表达式树基础

转载需注明出处：http://www.cnblogs.com/tianfan/

刚接触LINQ的人往往觉得表达式树很不容易理解。通过这篇文章我希望大家看到它其实并不像想象中那么难。您只要有普通的LINQ知识便可以轻松理解本文。

表达式树提供一个将可执行代码转换成数据的方法。如果你要在执行代码之前修改或转换此代码，那么它是非常有价值的。尤其是当你要将C#代码----如LINQ查询表达式转换成其他代码在另一个程序----如SQL数据库里操作它。

但是我在这里颠倒顺序，在文章最后你很容易发现为什么将代码转换到数据中去很有用。首先我需要提供一点背景知识。让我们开始看看相关的创建表达式树的简单语法。

表达式树的语法

考虑下面简单的Lambda表达式：

Func<int, int, int> function = (a,b) => a + b;

这个语句包含三个部分：

一个声明: Func<int, int, int> function

一个等号: =

一个lambda表达式: (a,b) => a + b;

变量function指向两个数字相加的原生可执行代码。上面三步的lambda表达式表示一个简短的如下的手写方法：

public int function(int a, int b)

{

return a + b;

}

上面的方法或lambda表达式都可以这样调用：

int c = function(3, 5);

当方法调用后，变量c将被设成3+5，即8。

上面声明中第一步委托类型Func是在System命名空间中为我们定义好的：

public delegate TResult Func<T1, T2, TResult>(T1 arg1, T2 arg2);

这个代码看上去很复杂，但它在这里只是用来帮我们定义变量function，变量function赋值为非常简单的两个数字相加的lambda表达式。即使你不懂委托和泛型，你仍然应该清楚这是一个声明可执行代码变量引用的方法。在这个例子里它指向一个非常简单的可执行代码。

将代码转换到数据中

在上一节，你看到怎么声明一个指向原生可执行代码的变量。表达式树不是可执行代码，它是一种数据结构。那么我们怎么从表达式的原生代码转换成表达式树？怎么从代码转换成数据？

LINQ提供一个简单语法用来将代码转换到名叫表达式树的数据结构。首先添加using语句引入Linq.Expressions命名空间：

using System.Linq.Expressions;

现在我们可以创建一个表达式树：

Expression<Func<int, int, int>> expression = (a,b) => a + b;

跟上个例子一样的lambda表达式用来转换到类型为Expression<T>的表达式树。标识expression不是可执行代码；它是一个名叫表达式树的数据结构。

Visual Studio 2008的samples包含一个叫ExpressionTreeVisualizer的程序。它可以用来呈现表达式树。图1你可以看到一个展示上面简单表达式语句的对话框截图。注意，对话框上面部分显示的是lambda表达式，下面是用TreeView控件显示的其组成部分。

ExpressionTree

图1：VS2008 C# Samples中的ExpressionTreeVisualizer创建一个表达式树的象征性的输出

编写代码来探索表达式树

我们的例子是一个Expression<TDelegate>。Expression<TDelegate>类有四个属性：

Body: 得到表达式的主体。

Parameters: 得到lambda表达式的参数.

NodeType: 获取树的节点的ExpressionType。共45种不同值，包含所有表达式节点各种可能的类型，例如返回常量，例如返回参数，例如取两个值的小值(<)，例如取两个值的大值(>)，例如将值相加(+)，等等。

Type: 获取表达式的一个静态类型。在这个例子里，表达式的类型是Func<int, int, int>。

如果我们折叠图1的树节点，Expression<TDelegate>的四个属性便显示得很清楚：

ExpressionTreeProperties

图2：将树节点折叠起来，你可以很容易的看到Expression<TDelegate>类的四个主要属性。

你可以使用这四个属性开始探索表达式树。例如，你可以通过这样找到参数的名称：

Console.WriteLine("参数1: {0}, 参数2: {1}", expression.Parameters[0], expression.Parameters[1]);

这句代码输出值a和b：

参数1: a, 参数2: b

这个很容易在图1的ParameterExpression节点找到。

让我们在接下来的代码探索表达式的Body，在这个例子里是(a + b)：

BinaryExpression body = (BinaryExpression)expression.Body;

