Zpravodaj moderníProgramování 1/2015: Skutečné typové konverze

Obtížnost: středně pokročilí

Tento a příští Zpravodaj budou zaměřeny na typové konverze. Typové konverze provádíme v situacích, kdy chceme s nějakou hodnotou pracovat v jiném datovém typu, než v jakém ji máme k dispozici. Charakteristickým příkladem je převod řetězce zadaného uživatelem na celé číslo voláním `Convert.ToInt32`.

Problematika typových konverzí je ovšem rozsáhlejší nežli ono zmíněné volání a v učebnicích *Moderní programování* je pokryta na různých místech. Vzhledem k její důležitosti jsem usoudil, že bude přínosné vše shrnout na místě jednom, zde ve Zpravodaji.

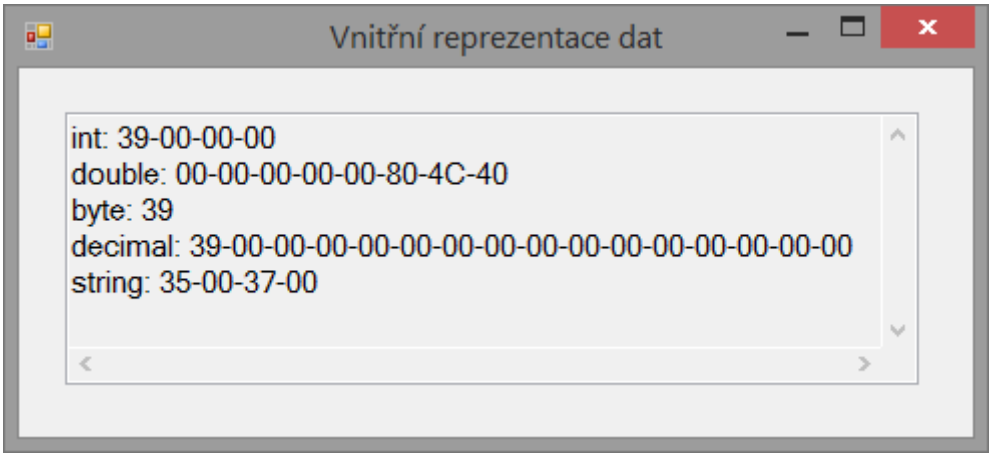
Aby to nebylo tak jednoduché, existují dva druhy typových konverzí:

- ***Skutečné typové konverze***, při kterých **dochází k transformaci** (přepočtu) konvertované hodnoty do vnitřní reprezentace jiného typu. Ty jsou vcelku názorné a nedělají problémy s porozuměním.
- ***Formální typové konverze***, při kterých se konvertovaná hodnota, tak jak je uložena v operační paměti, **nijak nemění** a které vždy souvisejí s potřebou jednotným způsobem zpracovat data různých typů. Tyto konverze jsou abstraktnější a déle trvá, než si na ně člověk zvykne, a především kvůli nim se na tomto místě tématice typových konverzí věnuji.

Dnešní Zpravodaj pokryje názornější konverze skutečné, Zpravodaj příští se bude věnovat obtížnějším konverzím formálním.

Vnitřní reprezentace dat

Data jsou v počítači uložena způsobem jedinečným pro každý datový typ. Pro názornou ilustraci jsem udělal prográmek, který zobrazuje vnitřní reprezentaci padesát sedmičky v několika datových typech. Výsledky jsou zobrazeny jako bajtové řetězce v šestnáctkové soustavě.



Program

Program vypadá následovně:

```
private void Form1_Load(object sender, EventArgs e)
{
    // Hodnoty, o které se zajímáme
    int celéČíslo = 57;
    double desetinnéČíslo = 57;
    byte bajtovéCeléČíslo = 57;
    decimal čísloDecimal = 57;
    string text = "57";

    // Základní číselné typy
    poleVýsledky.Clear();
    poleVýsledky.Text += "int: " +
        BitConverter.ToString(BitConverter.GetBytes(celéČíslo)) +
        Environment.NewLine;
    poleVýsledky.Text += "double: " +
        BitConverter.ToString(BitConverter.GetBytes(desetinnéČíslo)) +
        Environment.NewLine;
    poleVýsledky.Text += "byte: " +
        BitConverter.ToString(new byte[]{ bajtovéCeléČíslo }) +
        Environment.NewLine;

    // Decimal
    int[] poleIntů = decimal.GetBits(čísloDecimal);
    string výsledekDecimal = poleIntů.
        Select(jedenInt =>
               BitConverter.ToString(BitConverter.GetBytes(jedenInt))).
        Aggregate((mezivýsledek, hodnota) => mezivýsledek + "-" + hodnota);
    poleVýsledky.Text += "decimal: " +
        výsledekDecimal +
        Environment.NewLine;

    // String
    char[] poleZnaků = text.ToCharArray();
    string výsledekString = poleZnaků.
        Select(znak => BitConverter.ToString(BitConverter.GetBytes(znak))).
        Aggregate((mezivýsledek, hodnota) => mezivýsledek + "-" + hodnota);
    poleVýsledky.Text += "string: " +
        výsledekString +
        Environment.NewLine;
}
```

Vnitřní reprezentaci dat lze zjistit vícero způsoby, já jsem zde především využil třídu `BitConverter`. Ta má jednak metodu `GetBytes`, která vybrané datové typy umí převést na pole bytů, jednak metodu `ToString`, která pole bytů zobrazí jako řetězec v šestnáctkové soustavě.

