Define var outside the if-else statement scope for later use

Asked 4 years, 2 months ago Active 4 years, 2 months ago Viewed 1k times

2

I have an if-else statement which gets search results from database using Linq, each one is different than the other, but the returned type if the same.

My problem is that I can't initialize the `var` variable outside the if-else statement. I need it because after the statement I continue to query the results.

The result is of type `iQueryable<mission>` ;

How can I make the var in the function scope be able to accept the results?

I tried initializing it with `Enumerable.Empty<mission>().AsQueryable()` and it didn't work, tried using `dynamic` variable and set also set the `var missions = null` , and it didn't work either.

c# linq

asked Jul 1 '15 at 13:21

Idan Shechter
4,154 23 88 177

1 Can you show the code in question and the errors that you are receiving? – juharr Jul 1 '15 at 13:23

1 What's wrong with `IQueryable<Mission> = null;` ? – haim770 Jul 1 '15 at 13:24

1 Can you not just create an IQueryable<mission> instead of a var? – AllFallD0wn Jul 1 '15 at 13:25

1 It's not mandatory to define variables as `var` , you can define it as `IQueryable<mission>` outside `if` – Claudio Redi Jul 1 '15 at 13:25

1 The only time you actually **need** `var` is if you're using an anonymous type, in which case dealing with your sort of situation can be tricky and often impossible, forcing one to create a new type for the code to work. In other cases `var` is just syntactic sugar. – Jon Hanna Jul 1 '15 at 13:28

2 Answers

So... don't use `var` ?

5

```
IQueryable<Mission> variable;

if (...)
{
  ...
```

Also, as an alternative to `var variable = null` (which doesn't compile), you can use `default` :

```
var variable = default(IQueryable<Mission>);
```

Unlike the code above, though, this will hide errors stemming from "forgetting" to assign to the variable - you're assigning a "typed" `null` , so the variable has a value assigned. In the first example, `variable` isn't *assigned*, so if there's any branch of your code that tries to read the value of `variable` *without* writing to it first, the compiler will report an error. Also note that `default` only returns `null` for reference types. Value types will basically return whatever zero means for that particular type - for example, `default(int)` is simply zero, and `default(DateTime)` is `01/01/0001 00:00:00` (`DateTime.MinValue`).

edited Jul 1 '15 at 13:34

answered Jul 1 '15 at 13:25

Luaan

51.3k 6 63 88

Simple solution, just haven't thought about doing it this way. Noob me, but thanks for enlighten me. I'll accept this answer. – Idan Shechter Jul 1 '15 at 13:28

Thanks for the addition, learning a lot today. – Idan Shechter Jul 1 '15 at 13:36

4

The compiler has to be able to determine the *actual* type of a `var` variable *at compile time*. If you do this:

```
var foo = null;
```

Or this:

```
var foo;
```

So if you need to have a variable set to `null`, you have to *tell the compiler what type it is*:

```
IQueryable<Mission> foo = null;
```

Or, if you don't want to initialize it yet:

```
IQueryable<Mission> foo;
```

Look at the MSDN reference on implicitly typed local variables. In particular:

> The var keyword instructs the compiler to infer the type of the variable from the expression on the right side of the initialization statement.

You can't infer a type from `null` and you certainly can't do it if there is no expression on the right-hand side.

Also:

> var can only be used when a local variable is declared and initialized in the same statement; the variable cannot be initialized to null, or to a method group or an anonymous function.

edited Jul 1 '15 at 14:03

answered Jul 1 '15 at 13:27

Matt Burland

37.8k 11 74 139