

第三部分 远程访问服务

迄今为止，本书已介绍了可在 Microsoft Windows 操作系统中使用的所有网络 API 函数。利用这些函数，我们的应用程序可通过网络，建立与其他程序的通信联系。在那些讨论中，我们在很大程度上将重点放在七层 OSI 模型的应用层和表示层上面。在这个过程中，我们并没有就具体的某种网络协议进行剖析，大多数讨论都站在一个“与协议无关”的角度，解释如何使用函数。

本书的最后这一部分讨论了一种重要的服务，名为“远程访问服务”(Remote Access Service, RAS)，它允许用户从远程地点，将自己的计算机连接到一个本地计算机网络。一旦建立了连接，便可使用本书讲述的所有网络函数，即便你手上的计算机实际连接的是一个远程网络！

第16章 RAS 客户机

微软的所有 Windows 平台中都有 RAS 客户机，它允许我们将自己的计算机与另一个地方的远程计算机（其特色是一个远程访问服务器组件）相连。一般情况下，RAS 客户机利用连接了电话线的一个调制解调器，通过拨号的方式呼叫远程计算机。因此，有时，RAS 客户机也称作“拨号联网（DUN）客户机”。

服务器这方面，必须有一项等候 DUN 连接的服务。RAS 客户机能够和多种类型的远程访问服务器建立通信。RAS 通过使用工业标准分帧协议建立通信。比如这些协议：

点到点协议（PPP）可传输 IP 或 NetBEUI 通信协议。

串行线路网际协议（SLIP）只能传输 IP 通信协议。

异步 NetBEUI（微软的 Windows NT 3.1、微软的 Windows for Workgroups 3.11）只能传输 NetBEUI 通信协议。

这些分帧协议描述了数据是怎样通过 RAS 链接进行传输的，指令 RAS 链接上采用哪些网络通信协议（比如 TCP/IP 或 IPX）通信。如果 RAS 服务器支持前面定义的任何一种分帧协议，RAS 客户机便可以与之建立一个链接。Windows 95、Windows 98、Windows 2000 以及 Windows NT 的特色便是：RAS 服务器组件能支持前面列出的任何一种分帧协议。

RAS 客户机和服务器之间的连接建立之后，网络协议堆栈（与所用的分帧协议有关）就通过这个 RAS 连接，与远程计算机通信，就象通过 LAN 连接的一样。当然，如今，许多调制解调器的数据通信速率明显比直接的 LAN 连接慢。

RAS 服务器收到一次拨号连接时，处理前面列出的其中一种分帧协议之后，便与客户机开始通信。分帧协议一旦确立，RAS 服务器就会尝试对连接用户进行身份验证。本章描述的 RAS API 函数允许 RAS 客户机为这个 RAS 服务器指定用户名、口令和域登录凭据。Windows 2000 或 Windows NT RAS 服务器收到这条信息时，利用 Windows NT 域安全访问控制验证登录

凭据。注意，RAS服务器没有进入客户机登录的 Windows NT域；相反地，它采用客户机凭据验证是否允许用户建立 RAS连接。RAS连接过程和 Windows NT域登录过程不一样。RAS连接成功建立之后，计算机就可登录到 Windows NT域。本章不详细说明整个登录过程。Windows 95和 Windows 98平台上，RAS连接通过电话簿条目中可用的选项，经过验证后，RAS便可自动进入一台机器，登录到一个域，和我们将在本章稍后讲的一样。

RAS依赖“电话应用程序接口”(Telephony Application Programming Interface, TAPI) 设置和控制调制解调器之类的电话通信设备，TAPI控制这些拨号设备的硬件设置。在利用调制解调器设置一个 RAS连接时，TAPI会转向调制解调器，并自 RAS向调制解调器发出拨号信息。这样一来，RAS就会把调制解调器看成简单的 TAPI调制解调器端口，具备拨号并与远程服务器建立电话连接的能力。正如大家稍后将看到的那样，在设置 RAS连接信息时，有的 RAS API函数引用了TAPI调制解调器端口。

本章还将讨论如何通过编程的方式，利用 RAS建立与远程网络的通信。首先为大家介绍建立自己的应用程序时，需要哪些头文件和库文件。接下来介绍关于拨号的基本知识——如何真正建立一个远程连接。然后介绍如何设一个 RAS电话簿，定义有关的 RAS连接的详细通信属性。介绍完有关设置通信的基础知识之后，我们再为大家展示如何管理已建立的连接。

16.1 编译和链接

在开发一个RAS应用程序时，需要用这几个头文件和库文件来建立自己的应用程序：

Ras.h：包含RAS API函数所用的函数原型和数据结构。

Raserror.h：包含RAS API函数失败时所用的预先定义的错误代码。

Rasapi32.lib：所有RAS API函数的库。

Raserror.h列举了相当多预先定义的错误代码。这个文件中，大家将注意到一个错误说明字串，这个字串和 RAS中所用的各错误代码关联在一起。RAS特别突出的是一个名为 RasGetErrorString的函数，它非常有用，允许通过编程来获得与特定 RAS错误代码关联的错误字串。RasGetErrorString的定义如下：

```
DWORD RasGetErrorString(  
    UINT uErrorValue,  
    LPTSTR lpszErrorString,  
    DWORD cBufSize  
);
```

uErrorValue参数取得一个特定的 RAS 错误代码，这个代码是一个 RAS函数返回的。lpszErrorString参数是一个由应用程序提供的缓冲区，将接收一个错误字串，该字串与 uErrorValue参数中的错误代码关联到一起。这时应该有一个较大的缓冲区，以便能够容纳错误字串；否则，该函数就会失败，并返回 ERROR_UNINSUFFICIENT_BUFFER错误。我们建议缓冲区长度至少为256个字符，对时下的任何一个RAS错误字串来说，这个长度都比较恰当的。最后一个参数cBufSize是你为lpszErrorString提供的缓冲区的长度。

16.2 数据结构和平台兼容问题

在编译和建立自己的应用程序时，大家会发现：RAS函数所用的有些数据结构中，根据 WINVER定义的值，分别使用或不用额外的数据字段。但是，Windows CESDK 没有定义

WINVER, 因此, 前面提到的字段不能应用于 Windows CE。RAS数据结构中还有一个dwSize 字段, 必须把它设为你使用的 RAS结构的字节长度。这样做会对使用这些结构的 RAS函数产生影响, 因为对这些 RAS函数而言, 目标是针对某具体平台的。WINVER 标准适用于 Windows 95、Windows 98、Windows 2000和Windows NT平台。

WINVER = 0X400: 指明你的RAS应用程序的目标平台是 Windows 95、Windows 98或没有服务包的 Windows NT 4。

WINVER = 0X401: 指明你的RAS应用程序的目标平台是有服务包的 Windows NT。

WINVER = 0X500: 指明你的RAS应用程序的目标平台是 Windows 2000。

RAS本身没有可在所有平台上运行的一个可执行字段。但通过细致的编程, 利用一个可执行字段, 仍然可能获得所有平台的支持(当然 Windows CE除外)。尽管如此, 我们还是强烈建议大家在编写应用程序时, 应该有一个具体的目标平台。

16.3 DUN 1.3升级和Windows 95

在OSR的第一个版本到OSR 2期间, Windows 95发布了大量的版本。OSR 2是非卖品; 只供OEM厂商安装在它们的产品上。每次发布都对 RAS API的功能进行了沿伸。因此, 我们建议大家最好安装最新的 RAS升级包(DUN 1.3)以进一步发挥RAS的潜力。要获得 Windows 95的DUN 1.3升级版, 访问<http://www.microsoft.com/support>。本章的前提是你的机器上至少已经安装了DUN 1.3升级版; 至于以前的DUN版本中的RAS, 我们将不讨论。

16.4 RASDIAL

RAS客户机应用程序准备和一个远程计算机建立通信时, 必须调用 RasDial函数。RasDial函数相当复杂, 要提供许多调用参数, 这些参数用于拨号、身份验证以及和 RAS服务器建立远程连接。该函数的定义如下:

```
DWORD RasDial(
    LPRASDIALEXTENSIONS lpRasDialExtensions,
    LPCTSTR lpszPhonebook,
    LPRASDIALPARAMS lpRasDialParams,
    DWORD dwNotifierType,
    LPVOID lpvNotifier,
    LPHRASCONN lphRasConn
);
```

lpRasDialExtensions参数是一个可选指针, 指向一个RASDIALEXTENSIONS结构, 有了这个结构, 你的应用程序便可使用 RasDial函数的扩展特性了。在 Windows 95、Windows 98和 Windows CE上, 这个参数是被忽略的, 必须把它设为 NULL。RASDIALEXTENSIONS结构的格式如下:

```
typedef struct tagRASDIALEXTENSIONS {
    DWORD      dwSize;
    DWORD      dwfOptions;
    HWND       hwndParent;
    ULONG_PTR  reserved;
#ifdef WINVER >= 0x500
    ULONG_PTR  reserved1;
    RASEAPIFNO RasEapInfo;

