C# Reflection: How to get the type of a Nullable<int>?

27

3

What I want to do is something like this:

```
switch( myObject.GetType().GetProperty( "id") )
{
    case ??:
        // when Nullable<Int32>, do this
    case ??:
        // when string, do this
    case ??:
        // when Nullable<bool>, do this
```

What path under object.GetType() would have the string name of the datatype that I could compare using a case statement? I need to know the type so I can have one of many Convert.ToInt32(string) that will set the value of myObject using Reflection.

c# .net reflection typeof gettype

edited Dec 18 '11 at 6:33
Cody Gray ♦
198k 37 398 482

asked Dec 18 '11 at 6:25
Zachary Scott
11.2k 31 103 188

1 This is almost definitely doing things wrong. Why can't you take advantage of polymorphism, rather than using a giant switch statement? – Cody Gray ♦ Dec 18 '11 at 6:33

If you're suggesting to create several functions with different parameter types, I could. In this case, I am copying a set of properties of different types from one object to another where one is always a string type. So, I need to convert the value to assign it. That and I have very little experience with Reflection. – Zachary Scott Dec 18 '11 at 6:40

@CodyGray he might be doing something with regards to reflection, say writing his own ORM. Or writing a custom serializer for all this DAOs, you never know. – nawfal Jul 2 '16 at 7:06

That's why it is "almost definitely", rather than "certainly". And even if he is doing those things, there are arguments to be made about why those are

from two different sources, it took me much less time to reflect on members than go and implement an interface on all the types in project, if you need it quick and dirty. Along those lines.. When convention over configuration makes sense... Etc. Shoehorning OOP everywhere is a bad idea. – nawfal Jul 2 '16 at 12:10

5 Answers

12

Update: Looks like C# 7 will support switching on `Type` s as the asker of this question was trying to do. It's a little different though so watch out for syntax landmines.

You don't need a string name to compare it:

```
if (myObject.GetType().GetProperty("id").PropertyType == typeof(Nullable<Int32>))
    // when Nullable<Int32>, do this
else if (myObject.GetType().GetProperty("id").PropertyType == typeof(string))
    // when string, do this
else if (myObject.GetType().GetProperty("id").PropertyType == typeof(Nullable<bool>))
    // when Nullable<bool>, do this
```

edited Nov 10 '17 at 23:40

answered Dec 18 '11 at 6:27

M.Babcock

16.5k 3 43 77

1 Are you sure you compare .GetType() to typeof(Nullable<Int32>) or is it a property under GetType()? Assuming int? and Nullable<Int32> are the same type, a.GetType() != typeof(Nullable<Int32>). – Zachary Scott Dec 18 '11 at 6:35

3 Just like Eric said, there is no boxed `Nullable<T>` . Nullables get boxed as the underlying type if they are not null, or `null` which is typeless if they are. Thus your code won't work. – CodesInChaos Dec 19 '11 at 21:09

2 This structure exists. But `GetType()` will never return `Nullable<T>` , since `GetType()` operates on boxed values, and nullable values don't get boxed as `Nullable<T>` . So both the first and the third `if` in your example will never be satisfied. – CodesInChaos Dec 19 '11 at 21:53

5 He worked on `myObject.GetType().GetProperty("ID").PropertyType` . The static type of the property can be `Nullable<T>` , it's just `GetType()` that can't return a nullable. – CodesInChaos Dec 19 '11 at 22:16

61

I've been using the following type of code to check if the type is nullable and to get the actual type:

```
if (type.IsGenericType && type.GetGenericTypeDefinition() == typeof(Nullable<>))
{
    return Nullable.GetUnderlyingType(type);
}
```

If the type is e.g. Nullable this code returns the int part (underlying type). If you just need to convert object into specific type you could use `System.Convert.ChangeType` method.

edited Dec 18 '11 at 13:08
Jason Down
17.2k 9 76 109

answered Dec 18 '11 at 8:28
Toni Parviainen
1,887 1 10 13

+1 and worth noting this method does the same check inside and returns null if the given type is not `typeof(Nullable<>)` so doing a null check afterwards instead of the if statement would also work. – Connell Jan 14 '17 at 21:10

Works like a charm, thank you, Jason. – Sebastián Guerrero May 17 '18 at 21:35

18

The question is very confusing. Is "myObject" the object that might be a nullable int? Or is the property "id" possibly of type nullable int?

If the former, your question cannot be answered because it presupposes a falsehood. There is no such thing as a boxed nullable int. I note that all of the answers which propose `if (myobject.GetType() == typeof(int?))` are therefore incorrect; the condition will never be true.

When you convert a nullable int to object, either it becomes a null reference (if the nullable int had no value) or it becomes a boxed int. There is no way to determine if an object contains a nullable int because an object *never* contains a nullable int.

If the latter, compare the *property type* to `typeof(int?)`. You cannot use a switch; only constants may be used for switch cases and types are not constants.

All that said, this is a bad code smell. Why are you using reflection in the first place?

Update: in C# 7, switch statements no-longer require constant expressions :) – Sam Rueby Nov 10 '17 at 22:15

And provide a mechanism for switching on types. That feature was first proposed in... 2001? I think. So, 16 years, not bad. – Eric Lippert Nov 10 '17 at 22:19

2

In .net, instances of value types are just collections of bits, with no associated type information. For every value type other than `Nullable<T>`, however, the system also auto-generates a corresponding class type which derives from `System.ValueType`. A widening conversion exists from the value type to the auto-generated class type, and a narrowing conversion from the auto-generated class type to the value type. In the case of `Nullable<T>`, there is no corresponding auto-generated class type with conversions to/from the value type; instead, widening conversions exist in both directions between `Nullable<T>` and the class type associated with `T`.

As far as I can tell, this weird behavior was implemented to allow comparisons between `null` and an empty `Nullable<T>` to return true.

answered Jan 26 '12 at 23:56

supercat
59k 4 123 159

0

As @Cody Gray said if statements would probably be the best way

```
var t = myObject.GetType();

if (t == typeof(Nullable<int>))
{ }
else if (t == typeof(string))
{}
else if (t==typeof(Nullable<bool>))
{}
```

answered Dec 18 '11 at 6:42

heads5150
5,760 2 20 35

Wrong answer. `myObject.GetType` will always give `int` if it is declared either as `int` or `int?` – nawfal Jul 2 '16 at 7:00