

# 深圳市镭神智能系统 有限公司

## linux\_and\_win\_demo 说明

### 修改记录

版本号	修订日期	修订内容	修订人	备注
v2.1.0	2023/09/27	第一版说明	LSJ	
v2.2.0	2024/07/11	增加详细说明	LSJ	

拟制：

审核：

批准：



## 目录

<b>一、简介 .....</b>	<b>2</b>
1.1 目的 .....	2
1.2 介绍 .....	2
1.3 开发语言 .....	2
1.4 注意 .....	2
<b>二、文件夹结构说明 .....</b>	<b>4</b>
<b>三、 接口函数说明： .....</b>	<b>5</b>
3.1 启动程序 解析函数 .....	5
3.2 获取雷达设备参数信息 .....	9
3.3 修改雷达的参数 .....	12
<b>四、编译运行说明 .....</b>	<b>18</b>
4.1 Windows 系统 编译 .....	18
4.2 Linux 系统 编译 .....	22

## 一、简介

### 1.1 目的

linux\_and\_win\_demo 开发的 API 以及使用 demo 的方式说明。

### 1.2 介绍

linux\_and\_win\_demo 是针对深圳市镭神智能系统有限公司雷达用于进行二次开发使用的对应解析实例的 C/C++ 代码；使用者可根据代码的解析获取雷达一帧的数据，并进行二次的开发；

### 1.3 开发语言

linux\_and\_win\_demo 是基于 C/C++ 语言进行开发。

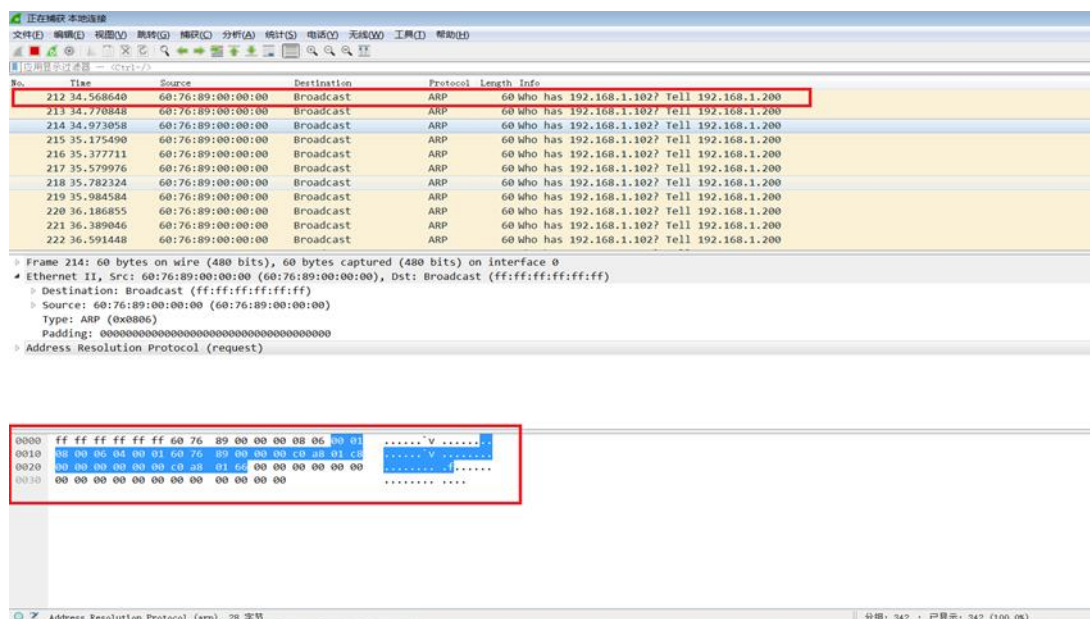
### 1.4 注意

表 1.1 雷达默认网络配置

	IP 地址	UDP 设备包端口号	UDP 数据包端口号
雷达	192.168.1.200	2368（固定不可配）	2369（固定不可配）
电脑	192.168.1.102	2369	2368

1) 雷达的初始 IP 地址为 192.168.1.200，目的 IP 地址为 192.168.1.102，默认目的数据包端口为 2368，设备包端口为 2369。如若未更改，使用雷达前，需要先在网络连接设置中设置以太网为固定 IP 192.168.1.102，子网掩码 255.255.255.0。用户可在使用时更新需要更改雷达 IP 地址，此时与雷达连接的 PC 等控制端设备 IP 也应进行相应修改以保证处于同一网段下。

如若雷达已经更改了 IP 地址，连接雷达时，电脑与雷达的 IP 在不同网段时，需要设置网关；相同网段时，设置不同 IP 即可，例如：192.168.1.x，子网掩码为 255.255.255.0。若需查找雷达的以太网配置信息，连接雷达后电脑可以使用 Wireshark 软件抓取设备 ARP 包进行分析，有关 ARP 包的特征识别，见下图。



注意: Wireshark 软件为第三方软件, 客户使用过程中造成的版权和商业纠纷等问题, 均与镭神智能无关。

2) 使用 SDK 时请禁用所有虚拟网卡, 防止干扰数据接收。

3) 使用 Visual studio 开发时, 优先使用 release 模式以减少数据阻塞、读写延时造成的问题。

## 二、文件夹结构说明

名称	修改日期	类型	大小
bin	2024/7/1 16:37	文件夹	
build	2024/7/1 16:37	文件夹	
demo	2024/7/1 16:38	文件夹	
doc	2024/7/1 17:04	文件夹	
Include	2024/7/1 16:38	文件夹	
lib	2024/7/1 16:38	文件夹	
CMakeLists.txt	2024/7/1 16:40	TXT 文件	3 KB

bin	编译生成的可执行文件	
build	编译的目录	
Include	头文件和源文件的目录	
doc	说明文档的路径	
demo：使用示例，调用启动 获取雷达的示例  包含三个例子：	main. cpp	一般示例，获取点云的文件， 输出点云数据
	main_PCL. cpp	获取点云的文件，增加 PCL 可视化的库，显示点云
	main_PCL_Pcap. cpp	获取点云的文件，增加 PCL 可视化的库，pcap 的离线解 析示例，显示点云：  <b>注意：离线的文件要放到 demo/PcapPacketPath 文件 夹内并在源文件中修改对应 的文件 main_PCL_Pcap. cpp 内修改离线包的名称</b>
CMakeLists.txt	CMake 文件，配置编译项目 配置编译规则	



## 三、接口函数说明:

### 3.1 启动程序 解析函数

`GetLidarData* m_GetLidarData = new GetLidarData_LS;` //使用的雷达, 初始化使用的雷达型号

#### 3.1.1: 设置端口和 IP:

```
void setPortAndIP(uint16_t mDataPort = 2368, uint16_t mDevPort = 2369, std::string mDestIP = "192.168.1.102", std::string mLidarIP = "192.168.1.200", std::string mGroupIp = "226.1.1.102");
```

