TCP / IP – Teil 2: Verbindungsaufbau

Verbindungsaufbau

Server:

1. Programm muss dem BS mitteilen, dass es nun Verbindungen über Port X annehmen möchte.

2. Der Firewall Port X öffnen, damit darüber kommuniziert werden kann

Server wartet nun auf eine Verbindung über Port X von außen

Client:

Programm möchte eine Verbindung über Port X zu Host herstellen.

1. Dem Betriebssystem mitteilen, dass es eine Verbindung herstellen möchte

2. Das Betriebssystem weist dem Client nun auch einen freien Port ab 1024 zu

3. Versuchen eine Verbindung zu Host herzustellen

Server:

1. erkennt, dass zu ihm eine Verbindung hergestellt werden will und nimmt diese an. Bekommt auch die Mitteilung auf welchem PC und an welcher Portnummer der Client läuft.

2. Damit der Server in der Lage ist mehrere Verbindungen von mehreren Clients anzunehmen und zu verwalten, erhält jede Verbindung eine Verbindungsnummer auch Handle genannt.

3. für jede Verbindung wird ein Prozess zur Verfügung gestellt, welcher nur diese Verbindung verarbeitet.

Der Server wartet also nur auf eine Anfrage vom Client und gibt dann die Antwort zurück.

Nach Abarbeitung kann sowohl der Server als auch der Client die Verbindung beenden.

Wenn eine Zeitlang keine Reaktion vom Client kommt, kann der Server sagen „timeout!“ und die Verbindung schließen.

TCP / IP – Teil 1: Begriffserklärung

kurze Begriffserklärung:

Server

ist kein Rechner an sich, sondern es ist nur ein Programm, oft ein Dienst welches auf einem Rechner läuft und seine Dienste zur Verfügung stellt.

Client

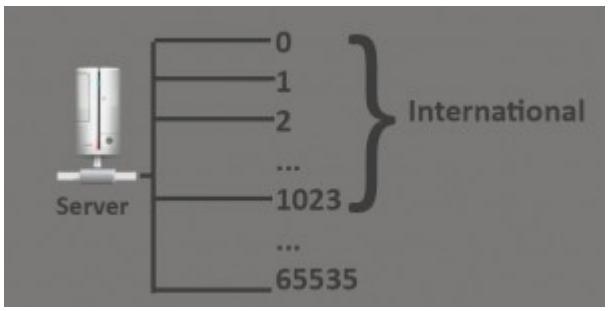
ist genauso wie der Server nur ein Programm, welches jedoch die Dienste empfängt.

Host

Der Rechner an sich wird dann als ein Host bezeichnet, wenn

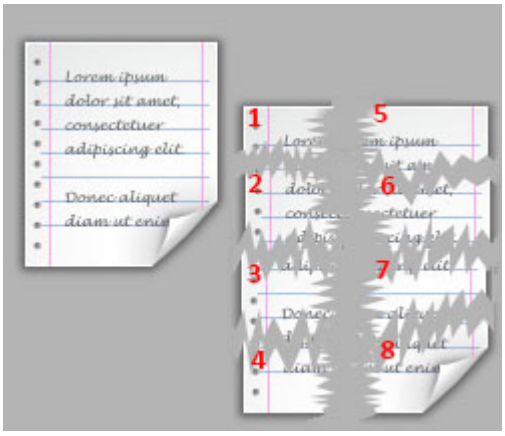
Datenübertragungen stattfinden.

Port



Vergleichbar mit einer Strasse mit 65536 Häusern. Jedes dieser Ports kann eine Aufgabe übernehmen. Berühmte standardisierte Ports sind z.B. Port 80 für das WWW / HTTP, Port 21 für FTP. Dabei sind die Ports von 0 – 1023 standardisiert für Internationale Zwecke bestimmt wie z.B. die eben genannten.

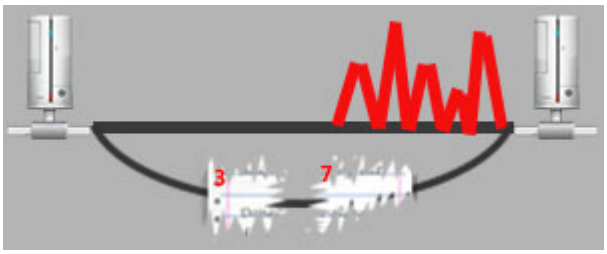
TCP – Transmission Control Protokoll



Die Aufgabe von TCP ist es, das Datenpaket, welches zugrunde liegt zu nehmen und in viele einzelne Teile zu splitten. Jedes dieser Teile wird nummeriert.

Nach der Übertragung auf dem Zielsocket, sind die einzelnen Datenpakete durcheinander. Es ist nun ebenfalls die Aufgabe von TCP diese nummerierten Pakete in richtiger Reihenfolge aufzustellen. Nachdem die einzelnen Elemente zusammengefügt wurden, erhält das Zielsocket eben dieselbe Datei wie die Quelle diese geschickt hat.

