实 验 八　HTTP 协议应用

8.1　实验目的

介绍超文本传输协议的原理和特点，给合最流行的网页服务器软软件 Apache，开发客户端程序应用编程的方法获取网页，程序对 HTTP 基本命令和协议字段进行解析，深入理解 HTTP 协议的请求应答过程及开发应用。

8.2　超文本传输协议介绍

超文本标记语言，即 HTML（Hypertext Markup Language），是用于描述网页文档的一种标记语言。网页以超文本为主要框架，还包括其它媒体类型，这些媒体文件使用 HTTP 协议由 Web 服务器下载到客户端。HTTP 规定了如何发布和接收 HTML 页面，除了文本数据，还进一步支持传输图片、音频、视频及各种程序文件。客户端使用浏览器软件它基于 HTTP 协议获取网页数据，根据 HTML 语言规范解释组成网页的文本及媒体文件，向用户展示或播放内容。HTTP 是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。

浏览器软件的重要作用就是解释超文本标记语言，它显示文本，图片，播放视频，还具有运行 javascript 语言的能力，能够执行与网页对象相关的程序逻辑。随着人们对网页应用需求的提高，浏览器还进一步支持异步请求技术，它被称为 Ajax 技术，Ajax 的核心是 JavaScript 中的 XmlHttpRequest 对象，它能够在不刷新整个页面的情况下对网页中的内容进行更新。

超文本传输协议是基于 TCP 的应用层协议，它符合客户机/服务端模式，服务端软件提供监听功能向用户提供网页数据，客户端向服务器请求页面，服务器向客户端发回响应数据。请求数据与响应数据由单行或多行文本组成，每行文本以换行符界定，为说明语法方便文本换行命令以 (CRLF) 表示。HTTP(版本 1.1) 中提供 8 种的请求方法，最常用的是 GET 方法和 POST 方法，WEB 服务器必须支持这两种方法。HTTP 请求由三部分组成，分别是：请求行、消息报头、请求正文。请求行以一个方法符号开头，以空格分开，后面跟着请求的 URI 和协议的版本。在浏览器的地址栏中输入网址的方式访问网页时，浏览器采用 GET 方法向服务器获取资源，使用 GET 方法请求网页的命令如下：

GET /index.html HTTP/1.1 (CRLF)

POST 方法用于提交表单，表单具有输入框，用户在网页上输入的数据由浏览器封装后附在协议头后面，服务器将会接收用户数据。Web 服务器接收到客户端发来的请求消息，进行一定解释后返回 HTTP 响应消息。HTTP 协议的响应由三个部分组成，包括状态行、消息报头、响应正文。状态行是单行文本格式如下：

HTTP-Version Status-Code Reason-Phrase (CRLF)

例如：HTTP/1.1 200 OK (CRLF)

这行文本以空格字符间隔为三个部分，其中 HTTP-Version 表示服务器 HTTP 协议的版本；Status-Code 表示服务器发回的响应状态代码；Reason-Phrase 表示状态代码的文本描述。

状态代码有三位数字组成，第一个数字定义了响应的类别，其取值意义分类如下：

1xx：指示信息 --表示请求已接收，继续处理

2xx：成功 --表示请求已被成功接收、理解、接受

3xx：重定向 --要完成请求必须进行更进一步的操作

4xx：客户端错误 --请求有语法错误或请求无法实现

5xx：服务器端错误 --服务器未能实现合法的请求

表8-1例举了部分服务器返回的状态码及其意义说明。

表 8-1 HTTP 状态码部分举例

状态代码	状态描述	说明
200	OK	客户端请求成功
400	Bad Request	客户端请求有语法错误，不能被服务器所理解
401	Unauthorized	请求未经授权，这个状态代码必须和 WWW-Authenticate 报头域一起使用
403	Forbidden	服务器收到请求，但是拒绝提供服务
404	Not Found	请求资源不存在，例如：输入了错误的 URL
500	Internal Server Error	服务器发生不可预期的错误
503	Server Unavailable	服务器当前不能处理客户端的请求，一段时间后可能恢复正常

HTTP 消息报头出现在客户端发出的请求中，也出现在服务端提供的响应。每一个报头域都是由名字 + “:” + 空格 + 值组成，消息报头域的名字是大小写无关的。

Cache-Control 用于指定缓存指令，缓存指令是单向的（响应中出现的缓存指令在请求中未必会出现），且是独立的（一个消息的缓存指令不会影响另一个消息处理的缓存机制），HTTP1.0 使用的类似的报头域为 Pragma。请求时的缓存指令包括：no-cache（用于指示请求或响应消息不能缓存）、no-store、max-age、max-stale、min-fresh、only-if-cached; 响应时的缓存指令包括：public、private、no-cache、no-store、no-transform、must-revalidate、proxy-revalidate、max-age、s-maxage.

