# Sniffer嗅探器项目报告

目录

[Sniffer嗅探器开发报告 1](#_Toc8505)

[1. 开发环境 1](#_Toc16236)

[2. 项目开发 2](#_Toc4909)

[2.1. 主要框架搭建 2](#_Toc18512)

[2.2. 初始化winpcap 3](#_Toc9569)

[2.3. SelectDev&&开始抓包 4](#_Toc12209)

[2.4 按钮触发函数 7](#_Toc10082)

[2.5 包列表函数 12](#_Toc19166)

[2.6 \*\*\*解析包\*\*\* 15](#_Toc5490)

[2.7 过滤器 17](#_Toc22670)

[2.8 IP分片重组 19](#_Toc8417)

[3. 实验结果测试 20](#_Toc25703)

[3.1 基本测试 20](#_Toc4102)

[3.2 IP分片重组 20](#_Toc1922)

[3.3 包过滤 23](#_Toc23348)

[3.4 数据包保存和读取 24](#_Toc30236)

[4. 问题与解决 26](#_Toc22045)

[4.1 代码问题 26](#_Toc3038)

[4.2 测试问题 27](#_Toc29055)

[参考列表 29](#_Toc21347)

## 

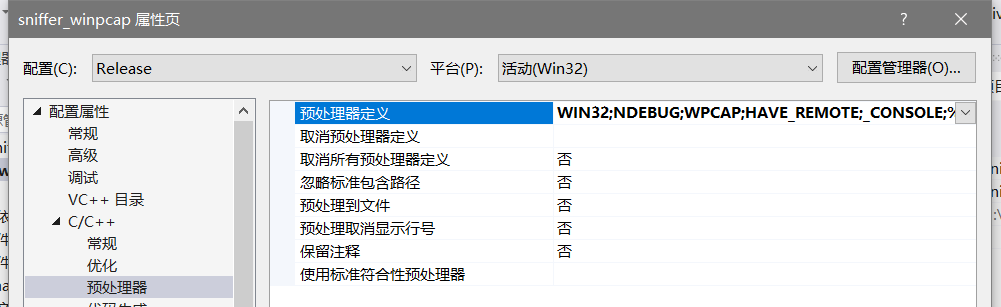
## 开发运行环境

项目基于vs2019配置WinPcap(开发包已放在目录WpdPack里)和MFC开发环境。

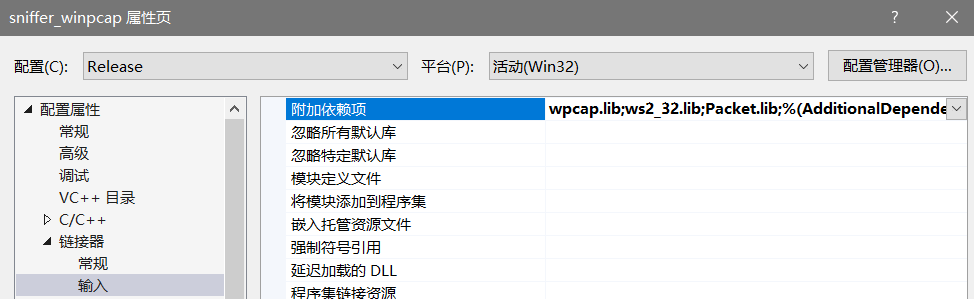
项目目前只能在**安装了winpcap**的x86平台运行。

MFC配置简单，直接创建项目即可。WinPcap具体配置如下：

