第 5 章　Internet 文件下载工具

Internet 不可逆转地改变了程序设计的过程。在 Internet 出现之前，大多数应用程序都在单独的机器上孤立地执行，或者使用小型的局域网。Internet 改变了这种状况。现在，大多数计算机都连接到 Internet，许多程序都使用了万维网的大量资源。对于现代程序员，将 Internet 功能整合到应用程序中不再是一种选择，而是一种必然。

尽管 Internet 很重要，但是 C++并没有提供对其内建的支持。这是因为 C++在 20 世纪 90 年代后期 Internet 出现的时候就已经成熟了。然而这并非是一个障碍，事实上却是一个优点。C++允许访问操作系统的 Internet 功能，而不是指定了所有程序员都必须使用的惟一方法。通过让您选择到 Internet 最好的接口，不仅提供了潜在的高效率和灵活性，而且可以让您以最适合底层执行环境的方法来编写具有 Internet 功能的程序。因此，如果您想要建立高性能的、具有 Internet 功能的代码，C++是您的第一选择。

为了说明 C++可以轻易地处理 Internet 任务，本章开发了一个文件下载系统，它可以被整合到许多不同的基于Internet的应用程序中。当给定文件的URL时，这个文件下载工具从Internet获取这个文件。这是一个独立的子系统，可以处理打开 Internet 连接、读取文件以及随后的关闭连接中所有的细节。这个文件下载工具有一个特殊的特征：它可以断点续传。例如，如果在下载的过程中，与 Internet 的连接丢失，当连接重新建立时，这个下载可以在中断的那个点重新启动。当通过慢速的调制解调器连接下载很大的文件时，这个特性尤其重要。

在开发这个下载子系统之后，建立了两个使用它的应用程序。第一个是简单的、基于控制台的应用程序，只是为了演示这个下载工具。第二个是可以用来下载文件的 GUI 应用程序。

由于 Internet 支持是由操作系统提供的，因此有必要选择一个操作系统。对几乎所有本书的读者都适用的操作系统是 Windows，在本章就使用它。然而，基本的技术也可以应用到其他环境。

提示：

本章假定读者对 Internet 具有一般的理解，并且具有 Windows 程序设计的工作经验。对这两个问题的讲述远远超出了本书的范围。

5.1　WinINet 库

为了访问 Internet，本章使用了 Windows 提供的易于使用的库。这个库称为 Windows Internet，或者简称 WinINet。WinINet API 提供了多种高级的函数，以一致的、稳定的方式处理各种协议，如 HTTP 和 FTP。Windows 为您处理底层细节(在某种意义上，这类似于<fstream>为文件操作提供一致的接口，从而为您处理细节)。正如您所看到的，如果您遵循少许规则，就可以向任何 Windows 应用程序加入对 Internet 的访问。

尽管 WinINet 是一个很大的库，但是您只需要使用下面一些函数：

InternetAttemptConnect	检查 Internet 连接是否有效
InternetOpen	打开 Internet 连接并返回一个句柄
InternetOpenUrl	打开 URL 并返回一个句柄
HttpQueryInfo	从最后一个响应报头获取信息
InternetReadFile	从开放的 URL 读取字节
InternetCloseHandle	关闭 Internet 句柄

