SinGooCMS 增加了一些新的工具类 History

1 contributor

584 lines (529 sloc) | 19.7 KB ...

```
1   using System;
2   using System.IO;
3   using System.Net;
4   using System.Net.Sockets;
5   using System.Runtime.Serialization;
6   using System.Runtime.Serialization.Formatters.Binary;
7   using System.Text;
8   using System.Threading;
9
10  namespace SinGooCMS.Utility
11  {
12      /// <summary>
13      /// Socket客户端操作类
14      /// </summary>
15      [Obsolete("似乎有点问题，暂时不建议使用")]
16      public static class SocketClient
17      {
18          #region 私有字段
19
20          /// <summary>
21          /// 设置数据缓冲区大小 默认1024
22          /// </summary>
23          private static readonly int m_maxpacket = 1024 * 4;
24
25          #endregion
26
27          #region 服务器侦听
28
29          /// <summary>
```

```
        /// 服务器侦听方法 返回null则说明没有链接上
        /// </summary>
        /// <returns>返回一个套接字(Socket)</returns>
        public static Socket ListenerSocket(this TcpListener listener)
        {
            try
            {
                return listener.AcceptSocket();
            }
            catch (Exception ex)
            {
                return null;
            }
        }

        /// <summary>
        /// 服务器侦听方法 返回null则说明没有链接上
        /// </summary>
        /// <param name="listener">TCP监听对象</param>
        /// <returns>返回一个网络流</returns>
        public static NetworkStream ListenerStream(this TcpListener listener)
        {
            try
            {
                return listener.AcceptTcpClient().GetStream();
            }
            catch (Exception)
            {
                return null;
            }
        }

        #endregion

        #region 客户端连接

        /// <summary>
        /// 从客户端连接获取socket对象
        /// </summary>
        /// <param name="tcpclient">TCP客户端</param>
        /// <param name="ipendpoint">客户端节点</param>
        /// <returns>客户端socket</returns>
        public static Socket ConnectSocket(this TcpClient tcpclient, IPEndPoint ipendpoint)
        {
            try
            {
                tcpclient.Connect(ipendpoint);
                return tcpclient.Client;
            }
```

```
            catch (Exception)
            {
                return null;
            }
        }

        /// <summary>
        /// 从客户端连接获取socket对象
        /// </summary>
        /// <param name="tcpclient">TCP客户端</param>
        /// <param name="ipadd">IP地址</param>
        /// <param name="port">端口号</param>
        /// <returns>客户端socket</returns>
        public static Socket ConnectSocket(this TcpClient tcpclient, IPAddress ipadd, int port)
        {
            try
            {
                tcpclient.Connect(ipadd, port);
                return tcpclient.Client;
            }
            catch (Exception)
            {
                return null;
            }
        }

        /// <summary>
        /// 从客户端获取网络流对象
        /// </summary>
        /// <param name="tcpclient">TCP客户端</param>
        /// <param name="ipendpoint">客户端节点</param>
        /// <returns>客户端的网络流</returns>
        public static NetworkStream ConnectStream(this TcpClient tcpclient, IPEndPoint ipendpoint)
        {
            try
            {
                tcpclient.Connect(ipendpoint);
                return tcpclient.GetStream();
            }
            catch (Exception)
            {
                return null;
            }
        }

        /// <summary>
        /// 从客户端获取网络流对象
        /// </summary>
        /// <param name="tcpclient">TCP客户端</param>
```

```
        /// <param name="ipadd">IP地址</param>
        /// <param name="port">端口号</param>
        /// <returns>客户端网络流对象</returns>
        public static NetworkStream ConnectStream(this TcpClient tcpclient, IPAddress ipadd, int p
        {
            try
            {
                tcpclient.Connect(ipadd, port);
                return tcpclient.GetStream();
            }
            catch (Exception)
            {
                return null;
            }
        }

        #endregion

        #region Socket接收数据

        /// <summary>
        /// 接受固定长度字符串
        /// </summary>
        /// <param name="socket">socket对象</param>
        /// <param name="size">字符串长度</param>
        /// <returns>字节数据</returns>
        public static byte[] ReceiveFixData(this Socket socket, int size)
        {
            int offset = 0;
            int dataleft = size;
            byte[] msg = new byte[size];
            while (dataleft > 0)
            {
                var recv = socket.Receive(msg, offset, dataleft, 0);
                if (recv == 0)
                {
                    break;
                }

                offset += recv;
                dataleft -= recv;
            }

            return msg;
        }

        /// <summary>
