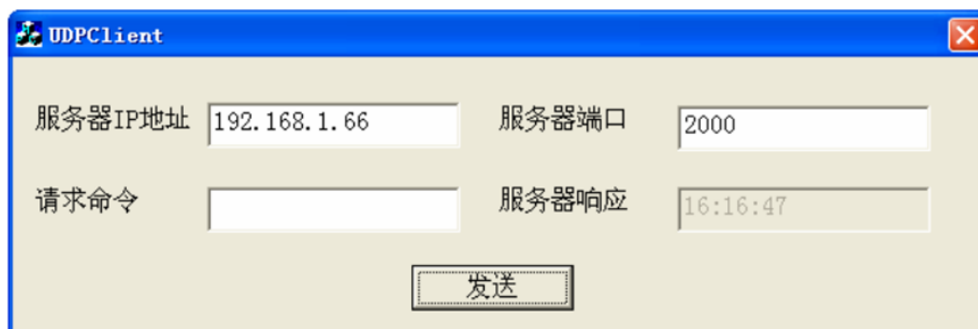


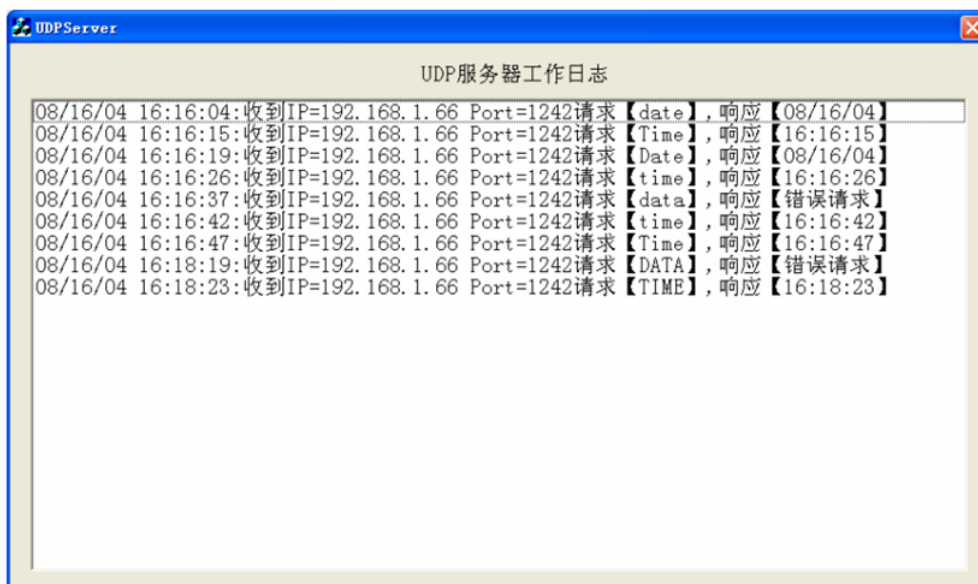
# H1 实验一：简单的客户机-服务器程序实验报告

## 1 实验要求

1. 利用CAsyncSocket类编写简单的客户-服务器程序
2. 客户-服务器之间使用数据报方式传送信息
3. 服务器收到客户发来的“Time”或“Date”请求后利用本地时间和日期分别进行响应
4. 界面如图：



客户程序界面示例



服务器程序界面示例

## 2 实验环境

Windows10、visual studio2019 preview、release X86

## 3 UDP协议介绍

1. 是无连接的。相比于TCP协议，UDP协议在传送数据前不需要建立连接，当然也就没有释放连接。
2. 是尽最大努力交付的。也就是说UDP协议无法保证数据能够准确的交付到目的主机。也不需要对接收到的UDP报文进行确认。
3. 是面向报文的。也就是说UDP协议将应用层传输下来的数据封装在一个UDP包中，不进行拆分或合并。因此，运输层在收到对方的UDP包后，会去掉首部后，将数据原封不动的交给应用进程。
4. 没有拥塞控制。因此UDP协议的发送速率不受网络的拥塞度影响。
5. UDP支持一对一、一对多、多对一和多对多的交互通信。

6. UDP的头部占用较小，只占用8个字节。

## 4 应用层协议介绍

### 4.1 客户端

1. 不区分大小写的字符串命令
2. 当是“date”或“time”时，会接收到来自服务器正确的date和time格式的回复命令，当是其他字符串时，会收到“错误请求”的回复命令。

### 4.2 服务器

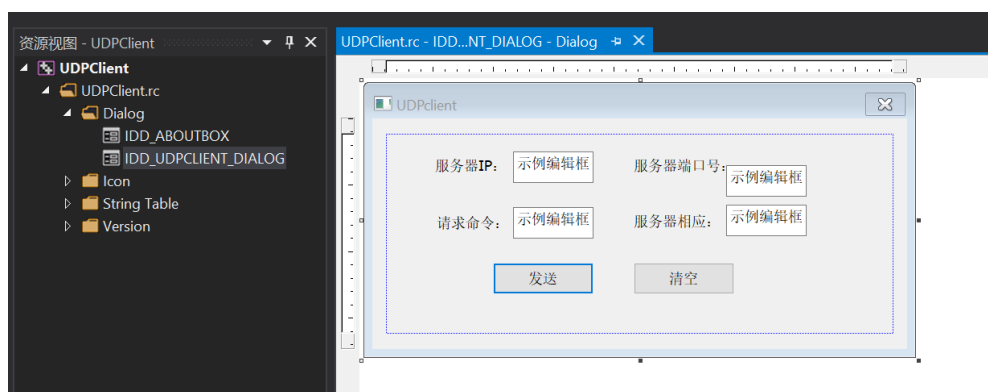
1. 当收到请求是不区分大小写的 date 字符串时，返回本机日期：格式是“xx/xx/xx”。
2. 当收到请求是不区分大小写的 time 字符串时，返回本机时间：格式是“xx:xx:xx”。
3. 当收到请求是其他字符串时，返回“错误请求”字符串。

