# Presenter Agent-Server通信

本指导（一指禅）主要是用于帮助您快速入门Presenter Agent与Presenter Server通信，这里的Presenter Agent指的是TCP通信中的客户端，一般在开发版一侧，宝藏一告诉你如何使用Python版本的Presenter Agent，宝藏二告诉你如何使用Python版本的Presenter Server，然后是一个常见问题（FAQ）的URL和常用操作作为附录。

## 宝藏一 Presenter Agent三步走

### 构建TCP通信的客户端对象以及连接

**1.1接口说明**

为了清楚客户类功能特列了如下接口的说明：  
类 PresenterSocketClient

构造函数参数：\_\_init\_\_(self, server\_address, reconnectiontime=5,recvCallback=None)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 参数说明 | 参数类型 | 默认值 | 备注 |
| server\_address | 服务器IP地址和端口号 | (str,int) | 无 | 参数为元组 |
| reconnectiontime | 重连时间(单位：秒) | int | 5 |  |
| recvCallback | 接收数据回调 |  | None | 接收服务器发送的数据 |

连接服务器：start\_connect(self) 返回值类型：无

发送数据参数：send\_data(self, data)返回值类型：无

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 参数说明 | 参数类型 | 默认值 | 备注 |
| data | 要发送的数据 | bytes | 无 |  |

关闭连接：close(self)返回值类型：无

传输数据结构说明

类 ChannelManager

打开通道：OpenChannel(self, channel\_name='video', content\_type=1)返回值类型：bytes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 参数说明 | 参数类型 | 默认值 | 备注 |
| channel\_name | 通道名称 | str | video | image或者video |
| content\_type | 内容类型 | int | 1 | kChannelContentTypeImage = 0;  kChannelContentTypeVideo = 1; |

打包数据：PackRequestData(self, image\_frame)返回值类型：bytes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 参数说明 | 参数类型 | 默认值 | 备注 |
| image\_frame | 图像信息结构图 | imageframe | 无 |  |

ImageFrame结构体参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 参数说明 | 参数类型 | 默认值 | 备注 |
| format | 格式 | ContentType | 0 | 0图像1视频 |
| width | 图像宽 | int | 0 |  |
| height | 图像高 | int | 0 |  |
| size | 图像大小 | int | 0 | 暂时未用上，可以不用赋值 |
| data | 图像数据 | bytes |  |  |
| detection\_results | 边框位置信息列表 | list | [] |  |

DetectionResult结构体参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 参数说明 | 参数类型 | 默认值 | 备注 |
| lt | 左上角坐标 | Point | (0,0) |  |
| rb | 右下角坐标 | Point | (0,0) |  |
| Result\_text | 文本信息 | str | “” |  |

1.2构建客户端对象

使用PresenterSocketClient类直接可以构建客户端对象，传递服务器IP地址和端口号、重连次数、以及接受数据回调的参数即可，然后建立连接，这里注意连接需要开启线程连接，否则会阻塞下面代码的执行，为了有充分建立连接的时间，需要在线程开启后延时一定时间，以下是简单的示例：

import PresenterSocketClient

import time

def recvdata(data):

print(data)

psc = PresenterSocketClient(("192.168.1.246", 7006), 5, recvdata)

threading.Thread(target=psc.start\_connect).start()

time.sleep(0.1)

**代码解读：**上述先定义一个回调函数，然后新建客户端对象，并与Presenter Server连接，最后延时100ms，注意客户端连接前需要先打开Presenter Server，就是服务器开启。客户端才能连接。比如腾讯服务器不开启，你的QQ就不能登录是一个道理。

### 和Presenter Server进行第一次握手，建立传输通道

和Presenter Server正式数据通信之前需要建立一个握手，告诉Server端我俩初次见面，请多关照。如果不事先和Server打招呼，则Server端认为咱们不是“一伙的”，不会进行通信，即客户端发送任何数据，Server端都不会理睬。为了能够正常通信我们进行握手机制分为如下步骤：

（1）和服务器建立连接

（2）打开通道返回通道信息

（3）发送通道信息

（4）正式建立通信

以下是简单的示例：

channel\_manager = ChannelManager.ChannelManager()

data = channel\_manager.OpenChannel()

psc.send\_data(data)