Date 普通报头域表示消息产生的日期和时间。Connection 普通报头域允许发送指定连接的选项。例如指定连接是连续，或者指定“close”选项，通知服务器，在响应完成后，关闭连接。

Accept 请求报头域用于指定客户端接受哪些类型的信息。例如：Accept：image/gif，表明客户端希望接受 GIF 图象格式的资源；Accept：text/html，表明客户端希望接受 html 文本。

Accept-Charset 请求报头域用于指定客户端接受的字符集。例如：Accept-Charset:iso-8859-1,gb2312。如果在请求消息中没有设置这个域，缺省是任何字符集都可以接受。

Accept-Encoding 请求报头域类似于 Accept，但是它是用于指定可接受的内容编码。例如：Accept-Encoding:gzip.deflate. 如果请求消息中没有设置这个域服务器假定客户端对各种内容编

码都可以接受。

Accept-Language 请求报头域类似于 Accept，但是它是用于指定一种自然语言。例如：Accept-Language:zh-cn. 如果请求消息中没有设置这个报头域，服务器假定客户端对各种语言都可以接受。

Authorization 请求报头域主要用于证明客户端有权查看某个资源。当浏览器访问一个页面时，如果收到服务器的响应代码为 401（未授权），可以发送一个包含 Authorization 请求报头域的请求，要求服务器对其进行验证。

Host 请求报头域主要用于指定被请求资源的 Internet 主机和端口号，它通常从 HTTP URL 中提取出来的，在浏览器中输入：http://www.zsc.edu.cn/index.html 浏览器发送的请求消息中，就会包含 Host 请求报头域，如下：Host：www.zsc.edu.cn 此处使用缺省端口号 80，若指定了端口号，则变成：Host：www.zsc.edu.cn: 指定端口。

Agent 这个请求报头域中获取到客户机操作系统的名称和版本，浏览器的名称和版本信息。User-Agent 请求报头域允许客户端将它的操作系统、浏览器和其它属性告诉服务器，这个报头域是可选的，如果没有，服务器端就无法得知客户机相关的信息。请求报头举例：

GET /form.html HTTP/1.1 (CRLF)

Accept:image/gif,image/x-xbitmap,image/jpeg,application/x-shockwave-flash,application/vnd.ms-excel,application/vnd.ms-powerpoint,application/msword,*/* (CRLF)

Accept-Language:zh-cn (CRLF)

Accept-Encoding:gzip,deflate (CRLF)

If-Modified-Since:Wed,05 Jan 2007 11:21:25 GMT (CRLF)

If-None-Match:W/"80b1a4c018f3c41:8317" (CRLF)

User-Agent:Mozilla/4.0(compatible;MSIE6.0;Windows NT 5.0) (CRLF)

Host:www.guet.edu.cn (CRLF)

Connection:Keep-Alive (CRLF)

(CRLF)

Location 响应报头域用于重定向客户端连接到一个新的链接位置。Location 响应报头域常用在更换域名的时候。Server 响应报头域包含了服务器用来处理请求的软件信息。例如：

Server：Apache-Coyote/1.1

WWW-Authenticate 响应报头域必须被包含在 401（未授权的）响应消息中，客户端收到 401 响应消息时候，并发送 Authorization 报头域请求服务器对其进行验证时，服务端响应报头就包含该报头域。例如：WWW-Authenticate:Basic realm="Basic Auth Test!" 服务器对请求资源采用的是基本验证机制。

Content-Encoding 实体报头域被用作媒体类型的修饰符，它的值指示了已经被应用到实体正文的附加内容的编码，因而要获得 Content-Type 报头域中所引用的媒体类型，必须采用相应的解码机制。Content-Encoding 这样用于记录文档的压缩方法，例如：Content-Encoding：gzip

Content-Language 实体报头域描述了资源所用的自然语言。没有设置该域则认为实体内容将提供给所有的语言阅读者。例如：Content-Language:da

Content-Length 实体报头域用于指明实体正文的长度，以字节方式存储的十进制数字来表示。

Content-Range 用于表示支持断继传或多线程下载时要下载的数据范围，例如：Content-

Range: bytes 9060352-9244383/9244384

Content-Type 实体报头域用语指明发送给接收者的实体正文的媒体类型。例如：Content-Type:text/html;charset=GB2312

Last-Modified 实体报头域用于指示资源的最后修改日期和时间。

Expires 实体报头域给出响应过期的日期和时间。为了让代理服务器或浏览器在一段时间以后更新缓存中 (再次访问曾访问过的页面时，直接从缓存中加载，缩短响应时间和降低服务器负载) 的页面，Expires 报头域指定页面过期的时间。例如：

Expires：Thu，15 Sep 2006 16:23:12 GMT

8.3 Apache 网页服务器

8.3.1 Apache 软件介绍

Apache 是世界使用排名第一的 Web 服务器软件，市场占有率达 60% 左右。Apache 是自由软件，源代码开放，提供源代码安装方式，用户可使用源代码进行编译安装，它也有不同平台的运行版本如 Uinx、Linux 和 Windows，有广泛的计算机平台，由于其跨平台和安全性被广泛使用。Apache 的特点是简单、速度快、性能稳定，并可做代理服务器来使用。用户可到 httpd.apache.org 下载 Apache 的安装软件，软件的图标是一支羽毛如图8-1所示。

