How to check programmatically if a type is a struct or a class?

Ask Question

40

8

How to check programmatically if a type is a struct or a class?

c# .net class struct types

edited Apr 15 '17 at 18:53
Cœur
19.3k 10 116 155

asked Dec 1 '09 at 16:44
Jader Dias
42.1k 134 385 588

1 Duplicate of stackoverflow.com/questions/1792586/… – bdukes Dec 1 '09 at 16:46

6 If this is a dup, then not from the given link. – Stefan Steinegger Dec 1 '09 at 16:57

5 Answers

Use `Type.IsValueType` :

Home

70

> Gets a value indicating whether the Type is a value type.

Use it either like this:

```
typeof(Foo).IsValueType
```

or at execution time like this:

```
fooInstance.GetType().IsValueType
```

Conversely there is also a `Type.IsClass` property (which should have been called `IsReferenceType` in my opinion but no matter) which may or may not be more appropriate for your uses based on what you are testing for.

Code always seems to read better without boolean negations, so use whichever helps the readability of your code.

As Stefan points out below, in order to properly identify *structs* you must be careful to avoid false positives when it comes to `enums` . An `enum` is a value type so the `IsValueType` property will return `true` for `enums` as well as `structs` .

So if you truly are looking for `structs` and not just value types in general you will need to do this:

```
Type fooType = fooInstance.GetType();
Boolean isStruct = fooType.IsValueType && !fooType.IsEnum;
```

edited Dec 3 '09 at 0:01

answered Dec 1 '09 at 16:46

Andrew Hare
281k 54 585 604

7 A primitive type is also a value type. – Stefan Steinegger Dec 1 '09 at 16:55

1 @Stephan - It is true that all C# primitives happen to be value types but that does not mean that all value types are therefore C# primitives. `System.Guid` and `System.DateTime` are both value types but are not language primitives. – Andrew Hare Dec 1 '09 at 16:57

1 To expand my point, the term "primitive" is special and really only is reserve for certain types that have overridden the `IsPrimitiveImpl` method from `System.Type` . There is nothing stopping Microsoft from implementing a new primitive that happens to be a reference type. There is nothing about a primitive that necessitates that it must also be a value type. – Andrew Hare Dec 1 '09 at 17:10

1 @Andrew: I see, primitive types are actually defined as special case of structs, so I'm wrong. I added a note to my answer. I apologize for the troubles. – Stefan Steinegger Dec 1 '09 at 18:12

3 Correct, enums and structs are the two value types that C# supports. A helpful way to remember this is that a struct is a kind of value type, not the other way around. – Andrew Hare Dec 1 '09 at 18:52

4

Extension method. It returns `true` for anything defined as a `struct` in my code but not for things like `int` which although they are technically structs are not for my purposes.

I needed to know when a type may have child fields or properties but was defined as a `struct` and not a `class` . Because when you alter a `struct` it just alters a copy, and then you have to set the original back to the altered copy to make the changes "stick".

```
public static bool IsStruct(this Type source)
{
    return source.IsValueType && !source.IsPrimitive && !source.IsEnum
}
```

edited Apr 12 '17 at 21:35

answered Apr 12 '17 at 21:26

toddmo
9,739 8 63 76

0

For every value type, there is a corresponding auto-generated class type which derives from `System.ValueType` , which in turn derives from `System.Object` . Note that value types themselves do not derive from anything, but are *implicitly convertible* to that class type, and instances of that class type may be explicitly converted to the value type.

Consider:

```
public static int GetSomething<T>(T enumerator) where T : IE
{
    T enumerator2 = enumerator;
    enumerator.MoveNext();
    enumerator2.MoveNext();
    return enumerator2.Current;
}
```

Calling this routine on a variable of type `List<int>.Enumerator` will yield very different behavior from calling it on a variable of type `IEnumerator<int>` which happens to have an instance of `List<int>.Enumerator` stored within it. Even though a variable of type `List<int>.Enumerator` is a value type, an instance of `List<int>.Enumerator` stored in a variable of type `IEnumerator<int>` will behave as a class type.

answered Jan 26 '12 at 17:08

supercat

58k 3 117 156

2 How is this answering the question? – nawfal Apr 19 '13 at 21:53

27

```
Type type = typeof(Foo);

bool isStruct = type.IsValueType && !type.IsPrimitive;
bool isClass = type.IsClass;
```

It could still be: a primitive type or an interface.

Edit: There is a lot of discussion about the definition of a struct. A struct and a value type are actually the same, so `IsValueType` is the correct answer. I usually had to know whether a type is a *user defined struct*, this means a type which is implemented using the keyword `struct` and not a primitive type. So I keep my answer for everyone who has the same problem then me.

Edit 2: According to the C# Reference, enums are not structs, while any other value type is. Therefore, the correct answer how to determine if a type is a struct is:

```
bool isStruct = type.IsValueType && !type.IsEnum;
```

IMHO, the definition of a struct is more confusing then logical. I actually doubt that this definition is of any relevance in praxis.

edited Dec 3 '09 at 9:23

answered Dec 1 '09 at 16:54

Stefan Steinegger
54.7k 14 111 178

The primitive type thing has been discussed to death in the comments of my answer :) You don't have to worry about an interface because and interface type instance will return `false` for both `IsClass` and `IsValueType` . Also any type that implements the interface will return its true type regardless of what the reference to the type is typed as an interface or not. – Andrew Hare Dec 1 '09 at 17:26

1 @Stefan: Are you saying that primitive types are precluded from being structs? If so, you're incorrect. For example, section 11 of the C# spec says "the simple types provided by C#, such as `int` , `double` , and `bool` , are in fact all `struct` types". – LukeH Dec 1 '09 at 17:27

1 @Luke: Yes, this is indeed the definition of primitive types. I always had to know if a type is a *user defined struct*, excluding primitive types, which is normally simply referenced to as "struct". But you are right, strictly speaking primitive types are also structs. – Stefan Steinegger Dec 1 '09 at 17:57

`type.IsValueType && !type.IsPrimitive` will not determine if the struct is a user-defined struct created using the keyword `struct` . It will return false positives for example for `decimal` , `int?` , or any enum. – thepirat000 Apr 23 '18 at 17:50

@thepirat000: Yes. There is no concept of "user-defined struct" in C#. Decimals and nullables are structs like any other. Of course you can exclude them explicitly, if it makes sense. There is a related (but different!) question with a better answer here: stackoverflow.com/questions/863881/ … – Stefan Steinegger Apr 24 '18 at 12:36

0

Try the following

```
bool IsStruct(Type t) {
  return t.IsValueType;
}
```

answered Dec 1 '09 at 16:46

JaredPar
583k 121 1077 1354

1 See answer from @Stefan: `type.IsValueType && !type.IsEnum` – gap Aug 31 '12 at 17:10