Zpravodaj moderníProgramování 07/2012: Kešování v LINQ

Obtížnost: pokročilí

Jak často instancovat

Veškerá práce s daty v LINQ to SQL se provádí prostřednictvím instance řídicí třídy LINQ, která je odvozená od třídy `DataContext`. V té souvislosti vzniká otázka, jak často tuto třídu instancovat - jednou při startu programu (a držet ji), nebo pokaždé znovu, když něco potřebujeme?

Odpověď najdeme na webu MDSN po vyhledání „DataContext class". Na stránce http://msdn.microsoft.com/en-us/library/system.data.linq.datacontext.aspx se v části „Remarks" píše

> *In general, a DataContext instance is designed to last for one "unit of work" however your application defines that term. A DataContext is lightweight and is not expensive to create. A typical LINQ to SQL application creates DataContext instances at method scope or as a member of short-lived classes that represent a logical set of related database operations.*

neboli

> *Třída DataContext je navržena tak, aby se instancovala na dobu jedné „jednotky práce", ať už to ve vaší aplikaci znamená cokoli. Objekt řídicí třídy LINQ je jednoduchý, nenáročný na vytvoření. Typická aplikace využívající LINQ to SQL instancuje řídicí třídu na úrovni metody, případně jako člen krátce žijícího objektu provádějícího množinu příbuzných databázových operací.*

Jinými slovy, doporučení Microsoftu zní vytvářet novou instanci řídicí třídy LINQ na každé čtení, na každý zápis, eventuálně ji chvíli ponechat, pokud máme několik za sebou jdoucích souvisejících operací.

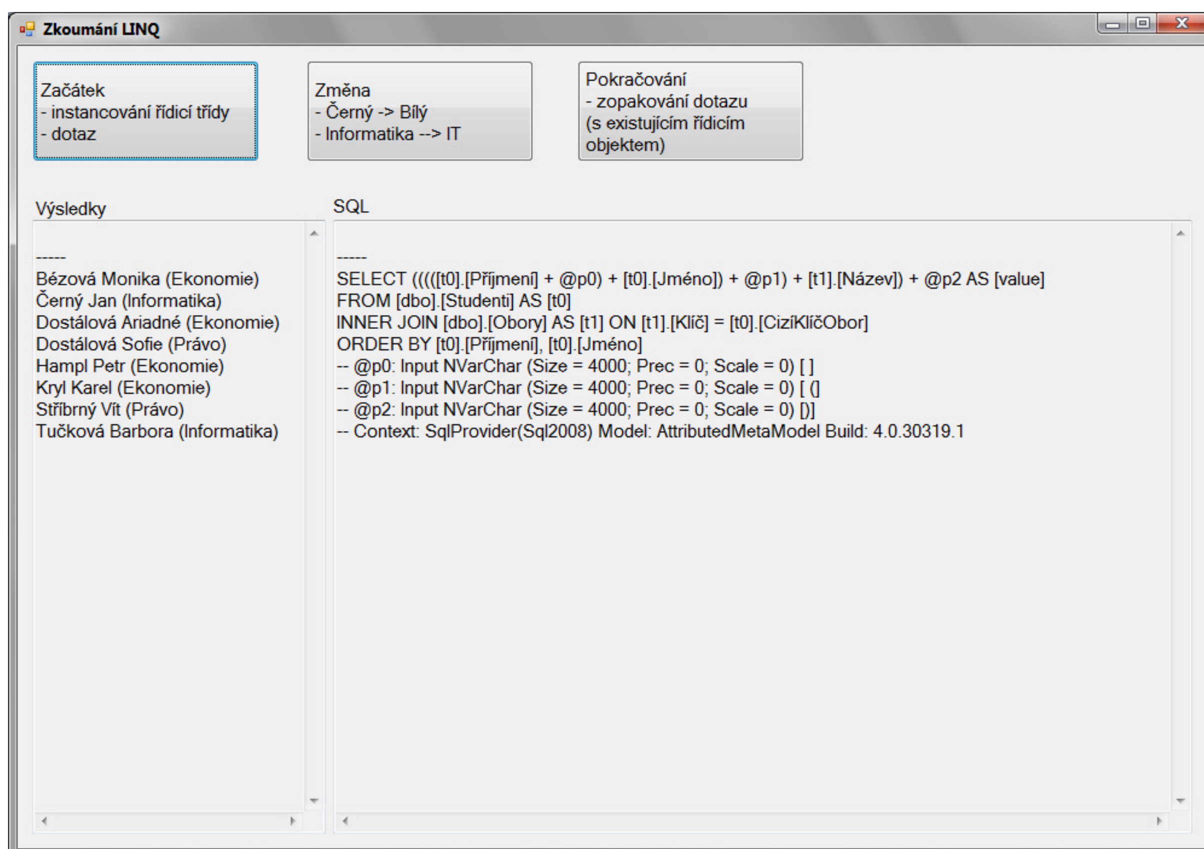
Tolik odpověď na důležitou otázku. Pokud vás zajímají příčiny a souvislosti, čtěte dál.