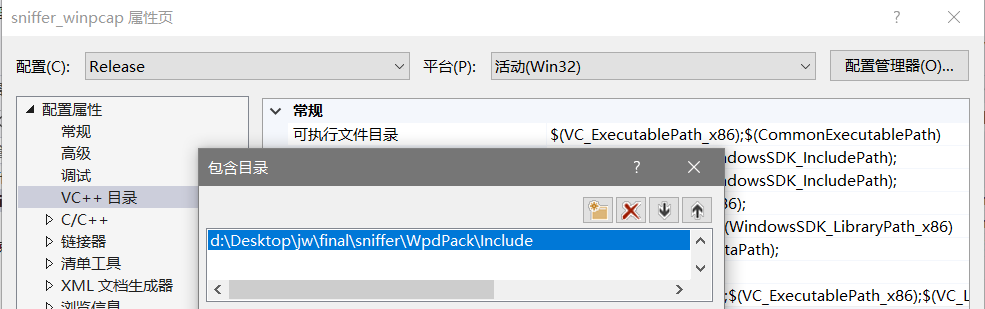
1. 添加宏WPCAP和HAVE\_REMOTE

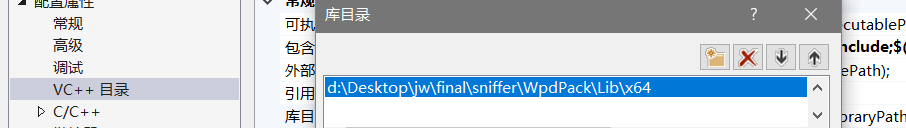


1. 添加动态链接库连接器输入wpcap.lib,ws2\_32.lib,Packet.lib



1. 添加静态、动态库文件夹地址

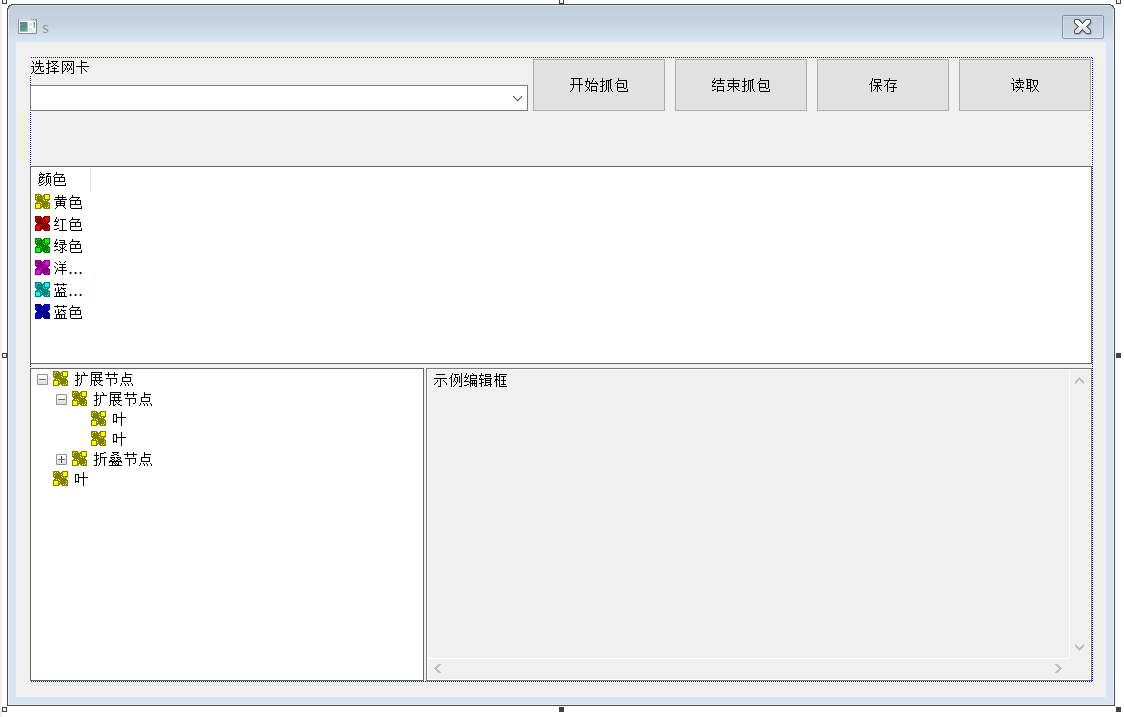


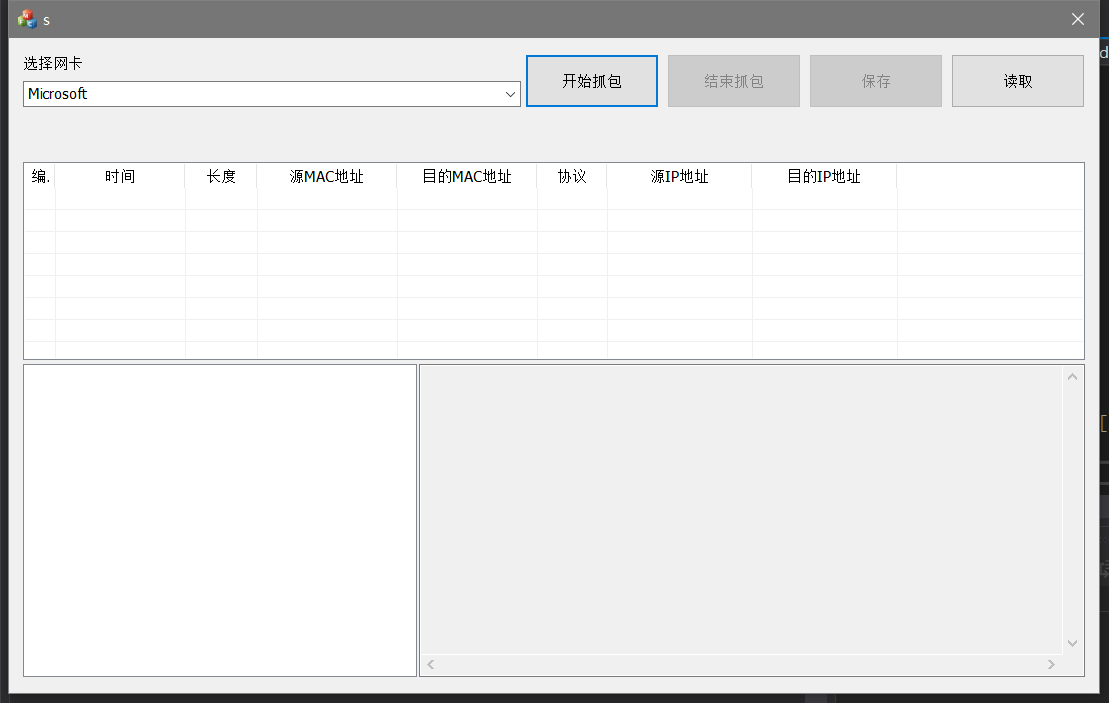


## 项目开发

### 主要框架搭建

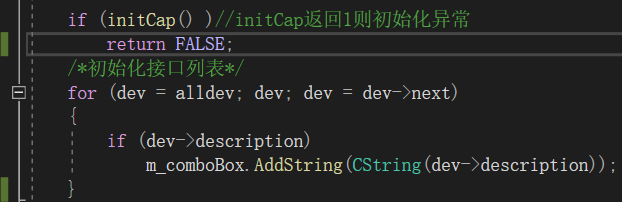
用MFC搭建主要框架。计划优先实现主要功能，之后再添加其他功能。



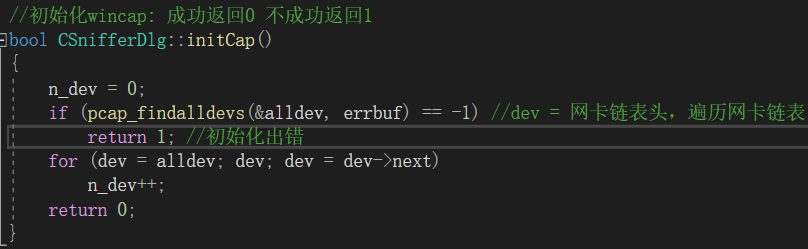


### 初始化winpcap

**#SnifferDlg.cpp**

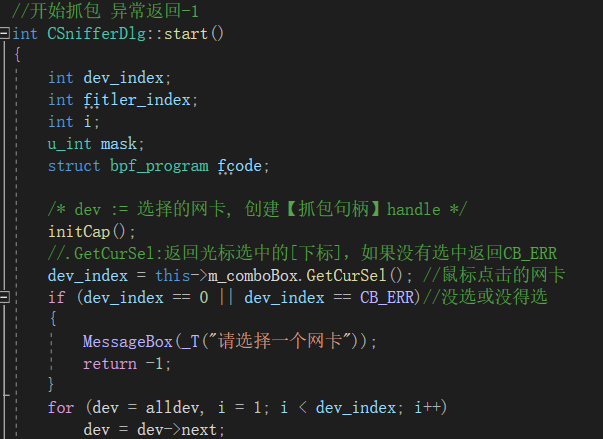


调用initCap初始化winpcap

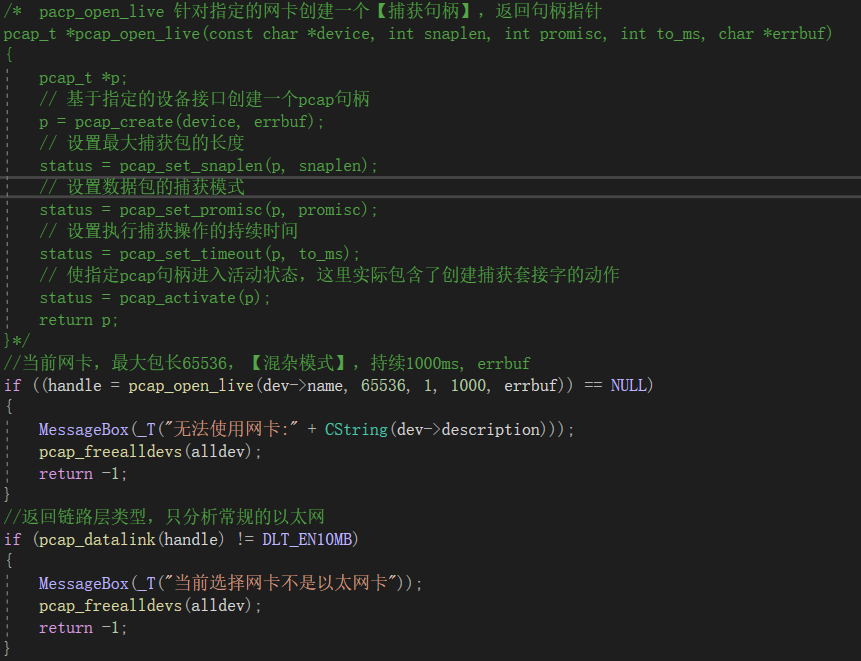


### SelectDev&&开始抓包

开始抓包，先调用m\_comboBox.GetCurSel()选择网卡，然后遍历并锁定网卡。



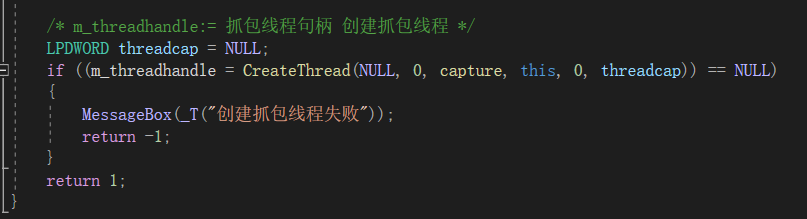
winpcap抓包需要创建一个抓包句柄handel，这里只分析常规的以太网网卡（即便是无线网卡其实最后也会被os虚拟化成以太网卡，无线网卡是特殊的专门网卡，一般不会用得到，为了方便和防止异常就做一下判断）。