S typy `decimal` a `string` je to složitější. U decimalu si musíme pomoct převodem na pole čtyř intů (volání `GetBits`), u `stringu` převodem na pole znaků. Abych si ušetřil nějaké cykly `foreach` pro procházení polí, použil jsem LINQ metody `Select` a `Aggregate`, to ale není až tak zásadní.

Jak provádět typové konverze

Převod hodnoty z jednoho typu na druhý lze provádět těmito způsoby:

- Implicitní, neboli tichou konverzí;
- Unárním operátorem typové konverze;
- Voláním vhodné metody.

Pojďme si jednotlivé možnosti postupně probrat.

Implicitní typová konverze

Implicitní, též automatická nebo tichá typová konverze je ta, která probíhá bez našeho přičinění, aniž bychom museli cokoli psát. Probíhá tehdy, když překladač usoudí, že se nic nemůže stát, což je v zásadě tehdy, když přiřazujeme „menší typ“ do „většího“, např. byte do `intu`, `int` do `doublu` či `float` do `doublu`.

Příklady:

```
// byte s intem
byte byte57 = 57;
int int57 = byte57;
// byte57 = int57; nelze!

// int s doublem
double double57 = int57;
// int57 = double57; nelze!

// float s doublem
float float57 = 57;
double57 = float57;
// float57 = double57; nelze!

// double s decimalem
decimal decimal57 = 57;
// double57 = decimal57; nelze!
// decimal57 = double57; nelze!
```

Např. bytová hodnota `byte57` se bez problémů převede na typ `int`, aby ji bylo možno přiřadit do proměnné `int57`.

Opačným směrem by mohlo dojít ke ztrátě části dat anebo minimálně ke ztrátě přesnosti, takže tu konverzi překladač sám neprovede.

Všimněte si mj. vztahu mezi `doublem` a `decimalem` - ani jeden z nich není „větší“ (tj. nadmnožinou) než ten druhý, proto ani jedním směrem nefunguje implicitní konverze.

Operátor typové konverze

Konverzi si můžeme výslovně vyžádat pomocí ***operátoru typové konverze***, který se zapisuje uvedením **názvu typu v závorkách**:

```
// Použití operátoru typové konverze
byte57 = (byte)int57;
int57 = (int)double57;
float57 = (float)double57;
double57 = (double)decimal57;
decimal57 = (decimal)double57;
```

Tady jsme si to vyžádali my, my sami tudíž neseme riziko, že se provede něco nesprávného. Pokud víme, co děláme, je to v pořádku. Pokud nevíme, můžeme ztratit data. Například při konverzi hodnoty `double` na hodnotu `int` dojde k odřezání desetinné části čísla.

Ještě horší případ nastane, když se hodnota „nevejde“ do cílového typu. Zkuste např. `int57` nastavit na 557, přiřadit ho do `byte57` a zobrazit výslednou hodnotu proměnné `byte57`. Nebudete se stačit divit!

Volání konverzní metody

Poslední případ asi znáte nejlépe, metody třídy `Convert` nebo metoda `ToString` (anebo také metody třídy `BitConverter` ze začátku tohoto Zpravodaje):

```
// Volání konverzní metody
string string57 = int57.ToString();
int57 = Convert.ToInt32(string57);
```

Za zvláštní zmínku zde stojí, že `Convert.ToInt32` lze zavolat nejen s obvyklým parametrem typu `string`, ale také rovněž s parametrem typu `double` - tehdy dojde k matematickému zaokrouhlení čísla předaného jako parametr.

Za zmínku stojí také to, že při převodech mezi čísly a řetězci nelze použít operátor typové konverze:

```
// string57 = (string)int57; nelze!
// int57 = (int)string57; nelze!
```

Převod mezi řetězcem a číslem je považován za natolik zásadní transformaci dat, že jej lze provést pouze specializovanými metodami, nikoli pouhým použitím operátoru.

Kombinace

V některých situacích přístupy kombinujete. Např. zaokrouhlení desetinného čísla vždy nahoru:

```
// Kombinace
double double57_2 = 57.2;
int zaokrouhlenoNahoru = (int)Math.Ceiling(double57_2);
```

`Math.Ceiling` hledá nejbližší vyšší nebo rovné celé číslo, vrací ho ale v typu double, takže potřebujeme ještě výslovný převod na `int`.

Závěr

Ukázali jsme si různé konverze hodnot mezi datovými typy. Jednalo se vždy o konverze skutečné, při nichž docházelo k fyzické změně (přepočtu, transformaci) dat. Příště si povíme o abstraktnějších konverzích formálních.

Radek Vystavěl, 1. ledna 2015

Pokud Vám Zpravodaje moderníProgramování připadají užitečné, doporučte jejich odběr svým známým. Mohou se přihlásit na webu www.moderniProgramovani.cz.