2020/08/10_网络编程_第4课_TCP

笔记本: 网络编程
创建时间: 2020/8/10 星期一 10:05
作者: ileemi

协议的定义和检测很重要，要注意其健壮性

TCP

UDP的缺点：发送的数据过多的时候，包易丢失，操作系统将收到的数据放入到缓冲区中，当缓冲区满的时候，剩下的包就将其丢掉。

> TCP：面向连接
> UDP：面向无连接

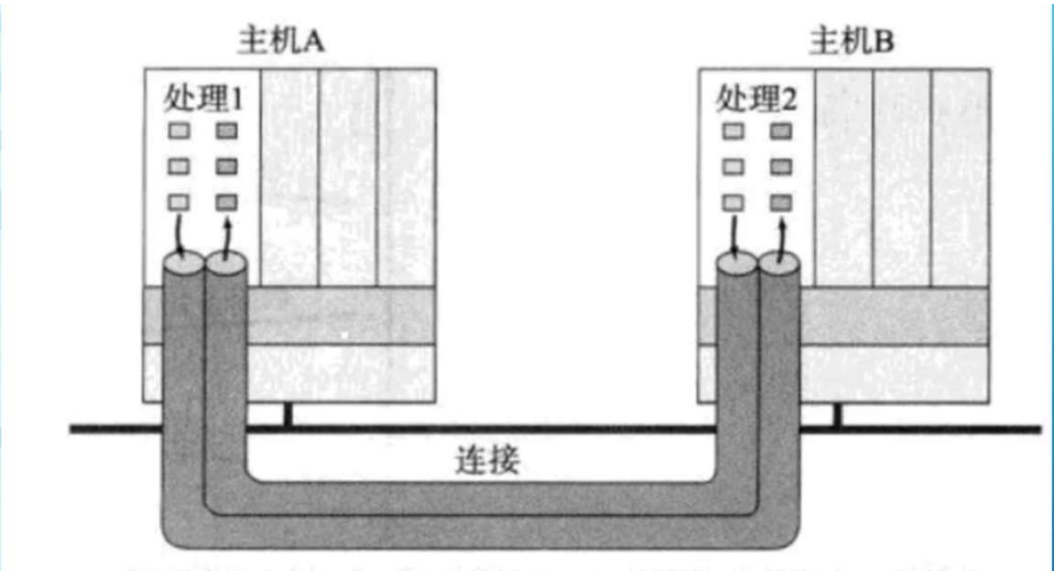
TCP -- 较为健壮

TCP与UDP的区别相当大。它充分地实现了数据传输时各种控制功能，可以进行丢包时的重发控制，还可以对次序乱掉的分包进行顺字控制。而这些UDP没有。此外，TCP作为一种面向有连接的协议，只有确认通信对端存在时才会发送数据，从而可以控制通信流量的浪费。

1. 确认机制（ACK -- 服务器收到消息后回复一个确认包）-- 确认数据是否发送，接收
2. 超时重发/传
3. 乱序重排（先发后致） -- 发送数据的线路导致发送数据到达的顺序颠倒，经过一段时间后将数据包进行排序
4. 流量控制（保证缓冲区中有空间可以存放数据）-- 控制数据发送的时间
5. 三次握手（连接）-- 确认对方（双方）身份，确定完才能传输数据（两次存在不可靠性）
6. 四次挥手（关闭连接）
7. 数据校验