Proč

V odpovědi z dokumentace mne dlouho znejišťovala nejednoznačnost té „jednotky práce". Jaktože si ji mohu definovat jakkoli? Co když si instanci řídicí třídy podržím déle, bude to něčemu vadit? V následujícím textu nastíním, k jakému názoru jsem se dopracoval. Současně při tom prozkoumáme některá vnitřní fungování LINQ.

Příprava

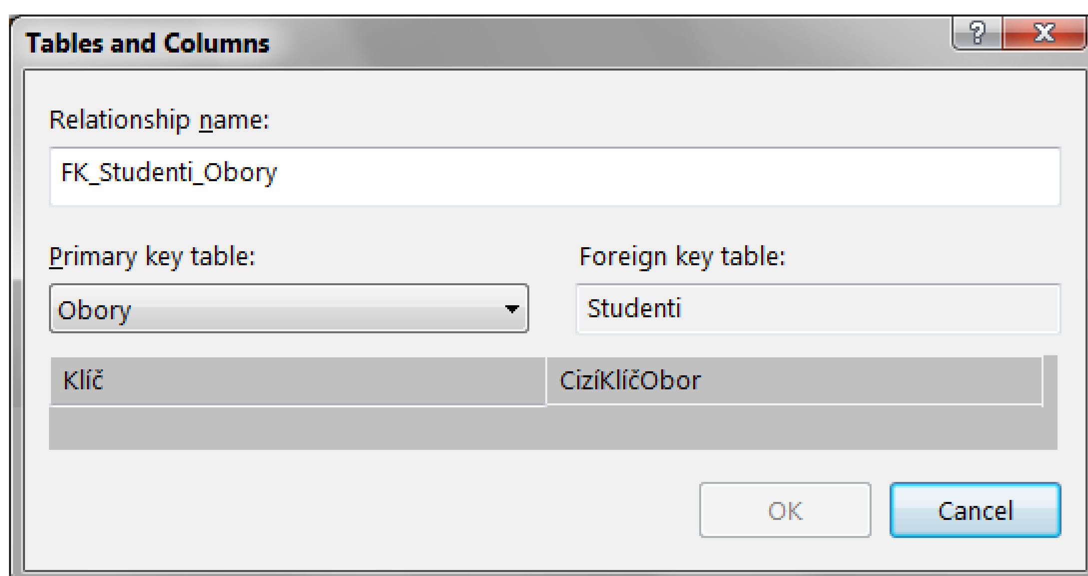
Detaily LINQ to SQL budeme zkoumat na programu, který zobrazí seřazený seznam studentů z databáze.



SŘBD použijeme jako obvykle MS SQL Express Edition, v databázi budou tabulky studentů a oborů, navzájem propojené omezením cizího klíče. Hodnoty primárních klíčů obou tabulek bude automaticky přidělovat SŘBD. Pro přípravu LINQ to SQL použijeme, jak jinak, O/R Návrháře.