图 8-1 Apache 图标

8.3.2 Apache 安装与配置

本实验下载的 Apache 软件是 2.2.22 版本号的 windows 平台安装版。在安装界面需要设置服务器名称，建议只输入英文字符而不要输入中文字符。

设置 Apache 软件的安装路径，本示例设置为 C:\Apache2\。

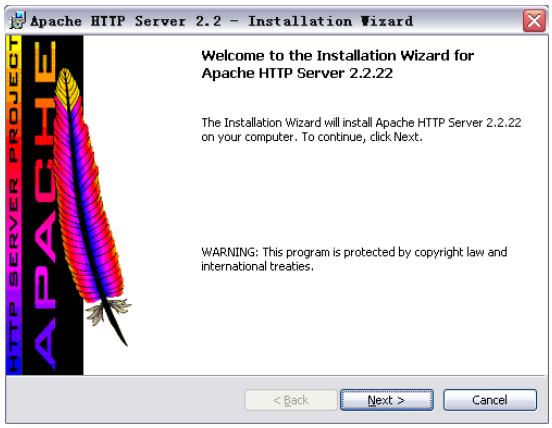

图 8-2 Apache 安装 –1

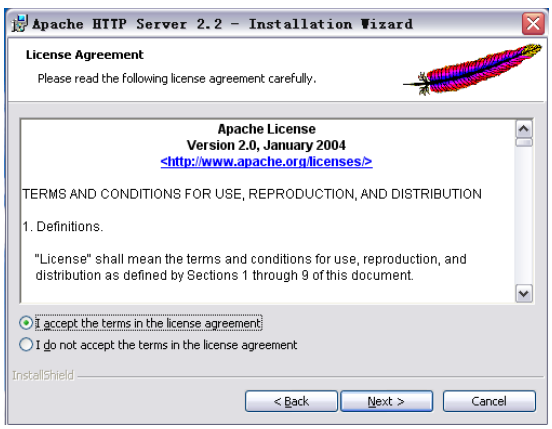

图 8-3 Apache 安装 –2

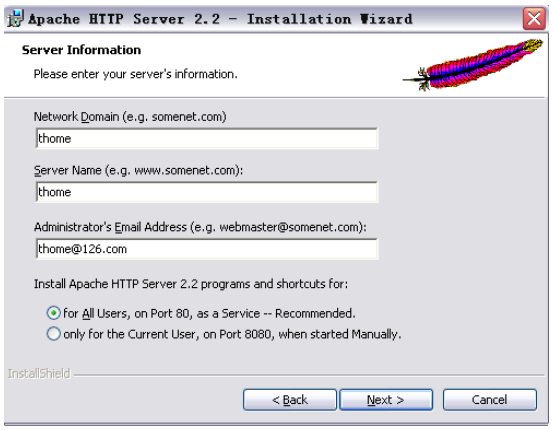

图 8-4 Apache 安装 --服务器名设置

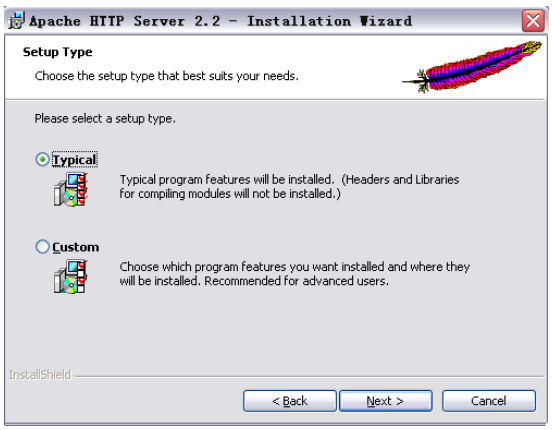

图 8-5 Apache 安装 --选择安装方式为 typical

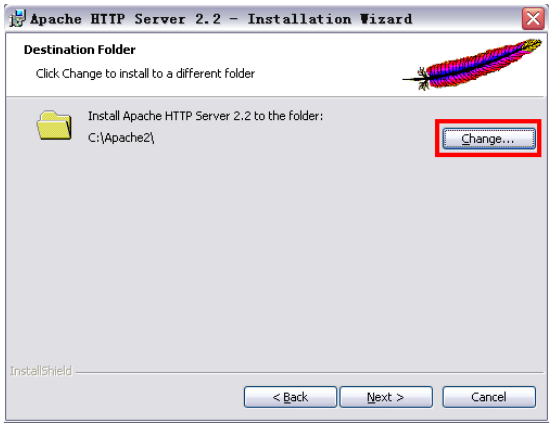

图 8-6 Apache 安装 --设置安装的路径

图 8-7 Apache 服务程序控制

经过基本的参数设置后 Web 服务可顺利安装到本机上，服务程序将会正常启动，如果本机已经有其它程序占用了 80 网络端口，则服务程序无法正常启动，但可通过修改配置文件绑定到其它端口来解决这个问题。在 windows 平台安装完成后，在任务栏的右下角的托盘区将出现 Apache 软件的控制图标，双击后可出现控制窗体如图8-7，可用于 web 服务的启动、停止、重启等操作。在本机浏览器输入地址：http://localhost/，正常情况能够看到一个网页，显示内容为 It works!，表示 Web 服务器软件安装成功。Apache 在 Windows 平台以服务程序运行而没有界面，主配置文件是 C:\Apache2\conf\httpd.conf。httpd.conf 配置 Apache 运行时的多个重要参数，例如 DocumentRoot 参数设置主文档的根目录。