**代码解读：**首先我们构建通道管理器类对象，然后调用OpenChannel打开通道返回一个通道数据，数据类型为bytes。Psc就是1.2节中构建的客户端对象，我们只需要调用send\_data即可完成第一次握手。

### 和Presenter Server进行数据通信

和Presenter Server通信。可以分为3个步骤：

1. 填充Image\_Frame结构体参数，结构体参数具体在1.1小结已经给出，请参考
2. 打包结构体数据  
   （3）发送数据

以下是简单的示例：

image\_data = b'' #图像数据

with open('test.jpg', mode='rb') as f: #打开文件

image\_data = f.read()#读取文件

image\_frame = ImageFrame() #新建结构体对象

image\_frame.format = 0 #Jpeg格式,0表示为Jpeg格式，目前Presenter Server只接收Jpeg图片

image\_frame.width = 300 #图像宽

image\_frame.height = 200 #图像高

image\_frame.data = image\_data # 图形bytes数据

image\_frame.size = 0 #为了与C++版本一样，这个参数保留，没有用上，赋值为0即可

dr= DetectionResult() # BOX1信息

dr.lt.x = 10 # 左上角X

dr.lt.y = 10 # 左上角Y

dr.rb.x = 100 # 右上角X

dr.rb.y = 100 # 右上角Y

dr.result\_text = 'hello!' # BOX1文本信息

image\_frame.detection\_results.append(dr) # 填充到image\_frame结构体中

dr= DetectionResult()# BOX2信息

dr.lt.x = 20 # 左上角X

dr.lt.y = 20 # 左上角Y

dr.rb.x = 200 # 右上角X

dr.rb.y = 200 # 右上角Y

dr.result\_text = 'world' # BOX2文本信息

image\_frame.detection\_results.append(dr)# 填充到image\_frame结构体中

all\_data = channel\_manager.PackRequestData(image\_frame)#打包结构体数据

psc.send\_data(data) #发送数据

**代码解读：**首先我们读取了一张图片，并赋值给image\_data，这里作为示例只是读取一张图片，如果是从摄像头还需要自己转成JPEG图像数据。然后构建通Image\_Frame类对象，依次填写我们要发送的图像信息。如图像的大小、宽高、数据以及需要画框的信息以及画框上的文本信息，数据类型为bytes。Psc就是1.2节中构建的客户端对象，我们只需要调用send\_data即可完成一次数据通信。

## 宝藏二 Presenter Server核心解读和回传数据

本部分将为大家讲解Presenter Server核心接收数据和发送数据规则，由于官方已经提供完成的代码，这里我们解读一下，让大家更加了解Server端的工作机制。然而官方并未给我们提供一个回传数据的案例，本部分，我们通过一个具体示例看看如何传输数据到客户端。

#### Presenter Server核心解读

1.1 Presenter Server接收数据解读

本部分我们以sample-facedetection的Presenter Server核心代码作为解读，首先我们下载sample-facedetection源码，https://github.com/Ascend/sample-facedetection，然后我们打开presenterserver\common\presenter\_socket\_server.py，这个就是我们重点解读的对象文件。首先我们看看核心处理的代码

def \_read\_sock\_and\_process\_msg(self, sock\_fileno, conns, msgs):

'''

Args:

sock\_fileno: a socket fileno, return value of socket.fileno()

conns: all socket connections registered in epoll

msgs: msg read from a socket

Returns:

ret: True or False

'''

# 读取包头

msg\_total\_len, msg\_name\_len = self.\_read\_msg\_head(sock\_fileno, conns)

if msg\_total\_len is None:

logging.error("msg\_total\_len is None.")

return False

# 读取包名

ret, msg\_name = self.\_read\_msg\_name(sock\_fileno, conns, msg\_name\_len)

if not ret:

return ret

# 读取包体

msg\_body\_len = msg\_total\_len - self.msg\_head\_len - msg\_name\_len

if msg\_body\_len < 0:

logging.error("msg\_total\_len:%u, msg\_name\_len:%u, msg\_body\_len:%u",

msg\_total\_len, msg\_name\_len, msg\_body\_len)

return False

ret = self.\_read\_msg\_body(sock\_fileno, conns, msg\_body\_len, msgs)

if not ret:

return ret

# 处理消息

ret = self.\_process\_msg(conns[sock\_fileno], msg\_name, msgs[sock\_fileno])

return ret

**代码解读：**上面我们可以清楚看到Presenter Server接收数据处理步骤：

1. 读取消息头，共计5个字节；
2. 读取消息名，是一个字符串；
3. 读取数据，这个数据是一个Protobuf序列化的bytes数据，其实就是我们在Presenter Agent打包的数据，就是Image\_Frame数据；
4. 对Protobuf序列化的bytes数据进行进一步处理。

