Can object constructor return a null?

Asked 15 years, 11 months ago Modified 6 years, 10 months ago Viewed 36k times

40

We have taken over some .NET 1.1 Windows Service code that spawns threads to read messages off a queue (SeeBeyond eGate JMS queue, but that is not important) and in turn spawns threads to process the message in the target application service. We are continually encountering logic and design decisions that is puzzling us to no end. Here is one example, where the message (lsMessage) has been retrieved from the queue and ready for processing

```
if(lsMessage != null)
{
    // Initialize a new thread class instance, pass in message
    WorkerThread worker = new WorkerThread(lsMessage);

Process:
    // Start a new thread to process the message
    Thread targetWorker = new Thread(new ThreadStart(worker.ProcessMessage));
    if(targetWorker != null)
    {
        targetWorker.Priority = ThreadPriority.Highest;
        targetWorker.Name = "Worker " + queueKey.ToString();
        targetWorker.Start();

        // wait for worker thread to join back in specified period
        bool isFinished = targetWorker.Join(SYNC_THREAD_TIMEOUT);

        string message = worker.replyMsg;

        if ( !isFinished )  // BF is timeout
        {
            targetWorker.Abort();

            // [obscure developer name] 25/10/2004: calling Join() to wait for
thread to terminate.
            // for EAI listener threads problem, ensure no new thread is
started
            // before the old one ends
            targetWorker.Join();

            // prepare reply message
            string errorMsg = string.Format("EAIMsg {0}: BF is timeout. Send
sync message back to caller.", worker.messageKey);
            log.Debug(errorMsg);

            message = worker.GenErrorCode(message, errorMsg);
        }

        // Commit message
        MQ.ReceiverCommit(queueKey, worker.messageKey, false);

        // Send back the response to the caller
        MQ.RespondSend(queueKey, message);
    }
```

```
    else
    {
        log.Debug(string.Format("Fail to start worker thread to process sync
message. Thread returned is null. Sleep for {0} milliseconds.",
LIMIT_RESOURCE_SLEEP));
        Thread.Sleep(LIMIT_RESOURCE_SLEEP);
        goto Process;
    }
}
```

Please ignore the *use of label and goto* for the moment; that is not the question. Our bewilderment is the **check whether the Thread object is null right after instantiation**. The else statement below seems to suggest the previous developers have encountered situations like this before. Of course, the original developers are long gone. So we would like to know, can the CLR really instantiate an object after the call to the constructor and return a null? We have no knowledge of such a possibility.

`.net` `multithreading`

Share
Improve this question
Follow

edited Feb 3, 2010 at 8:35

Andreas Bonini
44.7k ● 31 ● 124 ● 158

asked Jan 13, 2009 at 8:31

icelava
9,857 ● 7 ● 53 ● 75

2 Looks the the result of a refactoring or code change to me. Maybe the constructing line was something else such as `GetThread()`. – usr Dec 12, 2016 at 23:11

5 Answers

Sorted by: Highest score (default)

45

In my opinion, what the `else` statement suggests is that the previous developers didn't know their C#. A constructor always returns a constructed object or throws an exception.

In the very old times, C++ constructors could return `null`, so maybe the problem comes from that. This is no longer true in C++ either, at least for the default `new` operator.

Share
Improve this answer
Follow

edited Dec 12, 2016 at 22:17

answered Jan 13, 2009 at 8:39

Gorpik
11k ● 5 ● 38 ● 57

3 That may be a possibility; the original developers may have some C++ background. I have seen C++ programmers habitually trying to do "funny" stuff in .NET.... – icelava Jan 13, 2009 at 8:42

2 By the way in Objective-C constructor can returns null (nil) – fnc12 Jul 21, 2014 at 7:27

1 Swapping the first two sentences, or otherwise making the second sentence more prominent, would make the answer easier to parse (more direct) for use by people who've only read the "question title", not the entire question. – Jirka Hanika Jan 2, 2018 at 11:52

32

Edit: for clarification there is an *insane edge case* where you can get `null` from a class constructor, but frankly I don't think any real code should *ever* expect to deal with this level of crazy: What's the strangest corner case you've seen in C# or .NET? . To all *normal* intents : it won't happen.

No, you can't get null from a **class** constructor (`Thread` is a class). The only case I know of where a constructor can (seem to) return `null` is `Nullable<T>` - i.e.

```
object foo = new int?(); // this is null
```

This is a slightly bigger problem with generics:

```
static void Oops<T>() where T : new() {
    T t = new T();
    if (t == null) throw new InvalidOperationException();
}

static void Main() {
    Oops<int?>();
}
```

(of course, there are ways of checking/handling that scenario, such as `: class`)

Other than that, a constructor will always either return an object (or initialize a struct), or throw an exception.

