

# C#网络编程系列文章(三)之TcpListener实现异步TCP服务器

原创 2015年03月14日 18:02:51

8829

## 原创性声明

本文作者：小竹zz 本文地址http://blog.csdn.net/zhujunxxxxx/article/details/44258719 转载请注明出处

## 文章系列目录

- C#网络编程系列文章(一)之Socket实现异步TCP服务器 (http://blog.csdn.net/zhujunxxxxx/article/details/44258719)
- C#网络编程系列文章(二)之Socket实现同步TCP服务器 (http://blog.csdn.net/zhujunxxxxx/article/details/44261375)
- C#网络编程系列文章(三)之TcpListener实现异步TCP服务器 (http://blog.csdn.net/zhujunxxxxx/article/details/44261497)
- C#网络编程系列文章(四)之TcpListener实现同步TCP服务器 (http://blog.csdn.net/zhujunxxxxx/article/details/44278861)
- C#网络编程系列文章(五)之Socket实现异步UDP服务器 (http://blog.csdn.net/zhujunxxxxx/article/details/44280959)
- C#网络编程系列文章(六)之Socket实现同步UDP服务器 (http://blog.csdn.net/zhujunxxxxx/article/details/44303697)
- C#网络编程系列文章(七)之UdpClient实现异步UDP服务器 (http://blog.csdn.net/zhujunxxxxx/article/details/44303743)
- C#网络编程系列文章(八)之UdpClient实现同步UDP服务器 (http://blog.csdn.net/zhujunxxxxx/article/details/44303787)

## 本文介绍

TcpListener 类提供一些简单方法，用于在阻止同步模式下侦听和接受传入连接请求。可使用 TcpClient 或 Socket 来连接 TcpListener。可使用 IPEndPoint、本地 IP 地址及端口号或者仅使用端口号，来创建 TcpListener。可以将本地 IP 地址指定为 Any，将本地端口号指定为 0（如果希望基础服务提供程序为您分配这些值）。如果您选择这样做，可在连接套接字后使用 LocalEndpoint 属性来标识已指定的信息。使用 Start 方法，可开始侦听传入的连接请求。Start 将对传入连接进行排队，直至您调用 Stop 方法或它已经完成 MaxConnections 排队为止。可使用 AcceptSocket 或 AcceptTcpClient 从传入连接请求队列提取连接。这两种方法将阻止。如果要避免阻止，可首先使用 Pending 方法来确定队列中是否有可用的连接请求。

虽然TcpListener已经封装的比较不错了，我们于是就使用它在构造一个比较不错的异步TCP服务器，这里依然和前两章一样，给出服务器中的代码，代码中注释很详细，我也会给出相关的封装类。

## TcpListener异步TCP服务器



小竹zz (http://blog.csdn.net/zhujunxxxxx)

+ 关注

(http://blog.csdn.net/zhujunxxxxx)

码云

未开通

(https://github.com/zhujunxxxxx)

原创 129

粉丝 155

喜欢 1

utm\_source=

### 他的最新文章

更多文章 (http://blog.csdn.net/zhujunxxxxx)

记一次完整手机APP项目的开发 (http://blog.csdn.net/zhujunxxxxx/article/details/77824049)

树莓派+串口墨水电子屏幕+温度湿度传感器打造专属时钟 (http://blog.csdn.net/zhujunxxxxx/article/details/51944584)

MysqlProtocolAnalyzer一个Java实现的MySQL协议解析库 (http://blog.csdn.net/zhujunxxxxx/article/details/49837335)



### 博主专栏



数据结构的研究

(http://blog.csdn.net/column/details/18253)

18253



Web请求模拟

(http://blog.csdn.net/column/details/74853)

74853



c#网络编程

(http://blog.csdn.net/column/details/44258719)

44258719



c#网络编程

(http://blog.csdn.net/column/details/44258719)

