# UDP 接收端和UDP发送端的笔记一起做,因为他们是一对

# 学习大纲

|  |
| --- |
|  |
|  |

## sendto函数的用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **sendto** 函数将数据发送到特定目标。 语法 C++  int WSAAPI sendto(  [in] SOCKET s,  [in] const char \*buf,  [in] int len,  [in] int flags,  [in] const sockaddr \*to,  [in] int tolen  ); 参数 [in] s  标识可能连接的 () 套接字的描述符。  [in] buf  指向包含要传输的数据的缓冲区的指针。  [in] len  *buf* 参数指向的数据的长度（以字节为单位）。  [in] flags  一组指定调用方式的标志。  [in] to  指向包含目标套接字地址 [的 sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构的可选指针。  [in] tolen  *由 to* 参数指向的地址的大小（以字节为单位）。 返回值 如果未发生错误， **sendto** 将返回发送的总字节数，这可能小于 *len* 指示的数量。 否则，将返回值 SOCKET\_ERROR，并且可以通过调用 [WSAGetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsagetlasterror) 检索特定的错误代码。  展开表   |  |  | | --- | --- | | **错误代码** | **含义** | | [**WSANOTINITIALIZED**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 在使用此函数之前，必须成功调用 [WSAStartup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsastartup) 。 | | [**WSAENETDOWN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 网络子系统发生故障。 | | [**WSAEACCES**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | ：请求的地址是广播地址，但相应的标志未设置。 使用 SO\_BROADCAST 参数调用 [setsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt) 以允许使用广播地址。 | | [**WSAEINVAL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 指定了未知标志，或者为启用了SO\_OOBINLINE的套接字指定了MSG\_OOB。 | | [**WSAEINTR**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 阻止的 Windows 套接字 1.1 调用已通过 [WSACancelBlockingCall](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsacancelblockingcall) 取消。 | | [**WSAEINPROGRESS**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 阻止 Windows 套接字 1.1 调用正在进行，或者服务提供商仍在处理回调函数。 | | [**WSAEFAULT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | *buf* 或 *to* 参数不是用户地址空间的一部分，或者*被盗*参数太小。 | | [**WSAENETRESET**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 由于操作执行过程中持续的活动检测到故障，连接损坏。 | | [**WSAENOBUFS**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 未提供任何缓冲区空间。 | | [**WSAENOTCONN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 仅) (面向连接的套接字未连接套接字。 | | [**WSAENOTSOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | ：描述符不是套接字。 | | [**WSAEOPNOTSUPP**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | MSG\_OOB指定，但套接字不是流样式（如类型SOCK\_STREAM），与此套接字关联的通信域中不支持 OOB 数据，或者套接字是单向的，仅支持接收操作。 | | [**WSAESHUTDOWN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字已关闭;在[调用关闭](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-shutdown)*并设置为 SD\_SEND* 或 SD\_BOTH 后，无法在套接字上 sendto。 | | [**WSAEWOULDBLOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | ：套接字标记为非阻止，请求的操作会进行阻止。 | | [**WSAEMSGSIZE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字面向消息，消息大于基础传输支持的最大值。 | | [**WSAEHOSTUNREACH**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 目前无法从此主机访问远程主机。 | | [**WSAECONNABORTED**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 由于超时或其他故障，虚拟线路已终止。 因为套接字不可再用，应用程序应关闭套接字。 | | [**WSAECONNRESET**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 执行硬性或异常关闭的远程端重置了虚拟线路。 对于 UPD 套接字，远程主机无法传递以前发送的 UDP 数据报，并使用“端口无法访问”ICMP 数据包进行响应。 因为套接字不可再用，应用程序应关闭套接字。 | | [**WSAEADDRNOTAVAIL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 远程地址不是有效的地址，例如，ADDR\_ANY。 | | [**WSAEAFNOSUPPORT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 指定系列中的地址无法与此套接字一起使用。 | | [**WSAEDESTADDRREQ**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | ：需要目标地址。 | | [**WSAENETUNREACH**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 此时不可以从此主机访问该网络。 | | [**WSAEHOSTUNREACH**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字操作尝试访问无法访问的主机。 | | [**WSAETIMEDOUT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 由于网络故障或另一端的系统在未通知的情况下关闭，连接已断开。 |  备注 **sendto** 函数用于在套接字上写入传出数据。 对于面向消息的套接字，必须注意不要超过基础子网的最大数据包大小，这可以通过使用 [getsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockopt) 检索套接字选项的值SO\_MAX\_MSG\_SIZE获得。 如果数据太长，无法通过基础协议以原子方式传递，则会返回错误 [WSAEMSGSIZE](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) ，并且不会传输任何数据。  *to* 参数可以是套接字地址系列中的任何有效地址，包括广播地址或任何多播地址。 若要发送到广播地址，应用程序必须使用 [启用了SO\_BROADCAST的 setsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt) 。 否则， **sendto** 将失败并显示错误代码 [WSAEACCES](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。 对于 TCP/IP，应用程序可以发送到任何多播地址 (，而无需成为组成员) 。  **注意** 如果打开套接字，进行 [**setsockopt**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt) 调用，然后进行 **sendto** 调用，Windows 套接字将执行隐式 **绑定** 函数调用。    如果套接字未绑定，系统会将唯一值分配给本地关联，然后将套接字标记为绑定。 如果套接字已连接，则 [getsockname](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockname) 函数可用于确定与套接字关联的本地 IP 地址和端口。  如果未连接套接字，则为 [getockname](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockname) 函数可用于确定与套接字关联的本地端口号，但返回的 IP 地址设置为给定协议的通配符地址 (例如，INADDR\_ANY或 IPv4 的“0.0.0.0”，对于 IPv6) IN6ADDR\_ANY\_INIT或“：：”。  **sendto** 的成功完成并不表示数据已成功传递。  **sendto** 函数通常用于无连接套接字，以将数据报发送到由 *to* 参数标识的特定对等套接字。 即使无连接套接字以前已连接到特定地址， *to* 参数也仅覆盖该特定数据报的目标地址。 在面向连接的套接字上，忽略 *to* 和 *tolen* 参数，使 **sendto** 等效于 [send](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-send)。  **注意** 发出阻止的 Winsock 调用（如 **sendto）**时，Winsock 可能需要等待网络事件，然后才能完成呼叫。 在这种情况下，Winsock 执行可发出警报的等待， (在同一线程上计划的 APC) 异步过程调用可能会中断。 在 APC 内发出另一个阻止 Winsock 调用，该调用中断了同一线程上正在进行的阻止 Winsock 调用将导致未定义的行为，并且 Winsock 客户端绝不能尝试。   示例代码 以下示例演示如何使用 **sendto** 函数。  C++复制  #ifndef UNICODE  #define UNICODE  #endif  #define WIN32\_LEAN\_AND\_MEAN  #include <winsock2.h>  #include <Ws2tcpip.h>  #include <stdio.h>  // Link with ws2\_32.lib  #pragma comment(lib, "Ws2\_32.lib")  int main()  {  int iResult;  WSADATA wsaData;  SOCKET SendSocket = INVALID\_SOCKET;  sockaddr\_in RecvAddr;  unsigned short Port = 27015;  char SendBuf[1024];  int BufLen = 1024;  //----------------------  // Initialize Winsock  iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);  if (iResult != NO\_ERROR) {  wprintf(L"WSAStartup failed with error: %d\n", iResult);  return 1;  }  //---------------------------------------------  // Create a socket for sending data  SendSocket = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);  if (SendSocket == INVALID\_SOCKET) {  wprintf(L"socket failed with error: %ld\n", WSAGetLastError());  WSACleanup();  return 1;  }  //---------------------------------------------  // Set up the RecvAddr structure with the IP address of  // the receiver (in this example case "192.168.1.1")  // and the specified port number.  