
      "ttysWK0 (ttySWK)",
      "ttyS3 (rk_serial)",
      "ttyS0 (rk_serial)",
      "ttyS4 (rk_serial)",
      "ttyS1 (rk_serial)",
      "ttyFIQ0 (fiq-debugger)"
    ]
```

```
    }
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
data.devices	Array	设备列表
code	Integer	返回类型，0.成功，其他：失败

获取所有设备路径

- 用法如下：

```
```

- 参数说明

  无

- 示例

```
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
data.path	Array	设备路径列表
code	Integer	返回类型，0.成功，其他：失败

设置串口

方法名

`setPort`

> 该方法必须在串口打开之前设置

用法

- 用法如下：

```
module.setPort("/dev/ttyS1", res => {
  console.log(res )
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
port	String	是	/dev/ttyS1	串口路径

回调

- 示例

```
{
  "message": "设置成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

设置波特率

方法名

`setBaudRate`

> 该方法必须在串口打开之前设置

用法

- 用法如下：

```
module.setBaudRate(9600, res => {
  console.log(res )
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
baudRate	Integer	是	9600	波特率值

回调

- 示例

```
{
  "message": "设置成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

设置停止位

方法名

`setStopBits`

> 该方法必须在串口打开之前设置

用法

- 用法如下：

```
module.setStopBits(1, res => {
  console.log(res )
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
stopBits	Integer	是	1	值

回调

- 示例

```
{
  "message": "设置成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

设置数据位

方法名

`setDataBits`

> 该方法必须在串口打开之前设置

用法

- 用法如下：

```
module.setDataBits(1, res => {
  console.log(res )
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
dataBits	Integer	是	8	数据位值

回调

- 示例

```
{
  "message": "设置成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

设置校验位

方法名

`setParity`

> 该方法必须在串口打开之前设置

用法

- 用法如下：

```
module.setParity(1, res => {
  console.log(res )
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
parity	Integer	是	0	值

回调

- 示例

```
{
  "message": "设置成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

设置流控

方法名

> 该方法必须在串口打开之前设置

用法

- 用法如下：

```
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
flowCon	Integer	是	0	值

回调

- 示例

```
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

设置延迟

方法名

`setDelay`

> 该方法必须在串口打开之前设置

用法

- 用法如下：

```
module.setDelay(500, res => {
  console.log(res )
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
delay	Integer	是	500	值

回调

- 示例

```
{
  "message": "设置成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

开启串口

方法名

`open`

> 不设置属性值可以直接打开串口，属性值按默认值处理

用法

- 用法如下：

```
module.open(res => {
  console.log(res)
})
```

- 参数说明

  无

回调

- 示例

```
{
  "message": "开启成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

一键开启串口 v1.1.2

方法名

`fastOpen`

用法

- 用法如下：

```
module.fastOpen({
  //串口地址，必选
  port: "/dev/ttyS1",
  // 波特率，可选
  baudRate: 9600,
  //停止位，可选
  stopBits: 1,
  // 数据位，可选
  dataBits: 8,
  // 校验位，可选
  parity: 0,
  // 流控，可选
  flowCon: 0,
  //延迟，可选
  iDelay: 500
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
port	String	是	无	串口
baudRate	Integer	否	9600	波特率
stopBits	Integer	否	1	停止位
dataBits	Integer	否	8	数据位
parity	Integer	否	0	校验位
flowCon	Integer	否	0	流控
iDelay	Integer	否	500	延迟

回调

- 示例

```
{
  "message": "开启成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

关闭串口

方法名

`close`

用法

- 用法如下：

```
module.close(res => {
  console.log(res)
})
```

- 参数说明

  无

回调

- 示例

```
{
  "message": "关闭成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

监听字节串口消息

方法名

`onListenerBytes`

用法

- 用法如下：

```
module.onListenerBytes(res => {
 console.log(res)
})
```

- 参数说明

  无

回调

- 示例

```
{
  "message": "",
  "code": 0,
  "data": {
        "result":"123456",
        "port":"/dev/ttyS1",
        "index":1
    }
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
data.result	String	串口数据
data.port	String	串口号

参数名	参数类型	参数描述
data.index	Integer	当前加载模块的索引
data.time `v1.1.3`	Integer	当期时间戳
code	Integer	返回类型，0.成功，其他：失败

发送字节数据

方法名

`sendBytes`

用法

- 用法如下：

```
module.sendBytes("ab", res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
content	String	是	无	发送的内容

回调

- 示例

```
{
  "message": "发送成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

监听十六进制串口消息

方法名

`onListenerHex`

用法

- 用法如下：

```
module.onListenerHex(res => {
 console.log(res)
})
```

- 参数说明

  无

回调

- 示例

```
{
  "message": "",
  "code": 0,
  "data": {
        "result":"123456",
        "port":"/dev/ttyS1",
        "index":1
    }
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
data.result	String	串口数据
data.port	String	串口号

监听十六进制串口消息

参数名	参数类型	参数描述
data.index	Integer	当前加载模块的索引
data.time `v1.1.3`	Integer	当期时间戳
code	Integer	返回类型，0.成功，其他：失败

发送16进制消息

方法名

`sendHex`

用法

- 用法如下：