44258719

```
[csharp]
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Net.Sockets;
6. using System.Net;
7. namespace NetFrame.Net.TCP.Listener.Asynchronous
8. {
9.     /// <summary>
10.    /// TcpListener实现异步TCP服务器
11.    /// </summary>
12.    public class AsyncTCPServer : IDisposableable
13.    {
14.        #region Fields
15.        /// <summary>
16.        /// 服务器程序允许的最大客户端连接数
17.        /// </summary>
18.        private int _maxClient;
19.
20.        /// <summary>
21.        /// 当前的连接的客户端数
22.        /// </summary>
23.        private int _clientCount;
24.
25.        /// <summary>
26.        /// 服务器使用的异步TcpListener
27.        /// </summary>
28.        private TcpListener _listener;
29.
30.        /// <summary>
31.        /// 客户端会话列表
32.        /// </summary>
33.        private List<Object> _clients;
34.
35.        private bool disposed = false;
36.
37.        #endregion
38.
39.        #region Properties
40.
41.        /// <summary>
42.        /// 服务器是否正在运行
43.        /// </summary>
44.        public bool IsRunning { get; private set; }
45.        /// <summary>
46.        /// 监听的IP地址
47.        /// </summary>
48.        public IPAddress Address { get; private set; }
49.        /// <summary>
50.        /// 监听的端口
51.        /// </summary>
52.        public int Port { get; private set; }
53.        /// <summary>
54.        /// 通信使用的编码
55.        /// </summary>
56.        public Encoding Encoding { get; set; }
57.
58.
59.        #endregion
60.
61.        #region 构造函数
62.
63.        /// <summary>
64.        /// 异步TCP服务器
65.        /// </summary>
66.        /// <param name="listenPort">监听的端口</param>
67.        public AsyncTCPServer(int listenPort)
68.        : this(IPAddress.Any, listenPort)
69.        {
70.
71.
72.        }
73.
74.        /// <summary>
75.        /// 异步TCP服务器
76.        /// </summary>
77.        /// <param name="localEP">监听的终结点</param>
78.        public AsyncTCPServer(IPEndPoint localEP)
```



白盒引擎在WebAssembly中的实践  
([http://edu.csdn.net/huiyiCourse/series\\_detail/?utm\\_source=blog9](http://edu.csdn.net/huiyiCourse/series_detail/?utm_source=blog9))  
[http://edu.csdn.net/huiyiCourse/series\\_detail/?utm\\_source=blog9](http://edu.csdn.net/huiyiCourse/series_detail/?utm_source=blog9)



Apache Weex：移动研发的进阶之路  
([http://edu.csdn.net/huiyiCourse/detail/602?utm\\_source=blog9](http://edu.csdn.net/huiyiCourse/detail/602?utm_source=blog9))  
[http://edu.csdn.net/huiyiCourse/detail/602?utm\\_source=blog9](http://edu.csdn.net/huiyiCourse/detail/602?utm_source=blog9)

热门文章

python实现QQ机器人(自动登录，获取群消息，发送群消息) (<http://blog.csdn.net/zhujunxxxxx/article/details/29412297>)  
🔒 23304

[论文]内存数据库中的索引技术 (<http://blog.csdn.net/zhujunxxxxx/article/details/42490335>)  
🔒 19993

c#基于udp实现的p2p语音聊天工具 (<http://blog.csdn.net/zhujunxxxxx/article/details/40124773>)  
🔒 19120

C#高性能Socket服务器SocketAsyncEventArgs的实现(IOPC) (<http://blog.csdn.net/zhujunxxxxx/article/details/43573879>)  
🔒 18865

