auteur

Gert Jan Timmerman

C# 3.0

In Visual Studio 2008 is een nieuwe telg aan de C# familie toegevoegd: C# 3.0. In dit artikel worden de nieuwe features van deze taal uiteengezet. Daarbij wordt ervan uitgegaan dat de lezer goed bekend is met C# 2.0, want er wordt alleen aandacht besteed aan de nieuwe taalelementen.

Lambda expressies

In C# 2.0 zijn de anonieme delegates geïntroduceerd. Anonieme delegates hebben als voordeel dat er geen nieuwe functie met een volledige functiedefinitie geschreven hoeft te worden als deze functie slechts eenmalig gebruikt wordt als argument bij het instantiëren van een delegate. De syntax van de anonieme delegates kan in eerste instantie een beetje vreemd aandoen, omdat er een functie-body binnen een ander statement (in dit geval een method-call) gebruikt wordt (zie listing 1). C# 3.0 kent een verbetering van deze anonieme delegates, nl. de lambda expressie.

De syntax van een lambda expressie lijkt meer op de wiskundige notatie van een functie (zie listing 2). Deze eenvoudiger syntax maakt de lambda expressie leesbaarder dan een anonieme delegate. Verder kan een lambda expressie zowel een statement-block bevatten als een enkele expressie. We spreken dan van een statement lambda (zie listing 3) respectievelijk een expression lambda (zie listing 2). Verder zijn er eigenlijk niet veel verschillen met de anonieme delegate. Functioneel gezien biedt de lambda expressie dus niet heel veel nieuws: de leesbaarheid is het belangrijkste voordeel.

```
List<int> l = new List<int>();
// add some elements
l.Find(delegate(int element) { return element > 0; });
```

Listing 1

```
List<int> l = new List<int>();
// add some elements
l.Find(x => x > 0);
```

Listing 2

```
List<int> l = new List<int>();
// add some elements
l.Find(x => { return x > 0; });
```

Listing 3

Extension methods

Extension methods bieden de mogelijkheid als het ware een method toe te voegen aan een bestaande class, zonder de definitie van die class aan te passen. De method wordt dus ook niet echt toegevoegd aan de class, maar kan wel gebruikt worden alsof hij deze bij de class hoort. Ook hier zijn de resulterende syntax en de leesbaarheid het belangrijkste voordeel.

Stel dat we een method willen definiëren die vaststelt of een bepaalde integer een priemgetal is. We kunnen dan een static method schrijven die deze test uitvoert (zie listing 4). Als we deze static method willen gebruiken, moeten we de betreffende integer als eerste parameter meegeven (zie listing 5). Willen we deze method kunnen aanroepen alsof het een instance-method op de struct Int32 is, dan voegen we aan de (eerste) parameter van de method het keyword 'this' toe. Daarmee is de method een extension method geworden en kunnen we deze method gebruiken alsof het een instance method is (zie listing 6). Als de C#-compiler deze method-aanroep compileert, genereert deze een gewone static-method call.

C# 3.0 kent een aantal nieuwe features, maar de meeste hiervan zijn puur syntactisch

Onder water is er dus helemaal geen sprake van een instance-method en er kunnen dus (uiteraard) ook geen private members in de extension method gebruikt worden.

```
public static bool IsPrime(int x)
{
 if (x == 2) return true;
 if (x % 2 == 0 || x < 2) return false;
 for (int i = 3; i <= Math.Sqrt(x); i += 2)
 {
  if (x % i == 0) return false;
 }
 return true;
}
```

Listing 4

```
public static void Main()
{
  Console.WriteLine(IsPrime(49));
}
// result: False
```

Listing 5

```
public static bool IsPrime(this int x)
{
  if (x == 2) return true;
  if (x % 2 == 0 || x < 2) return false;
  for (int i = 3; i <= Math.Sqrt(x); i += 2)
  { if (x % i == 0) return false; }
  return true;
}

public static void Main()
{
  Console.WriteLine(49.IsPrime());
}
// result: False
```

Listing 6

Met behulp van extension methods kunnen ook methods aan interfaces toegevoegd worden (zie listing 7).

```
public static bool IsNull(this IComparable ic)
{
  return ic == null;
}

public static void Main()
{
 Console.WriteLine(49.IsNull());
}
// result: False
```