ParameterExpression left = (ParameterExpression)body.Left;

ParameterExpression right = (ParameterExpression)body.Right;

Console.WriteLine(expression.Body);

Console.WriteLine(" 表达式左边部分: " + "{0}{4} 节点类型: {1}{4} 表达式右边部分: {2}{4} 类型: {3}{4}", left.Name, body.NodeType, right.Name, body.Type, Environment.NewLine);

这段代码产生如下输入：

(a + b)

表达式左边部分: a

节点类型: Add

表达式右边部分: b

类型： System.Int32

同样，你会发现很容易在图1的Body节点中找到这些信息。

通过探索表达式树，我们可以分析表达式的各个部分发现它的组成。你可以看见，我们的表达式的所有元素都展示为像节点这样的数据结构。表达式树是代码转换成的数据。

编译一个表达式：将数据转换回代码

如果我们可以将代码转换到数据，那么我们也应该能将数据转换回代码。这里是让编译器将表达式树转换到可执行代码的简单代码。

int result = expression.Compile()(3, 5);

Console.WriteLine(result);

这段代码会输出值8，跟本文最初声明的lambda函数的执行结果一样。

IQueryable<T>和表达式树

现在至少你有一个抽象的概念理解表达式树，现在是时候回来理解其在LINQ中的关键作用了，尤其是在LINQ to SQL中。花点时间考虑这个标准的LINQ to SQL查询表达式：

var query = from c in db.Customers

where c.City == "Nantes"

select new { c.City, c.CompanyName };

你可能知道，这里LINQ表达式返回的变量query是IQueryable类型。这里是IQueryable类型的定义：

public interface IQueryable : IEnumerable

{

Type ElementType { get; }

Expression Expression { get; }

IQueryProvider Provider { get; }

}

你可以看见，IQueryable包含一个类型为Expression的属性，Expression是Expression<T>的基类。IQueryable的实例被设计成拥有一个相关的表达式树。它是一个等同于查询表达式中的可执行代码的数据结构。

花点时间考虑图3。你可能需要点击它使图片原尺寸显示。这是本节开始的查询表达式的表达式树的可视化显示。此图使用ExpressionTreeVisualizer创建，就像我使用它在图1创建基础的lambda表达式树一样。

LinqToSqlExpressionTree01

图3：此复杂的表达式树由上面的样例LINQ to SQL查询表达式生成。(点击图片看查看大图)

为什么要将LINQ to SQL查询表达式转换成表达式树呢？

你已经学习了表达式树是一个用来表示可执行代码的数据结构。但到目前为止我们还没有回答一个核心问题，那就是为什么我们要做这样的转换。这个问题是我们在本文开始时提出来的，现在是时候回答了。

一个LINQ to SQL查询不是在你的C#程序里执行的。相反，它被转换成SQL，通过网络发送，最后在数据库服务器上执行。换句话说，下面的代码实际上从来不会在你的程序里执行：

var query = from c in db.Customers

where c.City == "Nantes"

select new { c.City, c.CompanyName };

它首先被转换成下面的SQL语句然后在服务器上执行：

SELECT [t0].[City], [t0].[CompanyName]

FROM [dbo].[Customers] AS [t0]

WHERE [t0].[City] = @p0

从查询表达式的代码转换成SQL查询语句----它可以通过字符串形式被发送到其他程序。在这里，这个程序恰好是SQL Server数据库。像这样将数据结构转换到SQL显然比直接从原生IL或可执行代码转换到SQL要容易得多。这有些夸大问题的难度，只要试想转换0和1的序列到SQL！

现在是时候将你的查询表达式转换成SQL了，描述查询的表达式树是分解并解析了的，就像我们在上一节分解我们的简单的lambda表达式树一样。当然，解析LINQ to SQL表达式树的算法比我们用的那个要复杂得多，但规则是一样的。一旦解析了表达式树的各部分，那么LINQ开始斟酌以最好的方式生成返回被请求的数据的SQL语句。

表达式树被创建是为了制造一个像将查询表达式转换成字符串以传递给其他程序并在那里执行这样的转换任务。就是这么简单。没有巨大奥秘，不需要挥舞魔杖。只是简单的：把代码，转换成数据，然后分析数据发现其组成部分，最后转换成可以传递到其他程序的字符串。