wifidog用php实现验证流程 (<http://blog.csdn.net/zhujunxxxxx/article/details/25384909>)  
🔒 16061

```

77.         : this(localEP.Address, localEP.Port)
78.     {
79.     }
80.
81.     /// <summary>
82.     /// 异步TCP服务器
83.     /// </summary>
84.     /// <param name="localIPAddress">监听的IP地址</param>
85.     /// <param name="listenPort">监听的端口</param>
86.     public AsyncTCPServer(IPAddress localIPAddress, int listenPort)
87.     {
88.         Address = localIPAddress;
89.         Port = listenPort;
90.         this.Encoding = Encoding.Default;
91.
92.         _clients = new List<Object>();
93.
94.         _listener = new TcpListener(Address, Port);
95.         _listener.AllowNatTraversal(true);
96.     }
97.
98.     #endregion
99.
100.    #region Method
101.
102.    /// <summary>
103.    /// 启动服务器
104.    /// </summary>
105.    public void Start()
106.    {
107.        if (!IsRunning)
108.        {
109.            IsRunning = true;
110.            _listener.Start();
111.            _listener.BeginAcceptTcpClient(
112.                new AsyncCallback(HandleTcpClientAccepted), _listener);
113.        }
114.    }
115.
116.
117.    /// <summary>
118.    /// 启动服务器
119.    /// </summary>
120.    /// <param name="backlog">
121.    /// 服务器所允许的挂起连接序列的最大长度
122.    /// </param>
123.    public void Start(int backlog)
124.    {
125.        if (!IsRunning)
126.        {
127.            IsRunning = true;
128.            _listener.Start(backlog);
129.            _listener.BeginAcceptTcpClient(
130.                new AsyncCallback(HandleTcpClientAccepted), _listener);
131.        }
132.    }
133.
134.    /// <summary>
135.    /// 停止服务器
136.    /// </summary>
137.    public void Stop()
138.    {
139.        if (IsRunning)
140.        {
141.            IsRunning = false;
142.            _listener.Stop();
143.            lock (_clients)
144.            {
145.                //关闭所有客户端连接
146.                CloseAllClient();
147.            }
148.        }
149.    }
150.
151.    /// <summary>
152.    /// 处理客户端连接的函数
153.    /// </summary>
154.    /// <param name="ar"></param>
155.    private void HandleTcpClientAccepted(IAsyncResult ar)
156.    {

```



内容举报



返回顶部

```

157.         if (IsRunning)
158.         {
159.             //TcpListener tcpListener = (TcpListener)ar.AsyncState;
160.
161.             TcpClient client = _listener.EndAcceptTcpClient(ar);
162.             byte[] buffer = new byte[client.ReceiveBufferSize];
163.
164.             TCPClientState state
165.                 = new TCPClientState(client, buffer);
166.             lock (_clients)
167.             {
168.                 _clients.Add(state);
169.                 RaiseClientConnected(state);
170.             }
171.
172.             NetworkStream stream = state.NetworkStream;
173.             //开始异步读取数据
174.             stream.BeginRead(state.Buffer, 0, state.Buffer.Length, HandleDataReceived, state);
175.
176.             _listener.BeginAcceptTcpClient(
177.                 new AsyncCallback(HandleTcpClientAccepted), ar.AsyncState);
178.         }
179.     }
180.     /// <summary>
181.     /// 数据接受回调函数
182.     /// </summary>
183.     /// <param name="ar"></param>
184.     private void HandleDataReceived(IAsyncResult ar)
185.     {
186.         if (IsRunning)
187.         {
188.             TCPClientState state = (TCPClientState)ar.AsyncState;
189.             NetworkStream stream = state.NetworkStream;
190.
191.             int recv = 0;
192.             try
193.             {
194.                 recv = stream.EndRead(ar);
195.             }
196.             catch
197.             {
198.                 recv = 0;
199.             }
200.
201.             if (recv == 0)
202.             {
203.                 // connection has been closed
204.                 lock (_clients)
205.                 {
206.                     _clients.Remove(state);
207.                     //触发客户端连接断开事件
208.                     RaiseClientDisconnected(state);
209.                     return;
210.                 }
211.             }
212.
213.             // received byte and trigger event notification
214.             byte[] buff = new byte[recv];
215.             Buffer.BlockCopy(state.Buffer, 0, buff, 0, recv);
216.             //触发数据收到事件
217.             RaiseDataReceived(state);
218.
219.             // continue listening for tcp datagram packets
220.             stream.BeginRead(state.Buffer, 0, state.Buffer.Length, HandleDataReceived, state);
221.         }
222.     }
223.
224.     /// <summary>
225.     /// 发送数据
226.     /// </summary>
227.     /// <param name="state">接收数据的客户端会话</param>
228.     /// <param name="data">数据报文</param>
229.     public void Send(TCPClientState state, byte[] data)
230.     {
231.         RaisePrepareSend(state);
232.         Send(state.TcpClient, data);
233.     }
234.
235.     /// <summary>
236.     /// 异步发送数据至指定的客户端

