# 套接字编程可能是最复杂的进程间通信了.不过他的功能也是最强的的.

# 学习大纲,74和75笔记写在一起,方便对着学习

|  |
| --- |
|  |
|  |

# API用法说明

## WSAStartup函数的语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **WSAStartup** 函数通过进程启动对 Winsock DLL 的使用。 语法 C++复制  int WSAStartup(  [in] WORD wVersionRequired,  [out] LPWSADATA lpWSAData  ); 参数 [in] wVersionRequired  调用方可以使用的最高版本的 Windows 套接字规范。 高位字节指定次要版本号;低位字节指定主版本号。  [out] lpWSAData  指向 [WSADATA](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-wsadata) 数据结构的指针，该结构用于接收 Windows 套接字实现的详细信息。 返回值 如果成功， **WSAStartup** 函数返回零。 否则，它将返回下面列出的错误代码之一。  **WSAStartup** 函数直接返回此函数的返回值中的扩展错误代码。 不需要调用 [WSAGetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsagetlasterror) 函数，也不应使用。  展开表   |  |  | | --- | --- | | **错误代码** | **含义** | | [**WSASYSNOTREADY**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 基础网络子系统尚未准备好进行网络通信。 | | [**WSAVERNOTSUPPORTED**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 此特定 Windows 套接字实现不提供所请求的 Windows 套接字支持版本。 | | [**WSAEINPROGRESS**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 阻止 Windows 套接字 1.1 操作正在进行。 | | [**WSAEPROCLIM**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 已达到 Windows 套接字实现支持的任务数限制。 | | [**WSAEFAULT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | *lpWSAData* 参数不是有效的指针。 |  注解 **WSAStartup** 函数必须是应用程序或 DLL 调用的第一个 Windows 套接字函数。 它允许应用程序或 DLL 指定所需的 Windows 套接字版本，并检索特定 Windows 套接字实现的详细信息。 应用程序或 DLL 只能在成功调用 **WSAStartup** 后发出更多 Windows 套接字函数。  为了支持与最新版本的 Windows 套接字规范具有功能差异的各种 Windows 套接字实现和应用程序，在 **WSAStartup** 中进行了协商。 **WSAStartup** 的调用方将应用程序支持的最高版本的 Windows 套接字规范传入 *wVersionRequested* 参数。 Winsock DLL 指示它可以在其响应中支持的 Windows 套接字规范的最高版本。 Winsock DLL 还会回复它希望调用方使用的 Windows 套接字规范版本。  当应用程序或 DLL 调用 **WSAStartup** 函数时，Winsock DLL 检查在 *wVersionRequested* 参数中传递的应用程序请求的 Windows 套接字规范的版本。 如果应用程序请求的版本等于或高于 Winsock DLL 支持的最低版本，则调用成功，并且 Winsock DLL 将在 *lpWSAData* 参数指向的 [WSADATA](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-wsadata) 结构中返回详细信息。 **WSADATA** 结构的 **wHighVersion** 成员指示 Winsock DLL 支持的 Windows 套接字规范的最高版本。 **WSADATA** 结构的 **wVersion** 成员指示 Winsock DLL 希望调用方使用的 Windows 套接字规范的版本。  如果 **调用** 方无法接受 [WSADATA 结构的 wVersion](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-wsadata) 成员，则应用程序或 DLL 应调用 [WSACleanup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsacleanup) 来释放 Winsock DLL 资源，并且无法初始化 Winsock 应用程序。 为了支持此应用程序或 DLL，需要搜索要安装在平台上的 Winsock DLL 的更新版本。  Windows 套接字规范的当前版本为版本 2.2。 当前的 Winsock DLL *Ws2\_32.dll*支持请求以下任一版本的 Windows 套接字规范的应用程序：   * 1.0 * 1.1 * 2.0 * 2.1 * 2.2   若要完全访问更高版本的 Windows 套接字规范的新语法，应用程序必须协商此更高版本。 在这种情况下， *wVersionRequested* 参数应设置为请求版本 2.2。 应用程序还必须完全符合更高版本的 Windows 套接字规范，例如针对适当的头文件进行编译、与新库链接或其他特殊情况。 Microsoft Windows 软件开发工具包 (Windows SDK) (SDK) 随附了 Winsock 2 支持的 *Winsock2.h 头*文件。  Windows Server 2008、Windows Vista、Windows Server 2003、Windows XP、Windows 2000、Windows NT 4.0 service Pack 4 (SP4) 及更高版本、Windows Me、Windows 98 和 Windows 95 OSR2 支持 Windows 套接字版本 2.2。 在 上也支持 Windows 套接字版本 2.2 带有 Windows 套接字 2 更新的 Windows 95。 这些平台上的应用程序通常应相应地设置 *wVersionRequested* 参数来请求 Winsock 2.2。  在 Windows 95 和 Windows NT 3.51 及更早版本中，Windows 套接字版本 1.1 是受支持的 Windows 套接字规范的最高版本。  为使用 Winsock DLL 支持的较低版本的 Windows 套接字规范编写的应用程序或 DLL 可以使用 **WSAStartup** 函数成功协商此较低版本，这是合法的。 例如，应用程序可以在具有 Winsock 2.2 DLL 的平台上向 **WSAStartup** 函数传递的 *wVersionRequested* 参数中请求版本 1.1。 在这种情况下，应用程序应仅依赖于符合所请求版本的功能。 不应使用新的 Ioctl 代码、现有函数的新行为和新函数。 **WSAStartup** 提供的版本协商主要用于允许为 Windows 95 和 Windows NT 3.51 及更早版本开发的旧版 Winsock 1.1 应用程序以相同的行为在更高版本的 Windows 上运行。 *winsock 1.1* 支持的 Winsock.h 头文件包含在 Windows SDK中。  **WSAStartup** 函数中的这种协商允许使用 Windows 套接字和 Winsock DLL 的应用程序或 DLL 支持一系列 Windows 套接字版本。 如果版本范围有任何重叠，应用程序或 DLL 可以使用 Winsock DLL。 **WSAStartup** 函数返回的 [WSADATA](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-wsadata) 结构中提供了有关 Windows 套接字实现的详细信息。  下表显示了 **WSAStartup** 如何与不同的应用程序和 Winsock DLL 版本配合使用。  展开表   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **调用方版本支持** | **Winsock DLL 版本支持** | **已请求 wVersion** | **返回的 wVersion** | **wHighVersion 返回** | **最终结果** | | 1.1 | 1.1 | 1.1 | 1.1 | 1.1 | use 1.1 | | 1.0 1.1 | 1.0 | 1.1 | 1.0 | 1.0 | use 1.0 | | 1.0 | 1.0 1.1 | 1.0 | 1.0 | 1.1 | use 1.0 | | 1.1 | 1.0 1.1 | 1.1 | 1.1 | 1.1 | use 1.1 | | 1.1 | 1.0 | 1.1 | 1.0 | 1.0 | 应用程序失败 | | 1.0 | 1.1 | 1.0 | — | — | [WSAVERNOTSUPPORTED](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | | 1.0 1.1 | 1.0 1.1 | 1.1 | 1.1 | 1.1 | use 1.1 | | 1.1 2.0 | 1.0 1.1 | 2.0 | 1.1 | 1.1 | use 1.1 | | 2.0 | 1.0 1.1 2.0 | 2.0 | 2.0 | 2.0 | 使用 2.0 | | 2.0 2.2 | 1.0 1.1 2.0 | 2.2 | 2.0 | 2.0 | 使用 2.0 | | 2.2 | 1.0 1.1 2.0 2.1 2.2 | 2.2 | 2.2 | 2.2 | use 2.2 |     应用程序或 DLL 成功调用 **WSAStartup** 后，可以根据需要继续进行其他 Windows 套接字调用。 使用完 Winsock DLL 的服务后，应用程序必须调用 [WSACleanup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsacleanup) ，以允许 Winsock DLL 释放应用程序使用的内部 Winsock 资源。  如果需要多次获取 **WSADATA** 结构信息，应用程序可以多次调用 [WSAStartup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-wsadata) 。 每次调用此类时，应用程序都可以指定 Winsock DLL 支持的任何版本号。  **WSAStartup** 函数通常会导致加载特定于协议的帮助程序 DLL。 因此，不应从应用程序 DLL 中的 DllMain 函数调用 **WSAStartup** 函数。 这可能会导致死锁。 有关详细信息，请参阅 [DLL 主函数](https://learn.microsoft.com/zh-cn/windows/win32/dlls/dllmain)。  每次成功调用 [WSAStartup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsacleanup) 函数时，应用程序都必须调用 **WSACleanup** 函数。 这意味着，例如，如果应用程序调用 **WSAStartup** 三次，则必须调用 **WSACleanup** 三次。 对 **WSACleanup** 的前两个调用除了递减内部计数器外，什么都不做;任务的最终 **WSACleanup** 调用会为任务执行所有必要的资源解除分配。  **注意** 应用程序可以调用 [**WSAGetLastError**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsagetlasterror) 函数来确定其他 Windows 套接字函数的扩展错误代码，就像在 Windows 套接字中一样，即使 **WSAStartup** 函数失败或未调用 **WSAStartup** 函数来正确初始化 Windows 套接字，然后再调用 Windows 套接字函数。 **WSAGetLastError** 函数是 Winsock 2.2 DLL 中唯一可以在 **WSAStartup** 失败时调用的函数之一。    **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。 示例 以下代码片段演示仅支持 2.2 版 Windows 套接字的应用程序如何进行 **WSAStartup** 调用：  C++复制  #define WIN32\_LEAN\_AND\_MEAN  #include <windows.h>  #include <winsock2.h>  #include <ws2tcpip.h>  #include <stdio.h>  // Need to link with Ws2\_32.lib  #pragma comment(lib, "ws2\_32.lib")  int \_\_cdecl main()  {  WORD wVersionRequested;  WSADATA wsaData;  int err;  /\* Use the MAKEWORD(lowbyte, highbyte) macro declared in Windef.h \*/  wVersionRequested = MAKEWORD(2, 2);  err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  /\* Tell the user that we could not find a usable \*/  /\* Winsock DLL. \*/  printf("WSAStartup failed with error: %d\n", err);  return 1;  }  /\* Confirm that the WinSock DLL supports 2.2.\*/  /\* Note that if the DLL supports versions greater \*/  /\* than 2.2 in addition to 2.2, it will still return \*/  /\* 2.2 in wVersion since that is the version we \*/  /\* requested. \*/  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {  /\* Tell the user that we could not find a usable \*/  /\* WinSock DLL. \*/  printf("Could not find a usable version of Winsock.dll\n");  WSACleanup();  return 1;  }  else  printf("The Winsock 2.2 dll was found okay\n");    /\* The Winsock DLL is acceptable. Proceed to use it. \*/  /\* Add network programming using Winsock here \*/  /\* then call WSACleanup when done using the Winsock dll \*/    WSACleanup();  } 要求  |  |  | | --- | --- | | **目标平台** | Windows | | **标头** | winsock.h (包括 Winsock2.h) | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 [MAKEWORD](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/legacy/ms632663(v=vs.85))  [WSACleanup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsacleanup)  [WSAGetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsagetlasterror)  [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)  [send](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-send)  [sendto](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-sendto) |

## MAKEWORD 宏

|  |  |  |
| --- | --- | --- |
| 通过连接指定的值来创建 **WORD** 值。 语法 c++复制  WORD MAKEWORD(   BYTE a,   BYTE b  ); -parameters-param a 新值的低位字节。 -param b 新值的高位字节。 -return 类型：**WORD**  新值。 要求  |  |  | | --- | --- | | Header | minwindef.h（包括 Windows.h） |  另请参阅 [Windows 数据类型](https://learn.microsoft.com/zh-cn/windows/win32/winprog/windows-data-types) |

