SCPIInterface使用说明

项目简介

SCPIInterface服务端主要用于系统的设备控制，客户端与SCPI指令解析服务端数据通信过程如图1所示。客户端向服务端发送的指令分为查询指令和控制指令，查询指令用于查询设备状态及参数等信息，控制指令用于控制设备。查询指令下，服务端解析客户端下发的指令，根据指令从设备获取查询信息，并回传至客户端。控制指令下，服务端解析客户端下发的指令，根据指令向设备发送控制信息。

图1 客户端与SCPI指令解析服务端数据通信

环境依赖

操作系统

- Windows 7 32bit and 64bit
- Windows 10 32bit and 64bit

开发环境

- Microsoft Visual Studio Professional 2015 Update 3或者更高版本。

依赖库

- boost_1_74_0。

快速入门

建立SCPI指令解析服务端开发环境

- Step 1：打开Microsoft Visual Studio Professional 2019后，点击"File"->"New"->"Project"，在弹出的对话中框选择“Console App”工程类型，点击“Next”。如图2所示。

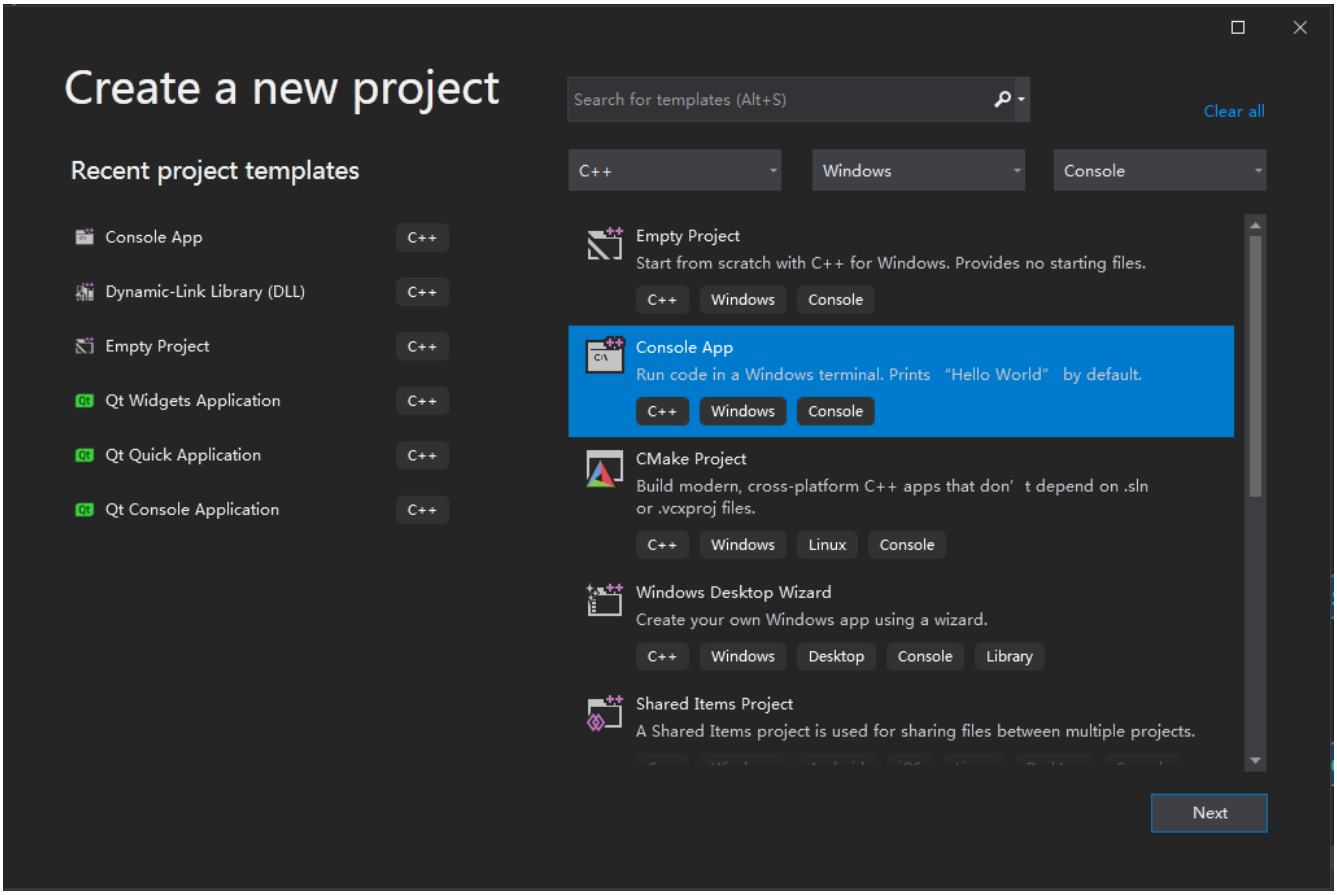

图2 创建控制台工程

- Step 2：配置工程名称及路径后，点击“Create”，创建工程。如图3所示。

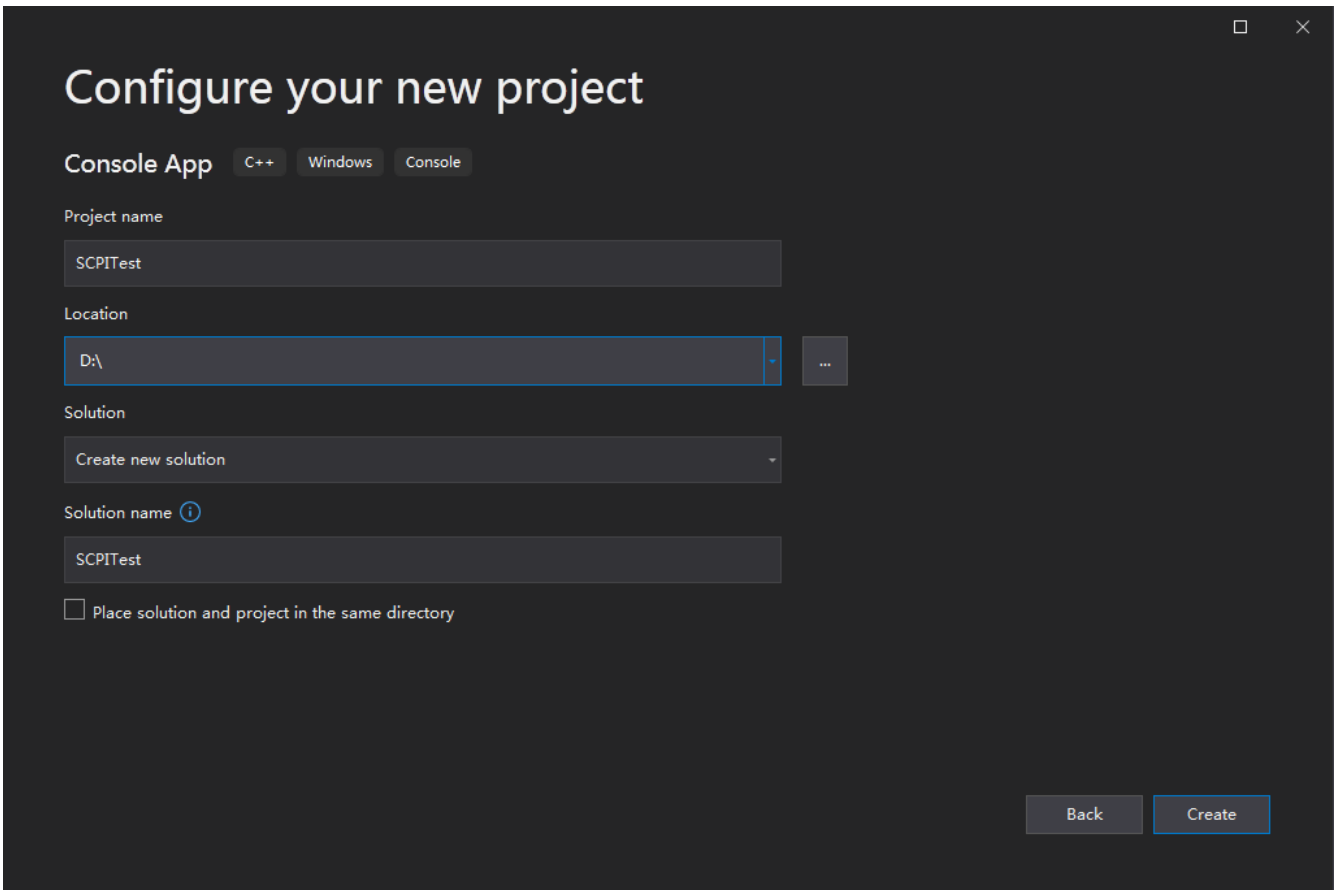

图3 配置工程

- Step 3：将工程依赖的源码文件夹SCPIInterface拷贝到工程文件同级目录下。
- Step 4：在工程名称上点击鼠标右键，点击"Add"->"NewFilter"，添加新文件夹并重命名为interface，用于存放将工程依赖的源码文件。
- Step 5：文件夹interface上点击鼠标右键，点击"Add"->"Existing Items..."，将SCPIInterface文件夹下所有源码添加到interface文件夹中，可根据需要在interface文件夹下新建文件夹将源码分组。如图4所示。

