# TCP/IP

## TCP/IP定义

TCP/IP（Transmission Control Protocol/Internet Protocol）即传输控制协议/网间协议，又叫网络通讯协议，这个协议是Internet最基本的协议、Internet国际互联网络的基础，简单地说，就是由网络层的IP协议和传输层的TCP协议组成的。

TCP/IP 是供已连接因特网的计算机进行通信的通信协议。 TCP/IP 定义了电子设备（比如计算机）如何连入因特网，以及数据如何在它们之间传输的标准。

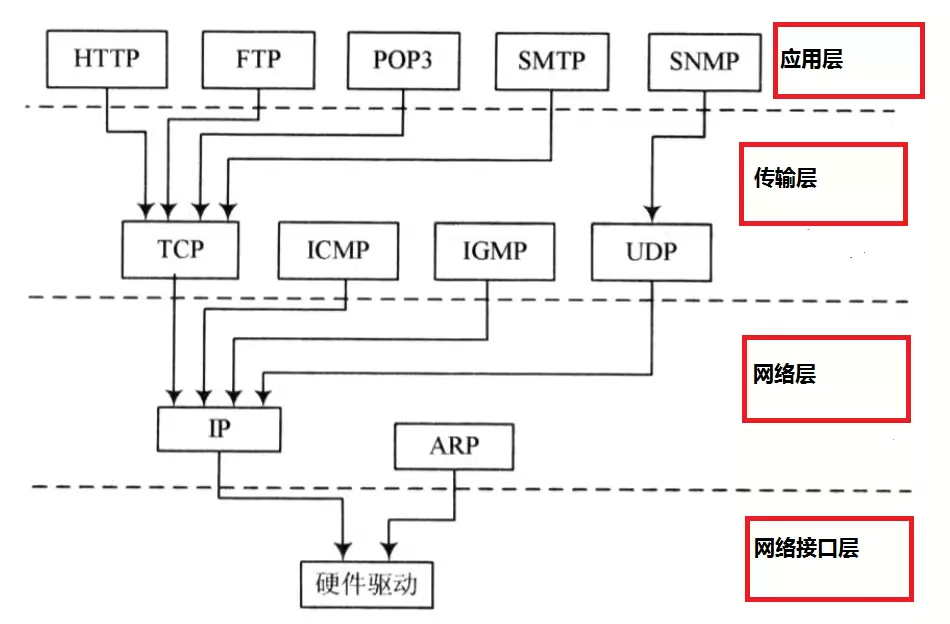
## 四层体系结构

　应用层：应用程序间沟通的层，如简单电子邮件传输（SMTP）、文件传输协议（FTP）、网络远程访问协议（Telnet）等。

　　传输层：在此层中，它提供了节点间的数据传送，应用程序之间的通信服务，主要功能是数据格式化、数据确认和丢失重传等。如传输控制协议（TCP）、用户数据报协议（UDP）等，TCP和UDP给数据包加入传输数据并把它传输到下一层中，这一层负责传送数据，并且确定数据已被送达并接收。

　　互连网络层：负责提供基本的数据封包传送功能，让每一块数据包都能够到达目的主机（但不检查是否被正确接收），如网际协议（IP）。

　　网络接口层（以太网设备驱动程序）：接收IP数据包并进行传输，从网络上接收物理帧，抽取IP数据报转交给下一层，对实际的网络媒体的管理，定义如何使用实际网络（如Ethernet、Serial Line等）来传送数据。



### IP

网际协议IP是TCP/IP的心脏，也是网络层中最重要的协议。

IP层接收由更低层（网络接口层例如以太网设备驱动程序）发来的数据包，并把该数据包发送到更高层---TCP或UDP层；相反，IP层也把从TCP或UDP层接收来的数据包传送到更低层。IP数据包是不可靠的，因为IP并没有做任何事情来确认数据包是按顺序发送的或者没有被破坏。IP数据包中含有发送它的主机的地址（源地址）和接收它的主机的地址（目的地址）。

高层的TCP和UDP服务在接收数据包时，通常假设包中的源地址是有效的。也可以这样说，IP地址形成了许多服务的认证基础，这些服务相信数据包是从一个有效的主机发送来的。IP确认包含一个选项，叫作IP source routing，可以用来指定一条源地址和目的地址之间的直接路径。对于一些TCP和UDP的服务来说，使用了该选项的IP包好像是从路径上的最后一个系统传递过来的，而不是来自于它的真实地点。这个选项是为了测试而存在的，说明了它可以被用来欺骗系统来进行平常是被禁止的连接。那么，许多依靠IP源地址做确认的服务将产生问题并且会被非法入侵。

### TCP

TCP提供一种面向连接的、可靠的字节流服务。面向连接意味着两个使用TCP的应用（通常是一个客户和一个服务器）在彼此交换数据包之前必须先建立一个TCP连接。这一过程与打电话很相似，先拨号振铃，等待对方摘机说“喂”，然后才说明是谁。在一个TCP连接中，仅有两方进行彼此通信。广播和多播不能用于TCP。

TCP会对数据进行分包和校验。

### UDP

UDP 是User Datagram Protocol的简称， 中文名是用户数据报协议。与TCP处于同一层，与TCP协议一样用于处理数据包，是一种无连接的协议。UDP有不提供数据包分组、组装和不能对数据包进行排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。

在选择使用协议的时候，选择UDP必须要谨慎。在网络质量令人十分不满意的环境下，UDP协议数据包丢失会比较严重。但是由于UDP的特性：它不属于连接型协议，因而具有资源消耗小，处理速度快的优点，所以通常音频、视频和普通数据在传送时使用UDP较多，因为它们即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。比如我们聊天用的ICQ和QQ就是使用的UDP协议。