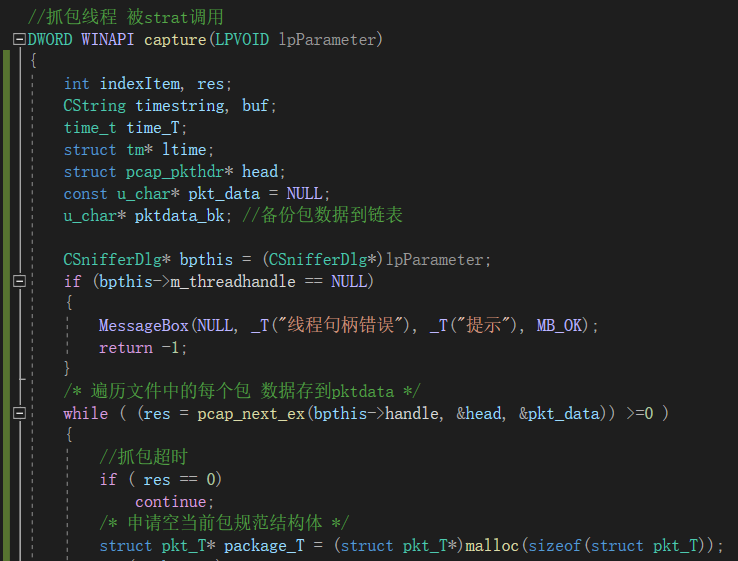
创建一下临时文件的保存位置，创建临时保存文件，文件名用时间：



创建抓包线程（由于需要窗口页面，所以MFC窗口是主线程，只能再创建一个抓包线程）：



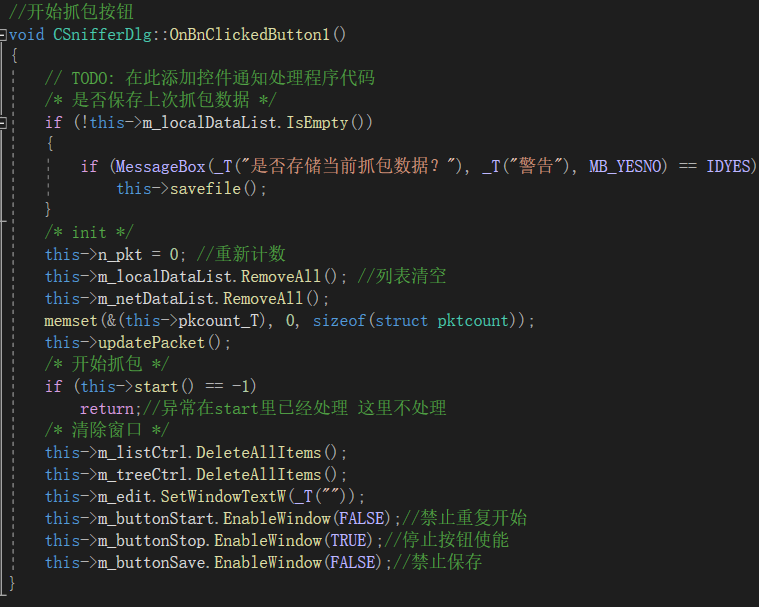
抓包线程函数：capture。其中最重要的是调用了pcap\_next\_ex来抓取下一个数据包，将其头部放入head，数据部分放入pkt\_data。然后创建pkt\_T package\_T来存储规范化数据包头部的信息，后来的部分和后面的“读取按钮”中的readfile一模一样：都是先调用analyze\_frame来分析包信息，赋值给package\_T，然后把package\_T中的规范信息输出在window上。