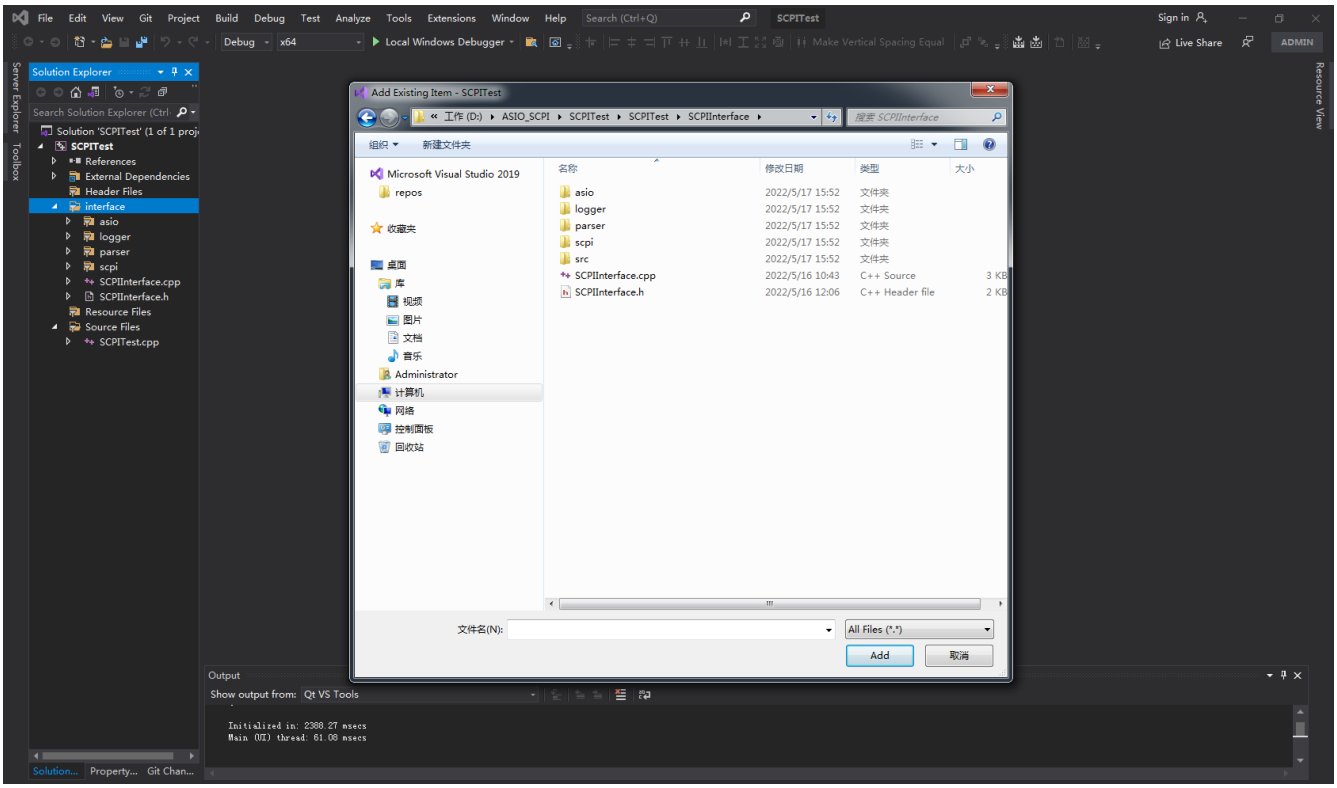

图4 添加依赖源码文件

- Step 6：在工程名称上点击鼠标右键，在弹出的菜单中点击“Properties”->"C/C++"->"General"，在Additional Include Directories中添加boost库文件路径。如图5所示。

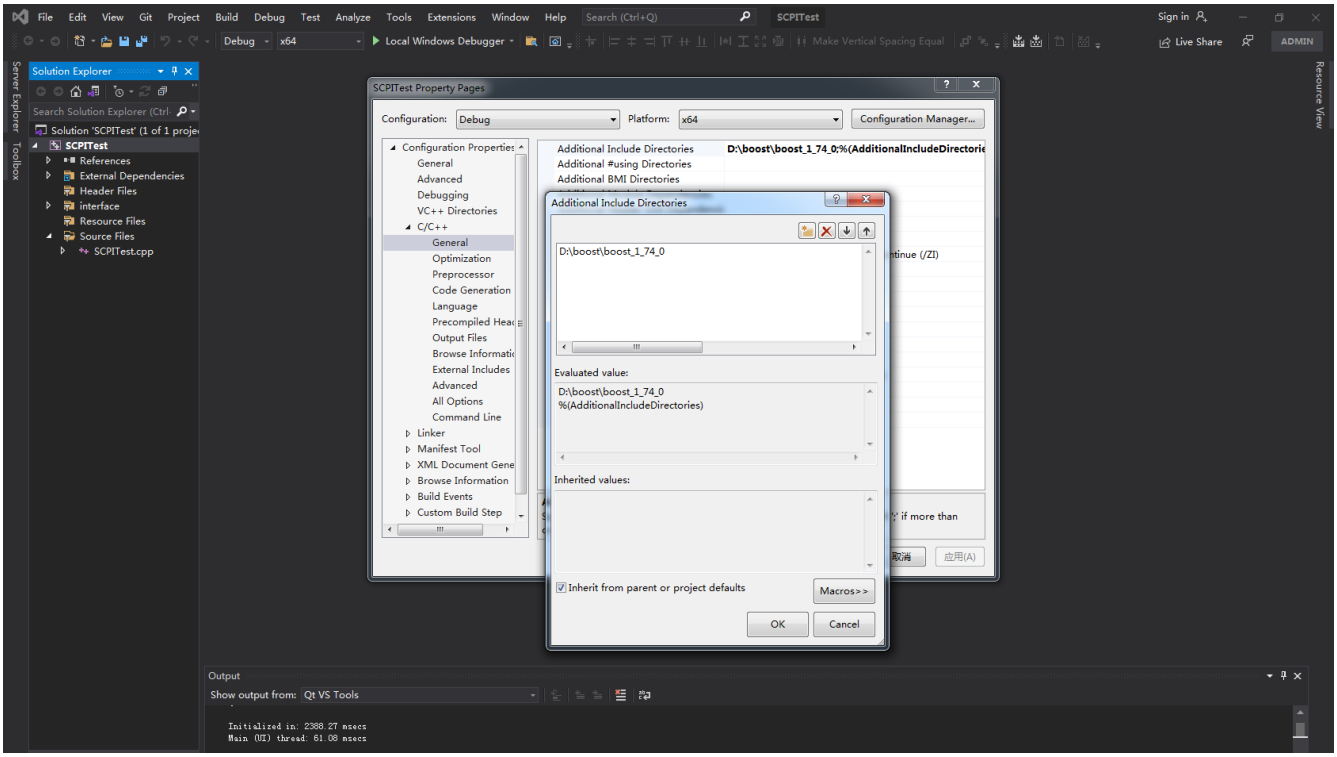

图5 包含boost库路径

- Step 7：在工程名称上点击鼠标右键，在弹出的菜单中点击“Properties”->"C/C++"->"Preprocessor"，在Preprocessor Definitions中添加宏定义_CRT_SECURE_NO_WARNINGS、NET_ASSIS（网络助手调试时用到）。如图6所示。