# Socket

## 定义

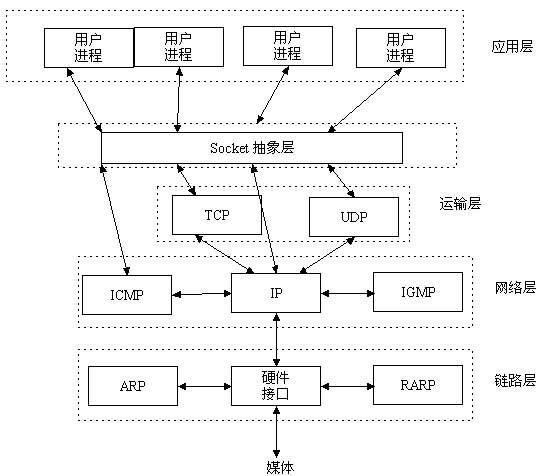
socket的英文原义是“孔”或“插座”。作为进程通信机制，取后一种意思。通常也称作“套接字”，用于描述IP地址和端口，是一个通信链的句柄。(其实就是两个程序通信用的。)

socket非常类似于电话插座。以一个电话网为例。电话的通话双方相当于相互通信的2个程序，电话号码就是IP地址。任何用户在通话之前，首先要占有一部电话机，相当于申请一个socket；同时要知道对方的号码，相当于对方有一个固定的socket。然后向对方拨号呼叫，相当于发出连接请求。对方假如在场并空闲，拿起电话话筒，双方就可以正式通话，相当于连接成功。双方通话的过程，是一方向电话机发出信号和对方从电话机接收信号的过程，相当于向socket发送数据和从socket接收数据。通话结束后，一方挂起电话机相当于关闭socket，撤消连接。

## Socket抽象层

Socket是位于应用层和运输层之间的抽象层。Socket是对TCP/IP协议的封装，Socket本身并不是协议，而是一个调用接口(API)。通过Socket，我们才能使用TCP/IP协议。

Socket的出现只是使得程序员更方便地使用TCP/IP协议栈而已，是对TCP/IP协议的抽象，



## 端口的分类

端口分为以下三类：

（1）公认端口（Well Known Ports）：从0到1023，它们紧密绑定（binding）于一些服务。通常这些端口的通讯 明确表明了某种服务的协议。例如：80端口实际上总是HTTP通讯。

（2）注册端口（Registered Ports）：从1024到49151。它们松散地绑定于一些服务。也就是说有许多服务绑定于 这些端口，这些端口同样用于许多其它目的。例如：许多系统处理动态端口从1024左右开始。

（3）动态和/或私有端口（Dynamic and/or Private Ports）：从49152到65535。理论上，不应为服务分配这些端 口。实际上，机器通常从1024起分配动态端口。但也有例外：SUN的RPC端口从32768开始。

## 端口与应用程序

在Internet上有很多这样的主机，这些主机一般运行了多个服务软件，同时提供几种服务。每种服务都打开一个Socket，并绑定到一个端口上，不同的端口对应于不同的服务(应用程序)。

例如：http 使用80端口。 ftp使用21端口。 smtp 25端口。23端口主要用于Telnet（远程登录）53端口为DNS（Domain Name Server，域名服务器。

## Socket分类

有两种类型：

流式Socket（STREAM）：  
是一种面向连接的Socket，针对于面向连接的TCP服务应用，安全，但是效率低；

数据报式Socket（DATAGRAM）：  
是一种无连接的Socket,对应于无连接的UDP服务应用.不安全(丢失,顺序混乱,在接收端要分析重排及要求重发),但效率高.

## Socket应用模式（客户端、服务器端）

### 服务器端的Socket(至少需要两个)

一个负责接收客户端连接请求(但不负责与客户端通信)

每成功接收到一个客户端的连接便在服务端产生一个对应的负责通信的Socket

在接收到客户端连接时创建.

为每个连接成功的客户端请求在服务端都创建一个对应的Socket(负责和客户端通信).

### 客户端的Socket

客户端Socket

必须指定要连接的服务端地址和端口。

通过创建一个Socket对象来初始化一个到服务器端的TCP连接。

### Socket通信过程

* 服务器端：

申请一个socket

绑定到一个IP地址和一个端口上

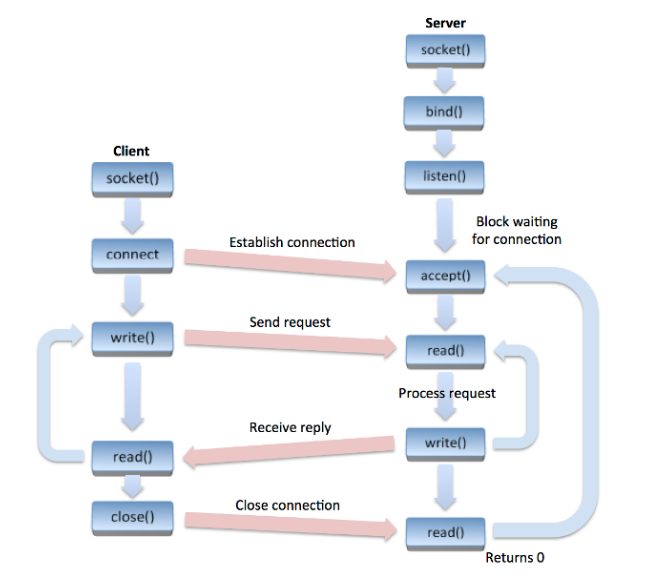
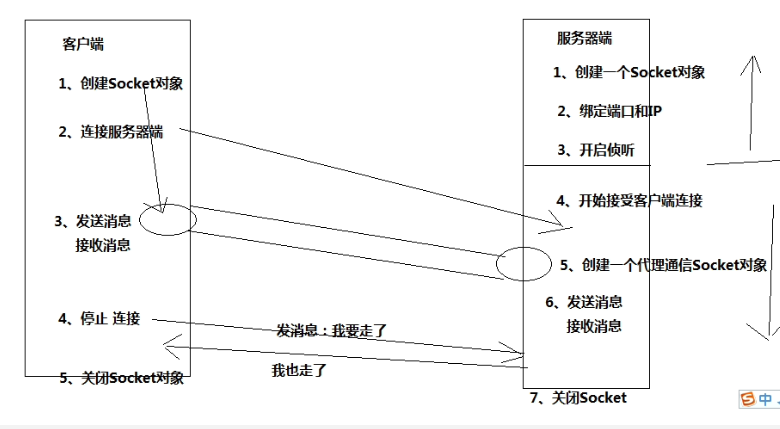
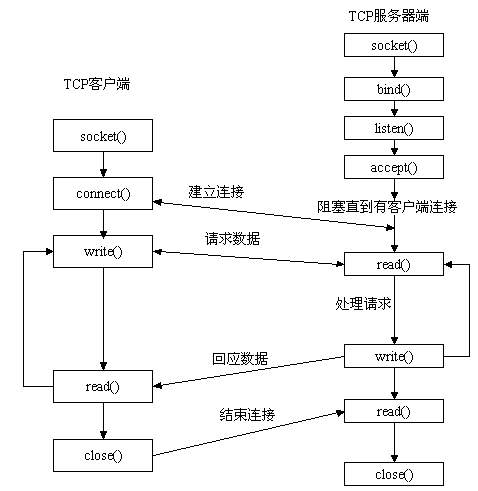
开启侦听，等待接授连接