RecvAddr.sin\_family = AF\_INET;  RecvAddr.sin\_port = htons(Port);  RecvAddr.sin\_addr.s\_addr = inet\_addr("192.168.1.1");  //---------------------------------------------  // Send a datagram to the receiver  wprintf(L"Sending a datagram to the receiver...\n");  iResult = sendto(SendSocket,  SendBuf, BufLen, 0, (SOCKADDR \*) & RecvAddr, sizeof (RecvAddr));  if (iResult == SOCKET\_ERROR) {  wprintf(L"sendto failed with error: %d\n", WSAGetLastError());  closesocket(SendSocket);  WSACleanup();  return 1;  }  //---------------------------------------------  // When the application is finished sending, close the socket.  wprintf(L"Finished sending. Closing socket.\n");  iResult = closesocket(SendSocket);  if (iResult == SOCKET\_ERROR) {  wprintf(L"closesocket failed with error: %d\n", WSAGetLastError());  WSACleanup();  return 1;  }  //---------------------------------------------  // Clean up and quit.  wprintf(L"Exiting.\n");  WSACleanup();  return 0;  } 对于使用 IP (版本 4 的套接字) 若要仅) SOCK\_DGRAM发送广播 (，可以构造 *由 参数* 指向的地址，以包含 Winsock2.h) 中定义的特殊 IPv4 地址INADDR\_BROADCAST (以及预期的端口号。 如果 *由 to* 参数指向的地址包含INADDR\_BROADCAST地址和预期端口，则广播将在所有接口上发送到该端口。  如果广播只应在特定接口上发送，则 *to* 参数指向的地址应包含接口的子网广播地址和预期端口。 例如，子网掩码为 255.255.255.0 的 IPv4 网络地址 192.168.1.0 将使用子网广播地址 192.168.1.255。  广播数据报通常不应超过出现的碎片的大小，这意味着数据报的数据部分（不包括标头）不应超过 512 字节。  如果传输系统中没有可用于保存要传输的数据的缓冲区空间，除非套接字已置于非阻止模式， **否则 sendto** 将阻止。 在非阻止、面向流的套接字上，写入的字节数可以介于 1 和请求的长度之间，具体取决于客户端和服务器系统上的缓冲区可用性。 [select](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-select)、[WSAAsyncSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsaasyncselect) 或 [WSAEventSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaeventselect) 函数可用于确定何时可以发送更多数据。  允许调用 *len 为零*的 **sendto**，并将返回零作为有效值。 对于面向消息的套接字，将发送零长度传输数据报。  *flags* 参数可用于影响函数调用的行为，超出为关联套接字指定的选项。 此函数的语义由套接字选项和 *flags* 参数确定。 后者是使用以下任何值的按位 OR 运算符构造的。  展开表   |  |  | | --- | --- | | **值** | **含义** | | MSG\_DONTROUTE | ：指定数据不应受到路由的约束。 Windows 套接字服务提供商可以选择忽略此标志。 | | MSG\_OOB | 仅) SOCK\_STREAM等流式套接字 (发送 OOB 数据。 |     **Windows Phone 8：**Windows Phone 8 及更高版本上的 Windows Phone 应用商店应用支持此函数。  **Windows 8.1** 和 **Windows Server 2012 R2**：Windows 8.1、Windows Server 2012 R2 及更高版本的 Windows 应用商店应用支持此功能。 要求  |  |  | | --- | --- | | **标头** | winsock2.h (包括 Winsock2.h) | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 [WSAAsyncSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsaasyncselect)  [WSAEventSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaeventselect)  [Winsock 函数](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/winsock-functions)  [Winsock 参考](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/winsock-reference)  [recv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recv)  [recvfrom](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recvfrom)  [select](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-select)  [send](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-send)  [socket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-socket) |

