



H2-November 14, 2008

الهم عجل لوليک الفرج

TCP/IP Socket Programming in Framework.Net2.0

سلام به تمام دوست داران برنامه نویسی !

هدف این PDF توضیح مختصری در خصوص برنامه نویسی سوکت ها، انحصاراً تحت پروتکل TCP/IP و در دات نت است.

انشا ... کمکی باشد برای دوستان دات نت کار، مخصوصاً C#.Net و VB.Net

امیدوارم دوستان آشنایی مختصری در مورد ذات وجودی سوکت ها داشته باشند.

بعد از توضیح مختصری در خصوص مفاهیم Socket و TCP و IP و Port در صفحات بعد کلاسهای System.Net.IPAddress و System.Net.IPEndPoint و System.Net.Sockets.Socket و System.Net.Dns به اختصار شرح داده میشوند.

مطالب بیان شده برای ساخت یک برنامه فوق العاده ساده چت (نمادی از برنامه شبکه ای) کافی است.

با توجه به اهمیت SQL Server در برنامه های شبکه ای دات نت، لازم دیدم قبل از شروع مطالب، مختصری در این باب توضیح دهم.

دوستان باید توجه داشته باشند که برای اینکه برنامه تحت شبکه کار کند، در خیلی از موارد نیازی نیست شما از برنامه نویسی مستقیم سوکت ها استفاده کنید.

تسلط بر سوکت ها خوب است ولی گاهی نیاز نیست.

اگر برنامه شما از یک بانک اطلاعاتی مثل SQL Server استفاده میکند و میخواهد دیتای واحدی را در رایانه های مختلف شبکه ای محلی یا حتی اینترنت در دو نقطه مختلف کره زمین، به اشتراک گذاشته و استفاده کند، لزومی به برنامه نویسی سوکت نیست.

فقط کافی است با چند کلیک:

- برنامه قدرتمند SQL Server را برای شبکه تنظیم کنید.
- برای دیتابیس مورد نظرتان یا کل دیتابیس ها Login و User و Password تعریف کنید، تا از آنها در رشته های اتصال ConnectionString در برنامه های روی کلاینتها یا حتی روی خود سرور استفاده کنید.

در برنامه بانک های اطلاعاتی شبکه ای مطلب فوق یک پیشنهاد نیست، بلکه روش استاندارد و منطقی پیاده سازی سیستم است.

سوکت مثل یک ترمینال است، یک پایانه ورودی و خروجی اطلاعات. میتوان انواع و اقسام بلوک های بایتی اطلاعات را به وسیله سوکت رد و بدل کرد.

اینکه چگونه و با چه فرمت و قالب و استاندارد همگانی تعریف شده ای اطلاعات را بین سوکت ها رد و بدل کنیم، نامش میشود پروتکل.

TCP/IP یکی از پروتکل های تحت شبکه و کار با سوکت ها است.

این پروتکل اتصال گرا است و اطمینان از دست نرفتن اطلاعات در آن زیاد است و یک تضمین هایی روی اتصال ایمن و رسیدن و تکرار بسته ها در صورت خطا هم دارد و به همین دلایل پروتکل استاندارد وب (www) هم شده.

شاید بتوان TCP را معروف ترین و پرکاربرد ترین پروتکل شبکه به حساب آورد.

هر رایانه میتواند بیش از یک IP داشته باشد، حداقل چون میتوانید بیش از یک کارت شبکه داشته باشد. !
همانطور که خودتان هم میدانید، IP نشانی و آدرس پستی رایانه شما در شبکه است.
که مشخص میکند بسته اطلاعات باید تحویل چه رایانه ای شود.

IP دو نسخه کاربردی دارد:

- آدرسهای ورژن 4 که از اعدادی 32 بیتی استفاده میکرده و در تئوری 2 به توان 32 حالت و رایانه را ساپورت میکند.

در نوشتار این IP چنین تایپ میشود : 217.85.72.32

- با گسترش شبکه جهانی اینترنت، کوچکی محدوده فوق در دسر ساز شد و نسخه 6 به وجود آمد.
آدرسهای ورژن 6 که از اعدادی 128 بیتی استفاده میکند، در تئوری 2 به توان 128 حالت و رایانه را ساپورت میکند. یعنی 2 به توان 96 حالت بیش از ورژن 4.

