NULOVATELNÉ TYPY POD LUPOU

Aleš Keprt

Katedra informatiky, FEI, VŠB Technická Univerzita Ostrava
17.listopadu 15, 708 00 Ostrava–Poruba
Aley@Keprt.cz

Abstrakt
Nová verze jazyka C# od Microsoftu obsahuje mimo jiné nový programovací prvek zvaný nulovatelné typy. Toto rozšíření umožňuje přiřazovat null do proměnných všech hodnotových typů a je vhodné zejména pro pohodlnou práci s SQL a relačními databázemi. Implementace nulovatelných typů v jazyce C# však skrývá řadu na první pohled ne zcela zřejmých vlastností. Jedná se zejména princip vazby na běžné hodnotové typy formou „lifted conversions“ a zpracování operátorů formou „lifted operators“. Tyto neobvyklé konstrukce vycházejí ze snahy obejít některá nepříjemná omezení ohledně využívání operátorů v jazyce C#. Příspěvek je zaměřen na podrobný popis této problematiky.

1 Úvod

Hodnotové typy (nullable types) jsou jednou z novinek, která se v jazyce C# objevuje v nové verzi 2.0. Pravděpodobně již konečná specifikace tohoto jazyka byla představena v květnu roku 2004 [1], zatímco nové Visual Studio 2005 je v době psaní tohoto textu teprve těsně před zveřejněním verze „beta 2“, která by se měla objevit koncem března.

Tento příspěvek představuje nulovatelné typy ve dvou fázích. V první části jsou srozumitelně představeny běžné konstrukce, s vynecháním detailů, které by mohly ztížit pochopení. Podobné informace je možné najít také v MSDN verze 2005 [9], článku [3] nebo seriálu [2]. Druhá část příspěvku odkrývá další detaily nulovatelných typů, které v běžné literatuře nenajdeme. V závěru jsou ještě zmíněny některé myšlenky a teze, které jsou důsledkem diskuzí odborné veřejnosti nad nulovatelnými typy.

2 Nulovatelné typy

2.1 Úvod

Jak známo, hodnotové datové typy (všechny typy **`struct`**, dále **`int`**, **`char`**, **`double`**, apod.) nepoužívají reference, nelze jim proto přiřadit **`null`**. Většinou to nemá žádný negativní dopad, někdy by však taková nepřiřazená reference mohla být využita k nějakým speciálním účelům. Například při práci s relačními databázemi se použití **`null`** přímo nabízí pro reprezentaci „neuvedených“ (tj. prázdných) hodnot v datech načtených z databáze do C#. Chceme-li však mít takovou neuvedenou hodnotu například u datového typu **`int`**, pak nezbývá, než jako „neuvedenou“ hodnotu použít některé z čísel, které nepotřebujeme. Může to být třeba **`-1`**, ale v případě, že daná proměnná může nabývat všech hodnot z domény příslušného datového typu, nastává obtížný problém. Používání vybraných čísel místo **`null`** navíc komplikuje kód a často vede k celé řadě (lidských) chyb.

Nyní máme v jazyce C# k dispozici tzv. nulovatelné typy. Jsou to hodnotové typy, kterým ovšem můžeme přiřadit i hodnotu **`null`**. Každý nulovatelný typ je „postaven“ nad nějakým

(libovolným) již existujícím hodnotovým typem (označujeme jej potom jako typ bázový – nemá to však nic společného s dědičností) a přidává do jeho domény **null**. Nulovatelnou proměnnou (tj. proměnnou nulovatelného typu) deklarujeme uvedením bázového typu a přidáme otazník. Jak ukazuje následující příklad, i základní použití nulovatelných proměnných je zcela intuitivní.

```
int? x;

Console.WriteLine(x==null ? "null" : x.Value.ToString());

if(x.HasValue) Console.WriteLine("x má hodnotu");
```

Nulovatelné typy mají tyto vlastnosti:

- Jsou vhodné pro proměnné, které mohou obsahovat nedefinovanou hodnotu.
- Syntaxe **T?** pouze zastupuje delší zápis **System.Nullable<T>**. Zápis s otazníkem je tedy ekvivalentní zápisu přes generický typ, vysvětlení generických typů je možno nalézt v [2], [3], [9].
- Vlastnost **HasValue** slouží ke zjištění, zda má daná proměnná hodnotu (pak vrací **true**), nebo je **null** (pak vrací false).
- Vlastnost **Value** zpřístupňuje hodnotu typu **T**.
- Přiřazení hodnoty **default** způsobí nastavení **HasValue = false**. (Slovo **default** je jednou z dalších novinek jazyka C# – vysvětlení viz [2], [3], [9].)

2.2 Přetypování

Přetypování z nulovatelného **T?** na jeho bázový typ (**T**) je možný explicitním přetypováním, převod z bázového typu je dokonce implicitní.

```
int a;
int? x;

x = a;      //implicitní přetypování int → int?

a = (int)x; //explicitní přetypování int? → int
            //vyvolá výjimku, když x bude null
```

2.3 Operátory

Nulovatelné typy mají všechny operátory stejné jako bázový typ. Výsledkem operací, do kterých vstupuje **null** hodnota, je vždy **null** hodnota. Chování operátorů je tedy zcela intuitivní, uvedeme jen několik příkladů.

```
double? a,b;
long? c;

a = b * 2.0;
c++;
if(a > c) {...}
```

Pro pohodlnou práci můžeme použít také nově zavedený operátor **??** (dva otazníky), který umožňuje definovat výchozí hodnotu při přetypování na bázový typ.

```
int? x;

int y = x ?? -1;
```

V této ukázce do proměnné **y** přiřazujeme hodnotu proměnné **x**. Pokud by **x** bylo **null**, pak je do **y** přiřazena hodnota **-1**. Operátor **??** můžeme použít i pro přiřazování mezi nulovatelnými proměnnými.

```
int? z = x ?? -1;
```

Přestože **z** je nulovatelná proměnná, tímto příkazem do ní vložíme vždy nenulovou (not null) hodnotu. (Operátor **??** totiž převede hodnotu na bázový typ **int**, při následném přiřazení do **int?** je pak použit implicitní konverzní operátor.) Když naopak chceme do nulovatelné proměnné vložit **null**, jednoduše jí přiřadíme literál **null**.

```
z = null;
```

3 Další detaily, aneb nulovatelné typy pod lupou

3.1 Přístup k vlastnostem Value a HasValue

Vlastnosti (properties) **Value** a **HasValue** jsou jen ke čtení, nelze jim tedy přiřadit hodnotu. Chceme-li změnit hodnotu **Value**, použijeme přiřazení přímo do proměnné.

```
int? a;
a.Value = 20; //chyba - Value je jen ke čtení
a = 20;       //toto je správně
```

Chceme-li přiřadit hodnotu false do **HasValue**, uděláme to přiřazením **null** do proměnné.

```
a.HasValue = false; //chyba - HasValue je jen ke čtení
a = null;           //toto je správně
```

Hodnotu **true** do **HasValue** samozřejmě přiřadíme jedině tak, že přímo přiřadíme nějakou konkrétní hodnotu do proměnné (tj. jinou než **null**).

3.1 Typové konverze

Princip generického programování zavedený v jazyce C# není dostatečně flexibilní, aby uměl popsat obecné typové konverze nulovatelných typů; tyto konverze tedy mají v jazyce zvláštní postavení. Obecně platí pravidlo, že nulovatelný typ poskytuje všechny typové konverze, které má jeho bázový typ. Navíc je možno kombinovat při převodech libovolné bázové i nulovatelné typy, pokud existují příslušné konverzní operátory pro příslušné bázové typy. Platnost tohoto pravidla je zřejmá již z sekce 2.2, při konverzích se totiž využívají výše zmíněné implicitní a explicitní konverze mezi nulovatelným typem a jeho bázovým typem. V následujících odstavcích si popíšeme všechny možné situace.

