Lambda Expression Tree Parsing

Asked 16 years, 1 month ago Modified 4 years, 2 months ago Viewed 17k times

16

I am trying to use Lambda Expressions in a project to map to a third party query API. So, I'm parsing the Expression tree by hand.

If I pass in a lambda expression like:

```
p => p.Title == "title"
```

everything works.

However, if my lambda expression looks like:

```
p => p.Title == myaspdropdown.SelectedValue
```

Using the .NET debugger, I don't see the actual value of that funciton. Instead I see something like:

```
p => p.Title =
(value(ASP.usercontrols_myaspusercontrol_ascx).myaspdropdown.SelectedValue)
```

What gives? And when I try to grab the right side of the expression as a string, I get `(value(ASP.usercontrols_myaspusercontrol_ascx).myaspdropdown.SelectedValue)` instead of the actual value. **How do I get the actual value?**

c# lambda expression-trees

Share
Improve this question
Follow

edited Jul 15, 2015 at 23:01

Enamul Hassan
5,445 ● 23 ● 43 ● 58

asked Oct 26, 2008 at 18:33

Keith Fitzgerald
5,701 ● 14 ● 45 ● 58

I think the issue is (and I'm not an expert with Expressions, so this isn't an answer) is that the expression isn't a lambda; it describes the data you wish to retrieve. It isn't executed, its interpreted. – user1228 Oct 26, 2008 at 19:20

That means that the value isn't substituted for your expression at runtime, its set permanently at compile time. In order to get the value, the code interpreting the lambda would have to understand the concept of the user control and how to extract it – user1228 Oct 26, 2008 at 19:22

And unless the code that is interpreting the expression has access to your conntrol and is coded to do this, it isn't possible. If the code has an overload or another method that does the same thing, but takes actual values, you can use that and a lambda to retrieve the value at runtime. – user1228 Oct 26, 2008 at 19:23

4 Answers

Sorted by: Highest score (default)

21

Remember that when you're dealing with the lambda expression as an expression tree, you don't have executable code. Rather you have a tree of expression elements, that make up the expression you wrote.

Charlie Calvert has a good post that discusses this in detail. Included is an example of using an expression visualiser for debugging expressions.

In your case, to get the value of the righthand side of the equality expression, you'll need to create a new lambda expression, compile it and then invoke it.

I've hacked together a quick example of this - hope it delivers what you need.

```
public class Class1
{
    public string Selection { get; set; }

    public void Sample()
    {
        Selection = "Example";
        Example<Book, bool>(p => p.Title == Selection);
    }

    public void Example<T,TResult>(Expression<Func<T,TResult>> exp)
    {
        BinaryExpression equality = (BinaryExpression)exp.Body;
        Debug.Assert(equality.NodeType == ExpressionType.Equal);

        // Note that you need to know the type of the rhs of the equality
        var accessorExpression = Expression.Lambda<Func<string>>
(equality.Right);
        Func<string> accessor = accessorExpression.Compile();
        var value = accessor();
        Debug.Assert(value == Selection);
    }
}

public class Book
{
    public string Title { get; set; }
}
```

Share

Improve this answer

edited Oct 21, 2020 at 5:36

Katie Kilian

6,965 ● 5 ● 42 ● 65

answered Oct 27, 2008 at 8:53

Bevan

44.3k ● 10 ● 84 ● 133

Follow

1

To get the actual value, you need to apply the logic of the expression tree to whatever context you've got.

The whole point of expression trees is that they represent the *logic* as data rather than evaluating the expression. You'll need to work out what the lambda expression truly means. That may mean evaluating some parts of it against local data - you'll need to decide that for yourself. Expression trees are very powerful, but it's not a simple matter to parse and use them. (Ask anyone who's written a LINQ provider... Frans Bouma has bemoaned the difficulties several times.)

Share Improve this answer Follow

answered Oct 26, 2008 at 19:46

Jon Skeet
1.5m ● 889 ● 9.3k ● 9.3k

hi john - maybe this question is clearer: stackoverflow.com/questions/238765/… – Keith Fitzgerald Oct 26, 2008 at 23:37

0

Just been struggling with exactly the same issue, thanks Bevan. On an extension, the following is a generic pattern you can use to extract the value (using this in my query engine).

```
    [TestFixture]
public class TestClass
{
    [Test]
    public void TEst()
    {
        var user = new User {Id = 123};
        var idToSearch = user.Id;
        var query = Creator.CreateQuery<User>()
            .Where(x => x.Id == idToSearch);
    }
}

public class Query<T>
{
    public Query<T> Where(Expression<Func<T, object>> filter)
    {
        var rightValue = GenericHelper.GetVariableValue(((BinaryExpression)
((UnaryExpression)filter.Body).Operand).Right.Type, ((BinaryExpression)
((UnaryExpression)filter.Body).Operand).Right);
        Console.WriteLine(rightValue);
        return this;
    }
}

internal class GenericHelper
```

```
{
    internal static object GetVariableValue(Type variableType, Expression
expression)
    {
        var targetMethodInfo =
typeof(InvokeGeneric).GetMethod("GetVariableValue");
        var genericTargetCall =
targetMethodInfo.MakeGenericMethod(variableType);
        return genericTargetCall.Invoke(new InvokeGeneric(), new[] { expression
});
    }
}

internal class InvokeGeneric
{
    public T GetVariableValue<T>(Expression expression) where T : class
    {
        var accessorExpression = Expression.Lambda<Func<T>>(expression);
        var accessor = accessorExpression.Compile();
        return accessor();
    }
}
```

Share Improve this answer Follow

answered Sep 16, 2010 at 12:20

squirrel
171 ● 2

Rather than using your reflection to call InvokeGeneric, what I would do is `Expression<Func<object>>(Expression.Convert(expression, typeof(object))).Compile()` – Double Down Mar 13, 2011 at 14:13

-1

I'm not sure I understand. Where are you "seeing" that? Is that at design-time or run-time? Lambda expressions can be thought of essentially as anonymous delegates, and will operate with deferred execution. So you shouldn't expect to see the value assigned until after execution has passed that line, obviously.
I don't think that's really what you mean though... if you clarify the question a bit maybe I can help :)

Share Improve this answer Follow

answered Oct 26, 2008 at 18:42

Grank
5,292 ● 8 ● 36 ● 36

updated question. it's in the debugger [and also when i try to grab the right side of the expression] – Keith Fitzgerald Oct 26, 2008 at 18:47

@Grank: Lambda expressions can be converted into either expression trees or delegates. It sounds like you're thinking of the conversion into delegates. – Jon Skeet Oct 26, 2008 at 19:43