## 5 实现步骤

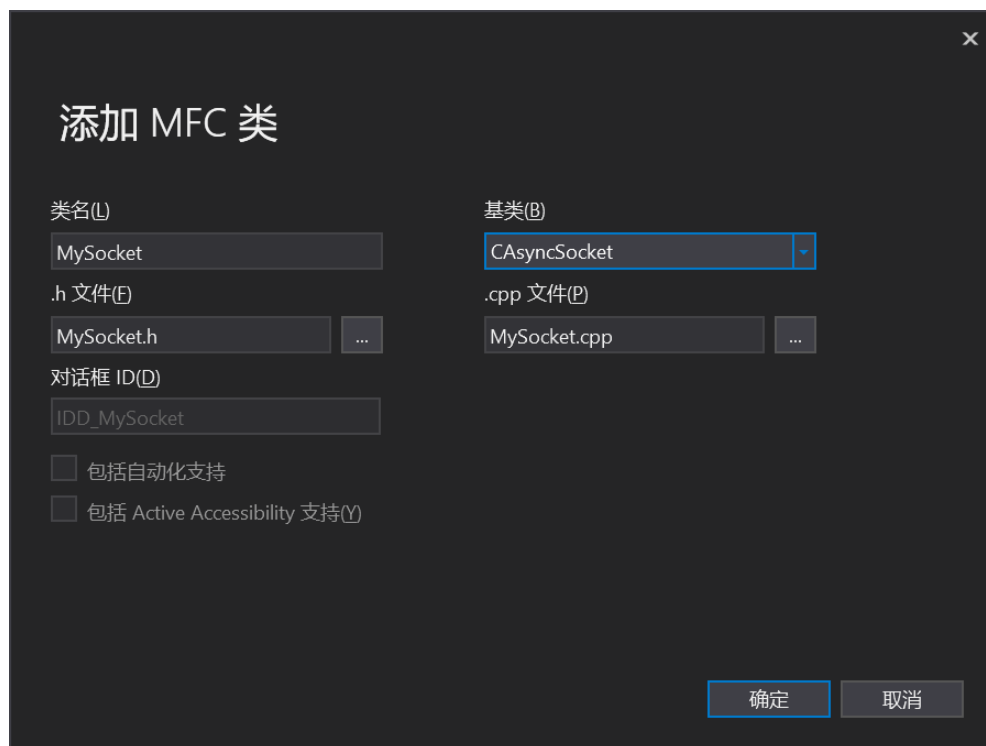
### 5.1 客户端

当客户端程序初始化创建的时候会创建一个继承于 CAsyncSocket 类的自定义类 MySocket，当用户点击“发送”按钮的时候，会触发一个响应函数，在此函数内会使用 SendToEx () 函数以 UDP 数据报方式向当前控件上显示的服务器 IP 和端口号的服务器发送请求命令。然后再重载 MySocket 类里面的 OnReceive () 函数，在其中执行当收到来自服务器的响应数据报时的操作，将其响应内容显示在控件上面。

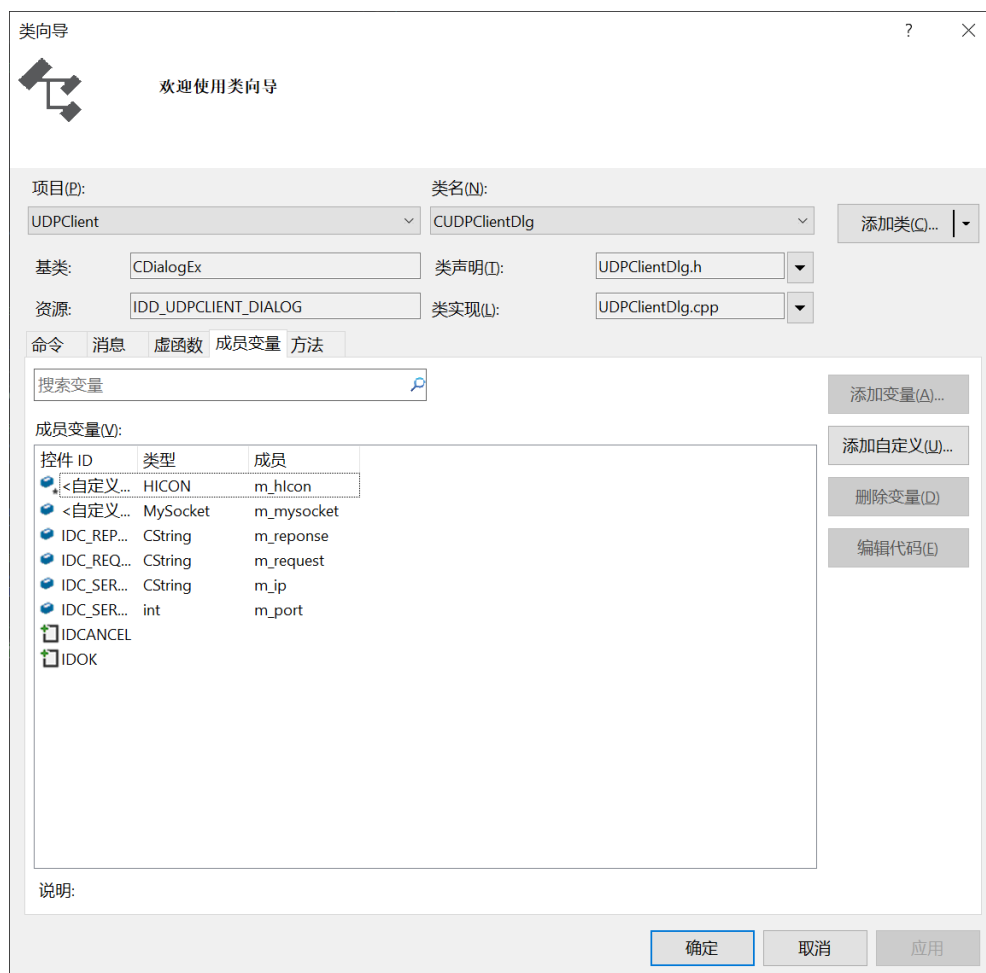
1. 首先新建一个基于对话框的MFC项目。
2. 然后在资源视图里画出如下对话框。



