

Re: Boxing and Unboxing of Value-Types when passed to a method call

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2005-01/5325.html>

From: ALI-R (*newbie_at_microsoft.com*)

Date: 01/25/05

Date: Mon, 24 Jan 2005 17:12:38 -0800

Excellent monitoring ,Excellent Cooperation ,Hopefully this thread could help those people (like me;-)) who still have the confusion about how the parameter passing is handled in C# and Generally what's the difference between ValueTypes VS ReferenceTypes in C#.

I'd like to point to some nice sentences from I extracted from Skeet's excellent article:

1)A reference type is a type which has as its value a reference to the appropriate data rather than the data itself.
Remember though that the value of a reference type variable is always a reference not the actual data

2)While reference types have a layer of indirection between the variable and the real data, value types don't. Variables of a value type directly contain the data

3)Saying that "value types go on the stack, reference types go on the heap" is an incorrect oversimplification (As Jon Skeet - <skeet@pobox.com> said here and in his excellent article). It depends on the context in which it is declared:

a.. Each *local variable (ie one declared in a method) is stored on the stack. That includes reference type variables - the variable itself is on the stack, but remember that the value of a reference type variable is only a reference (or null), not the object itself (Object is on the heap).

*Method parameters count as local variables too, but if they are declared with the ref modifier, they don't get their own slot, but share a slot with the variable used in the calling code.

b.. Instance variables for a reference type are always on the heap. That's where the object itself "lives".

c.. Instance variables for a value type are stored in the same context as the variable that declares the value type. The memory slot for the instance effectively contains the slots for each field within the instance. That means (given the previous two points) that a struct variable declared within a method will always be on the stack, whereas a struct variable which is an

instance field of a class will be on the heap.

Every static variable is stored on the heap, regardless of whether it's declared within a reference type or a value type. There is only one slot in total no matter how many instances are created.

4) Saying that "In method calls objects are passed by reference by default" is not true

5) Reference parameters don't pass the values of the variables used in the function member invocation – they use the variables themselves. Rather than creating a new storage location for the variable in the function member declaration, the same storage location is used, so the value of the variable in the function member and the value of the reference parameter will always be the same

6) What is the difference between passing a value object by reference and a reference object by value? Consider the following code?

Read skeet's article at <http://www.pobox.com/~skeet/csharp/memory.html>

Hope this helps,
Reza Alirezai

"Bruce Wood" <brucewood@canada.com> wrote in message
news:1106612873.561786.326740@z14g2000cwz.googlegroups.com...
> Ah, I see what you mean. Well, that's more a matter of point of view.
> It depends upon whether you consider the name "b" to refer to "the
> thing pointed to be the reference that was passed into my method", in
> which case "the reference" that is on the stack has no name, or you
> consider the name "b" to refer to "a reference to an int that is
> magically dereferenced whenever the name 'b' is mentioned". Being an
> old C hack, I tend to think of it in the latter way, but I admit that
> the former is equally valid. This will change the interpretation of
> some of the things I've said in other posts. My apologies for any
> confusion.
>
> So, yes, you could see "b" as just a synonym for "a" and therefore it
> has no special stack space of its own, and the reference on the stack
> that was passed into "myMethod" as an artifact of the language that has
> no name.
>