## recvfrom函数的语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **recvfrom** 函数接收数据报并存储源地址。 语法 C++  int recvfrom(  [in] SOCKET s,  [out] char \*buf,  [in] int len,  [in] int flags,  [out] sockaddr \*from,  [in, out, optional] int \*fromlen  ); 参数 [in] s  标识绑定套接字的描述符。  [out] buf  传入数据的缓冲区。  [in] len  *buf* 参数指向的缓冲区的长度（以字节为单位）。  [in] flags  一组选项，用于修改函数调用的行为，超出为关联套接字指定的选项。 有关更多详细信息，请参阅下面的“备注”。  [out] from  指向 [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构中的缓冲区的可选指针，该缓冲区将在返回时保存源地址。  [in, out, optional] fromlen  指向 *from* 参数指向的缓冲区大小（以字节为单位）的可选指针。 返回值 如果未发生错误， **recvfrom** 将返回收到的字节数。 如果连接已正常关闭，则返回值为零。 否则，将返回值 SOCKET\_ERROR，并且可以通过调用 [WSAGetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsagetlasterror) 来检索特定的错误代码。  展开表   |  |  | | --- | --- | | **错误代码** | **含义** | | [**WSANOTINITIALISED**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 在使用此函数之前，必须成功调用 [WSAStartup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsastartup) 。 | | [**WSAENETDOWN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 网络子系统失败。 | | [**WSAEFAULT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | *buf* 或 *from* 参数指向的缓冲区不在用户地址空间中，或者 *fromlen* 参数太小，无法容纳对等地址的源地址。 | | [**WSAEINTR**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | (阻止) 调用已通过 [WSACancelBlockingCall](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsacancelblockingcall) 取消。 | | [**WSAEINPROGRESS**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 阻止 Windows Sockets 1.1 调用正在进行，或者服务提供程序仍在处理回调函数。 | | [**WSAEINVAL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字未绑定 [绑定](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind)，或者指定了未知标志，或者为启用了SO\_OOBINLINE的套接字指定了MSG\_OOB，或者仅) *len* 为零或负数的字节流样式套接字 (。 | | [**WSAEISCONN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字已连接。 无论连接套接字是面向连接的还是无连接的套接字，都不允许使用此函数。 | | [**WSAENETRESET**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 对于数据报套接字，此错误显示生存时间已经过期。 | | [**WSAENOTSOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | *s* 参数中的描述符不是套接字。 | | [**WSAEOPNOTSUPP**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | MSG\_OOB已指定，但套接字不是流样式（如类型SOCK\_STREAM），与此套接字关联的通信域中不支持 OOB 数据，或者套接字是单向的，仅支持发送操作。 | | [**WSAESHUTDOWN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字已关闭;在调用[关闭](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-shutdown)后，无法[从](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recvfrom)套接字上重新显示SD\_RECEIVE或SD\_BOTH。 | | [**WSAEWOULDBLOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字标记为非阻塞， [recvfrom](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recvfrom) 操作将阻止。 | | [**WSAEMSGSIZE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 消息太大，无法容纳 *到 buf* 参数指向的缓冲区中，并且被截断。 | | [**WSAETIMEDOUT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 由于网络故障或另一端的系统在未通知的情况下出现故障，连接已断开。 | | [**WSAECONNRESET**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 执行硬性或异常关闭的远程端重置了虚拟线路。 应用程序应关闭套接字;它不再可用。 在 UDP 数据报套接字上，此错误指示以前的发送操作导致 ICMP *端口无法访问* 消息。 |  注解 **recvfrom** 函数读取已连接和未连接的套接字上的传入数据，并捕获从中发送数据的地址。 此函数通常用于无连接套接字。 套接字的本地地址必须是已知的。 对于服务器应用程序，这通常通过 [绑定](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind)显式完成。 不建议对客户端应用程序进行显式绑定。 对于使用此函数的客户端应用程序，套接字可以通过 [sendto](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-sendto)、 [WSASendTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasendto) 或 [WSAJoinLeaf](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsajoinleaf) 隐式绑定到本地地址。  对于面向流的套接字（如类型为 SOCK\_STREAM 的套接字），对 **recvfrom** 的调用将返回当前可用的信息量（最大为指定缓冲区的大小）。 如果套接字已配置为内联接收 OOB 数据 (套接字选项SO\_OOBINLINE) 且 OOB 数据尚未读取，则仅返回 OOB 数据。 应用程序可以使用 [ioctlsocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-ioctlsocket) 或 [WSAIoctl](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaioctl)**SIOCATMARK** 命令来确定是否还有更多 OOB 数据需要读取。 对于面向连接的套接字，将忽略 *from* 和 *fromlen* 参数。  对于面向消息的套接字，从第一个排队的消息中提取数据，最大大小为指定的缓冲区大小。 如果数据报或消息大于指定的缓冲区，则用数据报的第一部分填充缓冲区， **recvfrom** 将生成错误 [WSAEMSGSIZE](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。 例如，对于不可靠的协议 (，UDP) 会丢失多余的数据。 对于 UDP，如果收到的数据包不包含 (空) 的数据，则 **来自 recvfrom** 函数的返回值为零。  如果 *from* 参数不为零，并且套接字不面向连接， (类型SOCK\_DGRAM例如) ，则发送数据的对等方的网络地址将复制到相应的 [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构。 *fromlen* 指向的值初始化为此结构的大小，并在返回时进行修改，以指示**存储在 sockaddr** 结构中的地址的实际大小。  