3. 使用类向导新建一个继承CAsyncSocket的MFC类，命名为MySocket。



#### 4. 使用类向导，对对话框类的变量进行申明和初始化



结果如下

```

CUDPClientDlg::CUDPClientDlg(CWnd* pParent /*=nullptr*/)
    : CDialogEx(IDD_UDPCLIENT_DIALOG, pParent)
    , m_port(2000)//控件对应的变量
    , m_ip(_T(""))
    , m_reponse(_T(""))
    , m_request(_T(""))
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

```

5. 在OnInitDialog()函数里添加以下代码

```

// TODO: 在此添加额外的初始化代码
if (!AfxSocketInit())//初始化
{
    MessageBox(L"初始化失败", L"提示", MB_OK |
    MB_ICONSTOP);
}
//产生随机端口
srand((unsigned)time(NULL));
int max = RAND_MAX;
int client_port = (int)(rand() * 5000 / max + 2019);
if (!m_mysocket.Create(client_port, SOCK_DGRAM, FD_READ))
    //create函数第一个参数是指定的端口号, 第二个是连接类型, 这里是
    //drame, 第三个是监测的事件, 这里为读取。
{
    //以数据报方式创建socket
    MessageBox(L"Socket套接字创建失败", L"错误", MB_OK |
    MB_ICONSTOP);//一个确定按钮, 和一个黄色图标
}

```

6. 创建点击清空实现清除编辑框的功能的函数

```

void CUDPClientDlg::OnBnClickedCancel()
{
    // TODO: 在此添加控件通知处理程序代码
    //清空输入框, 这里是建一个指针指向EDIT框, 然后设为空字, 然后更新数
    //据
    CEdit* pedt = (CEdit*)GetDlgItem(IDC_SERVERIP);
    pedt->SetWindowText(_T(""));
    CEdit* req = (CEdit*)GetDlgItem(IDC_REQUEST);
    req->SetWindowText(_T(""));
    CEdit* rep = (CEdit*)GetDlgItem(IDC_REPONSE);
    rep->SetWindowText(_T(""));
    UpdateData();
}

```

7. 创建点击发送按钮的操作函数

```

void CUDPClientDlg::OnBnClickedOk()
{
    // TODO: 在此添加控件通知处理程序代码
    //先更新数据，然后进行错误处理
    UpdateData(true);
    if(m_ip.IsEmpty())//IP不能为空

    {
        MessageBox(_T("无服务器IP地址! "), _T("错误! "), MB_OK |
MB_ICONEXCLAMATION);
        return;
    }
    if (m_port <= 0 || m_port > 65535)//port在一定范围
    {
        MessageBox(_T("端口值不正确! "), _T("错误! "), MB_OK |
MB_ICONEXCLAMATION);
        return;
    }
    if (m_request.IsEmpty())//请求不能为空
    {
        MessageBox(_T("无请求命令! "), _T("错误! "), MB_OK |
MB_ICONEXCLAMATION);
        return;
    }
    //数据发送到特定目标，未发生错误函数返回发送字符总数，错误返回
    SOCKET_ERROR,
    //参数：包含数据的缓冲区，数据的长度（以字节为单位），端口号，一个计
    算机的IP。
    int flag = m_mysocket.SendToEx(m_request.GetBuffer()/*得到
指向缓冲区的非常量指针*/， (m_request.GetLength() + 1) *
sizeof(WCHAR), m_port, m_ip);
    if (flag == SOCKET_ERROR)
    {
        MessageBox(_T("请求失败! "), _T("错误! "), MB_OK |
MB_ICONSTOP);
        return;
    }
}

```

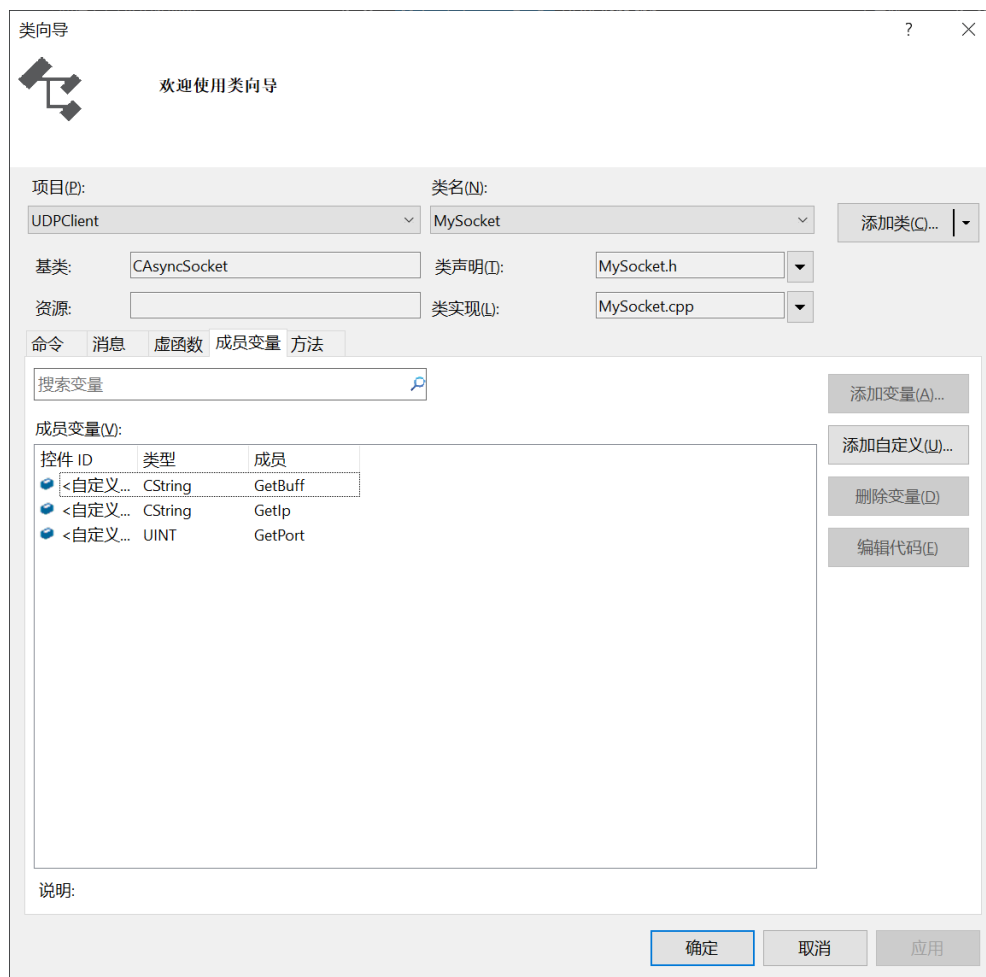
8. 关闭对话框时要关闭套接字并销毁窗口

```

void CUDPClientDlg::OnClose()
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值
    m_mysocket.Close();
    DestroyWindow();
    CDialogEx::OnClose();
}