        /// 接收变长字符串
        /// 为了处理粘包问题 ,每次发送数据时 包头(数据字节长度) + 正文
```

```
        /// 这个发送小数据
        /// 设置包头的字节为8,不能超过8位数的字节数组
        /// </summary>
        /// <param name="socket">客户端socket</param>
        /// <returns>byte[]数组</returns>
        public static byte[] ReceiveVarData(this Socket socket)
        {
            //每次接受数据时,接收固定长度的包头,包头长度为8
            byte[] lengthbyte = ReceiveFixData(socket, 8);
            //length得到字符长度 然后加工处理得到数字
            int length = GetPacketLength(lengthbyte);
            //得到正文
            return ReceiveFixData(socket, length);
        }

        /// <summary>
        /// 接收T类对象,反序列化
        /// </summary>
        /// <typeparam name="T">接收T类对象,T类必须是一个可序列化类</typeparam>
        /// <param name="socket">客户端socket</param>
        /// <returns>强类型对象</returns>
        public static T ReceiveVarData<T>(this Socket socket)
        {
            //先接收包头长度 固定8个字节
            byte[] lengthbyte = ReceiveFixData(socket, 8);
            //得到字节长度
            int length = GetPacketLength(lengthbyte);
            byte[] bytecoll = new byte[m_maxpacket];
            IFormatter format = new BinaryFormatter();
            MemoryStream stream = new MemoryStream();
            int offset = 0; //接收字节个数
            int lastdata = length; //还剩下多少没有接收,初始大小等于实际大小
            int receivedata = m_maxpacket; //每次接收大小
            //循环接收
            int mark = 0; //标记几次接收到的数据为0长度
            while (true)
            {
                //剩下的字节数是否小于缓存大小
                if (lastdata < m_maxpacket)
                {
                    receivedata = lastdata; //就只接收剩下的字节数
                }

                int count = socket.Receive(bytecoll, 0, receivedata, 0);
                if (count > 0)
                {
                    stream.Write(bytecoll, 0, count);
                    offset += count;
                    lastdata -= count;
```

```
                    mark = 0;
                }
                else
                {
                    mark++;
                    if (mark == 10)
                    {
                        break;
                    }
                }

                if (offset == length)
                {
                    break;
                }
            }

            stream.Seek(0, SeekOrigin.Begin); //必须要这个 或者stream.Position = 0;
            T t = (T)format.Deserialize(stream);
            return t;
        }

        /// <summary>
        /// 在预先得到文件的文件名和大小
        /// 调用此方法接收文件
        /// </summary>
        /// <param name="socket">socket服务端</param>
        /// <param name="path">路径必须存在</param>
        /// <param name="filename">文件名</param>
        /// <param name="size">预先知道的文件大小</param>
        /// <param name="progress">处理过程</param>
        public static bool ReceiveFile(this Socket socket, string path, string filename, long size
        {
            if (!Directory.Exists(path))
            {
                return false;
            }

            //主要是防止有重名文件
            string savepath = GetPath(path, filename); //得到文件路径
            //缓冲区
            byte[] file = new byte[m_maxpacket];
            int receivedata = m_maxpacket; //每次要接收的长度
            long offset = 0; //循环接收的总长度
            long lastdata = size; //剩余多少还没接收
            int mark = 0;
            using (var fs = new FileStream(savepath, FileMode.OpenOrCreate, FileAccess.Write))
            {
                if (size <= 0)
```

```
275                 {
276                     return false;
277                 }
278 
279                 bool ret = false;
280                 while (true)
281                 {
282                     if (lastdata < receivedata)
283                     {
284                         receivedata = Convert.ToInt32(lastdata);
285                     }
286 
287                     var count = socket.Receive(file, 0, receivedata, SocketFlags.None); //每次接收的
288                     if (count > 0)
289                     {
290                         fs.Write(file, 0, count);
291                         offset += count;
292                         lastdata -= count;
293                         mark = 0;
294                     }
295                     else
296                     {
297                         mark++; //连续5次接收为0字节 则跳出循环
298                         if (mark == 10)
299                         {
300                             break;
301                         }
302                     }
303 
304                     //接收进度
305                     progress(Convert.ToInt32(Convert.ToDouble(offset) / Convert.ToDouble(size) * 1
306                     //接收完毕
307                     if (offset == size)
308                     {
309                         ret = true;
310                         break;
311                     }