如果套接字上没有可用的传入数据， **则 recvfrom** 函数会根据为 [WSARecv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsarecv) 定义的阻止规则阻止并等待数据到达，除非套接字为非阻止，否则不会设置MSG\_PARTIAL标志。 在这种情况下，返回值 SOCKET\_ERROR，错误代码设置为 [WSAEWOULDBLOCK](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。 [select](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-select)、[WSAAsyncSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsaasyncselect) 或 [WSAEventSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaeventselect) 可用于确定何时到达更多数据。  如果套接字面向连接，并且远程端已正常关闭连接，则对 **recvfrom** 的调用将立即完成，并且接收了零个字节。 如果连接已重置 **，则 recvfrom** 将失败，并显示 [错误 WSAECONNRESET](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。  *flags* 参数可用于影响为关联套接字指定的选项之外的函数调用行为。 此函数的语义由套接字选项和 *flags* 参数确定。 后者是使用以下任一值的按位 OR 运算符构造的。  展开表   |  |  | | --- | --- | | **值** | **含义** | | MSG\_PEEK | 查看传入数据。 数据会复制到缓冲区中，但不会从输入队列中删除。 | | MSG\_OOB | 处理带外 (OOB) 数据。 |     **注意** 发出阻止 Winsock 调用（如 **recvfrom**）时，Winsock 可能需要等待网络事件，然后调用才能完成。 在这种情况下，Winsock 执行可发出警报的等待， (在同一线程上计划的 APC) 异步过程调用可能会中断该等待。 在 APC 内发出另一个阻止 Winsock 调用，该调用中断了同一线程上正在进行的阻止 Winsock 调用将导致未定义的行为，并且 Winsock 客户端绝不能尝试。   示例代码 以下示例演示如何使用 **recvfrom** 函数。  C++复制  #ifndef UNICODE  #define UNICODE  #endif  #define WIN32\_LEAN\_AND\_MEAN  #include <winsock2.h>  #include <Ws2tcpip.h>  #include <stdio.h>  // Link with ws2\_32.lib  #pragma comment(lib, "Ws2\_32.lib")  int main()  {  int iResult = 0;  WSADATA wsaData;  SOCKET RecvSocket;  struct sockaddr\_in RecvAddr;  unsigned short Port = 27015;  char RecvBuf[1024];  int BufLen = 1024;  struct sockaddr\_in SenderAddr;  int SenderAddrSize = sizeof (SenderAddr);  //-----------------------------------------------  // Initialize Winsock  iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);  if (iResult != NO\_ERROR) {  wprintf(L"WSAStartup failed with error %d\n", iResult);  return 1;  }  //-----------------------------------------------  // Create a receiver socket to receive datagrams  RecvSocket = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);  if (RecvSocket == INVALID\_SOCKET) {  wprintf(L"socket failed with error %d\n", WSAGetLastError());  return 1;  }  //-----------------------------------------------  // Bind the socket to any address and the specified port.  RecvAddr.sin\_family = AF\_INET;  RecvAddr.sin\_port = htons(Port);  RecvAddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  iResult = bind(RecvSocket, (SOCKADDR \*) & RecvAddr, sizeof (RecvAddr));  if (iResult != 0) {  wprintf(L"bind failed with error %d\n", WSAGetLastError());  return 1;  }  //-----------------------------------------------  // Call the recvfrom function to receive datagrams  // on the bound socket.  wprintf(L"Receiving datagrams...\n");  iResult = recvfrom(RecvSocket,  RecvBuf, BufLen, 0, (SOCKADDR \*) & SenderAddr, &SenderAddrSize);  if (iResult == SOCKET\_ERROR) {  wprintf(L"recvfrom failed with error %d\n", WSAGetLastError());  }    //-----------------------------------------------  // Close the socket when finished receiving datagrams  wprintf(L"Finished receiving. Closing socket.\n");  iResult = closesocket(RecvSocket);  if (iResult == SOCKET\_ERROR) {  wprintf(L"closesocket failed with error %d\n", WSAGetLastError());  return 1;  }  //-----------------------------------------------  // Clean up and exit.  wprintf(L"Exiting.\n");  WSACleanup();  return 0;  }  **Windows Phone 8：**Windows Phone 8 及更高版本上的 Windows Phone 应用商店应用支持此函数。  **Windows 8.1**和**Windows Server 2012 R2**：Windows 8.1、Windows Server 2012 R2 及更高版本的 Windows 应用商店应用支持此函数。 要求  |  |  | | --- | --- | | **标头** | winsock.h (包括 Winsock2.h) | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 [WSAAsyncSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsaasyncselect)  [WSAEventSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaeventselect)  [Winsock 函数](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/winsock-functions)  [Winsock 参考](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/winsock-reference)  [recv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recv)  [send](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-send)  [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2)  [socket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-socket) |

