How to use System.Linq.Expressions.Expression to filter based on children?

Asked 7 years, 5 months ago Active 7 months ago Viewed 21k times

10

3

I have a filter that I use across many methods:

```
Expression<Func<Child, bool>> filter = child => child.Status == 1;
```

(actually is more complex than that)

And I have to do the following

```
return db.Parents.Where(parent => parent.Status == 1 &&
                                  parent.Child.Status == 1);
```

where the condition is the same as in the filter above.

I want to reuse the filter in this method. But I don't know how. I tried

```
return db.Parents.Where(parent => parent.Status == 1 &&
                                  filter(parent.Child));
```

but an Expression can't be used as a method

c# .net linq-to-sql expression

asked Apr 27 '12 at 19:44

Jader Dias

43.6k 137 387 596

4

If you want to combine expressions and still be able to use linq-to-sql, you may want to have a look at LinqKit. It walks inside your expression and replaces all the function calls by their contents before the sql conversion.

This way you'll be able to use directly

```
return db.Parents
       .AsExpandable()
       .Where(parent => parent.Status == 1 && filter(parent.Child));
```

+50

answered May 7 '12 at 18:11

McX

867 8 14

1

You can try this:

```
var compiledFilter = filter.Compile();
foreach (var parent in db.Parents.Where(parent => parent.Status == 1))
    if (compiledFilter(parent.Child))
        yield return parent;
```

It requires you to pull all of the parents, but unlike @HugoRune's solution, it doesn't require a 1:1 relation of Parent:Child.

I don't think this will be useful for your situation because of the different types involved, but just in case, here is an example of how you can combine `Expression` s: How do I combine LINQ expressions into one?

Edit: I had previously suggested using `Compile()` , but that doesn't work over LINQ-to-SQL.

edited May 23 '17 at 11:53

Community ♦

1 1

answered Apr 27 '12 at 19:51

Tim S.

47.6k 5 75 109

I do not think you can compile expressions inside linq-to-sql statements. Lambdas only work in normal linq – HugoRune Apr 27 '12 at 19:54

1

Well, if there is a 1:1 relationship between parent and child (unlikely, but the example seems to imply that) then you could do it like this:

```
return db.Parents
.Where(parent => parent.Status == 1)
.Select(parent => parent.Child)
.Where(filter)
.Select(child=> child.Parent);
```

Otherwise it will be hard.

You could do it with dynamic linq but that is probably overkill.

You could generate your expression tree manually, but that is also quite complicated. I have not tried that myself.

As a last resort you could of course always call `yourQuery.AsEnumerable()` , this will cause linq-to-sql to translate your query into sql up to this point and perform the rest of the work on the client-side; then you can .compile() your expression. However you lose the performance benefits of linq-to-sql (and compile() itself is quite slow; whenever it is executed, it calls the JIT-compiler):

```
return db.Parents
.Where(parent => parent.Status == 1)
.AsEnumerable()
.Where(parent  => filter.Compile().Invoke(parent.Child))
```

Personally I'd just define the expression twice, once for child and once for parent.child:

```
Expression<Func<Child, bool>> filterChild = child => child.Status == 1;
Expression<Func<Parent, bool>> filterParent = parent => parent.Child.Status == 1;
```

Might not be the most elegant, but probably easier to maintain than the other solutions

edited Apr 27 '12 at 20:53

answered Apr 27 '12 at 20:01

HugoRune

8,614 6 46 116

0

```
public interface IStatus { public int Status { get; set; } }
public class Child : IStatus { }
public class Parent : IStatus
{public Child Child { get; set; }  }

Func<IStatus, bool> filter = (x) => x.Status == 1;
var list = Parents.Where(parent => filter(parent) && filter(parent.Child));
```

Hope this helps!

answered Apr 27 '12 at 20:29

Eduardo Crimi
1,241 12 13

1 Func does not work with linq-to-sql, it has to be Expression<Func>. And since parent and child are presumably auto-generated classes based on the db scheme, giving them a common base class is not that simple – HugoRune Apr 27 '12 at 20:33

Assuming the criteria are really so simply shared, this is actually a good approach: auto-generated classes are declared as `partial`, so that you can add things like interface declarations/implementations in another file. Since you'll need to use Expression instead of Func, the given code won't work as-is, but you can combine them using methods shown at stackoverflow.com/questions/1922497/…. – Tim S. Apr 27 '12 at 20:35

0

Could you just use the expression as a function instead?

Instead of:

```
Expression<Func<Child, bool>> filter = child => child.Status == 1;
```

Use that same expression as a generic function this way:

```
Func<Child, bool> filter = child => child.Status == 1;
```

Then you will be able to use the function in just the same way you were trying to use an expression:

Edit: I misunderstood the question. This is a bad answer. 6+ years out, I'm still getting comments to the effect that this doesn't work. I'm not sure, from a hygiene perspective, if it would be better to just delete the answer, or add this edit and let the answer stand as an example of something that decidedly doesn't work. I'm open to advisement on that.

edited Nov 15 '18 at 18:22

answered May 9 '12 at 14:30

jdmcnair

1,105 13 32

2 Your solution doesn't translate to SQL. – Jader Dias May 9 '12 at 16:43

In linq to sql, Expression works, instead of Func. – Lali Mar 27 '15 at 12:03

This does not work. If it does, please show how. – johnny Nov 15 '18 at 15:25

Yes, I misunderstood the question. This is a bad answer. – jdmcnair Nov 15 '18 at 18:16

0

There's no need for external libraries or mucking around with expression trees. Instead, write your lambda functions to use query chaining and take advantage of LINQ's deferred execution.

Instead of:

```
Expression<Func<Child, bool>> filter = child => child.Status == 1;
```

Rewrite it as:

```
Func<IQueryable<Parent>, IQueryable<Parent>> applyFilterOnParent = query => query.Where(parent => parent.Child.Status == 1);

Func<IQueryable<Child>, IQueryable<Child>> applyFilterOnChild = query => query.Where(child => child.Status == 1);
```

Now, instead of:

```
return db.Parents.Where(parent => parent.Status == 1 &&
                                  filter(parent.Child));
```

You can write:

```
var query = db.Parents.AsQueryable();
query = applyFilterOnParent(query);
```

And you can re-use the apply() filter functions in other LINQ queries. This technique works well when you want to use lambda functions together with LINQ-to-SQL, because LINQ will not translate a lambda function to SQL.

answered Feb 19 at 4:23

humbads

2,192 19 19