串口通信框架

功能：

- 实现串口基本通信；
- 一发多收，一发一收；
- 超时重试；
- 失败重试；
- 响应次数定义；
- 响应规则拆分；
- 全局响应规则和单个请求响应规则；
- 请求和响应拦截器；
- 阻塞式请求；

使用：

依赖配置

在根项目中的 `build.gradle` 文件中添加以下配置：

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        mavenCentral()
        maven { url 'https://jitpack.io' }
    }
}
```

在对应module的`build.gradle` 文件中添加以下配置：

```
dependencies {
    implementation 'com.github.chyi-dev:ok-serialport:Tag'
}
```

OkSerialPort API 文档

`OkSerialPort` 是一个用于管理串口连接和请求处理的类。提供了串口通信的核心功能，包括串口连接、数据发送与接收、请求处理、响应处理、粘包处理等。主要类包括 OkSerialPort、Request、Response、ResponseProcess、SerialPortProcess 等。

核心类 OkSerialPort

串口通信的核心管理类，负责串口连接、数据发送与接收、请求处理等。

OkSerialPort 类

方法

- `connect()`
  连接串口。如果已连接，则直接返回。
- `isConnect(): Boolean`
  检查串口是否已连接。
- `request(request: Request)`
  发送请求数据。如果串口未连接，则不会发送。
- `cancel(request: Request): Boolean`
  取消指定的请求。
- `addProcess(process: ResponseProcess)`
  添加数据处理逻辑。
- `removeProcess(process: ResponseProcess)`
  移除数据处理逻辑。
- `addConnectListener(onConnectListener: OnConnectListener)`
  添加串口连接状态监听器。
- `removeConnectListener()`
  移除串口连接状态监听器。
- `addDataListener(onDataListener: OnDataListener)`
  添加串口数据监听器。
- `removeDataListener()`
  移除串口数据监听器。
- `disconnect()`
  断开串口连接。

OkSerialPort.Builder 类

`Builder` 类用于配置和创建 `OkSerialPort` 实例。

方法

- `devicePath(devicePath: String)`
  设置串口地址。
- `baudRate(baudRate: Int)`
  设置波特率。

- `flags(flags: Int)`
  设置标志位。
- `dataBit(dataBit: Int)`
  设置数据位。
- `stopBit(stopBit: Int)`
  设置停止位。
- `parity(parity: Int)`
  设置校验位（0 表示无校验位，1 表示奇校验，2 表示偶校验）。
- `maxRetry(maxRetry: Int)`
  设置最大重试次数。
- `retryInterval(retryInterval: Long)`
  设置重试间隔时间（毫秒）。
- `sendInterval(sendInterval: Long)`
  设置发送数据的时间间隔（毫秒）。
- `readInterval(readInterval: Long)`
  设置读取数据的时间间隔（毫秒）。
- `maxRequestSize(maxRequestSize: Int)`
  设置最大请求数，默认为100。
- `logger(logger: SerialLogger)`
  设置日志记录器。
- `stickPacketHandle(stickPacketHandle: AbsStickPacketHandle)`
  设置串口粘包处理器。
- `addResponseRule(rule: ResponseRule)`
  添加响应匹配规则。
- `addRequestInterceptor(interceptor: Interceptor<Request>)`
  添加请求拦截器。
- `addResponseInterceptor(interceptor: Interceptor<Response>)`
  添加响应拦截器。
- `build(): OkSerialPort`
  构建并返回 `OkSerialPort` 实例。如果参数无效，会抛出异常。

`Request` 类

串口请求

方法

- `data(data: ByteArray): Request`
  设置请求数据
- `tag(tag: String): Request`
  设置请求标记
- `block(): Request`
  设置是否为阻塞式
- `timeout(timeout: Long): Request`
  设置超时时间
- `timeoutRetry(count: Int): Request`
  设置超时重试次数
- `addResponseRule(responseRule: ResponseRule): Request`
  添加响应规则
- `onResponseListener(listener: OnResponseListener): Request`
  设置响应监听器
- `responseCount(count: Int): Request`
  设置响应次数
- `infiniteResponse(): Request`
  设置无限响应
- `toHex(): String`
  将数据转换为十六进制字符串

`Response` 类

串口响应

方法

- `toHex(): String`
  将数据转换为十六进制字符串

`ResponseProcess` 类

响应处理类，用于处理串口响应。

方法

- `addResponseRule(responseRule: ResponseRule): ResponseProcess`
  添加响应规则
- `isResponseRule(): Boolean`
  检查是否存在响应规则