```
module.sendHex("F001020F", res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
content	String	是	无	发送的内容

回调

- 示例

```
{
  "message": "发送成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

监听ASCII消息串口消息

方法名

`onListenerText`

用法

- 用法如下：

```
module.onListenerText(res => {
 console.log(res)
})
```

- 参数说明

  无

回调

- 示例

```
{
  "message": "",
  "code": 0,
  "data": {
        "result":"123456",
        "port":"/dev/ttyS1",
        "index":1
    }
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
data.result	String	串口数据
data.port	String	串口号

参数名	参数类型	参数描述
data.index	Integer	当前加载模块的索引
data.time `v1.1.3`	Integer	当期时间戳
code	Integer	返回类型，0.成功，其他：失败

发送ASCII消息

方法名

`sendText`

用法

- 用法如下：

```
module.sendText("1234567", res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
content	String	是	无	发送的内容

回调

- 示例

```
{
  "message": "发送成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

串口是否开启

方法名

`isOpen`

用法

- 用法如下：

```
module.isOpen(res => {
  console.log(res)
})
```

- 参数说明

  无

回调

- 示例

```
{
  "message": "开启成功",
  "code": 0,
  "data": {
       "isOpen":true
   }
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
data.isOpen	Boolean	是否开启，true：开启，false：未开启
code	Integer	返回类型，0.成功，其他：失败

当前串口属性

方法名

`getSerialProperty`

用法

- 用法如下：

```
module.getSerialProperty(res => {
  console.log(res)
})
```

- 参数说明

  无

回调

- 示例

```
{
  "message": "查询成功",
  "code": 0,
  "data": {
    "delay": 500,
    "parity": 0,
    "baudRate": 9600,
    "port": "/dev/ttyS1",
    "stopBits": 1,
    "flowCon": 0,
    "dataBits": 8
  }
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示

当前串口属性

参数名	参数类型	参数描述
data	Object	数据对象
data.delay	Integer	延迟
data.parity	Integer	校验位
data.baudRate	Integer	波特率
data.port	String	串口号
data.stopBits	Integer	停止位
data.flowCon	Integer	流控
data.dataBits	Integer	数据位
code	Integer	返回类型，0.成功，其他：失败

取消监听 v1.2.0

方法名

`cancelListener`

用法

- 用法如下：

```
module.cancelListener(res => {
  console.log(res)
})
```

- 参数说明

  无

回调

- 示例

```
{
  "message": "操作成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

固定长度粘包处理

方法名

`setStaticLenStickPackage`

> 该方法必须在串口打开之前设置

用法

- 用法如下：

```
module.setStaticLenStickPackage({
  length: 16
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
length	Integer	是	无	字节长度

回调

- 示例

```
{
  "message": "设置成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象

参数名	参数类型	参数描述
code	Integer	返回类型，0.成功，其他：失败

可变长度粘包处理

方法名

`setVariableLenStickPackage`

> 该方法必须在串口打开之前设置
> 可变长度的粘包处理，使用于协议中有长度字段
> 例：协议为: type+dataLen+data+md5
> type:命名类型，两个字节
> dataLen字段的长度,两个字节
> data:数据字段,长度不定，长度为dataLen
> md5字段，8个字节
> 使用时注意：
> byteOrder:首先确定大小端
> lenSize字段的长度，这个例子为2
> lenIndex：len字段的位置，这个例子为2，因为len字段前面为type，它长度为2
> offset：整个包的长度-len，这个例子是，type+dataLen+md5 三个字段的长度，也就是2+2+8=12

用法

- 用法如下：

```
module.setVariableLenStickPackage({
  byteOrder: 1,
  lenSize: 2,
  lenIndex: 0,
  offset: 0
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
byteOrder	Integer	否	1	大端还是小端，1：大端，2：小端
lenSize	Integer	否	2	len字段的长度，一般是2或4
lenIndex	Integer	否	0	长度字段的位置

参数名	参数类型	是否必填	默认值	参数描述
offset	Integer	否	0	整体长度的偏移，比如有一个字段不算在len字段内

回调

- 示例

```
{
  "message": "设置成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败

首尾特殊字符粘包处理

方法名

`setSpecifiedStickPackage`

> 该方法必须在串口打开之前设置

用法

- 用法如下：

```
module.setSpecifiedStickPackage({
  head: "^",
  tail: "$"
}, res => {
  console.log(res)
})
```

- 参数说明

参数名	参数类型	是否必填	默认值	参数描述
head	String	是	无	首字符
tail	String	是	无	尾字符
isHex `v1.1.1`	Boolean	否	false	首尾字符是否为16进制数据

回调

- 示例

```
{
  "message": "设置成功",
  "code": 0,
  "data": {}
}
```

- 回调说明：

首尾特殊字符粘包处理

参数名	参数类型	参数描述
message	String	消息提示
data	Object	数据对象
code	Integer	返回类型，0.成功，其他：失败