* 客户端：

申请一个socket

连接服务器（指明IP地址和端口号）

* 服务器端接到连接请求后，产生一个新的socket(端口大于1024）与客户端建立连接并进行通讯，原监听socket继续监听。
* 断开连接时，客户端和服务器端会互相进行断开连接的通信，
* 最后，关闭连接，并释放socect对象。



# 服务器端

## 服务器一般创建流程

* 创建socket监听对象
* 绑定ip和端口
* 开始监听连接
* 有客户端连接时，再创建一个代理socket用来通信。
* 通过代理socket发送消息。
* 停止通信即停止接受和发送信息
* 释放代理资源
* 释放socket监听对象

在下面的代码中，点击按钮，创建服务端，发送当前时间到客户端。最后关闭客户端。

|  |
| --- |
| private void button1\_Click(object sender, EventArgs e)  {  //1.创建socket对象  //第一个参数：网络寻址的协议。InterNetwork表示为ipv4方式  //第二个参数：socket类型，定义数据传输的方式。Stream流式，双向链接传输针对TCP  //第三个参数：通信的协议。  Socket serverListenSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);  this.txtMsg.Text = "创建服务器Socket监听对象\r\n" + this.txtMsg.Text;  //2.绑定IP和端口  IPAddress ip = IPAddress.Parse(txtIP.Text.Trim());  IPEndPoint ipEndPoint = new IPEndPoint(ip, int.Parse(txtPort.Text.Trim()));  serverListenSocket.Bind(ipEndPoint);  //3.开始侦听  //参数为等待连接的队列的长度。同时来了100个请求，只能处理一个链接，队列里面放10个等待连接的客户端，其他的返回错误信息。  serverListenSocket.Listen(10);  this.txtMsg.Text = "开始接收客户端连接\r\n" + this.txtMsg.Text;  //4. 当有客户端连接时，会创建一个socket对象用来进行通信。  //accept方法一执行，当前线程阻塞，一直等到客户端连接上。  Socket proxySocket = serverListenSocket.Accept();  this.txtMsg.Text = string.Format("客户端:{0}连接上\r\n{1}", proxySocket.RemoteEndPoint.ToString(), this.txtMsg.Text);  //5.发送消息  string str = "北京时间：" + DateTime.Now.ToString();  byte[] bytes = Encoding.Default.GetBytes(str);  //以字节的方式发送数据  proxySocket.Send(bytes, 0, bytes.Length, SocketFlags.None);  //6.停止通信。SocketShutdown.Both表示不再接收和发送消息  proxySocket.Shutdown(SocketShutdown.Both);  this.txtMsg.Text = "关闭\r\n" + this.txtMsg.Text;  //7。关闭连接，释放资源  proxySocket.Close();  serverListenSocket.Close();  } |

## Accept阻塞

在3.1节中，执行时主窗体会“卡死”。这是因为当执行Accept时会阻塞当前线程，没有办法再操作主窗体了。因此应当把accept放到一个线程中。

首先创建一个用来传递给委托的方法

|  |
| --- |
| private void StartAcceptClient(object state)  {  //接受传递将来的socket对象并转换为Socket类型  Socket serverListenSocket = state as Socket;  //一直监听客户端连接  while (true)  {  Socket proxySocket = serverListenSocket.Accept();  this.txtMsg.Text = string.Format("客户端:{0}连接上\r\n{1}", proxySocket.RemoteEndPoint.ToString(), this.txtMsg.Text);  string str = "北京时间：" + DateTime.Now.ToString();  byte[] bytes = Encoding.Default.GetBytes(str);  proxySocket.Send(bytes, 0, bytes.Length, SocketFlags.None);  proxySocket.Shutdown(SocketShutdown.Both);  this.txtMsg.Text = "关闭\r\n" + this.txtMsg.Text;  proxySocket.Close();  //不要关闭监听socket对象，否则一个客户端连接后，就不再监听连接，第二个客户端连接会出现异常  // serverListenSocket.Close();  }  } |

在上面的代码中执行到txtMsg时会报异常，因为txtMsg不在这个线程中，我们暂时设置CheckForIllegalCrossThreadCalls为false来实现跨线程调用问题。

|  |
| --- |
| public Form1()  {  InitializeComponent();  Control.CheckForIllegalCrossThreadCalls = false;  } |

在主函数中用线程池调用委托方法，并把监听对象作为参数传递进来

|  |
| --- |
| private void button1\_Click(object sender, EventArgs e)  {  Socket serverListenSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);  this.txtMsg.Text = "创建服务器Socket监听对象\r\n" + this.txtMsg.Text;  IPAddress ip = IPAddress.Parse(txtIP.Text.Trim());  IPEndPoint ipEndPoint = new IPEndPoint(ip, int.Parse(txtPort.Text.Trim()));  serverListenSocket.Bind(ipEndPoint);  serverListenSocket.Listen(10);  this.txtMsg.Text = "开始接收客户端连接\r\n" + this.txtMsg.Text;  //开辟线程，传递监听socket对象  ThreadPool.QueueUserWorkItem(StartAcceptClient, serverListenSocket);  } |

这样再执行时就不会阻塞主窗体，多个客户端也能连接了。

## 群发消息

在上面的代码中，客户端连接上服务器后，自动发送当前时间到客服端，这里我们想要实现点击按钮发送文本框中的消息到所有连接的客户端。

首先创建一个Socket类型的List：

|  |
| --- |
| public partial class Form1 : Form  {    List<Socket> listProxySocket = new List<Socket>(); |

当客户端连接时，把代理的socket对象放到集合中

|  |
| --- |
| private void StartAcceptClient(object state)  {  Socket serverListenSocket = state as Socket;  while (true)  {  Socket proxySocket = serverListenSocket.Accept();  this.txtMsg.Text = string.Format("客户端:{0}连接上\r\n{1}", proxySocket.RemoteEndPoint.ToString(), this.txtMsg.Text);  //把代理socket放入集合  listProxySocket.Add(proxySocket);  }  } |

