# UDT函数接口

## ****accept****

UDTSOCKET accept(  
  UDTSOCKET u,

  struct sockaddr\* addr,  
  int\* addrlen  
);

一旦UDT套接字处于侦听状态，它将接受新的连接，并将挂起的连接维护在队列中。接受调用检索队列中的第一个连接，将其从队列中删除，并返回所关联的套接字描述符。  
如果在调用accept时队列中没有连接，则阻塞套接字将等待直到新连接建立，而非阻塞套接字将立即返回错误。  
接受的套接字将从侦听套接字继承所有的属性。

## ****bind****

int bind(  
  UDTSOCKET u,  
  struct sockaddr\* name,  
  int\* namelen  
);   
  
int bind(  
#ifndef WIN32  
  int udpsock  
#else  
  SOCKET udpsock  
#endif  
);

bind方法通常是为UDT套接字分配本地地址，包括IP地址和端口号。如果使用INADDR\_ANY，一旦设置了UDT连接，就会使用正确的IP地址。如果端口使用0，则会使用随机可用的端口号。getsockname方法可用于检索此端口号。  
bind的第二种形式允许UDT直接绑定到现有的UDP套接字上。在某些情况下，这是防火墙穿越的用法：1）创建UDP套接字，从指定服务器学习到其地址，不需要关闭UDP套接字并再次打开同一个地址上的UDT套接字; 2）对于某些防火墙，特别是本地系统的某些防火墙，端口映射可能会更改，或者当一个UDP套接字被关闭并重新打开时，可能会关闭“洞”，因此需要在UDT中直接使用UDP套接字。  
请谨慎使用第二种绑定形式，因为它违反了有关代码稳健性的某些编程规则。一旦将UDP套接字描述符传递给UDT，则不得再次使用。除非清楚了解相关系统的工作原理，否则请勿使用此功能。  
在所有情况下，bind调用是必需的，除了监听套接字。如果未调用bind，UDT将在连接建立时自动将套接字绑定到随机可用的地址。  
默认情况下，UDT允许重用现有的UDP端口用于新的UDT套接字，除非UDT\_REUSEADDR设置为false。当UDT\_REUSEADDR为false时，UDT将为此UDT套接字创建一个独占的UDP端口。绑定前必须调用UDT\_REUSEADDR。要重用现有的UDT/UDP端口，新的UDT套接字必须显式绑定到端口。如果端口已被UDT套接字使用，UDT\_REUSEADDR为false，则新绑定将返回错误。如果0作为端口号传递，bind将始终创建一个新端口，无论UDT\_REUSEADDR设置什么值。

## ****cleanup****

int cleanup(

);

cleanup方法释放UDT库。所有剩余的打开连接将关闭。后台垃圾回收关闭。但是，如果没有调用任何启动，此方法将不会执行任何操作，或者这是一个重复的调用。  
该方法必须在应用程序退出之前或UDT DLL释放之前调用，否则可能会发生内存泄漏。

## ****close****

int close(  
  UDTSOCKET u  
);

close方法正常关闭UDT连接，并释放与UDT套接字相关联的所有相关数据结构。如果没有与套接字关联的连接，则close只需释放套接字资源。  
在阻塞套接字上，如果UDT\_LINGER不为零，则close操作将等待发送缓冲区中的所有数据发送出去，或等待时间超过UDT\_LINGER设置的到期时间。然而，如果UDT\_SYNSND设置为false（即非阻塞发送），则close将立即返回，并且全部遗留数据将在后台发送，直到连续计时器期满。  
关闭UDT套接字将向对端发送关闭消息，以使对等端口也将被关闭。 这是一个尽力而为的消息。如果消息没有成功传递，对等端也将在超时后关闭。 在UDT中，不支持shutdown。  
不再使用的套接字都应该close释放

## ****connect****

int connect(  
  UDTSOCKET u,  
  const struct sockaddr\* name,  
  int\* namelen  
);

UDT是面向连接的，用于SOCK\_STREAM和SOCK\_DGRAM两种模式。建立连接时必须调用connect。 name参数是服务器或对端的地址。在常规（默认）客户端/服务器模式下，服务器端必须调用bind和listen。在会合模式下，双方必须在（大约）同时调用绑定并相互连接。会话连接不能用于同一UDP端口对上的多个连接，在这种情况下，UDT\_REUSEADDR可能设置为false。  
UDT连接需要至少一个往返行程才能完成。如果应用程序经常连接和断开同一地址，这可能会成为瓶颈。  
当UDT\_RCVSYN设置为false时，连接调用将立即返回，并在后台执行实际的连接操作。应用程序可以使用epoll等待连接完成。  
当连接失败时，仍可以使用UDT套接字再次连接。但是，如果套接字未绑定，则可能会隐式绑定，如上所述，即使连接失败。另外，在连接调用失败的情况下，UDT套接字不会被自动释放，如果不再需要套接字（例如重新连接），应用程序有责任关闭套接字。