```

9. 重载 MySocket 类里的 OnReceive()函数，添加接收到服务器返回数据的操作代码。当然之前也要使用类向导申明变量与方法。



```
void MySocket::OnReceive(int nErrorCode)
{
    // TODO: 在此添加专用代码和/或调用基类
    TCHAR buff[4096];
    //接受服务器发回的数据，放入缓冲区，数据的长度，
    ReceiveFrom(buff, sizeof(buff), GetIp, GetPort);
    CString data(buff); //数据赋值给data;
    GetBuff = data; //传给全局变量
    CUDPCClientDlg* dig = (CUDPCClientDlg*)theApp.m_pMainWnd; //
    获取窗口的句柄
    dig->m_reponse = GetBuff; //然后把值改为服务器相应的值
    dig->UpdateData(false); //不更新数据了。

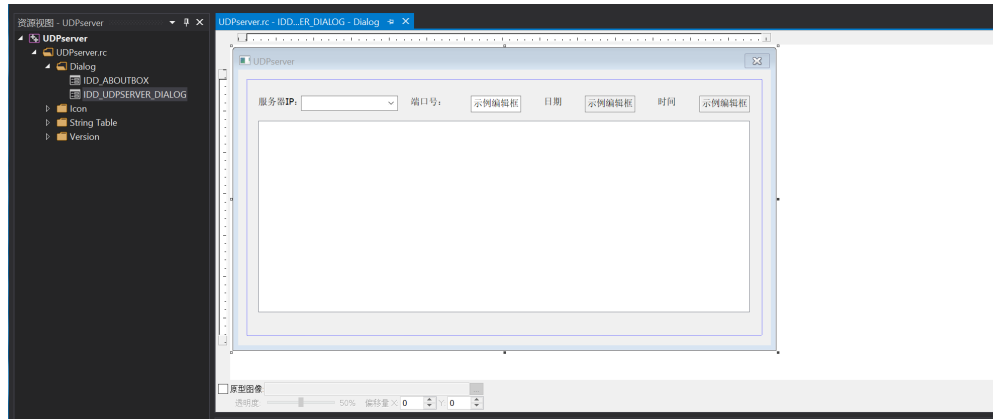
    CAsyncSocket::OnReceive(nErrorCode);
}
```

## 5.2 服务器

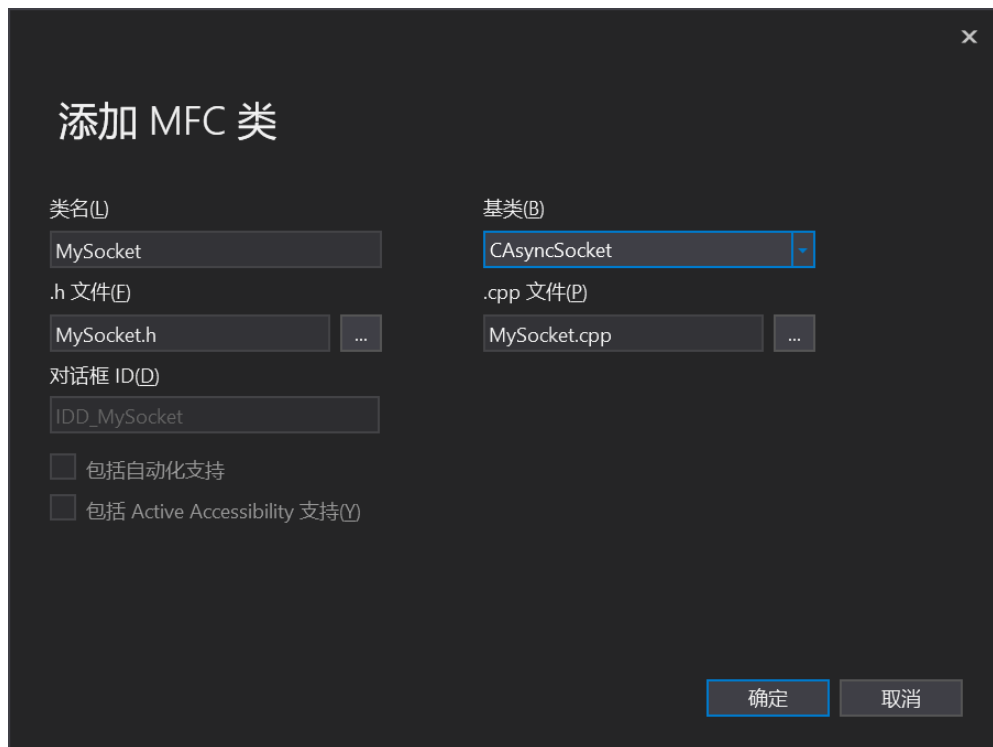
在服务器端主要的流程是当服务器端程序初始化创建的时候同样会创建一个继承于 CAsyncSocket 类的自定义类 MySocket，此时会查找本机可用的 IP 地址列表，并选择第一个 IP 和端口号创建一个 MySocket 类，然后重载 MySocket 类里面的 OnReceive () 函数执行监听操作，当有客户端发送数据报时，使用 SendToEx () 函数以 UDP 数据报方式向客户端返回响应内容。