در نوشتار این IP چنین تایپ میشود : fe80::225:96ef:dd26:d582

(در این خصوص مطالب دیگری هم وجود دارد، که با توجه به تمرکز بحث روی سوکت از آنها صرف نظر میکنیم.)

برخی اعداد IP ، انحصاری هستند و معنی خاصی دارند، فقط به عنوان نمونه :

Loopback (IPv4) 127.0.0.1 = (IPv6) ::1	همواره به رایانه جاری اشاره میکنند. دقت کنید که با توجه به آدرس Loopback حتی رایانه ای که کارت شبکه هم ندارد حداقل یک IP برای خودش دارد.
Broadcast (IPv4) 255.255.255.255	آدرسی است که اگر چیزی به آن ارسال شود، همه رایانه ها در شبکه آن دیتا را دریافت میکنند و در نسخه شش دیگر ساپورت نمیشود.
Wildcard (IPv4) 0.0.0.0 = (IPv6) ::-	معرف هر آدرس و هر رابطی است.
	و ...

Port را به نوعی میتوان کد صندوق پستی برنامه داخل رایانه بیان کرد. شما در یک لحظه میتوانید :

- در حال چک کردن ایمیلتان در برنامه ای باشید ...
- میتوانید با مرورگر خود، یک صفحه وب را ببینید ...
- به ftp وصل شده باشید ...
- یک برنامه چت هم باز باشد ...
- SQLServer هم در گوشه ای کار خودش را کند ... !
- و ...

در این بین و با یک IP مشخص چگونه اطلاعات تفکیک میشوند و به برنامه مناسب میرسند ؟
اینجا اهمیت Port مشخص میشود.

اعداد پورت های زیر 1024 برای استانداردهای خاصی رزو شده اند. مثلاً :

Port:	20	FTP data
Port:	21	FTP control
Port:	23	Telnet
Port:	25	SMTP
Port:	53	DNS
Port:	80	HTTP
Port:	110	POP3
Port:	135	RPC
Port:	143	IMAP
Port:	443	HTTPS
Port:	993	IMAP SSL
Port:	995	POP3 SSL
Port:	1433	SQLServer

اگر برایتان نام پورتهای مهم جالب است، سری به این دو لینک زیر بزنید:

http://www.microsoft.com/resources/documentation/windowsnt/4/server/reskit/en-us/net/port_nts.mspx?mfr=true

<http://www.iana.org/assignments/port-numbers>

البته این رزو به معنای این نیست که شما نمیتوانید به آنها وصل شوید و از آنها استفاده کنید، بلکه یک استاندارد جهانی است.

همه انتظار دارند یک سوکت سرور روی پورت 80 خدمات HTTP بدهد نه اینکه سیب زمینی بفروشد !

شما میتوانید از هر پورتی که خالی باشد و برنامه دیگری از آن استفاده نکند، استفاده کنید، برای کارهای خودتان و خارج از چارت پروتکل‌های پایه شبکه، بهتر است اعداد بالای 1024 را انتخاب کنید.

این کلاس صرفاً یک ظرف است، یک ظرف برای نگه داری IP همانطور که Integer یک ظرف برای نگه داری اعداد صحیح است. همانطور که String یک ظرف برای نگه داری متون رشته است. یعنی هیچ پویایی ندارند، به ذات به هیچ جا وصل نمیشوند.

شما میتوانید هر IP چه ورژن 4 و چه ورژن 6 را در یک کلاس System.Net.IPAddress جای دهید. برای پر کردن این کلاس میتوان از دستورات زیر استفاده کرد که همه با هم برابر هستند و در نهایت موجب تولید یک IPAddress واحد میشوند.

'VB.Net

```
Dim ip As New System.Net.IPAddress(New Byte(){127, 0, 0, 1})
Dim ip As New System.Net.IPAddress(&H100007FL)
Dim ip As System.Net.IPAddress = System.Net.IPAddress.Parse("127.0.0.1")
```

//C#.Net

```
System.Net.IPAddress ip = new System.Net.IPAddress(new Byte[]{127, 0, 0, 1});
System.Net.IPAddress ip = new System.Net.IPAddress(0x100007FL);
System.Net.IPAddress ip = System.Net.IPAddress.Parse("127.0.0.1");
```

- دستور اول آرایه بایتی میگردد، شامل تک تک بایت های IP .
 - دستور دوم یک عدد Long میگیرد که با اصول ساده باینری و مبنای شانزده میتواند با آن کار کرد.
 - دستور آخر یک String را سعی میکند به IP تبدیل کند.
- یعنی میتوان مقدار یک TextBox را عملاً به متد Parse داد و تبدیل شده اش به IPAddress را تحویل گرفت. اگر رشته متنی ورودی یک چیز اشتباه و بی ربط و غیر قابل تبدیل به IPAddress باشد یک خطا دریافت خواهید کرد.