312                 }
313 
314                 return ret;
315             }
316         }
317 
318         /// <summary>
319         /// 从socket服务端接收文件
320         /// </summary>
321         /// <param name="socket">socket服务端</param>
322         /// <param name="path">文件保存路径(必须存在)</param>
323         /// <param name="filename">文件名</param>
```

```
        /// <param name="size">预先知道的文件大小</param>
        /// <returns>处理结果</returns>
        public static bool ReceiveFile(this Socket socket, string path, string filename, long size
        {
            return ReceiveFile(socket, path, filename, size, null);
        }

        /// <summary>
        /// 预先不知道文件名和文件大小 用此方法接收
        /// 此方法对于的发送方法是SendFile()
        /// </summary>
        /// <param name="socket">socket服务端</param>
        /// <param name="path">要保存的目录</param>
        public static void ReceiveFile(this Socket socket, string path)
        {
            //得到包头信息字节数组 (文件名 + 文件大小 的字符串长度)
            //取前8位
            byte[] info_bt = ReceiveFixData(socket, 8);
            //得到包头信息字符长度
            int info_length = GetPacketLength(info_bt);
            //提取包头信息,(文件名 + 文件大小 的字符串长度)
            byte[] info = ReceiveFixData(socket, info_length);
            //得到文件信息字符串 (文件名 + 文件大小)
            string info_str = Encoding.UTF8.GetString(info);
            string[] strs = info_str.Split('|');
            string filename = strs[0]; //文件名
            long length = Convert.ToInt64(strs[1]); //文件大小
            //开始接收文件
            ReceiveFile(socket, path, filename, length);
        }

        private static int GetPacketLength(byte[] length)
        {
            string str = Encoding.UTF8.GetString(length);
            str = str.TrimEnd('*');
            return int.TryParse(str, out var _length) ? _length : 0;
        }

        private static int i;

        private static string markPath = string.Empty;

        /// <summary>
        /// 得到文件路径(防止有文件名重复)
        ///  如:aaa.txt已经在directory目录下存在,则会得到文件aaa(1).txt
        /// </summary>
        /// <param name="directory">目录名</param>
        /// <param name="file">文件名</param>
        /// <returns>文件路径</returns>
```

```
373         public static string GetPath(string directory, string file)
374         {
375             if (markPath == string.Empty)
376             {
377                 markPath = Path.Combine(directory, file);
378             }
379 
380             string path = Path.Combine(directory, file);
381             if (File.Exists(path))
382             {
383                 i++;
384                 string filename = Path.GetFileNameWithoutExtension(markPath) + "(" + i + ")";
385                 string extension = Path.GetExtension(markPath);
386                 return GetPath(directory, filename + extension);
387             }
388 
389             i = 0;
390             markPath = string.Empty;
391             return path;
392         }
393 
394         #endregion
395 
396         #region Socket发送数据
397 
398         /// <summary>
399         /// 发送固定长度消息
400         /// 发送字节数不能大于int型最大值
401         /// </summary>
402         /// <param name="socket">源socket</param>
403         /// <param name="msg">消息的字节数组</param>
404         /// <returns>返回发送字节个数</returns>
405         public static int SendFixData(this Socket socket, byte[] msg)
406         {
407             int size = msg.Length; //要发送字节长度
408             int offset = 0; //已经发送长度
409             int dataleft = size; //剩下字符
410             int senddata = m_maxpacket; //每次发送大小
411             while (true)
412             {
413                 //如过剩下的字节数 小于 每次发送字节数
414                 if (dataleft < senddata)
415                 {
416                     senddata = dataleft;
417                 }
418 
419                 int count = socket.Send(msg, offset, senddata, SocketFlags.None);
420                 offset += count;
421                 dataleft -= count;
```

```
422                 if (offset == size)
423                 {
424                     break;
425                 }
426             }
427 
428             return offset;
429         }
430 
431         /// <summary>
432         /// 发送变长信息 格式 包头(包头占8位) + 正文
433         /// </summary>
434         /// <param name="socket">发送方socket对象</param>
435         /// <param name="contact">发送文本</param>
436         /// <returns>发送的数据内容长度</returns>
437         public static int SendVarData(this Socket socket, string contact)
438         {
439             //得到字符长度
440             int size = Encoding.UTF8.GetBytes(contact).Length;
441             //包头字符
442             string length = GetSendPacketLengthStr(size);
443             //包头 + 正文
444             byte[] sendbyte = Encoding.UTF8.GetBytes(length + contact);
445             //发送
