How to check if an object is a certain type

Asked 13 years, 5 months ago Modified 5 years, 6 months ago Viewed 180k times

125

I am passing various objects to a subroutine to run the same process but using a different object each time. For example, in one case I am using a ListView and in another case I am passing a DropDownList.

I want to check if the object being passed is a DropDownList then execute some code if it is. How do I do this?

My code so far which doesn't work:

```
Sub FillCategories(ByVal Obj As Object)
    Dim cmd As New SqlCommand("sp_Resources_Categories", Conn)
    cmd.CommandType = CommandType.StoredProcedure
    Obj.DataSource = cmd.ExecuteReader
    If Obj Is System.Web.UI.WebControls.DropDownList Then

    End If
    Obj.DataBind()
End Sub
```

.net vb.net object drop-down-menu object-type

Share Improve this question Follow

edited Jul 5, 2011 at 8:59
Cody Gray ♦
244k ● 52 ● 501 ● 581

asked Jul 5, 2011 at 8:48
Leah
2,597 ● 5 ● 25 ● 28

2 Answers

Sorted by: Highest score (default)

212

In VB.NET, you need to use the `GetType` method to retrieve the type of an instance of an object, and the `GetType()` operator to retrieve the type of another known type.

Once you have the two types, you can simply compare them using the `Is` operator.

So your code should actually be written like this:

```
Sub FillCategories(ByVal Obj As Object)
    Dim cmd As New SqlCommand("sp_Resources_Categories", Conn)
    cmd.CommandType = CommandType.StoredProcedure
    Obj.DataSource = cmd.ExecuteReader
    If Obj.GetType() Is GetType(System.Web.UI.WebControls.DropDownList) Then
    End If
    Obj.DataBind()
End Sub
```

You can also use the `TypeOf` operator instead of the `GetType` method. Note that this tests if your object is *compatible* with the given type, not that it is the same type. That would look like this:

```
If TypeOf Obj Is System.Web.UI.WebControls.DropDownList Then

End If
```

Totally trivial, irrelevant nitpick: Traditionally, the names of parameters are camelCased (which means they always start with a lower-case letter) when writing .NET code (either VB.NET or C#). This makes them easy to distinguish at a glance from classes, types, methods, etc.

Share
Improve this answer
Follow

edited Aug 3, 2017 at 12:08
Sebastian Brosch
43.5k ● 15 ● 76 ● 89

answered Jul 5, 2011 at 9:04
Cody Gray ♦
244k ● 52 ● 501 ● 581

1 Thanks for your answer. I tried that code but actually the only thing is that it doesn't work with the '=' operator. I had to change it to 'Is'. The error I had when it was '=' was "Operator '=' is not defined for types 'System.Type' and 'System.Type'." – Leah Jul 5, 2011 at 9:28

1 @Leah: Yeah, sorry about that. Looks like I should start paying more attention when writing answers. `TypeOf` is probably an even simpler option, at least in terms of code readability; I've updated the answer with an example of that, too. – Cody Gray ♦ Jul 5, 2011 at 9:33

41 There is an important difference between the two, which is what led me to this post. The TypeOf check will return True if the object is of a class that inherits from the type you are checking against, whereas GetType will only return True if it is exactly the same class. – Abacus Aug 13, 2013 at 17:41

Totally trivial, irrelevant counterpoint: Even though the VS CodeAnalysis complains, I still feel the argument names are part of the public interface and so are PascalCase in my code. – Mark Hurd Aug 7, 2015 at 15:19

Is there a performance difference between the two? - What about `Select Case (Obj.GetType())` with multiple test cases Vs multiple `IF TypeOf Obj is ...`? – Luke T O'Brien May 22, 2017 at 16:27

6

Some more details in relation with the response from Cody Gray. As it took me some time to digest it I though it might be usefull to others.