اغلب ساختارها و کلاسهای که در دات نت که امکان تبدیل از String دارند متد Parse را هم دارند.

'VB.Net

```
Dim i As Integer = Integer.Parse(StringValue)
```

//C#.Net

```
int i = int.Parse(StringValue);
```

دیگر اعضای مهم کلاس IP Address:

- TryParse عمل Parse را به نوعی دیگر انجام میدهد به جای خطا، یک Boolean پس میدهد که نشانه موفقیت/عدم موفقیت در تبدیل است.
- ToString میتواند از این متد برای تبدیل IP به رشته متنی جهت نمایش و... استفاده کنید.
- Equals مقایسه با یک IP Address دیگر
- Loopback یک فیلد ثابت است که همواره آدرس IPv4-Loopback را بر میگرداند.
- Broadcast یک فیلد ثابت است که همواره آدرس IPv4-Broadcast را بر میگرداند.
- Any یک فیلد ثابت است که همواره آدرس IPv4-Wildcard را بر میگرداند.
- متدهای ثابت IPv6XXXXX هم مشابه موارد فوق هستند.

یعنی برای آدرس 127.0.0.1 کد زیر نیز با سه کد درج شده در اول این پست، برابر هستند و هر چهار کد یک نتیجه را در بر دارند:

'VB.Net

```
Dim ip As System.Net.IPAddress = System.Net.IPAddress.Loopback
```

//C#.Net

```
System.Net.IPAddress ip = System.Net.IPAddress.Loopback;
```

این کلاس هم مثل کلاس قبلی یعنی IPAddress فقط یک ظرف است.

ظرفی برای نگاه داشتن IP+Port

یعنی عملاً یک IPEndPoint شامل آدرس کامل اتصال به یک سوکت است. هم عدد IP را دارد و هم عدد Port را.

بدین شکل هم ساخته میشود و نتیجه هر سه IPEndPoint زیر برابر است.

'VB.Net

```
Dim ip As New System.Net.IPAddress(New Byte(){127, 0, 0, 1})
Dim port As New System.Net.IPEndPoint(ip, 5050)
Dim port As New System.Net.IPEndPoint(&H100007FL, 5050)
Dim port As New System.Net.IPEndPoint(System.Net.IPAddress.Parse("127.0.0.1"), 5050)
'etc ...
```

//C#.Net

```
System.Net.IPAddress ip = new System.Net.IPAddress(new Byte[]{127, 0, 0, 1});
System.Net.IPEndPoint port = new System.Net.IPEndPoint(ip, 5050);
System.Net.IPEndPoint port = new System.Net.IPEndPoint(0x100007FL, 5050);
System.Net.IPEndPoint port = new
System.Net.IPEndPoint(System.Net.IPAddress.Parse("127.0.0.1"), 5050);
//etc ...
```

بعد از نمونه سازی با دستور new اگر خودمان هم به IPAddress و Port نگه داری شده در آن نیاز داشتیم، این کلاس دو Property با نام Address و Port و به همین منظور دارد که به ترتیب مقدار IPAddress و Port را بازمیگردانند.

برای ساخت یک نمونه Socket جدید نیاز به سه پارامتر داریم:

System.Net.Sockets.AddressFamily (1)

پارامتر اول ساختار و نوع IP را مشخص میکنید که پرکاربرد ترین گزینه ها InterNetwork برای IPv4 و InterNetworkV6 برای IPv6 است.

کلاس IPAddress هم یک ReadOnly Property با نام AddressFamily دارد، که میتوان از این مقدار برای این پارامتر اول ساخت Socket هم استفاده کرد.

System.Net.Sockets.SocketType (2)

پارامتر دوم انتقال دیتاها را توسط سوکت مشخص میکند. که برای بحث ما و تحت TCP/IP گزینه Stream را باید استفاده کنیم، این گزینه یک زنجیره اطلاعات را مشخص میکند.