446             return SendFixData(socket, sendbyte);
447         }
448 
449         /// <summary>
450         /// 发送变成信息
451         /// </summary>
452         /// <param name="socket">发送方socket对象</param>
453         /// <param name="bytes">消息的 字节数组</param>
454         /// <returns>消息长度</returns>
455         public static int SendVarData(this Socket socket, byte[] bytes)
456         {
457             //得到包头字节
458             int size = bytes.Length;
459             string length = GetSendPacketLengthStr(size);
460             byte[] lengthbyte = Encoding.UTF8.GetBytes(length);
461             //发送包头
462             SendFixData(socket, lengthbyte); //因为不知道正文是什么编码所以没有合并
463             //发送正文
464             return SendFixData(socket, bytes);
465         }
466 
467         /// <summary>
468         /// 发送T类型对象,序列化
469         /// </summary>
470         /// <typeparam name="T">T类型</typeparam>
```

```
        /// <param name="socket">发送方的socket对象</param>
        /// <param name="obj">T类型对象,必须是可序列化的</param>
        /// <returns>消息长度</returns>
        public static int SendSerializeObject<T>(this Socket socket, T obj)
        {
            byte[] bytes = SerializeObject(obj);
            return SendVarData(socket, bytes);
        }

        /// <summary>
        /// 发送文件
        /// </summary>
        /// <param name="socket">socket对象</param>
        /// <param name="path">文件路径</param>
        /// <param name="issend">是否发送文件(头)信息,如果当前知道文件[大小,名称]则为false</param>
        /// <param name="progress">处理过程</param>
        /// <returns>处理结果</returns>
        public static bool SendFile(this Socket socket, string path, bool issend, Action<int> prog
        {
            if (!File.Exists(path))
            {
                return false;
            }

            var fileinfo = new FileInfo(path);
            string filename = fileinfo.Name;
            long length = fileinfo.Length;
            //发送文件信息
            if (issend)
            {
                SendVarData(socket, filename + "|" + length);
            }

            //发送文件
            long offset = 0;
            byte[] b = new byte[m_maxpacket];
            int mark = 0;
            using (var fs = new FileStream(path, FileMode.Open, FileAccess.Read))
            {
                int senddata = b.Length;
                //循环读取发送
                while (true)
                {
                    int count = fs.Read(b, 0, senddata);
                    if (count > 0)
                    {
                        socket.Send(b, 0, count, SocketFlags.None);
                        offset += count;
                        mark = 0;
```

```
520                     }
521                     else
522                     {
523                         mark++;
524                         if (mark == 10)
525                         {
526                             break;
527                         }
528                     }
529 
530                     progress(Convert.ToInt32(Convert.ToDouble(offset) / Convert.ToDouble(length) *
531                     if (offset == length)
532                     {
533                         return true;
534                     }
535 
536                     Thread.Sleep(50); //设置等待时间,以免粘包
537                 }
538             }
539 
540             return false;
541         }
542 
543         /// <summary>
544         /// 发送文件,不需要进度信息
545         /// </summary>
546         /// <param name="socket">socket对象</param>
547         /// <param name="path">文件路径</param>
548         /// <param name="issend">是否发生(头)信息</param>
549         /// <returns>处理结果</returns>
550         public static bool SendFile(this Socket socket, string path, bool issend)
551         {
552             return SendFile(socket, path, issend, null);
553         }
554 
555         /// <summary>
556         /// 发送文件,不需要进度信息和(头)信息
557         /// </summary>
558         /// <param name="socket">socket对象</param>
559         /// <param name="path">文件路径</param>
560         /// <returns>处理结果</returns>
561         public static bool SendFile(this Socket socket, string path)
562         {
563             return SendFile(socket, path, false, null);
564         }
565 
566         private static byte[] SerializeObject(object obj)
567         {
568             IFormatter format = new BinaryFormatter();
```

```
569             using (var stream = new MemoryStream())
570             {
571                 format.Serialize(stream, obj);
572                 return stream.ToArray();
573             }
574         }
575 
576         private static string GetSendPacketLengthStr(int size)
577         {
578             string length = size + "********"; //得到size的长度
579             return length.Substring(0, 8); //截取前前8位
580         }
581 
582         #endregion
583     }
584 }
```