在讨论文件下载工具的代码时，会详细讨论这些函数。

为了使用 WinINet，必须在您的程序中包含 wininet.h，并且将您的应用程序与 wininet.lib 相链接。

5.2 文件下载工具子系统

文件下载工具子系统的整个代码如下所示。代码在文件 dl.cpp 中。

```
// A file download subsystem.
#include <iostream>
#include <windows.h>
#include <wininet.h>
#include <fstream>
#include <cstdio>

using namespace std;

const int MAX_ERRMSG_SIZE = 80;
const int MAX_FILENAME_SIZE = 512;
const int BUF_SIZE = 1024;

// Exception class for download errors.
class DLExc {
  char err[MAX_ERRMSG_SIZE];
public:
  DLExc(char *exc) {
    if(strlen(exc) < MAX_ERRMSG_SIZE)
      strcpy(err, exc);
  }

  // Return a pointer to the error message.
  const char * geterr() {
    return err;
  }
};

// A class for downloading files from the Internet.
class Download {
  static bool ishttp(char *url);
  static bool httpverOK(HINTERNET hIurl);
  static bool getfname(char *url, char *fname);
  static unsigned long openfile(char *url, bool reload,
```

```
                                    ofstream &fout);
public:
  static bool download(char *url, bool restart=false,
    void (*update)(unsigned long, unsigned long)=NULL);
};


// Download a file.
//
// Pass the URL of the file to url.
//
// To reload a file, pass true to reload.
//
// To specify an update function that is called after
// each buffer is read, pass a pointer to that
// function as the third parameter. If no update
// function is desired, then let the third parameter
// default to null.
bool Download::download(char *url, bool reload,
       void (*update)(unsigned long, unsigned long)) {

  ofstream fout;             // output stream
  unsigned char buf[BUF_SIZE]; // input buffer
  unsigned long numrcved;    // number of bytes read
  unsigned long filelen;     // length of file on disk
  HINTERNET hIurl, hInet;    // Internet handles
  unsigned long contentlen; // length of content
  unsigned long len;         // length of contentlen
  unsigned long total = 0;  // running total of bytes received
  char header[80];           // holds Range header

  try {
    if(!ishttp(url))
      throw DLExc("Must be HTTP url.");

    // Open the file specified by url.
    // The open stream will be returned
    // in fout. If reload is true, then
    // any preexisting file will be truncated.
    // The length of any preexisting file (after
    // possible truncation) is returned.
    filelen = openfile(url, reload, fout);

    // See if Internet connection available.
    if(InternetAttemptConnect(0) != ERROR_SUCCESS)
      throw DLExc("Can't connect.");

    // Open Internet connection.
    hInet = InternetOpen("downloader",
                         INTERNET_OPEN_TYPE_DIRECT,
                         NULL, NULL, 0);
```

```
    if(hInet == NULL)
      throw DLExc("Can't open connection.");

    // Construct header requesting range of data.
    sprintf(header, "Range:bytes=%d-", filelen);

    // Open the URL and request range.
    hIurl = InternetOpenUrl(hInet, url,
              header, -1,
              INTERNET_FLAG_NO_CACHE_WRITE, 0);

    if(hIurl == NULL) throw DLExc("Can't open url.");

    // Confirm that HTTP/1.1 or greater is supported.
    if(!httpverOK(hIurl))
      throw DLExc("HTTP/1.1 not supported.");

    // Get content length.
    len = sizeof contentlen;
    if(!HttpQueryInfo(hIurl,
                      HTTP_QUERY_CONTENT_LENGTH |
                      HTTP_QUERY_FLAG_NUMBER,
                      &contentlen, &len, NULL))
      throw DLExc("File or content length not found.");

    // If existing file (if any) is not complete,
    // then finish downloading.
    if(filelen != contentlen && contentlen)
      do {
        // Read a buffer of info.
        if(!InternetReadFile(hIurl, &buf,
                             BUF_SIZE, &numrcved))
          throw DLExc("Error occurred during download.");

        // Write buffer to disk.
        fout.write((const char *) buf, numrcved);
        if(!fout.good())
          throw DLExc("Error writing file.");

        total += numrcved; // update running total

        // Call update function, if specified.
        if(update && numrcved > 0)
          update(contentlen+filelen, total+filelen);

      } while(numrcved > 0);
    else
      if(update)
        update(filelen, filelen);

  } catch(DLExc) {
    fout.close();
```

```
    InternetCloseHandle(hIurl);
    InternetCloseHandle(hInet);

    throw; // rethrow the exception for use by caller
  }

  fout.close();
  InternetCloseHandle(hIurl);
  InternetCloseHandle(hInet);

  return true;
}

// Return true if HTTP version of 1.1 or greater.
bool Download::httpverOK(HINTERNET hIurl) {
  char str[80];
  unsigned long len = 79;

  // Get HTTP version.
  if(!HttpQueryInfo(hIurl, HTTP_QUERY_VERSION, &str, &len, NULL))
    return false;

  // First, check major version number.
  char *p = strchr(str, '/');
  p++;
  if(*p == '0') return false; // can't use HTTP 0.x

  // Now, find start of minor HTTP version number.
  p = strchr(str, '.');
  p++;

  // Convert to int.
  int minorVerNum = atoi(p);

  if(minorVerNum > 0) return true;
  return false;
}

// Extract the filename from the URL. Return false if
// the filename cannot be found.
bool Download::getfname(char *url, char *fname) {
  // Find last /.
  char *p = strrchr(url, '/');

  // Copy filename after the last /.
  if(p && (strlen(p) < MAX_FILENAME_SIZE)) {
    p++;
    strcpy(fname, p);
    return true;
  }
  else
    return false;
```

```
}

// Open the output file, initialize the output
// stream, and return the file's length. If
// reload is true, first truncate any preexisting
// file.
unsigned long Download::openfile(char *url,
                                 bool reload,
                                 ofstream &fout) {
  char fname[MAX_FILENAME_SIZE];

  if(!getfname(url, fname))
    throw DLExc("File name error.");

  if(!reload)
    fout.open(fname, ios::binary | ios::out |
                     ios::app | ios::ate);
  else
    fout.open(fname, ios::binary | ios::out |
                     ios::trunc);

  if(!fout)
    throw DLExc("Can't open output file.");

  // Get current file length.
  return fout.tellp();
}

// Confirm that the URL specifies HTTP.
bool Download::ishttp(char *url) {
  char str[5] = "";

  // Get first four characters from URL.
  strncpy(str, url, 4);

  // Convert to lowercase
  for(char *p=str; *p; p++) *p = tolower(*p);

  return !strcmp("http", str);
}
```

注意，这个文件定义了两个类。第一个类是 DLExc，这个类封装了下载工具抛出的异常。DLExc 的构造函数传递了一个指向字符串的指针来描述这个异常，并存储这个字符串。调用成员函数 geterr()可以获得指向错误信息的指针。

第二个类是 Download，这个类处理文件的下载。注意，Download 类只包含静态函数。因此，将下载工具放入到一个类中，不仅是一种封装的手段，更是一种组织的技术。事实上，可以使用名字空间而不是类来达到这个目的。然而，使用类可以将几个函数设置为私有的，从而阻止了其他代码的使用。另外，使用类简化了以后新特性的扩展。

下面部分详细介绍 Download 的各个部分。

5.2.1 操作的一般理论

在检查 Download 的各个部分之前，理解它操作的一般理论是有帮助的。为了下载文件，这个文件的 URL 被传递给 download()函数。如果在磁盘上不存在同名文件，则这个文件被全部下载。然而，如果找到同名的已下载的部分文件，就只会下载这个文件的剩余部分。正是下载文件剩余部分的能力使得续传被中断的下载成为可能。

为了执行部分下载，Download 依赖于 HTTP 1.1 版本(或者更高版本)提供的功能：Range header，这个功能允许下载某个范围的数据。因此，这个下载工具只能使用支持 HTTP 1.1 版本或者更高版本的 URL。正是 Range header 使得被中断的下载可以从中断的那个点重启。由于需要 HTTP 1.1 版本，这个下载工具只支持 HTTP 下载。