احتمالاً باید بدانید Stream به چه مفهوم است، یک قطار دیتا، مثل نوارنقاله یک کارخانه که بسته های کالا رویش حرکت میکند.

به ذات بسته ها تکرار نمیشوند و قابلیت دسترسی به هر نقطه را هم ندارید، یک چیزی شبیه دانلود معمولی، که بایت به بایت را میگیرد و در هارد ذخیره میکنید.

System.Net.Sockets.ProtocolType (3)

پارامتر سوم و آخر هم پروتکل کاری را ست میکند، یک لایه استاندارد بالاتر.

پرکاربردترین گزینه ها هم Tcp و Udp هستند، که برای بحث ما فقط Tcp باید انتخاب شود.

باید توجه داشت که با ساخت Socket جدید هنوز هیچ اتفاق و وصل شدنی انجام نشده.

'VB.Net

```
Dim ip As System.Net.IPAddress = System.Net.IPAddress.Parse("127.0.0.1")
```

```
Dim point As New System.Net.IPEndPoint(ip, 80)
```

```
Dim socket As New System.Net.Sockets.Socket(System.Net.Sockets.AddressFamily.InterNetwork,  
System.Net.Sockets.SocketType.Stream, System.Net.Sockets.ProtocolType.Tcp)
```

```
Dim socket As New System.Net.Sockets.Socket(ip.AddressFamily,  
System.Net.Sockets.SocketType.Stream, System.Net.Sockets.ProtocolType.Tcp)
```

```
Dim socket As New System.Net.Sockets.Socket(point.AddressFamily,  
System.Net.Sockets.SocketType.Stream, System.Net.Sockets.ProtocolType.Tcp)
```

//C#.Net

```
System.Net.IPAddress ip = System.Net.IPAddress.Parse("127.0.0.1");
```

```
System.Net.IPEndPoint point = new System.Net.IPEndPoint(ip, 80);
```

```
System.Net.Sockets.Socket socket = new  
System.Net.Sockets.Socket(System.Net.Sockets.AddressFamily.InterNetwork,  
System.Net.Sockets.SocketType.Stream, System.Net.Sockets.ProtocolType.Tcp)
```

```
System.Net.Sockets.Socket socket = new System.Net.Sockets.Socket(ip.AddressFamily,  
System.Net.Sockets.SocketType.Stream, System.Net.Sockets.ProtocolType.Tcp)
```

```
System.Net.Sockets.Socket socket = new System.Net.Sockets.Socket(point.AddressFamily,  
System.Net.Sockets.SocketType.Stream, System.Net.Sockets.ProtocolType.Tcp)
```

مرحله بعد از ساخت سوکت، اتصال سوکت است.

برای این امر دو متد Bind و Connect وجود دارد که هر دو یک IPEndPoint میگیرند.

(لازم نیست یاد آوری کنم مطالب با محوریت پروتکل TCP بیان میشوند.)

متد Bind به سیستم عامل اعلام میکند که شما را به پورت خاصی وصل کند.

این متد را برای سوکت های سرور استفاده میکنیم تا به سیستم عامل اعلام کنیم تمام درخواست های رسیده به یک آدرس و پورت خاص را به سوکت و برنامه ما تحویل دهد.

متد Connect شما را برای گرفتن خدمات به آدرس و پورت خاصی وصل میکند.

این متد را فقط برای سوکت های سمت کلاینت استفاده میکنیم تا سیستم عامل ما را به یک سوکت برای خدمات گیری وصل کند.

این متد در زیر ساز داخلی سیستم عامل کار Bind را هم انجام میدهد و اصولاً نیاز به Bind را بر طرف میکند.

به صورت ساده میتوان این طور گفت که Bind متد اتصال برای سوکت سرور است و Connect متد اتصال برای سوکت کلاینت.

هر دو این متدها به دلایل مختلف میتوانند به خطا مواجه شوند مثلاً :

- در سرور شاید یک سوکت دیگری پورت را قبلاً گرفته باشد.
- در کلاینت شاید سوکت نظیر سرور فعال نباشد.
- یا ایرادی در ارتباط شبکه باشد.
- و...

در نهایت تنظیم خصوصیت ExclusiveAddressUse قبل از Bind معمولاً در سوکت های سمت سرور مشخص میکند که باید پورت انحصاراً در اختیار سوکت ما باشد یا نه.

