In C#, why is String a reference type that behaves like a value type?

A String is a reference type even though it has most of the characteristics of a value type such as being immutable and having == overloaded to compare the text rather than making sure they reference the same object.

Why isn't string just a value type then?

c# string clr value-type reference-type

edited Jun 25 '10 at 15:30
codekaizen
14.4k 3 33 70

asked Mar 12 '09 at 0:26
Davy8
12.1k 8 58 127

9 Answers

Strings aren't value types since they can be huge, and need to be stored on the heap. Value types are (in all implementations of the CLR as of yet) stored on the stack. Stack allocating strings would break all sorts of things: the stack is only 1MB, you'd have to box each string, incurring a copy penalty, you couldn't intern strings, and memory usage would balloon, etc...

(Edit: Added clarification about value type storage being an implementation detail, which leads to this situation where we have a type with value sematics not inheriting from System.ValueType. Thanks Ben.)

edited Aug 1 '11 at 18:54

answered Mar 12 '09 at 0:28
codekaizen
14.4k 3 33 70

1 I understand that, but why isn't it just a value type then? – Davy8 Mar 12 '09 at 0:29

1 I meant edit "or" make a new one btw. Waiting longer before marking accepted – Davy8 Mar 12 '09 at 0:40

30 I'm nitpicking here, but only because it gives me an opportunity to link to an blog post relevant to the question: value types are not necessarily stored on the stack. It's most often true in ms.net, but not at all specified by the CLI specification. The main difference between value and reference types is, that reference types follow copy-by-value semantics. See blogs.msdn.com/ericlippert/archive/2009/04/27/… and blogs.msdn.com/ericlippert/archive/2009/05/04/… – Ben Schwehn Jun 9 '09 at 22:36

3 @Qwertie: `String` is not variable size. When you add to it, you are actually creating another `String` object, allocating new memory for it. – codekaizen Jun 25 '10 at 15:49

1 That said, a string could, in theory, have been a value type (a struct), but the "value" would have been nothing more than a reference to the string. The .NET designers naturally decided to cut out the middleman (struct handling was inefficient in .NET 1.0, and it was natural to follow Java, in which strings were already defined as a reference, rather than primitive, type. Plus, if string were a value type then converting it to object would require it to be boxed, a needless inefficiency). – Qwertie Jun 25 '10 at 15:50

show **13** more comments

It is not a value type because performance (space and time!) would be terrible if it were a value type and its value had to be copied every time it were passed to and returned from methods, etc.

It has value semantics to keep the world sane. Can you imagine how difficult it would be to code if

```
string s = "hello";
string t = "hello";
bool b = (s == t);
```

set `b` to be `false` ? Imagine how difficult coding just about any application would be.

edited Mar 12 '09 at 0:37

answered Mar 12 '09 at 0:32

Jason
121k 12 199 365

19 Thats how java works, you have to use s.equals(t). – Matt Briggs Mar 12 '09 at 2:40

10 Java is not known for being pithy. – Jason Mar 12 '09 at 12:51

@Matt: exactly. When I switched over to C# this was kind of confusing, since I always used (an do still sometimes) .equals(..) for comparing strings while my teammates just used "==". I never understood why they didn't leave the "==" to compare the references, although if you think, 90% of the time you'll probably want to compare the content not the references for strings. – Juri Aug 6 '09 at 14:34

3 @Juri: Actually i think it's never desirable to check the references, since sometimes `new String("foo");` and another `new String("foo")` can evaluate in the same reference, which kind of is not what you would expect a `new` operator to do. (Or can you tell me a case where I would want to compare the references?) – Michael Sep 7 '10 at 5:59

1 ...having strings be a value type of of that style rather than being a class type would mean the default value of a `string` could behave as an empty string (as it was in pre-.net systems) rather than as a null reference. Actually, my own preference would be to have a value type `String` which contained a reference-type `NullableString`, with the former having a default value equivalent to `String.Empty` and the latter having a default of `null`, and with special boxing/unboxing rules (such that boxing a default-valued `NullableString` would yield a reference to `String.Empty`). – supercat Nov 12 '12 at 21:25

show **2** more comments

Not only strings are immutable reference types. **Multi-cast delegates too.** That is why it is safe to write

```
protected void OnMyEventHandler()
{
   delegate handler = this.MyEventHandler;
   if (null != handler)
   {
     handler(this, new EventArgs());
   }
}
```

I suppose that strings are immutable because this is the most safe method to work with them and allocate memory. Why they are not Value types? Previous authors are right about stack size etc. I would also add that making strings a reference types allow to save on assembly size when you use the same constant string in the program. If you define

```
string s1 = "my string";
//some code here
string s2 = "my string";
```

Chances are that both instances of "my string" constant will be allocated in your assembly only once.