按照前面的介绍安装 Apache 软件，安装成功能够通过浏览器正常查看到本机的网页信息。

8.3.3 网页获取

用户参考图8-8设计界面，单行文本框用于输入网址，中间多行文本框用于显示客户端发出的命令，下方的多行文本框显示服务端的响应消息。

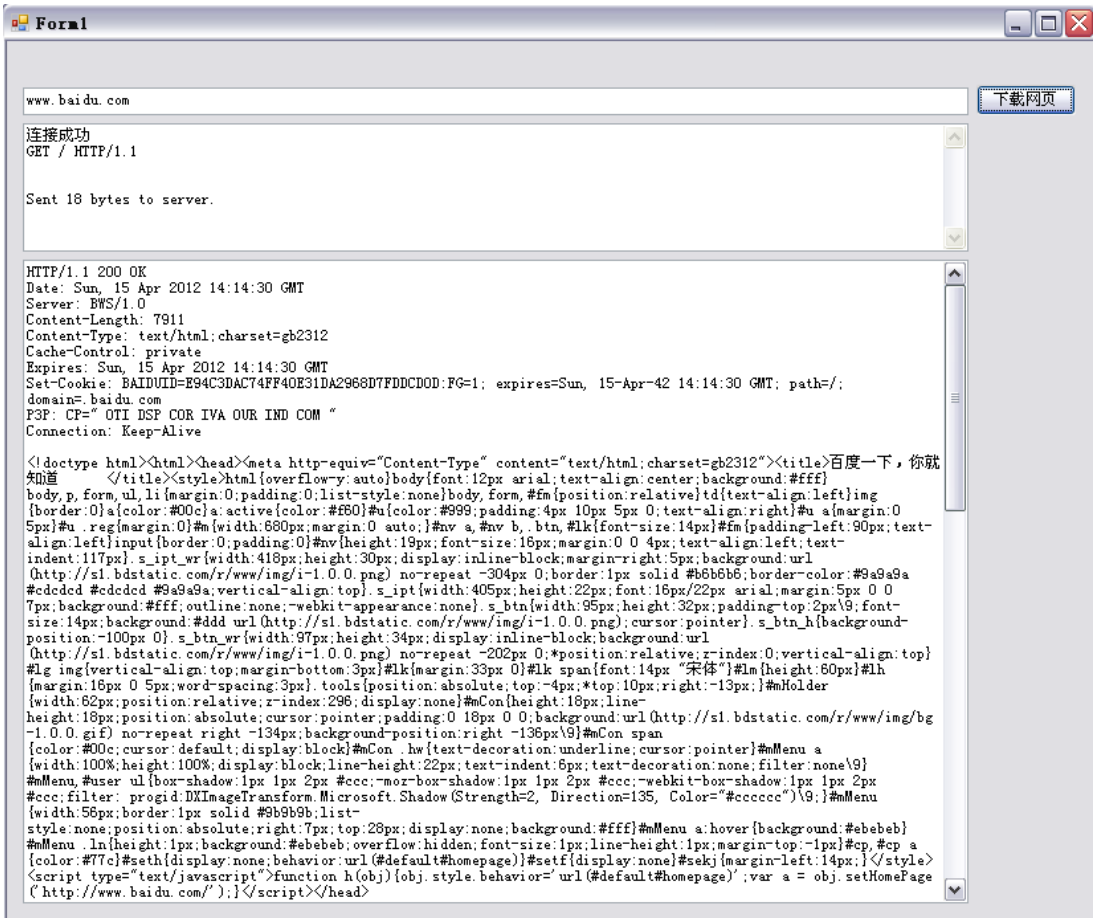

图 8-8 网页获取界面

工作线程使用 Socket 对象采用 TCP 方式连接服务器，向服务器发 HTTP 请求命令，服务器返回响应消息。进行网络通信时，使用 Socket 对象的异步调用方式，共包括三个方法：Connect，Send 和 Receive。

首先是常量和全局变量的定义，其中 ManualResetEvent 对象是用于同步线程的进度。

```
// The port number for the remote device.
private const int port = 80;
// ManualResetEvent instances signal completion.
private static ManualResetEvent connectDone = new ManualResetEvent(false);
private static ManualResetEvent sendDone = new ManualResetEvent(false);
private static ManualResetEvent receiveDone = new ManualResetEvent(false);
// The response from the remote device.
private static String request = String.Empty;
private static String response = String.Empty;
public static string url_str;
public const int UPDATE_SEND = 0x500;
public const int UPDATE_RECEIVE = 0x501;
public static IntPtr main_wnd_handle;
public static MemoryStream ms_recv;
//动态链接库引入
[DllImport("User32.dll", EntryPoint = "SendMessage")]
private static extern int SendMessage(
IntPtr hWnd, // handle to destination window
int Msg, // message
int wParam, // first message parameter
int lParam // second message parameter
);
```

类 StateObject 用在 Socket 对象的异步回调方法中，它封装通信中使用的 Socket 对象和用于接收数据的缓冲区。

```
// State object for receiving data from remote device.
public class StateObject
{
  // Client socket.
  public Socket workSocket = null;
  // Size of receive buffer.
  public const int BufferSize = 256;
  // Receive buffer.
  public byte[] buffer = new byte[BufferSize];
  // Received data string.
  public StringBuilder sb = new StringBuilder();
}
```