/\*

功能说明: 初始化数据包端口、设备端口、目的 IP、雷达 IP 和组播 IP, 传入参数, 使程序能够获取到雷达的数据包和设备包;

输入输出参数:

参数 1: mDataPort	数据包端口号	2368 (by default)
参数 2: mDevPort	设备包端口号	2369 (by default)
参数 3: mDestIP	目的 IP	"192.168.1.102" (by default)
参数 4: mLidarIP	雷达 IP	"192.168.1.200" (by default)
参数 5: mGroupIp	组播 IP	"226.1.1.102" (by default)
返回值:	无	

调用示例: `GetLidarData->setPortAndIP(2368, 2369, "192.168.1.102", "192.168.1.200", "226.1.1.102");`

注意: 在函数 `LidarStart()` 开始启动之前, 先调用 `setPortAndIP()` 函数指定当前雷达数据包的参数, 程序才能获取到雷达的数据;

\*/

#### 3.1.2: 设置回调函数: 获取雷达数据

```
void setCallbackFunction(FunDataPrt*);
```

/\*

功能说明: 传输回调函数, 雷达一帧数据完成解析后, 调用回调函数传输获取雷达数据

输入输出参数:

参数 1: FunDataPrt:	定义 回调函数, 传入回调函数指针
返回值:	无

定义的回调函数类型

```
typedef std::function<void(std::shared_ptr<std::vector<MuchLidarData>>, int, std::string)> FunDataPrt;
```

调用示例:

(1) 定义回调函数:

```
void callbackFunction(std::shared_ptr<std::vector<MuchLidarData>>, int, std::string);
```

```
FunDataPrt fun = std::bind(callbackFunction, std::placeholders::_1, std::placeholders::_2, std::placeholders::_3);
```

调用函数输入 回调函数指针: `GetLidarData->setCallbackFunction(&fun);`



(2) 查看回调函数 `callbackFunction(std::shared_ptr<std::vector<MuchLidarData>> PerData, int, std::string);`  
获取一帧的数据

注意：获取雷达的数据有两种方法：其中一种是使用回调函数的方式、另外一种 是使用标志位的方式；  
只有使用回调函数 方法才需要 传输回调函数；

\*/

### 3.1.3: 启动程序 开始获取雷达数据

```
void LidarStart();
```

/\*

功能说明： 启动程序， 开始获取雷达数据包和设备包，解析雷达的数据

输入输出参数：

参数 1: 无

返回值: 无

调用示例： `GetLidarData->LidarStart();`

注意：在函数 `LidarStart()` 开始启动之前，先调用 `setPortAndIP()` 函数指定当前雷数据包参数， 程序才能获取到雷达的数据；

\*/

### 3.1.4: 停止程序 停止获取雷达数据

```
void LidarStop();
```

/\*

功能说明： 停止程序， 停止获取雷达数据包和设备包， 停止线程解析数据

输入输出参数：

参数 1: 无

返回值: 无

调用示例： `GetLidarData->LidarStop();`

注意：无

\*/

### 3.1.5: 调用接口，获取雷达一帧的数据

```
bool getLidarPerFrameData(std::shared_ptr<std::vector<MuchLidarData>>& preFrameData, std::string& Info);
```

/\*

功能说明： 获取一帧的点云数据，

输入输出参数： SDK 中定义了 每个数据点的 信息输出，可获取的点云信息

```
typedef struct _MuchLidarData
```

```
{
```

```
    float X = 0.0;           //坐标 X 值
```

```
    float Y = 0.0;           //坐标 Y 值
```

```
    float Z = 0.0;           //坐标 Z 值
```

```
    int ID = 0;               //通道号
```

```
    float H_angle = 0.0;      //水平角度
```

```
    float V_angle = 0.0;      //垂直角度
```



```
float Distance = 0.0;          //距离值
int Intensity = 0;             //强度值
u_int64 Mtimestamp_nsce = 0;   //时间戳
}MuchLidarData;
```

参数 1: preFrameData: 传入获取一帧雷达参数的引用, 返回一帧的点云数据

参数 2: Info: 传入字符参数, 获取函数调用信息, 返回值为 false 时, 输出获取失败的信息

返回值: bool 值; true, 成功获取到雷达的一帧数据 ; false: 获取失败, 查看 Info 消息值

调用示例:

(1) 判断 isFrameOK 是否为 true: 程序解析完成一帧的标记;

(2) isFrameOK = true 时, 定义:

```
std::shared_ptr<std::vector<MuchLidarData>> m_LidarData_temp;
std::string mInfo;
```

调用: (3) getLidarPerFrameDate(LidarData, mInfo); 获取雷达数据

参考: main.cpp 的 调用

```
while (true)
{
    //method one, get one frame of data
    if (m_GetLidarData->isFrameOK)
    {
        std::shared_ptr<std::vector<MuchLidarData>> m_LidarData_temp;
        std::string mInfo;
        if (!m_GetLidarData->getLidarPerFrameDate(m_LidarData_temp, mInfo))
        {
            std::cout << mInfo << std::endl;
            std::this_thread::sleep_for(std::chrono::milliseconds(1));
            continue;
        }

        //output the number of point cloud
        std::cout << m_LidarData_temp->size() << std::endl;
    }
    else
    {
        std::this_thread::sleep_for(std::chrono::milliseconds(1));
    }
}
```

注意: 获取雷达的数据有两种方法: 这是使用标志位的方式; 通过标志位, 不断获取雷达参数;

\*/



## 3.1.6: 完整的初始化 + 获取雷达数据的过程

(可参数 `./demo/main.cpp`) 通过标志位获取数据方法

```
GetLidarData* m_GetLidarData = new GetLidarData_LS; //初始化使用的雷达型号
m_GetLidarData->setPortAndIP(2368, 2369, "192.168.1.102", "192.168.1.200", "226.1.1.102"); //设置参数
m_GetLidarData->LidarStart(); //启动程序获取雷达数据，解析
while (true)
{
    if (m_GetLidarData->isFrameOK)
    {
        std::shared_ptr<std::vector<MuchLidarData>> m_LidarData_temp;
        std::string mInfo;
        if (!m_GetLidarData->getLidarPerFrameData(m_LidarData_temp, mInfo))
        {
            std::cout << mInfo << std::endl;
            std::this_thread::sleep_for(std::chrono::milliseconds(1));
            continue;
        }

        //output the number of point cloud
        std::cout << m_LidarData_temp->size() << std::endl;
    }
    else
    {
        std::this_thread::sleep_for(std::chrono::milliseconds(1));
    }
}
m_GetLidarData->LidarStop(); //停止程序，停止解析雷达数据
```