点击按钮时，通过代理socket群发消息

|  |
| --- |
| private void button2\_Click(object sender, EventArgs e)  {  string msg = this.txtSend.Text;  byte[] bytes = Encoding.Default.GetBytes(msg);  //变量list群发消息。  foreach (var item in listProxySocket)  {  item.Send(bytes, 0, bytes.Length, SocketFlags.None);  }  } |

## 接受消息

|  |
| --- |
| private void StartAcceptClient(object state)  {  Socket serverListenSocket = state as Socket;  while (true)  {  Socket proxySocket = serverListenSocket.Accept();  this.txtMsg.Text = string.Format("客户端:{0}连接上\r\n{1}", proxySocket.RemoteEndPoint.ToString(), this.txtMsg.Text);  listProxySocket.Add(proxySocket);  //接受消息  //缓冲区，每次可以接受字节数量  byte[] date = new byte[1024\*1024];  while (true)  {  //接受到的信息为字节数组。返回值为数组长度  int len = proxySocket.Receive(date, 0, date.Length, SocketFlags.None);  string fromClientMsg = Encoding.Default.GetString(date, 0, len);  this.txtMsg.Text = string.Format("接受到客户端{0}的消息：{1}\r\n{2}", proxySocket.RemoteEndPoint.ToString(), fromClientMsg, txtMsg.Text);  }  } |

上面的代码，当第二个客户端连接时，无法接受和发送信息这时因为Receive也会阻塞当前线程即Accept无法执行，创建不了代理socket，也就无法与第二个客户端进行通信。

## Receive阻塞

与Accept一样，需要为Receive创建线程，来解决阻塞

|  |
| --- |
| private void RecieveMsgFromClient(object state)  {  Socket proxySocket = state as Socket;  byte[] date = new byte[1024 \* 1024];  int len =-1;  while (true)  {    try//异常退出时  {  //接受到的信息为字节数组。返回值为字节长度  len = proxySocket.Receive(date, 0, date.Length, SocketFlags.None);  }  catch (Exception e)  {  MessageBox.Show(e.Message);  }  //当接受到信息的长度为0时，表示客户端退出  if (len == 0)  {  this.txtMsg.Text = string.Format("客户端：{0}已关闭\r\n{1}", proxySocket.RemoteEndPoint.ToString(), txtMsg.Text);  //关闭代理socket  proxySocket.Shutdown(SocketShutdown.Both);  proxySocket.Close();  return;  }  string fromClientMsg = Encoding.Default.GetString(date, 0, len);  this.txtMsg.Text = string.Format("接受到客户端{0}的消息：{1}\r\n{2}", proxySocket.RemoteEndPoint.ToString(), fromClientMsg, txtMsg.Text);  }  } |

在StartAcceptClient创建的线程中，把RecieveMsgFromClient放入线程池。

|  |
| --- |
| private void StartAcceptClient(object state)  {  Socket serverListenSocket = state as Socket;  while (true)  {  Socket proxySocket = serverListenSocket.Accept();  this.txtMsg.Text = string.Format("客户端:{0}连接上\r\n{1}", proxySocket.RemoteEndPoint.ToString(), this.txtMsg.Text);  listProxySocket.Add(proxySocket);  //接受消息  ThreadPool.QueueUserWorkItem(RecieveMsgFromClient, proxySocket);  }  } |

## 完整代码

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Net;  using System.Net.Sockets;  using System.Text;  using System.Threading;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace Socket编程  {  public partial class Form1 : Form  {  List<Socket> listProxySocket = new List<Socket>();  public Form1()  {  InitializeComponent();  Control.CheckForIllegalCrossThreadCalls = false;  }  private void button1\_Click(object sender, EventArgs e)  {  Socket serverListenSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);  this.txtMsg.Text = "创建服务器Socket监听对象\r\n" + this.txtMsg.Text;  IPAddress ip = IPAddress.Parse(txtIP.Text.Trim());  IPEndPoint ipEndPoint = new IPEndPoint(ip, int.Parse(txtPort.Text.Trim()));  serverListenSocket.Bind(ipEndPoint);  serverListenSocket.Listen(10);  this.txtMsg.Text = "开始接收客户端连接\r\n" + this.txtMsg.Text;  //开辟线程，传递监听socket对象  ThreadPool.QueueUserWorkItem(StartAcceptClient, serverListenSocket);  }  private void StartAcceptClient(object state)  {  Socket serverListenSocket = state as Socket;  while (true)  {  Socket proxySocket = serverListenSocket.Accept();  this.txtMsg.Text = string.Format("客户端:{0}连接上\r\n{1}", proxySocket.RemoteEndPoint.ToString(), this.txtMsg.Text);  listProxySocket.Add(proxySocket);  //接受消息  ThreadPool.QueueUserWorkItem(RecieveMsgFromClient, proxySocket);  }  }  private void RecieveMsgFromClient(object state)  {  Socket proxySocket = state as Socket;  byte[] date = new byte[1024 \* 1024];  int len =-1;  while (true)  {    try//异常退出时  {  //接受到的信息为字节数组。返回值为字节长度  len = proxySocket.Receive(date, 0, date.Length, SocketFlags.None);  }  catch (Exception e)  {  this.txtMsg.Text = string.Format("客户端：{0}非正常退出\r\n{1}", proxySocket.RemoteEndPoint.ToString(), txtMsg.Text);  //关闭代理socket  proxySocket.Shutdown(SocketShutdown.Both);  proxySocket.Close();  listProxySocket.Remove(proxySocket);  return;  }  //当接受到信息的长度为0时，表示客户端退出  if (len == 0)  {  this.txtMsg.Text = string.Format("客户端：{0}已关闭\r\n{1}", proxySocket.RemoteEndPoint.ToString(), txtMsg.Text);  //关闭代理socket  proxySocket.Shutdown(SocketShutdown.Both);  proxySocket.Close();  listProxySocket.Remove(proxySocket);  return;  }  string fromClientMsg = Encoding.Default.GetString(date, 0, len);  this.txtMsg.Text = string.Format("接受到客户端{0}的消息：{1}\r\n{2}", proxySocket.RemoteEndPoint.ToString(), fromClientMsg, txtMsg.Text);  }  }  private void button2\_Click(object sender, EventArgs e)  {  string msg = this.txtSend.Text;  byte[] bytes = Encoding.Default.GetBytes(msg);  //变量list群发消息。  foreach (var item in listProxySocket)  {  item.Send(bytes, 0, bytes.Length, SocketFlags.None);  }  }  }  } |

