## typeof generic and casted type [duplicate]

Asked 5 years, 10 months ago Modified 5 years, 10 months ago Viewed 2k times



25

This question already has answers here:

<u>Type Checking: typeof, GetType, or is?</u> (16 answers)

Closed 5 years ago.



Let's say we have generic method:



```
public void GenericMethod<T>(T item)
{
   var typeOf = typeof(T);
   var getType = item.GetType();
}
```

And we are invoking it with the following parameters:

```
GenericMethod(1)
GenericMethod((object) 1)
```

The results are:

```
typeOf = System.Int32
getType = System.Int32
```

and

```
typeOf = System.Object
getType = System.Int32
```

Can someone explain me why typeof integer casted to object returns System. Object, but .GetType() returns System. Int32?

c# generics

Share

Improve this question

Follow

edited Feb 22, 2019 at 12:59

Uwe Keim

40.7k • 61 • 185 • 301

asked Feb 22, 2019 at 12:44

user3450929
327 • 2 • 10

- typeof takes a type name (which you specify at compile time), GetType gets the runtime type of an instance. - João Paulo Amorim Feb 22, 2019 at 12:47
- Also, if you find yourself doing any kind of type test inside a generic, ask yourself whether you've picked the right tool for the job. Because it may mean that you fail to work properly at runtime something that you "promised" at compile time you could do (by saying you could work for any type, subject to any generic type constraints on that type parameter) - Damien The Unbeliever Feb 22, 2019 at 12:49

## 5 Answers

Sorted by: Highest score (default) \$



typeof returns the **static (compile-time)** type of the generic parameter T.

**32** 

GetType returns the **dynamic (run-time)** type of the **value** contained in variable item.





The difference is easier to see if you make your method non-generic. Let's assume that B is a subtype of A:



```
public void NonGenericMethod(A item)
{
    var typeOf = typeof(A);
    var getType = item.GetType();
}
```

In that case, calling NonGenericMethod(new B()) would yield

```
Α
В
```

Recommended further reading:

Run-time type vs compile-time type in C#

Now, you might ask: Why did you use NonGenericMethod(A item) in your example instead of NonGenericMethod(B item)? That's a very good question! Consider the following (non-generic) example code:

```
public static void NonGenericMethod(A item)
{
   Console.WriteLine("Method A");
   var typeOf = typeof(A);
   var getType = item.GetType();
public static void NonGenericMethod(B item)
```

```
{
    Console.WriteLine("Method B");
    var typeOf = typeof(B);
    var getType = item.GetType();
}
```

What do you get when you call NonGenericMethod((A) new B()) (which is analogous to the argument (object) 1 in your example)?

```
Method A
A
B
```

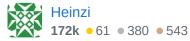
Why? Because overload resolution is done at compile-time, not at run-time. At *compile-time*, the type of the expression (A) new B() is A, just like the compile-time type of (object) 1 is object.

Recommended further reading:

When is the generic type resolved in c#?

Share edited Feb 22, 2019 at 13:22

answered Feb 22, 2019 at 12:47



Follow

Improve this answer

Although typeOf will get resolved before the first time a function is executed with a particular T, it may occur after the program has started execution. Unlike C++, C# makes it possible for a program to create an unbounded number of different types based upon input, even without using Reflection. In such cases, it may be impossible to produce a list of every type T with which a function could be invoked, and thus impossible to determine what T would be prior to program execution. – supercat Feb 22, 2019 at 19:08



In GenericMethod((object) 1), T will be object typeof reflects that.

2

But item.GetType(); is a virtual method and will execute at runtime on Int32.



Share Improve this answer Follow

answered Feb 22, 2019 at 12:49









The call to GetType gets resolved at runtime, while typeof is resolved at compile time. That is why it is giving different results. you can check here - When and where to use GetType() or typeof()?





1



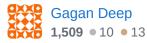
<u>This Tells me</u> Typeof gives you compile time type whereas GetType gives you Exact Run time type.





Share Improve this answer Follow

answered Feb 22, 2019 at 12:50









A lot is made clear when you leave out the type interference:

GenericMethod(1) is actually GenericMethod<int>(1).



0

GenericMethod((object) 1) is inferred as GenericMethod<object>((object) 1).



When you ask typeof(T), it returns the T you specified in the method call. You could also do GenericMethod < object > ("a"), which will return object on typeof(T).



GetType returns the actual runtime type of the instance provided.

Share Improve this answer Follow

answered Feb 22, 2019 at 19:13