```

```
#endif
} RASDIALEXTENSIONS;
```

注意，前面已经讲过，编译期间，由于 WINVER 值不相同，这个结构的长度也会不同。它的各个字段是这样定义的：

dwSize：必须设为 RASDIALEXTENSIONS 结构的长度（按字节算）。

dwfOptions：允许你为使用 RasDial 扩展特性而设置位标志。表 16-1 对这些标志进行了说明。

hwndParent：不用，应该设为 NULL。

reserved：不用，应该设为 0。

reserved1：保留，供以后的 Windows 2000 RAS 扩展特性使用。应该设为 0。

RasEapInfo：在 Windows 2000 上，允许你指定“扩展性身份验证协议”（EAP）信息。EAP 不在本书讨论范围内。

表 16-1 RasDial 位标志

标 志	说 明
RDEOPT_UsePrefixSuffix	令 RasDial 使用与指定拨号设备关联的前缀和后缀
RDEOPT_PausedStates	允许 RasDial 进入暂停操作状态，这样一来，用户就可以重试登录、改变密码和设置回拨号码
RDEOPT_IgnoreModemSpeaker	令 RasDial 忽略 RAS 电话簿中的 Modem 喇叭设置
RDEOPT_SetModemSpeaker	如果设置了 RDEOPT_IgnoreModemSpeaker 标志，那么利用这个标志，就可以打开 Modem 喇叭
RDEOPT_IgnoreSoftwareCompression	令 RasDial 忽略软件压缩设置
RDEOPT_SetSoftwareCompression	在设置了 RDEOPT_IgnoreSoftwareCompression 标志的前提下，再用这个标志，便会打开软件压缩
RDEOPT_PauseOnScript	供 RasDialDlg 内部使用。不能设置这个标志

在 Windows 2000 和 Windows NT 中，RasDial 函数的 lpszPhonebook 参数用于识别到一个电话簿文件的路径。如在 Windows 95、Windows 98 和 Windows CE 上，这个参数必须设为 NULL，因为电话簿是保存在系统注册表内的。所谓电话簿，就是一个 RAS 拨号属性的集合，这些属性对如何设置一个 RAS 连接进行了定义。但并不要求你利用这个电话簿建立一个 RAS 连接。RasDial 的特色在于其丰富的拨号参数，允许你设置一个基本连接。我们稍后将详细讨论 RAS 电话簿。

RASDIALPARAMS 结构指针 lpRasDialParams 定义了拨号和用户身份验证参数，RasDial 函数使用这些参数建立一个远程连接。该结构的格式如下：

```
typedef struct _RASDIALPARAMS {
    DWORD dwSize;
    TCHAR szEntryName[RAS_MaxEntryName + 1];
    TCHAR szPhoneNumber[RAS_MaxPhoneNumber + 1];
    TCHAR szCallbackNumber[RAS_MaxCallbackNumber + 1];
    TCHAR szUserName[UNLEN + 1];
    TCHAR szPassword[PWLEN + 1];
    TCHAR szDomain[DNLEN + 1];
#ifdef WINVER_0x401
    DWORD dwSubEntry;
    DWORD dwCallbackId;
#endif
}
```

```
} RASDIALPARAMS;
```

RASDIALPARAMS结构的各个字段是这样定义的：

dwSize：应该设为一个RASDIALPARAMS结构的长度（按字节算）。有了这个标志，RAS便可对你编辑程序时所用的WINVER版本进行内部判断。

szEntryName：一个字串，允许你标识一个电话簿条目，该条目中包含在 RasDial函数的lpszPhonebook参数中列出的电话簿文件内。这个参数非常重要，因为电话簿条目能够使你更好地指定RAS连接属性，比如选定一个Modem或选定一个分帧协议等等。但是，为使用RasDial而指定一个电话簿条目并不是非用不可的，可选择使用。如果这个字段是空字串（“ ”），RasDial就会选择系统上已安装的第一个可用的Modem，并依据下一个参数szPhoneNumber来拨叫连接。

szPhonebookNumber：一个字串，代表一个电话号码，这个号码优先于 szEntryName字段中指定的电话簿条目内包含的电话号码。

szCallbackNumber：允许你指定一个电话号码，RAS服务器可以根据这个号码回拨。如果RAS服务器允许有一个回拨号码，它就会中断原来的连接，利用你指定的回拨号码回拨。这个特性非常不错，因为有了它，服务器就可知道连接的用户来自何处。

szUserName：一个字串，标识RAS服务器上的用户进行身份验证时所用的登录名。

szPassword：一个字串，标识RAS服务器上的用户进行身份验证时所用的密码。

dwSubentry：标识用户账号所在的Windows 2000或Windows NT区域。

szDomain：可选。允许你指定最初的电话簿子条目拨叫一个RAS多链路连接（我们稍后将讨论RAS电话簿子条目和多链路连接）。

dwCallbackId：允许你把一个应用程序定义的值投递到一个 RasDialFunc2回拨函数中，这个字段没有使用。

下面这两个RasDial参数——dwNotifier和lpvNotifier，用于对RasDial操作模式进行判断（可同步调用还是异步调用）。最后一个RasDial参数是lphRasConn，它是一个指针，指向HRASCONN类型的RAS连接句柄。在调用RasDial之前，必须把这个参数设为NULL。如果RasDial调用成功，就会返回指向RAS连接的引用句柄。

下面，我们将为大家演示如何调用RasDial。正如我们前面提到的那样，RasDial调用可以在两种操作模式中执行：同步和异步。同步模式中，在这个函数成功建立连接或以失败告终以前，RasDial会处于锁定状态，而异步模式中，RasDial则立即执行连接，同时允许你的应用程序在连接期间执行别的操作。

16.4.1 同步模式

如果把RasDial的lpvNotifier参数设为NULL，RasDial就会置入同步模式。lpvNotifier参数是NULL时，dwNotifierType参数就会被忽略。同步调用RasDial是使用该函数的最简单的作用；美中不足的是，同步模式下不能对连接进行监视，稍后，我们将就此进行详述。程序清单16-1演示了如何同步调用RasDial。注意，这段代码中没有指定电话簿或电话簿条目，而是演示如何简单地建立RAS连接。

程序清单 16-1 同步调用RasDial

```
// Always set the size of the RASDIALPARAMS structure
```



```
RasDialParams.dwSize = sizeof(RASDIALPARAMS);
hRasConn = NULL;

// Setting this field to an empty string will allow
// RasDial to use default dialing properties

lstrcpy(RasDialParams.szEntryName, "");

lstrcpy(RasDialParams.szPhoneNumber, "867-5309");
lstrcpy(RasDialParams.szUserName, "jenny");
lstrcpy(RasDialParams.szPassword, "mypassword");
lstrcpy(RasDialParams.szDomain, "mydomain");

// Call RasDial synchronously (the fifth parameter
// is set to NULL)
Ret = RasDial(NULL, NULL, &RasDialParams, 0, NULL, &hRasConn);

if (Ret != 0)
{
    printf("RasDial failed: Error = %d\n", Ret);
}
```

16.4.2 异步模式

比起同步调用RasDial函数来，异步调用要复杂得多。如果RasDial的lpvNotifier参数没有设为NULL，RasDial就会进入异步模式，意味着进行连接的同时，函数调用会立即返回。在进行RAS连接时，异步调用RasDial是优选方法，因为你可以对连接进程进行监视。lpcNotifier参数既可以是一个指针，指向RasDial中发生连接活动时被调用的函数，又可以是一个窗口句柄（通过Windows消息接收进程通知）。RasDial函数的dwNotifierType参数用于判断函数类型或被投入lpvNotifier的窗口句柄。至于dwNotifierType中可以指定哪些值呢？表16-2对此进行了说明。

表16-2 RasDial异步通知方法

通知类型	含 义
0	lpvNotifier参数令RasDial使用RasDialFunc函数指针管理连接事件
1	lpvNotifier参数令RasDial利用RasDialFunc1函数指针管理连接事件
2	lpvNotifier参数令RasDial利用RasDialFunc2函数指针管理连接事件
0xFFFFFFFF	lpvNotifier参数令RasDial在连接事件期间发送一个窗口消息

表16-2展示了lpvNotifier参数中的三个函数原型，你可以把它们提供给RasDial函数，用来接收连接事件的回拨通知。它们是：RasDialFunc、RasDialFunc1和RasDialFunc2。RasDialFunc的原型如下：

```
VOID WINAPI RasDialFunc(
    UINT unMsg,
    RASCONNSTATE rasconnstate,
    DWORD dwError
);
```

unMsg参数接收已发生的事件类型。当前的这个事件只可能是WM_RASDIALEVENT，它意味着这个参数没有用处。rasconnstate参数接收RasDial函数即将发生的连接活动。表16-3

定义了可能发生的连接活动。如果连接活动失败，dwError参数就会收到RAS错误代码。

表16-3展示了异步RasDial调用中，连接活动有三种状态：运行、暂停和中断。运行状态表明RasDial调用仍处于进程中，每一个处于运行状态的活动都将提供进程状态信息。

暂停状态，是指RasDial需要更多信息来建立连接。默认设置是取消暂停状态。在RASDIALEXTENSIONS结构中设置RDEOPT_PausedStates标志，启用通知进程。关于这一点，我们在前面曾经讲过。连接活动处于暂停状态时，就会指明相应的情况。