当涉及到 HTTP 请求时，报头是伴随着这个请求的信息。Range 报头是一个字符串，其通常的形式为：

```
Range: bytes=start-end
```

在此，start 指定了这个范围的开始，end 指定了结尾。如果没有 end，这个范围从开头到文件的结束。

Download 定义了一个公有函数 download()，这个函数负责下载文件。因此，它是下载的入口点。Download 定义了 4 个支持函数：

ishttp	确定 URL 是否指定一个 HTTP 请求
httpverOK	确定使用的是否为 HTTP 1.1 版本或更高版本
getfname	获取 URL 的文件名部分
openfile	打开文件并返回其长度，如果已存在部分下载，那么长度将大于 0

在 Download 内，这些支持函数被声明为 private。

5.2.2 download()函数

download()函数是 Download 定义的惟一的公有函数。它由用户代码调用来处理文件的下载。因此，为了下载文件，需要调用 download()函数。由于它的重要性，我们对其逐行检查。它的开始代码为：

```
bool Download::download(char *url, bool reload,
void (*update)(unsigned long, unsigned long)) {
```

download()函数具有 3 个参数。首先是 url，这是一个指向字符串的指针，这个字符串包含了文件完整的 URL(包括文件名)。文件名假定为 URL 中最后一个“/”后面的名称。例如，下面的 URL 中，文件名为 MyFile.dat:

```
http://www.SoeSite.com/SomeDir/MyFile.dat
```

记住，必须使用规范的形式指定完整的 URL，包括 http: //。

第二个参数 reload 判断是否再次下载已经下载过的某个文件。如果 reload 为 true，则会下载整个文件，而不管磁盘上是否存在这个文件(无论存在部分或者全部)。如果 reload 为 false，那么只下载文件的剩余部分(如果有的话)。因此，如果您想要获取一个已下载文件的新副本，就将 reload 设置为 true。

第三个参数是 update。这是一个函数指针，它作为用户代码了解下载过程的一种方法，被 download()周期性地调用。如果这个参数为 null，则不会使用 update 函数。我们将在稍后讨论对 update 函数参数的需求。

随后，声明了如下的局部变量：

```
ofstream fout;            // output stream
unsigned char buf[BUF_SIZE];     // input buffer
unsigned long numrcved;   // number of bytes read
unsigned long filelen;    // length of file on disk
HINTERNET hIurl, hInet;   // Internet handles
unsigned long contentlen; // length of content
unsigned long len;        // length of contentlen
unsigned long total = 0;  // running total of bytes received
char header[80];          // holds Range header
```

注意变量 hIurl 和 HInet。它们分别拥有到被下载的 URL 的句柄和到 Internet 连接的句柄。WinINet API 函数会用到其中的一个句柄。

main 函数以下面所示的代码开始。注意整个下载代码被包装到一个 try/catch 块中，用来处理通信和/或者文件错误。

```
try {
  if(!ishttp(url))
    throw DLExc("Must be HTTP url.");
```

download()做的第一件事是调用 ishttp()，通过验证指定的 URL 是否以“http”开始来判断它否是一个 HTTP 请求。如前所述，这个下载工具只能处理 HTTP 请求。

随后，打开接收文件并获取它的长度。

```
// Open the file specified by url.
// The open stream will be returned
// in fout. If reload is true, then
// any preexisting file will be truncated.
// The length of any preexisting file (after
// possible truncation) is returned.
filelen = openfile(url, reload, fout);
```

正如注释所说明的那样，如果这个文件不存在或者被重新下载，那么删除已经存在的同名文件。在此情况下，openfile()返回的文件长度为 0。否则，如果这个文件已经存在，就会返回它的长度。

记住，如果某个下载被中断，当这个下载重启时，部分文件已经存在。

文件打开之后，下面的代码建立与 Internet 的连接：

```
// See if Internet connection available.
if(InternetAttemptConnect(0) != ERROR_SUCCESS)
  throw DLExc("Can't connect.");

// Open Internet connection.
hInet = InternetOpen("downloader",
                     INTERNET_OPEN_TYPE_DIRECT,
```

```
                        NULL, NULL, 0);

  if(hInet == NULL)
    throw DLExc("Can't open connection.");
```

首先，通过调用 WinINet API 函数 InternetAttemptConnect()来试图与 Internet 建立连接。如果可能建立连接，则这个函数返回 ERROR_SUCCESS。这个函数只保留了一个参数，并且这个参数必须为 0。如果计算机当前没有连接，那么(取决于您的计算机设置)将会看到一个对话框，询问您是否想要连接。如果没有可用的连接，InternetAttemptConnect()返回一个错误，从而导致异常抛出。

假定某个连接可用，就会调用 InernetOpen()打开这个连接。这是另一个 WinINet 函数，原型如下：

```
HINTERNET InternetOpen(LPCTSTR agent, DWORD access,
                       LPCTSTR proxy, LPCTSTR proxopt,
                       DWORD options)
```

在此，*agent* 指定了应用程序的名称，*access* 指定了使用的访问类型。对于这个下载工具，这个访问的类型为 INTERNET_OPEN_TYPE_DIRECT。当 *access* 为 INTERNET_OPEN_TYPE_DIRECT 时，*proxy* 和 *proxopt* 都不会被使用，并且必须为 null。参数 *options* 指定了任意的选项，下载工具不需要这些选项。函数返回一个打开连接的句柄。如果这个连接没有被打开，则返回 null。

一旦打开连接，就会生成下面的 Range 报头：

```
// Construct header requesting range of data.
sprintf(header, "Range:bytes=%d-", filelen);
```