```

广告



内容举报



返回顶部

```

237.     /// </summary>
238.     /// <param name="client">客户端</param>
239.     /// <param name="data">报文</param>
240.     public void Send(TcpClient client, byte[] data)
241.     {
242.         if (!IsRunning)
243.             throw new InvalidProgramException("This TCP Socket server has not been started.");
244.
245.         if (client == null)
246.             throw new ArgumentNullException("client");
247.
248.         if (data == null)
249.             throw new ArgumentNullException("data");
250.         client.GetStream().BeginWrite(data, 0, data.Length, SendDataEnd, client);
251.     }
252.
253.     /// <summary>
254.     /// 发送数据完成处理函数
255.     /// </summary>
256.     /// <param name="ar">目标客户端Socket</param>
257.     private void SendDataEnd(IAsyncResult ar)
258.     {
259.         ((TcpClient)ar.AsyncState).GetStream().EndWrite(ar);
260.         RaiseCompletedSend(null);
261.     }
262.     #endregion
263.
264.     #region 事件
265.
266.     /// <summary>
267.     /// 与客户端的连接已建立事件
268.     /// </summary>
269.     public event EventHandler<AsyncEventArgs> ClientConnected;
270.     /// <summary>
271.     /// 与客户端的连接已断开事件
272.     /// </summary>
273.     public event EventHandler<AsyncEventArgs> ClientDisconnected;
274.
275.
276.     /// <summary>
277.     /// 触发客户端连接事件
278.     /// </summary>
279.     /// <param name="state"></param>
280.     private void RaiseClientConnected(TCPClientState state)
281.     {
282.         if (ClientConnected != null)
283.         {
284.             ClientConnected(this, new AsyncEventArgs(state));
285.         }
286.     }
287.     /// <summary>
288.     /// 触发客户端连接断开事件
289.     /// </summary>
290.     /// <param name="client"></param>
291.     private void RaiseClientDisconnected(TCPClientState state)
292.     {
293.         if (ClientDisconnected != null)
294.         {
295.             ClientDisconnected(this, new AsyncEventArgs("连接断开"));
296.         }
297.     }
298.
299.     /// <summary>
300.     /// 接收到数据事件
301.     /// </summary>
302.     public event EventHandler<AsyncEventArgs> DataReceived;
303.
304.     private void RaiseDataReceived(TCPClientState state)
305.     {
306.         if (DataReceived != null)
307.         {
308.             DataReceived(this, new AsyncEventArgs(state));
309.         }
310.     }
311.
312.     /// <summary>
313.     /// 发送数据前的事件
314.     /// </summary>
315.     public event EventHandler<AsyncEventArgs> PrepareSend;
316.