由于查询来自编译器封装的抽象的数据结构，编译器可以获取任何它想要的信息。它不要求执行查询要在特定的顺序，或用特定的方式。相反，它可以分析表达式树，寻找你要做的是什么，然后再决定怎么去做。至少在理论上，我们可以自由的考虑各种因素，比如网络状况，数据库负载，结果集是否有效，等等。在实际中LINQ to SQL不考虑所有这些因素，但它理论上可以自由的做几乎所有想做的事。此外，人们可以通过表达式树将自己编写的代码，分析并转换成跟LINQ to SQL提供的完全不同的东西。

IQueryable<T>和IEnumerable<T>

正如你可能知道的，LINQ to Objects的查询表达式返回IEnumerable<T>而不是IQueryable<T>。为什么LINQ to Objects使用IEnumerable<T>而LINQ to SQL使用IQueryable<T>？

这里是IEnumerable<T>的定义：

public interface IEnumerable<T> : IEnumerable

{

IEnumerator<T> GetEnumerator();

}

正如你看到的，IEnumerable<T>并不包含类型为Expression的属性。这指出LINQ to Objects和LINQ to SQL的根本区别。后者大量使用了表达式树，但LINQ to Objects很少使用。

为什么表达式树不是LINQ to Objects的标准部分？虽然答案不一定会马上出现，但这是很有意义的一旦你发现这个问题。

考虑这个简单LINQ to Objects查询表达式：

List<int> list = new List<int>() { 1, 2, 3 };

var query = from number in list

where number < 3

select number;

这个LINQ查询返回在我们的list中比3小的数字；就是说，这里返回数字1和2。显然没有必要将查询转换成字符串来顺序传递给其他程序并获取正确的结果。相反，可以直接转换查询表达式为可执行的.NET代码。这里并不需要将它转换成字符串或对它执行任何其他复杂操作。

可是这有点理论化，在实际中某些特殊情况下其分隔线可能有些模糊，总体上讲规则相当简单：

如果代码可以在程序里执行那么可以使用名为IEnumerable<T>的简单类型完成任务

如果你需要将查询表达式转换成将传递到其他程序的字符串，那么应该使用IQueryable<T>和表达式树。

像LINQ to Amazon这样的项目需要将查询表达式转换成web service调用执行外部程序，通常使用IQueryable<T>和表达式树。LINQ to Amazon将它的查询表达式转换成数据，通过web service传递给另一个甚至不是C#写的程序。将C#代码转换成到某些能传递到web service的东西时，表达式树内在的抽象是非常有用的。要在程序内执行的代码，仍然可以经常使用而抛开表达式树。例如下面的查询使用IEnumerable<T>，因为它调用到当前程序的.NET反射API：

var query = from method in typeof(System.Linq.Enumerable).GetMethods()

orderby method.Name

group method by method.Name into g

select new { Name = g.Key, Overloads = g.Count() };

概要

本文覆盖了表达式树的一些基本情况。通过将代码转换成数据，这些数据结构揭示并描绘表达式的组成部分。从最小的概念上讲，理解表达式树是相当简单的。它获取可执行表达式并获取其组成部分放入树形数据结构。例如，我们检测这个简单的表达式：

(a,b) => a + b;

通过研究来源于这个表达式的树，你能看到创建树的基本规则，见图1。

你同样可以看到表达式树在LINQ to SQL里扮演非常重要的角色。尤其，他们是LINQ to SQL查询表达式用来获取逻辑的数据抽象。解析并分析此数据得到SQL语句，然后发送到服务器。

LINQ使查询C#语言的一个普通类即有类型检查也有智能感知。其代码是类型检查和智能感知的，必须使用正确的C#语法，它能直接转换到可执行代码，就像任何其他C#代码一样被转换和执行。表达式树使将可执行代码转换成能传递到服务器的SQL语句相对容易。

查询返回IEnumerable<T>优于IQueryable<T>表示不使用表达式树。作为一般性规则，可以这么说：LINQ查询在程序内执行时不需要表达式树，当代码在程序外执行时可以利用表达式树。

分类: C#

好文要顶 关注我 收藏该文