(کد خاصی ندارد، کافیه نمونه IPEndPoint را تحویل متد بدهید.)

پس از اتمام ارسال ها و دریافت ها، در نهایت برای قطع ارتباط یک سوکت میتوان از دستورات Close و Shutdown و Dispose استفاده کرد.

در خصوص شی سوکت دستور Close و Dispose عملکرد یکسانی دارند و سوکت را کاملاً بسته و به نوعی نابود میکند و منابع سیستم عامل را آزاد میکنند.

اما دستور Shutdown یک پارامتر از نوع System.Net.Sockets.SocketShutdown میگیرد و میتواند خط ارسال و دریافت را جداگانه ببندد.

پیغامی که در عوض بسته شدن سوکت در یک سر به سوکت نظیر در آن سر میرسد، دریافت اطلاعات جدید با طول صفر بایت است.

هنوز دستور دریافت اطلاعات را نگفته ایم ولی فعلاً بدانید که اگر اطلاعاتی با تعداد بایت صفر، دریافت شود، یعنی سوکت نظیر در آن سمت دیگر وجود ندارد و قاطعاً باید در این سمت هم در صورت نیاز منابع سیستم را آزاد کنیم.

در برخی برنامه ها که بسته شدن کامل و امن هر دو سوکت برایشان خیلی مهم است، اول Shutdown را با Send اجرا میکنند و بعد هم منتظر رسیدن پیغام نظیر از سوکت سمت دیگر میشوند و بعد سوکت را کامل مینند.

ولی اکثراً یک Close ساده، کافی است.

سه دستور زیر تقریباً یک نتیجه دارند:

'VB.Net

socket.Close()

DirectCast(socket, System.IDisposable).Dispose()

socket.Shutdown(System.Net.Sockets.SocketShutdown.Both)

//C#.Net

socket.Close();

((System.IDisposable)socket).Dispose();

socket.Shutdown(System.Net.Sockets.SocketShutdown.Both);

سوکت کلاینت به محض اجرای دستور Connect آماده ارسال و دریافت اطلاعات است ولی در سرور عملاً باید دستورات Listen و Accept نیز اجرا شوند.

متد Listen سرور را برای پذیرفتن درخواستهای کلاینتها به یک تعداد صف مشخص آماده میکند، معمولاً آن را با حداکثر مقدار ممکن فراخوانی میکنند تا در آینده و با کوچک و بزرگ شدن شبکه و توسعه و ... همواره کد صحیح کار کند، ولی مقداردهی یک عدد منطقی و مورد انتظار شاید بد نباشد و مثلاً میتواند یک نیمچه سدی در مقابل حملات DOS باشد.

سه دستور زیر دقیقاً یک کد IL تولید میکنند.

'VB.Net

'socket = New ...

'socket.Bind (...)

socket.Listen(Integer.MaxValue)

socket.Listen(&H7FFFFFFF)

socket.Listen(2147483647)

//C#.Net

//socket = new ...

//socket.Bind (...)

socket.Listen(int.MaxValue);

socket.Listen(0x7FFFFFFF);

socket.Listen(2147483647);

در کلاینت اوضاع خیلی ساده است یک شی سوکت میسازیم و با همان اقدام به دریافت و ارسال اطلاعات میکنیم، در واقعاً هر کلاینت فقط یک وظیفه دارد و آن اتصال به سرور است !

ولی در سرور خیلی جزئی اوضاع فرق میکند و سرور باید یک خط ارتباطی با هر یک از کلاینت ها به طور مجزا داشته باشد، در نتیجه همان یک نمونه سوکتی که اول در سرور میسازیم کفایت نمیکند و ما نیاز به یک خط ویژه ارتباطی با هر کلاینت داریم.

برای آنکه بهتر متوجه شوید، مرکز تلفن 118 را در نظر بگیرید، در ظاهر فقط یک تلفن 118 وجود دارد و این خیلی بدیهی است که از هر جایی کافی است فقط 118 را بگیریم، این لفظ کلی تلفن 118، مثل همان شی سوکت اصلی سرور میماند که خودمان در سرور با New آن را میسازیم.