Listing 7

IEnumerable extension methods

In .NET Framework 3.5 (meegeleverd met Visual Studio 2008) is een aantal extension methods gedefinieerd op de interface IEnumerable. Het gaat onder andere om the methods: Where(), OrderBy() en Select(). Deze methodes leveren allemaal weer een IEnumerable interface als returnwaarde, dus kunnen ze achtereenvolgens op elkaars resultaat aangeroepen worden (zie listing 8). Al deze methodes hebben een delegate (lambda expressie) als argument. Deze delegate wordt op elk element in de collectie (waar de IEnumerable interface doorheen itereert) toegepast. De resultaten vormen de collectie die als returnwaarde teruggegeven wordt (via een IEnumerable interface).

```
int[] inttable = new int[3];
inttable[0] = 5;
inttable[1] = 10;
inttable[2] = -3;

IEnumerable<int> res =
  inttable.Where(x => x > 0).
    OrderBy(x => x % 10).Select(x => x * x);

foreach (int i in res) Console.WriteLine(i);
// result: 100 & 25
```

Listing 8

Delphi Tip

NeverSleepOnMMThreadContention

In Delphi 2007 is er iets extra's bijgekomen in de vorm van een global variabele met de naam NeverSleepOnMMThreadContention. Hiermee kun je aangeven of de huidige thread gaat wachten met "sleep" of in een loop indien de memory manager op dit moment "busy' is in een andere thread. De default optie is om met "sleep" te wachten, maar als je de NeverSleepOn MMThreadContention op True zet dan wordt er in een loop gewacht. Het resultaat is dat je thread eerder bij de memory manager kan komen (maar in de tussentijd je CPU wel meer belast zal worden).

```
NeverSleepOnMMThreadContention := True;
```

Deze methodes werken op basis van 'deferred execution', dat wil zeggen dat alle iterators pas uitgevoerd worden op het moment dat de uiteindelijke iterator (laatst geretourneerde iterator) gebruikt wordt in bijvoorbeeld een foreach lus. Dit principe wordt duidelijk gemaakt in listing 9. Hierin wordt een collectie gecreëerd waarin in eerste instantie één element gestopt wordt. Vervolgens wordt hier de Select() method op losgelaten. Dit levert een nieuwe IEnumerable-implementatie op. Vervolgens wordt er een element aan de oorspronkelijke collectie toegevoegd. Daarna wordt de eerder geretourneerde IEnumerable interface gebruikt in een iteratie. Dit levert uiteindelijk beide elementen op: zowel het element dat al in de collectie zat toen de Select() method uitgevoerd werd, als het element dat daarna toegevoegd werd.

```
List<int> intlist = new List<int>();
intlist.Add(5);

IEnumerable<int> res = intlist.Select(x => x * x);
intlist.Add(10);

foreach (int i in res) Console.WriteLine(i);
// result: 25 & 100
```

Listing 9

Welke nieuwe mogelijkheden deze functionele toevoegingen aan het .NET Framework , in combinatie met de nieuwe syntax van C# 3.0, ons bieden, wordt in het vervolg duidelijk.

Query Comprehension syntax

C# 3.0 kent een alternatieve, op SQL lijkende, syntax waarmee IEnumerable iterators bewerkt kunnen worden door middel van de eerder genoemde extension methods. Listing 10 toont een statement dat gebruik maakt van de extension methods. Uit een array met strings worden eerst de strings gefilterd die groter zijn dan 4 letters, daarna worden ze gesorteerd op alfabet en vervolgens worden de strings geconverteerd naar uppercase. Listing 11 bevat een statement dat hetzelfde doet, maar nu met gebruikmaking van Query Comprehension syntax. Deze syntax wordt door de C#-compiler geconverteerd naar het equivalente statement dat gebruik maakt van de extension methods. Listing 10 en listing 11 leveren dus dezelfde code op.
Dit mechanisme wordt ook wel Language Integrated Query (LINQ) genoemd. Ook bij deze feature van C# 3.0 gaat het weer om een syntactische variatie, niet om een functionele vernieuwing. De nieuwe functionaliteit waarvan hierbij gebruik wordt gemaakt zit in het .NET Framework 3.5.

```
string[] stringtable = new string[3];
stringtable[0] = "abc";
stringtable[1] = "defghi";
stringtable[2] = "dit is een test";

IEnumerable<string> res =
  stringtable.Where(x => x.Length > 4).
    OrderBy(x => x).Select(x => x.ToUpper());

foreach (string s in res) Console.WriteLine(s);
// result: DEFGHI & DIT IS EEN TEST
```