# 案例：完整的服务器与客户端

## 服务器端

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Net;  using System.Net.Sockets;  using System.Text;  using System.Threading;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace \_02\_服务端完整版  {  public partial class Form1 : Form  {  List<Socket> socketList = new List<Socket>();  public Form1()  {  InitializeComponent();  }  #region 开启服务  private void btmStartServer\_Click(object sender, EventArgs e)  {  //1 创建Socket  Socket ListenSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);  //2 绑定端口ip  ListenSocket.Bind(new IPEndPoint(IPAddress.Parse(txtIP.Text), int.Parse(txtPort.Text)));  //3 开启侦听  ListenSocket.Listen(10);  //4 开辟线程，开始接受客户端的连接  ThreadPool.QueueUserWorkItem(new WaitCallback(CreateCommunicationSocket), ListenSocket);  }  #endregion  #region 接受信息  private void ReceiveMsgFromClient(object state)  {  Socket commnunicationSocekt = state as Socket;  if(commnunicationSocekt ==null)  {  return;  }  byte[] bytes = new byte[1024 \* 1024];  while (true)  {  int receiveNums = 0;  try  {  receiveNums = commnunicationSocekt.Receive(bytes, 0, bytes.Length, SocketFlags.None);  }  catch  {  //异常退出  AppendMsgToText(string.Format("客户端：{0}异常退出", commnunicationSocekt.RemoteEndPoint.ToString()));  socketList.Remove(commnunicationSocekt);  StopConnection(commnunicationSocekt);  return;  }  if (receiveNums == 0)//正常退出  {  AppendMsgToText(string.Format("客户端：{0}已正常关闭。", commnunicationSocekt.RemoteEndPoint.ToString()));  socketList.Remove(commnunicationSocekt);  StopConnection(commnunicationSocekt);  return;  }  //接受的信息  string receiveMsg = Encoding.Default.GetString(bytes, 0, receiveNums);  AppendMsgToText(string.Format("客户端：{0} 发来消息：{1}", commnunicationSocekt.RemoteEndPoint.ToString(), receiveMsg));  }  }  #endregion  #region 关闭连接释放资源  private void StopConnection(Socket socket)  {  if (socket.Connected)  {  socket.Shutdown(SocketShutdown.Both);  socket.Close();  }  }  #endregion  #region 创建通信socket，并接受信息  private void CreateCommunicationSocket(object state)  {  Socket ListenSocket = state as Socket;  while (true)  {  Socket commnunicationSocekt = ListenSocket.Accept();  socketList.Add(commnunicationSocekt);  AppendMsgToText(string.Format("客户端：{0}已连接上", commnunicationSocekt.RemoteEndPoint.ToString()));  //开辟线程，接受信息。  ThreadPool.QueueUserWorkItem(ReceiveMsgFromClient, commnunicationSocekt);  }  }  #endregion  #region 发送消息  private void btnSend\_Click(object sender, EventArgs e)  {  byte[] bytes = Encoding.Default.GetBytes(txtSendMsg.Text);  foreach (Socket item in socketList)  {  if (item.Connected)  {  item.Send(bytes, 0, bytes.Length, SocketFlags.None);  }  }  }  #endregion  #region 追加信息到文本框  public void AppendMsgToText(string txt)  {  if (txtMsg.InvokeRequired)  {  txtMsg.BeginInvoke(new Action<string>((s) =>  {  this.txtMsg.Text = string.Format("{0}\r\n{1}", s, txtMsg.Text);  }  ), txt);  }  else  {  this.txtMsg.Text = string.Format("{0}\r\n{1}", txt, txtMsg.Text);  }  }  #endregion  #region 关闭窗体时通知客户端退出。  private void Form1\_FormClosing(object sender, FormClosingEventArgs e)  {  foreach (var item in socketList)  {  StopConnection(item);  }  }  #endregion  }  } |

## 客户端

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Net;  using System.Net.Sockets;  using System.Text;  using System.Threading;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace \_03\_客户端完整版  {  public partial class 客户端 : Form  {  Socket clientSocket;  public 客户端()  {  InitializeComponent();  }  #region 连接服务器  private void btnConnect\_Click(object sender, EventArgs e)  {  //1.创建Socket对象  clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);  try  {  //2.连接服务器端  clientSocket.Connect(IPAddress.Parse(txtIP.Text), int.Parse(txtPort.Text));  }  catch  {  appendMsgToText("连接失败重新连接");  return;  }  appendMsgToText("已连接上服务器");  //3 发送消息，接收消息。  ThreadPool.QueueUserWorkItem(recieveMsgFromServer);  }  #endregion  #region 接受信息  private void recieveMsgFromServer(object state)  {  byte[] buffer = new byte[1024 \* 1024];  int receiveNums = 0; ;  while (true)  {  try  {  receiveNums = clientSocket.Receive(buffer, 0, buffer.Length, SocketFlags.None);  }  catch  {  appendMsgToText("服务器异常退出！");  StopConnection(clientSocket);  return;  }  if (receiveNums == 0)  {  appendMsgToText("服务器异常退出！");  StopConnection(clientSocket);  return;  }  string msg = Encoding.Default.GetString(buffer, 0, receiveNums);  appendMsgToText(string.Format("服务器发来消息：{0}", msg));  }  }  #endregion  #region 追加数据到文本框  private void appendMsgToText(string msg)  {  if (txtMsg.InvokeRequired)  {  txtMsg.Invoke(new Action<string>((s) =>  {  txtMsg.Text = string.Format("{0}\r\n{1}", s, txtMsg.Text);  }), msg);  }  else  {  txtMsg.Text = string.Format("{0}\r\n{1}", msg, txtMsg.Text);  }  }    #endregion  #region 发送信息  private void btnSend\_Click(object sender, EventArgs e)  {  byte[] buffer = Encoding.Default.GetBytes(txtSendMsg.Text);  clientSocket.Send(buffer, 0, buffer.Length, SocketFlags.None);  }  #endregion  #region 主窗口退出前关闭连接  private void 客户端\_FormClosing(object sender, FormClosingEventArgs e)  {  if (clientSocket.Connected)  {  StopConnection(clientSocket);  }  }  #endregion  #region 关闭连接  private void StopConnection(Socket clientSocket)  {  clientSocket.Shutdown(SocketShutdown.Both);  clientSocket.Close();  }  #endregion  }  } |