```



内容举报



返回顶部

```

317.     /// <summary>
318.     /// 触发发送数据前的事件
319.     /// </summary>
320.     /// <param name="state"></param>
321.     private void RaisePrepareSend(TCPClientState state)
322.     {
323.         if (PrepareSend != null)
324.         {
325.             PrepareSend(this, new AsyncEventArgs(state));
326.         }
327.     }
328.
329.     /// <summary>
330.     /// 数据发送完毕事件
331.     /// </summary>
332.     public event EventHandler<AsyncEventArgs> CompletedSend;
333.
334.     /// <summary>
335.     /// 触发数据发送完毕的事件
336.     /// </summary>
337.     /// <param name="state"></param>
338.     private void RaiseCompletedSend(TCPClientState state)
339.     {
340.         if (CompletedSend != null)
341.         {
342.             CompletedSend(this, new AsyncEventArgs(state));
343.         }
344.     }
345.
346.     /// <summary>
347.     /// 网络错误事件
348.     /// </summary>
349.     public event EventHandler<AsyncEventArgs> NetError;
350.     /// <summary>
351.     /// 触发网络错误事件
352.     /// </summary>
353.     /// <param name="state"></param>
354.     private void RaiseNetError(TCPClientState state)
355.     {
356.         if (NetError != null)
357.         {
358.             NetError(this, new AsyncEventArgs(state));
359.         }
360.     }
361.
362.     /// <summary>
363.     /// 异常事件
364.     /// </summary>
365.     public event EventHandler<AsyncEventArgs> OtherException;
366.     /// <summary>
367.     /// 触发异常事件
368.     /// </summary>
369.     /// <param name="state"></param>
370.     private void RaiseOtherException(TCPClientState state, string descrip)
371.     {
372.         if (OtherException != null)
373.         {
374.             OtherException(this, new AsyncEventArgs(descrip, state));
375.         }
376.     }
377.     private void RaiseOtherException(TCPClientState state)
378.     {
379.         RaiseOtherException(state, "");
380.     }
381.
382. #endregion
383.
384. #region Close
385.     /// <summary>
386.     /// 关闭一个与客户端之间的会话
387.     /// </summary>
388.     /// <param name="state">需要关闭的客户端会话对象</param>
389.     public void Close(TCPClientState state)
390.     {
391.         if (state != null)
392.         {
393.             state.Close();
394.             _clients.Remove(state);
395.             _clientCount--;
396.             //TODO 触发关闭事件

```



内容举报



返回顶部

```

397.     }
398. }
399. /// <summary>
400. /// 关闭所有的客户端会话,与所有的客户端连接会断开
401. /// </summary>
402. public void CloseAllClient()
403. {
404.     foreach (TCPClientState client in _clients)
405.     {
406.         Close(client);
407.     }
408.     _clientCount = 0;
409.     _clients.Clear();
410. }
411. #endregion
412.
413. #region 释放
414. /// <summary>
415. /// Performs application-defined tasks associated with freeing,
416. /// releasing, or resetting unmanaged resources.
417. /// </summary>
418. public void Dispose()
419. {
420.     Dispose(true);
421.     GC.SuppressFinalize(this);
422. }
423.
424. /// <summary>
425. /// Releases unmanaged and - optionally - managed resources
426. /// </summary>
427. /// <param name="disposing"><c>true</c> to release
428. /// both managed and unmanaged resources; <c>false</c>
429. /// to release only unmanaged resources.</param>
430. protected virtual void Dispose(bool disposing)
431. {
432.     if (!this.disposed)
433.     {
434.         if (disposing)
435.         {
436.             try
437.             {
438.                 Stop();
439.                 if (_listener != null)
440.                 {
441.                     _listener = null;
442.                 }
443.             }
444.             catch (SocketException)
445.             {
446.                 //TODO
447.                 RaiseOtherException(null);
448.             }
449.         }
450.         disposed = true;
451.     }
452. }
453. #endregion
454. }
455. }

```

## 客户端处理封装类

```

[csharp]
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Net.Sockets;
6.
7. namespace NetFrame.Net.TCP.Listener.Asynchronous
8. {
9.     public class TCPClientState
10.    {
11.        /// <summary>
12.        /// 与客户端相关的TcpClient
13.        /// </summary>
14.        public TcpClient TcpClient { get; private set; }
15.
16.        /// <summary>

```



内容举报



返回顶部

```

17.    /// 获取缓冲区
18.    /// </summary>
19.    public byte[] Buffer { get; private set; }
20.
21.    /// <summary>
22.    /// 获取网络流
23.    /// </summary>
24.    public NetworkStream NetworkStream
25.    {
26.        get { return TcpClient.GetStream(); }
27.    }
28.
29.    public TCPClientState(TcpClient tcpClient, byte[] buffer)
30.    {
31.        if (tcpClient == null)
32.            throw new ArgumentNullException("tcpClient");
33.        if (buffer == null)
34.            throw new ArgumentNullException("buffer");
35.
36.        this.TcpClient = tcpClient;
37.        this.Buffer = buffer;
38.    }
39.    /// <summary>
40.    /// 关闭
41.    /// </summary>
42.    public void Close()
43.    {
44.        //关闭数据的接受和发送
45.        TcpClient.Close();
46.        Buffer = null;
47.    }
48.    }
49. }

