TCP / IP – Teil 3: HTTP – Protokoll

Das HTTP (Hypertext Transfer Protocol) baut auf das TCP auf, bildet in seiner Übertragung jedoch weitreichende eigene Konzepte, als das die Übertragung über TCP/IP.

Ebenfalls ist das HTTP ist heute das Standardprotokoll um Webseiten im Webbrowser darzustellen, welches sich laut W3C im 1.1 Standard befindet. HTTP/2 befindet sich aber bereits in Entwicklung.

Eine Ausführung sieht im kurzem immer so aus:

1. Server wartet über einen Port i.d.R. Port 80 auf eine Anfrage

2. Client sendet über POST oder GET eine Anfrage (Request) an den Server

3. Falls dieser Erreichbar ist sendet der Server eine Antwort (Response) zurück

URL

Eine URL kann z.B. so aussehen:
[crayon-677edd5224115823243932/]
http – ist das verwendete Protokoll

www.devandy.de – hostname oder Domain

/meinOrdner/2015-04-01/ – Verzeichnis wo sich die Datei auf dem Server befindet

datei01.php – die Datei an die ein Request gesendet wird

?vorname=peter&nachname=lustig – Querystring, der immer mit einem ? beginnt. vorname= und nachname= bilden dabei die

variablen und peter und lustig sind die darin enthaltenen werte.

Übertragungsmethode GET

Die Übertragung findet über die URI, über den Querystring statt. Die Anweisungen aus der method sind in der URL Sichtbar und können für spätere selbe Zwecke gespeichert werden

Übertragungsmethode POST

Die Übertragung über POST hingegen kann man nicht bookmarken, da die Übertragung im Content geschieht. Dabei werden die Informationen im Header weiter gegeben. Ein Dateiupload ist z.B. nur mit Post möglich, ein Textfeld mit einem Roman macht auch nur über Post Sinn, weil dieser nicht begrenzt ist.

1. Schritt: Server wartet auf Anfrage

In der Regel sind die Serverprogramme wie der IIS Server oder Apache standardisiert eingerichtet, so dass immer über Port 80 gelauscht wird. Man kann dies aber Serverseitig auch auf einen anderen Port umlegen.

2. Schritt: Client Request

Der Client wie der Webbrowser sendet an den Host über das http Protokoll eine Anfrage in Form einer URL.

Dabei enthält diese Anfrage 2 Teile. Zum einen die Anforderungszeile (URL) und zum anderen einen Header, der so aussehen kann:
[crayon-677edd522411e756638384/]
Die erste Zeile beinhaltet mit GET die Methode über den Request, index.html die zu aufrufende Datei und dann die Protokollart und Versionsnummer.

Darunter folgen weitere Informationen wie Ursprungsland, Bilder erlauben, Browsertyp und Versionsnr, Referer (welche Seite man vorher besucht hat), usw.

3. Schritt Server Response

Ist der Server erreichbar, sendet er eine 3 teilige Antwort wieder. Diese besteht aus

1. Statuszeile
[crayon-677edd5224122881282193/]
bestehend aus dem Protokoll, Versionsnummer, Statusnummer und Status.

2. Header

Der Header beinhaltet das Date, Datum und Uhrzeit wann die Antwort geschickt wurde, Server Name und Versionsnummer des Webservers, Content-Length, Länge des Nachrichten-Body in Byte usw.

3. Body

Darin befindet sich der eigentliche HTML Code, der dann im Browser angezeigt wird.

Beispiel Projekt in C#, ein Http Server:

HTTP Server

Quellen:

SelfPHP Get/Post

Andreas Olesch

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

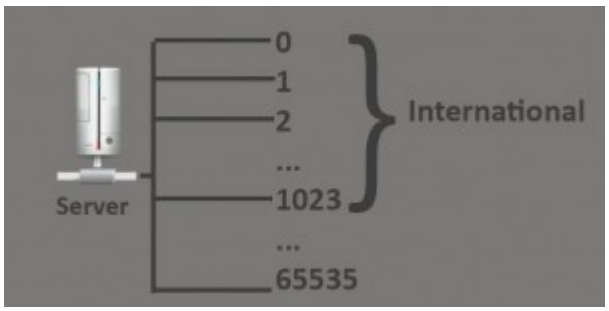
ist genauso wie der Server nur ein Programm, welches jedoch die Dienste empfängt.