注意：输出的目前只输出雷达每一帧的点数，查看 `MuchLidarData` 点的数据类型，需要获取每个点的信息



## 3.2 获取雷达设备参数信息

**3.2.1: 定义雷达输出 雷达参数信息的数据类型，默认值是-1: 以及返回值的主要信息判断 或者单位**

`typedef struct _LidarStateParam`

```
{  
    std::string LidarIP = "-1,-1,-1,-1";           //雷达 IP  
    std::string ComputerIP = "-1,-1,-1,-1";        //目的 IP  
    std::string NtpIP = "-1,-1,-1,-1";             //NTP IP  
    std::string GatewayIP = "-1,-1,-1,-1";         //网关 IP  
    std::string SubnetMaskIP = "-1,-1,-1,-1";      //子网掩码 IP  
    c MacAddress = "-1,-1,-1,-1,-1,-1";           //雷达的物理 MAC 地址 50 3E 7C ** ** **  
    int DataPort = -1;                             //数据包端口  
    int DevPort = -1;                              //设备包端口  
  
    float ReceiverTemperature = -1.0;              //雷达温度 单位: :°  
    float ReceiverHighVoltage = -1.0;              //接收板高压 单位: V  
    float MotorSpeed = -1;                         //转速  
    int FrameRateMode = -1;                        //帧率模式 模式 0、1、2  
  
    int PTP_State = -1;                            //PTP 状态: 1:丢失; 0:未丢失  
    int GPS_State = -1;                            //GPS 状态: 1:丢失; 0:未丢失  
    int PPS_State = -1;                            //PPS 状态: 1:丢失; 0:未丢失  
    int StandbyMode = -1;                          //休眠状态 0:正常 1: 休眠  
    int Clock_Source = -1;                         //授时源: 0:GPS; 1: PTP_L2; 2:NTP; 3:PTP_UDPV4  
    int PhaseLockedSwitch = -1;                    //相位锁: 0:关闭; 1: 开启  
    float PhaseLockedAngle = -1;                   //相位锁 相位锁角度 单位°  
    int PhaseLockedState = -1;                     //相位锁状态: 0:未锁住; 1: 锁住  
    std::string FaultCode = "-1";                  //故障码 4 个字节,按位显示, 每一位代表一个故障状态  
    double RunningTime = -1;                       //运行时间 单位: 小时  
}  
}LidarStateParam;
```

注意：默认参数是 -1，如果其他项有值，但某一项 是 -1，说明此款雷达没有此类参数 输出；

### 3.2.2: 获取雷达参数的接口

`bool getLidarParamState(LidarStateParam& mLidarStateParam, std::string& InfoString);`

`/*`

功能说明： 获取当前雷达内部的参数返回值

输入输出参数：

参数 1: mLidarStateParam: 传入获取雷达参数自定义数据类型 的引用，返回当前获取的雷达参数

参数 2: InfoString: 传入字符串参数， 获取函数调用信息， 返回值为 false 时， 输出获取失败的信息

返回值: bool 值; true, 成功获取到雷达的一帧数据 ; false:获取失败, 查看 Info 消息值

调用示例：

(1) 定义参数类型：



```
LidarStateParam mLidarStateParam;  
std::string mInfo1;
```

(2) 调用接口传入引用，输出

```
if (!m_GetLidarData->getLidarParamState(mLidarStateParam, mInfo1))  
{  
    std::cout << mInfo1 << std::endl;  
}  
else  
{  
    std::cout << "LidarIP    =    " << mLidarStateParam.LidarIP << std::endl;  
}
```

参考：main.cpp 的 调用

### 3.2.3：完整的初始化 + 获取雷达参数的过程

```
GetLidarData* m_GetLidarData = new GetLidarData LS;           //初始化使用的雷达型号  
m_GetLidarData->setPortAndIP(2368, 2369, "192.168.1.102", "192.168.1.200", "226.1.1.102"); //设置参数  
m_GetLidarData->LidarStart();           //启动程序获取雷达数据，解析  
  
//只获取一次参数  
LidarStateParam mLidarStateParam;  
std::string mInfo1;  
std::this_thread::sleep_for(std::chrono::milliseconds(2000)); //启动 雷达后需要等待 2s 在获取雷达数据;  
if (!m_GetLidarData->getLidarParamState(mLidarStateParam, mInfo1)) //获取雷达参数调用 判断返回值  
{  
    std::cout << mInfo1 << std::endl;           //失败打印信息  
}  
else  
{  
    std::cout << "***** Lidar Parameters Display Start *****" << std::endl;  
  
    std::cout.width(20); std::cout << "LidarIP    =    " << mLidarStateParam.LidarIP << std::endl;  
    std::cout.width(20); std::cout << "ComputerIP  =    " << mLidarStateParam.ComputerIP << std::endl;  
    std::cout.width(20); std::cout << "GatewayIP   =    " << mLidarStateParam.GatewayIP << std::endl;  
    std::cout.width(20); std::cout << "SubnetMaskIP =    " << mLidarStateParam.SubnetMaskIP << std::endl;  
    std::cout.width(20); std::cout << "DataPort    =    " << mLidarStateParam.DataPort << std::endl;  
    std::cout.width(20); std::cout << "DevPort     =    " << mLidarStateParam.DevPort << std::endl;  
  
    std::cout.width(20); std::cout << "ReceiverTemperature =    " << mLidarStateParam.ReceiverTemperature << std::endl;  
    std::cout.width(20); std::cout << "ReceiverHighVoltage =    " << mLidarStateParam.ReceiverHighVoltage << std::endl;  
    std::cout.width(20); std::cout << "MotorSpeed    =    " << mLidarStateParam.MotorSpeed << std::endl;  
    std::cout.width(20); std::cout << "PTP_State     =    " << mLidarStateParam.PTP_State << std::endl;  
    std::cout.width(20); std::cout << "GPS_State     =    " << mLidarStateParam.GPS_State << std::endl;  
    std::cout.width(20); std::cout << "PPS_State     =    " << mLidarStateParam.PPS_State << std::endl;
```



```
std::cout.width(20); std::cout << "StandbyMode    =    " << mLidarStateParam.StandbyMode << std::endl;
std::cout.width(20); std::cout << "Clock_Source  =    " << mLidarStateParam.Clock_Source << std::endl;
std::cout.width(20); std::cout << "PhaseLockedSwitch =    " << mLidarStateParam.PhaseLockedSwitch << std::endl;
std::cout.width(20); std::cout << "PhaseLockedState=    " << mLidarStateParam.PhaseLockedState << std::endl;
std::cout.width(20); std::cout << "FaultCode    =    " << mLidarStateParam.FaultCode << std::endl;
std::cout.width(20); std::cout << "RunningTime   =    " << mLidarStateParam.RunningTime << std::endl;

std::cout << "" << "***** Lidar Parameters Display End *****" << std::endl << std::endl;
}
```

注意：由于设备包时 1s 一个，需要再程序启动 ;LidarStart() 后 间隔 1s 以上在获取数据，上面 为了保证获取雷达参数，设定了 2s;