# 演练

## 先编写服务端程序

## 1.新建一个项目,取名:Lesson76-77-udp,然后只保留解决方案,把项目删除

|  |
| --- |
|  |

## 2.在解决方案上面点击右键->添加->新建项目,取名upd-server,是一个常规空项目

|  |
| --- |
|  |

## 3.一个简单的udp服务器的代码如下,下面注释掉的是微软提供的UNICODE版本程序也是ok的

|  |
| --- |
| #pragma comment(lib,"Ws2\_32.lib")  #include<stdio.h>  #include<stdlib.h>  #include <Winsock2.h>  void main()  {  //变量定义  WSADATA wsaData;  SOCKET serverSocket;//套接字  SOCKADDR\_IN serverAddr;//保存服务端地址的变量  SOCKADDR\_IN clientAddr;//保存客户端地址的变量  int addrlen;//客户端地址长度  int port = 3000;//定义服务器端口  int ret;  //1.初始化套接字库  if((ret = WSAStartup(MAKEWORD(2,2),&wsaData))!=0)  {  printf("初始化套接字库失败\n");  return;  }  else  {  if(LOBYTE(wsaData.wVersion)!=2 || HIBYTE(wsaData.wVersion)!=2)  {  //版本不对就清理  printf("套接字库版本错误\n");  WSACleanup();  return;  }  }  //2.创建套接字  serverSocket = socket(AF\_INET,SOCK\_DGRAM,IPPROTO\_UDP);    //2.2设置服务器地址和端口  serverAddr.sin\_family = AF\_INET;  serverAddr.sin\_port = htons(port);  serverAddr.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);  //3.绑定套接字和服务器端口地址  if(bind(serverSocket,(SOCKADDR\*)&serverAddr,sizeof(serverAddr))==SOCKET\_ERROR)  {  printf("绑定服务器地址错误,错误码：%d\n",WSAGetLastError());  closesocket(serverSocket);  WSACleanup();  return;  }  //4.接收数据  addrlen = sizeof(clientAddr);  while(1)  {  char buffer[1024] = {0};  printf("等等客户端信息...\n");  if(recvfrom(serverSocket,buffer,sizeof(buffer),0,(SOCKADDR\*)&clientAddr,&addrlen)==SOCKET\_ERROR)  {  printf("接收数据错误\n");  closesocket(serverSocket);  WSACleanup();  return;  }  else  {  printf("客户-%s说:%s",inet\_ntoa(clientAddr.sin\_addr),buffer);  //偷懒方法，把客户分数过来的消息有原样发送给客户  if(sendto(serverSocket,buffer,sizeof(buffer),0,(SOCKADDR\*)&clientAddr,addrlen)==SOCKET\_ERROR)  {  printf("发送数据错误\n");  closesocket(serverSocket);  WSACleanup();  return;  }  }  Sleep(500);  }  closesocket(serverSocket);  WSACleanup();  system("pause");  }  //下面这个是UNICODE版本的程序，是OK的。  //#ifndef UNICODE  //#define UNICODE  //#endif  //  //#define WIN32\_LEAN\_AND\_MEAN  //  //#include <winsock2.h>  //#include <Ws2tcpip.h>  //#include <stdio.h>  //  //// Link with ws2\_32.lib  //#pragma comment(lib, "Ws2\_32.lib")  //  //int main()  //{  //  // int iResult = 0;  //  // WSADATA wsaData;  //  // SOCKET RecvSocket;  // struct sockaddr\_in RecvAddr;  //  // unsigned short Port = 27015;  //  // char RecvBuf[1024];  // int BufLen = 1024;  //  // struct sockaddr\_in SenderAddr;  // int SenderAddrSize = sizeof (SenderAddr);  //  // //-----------------------------------------------  // // Initialize Winsock  // iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);  // if (iResult != NO\_ERROR) {  // wprintf(L"WSAStartup failed with error %d\n", iResult);  // return 1;  // }  // //-----------------------------------------------  // // Create a receiver socket to receive datagrams  // RecvSocket = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);  // if (RecvSocket == INVALID\_SOCKET) {  // wprintf(L"socket failed with error %d\n", WSAGetLastError());  // return 1;  // }  // //-----------------------------------------------  // // Bind the socket to any address and the specified port.  // RecvAddr.sin\_family = AF\_INET;  // RecvAddr.sin\_port = htons(Port);  // RecvAddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  //  // iResult = bind(RecvSocket, (SOCKADDR \*) & RecvAddr, sizeof (RecvAddr));  // if (iResult != 0) {  // wprintf(L"bind failed with error %d\n", WSAGetLastError());  // return 1;  // }  // //-----------------------------------------------  // // Call the recvfrom function to receive datagrams  // // on the bound socket.  // wprintf(L"Receiving datagrams...\n");  // iResult = recvfrom(RecvSocket,  // RecvBuf, BufLen, 0, (SOCKADDR \*) & SenderAddr, &SenderAddrSize);  // if (iResult == SOCKET\_ERROR) {  // wprintf(L"recvfrom failed with error %d\n", WSAGetLastError());  // }  //  // //-----------------------------------------------  // // Close the socket when finished receiving datagrams  // wprintf(L"Finished receiving. Closing socket.\n");  // iResult = closesocket(RecvSocket);  // if (iResult == SOCKET\_ERROR) {  // wprintf(L"closesocket failed with error %d\n", WSAGetLastError());  // return 1;  // }  //  // //-----------------------------------------------  // // Clean up and exit.  // wprintf(L"Exiting.\n");  // WSACleanup();  // return 0;  //} |