将这个报头传递给如下所示的 WinINet 函数 InternetOpenUrl()，这个函数打开 url 中指定的 URL：

```
// Open the URL and request range.
hIurl = InternetOpenUrl(hInet, url,
          header, -1,
          INTERNET_FLAG_NO_CACHE_WRITE, 0);

if(hIurl == NULL) throw DLExc("Can't open url.");
```

InternetOpenUrl()函数的原型如下：

```
HINTERNET InternetOpenUrl(HINTERNET hInet, LPCTSTR url,
                          LPCTSTR headerstr, DWORD headlen,
                          DWORD options, LPDWORD extra);
```

InternetOpen()获得的 Internet 句柄传递给 hInet。指向包含打开的 URL 的字符串的指针传递给 url。一个指向字符串的指针传递给了 headerstr，这个字符串包含了一个或者多个将被整合到请求中的附加报头。headerstr 的长度传递给 headlen，如果 headerstr 指向一个 null-terminated 字符串，则它可以为 - 1，就如同下载工具那样。在 option 中可以指定各种标记。这个下载工具所使用的是：

```
INTERNET_FLAG_NO_CACHE_WRITE
```

这样做可防止缓存下载的文件。任何附加的、应用程序指定的信息都可以传递给 extra。由于这个下载工具不需要这些信息，因此使用 0。这个函数将返回开放 URL 的句柄，如果失败，则返回空值。

服务器的每个响应都带有一个报头。这个报头的组件之一包含 HTTP 的版本信息。由于这个下载工具要求服务器支持 HTTP 1.1 版本或者更高版本，那么在开始处理之前，有必要检查 HTTP 的版本。这个任务是通过调用 httpverOK()来完成的，如下所示：

```
// Confirm that HTTP/1.1 or greater is supported.
if(!httpverOK(hIurl))
  throw DLExc("HTTP/1.1 not supported.");
```

假如 HTTP 的版本是可以接受的，则下一步将获取下载内容的数量。调用 WinINet 函数 HttpQueryInfo()可以完成这个任务，如下所示：

```
// Get content length.
len = sizeof contentlen;
if(!HttpQueryInfo(hIurl,
                  HTTP_QUERY_CONTENT_LENGTH |
                  HTTP_QUERY_FLAG_NUMBER,
                  &contentlen, &len, NULL))
  throw DLExc("File or content length not found.");
```

当 HttpQueryInfo()返回时，内容的长度(按字节数)存储在 contentlen 中。内容长度自动考虑 Ranger header 请求的范围。因此，如果整个文件将被下载，则请求的范围将会是“0-”(它指定整个文件)，内容的长度将与文件的长度相等。当续传一个被中断的下载时，范围值将指定文件中间的一个点作为起始点，内容的长度等于剩余的字节数。

HttpQueryInfo()从报头获取信息。其原型如下：

```
BOOL HttpQueryInfo(HINTERNET hIurl, DWORD what, LPVOID buf,
                   LPDWORD buflen, LPDWORD index)
```

在此情况下，将由 InternetOpenUrl()返回的请求的句柄传递给 hIurl。将指定被获取的信息项的值传递给 what。对于这个下载工具，需要两个值。第一个值是

```
HTTP_QUERY_CONTENT_LENGTH
```

用来获取被请求的内容的长度(在此情况下是文件)，第二个值是

```
HTTP_QUERY_FLAG_NUMBER
```

它要求这个值以整型值表示。把指向接收信息的缓冲区的指针传递给 buf，指定缓冲区长度的整型值的指针传递给 buflen。参数 index 允许您指定报头的索引，但是这个下载工具不需要这个特征，因此传递 null。如果能够获取请求的信息，函数则返回 true。内容的长度有可能不可使用，在此情况下 HttpQueryInfo()返回 false。

假定内容的长度可用，如果磁盘上现有文件的长度小于内容的长度，并且内容的长度不为 0，这个文件(或者文件的剩余部分)用下面的代码下载：

```
// If existing file (if any) is not complete,
// then finish downloading.
if(filelen != contentlen && contentlen)
  do {
    // Read a buffer of info.
    if(!InternetReadFile(hIurl, &buf,
                         BUF_SIZE, &numrcved))
      throw DLExc("Error occurred during download.");

    // Write buffer to disk.
    fout.write((const char *) buf, numrcved);
    if(!fout.good())
      throw DLExc("Error writing file.");

    total += numrcved; // update running total

  // Call update function, if specified.
    if(update && numrcved > 0)
      update(contentlen+filelen, total+filelen);
  } while(numrcved > 0);
else.
  if(update)
    update(filelen, filelen);
```

这段代码使用了 WinINet 的函数 InternetReadFile()来读取文件，每次读取一个缓冲区。每循环一次，都会调用 update 指向的函数，除非 update 是 null。如前所述，update 所指向的函数用来报告文件的下载进度。

InternetReadFile()是一个非常有用的函数，因为它使得从 Internet 读取文件的方式与从磁盘读取文件的方式相同。其原型如下：

```
BOOL InternetReadFile(HINTERNET hIurl, LPVOID buf, DWORD numbytes,
                      LPDWORD numrcvc);
```

文件的句柄传递给 hIurl。对于这个下载工具，这是由 InternetOpenUrl()返回的句柄。将指向接收数据的缓冲区的指针传递给 buf，将要读取的字节数传递给 numbytes。这个值不能超过 buf 的长度。实际读取的字节数返回到 numrcvd 所指的变量中。当不再需要获取字节时，这个值为 0。如果成功，这个函数返回 true，否则返回 false。

download()函数以下面的代码结束：

```
  } catch(DLExc) {
fout.close();
    InternetCloseHandle(hIurl);
    InternetCloseHandle(hInet);

    throw; // rethrow the exception for use by caller
  }

  fout.close();
  InternetCloseHandle(hIurl);
  InternetCloseHandle(hInet);
```

```
  return true;
}
```