## WSAAccept 函数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **WSAAccept** 函数根据条件函数的返回值有条件地接受连接，提供服务质量流规范，并允许传输连接数据。 语法 C++复制  SOCKET WSAAPI WSAAccept(  [in] SOCKET s,  [out] sockaddr \*addr,  [in, out] LPINT addrlen,  [in] LPCONDITIONPROC lpfnCondition,  [in] DWORD\_PTR dwCallbackData  ); 参数 [in] s  一个描述符，用于标识在调用 [侦听](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-listen) 函数后侦听连接的套接字。  [out] addr  指向 [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构的可选指针，该结构接收连接实体的地址，称为通信层。 *addr* 参数的确切格式由创建套接字时建立的地址系列确定。  [in, out] addrlen  指向一个整数的可选指针，该整数包含 *addr* 参数所指向的 [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构的长度（以字节为单位）。  [in] lpfnCondition  可选、应用程序指定的条件函数的地址，该函数将根据作为参数传递的调用方信息做出接受/拒绝决策，并选择性地通过将适当的值分配给此函数的结果参数 *g* 来创建或联接套接字组。 如果此参数 **NULL**，则不调用条件函数。  [in] dwCallbackData  作为传递给条件函数的 *dwCallbackData* 参数的值传递回应用程序指定的条件函数的回调数据。 仅当 *lpfnCondition* 参数不 **NULL**时，此参数才适用。 Windows 套接字不解释此参数。 返回值 如果未发生错误，**WSAAccept** 将返回一个类型为 SOCKET 的值，该值是接受套接字的描述符。 否则，将返回INVALID\_SOCKET值，并通过调用 [WSAGetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsagetlasterror)来检索特定的错误代码。  *addrlen* 引用的整数最初包含由 *加法器*指向的空间量。返回时，它将包含返回的地址的实际长度（以字节为单位）。  展开表   |  |  | | --- | --- | | **错误代码** | **意义** | | [**WSAEACCES**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 尝试以禁止其访问权限的方式访问套接字。 如果提供的连接请求已超时或已撤回，则返回此错误。 | | [**WSAECONNREFUSED**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 无法建立连接，因为目标计算机主动拒绝连接。 如果在条件函数的返回值（CF\_REJECT）中指示强制拒绝了连接请求，则返回此错误。 | | [**WSAECONNRESET**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 远程主机强行关闭了现有连接。 指示传入连接返回此错误，但在接受呼叫之前，远程对等方随后终止。 | | [**WSAEFAULT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 系统在尝试在调用中使用指针参数时检测到无效的指针地址。 *addrlen* 参数太小或 *加载项* 或 *lpfnCondition* 不属于用户地址空间，则返回此错误。 | | [**WSAEINTR**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 阻止操作被调用 [WSACancelBlockingCall](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsacancelblockingcall)中断。 如果通过 **WSACancelBlockingCall**取消阻止 Windows 套接字 1.1 调用，则返回此错误。 | | [**WSAEINPROGRESS**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 阻止操作当前正在执行。 如果正在进行阻止的 Windows 套接字 1.1 调用，则返回此错误。 | | [**WSAEINVAL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 提供了无效的参数。 如果在 [WSAAccept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaaccept)之前未调用 [侦听](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-listen)，则返回此错误，条件函数的返回值无效，或者指定套接字处于无效状态的任何情况。 | | [**WSAEMFILE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 打开的套接字太多。 如果在 [WSAAccept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaaccept) 条目时队列为 nonempty，并且没有可用的套接字描述符，则返回此错误。 | | [**WSAENETDOWN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字操作遇到死网。 如果网络子系统失败，则返回此错误。 | | [**WSAENOBUFS**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 无法对套接字执行操作，因为系统缺少足够的缓冲区空间或队列已满。 如果没有可用的缓冲区空间，则返回此错误。 | | [**WSAENOTSOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 尝试对不是套接字的内容执行操作。 如果传入 *s* 参数的套接字描述符不是套接字，则返回此错误。 | | [**WSAEOPNOTSUPP**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 协议系列尚未配置到系统中，或者不存在任何实现。 如果引用的套接字不是支持面向连接的服务的类型，则返回此错误。 | | [**WSAEWOULDBLOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 无法立即完成非阻塞套接字操作。 如果套接字被标记为非阻止且未接受任何连接，则返回此错误。 | | [**WSANOTINITIALIZED**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 应用程序未调用 [WSAStartup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsastartup)，或 **WSAStartup** 失败。 此错误返回对 **WSAStartup** 函数 dit 的成功调用，然后再使用此函数。 | | [**WSATRY\_AGAIN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 这通常是主机名解析期间的临时错误，这意味着本地服务器未收到来自权威服务器的响应。 如果接受连接请求被延迟，则返回此错误，如条件函数的返回值（CF\_DEFER）。 |  言论 **WSAAccept** 函数提取套接字上挂起连接队列中的第一个连接，*s*，并检查它是否与条件函数，前提是指定了条件函数（即，而不是 **NULL**）。 如果条件函数返回CF\_ACCEPT，**WSAAccept** 创建新的套接字。 新创建的套接字的属性与套接字 相同，包括 [WSAAsyncSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsaasyncselect) 注册的异步事件，或 [WSAEventSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaeventselect)。 如果条件函数返回CF\_REJECT，**WSAAccept** 拒绝连接请求。 条件函数与此函数运行在同一线程中，应尽快返回。 如果无法立即做出决策，条件函数应返回CF\_DEFER以指示尚未做出任何决定，并且服务提供商不应对此连接请求采取任何操作。 当应用程序准备好对连接请求执行操作时，它将再次调用 **WSAAccept**，并将CF\_ACCEPT或CF\_REJECT作为条件函数的返回值返回。  当应用程序调用 **WSAAccept** 且队列上没有挂起的连接时，默认模式下的套接字（阻止）将阻止连接。  当应用程序调用 **WSAAccept** 且队列上没有挂起的连接时，非阻止模式（阻止）中的套接字失败，[WSAEWOULDBLOCK](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 错误。 **WSAAccept** 成功并返回新的套接字句柄后，接受的套接字不能用于接受任何其他连接。 原始套接字保持打开状态，并侦听新的连接请求。  *addr* 参数是用连接实体的地址填充的结果参数，称为通信层。 *addr* 参数的确切格式由通信所在的地址系列决定。 *addrlen* 是值结果参数;它最初应包含 *addr*指向的空间量。返回时，它将包含返回的地址的实际长度（以字节为单位）。 此调用用于面向连接的套接字类型，例如SOCK\_STREAM。 如果 *addr* 和/或 *addrlen* 等于 **NULL**，则不会返回有关接受套接字的远程地址的信息。 否则，如果成功接受连接，将填充这两个参数。  条件函数的原型在 Winsock2.h 头文件中定义为 **LPCONDITIONPROC**，如下所示。  C++复制  int CALLBACK  ConditionFunc(  IN LPWSABUF lpCallerId,  IN LPWSABUF lpCallerData,  IN OUT LPQOS lpSQOS,  IN OUT LPQOS lpGQOS,  IN LPWSABUF lpCalleeId,  IN LPWSABUF lpCalleeData,  OUT GROUP FAR \* g,  IN DWORD\_PTR dwCallbackData  );  **ConditionFunc** 是应用程序指定的回调函数的占位符。 实际条件函数必须驻留在 DLL 或应用程序模块中。 它在模块定义文件中导出。  *lpCallerId* 参数指向包含连接实体地址的 WSABUF 结构，其中其 *len* 参数是缓冲区的长度（以字节为单位），其 *buf* 参数是指向缓冲区的指针。 *lpCallerData* 是包含任何用户数据的值参数。 这些参数中的信息随连接请求一起发送。 如果没有调用方标识或调用方数据可用，则相应的参数将 **NULL**。 许多网络协议不支持连接时调用方数据。 大多数传统的网络协议可以在连接请求时支持调用方标识符信息。 *lpCaller* Id 指向的 [WSABUF](https://learn.microsoft.com/zh-cn/windows/desktop/api/ws2def/ns-ws2def-wsabuf) 的 buf 部分指向 [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2)。 **sockaddr** 结构根据其地址系列进行解释（通常通过将 **sockaddr** 转换为特定于地址系列的某种类型）。  *lpSQOS* 参数引用调用方指定的套接字 [FLOWSPEC](https://learn.microsoft.com/zh-cn/windows/desktop/api/qos/ns-qos-flowspec) 结构，每个方向对应一个，后跟任何其他提供程序特定的参数。 对于任何单向套接字，将忽略发送或接收流规范值。 **NULL** 值表示没有调用方提供的服务质量，并且无法进行协商。 非**NULL***lpSQOS* 指针表示要进行服务质量协商，或者提供商准备接受服务质量请求而不进行协商。  *lpGQOS* 参数保留，应 **NULL**。 （保留供将来与套接字组一起使用）引用调用方要创建的套接字组 [FLOWSPEC](https://learn.microsoft.com/zh-cn/windows/desktop/api/qos/ns-qos-flowspec) 结构，每个方向各有一个，后跟任何其他提供程序特定的参数。 *lpGQOS* 的 **NULL** 值表示没有调用方指定的组服务质量。 如果协商发生，则可以返回服务质量信息。  *lpCalleeId* 是包含已连接实体的本地地址的参数。 *lpCallee* Id 指向的 WSABUF *buf* 部分指向 [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构。 **sockaddr** 结构根据其地址系列进行解释（通常通过将 **sockaddr** 转换为特定于地址系列（如结构 **sockaddr\_in**）的某些类型）。  *lpCalleeData* 是条件函数用来将用户数据送回连接实体的结果参数。 *lpCalleeData->len* 最初包含服务提供商分配的缓冲区的长度，并由 *lpCalleeData->buf*指向。 零值表示不支持将用户数据传回调用方。 条件函数应最多将 *lpCalleeData->len* 字节的数据复制到 *lpCalleeData->buf*，然后更新 *lpCalleeData->len*，以指示传输的实际字节数。 如果未将用户数据传回调用方，则条件函数应将 *lpCalleeData->len* 设置为零。 所有地址和用户数据的格式都特定于套接字所属的地址系列。  *g* 参数在条件函数中分配，以指示以下任一操作：   * 如果 *g* 是现有的套接字组标识符，则向此组添加 ，前提是满足此组设置的所有要求。 * 如果 *g* = SG\_UNCONSTRAINED\_GROUP，请创建一个不受约束的套接字组，并将 作为第一个成员。 * 如果 *g* = SG\_CONSTRAINED\_GROUP，请创建一个受约束的套接字组，并将 作为第一个成员。 * 如果 *g* = 零，则不执行组操作。   对于不受约束的组，只要单个服务提供商支持这些套接字集，就可以将其组合在一起。 受约束套接字组只能包含面向连接的套接字，并且要求所有分组套接字上的连接都位于同一主机上的同一地址。 对于新创建的套接字组，可以使用 [getsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockopt) 函数检索新组标识符，并将 *级别* 参数设置为 [SOL\_SOCKET](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sol-socket-socket-options)，*optname* 参数设置为 **SO\_GROUP\_ID**。 在关闭属于此套接字组的最后一个套接字之前，套接字组及其关联的套接字组 ID 保持有效。 套接字组 ID 在给定服务提供商的所有进程中都是唯一的。 在关闭属于此套接字组的最后一个套接字之前，套接字组及其关联的标识符保持有效。 套接字组标识符在给定服务提供商的所有进程中都是唯一的。 有关套接字组的详细信息，请参阅 [WSASocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasocketa) 函数的备注。  传递给条件函数的 dwCallbackData 参数值 是作为原始 WSAAccept 调用中的 dwCallbackData 参数传递的值。 此值仅由 Windows 套接字版本 2 客户端解释。 这样，客户端就可以将 **WSAAccept** 调用站点中的一些上下文信息传递给条件函数。 这也为条件函数提供了确定是否接受连接所需的任何其他信息。 典型的用法是将（适当强制转换）指针传递给包含对此套接字关联的应用程序定义对象的引用的数据结构。  **注意** 为了防止使用 **WSAAccept** 函数免受 SYN 攻击，应用程序在报告连接请求之前必须执行完整的 TCP 握手（SYN-SYNACK-ACK）。 以这种方式防止 SYN 攻击导致SO\_CONDITIONAL\_ACCEPT套接字选项变得无效;条件函数仍在调用，**WSAAccept** 函数正常运行，但依赖客户端无法执行握手的服务器应用程序将无法正常运行。    **注意** 发出阻止的 Winsock 调用（如 **WSAAccept**）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 会执行可警报的等待，这可以通过在同一线程上计划的异步过程调用（APC）中断。 在 APC 中发出另一个阻止 Winsock 调用，中断同一线程上的持续阻止 Winsock 调用将导致未定义的行为，并且永远不会由 Winsock 客户端尝试。   示例代码 以下示例演示如何使用 **WSAAccept** 函数。  C++复制  #include <winsock2.h>  #include <stdio.h>  #include <windows.h>  /\* Define an example conditional function that depends on the pQos field \*/  int CALLBACK ConditionAcceptFunc(  LPWSABUF lpCallerId,  LPWSABUF lpCallerData,  LPQOS pQos,  LPQOS lpGQOS,  LPWSABUF lpCalleeId,  LPWSABUF lpCalleeData,  GROUP FAR \* g,  DWORD\_PTR dwCallbackData  )  {  if (pQos != NULL) {  RtlZeroMemory(pQos, sizeof(QOS));  return CF\_ACCEPT;  } else  return CF\_REJECT;  }  int main() {  /\* Declare and initialize variables \*/  WSADATA wsaData;  SOCKET ListenSocket, AcceptSocket;  struct sockaddr\_in saClient;  int iClientSize = sizeof(saClient);  u\_short port = 27015;  char\* ip;  sockaddr\_in service;  int error;  /\* Initialize Winsock \*/  error = WSAStartup(MAKEWORD(2,2), &wsaData);  if (error) {  printf("WSAStartup() failed with error: %d\n", error);  return 1;  }  /\* Create a TCP listening socket \*/  ListenSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (ListenSocket == INVALID\_SOCKET) {  printf("socket() failed with error: %d\n", WSAGetLastError() );  WSACleanup();  return 1;  }  /\*-----------------------------------------  \* Set up the sock addr structure that the listening socket  \* will be bound to. In this case, the structure holds the  \* local IP address and the port specified. \*/  service.sin\_family = AF\_INET;  service.sin\_port = htons(port);  hostent\* thisHost;  thisHost = gethostbyname("");  ip = inet\_ntoa (\*(struct in\_addr \*)\*thisHost->h\_addr\_list);  service.sin\_addr.s\_addr = inet\_addr(ip);  /\*-----------------------------------------  \* Bind the listening socket to the IP address.  \* and port number specified by the sockaddr structure. \*/  error = bind(ListenSocket, (SOCKADDR \*) &service, sizeof(SOCKADDR));  if (error == SOCKET\_ERROR) {  printf("bind() failed with error: %d\n", WSAGetLastError() );  closesocket(ListenSocket);  WSACleanup();  return 1;  }    /\* Make the socket listen for incoming connection requests \*/  error = listen(ListenSocket, 1);  if (error == SOCKET\_ERROR) {  printf("listen() failed with error: %d\n", WSAGetLastError() );  closesocket(ListenSocket);  WSACleanup();  return 1;  }  printf("Listening...\n");    /\*-----------------------------------------  \* Accept an incoming connection request on the  \* listening socket and transfer control to the  \* accepting socket. \*/  AcceptSocket = WSAAccept(ListenSocket, (SOCKADDR\*) &saClient, &iClientSize,  &ConditionAcceptFunc, NULL);    /\* Now do some work with the AcceptSocket  \* At this point, the application could  \* handle data transfer on the socket, or other socket  \* functionality.\*/    /\* Then clean up and quit \*/  closesocket(AcceptSocket);  closesocket(ListenSocket);  WSACleanup();  return 0;  }   |  |  | | --- | --- | | **标头** | winsock2.h | | **库** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |   [WSAAsyncSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsaasyncselect)  [WSAConnect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaconnect)  [WSASocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasocketa)  [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)  [接受](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept)  [绑定](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind)  [连接](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect)  [getsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockopt)  [侦听](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-listen)  [选择](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-select)  [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2)  [套接字](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-socket) |

## WSACleanup 函数语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **WSACleanup** 函数终止使用 Winsock 2 DLL (*Ws2\_32.dll*) 。 语法 C++复制  int WSACleanup(); 返回值 如果操作成功，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) 检索特定的错误号。  在多线程环境中， **WSACleanup** 终止所有线程的 Windows 套接字操作。  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。 | | [**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。 |  注解 应用程序或 DLL 需要执行成功的 [WSAStartup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsastartup) 调用，然后才能使用 Windows 套接字服务。 完成 Windows 套接字的使用后，应用程序或 DLL 必须调用 **WSACleanup** 以从 Windows 套接字实现中注销自身，并允许实现释放代表应用程序或 DLL 分配的任何资源。  调用 **WSACleanup** 时，将取消此进程中任何线程发出的任何挂起的阻止或异步 Windows 套接字调用，而不会发布任何通知消息或发出任何事件对象的信号。 ([WSASend](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasend)、[WSASendTo、WSARecv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasendto) 或 [WSARecvFrom](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsarecvfrom) 的任何挂起的重叠发送或接收操作（例如，此进程中任何线程发出的) ）也会被取消，而无需设置事件对象或调用完成例程（如果指定了一个）。 在这种情况下，挂起的重叠操作会失败，WSA\_OPERATION\_ABORTED**错误状态。**  调用 **WSACleanup** 时打开的套接字将重置并自动解除分配，就像调用 [closesocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-closesocket) 一样。 调用 **WSACleanup** 时，已使用 **closesocket** 关闭但仍具有待发送数据的套接字可能会受到影响。 在这种情况下，如果在应用程序退出时从内存中卸载 WS2\_32.DLL，则挂起的数据可能会丢失。 为了确保发送所有挂起的数据，应用程序应使用 [关闭](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-shutdown) 来关闭连接，然后等待关闭完成，然后再调用 **closesocket** 和 **WSACleanup**。 必须解除分配所有资源和内部状态（如排队未发布或已发布的消息），以便可供下一个用户使用。  每次成功调用 **WSAStartup** 时，都必须调用 [WSACleanup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsastartup)。 只有最终 **的 WSACleanup** 函数调用才会执行实际清理。 前面的调用只是递减 WS2\_32.DLL 中的内部引用计数。  **注意WSACleanup** 不会取消注册对等名称 (对等名称，例如) 可能已注册到 Windows 套接字命名空间提供程序，例如对等名称解析协议 (PNRP) 命名空间提供程序。    在 Windows 套接字 1.1 中，尝试从阻塞挂钩中调用 **WSACleanup**，但未能检查返回代码是常见的编程错误。 如果 Winsock 1.1 应用程序需要在阻止调用未完成时退出，则应用程序必须先使用 [WSACancelBlockingCall](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsacancelblockingcall) 取消阻止调用，然后在将控制权返回到应用程序后发出 **WSACleanup** 调用。 在 Windows 套接字 2 中，此问题不存在， **WSACancelBlockingCall** 函数已删除。  **WSACleanup** 函数通常会导致卸载特定于协议的帮助程序 DLL。 因此，不应从应用程序 DLL 中的 DllMain 函数调用 **WSACleanup** 函数。 这可能会导致死锁。 有关详细信息，请参阅 [DLL 主函数](https://learn.microsoft.com/zh-cn/windows/win32/dlls/dllmain)。  **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 [PNRP 命名空间提供程序 API](https://learn.microsoft.com/zh-cn/windows/desktop/P2PSdk/pnrp-namespace-provider-api)  [WSAStartup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsastartup)  [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)  [closesocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-closesocket)  [shutdown](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-shutdown) |