读取消息头具体是怎么读取的呢，我们进去 self.\_read\_msg\_head看看

def \_read\_msg\_head(self, sock\_fileno, conns):

'''

Args:

sock\_fileno: a socket fileno

conns: all socket connections which created by server.

Returns:

msg\_total\_len: total message length.

msg\_name\_len: message name length.

'''

ret, msg\_head = self.\_read\_socket(conns[sock\_fileno], self.msg\_head\_len)

if not ret:

logging.error("socket %u receive msg head null", sock\_fileno)

return None, None

# in Struct(), 'I' is unsigned int, 'B' is unsigned char

msg\_head\_data = struct.Struct('IB')

(msg\_total\_len, msg\_name\_len) = msg\_head\_data.unpack(msg\_head)#解包数据

msg\_total\_len = socket.ntohl(msg\_total\_len)

return msg\_total\_len, msg\_name\_len

**代码解读：**我们可以看到首先读取了5个字节数据，然后用了一个结构体类包数据到元组，就可以直接得到消息总长度和消息名长度，然后消息总长度并不是是一个整数，需要字节序列反解才是真正的数据长度。

def \_read\_msg\_name(self, sock\_fd, conns, msg\_name\_len):

'''

Args:

sock\_fd: a socket fileno

conns: all socket connections which created by server.

msg\_name\_len: message name length.

Returns:

ret: True or False

msg\_name: message name.

'''

ret, msg\_name = self.\_read\_socket(conns[sock\_fd], msg\_name\_len)

if not ret:

logging.error("socket %u receive msg name null", sock\_fd)

return False, None

try:

msg\_name = msg\_name.decode("utf-8")

except UnicodeDecodeError:

logging.error("msg name decode to utf-8 error")

return False, None

return True, msg\_name

**代码解读：**在传入消息名长度参数后，socket直接读取指定消息名长度数据，然后调用decode函数解码即可得到消息名。

def \_read\_msg\_body(self, sock\_fd, conns, msg\_body\_len, msgs):

'''

Args:

sock\_fd: a socket fileno

conns: all socket connections which created by server.

msg\_name\_len: message name length.

msgs: msg read from a socket

Returns:

ret: True or False

'''

ret, msg\_body = self.\_read\_socket(conns[sock\_fd], msg\_body\_len)

if not ret:

logging.error("socket %u receive msg body null", sock\_fd)

return False

msgs[sock\_fd] = msg\_body

return True

**代码解读：**在传入消息体长度参数后，socket直接读取指定消息体长度数据，然后传给msgs[sock\_fd] ，msg\_body就是Protobuf序列化的bytes数据。

综上可以看出，整个从Presenter Agent发来的数据为：

包头+包名+包体

其中包头=结构体打包((字节序列(消息总长度), 消息名长度))

包名=消息名称,由protbuf定义

包体=序列化的Image\_Frame结构体bytes数据

1.2 Presenter Server 发送数据解读

Presenter回传数据只要调用了sned\_message函数。定义如下

def send\_message(self, conn, protobuf, msg\_name):

'''

API for send message

Args:

conn: a socket connection.

protobuf: message body defined in protobuf.

msg\_name: msg name.