## 自定义协议

在第 4 章的案例中，我们可以发送文本消息。只需要把传过来的字节还原为字符串即可。但是如果我们想要发送一个命令（如关机）或者发送一个文件时，该如何处理呢？

这时就需要约定一个协议。在传输的字节前加上特殊的标识来区别传输的是文字、命令还是文件。

例如：

第一字节约定类型：1：表示文字、2：表示命令、3：表示文件。

第二个字节约定类型的格式：

10：表示传输的是无格式字符

21：表示传输的是一个闪屏命令

31：表示传输的是一个txt文件。

### 发送字符

#### 服务器端

|  |
| --- |
| private void btnSend\_Click(object sender, EventArgs e)  {  //1.原始字符转换成字节数组  byte[] bytes = Encoding.Default.GetBytes(txtSendMsg.Text);  //2.定义一个新的字节数组：比原始字节数组长度多2，用来存储协议  byte[] result = new byte[bytes.Length + 2];  //设置当前协议头部字节: 10代表字符串  result[0] = 1;  result[1] = 0;  Buffer.BlockCopy(bytes, 0, result, 2, bytes.Length);  foreach (Socket item in socketList)  {  if (item.Connected)  {  item.Send(result, 0, result.Length, SocketFlags.None);  }  }  } |

#### 客户端

接受时判断协议头部

|  |
| --- |
| private void recieveMsgFromServer(object state)  {  byte[] buffer = new byte[1024 \* 1024];  int receiveNums = 0; ;  while (true)  {  try  {  receiveNums = clientSocket.Receive(buffer, 0, buffer.Length, SocketFlags.None);  }  catch  {  appendMsgToText("服务器异常退出！");  StopConnection(clientSocket);  return;  }  if (receiveNums == 0)  {  appendMsgToText("正常退出！");  StopConnection(clientSocket);  return;  }  //遍历协议的第一个字节  switch (buffer[0])  {  //如果为1表示是文字  case 1:  receiveString(buffer);  break;  }  }  } |

如果传过来的字节数组第一个是1，则调用receiveString方法接受字符串信息

|  |
| --- |
| private void receiveString(byte[] buffer)  {  if (buffer[1] == 0)  {  //从偏移为2的字节开始转换为字符，转换大小为字节-2即正文的长度  string msg = Encoding.Default.GetString(buffer, 2, buffer.Length - 2);  appendMsgToText(string.Format("服务器发来消息：{0}", msg));  }  } |

### 发送命令

#### 服务器端

只需要发送协议即可

|  |
| --- |
| private void btnSendCommand\_Click(object sender, EventArgs e)  {  //只需要发送协议，20表示闪屏  byte[] bytes = new byte[2] { 2, 0 };  foreach (var item in socketList)  {  item.Send(bytes, 0, bytes.Length, SocketFlags.None);  }  } |

#### 客户端

|  |
| --- |
| //遍历协议的第一个字节  switch (buffer[0])  {  //如果为1表示是文字  case 1:  receiveString(buffer);  break;  //如果为2表示是命令  case 2:  receiveCommand(buffer);  break;  } |

调用receiveCommand方法

|  |
| --- |
| private void receiveCommand(byte[] buffer)  {  //如果为0表示是闪屏  if (buffer[1] == 0)  {  Point oldLocation = this.Location;  Random rand = new Random();  if (this.InvokeRequired)  {  //跨线程调用掉发  this.Invoke(new Action(() =>  {  for (int i = 0; i < 20; i++)  {  this.Location = new Point(rand.Next(oldLocation.X - 5, oldLocation.X + 5), rand.Next(oldLocation.Y - 5, oldLocation.Y + 5));  Thread.Sleep(30);  }  }));  }  }  } |

### 发送文件

#### 服务器端

|  |
| --- |
| private void btnSendFile\_Click(object sender, EventArgs e)  {  OpenFileDialog ofd = new OpenFileDialog();  ofd.DefaultExt = "txt";  ofd.Filter = "文本文件(\*.txt)|\*.txt|图片(\*.jpg)|\*.jpg|所有文件(\*.\*)|\*.\*";  if (ofd.ShowDialog(this) != DialogResult.OK)  {  return;  }  //读取文件  byte[] file = File.ReadAllBytes(ofd.FileName);  //获取后缀  string ext = Path.GetExtension(ofd.FileName);  //用来存储带协议的字节数组  byte[] result = new byte[file.Length + 2];  //设置协议  result[0] = 3;  switch (ext)  {  case ".txt":  result[1] = 1;  break;  case ".jpg":  result[1] = 2;  break;  default:  result[1] = 0;  break;  }  //原数组拷贝到带协议的数组中。  Buffer.BlockCopy(file, 0, result, 2,file.Length);  //广播  foreach (var item in socketList)  {  if (!item.Connected)  {  return;  }  item.Send(result, SocketFlags.None);  }  } |

#### 客户端

|  |
| --- |
| private void receiveFile(byte[] buffer, int receiveNums)  {  SaveFileDialog sfd = new SaveFileDialog();  //根据协议自动加后缀  switch (buffer[1])  {  case 1:  sfd.DefaultExt = "txt";  break;  case 2:  sfd.DefaultExt = "jpg";  break;  }  sfd.AddExtension = true;  //跨线程调用  if (this.InvokeRequired)  {  this.Invoke(new Action(() =>  {  if (sfd.ShowDialog(this) != DialogResult.OK)  {  return;  }  }  ));  }  //数组用来还原数据  byte[] fileData = new byte[receiveNums - 2];  //去掉协议  Buffer.BlockCopy(buffer, 2, fileData, 0, fileData.Length);  //写文件到硬盘。  File.WriteAllBytes(sfd.FileName, fileData);  } |