- `onResponseListener(listener: OnResponseListener): ResponseProcess`
  设置响应监听器
- `responseCount(count: Int): ResponseProcess`
  设置响应次数
- `infiniteResponse(): ResponseProcess`
  设置无限响应

- `match(request: Request?, receive: ByteArray): Boolean`
  匹配响应规则
- `deductCount(): Boolean`
  扣除响应次数

`SerialPortProcess` 类

串口数据处理类，负责数据的发送和接收。

方法

- `start(coroutineScope: CoroutineScope)`
  启动串口处理
- `addRequest(request: Request)`
  添加请求
- `cancelRequest(request: Request): Boolean`
  取消请求
- `addResponseProcess(responseProcess: ResponseProcess)`
  添加响应处理器
- `removeResponseProcess(process: ResponseProcess)`
  移除响应处理器

内部类与接口

`SerialPort` 类

串口数据处理类，负责数据的发送和接收，与C++交互。

方法

- `connect(coroutineScope: CoroutineScope)`
  连接串口调用C++的open方法
- `readStream(): InputStream?`
  获取输入流
- `write(data: ByteArray)`
  写入数据
- `disconnect()`
  关闭连接

`OnConnectListener` 接口

串口连接监听器接口。

- `onConnect(devicePath: String)`
  连接成功回调

- `onDisconnect(devicePath: String, e: Exception?)`
  连接失败回调。

`OnDataListener` 接口

用于监听串口数据。

- `onRequest(data: ByteArray)`
  数据请求回调。
- `onRequest(data: ByteArray)`
  数据响应回调。

`Interceptor<T>` 接口

用于拦截请求或响应。

- `intercept(chain: Chain): T`
  拦截并处理请求或响应。

`Interceptor.Chain<T>` 接口

用于拦截请求或响应链式。

- `data(): T`
  拦截的数据。
- `proceed(data: T): T`
  下一个拦截处理。

`RealInterceptorChain` 类

拦截请求或响应链式构建，继承Interceptor.Chain。

- `data(): T`
  拦截的数据。
- `proceed(data: T): T`
  下一个拦截处理。

`ResponseRule` 接口

用于定义响应匹配规则。

- `match(request: Request?, receive: ByteArray): Boolean`
  拦截并处理请求或响应。

`AbsStickPacketHandle` 抽象类

用于处理串口粘包问题。

- `execute(inputStream: InputStream): ByteArray?`
  串口粘包处理。

示例代码

```
val serialPort = OkSerialPort.Builder()
    .devicePath("/dev/ttyUSB0")
    .baudRate(9600)
    .sendInterval(300)
    .addRequestInterceptor(RequestInterceptor())
    .addResponseInterceptor(ResponseInterceptor())
    .addResponseRule(object : ResponseRule {
        override fun match(request: Request?, receive: ByteArray): Boolean {
            request?.let {
                return receive.size >= 6 && it.data[3] == receive[3]
            } ?: run {
                return false
            }
        }
    })
    .build()

serialPort.addConnectListener(object : OnConnectListener {
    override fun onConnect(devicePath: String) {
        println("Connected to $devicePath")
    }

    override fun onDisconnect(devicePath: String, e: Exception?) {
        println("Disconnected from $devicePath: ${e?.message}")
    }
})

serialPort.connect()

val request = Request(byteArr)
    .responseCount(3)
    .onResponseListener(object : OnResponseListener {
        override fun onResponse(response: Response) {
            Log.i("Ok-Serial", "response onResponse:${response.toHex()}")
        }

        override fun onFailure(request: Request?, e: Exception) {
            Log.i("Ok-Serial", "response onFailure:${e.message}")
        }
    })
serialPort?.request(request)
```