对窗体消息处理函数重载用于接收自定义消息。

```
protected override void DefWndProc(ref Message m)
{//窗体消息处理重载
  switch (m.Msg)
```

```
  {
    case UPDATE_SEND:
      //对发送的字节信息进行显示
      textBox1.AppendText(request+"\r\n");
      textBox1.ScrollToCaret();
      break;
    case UPDATE_RECEIVE:
      //对接收的字节信息进行显示
      textBox3.AppendText(response);
      textBox3.ScrollToCaret();
      break;
    default:
      base.DefWndProc(ref m);
      break;
  }
}
```

工作线程由静态函数 thread_GET_html() 定义。

```
static void thread_GET_html()
{
  try
  {
    // Establish the remote endpoint for the socket.
    IPAddress ipAddress = Dns.GetHostEntry(url_str).AddressList[0];
    IPEndPoint remoteEP = new IPEndPoint(ipAddress, port);
    // Create a TCP/IP socket.
    Socket client = new Socket(AddressFamily.InterNetwork,
      SocketType.Stream, ProtocolType.Tcp);
    // Connect to the remote endpoint.
    client.BeginConnect(remoteEP,
      new AsyncCallback(ConnectCallback), client);
    connectDone.WaitOne();
    // Send test data to the remote device.
    request="GET / HTTP/1.1\r\n\r\n";
    Send(client, request);
    SendMessage(main_wnd_handle, UPDATE_SEND, 100, 100);
    sendDone.WaitOne();
    // Receive the response from the remote device.
    Receive(client);
    receiveDone.WaitOne();
    // Release the socket.
```

```
        client.Shutdown(SocketShutdown.Both);
        client.Close();
    }
    catch (Exception e)
    {
        MessageBox.Show(e.ToString());
    }
}
```

Socket 对象异步方法 BeginConnect 指定的回调函数为 ConnectCallback，在成功连接后由事件控制线程的同步。

```
private static void ConnectCallback(IAsyncResult ar)
{
    try
    {
        // Retrieve the socket from the state object.
        Socket client = (Socket)ar.AsyncState;
        // Complete the connection.
        client.EndConnect(ar);
        request = " 连接成功";
        SendMessage(main_wnd_handle, UPDATE_SEND, 100, 100);
        connectDone.Set();
    }
    catch (Exception e)
    {
        MessageBox.Show(e.ToString());
    }
}
```

Send 方法调用 Socket 的异步方法 BeginSend，用于异步方式发送 HTTP 请求命令。

```
private static void Send(Socket client, String data)
{
    // Convert the string data to byte data using ASCII encoding.
    byte[] byteData = Encoding.ASCII.GetBytes(data);
    // Begin sending the data to the remote device.
    client.BeginSend(byteData, 0, byteData.Length, 0,
        new AsyncCallback(SendCallback), client);
}
```

SendCallback 回调函数的定义，它在 Socket 对象成功完成数据发送后被调用。

```
private static void SendCallback(IAsyncResult ar)
{
    try
```

```
    {
        // Retrieve the socket from the state object.
        Socket client = (Socket)ar.AsyncState;
        // Complete sending the data to the remote device.
        int bytesSent = client.EndSend(ar);
        request = String.Format("Sent {0} bytes to server.", bytesSent);
        SendMessage(main_wnd_handle, UPDATE_SEND, 100, 100);
        // Signal that all bytes have been sent.
        sendDone.Set();
    }
    catch (Exception e)
    {
        MessageBox.Show(e.ToString());
    }
}
```

Receive 方法调用 Socket 对象的 BeginReceive 方法。

```
private static void Receive(Socket client)
{
    try
    {
        // Create the state object.
        StateObject state = new StateObject();
        state.workSocket = client;
        // Begin receiving the data from the remote device.
        client.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
            new AsyncCallback(ReceiveCallback), state);
    }
    catch (Exception e)
    {
        MessageBox.Show(e.ToString());
    }
}
```

异步接收数据回调函数 ReceiveCallback 的定义，当接收到服务端响应数据后，此函数被调用，注意此函数在收到的数据长度不为零情况下，会继续调用 Socket 的 BeginReceive 方法，ReceiveCallback 会在数据接收的过程中被递归调用，直到所有数据接收完毕。

```
private static void ReceiveCallback(IAsyncResult ar)
{
    try
    {
        // Retrieve the state object and the client socket
```

```
        // from the asynchronous state object.
        StateObject state = (StateObject)ar.AsyncState;
        Socket client = state.workSocket;
        // Read data from the remote device.
        int bytesRead = client.EndReceive(ar);
        if (bytesRead > 0)
        {
          // There might be more data, so store the data received so far.
          response = Encoding.ASCII.GetString(state.buffer, 0, bytesRead);
          SendMessage(main_wnd_handle, UPDATE_RECEIVE, 100, 100);
          // Get the rest of the data.
          client.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
            new AsyncCallback(ReceiveCallback), state);
        }
        else
        {
          // All the data has arrived; put it in response.
          if (state.sb.Length > 1)
          {
            response = state.sb.ToString();
          }
          // Signal that all bytes have been received.
          receiveDone.Set();
        }
      }
      catch (Exception e)
      {
        MessageBox.Show(e.ToString());
      }
    }
```