## accept函数的语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **accept** 函数允许在套接字上进行传入连接尝试。 语法 C++复制  SOCKET WSAAPI accept(  [in] SOCKET s,  [out] sockaddr \*addr,  [in, out] int \*addrlen  ); 参数 [in] s  一个描述符，用于标识已使用 [侦](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-listen) 听函数置于侦听状态的套接字。 连接实际上是使用 **accept** 返回的套接字建立的。  [out] addr  指向接收连接实体地址的缓冲区的可选指针，该地址称为通信层。 *addr* 参数的确切格式由创建 [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构中的套接字时建立的地址系列确定。  [in, out] addrlen  指向包含 *addr* 参数指向的结构长度的整数的可选指针。 返回值 如果未发生错误， **则 accept** 将返回 **类型为 SOCKET** 的值，该值是新套接字的描述符。 此返回值是建立实际连接的套接字的句柄。  否则，将返回 **值 INVALID\_SOCKET** ，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。  *addrlen* 引用的整数最初包含 *addr* 指向的空间量。返回时，它将包含返回的地址的实际长度（以字节为单位）。  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) 。 | | [**WSAECONNRESET**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 指示了传入连接，但随后在接受呼叫之前被远程对等方终止。 | | [**WSAEFAULT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | *addrlen* 参数太小或 *addr* 不是用户地址空间的有效部分。 | | [**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。 | | [**WSAEINVAL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 在[接受](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept)之前未调用 [listen](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-listen) 函数。 | | [**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。 | | [**WSAEMFILE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 队列在 [输入接受时](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept) 为非空，并且没有可用的描述符。 | | [**WSAENETDOWN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 网络子系统失败。 | | [**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。 | | [**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) | ：引用的套接字不是支持面向连接的服务的类型。 | | [**WSAEWOULDBLOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | ：套接字标记为非阻止，没有要接受的连接。 |  注解 **accept** 函数提取套接字 上挂起连接队列中的第一*个*连接。 然后，它会创建并返回新套接字的句柄。 新创建的套接字是将处理实际连接的套接字;它具有 *与套接字相同的*属性，包括向 [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) 函数注册的异步事件。  如果队列中没有挂起的连接，并且套接字标记为阻塞，则 **accept** 函数可以阻止调用方，直到存在连接。 如果套接字标记为非阻止，并且队列中没有挂起的连接， **则接受** 将返回如下所述的错误。 成功完成 **accept** 返回新的套接字句柄后，接受的套接字不能用于接受更多连接。 原始套接字保持打开状态，并侦听新的连接请求。  参数 *添加器* 是一个结果参数，它用连接实体的地址填充，如通信层所称。 *addr* 参数的确切格式由发生通信的地址系列确定。 *addrlen* 是 value-result 参数;它最初应包含 *addr* 指向的空间量;返回时，它将包含返回的地址的实际长度 () 字节。  **accept** 函数与面向连接的套接字类型（如 SOCK\_STREAM）一起使用。 如果 *addr* 和/或 *addrlen* 等于 **NULL**，则不会返回有关接受套接字的远程地址的信息。  **注意** 发出阻止 Winsock 调用（如 **accept**）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 **accept** 函数。  C++复制  #ifndef UNICODE  #define UNICODE  #endif  #include <winsock2.h>  #include <WS2tcpip.h>  #include <stdio.h>  #include <windows.h>  // Need to link with Ws2\_32.lib  #pragma comment(lib, "Ws2\_32.lib")  int wmain(void)  {  //----------------------  // Initialize Winsock.  WSADATA wsaData;  int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);  if (iResult != NO\_ERROR) {  wprintf(L"WSAStartup failed with error: %ld\n", iResult);  return 1;  }  //----------------------  // Create a SOCKET for listening for  // incoming connection requests.  SOCKET ListenSocket;  ListenSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (ListenSocket == INVALID\_SOCKET) {  wprintf(L"socket failed with error: %ld\n", WSAGetLastError());  WSACleanup();  return 1;  }  //----------------------  // The sockaddr\_in structure specifies the address family,  // IP address, and port for the socket that is being bound.  sockaddr\_in service;  service.sin\_family = AF\_INET;  service.sin\_port = htons(27015);  inet\_pton(AF\_INET, "127.0.0.1", &service.sin\_addr);  if (bind(ListenSocket,  (SOCKADDR \*) & service, sizeof (service)) == SOCKET\_ERROR) {  wprintf(L"bind failed with error: %ld\n", WSAGetLastError());  closesocket(ListenSocket);  WSACleanup();  return 1;  }  //----------------------  // Listen for incoming connection requests.  // on the created socket  if (listen(ListenSocket, 1) == SOCKET\_ERROR) {  wprintf(L"listen failed with error: %ld\n", WSAGetLastError());  closesocket(ListenSocket);  WSACleanup();  return 1;  }  //----------------------  // Create a SOCKET for accepting incoming requests.  SOCKET AcceptSocket;  wprintf(L"Waiting for client to connect...\n");  //----------------------  // Accept the connection.  AcceptSocket = accept(ListenSocket, NULL, NULL);  if (AcceptSocket == INVALID\_SOCKET) {  wprintf(L"accept failed with error: %ld\n", WSAGetLastError());  closesocket(ListenSocket);  WSACleanup();  return 1;  } else  wprintf(L"Client connected.\n");  // No longer need server socket  closesocket(ListenSocket);  WSACleanup();  return 0;  }  有关使用 **accept** 函数的另一个示例，请参阅 [入门 With Winsock](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/getting-started-with-winsock)。 ATM 说明 以下是与连接设置相关的重要问题，在将异步传输模式 (ATM) 与 Windows 套接字 2 配合使用时，必须考虑这些问题：   * **accept** 和 [WSAAccept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaaccept) 函数不一定设置远程地址和地址长度参数。 因此，使用 ATM 时，调用方应使用 **WSAAccept** 函数并将ATM\_CALLING\_PARTY\_NUMBER\_IE放在 [QoS](https://learn.microsoft.com/zh-cn/windows/win32/api/winsock2/ns-winsock2-qos) 结构的 **ProviderSpecific** 成员中，该成员本身包含在根据 **WSAAccept** 使用的回调函数的 *lpSQOS* 参数中。 * 使用 **accept** 函数时，请注意，在连接建立已遍历发送方和接收方之间的整个距离之前，函数可能会返回 。 这是因为 **accept** 函数在收到 CONNECT ACK 消息后立即返回 ;在 ATM 中，一旦将 CONNECT 消息处理 (，路径中的下一个开关就会返回 CONNECT ACK 消息，而不是最终) 建立连接的终结点发送 CONNECT ACK。 因此，应用程序应意识到，如果在收到 CONNECT ACK 消息后立即发送数据，则可能会丢失数据，因为可能尚未在发送方和接收方之间建立连接。   **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** | winsock2.h | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 [WSAAccept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaaccept)  [WSAAsyncSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsaasyncselect)  [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)  [bind](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind)  [connect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect)  [listen](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-listen)  [select](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-select)  [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) |

## bind 函数语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **bind** 函数将本地地址与套接字相关联。 语法 C++复制  int bind(  [in] SOCKET s,  const sockaddr \*addr,  [in] int namelen  ); 参数 [in] s  标识未绑定套接字的描述符。  addr  指向要分配给绑定套接字 的本地地址 [的 sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构的指针。  [in] namelen  addr 指向的值的长度（以字节为单位）。 返回值 如果未发生错误， **绑定** 将返回零。 否则，它将返回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** | | [**WSANOTINITIALIZED**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | **注意** 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) | 试图以套接字访问权限所禁止的方式访问套接字。  如果 nn 尝试将数据报套接字绑定到广播地址失败，因为 [未启用 setsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt) 选项SO\_BROADCAST，则返回此错误。 | | [**WSAEADDRINUSE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 通常每个套接字地址只允许使用一次（协议/网络地址/端口）。  如果计算机上的进程已绑定到相同的完全限定地址，并且套接字尚未标记为允许使用 SO\_REUSEADDR 重用地址，则返回此错误。 例如， *name* 参数中指定的 IP 地址和端口已绑定到另一个应用程序正在使用的另一个套接字。 有关详细信息，请参阅SOL\_SOCKET套接字选项参考中的 [SO\_REUSEADDR套接字选项](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sol-socket-socket-options) 、 [使用SO\_REUSEADDR和SO\_EXCLUSIVEADDRUSE](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/using-so-reuseaddr-and-so-exclusiveaddruse)和 [SO\_EXCLUSIVEADDRUSE](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/so-exclusiveaddruse)。 | | [**WSAEADDRNOTAVAIL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 请求的地址在其上下文中无效。  如果 *name* 参数指向的指定地址不是此计算机上的有效本地 IP 地址，则返回此错误。 | | [**WSAEFAULT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 系统尝试在调用中使用指针参数时检测到指针地址无效。  如果 *name* 参数为 *NULL，name* 或 *namelen* 参数不是用户地址空间的有效部分， *namelen* 参数太小， *name* 参数包含关联的地址系列地址格式不正确，或者 *name* 指定的内存块的前两个字节与与套接字描述符关联的地址系列 *不匹配，则*返回此错误。 | | [**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。 | | [**WSAEINVAL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 提供的参数无效。  如果 *套接字 已* 绑定到地址，则返回此错误。 | | [**WSAENOBUFS**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 通常， **WSAENOBUFS** 指示没有足够的临时端口来为绑定分配。 | | [**WSAENOTSOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 某个操作尝试对非套接字执行操作。  如果 *s* 参数中的描述符不是套接字，则返回此错误。 |  注解 在未连接的套接字上需要 **绑定** 函数，然后才能对 [侦听](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-listen) 函数进行后续调用。 它通常用于绑定到面向连接的 (流) 或无连接 (数据报) 套接字。 **绑定**函数还可用于绑定到原始套接字， (套接字是通过调用*类型*参数设置为 SOCK\_RAW) 的[套接字](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-socket)创建的。 在后续调用 [connect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect)、[ConnectEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/mswsock/nc-mswsock-lpfn_connectex)、[WSAConnect、WSAConnectByList](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaconnect) 或 [WSAConnectByName](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaconnectbylist) 函数之前，还可以在未连接的套接字上使用**绑定**函数，然后再发送操作。  使用调用套接字函数创建 [套接字](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-socket) 时，它存在于命名空间 (地址系列) 中，但它没有为其分配名称。 使用 **bind** 函数通过将本地名称分配给未命名的套接字来建立套接字的本地关联。  使用 Internet 地址系列时，名称由三个部分组成：   * 地址系列。 * 主机地址。 * 标识应用程序的端口号。   在 Windows 套接字 2 中， *名称* 参数不会严格解释为指向 [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构的指针。 它以这种方式强制转换，以便与 Windows 套接字 1.1 兼容。 服务提供商可以随意将其视为 *指向名称大小*内存块的指针。 此块中的前 2 个字节 (对应于 **sockaddr** 结构的**sa\_family**成员、**sockaddr\_in** 结构的**sin\_family**成员或**sockaddr\_in6**结构的**sin6\_family成员)**必须包含用于创建套接字的地址系列。 否则，会发生错误 WSAEFAULT。  如果应用程序不关心分配的本地地址，请在 *name* 参数**的 sa\_data** 成员中指定 IPv4 本地地址的常量值**INADDR\_ANY**或 IPv6 本地地址的常量值**in6addr\_any**。 这样，基础服务提供商就可以使用任何适当的网络地址，在 *多宿主* 主机 (即具有多个网络接口和地址) 的主机的情况下，可能会简化应用程序编程。  对于 TCP/IP，如果端口指定为零，则服务提供商会从动态客户端端口范围向应用程序分配唯一端口。 在 Windows Vista 及更高版本中，动态客户端端口范围是介于 49152 和 65535 之间的值。 这与 Windows Server 2003 及更早版本不同，其中动态客户端端口范围是介于 1025 和 5000 之间的值。 可以通过在以下注册表项下设置值来更改客户端动态端口范围的最大值：  **HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters**  **MaxUserPort** 注册表值设置要用于动态客户端端口范围最大值的值。 必须重启计算机才能使此设置生效。  在 Windows Vista 及更高版本上，可以使用 **netsh** 命令查看和更改动态客户端端口范围。 动态客户端端口范围可以针对 UDP 和 TCP 以及 IPv4 和 IPv6 进行不同的设置。 有关详细信息，请参阅 [知识库929851](https://support.microsoft.com/kb/929851)。  应用程序可以在调用 **bind** 后使用 [getsockname](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockname) 来了解已分配给套接字的地址和端口。 如果 Internet 地址等于 **INADDR\_ANY** 或 **in6addr\_any**，则在连接套接字之前 **，getsockname** 不一定提供地址，因为如果主机是多宿主的，则多个地址可能有效。 对于客户端应用程序，不建议绑定到除端口 0 以外的特定端口号，因为存在与本地计算机上已使用该端口号的另一个套接字发生冲突的危险。  **注意** 将 **绑定** 与 SO\_EXCLUSIVEADDRUSE 或 SO\_REUSEADDR 套接字选项一起使用时，必须在执行 **绑定** 之前设置套接字选项，才能产生任何影响。 有关详细信息，请参阅 [**SO\_EXCLUSIVEADDRUSE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/so-exclusiveaddruse) 和使用 [**SO\_REUSEADDR和SO\_EXCLUSIVEADDRUSE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/using-so-reuseaddr-and-so-exclusiveaddruse)。    对于多播操作，首选方法是调用 **绑定** 函数以将套接字与本地 IP 地址相关联，然后加入多播组。 尽管此操作顺序不是必需的，但强烈建议这样做。 因此，多播应用程序将首先在本地计算机上选择 IPv4 或 IPv6 地址、通配符 IPv4 地址 (**INADDR\_ANY)**，或者通配符 IPv6 地址 (**in6addr\_any**) 。 然后，多播应用程序将使用 *name* 参数的 **sa\_data** 成员中的 此地址调用**绑定**函数，以将本地 IP 地址与套接字相关联。 如果指定了通配符地址，则 Windows 将选择要使用的本地 IP 地址。 **绑定**函数完成后，应用程序将加入感兴趣的多播组。 有关如何加入多播组的详细信息，请参阅多 [播编程](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/multicast-programming)部分。 然后，可以使用此套接字通过 recv、[recvfrom](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recvfrom)、[WSARecv、WSARecvEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsarecv)、[WSARecvFrom](https://learn.microsoft.com/zh-cn/windows/desktop/api/mswsock/nf-mswsock-wsarecvex) 或 [LPFN\_WSARECVMSG (WSARecvMsg)](https://learn.microsoft.com/zh-cn/windows/win32/api/mswsock/nc-mswsock-lpfn_wsarecvmsg)函数从多播组接收多播数据包。  将操作发送到多播组通常不需要 **绑定** 函数。 如果套接字尚未绑定， [sendto](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-sendto)、[WSASendMsg](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasendmsg) 和 [WSASendTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasendto) 函数会将套接字隐式绑定到通配符地址。 在使用[发送或](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-send)[WSASend](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasend) 函数之前，需要**绑定**函数，这些函数不执行隐式绑定，并且仅在连接的套接字上允许，这意味着必须已绑定套接字才能连接。 如果应用程序想要在具有多个网络接口和本地 IP 地址的本地计算机上选择特定的本地 IP 地址，则可以在使用 **sendto**、**WSASendMsg** 或 **WSASendTo** 函数的发送操作之前使用**绑定**函数。 否则，使用 **sendto**、**WSASendMsg** 或 **WSASendTo** 函数隐式绑定到通配符地址可能会导致不同的本地 IP 地址用于发送操作。  **注意** 发出阻止的 Winsock 调用（如 **绑定**）时，Winsock 可能需要等待网络事件，然后才能完成调用。 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。   IrDA 套接字说明  * 必须显式包含 Af\_irda.h 头文件。 * 本地名称不会在 IrDA 中公开。 因此，IrDA 客户端套接字不得在 [connect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect) 函数之前调用**绑定**函数。 如果 IrDA 套接字以前使用 **绑定**绑定到服务名称， **则 connect** 函数将失败并SOCKET\_ERROR。 * 如果服务名称的格式为“LSAP-SELxxx”，其中 xxx 是范围 1-127 中的十进制整数，则地址指示特定的 LSAP-SEL xxx，而不是服务名称。 此类服务名称允许服务器应用程序接受定向到特定 LSAP-SEL 的传入连接，而无需先执行 ISA 服务名称查询来获取关联的 LSAP-SEL。 此服务名称类型的一个示例是不支持 IAS 的非 Windows 设备。   **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。 示例 以下示例演示如何使用 **bind** 函数。 有关使用 **bind** 函数的另一个示例，请参阅[使用 Winsock 入门](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/getting-started-with-winsock)。  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()  {  // Declare some variables  WSADATA wsaData;  int iResult = 0; // used to return function results  // the listening socket to be created  SOCKET ListenSocket = INVALID\_SOCKET;  // The socket address to be passed to bind  sockaddr\_in service;  //----------------------  // Initialize Winsock  iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);  if (iResult != NO\_ERROR) {  wprintf(L"Error at WSAStartup()\n");  return 1;  }  //----------------------  // Create a SOCKET for listening for  // incoming connection requests  ListenSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (ListenSocket == INVALID\_SOCKET) {  wprintf(L"socket function failed with error: %u\n", WSAGetLastError());  WSACleanup();  return 1;  }  //----------------------  // The sockaddr\_in structure specifies the address family,  // IP address, and port for the socket that is being bound.  service.sin\_family = AF\_INET;  service.sin\_addr.s\_addr = inet\_addr("127.0.0.1");  service.sin\_port = htons(27015);  //----------------------  // Bind the socket.  iResult = bind(ListenSocket, (SOCKADDR \*) &service, sizeof (service));  if (iResult == SOCKET\_ERROR) {  wprintf(L"bind failed with error %u\n", WSAGetLastError());  closesocket(ListenSocket);  WSACleanup();  return 1;  }  else  wprintf(L"bind returned success\n");  WSACleanup();  return 0;  } 要求  |  |  | | --- | --- | | **标头** | winsock.h (包括 Winsock2.h) | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 [多播编程](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/multicast-programming)  [SOL\_SOCKET套接字选项](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sol-socket-socket-options)  [SO\_EXCLUSIVEADDRUSE](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/so-exclusiveaddruse)  [TCP/IP 原始套接字](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/tcp-ip-raw-sockets-2)  [使用SO\_REUSEADDR和SO\_EXCLUSIVEADDRUSE](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/using-so-reuseaddr-and-so-exclusiveaddruse)  [WSACancelBlockingCall](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsacancelblockingcall)  [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)  [connect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect)  [getsockname](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockname)  [listen](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-listen)  [setsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt)  [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) |