	Column Name	Data Type	Allow Nulls
🔑	Klíč	int	☐
	Jméno	nvarchar(50)	☐
	Příjmení	nvarchar(50)	☐
	CizíKlíčObor	int	☐

	Column Name	Data Type	Allow Nulls
🔑	Klíč	int	☐
	Název	nvarchar(50)	☐



Ukázková data:

Obory: Query(m...E_2012_07.MDF)

Klíč	Název
1	Informatika
2	Ekonomie
3	Právo

Studenti: Query...ZE_2012_07.MDF)

Klíč	Jméno	Příjmení	CizíKlíčObor
2	Monika	Bézová	2
3	Sofie	Dostálová	3
4	Petr	Hampl	2
5	Karel	Kryl	2
6	Vít	Stříbrný	3
7	Barbora	Tučková	1
8	Ariadné	Dostálová	2
9	Jan	Černý	1

Pro ukázky bude důležité, abychom v tabulce oborů měli obor Informatika a v tabulce studentů Černého.

Všechno máte samozřejmě přiloženo. Otevřete-li si hotový projekt, nezapomeňte si změnit cestu k databázi v souboru `app.config`.

Co budeme zkoumat

Chci provést následující pokus:

1. Prvním tlačítkem provedeme dotaz do DB a vypíšeme studenty;
2. Nějakého studenta a nějaký obor **změníme bez použití řídicího objektu z bodu 1** (tj. buď ručně ve vývojovém prostředí, nebo použitím jiné instance řídicí třídy);
3. Provedeme stejný dotaz jako v bodě 1 a použijeme stejný řídicí objekt jako v bodě 1.

Cílem je zjistit, zda se externí změna provedená v bodě 2 promítne v bodě 3.

Struktura programu

Program bude v jádru vypadat takto:

```
public partial class oknoProgramu : Form
{
    // Proměnná pro instanci řídicí třídy LINQ, kterou budeme
    //   v programu opakovaně používat
    LINQDataContext linq;

    private void Dotaz()
    // Provede dotaz na studenty,
    //   využívá existující instanci řídicí třídy LINQ
    {
    }

    private void ZměňPříjmeníStudenta(string původní, string nové)
    // Změní příjmení prvního studenta,
    //   kterého najde dle hodnoty parametru původní
    {
    }

    private void ZměňNázevOboru(string původní, string nový)
    // Změní název prvního oboru,
    //   který najde dle hodnoty parametru původní
    {
    }

    private void tlačítkoZačátek_Click(object sender, EventArgs e)
    // Instancuje řídicí třídu LINQ a provede dotaz
    {
        linq = new LINQDataContext();
        Dotaz();
    }

    private void tlačítkoPokračování_Click(object sender, EventArgs e)
    // Provede dotaz, řídicí třídu LINQ znovu neinstancuje
    {
        Dotaz();
    }
```

```
    private void tlačítkoZměna_Click(object sender, EventArgs e)
    // Změní jednoho studenta a název jednoho oboru
    {
        ZměňPříjmeníStudenta("Černý", "Bílý");
        ZměňNázevOboru("Informatika", "IT");
    }

    private void oknoProgramu_Load(object sender, EventArgs e)
    // Uvede případné změny z předchozího běhu programu do pův. stavu
    {
        ZměňPříjmeníStudenta("Bílý", "Černý");
        ZměňNázevOboru("IT", "Informatika");
    }
}
```

Detaily

Metoda `Dotaz` zapíše výsledky do textového `poleVýsledky` a SQL příkazy, na které se LINQ zápis přeložil, do textového `poleSQL`.

```
private string Oddělovač()
{
   return Environment.NewLine + "-----" + Environment.NewLine;
}
private void Dotaz()
{
  linq.Log = new StringWriter();
  poleVýsledky.Text += Oddělovač() +
    linq.Studenti.
    OrderBy(st => st.Příjmení).ThenBy(st => st.Jméno).
    Select(st => st.Příjmení + " " + st.Jméno +
                 " (" + st.Obory.Název + ")").
    ToArray().
    Aggregate((mezivýsledek, prvek) =>
                 mezivýsledek + Environment.NewLine + prvek);
   poleSQL.Text += Oddělovač() + linq.Log.ToString();
}
```

- `OrderBy`, `ThenBy` - řazení podle abecedy;
- `Select` - převádí celý záznam na zobrazovaný řetězec;
- `Aggregate` - z celé sekvence udělá jediný textový řetězec; musí mu předcházet `ToArray`, neboť `Aggregate` nelze provést na SQL Serveru, je třeba provést lokálně v aplikaci;

Změna je přímočará (všimněte si použití jiného řídicího objektu!):

```
private void ZměňPříjmeníStudenta(string původní, string nové)
{
   var jinéLinq = new LINQDataContext();
   Studenti student = jinéLinq.Studenti.
                      FirstOrDefault(st => st.Příjmení == původní);
   if (student != null)
       student.Příjmení = nové;
   jinéLinq.SubmitChanges();
}
```

Výsledky

Program můžeme spustit. Postupným stiskem všech tří tlačítek zjistíme, že Černý se změnil na Bílého a Informatika na IT. Pro výpis byl dvakrát použit SQL příkaz Select se spojením obou tabulek. Vše funguje, jak bychom asi čekali.