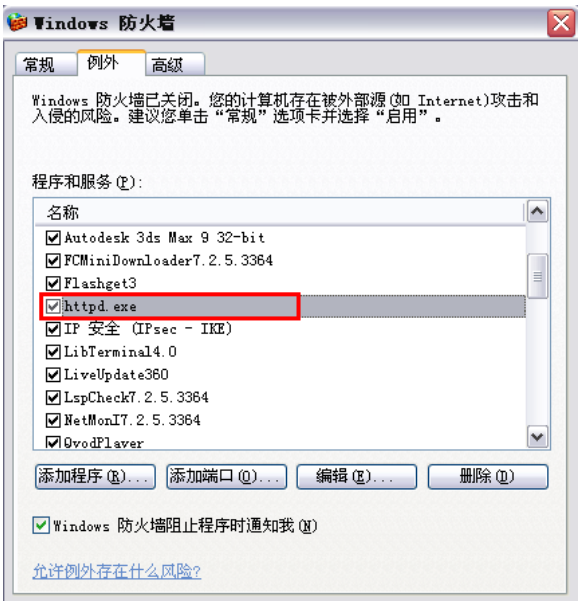

图 8-9 设置 Windows XP 网络防火墙

图 8-10 发送 HEAD 请求包与响应

8.3.4 断点续传与多线程下载

本小节实现 HTTP 协议的断点续传和多线程下载功能。为了模拟实际的网络应用，客户机与服务器都运行在 Windows XP 系统，并且分别位于不同的机器上，实验中 Apache 软件服务器地址为 192.168.1.100，客户机地址为 192.168.1.102，读者应根据自己机器环境进行相应设置。服务器要提供 Web 服务，需设置防火墙例外条件，方法是开始 | 控制面版 | 安全中心 | Windows 防火墙，在例外标签页 | 添加程序，添加程序 c:\Apache2\bin\httpd.exe，图8-9显示防火墙成功添加了 httpd 例外的载图。如果从客户机浏览器能够访问服务器的网页，则说明设置成功。

HTTP 协议不但能够传输 HTML 网页文本，还可以传其它多种格式文件，如 JPG、MP3 和 RAR 等类型文件，其它格式的文件的尺寸往往较大，网络环境容易发生网络连接中断的情况，为应对复杂的网络条件，HTTP 协议支持对大文件的断点续传和多线程下载功能。HTTP 协议有专门的报头域对要传输的文件进行分段标识，Range 实体域用在请求头中，指定第一个字节和最后一个位置：如 Range:200-300，相应的服务端的响应头中有 Content-Range 实体域，它除了响应发来的数据范围，还给出文件的总大小。如 Content-Range: bytes 200-300/2350，其中 2350 指出了文件总大小。

对于客户采用 GET 或者 POST 发出的请求，服务器在响应报文中将协议头部分与实际数据拼在一起发送，这给多线程下载中根据响应的内容来确定线程究竟从何处下载分段的文件数据带了一定的麻烦。例如需要先获得文件的总长度再确定线程要下载的位置，这可以采用 HEAD 请求，服务器收到 HEAD 请求后，与 GET 或者 POST 方式在响应头部分的内容是一样的，区别是服务器不提供数据部分。例如图8-10显示向服务器发送如下的请求包：

```
HEAD /httpd.rar HTTP/1.1
Connection: close
Range: bytes=100-1000
Host: 192.168.1.100
```

User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)

服务器返回的响应消息如下：

```
HTTP/1.1 206 Partial Content
Date: Mon, 07 Jan 2008 14:52:48 GMT
Server: Apache/2.2.22 (Win32)
Last-Modified: Mon, 07 Jan 2008 09:08:33 GMT
ETag: "24000000025534-4bfa49-4431e345ab0e2"
Accept-Ranges: bytes
Content-Length: 901
Content-Range: bytes 100-1000/4979273
Connection: close
Content-Type: application/x-rar-compressed
```

服务端返回的 Content-Range 域指出了文件的大小为 4979273 字节，在这里采用两个线程实例分别下载文件的前半部分与后半部分，下载完成后，将获得的数据合并成一个文件，这就是多线程下载/断点续传的具体实现。编写线程代码函数 MultiThreadsDown，两次启动这个线程，通过每次传入不同的参数指定下载文件的不同部分。

在获取网页的线程里，修改原来的的 GET 请求命令，换成本小节的 HEAD 请求报文，解析收到的返回报文获取要下载的文件大小，参考代码如下：

```
//获取文件长度信息
long dFileLen = 0;
string strFilelen;
string[] responLines = totalrespon.Split("\r\n".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);
for (int i = 0; i < responLines.Length;i++ )
{
  if (responLines[i].IndexOf("Content-Range")>-1)
  {
    strFilelen = responLines[i].Substring(responLines[i].IndexOf("/") + 1);
    dFileLen=Int64.Parse(strFilelen);
    break;
  }
}
```