ولی بعد هر کس با 118 تماس میگیرد، به یک اپراتور مجزا و یک گوش تلفن مجزا وصل میشود، شماره تماس همه گوشی‌های تلفن مرکز، همان 118 است ولی مثل تماس گیرنده، اینجا یک گوشی وجود ندارد و کفایت نمیکند، اگر یک گوشی در 118 باشد، طبیعی است که همه صحبت‌های تماس گیرنده و همه جواب‌های مرکز قاطی میشوند.

متد Accept مسئول برداشتن گوشی جدید در سرور و ایجاد یک خط ارتباطی مستقل است.

با فراخوانی این متد ریسمان شما بلوکه و متوقف میشود تا اولین درخواست برقراری ارتباط از یک کلاینت برسد، به محض رسیدن درخواست، متد Accept ارتباط را میپذیرد و آن را برقرار میکند و یک نمونه جدید از شی سوکت به ما میدهد که این نمونه فقط مخصوص ارتباط ما با آن همان یک کلاینت است.

یعنی تمام گرفتن و ارسال اطلاعات ما به یک کلاینت خاص باید توسط این نمونه سوکت جدید انجام میشود.

دستور زیر (در سمت سرور) آنقدر صبر میکند تا فقط یک کلاینت در خواست اتصال بدهد، آنگاه اتصال را پذیرفته و یک پیغام متنی Hello برای کلاینت ارسال میکند و اتصال را پایان میدهد و میندد و همه چیز تمام میشود !

(دستور Send را توضیح خواهیم داد، فعلاً بدانید که وظیفه ارسال اطلاعات را به عهده دارد.)

```

'VB.Net

'socket = New ...

'socket.Bind ( ...

'socket.Listen ( ...


Dim msgbuffer() As Byte = System.Text.Encoding.UTF8.GetBytes("Hello")


Dim clientsocket As System.Net.Sockets.Socket = socket.Accept()
clientsocket.Send(msgbuffer)
clientsocket.Close()


//C#.Net

//socket = new ...

//socket.Bind ( ...

//socket.Listen ( ...


byte[] msgbuffer = System.Text.Encoding.UTF8.GetBytes("Hello");


System.Net.Sockets.Socket clientsocket = socket.Accept();
clientsocket.Send(msgbuffer);
clientsocket.Close();

```

برای یک برنامه واقعی باید Accept را در یک حلقه تکرار کرد تا تمام درخواستهای همه کلاینتها پردازش شود. مثلاً کد زیر هر (در سمت سرور) درخواستی را که برسد قبول میکند و یک پیغام Hello به آن ارسال میکند و مکالمه را پایان میدهد و مجدداً منتظر درخواست بعدی میشود.

'VB.Net

'socket = New ...

'socket.Bind (...

'socket.Listen(...

Dim msgbuffer() As Byte = System.Text.Encoding.UTF8.GetBytes("Hello")

Do

Dim clientsocket As System.Net.Sockets.Socket = socket.Accept()

clientsocket.Send(msgbuffer)

clientsocket.Close()

Loop

//C#.Net

//socket = new ...

//socket.Bind (...

//socket.Listen (...

byte[] msgbuffer = System.Text.Encoding.UTF8.GetBytes("Hello");

while (true)

{

System.Net.Sockets.Socket clientsocket = socket.Accept();

clientsocket.Send(msgbuffer);

clientsocket.Close();

}

برای ارسال و دریافت اطلاعات، تنها فرمت مورد قبول در سطح سوکت، اطلاعات باینری خالص است، آرایه بایتی.

وظیفه کد ما است که آیتم مورد نظر برای ارسال را Serialize به آرایه بایتی کند و پس از دریافت هم Deserialize کردن به دیتا بر عهده کد ما است.

- تبدیلات دیتاهای معمول مثل Integer و Double و... میتوان با کلاسهای System.IO.BinaryReader و System.IO.BinaryWriter انجام داد.

- تبدیل String را هم میتوان با کلاس System.Text.Encoding.UTF8 انجام داد.

- تبدیل Image هم میتواند با کلاسهای System.IO.MemoryStream و متدهای FromStream و Save انجام شود.

- برای اشیای پیچیده و ترکیبی که تبدیلات آنها مشکل است میتوان از System.Runtime.Serialization.Formatters.Binary.BinaryFormatter استفاده کرد ولی این تبدیلات از تبدیلات مشابه فوق الذکر کندتر و حجیم تر هستند.

- و...