### 2.4 按钮触发函数

#### 2.4.1 开始按钮

虽然每次为了方便都会创建tmp文件保存抓包记录，但是是否真的存储要在下次抓包前判断。

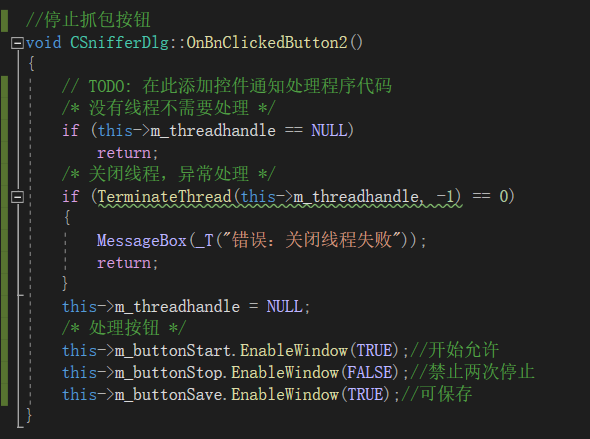


初始化，清空各个变量，然后start()开始抓包。

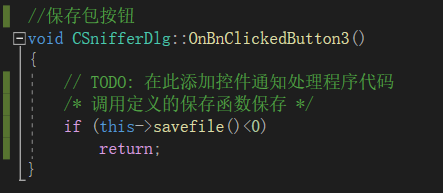
这里不再处理异常，异常都在start()里处理完毕了。

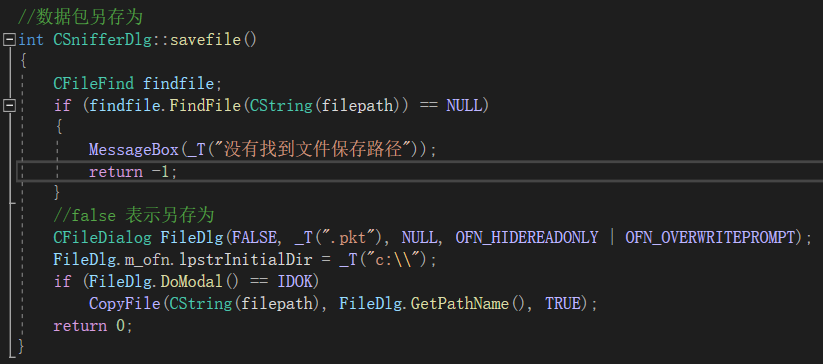


#### 2.4.2 结束按钮

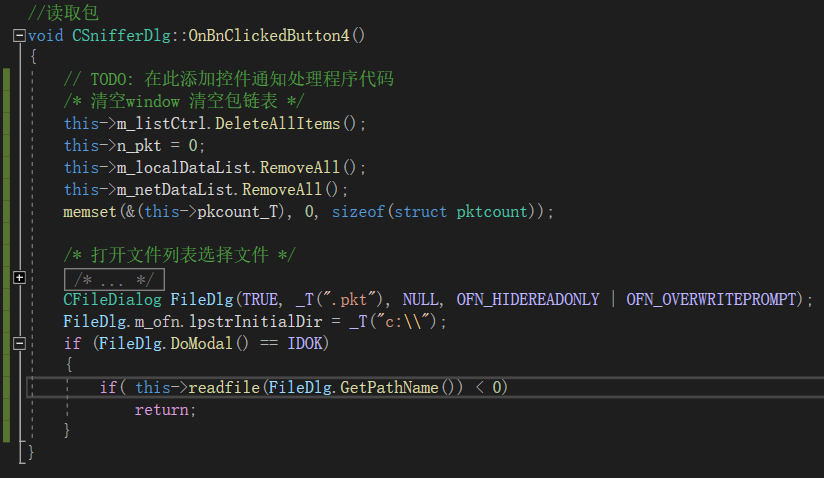


#### 2.4.3 保存按钮

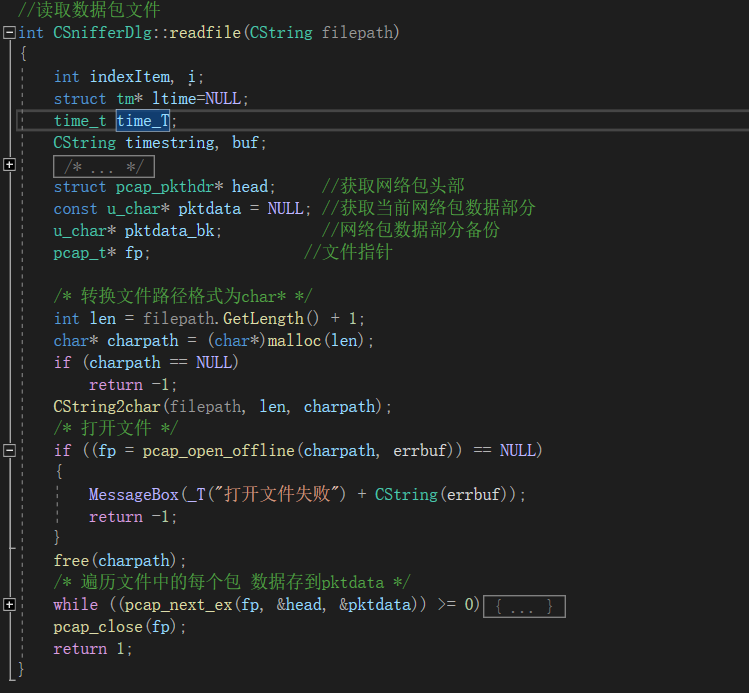




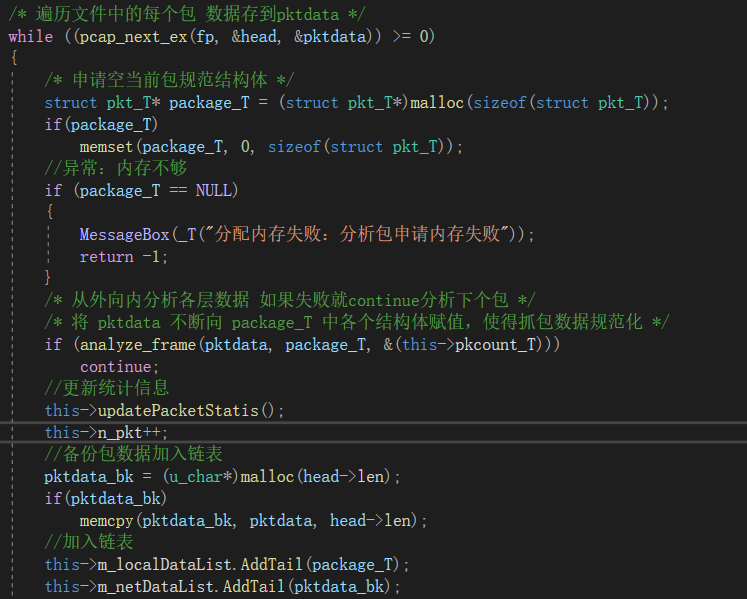