Returns: NA

'''

message\_data = protobuf.SerializeToString() #将protobuf数据序列bytes

message\_len = len(message\_data)#求序列化的长度

msg\_name\_size = len(msg\_name) #消息名长度

msg\_total\_size = self.msg\_head\_len + msg\_name\_size + message\_len#消息总长度

# in Struct(), 'I' is unsigned int, 'B' is unsigned char

s = struct.Struct('IB')

msg\_head = (socket.htonl(msg\_total\_size), msg\_name\_size)#包头是个元组类型，其中消息总长度被字节序列

packed\_msg\_head = s.pack(\*msg\_head) #打包结构体

msg\_data = packed\_msg\_head + \

bytes(msg\_name, encoding="utf-8") + message\_data #消息名被encode

conn.sendall(msg\_data) # socket发送数据到Presenter Agent端

**代码解读：**从发送的函数可以看出，Presenter Server接收的数据结构和发送到Presenter Agent数据结构一样。我们必须遵循这个结构，否则我们需要自己定义一套TCP通信规则完成通信任务。

#### Presenter Server回传数据

本部分通过一个具体示例讲解，如何从Presenter Server回传数据到Presenter Agent。我们仍然以宝藏一第3小结作为示例，我们读取一张图片，然后由Presnter Server接收再转发回来保存这个图片。经过宝藏一的详细介绍，我们知道Agent和Server通信数据结构一样，因此我们只需要将Server处理消息的结构代码搬移到Agent稍微改下即可正常运行。请看下面片段代码，后面我们将给出完整待代码和程序案例给大家研究。

def \_\_start\_listenning(self):

while self.\_bstart:

try:

self.\_read\_sock\_and\_process\_msg()

except Exception as e:

print(e)

self.\_sock\_client.close()

self.start\_connect()

break

def \_read\_socket(self, read\_len):# 读取指定长度read\_len的socket字节数据

has\_read\_len = 0

read\_buf = b''

total\_buf = b''

while has\_read\_len != read\_len:

try:

read\_buf = self.\_sock\_client.recv(read\_len - has\_read\_len)

except socket.error:

print("socket error")

return False, None

if read\_buf == b'':

return False, None

total\_buf += read\_buf

has\_read\_len = len(total\_buf)

return True, total\_buf

def \_read\_msg\_head(self,read\_len):# 读取包头

ret, msg\_head = self.\_read\_socket(read\_len)

print("msg head data is :", msg\_head)

if not ret:

print("socket receive msg head null")

return None, None

# in Struct(), 'I' is unsigned int, 'B' is unsigned char

msg\_head\_data = struct.Struct('IB')

(msg\_total\_len, msg\_name\_len) = msg\_head\_data.unpack(msg\_head)

msg\_total\_len = socket.ntohl(msg\_total\_len)

print("msg total length is :",msg\_total\_len)

print("msg name is :", msg\_name\_len)

return msg\_total\_len, msg\_name\_len

def \_read\_msg\_name(self, msg\_name\_len): # 读取包名

ret, msg\_name = self.\_read\_socket(msg\_name\_len)

print("direct msg name is :", msg\_name)

if not ret:

print("socket receive msg name null")

return False, None

try:

msg\_name = msg\_name.decode("utf-8")

print("decode msg name is :", msg\_name)

except Exception as e:

print("msg name decode to utf-8 error")

return False, None

return True, msg\_name

def \_read\_msg\_body(self, msg\_body\_len):# 读取包体

print("msg body length is :", msg\_body\_len)

ret, msg\_body = self.\_read\_socket(msg\_body\_len)

if not ret:

print("socket receive msg body null")

return False,None

return True, msg\_body

def \_read\_sock\_and\_process\_msg(self): #处理消息

# Step1: read msg head

msg\_total\_len, msg\_name\_len = self.\_read\_msg\_head(5)

if msg\_total\_len is None:

print("msg\_total\_len is None.")

return False

# Step2: read msg name

ret, msg\_name = self.\_read\_msg\_name(msg\_name\_len)

if not ret:

return ret

# Step3: read msg body

msg\_body\_len = msg\_total\_len - 5 - msg\_name\_len

if msg\_body\_len < 0:

print("msg\_total\_len is 0")

return False

ret, msg\_body = self.\_read\_msg\_body(msg\_body\_len)

if not ret:

return ret

# Step4: process msg

ret = self.\_process\_msg(msg\_name, msg\_body)

return ret

def \_process\_msg(self, msg\_name,msg\_data):#保存我们从Presenter Server发来的数据

if msg\_name == pb2.\_PRESENTIMAGEREQUEST.full\_name:

request = pb2.PresentImageRequest()

try:

print("ParseFromString start")

print("msg\_data type is :", type(msg\_data))

request.ParseFromString(msg\_data)

except Exception as e:

print("ParseFromString exception: Error parsing message")

return

with open('result.jpg','wb') as f:

f.write(request.data)

