How can I get the data type of a variable in C#?

73

7

How can I find out what data type some variable is holding? (e.g. int, string, char, etc.)

I have something like this now:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Testing
{
    class Program
    {
        static void Main()
        {
            Person someone = new Person();
            someone.setName(22);
            int n = someone.getName();
            Console.WriteLine(n.typeOf());
        }
    }

    class Person
    {
        public int name;

        public void setName(int name)
        {
            this.name = name;
        }

        public int getName()
        {
            return this.name;
        }
```

edited Oct 16 '14 at 13:33

APerson
5,205 4 29 46

asked Jul 24 '12 at 15:21

Limeni
2,046 6 22 33

6 You've already defined the type - `int` – Jamiec Jul 24 '12 at 15:23

This is unclear what you mean by "find out the data type". Normally, the answer is "you just look at the class member signature and the type is stated there in an explicit way". Is your intention to inspect class members in the run time? – Wiktor Zychla Jul 24 '12 at 15:25

1 Out of subject, but what you wrote here is better written like this with C# : `class Person { public string Name { get; set; } }` or `class Person { private string m_Name; public string Name { get {return m_Name;} set { m_Name = value; } }` . Read the documentation about Properties – Steve B Jul 24 '12 at 15:26

1 Jamiec is right. Static typing means the declaration sets your type forever. Your variable n can only be of the type you declared, or an inherited type. In your specific case you chose to display an int, that's a type you can't inherit from, so n can only be an int. – Ksempac Jul 24 '12 at 15:27

9 Answers

95

Other answers offer good help with this question, but there is an important and subtle issue that none of them addresses directly. There are two ways of considering type in C#: *static type* and *run-time type*.

Static type is the type of a variable in your source code. It is therefore a compile-time concept. This is the type that you see in a tooltip when you hover over a variable or property in your development environment.

Run-time type is the type of an object in memory. It is therefore a run-time concept. This is the type returned by the `GetType()` method.

An object's run-time type is frequently different from the static type of the variable, property, or method that holds or returns it. For example, you can have code like this:

```
object o = "Some string";
```

The static type of the variable is `object` , but at run time, the type of the variable's *referent* is `string` . Therefore, the next line will print "System.String" to the console:

But, if you hover over the variable `o` in your development environment, you'll see the type `System.Object` (or the equivalent `object` keyword).

For value-type variables, such as `int`, `double`, `System.Guid`, you know that the run-time type will always be the same as the static type, because value types cannot serve as the base class for another type; the value type is guaranteed to be the most-derived type in its inheritance chain. This is also true for sealed reference types: if the static type is a sealed reference type, the run-time value must either be an instance of that type or `null`.

Conversely, if the static type of the variable is an abstract type, then it is guaranteed that the static type and the runtime type will be different.

To illustrate that in code:

```
// int is a value type
int i = 0;
// Prints True for any value of i
Console.WriteLine(i.GetType() == typeof(int));

// string is a sealed reference type
string s = "Foo";
// Prints True for any value of s
Console.WriteLine(s == null || s.GetType() == typeof(string));

// object is an unsealed reference type
object o = new FileInfo("C:\\f.txt");
// Prints False, but could be true for some values of o
Console.WriteLine(o == null || o.GetType() == typeof(object));

// FileSystemInfo is an abstract type
FileSystemInfo fsi = new DirectoryInfo("C:\\");
// Prints False for all non-null values of fsi
Console.WriteLine(fsi == null || fsi.GetType() == typeof(FileSystemInfo));
```

answered Jul 24 '12 at 16:02

phoog

36k 2 65 92

`typeof(int)` it will return int, but isn't checking the variable 'a' and showing you the type of 'a'.. You could say that you don't need to show the static type of 'a', that may be so, but the fact is that it isn't showing it. So I don't see how the use of typeof here is useful. – barlop Dec 24 '15 at 18:34

@barlop To go back one step, how is knowing the static type at runtime useful? – phoog Dec 24 '15 at 18:55

3 @barlop you could do this to let type inference take care of it for you: `Type GetStaticType < T > (T x) { return typeof(T); }` – phoog Dec 24 '15 at 19:35

16

Generally speaking, you'll hardly ever need to do type comparisons unless you're doing something with reflection or interfaces. Nonetheless:

If you know the type you want to compare it with, use the `is` or `as` operators:

```
if( unknownObject is TypeIKnow ) { // run code here
```

The `as` operator performs a cast that returns null if it fails rather than an exception:

```
TypeIKnow typed = unknownObject as TypeIKnow;
```

If you don't know the type and just want runtime type information, use the .GetType() method:

```
Type typeInformation = unknownObject.GetType();
```

In newer versions of C#, you can use the `is` operator to declare a variable without needing to use `as`:

```
if( unknownObject is TypeIKnow knownObject ) {
    knownObject.SomeMember();
}
```

Previously you would have to do this:

```
TypeIKnow knownObject;
```

Dai
76.4k 16 126 214

10

Its Very simple

```
variable.GetType().Name
```

it will return your datatype of your variable

answered Mar 28 '14 at 18:43

Sagar Chavan
258 3 15

4

Use the GetType() method

http://msdn.microsoft.com/en-us/library/system.object.gettype.aspx

answered Jul 24 '12 at 15:23

Stephen Oberauer
3,465 5 39 68

4

Just hold cursor over member you interested in, and see tooltip - it will show memeber's type:

```
Person someone = new Person();
someone.setName(22);
int n = someone.getName();
```

int Person.getName()

`GetType()` method

1

```
int n=34;
Console.WriteLine(n.GetType());
string name="Smome";
Console.WriteLine(name.GetType());
```

answered Jul 24 '12 at 15:24

Shyju
151k 89 342 450

1

PHP and C# are related syntactically but quite different, whilst I could answer the question at face value (See this article http://msdn.microsoft.com/en-us/library/58918ffs(v=vs.71).aspx) I strongly advise you to get hold of a copy of CLR via C# (Third or second edition) by Jeffrey Richter and read it. Its the finest book related to programming I think I've ever read and would answer almost all of your type related questions and give you a very deep understanding of what's going on under the hood!

answered Jul 24 '12 at 15:29

bUKaneer
4,018 2 24 48

1

One option would be to use a helper extension method like follows:

```
public static class MyExtensions
{
    public static System.Type Type<T>(this T v)=>typeof(T);
}

var i=0;
console.WriteLine(i.Type().FullName);
```

check out one of the simple way to do this

0

```
// Read string from console
        string line = Console.ReadLine();
        int valueInt;
        float valueFloat;
        if (int.TryParse(line, out valueInt)) // Try to parse the string as an integer
        {
            Console.Write("This input is of type Integer.");
        }
        else if (float.TryParse(line, out valueFloat))
        {
            Console.Write("This input is of type Float.");
        }
        else
        {
            Console.WriteLine("This input is of type string.");
        }
```

answered Aug 9 '17 at 12:06

KiranSolkar

793 3 14 27