

西安科技大学

**网络管理与维护**

**大作业**

**学院：** 计算机科学与技术学院

**专业：** 网络工程系   
 **班级：** 网络工程1801班

**姓名：** 吴斌

**学号：** 18408020129

**日期：** 2021年5月

目录

[1.SNMP介绍 3](#_Toc73022014)

[1.1 SNMP的工作原理: 3](#_Toc73022015)

[1.2 SNMP的基本思想： 3](#_Toc73022016)

[1.3 SNMP结构概述: 3](#_Toc73022017)

[1.4 SNMP支持的网管操作 4](#_Toc73022018)

[1.5 SNMP的实现结构 4](#_Toc73022019)

[1.6 SNMP的技术内容 5](#_Toc73022020)

[1.7 SNMP的发展历史 5](#_Toc73022021)

[1.8 SNMP的技术术语 6](#_Toc73022022)

[2.程序设计思路 7](#_Toc73022023)

[3.实验结果展示及分析 7](#_Toc73022024)

[4.程序源码及运行说明 8](#_Toc73022025)

[4.1代理站程序 8](#_Toc73022026)

[4.1.1 AnalyzPaper.h 8](#_Toc73022027)

[4.1.2 Head\_Files.h 11](#_Toc73022028)

[4.1.3 UdpTalk.h 11](#_Toc73022029)

[4.1.4 UidClass.h 13](#_Toc73022030)

[4.1.5 server.cpp 15](#_Toc73022031)

[4.2 客户端程序 15](#_Toc73022032)

[4.2.1 UdpTalk.h 15](#_Toc73022033)

[4.2.2 Checkpaper.h 17](#_Toc73022034)

[4.2.3 Client.h 18](#_Toc73022035)

[4.3 运行说明 18](#_Toc73022036)

# 1.SNMP介绍

## 1.1 SNMP的工作原理:

SNMP：“简单网络管理协议”，用于网络管理的协议。SNMP用于网络设备的管理。SNMP的工作方式：管理员需要向设备获取数据，所以SNMP提供了“读”操作；管理员需要向设备执行设置操作，所以SNMP提供了“写”操作；设备需要在重要状况改变的时候，向管理员通报事件的发生，所以SNMP提供了“Trap”操作。

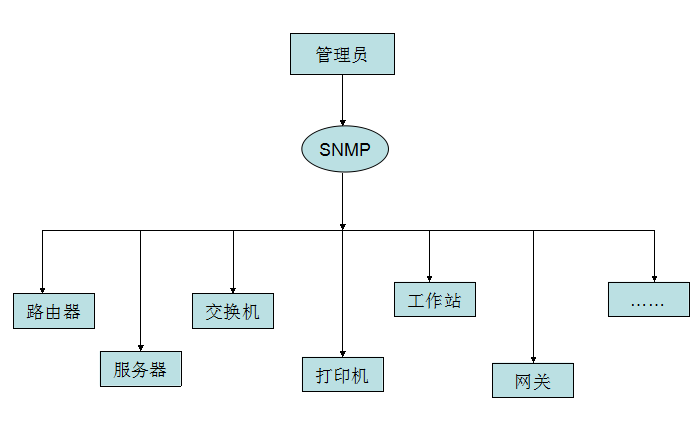
## 1.2 SNMP的基本思想：

为不同种类的设备、不同厂家生产的设备、不同型号的设备，定义为一个统一的接口和协议，使得管理员可以是使用统一的外观面对这些需要管理的网络设备进行管理。通过网络，管理员可以管理位于不同物理空间的设备，从而大大提高网络管理的效率，简化网络管理员的工作。

## 1.3 SNMP结构概述:

SNMP被设计为工作在TCP/IP协议族上。SNMP基于TCP/IP协议工作，对网络中支持SNMP协议的设备进行管理。所有支持SNMP协议的设备都提供SNMP这个统一界面，使得管理员可以使用统一的操作进行管理，而不必理会设备是什么类型、是哪个厂家生产的。

如下图:



## 1.4SNMP支持的网管操作

对于网络管理，我们面对的数据是设备的配置、参数、状态等信息，面对的操作是读取和设置；同时，因为网络设备众多，为了能及时得到设备的重要状态，还要求设备能主动地汇报重要状态，这就是报警功能。