用户需要提供新的登录凭据，因为身份验证失败。

用户需要提供一个新密码，因为密码已到期。

用户需要提供一个回拨号码。

表16-3 RAS连接活动

活 动	状 态	说 明
RASCS_OpenPort	运行	即将打开一个通信端口
RASCS_PortOpened	运行	通信端口已打开
RASCS_ConnectDevice	运行	准备与一个设备连接
RASCS_DeviceConnected	运行	已成功于设备连接
RASCS_AllDevicesConnected	运行	物理链接已建立
RASCS_Authenticate	运行	RAS身份验证进程已开始
RASCS_AuthNotify	运行	身份验证事件已发生
RASCS_AuthRetry	运行	服务器已请求尝试另一个身份验证
RASCS_AuthCallback	运行	服务器已请求一个回拨号码
RASCS_AuthChangePassword	运行	客户机已请求改变RAS账号上的密码
RASCS_AuthProject	运行	协议发送准备进行
RASCS_AuthLinkSpeed	运行	链接速度正在计算过程中
RASCS_AuthAck	运行	身份验证据请求正在确认过程中
RASCS_ReAuthenticate	运行	回拨之后的身份验证进程准备开始
RASCS_Authenticated	运行	客户机已成功结束身份验证
RASCS_PrepareForCallback	运行	线路即将取消连接，准备回拨
RASCS_WaitForModemReset	运行	准备回拨之前，客户机等待Modem的重新设置
RASCS_WaitForCallback	运行	客户机等待服务器发出的接入拨号
RASCS_Projected	运行	协议发送已完成
RASCS_StartAuthentication	运行	用户身份验证已开始或已完成（只适用于 Windows 95和 Windows 98）
RASCS_CallbackComplete	运行	已经回拨客户机（只适用于 Windows 95和Windows 98）
RASCS_LogonNetwork	运行	客户机正在登录远程网络（只适用于 Windows 95 和 Windows 98）
RASCS_SubEntryConnected	运行	多链路电话簿条目的子条目已接入。RasDialFunc2的dwSubEntry参数中将包括一个已连接子条目的索引
RASCS_SubEntryDisconnected	运行	多链路电话簿条目已取消连接。RasDialFunc2的dwSubEntry参数中将包含这个已取消连接的子条目的索引
RASCS_RetryAuthentication	暂停	RasDial正在等待新的用户凭据
RASCS_CallbackSetByCaller	暂停	RasDial正在等待客户机的回拨号码
RASCS_PasswordExpired	暂停	RasDial希望用户提供一个新密码
RASCS_InvokeEapUI	暂停	Windows 2000平台上，RasDial等待自定义用户接口，以便获得EAP信息
RASCS_Connected	中止	RAS连接成功，处于活动状态
RASCS_Disconnected	中止	RAS连接失败或处于不活动状态

这些活动与本章前面提到的 RASDIALPARAMS 结构中的信息息息相关。发生处于暂停状态的连接活动时，RasDial 就会向你的回调函数（或窗口进程）发出通知。如果把暂停状态取消，RAS 就会向你的通知函数发送一个错误，RasDial 就会失败。如果启用暂停，RasDial 函数就会进入暂停状态，这样，你的应用程序就可以通过 RASDIALPARAMS 结构，提供必要的信息。RasDial 处于暂停状态时，通过原来的调用连接句柄（lphRasConn）和通知函数（lpvNotifier），再次调用这个函数，便可恢复运行。亦可以调用 RasHangUp（稍后将讨论），简单地中止暂停操作，恢复运行。恢复暂停连接时，需要通过投递给已恢复的 RasDial 调用的 RASDIALPARAMS 结构提供必要的用户输入。

注意 恢复暂停状态时，不要直接从一个通知句柄函数（比如 RasDialFunc）调用 RasDial。RasDial 尚不能处理这种情况，所以应该直接从你的应用程序线程中恢复暂停状态。

最后一个状态（中止）表明 RasDial 连接不是已成功，就是失败。还可以表示 RasHangUp 函数关闭了连接。

现在，对于监视异步 RasDial 调用的连接，大家已有初步的认识了。接下来，我们将演示如何设计一个建立的程序，使其能够异步调用 RasDial。程序清单 16-2 展示了这一过程。大家可在本书配套光盘上找到一个异步调用 RasDial 示例。

程序清单 16-2 异步调用 RasDial

```
void main(void)
{
    DWORD Ret;
    RASDIALPARAMS RasDialParams;
    HRASCONN hRasConn;

    // Fill in the RASDIALPARAMS structure with call parameters
    // as was done in the synchronous example
    ...

    if ((Ret = RasDial(NULL, NULL, &RasDialParams, 0,
        &RasDialFunc, &hRasConn)) != 0)
    {
        printf("RasDial failed with error %d\n", Ret);
        return;
    }

    // Perform other tasks while RasDial is processing
    ...
}

// Callback function RasDialFunc()
void WINAPI RasDialFunc(UINT unMsg, RASCONNSTATE rasconnstate,
    DWORD dwError)
{
    char szRasString[256]; // Buffer for error string

    if (dwError)
    {
        RasGetErrorString((UINT)dwError, szRasString, 256);
    }
}
```



```

        printf("Error: %d - %s\n",dwError, szRasString);
        return;
    }
    // Map each of the states of RasDial, and display on the
    // screen the next state that RasDial is entering

    switch (rasconnstate)
    {
        case RASCS_ConnectDevice:
            printf ("Connecting device...\n");
            break;
        case RASCS_DeviceConnected:
            printf ("Device connected.\n");
            break;

        // Add other connection activities here
        ...

        default:
            printf ("Unmonitored RAS activity.\n");
            break;
    }
}

```

表16-2中，还提到了另外两个回调通知函数，它们是：RasDialFunc1和RasDialFunc2。它们的原型如下：

```

VOID WINAPI RasDialFunc1(
    HRASCONN hrasconn,
    UINT unMsg,
    RASCONNSTATE rascs,
    DWORD dwError,
    DWORD dwExtendedError
);

DWORD WINAPI RasDialFunc2(
    DWORD dwCallbackId,
    DWORD dwSubEntry,
    HRASCONN hrasconn,
    UINT unMsg,
    RASCONNSTATE rascs,
    DWORD dwError,
    DWORD dwExtendedError
);

```

RasDialFunc1函数和我们前面讨论的RasDialFunc函数极其相似，只不过增加了两个参数：hbrasconn和dwExtendedError。hbrasconn参数是一个指向连接的句柄。这个连接是RasDial函数返回的。dwExtendedError参数，允许你在连接期间获得dwError参数所发生的这些错误的扩展错误信息。

ERROR_SERVER_NOT_RESPONDING：dwExtendedError收到一个NetBIOS特有的错误代码。

ERROR_NETBIOS_ERROR：dwExtendedError收到一个NetBIOS特有的错误代码。

ERROR_AUTH_INTERNAL：dwExtendedError收到一个内部诊断性错误代码。这些错误代码尚未归入文档。

ERROR_CANNOT_GET_LANA：dwExtendedError收到一个路由RAS特有的错误代码。

RasDialFunc2函数和RasDialFunc1函数极其相似，但它比后者多了两个参数：dwCallbackId和dwSubEntry。dwCallbackId参数中包含了一个应用程序定义的值，这个值最初是设在RASDIALPARAMS结构的dwCallbackId字段中的，我们前面曾讲过这个结构被投给了RasDial调用。dwSubEntry参数接收子条目电话簿索引，这个索引会再次回调RasDialFunc2函数。

16.4.3 状态通知

RAS的特征之一是单立函数RasConnectionNotification。它允许你的应用程序决定何时建立或中断RAS连接。它的定义如下：

```
DWORD RasConnectionNotification(  
    HRASCONN hrasconn,  
    HANDLE hEvent,  
    DWORD dwFlags  
);
```

hrasconn参数是RasDial返回的一个连接句柄。hEvent参数是一个事件句柄，该句柄是你的应用程序利用CreateEvent函数创建的。dwFlags参数可以设为下列连接活动标志的任何一种组合：

RASCN_Connection：通知你RAS连接已经建立。如果hrasconn参数设为INVALID_HANDLE_VALUE，任何一次RAS连接都会触发这一事件。

RASCN_Disconnection：通知你RAS连接已经中止。如果hrasconn参数设为INVALID_HANDLE_VALUE，任何连接中止都会触发这一事件。

RASCN_BandwidthAdded：在一个多链路连接上，子条目连接会触发这一事件。

RASCN_BandwidthRemoved：在多链路连接上，子条目取消连接时，会触发这一事件。

注意，这些标志的活动方式和表16-3中描述的连接活动标志一样。如果连接期间发生了这些活动，就会触发你的事件。因此，你的应用程序应该使用Win32等待函数，比方说WaitForSingleObject，利用它们来决定什么时候向对象发出通知。

16.4.4 关闭连接

关闭RasDial建立的连接很简单。只须调用RasHangUp即可。该函数的定义如下：

```
DWORD RasHangUp(  
    HRASCONN hrasconn  
);
```

hrasconn参数是RasDial返回的一个句柄。尽管这个函数非常易于使用，但仍然需要了解RAS中，连接的内部管理是怎么回事。连接在利用一个调制解调器端口时，如果连接关闭，这个端口需要花时间去重新设置这个连接。因此，你应该一直等下去，直到端口连接完全关闭为止。要做到这一点，在重新设置自己的连接时，可调用RasGetConnectionStatus来判断连接是否完全关闭。RasGetConnectionStatus的定义是这样的：

```
DWORD RasGetConnectStatus(  
    HRASCONN hrasconn,  
    LPRASCONNSTATUS lprasconnstatus  
);
```

hrasconn参数是RasDial返回的一个句柄。lprasconnstatus参数是一个RASCONNSTATUS结构，取得当前的连接状态。RASCONNSTATUS结构的格式如下：

```
typedef struct _RASCONNSTATUS
{
    DWORD dwSize;
    RASCONNSTATE rasconnstate;
    DWORD dwError;
    TCHAR szDeviceType[RAS_MaxDeviceType + 1];
    TCHAR szDeviceName[RAS_MaxDeviceName + 1];
} RASCONNSTATUS;
```