### //循环获取参数

```
While(true)
{
    LidarStateParam mLidarStateParam;
    std::string mInfo1;
    if (!m_GetLidarData->getLidarParamState(mLidarStateParam, mInfo1))
    {
        std::cout << mInfo1 << std::endl;
    }
    else
    {
        std::cout << "ReceiverTemperature    =    " << mLidarStateParam.ReceiverTemperature << std::endl;
    }
    std::this_thread::sleep_for(std::chrono::milliseconds(1000))    //休眠 1s;
}
```

### 3.2.4: 获取雷达数据包当前状态信息

```
std::string getDataPacketState();
/*
```

功能说明： 获取 当前获取的数据包是否有异常,判断程序获取的数据有无错误

输入输出参数：

参数 1: 无

返回值: std::string: 返回字符说明信息

调用示例: std::string infoStr = getDataPacketState();

```
std::cout << "infoStr = " << infoStr << std::endl;
```

注意：启动程序获取雷达数据 LidarStart() 后 ， 在调用接口获取雷达参数

\*/



### 3.2.5: 获取雷达数据包当前状态信息

```
std::string getDevPacketState();
```

/\*

功能说明： 获取 当前获取的设备包是否有异常, 判断程序 是否正常获取到设备包信息

输入输出参数:

参数 1: 无

返回值: std::string: 返回字符说明信息

调用示例: std::string infoStr = getDevPacketState();

```
std::cout << "infoStr = " << infoStr << std::endl;
```

注意: 启动程序获取雷达数据 LidarStart() 后 , 在调用接口获取雷达参数

## 3.3 修改雷达的参数

### 3.3.1: 修改雷达转速

```
bool setLidarRotateSpeed(int SpeedValue, std::string& InfoString);
```

/\*

功能说明: 修改雷达转速参数;

输入输出参数:

参数 1: SpeedValue 转速值, 支持输入 300, 600, 1200 (分别代表 5hz、10hz、20hz)

参数 2: InfoString: 传入字符参数, 获取设置的信息, 返回值为 false 时, 输出设置失败的信息

返回值: bool 值; true, 成功设置转速 ; false:设置失败, 查看 Info 消息值

调用示例:

```
std::string& InfoString; //创建回读信息字符
```

```
GetLidarData->setLidarRotateSpeed(300, InfoString); //调用接口, 设置 300 转 5hz
```

```
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 setLidarRotateSpeed() ;出现 This version of Lidar does not support \*\*\*'!!! 说明此雷达不支持此函数的调用; (无此功能);

2: 调用 setLidarRotateSpeed() 后需要 调用 sendPackUDP(), 才能发 UDP 包到雷达, 修改雷达参数;

3: 1550nm(LS)系列雷达 不可用, 不支持此功能

\*/

### 3.3.2: 修改雷达 IP

```
bool setLidarIP(std::string IPString, std::string& InfoString);
```

/\*

功能说明: 修改雷达 IP 参数;

输入输出参数:

参数 1: IPString 需要修改的 IP 值, 例如: "192.168.1.200"

参数 2: InfoString: 传入字符参数, 获取设置的信息, 返回值为 false 时, 输出设置失败的信息



返回值: bool 值; true, 成功设置转速 ; false:设置失败, 查看 Info 消息值

调用示例:

```
std::string& InfoString; //创建回读信息字符
GetLidarData->setLidarIP("192.168.1.200", InfoString); //调用接口, 设置 IP
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 setLidarIP () 后需要 调用 sendPackUDP(), 才能发 UDP 包到雷达, 修改雷达参数;

\*/

### 3.3.3: 修改目的 IP

```
bool setComputerIP(std::string IPString, std::string& InfoString);
```

/\*

功能说明: 修改目的 IP 参数;

输入输出参数:

参数 1: IPString 需要修改的 IP 值, 例如: "192.168.1.102"

参数 2: InfoString: 传入字符参数, 获取设置的信息, 返回值为 false 时, 输出设置失败的信息

返回值: bool 值; true, 成功设置转速 ; false:设置失败, 查看 Info 消息值

调用示例:

```
std::string& InfoString; //创建回读信息字符
GetLidarData->setComputerIP("192.168.1.102", InfoString); //调用接口, 设置 IP
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 setComputerIP () 后需要 调用 sendPackUDP(), 才能发 UDP 包到雷达, 修改雷达参数;

\*/

### 3.3.4: 修改 NTP IP

```
bool setNTP_IP(std::string IPString, std::string& InfoString);
```

/\*

功能说明: 修改 NTP IP 参数;

输入输出参数:

参数 1: IPString 需要修改的 IP 值, 例如: "192.168.1.102"

参数 2: InfoString: 传入字符参数, 获取设置的信息, 返回值为 false 时, 输出设置失败的信息

返回值: bool 值; true, 成功设置转速 ; false:设置失败, 查看 Info 消息值

调用示例:

```
std::string& InfoString; //创建回读信息字符
GetLidarData->setNTP_IP("192.168.1.102", InfoString); //调用接口, 设置 IP
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 setNTP\_IP () ;出现 This version of Lidar does not support \*\*\*'!!! 说明此雷达不支持此函数的调用; (无此功能);



2: 调用 setNTP\_IP () 后需要 调用 sendPackUDP(), 才能发 UDP 包到雷达, 修改雷达参数;

\*/

### 3.3.5: 修改 网关 IP

```
bool setGatewayIP(std::string IPString, std::string& InfoString);
```

/\*

功能说明: 修改网关 IP 参数;

输入输出参数:

参数 1: IPString 需要修改的 IP 值, 例如: "192.168.1.254"

参数 2: InfoString: 传入字符参数, 获取设置的信息, 返回值为 false 时, 输出设置失败的信息

返回值: bool 值; true, 成功设置转速; false: 设置失败, 查看 Info 消息值

调用示例:

```
std::string& InfoString; //创建回读信息字符
GetLidarData->setGatewayIP("192.168.1.254", InfoString); //调用接口, 设置 IP
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 setGatewayIP () ;出现 This version of Lidar does not support \*\*\*'!!! 说明此雷达不支持此函数的调用; (无此功能);

2: 调用 setGatewayIP () 后需要 调用 sendPackUDP(), 才能发 UDP 包到雷达, 修改雷达参数;

\*/

### 3.3.6: 修改 子网掩码 IP

```
bool setSubnetMaskIP(std::string IPString, std::string& InfoString);
```

/\*

功能说明: 修改子网掩码 IP 参数;

输入输出参数:

参数 1: IPString 需要修改的 IP 值, 例如: "255.255.255.0"

参数 2: InfoString: 传入字符参数, 获取设置的信息, 返回值为 false 时, 输出设置失败的信息

返回值: bool 值; true, 成功设置转速; false: 设置失败, 查看 Info 消息值

调用示例:

```
std::string& InfoString; //创建回读信息字符
GetLidarData->SetSubnetMaskIP("192.168.1.254", InfoString); //调用接口, 设置 IP
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 setSubnetMaskIP () ;出现 This version of Lidar does not support \*\*\*'!!! 说明此雷达不支持此函数的调用; (无此功能);

2: 调用 setSubnetMaskIP () 后需要 调用 sendPackUDP(), 才能发 UDP 包到雷达, 修改雷达参数;

\*/

### 3.3.7: 修改 数据包端口

```
bool setDataPort(int PortNum, std::string& InfoString);
```

/\*

功能说明: 修改数据包端口;

输入输出参数:

参数 1: PortNum, : 需要修改的数据包端口值, 例如: 2368

参数 2: InfoString: 传入字符参数, 获取设置的信息, 返回值为 false 时, 输出设置失败的信息



返回值: bool 值; true, 成功设置转速 ; false:设置失败, 查看 Info 消息值

调用示例:

```
std::string& InfoString; //创建回读信息字符
GetLidarData->setDataPort(2368, InfoString); //调用接口, 设置端口
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 setDataPort () 后需要 调用 sendPackUDP(), 才能发 UDP 包到雷达, 修改雷达参数;

\*/

### 3.3.8: 修改 设备包端口

```
bool setDevPort(int PortNum, std::string& InfoString);
```

/\*

功能说明: 修改设备包端口;

输入输出参数:

参数 1: PortNum, : 需要修改的数据包端口值, 例如: 2369  
参数 2: InfoString: 传入字符参数, 获取设置的信息, 返回值为 false 时, 输出设置失败的信息  
返回值: bool 值; true, 成功设置转速 ; false:设置失败, 查看 Info 消息值

调用示例:

```
std::string& InfoString; //创建回读信息字符
GetLidarData->setDevPort(2369, InfoString); //调用接口, 设置端口
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 setDevPort () 后需要 调用 sendPackUDP(), 才能发 UDP 包到雷达, 修改雷达参数;

\*/

### 3.3.9: 修改 雷达授时源

```
bool setLidarSourceSelection(int StateValue, std::string& InfoString)
```

/\*

功能说明: 修改 雷达授时源 参数;

输入输出参数:

参数 1: StateValue 需要修改值, 0:GPS; 1: PTP\_L2; 2:NTP; 3:PTP\_UDPv4;  
参数 2: InfoString: 传入字符参数, 获取设置的信息, 返回值为 false 时, 输出设置失败的信息  
返回值: bool 值; true, 成功设置转速 ; false:设置失败, 查看 Info 消息值

调用示例:

```
std::string& InfoString; //创建回读信息字符
GetLidarData->setLidarSourceSelection(0, InfoString); //调用接口, 设置
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 setLidarSourceSelection () ;出现 This version of Lidar does not support \*\*\*'!!! 说明此雷达不支持此函数的调用; (无此功能);



2: 调用 setLidarSourceSelection () 后需要 调用 sendPackUDP (), 才能发 UDP 包到雷达, 修改雷达参数;

\*/

### 3.3.10: 修改 雷达工作状态

```
bool setLidarWorkState(int StateValue, std::string& InfoString)
```

/\*

功能说明: 修改雷达工作状态; 正常模式 或者 低功耗模式 (只发设备包, 不发数据包, 雷达不发光)

输入输出参数:

参数 1: StateValue 需要修改值, 0:正常模式, 1: 低功耗模式;

参数 2: InfoString: 传入字符参数, 获取设置的信息, 返回值为 false 时, 输出设置失败的信息

返回值: bool 值; true, 成功设置转速 ; false:设置失败, 查看 Info 消息值

调用示例:

```
std::string& InfoString; //创建回读信息字符
GetLidarData->setLidarWorkState(0, InfoString); //调用接口, 设置
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 setLidarWorkState () ;出现 This version of Lidar does not support \*\*\*'!!! 说明此雷达不支持此函数的调用; (无此功能);

2: 调用 setLidarWorkState () 后需要 调用 sendPackUDP (), 才能发 UDP 包到雷达, 修改雷达参数;

\*/

### 3.3.11: 修改 雷达帧率切换

```
bool setFrameRateMode(int StateValue, std::string& InfoString)
```

/\*

功能说明: 修改雷达 帧率切换

输入输出参数:

参数 1: StateValue 需要修改值, 0:正常帧率, 1: 50%帧率; 2: 25%帧率

参数 2: InfoString: 传入字符参数, 获取设置的信息, 返回值为 false 时, 输出设置失败的信息

返回值: bool 值; true, 成功设置转速 ; false:设置失败, 查看 Info 消息值

调用示例:

```
std::string& InfoString; //创建回读信息字符
GetLidarData->setFrameRateMode(0, InfoString); //调用接口, 设置
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 setFrameRateMode () ;出现 This version of Lidar does not support \*\*\*'!!! 说明此雷达不支持此函数的调用; (无此功能);

2: 调用 setFrameRateMode () 后需要 调用 sendPackUDP (), 才能发 UDP 包到雷达, 修改雷达参数;

3: 目前仅用于 1550nm(LS) 系列的雷达

\*/

### 3.3.12: 修改 相位锁开关

```
bool setPhaseLockedSwitch(int StateValue, std::string& InfoString)
```



/\*

功能说明： 修改雷达相位锁开关

输入输出参数：

参数 1: StateValue                      需要修改值, 0:是关闭, 1: 开启

参数 2: InfoString: 传入字符参数, 获取设置的信息, 返回值为 false 时, 输出设置失败的信息

返回值: bool 值; true, 成功设置转速 ; false:设置失败, 查看 Info 消息值

调用示例:

```
std::string& InfoString; //创建回读信息字符
GetLidarData->setPhaseLockedSwitch(0, InfoString); //调用接口, 设置相位锁开关
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 setPhaseLockedSwitch () ;出现 This version of Lidar does not support \*\*\*'!!! 说明此雷达不支持此函数的调用;(无此功能);

2: 调用 setPhaseLockedSwitch () 后需要 调用 sendPackUDP(), 才能发 UDP 包到雷达, 修改雷达参数;

3: 目前仅用于 LS 系列的雷达

\*/

### 3.3.13: 发送 UDP 修改的配置包 到 雷达

bool sendPackUDP ()

/\*

功能说明： 发送 配置包 修改雷达参数

输入输出参数：

参数 1: 无

返回值: 无

调用示例:

```
GetLidarData->sendPackUDP(); //调用接口 发送 UDP 包给雷达
```

注意: 1: 调用 以上的修改接口后 都 需要 调用 sendPackUDP(), 才能发 UDP 包到雷达, 修改雷达参数;

2: 可同时修改多个参数后, 最后调用 sendPackUDP() 统一发包修改参数;