Listing 10

```
string[] stringtable = new string[3];
stringtable[0] = "abc";
stringtable[1] = "defghi";
stringtable[2] = "dit is een test";

IEnumerable<string> res =
  from s in stringtable
  where s.Length > 4
  orderby s
  select s.ToUpper();

foreach (string s in res) Console.WriteLine(s);
// result: DEFGHI & DIT IS EEN TEST
```

Listing 11

Overige nieuwtjes

Tenslotte kent C# 3.0 nog een aantal minder belangrijke nieuwtjes: zo is het mogelijk variabelen te definiëren zonder het datatype op te geven, mits de compiler het datatype kan afleiden uit de expressie die aan de variabele wordt toegekend (zie listing 12). De variabele x in dit voorbeeld is van het type int omdat de expressie die voor de initiële waarde zorgt, van dit type is. Zo is de variabele y in dit voorbeeld van het type double.

```
var x = 5;
var y = 2.3;
Console.WriteLine("{0},{1}", x, y);
// result: 5,2.3
```

Listing 12

'Object Initializers' stellen ons in staat een nieuw gecreëerd object onmiddellijk bij de creatie te initialiseren (zonder dat er een constructor voor gedefinieerd is) door de public properties of fields een waarde te geven (zie listing 13). Op deze manier is het ook mogelijk anonieme classes te definiëren door na het new-statement (zonder typenaam) een aantal properties te initialiseren. Daarmee wordt dan impliciet een nieuw type gedefinieerd (zie listing 14).

```
class Employee
{
  public string Name;
  public DateTime BirthDate;
}

public static void Main()
{
  Employee emp = new Employee
  {
    Name = "Willem",
    BirthDate = new DateTime(1991, 11, 10)
  };
}
Listing 13

var manager =
  new {
    ManagerName = "Jansen",
    ManagerBirthDate = new DateTime(1941, 2, 28)
  };
```

Listing 14

Het definiëren van properties wordt ook eenvoudiger in C# 3.0 door gebruik te maken van 'automatic properties'. Listing 15 toont hoe properties gedefinieerd kunnen worden zonder dat de implementatie door de programmeur geschreven hoeft te worden: de compiler genereert een private member waarin de waarde opgeslagen wordt en de implementatie van de get- en set-method.

```
class Employee
{
  public string Name;
  public DateTime BirthDate
  {
    get;
    set;
  }
}
```

Listing 15

Tenslotte kent C# 3.0 'partial methods'. Partial methods zijn bedoeld om gebruikt te worden in partial classes, waarbij een deel van de class gegenereerd wordt door een tool. Wanneer deze tool een partial method genereert, bepaalt de tool de signature van deze functie en kan de gegenereerde code deze methode aanroepen. Wanneer de ontwikkelaar dat wil, kan hij de betreffende methode in een ander deel van de class implementeren (uiteraard moet de signature overeenkomen). Als dat het geval is, wordt deze methode door de gegenereerde code aangeroepen. Als de ontwikkelaar er echter voor kiest deze methode niet te implementeren, verwijdert de compiler uiteindelijk de method-calls, zodat er geen compile-time of run-time foutmelding ontstaat. Om die reden moet de partial method een returntype void hebben. Listing 16 bevat een voorbeeld hiervan.

```
// gegenereerde code:
partial class TestClass
{
  partial void f(int i);
  public void g(int i)
  {
    f(i);
  }
}

// door developer geschreven code:
partial class TestClass
{
  partial void f(int i)
  {
    Console.WriteLine("Dit is method f");
  }
}
```

Listing 16

Conclusie

C# 3.0 kent een aantal nieuwe features, maar de meeste hiervan zijn puur syntactisch. Dat houdt in dat door deze nieuwe features niet zozeer nieuwe functionaliteit beschikbaar gesteld wordt, maar dat hierdoor de code eenvoudiger leesbaar en daardoor onderhoudbaarder wordt. In combinatie met een aantal nieuwe Framework elementen (extension methods) zijn er hierdoor wel heel krachtige C# statements mogelijk (Query Comprehension syntax), waarmee een SQL-achtige constructie in C# beschikbaar komt. Al met al toch een uitbreiding van C# die niet onderschat moet worden.

Gert Jan Timmerman is werkzaam bij Info Support in de functie Hoofd Kenniscentrum. Hij is gespecialiseerd in Microsoft .NET en Java. Hij heeft in het verleden al vaak lezingen gehouden en artikelen geschreven over C#.