如果成功地完成，则 download()函数将通过关闭文件和 Internet 句柄结束，并返回 true。如果发生错误，则仍然会关闭句柄，并抛出异常。这样做允许用户代码对错误做出响应。

5.2.3 ishttp()函数

如前所述，这个下载工具只支持 HTTP 文件下载。Download 使用这里所示的 ishttp()函数来确定文件的 URL 指定了 HTTP 协议：

```
// Confirm that the URL specifies HTTP.
bool Download::ishttp(char *url) {
  char str[5] = "";

  // Get first four characters from URL.
  strncpy(str, url, 4);

  // Convert to lowercase
  for(char *p=str; *p; p++) *p = tolower(*p);

  return !strcmp("http", str);
}
```

ishttp()的操作相当直接。它只是检查 URL 中前四个字符是否包含了字符串“http”。如果确实如此，它返回 true，否则返回 false。

5.2.4 httpverOK()函数

如下所示的 httpverOK()函数确定处理请求的服务器是否支持 HTTP 1.1 版本：

```
// Return true if HTTP version of 1.1 or greater.
bool Download::httpverOK(HINTERNET hIurl) {
  char str[80];
  unsigned long len = 79;

  // Get HTTP version.
  if(!HttpQueryInfo(hIurl, HTTP_QUERY_VERSION, &str, &len, NULL))
    return false;

  // First, check major version number.
  char *p = strchr(str, '/');
  p++;
  if(*p == '0') return false; // can't use HTTP 0.x

  // Now, find start of minor HTTP version number.
  p = strchr(str, '.');
  p++;

  // Convert to int.
  int minorVerNum = atoi(p);
```

```
  if(minorVerNum > 0) return true;
  return false;
}
```

URL 的句柄传递给 httpverOK()。随后，通过对这个句柄调用 HttpQueryInfo()，并指定 HTTP_QUERY_VERSION(它请求版本信息)，获取了 HTTP 的版本(以字符串的形式)。这个查询将版本字符串保存在 str 中。对于 HTTP 1.1 版本，字符串的形式如下：

```
HTTP/1.1
```

然后，这个函数调用标准库函数 strchr()将 p 指向字符“/”。接着增加 p，并确保它所指的数字不为 0。下一步，建立一个指向句号的指针来分割主版本号和次版本号。然后增加这个指针，使它指向句号后数值的第一个数字。最后，通过使用标准库中的 atoi()函数将这个值转换为整型值。如果次版本号大于等于 1，那么 HTTP 版本大于等于 1.1。

5.2.5 getfname()函数

如下所示的 getfname()函数从 URL 中提取文件名：

```
// Extract the filename from the URL. Return false if
// the filename cannot be found.
bool Download::getfname(char *url, char *fname) {
  // Find last /.
  char *p = strrchr(url, '/');

  // Copy filename after the last /.
  if(p && (strlen(p) < MAX_FILENAME_SIZE)) {
    p++;
    strcpy(fname, p);
    return true;
  }
  else
    return false;
}
```

在函数返回时，这个函数传递了一个指向保存 URL 的字符串的指针，以及一个指向保存了文件名的字符数组的指针。文件名假定为 URL 中最后一个“/”到结尾之间的部分。如果成功，则这个函数返回 true；如果找不到文件名，则返回 false。

5.2.6 openfile()函数

openfile()函数打开一个磁盘文件，下载的文件将写入到这个文件中。它还返回现有文件的长度。如下所示：

```
// Open the output file, initialize the output
// stream, and return the file's length. If
// reload is true, first truncate any existing
// file.
```

```
unsigned long Download::openfile(char *url,
                                 bool reload,
                                 ofstream &fout) {
 char fname[MAX_FILENAME_SIZE];

 if(!getfname(url, fname))
   throw DLExc("File name error.");

 if(!reload)
   fout.open(fname, ios::binary   ios::out |
                    ios::app | ios::ate);
 else
   fout.open(fname, ios::binary   ios::out |
                    ios::trunc);

 if(!fout)
   throw DLExc("Can't open output file.");

 // Get current file length.
 return fout.tellp();
}
```

openfile()函数有 3 个参数：url、reload 和 fout。将指向包含 URL 的字符串的指针传递给 url。判断是否删除任何相同名称的现有文件的值传递给 reload。传递给 fout 一个指针，这个指针指向一个变量，当 openfile()返回时，这个变量包含了打开的文件流。

首先，openfile()从 url 指向的字符串获取文件名，然后检查 reload 的值。如果 reload 为 true，则打开输出文件，同时任何现有文件的内容都被删除。如果 reload 为 false，则会打开文件，而不会删除现有文件的任何内容，并指定所有的输出都在文件的结尾发生。然后将文件指针移动到文件结尾。随后，通过调用 tellp()来获取文件的长度。对于新建的文件，长度为 0。然而，如果找到现有的文件，这个值就是现有文件的长度，因为当打开文件时，文件指针在文件的结尾处。返回文件的长度。

5.2.7 update()函数

如前所述，如果想要跟踪下载的进度，就必须向 download()的 update 参数传递一个指向函数的指针，这个函数将会接收跟踪的信息。(对于后台操作，update 可以使用默认的 null)。跟踪函数必须具有如下的原型。(当然，函数名可以不同)。

```
void update(unsigned long total, unsigned long part);
```