上面这个结构的各个字段是这样定义的：

dwSize：应该设置为RASCONNSTATUS结构的长度（按字节算）。

rasconnstate：取得表16-3中定义的一种连接活动。

dwError：若RasGetConnectStatus没有返回0，就取得一个具体的RAS错误代码。

szDeviceType：取得一个字串，该字串代表连接所用的设备类型。

szDeviceName：取得当前的设备名。

我们建议在收取RASCS_Disconnected活动状态之前，先检查一下自己的连接状态。显而易见，在重新设置连接之前，可能需要多次调用RasGetConnectionStatus。一旦连接被重新设置，就可以退出应用程序或建立另一个连接了。

16.5 电话簿

利用电话簿条目对建立连接所需的属性进行保存和管理，通过这一方式，RAS便可以设置与远程服务器的通信。电话簿不过是一个RASENTRY结构的集合，这些结构中包含了电话号码、数据速率、用户身份验证信息和其他连接信息。在Windows 95、Windows 98和Windows CE平台上，电话簿保存在系统的“注册表”内。在Windows NT和Windows 2000平台上，电话簿的扩展名一般是.PBK的文件中。RASENTRY结构的格式如下：

```
typedef struct tagRASENTRY
{
    DWORD dwSize;
    DWORD dwfOptions;
    DWORD dwCountryID;
    DWORD dwCountryCode;
    TCHAR szAreaCode[RAS_MaxAreaCode + 1];
    TCHAR szLocalPhoneNumber[RAS_MaxPhoneNumber + 1];
    DWORD dwAlternateOffset;
    RASIPADDR ipaddr;
    RASIPADDR ipaddrDns;
    RASIPADDR ipaddrDnsAlt;
    RASIPADDR ipaddrWins;
    RASIPADDR ipaddrWinsAlt;
    DWORD dwFrameSize;
    DWORD dwfNetProtocols;
    DWORD dwFramingProtocol;
    TCHAR szScript[MAX_PATH];
    TCHAR szAutodialDial[MAX_PATH];
    TCHAR szAutodialFunc[MAX_PATH];
    TCHAR szDeviceType[RAS_MaxDeviceType + 1];
}
```

```

TCHAR    szDeviceName[RAS_MaxDeviceName + 1];
TCHAR    szX25PadType[RAS_MaxPadType + 1];
TCHAR    szX25Address[RAS_MaxX25Address + 1];
TCHAR    szX25Facilities[RAS_MaxFacilities + 1];
TCHAR    szX25UserData[RAS_MaxUserData + 1];
DWORD    dwChannels;
DWORD    dwReserved1;
DWORD    dwReserved2;
#if (WINVER >= 0x401)
DWORD    dwSubEntries;
DWORD    dwDialMode;
DWORD    dwDialExtraPercent;
DWORD    dwDialExtraSampleSeconds;
DWORD    dwHangUpExtraPercent;
DWORD    dwHangUpExtraSampleSeconds;
DWORD    dwIdleDisconnectSeconds;

#endif
#if (WINVER >= 0x500)
DWORD    dwType;
DWORD    dwEncryptionType;
DWORD    dwCustomAuthKey;
GUID     guidId;
TCHAR    szCustomDialDll[MAX_PATH];
DWORD    dwVpnStrategy;
#endif
} RASENTRY;

```

正如大家看到的那样，上面这个结构由许多字段组成。这些字段的定义如下：

dwSize：指定一个RASENTRY结构的长度（按字节算）。

dwfOptions：可以设为一个或多个的标志，这些标志的描述可参见表 16-4。

表16-4 RASENTRY结构选项的标志

选项标志	描 述
RASEO_Custom	在Windows 2000平台上，使用自定义加密
RASEO_DisableLcpExtensions	RAS取消定义在RFC 1570中的PPP LCP扩展
RASEO_IpHeaderCompression	RAS将对PPP连接上的IP头压缩进行处理
RASEO_ModemLights	在Windows 2000上，这个任务栏将显示状态监视器
RASEO_NetworkLogon	在Windows 95和Windows 98上，只有在RAS连接通过身份验证之后，RAS才会将让用户登录到一个域
RASEO_PreviewDomain	在Windows 2000上，RAS将在拨号之前，显示用户域
RASEO_PreviewPhoneNumber	Windows 2000上，RAS将显示准备拨打的那个电话号码
RASEO_PromoteAlternates	如果利用备用号码进行的连接成功的话，RAS可以把备用号码转为首要号码
RASEO_RemoteDefaultGateway	RAS连接处于活动状态时，IP包的默认路由通过的是拨号适配器，而不是其他的网络适配器
RASEO_RequireCHAP	Windows 2000上，将采用“盘问联络身份验证协议”(CHAP)进行身份验证
RASEO_RequireDataEncryption	必须成功地协商数据加密；不然就应该丢弃这个连接。此外，还必须设置RASEO_RequireEncryptedPw标志
RASEO_RequireEAP	Windows 2000平台上，将采用EAP进行身份验证
RASEO_RequireEncryptedPw	对客户机进行身份验证时，设置这个标志可避免PPP使用“密码身份验证协议”(PAP)，而是用CHAP和Shiva的密码身份验证协议(SPAP)

(续)

选项标志	描 述
RASEO_RequireMsCHAP	Windows 2000平台上, 将采用微软的CHAP进行身份验证
RASEO_RequireMsCHAP2	Windows 2000平台上, 将采用微软的CHAP第2版进行身份验证
RASEO_RequireMsEncryptedPw	这个标志优先于RASEO_RequireEncryptedPw, 允许RAS使用微软的安全密码方案, 比如说微软的CHAP
RASEO_RequirePAP	Windows 2000平台上, 将采用PAP进行身份验证
RASEO_RequireSPAP	Windows 2000平台上, 将采用SPAP进行身份验证
RASEO_RequireW95MSCHAP	Windows 2000平台上, 采用老版本的微软CHAP(这是为Windows 95 RAS服务器设计的)进行身份验证
RASEO_ReviewUserPW	Windows 2000平台上, RAS将在拨号之前, 显示用户名和密码
RASEO_SecureLocalFiles	Windows 2000和Windows NT平台上, RAS在与一个条目建立连接之前, 对现成的远程文件系统和远程打印机的绑定进行检查
RASEO_SharedPhoneNumbers	Windows 2000上, 电话号码是共享的
RASEO_ShowDialingProgress	Windows 2000上, RAS将显示拨号过程
RASEO_SpecificIpAddr	这个标志要求RAS使用ipaddr字段中指定的IP地址
RASEO_SpecificNameServers	这个标志要求RAS使用ipaddrDns、ipaddrDnsAlt、ipaddrWins和ipaddrWinsAlt这几个字段中指定的IP信息
RASEO_SwCompression	这个标志让RAS对连接上发送的数据的软件压缩进行协商
RASEO_TerminalAfterDial	拨号连接后, RAS显示用户输入的终端窗口
RASEO_TerminalBeforeDial	拨号连接之前, RAS显示用户输入的终端窗口
RASEO_UseCountryAndAreaCodes	这个标志要求RAS利用dwCountryID、dwCountryCode和szAreaCode字段以及szLocalPhoneNumber字段, 构建一个电话号码
RASEO_UseLogonCredentials	这个标志要求RAS在拨号时, 采用用户名、密码和当前正在登录的用户所在区域。只有设置 RASEO_RequiredMsEncryptedPw标志时, 才会发生这种情况

dwCountryID: 如果设置了RASEO_UseCountryAndAreaCodes选项标志, 就会指定一个TAPI国家标识符。调用RasGetCountryInfo函数之后, 便可获取国家标识码信息(我们稍后将对此进行讨论)。

CountryCode: 如果设置了RASEO_UseCountryAndAreaCodes选项标志, 就会指定一个与dwCountryID字段关联的国家代码。如果该字段为 0, 就采用 Windows 中与dwCountryID关联的国家代码。

szAreaCode: 如果设置了RASEO_UseCountryAndAreaCodes标志, 就会指定一个地区代码。

szLocalPhoneNumber: 指定准备拨打的电话号码。如果设置了 RASEO_UseCountry-AndAreaCodes标志, RAS就会把dwCountryID、dwCountryCode和szAreaCode这三个字段的值和你的电话号码合并在一起。

dwAlternateOffset: 指定按字节算的偏移量, 从这个结构的开始处算起, 为 RAS电话簿条目准备的备用电话号码就保存在这里。备用电话号码被保存为一个连续的空中止字串集。这个集合中的最后一个字串是以两个连续的空白字符结尾的。

ipaddr: 如果设置了RASEO_SpecificIpAddr标志, 就为这个连接指定一个IP地址, 供它使用。

ipaddrdns: 如果设置了RASEO_SpecificNameServers标志, 就为这个连接指定一个DNS服务器IP地址。

ipaddrDnsAlt：如果设置了RASEO_SpecificNameServers标志，就为这个连接指定一个备用的DNS服务器IP地址。

ipaddrDWins：如果设置了RASEO_SpecificNameServers标志，就为这个连接指定一个WINS服务器IP地址。

ipaddrDWinsAlt：如果设置了RASEO_SpecificNameServers标志，就为这个连接指定一个备用的WINS服务器IP地址。

dwFrameSize：若在dwFramingProtocol字段中设置RASFP_Slip标志，分帧协议的长度就会变为1006或1500个字节。

dwfNetProtocols：指定标志，这些标志表示分帧协议上将使用哪些网络协议。可能有这几个标志；针对 NetBEUI的RASNP_NetBEUI、针对IPX的RASNP_Ipx、针对IP的RASNP_Ip。