### 编译程序没有错误,不过现在还不能测试,等我们下面的客户端程序编写完成就可以测试了

## 然后编写客户端

## 1.在解决方案上面点击右键->添加->新建项目,新建一个常规空项目,取名:upd-cient,然后添加一个cpp文件,一个简单的udp客户端的应用程序代码如下

|  |
| --- |
| #pragma comment(lib,"Ws2\_32.lib")  #include<stdio.h>  #include<stdlib.h>  #include <Winsock2.h>  void main(int argc,char\*\* argv)  {    //变量定义  WSADATA wsaData;  SOCKET clientSocket;//套接字  SOCKADDR\_IN serverAddr;//保存服务端地址的变量  SOCKADDR\_IN clientAddr;//保存客户端地址的变量  int addrlen;//客户端地址长度  int port = 3000;//定义服务器端口  int ret;  if(argc < 2)//检查命令行参数是否正确  {  printf("Usage :udp-client <Server address>\n");  return;  }  //1.初始化套接字库  if((ret = WSAStartup(MAKEWORD(2,2),&wsaData))!=0)  {  printf("初始化套接字库失败\n");  WSACleanup();  return;  }  //2.创建套接字  clientSocket = socket(AF\_INET,SOCK\_DGRAM,IPPROTO\_UDP);  //2.2设置服务器地址和端口  serverAddr.sin\_family = AF\_INET;  serverAddr.sin\_port = htons(port);  serverAddr.sin\_addr.S\_un.S\_addr = inet\_addr(argv[1]);  addrlen = sizeof(clientAddr);  //发送数据  while(1)  {  char buffer[1024] = "\0";  printf("请输入消息:\n ");  //scanf("%s",buffer);//scanf函数有一个致命缺点,就是一碰到空格就会返回.也就是不能读取以空格分割的字符串  fgets(buffer,sizeof(buffer),stdin); //使用fgets代替scanf是一个不错的选择  if(strcmp(buffer,"bye")==0) //输入bye就退出程序  {  break;  }    if(!sendto(clientSocket,buffer,sizeof(buffer),0,(SOCKADDR\*)&serverAddr,addrlen))  {  printf("发送消息错误:%d\n ",WSAGetLastError());  break;  }  else  {  if(recvfrom(clientSocket,buffer,sizeof(buffer),0,(SOCKADDR\*)&clientAddr,&addrlen)!=SOCKET\_ERROR)  {  printf("%s : %s\n",inet\_ntoa(serverAddr.sin\_addr),buffer);  }  }  //Sleep(500);  }  closesocket(clientSocket);  WSACleanup();  system("pause");  } |

### 编译生成项目后可以来测试了,先启动服务器,然后注意编写因命令行的方式来启动客户端并且输入服务器的ip地址作为参数,效果如下

|  |
| --- |
|  |

## 注意: scanf函数有一个致命缺点,就是一碰到空格就会返回.也就是不能读取以空格分割的字符串,需要多次读取才行,也就是相当于把字符串分割了.使用fgets代替scanf是一个不错的选择

# 这些只是一些非常简单的程序,只能学习一点点基本原理.没有什么用处需要多多思考,多多参考别人的例子或者多多看书,测试开发有实用价值的程序