```

## 服务器事件参数类

```

[csharp]
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5.
6. namespace NetFrame.Net.TCP.Listener.Asynchronous
7. {
8.     /// <summary>
9.     /// 异步TcpListener TCP服务器事件参数类
10.    /// </summary>
11.    public class AsyncEventArgs:EventArgs
12.    {
13.        /// <summary>
14.        /// 提示信息
15.        /// </summary>
16.        public string _msg;
17.
18.        /// <summary>
19.        /// 客户端状态封装类
20.        /// </summary>
21.        public TCPClientState _state;
22.
23.        /// <summary>
24.        /// 是否已经处理过了
25.        /// </summary>
26.        public bool IsHandled { get; set; }
27.
28.        public AsyncEventArgs(string msg)
29.        {
30.            this._msg = msg;
31.            IsHandled = false;
32.        }
33.        public AsyncEventArgs(TCPClientState state)
34.        {
35.            this._state = state;
36.            IsHandled = false;
37.        }
38.        public AsyncEventArgs(string msg, TCPClientState state)
39.        {
40.            this._msg = msg;
41.            this._state = state;
42.            IsHandled = false;

```



内容举报



返回顶部



```
43.     }  
44.     }  
45. }
```

本文作者：小竹zz 本文地址<http://blog.csdn.net/zhujunxxxxx/article/details/44258719> 转载请注明出处

本文已收录于以下专栏：c#网络编程 (<http://blog.csdn.net/column/details/zhujun-csharp-net.html>)



suchangya (/suchangya) 2017-10-12 21:19

2楼

(/suchangya)  
class AsyncTCPServer 在220行  
// continue listening for tcp datagram packets  
stream.BeginRead(state.Buffer, 0, state.Buffer.Length, HandleDataReceived, state);  
是否应该加上while (ar.IsCompleted == false)包围

我用该段代码会报System.ObjectDisposedException: 无法访问已释放的对象，加上while循环后，程序就不报错。

ps: 我的客户端数量是200

回复



jacksu510 (/jacksu510) 2017-08-24 22:04

1楼

(/jacksu510)写的很详细，有启发！

回复

## 相关文章推荐

### c# socket 、TCPClient、 TCPListener 用法详解 (<http://blog.csdn.net/zgl159040290/a...>)

原文地址： [http://blog.sina.com.cn/s/blog\\_40d47c89010101vj.html](http://blog.sina.com.cn/s/blog_40d47c89010101vj.html) Visual C#.Net网络程序开发-Socket篇 Micros...

zgl159040290 (<http://blog.csdn.net/zgl159040290>) 2016年11月14日 11:05 2247

### C# 的TCPClient异步连接与异步读数据 (<http://blog.csdn.net/u011555996/article/detai...>)

Socket的TCP通讯 一、 socket的通讯原理 服务器端的步骤如下。（1）建立服务器端的Socket，开始侦听整个网络中的连接请求。（2）当检测到来自客户端的连接请...

u011555996 (<http://blog.csdn.net/u011555996>) 2016年03月10日 14:08 8489

### 异步调用后如何停止tcplistener ([http://blog.csdn.net/ropin\\_os/article/details/6736471](http://blog.csdn.net/ropin_os/article/details/6736471))

我有这个代码... internal static void Start() { TcpListener listenerSocket = new TcpListener(IPAddre...

ropin\_os ([http://blog.csdn.net/ropin\\_os](http://blog.csdn.net/ropin_os)) 2011年08月31日 16:49 3305

### [C#基础]网络编程(二):TcpListener & TcpClient (<http://blog.csdn.net/lyh916/article/d...>)



TcpListener & TcpClient，可以看作是对socket的进一步封装(基于tcp协议)，TcpListener为服务器端，TcpClient为客户端。 TcpListen...