#### 2.4.4 读取按钮



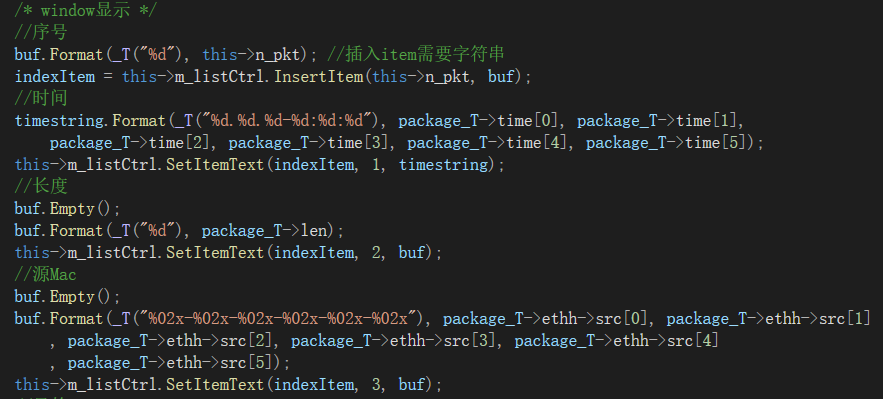
调用功能函数：readfile来读取选择的抓包文件里的包：



主要循环：

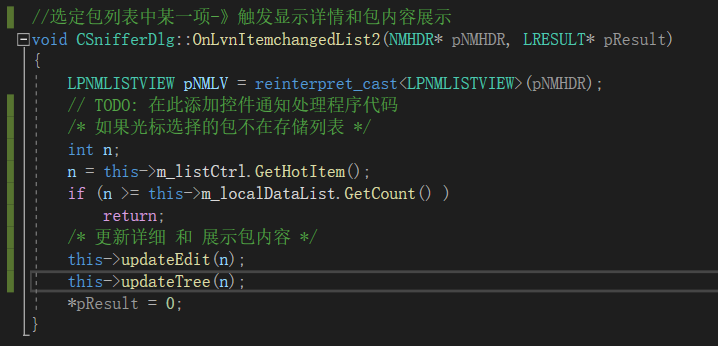


之后是为展示窗口的各个项目赋值的过程，例如：



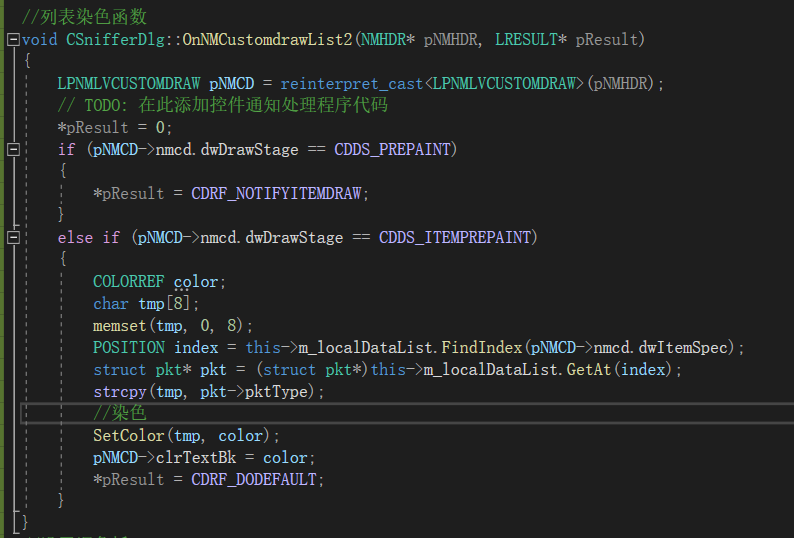
### 2.5 包列表函数

#### 2.5.1 选中包列表的某个包

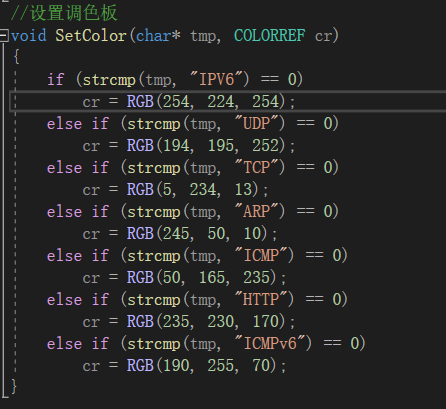


#### 2.5.2 自动染色

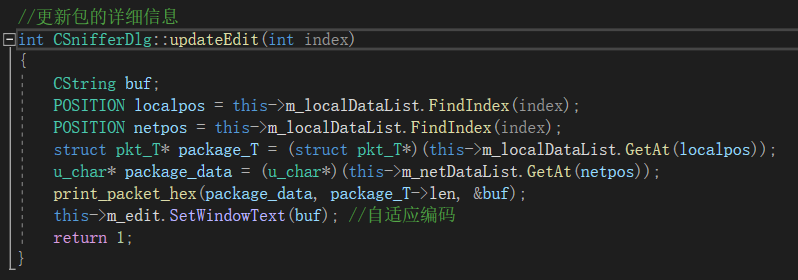
染色函数：



调色板：



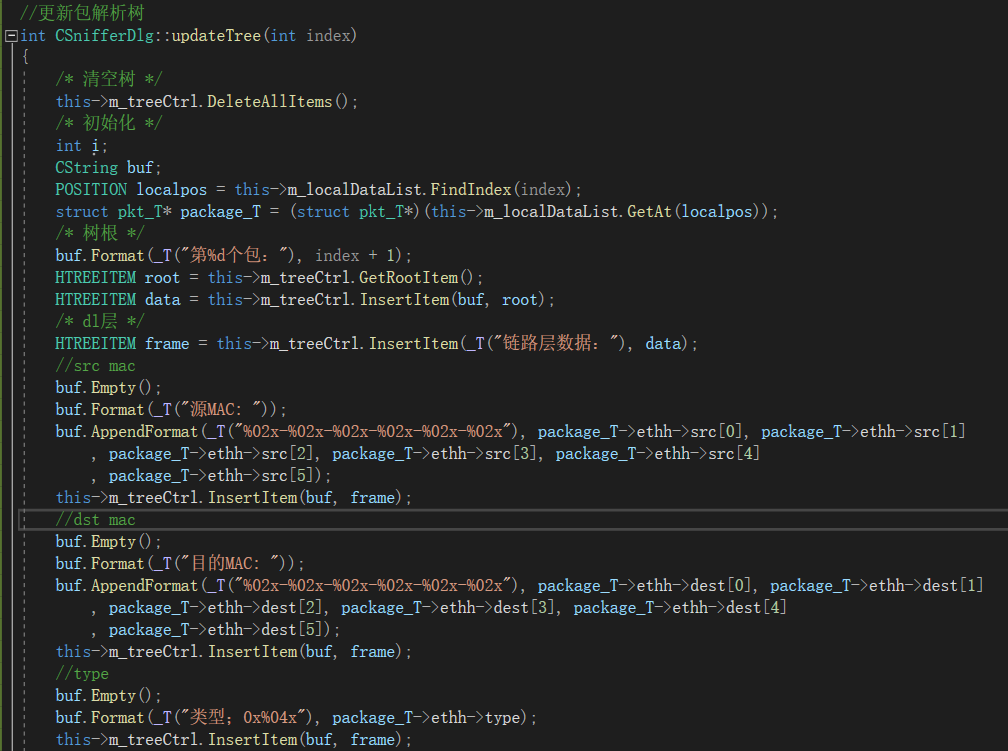