## listen 函数语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **侦听**函数将套接字置于侦听传入连接的状态。 语法 C++复制  int WSAAPI listen(  [in] SOCKET s,  [in] int backlog  ); 参数 [in] s  标识绑定的未连接的套接字的描述符。  [in] backlog  挂起的连接队列的最大长度。 如果设置为 **SOMAXCONN**，则负责套接字 *的基础* 服务提供商会将积压工作设置为最大合理值。 如果设置为 **SOMAXCONN\_HINT (N)**(其中 N 为数字) ，则积压工作值为 N，调整为在 200、65535) (范围内。 请注意， **SOMAXCONN\_HINT** 可用于将积压工作设置为比 SOMAXCONN 更大的值。  **SOMAXCONN\_HINT** 仅受 Microsoft TCP/IP 服务提供商支持。 没有用于获取实际积压工作值的标准预配。 返回值 如果未发生错误， **则侦听** 返回零。 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** | | [**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。 | | [**WSAEADDRINUSE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字的本地地址已在使用中，并且未标记为允许使用 SO\_REUSEADDR 重用地址。 此错误通常在 [绑定](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind) 函数执行期间发生，但如果 **绑定到** 涉及ADDR\_ANY) 的部分通配符地址 (，并且需要在此函数时提交特定地址，则可能会延迟到此函数。 | | [**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。 | | [**WSAEINVAL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字[尚未绑定。](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind) | | [**WSAEISCONN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | ：套接字已连接。 | | [**WSAEMFILE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 无法提供更多套接字描述符。 | | [**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。 | | [**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) | 引用的套接字不是支持 [侦听](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-listen) 操作的类型。 |  注解 若要接受连接，首先使用套接字函数创建 [套接字](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-socket) ，并使用 [绑定](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind) 函数绑定到本地地址。 使用 **listen** 指定传入连接的积压工作，然后使用 [accept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept) 函数接受连接。 面向连接的套接字（例如 **SOCK\_STREAM** 类型的套接字）用于 **侦听**。 *套接字将*置于被动模式，其中传入的连接请求被进程确认并排队等待接受。  **SOMAXCONN***积压*工作的值是一个特殊常量，指示负责套接字 *的基础*服务提供商将挂起连接队列的长度设置为最大合理值。  在 Windows 套接字 2 上，此最大值默认为一个大值， (通常为数百个或更多) 。  在蓝牙应用程序中调用 **侦听** 函数时，强烈建议对 *积压工作* 参数使用低得多的值， (通常为 2 到 4) ，因为只接受少数客户端连接。 这会减少分配供侦听套接字使用的系统资源。 这一相同建议适用于其他只需要几个客户端连接的网络应用程序。  **侦听**函数通常由一次可以有多个连接请求的服务器使用。 如果连接请求到达且队列已满，客户端将收到指示 [WSAECONNREFUSED](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 的错误。  如果没有可用的套接字描述符， **则侦听** 会尝试继续正常运行。 如果描述符可用，则稍后对 **listen** 或 [accept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept) 的调用会将队列重新填充为 *积压工作* 参数指定的当前或最新值（如果可能），并恢复侦听传入连接。  如果在已侦听套接字上调用 **listen** 函数，它将返回成功，而不会更改 *积压工作* 参数的值。 在侦**听**套接字的后续调用中，将*积压工作*参数设置为 0 不被视为正确重置，尤其是在套接字上存在连接时。  **注意** 发出阻止的 Winsock 调用（例如 **侦听**）时，Winsock 可能需要等待网络事件，然后才能完成调用。 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 **listen** 函数。  C++复制  #ifndef UNICODE  #define UNICODE  #endif  #define WIN32\_LEAN\_AND\_MEAN  #include <winsock2.h>  #include <ws2tcpip.h>  #include <stdio.h>  // Need to link with Ws2\_32.lib  #pragma comment(lib, "ws2\_32.lib")  int wmain()  {  //----------------------  // Initialize Winsock  WSADATA wsaData;  int iResult = 0;  SOCKET ListenSocket = INVALID\_SOCKET;  sockaddr\_in service;  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 listening for incoming connection requests.  ListenSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (ListenSocket == INVALID\_SOCKET) {  wprintf(L"socket function failed with error: %ld\n", WSAGetLastError());  WSACleanup();  return 1;  }  //----------------------  // The sockaddr\_in structure specifies the address family,  // IP address, and port for the socket that is being bound.  service.sin\_family = AF\_INET;  service.sin\_addr.s\_addr = inet\_addr("127.0.0.1");  service.sin\_port = htons(27015);  iResult = bind(ListenSocket, (SOCKADDR \*) & service, sizeof (service));  if (iResult == SOCKET\_ERROR) {  wprintf(L"bind function failed with error %d\n", WSAGetLastError());  iResult = closesocket(ListenSocket);  if (iResult == SOCKET\_ERROR)  wprintf(L"closesocket function failed with error %d\n", WSAGetLastError());  WSACleanup();  return 1;  }  //----------------------  // Listen for incoming connection requests  // on the created socket  if (listen(ListenSocket, SOMAXCONN) == SOCKET\_ERROR)  wprintf(L"listen function failed with error: %d\n", WSAGetLastError());  wprintf(L"Listening on socket...\n");  iResult = closesocket(ListenSocket);  if (iResult == SOCKET\_ERROR) {  wprintf(L"closesocket function failed with error %d\n", WSAGetLastError());  WSACleanup();  return 1;  }  WSACleanup();  return 0;  } 示例代码 有关使用 **listen** 函数的另一个示例，请参阅 [入门 With Winsock](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/getting-started-with-winsock)。 IrDA 套接字说明  * 必须显式包含 Af\_irda.h 头文件。  兼容性 *积压工作*参数限制 (无提示) 基础服务提供商确定的合理值。 非法值将替换为最接近的法定值。 没有标准预配来找出实际的积压工作值。  **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** | winsock2.h | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 [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)  [accept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept)  [connect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect)  [socket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-socket) |

## socket 函数语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **套接字**函数创建绑定到特定传输服务提供程序的套接字。 语法 C++复制  SOCKET WSAAPI socket(  [in] int af,  [in] int type,  [in] int protocol  ); 参数 [in] af  地址系列规范。 地址系列的可能值在 *Winsock2.h* 头文件中定义。  在为 Windows Vista 及更高版本发布的Windows SDK中，头文件的组织方式已更改，地址系列的可能值在 *Ws2def.h* 头文件中定义。 请注意， *Ws2def.h* 头文件会自动包含在 *Winsock2.h 中*，永远不应直接使用。  当前支持的值是 AF\_INET 或 AF\_INET6，它们是 IPv4 和 IPv6 的 Internet 地址系列格式。 用于 NetBIOS 的地址系列 (AF\_NETBIOS 的其他选项，例如，如果安装了地址系列的 Windows 套接字服务提供商，则支持) 。 请注意，AF\_地址系列和PF\_协议系列常量的值 (相同，例如 **，AF\_INET** 和 **PF\_INET**) ，因此可以使用任一常量。  下表列出了地址系列的常见值，尽管许多其他值是可能的。  展开表   |  |  | | --- | --- | | **Af** | **含义** | | **AF\_UNSPEC**  0 | 地址系列未指定。 | | **AF\_INET**  2 | Internet 协议版本 4 (IPv4) 地址系列。 | | **AF\_IPX**  6 | IPX/SPX 地址系列。 仅当安装了 NWLink IPX/SPX NetBIOS 兼容传输协议时，才支持此地址系列。  Windows Vista 及更高版本不支持此地址系列。 | | **AF\_APPLETALK**  16 | AppleTalk 地址系列。 仅当安装了 AppleTalk 协议时，才支持此地址系列。  Windows Vista 及更高版本不支持此地址系列。 | | **AF\_NETBIOS**  17 | NetBIOS 地址系列。 仅当安装了适用于 NetBIOS 的 Windows 套接字提供程序时，才支持此地址系列。  32 位版本的 Windows 支持 NetBIOS 的 Windows 套接字提供程序。 默认情况下，此提供程序安装在 32 位版本的 Windows 上。  64 位版本的 Windows 不支持 NetBIOS 的 Windows 套接字提供程序，包括 Windows 7、Windows Server 2008、Windows Vista、Windows Server 2003 或 Windows XP。  适用于 NetBIOS 的 Windows 套接字提供程序仅支持 *类型* 参数设置为 **SOCK\_DGRAM**的套接字。  适用于 NetBIOS 的 Windows 套接字提供程序与 [NetBIOS](https://learn.microsoft.com/zh-cn/previous-versions/windows/desktop/netbios/portal) 编程接口不直接相关。 Windows Vista、Windows Server 2008 及更高版本不支持 NetBIOS 编程接口。 | | **AF\_INET6**  23 | Internet 协议版本 6 (IPv6) 地址系列。 | | **AF\_IRDA**  26 | IrDA (IrDA) 地址系列。  仅当计算机安装了红外端口和驱动程序时，才支持此地址系列。 | | **AF\_BTH**  32 | 蓝牙地址系列。  如果计算机安装了蓝牙适配器和驱动程序，则 SP2 或更高版本的 Windows XP 支持此地址系列。 |   [in] type  新套接字的类型规范。  套接字类型的可能值在 *Winsock2.h* 头文件中定义。  下表列出了 Windows 套接字 2 支持 *的类型* 参数的可能值：  展开表   |  |  | | --- | --- | | **类型** | **含义** | | **SOCK\_STREAM**  1 | 一种套接字类型，它通过 OOB 数据传输机制提供排序的、可靠的双向、基于连接的字节流。 此套接字类型对 Internet 地址系列 (AF\_INET 或AF\_INET6) 使用传输控制协议 (TCP) 。 | | **SOCK\_DGRAM**  2 | 支持数据报的套接字类型，这些数据报是固定 (通常较小) 最大长度的无连接、不可靠的缓冲区。 此套接字类型对 Internet 地址系列 (AF\_INET 或AF\_INET6) 使用用户数据报协议 (UDP) 。 | | **SOCK\_RAW**  3 | 一种套接字类型，它提供允许应用程序操作下一层协议标头的原始套接字。 若要操作 IPv4 标头，必须在套接字上设置 [IP\_HDRINCL](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/ipproto-ip-socket-options) 套接字选项。 若要操作 IPv6 标头，必须在套接字上设置 [IPV6\_HDRINCL](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/ipproto-ipv6-socket-options) 套接字选项。 | | **SOCK\_RDM**  4 | 提供可靠消息数据报的套接字类型。 此类型的一个示例是 Windows 中的实用常规多播 (PGM) 多播协议实现，通常称为 [可靠多播编程](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/reliable-multicast-programming--pgm-)。  仅当安装了可靠多播协议时，才支持此 *类型* 值。 | | **SOCK\_SEQPACKET**  5 | 提供基于数据报的伪流数据包的套接字类型。 |     在 Windows 套接字 2 中，引入了新的套接字类型。 应用程序可以通过 [WSAEnumProtocols](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaenumprotocolsa) 函数动态发现每个可用传输协议的属性。 因此，应用程序可以确定地址系列的可能的套接字类型和协议选项，并在指定此参数时使用此信息。 *Winsock2.h* 和 *Ws2def.h* 头文件中的套接字类型定义将随着新的套接字类型、地址系列和协议的定义而定期更新。  在 Windows 套接字 1.1 中，唯一可能的套接字类型是 **SOCK\_DGRAM** 和 **SOCK\_STREAM**。  [in] protocol  要使用的协议。 *协议*参数的可能选项特定于指定的地址系列和套接字类型。 *协议*的可能值在 *Winsock2.h* 和 *Wsrm.h* 头文件中定义。  在 Windows Vista 及更高版本发布的Windows SDK中，头文件的组织方式已更改，此参数可以是 *Ws2def.h* 头文件中定义的 **IPPROTO** 枚举类型中的值之一。 请注意， *Ws2def.h* 头文件会自动包含在 *Winsock2.h 中*，永远不应直接使用。  如果指定值 0，则调用方不希望指定协议，服务提供商将选择要使用的 *协议* 。  当 *af* 参数AF\_INET或AF\_INET6且*类型***为SOCK\_RAW**时，为*协议*指定的值在 IPv6 或 IPv4 数据包标头的协议字段中设置。  下表列出了 *协议* 的常见值，尽管许多其他值是可能的。  展开表   |  |  | | --- | --- | | **protocol** | **含义** | | **IPPROTO\_ICMP**  1 | Internet 控制消息协议 (ICMP) 。 当 *af* 参数 **AF\_UNSPEC**、 **AF\_INET**或 **AF\_INET6** 且 *类型* 参数 **SOCK\_RAW** 或未指定时，此值可能为 。  Windows XP 及更高版本支持此 *协议* 值。 | | **IPPROTO\_IGMP**  2 | Internet 组管理协议 (IGMP) 。 当 *af* 参数 **AF\_UNSPEC**、 **AF\_INET**或 **AF\_INET6** 且 *类型* 参数 **SOCK\_RAW** 或未指定时，此值可能为 。  Windows XP 及更高版本支持此 *协议* 值。 | | **BTHPROTO\_RFCOMM**  3 | 蓝牙射频通信 (蓝牙 RFCOMM) 协议。 当 *af* 参数AF\_BTH且*类型*参数**SOCK\_STREAM**时，这是一个可能**的值。**  具有 SP2 或更高版本的 Windows XP 支持此 *协议* 值。 | | **IPPROTO\_TCP**  6 | 传输控制协议 (TCP) 。 当 *af* 参数**AF\_INET**或**AF\_INET6**且*类型*参数SOCK\_STREAM时，这是一个可能**的值。** | | **IPPROTO\_UDP**  17 | 用户数据报协议 (UDP) 。 当 *af* 参数**AF\_INET**或**AF\_INET6**且*类型*参数SOCK\_DGRAM时，这是一个可能**的值。** | | **IPPROTO\_ICMPV6**  58 | Internet 控制消息协议版本 6 (ICMPv6) 。 当 *af* 参数 **AF\_UNSPEC**、 **AF\_INET**或 **AF\_INET6** 且 *类型* 参数 **SOCK\_RAW** 或未指定时，此值可能为 。  Windows XP 及更高版本支持此 *协议* 值。 | | **IPPROTO\_RM**  113 | 可靠多播的 PGM 协议。 当 *af* 参数AF\_INET且*类型*参数**SOCK\_RDM**时，这是一个可能**的值。** 在 Windows Vista 及更高版本发布的Windows SDK中，此协议也称为**IPPROTO\_PGM**。  仅当安装了可靠多播 *协议* 时，才支持此协议值。 |  返回值 如果未发生错误， **套接字** 将返回引用新套接字的描述符。 否则，将返回值 INVALID\_SOCKET，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** | | [**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) | 网络子系统或关联的服务提供商发生故障。 | | [**WSAEAFNOSUPPORT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | ：不支持指定的地址系列。 例如，应用程序尝试为 **AF\_IRDA** 地址系列创建套接字，但未在本地计算机上安装红外适配器和设备驱动程序。 | | [**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。 | | [**WSAEMFILE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 无法提供更多套接字描述符。 | | [**WSAEINVAL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 提供的参数无效。 如果 *af* 参数设置为 **AF\_UNSPEC** 且*类型和协议*参数未指定，则返回此错误。 | | [**WSAEINVALIDPROVIDER**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 服务提供商返回了 2.2 以外的版本。 | | [**WSAEINVALIDPROCTABLE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 服务提供商向 [WSPStartup](https://learn.microsoft.com/zh-cn/windows/desktop/api/ws2spi/nf-ws2spi-wspstartup) 返回了无效或不完整的过程表。 | | [**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。 无法创建套接字。 | | [**WSAEPROTONOSUPPORT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 指定的协议不受支持。 | | [**WSAEPROTOTYPE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 指定的协议是此套接字的错误类型。 | | [**WSAEPROVIDERFAILEDINIT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 服务提供程序初始化失败。 如果分层服务提供程序 (LSP) 或命名空间提供程序安装不正确或提供程序无法正常运行，则返回此错误。 | | [**WSAESOCKTNOSUPPORT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | ：在此地址系列中，不支持指定的套接字类型。 |  注解 **套接字**函数会导致分配套接字描述符和任何相关资源并将其绑定到特定的传输服务提供程序。 Winsock 将使用第一个可用的服务提供程序，该提供程序支持所请求的地址系列、套接字类型和协议参数的组合。 创建的套接字将具有重叠属性作为默认值。 对于 Windows，Mswsock.h 中定义的特定于 Microsoft 的套接字选项SO\_OPENTYPE可能会影响此默认值。 有关SO\_OPENTYPE的详细说明，请参阅特定于 Microsoft 的文档。  可以使用 [WSASocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasocketa) 创建没有重叠属性的套接字。 如果与重叠操作相关的参数的值为 **NULL**，则允许 ([WSASend](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasend)、[WSARecv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsarecv)、[WSASendTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasendto)、[WSARecvFrom](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsarecvfrom) 和 [WSAIoctl](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaioctl)) 的所有函数也支持在重叠套接字上使用非重叠操作。  选择协议及其支持服务提供商时，此过程将仅选择基本协议或协议链，而不是协议层本身。 未链的协议层不被视为在 *类型* 或 *af* 上具有部分匹配项。 也就是说，如果未找到合适的协议，则它们不会导致 [WSAEAFNOSUPPORT](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 或 [WSAEPROTONOSUPPORT](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 的错误代码。  **注意** 清单常 **量AF\_UNSPEC** 继续在头文件中定义，但强烈建议不要使用它，因为这样可能会导致 *解释协议* 参数的值不明确。    建议应用程序将 **AF\_INET6** 用于 *af* 参数，并创建可与 IPv4 和 IPv6 一起使用的双模式套接字。  面向连接的套接字（如 **SOCK\_STREAM** ）提供全双工连接，并且必须处于连接状态，然后才能发送或接收任何数据。 使用 [连接](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect) 调用创建到另一个套接字的连接。 连接后，可以使用 [发送](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-send) 和 [recv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recv) 调用传输数据。 会话完成后，必须执行 [closesocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-closesocket) 。  用于实现面向连接的可靠套接字的通信协议可确保数据不会丢失或重复。 如果对等协议具有缓冲区空间的数据无法在合理的时间内成功传输，则认为连接断开，后续调用将失败，错误代码设置为 [WSAETIMEDOUT](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。  无连接、面向消息的套接字允许使用 [sendto](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-sendto) 和 [recvfrom](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recvfrom) 向任意对等方发送和接收数据报。 如果此类套接字连接到特定对等方，则可以使用 [send](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-send) 将数据报发送到该对等方，并且只能使用 [recv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recv) 从此对等方接收数据报。  接收具有**SOCK\_RAW***类型的*套接字时，IPv6 和 IPv4 的运行方式不同。 IPv4 接收数据包包括数据包有效负载、下一个上层标头 (例如，TCP 或 UDP 数据包) 的 IP 标头以及 IPv4 数据包标头。 IPv6 接收数据包包括数据包有效负载和下一个上层标头。 IPv6 接收数据包绝不包括 IPv6 数据包标头。  **注意**在Windows NT，原始套接字支持需要管理权限。    *类型*参数为 **SOCK\_SEQPACKET** 的套接字基于数据报，但充当伪流协议。 对于发送和接收数据包，使用单独的数据报。 但是，Windows 套接字可以将多个接收数据包合并为单个数据包。 因此，应用程序可以发出接收调用 (例如， [recv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recv) 或 [WSARecvEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/mswsock/nf-mswsock-wsarecvex)) ，并在单个调用中检索多个合并的多个数据包中的数据。 AF\_NETBIOS地址系列支持 **SOCK\_SEQPACKET***的类型*参数。  当通过 TCP/IP **为** NetBIOS AF\_NETBIOS *af参数时*，类型参数可以是**SOCK\_DGRAM**或**SOCK\_SEQPACKET**。 对于 **AF\_NETBIOS** 地址系列， *协议* 参数是表示为负数的 LAN 适配器编号。  在 Windows XP 及更高版本中，以下命令可用于列出 Windows 套接字目录，以确定安装的服务提供商以及支持的地址系列、套接字类型和协议。  **netsh winsock show catalog**  不需要支持 **类型为 SOCK\_RAW** 的套接字，但建议服务提供商尽可能支持原始套接字。 IrDA 套接字说明 请记住以下几点：   * 必须显式包含 Af\_irda.h 头文件。 * 仅支持 **SOCK\_STREAM** ;IrDA 不支持 **SOCK\_DGRAM** 类型。 * 对于 IrDA， *协议* 参数始终设置为 0。   仅当本地计算机安装了红外端口和驱动程序时，才能创建与 AF\_IRDA 地址系列配合使用的套接字。 否则，对*将 af* 参数设置为 AF\_IRDA 的**套接字**函数的调用将失败，[并且 WSAGetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsagetlasterror) 返回 [WSAEPROTONOSUPPORT](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。 示例代码 下面的示例演示如何使用 **套接字** 函数创建绑定到特定传输服务提供程序的套接字。  C++复制  #ifndef UNICODE  #define UNICODE 1  #endif  // link with Ws2\_32.lib  #pragma comment(lib,"Ws2\_32.lib")  #include <winsock2.h>  #include <ws2tcpip.h>  #include <stdio.h>  #include <stdlib.h> // Needed for \_wtoi  int \_\_cdecl wmain(int argc, wchar\_t \*\*argv)  {  //-----------------------------------------  // Declare and initialize variables  WSADATA wsaData = {0};  int iResult = 0;  // int i = 1;  SOCKET sock = INVALID\_SOCKET;  int iFamily = AF\_UNSPEC;  int iType = 0;  int iProtocol = 0;  // Validate the parameters  if (argc != 4) {  wprintf(L"usage: %s <addressfamily> <type> <protocol>\n", argv[0]);  wprintf(L"socket opens a socket for the specified family, type, & protocol\n");  wprintf(L"%ws example usage\n", argv[0]);  wprintf(L" %ws 0 2 17\n", argv[0]);  wprintf(L" where AF\_UNSPEC=0 SOCK\_DGRAM=2 IPPROTO\_UDP=17\n", argv[0]);  return 1;  }  iFamily = \_wtoi(argv[1]);  iType = \_wtoi(argv[2]);  iProtocol = \_wtoi(argv[3]);    // Initialize Winsock  iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);  if (iResult != 0) {  wprintf(L"WSAStartup failed: %d\n", iResult);  return 1;  }  wprintf(L"Calling socket with following parameters:\n");  wprintf(L" Address Family = ");  switch (iFamily) {  case AF\_UNSPEC:  wprintf(L"Unspecified");  break;  case AF\_INET:  wprintf(L"AF\_INET (IPv4)");  break;  case AF\_INET6:  wprintf(L"AF\_INET6 (IPv6)");  break;  case AF\_NETBIOS:  wprintf(L"AF\_NETBIOS (NetBIOS)");  break;  case AF\_BTH:  wprintf(L"AF\_BTH (Bluetooth)");  break;  default:  wprintf(L"Other");  break;  }  wprintf(L" (%d)\n", iFamily);    wprintf(L" Socket type = ");  switch (iType) {  case 0:  wprintf(L"Unspecified");  break;  case SOCK\_STREAM:  wprintf(L"SOCK\_STREAM (stream)");  break;  case SOCK\_DGRAM:  wprintf(L"SOCK\_DGRAM (datagram)");  break;  case SOCK\_RAW:  wprintf(L"SOCK\_RAW (raw)");  break;  case SOCK\_RDM:  wprintf(L"SOCK\_RDM (reliable message datagram)");  break;  case SOCK\_SEQPACKET:  wprintf(L"SOCK\_SEQPACKET (pseudo-stream packet)");  break;  default:  wprintf(L"Other");  break;  }  wprintf(L" (%d)\n", iType);  wprintf(L" Protocol = %d = ", iProtocol);  switch (iProtocol) {  case 0:  wprintf(L"Unspecified");  break;  case IPPROTO\_ICMP:  wprintf(L"IPPROTO\_ICMP (ICMP)");  break;  case IPPROTO\_IGMP:  wprintf(L"IPPROTO\_IGMP (IGMP)");  break;  case IPPROTO\_TCP:  wprintf(L"IPPROTO\_TCP (TCP)");  break;  case IPPROTO\_UDP:  wprintf(L"IPPROTO\_UDP (UDP)");  break;  case IPPROTO\_ICMPV6:  wprintf(L"IPPROTO\_ICMPV6 (ICMP Version 6)");  break;  default:  wprintf(L"Other");  break;  }  wprintf(L" (%d)\n", iProtocol);  sock = socket(iFamily, iType, iProtocol);  if (sock == INVALID\_SOCKET)  wprintf(L"socket function failed with error = %d\n", WSAGetLastError() );  else {  wprintf(L"socket function succeeded\n");  // Close the socket to release the resources associated  // Normally an application calls shutdown() before closesocket  // to disables sends or receives on a socket first  // This isn't needed in this simple sample  iResult = closesocket(sock);  if (iResult == SOCKET\_ERROR) {  wprintf(L"closesocket failed with error = %d\n", WSAGetLastError() );  WSACleanup();  return 1;  }  }  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** | winsock2.h | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 [IPPROTO\_IP套接字选项](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/ipproto-ip-socket-options)  [IPPROTO\_IPV6套接字选项](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/ipproto-ipv6-socket-options)  [可靠的多播编程](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/reliable-multicast-programming--pgm-)  [WSASocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasocketa)  [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)  [accept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept)  [bind](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind)  [closesocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-closesocket)  [connect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect)  [getsockname](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockname)  [getsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockopt)  [ioctlsocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-ioctlsocket)  [listen](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-listen)  [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)  [sendto](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-sendto)  [setsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt)  [shutdown](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-shutdown) |