Nyní provedeme v programu zdánlivě drobnou změnu. V metodě `Dotaz` vyměníme pořadí LINQ metod `Select` a `ToArray` (stačí prostě prohodit oba řádky kódu).

Co to bude znamenat? Zpracování na SQL Serveru skončí v okamžiku volání `ToArray`. Z SQL Serveru se tedy vrátí sekvence studentů, LINQ metoda Select je zpracována již lokálně. Jelikož ale zpracování vyžaduje znalost názvů oborů (nestačí pouze hodnota cizího klíče), musí proběhnout další dotazy na SQL Server. Přesně to je vidět v překladu na SQL, který se zobrazuje v pravém poli.

To ale není všechno, co uvidíte po spuštění pozměněného programu. Zajímavé tam je dále:

- Při druhém volání metody `Dotaz` (po provedení změn) se vykonal pouze jediný SQL příkaz Select do tabulky studentů;
- V poli zobrazujícím výsledky dotazu i na podruhé zůstává Černý a Informatika!
- Pokud se po skončení běhu programu ve vývojovém prostředí podíváme na databázi, zjistíme, že mezi studenty je Bílý a v oborech IT!

Kešování

Jak lze zmíněné chování pozměněného programu vysvětlit? **LINQ to SQL provádí kešování dat získaných z SQL Serveru**. Jinými slovy, vytváří si dočasné lokální kopie těchto dat (tím se trochu podobá odpojenému režimu ADO.NET).

V testovacích datech jsem měl 2 informatiky, 2 právníky a 4 ekonomy. Přesto dotaz do databáze na název oboru proběhl za každý obor jen jednou. Neprováděl se za každého studenta jednou. Druhý, třetí a čtvrtý ekonom již bral název oboru z lokálně zakešovaných dat zjištěných z databáze u prvního ekonoma. Navíc při druhém volání metody `Dotaz` už se názvy nezjišťovaly vůbec. Kešování tedy zřejmě existuje z výkonových důvodu - aby se databáze a následně program nezdržovaly zjišťováním dat, která jsou již jednou zjištěná.

Kešování vysvětluje, proč se při druhém volání zobrazuje stále Černý, přestože v databázi změna na Bílého evidentně proběhla. LINQ to SQL jednoduše používá dříve zjištěné příjmení.

Poznamenám, že data se kešují vždy v rámci jedné instance řídicí třídy LINQ. Každá nová instance si vytváří svou vlastní kopii těch dat, o nichž to „uzná za vhodné“. Tím se dostáváme k vysvětlení toho, proč v původní verzi programu k žádnému kešování nedocházelo. Mám za to, že se uchovávají pouze záznamy, které patří k nějaké tabulce a mají primární klíč. Prozkoumáte-li SQL příkaz Select z původní verze, zjistíte, že SQL Server tehdy vracel pouze sérii řetězců vytvořených vždy ze dvou tabulek, nikoli sérii záznamů s primárními klíči.

Závěr

Pokud uznáme za potřebné, můžeme instanci řídicího objektu LINQ to SQL udržovat libovolně dlouho. Domnívám se, že jediným skutečným omezením její doby života je postupná ztráta aktuálnosti zakešovaných dat činností buď jiných instancí řídicí třídy, nebo (častěji) jiných programů pracujících s toutéž databází, pravděpodobně spuštěných na jiném stroji.

Vzhledem k uváděné nenáročnosti vytvoření řídicího objektu sice nejspíš jeho dlouhodobé držení nějak potřebné není, ale minimálně tak dostáváme odpověď na znepokojující otázku, jak to je s tou vlastní definicí „jednotky práce“.

Radek Vystavěl, 13. srpna 2012

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