LibSerial库使用指南

1. 简介

LibSerial是一个C++库，用于在Linux系统上进行串口通信。它提供了简单易用的接口来配置和操作串口设备。

2. 安装方法

```
# 安装依赖
sudo apt-get install libboost-all-dev

# 安装LibSerial
sudo apt-get install libserial-dev
```

3. 基本用法

3.1 头文件包含

```
#include <libserial/SerialPort.h>
```

3.2 主要类和方法

SerialPort类

```
LibSerial::SerialPort serial_port;
```

常用方法

1. 打开串口

```
// 打开指定串口
serial_port.Open("/dev/ttyUSB0");

// 设置串口参数
serial_port.SetBaudRate(LibSerial::BaudRate::BAUD_115200);
serial_port.SetCharacterSize(LibSerial::CharacterSize::CHAR_SIZE_8);
serial_port.SetFlowControl(LibSerial::FlowControl::FLOW_CONTROL_NONE);
serial_port.SetParity(LibSerial::Parity::PARITY_NONE);
serial_port.SetStopBits(LibSerial::StopBits::STOP_BITS_1);
```

2. 读写数据

```
// 写入数据
std::string write_data = "Hello World";
serial_port.Write(write_data);

// 读取数据
std::string read_data;
serial_port.Read(read_data, 1000, 100);  // 超时1000ms，读取100字节
```

3. 关闭串口

```
serial_port.Close();
```

3.3 完整示例

```
#include <libserial/SerialPort.h>
#include <iostream>

int main() {
    try {
        // 创建串口对象
        LibSerial::SerialPort serial_port;

        // 打开串口
        serial_port.Open("/dev/ttyUSB0");

        // 配置串口参数
        serial_port.SetBaudRate(LibSerial::BaudRate::BAUD_115200);

serial_port.SetCharacterSize(LibSerial::CharacterSize::CHAR_SIZE_8);

serial_port.SetFlowControl(LibSerial::FlowControl::FLOW_CONTROL_NONE);
        serial_port.SetParity(LibSerial::Parity::PARITY_NONE);
        serial_port.SetStopBits(LibSerial::StopBits::STOP_BITS_1);

        // 写入数据
        std::string write_data = "Hello World";
        serial_port.Write(write_data);

        // 读取数据
        std::string read_data;
        serial_port.Read(read_data, 1000, 100);

        // 输出读取的数据
        std::cout << "Received: " << read_data << std::endl;

        // 关闭串口
        serial_port.Close();
    }
```

```
    catch (const LibSerial::OpenFailed&) {
        std::cerr << "串口打开失败" << std::endl;
    }
    catch (const LibSerial::ReadTimeout&) {
        std::cerr << "读取超时" << std::endl;
    }
    catch (const std::exception& e) {
        std::cerr << "发生错误: " << e.what() << std::endl;
    }

    return 0;
}
```

```
class UART {
private:
    LibSerial::SerialPort serial_port;
    bool is_open = false;

public:
    /**
     * @brief 打开串口
     * @param port_name 串口设备名（如"/dev/ttyUSB0"）
     * @param baud_rate 波特率
     * @return 是否成功打开
     */
    bool open(const std::string& port_name, int baud_rate = 115200) {
        try {
            serial_port.Open(port_name);
            serial_port.SetBaudRate(LibSerial::BaudRate::BAUD_115200);

serial_port.SetCharacterSize(LibSerial::CharacterSize::CHAR_SIZE_8);

serial_port.SetFlowControl(LibSerial::FlowControl::FLOW_CONTROL_NONE);
            serial_port.SetParity(LibSerial::Parity::PARITY_NONE);
            serial_port.SetStopBits(LibSerial::StopBits::STOP_BITS_1);
            is_open = true;
            return true;
        }
        catch (const LibSerial::OpenFailed&) {
            std::cerr << "串口打开失败" << std::endl;
            return false;
        }
    }

    /**
     * @brief 关闭串口
     */
    void close() {
        if (is_open) {
            serial_port.Close();
            is_open = false;
```

```
        }
    }

    /**
     * @brief 发送数据
     * @param data 要发送的数据
     * @return 是否发送成功
     */
    bool write(const std::string& data) {
        if (!is_open) return false;
        try {
            serial_port.Write(data);
            return true;
        }
        catch (const std::exception& e) {
            std::cerr << "发送数据失败: " << e.what() << std::endl;
            return false;
        }
    }

    /**
     * @brief 读取数据
     * @param data 存储读取的数据
     * @param timeout_ms 超时时间（毫秒）
     * @param size 要读取的字节数
     * @return 是否读取成功
     */
    bool read(std::string& data, int timeout_ms = 1000, int size = 100) {
        if (!is_open) return false;
        try {
            serial_port.Read(data, timeout_ms, size);
            return true;
        }
        catch (const LibSerial::ReadTimeout&) {
            std::cerr << "读取超时" << std::endl;
            return false;
        }
        catch (const std::exception& e) {
            std::cerr << "读取数据失败: " << e.what() << std::endl;
            return false;
        }
    }

    /**
     * @brief 检查串口是否打开
     * @return 串口是否打开
     */
    bool isOpen() const {
        return is_open;
    }

    /**
     * @brief 析构函数
     */
```

```
    ~UART() {
        close();
    }
};
```

4. 串口参数说明

4.1 波特率（BaudRate）

常用波特率：

- BAUD_9600
- BAUD_19200
- BAUD_38400
- BAUD_57600
- BAUD_115200

4.2 数据位（CharacterSize）

- CHAR_SIZE_5
- CHAR_SIZE_6
- CHAR_SIZE_7
- CHAR_SIZE_8

4.3 停止位（StopBits）

- STOP_BITS_1
- STOP_BITS_2

4.4 校验位（Parity）

- PARITY_NONE
- PARITY_ODD
- PARITY_EVEN

4.5 流控制（FlowControl）

- FLOW_CONTROL_NONE
- FLOW_CONTROL_HARDWARE
- FLOW_CONTROL_SOFTWARE

5. 错误处理

LibSerial提供了多种异常类：

- OpenFailed：打开串口失败
- AlreadyOpen：串口已经打开
- ReadTimeout：读取超时
- WriteTimeout：写入超时
- PortNotOpen：串口未打开

6. 注意事项

1. 确保有串口设备的访问权限

```
sudo chmod 666 /dev/ttyUSB0
```

2. 检查串口设备是否存在

```
ls -l /dev/ttyUSB*
```

3. 在程序退出前记得关闭串口
4. 使用try-catch处理可能的异常
5. 注意串口参数的匹配（波特率、数据位等）

7. 常见问题解决

1. 串口打开失败
    - 检查设备名称是否正确
    - 检查访问权限
    - 检查设备是否被其他程序占用
2. 数据读取超时
    - 检查波特率设置
    - 检查数据格式
    - 适当增加超时时间
3. 数据错误
    - 检查串口参数设置
    - 检查数据格式
    - 检查硬件连接