\*/

### 3.3.14: 完整的初始化 + 修改雷达参数的过程

```
GetLidarData* m_GetLidarData = new GetLidarData_LS; //初始化使用的雷达型号
m_GetLidarData->setPortAndIP(2368, 2369, "192.168.1.102", "192.168.1.200", "226.1.1.102"); //设置参数
m_GetLidarData->LidarStart(); //启动程序获取雷达数据, 解析
std::this_thread::sleep_for(std::chrono::milliseconds(2000)); //修改 2s 等待程序 获取到设备包
std::string mInfo;
m_GetLidarData->setLidarRotateSpeed(600, mInfo);
m_GetLidarData->setLidarIP("192.168.1.200", mInfo);
m_GetLidarData->setComputerIP("192.168.1.102", mInfo);
m_GetLidarData->setDataPort(2368, mInfo);
m_GetLidarData->setDevPort(2369, mInfo);
m_GetLidarData->setLidarSoureSelection(0, mInfo);
m_GetLidarData->setLidarWorkState(0, mInfo);

m_GetLidarData->sendPackUDP(); //发送 UDP 包
```

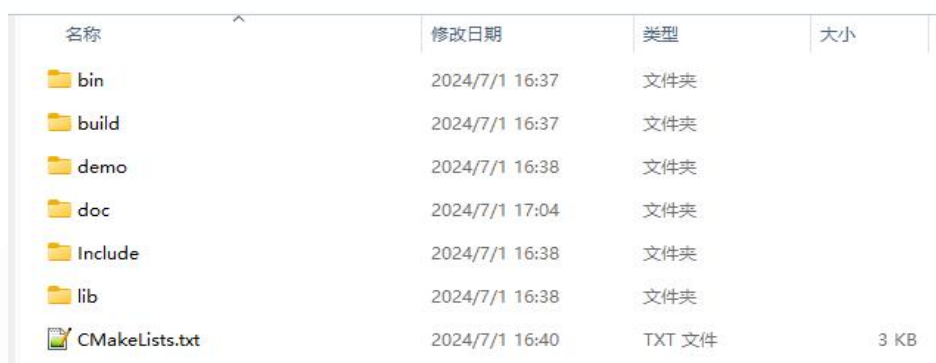
```
m_GetLidarData->LidarStop(); //停止程序，停止解析雷达数据
```

注意：由于设备包时 1s 一个，需要再程序启动 ;LidarStart() 后 间隔 1s 以上获取设备包后 在设置雷达的参数； 为保证能正常设置参数，先休眠 2s;

## 四、编译运行说明

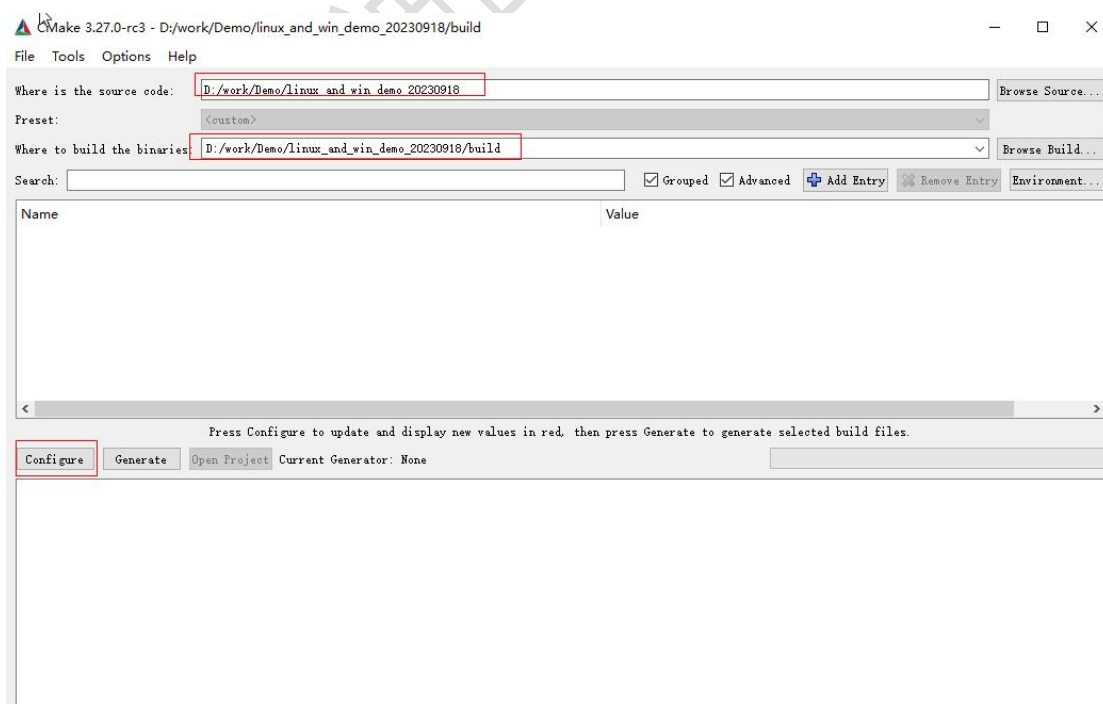
### 4.1 Windows 系统 编译

使用 cmake-gui 配置编译项目，源码目录选择 CMakeList 和源码文件所在目录（linux\_and\_win\_demo），编译工作目录选择所在目录的 build。



名称	修改日期	类型	大小
bin	2024/7/1 16:37	文件夹	
build	2024/7/1 16:37	文件夹	
demo	2024/7/1 16:38	文件夹	
doc	2024/7/1 17:04	文件夹	
Include	2024/7/1 16:38	文件夹	
lib	2024/7/1 16:38	文件夹	
CMakeLists.txt	2024/7/1 16:40	TXT 文件	3 KB