## connect 函数语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **connect** 函数与指定的套接字建立连接。 语法 C++  int WSAAPI connect(  [in] SOCKET s,  [in] const sockaddr \*name,  [in] int namelen  ); 参数 [in] s  标识未连接的套接字的描述符。  [in] name  指向应建立连接的 [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构的指针。  [in] namelen  *name* 参数指向的 [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构的长度（以字节为单位）。 返回值 如果未发生错误， **则连接** 返回零。 否则，它将返回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。  在阻塞套接字上，返回值指示连接尝试成功或失败。  使用非阻止套接字时，无法立即完成连接尝试。 在这种情况下， **连接** 将返回SOCKET\_ERROR， [WSAGetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsagetlasterror) 将返回 [WSAEWOULDBLOCK](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。 在这种情况下，有三种可能的方案：   * 使用 [select](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-select) 函数通过检查套接字是否可写来确定连接请求的完成情况。 * 如果应用程序使用 [WSAAsyncSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsaasyncselect) 来指示对连接事件感兴趣，则应用程序将收到FD\_CONNECT通知，指示 **连接** 操作已完成 (成功或未) 。 * 如果应用程序使用 [WSAEventSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaeventselect) 来指示对连接事件的兴趣，则将向关联的事件对象发出信号，指示 **连接** 操作已完成 (成功或未) 。   在非阻止套接字上完成连接尝试之前，在连接成功完成时，在同一套接字上进行 **连接** 的所有后续调用都将失败并出现错误代码 [WSAEALREADY](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 和 [WSAEISCONN](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 。 由于 Windows 套接字规范版本 1.1 中的多义性，在连接已挂起时从 **连接** 返回的错误代码可能因实现而异。 因此，不建议应用程序使用多个调用进行连接以检测连接完成情况。 如果他们这样做，则必须准备好处理 [WSAEINVAL](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 和 [WSAEWOULDBLOCK](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 错误值的方式与处理 [WSAEALREADY](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 的方式相同，以确保可靠的操作。  如果返回的错误代码指示连接尝试失败 (即 [WSAECONNREFUSED](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)、 [WSAENETUNREACH](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)、 [WSAETIMEDOUT](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)) 应用程序可以为同一套接字再次调用 **connect** 。  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。 | | [**WSAEADDRINUSE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字的本地地址已在使用中，并且未将套接字标记为允许使用 SO\_REUSEADDR 重用地址。 此错误通常在执行[绑定](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind)时发生，但如果**绑定**是本地 IP 地址的通配符地址 (**INADDR\_ANY**或**in6addr\_any**) ，则可能会延迟到[连接](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect)功能。 特定地址需要由 **connect** 函数隐式绑定。 | | [**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 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。 | | [**WSAEALREADY**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 正在指定的套接字上进行非阻止 [连接](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect) 调用。  **注意** 为了保持向后兼容性，此错误以 [**WSAEINVAL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 的形式报告给链接到 Winsock.dll 或 Wsock32.dll 的 Windows 套接字 1.1 应用程序。 | | [**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 **INADDR\_ANY** 或 **in6addr\_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。 | | [**WSAECONNREFUSED**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 尝试连接被强制性拒绝。 | | [**WSAEFAULT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | *名称*指向[的 sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构包含关联的地址系列地址格式不正确，或者 *namelen* 参数太小。 如果 *name* 参数所指向的长度在 *namelen* 参数中指定的 **sockaddr** 结构不在用户地址空间的有效部分，则也会返回此错误。 | | [**WSAEINVAL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 参数 *s* 是侦听套接字。 | | [**WSAEISCONN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 仅) 面向连接的套接字 (已连接套接字。 | | [**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。 | | [**WSAENOBUFS**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | **注意** 没有可用的缓冲区空间。 无法连接套接字。 | | [**WSAENOTSOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | *s* 参数中指定的描述符不是套接字。 | | [**WSAETIMEDOUT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 尝试连接超时，但未建立连接。 | | [**WSAEWOULDBLOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | ：套接字标记为非阻止，无法立即完成连接。 | | [**WSAEACCES**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 尝试将数据报套接字连接到广播地址失败，因为未启用 [setsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt) 选项SO\_BROADCAST。 |  备注 **connect** 函数用于创建到指定目标的连接。 如果套接字 *未*绑定，则系统会将唯一值分配给本地关联，并将套接字标记为绑定。  对于面向连接的套接字 (例如，SOCK\_STREAM) 类型，将使用 *名称* (套接字命名空间中的地址发起到外部主机的活动连接;有关详细说明，请参阅 [bind](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind) 和 [sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2)) 。  **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，然后发出 [**sendto**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-sendto) 调用，Windows 套接字将执行隐式 [**绑定**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind) 函数调用。    套接字调用成功完成后，套接字已准备好发送和接收数据。 如果 *name* 参数指定的结构的地址成员用零填充， **则连接** 将返回错误 [WSAEADDRNOTAVAIL](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。 重新连接活动连接的任何尝试都将失败，错误代码为 [WSAEISCONN](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。  对于面向连接的非阻塞套接字，通常无法立即完成连接。 在这种情况下，此函数返回错误 [WSAEWOULDBLOCK](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。 但是，该操作继续进行。  当成功或失败结果已知时，可能会以两种方式之一进行报告，具体取决于客户端注册通知的方式。   * 如果客户端使用 [select](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-select) 函数，则会在 writefds 集中报告成功，在 exceptfds 集中报告失败。 * 如果客户端使用函数 [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)，则会使用 FD\_CONNECT 通知，并且与FD\_CONNECT关联的错误代码指示成功或失败的特定原因。   如果未立即完成连接，客户端应等待连接完成，然后尝试使用 [setsockopt](https://learn.microsoft.com/zh-cn/windows/win32/api/winsock/nf-winsock-setsockopt) 设置套接字选项。 不支持在连接正在进行时调用 setsockopt。  对于无连接套接字 (例如，SOCK\_DGRAM) 类型， **连接** 执行的操作只是为了建立可用于后续 [发送](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-send)/ [WSASend](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasend) 和 [recv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recv)/ [WSARecv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsarecv) 调用的默认目标地址。 从指定的目标地址以外的地址接收的任何数据报将被丢弃。 如果 *name* 指定的结构的地址成员用零填充，则套接字将断开连接。 然后，默认远程地址将不确定，因此 [send](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-send)/ [WSASend](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasend) 和 [recv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recv)/ [WSARecv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsarecv) 调用将返回错误代码 [WSAENOTCONN](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。 但是， [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) 和 [recvfrom](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recvfrom)/ [WSARecvFrom](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsarecvfrom) 仍可使用。 只需再次调用 **连接** 即可更改默认目标，即使套接字已连接也是如此。 如果 *名称* 与上一个 **连接**不同，则放弃排队接收的任何数据报。  对于无连接套接字， *名称* 可以指示任何有效的地址，包括广播地址。 但是，若要连接到广播地址，套接字必须使用 [setsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt) 来启用SO\_BROADCAST选项。 否则， **连接** 将失败，错误代码 [为 WSAEACCES](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。  当套接字之间的连接断开时，应放弃已连接的套接字，并创建新的套接字。 当在连接的套接字上出现问题时，应用程序必须放弃套接字并再次创建套接字，以便返回到稳定点。  **注意** 发出阻止的 Winsock 调用（如 **连接**）时，Winsock 可能需要等待网络事件，才能完成调用。 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 **connect** 函数。  C++复制  #ifndef UNICODE  #define UNICODE  #endif  #define WIN32\_LEAN\_AND\_MEAN  #include <winsock2.h>  #include <ws2tcpip.h>  #include <stdio.h>  // Need to link with Ws2\_32.lib  #pragma comment(lib, "ws2\_32.lib")  int wmain()  {  //----------------------  // Initialize Winsock  WSADATA wsaData;  int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);  if (iResult != NO\_ERROR) {  wprintf(L"WSAStartup function failed with error: %d\n", iResult);  return 1;  }  //----------------------  // Create a SOCKET for connecting to server  SOCKET ConnectSocket;  ConnectSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (ConnectSocket == INVALID\_SOCKET) {  wprintf(L"socket function failed with error: %ld\n", WSAGetLastError());  WSACleanup();  return 1;  }  //----------------------  // The sockaddr\_in structure specifies the address family,  // IP address, and port of the server to be connected to.  sockaddr\_in clientService;  clientService.sin\_family = AF\_INET;  clientService.sin\_addr.s\_addr = inet\_addr("127.0.0.1");  clientService.sin\_port = htons(27015);  //----------------------  // Connect to server.  iResult = connect(ConnectSocket, (SOCKADDR \*) & clientService, sizeof (clientService));  if (iResult == SOCKET\_ERROR) {  wprintf(L"connect function failed with error: %ld\n", WSAGetLastError());  iResult = closesocket(ConnectSocket);  if (iResult == SOCKET\_ERROR)  wprintf(L"closesocket function failed with error: %ld\n", WSAGetLastError());  WSACleanup();  return 1;  }  wprintf(L"Connected to server.\n");  iResult = closesocket(ConnectSocket);  if (iResult == SOCKET\_ERROR) {  wprintf(L"closesocket function failed with error: %ld\n", WSAGetLastError());  WSACleanup();  return 1;  }  WSACleanup();  return 0;  }  有关使用 **connect** 函数的另一个示例，请参阅 [入门 With Winsock](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/getting-started-with-winsock)。 IrDA 套接字说明  * 必须显式包含 Af\_irda.h 头文件。 * 如果在媒体访问级别检测到现有的 IrDA 连接，则返回 [WSAENETDOWN](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 。 * 如果存在与具有不同地址的设备的活动连接，则返回 [WSAEADDRINUSE](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 。 * 如果套接字已连接或独占/多路复用模式更改失败，则返回 [WSAEISCONN](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 。 * 如果套接字以前绑定到本地服务名称以使用 [绑定](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind)接受传入连接，则返回 [WSAEINVAL](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 。 请注意，绑定套接字后，它不能用于建立出站连接。   IrDA 使用表单sockaddr\_irda地址实现 connect 函数。 通常，客户端应用程序将使用套接字功能创建一个套接字，使用“IRLMP\_ENUMDEVICES套接字”选项扫描 IrDA 设备的邻近区域，从返回的列表中选择设备，形成地址，然后调用 **connect**。 阻止语义和非阻止语义之间没有区别。 要求  |  |  | | --- | --- | | **标头** | 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)  [WSAConnect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaconnect)  [ConnectEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/mswsock/nc-mswsock-lpfn_connectex)  [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)  [accept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept)  [bind](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind)  [getsockname](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockname)  [select](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-select)  [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) |