## 完整代码

### 服务器端

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.IO;  using System.Linq;  using System.Net;  using System.Net.Sockets;  using System.Text;  using System.Threading;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace \_02\_服务端完整版  {  public partial class Form1 : Form  {  List<Socket> socketList = new List<Socket>();  public Form1()  {  InitializeComponent();  }  #region 开启服务  private void btmStartServer\_Click(object sender, EventArgs e)  {  //1 创建Socket  Socket ListenSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);  //2 绑定端口ip  ListenSocket.Bind(new IPEndPoint(IPAddress.Parse(txtIP.Text), int.Parse(txtPort.Text)));  //3 开启侦听  ListenSocket.Listen(10);  //4 开辟线程，开始接受客户端的连接  ThreadPool.QueueUserWorkItem(new WaitCallback(CreateCommunicationSocket), ListenSocket);  }  #endregion  #region 接受信息  private void ReceiveMsgFromClient(object state)  {  Socket commnunicationSocekt = state as Socket;  if (commnunicationSocekt == null)  {  return;  }  byte[] bytes = new byte[1024 \* 1024];  while (true)  {  int receiveNums = 0;  try  {  receiveNums = commnunicationSocekt.Receive(bytes, 0, bytes.Length, SocketFlags.None);  }  catch  {  //异常退出  AppendMsgToText(string.Format("客户端：{0}异常退出", commnunicationSocekt.RemoteEndPoint.ToString()));  socketList.Remove(commnunicationSocekt);  StopConnection(commnunicationSocekt);  return;  }  if (receiveNums == 0)//正常退出  {  AppendMsgToText(string.Format("客户端：{0}已正常关闭。", commnunicationSocekt.RemoteEndPoint.ToString()));  socketList.Remove(commnunicationSocekt);  StopConnection(commnunicationSocekt);  return;  }  //接受的信息  string receiveMsg = Encoding.Default.GetString(bytes, 0, receiveNums);  AppendMsgToText(string.Format("客户端：{0} 发来消息：{1}", commnunicationSocekt.RemoteEndPoint.ToString(), receiveMsg));  }  }  #endregion  #region 关闭连接释放资源  private void StopConnection(Socket socket)  {  if (socket.Connected)  {  socket.Shutdown(SocketShutdown.Both);  socket.Close();  }  }  #endregion  #region 创建通信socket，并接受信息  private void CreateCommunicationSocket(object state)  {  Socket ListenSocket = state as Socket;  while (true)  {  Socket commnunicationSocekt = ListenSocket.Accept();  socketList.Add(commnunicationSocekt);  AppendMsgToText(string.Format("客户端：{0}已连接上", commnunicationSocekt.RemoteEndPoint.ToString()));  //开辟线程，接受信息。  ThreadPool.QueueUserWorkItem(ReceiveMsgFromClient, commnunicationSocekt);  }  }  #endregion  #region 发送消息  private void btnSend\_Click(object sender, EventArgs e)  {  //1.原始字符转换成字节数组  byte[] bytes = Encoding.Default.GetBytes(txtSendMsg.Text);  //2.定义一个新的字节数组：比原始字节数组长度多2，用来存储协议  byte[] result = new byte[bytes.Length + 2];  //设置当前协议头部字节: 10代表字符串  result[0] = 1;  result[1] = 0;  Buffer.BlockCopy(bytes, 0, result, 2, bytes.Length);  foreach (Socket item in socketList)  {  if (item.Connected)  {  item.Send(result, 0, result.Length, SocketFlags.None);  }  }  }  #endregion  #region 追加信息到文本框  public void AppendMsgToText(string txt)  {  if (txtMsg.InvokeRequired)  {  txtMsg.BeginInvoke(new Action<string>((s) =>  {  this.txtMsg.Text = string.Format("{0}\r\n{1}", s, txtMsg.Text);  }  ), txt);  }  else  {  this.txtMsg.Text = string.Format("{0}\r\n{1}", txt, txtMsg.Text);  }  }  #endregion  #region 关闭窗体时通知客户端退出。  private void Form1\_FormClosing(object sender, FormClosingEventArgs e)  {  foreach (var item in socketList)  {  StopConnection(item);  }  }  #endregion  private void btnSendCommand\_Click(object sender, EventArgs e)  {  //只需要发送协议，20表示闪屏  byte[] bytes = new byte[2] { 2, 0 };  foreach (var item in socketList)  {  item.Send(bytes, 0, bytes.Length, SocketFlags.None);  }  }  private void btnSendFile\_Click(object sender, EventArgs e)  {  OpenFileDialog ofd = new OpenFileDialog();  ofd.DefaultExt = "txt";  ofd.Filter = "文本文件(\*.txt)|\*.txt|图片(\*.jpg)|\*.jpg|所有文件(\*.\*)|\*.\*";  if (ofd.ShowDialog(this) != DialogResult.OK)  {  return;  }  //读取文件  byte[] file = File.ReadAllBytes(ofd.FileName);  //获取后缀  string ext = Path.GetExtension(ofd.FileName);  //用来存储带协议的字节数组  byte[] result = new byte[file.Length + 2];  //设置协议  result[0] = 3;  switch (ext)  {  case ".txt":  result[1] = 1;  break;  case ".jpg":  result[1] = 2;  break;  default:  result[1] = 0;  break;  }  //原数组拷贝到带协议的数组中。  Buffer.BlockCopy(file, 0, result, 2,file.Length);  //广播  foreach (var item in socketList)  {  if (!item.Connected)  {  return;  }  item.Send(result, SocketFlags.None);  }  }  }  } |