内容举报





返回顶部

 lyh916 (<http://blog.csdn.net/lyh916>) 2015年11月15日 16:40  1617

## C#:TcpClient(客户端) and TcpServer(服务端) (<http://blog.csdn.net/xiaofengsheng/ar...>)

服务端:using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.W...

 xiaofengsheng (<http://blog.csdn.net/xiaofengsheng>) 2009年10月08日 23:02  16571





## Delphi7高级应用开发随书源码 (<http://download.csdn.net/detail/chen...>)

<http://download.csdn.net/detail/chen...> 2003年04月30日 00:00 676KB [下载](#)



## 高性能TCPServer (<http://blog.csdn.net/jjp837661103/article/details/24959227>)

最近两天正在学习TCPServer，在网上看到比较好的两篇文章，z

 jjp837661103 (<http://blog.csdn.net/jjp837661103>) 2014年05月04日 09:24  6776

## C#网络编程系列文章(一)之Socket实现异步TCP服务器 (<http://blog.csdn.net/zhujunxxxx...>)

原创性声明 开篇 本人因为对于网络编程的喜爱，经常性的使用c#编写各类服务器(e.g TCP服务器，UDP服务器)，但是基本上都是搞着玩，网上也有很多讲c#网络编程的文章，当然我也参考了很多作者写...

 zhujunxxxx (<http://blog.csdn.net/zhujunxxxx>) 2015年03月14日 13:43  14654



## Delphi7高级应用开发随书源码 (<http://download.csdn.net/detail/chen...>)

<http://download.csdn.net/detail/chen...> 2003年04月30日 00:00 676KB [下载](#)





## Delphi7高级应用开发随书源码 (<http://download.csdn.net/detail/chen...>)

<http://download.csdn.net/detail/chen...> 2003年04月30日 00:00 676KB [下载](#)



## C#异步TCP服务器完整实现 (/zhujunxxxxx/article/details/40621295)

TCP异步Socket模型 C#的TCP异步Socket模型是通过Begin-End模式实现的。例如提供 BeginConnect、BeginAccept、BeginSend 和 BeginRe...

 zhujunxxxx (<http://blog.csdn.net/zhujunxxxx>) 2014-10-30 16:02  9829



## C#网络编程系列文章(二)之Socket实现同步TCP服务器 (/zhujunxxxxx/article/details/442...

原创性声明 本文作者:小竹zz 本文地址<http://blog.csdn.net/zhujunxxxxx/article/details/44258719> 转载请注明出处 本文介绍 在上一篇博客中我...

 zhujunxxxx (<http://blog.csdn.net/zhujunxxxx>) 2015-03-14 17:54  5796

## c#异步Socket Tcp服务器实现 (/zhujunxxxxx/article/details/43574655)

原创性申明 本文作者: 小竹zz 本文地址:<http://blog.csdn.net/zhujunxxxx> 转载请注明出处。 介绍 在c#中微软已经提供了TcpListener和TcpClient...

 zhujunxxxx (<http://blog.csdn.net/zhujunxxxx>) 2015-02-06 21:31  5420

## C#TCP服务器简单程序 (/fenglifeng1987/article/details/17447559)



参考网上的程序写的，仅仅都是概念，离实际应用还差十万八千里 using System.Net; using System.Net.Sockets; using System.Threading; 用...



内容举报





返回顶部

 fenglifeng1987 (<http://blog.csdn.net/fenglifeng1987>) 2013-12-20 15:24  10488

---

## C# Socket简单例子（服务器与客户端通信）(/andrew\_wx/article/details/6629721)

这个例子只是简单实现了如何使用 Socket 类实现面向连接的通信。注意：此例子的目的只是为了说明用套接字写程序的大概思路，而不是实际项目中的使用程序。在这个例子中，实际上还有很多问题没有解决，如消息...

 Andrew\_wx ([http://blog.csdn.net/Andrew\\_wx](http://blog.csdn.net/Andrew_wx)) 2011-07-24 15:55  213777

---

  
内容举报

  
返回顶部