缺点：效率比较低
但是为了追求数据的可靠性，TCP使用度要比UCP的使用度高，可根据合适的应用场景选择合适的通信协议。

当连接建立好以后进行通信时，应用程序只需要通过管道的出入口发送或接受数据，就可以实现与对端的网络通信。

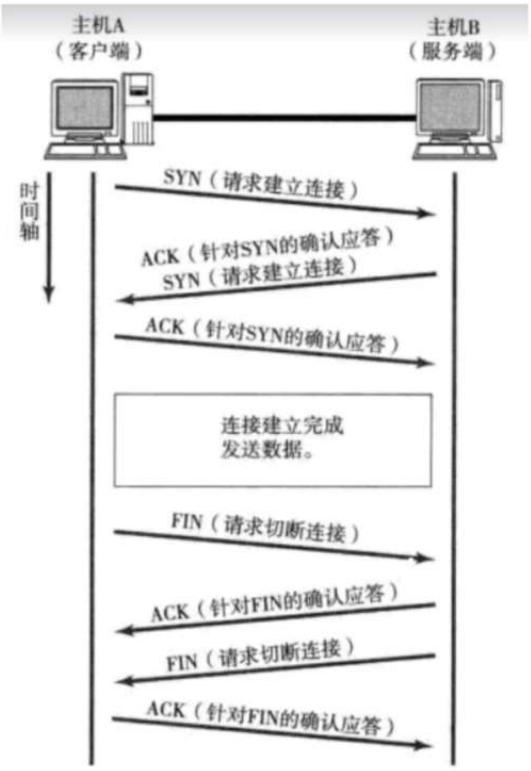


一般游戏的发送的数据量比较大的时候，都是用UDP，并在其基础上进行扩展以满足自己的需求。

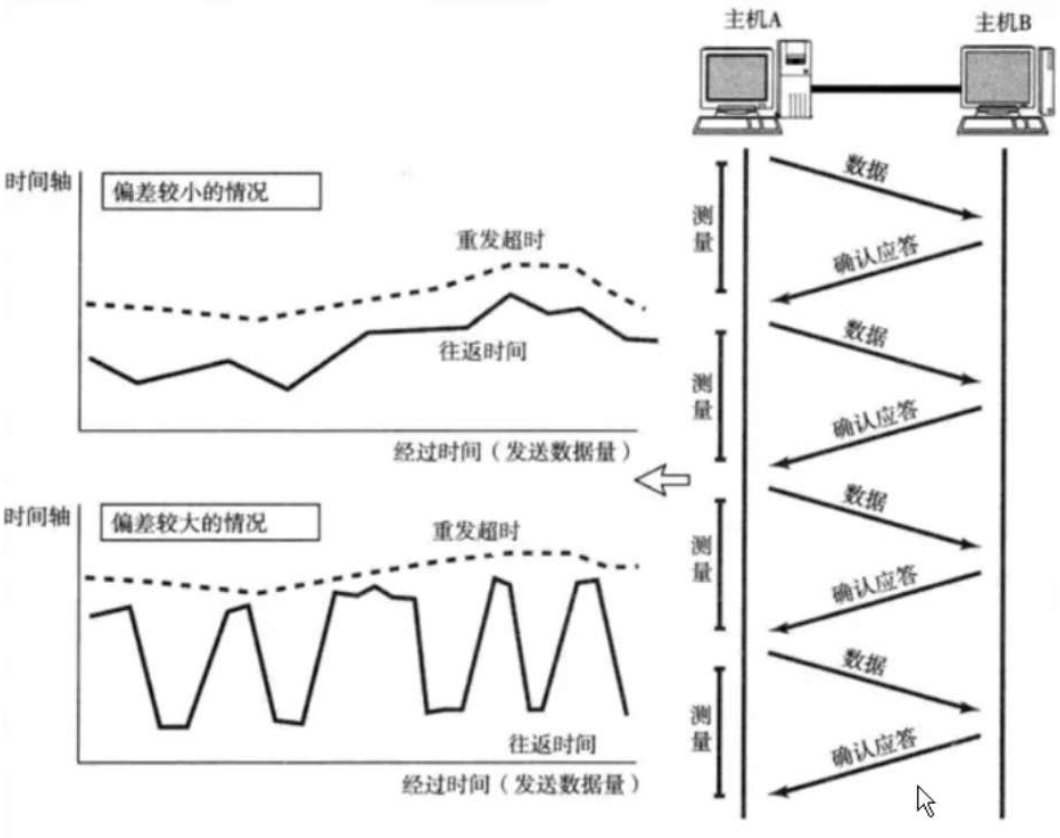
三次握手 -- 建立连接

产生一个随机数（SEQ ）
主机A -- SEQ --> 主机B
主机B -- SYN,ACK --> 主机A
主机A -- ACK -- > 主机B

四次挥手 -- 关闭连接

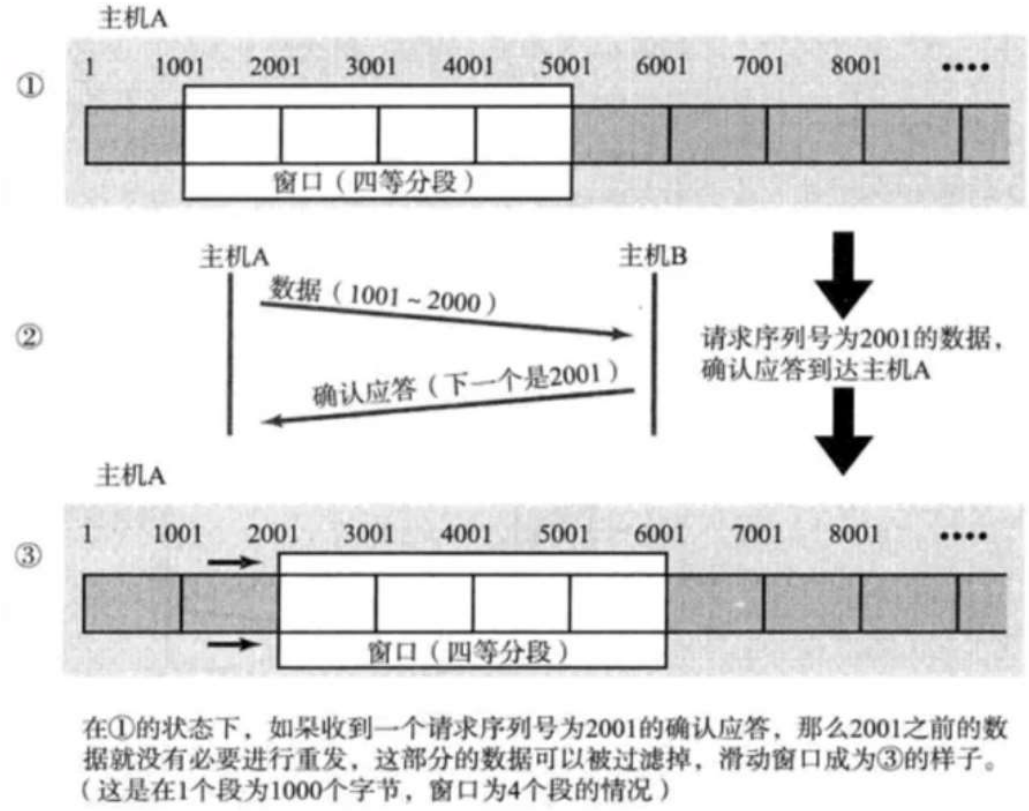