Share

Improve this answer

Follow

edited May 23, 2017 at 11:53

Community Bot

1 • 1

answered Jan 13, 2009 at 8:36

Marc Gravell

1.1m • 273 • 2.6k • 3k

3 True enough - but I thought it was worth mentioning as an edge-case ;-p – Marc Gravell Jan 13, 2009 at 8:40

And it's *almost* always a reference to a newly created object, too. The only exception to that that I know of is a string corner case. – Jon Skeet Jan 13, 2009 at 8:45

what if in the last line of constructor you pass "this" by reference, in a method, and that method sets that reference to "null"? – serhio Jan 23, 2013 at 11:15

1 @serhio in the case of a `class`, you get "Error 1 Cannot pass '<this>' as a ref or out argument because it is read-only". In the case of a `struct`, the `ref` is fine, but you can't set a `struct` to `null` (except for `Nullable<T>`, and you don't control the constructor for `Nullable<T>`) – Marc Gravell Jan 23, 2013 at 11:37

@serhio that said: there *is* an evil case when you **can** get a `null` from a constructor: stackoverflow.com/questions/194484/… - but very unlikely to see that in real code. – Marc Gravell Jan 23, 2013 at 11:40

3

As `core` mentions, operator overloading can make it appear that a constructor returned `null`, when that's not what really happened. The authors of the article `core` found say they haven't seen it used, but it actually is used in a very popular product: Unity.

This will compile and log a message at run time:

```
using UnityEngine;

public class Test:MonoBehaviour
{
    void Start()
    {
        AudioSource as = new AudioSource();

        if (as == null)
        {
            Debug.Log("Looks null to me.");
        }
    }
}
```

Now, the fault here is mine, because one should *not* call the `AudioSource` constructor directly. But one *should* know that the `==` operator is overloaded at the root of the inheritance tree for the objects Unity can reference. Here's what the Unity manual says about `UnityEngine.Object`'s `==` operator:

> Be careful when comparing with null.
>
> e.g.
>
> ```
> GameObject go = new GameObject();
> Debug.Log (go == null); // false
>
> Object obj = new Object();
> Debug.Log (obj == null); // true
> ```
>
> Instatiating a GameObject adds it to the scene so it's completely initialized (!destroyed). Instantiating a simple UnityEngine.Object has no such

> semantics, so the(*sic*) it stays in the 'destroyed' state which compares true to null.

While instantiating a `GameObject` initializes it, instantiating an `AudioSource` object doesn't, so the comparison with `null` returns `true`.

This unusual idiom is made even more stealthy by virtue of the fact that attempts to reference properties of the uninitialized `AudioSource` object will throw null-reference exceptions, which I initially misinterpreted as meaning the object reference was `null`, not the property.

Others have answered the OP's question, but I wanted to add this answer because the OP's code might actually make sense if the `Thread` class therein isn't the one we would expect it to be, just as `Object` (and its descendants) isn't quite what you might expect it to be in a Unity script (that is, it is actually `UnityEngine.Object`, rather than `System.Object`, which gets you the overloaded `==` operator that so confused me).

Share Improve this answer Follow

answered Feb 8, 2018 at 16:30

Stevens Miller
1,480 ● 10 ● 27

Wow, not that it's relevant, but that looks like a pretty egregious abuse of the == operator.
– Frank Hagenson Dec 15, 2019 at 20:19

1 Unity maintains that this is a side-effect of it having originally been developed in C++, with its C# portion coming later. A lot of that original C++ code remains in Unity, so the dynamics of object destruction are not what you would ordinarily expect in a managed language.
– Stevens Miller Dec 18, 2019 at 13:09

2

NO! that null check is redundant. Lot of C++ devs who moved to C# have this habit of a null check and I guess it is the same here.

The only thing is you should check the documentation to see if the constructor can throw any exception. In your case refer to http://msdn.microsoft.com/en-us/library/xx3ezzs2.aspx and as mentioned the constructor will always return a valid obj.

Share Improve this answer Follow

answered Jan 13, 2009 at 8:41

Sesh
6,174 ● 4 ● 33 ● 40

You can make it appear like an object ctor returns null:

2

http://seattlesoftware.wordpress.com/2008/03/05/returning-null-from-a-class-constructor/

Search for "Another pattern which I haven't seen used allows an invalid object to emulate a null reference" and read from there.

Share Improve this answer Follow

answered Mar 17, 2009 at 3:28

core

33k ● 45 ● 140 ● 195