#### 2.5.3 展示包数据信息



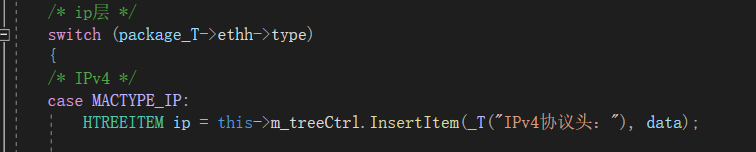
#### 2.5.4 展示包头信息

package\_T保存了鼠标选定的包的格式化的包头数据，不断从中取出数据来嵌入树的节点。

这里链路层数据的处理方式是一样的（网卡选定为以太网卡了）。



网络层涉及到IPv4, ARP, IPv6三种协议，用switch语句来分别处理。



ip分段部分标志位0b1110 0000

0000 0000 0000 0000

011X XXXX XXXX XXXX

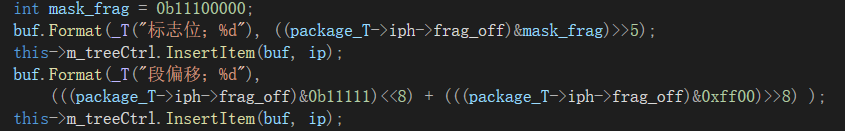
0 15

0001 1111 1111 1111

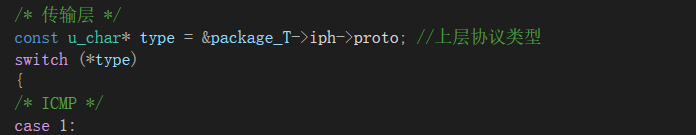
考虑大小端序，存储下来是小端序，段偏移:

1111 1111 0001 1111

x&0b1 1111<<8 + x&0b1111 1111 0000 0000>>8



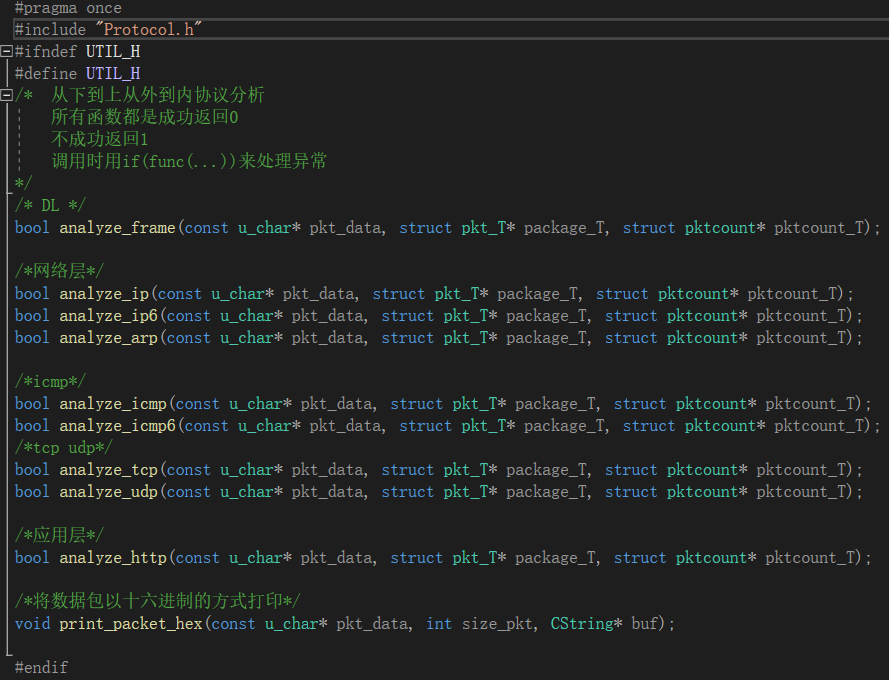
传输层考虑TCP, UDP, ICMP(v6)三种协议，也用switch。



### 2.6 \*\*\*解析包\*\*\*

**#util.h**

主要思路为从DL层开始逐层分析(判断上层协议类型，调用响应上层的解析函数)数据包的头部结构，最后调用print\_packet\_hex将数据转换成16进制保存。

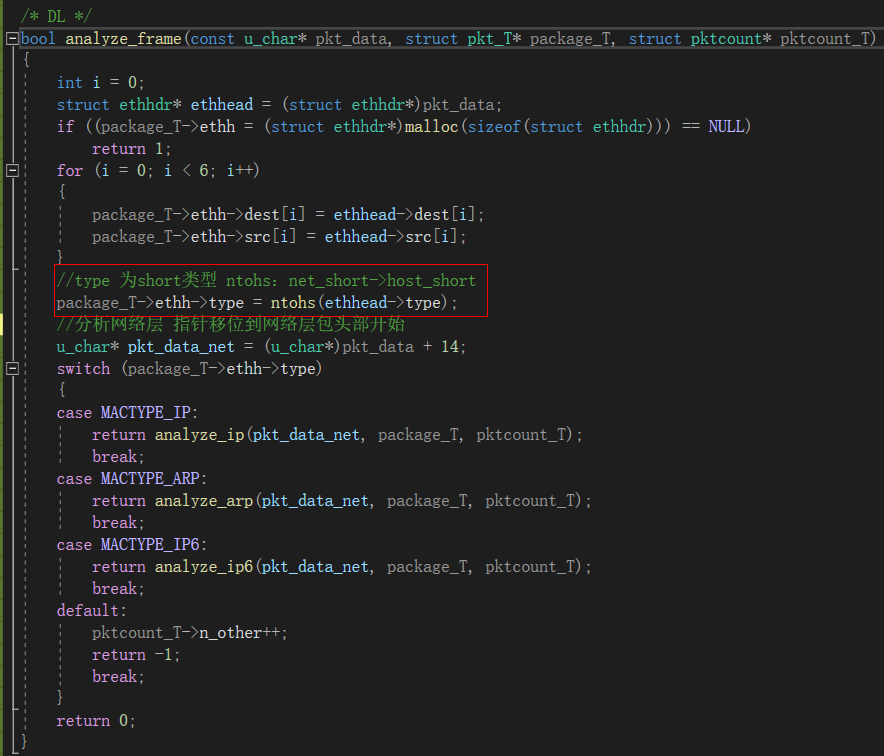


具体实现在**#util.c**

#### 2.6.1 analyze\_frame

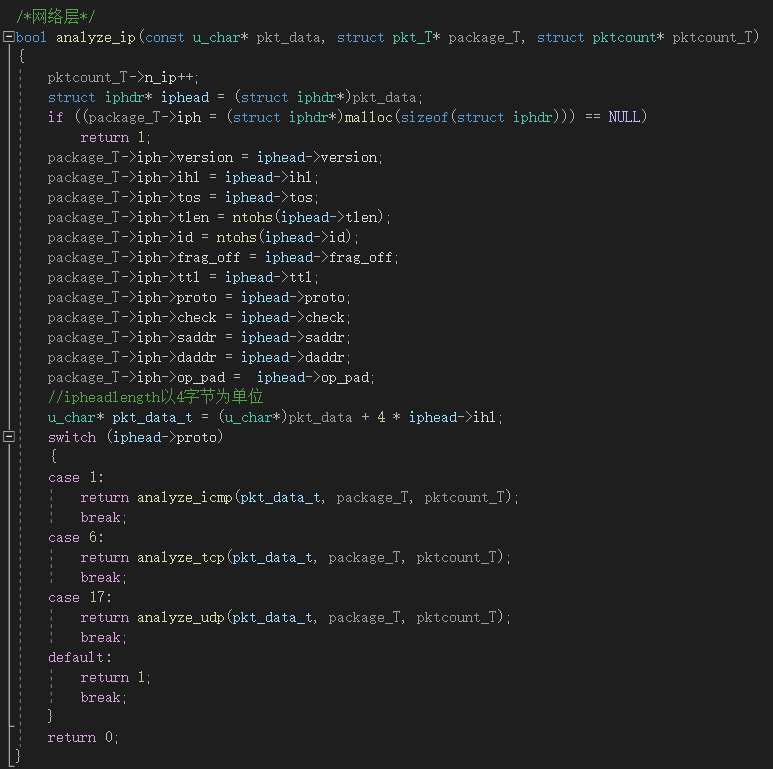
解析链路层数据，type是short类型，需要大小端的转换（网络流是大端，一般主机是小端）。