dwFramingProtocol：指定标志，这些标志表示 RAS连接上将使用哪些分帧协议。可能有这几个标志；针对PPP的RASFP_Ppp、针对SLIP的RASFP_Slip、针对异步。

szScript：指定到一个拨号 Script的完整路径，一开始连接，就会执行这个拨号。

szAutodialDll：指定一个定制的 DLL，该DLL可用于设置RAS的自动拨号（autodial）特性。本书没有讨论如何管理RAS自动拨号特性。

szAutodialFunc：对szAutodialDll字段中定制DLL输出的函数名进行指定。

szDeviceType：指定用于连接的设备类型。其值应该标识成字符串。表 16-5列出了它可能的值。

表16-5 RAS设备类型

设备名字串	说 明
"RASDT_Modem"	COM端口上的调制解调器
"RASDT_Isdn"	ISDN适配器
"RASDT_X25"	X.25网卡
"RASDT_Vpn"	虚拟专用网（VPN）连接
"RASDT_Pad"	包汇编码器/取消汇编码器
"RASDT_Generic"	一般类型
"RASDT_Serial"	串口
"RASDT_FrameRelay"	帧中继设备
"RASDT_Atm"	ATM设备
"RASDT_Sonet"	同步光纤网络（SONET）设备
"RASDT_SW56"	交换56-Kbps访问
"RASDT_Irda"	红外线设备
"RASDT_Parallel"	并口

szDeviceName：对连接所用的 TAPI设备进行识别。可利用 RasEnumDevices函数获取TAPI设备，稍后将对此进行讨论。

szX25PadType：指定一个X.25 PAD类型。

szX25Address：指定一个X.25地址。

szX25Facilities：指定设备应答X.25主机发出的请求。

szX25UserData：为X.25连接指定额外的信息。

dwChannels：未使用。

dwReserved1：未使用，但必须设为0。

dwReserved2：未使用，但必须设为0。

dwSubEntries：识别与该电话簿条目关联的多链路子条目有多少。应该把这个字段设为0，并令RasSetSubEntryProperties函数管理该字段的多链路子条目（关于这个函数，我们稍后将详述）。

dwDialMode：Windows 2000中，利用RASEDM_DialAll和RASEDM_DialAsNeeded的定义值进行连接时，这个字段指定RAS应该怎样拨多链路子条目。如果这个字段设为RASEDM_DialAll，就会拨叫所有的多链路子条目。如果设为RASEDM_DialAsNeeded，RAS就会利用dwDialExtraPercent、dwDialExtraSampleSeconds、dwHangUpExtraPercent和dwHangUpExtraSampleSeconds这四个字段来决定何时拨叫和取消新增的多链路子条目。

dwDialExtraPercent：Windows 2000中，该字段指定一个连接当前总带宽的百分比。所用的总带宽超过这个百分比超过了dwDialExtraSampleSeconds指定的时间时，RAS就会拨叫另一个子条目。

dwDialExtraSampleSeconds：Windows 2000中，利用该字段，表明RAS拨叫另一个子条目之前，当前带宽的利用必须超过dwDialExtraPercent中指定的带宽百分比多少秒。

dwHangUpExtraPercent：Windows 2000中，从连接的子条目可用的总带宽的百分比。总带宽的使用率低于dwDialExtraSampleSeconds中指定的百分比时，RAS将中断子条目连接。

dwHangUpExtraSampleSeconds：Windows 2000中，利用该字段，表明RAS在取消一个子条目之前，当前带宽的使用必须降到dwHangUpExtraPercent中定义的带宽百分比以下的限定时间。

dwIdleDisconnectSeconds：利用这个字段，表明在RAS中断连接之前，允许连接闲置多长时间。还可把这个字段设为RASIDS_Disabled（防止连接中断），或RASIDS_UseGlobalValue（利用系统的默认值）。

dwType：Windows 2000中，指定电话簿条目的类型。表16-6列出了这些类型可能的值

表16-6 RASENTRY电话簿类型

类 型	说 明
RASET_Direct	直接的串行或并行连接
RASET_Internet	互联网连接服务（ICS）
RASET_Phone	电话线
RASET_Vpn	虚拟专用网

dwEncryptionType：Windows 2000中，指定连接上投递的数据所使用的加密类型。表16-7列出了可能的值。

表16-7 RAS连接上所使用的数据加密值

值	说 明
ET_40Bit	40位数据加密
ET_128Bit	128位数据加密

dwCustomAuthkey：Windows 2000平台上，指定一个身份验证密钥，这是为EAP厂商

提供。

guidId：Windows 2000平台上，标识与一个电话簿条目关联在一起的 GUID。

szCustomDialDll：Windows 2000平台上，指定到一个DLL的路径，这个DLL中包含了自定义RAS拨叫函数。如果这个字段是 NULL，RAS将采用默认的系统拨叫者。至于如何为RAS开发自定义拨叫者的种种细节，本书没有涉及。

dwVpnStrategy：Windows 2000平台上，指定VPN连接将采用的VPN拨叫方案表16-8列出了可能的值。

表16-8 RAS VPN拨号方案

值	说 明
VS_Default	RAS先拨叫点到点通道传输协议(PPTP)。如果失败，就尝试第2层通道传输协议(L2TP)。
VS_L2tpFirst	先拨叫L2TP
VS_L2tpOnly	只拨叫L2TP
VS_PptpFirst	先拨叫PPTP
VS_PptpOnly	只拨叫PPT

在调用RAS API时，如果它把一个电话簿文件当作参数（ lpszPhonebook ），就可识别出到电话簿文件的路径了。正如我们前面提到的那样， Windows 95、Windows 98和Windows CE平台上，这个参数必须是 NULL，因为电话簿条目是保存在系统注册表内的。而在 Windows 2000和Windows NT平台上，这是到一个电话簿文件的路径。通常，这个电话簿文件有一个扩展名.pbk。另外，Windows 2000和Windows NT上的系统默认电话簿位于% SystemRoot%\System32\Ras\RasPhone.pbk。如果你将电话簿指定为 NULL，便使用系统默认电话簿文件。

有三个支持函数可帮助你建立和管理电话簿条目。它们是： RasValidateEntry、RasEnumDevices和RasGetCountryInfo。下面展示的RaValidateEntryName函数，用来判断名字的格式是否正确，是否已包含在电话簿中。

```
DWORD RasValidateEntryName(
    LPCTSTR lpszPhonebook,
    LPCTSTR lpszEntry
);
```

lpszPhonebook参数是一个指针，指向电话簿文件名。 lpszEntry参数是一个字串，表示正在核对的电话簿条目名。电话簿中没有这个名字，但该名字格式无误时，便返回 ERROR_SUCESS。若名字格式有错，这个函数就会失败，返回 ERROR_INVALID_NAME；如果电话簿中有这个名字，就会返回 ERROR_ALREADY_EXISTS。

对具体的计算机而言，可利用 RasEnumDevice函数获得所有具有RAS能力的设备名及类型。

```
DWORD RasEnumDevices(
    LPRASDEVINFO lpRasDevInfo,
    LPDWORD lpcb,
    LPDWORD lpcDevices
);
```

lpRasDevInfo参数是一个指针，指向一个你必须提供的应用程序缓冲区，这个缓冲区用来接收RASDEVINFO结构数组。RASDEVINFO结构的格式如下：

```
typedef struct tagRASDEVINFO {
    DWORD dwSize;
```

```
TCHAR szDeviceType[RAS_MaxDeviceType + 1];
TCHAR szDeviceName[RAS_MaxDeviceName + 1];
} RASDEVINFOW;
```

该结构的字段是这样定义的：

dwSize：在调用RasEnumDevices之前，必须把它设为RASDEVINFO结构的长度（按字节算）。

szDeviceType：接收对设备类型进行描述的字串，比如RASDT_Modem。

szDeviceName：接收TAPI设备的正式名称。

这时，必须确定提供的缓冲区足以容纳若干个结构；否则，RasEnumDevices就会失败，返回ERROR_BUFFER_TOO_SMALL。下一个参数lpcb，是一个变量指针，用于取得列举设备时所需的字节数。这个参数必须设为你自己的lpRasDevInfo缓冲区的长度（按字节算）。最后一个参数lpcDevices，是一个变量指针，用于取得写入lpRasDevInfo中的lpcDRASDEVINFO结构的个数。

RasGetCountryInfo函数允许从Windows中取得各个国家特有的TAPI拨叫信息。

```
DWORD RasGetCountryInfo(
    LPRASCTRYINFO lpRasCtryInfo,
    LPDWORD lpdwSize
);
```

lpRasCtryInfo参数是一个缓冲区，用于接收拨号的前缀和与指定国家相关的其他信息。这个缓冲区必须是一个RASCTRYINFO结构，后面再跟上额外的字节（接收一个国家描述字串）。为了能够容纳RASCTRYINFO结构和描述性字串，我们建议至少分配256个字节的缓冲区。RASCTRYINFO结构的格式如下：

```
typedef struct RASCTRYINFO
{
    DWORD dwSize;
    DWORD dwCountryID;
    DWORD dwNextCountryID;
    DWORD dwCountryCode;
    DWORD dwCountryNameOffset;
} RASCTRYINFO;
```