IP – Internet Protokoll



Die Aufgabe des Internet Protokolls ist es nun die kürzeste Verbindung zu finden. Ist diese aber Fehlerhaft, so sucht die IP nach einer Alternativen Route und übermittelt darüber. Dies sind die 2 Aufgaben der IP.

Sockets

Auf Deutsch auch Sockel sind die beiden Endpunkte vom Server und Client. Diese kann man sich wie eine Steckdose vorstellen. Dabei kann entweder darüber das TCP oder das UDP Protokoll genutzt werden.

Extension Method Time zu Decimal und zurück

wiederum aufbauen auf den letzten Beitrag möchte ich hier ein Snippet vorstellen, mit dem man Zeit in Decimal und Decimal in Zeit umwandeln kann.

Beispiel 12:45 -> 12.75 oder eben 12.75 -> 12:45

[crayon-66291f10cee3e116072509/]

ExtensionMethods – ToInt

Im vorherigen Beitrag habe ich etwas zu der Extension Method ToString erklärt. ToInt() gibt es von Haus aus nicht. Aber das ist nicht schlimm, denn diese können wir uns selber basteln:

[crayon-66291f10cf395339142757/]

Das Ausschlaggebende ist, dass wir als Rückgabewert von der Methode ToInt einen Datentypen haben. Nämlich den Int32 Datentyp.

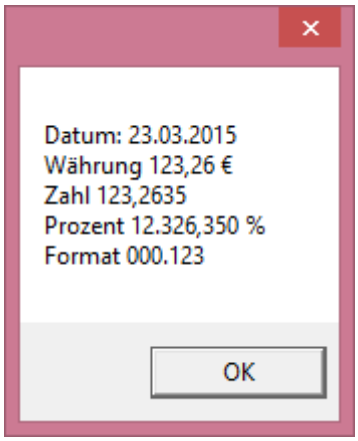
Weiter kann diese Methode jeden Datentyp implementieren, weil diese Methode generischer Natur ist. Leider ist aber auch hier ein try/catch Notwendig, denn wenn wir versuchen einen String “Apfel” in ein Integer zu konvertieren, ist dies nicht möglich und es folgt die Ausnahme.

ToString

Möchte man einen Wert in String konvertieren, gibt es neben Convert.ToString(…) auch einfach .ToString(). Die 2. Möglichkeit ist weit mächtiger als dass diese einfach nur konvertieren kann. Damit lassen sich Datumsformate individuell darstellen, zahlen Runden usw. Mal ein paar Beispiele:

[crayon-66291f10cf509872698098/]

als Ausgabe erhalten wir:



Dies war nur ein Bruchteil an Möglichkeiten. Weitere Formatzeichen und eine größere Erklärung gibt es hier:

http://openbook.rheinwerk-verlag.de/csharp/kap30.htm

Datum nach darauffolgenden Tagen gruppieren

Ehrlich gesagt habe ich etwas gebraucht um den Titel richtig zu wählen. Aber ich möchte einmal aufzeigen, welches Szenario ich meine:

Man hat eine Collection mit folgenden Datumelementen:

31.12.2014
01.01.2015
02.01.2015
16.02.2015
19.02.2015
20.02.2015

und möchte sie so sortieren:

Von – Bis

31.12.2014	02.01.2015
16.02.2015	16.02.2015
19.02.2015	20.02.2015

Als Quelle steht uns eine gefüllte List zur Verfügung:
[crayon-66291f10cf677719965898/]
Das heißt, wenn der nächste Eintrag nicht der darauffolgende Tag ist, wird eine neue Gruppe erstellt.
[crayon-66291f10cf67b998724033/]
Möchte man nun noch die von und bis Werte auswerten, kann man folgendes nutzen:
[crayon-66291f10cf67c993219621/]

Quelle: http://stackoverflow.com/questions/27393626/in-c-what-is-the-best-way-to-group-consecutive-dates-in-a-list

---

ADODB Recordset to Arraylist

Ich weiß, ADODB ist veraltet und sollte nicht verwendet werden. Wer jedoch doch damit arbeiten muss, kann hier mal weiter lesen.

Namespace:

[crayon-66291f10cf824779636850/]
Typen:
[crayon-66291f10cf827739022472/]
Connections:
[crayon-66291f10cf828714520271/]

Methode zu ArrayList
[crayon-66291f10cf829084073829/]

Aktuelles Datum binden

als namespace hinzufügen:
[crayon-66291f10cf9d9719681962/]
Den DateTimePicker erstellen:
[crayon-66291f10cf9dd379586603/]

---

generische Listen / Collection

Zwar braucht die Dictionary<T, K> weniger Zeit beim hinzufügen/ löschen von Werten, braucht die SortedList doch weniger Ramkapazitäten und ist im großen und ganzen schneller. Deutlich langsamer ist die Hashtable und SortedDictionary.