Host

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

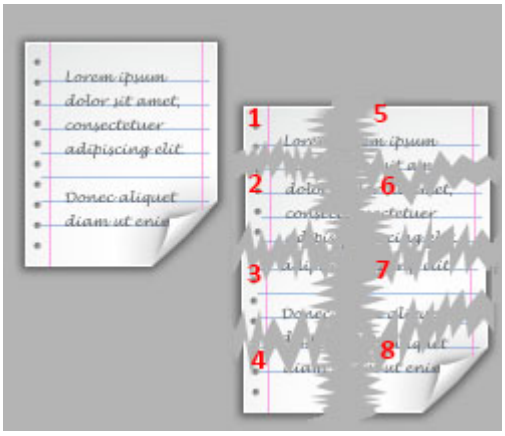
Datenübertragungen stattfinden.

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.

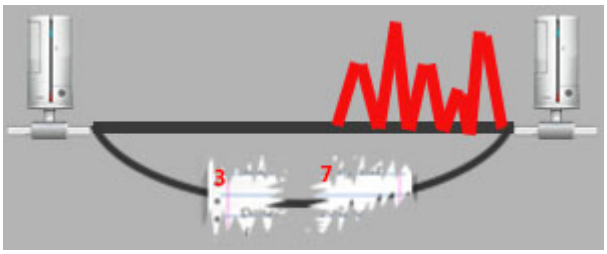
TCP – Transmission Control Protokoll



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.

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-677edd5224663786321217/]

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

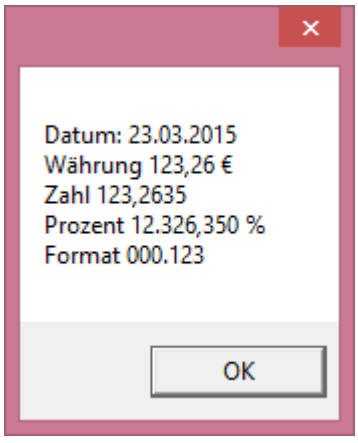
Weiter kann diese Methode jeden Datentyp implementieren, weil diese Methode generischer Natur ist. Leider ist aber auch hier ein try/catch Notwendig, denn wenn wir versuchen einen String „Apfel“ in ein Integer zu konvertieren, ist dies nicht möglich und es folgt die Ausnahme.

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-677edd52247d2104244285/]

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

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-677edd5224953723868759/]
Typen:
[crayon-677edd5224957837953059/]
Connections:
[crayon-677edd5224958196899086/]

Methode zu ArrayList
[crayon-677edd522495a760438716/]

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-677edd5224b0e161706182/]
SortedList<V, K>
[crayon-677edd5224b11173721113/]
Beinhaltet einen Key und Value. Über den Key lässt sich das Value herausfinden. Beispiel:
[crayon-677edd5224b12411934279/]
liefert den Wert Boolean Wert True

Dictionary<V, K>
[crayon-677edd5224b13276622969/]
ArrayList (Object)
[crayon-677edd5224b14488120204/]
Hashtable (Object K, Object V)
[crayon-677edd5224b16308648596/]
ObservableCollection
[crayon-677edd5224b17540280347/]
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-677edd5224b19091156247/]

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-677edd5224ccd255451425/]

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-677edd5224cd0314016144/]
Das neue Window1:
[crayon-677edd5224cd2364490896/]
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-677edd5224cd3742678542/]
Dem MainWindow fügen wir hinzu:
[crayon-677edd5224cd4534016111/]
Ausgabe ist eine Messagebox mit dem Wert 100 aus der Klasse in Window2 heraus

---

generische Klassen

Mit generischen Klassen kann man Datentyp unabhängigen Aufbau einer Klasse erreichen. Das bedeutet, man kann dann ein Objekt der Klasse erzeugen und sagen, dass die Methoden dort eben mit diesen Datentyp arbeiten soll, dem wir dem Objekt übergeben. Auch eine dort deklarierte Variable ist eben der Typ, der übergeben wurde.