Get：读取网络设备的状态信息。

Set：远程配置设备参数。

Trap：管理站及时获取设备的重要信息。

## 1.5 SNMP的实现结构

在具体实现上，SNMP为管理员提供了一个网管平台(NMS)，又称为管理站，负责网管命令的发出、数据存储、及数据分析。被监管的设备上运行一个SNMP代理(Agent))，代理实现设备与管理站的SNMP通信。

管理站与代理端通过MIB进行接口统一，MIB定义了设备中的被管理对象。管理站和代理都实现了相应的MIB对象，使得双方可以识别对方的数据，实现通信。管理站向代理申请MIB中定义的数据，代理识别后，将管理设备提供的相关状态或参数等数据转换为MIB定义的格式，应答给管理站，完成一次管理操作。

已有的设备，只要新加一个SNMP模块就可以实现网络支持。旧的带扩展槽的设备，只要插入SNMP模块插卡即可支持网络管理。网络上的许多设备，路由器、交换机等，都可以通过添加一个SNMP网管模块而增加网管功能。服务器可以通过运行一个网管进程实现。其他服务级的产品也可以通过网管模块实现网络管理，如Oracle、WebLogic都有SNMP进程，运行后就可以通过管理站对这些系统级服务进行管理。

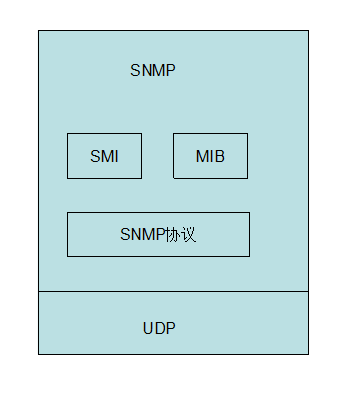
网络管理系统：又称管理站、NMS。是系统的控制台，向管理员提供界面以获取与改变设备的配置、信息、状态、操作等信息。管理站与Agent进行通信，执行相应的Set和Get操作，并接收代理发过来的警报(Trap)。

代理：Agent是网络管理的代理人，负责管理站和设备SNMP操作的传递。介于管理站和设备之间，与管理站通信并相应管理站的请求，从设备获取相应的数据，或对设备进行相应的设置，来响应管理站的请求。代理也需要具有根据设备的相应状态使用MIB中定义的Trap向管理站发送报告的能力。

代理服务器：Proxy是一种特殊的代理，在不能直接使用SNMP协议的地方，如：异种网络、不同版本的SNMP代理等情况，Proxy代替相关设备向管理站提供一种外观，为设备代理SNMP协议的实现。Proxy做了异种网络或不同版本代理和相应SNMP数据请求的转换工作。

管理信息库MIB：定义了设备上可以使用的管理信息。代理和管理站使用MIB作为统一的数据接口通信。

## 1.6SNMP的技术内容



## 1.7 SNMP的发展历史

* 1989年------ SNMPv1
* 1991年------ RMON(Remote Network Monitoring 远程网络监视)，它扩充了SNMP的功能，包括对LAN的管理及对依附于这些网络的设备的管理。RMON 没有修改和增加SNMPv1，只是增加了SNMP监视子网的能力。
* 1993年------ SNMPv2(SNMPv1的升级版)
* 1995年------ SNMPv2正式版，其中规定了如何在基于OSI的网络中使用SNMP
* 1995年------ RMON扩展为RMON2
* 1998年------ SNMPv3，一系列文档定义了SNMP的安全性，并定义了将来改进的总体结构，SNMPv3可以和SNMPv2、SNMPv1一起使用。

## 1.8 SNMP的技术术语

* SNMP：Simple Network Management Protocol(简单网络管理协议)，是一个标准的用于管理基于IP网络上设备的协议。
* MIB：Management Information Base(管理信息库)，定义代理进程中所有可被查询和修改的参数。
* SMI：Structure of Management Information(管理信息结构)，SMI定义了SNMP中使用到的ASN.1类型、语法，并定义了SNMP中使用到的类型、宏、符号等。SMI用于后续协议的描述和MIB的定义。每个版本的SNMP都可能定义自己的SMI。
* ASN.1：Abstract Syntax Notation One(抽象语法定义)。用于定义语法的正式语言，在SNMP中定义SNMP的协议数据单元PDU和管理对象MIB的格式。SNMP只使用了ASN.1中的一部分，而且使用ASN.1的语言特性定义了一些自定义类型和类型宏 ，这些组成了SMI。
* PDU： Protocol Data Unit(协议数据单元)，它是网络中传送的数据包。每一种SNMP操作，物理上都对应一个PDU。
* NMS： Network Management System，网络管理系统，又名网络管理站，简称“管理站”。它是SNMP的总控机，提供统一的用户界面访问支持SNMP的设备，一般提供UI界面，并有统计、分析等功能，是网管系统的总控制台。NMS是网络管理操作的发起者。