当调用 update()时，文件的总长度传递给 total，当前下载的数量传递给 part。可以使用这些信息来监视下载的进度，并让用户知道。

5.3 Download 头文件

任何使用这个下载工具的文件都必须包含如下的头文件。这个文件的名称为 dl.h。

```
// Header file for downloader. Call this file dl.h.
#include <iostream>
#include <string>
#include <windows.h>
#include <wininet.h>
#include <fstream>
using namespace std;

const int MAX_ERRMSG_SIZE = 80;

// Exception class for downloading errors.
class DLExc {
  char err[MAX_ERRMSG_SIZE];
public:
  DLExc(char *exc) {
    if(strlen(exc) < MAX_ERRMSG_SIZE)
      strcpy(err, exc);
  }

  const char * geterr() {
    return err;
  }
};

class Download {
  static bool httpverOK(HINTERNET hIurl);
  static bool getfname(char *url, char *fname);
  static unsigned long openfile(char *url, bool reload,
                                ofstream &fout);
public:
  static bool download(char *url, bool restart=false,
    void (*update)(unsigned long, unsigned long)=NULL);
};
```

5.4 文件下载工具的演示

下面的程序建立了一个简单的下载文件的控制台应用程序来演示这个文件下载工具。为了前后一致，称其为 dltest.cpp。

```
// A Sample program that uses Download.
#include <iostream>
#include "dl.h"

// This function displays the download progress as a percentage.
void showprogress(unsigned long total, unsigned long part) {
```

```
  int val = (int) ((double) part/total*100);
  cout << val << "%" << endl;
}

int main(int argc, char *argv[])
{

  // This URL is for demonstration purposes only. Substitute
  // the URL of the file that you want to download.
  char url[] =
    "http://www.osborne.com/products/0072226803/0072226803_code.zip";

  bool reload = false;

  if(argc==2 && !strcmp(argv[1], "reload"))
    reload = true;

  cout << "Beginning download.\n";

  try {
    if(Download::download(url, reload, showprogress))
      cout << "Download Complete\n";
  } catch(DLExc exc) {
    cout << exc.geterr() << endl;
    cout << "Download Interrupted\n";
  }

  return 0;
}
```

在这个程序中，有3件很有趣的事情。首先，要注意到它包含了一个硬编码的URL。这个URL指定了一个文件，这个文件包含了免费的、在线的本书作者著的另一本书(*C++: The Complete Reference,4th Edition*)的代码。这个文件方便地测试了这个下载工具。当然，您可以指定自己选择的URL。

随后，注意showprogress()函数。指向这个函数的一个指针被传递给download()的update参数。这意味着每次下载一个数据缓冲区时都会调用此函数。显示了一个值来指示完成的百分比。

最后要注意，dltest接收一个命令行参数。如果这个参数为“reload”，那么不管磁盘上是否存在这个文件，都会将其全部下载。如果没有给出这个参数，只有在磁盘上这个文件不完整时，这个文件(或者这个文件的剩余部分)才会被下载。

为了编译dltest.cpp，必须链接wininet.lib。例如，如果使用Visual C++的命令行编译，就可以使用如下的命令行来编译这个程序：

```
cl -GX dl.cpp dltest.cpp wininet.lib
```

如果使用IDE，就必须记得在链接中加入wininet.lib。

5.5 基于GUI的下载工具

无论何时您需要从Internet下载程序代码内的文件，都可以使用Download类。例如，可以使用下载工具获取每周的销售报表。如果加入前端用户界面，它还可以用作独立的Internet实用工具。为了说明第二个用途，本章为Windows开发了一个完整的、基于GUI的文件下载工具，其名称为WinDL。

WinDL显示了一个对话框，允许用户输入将要下载的文件的URL。它有一个进度条来显示下载的状态。它还有一个复选框来让用户请求完全下载文件。图5-1显示了这个文件下载对话框。