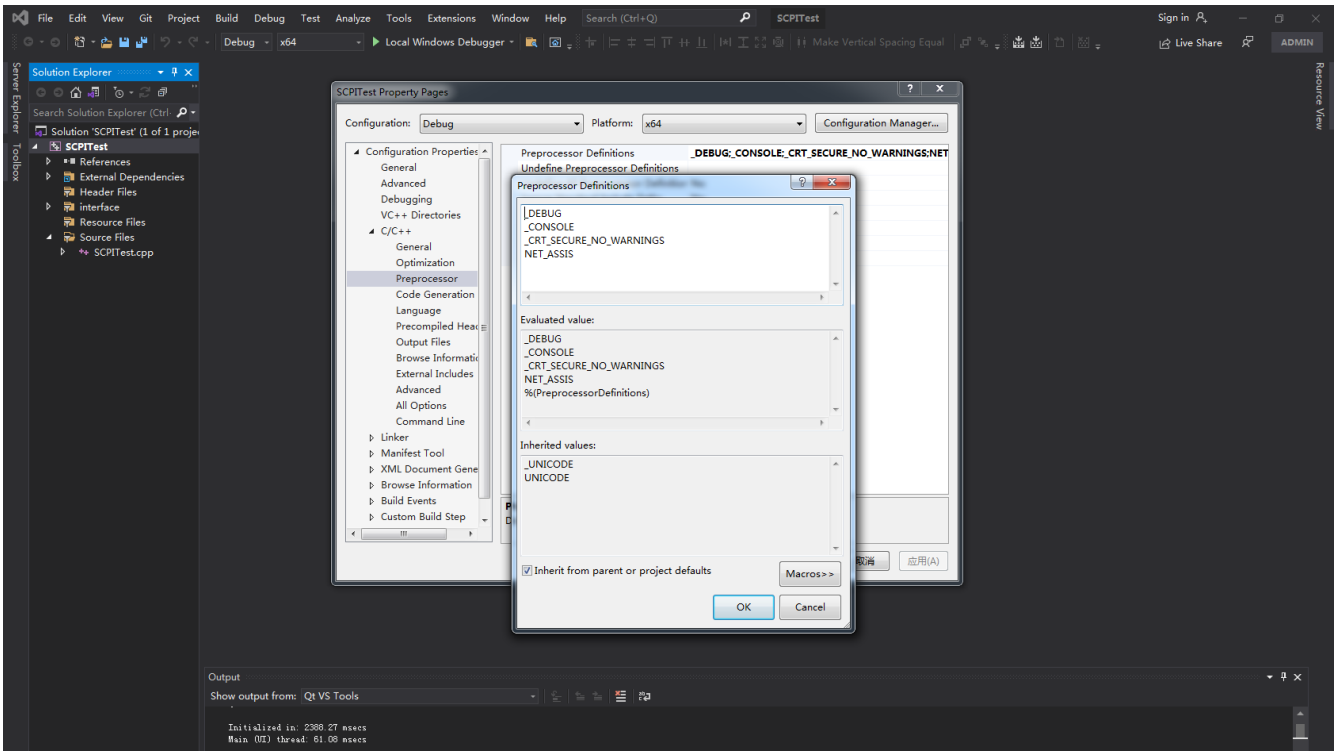

图6 添加宏定义

- Step 8：在工程名称上点击鼠标右键，在弹出的菜单中点击“Properties”->"Linker"->"General"，在Additional LibraryDirectories中添加boost库路径。如图7所示。