First, some definitions:

1. There are TypeNames, which are string representations of the type of an object, interface, etc. For example, `Bar` is a TypeName in `Public Class Bar`, or in `Dim Foo as Bar`. TypeNames could be seen as "labels" used in the code to tell the compiler which type definition to look for in a dictionary where all available types would be described.
2. There are [`System.Type`](System.Type) objects which contain a value. This value indicates a type; just like a `String` would take some text or an `Int` would take a number, except we are storing types instead of text or numbers. `Type` objects contain the type definitions, as well as its corresponding TypeName.

Second, the theory:

1. `Foo.GetType()` returns a `Type` object which contains the type for the variable `Foo`. In other words, it tells you what `Foo` is an instance of.
2. `GetType(Bar)` returns a `Type` object which contains the type for the TypeName `Bar`.
3. In some instances, the type an object has been `Cast` to is different from the type an object was first instantiated from. In the following example, MyObj is an `Integer` cast into an `Object`:

   ```
   Dim MyVal As Integer = 42
   Dim MyObj As Object = CType(MyVal, Object)
   ```

So, is `MyObj` of type `Object` or of type `Integer`? `MyObj.GetType()` will tell you it is an `Integer`.

4. But here comes the `Type Of Foo Is Bar` feature, which allows you to ascertain a variable `Foo` is compatible with a TypeName `Bar`. `Type Of MyObj Is Integer` and `Type Of MyObj Is Object` will both return True. For most cases, TypeOf will indicate a variable is compatible with a TypeName if the variable is of that Type or a Type that derives from it. More info here: https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/operators/typeof-operator#remarks

The test below illustrate quite well the behaviour and usage of each of the mentionned keywords and properties.

```
Public Sub TestMethod1()

    Dim MyValInt As Integer = 42
    Dim MyValDble As Double = CType(MyValInt, Double)
    Dim MyObj As Object = CType(MyValDble, Object)

    Debug.Print(MyValInt.GetType.ToString) 'Returns System.Int32
    Debug.Print(MyValDble.GetType.ToString) 'Returns System.Double
    Debug.Print(MyObj.GetType.ToString) 'Returns System.Double

    Debug.Print(MyValInt.GetType.GetType.ToString) 'Returns System.RuntimeType
```

```
    Debug.Print(MyValDble.GetType.GetType.ToString) 'Returns System.RuntimeType
    Debug.Print(MyObj.GetType.GetType.ToString) 'Returns System.RuntimeType

    Debug.Print(GetType(Integer).GetType.ToString) 'Returns System.RuntimeType
    Debug.Print(GetType(Double).GetType.ToString) 'Returns System.RuntimeType
    Debug.Print(GetType(Object).GetType.ToString) 'Returns System.RuntimeType

    Debug.Print(MyValInt.GetType = GetType(Integer)) '# Returns True
    Debug.Print(MyValInt.GetType = GetType(Double)) 'Returns False
    Debug.Print(MyValInt.GetType = GetType(Object)) 'Returns False

    Debug.Print(MyValDble.GetType = GetType(Integer)) 'Returns False
    Debug.Print(MyValDble.GetType = GetType(Double)) '# Returns True
    Debug.Print(MyValDble.GetType = GetType(Object)) 'Returns False

    Debug.Print(MyObj.GetType = GetType(Integer)) 'Returns False
    Debug.Print(MyObj.GetType = GetType(Double)) '# Returns True
    Debug.Print(MyObj.GetType = GetType(Object)) 'Returns False

    Debug.Print(TypeOf MyObj Is Integer) 'Returns False
    Debug.Print(TypeOf MyObj Is Double) '# Returns True
    Debug.Print(TypeOf MyObj Is Object) '# Returns True


End Sub
```

EDIT

You can also use `Information.TypeName(Object)` to get the TypeName of a given object. For example,

```
Dim Foo as Bar
Dim Result as String
Result = TypeName(Foo)
Debug.Print(Result) 'Will display "Bar"
```

Share

Improve this answer

Follow

edited Jun 12, 2019 at 11:57

answered Apr 17, 2019 at 16:26

Ama's

Ama
1,565 ● 14 ● 28