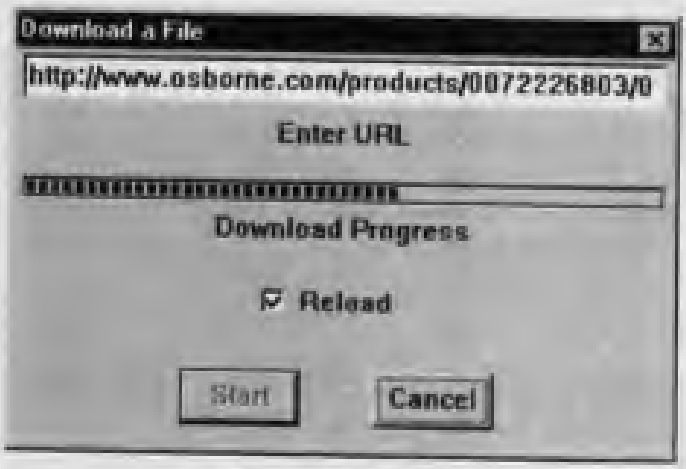

图5-1 文件下载对话框

5.5.1 WinDL代码

WinDL由两个主要部分组成。首先当然是Download的代码，这在前面已经讲述过。第二个部分是处理GUI界面的代码。这些代码在下面给出。为了与前面一致，称这个文件为windl.cpp。

```
// WinDL: A GUI-based file download utility.

#include <windows.h>
#include <commctrl.h>
#include <cstring>
#include <cstdio>
#include "windl.h"
#include <process.h>
#include "dl.h"

const int URL_BUF_SIZE = 1024;

LRESULT CALLBACK WindowFunc(HWND, UINT, WPARAM, LPARAM);
BOOL CALLBACK DialogFunc(HWND, UINT, WPARAM, LPARAM);

void showprogress(unsigned long total,
                  unsigned long part);
```

```
void resetprogress();
unsigned __stdcall dlstart(void * reload);

char szWinName[] = "Download"; // name of window class

HINSTANCE hInst;  // instance handle
HWND hwnd;        // handle of main window
HWND hProgWnd;    // handle of progress bar

HANDLE hThrd = 0;   // thread handle
unsigned long Tid; // thread ID

// Progress counters.
int percentdone = 0;
int oldpercentdone = 0;

// A small struct for passing info to dlstart().
struct ThrdInfo {
  char *url; // pointer to URL string
  int reload; // reload flag
  HWND hPBStart; // handle of Start button
};

int WINAPI WinMain(HINSTANCE hThisInst, HINSTANCE hPrevInst,
                   LPSTR lpszArgs, int nWinMode)
{
  MSG msg;
  WNDCLASSEX wcl;
  INITCOMMONCONTROLSEX cc;

  // Define a window class.
  wcl.cbSize = sizeof(WNDCLASSEX);

  wcl.hInstance = hThisInst;      // handle to this instance
  wcl.lpszClassName = szWinName; // window class name
  wcl.lpfnWndProc = WindowFunc;  // window function
  wcl.style = 0;                 // default style

  wcl.hIcon = LoadIcon(NULL, IDI_APPLICATION); // large icon
  wcl.hIconSm = NULL; // use small version of large icon
  wcl.hCursor = LoadCursor(NULL, IDC_ARROW); // cursor style

  wcl.lpszMenuName = NULL; // no menu

  wcl.cbClsExtra = 0; // no extras
  wcl.cbWndExtra = 0;

  wcl.hbrBackground = NULL; // not used

  // Register the window class.
  if(!RegisterClassEx(&wcl)) return 0;
```

```
  // Create a main window that won't be visible.
  hwnd = CreateWindow(
    szWinName, // name of window class
    "File Downloader", // title
    0, // no style needed
    0, 0, 0, 0, // no dimensions
    NULL, // no parent window
    NULL, // no menu
    hThisInst, // instance handle
    NULL // no additional arguments
  );

  hInst = hThisInst; // save the current instance handle

  // Initialize the common controls. This is
  // needed because of the progress bar.
  cc.dwSize = sizeof(INITCOMMONCONTROLSEX);
  cc.dwICC = ICC_PROGRESS_CLASS;
  InitCommonControlsEx(&cc);

  // Show the window minimized.
  ShowWindow(hwnd, SW_SHOWMINIMIZED);

  // Create the download dialog box.
  DialogBox(hInst, "DLDB", hwnd, (DLGPROC) DialogFunc);

  // Create the message loop.
  while(GetMessage(&msg, NULL, 0, 0))
  {
    TranslateMessage(&msg); // translate keyboard messages
    DispatchMessage(&msg); // return control to Windows
  }

  return msg.wParam;
}

// Window function.
LRESULT CALLBACK WindowFunc(HWND hwnd, UINT message,
                            WPARAM wParam, LPARAM lParam)
{
  switch(message) {
    case WM_DESTROY:
      PostQuitMessage(0); // terminate the program
      break;
    default:
      return DefWindowProc(hwnd, message, wParam, lParam);
  }
  return 0;
}

// Downloader Dialog function.
BOOL CALLBACK DialogFunc(HWND hdwnd, UINT message,
```

```
                        WPARAM wParam, LPARAM lParam)
{
  // Here, url is initialized with a sample url for
  // demonstration purposes only.
  static char url[URL_BUF_SIZE] =
    "http://www.osborne.com/products/0072226803/0072226803_code.zip";

  static ThrdInfo ti;
  switch(message) {
    case WM_INITDIALOG:
      // Initialize edit box with URL.
      SetDlgItemText(hdwnd, IDD_EB1, url);

      // Create progress bar.
      hProgWnd = CreateWindow(PROGRESS_CLASS,
                  "",
                  WS_CHILD | WS_VISIBLE | WS_BORDER,
                  4, 64, 320, 12,
                  hdwnd, NULL, hInst, NULL);

      // Set step increment to 1.
      SendMessage(hProgWnd, PBM_SETSTEP, 1, 0);

      return 1;
    case WM_COMMAND:
      switch(LOWORD(wParam)) {
        case IDCANCEL:
          EndDialog(hdwnd, 0);
          PostQuitMessage(0);

          return 1;

        case IDD_START: // start download
          // Set position to zero.
          SendMessage(hProgWnd, PBM_SETPOS, 0, 0);

          // Get URL from edit box.
          GetDlgItemText(hdwnd, IDD_EB1, url, URL_BUF_SIZE);
          ti.url = url;

          // Get reload status.
          ti.reload = SendDlgItemMessage(hdwnd, IDD_CB1,
                        BM_GETCHECK, 0, 0);

          // Get handle to Start button.
          ti.hPBStart = GetDlgItem(hdwnd, IDD_START);

          // Reset progress counters.
          resetprogress();

          // Start download thread.
          if(!hThrd)
```

```
          hThrd = (HANDLE) _beginthreadex(NULL, 0, dlstart,
          (void *) &ti, 0, (unsigned *) &Tid);

          return 1;
      }
    }

    return 0;
}

// Show progress in the progress bar. This is called
// by the download() function.
void showprogress(unsigned long total,
                  unsigned long part) {

    percentdone = (part*100)/total;

    if(percentdone > oldpercentdone) {
      for(int i= oldpercentdone; i < percentdone; i++) {
        // Advance the progress bar.
        SendMessage(hProgWnd, PBM_STEPIT, 0, 0);
      }
      oldpercentdone = percentdone;
    }
}

// Reset the progress counters.
void resetprogress() {
    percentdone = 0;
    oldpercentdone = 0;
}

// Thread entry function that begins downloading.
unsigned __stdcall dlstart(void * param) {
    ThrdInfo *tip = (ThrdInfo *) param;

    // Disable Start button.
    EnableWindow(tip->hPBStart, 0);

    try {
      if(tip->reload == BST_CHECKED)
        Download::download(tip->url, true, showprogress);
      else
        Download::download(tip->url, false, showprogress);
    } catch(DLExc exc) {
        MessageBox(hwnd, exc.geterr(),
                   "Download Error", MB_OK);
    }

    // Enable Start button.
    EnableWindow(tip->hPBStart, 1);
```

```
  CloseHandle(hThrd); // close the thread handle
  hThrd = 0; // set thread handle to inactive

  return 0;
}
```