超时重发机制



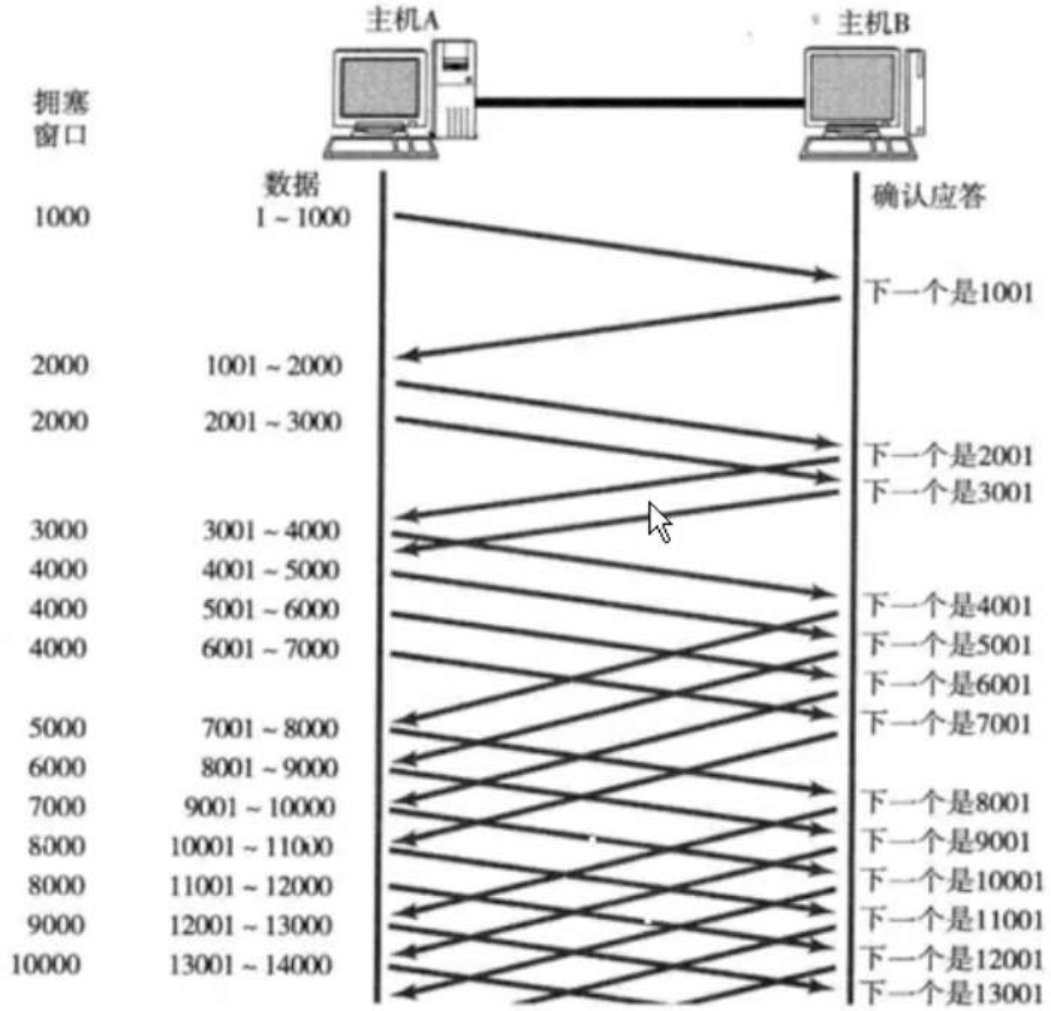
先发后至问题

有一个窗口机制 -- 根据情况确定发送数据包的个数，根据网速等



在①的状态下，如果收到一个请求序列号为2001的确认应答，那么2001之前的数据就没有必要进行重发，这部分的数据可以被过滤掉，滑动窗口成为③的样子。（这是在1个段为1000个字节，窗口为4个段的情况）

拥塞机制



最初将发送端的窗口（拥塞窗口）设置为1。每收到一个确认应答，窗口的值会增加1个段。（图中所示为没有延迟确认应答的情况，因此与实际情况有所不同）

TCP协议结构

控制位：ACK FIN SYN
校验和：不对，将数据包丢弃， 强制使用，确保数据包的正确性（网际校验和算法）。
紧急指针：额外带其它包的数据

使用TCP 建立服务器客户端

服务器：

1. **包含头文件**
2. **初始化套接字** 绑定协议（使用 **SOCK_STREAM**）
3. **绑定端口**
4. **监听** API：`listen` 客户端建立连接会产生对应缓冲区数据
5. **接受客户端连接**：与客户端进行三次握手 API：`accept` -- **通过接受客户端连接的返回值进行数据通信（不是通过客户端的IP地址和端口进行数据通信）**
6. **收发数据** 接受数据：`recv` 发送数据：`send` （接收数据和发送数据API的参数不需要填写目标IP以及端口）

客户端：不需要 绑定端口，监听

1. **包含头文件**
2. **初始化套接字** 绑定协议（使用TCP）
3. **连接服务器** API：`connect` （服务器连接失败，客户端不允许发送数据）
4. **收发数据** 发送数据 API -- send