(Poznámka: Těmto konverzím se v originální anglické specifikaci jazyka [1] říká „lifted conversions“, čili česky vyzvednuté nebo pozvednuté konverze.)

3.1.1 Převod z hodnotového (nenulovatelného) na nulovatelný typ

Převod z hodnotového (nenulovatelného) na nulovatelný typ probíhá s využitím implicitní konverze z bázového typu na jeho nulovatelný typ. Tato konverze je tedy také implicitní, pokud je konverze mezi bázovými typy implicitní, jinak je explicitní.

```
int? a = 'A';         //implicitní konverze char → int?

char? b = (char?)65; //explicitní konverze int → char?

char? b = (char)65;  //explicitní konverze int → char
                     //potom implicitní char → char?
```

3.1.2 Převod z nulovatelného typu na nulovatelný typ

Převod z nulovatelného typu na (jiný) nulovatelný typ probíhá tak, že hodnota null je zachována a pro ostatní hodnoty je použit příslušný konverzní operátor bázového typu. Tato konverze je tedy implicitní, pokud je konverze mezi bázovými typy implicitní, jinak je explicitní.

```
int? a = (char?)'A';       //implicitní konverze char? → int?

char? b = (char?)(int?)65; //explicitní konverze int? → char?
```

3.1.3 Převod z nulovatelného typu na hodnotový typ

Převod z nulovatelného typu na (jiný než jeho vlastní bázový) hodnotový typ vyvolá výjimku v případě, že výchozí hodnota je **`null`**. V ostatních případech je použita explicitní konverze na bázový typ a potom příslušný konverzní operátor bázového typu. Tato konverze je tedy vždy explicitní (tj. je třeba toto přetypování vždy explicitně uvést, není nikdy provedeno automaticky). Je-li konverzní operátor bázového typu explicitní, dochází tak de facto ke dvojí explicitní konverzi. Překladač však tuto konverzi považuje za přímou explicitní konverzi – první řádek z následujícího příkladu tak ukazuje zbytečně složitou dvojí konverzi.

```
char a = (char)(int)(int?)65; //explicitní konverze int? → int
                              //potom druhá explicitní int → char

char a = (char)(int?)65; //přímá explicitní konverze int? → char
```

3.2 Jak fungují operátory

Stejně jako přetypování, i operátory jsou nulovatelným typům zavedeny pomocí explicitní a implicitní konverze mezi nulovatelným a jeho bázovým typem (viz sekci 2.2). (Specifikace jazyka [1] pak hovoří o „lifted operators“, čili česky o vyzvednutých nebo pozvednutých operátorech.)

3.2.1 Unární operace s nulovatelným typem

Definuje-li bázový typ nějaký unární operátor, pak nulovatelný typ nad ním má tento operátor také. Je-li vstupní hodnotou **`null`**, pak výsledkem je také **`null`**. V opačném případě je hodnota konvertována na bázový typ, na něm je provedena operace a výsledek je konvertován zpět na nulovatelný typ. Výsledkem je pak tedy nenulová hodnota nulovatelného typu.

(Paradoxně se zde tedy objevuje implicitní použití explicitní konverze na bázový typ – je to však bezpečné, neboť k tomu dojde jen při nenulové hodnotě vstupu.)

```
int? a = 20;
a++;          //pozvednutý unární operátor ++
```

3.2.2 Binární operace s nulovatelným typem

Binární operace mají dvě vstupní hodnoty, může zde tedy nastat více situací. Je-li alespoň jedna ze vstupních hodnot **`null`**, potom výsledkem operace je také **`null`**. Není-li ani jedna ze vstupních hodnot **`null`**, pak jsou obě přetypovány na své bázové typy a je použit příslušný operátor bázového typu. Výsledek je pak převeden na příslušný nulovatelný typ. (Opět je zde tedy implicitní použití explicitní typové konverze a opět je bezpečné.)