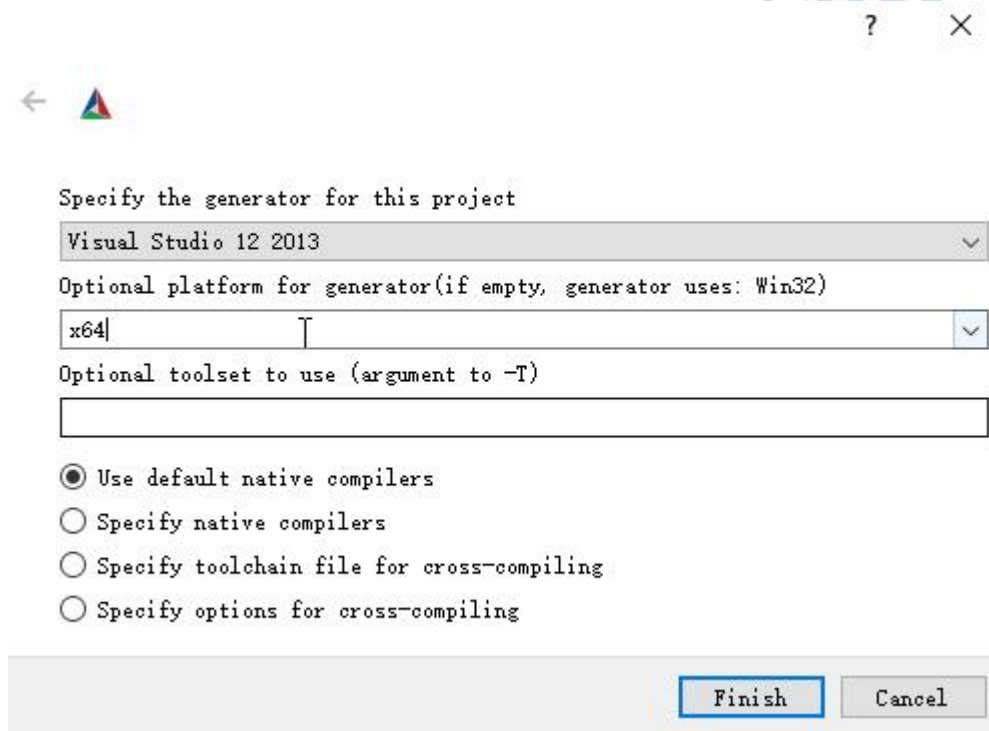
图表 4.1 源码目录下应该包含的文件



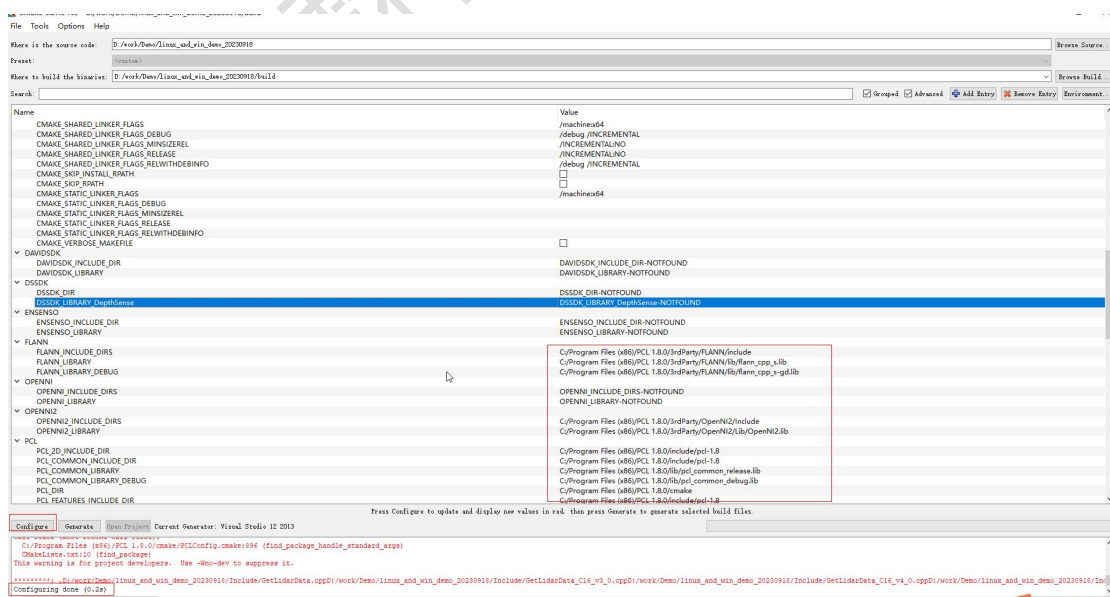
图表 4.2 cmake-gui 配置



点击 **Configure** 按钮配置项目，选择当前使用的编译器版本；

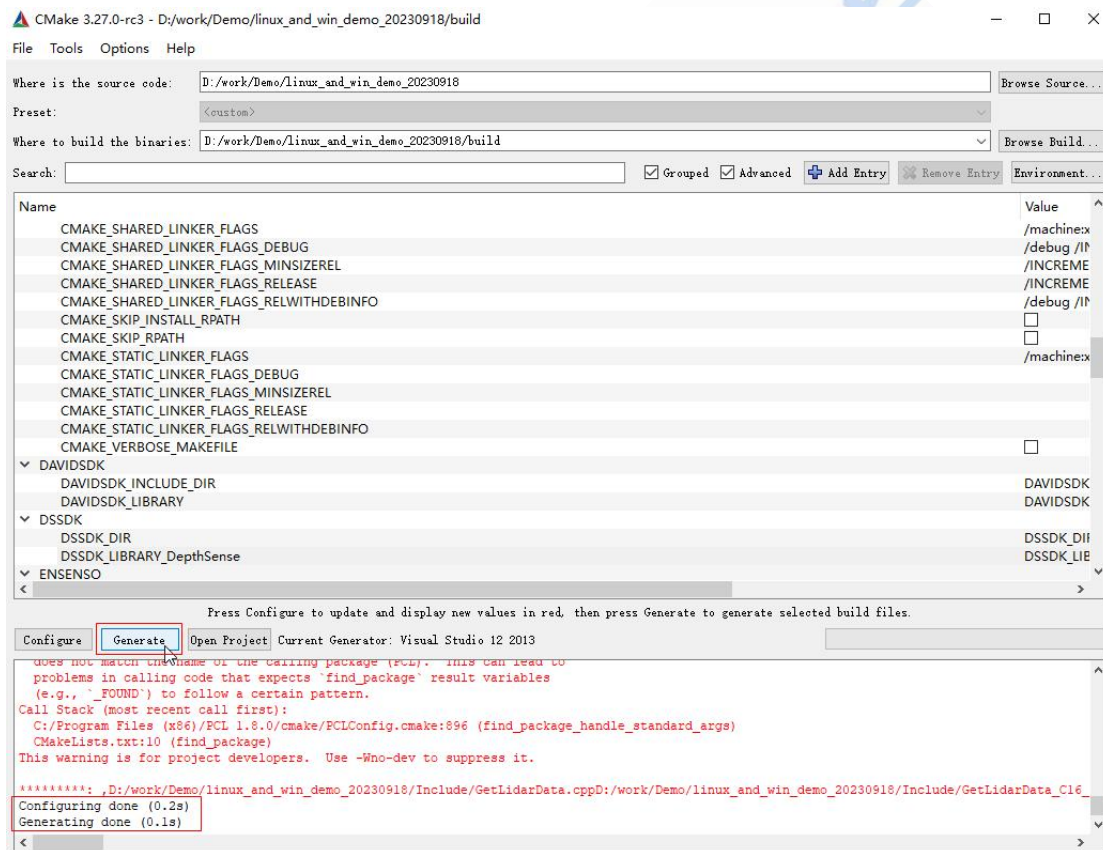


图表 4.3 编译器选择



图表 4.4 PCL 的配置

配置安装的 PCL 版本路径，点击 Configure 按钮配置项目。当出现 Configuring done 提示时表示配置完成。



图表 4.5 Generate 按钮生成项目

点击 Generate 按钮生成项目，出现 Generating done 提示时表示生成项目成功；打开 build 目录就可以看到生成的项目

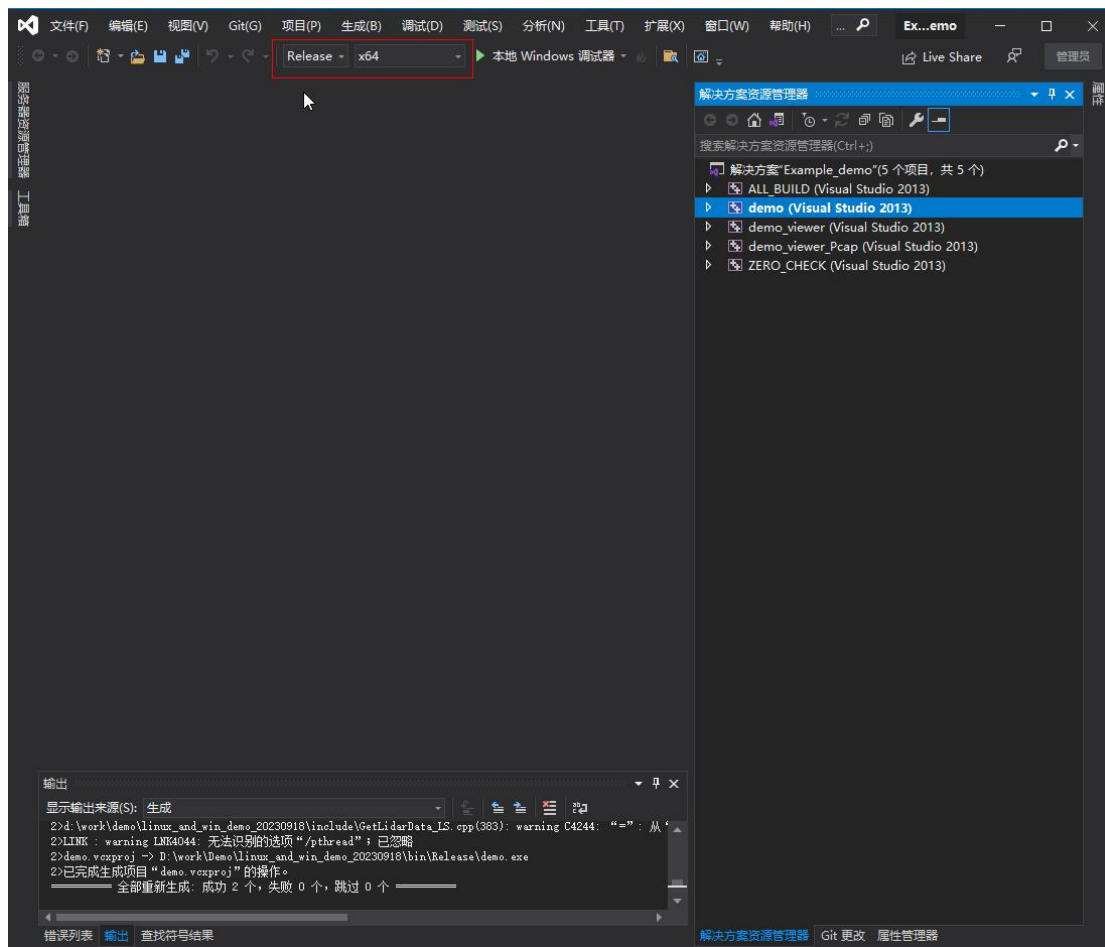
电脑 &gt; 新加卷 (D:) &gt; work &gt; Demo &gt; linux\_and\_win\_demo\_20230918 &gt; build

名称	修改日期	类型	大小
CMakeFiles	2023/9/18 11:36	文件夹	
ALL_BUILD.vcxproj	2023/9/18 11:36	VC++ Project	66 KB
ALL_BUILD.vcxproj.filters	2023/9/18 11:36	VC++ Project Fil...	1 KB
cmake_install.cmake	2023/9/18 11:36	CMAKE 文件	2 KB
CMakeCache.txt	2023/9/18 11:33	TXT 文件	50 KB
demo.vcxproj	2023/9/18 11:36	VC++ Project	84 KB
demo.vcxproj.filters	2023/9/18 11:36	VC++ Project Fil...	1 KB
demo_viewer.vcxproj	2023/9/18 11:36	VC++ Project	184 KB
demo_viewer.vcxproj.filters	2023/9/18 11:36	VC++ Project Fil...	1 KB
demo_viewer_Pcap.vcxproj	2023/9/18 11:36	VC++ Project	184 KB
