SerialPortInfo.cpp 串口信息类

```
SerialPortInfo::SerialPortInfo()
= default;
//这是SerialPortInfo类的默认构造函数,使用"default"指示编译器生成默认实现
```

```
SerialPortInfo::SerialPortInfo(std::string portName, const unsigned& baudRate, const unsigne
    CharacterSize(characterSize),
    BaudRate(baudRate),
    DeviceName(std::move(portName))
{
}
/*接受三个参数：portName（串行端口名称）、baudRate（波特率）和characterSize（字符大小）
将characterSize赋值给CharacterSize成员变量
将baudRate赋值给BaudRate成员变量
使用std::move()将portName从外部移动到DeviceName成员变量内，以避免内存拷贝*/
```

```
SerialPortInfo::SerialPortInfo(const SerialPortInfo& other)
= default;
这是SerialPortInfo类的复制构造函数的实现
```

```
SerialPortInfo::SerialPortInfo(SerialPortInfo&& other) noexcept
= default;
这是SerialPortInfo类的移动构造函数的实现
```

```
SerialPortInfo& SerialPortInfo::operator=(const SerialPortInfo& other)
= default;
这是SerialPortInfo类的复制赋值运算符的实现
```

```
SerialPortInfo& SerialPortInfo::operator=(SerialPortInfo&& other) noexcept
= default;
这是SerialPortInfo类的移动赋值运算符的实现
```

```
std::string SerialPortInfo::ToString() const
{
    return fmt::format(
        R"({{"IsConnected":"{}","CharacterSize":"{}","BaudRate":"{}","DeviceName":"{}"}})",
        IsConnected,
        CharacterSize,
        BaudRate,
```

```
        DeviceName
    );
}
```

ToString函数的实现

使用fmt库的format函数将类的成员变量格式化为一个JSON字符串

JSON字符串的格式为：{"IsConnected":"<IsConnected值>","CharacterSize":"<CharacterSize值>","