متد Send جهت ارسال اطلاعات استفاده میشود و چندین Overload دارد، یکی از کاملترین Overload های این متد چهار پارامتر میگیرد.

buffer() As Byte (1)

پارامتر اول خود آرایه بایتی است که باید ارسال شود.

offset As Integer (2)

پارامتر دوم ایندکس آیتم شروع در آرایه است که معمولاً عدد صفر است.

size As Integer (3)

پارامتر سوم تعداد بایتهایی است که از محل مشخص شده باید ارسال شوند.

socketFlags As System.Net.Sockets.SocketFlags (4)

پارامتر آخر فلاگهای مشخصه ارسال است که ما گزینه None را انتخاب میکنیم.

متد Receive جهت دریافت اطلاعات استفاده میشود و چندین Overload دارد، یکی از کاملترین Overload های این متد سه پارامتر میگیرد و یک عدد باز میگرداند، عدد مذکور میزان بایتی اطلاعات رسیده را نشان میدهد.

این دستور باید در یک حلقه فراخانی شود و کد ما مدام گوش بزرگ رسیدن دیتای جدید باشد.

buffer() As Byte (1)

پارامتر اول آرایه خالی است که ما خودمان به متد مذکور میدهیم تا به عنوان بافر، اطلاعات رسیده را در آن بریزد.

(البته خالی نه به معنای پر از صفر! خالی به معنای آرایه ای که محتویات جاری اش دیگر برای ما مهم نیست و آماده است تا اطلاعات جدید در آن ریخته شود.)

size As Integer (2)

پارامتر دوم، اندازه ای از آرایه ای است که متد مذکور اجازه دارد از آن استفاده کند.

socketFlags As System.Net.Sockets.SocketFlags (3)

پارامتر آخر هم همانطور که در قبل گفتیم فلاگ های خاص نحوه دریافت و ارسال است که ما از گزینه None استفاده خواهیم کرد.

توجه کنید که اگر ما آرایه ای 10K بایتی بدهیم، لزومی به صبر برای پر شدن کل آرایه نیست و یک بسته 1 بایتی اطلاعات هم برسد، کافیت تا مدت انجام شده و به انتها برسد.

اگر بلوک بسته ای اطلاعات رسیده شده، بیشتر از حجم بافر ما باشد، تا حدی که بافر ما اجازه دهد اطلاعات را دریافت میکنیم و مابقی را در اجرای بعدی Receive دریافت خواهیم کرد.

کد خاصی هم که فعلاً ارزش نوشتن داشته باشد، ندارد!

دستورات گفته شده تا این لحظه همه از ریسمان جاری استفاده میکنند و تا اتمام کار، ریسمان بلوکه میشود و منتظر خواهد ماند.

در نتیجه باید از چند ریسمانی استفاده کنیم، یا خودمان اقدام به ساخت ریسمان جدید کنیم و یا از جفت دستورات BeginXXXXXX و EndXXXXXX استفاده کنیم.

جفت دستورات نا همزمان فوق نسبت به دستورات مشابه عادی دو پارامتر بیشتر میگیرند.

requestCallback As System.AsyncCallback (1)

پارامتر اول هندلر و اشاره گر روالی است که در پایان عملیات فراخوانی میشود.

state As Object (2)

پارامتر دوم هم اختیاری است و میتواند هر مقداری داشته باشد، هر چه به این مقدار بدهید در اجرای روال پایانی به شما داده میشود و به نوعی شما میتوانید وضعیت جاری و یا مقداری برای مشخص کردند حالات مختلف را به روال پایانی ارسال کنید.

روال پایانی که در انتهای انجام عمل نا همزمان اجرا میشود و باید اشاره گر آن به پارامتر requestCallback داده شود و دارای امضا و فرمت زیر است:

```
'VB.Net
```

```
Sub MethodName (ByVal ar As System.IAsyncResult)
```

```
//C#.Net
```

```
void MethodName (System.IAsyncResult ar)
```

ar که به ما تحویل داده میشود مشخص کنند مواردی از جمله موفقیت و اتمام صحیح عملیات و همان پارامتر state هم است (البته با نام AsyncState)

برای اتمام کلی فرایند و چرخه اجرا، این پارامتر ar باید به EndXXXXXX پاس داده شود و اگر عملیات مقدار بازگشتی هم داشته باشد EndXXXXXX ان را به ما میدهد.