<T> = Typenparameter, erwartet wird ein Datentyp als Parameter. Beispiel string, int, object,…

<V, K> = siehe <T>, jedoch wegen besserer Lesbarkeit stellt dieser Typenparameter den Wert des **V**alue bzw. **K**ey da.

List<T>
[crayon-66291f10cfb76929759754/]
SortedList<V, K>
[crayon-66291f10cfb7a858879551/]
Beinhaltet einen Key und Value. Über den Key lässt sich das Value herausfinden. Beispiel:
[crayon-66291f10cfb7c007793386/]
liefert den Wert Boolean Wert True

Dictionary<V, K>
[crayon-66291f10cfb7d807617220/]
ArrayList (Object)
[crayon-66291f10cfb7e157504379/]
Hashtable (Object K, Object V)
[crayon-66291f10cfb7f960397297/]
ObservableCollection
[crayon-66291f10cfb81346565656/]
Im Gegensatz zur List<T> nutzt die ObservableCollection die INotifyCollectionChanged Schnittstelle. Diese gibt eine Meldung, sobald sich in der Collection etwas geändert hat. Sehr Sinnvoll, wenn man diese Collection an ein Steuerelement per WPF binden möchte, aktualisiert sich das Element so automatisch. Ein Nachteil ist jedoch, dass man die Liste nicht aus der Liste suchen/sortieren kann. Hier kann man nachlesen, wie man dies doch mit einbauen kann -> Link

List<Tuble<T,A,B>

Bisher hatten wir immer nur die Möglichkeit über Key / Value ein Pärchen vom Eintrag zu bilden. Manchmal kommt man aber in die Situation wo man nicht das Key/Value Prinzip haben möchte,

oder mehr als 2 Argumente übergeben möchte. Da kommt das seit .NET 4.0 eingeführte Tuble ins Spiel. Die einzelnen Einträge werden dann als Item1, Item2 usw. innerhalb des Eintrags geführt.
[crayon-66291f10cfb83263671918/]

Quelle: http://blog.bodurov.com/Performance-SortedList-SortedDictionary-Dictionary-Hashtable/

Delegaten und Events

Delegaten sind in C# aufgebaut wie normale Methoden, jedoch besitzen sie keinen Code der ausgeführt wird, sondern weisen lediglich auf eine Methode mit Code hin.

Das bedeutet, dass die Delegaten genau so wie Methoden

einen Zugriffsmodifikator (private, public), einen Rückgabe wert und Parameter haben. Die Parameter müssen aber in der Anzahl und Datentyp den Methoden identisch sein.

[crayon-66291f10cfdad893968747/]

Interessant werden Delegaten in Verbindung mit Events auf Deutsch Ereignisse. Events werden ausgelöst. Wenn man einen Button klickt, löst man das Click-Event aus. In WPF und Windows Forms haben die Steuerelemente z.B. viele vordefinierte Events. So kann man ein Mouseover Event auslösen lassen, wenn man z.B. mit der Maus über ein Steuerelement fährt. Sobald nun ein Event ausgelöst wird, versucht das an Ihm hängende Delegat eine Methode auszulösen mit den

Rückgabewerten und Parameter, die dem Delegat zur Verfügung stehen. Das Interessante ist, dass ein Delegat unabhängig vom Zugriffsmodifikator arbeitet. An beide wird eine Methode durch überladung angehängt ” += ” oder abgezogen ” -= “.

Mal ein kleines Beispiel, wie wir mithilfe eines Delegaten und Event von einem neuen Window in das Mainwindow Label etwas schreiben können:

MainWindow:
[crayon-66291f10cfdb0040605728/]
Das neue Window1:
[crayon-66291f10cfdb2914865045/]
Hier sehen wir, dass zunächst im MainWindow dem Event vom Objekt w1 die Methode wnd_MyCustomEvent hinzugetan wurde. Jedesmal also, wenn das Event ausgeführt wird, wird auch diese Methode ausgeführt.

Im Window1 wird nun die Methode durch das klicken auf den cmd_w1_Button das Event MyEvent ausgelöst und dem dazugehörigen Delegaten wird der Parameter von 100 übergeben.

Abschließend wird nun im MainWindow dieser Wert angezeigt

Der nächste Schritt, den man gehen könnte ist, noch ein neues Window zu erstellen und dort eine Methode hinzufügen:
[crayon-66291f10cfdb3065989704/]
Dem MainWindow fügen wir hinzu:
[crayon-66291f10cfdb4585479418/]
Ausgabe ist eine Messagebox mit dem Wert 100 aus der Klasse in Window2 heraus