然后将指针位移到网络层数据包头部的开始，在switch中选择上层协议类型，进行进一步的解析。



#### 2.6.2 analyze\_ip

IPv4协议头解析。



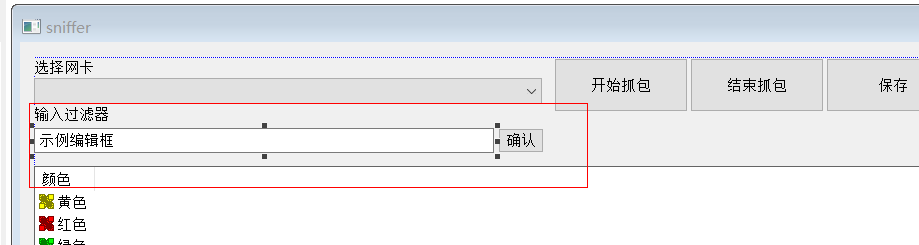
其他的解析也是类似，都是把原始的数据直接赋值给相应的头部结构体，结构体是按照网络头部的格式安排的，所以直接取出各项的值赋给package\_T即完成。最后根据再上一层协议的类型来分别调用上一层的协议解析函数即可。

### 2.7 过滤器

由于winpcap里的过滤器规则比较复杂，如果说做成选项的话比较困难，考虑的东西太多，所以干脆做成输入框，输入规则然后点击确认就生效。

#### 2.7.1 控件

添加两个控件：edit2和button5，分别用来输入过滤规则和确认过滤器的设置。

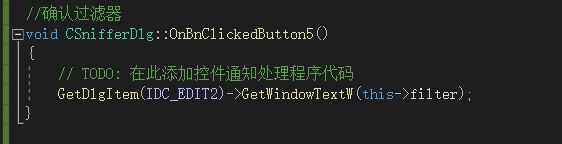


#### 2.7.2 过滤器设置

添加过滤器字符串filter：



点击“确认”按钮触发设置：



在开始抓包的时候设置过滤器(在函数start())：

这里注意需要一个mask值，还有pcap\_compile编译过滤器的时候必须要输入char\*类型，所以需要CString转成char，转换的时候len+1不然没有’\0’会出错。

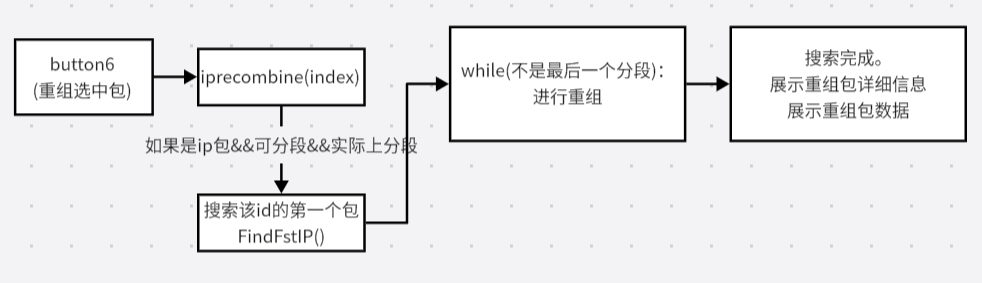


### 2.8 IP分片重组

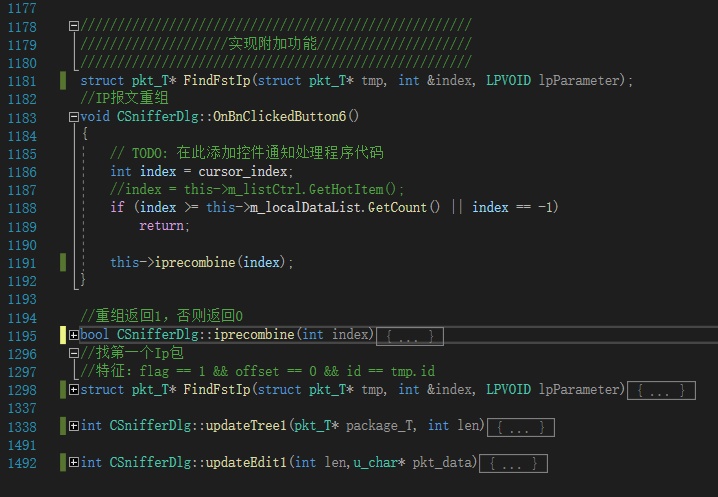
如果有分片则有如下特点：

1. 只有第一个包带上层协议头（TCP\UDP\ICMP）
2. 开始包：flag = 1; offset == 0;
3. 中间包：flag = 1; offset != 0;
4. 最后包：flag = 0; offset != 0;
5. 不分段：flag = 0; offset == 0;

实现逻辑：



代码框架如下：都在文件**SnifferDlg.cpp**里实现



## 参考列表

1. 文件列表相关：<https://blog.csdn.net/ivandark/article/details/7645618>
2. vs的参考手册：<https://learn.microsoft.com/en-us/cpp/mfc/reference>
3. vs的MFC各类工具箱控件参考手册：

[https://learn.microsoft.com/zh-cn/windows/win32/controls](https://learn.microsoft.com/zh-cn/windows/win32/controls/nm-customdraw-toolbar)