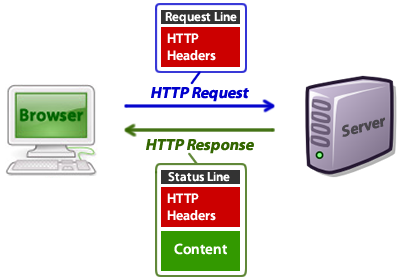
### 客户端

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.IO;  using System.Linq;  using System.Net;  using System.Net.Sockets;  using System.Text;  using System.Threading;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace \_03\_客户端完整版  {  public partial class 客户端 : Form  {  Socket clientSocket;  public 客户端()  {  InitializeComponent();  }  #region 连接服务器  private void btnConnect\_Click(object sender, EventArgs e)  {  //1.创建Socket对象  clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);  try  {  //2.连接服务器端  clientSocket.Connect(IPAddress.Parse(txtIP.Text), int.Parse(txtPort.Text));  }  catch  {  appendMsgToText("连接失败重新连接");  return;  }  appendMsgToText("已连接上服务器");  //3 发送消息，接收消息。  ThreadPool.QueueUserWorkItem(recieveMsgFromServer);  }  #endregion  #region 接受信息  private void recieveMsgFromServer(object state)  {  byte[] buffer = new byte[1024 \* 1024];  int receiveNums = 0; ;  while (true)  {  try  {  receiveNums = clientSocket.Receive(buffer, 0, buffer.Length, SocketFlags.None);  }  catch  {  appendMsgToText("服务器异常退出！");  StopConnection(clientSocket);  return;  }  if (receiveNums == 0)  {  appendMsgToText("正常退出！");  StopConnection(clientSocket);  return;  }  //遍历协议的第一个字节  switch (buffer[0])  {  //如果为1表示是文字  case 1:  receiveString(buffer);  break;  //如果为2表示是命令  case 2:  receiveCommand(buffer);  break;  case 3:  receiveFile(buffer, receiveNums);  break;  }  }  }  private void receiveFile(byte[] buffer, int receiveNums)  {  SaveFileDialog sfd = new SaveFileDialog();  //根据协议自动加后缀  switch (buffer[1])  {  case 1:  sfd.DefaultExt = "txt";  break;  case 2:  sfd.DefaultExt = "jpg";  break;  }  sfd.AddExtension = true;  //跨线程调用  if (this.InvokeRequired)  {  this.Invoke(new Action(() =>  {  if (sfd.ShowDialog(this) != DialogResult.OK)  {  return;  }  }  ));  }  //数组用来还原数据  byte[] fileData = new byte[receiveNums - 2];  //去掉协议  Buffer.BlockCopy(buffer, 2, fileData, 0, fileData.Length);  //写文件到硬盘。  File.WriteAllBytes(sfd.FileName, fileData);  }  private void receiveCommand(byte[] buffer)  {  if (buffer[1] == 0)  {  Point oldLocation = this.Location;  Random rand = new Random();  if (this.InvokeRequired)  {  this.Invoke(new Action(() =>  {  for (int i = 0; i < 20; i++)  {  this.Location = new Point(rand.Next(oldLocation.X - 5, oldLocation.X + 5), rand.Next(oldLocation.Y - 5, oldLocation.Y + 5));  Thread.Sleep(30);  }  }));  }  }  }  private void receiveString(byte[] buffer)  {  if (buffer[1] == 0)  {  //从偏移为2的字节开始转换为字符，转换大小为字节-2即正文的长度  string msg = Encoding.Default.GetString(buffer, 2, buffer.Length - 2);  appendMsgToText(string.Format("服务器发来消息：{0}", msg));  }  }  #endregion  #region 追加数据到文本框  private void appendMsgToText(string msg)  {  if (txtMsg.InvokeRequired)  {  txtMsg.Invoke(new Action<string>((s) =>  {  txtMsg.Text = string.Format("{0}\r\n{1}", s, txtMsg.Text);  }), msg);  }  else  {  txtMsg.Text = string.Format("{0}\r\n{1}", msg, txtMsg.Text);  }  }  #endregion  #region 发送信息  private void btnSend\_Click(object sender, EventArgs e)  {  byte[] buffer = Encoding.Default.GetBytes(txtSendMsg.Text);  clientSocket.Send(buffer, 0, buffer.Length, SocketFlags.None);  }  #endregion  #region 主窗口退出前关闭连接  private void 客户端\_FormClosing(object sender, FormClosingEventArgs e)  {  if (clientSocket.Connected)  {  StopConnection(clientSocket);  }  }  #endregion  #region 关闭连接  private void StopConnection(Socket clientSocket)  {  clientSocket.Shutdown(SocketShutdown.Both);  clientSocket.Close();  }  #endregion  }  } |

# HTTP

## 什么是HTTP协议

一个基于应用层的通信规范



## 简介

Web开发是和Http协议打交道的，必须了解Http协议。Http协议版本：Http/0.9、Http/1.0、Http/1.1，现在主流的是Http/1.1版本

Http协议分析工具：

1、DebugBar，Http(S)标签的内容。免费的。只能分析当前浏览器中的内容。

2、httpwatch，收费的，只能分析当前浏览器中的内容。推荐使用

3、HttpAnalyzer，收费的，能分析计算机上所有的Http请求数据。

4、WFetch\_1.4，免费，小型分析工具

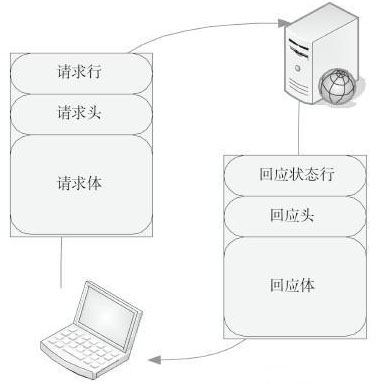
Http协议的几个概念：

1.连接(Connection)：浏览器和服务器之间传输数据的通道。 一般请求完毕就关闭，http不保持连接。不保持连接会降低处理速度（因为建立连接速度很慢），保持连接的话就会降低服务器的处理的客户端请求数，而不保持连接服务器可以处理更多的请求。

2.请求(Request)：浏览器向服务器发送的“我要\*\*\*”的消息，包含请求的类型、请求的数据、浏览器的信息（语言、浏览器版本等）。

3.响应(Response)：服务器对浏览器请求的返回的数据，包含是否成功、状态码等。

## HTTP协议消息结构图



## HTTP协议-请求报文格式



### 请求报文示例



## 响应码

浏览器向服务器发出请求，服务器处理可能是成功、可能是失败、可能没有权限访问等原因，服务器会通过响应码来告诉浏览器处理结果。

"200" : OK

"302" : Found 重定向.

"400" : Bad Request 错误请求，发出错误的不符合Http协议的请求

"403" : Forbidden 禁止

"404" : Not Found 未找到。演示访问一个不存在的页面看报文

"500" : Internal Server Error 服务器内部错误。演示页面抛出异常。

"503" : Service Unavailable。一般是访问人数过多。

200段是成功；300段需要对请求做进一步的处理；400段表示客户端请求错误；500段是服务器的错误。

## Http协议特点

http是无状态的，不会记得“上个请求”，所以哪怕是同一个页面中的js、css、jpg也都要重复的提交Accept-Language、Accept-Encoding、Cookie等。

网页中如果有图片、css、js等外部文件的话图片、css、js都在单独的请求中，也就是并不是页面的所有内容都在一个请求中完成，而是每个资源一个请求。

一般情况下，只有浏览器请求服务器端，服务器端才有给浏览器响应数据，服务器不会主动向浏览器推送数据，这样是安全考虑，也是提高服务器的性能考虑。如果要服务器向浏览器推送数据，则需要使用ServerPush(ajax隔一段时间到服务器请求最新的数据)等额外的技术。

Http是“请求—响应”的工作方式。

## HTTP请求详解图