3.2.3 Test na null

Jak již bylo řečeno, test na **`null`** provádíme pomocí vlastnosti **`HasValue`**. Můžeme k tomu však použít i speciální tvar operátorů **`==`** a **`!=`**, jak ukazuje následující příklad.

```
int? a;
if(a.HasValue==false) { /* je to null */ }
if(a==null) { /* je to null */ }
```

Obě tyto konstrukce jsou ekvivalentní, ta druhá je zřejmě více intuitivní.

3.3 Trojhodnotový typ `bool?`

Specifické postavení mezi nulovatelnými typy má **`bool?`**. Jak ukazuje tabulka 1, operátory logického součtu **`|`** a logického součinu **`&`** dávají u tohoto typu v některých případech nenulové výsledky, i když jeden z operandů je **`null`**. Toto chování odpovídá klasické (matematické) trojstavové logice, potažmo chování trojstavové logiky v jazyce SQL a je specifické právě pro typ **`bool?`** – u všech ostatních nulovatelných typů platí, že výsledkem binárních operací, kde jeden (nebo i oba) operandy jsou **`null`**, je vždy hodnota **`null`**.

Tabulka 1.
Tabulka pravdivostních hodnot typu **`bool?`** pro operátory **`&`** a **`|`**.

& (and)	null	false	true
null	null	false	null
false	false	false	false
true	null	false	true

| (or)	null	false	true
null	null	null	true
false	null	false	true
true	true	true	true

3.4 Nulovatelné typy vyšších řádů

Přímo z definice nulovatelných typů (viz 2.1) je zřejmé, že každý nulovatelný typ je sám také typem hodnotovým a tudíž může být použit jako bázový typ pro další nulovatelný typ. V praxi to znamená, že C# umožňuje tzv. „zřetězení otazníků“, jak ukazuje další příklad.

```
int?? a = null;
```

Tato konstrukce tedy vytvoří nulovatelnou proměnnou typu **int??**, což je nulovatelný typ nad bázovým typem **int?** (a **int?** je nulovatelný typ nad **int**). Tato konstrukce nápadně připomíná ukazatele vyšších řádů a principiálně také funguje obdobně.[1]

```
unsafe {
  int** p = null;
}
```

Jeden zásadní rozdíl oproti ukazatelům vyšších řádů tu však je a týká se právě přiřazování **null** použitém ve výše uvedených příkladech. U ukazatelů se totiž rozlišuje „ukazatel roven null“ a „ukazatel ukazující na proměnnou rovnou null“. U nulovatelných typů se však „nulovatelná proměnná s hodnotou null“ a „nulovatelná proměnná s hodnotou nulovatelného typu, jejíž hodnota je null“ nerozlišuje. Tento rozdíl je možno vyzkoušet například takto.

```
//toto vypíše true
unsafe {
  int* p1 = null;
  int** p2 = &p1;
  System.Console.WriteLine("p2 má hodnotu? {0}", p2!=null);
}

//toto vypíše false
int? v1 = null;
int?? v2 = v1;
System.Console.WriteLine("v2 má hodnotu? {0}", v2.HasValue);
```

Přestože na předposledním řádku přiřazujeme „nějakou hodnotu“ typu **int?**, což jistě není totéž jako přiřazovat literál **null**, proměnná **v2** se nastaví na hodnotu **null**.

4 Další myšlenky

Nad nulovatelnými typy se strhla bouřlivá diskuze na internetu. Na jedné straně totiž zjednodušují kód, zejména práci s relačními databázemi, na druhé straně však přinášejí několik sporných bodů:

1. Jednoduchá deklarace ve tvaru **int?** se každému nemusí nelíbí. Především kvůli tomu, že jde jen o syntaktický cukr, se programátoři rozdělili na dva proti sobě stojící tábory. Zástupci prvního tábora preferují jazyk C a odvozené jazyky pro jejich stručný zápis mnoha konstrukcí (viz operátor **?:**), druhý tábor to naopak považuje za zbytečné zkracování deklarací a prvek znepřehledňující kód.
2. Nulovatelné typy můžeme chápat také jako předkrok k zavedení podpory proměnných ve formě jednoduchých regulárních výrazů. Výraz **int?** totiž můžeme chápat jako „žádný nebo jeden **int**“. Potom **int!** by odpovídalo **int** a znamenalo by „právě jeden **int**“. Dále **int+** by znamenalo „jeden nebo více **int**ů“ a **int*** by znamenalo „libovolný počet **int**ů“. Tyto konstrukce by de facto deklarovaly kolekce a umožnily by pohodlnější práci s daty. Na druhé straně by však přinesly (další) zesložitění jazyka a je možné, že někteří programátoři by je ani nemuseli pochopit. Navíc pak zejména konstrukce **int*** je pro svou zaměnitelnost se zcela odlišným prvkem jazyka

[1] Poznámka: Jazyk C# podporuje ukazatele stejně jako C++, pouze musíme části kódu, kde ukazatele používáme, uzavřít do bloku unsafe.

(s ukazateli) mnoha lidem vyloženě trnem v oku. Podrobněji je tato problematika rozebrána v dokumentu z Microsoft Research [4].

3. Zatímco zde byl vyřešen problém nulovatelných typů, ticho zůstalo v otázce „nenulovatelných“ typů, jak je známe z C++. Jazyk C# totiž dodnes neobsahuje konstrukci, jak zapsat kód ekvivalentní referencím jazyka C++, tedy referencím, do kterých nelze přiřadit **`null`**. Zatímco v C++ to lze obejít přetypováním přes ukazatel, v C# by to díky silnější typové kontrole mohlo fungovat ještě lépe. Jak ukazují některé studie, přidáním tohoto prvku by se mohla zvýšit bezpečnost i jednoduchost kódu, neboť na většině míst opravdu **`null`** hodnoty (místo skutečných objektů) vůbec nechceme. Pro podrobnější diskuzi viz [5].

5 Závěr

Nulovatelné typy jsou novým prvkem jazyka C#. Ačkoliv jejich použití se v praxi možná omezí jen na práci s relačními databázemi, principiálně jde o velmi zajímavý prvek, který nabízí jistý zobecněný pohled na chápání hodnotových datových typů. Při hlubším zkoumání nulovatelných typů navíc vidíme i prostor k debatám nad dalšími možnými zobecněními, což můžeme pro výzkum v oblasti programovacích jazyků považovat za velmi přínosné.

Další informace je možno hledat ve všech publikacích zmíněných v textu a také na stránkách MSDN [6], MSDN Lab. [7] nebo MSDN Magazine [8].

Literatura

1. Hejlsberg, A., Golde, P. a Katzenberger, S. *C# Version 2.0 Specification.* Microsoft 2004. http://msdn.microsoft.com/vcsharp/team/language/
2. Keprt, A. Horké novinky v jazyce C# 2.0 (1.část). V časopise *ComputerWorld.* Ročník 15, číslo 42/2004. ISSN 1210-9924.
3. Keprt, A. Nové prvky jazyka Visual C# 2.0 (2005). Ve sborníku konference *Objekty 2004*. Ed. David Ježek, Vojtech Merunka, VŠB Technická Univerzita, Ostrava, 2004, pp. 15–32, ISBN 80-248-0672-X.
4. Meijer, E. a Schulte, W. *Unifying Tables, Objects and Documents.* Microsoft Research. http://research.microsoft.com/~emeijer/Papers/XS.pdf
5. Najmabadi, C. *Who Wants Non-Nullable Types?* In MSDN Blogs. http://blogs.msdn.com/cyrusn/archive/2004/06/03/147778.aspx
6. MSDN – centrum nápovědy a dokumentace http://msdn.microsoft.com/
7. MSDN Lab. – laboratoř pro beta verze produktů apod. http://lab.msdn.microsoft.com/
8. MSDN Magazine – časopis serveru MSDN http://msdn.microsoft.com/msdnmag/
9. Visual Studio 2005 Beta Documentation http://msdn2.microsoft.com/