它的字段是这样定义的：

dwSize：必须设为RASCTRYINFO结构的长度（按字节算）。

dwCountryID：允许你在Windows国家列表中指定一个TAPI国家识别符（用于RASENTRY结构的dwCountryID字段）。如果把这个字段设为1，就会收到列表中的第一个条目。

dwNextCountryID：收到列表中的下一个TAPI国家识别符。如果把这个字段设为0，就会收到列表中的最后一个识别符。

dwCountryCode：收到指定在dwCountryID参数中的国家相关的拨号前缀代码。

dwCountryNameOffset：指定字节数，表示这个结构的起始处到空中中止字串的起始处之间的字节是多少，这个空中中止字串对RASCTRYINFO结构后面的国家进行了描述。

RasGetCountryInfo的另一个参数是lpdwsize，它是一个变量指针，接收lpRasCtryInfo放在lpRasCtryInfo缓冲区中的字节数。在调用这个函数之前，必须把这个参数设为你的应用程序的缓冲区长度。

16.5.1 电话簿条目的增添

RAS有四个函数，允许通过程序对电话簿RASENTRY结构进行管理。它们是：RasSetEntry、RasGetEntryProperties、RasRenameEntry和RasDeleteEntry。如要建立一个新条目或对一个现成的条目进行修改，可使用RasSetEntryProperties函数，它的定义如下：

```
DWORD RasSetEntryProperties(  
    LPCTSTR lpszPhonebook,  
    LPCTSTR lpszEntry,  
    LPRASENTRY lpRasEntry,  
    DWORD dwEntryInfoSize,  
    LPBYTE lpbDeviceInfo,  
    DWORD dwDeviceInfoSize  
);
```

lpszPhonebook参数是一个指针，指向电话簿文件名。lpszEntry参数是一个指向字串的指针，这个字串用于识别现成的或新建的条目。RASENTRY结构以这个名字存在，其属性就会被修改；否则，便在这个电话簿中建立一个新条目。lpRasEntry参数是一个指向RASENTRY结构的指针。可以把空中中止字符串列表放在定义备用电话号码的RASENTRY结构之后。最后一个字串的结尾是两个连续的空字符串。dwEntryInfoSize参数是lpRasEntry参数中的那个结构的长度（按字节数算）。lpbDeviceInfo参数是一个指向缓冲区的指针，该缓冲区中包含TAPI设备的配置信息。在Windows 2000和Windows NT平台上，没有使用这个参数，因此，应该设为NULL。最后一个参数是dwDeviceInfoSize，它代表lpbDeviceInfo缓冲区的长度（按字节数算）。

RasGetEntryProperties函数，可用于获得一个现成电话簿条目的属性或一个新电话簿条目的值。它的定义如下：

```
DWORD RasGetEntryProperties(  
    LPCTSTR lpszPhonebook,  
    LPCTSTR lpszEntry,  
    LPRASENTRY lpRasEntry,  
    LPDWORD lpdwEntryInfoSize,  
    LPBYTE lpbDeviceInfo,  
    LPDWORD lpdwDeviceInfoSize  
);
```

lpszPhonebook参数是一个指向电话簿文件名的指针。lpszEntry参数是一个指向一个字串的指针，该字串标识一个现成电话簿条目。如果把这个参数设为NULL，lpRasEntry和lpbDeviceInfo参数就会收到一个电话簿条目的默认值。获得这些默认值是非常有用的：如果需要建立一个新RAS电话簿条目，可在调用RasSetEntryProperties函数之前，用恰当的系统信息填充lpRasEntry和lpbDeviceInfo这两个字段。

lpRasEntry参数是指向一个缓冲区的指针，这个缓冲区是你的应用程序为接收RASENTRY结构而提供的。我们在讨论RasSetEntryProperties函数时，曾经提过这个结构的后面可以跟一个空中中止字符串数组，这些字符串为请求的电话簿条目识别备用电话号码。因此，供接收结构用的缓冲区长度应该大于RASENTRY结构的长度。如果向它传递一个NULL指针，lpdwEntryInfoSize参数就会收到一个总字节数，这个数是保存RASENTRY结构的所有元素和所有的备用电话号码所需要的。lpdwEntryInfoSize参数是一个指针，指向包含接收缓冲区中的那个字节数的DWORD，这个缓冲区是你的应用程序为lpRasEntry参数提供的。RasSetEntry Properties函数

结束时，会把 `lpdwEntryInfoSize` 更新成 `lpRasEntry` 中实际上收到的字节数。我们强烈建议大家调用这个函数时，把 `lpRasEntry` 设为 `NULL`，把 `lpdwEntryInfoSize` 设为 0，以便获得缓冲区长度的有关信息。一旦有了正确的缓冲区长度，就可以再次调用这个函数，准确无误地获得所有的信息。

`lpbDeviceInfo` 参数是一个指针，指向一个应用程序提供的缓冲区，这个缓冲区用来接收这个电话簿条目的 TAPI 设备相关信息。如果把它设为 `NULL`，`lpdwDeviceInfoSize` 参数就会收到获得该信息所需的字节数。如果你使用的是 Windows 2000 和 Windows NT，就应该把 `lpbDeviceInfo` 参数设为 `NULL`。最后一个参数是 `lpdwDeviceInfoSize`，它是一个指向 `DWORD` 的指针，`DWORD` 应该被设为字节数，从这个数可看出为 `lpbDeviceInfo` 提供的缓冲区中包含的字节有多少。`RasGetEntryProperties` 返回时，`lpbDeviceInfoSize` 就会返回 `lpbDeviceInfo` 缓冲区内返回的字节数。

程序清单 16-3 演示了一个应用程序应该怎样利用 `RasGetEntryProperties` 和 `RasSetEntryProperties` 来建立一个新的电话簿条目。

程序清单 16-3 利用默认属性建立一个新的 RAS 电话簿条目

```
#include <windows.h>
#include <ras.h>
#include <raserror.h>
#include <stdio.h>

void main(void)
{
    DWORD EntryInfoSize = 0;
    DWORD DeviceInfoSize = 0;
    DWORD Ret;
    LPRASENTRY lpRasEntry;
    LPBYTE lpDeviceInfo;

    // Get buffer sizing information for a default phonebook entry
    if ((Ret = RasGetEntryProperties(NULL, "", NULL,
        &EntryInfoSize, NULL, &DeviceInfoSize)) != 0)
    {
        if (Ret != ERROR_BUFFER_TOO_SMALL)
        {
            printf("RasGetEntryProperties sizing failed "
                "with error %d\n", Ret);
            return;
        }
    }

    lpRasEntry = (LPRASENTRY) GlobalAlloc(GPTR, EntryInfoSize);

    if (DeviceInfoSize == 0)
        lpDeviceInfo = NULL;
    else
        lpDeviceInfo = (LPBYTE) GlobalAlloc(GPTR, DeviceInfoSize);

    // Get default phonebook entry
    lpRasEntry->dwSize = sizeof(RASENTRY);

    if ((Ret = RasGetEntryProperties(NULL, "", lpRasEntry,
```

```

        &EntryInfoSize, lpDeviceInfo, &DeviceInfoSize)) != 0)
    {
        printf("RasGetEntryProperties failed with error %d\n",
            Ret);
        return;
    }

    // Validate new phonebook name "Testentry"
    if ((Ret = RasValidateEntryName(NULL, "Testentry")) !=
        ERROR_SUCCESS)
    {
        printf("RasValidateEntryName failed with error %d\n",
            Ret);
        return;
    }

    // Install a new phonebook entry, "Testentry", using
    // default properties
    if ((Ret = RasSetEntryProperties(NULL, "Testentry",
        lpRasEntry, EntryInfoSize, lpDeviceInfo,
        DeviceInfoSize)) != 0)
    {
        printf("RasSetEntryProperties failed with error %d\n",
            Ret);
        return;
    }
}

```

16.5.2 电话簿条目的重命名

现在，大家已对建立和修改电话簿条目有了一定的认识。接下来看看 RasRenameEntry 函数。RasRenameEntry 只允许对电话簿条目进行重命名。它的定义如下：

```

DWORD RasRenameEntry(
    LPCTSTR lpszPhonebook,
    LPCTSTR lpszOldEntry,
    LPCTSTR lpszNewEntry
);

```

lpszPhonebook 参数是一个指向电话簿文件的指针。lpszOldEntry 参数是一个指向字串的指针，这个字串用于识别准备进行重命名的那个现成的电话簿条目。lpszNewEntry 参数是指向一个字串的指针，这个字串中包含了电话簿条目的新名字。你的应用程序在调用 RasRenameEntry 之前，应该根据新名字来调用 RasValidateEntryName。如果 RasRenameEntry 调用成功，就会返回 0。如果失败，就会返回下面这些错误：

ERROR_INVALID_NAME：表示 lpszNewEntry 名字无效。

ERROR_ALREADY_EXISTS：表示电话簿中已存在 lpszNewEntry 名字。

ERROR_CANNOT_FIND_PHONEBOOK_ENTRY：表示电话簿中未找到 lpszOldEntry 名。

16.5.3 电话簿条目的删除

删除一个电话簿条目很简单。调用 RasDeleteEntry 函数即可。这个函数的定义如下：


```
DWORD RasDeleteEntry(  
    LPCTSTR lpszPhonebook,  
    LPCTSTR lpszEntry  
);
```

lpszPhonebook参数是一个指向电话簿文件的指针。lpszEntry参数是一个字串，这个字串代表一个现成的电话簿条目。如果调用这个函数成功，就会返回 ERROR_SUCCESS；否则，就返回ERROR_INVALID_NAME。