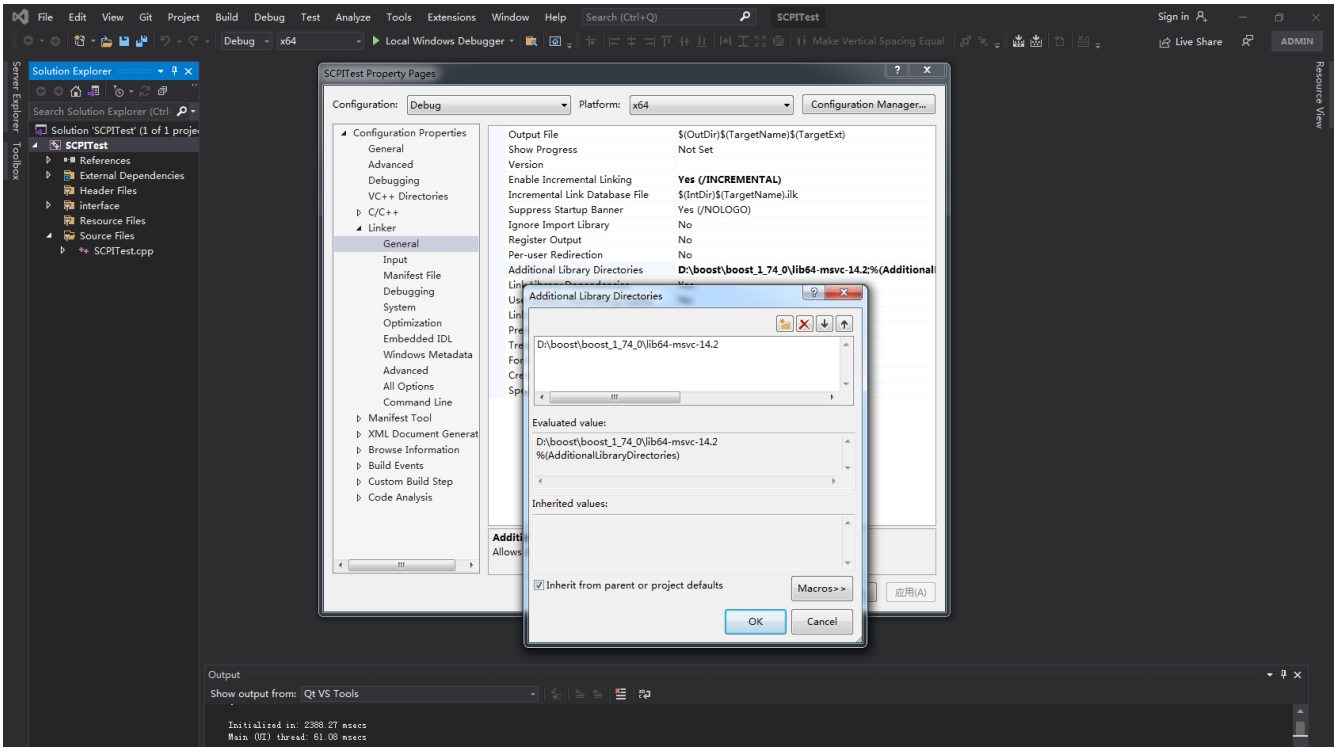

图7 添加boost库路径

启动SCPI指令解析服务端

- Step 1：替换SCPITest.cpp文件内容如下：

```
#include "SCPIInterface/SCPIInterface.h" //包含接口头文件

using namespace SCPI;//命名空间SCPI

//测试类
class SCPITest
{
public:
    SCPITest() {};
    ~SCPITest() {};

public:
    void addCommand()
    {
        //添加设置频率指令
        m_iterfc.addCommand("[:SENSe]:FREQuency:CENTer", bind(&SCPITest::SetFreq));
        //添加查询迹线指令
        m_iterfc.addCommand("TRAC?", bind(&SCPITest::Trace));
        //添加查询频率指令
        m_iterfc.addCommand("[:SENSe]:FREQuency:CENTer?", bind(&SCPITest::GetFreq));
        //结束添加指令
        m_iterfc.endAdd();
    }

void start()
{
```

```
    m_iterfc.startListen(5050);//开始监听5050端口
}

void stop()
{
    m_iterfc.stopListen();//停止监听
}

private:
    CommandHandler bind(bool(SCPITest::* funPtr)(const SCPIInterface& inter))
    {
        return std::bind(funPtr, this, std::placeholders::_1);
    }

//设置频率指令回调函数
bool SetFreq(const SCPIInterface& inter)
{
    //获得数值
    /*
    double param;
    while (inter.getParam(param))//获得客户端发来的double类型参数
    {
        int doSomething = 0;
    }*/

//获得参数
scpi_number_t paramNum;
while (inter.getParam(paramNum))//获得客户端发来的scpi_number_t类型参数