代码解读：通过上面的改写，我们明显看到，我们可以直接从Presenter Server接收处理消息的代码复制到Agent端稍作修改就可以完成接收数据的任务。为了发送数据到Agent，我们只需要在presenterserver\face\_detection\src\face\_detection\_server.py第173行下面添加一条发送代码即可，看下面标红的数据。

try:

request.ParseFromString(msg\_data)

except DecodeError:

logging.error("ParseFromString exception: Error parsing message")

err\_code = pb2.kPresentDataErrorOther

return self.\_response\_image\_request(conn, response, err\_code)

self.send\_message(conn, request, pb2.\_PRESENTIMAGEREQUEST.full\_name)

sock\_fileno = conn.fileno()

完整的代码案例请参考https://github.com/futureflsl/TcpCommunication

## 常见问题（FAQ）

如果遇到一些问题，可以在如下一站式FAQ中查找问题解决办法；

FAQ：[http://122.112.191.122:4000](http://122.112.191.122:4000/)

## 求助渠道（昇腾社区）

可以在社区中查找解决方案或者提问题求助：

<https://bbs.huaweicloud.com/forum/forum-949-1.html>

## 附录（常用操作）：

#### USB虚拟网卡IP配置（for linux）

以普通用户登录Ubuntu服务器，执行如下命令切换到root用户。

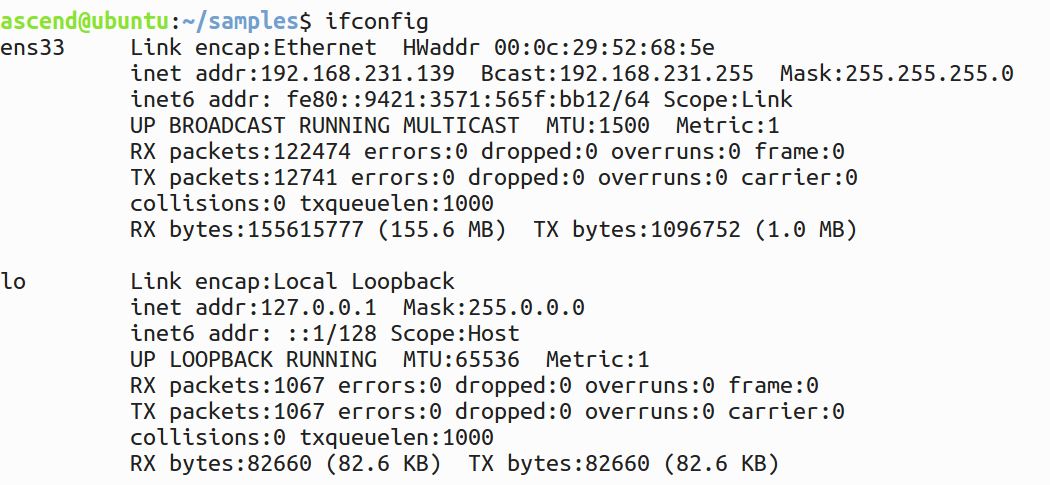
su - root

获取USB网卡名

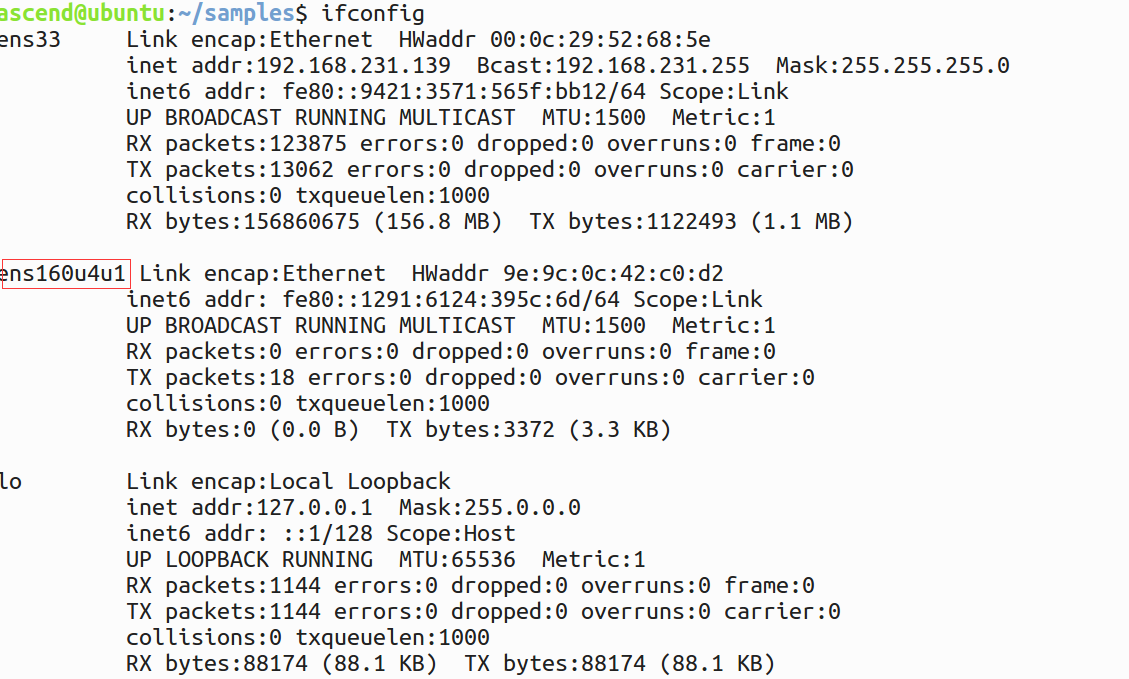
ifconfig -a

若系统中有多个USB网卡，可以通过拔插开发者板进行判定。

例如，在USB线没有插入主机前（USB线另外一端连接开发板）：



USB线插入主机后，如下的ens160u4u1就是你插入后多出来的一个网卡，但此时该网卡还未分配IP：



在“/etc/network/interfaces”文件中添加USB网卡的静态IP。

执行如下命令打开interfaces文件；

vi /etc/network/interfaces

并在文末追加如下内容（例如：为ens160u4u1配置静态IP为192.168.1.123）：

auto ens160u4u1

iface ens160u4u1 inet static

address 192.168.1.123

netmask 255.255.255.0

（**注意**：USB网卡名一定要与你实际查出来的一致，配置静态IP为 192.168.1.xxx网段即可，自己可以改，另外，如果192.168.1.xxx网段已经被用于你的局域网通讯，为了避免IP冲突，在配置并连接成功后需要解决冲突：[https://bbs.huaweicloud.com/forum/forum.php?mod=viewthread&tid=16966&page=1&extra=#pid56395](https://bbs.huaweicloud.com/forum/forum.php?mod=viewthread&tid=16966&page=1&extra=" \l "pid56395)）