## closesocket 函数语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **closesocket** 函数关闭现有套接字。 语法 C++复制  int closesocket(  [in] SOCKET s  ); 参数 [in] s  标识要关闭的套接字的描述符。 返回值 如果未发生错误， **则 closesocket** 返回零。 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** | | [**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。 | | [**WSAENOTSOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | ：描述符不是套接字。 | | [**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。 | | [**WSAEINTR**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | (阻止) 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。 | | [**WSAEWOULDBLOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字标记为非阻止，但 [linger](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-linger) 结构的**l\_onoff**成员设置为非零，而 **linger** 结构的**l\_linger**成员设置为非零超时值。 |  注解 **closesocket** 函数关闭套接字。 使用它释放在 *s* 参数中传递的套接字描述符。 请注意，一旦发出 **closesocket** 函数，系统可能会立即重用传入 *s* 参数的套接字描述符。 因此，预期对 *s* 参数中传递的套接字描述符的进一步引用失败并出现错误 [WSAENOTSOCK](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 是不可靠的。 Winsock 客户端不得在另一个 Winsock 函数调用的同时在 *上*发出 **closesocket**。  ( [WSASendTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasend)/ [WSARecvWSendTo](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasendto)/ [WSARecvFrom](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsarecv)/ 的任何挂起的重叠发送和接收操作，以及此进程中任何线程颁发的重叠套接字) 也会被取消。 将执行为这些重叠操作指定的任何事件、完成例程或完成端口操作。 挂起的重叠操作失败，错误状态 [WSA\_OPERATION\_ABORTED](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。  应用程序不应假定当 **closesocket** 返回时，套接字上任何未完成的 I/O 操作都保证完成。 **closesocket** 函数将对未完成的 I/O 操作启动取消，但这并不意味着应用程序将在 **closesocket** 函数返回时收到这些 I/O 操作的 I/O 完成。 因此，应用程序不应清除 [WSAOVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/ns-winsock2-wsaoverlapped) 结构 (任何资源，例如) 未完成的 I/O 请求引用，直到 I/O 请求确实完成。  对于每次成功调用[套接字](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-socket)，应用程序应始终对 **closesocket** 进行匹配调用，以便将任何套接字资源返回到系统。  [linger](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-linger) 结构维护有关特定套接字的信息，该套接字指定在排队发送数据并在套接字上调用 **closesocket** 函数时该套接字的行为方式。  **linger** 结构的**l\_onoff**成员确定套接字在**调用 closesocket** 函数后是否应在指定的时间内保持打开状态，以允许发送排队数据。 可以通过两种方式修改此成员：   * 调用 [setockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt) 函数，并将 *optname* 参数设置为 **SO\_DONTLINGER**。 *optval* 参数确定**如何修改 l\_onoff** 成员。 * 调用 [setockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt) 函数，并将 *optname* 参数设置为 **SO\_LINGER**。 *optval* 参数指定如何修改**l\_onoff**和**l\_linger**成员。   **linger** 结构的 **l\_linger** 成员确定套接字应保持打开状态的时间量（以秒为单位）。 仅当 **linger** 结构的**l\_onoff**成员为非零时，此成员才适用。  套接字的默认参数是 **linger** 结构的**l\_onoff**成员为零，表示套接字不应保持打开状态。 **linger** 结构的**l\_linger**成员的默认值为零，但当**l\_onoff**成员设置为零时，此值将被忽略。  若要使套接字保持打开状态，应用程序应将 **l\_onoff** 成员设置为非零值，并将 **l\_linger** 成员设置为所需的超时（以秒为单位）。 若要禁用套接字保持打开状态，应用程序只需将 **linger** 结构的 **l\_onoff** 成员设置为零。  如果应用程序调用 *optname* 参数设置为 **SO\_DONTLINGER**[的 setsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt) 函数，将 **l\_onoff** 成员设置为非零值，则不指定 **l\_linger** 成员的值。 在这种情况下，使用的超时依赖于实现。 如果以前通过调用 **setockopt** 函数（ *optname* 参数设置为 SO\_LINGER) ）为套接字 **(** 建立了以前的超时，则服务提供程序应恢复此超时值。  **closesocket** 函数的语义受设置 **linger** 结构成员的套接字选项的影响。  展开表   |  |  |  |  | | --- | --- | --- | --- | | **l\_onoff** | **l\_linger** | **关闭类型** | **等待关闭？** | | 零 | 不在乎 | 优雅关闭 | 否 | | 零 | 零 | 硬色调 | 否 | | 零 | 零 | 如果所有数据在 **l\_linger** 成员中指定的超时值内发送，则为正常。  如果无法在 **l\_linger** 成员中指定的超时值内发送所有数据，则很难。 | 是 |     如果 [LINGER](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-linger) 结构的**l\_onoff**成员在流套接字上为零，**则 closesocket** 调用将立即返回，并且无论套接字是阻塞还是非阻塞，都不会收到 [WSAEWOULDBLOCK](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。 但是，如果可能，在关闭基础套接字之前，将发送任何排队等待传输的数据。 这也称为正常断开连接或关闭。 在这种情况下，Windows 套接字提供程序无法在任意时间段内释放套接字和其他资源，从而影响希望使用所有可用套接字的应用程序。 这是套接字的默认行为。  如果 [linger](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-linger) 结构的**l\_onoff**成员为非零且**l\_linger**成员为零，则即使尚未发送或确认排队数据，**也不会阻止 closesocket**。 这称为硬关闭或中止关闭，因为套接字的虚拟线路会立即重置，并且任何未发送的数据都将丢失。 在 Windows 上，线路远程端的任何 **recv** 调用都将失败并出现 [WSAECONNRESET](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。  如果 [linger](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-linger) 结构的**l\_onoff**成员设置为非零，**并且l\_linger**成员在阻止套接字上设置为非零超时，则 **closesocket** 调用将一直阻止，直到发送剩余数据或超时过期为止。 如果在 **l\_linger** 成员中指定的超时值内发送所有数据，则这称为正常断开连接或关闭。 如果超时在发送所有数据之前过期，Windows 套接字实现会在 **closesocket** 返回之前终止连接，这称为硬关闭或中止关闭。  不建议将 [linger](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-linger) 结构的 **l\_onoff** 成员设置为非零，并将非阻止套接字上具有非零超时间隔**的 l\_linger** 成员。 在这种情况下，如果关闭操作无法立即完成，对 **closesocket 的** 调用将失败并显示 [WSAEWOULDBLOCK](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) 错误。 如果 **closesocket** 失败并出现 [WSAEWOULDBLOCK](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) ，则套接字句柄仍然有效，并且不会启动断开连接。 应用程序必须再次调用 **closesocket** 才能关闭套接字。  如果 [linger](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-linger) 结构的**l\_onoff**成员为非零值，**并且 l\_linger** 成员是阻塞套接字上的非零超时间隔，则 **closesocket** 函数的结果不能用于确定是否所有数据都已发送到对等方。 如果在 **l\_linger** 成员中指定的超时过期之前发送数据，或者如果连接中止， **则 closesocket** 函数不会返回错误代码， (**closesocket** 函数的返回值为零) 。  **closesocket** 调用只会阻止，直到所有数据都已传递到对等方或超时过期。 如果连接因超时过期而重置，则套接字不会进入TIME\_WAIT状态。 如果所有数据在超时期限内发送，则套接字可以进入TIME\_WAIT状态。  如果 [linger](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-linger) 结构的**l\_onoff**成员为非零，并且 **l\_linger** 成员在阻塞套接字上为零超时间隔，则对 **closesocket** 的调用将重置连接。 套接字不会进入TIME\_WAIT状态。  可以在将 *optname* 参数设置为 **SO\_LINGER** 的情况下调用 [getsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockopt) 函数，以检索与套接字关联的 **linger** 结构的当前值。  **注意**为了确保在连接上发送和接收所有数据，应用程序应在调用 **closesocket** 之前调用[**关闭**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-shutdown)， (请参阅[**正常关闭、徘徊选项和套接字关闭**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/graceful-shutdown-linger-options-and-socket-closure-2)以获取详细信息) 。 另请注意，调用 **closesocket** 后不会发布FD\_CLOSE网络事件。    下面是 **closesocket** 行为的摘要：   * 如果 [LINGER](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-linger) 结构的**l\_onoff**成员为零， (套接字) 的默认值，**则 closesocket** 将立即返回，连接在后台正常关闭。 * 如果 [linger](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-linger) 结构的**l\_onoff**成员设置为非零，**并且 l\_linger** 成员设置为零 (则没有超时) **closesocket** 会立即返回并且连接重置或终止。 * 如果 [linger](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-linger) 结构的 **l\_onoff** 成员设置为非零值，并将**l\_linger**成员设置为非零超时：– 对于阻塞套接字，**则 closesocket** 会阻止，直到发送所有数据或超时到期为止。   – 对于非阻止套接字， **closesocket** 会立即返回指示失败。  有关详细信息 [，请参阅正常关闭、挥之不去的选项和套接字关闭](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/graceful-shutdown-linger-options-and-socket-closure-2) 。  **注意** 发出阻止 Winsock 调用（如 **closesocket）**时，Winsock 可能需要等待网络事件，才能完成调用。 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。   IrDA 套接字说明 请记住以下几点：   * 必须显式包含 Af\_irda.h 头文件。 * 支持标准挥之不去的选项。 * 尽管 IrDA 不提供正常关闭，但 IrDA 将推迟关闭，直到清除接收队列。 因此，应用程序可以发送数据并立即调用 [套接字](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-socket) 函数，并确信接收方会在接收FD\_CLOSE消息之前复制数据。  ATM 说明 以下是使用异步传输模式 (ATM) 和 Windows 套接字 2 时与连接断开相关的重要问题：   * 将 **closesocket** 或 [关闭](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-shutdown) 函数与SD\_SEND或SD\_BOTH会导致在控制通道上发出 RELEASE 信号。 由于 ATM 使用单独的信号和数据通道，RELEASE 信号可能会在最后一个数据到达其目标之前到达远程端，从而导致数据丢失。 一种可能的解决方案是在发送的最后一个数据与 ATM 套接字的 **closesocket** 或 [shutdown](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-shutdown) 函数调用之间设置足够的延迟。 * ATM 不支持半关闭。 * 中止和正常断开连接都会导致发布信号以相同的原因字段发出。 在任一情况下，在套接字的远程端接收的数据仍会传递到应用程序。 有关详细信息 [，请参阅正常关闭、挥之不去的选项和套接字关闭](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/graceful-shutdown-linger-options-and-socket-closure-2) 。   **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 [正常关闭、挥之不去的选项和套接字关闭](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/graceful-shutdown-linger-options-and-socket-closure-2)  [WSAAsyncSelect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsaasyncselect)  [WSADuplicateSocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaduplicatesocketa)  [WSAOVERLAPPED](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/ns-winsock2-wsaoverlapped)  [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)  [accept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept)  [getsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockopt)  [ioctlsocket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-ioctlsocket)  [萦绕](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-linger)  [setsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt)  [socket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-socket) |

## send函数用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **send** 函数在连接的套接字上发送数据。 语法 C++复制  int WSAAPI send(  [in] SOCKET s,  [in] const char \*buf,  [in] int len,  [in] int flags  ); 参数 [in] s  标识已连接套接字的描述符。  [in] buf  指向包含要传输的数据的缓冲区的指针。  [in] len  *buf* 参数指向的缓冲区中数据的长度（以字节为单位）。  [in] flags  一组指定调用方式的标志。 此参数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\_DONTROUTE** | ：指定数据不应受到路由的约束。 Windows 套接字服务提供程序可以选择忽略此标志。 | | **MSG\_OOB** | 仅) (流式套接字（例如SOCK\_STREAM）发送 OOB 数据。 |  返回值 如果未发生错误， **send** 将返回发送的总字节数，该字节数可能小于 *在 len* 参数中请求发送的字节数。 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。 | | [**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 套接字选项调用 [setsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt) 以启用广播地址的使用。 | | [**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 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。 | | [**WSAEFAULT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | *buf* 参数未完全包含在用户地址空间的有效部分。 | | [**WSAENETRESET**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 由于执行操作过程中保持活动状态的活动检测到故障，连接已经损坏。 | | [**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) | 套接字未连接。 | | [**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) 后，无法在套接字上发送 ， *并将 如何* 设置为 SD\_SEND 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大小。 | | [**WSAEHOSTUNREACH**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 此时无法从此主机访问远程主机。 | | [**WSAEINVAL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 尚未 [绑定套接字](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。 | | [**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 UDP 套接字，远程主机无法传送以前发送的 UDP 数据报，并响应了“端口无法访问”ICMP 数据包。 因为套接字不可再用，应用程序应关闭套接字。 | | [**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。 |  注解 **send** 函数用于在连接的套接字上写入传出数据。  对于面向消息的套接字 (**地址系列AF\_INET** 或 **AF\_INET6**、 **SOCK\_DGRAM**类型和 **IPPROTO\_UDP**协议（例如) ），必须注意不要超过基础提供程序的最大数据包大小。 可以通过调用将 *optname* 参数设置为 **SO\_MAX\_MSG\_SIZE**[的 getsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockopt) 来获取提供程序的最大消息数据包大小，以检索 socket 选项的值。 如果数据太长，无法通过基础协议以原子方式传递，则返回错误 [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。  **发送**函数的成功完成并不表示数据已成功传递并接收给收件人。 此函数仅指示已成功发送数据。  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，除非套接字处于非阻止模式，否则 **发送** 将阻塞。 在面向非阻止流的套接字上，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计算机上的缓冲区可用性。 [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。  允许使用 *len* 参数为零调用 **send**，并且实现将被视为成功。 在这种情况下， **send** 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* 参数可用于影响为关联套接字指定的选项之外的函数的行为。 **send** 函数的语义由之前在 *s* 参数中指定的套接字上设置的任何选项以及传递给 **send** 函数的 *flags* 参数确定。  调用 **发送** 的顺序也是缓冲区传输到传输层的顺序。 不应从不同的线程同时在同一个面向流的套接字上调用 **send**，因为某些 Winsock 提供程序可能会将大型发送请求拆分为多个传输，这可能会导致意外的数据从同一流导向套接字上的多个并发发送请求交错。  **注意** 发出阻止 Winsock 调用（如 **send**）时，Winsock 可能需要等待网络事件，然后才能完成调用。 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 **send** 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")  #define DEFAULT\_BUFLEN 512  #define DEFAULT\_PORT 27015  int main() {  //----------------------  // Declare and initialize variables.  int iResult;  WSADATA wsaData;  SOCKET ConnectSocket = INVALID\_SOCKET;  struct sockaddr\_in clientService;  int recvbuflen = DEFAULT\_BUFLEN;  char \*sendbuf = "Client: sending data test";  char recvbuf[DEFAULT\_BUFLEN] = "";  //----------------------  // 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 connecting to server  ConnectSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (ConnectSocket == INVALID\_SOCKET) {  wprintf(L"socket failed with error: %ld\n", WSAGetLastError());  WSACleanup();  return 1;  }  //----------------------  // The sockaddr\_in structure specifies the address family,  // IP address, and port of the server to be connected to.  clientService.sin\_family = AF\_INET;  clientService.sin\_addr.s\_addr = inet\_addr( "127.0.0.1" );  clientService.sin\_port = htons( DEFAULT\_PORT );  //----------------------  // Connect to server.  iResult = connect( ConnectSocket, (SOCKADDR\*) &clientService, sizeof(clientService) );  if (iResult == SOCKET\_ERROR) {  wprintf(L"connect failed with error: %d\n", WSAGetLastError() );  closesocket(ConnectSocket);  WSACleanup();  return 1;  }  //----------------------  // Send an initial buffer  iResult = send( ConnectSocket, sendbuf, (int)strlen(sendbuf), 0 );  if (iResult == SOCKET\_ERROR) {  wprintf(L"send failed with error: %d\n", WSAGetLastError());  closesocket(ConnectSocket);  WSACleanup();  return 1;  }  printf("Bytes Sent: %d\n", iResult);  // shutdown the connection since no more data will be sent  iResult = shutdown(ConnectSocket, SD\_SEND);  if (iResult == SOCKET\_ERROR) {  wprintf(L"shutdown failed with error: %d\n", WSAGetLastError());  closesocket(ConnectSocket);  WSACleanup();  return 1;  }  // Receive until the peer closes the connection  do {  iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);  if ( iResult > 0 )  wprintf(L"Bytes received: %d\n", iResult);  else if ( iResult == 0 )  wprintf(L"Connection closed\n");  else  wprintf(L"recv failed with error: %d\n", WSAGetLastError());  } while( iResult > 0 );  // close the socket  iResult = closesocket(ConnectSocket);  if (iResult == SOCKET\_ERROR) {  wprintf(L"close failed with error: %d\n", WSAGetLastError());  WSACleanup();  return 1;  }  WSACleanup();  return 0;  } 示例代码 有关使用 **send** 函数的另一个示例，请参阅 [入门 With Winsock](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/getting-started-with-winsock)。 IrDA 套接字说明  * 必须显式包含 Af\_irda.h 头文件。   **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** | winsock2.h | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 [使用 Winsock 入门](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/getting-started-with-winsock)  [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)  [sendto](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-sendto)  [socket](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-socket) |