## ****epoll****

#ifndef WIN32  
   typedef int SYSSOCKET;  
#else  
   typedef SOCKET SYSSOCKET;  
#endif  
int epoll\_create();  
int epoll\_add\_usock(const int eid, const UDTSOCKET usock, const int\* events = NULL);  
int epoll\_add\_ssock(const int eid, const UDTSOCKET ssock, const int\* events = NULL);  
int epoll\_remove\_usock(const int eid, const UDTSOCKET usock);  
int epoll\_remove\_ssock(const int eid, const UDTSOCKET ssock);  
int epoll\_wait(const int eid, std::set<UDTSOCKET>\* readfds, std::set<UDTSOCKET>\* writefds, int64\_t msTimeOut, std::set<SYSSOCKET>\* lrfds = NULL, std::set<SYSSOCKET>\* wrfds = NULL);  
int epoll\_release(const int eid);

epoll提供可扩展和高效的方式来等待UDT套接字IO事件。当应用程序需要等待大量的套接字时，应该使用select而不是selectEx。此外，epoll还提供同时等待系统套接字，当应用程序同时使用UDT和TCP/UDP时，这可以很方便。  
应用程序应该使epoll\_create创建一个epoll ID，并使用epoll\_add\_usock/ssock和epoll\_remove\_usock/ssock添加/删除套接字。如果套接字已经在epoll集中，如果再次添加，它将被忽略。添加无效或关闭的套接字会导致错误。但是，它们将被忽略，而在删除时不会返回任何错误。  
可以创建多个epoll实体，只要系统资源允许，就不会有上限。 UDT套接字的数量也没有硬限制。 UDT::epoll支持的数字系统描述符取决于平台。  
对于Linux上的系统套接字，开发人员可以选择从EPOLLIN（读），EPOLLOUT（写）和EPOLLERR（异常）中观察各种事件。当使用epoll\_remove\_ssock时，如果套接字正在等待多个事件，则只会删除在事件中指定的事件。事件可以是以下任何值的组合（使用“|”操作）。

enum EPOLLOpt  
{  
    UDT\_EPOLL\_IN = 0x1,  
    UDT\_EPOLL\_OUT = 0x4,  
    UDT\_EPOLL\_ERR = 0x8  
};

对于所有其他情况，events参数将被忽略，所有事件都将被监听。  
请注意，异常被分类为写事件，因此当应用程序选择写入此套接字时，它会检测到异常。  
最后，对于epoll\_wait，超时值为负值时将使该函数等待直到事件发生。 如果超时值为0，则该函数将立即返回与IO事件关联的任何套接字。如果任何事件发生之前发生超时，该函数返回0。

## ****getlasterror****

ERRORINFO& getlasterror(  
);

Getlasterror获取调用此方法的线程中的最后一个UDT错误。错误信息存储在线程特定存储中。

## ****getpeername****

int getpeername(  
  UDTSOCKET u,  
  struct sockaddr\* name,  
  int\* namelen  
);

getpeername用于获取连接的对端地址。getpeername调用时连接必须处于已连接状态。 namelen提供name参数的leangth，空间必须足以容纳地址信息。返回时，namelen包含结果的长度。

## ****getsockname****

int getsockname(  
  UDTSOCKET u,  
  struct sockaddr\* name,  
  int\* namelen  
);

getsockname获取与套接字关联的本地地址。UDT套接字必须明确绑定（通过bind）或隐式绑定（通过connect）绑定，否则该方法将失败，因为没有有效的地址绑定到套接字。  
如果在显式绑定之后但在连接之前调用getsockname，返回的IP地址将完全是用于绑定的IP地址，如果使用ADDR\_ANY，则可能为0.0.0.0。如果在连接后调用getsockname，返回的IP地址将是对端套接字看到的地址。在存在代理（例如NAT）的情况下，返回的IP地址将是代理的转换地址，而不是本地地址。如果没有代理，返回的IP地址将是本地地址。在任一情况下，端口号是本地的（即不是转换的代理端口）。  
因为UDP是无连接的，所以在UDP端口上使用getsockname几乎总是返回0.0.0.0作为IP地址（除非绑定到显式IP）。作为面向连接的协议，如果没有代理转换存在，则UDT将通过getsockname返回有意义的IP地址。  
UDT还没有多重支持。当有多个本地地址并且其中一个可以路由到目的地址时，由于多路径效应，UDT可能无法正常运行。在这种情况下，UDT套接字必须明确绑定到其中一个本地地址。

## ****Getsockopt/setsockopt****

int getsockopt(  
  UDTSOCKET u,  
  int level,  
  SOCKOPT optname,  
  char\* optval,  
  int\* optlen  
);   
  
int setsockopt(  
  UDTSOCKET u,  
  int level,  
  SOCKOPT optname,  
  const char\* optval,  
  int optlen  
);

setsockopt方法设置具有optval值的UDT选项optName。检查optlen的参数以验证选项值的合法性。不是所有的选项都可以在UDT的任何状态下设置。 实际上，大多数选项必须在绑定或连接被调用之前进行设置，因为这些值在创建UDT连接时初始化某些数据结构是必要的。  
getsockopt方法读取当前选项值。该值写入由optval指向的缓冲区，并在optlen中返回长度。