.net column

Generics in 'Whidbey'-versie van .NET

Microsoft heeft de Common Language Runtime (CLR) aangepast om zogenaamde Generics te kunnen ondersteunen. Deze nieuwe versie van de CLR en de language-compilers zullen gereleased worden in de"Whidbey" versie van .NET. Generics is een uitbreiding op het type systeem van de CLR, die het mogelijk maakt om types te definiëren waarvan bepaalde details ongespecificeerd gelaten worden. Deze details worden pas gespecificeerd op het moment dat het generic type gebruikt wordt, zodat er runtime een op maat gemaakt type ontstaat.

De naam 'Generics' geeft precies het doel van deze nieuwe feature aan, namelijk het schrijven van code zonder details te specificeren die de scope of het nut ervan kunnen beperken. Het concept van generics is niet nieuw en mensen die bekend zijn met C++ templates zullen veel overeenkomsten ontdekken.

De voordelen van CLR generics zijn: compile-time type safety, binair hergebruik van code en performance. Aan de hand van een simpel voorbeeld zullen deze aspecten worden verduidelijkt. Als voorbeeld gaan we uit van twee hypothetische collection classes: *HashTable*, een collectie van *object* referenties en *GenericHashTable<T>*, een collectie van alle mogelijke typen.

Type safety: Als een gebruiker een string toevoegt aan de HashTable, vindt er een impliciete conversie plaats naar *object*. Vergelijkbaar is het als een string object uit de HashTable wordt gelezen, dan moet er runtime een conversie plaats vinden van *object* naar een string referentie. Het gebrek aan type safety tijdens het compileren is niet alleen lastig voor de ontwikkelaar, maar verhoogt tevens ook de foutgevoeligheid doordat er runtime conversieproblemen kunnen optreden. Het gebruik van de *GenericHashTable<String>*, waarbij T getypeerd is als string, zorgt ervoor dat alle methoden voor het toevoegen en raadplegen, werken met String referenties. Hierdoor wordt het mogelijk om de types van alle elementen compile-time te controleren in plaats van runtime.

Binary code re-use: Om toch compile-time type safety te verkrijgen zou een ontwikkelaar ervoor kunnen kiezen om van de HashTable klasse een klasse af te leiden, bijvoorbeeld *HashTableOfStrings*. Het probleem van deze aanpak is dat er voor elk type een nieuwe klasse moet worden geschreven, hetgeen veel werk is en veel onderhoud vergt.

Het enige wat gedaan moet worden om de *GenericHashTable<T>* geschikt te maken voor een bepaald type, is het instantiëren ervan met het gewenste type. Doordat de code voor een generic runtime gegenereerd wordt is het mogelijk dat verschillende ongerelateerde element types, zoals bijvoorbeeld *GenericHashTable<String>* en *GenericHashTable<int>*, het grootste deel van de just-in-time (JIT) gecompileerde code kunnen hergebruiken.

Performance: Indien type checking compile-time kan worden gedaan in plaats van runtime, zal de performance worden verhoogd. In managed code vindt er *boxing* plaats als een *value type* '*gecast*' wordt naar een *object* type. Tijdens het boxen wordt een nieuw object gealloceerd, en wordt de waarde van het value type hierin gekopieerd. Als we bijvoorbeeld honderd getallen in de HashTable stoppen, vindt er dus honderd keer een impliciete conversie plaats van een value type naar een reference type, hetgeen negatieve gevolgen zal hebben voor de performance. Door gebruik te maken van de *GenericHashTable<int>* kan voorkomen worden dat er geboxed moet worden, waardoor de performance verhoogd wordt.

Microsoft heeft met de introductie van generics in de CLR een krachtige nieuwe feature toegevoegd aan het .NET platform. Belangrijke voordelen van generics zijn: compile-time type safety, binair hergebruik van code en performance.

Ing. Xander Buffart is werkzaam als IT-architect bij Info Support te Veenendaal (e-mail: xanderb@infosupport.com).