Dabei arbeitet man mit einem sogenannten **T**yp-Parameter, welcher ähnlich einem Platzhalter für einen Datentyp stehen soll. Deutlicher wird das vielleicht durch das folgende Beispiel:

[crayon-677edd5224eac232954222/]
Die Klasse erhält einen Typenparameter T. Man könnte auch einen anderen Buchstaben nehmen (i.d.R. ist der erste immer ein T), oder durch Komma getrennt weitere Datentypen anfügen.

Etwas ungewöhnlicher sieht da die Klassen-Eigenschaft in der 2. Zeile aus. Denkt man sich das T weg, könnte dort ein int, string[], double oder sonstwas stehen.

Dann folgt ein Konstruktor, der dieser Eigenschaft nun einen Wert zuweist

und zum Schluss noch eine Methode, die diese Eigenschaft ausgibt.

Contraints

im oberen Beispiel hätte man rein theoretisch die Möglichkeit alle möglichen Datentypen zu setzen. Möchte man dies aber auf eine bestimmte Art von Datentypen beschränken, so gibt es eine where Klausel. Das Muster erinnert dann ein wenig an SQL.

Dann nimmt die Klasse folgende Bezeichnung ein:
[crayon-677edd5224eaf551558909/]

Dabei kann man daraus den Satz machen: „Filtere alle, die von der Elternklasse IComparable erben“. Dabei macht man sich am besten im Objektexplorer schlau, welche Einschränkungen man treffen möchte.

Möchte man ein Objekt erstellen und dieser Datentyp passt nicht unter das Gefilterte, wird bereits zur Entwicklungszeit ein Fehler angezeigt.

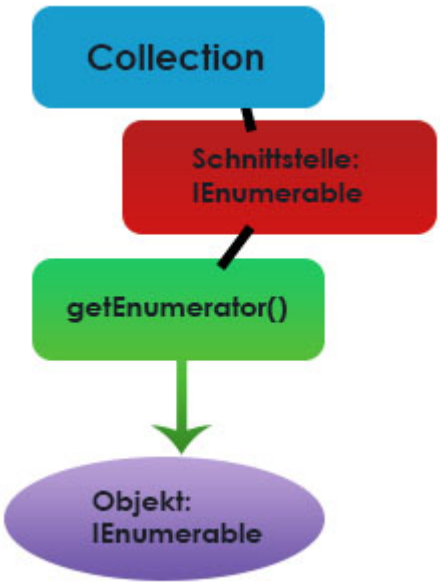
Quelle: http://www.dotnetperls.com/generic

Iteratoren

Das Verb iterieren bedeutet aus einer Kollektion (engl. Collection) nacheinander durch die enthaltenen Elemente durch zulaufen. Nichts anderes tut ja auch die foreach Schleife.

Doch was ist dazu Notwendig, damit eine Iteration stattfinden kann? Die Collection muss die IEnumerable Schnittstelle unterstützen.

IEnumerable besitzt einen Inhaltsverzeichnis (engl. Index) welcher alle Elemente der Liste durchnummeriert beinhaltet. Weiter hat die Schnittstelle eine Methode getEnumerable() um diesen Index auszugeben. Genau dies tut auch die foreach Schleife. Sie ruft die Methode getEnumerable() auf und durchläuft vom ersten bis zum letzten Enumerator.



Glücklicherweise beinhalten die meisten Collections in .NET diese Schnittstelle (Array, List, DataRow usw.)

Möchte man nun, dass eine Klasse nach einem bestimmten Muster iteriert wird, bietet es sich an, eine Metode in dieser Klasse zu erstellen welche vom Typ
[crayon-677edd522500a236182390/]
ist und die einzelnen Collection Elemente neu setzt. Beispiel:
[crayon-677edd522500c344736766/]
[crayon-677edd522500e782296185/]

durch einen Aufruf von
[crayon-677edd522500f002297957/]
wird auch nur die ersten 3 durchiteriert

Quelle: http://www.s-line.de/homepages/trac/wissen/dot-net/csharp-2.html