If you would like to manage strings like usual reference type, put the string inside a new StringBuilder(string s). Or use MemoryStreams.

If you are to create a library, where you expect a huge strings to be passed in your functions, either define a parameter as a StringBuilder or as a Stream.

edited Apr 6 '11 at 10:00

gehho
4,371 13 33

answered Jun 23 '09 at 10:17

Bogdan_Ch
2,006 2 10 25

There are plenty of examples of immutable reference-types. And re the string example, that is indeed pretty-much guaranteed under the current implementations - *technically* it is is per *module* (not per-assembly) - but that is almost always the same thing... – Marc Gravell ♦ Jun 23 '09 at 10:21

1 Re the last point: StringBuilder doesn't help if you trying to *pass* a large string (since it is actually implemented as a string anyway) - StringBuilder is useful for **manipulating** a string multiple times. – Marc Gravell ♦ Jun 23 '09 at 10:23

Did u mean delegate handler, not hadler? (sorry to be picky .. but it's very close to a (not common) surname i know....) – Pure.Krome Jun 23 '09 at 10:24

Also, the way strings are implemented (different for each platform) and when you start stitching them together. Like using a `StringBuilder`. It allocats a buffer for you to copy into, once you reach the end, it allocates even more memory for you, in the hopes that if you do a large concatenation performance won't be hindered.

Maybe Jon Skeet can help up out here?

answered Mar 12 '09 at 0:34

Chris
3,188 6 28 48

It is mainly a performance issue.

Having strings behave LIKE value type helps when writing code, but having it BE a value type would make a huge performance hit.

For an in-depth look, take a peek at a nice article on strings in the .net framework.

answered Mar 12 '09 at 2:35

Denis Troller
5,659 8 29

Actually strings have very few resemblances to value types. For starters, not all value types are immutable, you can change the value of an Int32 all you want and it it would still be the same address on the stack.

Strings are immutable for a very good reason, it has nothing to do with it being a reference type, but has a lot to do with memory management. It's just more efficient to create a new object when string size changes than to shift things around on the managed heap. I think you're mixing together value/reference types and immutable objects concepts.

As far as "==": Like you said "==" is an operator overload, and again it was implemented for a very good reason to make framework more useful when working with strings.

edited Nov 11 '12 at 15:07

codesparkle
5,677 14 32

answered Mar 12 '09 at 1:02

WebMatrix
766 2 6 13

I realize that value types aren't by definition immutable, but most best practice seems to suggest that they should be when creating your own. I said characteristics, not properties of value types, which to me means that often value types exhibit these, but not necessarily by definition – Davy8 Mar 12 '09 at 12:59

Good information, but I think a misinterpretation of the question – Davy8 Mar 12 '09 at 12:59

4 @WebMatrix, @Davy8: The primitive types (int, double, bool, ...) are immutable. – Jason Mar 12 '09 at 13:11

1 @Jason, I thought immutable term mostly apply to objects (reference types) which can not change after initialization, like strings when strings value changes, internally a new instance of a string is created, and original object remains unchanged. How does this apply to value types? – WebMatrix Mar 12 '09 at 14:23

6 Somehow, in "int n = 4; n = 9;", it's not that your int variable is "immutable", in the sense of "constant"; it's that the value 4 is immutable, it doesn't change to 9. Your int variable "n" first has a value of 4 and then a different value, 9; but the values themselves are immutable. Frankly, to me this is very close to wtf. – Daniel Daranas Jun 23 '09 at 10:36

show **2** more comments

Isn't just as simple as Strings are made up of characters arrays. I look at strings as character arrays[]. Therefore they are on the heap because the reference memory location is stored on the stack and points to the beginning of the array's memory location on the heap. The string size is not known before it is allocated ...perfect for the heap.