1. 首先新建一个基于对话框的MFC项目。

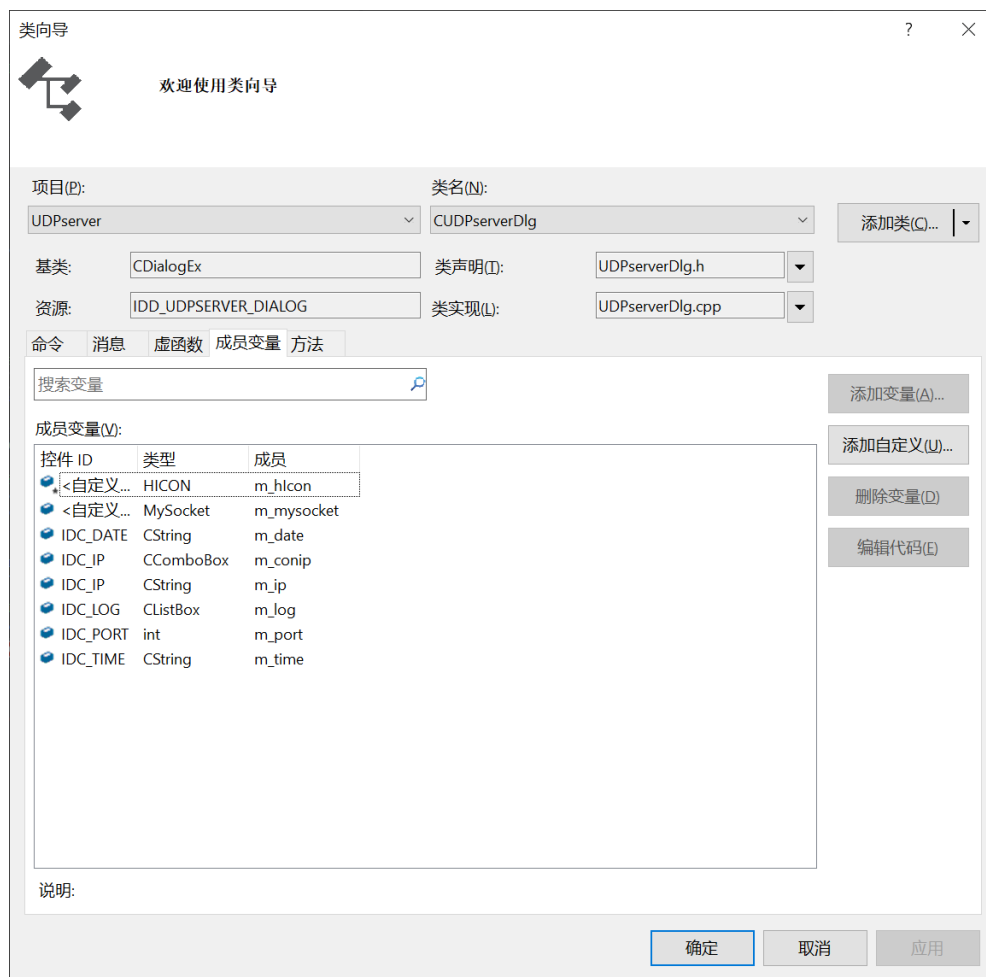
2. 然后在资源视图里画出如下对话框。



3. 使用类向导新建一个继承CAsyncSocket的MFC类，命名为MySocket。



4. 使用类向导，对对话框类的变量进行申明和初始化



结果如下：

```
CUDPServerDlg::CUDPServerDlg(CWnd* pParent /*=nullptr*/)
: CDialogEx(IDD_UDPSERVER_DIALOG, pParent)
, m_date(_T(""))
, m_port(2000)
, m_time(_T(""))
, m_ip(_T(""))
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```

5. 在 `OnInitDialog()` 函数里添加以下代码获取主机名：

```
// TODO: 在此添加额外的初始化代码
//获取主机名
update_date_and_time(m_date, m_time); //获取当前日期和时间放入。

if (!AfxSocketInit()) { //初始化

    MessageBox(L"初始化失败", L"提示", MB_OK |
MB_ICONSTOP);

}
```



```

        update_date_and_time(m_date, m_time); //获取当前日期和时间

        m_log.AddString(m_date + _T(" ") + m_time + _T(":") +
        _T("Socket初始化成功! "));
        //listbox添加记录

        char hostname[20] = "";
        int errorcode;
        if ((errorcode = gethostname(hostname, sizeof(hostname)))
        != 0) //通过获取主机名
        {
            char char_error[100];
            _itoa_s(errorcode, char_error, 10); //转化为字符串
            MessageBoxA(this->GetSafeHwnd(), char_error,
            "errorcode", MB_OK); //获取对话框的句柄然后输出错误信息
            AfxGetMainWnd()->SendMessage(WM_CLOSE); //程序退出

        }
        CString hostname_cstring(hostname); //把char*转化为CString
        CString record;
        record.Format(L"当前获取的主机名是: 【%s】",
        hostname_cstring); //记录格式化
        update_date_and_time(m_date, m_time);
        m_log.AddString(m_date + _T(" ") + m_time + _T(":") +
        record)); //记录时间

```

## 6. 获取IP地址并显示