## recv函数的语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **recv** 函数从连接的套接字或绑定的无连接套接字接收数据。 语法 C++复制  int recv(  [in] SOCKET s,  [out] char \*buf,  [in] int len,  [in] int flags  ); 参数 [in] s  标识连接的套接字的描述符。  [out] buf  指向用于接收传入数据的缓冲区的指针。  [in] len  *buf* 参数指向的缓冲区的长度（以字节为单位）。  [in] flags  影响此函数行为的一组标志。 请参阅下面的备注。 有关此参数的可能值的详细信息，请参阅“备注”部分。 返回值 如果未发生错误， **recv** 将返回收到的字节数， *buf* 参数指向的缓冲区将包含接收的此数据。 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* 参数未完全包含在用户地址空间的有效部分。 | | [**WSAENOTCONN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字未连接。 | | [**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。 | | [**WSAENETRESET**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 对于面向连接的套接字，此错误表示连接已中断，因为 *保持活动* 活动检测到操作正在进行时失败。 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) | ：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;调用[关闭](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-shutdown)后，无法接收套接字上的SD\_RECEIVE或SD\_BOTH。 | | [**WSAEWOULDBLOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 套接字标记为非阻止，接收操作将阻止。 | | [**WSAEMSGSIZE**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 消息太大，无法放入指定的缓冲区，并且已被截断。 | | [**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，或者仅对字节流套接字 () *len* 为零或负数。 | | [**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。 | | [**WSAETIMEDOUT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 因为网络故障或对等系统无法响应，已经丢弃了连接。 | | [**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。 因为套接字不可再用，应用程序应关闭套接字。 在 UDP 数据报套接字上，此错误将指示以前的发送操作导致 ICMP“端口无法访问”消息。 |  注解 **recv** 函数用于读取面向连接的套接字或无连接套接字上的传入数据。 使用面向连接的协议时，必须在调用 **recv** 之前连接套接字。 使用无连接协议时，必须在调用 **recv** 之前绑定套接字。  套接字的本地地址必须是已知的。 对于服务器应用程序，请使用显式 [绑定](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-bind) 函数或隐式 [accept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept) 或 [WSAAccept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaaccept) 函数。 不建议对客户端应用程序进行显式绑定。 对于客户端应用程序，套接字可以使用 [connect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect)、 [WSAConnect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaconnect)、 [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。  对于连接的或无连接的套接字， **recv** 函数限制接收消息的地址。 函数仅返回来自连接中指定的远程地址的消息。 来自其他地址的消息 (静默) 丢弃。  对于面向连接的套接字 (类型SOCK\_STREAM例如) ，调用 **recv** 将返回当前可用的数据量，最大为指定的缓冲区大小。 如果已将套接字配置为对 OOB 数据进行内联接收， (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 (类型SOCK\_DGRAM或其他面向消息的套接字) 的无连接套接字，数据从 [连接](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect) 函数指定的目标地址提取第一个排队数据报 (消息) 。  如果数据报或消息大于指定的缓冲区，则会使用数据报的第一部分填充缓冲区，而 **recv** 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;对于可靠的协议，数据由服务提供商保留，直到使用足够大的缓冲区调用 **recv** 成功读取。  如果套接字上没有可用的传入数据， **则 recv** 调用会根据为 [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) 函数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，并且已接收所有数据， **则 recv** 将立即完成，且接收了零个字节。 如果连接已重置， **则 recv** 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。 *flags* 参数的可能值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。 数据将复制到缓冲区中，但不会从输入队列中删除。 | | MSG\_OOB | 处理带外 (OOB) 数据。 | | MSG\_WAITALL | 仅当发生以下事件之一时，接收请求才会完成：   * 调用方提供的缓冲区已完全满。 * 该连接已关闭。 * 请求已取消或发生错误。   请注意，如果基础传输不支持MSG\_WAITALL，或者套接字处于非阻塞模式，则此调用将失败并出现 **WSAEOPNOTSUPP**。 此外，如果MSG\_WAITALL与MSG\_OOB、MSG\_PEEK或MSG\_PARTIAL一起指定，则此调用将失败并显示 **WSAEOPNOTSUPP**。 数据报套接字或面向消息的套接字不支持此标志。 |     **注意** 发出阻止 Winsock 调用（如 **recv**）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 下面的代码示例演示如何使用 **recv** 函数。  C++复制  #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")  #define DEFAULT\_BUFLEN 512  #define DEFAULT\_PORT "27015"  int \_\_cdecl main() {  //----------------------  // Declare and initialize variables.  WSADATA wsaData;  int iResult;  SOCKET ConnectSocket = INVALID\_SOCKET;  struct sockaddr\_in clientService;  char \*sendbuf = "this is a test";  char recvbuf[DEFAULT\_BUFLEN];  int recvbuflen = DEFAULT\_BUFLEN;    //----------------------  // Initialize Winsock  iResult = WSAStartup(MAKEWORD(2,2), &wsaData);  if (iResult != NO\_ERROR) {  printf("WSAStartup failed: %d\n", iResult);  return 1;  }  //----------------------  // Create a SOCKET for connecting to server  ConnectSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (ConnectSocket == INVALID\_SOCKET) {  printf("Error at socket(): %ld\n", WSAGetLastError() );  WSACleanup();  return 1;  }  //----------------------  // The sockaddr\_in structure specifies the address family,  // IP address, and port of the server to be connected to.  clientService.sin\_family = AF\_INET;  clientService.sin\_addr.s\_addr = inet\_addr( "127.0.0.1" );  clientService.sin\_port = htons( 27015 );  //----------------------  // Connect to server.  iResult = connect( ConnectSocket, (SOCKADDR\*) &clientService, sizeof(clientService) );  if ( iResult == SOCKET\_ERROR) {  closesocket (ConnectSocket);  printf("Unable to connect to server: %ld\n", WSAGetLastError());  WSACleanup();  return 1;  }  // Send an initial buffer  iResult = send( ConnectSocket, sendbuf, (int)strlen(sendbuf), 0 );  if (iResult == SOCKET\_ERROR) {  printf("send failed: %d\n", WSAGetLastError());  closesocket(ConnectSocket);  WSACleanup();  return 1;  }  printf("Bytes Sent: %ld\n", iResult);  // shutdown the connection since no more data will be sent  iResult = shutdown(ConnectSocket, SD\_SEND);  if (iResult == SOCKET\_ERROR) {  printf("shutdown failed: %d\n", WSAGetLastError());  closesocket(ConnectSocket);  WSACleanup();  return 1;  }  // Receive until the peer closes the connection  do {  iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);  if ( iResult > 0 )  printf("Bytes received: %d\n", iResult);  else if ( iResult == 0 )  printf("Connection closed\n");  else  printf("recv failed: %d\n", WSAGetLastError());  } while( iResult > 0 );  // cleanup  closesocket(ConnectSocket);  WSACleanup();  return 0;  } 示例代码 有关详细信息，以及 **recv** 函数的另一个示例，请参阅 [入门 With Winsock](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/getting-started-with-winsock)。  **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** | 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)  [WSARecv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsarecv)  [WSARecvEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/mswsock/nf-mswsock-wsarecvex)  [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)  [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) |

## select函数的语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **select** 函数确定一个或多个套接字的状态，并在必要时等待执行同步 I/O。 语法 C++  int WSAAPI select(  [in] int nfds,  [in, out] fd\_set \*readfds,  [in, out] fd\_set \*writefds,  [in, out] fd\_set \*exceptfds,  [in] const timeval \*timeout  ); 参数 [in] nfds  已忽略。 包含 *nfds* 参数只是为了与 Berkeley 套接字兼容。  [in, out] readfds  一个可选指针，指向要检查的一组套接字的可读性。  [in, out] writefds  指向要检查可写性的一组套接字的可选指针。  [in, out] exceptfds  指向要检查错误的一组套接字的可选指针。  [in] timeout  **选择**等待的最长时间，以 [TIMEVAL](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-timeval) 结构的形式提供。 将阻止操作的 *超时* 参数设置为 **null** 。 返回值 **select** 函数返回[fd\_set](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/ns-winsock2-fd_set)结构中已就绪并包含的套接字句柄总数;如果时间限制过期，则返回 0;如果发生错误，则返回SOCKET\_ERROR。 如果返回值SOCKET\_ERROR，则 [WSAGetLastError](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsagetlasterror) 可用于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) 。 | | [**WSAEFAULT**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | Windows 套接字实现无法为其内部操作分配所需的资源，或者 *readfds*、 *writefds*、 *exceptfds* 或 *timeval* 参数不是用户地址空间的一部分。 | | [**WSAENETDOWN**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 网络子系统失败。 | | [**WSAEINVAL**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | *超时*值无效，或者三个描述符参数全部为 **null**。 | | [**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 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。 | | [**WSAENOTSOCK**](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2) | 其中一个描述符集包含不是套接字的条目。 |  注解 **select** 函数用于确定一个或多个套接字的状态。 对于每个套接字，调用方可以请求有关读取、写入或错误状态的信息。 请求给定状态的套接字集由 [fd\_set](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/ns-winsock2-fd_set) 结构指示。 **包含在 fd\_set** 结构中的套接字必须与单个服务提供程序相关联。 出于此限制的目的，如果描述其协议 [的WSAPROTOCOL\_INFO](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/ns-winsock2-wsaprotocol_infoa) 结构具有相同的 providerId 值，则将套接字视为来自同 *一服务* 提供程序。 返回后，将更新结构以反映满足指定条件的这些套接字的子集。 **select** 函数返回满足条件的套接字数。 提供了一组宏用于操作 **fd\_set** 结构。 这些宏与伯克利软件中使用的宏兼容，但基础表示形式完全不同。  参数 *readfds* 标识要检查的可读性套接字。 如果套接字当前处于 [侦听](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-listen) 状态，则如果已收到传入的连接请求，则它将标记为可读，以便保证 [接受](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept) 在不阻塞的情况下完成。 对于其他套接字，可读性意味着排队的数据可用于读取，以便保证不阻止对 [recv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recv)、[WSARecv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsarecv)[、WSARecvFrom](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsarecvfrom) 或 [recvfrom](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recvfrom) 的调用。  对于面向连接的套接字，可读性还可以指示已从对等方收到关闭套接字的请求。 如果虚拟线路正常关闭，并且接收了所有数据，则 [recv](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-recv) 将立即返回且读取零个字节。 如果重置了虚拟线路，则 **recv** 将立即完成，错误代码为 [WSAECONNRESET](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/windows-sockets-error-codes-2)。 如果已启用套接字选项SO\_OOBINLINE，将检查 OOB 数据是否存在 (请参阅 [setsockopt](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-setsockopt)) 。  参数 *writefds* 标识要检查的可写性套接字。 如果套接字正在处理 (非阻止) [的连接](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect) 调用，则如果连接建立成功完成，则套接字是可写的。 如果套接字未处理 **连接** 调用，可写性意味着 [可以保证 send](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-send)、 [sendto](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-sendto) 或 [WSASendto](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsasendto) 成功。 但是，如果 *len* 参数超过可用的传出系统缓冲区空间量，则它们可以在阻止套接字上阻塞。 未指定可以假定这些保证有效多长时间，尤其是在多线程环境中。  参数 *exceptfds* 标识要检查的套接字是否存在 OOB 数据或任何异常错误条件。  **注意** 仅当选项SO\_OOBINLINE为 **FALSE** 时，才会以这种方式报告带外数据。 如果套接字正在处理 (非阻止) [**的连接**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect) 调用，则连接尝试失败的 *指示 (应用程序* 必须调用 [**getsockopt**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-getsockopt) SO\_ERROR来确定错误值，以描述) 失败的原因。 本文档不定义将包含哪些其他错误。    任何两个参数（ *readfds*、 *writefds* 或 *exceptfds*）都可以指定为 **null**。 必须至少有一个非 **null**，并且任何非 **null** 描述符集必须至少包含一个套接字的句柄。  总之，如果出现如下情况， **则 select** 返回时，将在特定集中标识套接字：  *readfds*：   * 如果已调用 [侦听](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-listen) 且连接挂起， [则接受](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept) 将成功。 * 如果) 启用SO\_OOBINLINE，则数据可用于读取 (包括 OOB 数据。 * 连接已关闭/重置/终止。   *writefds*：   * 如果处理 [连接](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect) 调用 (非阻止) ，则表示连接已成功。 * 可以发送数据。   *exceptfds*：   * 如果处理 **连接** 调用 (非阻止) ，则连接尝试失败。 * 仅当) 禁用SO\_OOBINLINE时，OOB 数据才可用于读取 (。   头文件 Winsock2.h 中定义了四个宏，用于操作和检查描述符集。 变量FD\_SETSIZE确定集中描述符的最大数目。 (FD\_SETSIZE 的默认值为 64，可通过先将FD\_SETSIZE定义为另一个值进行修改，然后再包括 Winsock2.h.) 内部， [fd\_set](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/ns-winsock2-fd_set) 结构中的套接字句柄不如在 Berkeley Unix 中表示为位标志。 其数据表示是不透明的。 使用这些宏将保持不同套接字环境之间的软件可移植性。 用于操作和检查**fd\_set**内容的宏包括：   * *FD\_ZERO (\*set)*- 初始化设置为空集。 在使用之前，应始终清除集。 * *FD\_CLR (，\*set)*- 从集中删除套接字。 * *FD\_ISSET (，\*set)*- 检查是否为 set 的成员，如果为 ，则返回 TRUE。 * *FD\_SET (，\*set)*- 添加要设置的套接字。   参数 *超时* 控制 **选择** 需要多长时间才能完成。 如果 *超时* 是 **空** 指针， **则 select** 将无限期阻止，直到至少有一个描述符满足指定条件。 否则， *超时* 将指向 [TIMEVAL](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-timeval) 结构，该结构指定 **选择** 在返回之前应等待的最长时间。 **当 select** 返回时，不会更改 **TIMEVAL** 结构的内容。 如果 **TIMEVAL** 初始化为 {0， 0}， **则 select** 将立即返回;用于轮询所选套接字的状态。 如果 **select** 立即返回，则 **select** 调用被视为非阻止调用，且非阻止调用的标准假设适用。 例如，不会调用阻塞挂钩，并且 Windows 套接字不会生成。  **注意select** 函数对向 [**WSAAsyncSelect 或 WSAEventSelect**](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsaasyncselect) 注册的套接字事件的持久性没有影响。    **注意**发出阻止 Winsock 调用时，例如 **select***，超时参数*设置为 **NULL** 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。    **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** | winsock2.h | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 [TIMEVAL](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/ns-winsock-timeval)  [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)  [accept](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-accept)  [connect](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-connect)  [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)  [send](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-send) |