به مثال فرضی زیر که نمونه کد همزمان و نا همزمان را استفاده کرده، دقت کنید:

'VB.Net - Synchronous

```
Dim ret As TypeName = ClassVariantName.FunctionName(125)
```

```
\...
```

```
'
```

'VB.Net - Asynchronous

```
ClassVariantName.BeginFunctionName(125, AddressOf MethodName, "Mahdi")
```

```
Private Sub MethodName (ByVal ar As System.IAsyncResult)
```

```
    ' "Mahdi" === ar.AsyncState
```

```
    Dim ret As TypeName = ClassVariantName.EndFunctionName(ar)
```

```
    '...
```

```
End Sub
```

//C#.Net - Synchronous

```
TypeName ret = ClassVariantName.FunctionName(125);
```

```
//...
```

```
//
```

//C#.Net - Asynchronous

```
ClassVariantName.BeginFunctionName(125, new System.AsyncCallback(MethodName), "Mahdi");
```

```
private void MethodName (System.IAsyncResult ar)
```

```
{
```

```
    // "Mahdi" === ar.AsyncState;
```

```
    TypeName ret = ClassVariantName.EndFunctionName(ar);
```

```
    //...
```

```
}
```

فراموش نکنید مزیت روش غیر همزمان آن است که ریسمان شما بلوکه نمیشود و ریسمان شما میتواند به کارهای دیگر برسد و هر موقع عملیات تکمیل شد، روال پایانی اجرا شود و مقدار بازگشتی را هم تحویل بگیرید.

مطالب فوق را قابل تعمیم بیان کردم و عیناً میتوانید برای هر کلاسی در دات نت که جفت روال BeginXXXX و EndXXXX را دارد اجرا کنید، از جمله:

BeginConnect و BeginAccept و BeginSend و BeginReceive و...

کافی است تست کنید !

اغلب مواقع، در کدها و برنامه ها به جای آنکه IP را مستقیم از کاربر بگیریم از نام استفاده میکنیم. یک رایانه یا سرور در شبکه میتواند نام هم داشته باشد و همانطور که قبلاً هم گفتیم، هر رایانه میتواند چندین IP داشته باشد.

کلاس Dns وظیفه ترجمه نام به IP ها را بر عهده دارد، یعنی ما وقتی به شبکه وصل هستیم نام یک رایانه در شبکه را میدهیم و IP هایش را تحویل میگیریم.

مثلاً اگر به شبکه اینترنت وصل باشیم میتوانیم نام `www.microsoft.com` را بدهیم و IP هایش را به دست آوریم.

با این کلاس و کمی کد اضافه برنامه میتواند علاوه بر IP با نام هم تغذیه شود و کار کند و خودش خودکار از نام به IP برسد.

مثلاً کد زیر سعی میکند به همه IP های متناظر یک نام وصل شود و به اولین IP که وصل شد و کار کرد، جستجو را تمام کند.

(میتوانند هر کجا نیاز بود، IP را به طریقه فوق به دست آورید.)

'VB.Net

Dim port As Integer = 80

Dim socket As System.Net.Sockets.Socket = Nothing

For Each ip As System.Net.IPAddress In
System.Net.Dns.GetHostEntry("www.microsoft.com").AddressList

Try

socket = _

New System.Net.Sockets.Socket(_

ip.AddressFamily, _

System.Net.Sockets.SocketType.Stream, _

System.Net.Sockets.ProtocolType.Tcp)

socket.Connect(New System.Net.IPEndPoint(ip, port))

Exit For

Catch

socket = Nothing

End Try

Next

If (socket IsNot Nothing) Then

' ...

' ...

socket.Close()

End If

```

//C#.Net

int port = 80;

System.Net.Sockets.Socket socket = null;

foreach(System.Net.IPAddress ip in
System.Net.Dns.GetHostEntry("www.microsoft.com").AddressList)
{
    Try {
        socket =
            new System.Net.Sockets.Socket(
                ip.AddressFamily,
                System.Net.Sockets.SocketType.Stream,
                System.Net.Sockets.ProtocolType.Tcp);

        socket.Connect(new System.Net.IPEndPoint(ip, port));
        break;
    }
    Catch {
        socket = null;
    }
}

if (socket != null) {
    //...
    //...
    socket.Close();
}

```

H2
حاجی شریفی
November 14, 2008
info [at] h02 [dot] ir
http://h02.ir
http://persianscience.ir