* Agent： 是SNMP的访问代理，简称“代理”，为设备提供SNMP能力，负责设备与NMS的通信。
* Proxy： 代理服务器，对实现不同协议的设备进行协议转换，使非IP协议的设备也能被管理。
* Trap： 是由设备主动发出的报警数据，用于提示重要的状态的改变。
* BER： Basic Encoding Rule，基本编码规格。描述如何将ASN.1类型的值编码为字符串的方法。它是ASN.1标准的一部分。BER编码将数据分成TLV三部分，T为Tag的缩写，是类型标识；L为Length的缩写，标识类型的长度；V为Value的缩写，标识数据内容。按照TLV的顺序对数据进行编码，生成字节流。SNMP使用BER将SNMP的操作请求和应答编码后进行传输，并用于接收端进行解码。

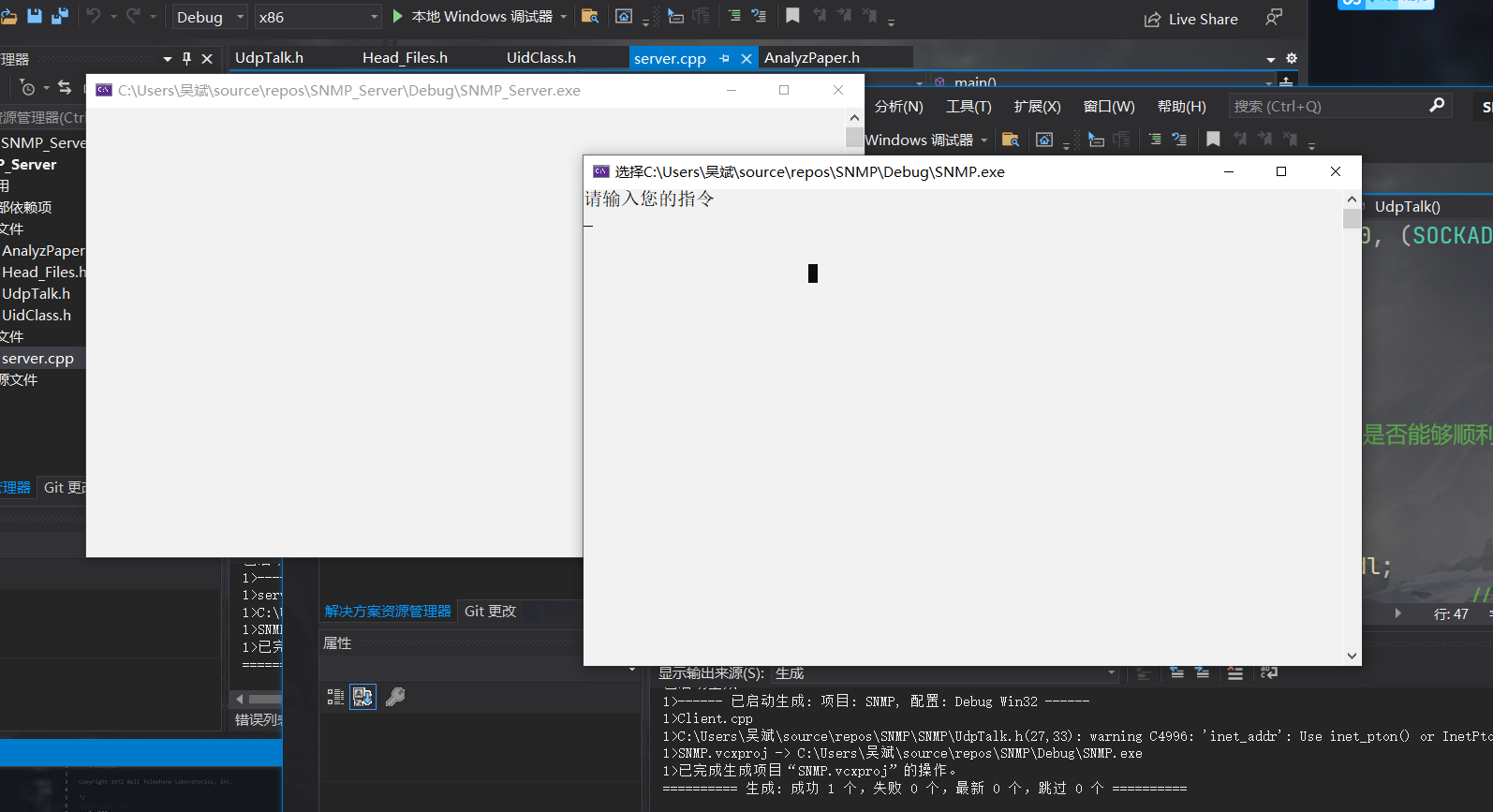
# 2.程序设计思路

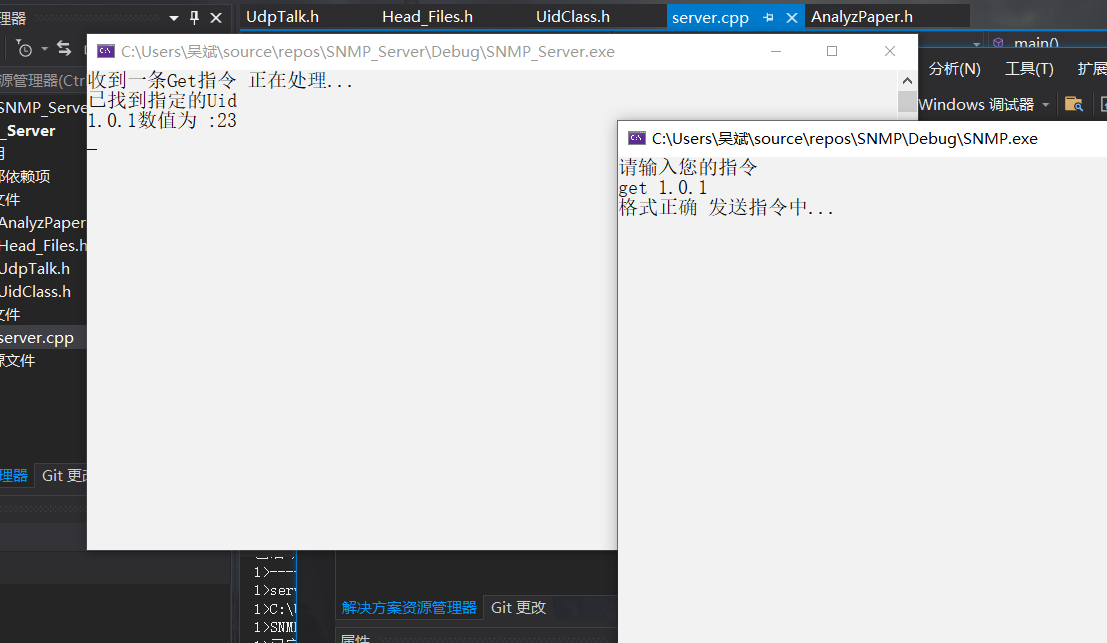
在理解了SNMP网络管理协议的前提下，考虑到具体实现的难度和人手的局限性只做了简易实现。

首先我设计一个代理端存储了被管对象的资源（在实际情况下通常有一个MIB库来存储被管对象的资源，这里只做简单模拟，并未考虑面向应用开发）；然后开启后会一直等待管理站的管理请求，接受来自管理站的报文并且解析报文。然后与被管对象进行交互修改被管对象的资源内容。

然后是一个管理站的设计；用来实现和用户进行交互；读取用户输入的命令然后将其封装成特定报文的格式；发送给代理站让代理站去处理管理请求。

# 3.实验结果展示及分析





# 4.程序源码及运行说明

## 4.1代理站程序

### 4.1.1 AnalyzPaper.h

#pragma once

#include "UidClass.h"

string UID;

int Send\_data;

string NotFind = "Can't Find The Uid You Given";

int AnalyzPaper(string);

//分析收到的报文(因为在下位机已经检测报文格式)

//所以收到的报文误差率很低

void Response(int);

int AnalyzPaper(string PaperStr)

{

if (0 == strncmp(PaperStr.c\_str(), "get ", 4))

{

cout << "收到一条Get指令 正在处理..." << endl;