16.5.4 电话簿条目的列举

RAS提供了一个非常方便的函数——RasEnumEntries，它可以获得电话簿文件中所有可用的电话簿条目。它的定义如下：

```
DWORD RasEnumEntries (  
    LPCTSTR reserved,  
    LPCTSTR lpszPhonebook,  
    LPRASENTRYNAME lprasentryname,  
    LPDWORD lpcb,  
    LPDWORD lpcEntries  
);
```

reserved参数没有采用，但必须把它设为 NULL。lpszPhonebook参数是一个指向电话簿文件名的指针。lprasentryname参数是一个指向应用程序缓冲区的指针，这个缓冲区是你必须提供的，用来接收 RASENTRYNAME 结构数组。RASENTRYNAME 结构的格式如下：

```
typedef struct _RASENTRYNAME  
{  
    DWORD dwSize;  
    TCHAR szEntryName[RAS_MaxEntryName + 1];  
#if (WINVER >= 0x500)  
    DWORD dwFlags;  
    CHAR  szPhonebookPath[MAX_PATH + 1];  
#endif  
} RASENTRYNAME;
```

字段如下定义：

dwSize：在调用 RasEnumEntries 之前，必须把它设为 RASENTRYNAME 结构的长度（按字节算）。

szEntryName：接收电话簿条目的名字。

dwFlags：在 Windows 2000 平台上，这个标志表示电话簿条目是在采用 REN_AllUsers 标志的系统默认电话簿（前面曾讲过）内还是在采用 REN_User 标志的一个用户配置电话簿内。

szPhonebookPath：Windows 2000 平台上，指定到电话簿文件的完整路径。

这时，必须提供足够大的缓冲区，以便能够容纳若干个结构；若不然，RasEnumEntries 调用就会失败，出现 ERROR_BUFFER_TOO_SMALL 错误。下一个参数 lpcb 是一个变量指针，表示列举条目所需要的字节数有多少。这个参数必须设为你自己的 lprasentryname 缓冲区的长度（按字节算）。最后一个参数是 lpcEntries，它是一个变量指针，表示写入 lprasentryname 缓冲区中的 RASENTRYNAME 结构数有多少。

16.5.5 用户凭据的管理

RAS 客户机通过 RasDial 利用电话簿条目进行连接时，便将该用户的安全凭据保存下来，

并把它和电话簿条目关联起来。RasGetCredentials、RasSetCredentials、RasGetEntryDialParams和RasSetEntryDialParams这四个函数允许你对与一个电话条目关联的用户安全凭据进行管理。RasGetCredentials和RasSetCredentials这两个函数是Windows NT 4中引入的（也可用于Windows 2000）。这两个函数取代了RasGetEntryDialParams和RasSetEntryDialParams。由于RasGetCredentials和RasSetCredentials不能用于Windows 95、Windows 98和Windows CE，只有采用RasGetEntryDialParams和RasSetEntryDialParams，它们可用于所有的平台。

RasGetCredentials函数，可获得与一个电话簿条目关联的用户凭据。它的定义如下：

```
DWORD RasGetCredentials(  
    LPCTSTR lpszPhonebook,  
    LPCTSTR lpszEntry,  
    LPRASCREDENTIALS lpCredentials  
);
```

lpszPhonebook参数是一个指向电话簿文件的指针。lpszEntry参数是一个代表现成电话簿条目的字串。lpCredentials参数是一个指针，指向RASCREDENTIALS结构，这个结构可接收与指定电话簿条目相关的用户名、密码和区域。RASCREDENTIALS结构的格式如下：

```
typedef struct {  
    DWORD dwSize;  
    DWORD dwMask;  
    TCHAR szUserName[UNLEN + 1];  
    TCHAR szPassword[PWLEN + 1];  
    TCHAR szDomain[DNLEN + 1];  
} RASCREDENTIALS, *LPRASCREDENTIALS;
```

它的字段是这样定义的：

dwSize：指定RASCREDENTIALS结构的长度（按字节算）。这个字段应该始终设为结构的长度。

dwMask：是一个位掩码字段，通过预先定义标志，识别这个结构中接下来的哪三个字段是有效的。预先定义标志有：用于szUserName的RASCMD_UserName、用于szPassword的RASCMD_Password、用于szDomain的RASCMD_Domain。

szUserName：是一个空中止字串，其中包含用户登录名。

szPassword：是一个空中止字串，其中包含用户密码。

szDomain：是一个空中止字串，其中包含用户登录的区域。

若RasGetCredentials调用成功，就会返回0。根据lpCredentials结构的dwMask字段中设置的标志，你的应用程序便可知道设置了哪些安全凭据。

RasSetCredentials函数类似于RasGetCredentials，但有一点除外，它还允许你改变与一个电话簿条目关联的安全凭据。除了多一个fClearCredentials参数外，它的其余参数均和RasGetCredentials一样。RasSetCredentials函数的定义如下：

```
DWORD RasSetCredentials(  
    LPCTSTR lpszPhonebook,  
    LPCTSTR lpszEntry,  
    LPRASCREDENTIALS lpCredentials,  
    BOOL fClearCredentials  
);
```

fClearCredentials参数是一个布尔运算符，如果设为TRUE，就会导致RasSetCredentials把凭据改为一个空字串（“ ”）值，凭据是在lpCredentials结构中dwMask字段内进行识别的。比

如，如果 dwMask 中包含 RASCM_Password 标志，原来保存的密码就会被替换成一个空字符串。如果 RasSetCredentials 函数调用成功，就会返回 0。

另外，还可以用 RasGetEntryDialParams 和 RasSetEntryDialParams 函数来管理与具体电话簿条目相关的用户安全凭据。RasGetEntryDialParams 的定义如下：

```
DWORD RasGetEntryDialParams(
    LPCTSTR lpszPhonebook,
    LPRASDIALPARAMS lprasdialparams,
    LPBOOL lpfPassword
);
```

lpszPhonebook 参数是一个指向电话簿文件名的指针。lprasdialparams 参数是指向一个 RASDIALPARAMS 结构的指针。lpfPassword 参数是一个布尔标志，如果在 lprasdialparams 结构中获得了用户密码，就会返回 TRUE。

RasSetEntryDialParams 函数用于改变上一次的连接信息，该信息是 RasDial 调用时设置的，与特定的电话簿条目相关。RasSetEntryDialParams 的定义如下：

```
DWORD RasSetEntryDialParams(
    LPCTSTR lpszPhonebook,
    LPRASDIALPARAMS lprasdialparams,
    BOOL fRemovePassword
);
```

lpszPhonebook 和 lprasdialparams 参数与 RasGetEntryDialParams 函数中的前两个参数是一样的。fRemovePassword 参数是一个布尔标志，如果设为 TRUE，则是要求 RasSetEntryDialParams 删除与指定电话簿条目相关的密码，这个条目是在 lprasdialparams 结构中识别的。

16.5.6 多链接电话簿的子条目

Windows 2000 和 Windows NT 平台上，RAS 允许你对多链路电话簿条目进行管理，以增强通信能力。多链路条目能够使一个 RAS 连接上关联更多的通信设备，进而增大连接的总带宽。RAS 允许你利用 RasGetSubEntryProperties 和 RasSetSubEntryProperties 这两个函数对多链路电话簿条目进行管理。RasGetSubEntryProperties 函数的定义如下：

```
DWORD RasGetSubEntryProperties(
    LPCTSTR lpszPhonebook,
    LPCTSTR lpszEntry,
    DWORD dwSubEntry,
    LPRASSUBENTRY lpRasSubEntry,
    LPDWORD lpdwcb,
    LPBYTE lpbDeviceConfig,
    LPDWORD lpcbDeviceConfig
);
```

lpszPhonebook 参数是一个指向电话簿文件名的指针。lpszEntry 参数是一个电话簿条目。dwSubEntry 参数指定的是一个子条目的索引，这个子条目包含在电话簿条目内。lpRasSubEntry 参数是一个指向缓冲区的指针，这个缓冲区将接收一个 RASSUBENTRY 结构和一个供选择的备用电话号码列表。RASSUBENTRY 结构的格式如下：

```
typedef struct tagRASSUBENTRY
{
```

```

    DWORD dwSize;
    DWORD dwfFlags;
    TCHAR szDeviceType[RAS_MaxDeviceType + 1];
    TCHAR szDeviceName[RAS_MaxDeviceName + 1];
    TCHAR szLocalPhoneNumber[RAS_MaxPhoneNumber + 1];
    DWORD dwAlternateOffset;
} RASSUBENTRY;

```

上面这个结构的定义是这样的：

dwSize：必须设成一个RASSUBENTRY结构的长度（按字节算）。

dwFlags：未用。

szDeviceType：接收一个代表设备类型的字串，该设备用于具体的连接上。

szDeviceName：接收TAPI设备的真名。

szLocalPhoneNumber：识别该设备所用的电话号码。

dwAlternateOffset：指定字节数，表示 RASSUBENTRY结构的开始处到跟在该结构后面的连续空中止字串字节表之间的字节数是多少。

lpRasSubEntry缓冲区必须足够大，以便能够容纳一个 RASSUBENTRY结构和备用的电话号码串；否则，RasGetSubEntryProperties就会失败，并返回ERROR_BUFFER_TOO_SMALL错误。lpdwcb参数应该设为你自己的lpRasSubEntry缓冲区中的字节数。函数返回时，lpdwcb将收到总的字节数，这个数表示包含 RASSUBENTRY结构和备用电话号码需要这么多字节。lpbDeviceConfig和lpcbDeviceConfig参数都没有使用，应该设为NULL。