客户端强制退出，会自动四次挥手。

在 MFC 工程中，勾选 "Windows 套接字" 后就不在需要在项目工程中 "初始化套接字"。

nagle 算法

建立一个缓冲区，先将数据发送到缓冲区中，缓冲区中有一个计时器，没有数据到来，就将缓冲区中的数据发送给服务器，缓冲区满的时候也将数据发送给服务器。

客户端使用 **send** 发送数据，返回值正确的时候不代表数据到达服务器，一次性发送。发送的数据量过大的时候，根据缓冲区的大小接受客户端发送的数据包。

粘包：数据包会分指定的批次粘贴在一起发送给服务器，数据包发送效率提高。在Windows上不建议关闭，强制开启。不粘包发送，效率就会变低。

解决粘包问题：发送数据的时候，同时发送数据包的长度，服务器接受数据包的时候，优先接受数据包的长度，之后服务器根据数据包的长度从缓冲区中读取对应长度的数据。

UDP 不存在粘包情况，TCP 发送的数据超过缓冲区的内存时，会拆分成多次发送。

服务器接受客服端发送数据的缓冲区不能大于客户端发送的数据长度，读取数据会越界。

线程创建的最大数量有计算机硬件决定的

代码示例

服务器：

```
// TCPServer.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
```

```
//
#include <stdio.h>
#include <Winsock2.h>
#include <ws2tcpip.h>
#pragma comment(lib, "Ws2_32.lib")

SOCKET g_Socket;

class CSocket {
public:
    CSocket() {
        //初始化套接字库
        WSADATA wsaData;
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            printf("
[server] WSAStartup error:%08X\n", WSAGetLastError());
        }
    }
    ~CSocket() {
        //反初始化库
        WSACleanup();
    }
}g_init;

void show_error_msg(const char* pre) {
    LPVOID lpMsgBuf;
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        WSAGetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language

        (LPTSTR)& lpMsgBuf,
        0,
        NULL
    );
    printf("%s:%s", pre, (LPCTSTR)lpMsgBuf);
    // Free the buffer.
    LocalFree(lpMsgBuf);
}

// 接收客户端发送数据的线程
DWORD __stdcall WorkThread(LPVOID lpParam) {
    SOCKET s = (SOCKET)lpParam;
    //收发数据
```

```
    char* pBuff = NULL;    // 存储从客户端发送到缓冲区中的数据
    int nDataLen;      // 存储客户端发送数据的长度
    int nRet;
    while (true) {
        //先接收长度
        nRet = recv(s, (char*)&nDataLen, sizeof(nDataLen), 0);    //
阻塞
        if (nRet <= 0) {
            show_error_msg("[server] recv error");
            break;
        }
        // 根据客户端发送数据的长度申请对应的缓冲区
        pBuff = (char*)malloc(nDataLen + 1);
        if (pBuff == NULL) {
            break;
        }
        //循环接收数
据        recv()    len=2000    while()  {recv(2000)    ret = 1000}
        int cur_len = 0;                          // 当前收取的字
节数
        int total = nDataLen;            // 总共收取的字节数
        while (cur_len < total)
        {
            // 每次收取的字节数为 总字节数 - 当前收取的字节数
            nRet = recv(s, pBuff, total - cur_len, 0); //2000

            if (nRet <= 0) {
                show_error_msg("[server] recv error");
                break;
            }
            cur_len += nRet;
        }
        pBuff[cur_len] = '\0';
        printf("[server] buf:%s len:%d\n", pBuff, nRet);
        //nagle算法     3       6    流    沾包（如何解决） len msg
        free(pBuff);
        //send
    }
    // 关闭客户端sokcet
    closesocket(s);
    return 0;
}

int main()
{
    //TCP服务器
    /*
```

```
        面向连接：
        1.确认机制（ACK）
        2.超时重传
        3.乱序重排（先发后致）
        4.流量控制
        5.三次握手（连接）
        6.四次挥手（关掉连接）
        7.数据校验
        UDP：面向无连接
    */
    //  1.初始化套接字(说明使用的协议)
    g_Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); //TCP协议
    if (INVALID_SOCKET == g_Socket) {
        printf("
[server] socket init error:%08X\n", WSAGetLastError());
        return 0;
    }
    printf("[server] socket init ok s=%08X\n", g_Socket);

    //  2.绑定端口
    sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(5566);
    inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);

    //  2.绑定
    if (bind(g_Socket, (sockaddr*)&addr, sizeof(addr)) == SOCKET_ERROR)

        show_error_msg("bind error");
        return 0;
    }
    printf("[server] bind ok\n");

    //3.监听
    if (listen(g_Socket, SOMAXCONN) == SOCKET_ERROR) {
        show_error_msg("[server] listen error");
        return 0;
    }
    printf("[server] listen OK\n");

    //  4.接受客户端连接（与客户端进行三次握手）
    printf("[server] accept...\n");
    sockaddr_in caddr;
    int len = sizeof(caddr);

    //  接受多个客户端连接
    while (true) {
```

```
        // 阻塞，等待客户端连接
        SOCKET s = accept(g_Socket, (sockaddr*)& caddr, &len);
        if (s == INVALID_SOCKET) {
            show_error_msg("[server] accept error");
            return 0;
        }
        printf("[server] accept OK ip:%s port:%d\n",
            inet_ntoa(caddr.sin_addr), htons(caddr.sin_port));
        // 为每个连接的客户端创建一个线程
        CreateThread(NULL, 0, WorkThread, (LPVOID)s, 0, NULL);
    }

    // 关闭服务器socket
    closesocket(g_Socket);
    return 0;
}
```