demo_viewer_Pcap.vcxproj.filters	2023/9/18 11:36	VC++ Project Fil...	1 KB
Example_demo.sln	2023/9/18 11:36	Visual Studio Sol...	6 KB
ZERO_CHECK.vcxproj	2023/9/18 11:36	VC++ Project	65 KB
ZERO_CHECK.vcxproj.filters	2023/9/18 11:36	VC++ Project Fil...	1 KB

图表 4.6 build 目录下生成的文件

点击 Open Project 按钮或打开编译工作目录下\*.sln 文件打开项目。

将项目改为 Release 模式（**推荐**）和对应 32/64 位模式，根据需要编译对应的项目。



图表 4.7 编译对应的项目

此电脑 > 新加卷 (D:) > work > Demo > linux\_and\_win\_demo\_20230918 > bin > Release

名称	修改日期	类型	大小
demo.exe	2023/9/18 11:57	应用程序	103 KB
demo_viewer.exe	2023/9/18 11:57	应用程序	5,013 KB
demo_viewer_Pcap.exe	2023/9/18 11:58	应用程序	5,012 KB

图表 4.8 bin 目录生成对应的示例运行程序

## 4.2 Linux 系统 编译

### ##依赖

如果没有事先安装好 pcl 点云库，需要安装

pcl 点云库： `sudo apt-get install libpcl-dev`

参考：<https://github.com/PointCloudLibrary/pcl>

# 编译和运行

linux 系统下编译过程:

切换到文件夹所在目录

```
cd build
```

```
cmake ..
```

```
make -j4
```

切换到生成可执行文件的文件目录

```
cd ../bin
```

获取数据 demo: `./demo`

可视化 demo\_viewer: `./demo_viewer`

可视化离线包 demo\_viewer\_Pcap: `./demo_demo_viewer_Pcap`

注意: 离线的文件要放到 `./demo/PcapPacketPath` 文件夹中并在源文件 `main_PCL_Pcap.cpp` 中修改对应的文件名, 修改离线包的名称