WinDL 使用下列源文件：

```
// Resources for file downloader.
#include <windows.h>
#include "windl.h"

DLDB DIALOGEX 18, 18, 164, 100
CAPTION "Download a File"
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION |
      WS_SYSMENU | WS_VISIBLE
{
  PUSHBUTTON "Start", IDD_START, 42, 80, 30, 14
  PUSHBUTTON "Cancel", IDCANCEL, 90, 80, 30, 14

  CTEXT "Download Progress", IDD_TEXT2, 2, 40, 160, 12

CTEXT "Enter URL", IDD_TEXT1, 2, 16, 160, 12
  EDITTEXT IDD_EB1, 2, 1, 160, 12, ES_LEFT | WS_CHILD |
           WS_VISIBLE | WS_BORDER | ES_AUTOHSCROLL

  AUTOCHECKBOX "Reload", IDD_CB1, 62, 56, 36, 14
}
```

源文件和程序代码都需要包含头文件 windl.h，如下所示：

```
#define IDD_START 100

#define IDD_CB1 200

#define IDD_EB1 300

#define IDD_TEXT1 401
#define IDD_TEXT2 402
```

为了编译 WinDL，必须创建一个包含下面文件的项目：

```
dl.cpp    windl.cpp    windl.rc
```

头文件 dl.h 和 windl.h 也必须可用。您一定还要记得链接 wininet.lib 和 comctl32.lib。(进度控制需要 comctl32.lib 库)。最后，由于 download()在它自己的线程内运行，因此必须链接多线程库。

5.5.2 WinDL 的运行方式

WinDL 为 Download 类提供了可视化的前端来处理用户的输入，并显示下载的进度。WinDL 开始时建立了一个最小化的主窗口，然后显示了下载对话框。因此，WinDL 是基于对话框的应

用程序，永远不会显示主窗口。如前所述，对于 WinDL 中支持所有 Windows 程序都通用的基本 Windows 框架部分的描述超出了本书的范围。然而，这些与下载工具有特殊关联的代码会在下面讲述。

处理用户与对话框交互的函数是 DialogFunc()。它声明了两个静态变量。第一个是名为 url 的数组，它为文本编辑框提供了支持数组。第二个是 ti，这是一个很小的结构，包含了将要传递给线程入口函数的信息。

当建立这个对话框时，初始化了文本编辑框和进度条。正如注释提到的那样，这个编辑框用 URL 来初始化只是为了演示(通常，此文本编辑框不会包含初始化字符串)。进度条的增量设置为 1。在默认情况下，进度条的范围为 0 到 100，因此当增量设置为 1 时，进度条的每次增加都是 1%。

为了下载文件，将所需文件的 URL 输入到文本编辑框中，然后单击 Start 按钮。这产生了如下的事件序列。首先，进度条设置为 0。然后从文本编辑框中获取包含了 URL 的字符串，从复选框中获取续传状态，并获取 Start 按钮的句柄。这些都存储在 ti 中，ti 将被传递给线程入口函数。随后，进度计数器被设置为 0。全局变量 percentdone 和 oldpercentdone 用来更新下载期间进度条的位置。最后，启动将处理下载的新线程。

必须在自己的线程中运行 download()，因为 Windows 消息传送系统假定控制权会相对快速地交还给 Windows。(也就是说，DialogFunc()不能采取一些其他的操作阻止新消息的处理)。这个线程的入口函数为 dlstart()。正如第 3 章讲述的那样，在 Windows 中，所有的线程入口函数都只有一个 void*参数。可以给这个参数传递函数需要的任何内容。在此情况下，传递了一个指向 ti 结构的指针，ti 包含了 3 个字段：ur、reload 以及 hPBStart。url 字段指向用来下载的 URL，Reload 复选框的状态存储在 reload 字段中。它决定这个文件是否被完全重新下载。Start 按钮的句柄在 hPBStart 字段中。当开始下载时，dlstart()用它将 Start 按钮变得不可用，当下载结束时，将其设置为可用。

dlstart()函数调用 download()来处理文件的下载。当这样做的时候，它向 update 参数传递 showprogress 的地址。当 download()结束时，Start 按钮可用，线程句柄关闭。

download()调用 showprogress()函数，以显示下载的进度。它只是在文件的百分之一下载完成时，简单地增加进度条。

5.6 尝试完成以下任务

Download 可以添加几个增强的功能。首先您想要尝试的是加入下载给定 FTP 地址的文件的能力。由于 InternetOpenUrl()支持 FTP，因此这个任务相对简单。尽管 HTTP 1.0 版本使用地越来越少，但还是想加入对它的支持。这也很容易，因为只需要总是下载全部的文件就可以了。另一个对下载工具有意义的增强是使得它能够获取一个文件的列表。做到这一点的方法之一是让它从磁盘文件读取 URL。最后，您可能想要添加自动重试功能，在下载被中断时自动尝试完成下载。

一般文件下载机制的应用功能不仅仅是基本的文件下载工具。Download 类可以在任何需要获取文件的时候使用。例如，您可以使用 Download 建立一个远程数据采集程序，从固定的站点下载数据文件，如一个库存报告单。