返回生成的JSON字符串作为函数的结果

SerialPort.hpp 串口通信类

```
SerialPort
::SerialPort(boost::asio::io_context& ioContext, SerialPortInfo info) :
UniqueSerialPort(std::make_uniqueboost::asio::serial_port(ioContext)),
Info(std::move(info))
{
}//构造函数
```

```
//应用设置选项
void
SerialPort
::ApplyOption(boost::system::error_code& result) const
{
using AsioSerial = boost::asio::serial_port;

// 设置波特率
UniqueSerialPort->set_option(AsioSerial::baud_rate(Info.BaudRate), result);
if (result.failed()) return;

// 设置流控制
UniqueSerialPort->set_option(AsioSerial::flow_control(AsioSerial::flow_control::none), resul
if (result.failed()) return;

// 设置校验位
UniqueSerialPort->set_option(AsioSerial::parity(AsioSerial::parity::none), result);
if (result.failed()) return;

// 设置停止位
UniqueSerialPort->set_option(AsioSerial::stop_bits(AsioSerial::stop_bits::one), result);
if (result.failed()) return;

// 设置字符大小
```

```
UniqueSerialPort->set_option(AsioSerial::character_size(AsioSerial::character_size(Info.Char
)
```

```
// 刷新连接
void
SerialPort
::RefreshConnection(boost::system::error_code& result)
{
using namespace boost::asio;

spdlog::info("SerialPort({})> Refreshing connection", Info.DeviceName);

// 如果已经连接，则直接返回
if (UniqueSerialPort->is_open())
{
    spdlog::info("SerialPort({})> Is connected, don't need refresh.");
    Info.IsConnected = true;
    result = boost::system::error_code();
    return;
}

// 标记为未连接
Info.IsConnected = false;

// 打开串口
UniqueSerialPort->open(Info.DeviceName, result);
if (result.failed())
{
    spdlog::warn("SerialPort({})> Cannot open: {}", Info.DeviceName, result.what());
    return;
}

// 应用设置选项
ApplyOption(result);
if (result.failed())
{
    spdlog::warn("SerialPort({})> Cannot set option: {}", Info.DeviceName, result.what());
    return;
}

// 标记为已连接
```

```
Info.IsConnected = true;
}
```

```
// 构建共享指针
std::shared_ptr<SerialPort>
SerialPort::
BuildShared(boost::asio::io_context& ioContext, const SerialPortInfo& info)
{
spdlog::info("Building Shared SerialPort({})", info.ToString());
auto ptr = std::shared_ptr<SerialPort>(new SerialPort{ioContext, info});
spdlog::info("Finished building SerialPort({})", info.DeviceName);
return ptr;
}

// 析构函数
SerialPort
::~SerialPort()
{
spdlog::info("SerialPort({})> Trying closing", Info.DeviceName);
Disconnect();
spdlog::info("SerialPort({})> Closed", Info.DeviceName);
}

// 连接串口
void
SerialPort
::Connect()
{
boost::system::error_code result;
RefreshConnection(result);
if (result.failed())
{
spdlog::warn("SerialPort({})> Cannot connect: {}", Info.DeviceName, result.what());
}
}

// 判断是否已连接
bool
SerialPort
::IsConnected()
{
return Info.IsConnected && UniqueSerialPort->is_open();
}
```

```
// 断开连接
void
SerialPort
::Disconnect()
{
boost::system::error_code result;
UniqueSerialPort->close(result);
if (result.failed())
{
spdlog::warn("SerialPort({})> Cannot close serial port: {}", Info.DeviceName, result.what())
}

Info.IsConnected = false;
}

// 获取信息
const
SerialPortInfo&
SerialPort::GetInfo() const
{
return Info;
}

// 读取数据
MemoryView::SizeType
SerialPort
::Read(const MemoryView& view)
{
boost::system::error_code result{};
auto bytes = read(*UniqueSerialPort, view.ToBuffer(), result);
spdlog::info(
"SerialPort({})> Read {}:{} bytes, failed({}), failure({})",
Info.DeviceName,
bytes,
view.Size,
result.failed(),
result.what()
);

return bytes;
}
```

```
// 写入数据
MemoryView::SizeType
SerialPort
::Write(const MemoryView& view)
{
boost::system::error_code result{};
auto bytes = write(*UniqueSerialPort, view.ToBuffer(), result);
spdlog::info(
"SerialPort({})> Write {}:{} bytes, failed({}), failure({})",
Info.DeviceName,
bytes,
view.Size,
result.failed(),
result.what()
);

return bytes;
}
```

MemoryView.hpp 内存数据操作类

```
MemoryView(): 默认构造函数。
MemoryView(const MemoryView& other): 拷贝构造函数。
MemoryView(MemoryView&& other) noexcept: 移动构造函数。
MemoryView& operator=(const MemoryView& other): 拷贝赋值运算符。
MemoryView& operator=(MemoryView&& other) noexcept: 移动赋值运算符。
boost::asio::mutable_buffer ToBuffer() const: 将MemoryView转换为boost::asio::mutable_buffer
MemoryView::ByteType& operator[](const SizeType size) const: 重载索引运算符，返回指定位置
MemoryView::SizeType CopyTo(void* head) const: 将MemoryView的数据复制到指定的内存地址，并
MemoryView::SizeType CopyTo(const MemoryView& destination) const: 将MemoryView的数据复制到
MemoryView::SizeType ReadFrom(const void* head) const: 从指定的内存地址读取数据到MemoryVi
`MemoryView::SizeType ReadFrom [Something went wrong, please try again later.]
```

TCPConnection.hpp TCP连接类

```
TCPConnection::TCPConnection()//构造函数成员变量 Endpoint 被初始化为传入的远程端点，Rem
TCPConnection::RefreshConnection()//刷新连接状态
CPConnection::BuildShared()//构建一个指向 TCPConnection 对象的 shared_ptr 智能指针
TCPConnection::~TCPConnection()//析构函数。析构函数被调用时，会尝试关闭 TCP 连接，并记录
TCPConnection::Connect()//尝试连接到远程端点
TCPConnection::IsConnected()//返回当前连接状态
CPConnection::Disconnect()//关闭 TCP 连接
```

```
TCPConnection::Read()//从 TCP 连接中读取数据，并返回读取的字节数
TCPConnection::Write()//向 TCP 连接中写入数据，并返回写入的字节数。
```