要启用相同的线程代码两次，生成两个线程实例，需要使用不同的线程参数下载文件的不同部分，要使用的内存对象与线程参数类 ThreadPara 参考定义如下：

```
//用于同步下载线程
static ManualResetEvent[] threadsDone = new ManualResetEvent[2];
//线程使用的内存区，暂存不同的数据部分
public static MemoryStream msFileData0;
public static MemoryStream msFileData1;
//线程参数类，用于标识文件下载位置，使用的内存区
public class ThreadPara
```

```
{
    public long StartPos;
    public long EndPos;
    public ManualResetEvent eventThreadDone;
    public MemoryStream msFData;
    public long dataLen;
}
```

文件多线程下载函数 MultiThreadsDown 的定义如下，注意线程在定义时使用了参数，在启动线程时将根据不同参数值下载文件的不同位置：

```
static void MultiThreadsDown(Object tdata)
{
    try
    {
        // Establish the remote endpoint for the socket.
        IPAddress ipAddress = Dns.GetHostEntry(url_str).AddressList[0];
        IPEndPoint remoteEP = new IPEndPoint(ipAddress, port);
        // Create a TCP/IP socket.
        Socket client = new Socket(AddressFamily.InterNetwork,
            SocketType.Stream, ProtocolType.Tcp);
        // Connect to the remote endpoint.
        client.Connect(remoteEP);
        ThreadPara thdata = (ThreadPara)tdata;
        string httprequest;
        httprequest = "GET /" + httpfile + " HTTP/1.1\r\n";
        httprequest += "Connection: close\r\n";
        httprequest += "Range: bytes=" + thdata.StartPos.ToString() + "-" + thdata.EndPos.ToString()
+ "\r\n";
        httprequest += "Host: 192.168.1.100\r\n";
        httprequest += "User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Tri-
dent/5.0)\r\n\r\n";
        thdata.dataLen = thdata.EndPos - thdata.StartPos+1;
        thdata.msFData.Seek(0, SeekOrigin.Begin);
        byte[] sendData = Encoding.ASCII.GetBytes(httprequest);
        client.Send(sendData, SocketFlags.None);
        byte[] receiveData = new byte[1024];
        Int32 recvLen = client.Receive(receiveData, 1024, SocketFlags.None);
        thdata.msFData.Write(receiveData, 0, recvLen);
        while (recvLen > 0)
        {
            recvLen = client.Receive(receiveData, 1024, SocketFlags.None);
```

```
            thdata.msFData.Write(receiveData, 0, recvLen);
        }
        client.Shutdown(SocketShutdown.Both);
        client.Close();
        thdata.eventThreadDone.Set();
    }
    catch (Exception e)
    {
        MessageBox.Show(e.ToString());
    }
}
```

在本实验中根据文件总长度以及 HTTP 断点续传报头定义生成两个参数，使用两个参数值启动线程两次，参考代码如下：

```
long partpos = dFileLen/2;
msFileData0 = new MemoryStream(5000000);
msFileData1 = new MemoryStream(5000000);
threadsDone[0] = new ManualResetEvent(false);
threadsDone[1] = new ManualResetEvent(false);
//初始化线程参数
ThreadPara thrPara0 = new ThreadPara();
thrPara0.msFData = msFileData0;
thrPara0.eventThreadDone = threadsDone[0];
thrPara0.StartPos = 0;
thrPara0.EndPos = partpos;
ThreadPara thrPara1 = new ThreadPara();
thrPara1.msFData = msFileData1;
thrPara1.eventThreadDone = threadsDone[1];
thrPara1.StartPos = partpos+1;
thrPara1.EndPos = dFileLen-1;
//启动线程下载文件不同部分
Thread Thread0 = new Thread(new ParameterizedThreadStart(MultiThreadsDown));
Thread0.Start(thrPara0);
Thread Thread1 = new Thread(new ParameterizedThreadStart(MultiThreadsDown));
Thread1.Start(thrPara1);
//同步两个下载线程的结束
WaitHandle.WaitAll(threadsDone);
```

下载线程收到的服务器响应报文是报文头部分 + 文件数据部分，需要界定出报文头部分，报文头部分在结束时使用连续的 \r\n\r\n 与数据区别，其字节值为 13、10、13、10，合并生成文件时只复制数据部分，参考代码如下：

```
//合并文件内容
```

```
byte[] temBuf=new byte[1024];
string filename = "d:\\" + httpfile ;
FileStream fs = new FileStream(filename, FileMode.CreateNew);
ThreadPara[] thParas = new ThreadPara[2];
thParas[0] = thrPara0;
thParas[1] = thrPara1;
for (int i = 0; i < 2;i++ )
{
  thParas[i].msFData.Seek(0, SeekOrigin.Begin);
  byte[] tData = new byte[4];
  int j = 0;
  //确定报头后的数据位置
  for (; j < 1000; j++)
  {
    thParas[i].msFData.Seek(j, SeekOrigin.Begin);
    thParas[i].msFData.Read(tData, 0, 4);
    if ((tData[0] == 13) && (tData[1] == 10) && (tData[2] == 13) && (tData[3] == 10))
    {
      thParas[i].msFData.Seek(j + 4, SeekOrigin.Begin);
      break;
    }
  }
  long totalLen = 0;
  int writeLen = 0;
  do
  {
    writeLen = thParas[i].msFData.Read(temBuf, 0, 1024);
    fs.Write(temBuf, 0, writeLen);
    totalLen += writeLen;
  }
  while (totalLen < thParas[i].dataLen);
}
fs.Flush();
fs.Close();
fs.Dispose();
```