UID = PaperStr.substr(4);

for (vector<Uid\_Info>::iterator Uit = V\_Uid.begin(); Uit != V\_Uid.end(); Uit++)

{

if (!Uit->GetUid().compare(UID))

{

Send\_data = Uit->Getdata();

return 1;

}

}

return 0; //没有找到给定的Uid

} //判断为Get指令

else if (0 == strncmp(PaperStr.c\_str(), "getnext ", 8))

{

cout << "收到一条GetNext指令 正在处理..." << endl;

UID = PaperStr.substr(8);

for (vector<Uid\_Info>::iterator Uit = V\_Uid.begin(); Uit != V\_Uid.end(); Uit++)

{

if (!Uit->GetUid().compare(UID))

{

Send\_data = (++Uit)->Getdata();

return 2;

}

}

return 0; //没有找到给定的Uid

} //判断为getNext指令

if (0 == strncmp(PaperStr.c\_str(), "set ", 4))

{

cout << "收到一条Set指令 正在处理..." << endl;

UID = PaperStr.substr(4);

for (vector<Uid\_Info>::iterator Uit = V\_Uid.begin(); Uit != V\_Uid.end(); Uit++)

{

if (!strncmp(UID.c\_str(),Uit->GetUid().c\_str(),5))

{

Uit->SetData(PaperStr.back());

return 3;

}

}

return 0; //没有找到给定的Uid

} //判断为Set指令

}

void Response(int flag)

{

switch (flag)

{

case 0:

cout << "没有找到您输入的Uid信息" << endl;

break;

case 1:

cout << "已找到指定的Uid" << endl;

cout << UID << "数值为 :" << Send\_data << endl;

break;

case 2:

cout << "已找到指定的Uid" << endl;

cout << UID << "的GetNext数值为 :" << Send\_data << endl;

break;

case 3:

cout << "已找到指定的Uid" << endl;

break;

default:

break;

};

}

### 4.1.2 Head\_Files.h

#pragma once

#include <string>

#include <Winsock2.h>

#include <iostream>

#include <algorithm>

#include <vector>

#pragma comment(lib,"ws2\_32.lib")

using namespace std;

### 4.1.3 UdpTalk.h

#pragma once

#include "AnalyzPaper.h"

WORD wVersionRequested;

WSADATA wsaData;

SOCKADDR\_IN addrSrv;

SOCKET sockSrv;

SOCKADDR\_IN addrClient;

int len = sizeof(SOCKADDR);

int err;

char recvBuf[100]; //定义用到的变量

int UdpTalk();

void GetPaper();

void SendPaper();

int InitUdp(); //函数声明

int InitUdp()

{

wVersionRequested = MAKEWORD(1, 1);

err = WSAStartup(wVersionRequested, &wsaData);

if (0 != err)

{

return 0;

}

if (1 != LOBYTE(wsaData.wVersion) || 1 != HIBYTE(wsaData.wVersion))

{

WSACleanup();

return 0;

}

sockSrv = socket(AF\_INET, SOCK\_DGRAM, 0);

addrSrv.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);

addrSrv.sin\_family = AF\_INET;

addrSrv.sin\_port = htons(6000);

bind(sockSrv, (SOCKADDR\*)&addrSrv, sizeof(SOCKADDR));

return 1;

}

void GetPaper()

{

recvfrom(sockSrv, recvBuf, 100, 0, (SOCKADDR\*)&addrClient, &len);

}

//void SendPaper()

//{

// sendto();

//}

int UdpTalk()

{

SetVuid();

if (InitUdp()) //udp连接可以成功

{

while (1)

{

GetPaper();

Response(AnalyzPaper(recvBuf)); //本地响应

//处理分析报文

Sleep(4000); system("cls");

}

}

WSACleanup();

closesocket(sockSrv);

return 0;

}

### 4.1.4 UidClass.h

#pragma once

#include "Head\_Files.h"

class Uid\_Info

{

public:

Uid\_Info(string Uid, size\_t data)

{

this->data = data;

this->Uid = Uid;

} //构造函数

~Uid\_Info()

{

} //默认析构函数

string GetUid()

{

return this->Uid;

} //得到对象实例的UID

size\_t Getdata()

{

return this->data;

} //得到对象实例的Info

void SetData(size\_t data)

{

this->data = data;

} //Set对象实例的数值

private:

string Uid;

size\_t data; //简易的把Uid和一个数值绑定起来

//进行简单模拟获取以及更改

};

vector<Uid\_Info> V\_Uid;