你可以建立一个新的子条目或对一个指定电话簿条目的现成的子条目进行修改，这是通过调用RasSetSubEntryProperties函数来完成的。该函数的定义如下：

```

DWORD RasSetSubEntryProperties(
    LPCTSTR lpszPhonebook,
    LPCTSTR lpszEntry,
    DWORD dwSubEntry,
    LPRASSUBENTRY lpRasSubEntry,
    DWORD dwcbRasSubEntry,
    LPBYTE lpbDeviceConfig,
    DWORD dwcbDeviceConfig
);

```

这个函数的各个参数中，lpRasSubEntry指定准备添加到电话簿中的子条目。其余参数则和RasGetSubEntryProperties中的参数一样。

16.6 连接管理

RAS有三个非常有用的函数。通过它们，可获得自己系统上已建立的各个连接的属性。它们是：RasEnumConnections、RasGetSubEntryHandle和RasGetProjectionInfo。RasEnumConnections函数列出了所有活动的RAS连接，它的定义如下：

```

DWORD RasEnumConnections(
    LPRASCONN lprasconn,
    LPDWORD lpcb,
    LPDWORD lpcConnections
);

```

lprasconn参数是一个应用程序缓冲区，将收到一个 RASCONN结构数组。RASCONN结构的格式如下：

```
typedef struct _RASCONN
{
    DWORD dwSize;
    HRASCONN hrasconn;
    TCHAR szEntryName[RAS_MaxEntryName + 1];
#ifdef WINVER >= 0x400
    CHAR szDeviceType[RAS_MaxDeviceType + 1];
    CHAR szDeviceName[RAS_MaxDeviceName + 1];
#endif
#ifdef WINVER >= 0x401
    CHAR szPhonebook[MAX_PATH];
    DWORD dwSubEntry;
#endif
#ifdef WINVER >= 0x500
    GUID guidEntry;
#endif
} RASCONN;
```

该结构的字段定义如下：

dwSize：指出RASCONN结构的长度（按字节算）。

brasconn：取得RasDial建立的连接句柄。

szEntryName：取得用来建立连接的电话簿条目。如果采用了空字符串，这个字段就会返回一个带有句号（.）的字符串，后面还有全部电话号码。

szDeviceType：取得一个字串，描述连接所用的设备类型。

szDeviceName：取得一个带有设备名的字符串，该设备用于建立连接。

szPhoneName：为建立连接的条目取得电话簿的完整路径。

dwSubEntry：取得一个多链接电话簿条目的子条目索引。

guidEntry：在Windows 2000平台上，为建立连接所用的电话簿条目取得一个GUID。

对RasEnumConnections函数而言，需要把它投到一个够大的缓冲区，使之能够容纳若干个RASCONN结构；不然的话，这个函数就会失败，并返回ERROR_BUFFER_TOO_SMALL错误。此外，你的缓冲区中，第一个RASCONN结构中的dwSize字段必须被设成一个RASCONN结构的字节长。下一个参数 lpcb，是一个指向一个变量的指针，必须把这个变量设为自己的lprasconn数组的长度（按字节算）。这个函数返回时，lpcb就会包含列举所有连接时所需要的一切字节。即使你没有提供足够大的缓冲区，也可以用 lpcb中返回的正确的缓冲区长度再试一次。lpcConnections参数是一个变量指针，取得被写成 lprasconn的RASCONN结构的个数。

RasGetSubEntryHandle函数的定义在下一页。它允许你为多链接连接的一个具体的子条目取得一个连接句柄。

```
DWORD RasGetSubEntryHandle(
    HRASCONN hrasconn,
    DWORD dwSubEntry,
    LPHRASCONN lphrasconn
);
```

brasconn参数是一个多链接连接的RAS连接句柄。dwSubEntry参数是多链接连接中的一个

设备的子条目索引。lpbrasconn参数为子条目设备取得连接句柄。

有了取自RasEnumConnections和RasGetSubEntryHandle的RAS连接句柄，就可获得网络协议特有的信息，该信息用于一个已建立的 RAS连接。这个新的网络协议特有的信息就是人们所说的“Projection information”(投影信息)。远程访问服务器利用投影信息来表示网络上的一个远程客户机。比方说，在一个分帧协议上建立一个连接时，这个连接采用的是 IP协议，IP配置信息（比如分配得到的 IP地址）就会从RAS服务到客户机之间建立起来。要想获得 PPP分帧协议上的协议投影信息，调用 RasGetProjectionInfo函数即可，该函数的定义如下：

```
DWORD RasGetProjectionInfo(
    HRASCONN hrasconn,
    RASPROJECTION rasprojection,
    LPVOID lpprojection,
    LPDWORD lpcb
);
```

brasconn参数是一个RAS连接句柄。rasprojection参数是一个RASPROJECTION列举类型，允许你指定一个协议来接收连接信息。lpprojection参数取得一个数据结构，这个结构和rasprojection中指定的列举类型有关联。最后一个参数 lpcb是一个变量指针，必须把它设为自己的lpprojection结构的长度。这个函数结束时，该变量中会包含获得投影信息所需要的缓冲区的长度。

下面的RASPROJECTION列举类型允许你取得连接信息：

```
RASP_Amb。
RASP_PppNbf。
RASP_PppIpx。
RASP_PppIp。
```

如果指定一个RASP_Amb类型，就会收到一个RASAMB结构，该结构的格式如下：

```
typedef struct _RASAMB
{
    DWORD dwSize;
    DWORD dwError;
    TCHAR szNetBiosError[NETBIOS_NAME_LEN + 1];
    BYTE bLana;
} RASAMB;
```

它的各个字段是这样定义的：

dwSize：应该设为RASAMB结构的长度（按字节算）。

dwError：从PPP协商进程中取得一个错误代码。

szNetBiosError：如果在 PPP身份验证进程中，发生了名字冲突，就会收到一个 NetBIOS名。如果dwError返回ERROR_NAME_EXISTS_ON_NET，szNetBiosError就会收到导致这一错误的那个名字。

bLana：对用于建立远程访问连接的 NetBIOS LAN适配器（LANA）的编号进行识别。

如果把RASP_PppNbf指定为RasGetProjectionInfo，就会收到一个RASPPPNBF结构，该结构的定义如下：

```
typedef struct _RASPPPNBF
{
    DWORD dwSize;
    DWORD dwError;
```



```

DWORD dwNetBiosError;
TCHAR szNetBiosError[NETBIOS_NAME_LEN + 1];
TCHAR szWorkstationName[NETBIOS_NAME_LEN + 1];
BYTE bLana;
} RASPPPNBF;

```

RASPPPNBF的字段和RASAMB结构中的字段一样，但是，前者包含的字段比后者多了两个：szWorkstationName和 dwNetBiosError。szWorkstationName字段收到NetBIOS名，这个名用来识别网络上你正在连接的那个工作站。dwNetBiosError字段收到的是发生的NetBIOS错误。

如果把RASP_PppIp枚举类型指定为 RasGetProjectionInfo，收到的则是一个RASPPPIPX结构，该结构的定义如下：

```

typedef struct _RASPPPIPX
{
    DWORD dwSize;
    DWORD dwError;
    TCHAR szIpAddress[RAS_MaxIpAddress + 1];
} RASPPPIPX;

```

它的字段是这样定义的：

dwSize：应该设为一个RASPPPIPX结构的长度（按字节算）。

dwError：从PPP协商进程中取得一个错误代码。

szIpAddress：取得一个字串，该字串代表客户机的IPX地址。

如果把RASP_PppIp枚举类型指定为 RasGetProjectionInfo，就会收到RASPPPIP结构。该结构的格式如下：

```

typedef struct _RASPPPIP
{
    DWORD dwSize;
    DWORD dwError;
    TCHAR szIpAddress[RAS_MaxIpAddress + 1];
    TCHAR szServerIpAddress[RAS_MaxIpAddress + 1];
} RASPPPIP;

```

它的各个字段是这样定义的：

dwSize：应该设为一个RASPPPIP结构的长度（按字节算）。

dwError：从PPP协商进程中取得一个错误代码。

szIpAddress：取得一个代表客户机IP地址的字串。

szServerIpAddress：取得一个代表服务器IP地址的字串。

对在RAS基础上建立的IP连接来说，怎样才能获得为它分配的IP地址呢？看看程序清单16-4，你就明白了。

程序清单 16-4 在一个IP连接上使用 RasGetProjectionInfo

```

lpProjection = (RASPPPIP *) GlobalAlloc(GPTR, cb);
lpProjection->dwSize = sizeof(RASPPPIP);
cb = sizeof(RASPPPIP);

Ret = RasGetProjectionInfo(hRasConn, RASP_PppIp,
    lpProjection, &cb);

if (Ret != ERROR_SUCCESS)

```

```
{
    printf("RasGetProjectionInfo failed with error %d", Ret);
    return;
}
else
{
    printf("\nRas Client IP address: %s\n",
        lpProjection->szIpAddress);
    printf("Ras Server IP address: %s\n",
        lpProjection->szServerIpAddress);
}
```

16.7 小结

本章介绍了利用 RAS来扩展计算机连网能力的基础知识。还详细地说明了如何调用 RasDial函数与远程网络进行通信。另外，我们还讨论了如何利用建立电话簿条目充分挖掘 RAS的潜力。本章最后，对RAS和本书第三部分进行了全面的总结。