适当修改获取网页的线程 thread_GET_html，启用两个线程下载文件数据。实际的网络平台会有一定变化，需要用户根据实际情况调整程序以适应。

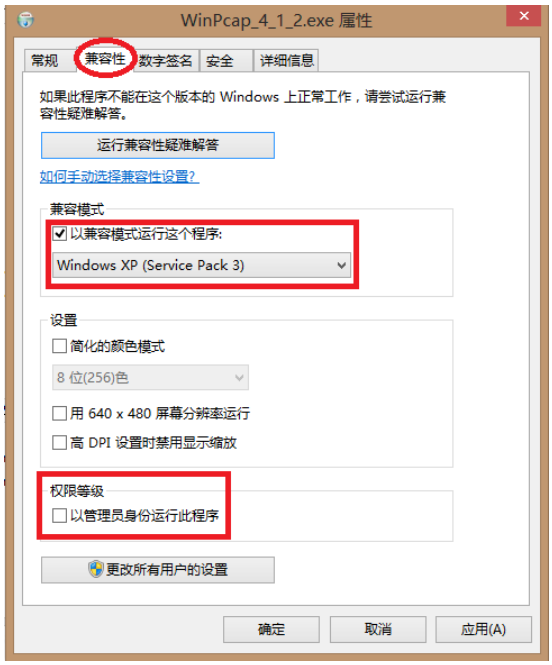

图 8-11 设置 Wireshark 的兼容安装方式

图 8-12 忽略软件兼容性提示

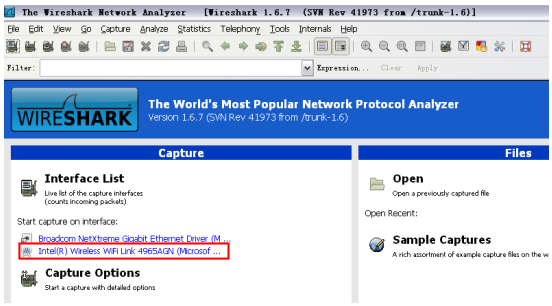

图 8-13 WireShark 软件初始运行界面

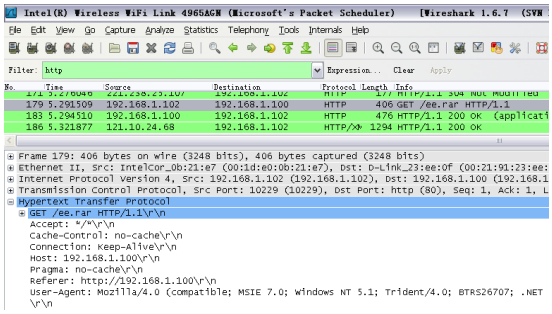

图 8-14 WireShark 抓取网络数据包界面

8.4 WireShark 抓包软件

网络数据在局域网内以网络广播方式传播，可通过软件方式获取同网段内数据包，比较出名的软件有 sniffer 和 WireShark，本实验内容基于 WireShark 软件进行网络协议分析。

WireShark 软件曾叫作"Ethereal"，它是一个网络数据分析工具，可以在线捕获网络数据包，网络管理人员使用它监视网络或者进行故障排错。WireShark 软件还是用于研究学习多种网络协议的优秀工具，抓取已有软件的网络包通过模拟的方法能迅速掌握协议应用。WireShark 软件包含一个名为 WinpCap 的软件包，WinpCap 完成抓包的任务。在 Windows XP 平台安装 WireShark 软件的过程非常简单，如果在 Win8 平台安装这个软件包时需设置为兼容的运行模式，并且要以管理员身份运行，右键点击软件后选择属性菜单项进行设置，操作如图8-11所示，忽略安装过程的兼容性提示如图8-12所示。

Wireshark 软件成功安装后的初始运行界面如图8-13。

在 WireShark 软件初始运行界面，列举了机器上可用的网卡列表 (Interface List)，图8-13表示了 Brodcom 的以太网卡和 Intel 的 4965AGN 无线网卡。点击需要进行抓包的网卡，

WireShark 马上进行抓包过程，机器在运行时一般都会有大量的实时网络数据包，抓包时间不宜过长，要及时停止抓包过程，防止产生的抓包文件过大。例如用户希望掌握实际的 HTTP 协议的请求包和服务器的响应数据包，操作方法如下：

1. 启动 WireShark 抓包功能；
2. 让浏览器向服务器发送请求，例如输入地址 http://192.168.1.100/ee.rar；
3. 得到服务器请求后，停止 WireShark 的抓包功能；
4. 在过滤条件栏中输入 http，只查看 http 协议数据，找到目标地址为 192.168.1.100 的记录行；
5. 观察和学习 http 协议的请求数据包和响应数据包；

图8-14演示了从地址 http://192.168.1.100/ee.rar 下载文件的网络抓包载图。

8.5 实验作业

1. 获取本机主页。
2. 应用程序代码获取百度网页文本。
3. 应用程序代码获取网易（或其它门户型网站）网页文本，将网页正文与服务器响码进行分离，比较自己的程序与使用浏览器的查看源代码获取的网页文本时否相同，请分析原因。
4. 调试程序实现多线程下载文件任务。