That is why a string is really immutable because when you change it even if it is of the same size the compiler doesn't know that and has to allocate a new array and assign characters to the positions in the array. It makes sense if you think of strings as a way that languages protect you from having to allocate memory on the fly (read C like programming)

edited Sep 10 '13 at 10:03

Devraj Gadhavi
972 1 7 23

answered Jun 23 '12 at 14:48

BionicCyborg
29 1

1 "string size is not known before it is allocated " - this is incorrect in the CLR. – codekaizen Jan 2 '13 at 9:04

The distinction between reference types and value types are basically a performance tradeoff in the design of the language. Reference types have some overhead on construction and destruction and garbage collection, because they are created on the heap. Value types on the other hand have overhead on method calls (if the data size is larger than a pointer), because the whole object is copied rather than just a pointer. Because strings can be (and typically are) much larger than the size of a pointer, they are designed as reference types. Also, as Servy pointed out, the size of a value type must be known at compile time, which is not always the case for strings.

The question of mutability is a separate issue. Both reference types and value types can be either mutable or immutable. Value types are typically immutable though, since the semantics for mutable value types can be confusing.

Reference types are generally mutable, but can be designed as immutable if it makes sense. Strings are defined as immutable because it makes certain optimizations possible. For example, if the same string literal occurs multiple times in the same program (which is quite common), the compiler can reuse the same object.

So why is "==" overloaded to compare strings by text? Because it is the most useful semantics. If two strings are equal by text, they may or may not be the same object reference due to the optimizations. So comparing references are pretty useless, while comparing text are almost always what you want.

Speaking more generally, Strings has what is termed **value semantics**. This is a more general concept than value types, which is a C# specific implementation detail. Value types have value semantics, but reference types may also have value semantics. When a type have value semantics, you can't really tell if the underlying implementation is a reference type or value type, so you can consider that an implementation detail.

edited Nov 8 '13 at 8:45

answered Nov 7 '13 at 14:16

JacquesB
19.3k 7 33 57

The distinction between value types and reference types isn't really about performance at all. It's about whether a variable contains an actual object or a reference to an object. A string could never possibly be a value type because the size of a string is variable; it would need to be constant to be a value type; performance has almost nothing to do with it. Reference types are also not expensive to create at all. – Servy Nov 7 '13 at 16:06

@Sevy: The size of a string *is* constant. – JacquesB Nov 7 '13 at 16:09

Because it just contains a reference to a character array, which is of variable size. Having a value type who's only real "value" was a reference type would just be all the more confusing, as it would still have reference semantics for all intensive purposes. – Servy Nov 7 '13 at 16:11

@Sevy: The size of an array is constant. – JacquesB Nov 7 '13 at 16:35

The size of a reference to an array is constant. The size of an array itself is dependent on the number of items in the array and the size of the type the array holds. – Servy Nov 7 '13 at 16:37

show **2** more comments

How can you tell `string` is a reference type? I'm not sure that it matters how it is implemented. Strings in C# are immutable precisely so that you don't have to worry about this issue.

answered Mar 12 '09 at 3:17
please delete me

It's a reference type (I believe) because it doesn't derives from System.ValueType From MSDN Remarks on System.ValueType: Data types are separated into value types and reference types. Value types are either stack-allocated or allocated inline in a structure. Reference types are heap-allocated. – Davy8 Mar 12 '09 at 13:09

Both reference and value types are derived from the ultimate base class Object. In cases where it is necessary for a value type to behave like an object, a wrapper that makes the value type look like a reference object is allocated on the heap, and the value type's value is copied into it. – Davy8 Mar 12 '09 at 13:10

The wrapper is marked so the system knows that it contains a value type. This process is known as boxing, and the reverse process is known as unboxing. Boxing and unboxing allow any type to be treated as an object. (In hind site, probably should've just linked to the article.) – Davy8 Mar 12 '09 at 13:11

Not the answer you're looking for? Browse other questions tagged c# string clr

value-type reference-type **or ask your own question.**