```

hostent* hn;

hn = gethostbyname(hostname); //通过主机名获取主机信息

/*
返回hostent结构体类型指针
struct hostent
{
    char    *h_name;
    char    **h_aliases;
    int     h_addrtype;
    int     h_length;
    char    **h_addr_list;
    #define h_addr h_addr_list[0]
};

hostent->h_name
表示的是主机的规范名。例如www.google.com的规范名其实是
www.l.google.com。

hostent->h_aliases

```

表示的是主机的别名。www.google.com就是google他自己的别名。有的时候，有的主机可能有好几个别名，这些，其实都是为了易于用户记忆而为自己的网站多取的名字。

hostent->h\_addrtype

表示的是主机ip地址的类型，到底是ipv4(AF\_INET)，还是pv6(AF\_INET6)

hostent->h\_length

表示的是主机ip地址的长度

hostent->h\_addr\_list

表示的是主机的ip地址，注意，这个是以网络字节序存储的。千万不要直接用printf带%s参数来打这个东西，会有问题的哇。所以到真正需要打印出这个IP的话，需要调用inet\_ntop()。

const char \*inet\_ntop(int af, const void \*src, char \*dst, socklen\_t cnt) :

这个函数，是将类型为af的网络地址结构src，转换成主机序的字符串形式，存放在长度为cnt的字符串中。返回指向dst的一个指针。如果函数调用错误，返回值是NULL。

```
*/
int i = 0;

while (hn->h_addr_list[i] != 0) {

    char* p = inet_ntoa(*(in_addr*)hn->h_addr_list[i++]);

    wchar_t pw[20];

    SHAnsiToUnicode(p, pw, 20);

    CString str;

    str.Format(L"%s", pw); //转化为CSTRING

    m_conip.AddString(str);

    record.Format(L"当前主机第 %d 个可用IP地址是: 【 %s 】",
i, pw);

    update_date_and_time(m_date, m_time); //更新日期和时间

    m_log.AddString(m_date + _T(" ") + m_time + _T(":") +
record); //增加记录

}
if (i == 0) {

    AfxGetMainWnd()->SendMessage(WM_CLOSE);
    //关闭程序

}
```

```

UpdateData(false);

m_conip.SetCurSel(0); //强制选择第一个项
UpdateData(true);

```

## 7. 启动服务器

```

if (m_ip.IsEmpty())

{
    MessageBox(_T("无服务器Ip地址! "), _T("错误! "), MB_OK |
MB_ICONEXCLAMATION); return TRUE;
}

if (m_port <= 0 || m_port > 65535)

{
    MessageBox(_T("端口值设置错误! "), _T("错误! "), MB_OK |
MB_ICONEXCLAMATION); return TRUE;
}

UpdateData(true);

if (!m_mysocket.Create(m_port, SOCK_DGRAM, FD_READ,
m_ip)) { //以数据报方式创建socket

    MessageBox(L"Socket套接字创建失败", L"错误", MB_OK |
MB_ICONSTOP); return TRUE;

};

record.Format(L"启动服务器成功! 当前主机IP是: 【 %s 】, 端口号
是: 【 %u 】! ", m_ip, m_port);

update_date_and_time(m_date, m_time); //更新日期和时间

m_log.AddString(m_date + _T(" ") + m_time + _T(":") +
record);

```

## 8. 获取当前时间的函数

```

void CUDPserverDlg::update_date_and_time(CString& date,
CString& time)//获取当前的信息
{
    date = CTime::GetCurrentTime().Format(L"%Y/%m/%d");

    time = CTime::GetCurrentTime().Format("%H:%M:%S");

    UpdateData(false);// TODO: 在此处添加实现代码。
}

```

## 9. 监听客户机发送来的信息

```

void MySocket::OnReceive(int nErrorCode)
{
    // TODO: 在此添加专用代码和/或调用基类
    CUDPserverDlg* dig = (CUDPserverDlg*)theApp.m_pMainWnd;//
    获取窗口句柄
    TCHAR buff[4096];
    CString GetIp;//客户端的IP
    UINT GetPort;//客户端的端口号
    ReceiveFrom(buff, sizeof(buff), GetIp, GetPort);//接受发回
    的数据存在各个变量里。这个为客户端发来的请求数据
    CString data(buff);
    CString temp = data;
    dig->update_date_and_time(dig->m_date, dig->m_time);//把当前
    信息赋值给date&time

    CString Answer = _T("错误请求");
    data.MakeLower();//不管大小写全部置为小写

    if (data == "time")
        Answer = dig->m_time;//结果为当前时间
    if (data == "date")
        Answer = dig->m_date;//结果为当前日期