{
    m_params.emplace_back(paramNum);
}
return true;
}

//查询迹线指令回调函数
bool Trace(const SCPIInterface& inter)
{
    return inter.sendArray(data());//向客户端发送迹线数据
}

//查询频率指令回调函数
bool GetFreq(const SCPIInterface& inter)
{
    for (auto par : m_params)
    {
        inter.sendValue(par.content.value);//向客户端发送频率
    }
    return true;
}

private:
    //迹线数据
    std::vector<float> data()
```

```
    {
        std::vector<float> vec{7.1,8.2,9.3};
        return vec;
    }

private:
    SCPIInterface m_iterfc;
    std::vector<scpi_number_t> m_params;
};

int main()
{
    SCPITest test;
    test.addCommand();
    test.start();
    system("pause");
    test.stop();
    return 0;
}
```

Step 2：工程编译运行成功后即可启动服务端，监听端口5050。如图8所示。

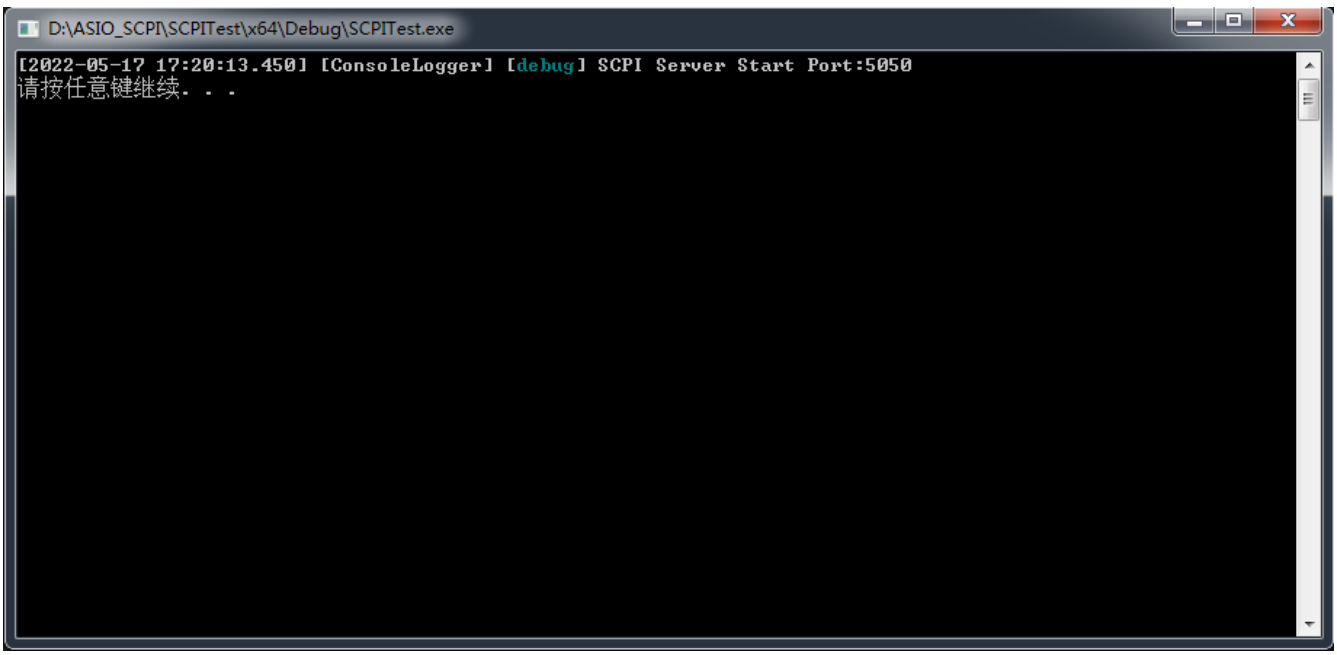

图8 启动服务

通过网络助手发送SCPI指令

- Step 1：运行网络调试助手，协议类型选择”TCP客户端“，服务器端口设置为”5050“，点击”连接“，即可连接到服务端：

图9 连接服务端

- Step 2：在网络助手发送区输入“:FREQ:CENT 5 GHz”，点击发送按钮，即可向服务端发送设置频率SCPI指令，设置的频率为 5GHz。
- Step 3：在网络助手发送区输入“TRAC?”，点击发送按钮，即可向服务端发送查询迹线SCPI指令，此时网络助手接收区会显示“7.1,8.2,9.3”，这就是服务端回传的迹线极限数据。
- Step 4：在网络助手发送区输入“:FREQ:CENT?”，点击发送按钮，即可向服务端发送查询频率SCPI指令，此时网络助手接收区会显示频率“5000000000”，这就是服务端回传的迹线极限数据。此值为Step 2设置的频率。
- Step 5：点击”断开“，即可断开与服务端的连接。