## inet\_addr函数用法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **inet\_addr** 函数将包含 IPv4 点十进制地址的字符串转换为[IN\_ADDR](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/ns-winsock2-in_addr)结构的正确地址。 语法 C++复制  unsigned long WSAAPI inet\_addr(  const char \*cp  ); 参数 cp  TBD 返回值 如果未发生错误， **inet\_addr** 函数将返回一个无符号长值，其中包含给定的 Internet 地址的合适二进制表示形式。  如果 *cp* 参数中的字符串不包含合法的 Internet 地址，例如，如果“a.b.c.d”地址的一部分超过 255，则 **inet\_addr** 返回 **INADDR\_NONE**的值。  在 Windows Server 2003 及更高版本中，如果 *cp* 参数中的字符串为空字符串，**则inet\_addr返回值INADDR\_NONE**。 如果在 *cp* 参数中传递 **NULL**，则**inet\_addr**返回**值INADDR\_NONE**。  在 Windows XP 及更早版本中，如果 *cp* 参数中的字符串为空字符串，**则inet\_addr返回值INADDR\_ANY**。 如果在 *cp* 参数中传递 **NULL**，则**inet\_addr**返回**值INADDR\_NONE**。 注解 **inet\_addr** 函数解释 *cp* 参数指定的字符串。 此字符串表示以 Internet 标准“.”表示的数字 Internet 地址。 符号。 返回的值是一个适合用作 Internet 地址的数字。 所有 Internet 地址都按 IP 的网络顺序返回 (字节从左到右) 。 如果将“” (空格) 传入 **inet\_addr** 函数， **inet\_addr** 返回零。  在 Windows Vista 及更高版本上， [RtlIpv4StringToAddress](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv4stringtoaddressa) 函数可用于将 IPv4 地址的字符串表示形式转换为表示为 [IN\_ADDR](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/ns-winsock2-in_addr) 结构的二进制 IPv4 地址。 在 Windows Vista 及更高版本上， [RtlIpv6StringToAddress](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv6stringtoaddressa) 函数可用于将 IPv6 地址的字符串表示形式转换为表示为 **IN6\_ADDR** 结构的二进制 IPv6 地址。 Internet 地址 使用“.”指定的值 表示法采用以下形式之一：  a.b.c.d a.b.c.a.b  当指定四个部分时，每个部分都解释为数据的一个字节，并从左到右分配给 Internet 地址的 4 个字节。 当 Internet 地址在 Intel 体系结构中被视为 32 位整数数量时，上面提到的字节将显示为“d.c.b.a'”。 也就是说，Intel 处理器上的字节从右到左排序。  构成“.”表示法中地址的部分可以是 C 语言中指定的十进制、八进制或十六进制。 以“0x”或“0X”开头的数字表示十六进制。 以“0”开头的数字表示八进制。 所有其他数字都解释为十进制。  展开表   |  |  | | --- | --- | | **Internet 地址值** | **含义** | | "4.3.2.16" | 小数 | | "004.003.002.020" | 八进制 | | “0x4.0x3.0x2.0x10” | 十六进制 | | “4.003.002.0x10” | 组合 |     **inet\_addr** 函数支持 *cp* 参数中传递的字符串的十进制、八进制、十六进制和混合表示法。  **注意** 以下表示法仅供伯克利软件使用，而 Internet 上没有其他位置使用。 为了与 Berkeley 软件兼容， **inet\_addr** 函数还支持下面指定的其他表示法。    指定由三部分构成的地址时，最后一部分将解释为 16 位数量，并放置在网络地址最右边的 2 个字节中。 这使得三部分地址格式便于将 B 类网络地址指定为“128.net.host”  指定由两部分构成的地址时，最后一部分将解释为 24 位数量，并放置在网络地址最右边的 3 个字节中。 这使得由两部分构成的地址格式便于将类 A 网络地址指定为“net.host”。  如果只提供一个部分，则该值直接存储在网络地址中，而无需重新排列任何字节。  **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。 示例 下面的代码示例演示如何使用 **inet\_addr** 函数。  C++复制  #define WIN32\_LEAN\_AND\_MEAN  #include <winsock2.h>  #include <stdio.h>  #include <windows.h>  // need link with Ws2\_32.lib  #pragma comment(lib, "Ws2\_32.lib")  int \_\_cdecl main(int argc, char \*\*argv)  {  //-----------------------------------------  // Declare and initialize variables  WSADATA wsaData;  int iResult;  unsigned long ulAddr = INADDR\_NONE;  // Validate the parameters  if (argc != 2) {  printf("usage: %s <IPv4 address>\n", argv[0]);  printf(" inetaddr converts a string containing an\n");  printf(" IPv4 address in one of the supported formats\n");  printf(" to a unsigned long representing an IN\_ADDR\n");  printf(" %s 192.168.16.34\n", argv[0]);  return 1;  }  // Initialize Winsock  iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);  if (iResult != 0) {  printf("WSAStartup failed: %d\n", iResult);  return 1;  }  //--------------------------------  // Call inet\_addr(). If the call succeeds,  // the result variable will hold a IN\_ADDR  ulAddr = inet\_addr(argv[1]);  if ( ulAddr == INADDR\_NONE ) {  printf("inet\_addr failed and returned INADDR\_NONE\n");  WSACleanup();  return 1;  }    if (ulAddr == INADDR\_ANY) {  printf("inet\_addr failed and returned INADDR\_ANY\n");  WSACleanup();  return 1;  }  printf("inet\_addr returned success\n");    // Here we could implement code to retrieve each address and  // print out the hex bytes  // for(i=0, ptr= (Char\*) &ulAddr; i < 4; i++, ptr++) {  WSACleanup();  return 0;  } 要求  |  |  | | --- | --- | | **标头** | winsock2.h (包括 Winsock2.h、Winsock.h) | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 **IN6\_ADDR**  [IN\_ADDR](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/ns-winsock2-in_addr)  [InetNtop](https://learn.microsoft.com/zh-cn/windows/desktop/api/ws2tcpip/nf-ws2tcpip-inetntopw)  [RtlIpv4AddressToString](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv4addresstostringa)  [RtlIpv4AddressToStringEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv4addresstostringexw)  [RtlIpv4StringToAddress](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv4stringtoaddressa)  [RtlIpv4StringToAddressEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv4stringtoaddressexw)  [RtlIpv6AddressToString](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv6addresstostringa)  [RtlIpv6AddressToStringEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv6addresstostringexw)  [RtlIpv6StringToAddress](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv6stringtoaddressa)  [RtlIpv6StringToAddressEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv6stringtoaddressexw)  [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)  [inet\_ntoa](https://learn.microsoft.com/zh-cn/windows/desktop/api/wsipv6ok/nf-wsipv6ok-inet_ntoa) |

## inet\_ntoa宏

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **inet\_ntoa** 函数将 internet 网络地址) (Ipv4 转换为 Internet 标准点十进制格式的 ASCII 字符串。 语法 C++复制  void inet\_ntoa(  [in] a  ); 参数 [in] a  表示 Internet 主机地址 [的in\_addr](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/ns-winsock2-in_addr) 结构。 返回值 无 备注 **inet\_ntoa**函数采用*由 in* 参数指定的 Internet 地址结构，并返回一个**以 NULL** 结尾的 ASCII 字符串，该字符串表示“.”中的地址， (点) 表示法，如“192.168.16.0”中所示，这是以点数十进制表示法表示的 IPv4 地址的示例。 **inet\_ntoa**返回的字符串驻留在 Windows 套接字分配的内存中。 应用程序不应对内存的分配方式做出任何假设。 保证返回的字符串仅在同一线程中进行下一次 Windows 套接字函数调用之前有效。 因此，应在执行另一个 Windows 套接字调用之前复制数据。  [WSAAddressToString](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaaddresstostringa) 函数可用于将包含 IPv4 地址[的 sockaddr](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2) 结构转换为 Internet 标准点十进制表示法中 IPv4 地址的字符串表示形式。 **WSAAddressToString** 函数的优点是它支持 IPv4 和 IPv6 地址。 **WSAAddressToString** 函数的另一个优点是此函数同时存在 ASCII 和 Unicode 版本。  在 Windows Vista 及更高版本上， [RtlIpv4AddressToString](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv4addresstostringa) 函数可用于将表示为 [IN\_ADDR](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/ns-winsock2-in_addr) 结构的 IPv4 地址转换为 Internet 标准点十进制表示法中 IPv4 地址的字符串表示形式。 在 Windows Vista 及更高版本上， [可以使用 RtlIpv6AddressToString](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv6addresstostringa) 函数将表示为 **IN6\_ADDR** 结构的 IPv6 地址转换为 IPv6 地址的字符串表示形式。  **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** | wsipv6ok.h (包括 Winsock2.h、Winsock.h) | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 **IN6\_ADDR**  [InetNtop](https://learn.microsoft.com/zh-cn/windows/desktop/api/ws2tcpip/nf-ws2tcpip-inetntopw)  [RtlIpv4AddressToString](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv4addresstostringa)  [RtlIpv4AddressToStringEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv4addresstostringexw)  [RtlIpv4StringToAddress](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv4stringtoaddressa)  [RtlIpv4StringToAddressEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv4stringtoaddressexw)  [RtlIpv6AddressToString](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv6addresstostringa)  [RtlIpv6AddressToStringEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv6addresstostringexw)  [RtlIpv6StringToAddress](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv6stringtoaddressa)  [RtlIpv6StringToAddressEx](https://learn.microsoft.com/zh-cn/windows/desktop/api/ip2string/nf-ip2string-rtlipv6stringtoaddressexw)  [SOCKADDR](https://learn.microsoft.com/zh-cn/windows/desktop/WinSock/sockaddr-2)  [WSAAddressToString](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsaaddresstostringa)  [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)  [in\_addr](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/ns-winsock2-in_addr)  [inet\_addr](https://learn.microsoft.com/zh-cn/windows/desktop/api/wsipv6ok/nf-wsipv6ok-inet_addr) |

## htons函数的语法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **htons** 函数将**主机u\_short**转换为 TCP/IP 网络字节顺序 (这是大端) 。 语法 C++复制  u\_short WSAAPI htons(  [in] u\_short hostshort  ); 参数 [in] hostshort  主机字节顺序为 16 位数字。 返回值 **htons** 函数以 TCP/IP 网络字节顺序返回值。 注解 **htons** 函数采用主机字节顺序为 16 位数字，并返回 TCP/IP 网络 (AF\_INET或AF\_INET6地址系列) 中使用的 16 位数字。  **htons** 函数可用于将主机字节顺序中的 IP 端口号转换为网络字节顺序的 IP 端口号。  **htons** 函数不要求 Winsock DLL 之前已通过对 [WSAStartup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsastartup) 函数的成功调用加载。  **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** | winsock2.h (包括 Winsock2.h) | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 [InetNtop](https://learn.microsoft.com/zh-cn/windows/desktop/api/ws2tcpip/nf-ws2tcpip-inetntopw)  [InetPton](https://learn.microsoft.com/zh-cn/windows/desktop/api/ws2tcpip/nf-ws2tcpip-inetptonw)  [WSAHtonl](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsahtonl)  [WSAHtons](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsahtons)  [WSANtohl](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsantohl)  [WSANtohs](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsantohs)  [htond](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-htond)  [htonf](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-htonf)  [htonl](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-htonl)  [htonll](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-htonll)  [inet\_addr](https://learn.microsoft.com/zh-cn/windows/desktop/api/wsipv6ok/nf-wsipv6ok-inet_addr)  [inet\_ntoa](https://learn.microsoft.com/zh-cn/windows/desktop/api/wsipv6ok/nf-wsipv6ok-inet_ntoa)  [ntohd](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-ntohd)  [ntohf](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-ntohf)  [ntohl](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-ntohl)  [ntohll](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-ntohll)  [ntohs](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-ntohs) |

## htonl 函数语法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **htonl** 函数将**u\_long**从主机转换为 TCP/IP 网络字节顺序 (这是 big-endian) 。 语法 C++  u\_long WSAAPI htonl(  [in] u\_long hostlong  ); 参数 [in] hostlong  主机字节顺序中的 32 位数字。 返回值 **htonl** 函数以 TCP/IP 的网络字节顺序返回值。 注解 **htonl** 函数采用主机字节顺序的 32 位数字，并在 TCP/IP 网络中使用的网络字节顺序中返回一个 32 位数字， (AF\_INET或AF\_INET6地址系列) 。  **htonl** 函数可用于将主机字节顺序中的 IPv4 地址转换为网络字节顺序中的 IPv4 地址。 此函数不会执行任何检查来确定 *hostlong* 参数是否为有效的 IPv4 地址。  **htonl** 函数不要求之前已通过成功调用 [WSAStartup](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-wsastartup) 函数加载 Winsock DLL。  **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** | winsock2.h (包括 Winsock2.h) | | **Library** | Ws2\_32.lib | | **DLL** | Ws2\_32.dll |  另请参阅 [InetNtop](https://learn.microsoft.com/zh-cn/windows/desktop/api/ws2tcpip/nf-ws2tcpip-inetntopw)  [InetPton](https://learn.microsoft.com/zh-cn/windows/desktop/api/ws2tcpip/nf-ws2tcpip-inetptonw)  [WSAHtonl](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsahtonl)  [WSAHtons](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsahtons)  [WSANtohl](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsantohl)  [WSANtohs](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-wsantohs)  [htond](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-htond)  [htonf](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-htonf)  [htonll](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-htonll)  [htons](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-htons)  [inet\_addr](https://learn.microsoft.com/zh-cn/windows/desktop/api/wsipv6ok/nf-wsipv6ok-inet_addr)  [inet\_ntoa](https://learn.microsoft.com/zh-cn/windows/desktop/api/wsipv6ok/nf-wsipv6ok-inet_ntoa)  [ntohd](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-ntohd)  [ntohf](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-ntohf)  [ntohll](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock2/nf-winsock2-ntohll)  [ntohs](https://learn.microsoft.com/zh-cn/windows/desktop/api/winsock/nf-winsock-ntohs) |

# 演练

# 先来做TCP服务器

## 1.新建一个文件夹,取名:Lesson74-tcp,然后在里面新建一个常规空项目取名:tcp-server,然后新建一个cpp文件,也叫做tcpserver.cpp,写好骨架代码

|  |
| --- |
|  |

## 2.然后需要导入socket头文件和lib文件

|  |
| --- |
|  |

## 3.最简单的tcp socket服务器代码如下,这里只能连接一次,实际开发中需要循环等待用户连接,需要一直等待

## tcpserver.cpp

|  |
| --- |
| #pragma comment(lib,"Ws2\_32.lib")  #include<stdio.h>  #include<stdlib.h>  #include <Winsock2.h>  void main()  {  //变量定义  WSADATA wsaData;  SOCKET sListen;//监听套接字  SOCKET sConn;//连接套接字  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 claddrlen;//客户端地址长度  int port = 5000;//定义服务器端口  int ret;  char buffer[1024];//接收数据的缓冲区  //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;  }  //2.创建监听套接字  sListen = socket(AF\_INET,SOCK\_STREAM,IPPROTO\_TCP);  if(sListen == INVALID\_SOCKET)  {  printf("创建监听套接字失败：%d\n",WSAGetLastError());  WSACleanup();//创建套接字失败了,就需要清理套接字库  return;  }  //2.2设置服务端的地址和端口,需要设置为网络字节序  serverAddr.sin\_family = AF\_INET; //协议簇为internet协议  serverAddr.sin\_port = htons(port);//端口转换  serverAddr.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);//设置服务器接受来自任何地址的连接  //3.绑定套接字和服务端的地址  if(SOCKET\_ERROR == bind(sListen,(SOCKADDR\*)&serverAddr,sizeof(serverAddr)))  {  printf("绑定端口失败：%d\n",WSAGetLastError());  closesocket(sListen);//绑定失败需要关闭套接字  WSACleanup();//绑定失败了,也需要清理套接字库  return;  }  //4.监听  if(SOCKET\_ERROR == listen(sListen,10))  {  printf("监听失败：%d\n",WSAGetLastError());  closesocket(sListen);//监听失败需要关闭套接字  WSACleanup();//监听失败了,也需要清理套接字库  return;  }  printf("端口:%d,等待客户端连接...\n",port);  //5.接受客户端连接  claddrlen = sizeof(SOCKADDR\_IN);  sConn = accept(sListen,(SOCKADDR\*)&clientAddr,&claddrlen);  if(INVALID\_SOCKET ==sConn )  {  printf("监听失败：%d\n",WSAGetLastError());  closesocket(sListen);//连接失败需要关闭套接字  closesocket(sConn);//连接失败需要关闭连接套接字  WSACleanup();//连接失败了,也需要清理套接字库  return;  }  printf("客户端%s:%d成功连接...\n",inet\_ntoa(clientAddr.sin\_addr),ntohs(clientAddr.sin\_port));  closesocket(sListen);//这里只做一次监听,所以这里关闭了服务器套接字,在实际开发中是使用循环来持续通信的.  //6.接收数据  printf("接收数据中...\n");  if((ret = recv(sConn,buffer,sizeof(buffer),0))== SOCKET\_ERROR)  {  printf("接收数据失败：%d\n",WSAGetLastError());  closesocket(sConn);//接收数据失败需要关闭连接套接字  WSACleanup();//接收数据失败了,也需要清理套接字库  return;  }  buffer[ret] = '\0';//给接收到的数据的末尾添加结束符  printf("客户信息: %s\n",buffer);  printf("现在关闭连接\n");  //7.关闭客户连接  closesocket(sConn);//结束通信  WSACleanup();//不要忘记了清理工作  system("pause");  } |

# 然后我们来做TCP客户端

## 0把tcp-server解决方案改名为tcp-server+tcp-client,把文件夹改名Lesson74-75-tcp

## 1.在tcp-server+tcp-client解决方案上面点击右键->添加->新建项目,也是一个常规项目,取名tcp-client,然后新建一个cpp文件取名tcpclient.cpp

|  |
| --- |
|  |

## 2.然后Tcp客户端的最简单程序如下tcpclient.cpp

|  |
| --- |
| #pragma comment(lib,"Ws2\_32.lib")  #include<stdio.h>  #include<stdlib.h>  #include <Winsock2.h>  void main(int argc,char \*argv[])  {  if(argc < 2)//格式不对就返回.  {  printf("需要提供服务器的ip地址\n");  printf("Usage: tcp\_client.exe x.x.x.x\n");  return;  }  //变量定义  WSADATA wsaData;  SOCKET s;//连接套接字  SOCKADDR\_IN serverAddr;//保存服务端地址的变量  int port = 5000;  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;  }  //2.创建套接字  s = socket(AF\_INET,SOCK\_STREAM,IPPROTO\_TCP);  if(s == INVALID\_SOCKET)  {  printf("创建监听套接字失败：%d\n",WSAGetLastError());  WSACleanup();//创建套接字失败了,就需要清理套接字库  return;  }  //2.2设置服务端的地址和端口,需要设置为网络字节序  serverAddr.sin\_family = AF\_INET; //协议簇为internet协议  serverAddr.sin\_port = htons(port);//端口转换  serverAddr.sin\_addr.S\_un.S\_addr = inet\_addr(argv[1]);  //3.连接服务器  printf("正在连接服务器:%s:%d...\n",inet\_ntoa(serverAddr.sin\_addr),htons(serverAddr.sin\_port));  if((ret = connect(s,(SOCKADDR\*)&serverAddr,sizeof(serverAddr)))==SOCKET\_ERROR)  {  printf("连接失败：%d\n",WSAGetLastError());  closesocket(s);//连接失败需要关闭连接套接字  WSACleanup();//连接失败了,也需要清理套接字库  return;  }  //4.发送数据  printf("连接成功,开始发送数据\n");  if((ret = send(s,"hello server",12,0))==SOCKET\_ERROR)  {  printf("发送数据失败%d\n",WSAGetLastError());  closesocket(s);//失败发送数据需要关闭连接套接字  WSACleanup();//失败发送数据了,也需要清理套接字库  return;  }  //5.关闭连接  printf("发送数据成功\n");  printf("现在关闭套接字\n");  closesocket(s);  WSACleanup();  system("pause");  } |

## 注意:客户端一定要在命令行窗口里面启动,不要双击运行,服务器可以双击运行

### 运行两个程序,效果如下

|  |
| --- |
|  |

# 注意:这个课程有些章节,如:这里的内容太过于简单,我们入门后需要多上网找找一些资料来加深学习

# 这一节的信息到此为止,完整代码声明已经给出