修改NetworkManager.conf文件，避免重启后网络配置失效。

若是Ubuntu Sever版本，此步骤请忽略。

执行如下命令打开“NetworkManager.conf”文件。

vi /etc/NetworkManager/NetworkManager.conf

修改文件中的“managed=false”为“managed=true” 。

配置静态IP生效

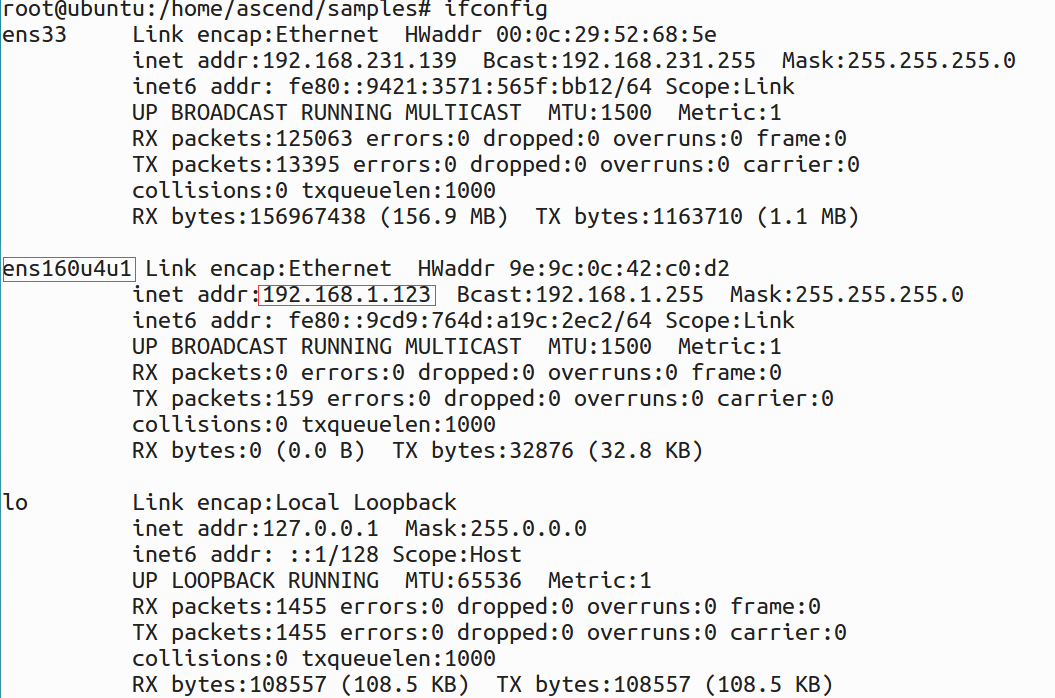
执行以下命令：

ifdown enp0s20f0u4

ifup enp0s20f0u4

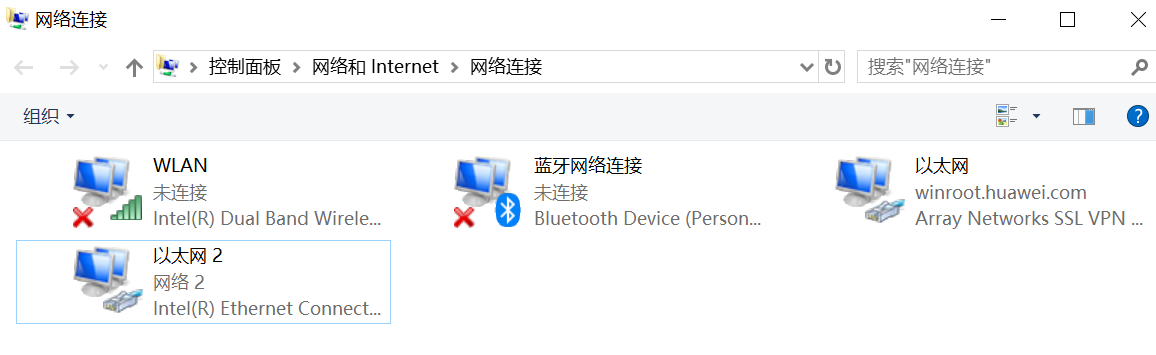
service NetworkManager restart //Ubuntu Server版本，此步骤请忽略