执行Step 1~Step5后服务端接收到SCPI指令，控制台输出如下：

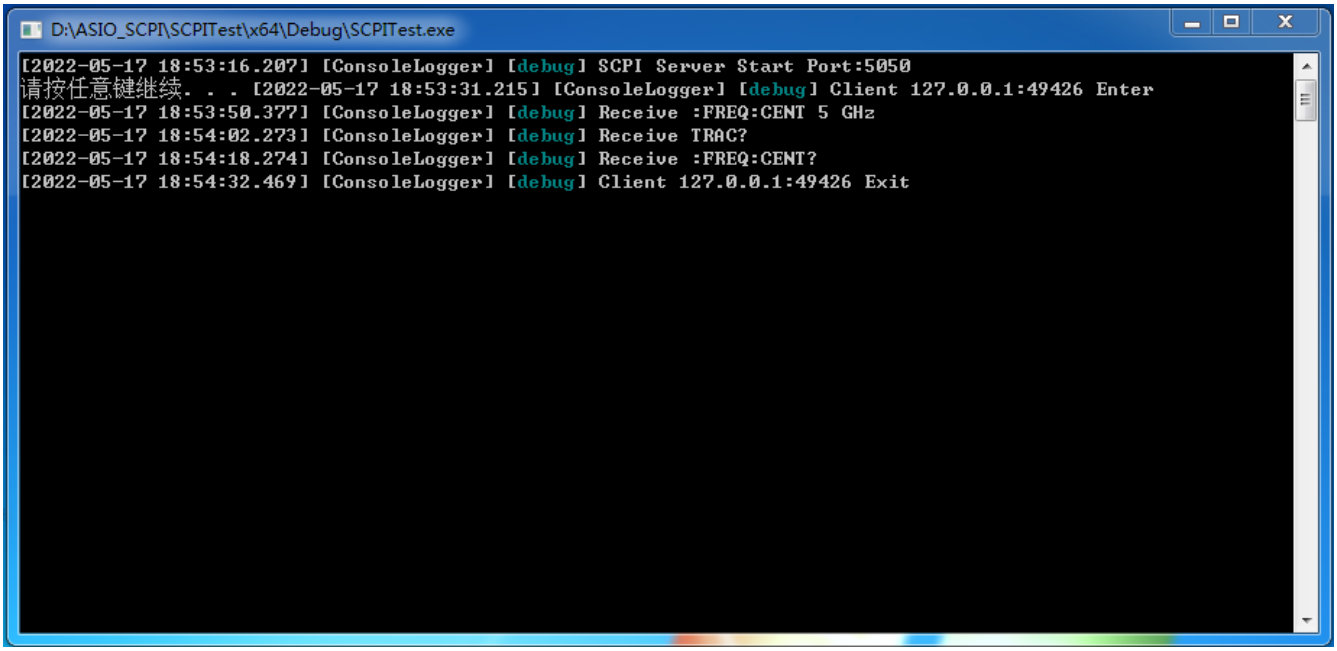

图10 服务端输出

执行Step 1~Step5后网络助手接收区显示SCPI指令响应，输出如下：

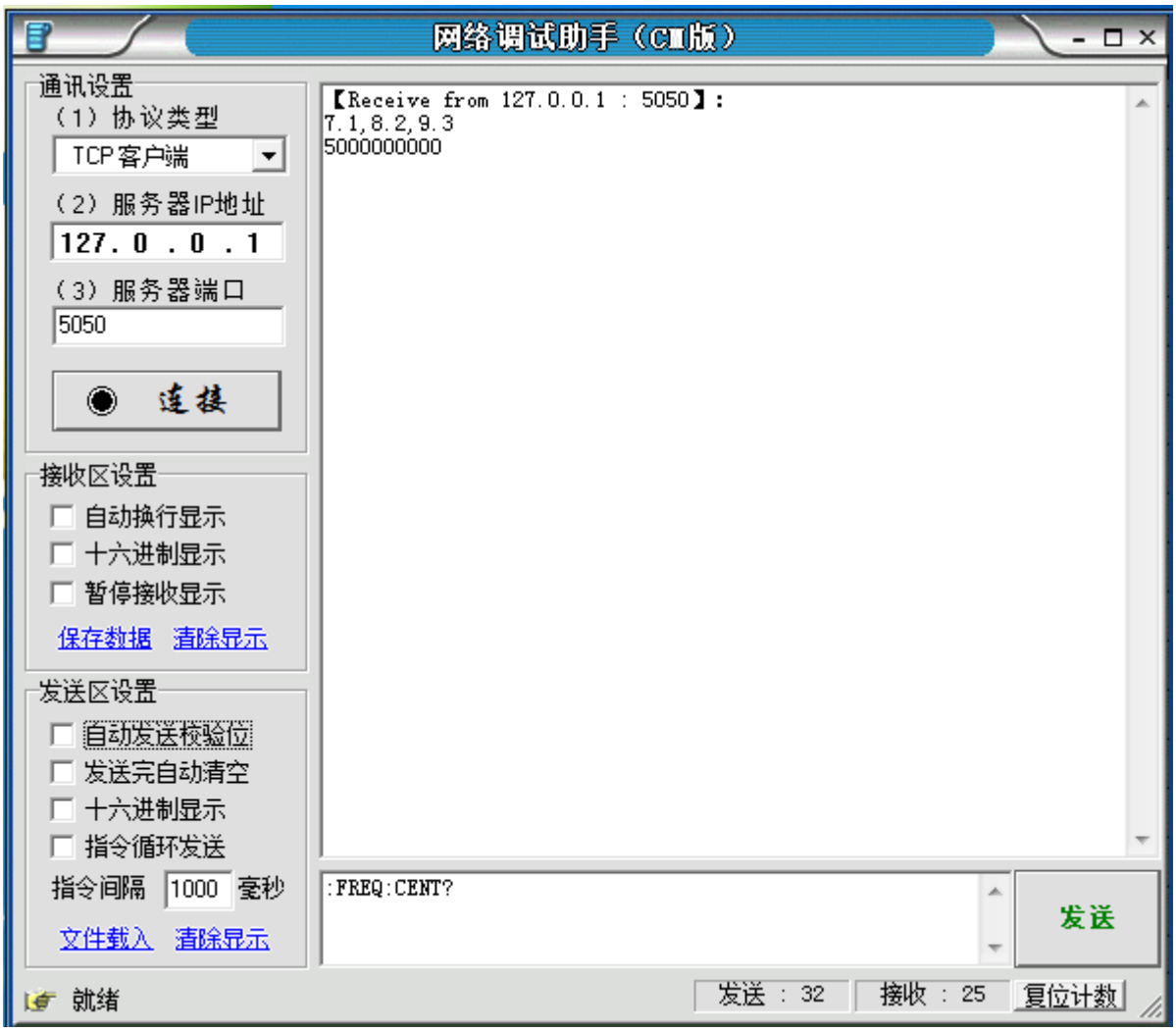

图11 客户端响应

使用详细说明

技术原理介绍

SCPIInterface依赖于Boost.Asio、SCPIParser。Boost.Asio为异步网络通信库，用于异步接受客户端请求、向客户端发送响应，SCPIParser为SCPI指令解析器，用于解析SCPI指令。利用SCPIInterface进行客户端与设备通信的过程如图12所示。客户端向服务端发送的SCPI指令分为查询指令和控制指令，查询指令用于查询设备状态及参数等信息，控制指令用于控制设备。TCPListner接收到客户端SCPI指令后，交给NetParser解析，执行与SCPI指令绑定的处理例程。若为查询指令，则通过TCPListner将设备响应发送到客户端；若为控制指令，则直接向设备发送控制信息。

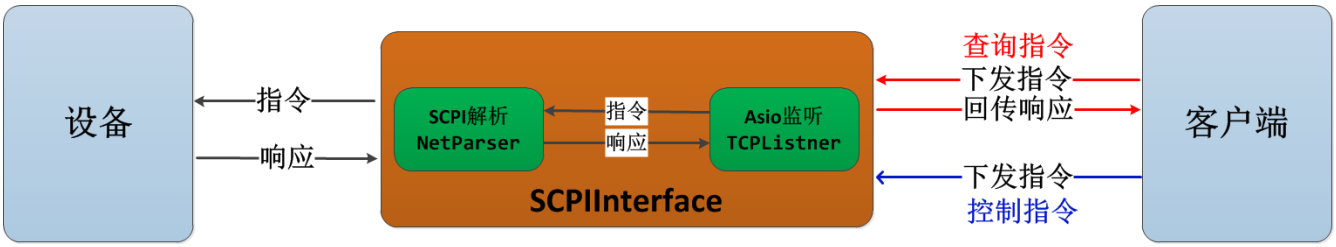

图12 客户端与设备通信的过程

调用接口主要步骤

Step 1：#include "SCPIInterface/SCPIInterface.h"

包含头文件SCPIInterface.h，该文件位于文件夹SCPIInterface中，主要内容为声明对外接口函数及数据类型，详细介绍参见接口及数据类型说明。

Step 2：using namespace SCPI

使用命名空间SCPI，所有接口函数及数据类型均位于命名空间SCPI下。

Step 3：调用SCPIInterface创建SCPIInterface类的实例

通过该函数创建SCPIInterface类的实例，通过实例访问该类的公开接口。SCPIInterface()函数原型为：

```
SCPIInterface(const IDN& idn);
```

该函数的唯一参数idn为一个结构体常量，为输入参数，代表用户自定义的仪器标识，调用时需要将该位置替换为仪器标识结构体。IDN结构如下：

```
struct IDN
{
    std::string idn1;
    std::string idn2;
    std::string idn3;
    std::string idn4;
};
```

Step 4：调用addCommand添加SCPI指令及其响应函数

通过该函数向SCPI指令解析服务端的SCPI指令集合中添加新的SCPI指令，及服务端接收该SCPI指令后触发的回调函数。addCommand()函数原型为：

```
void addCommand(const std::string& pattern, const CommandHandler& handler);