    SendToEx(Answer.GetBuffer(), (Answer.GetLength() + 1) *
    sizeof(WCHAR), GetPort, GetIp);//再把处理后的结果发回客户端
    CString record;
    record.Format(L": 收到IP=%s, Port=%u请求【%s】,响应【%s】",
    GetIp, GetPort, temp, Answer);//把收到的信息进行格式化, s为
    string, u为UINT
    dig->m_log.AddString(dig->m_date + _T(" ") + dig->m_time
    + record);//把记录发送到listbox
    CAsyncSocket::OnReceive(nErrorCode);
}

```

## 6 实验实现功能介绍

### 6.1 客户端

客户端“UDPClient.exe”打开后显示界面如下：



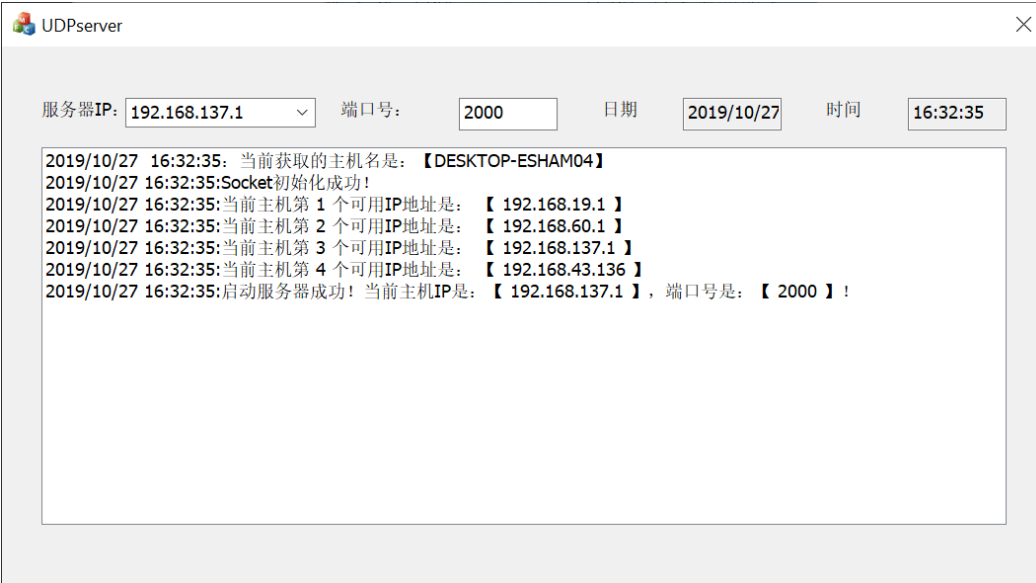
The UDPClient window contains the following fields and buttons:

- 服务器IP:
- 服务器端口号:
- 请求命令:
- 服务器相应:
- 发送:
- 清空:

1. 服务器 IP：此控件上由用户输入 Ip 地址值作为即将发送的服务器 IP 地址。
2. 服务器端口号：此控件上由用户输入端口值作为即将发送的服务器端口值。
3. 请求命令：在此控件上由用户输入请求命令作为数据报发往服务器。按照此程序设计协议，请求命令可以是 date 和 time（不区分大小写）会返回服务器本地的日期和时间，其余命令都会返回“错误请求”。
4. 服务器响应：在此控件上会显示服务器响应发送请求命令的值。按照此程序设计协议，响应会是服务器本地的日期、时间和“错误请求”三种情况。
5. 发送：用户点击此按钮之后客户端会将请求命令按照服务器 IP 地址值和端口发送到服务器，并在服务器响应控件上显示服务器响应内容。
6. 清空：用户点击此按钮之后会清空 `服务器IP`、`请求命令`、`服务器响应`。

## 6.2 服务器端

服务器“UDPServer.exe”打开后显示界面如下：



The UDPServer window displays the following information:

- 服务器IP:
- 端口号:
- 日期:
- 时间:
- 日志内容:
 