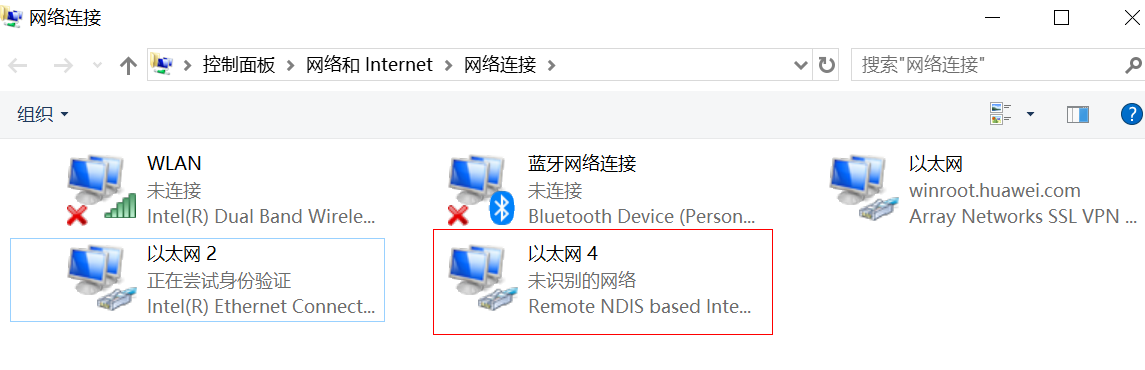
然后，ifconfig查看，可以看到虚拟网卡已经被分配了IP：



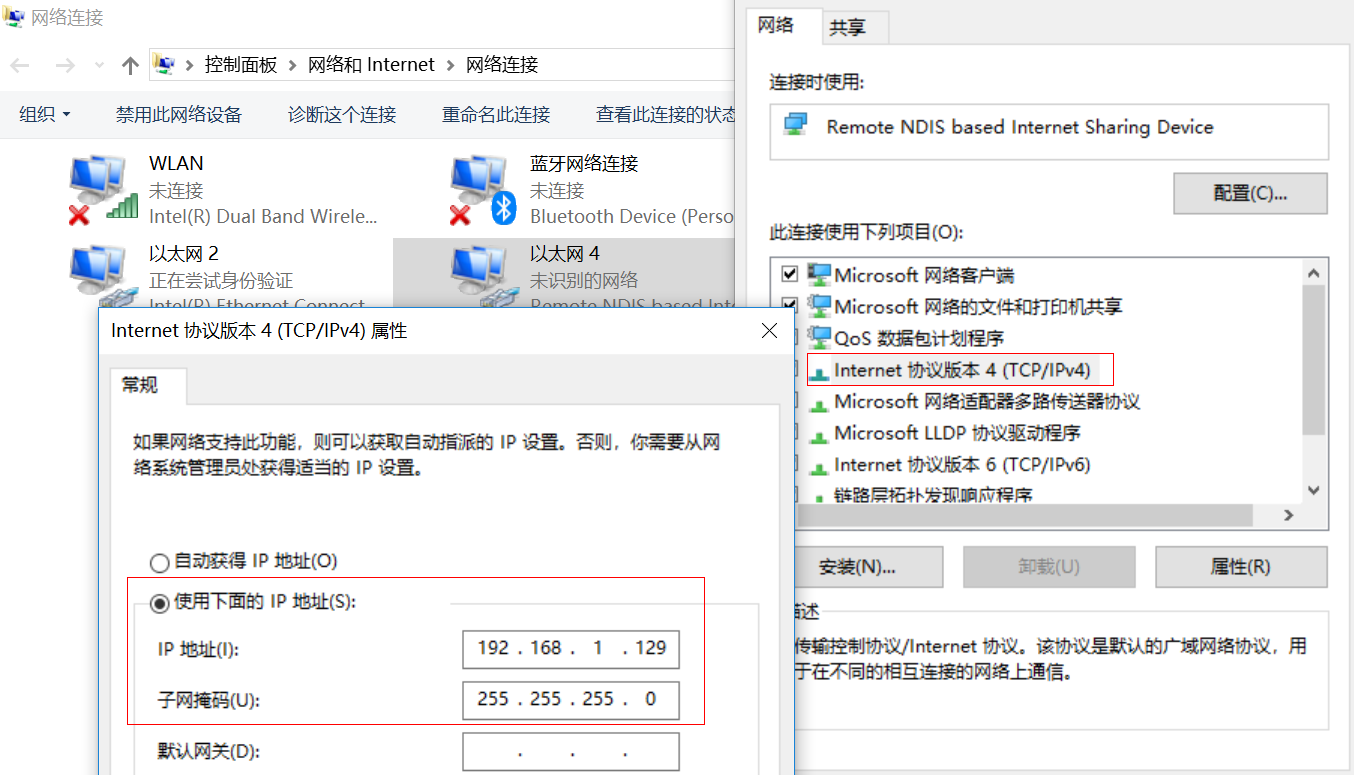
#### USB虚拟网卡IP配置（for windows）

连接usb连接线前先查看网络连接

将开发板通过usb线与windows连接，再查看网络连接，会发现多了一个以太网设备，这个就是windows虚拟出的一个usb网卡，以下图为例，就是“以太网 4”，接下来为该网卡配置静态IP，与开发板在同一网段即可；



为usb虚拟网卡配置静态IP，因为开发板默认有两个IP（usb设备的IP默认是192.168.1.2， eth0设备的IP默认是192.168.0.2），这里是通过usb连接，所以配置usb虚拟网卡为192.168.1.xxx，例如这里我们将其设置为192.168.1.129，然后确认退出；



打开cmd窗口，ping 192.168.1.2，应该就可以通了：）