```

在该函数的参数表中，第一个参数pattern为一个字符串引用常量，为输入参数，代表用户自定义的SCPI指令，调用时需要将该位置替换为常量字符串。第二个参数handler为CommandHandler类型引用常量，为输入参数，是服务端接收SCPI指令后执行的**回调函数**，调用时需要将该位置替换为CommandHandler类型函数对象。CommandHandler结构如下：

```
using CommandHandler = std::function <bool(const SCPIInterface& inter)>;
```

该函数的输入参数为const SCPIInterface&，返回值类型为bool。

对于控制类的SCPI指令，需要在**回调函数**中调用**getParam**接口获取客户端控制参数。getParam()函数原型为：

```
template<class T>
bool getParam(T& param) const;
```

该函数为一个模板函数。该函数唯一的参数param为不定类型变量，为输出参数，你需要先定义一个需要获取的参数类形的变量，然后将其放到该参数位置上。支持的参数类型包括int32_t、uint32_t、int64_t、uint64_t、float、double及scpi_number_t。scpi_number_t结构如下：

```
struct scpi_number_t {
    scpi_bool_t special;
    union {
        double value;
        int32_t tag;
    } content;
    scpi_unit_t unit;
    int8_t base;
};
```

special为true，表明参数不是数值，而是最大、最小或者默认值。

value为标准单位下参数的数值部分，只有special为false时，value才有效。

unit为参数的标准单位，结构见数据类型说明。

base为进制类型，如DEC、HEX、BIN等。

对于查询类的SCPI指令，需要在**回调函数**中调用**sendValue**接口或**sendArray**接口向客户端发送响应。sendValue()函数原型为：

```
template<class T>
bool sendValue(const T& val) const;
```

该函数为一个模板函数。通过该函数将单个数值发送到客户端，该函数唯一的参数val为不定类型常量，为输入参数，你需要将要发送的数值放到该参数位置上。支持的参数类型包括int32_t、int64_t、float、double。

sendArray()函数原型为：

```
template<class T>
bool sendArray(const std::vector<T>& data) const;
```