//用STL创建一个数组（或者集合种种都可以，我这里选择数组）

//这个数组用来存储“这种绑定关系”

Uid\_Info\* Uid1 = new Uid\_Info("1.0.1", 23);

Uid\_Info\* Uid2 = new Uid\_Info("1.0.2", 14);

Uid\_Info\* Uid3 = new Uid\_Info("1.0.3", 87);

Uid\_Info\* Uid4 = new Uid\_Info("1.0.4", 48);

Uid\_Info\* Uid5 = new Uid\_Info("1.0.5", 60); //创建了五个“绑定关系”

void SetVuid()

{

V\_Uid.push\_back(\*Uid1);

V\_Uid.push\_back(\*Uid2);

V\_Uid.push\_back(\*Uid3);

V\_Uid.push\_back(\*Uid4);

V\_Uid.push\_back(\*Uid5);

} //将这五个绑定关系放到数组里

### 4.1.5 server.cpp

#include "UdpTalk.h"

int main()

{

UdpTalk();

system("pause");

return 0;

}

## 4.2 客户端程序

### 4.2.1 UdpTalk.h

#pragma once

#include "CheckPaper.h"

int InitUdp();

int UdpTalk();

void SendPaper(); //需要用到的函数声明

int InitUdp()

{

wVersionRequested = MAKEWORD(1, 1);

err = WSAStartup(wVersionRequested, &wsaData);

if (0 != err)

{

return 0;

}

if (1 != LOBYTE(wsaData.wVersion) || 1 != HIBYTE(wsaData.wVersion))

{

WSACleanup();

return 0;

}

sockClient = socket(AF\_INET, SOCK\_DGRAM, 0);

addrSrv.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");

addrSrv.sin\_family = AF\_INET;

addrSrv.sin\_port = htons(6000);

return 1; //实现Udp通信

}

void SendPaper()

{

sendto(sockClient, str.data(), 100, 0, (SOCKADDR\*)&addrSrv, sizeof(SOCKADDR));

}

//把符合格式的报文发送出去

int UdpTalk()

{

if (InitUdp()) //判断Udp是否能够顺利通信

{

while (1)

{

cout << "请输入您的指令" << endl;

getline(cin, str); //获取本地输入的指令

if (0 == CheckPaper(str)) //格式判断有误

{

cout << "格式有误 请重新输入..." << endl;

Sleep(1000); system("cls");

}

else //格式正确的话 执行下列操作

{

cout << "格式正确 发送指令中..." << endl;

SendPaper();

Sleep(1000); system("cls");

}

}

}

WSACleanup();

closesocket(sockClient);

return 0;

}

### 4.2.2 Checkpaper.h

#pragma once

#include <Winsock2.h>

#include <string>

#include <algorithm>

#include <iostream>

#pragma comment(lib,"ws2\_32.lib")

using namespace std; //先在本地检查输入指令的格式是否正确

int err; string str;

WORD wVersionRequested;

int len = sizeof(SOCKADDR);

WSADATA wsaData;

SOCKET sockClient;

SOCKADDR\_IN addrSrv; //定义需要用到的变量

int CheckPaper(string); //来确保发送出去的包的格式都是正确的

int CheckPaper(string PaperStr)

{

string CommandCh;

transform(PaperStr.begin(), PaperStr.end(), back\_inserter(CommandCh), ::tolower);

//把指令变为小写方便对比

if (0 == strncmp(CommandCh.c\_str(), "getnext ", 8) && CommandCh.size() > 8)

{

return 2;

} //识别getNext指令

else if (0 == strncmp(CommandCh.c\_str(), "get ", 4) && CommandCh.size() > 4)

{

return 1;

} //识别Get指令

else if (0 == strncmp(CommandCh.c\_str(), "set ", 4) && CommandCh.size() > 4)

{

return 1;

} //识别Set指令

return 0;

//如三个前缀都不能满足，则判定为格式错误(暂时只实现这几个指令)

}

### 4.2.3 Client.h

#include "UdpTalk.h"

int main()

{

UdpTalk();

//调用通信函数发包

system("pause");

return 0;

}

## 4.3 运行说明

目前Version 0.1实现了基本的Get、GetNext、Set等命令；只需开启管理站（客户端）程序后；输入Get（Getnext、Set）+ OID即可（Set 命令需在OID后加上修改后的值）。