```

2019/10/27 16:32:35: 当前获取的主机名是: 【DESKTOP-ESHAM04】
2019/10/27 16:32:35: Socket初始化成功!
2019/10/27 16:32:35: 当前主机第 1 个可用IP地址是: 【 192.168.19.1 】
2019/10/27 16:32:35: 当前主机第 2 个可用IP地址是: 【 192.168.60.1 】
2019/10/27 16:32:35: 当前主机第 3 个可用IP地址是: 【 192.168.137.1 】
2019/10/27 16:32:35: 当前主机第 4 个可用IP地址是: 【 192.168.43.136 】
2019/10/27 16:32:35: 启动服务器成功! 当前主机IP是: 【 192.168.137.1 】, 端口号是: 【 2000 】!

```

1. 服务器 IP：在此控件上会显示本程序当前运行环境的 IP 地址列表，并且获得的第一个 IP 地址创建服务器 Socket 类。
2. 端口号：会以此控件上显示的端口号创建服务器 Socket 类，默认不输入之前端口号是 2000。
3. 日期：在此控件上会显示最近一次 UDP 服务器动作时的本地的日期值。
4. 时间：在此控件上会显示最近一次 UDP 服务器动作时的本地的时间值。
5. 空白对话框：此部分会按照动作执行的顺序显示服务器的工作日志，在开始阶段会显示“初始化成功或失败”，接着会显示本机的主机名，然后逐条显示本机的所有可用的 IP 地址，最后会显示以 IP 地址列表里的第一个 IP 地址和端口号显示的数值创建服务器

Socket类。然后就会以下的格式显示客户端连接记录。 日期 时间 收到 IP =  
xx.xxx.xx.xxx Port=xxxx 请求【】， 相应【】

### 6.3 交互效果展示

1. 客户端发送正确的 date 指令，服务器显示客户端连接日志，然后响应客户端并在客户端上显示响应日期内容：

The image shows two windows: UDPClient and UDPserver. UDPClient has fields for Server IP (192.168.137.1), Server Port (2000), Request Command (date), and Server Response (2019/10/27). UDPserver shows a log of events, including the successful initialization of the Socket and the receipt of a 'date' request from 192.168.137.1, which it responds to with '2019/10/27'.

UDPCient

服务器IP: 192.168.137.1 服务器端口号: 2000

请求命令: date 服务器相应: 2019/10/27

发送 清空

UDPserver

服务器IP: 192.168.137.1 端口号: 2000 日期: 2019/10/27 时间: 16:43:30

2019/10/27 16:37:12: 当前获取的主机名是: 【DESKTOP-ESHAM04】  
2019/10/27 16:37:12:Socket初始化成功!  
2019/10/27 16:37:12:当前主机第 1 个可用IP地址是: 【 192.168.19.1 】  
2019/10/27 16:37:12:当前主机第 2 个可用IP地址是: 【 192.168.60.1 】  
2019/10/27 16:37:12:当前主机第 3 个可用IP地址是: 【 192.168.137.1 】  
2019/10/27 16:37:12:当前主机第 4 个可用IP地址是: 【 192.168.43.136 】  
2019/10/27 16:37:12:启动服务器成功!当前主机IP是: 【 192.168.137.1 】, 端口号是: 【 2000 】!  
2019/10/27 16:43:30: 收到IP=192.168.137.1, Port=2808请求【date】,响应【2019/10/27】

2. 客户端发送正确的 time 指令，服务器显示客户端连接日志，然后响应客户端并在客户端上显示响应时间内容：

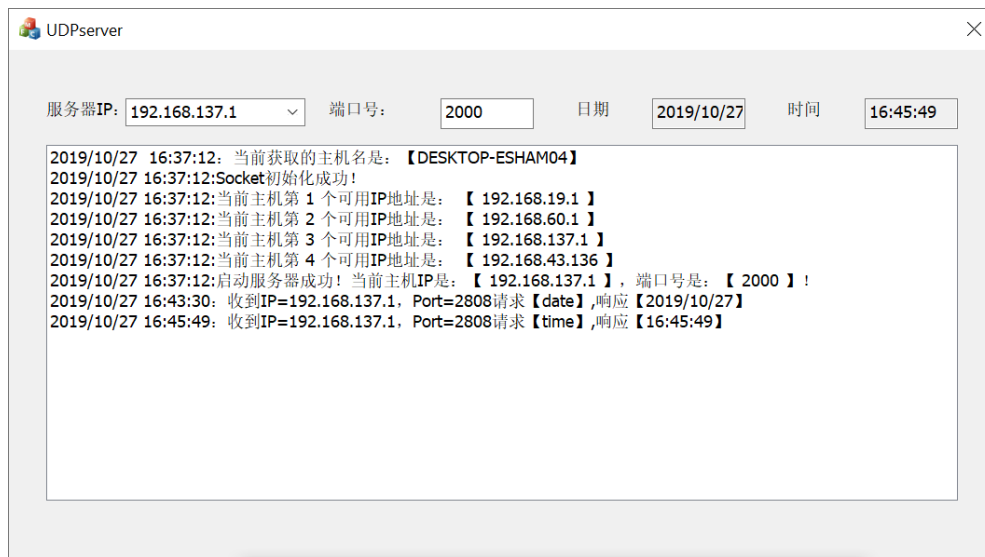
The image shows the UDPClient window with the Request Command field set to 'time' and the Server Response field showing '16:45:49'.

UDPCient

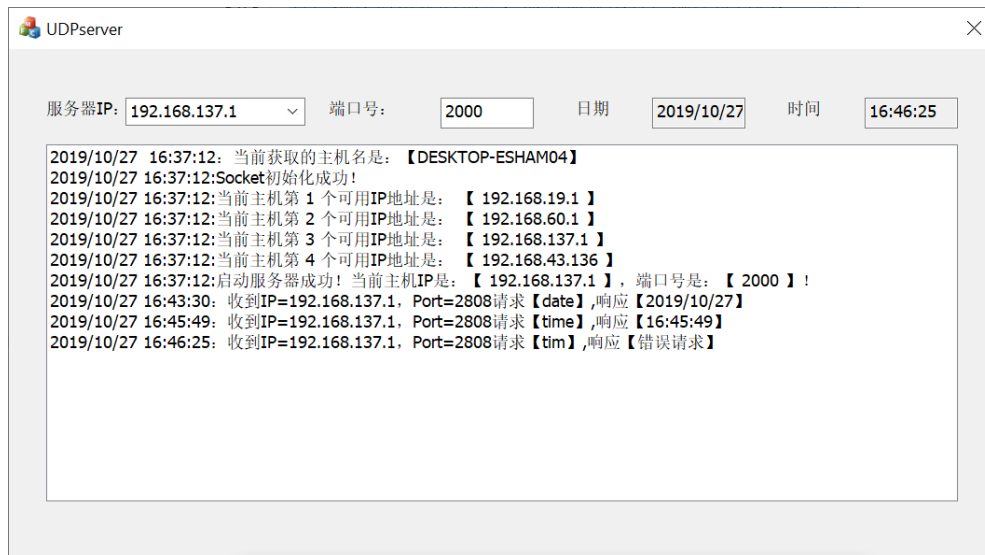
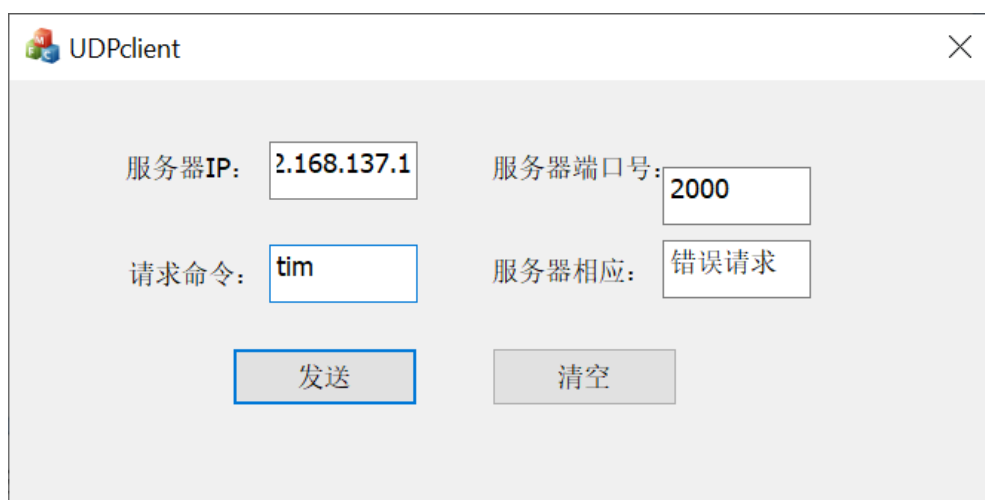
服务器IP: 192.168.137.1 服务器端口号: 2000

请求命令: time 服务器相应: 16:45:49

发送 清空



3. 客户端发送错误的指令，服务器显示客户端连接日志，然后响 应客户端并在客户端上显示响应“请求错误”：



4. 客户端点击清空按钮，清空编辑框：

UDPclient

服务器IP:  服务器端口号:

请求命令:  服务器相应:

5. 不输入服务器IP

UDPclient

服务器IP:  服务器端口号:

请求命令:  服务器相应:

错误!

! 无服务器IP地址!

OK

6. 不输入请求命令

UDPclient

服务器IP:  服务器端口号:

请求命令:  服务器相应:

错误!

! 无请求命令!

OK

7. 输入错误端口值

UDPclient

服务器IP:  服务器端口号:

请求命令:  服务器相应:

错误!

! 端口值不正确!

OK



## 8. 输入错误IP