该函数为一个模板函数。通过该函数将数组发送到客户端，该函数唯一的参数data为不定类型数组常量，为输入参数，你需要将要发送的数组放到该参数位置上。支持的参数类型包括int8_t、uint8_t、int16_t、uint16_t、int32_t、uint32_t、int64_t、uint64_t、float、double。

Step 5：调用endAdd结束添加SCPI指令

该函数为添加完所有SCPI指令结束后必须调用的函数，通过该函数初始化SCPI指令解析服务端需要的一些资源。

Step 6：调用startListen启动服务

通过该函数启动SCPI指令解析服务，服务异步读取客户端发来的SCPI指令，并以异步方式将指令响应发送到客户端。startListen()函数原型为：

```
void startListen(const unsigned short port);
```

该函数的唯一参数port为无符号short类型常量，表示服务端需要监听的端口号。

Step 7：调用stopListen停止服务端

通过该函数停止SCPI指令解析服务。

接口说明

构造函数 SCPIInterface

函数声明：

```
SCPIInterface(const IDN& idn);
```

函数功能：

创建SCPIInterface类的实例。

参数列表：

参数类型	参数名	参数说明
const IDN&	idn	[in] 仪器标识

添加SCPI指令接口 addCommand

函数声明：

```
void addCommand(const std::string& pattern, const CommandHandler& handler);
```

函数功能：

添加SCPI指令及该指令响应的回调函数。

参数列表：

参数类型	参数名	参数说明
const std::string&	pattern	[in] scpi指令
const CommandHandler&	handler	[in] scpi指令响应的回调函数

结束添加SCPI指令接口 endAdd

函数声明：

```
void endAdd();
```

函数功能：

结束添加SCPI指令。

注意：

当调用addCommand接口添加完所有SCPI指令后，必须调用endAdd接口完成结束添加指令，以便于scpi指令解析器配置其程序执行的上下文当环境。当调用endAdd接口后最好不要再次调用addCommand接口。

开始监听接口 startListen

函数声明：

```
void startListen(const unsigned short port);
```

函数功能：

启动SCPI指令解析服务端，该服务监听指定端口。

参数列表：

参数类型	参数名	参数说明
const unsigned short	port	[in] 服务监听的端口号

结束监听接口 stopListen

函数声明：

```
void stopListen();
```

函数功能：

停止SCPI指令解析服务端。

获取参数接口 getParam

函数声明：

```
template<class T>
bool getParam(T& param) const;
```

函数功能：

获取客户端发送到服务端的参数，一般为设置类型的scpi指令回调函数中调用此接口获取参数。

参数列表：

参数类型	参数名	参数说明
int32_t，uint32_t，int64_t，uint64_t，float，double	param	[out] 存储客户端设置的参数，该参数仅为数值
scpi_number_t	param	[out] 存储客户端设置的参数，该参数可带单位

发送数值接口 sendValue

函数声明：

```
template<class T>
bool sendValue(const T& val) const;
```

函数功能：

向客户端发送数值，一般为查询类型的scpi指令回调函数中调用此接口发送数值。

参数列表：

参数类型	参数名	参数说明
int32_t，int64_t，float，double	val	[in] 发送到客户端的数值

发送数组接口 sendArray

函数声明：

```
template<class T>
bool sendArray(const std::vector<T>& data) const;
```

函数功能：

向客户端发送一组数据，一般为查询类型的scpi指令回调函数中调用此接口发送数据。

参数列表：

参数类型	参数名	参数说明
int8_t , uint8_t , int16_t , uint16_t , int32_t , uint32_t , int64_t , uint64_t , float , double	data	[in] 发送到客户端的数据

数据类型说明

scpi_unit_t

```
enum _scpi_unit_t {
    SCPI_UNIT_NONE,
    SCPI_UNIT_VOLT,
    SCPI_UNIT_AMPER,
    SCPI_UNIT_OHM,
    SCPI_UNIT_HERTZ,
    SCPI_UNIT_CELSIUS,
    SCPI_UNIT_SECOND,
    SCPI_UNIT_METER,
    SCPI_UNIT_GRAY,
    SCPI_UNIT_BECQUEREL,
    SCPI_UNIT_MOLE,
    SCPI_UNIT_DEGREE,
    SCPI_UNIT_GRADE,
    SCPI_UNIT_RADIAN,
    SCPI_UNIT_REVOLUTION,
    SCPI_UNIT_STERADIAN,
    SCPI_UNIT_SIEVERT,
    SCPI_UNIT_FARAD,
    SCPI_UNIT_COULOMB,
    SCPI_UNIT_SIEMENS,
    SCPI_UNIT_ELECTRONVOLT,
    SCPI_UNIT_JOULE,
    SCPI_UNIT_NEWTON,
    SCPI_UNIT_LUX,
    SCPI_UNIT_HENRY,
    SCPI_UNIT_ASTRONOMIC_UNIT,
    SCPI_UNIT_INCH,
    SCPI_UNIT_FOOT,
    SCPI_UNIT_PARSEC,
    SCPI_UNIT_MILE,
    SCPI_UNIT_NAUTICAL_MILE,
    SCPI_UNIT_LUMEN,
    SCPI_UNIT_CANDELA,
    SCPI_UNIT_WEBER,
    SCPI_UNIT_TESLA,
    SCPI_UNIT_ATOMIC_MASS,
    SCPI_UNIT_KILOGRAM,
    SCPI_UNIT_WATT,
    SCPI_UNIT_DBM,
    SCPI_UNIT_ATMOSPHERE,
```

```
    SCPI_UNIT_INCH_OF_MERCURY,
    SCPI_UNIT_MM_OF_MERCURY,
    SCPI_UNIT_PASCAL,
    SCPI_UNIT_TORT,
    SCPI_UNIT_BAR,
    SCPI_UNIT_DECIBEL,
    SCPI_UNIT_UNITLESS,
    SCPI_UNIT_FAHRENHEIT,
    SCPI_UNIT_KELVIN,
    SCPI_UNIT_DAY,
    SCPI_UNIT_YEAR,
    SCPI_UNIT_STROKES,
    SCPI_UNIT_POISE,
    SCPI_UNIT_LITER
};
```