客户端：

```
// Client.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//
#include <stdio.h>
#include <Winsock2.h>
#include <ws2tcpip.h>
#pragma comment(lib, "Ws2_32.lib")

SOCKET g_Socket;

class CSocket {
public:
    CSocket() {
        //初始化套接字库
        WSADATA wsaData;
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            printf("[server] WSAStartup error:%08X\n",
            WSAGetLastError());
        }
    }
    ~CSocket() {
        //反初始化库
        WSACleanup();
    }
}g_init;
```

```
// 显示错误信息
void show_error_msg(const char* pre) {
    LPVOID lpMsgBuf;
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        WSAGetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language

        (LPTSTR)& lpMsgBuf,
        0,
        NULL
    );
    printf("%s:%s", pre, (LPCTSTR)lpMsgBuf);
    // Free the buffer.
    LocalFree(lpMsgBuf);
}

int main()
{
    //1.初始化套接字(说明使用的协议)
    g_Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); //TCP协议
    if (INVALID_SOCKET == g_Socket) {
        show_error_msg("[client] socket init error");
        return 0;
    }
    printf("[client] socket init ok s=%08X\n", g_Socket);

    // 2. 连接服务器（等价于三次握手）
    sockaddr_in addr; // 存储服务器的相关信息
    addr.sin_family = AF_INET;
    addr.sin_port = htons(5566);
    inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);
    // 在TCP中客户端没有连接服务器，是不能向服务器发送数据的
    if (connect(g_Socket, (sockaddr*)& addr, sizeof(addr)) == SOCKET_

        show_error_msg("[client] connect server error");
        return 0;
    }
    // 连接服务器成功
    printf("[client] connect server ok\n");

    // 3. 收发数据
    char szBuff[260];
```

```
    int  nRet;    //  存储向服务器发送的数据包长度
    int  nLen;    //  存储发送信息的字节长度
    while  (true)  {
            printf("[msg]:");
            scanf_s("%s",  szBuff,  sizeof(szBuff));
            nLen  =  strlen(szBuff);  //  int：65535、short：32765
            //  发送数据包的长度
            nRet  =  send(g_Socket,  (char*)&nLen,  sizeof(nLen),  0);
            if  (nRet  <=  0)  {
                show_error_msg("[client]  send  server  error");
                break;
            }
            //  发送数据
            nRet  =  send(g_Socket,  (char*)szBuff,  nLen,  0);
            if  (nRet  <=  0)  {
                show_error_msg("[client]  send  server  error");
                break;
            }
            printf("[client]  send  server  ok  bytes:%d\n",  nRet);
    }
    //关闭socket
    closesocket(g_Socket);
    return  0;
}
```