How to properly cast objects created through reflection

Asked 16 years, 3 months ago Modified 13 years, 11 months ago Viewed 13k times

10

I'm trying to wrap my head around reflection, so I decided to add plugin capability to a program that I'm writing. The only way to understand a concept is to get your fingers dirty and write the code, so I went the route of creating a simple interface library consisting of the IPlugin and IHost interfaces, a plugin implementation library of classes that implement IPlugin, and a simple console project that instantiates the IHost implementation class that does simple work with the plugin objects.

Using reflection, I wanted to iterate through the types contained inside my plugin implementation dll and create instances of types. I was able to sucessfully instantiate classes with this code, but I could not cast the created object to the interface.

I tried this code but I couldn't cast object o as I expected. I stepped through the process with the debugger and the proper constructor was called. Quickwatching object o showed me that it had the fields and properties that I expected to see in the implementation class.

```
loop through assemblies
  loop through types in assembly
    // Filter out unwanted types
    if (!type.IsClass || type.IsNotPublic || type.IsAbstract )
      continue;
    // This successfully created the right object
    object o = Activator.CreateInstance(type);
    // This threw an Invalid Cast Exception or returned null for an "as" cast
    // even though the object implemented IPlugin
    IPlugin i = (IPlugin) o;
```

I made the code work with this.

```
using System.Runtime.Remoting;
ObjectHandle oh = Activator.CreateInstance(assembly.FullName, type.FullName);
// This worked as I intended
IPlugin i = (IPlugin) oh.Unwrap();
i.DoStuff();
```

Here are my questions:

1. Activator.CreateInstance(Type t) returns an object, but I couldn't cast the object to an interface that the object implemented. Why?
2. Should I have been using a different overload of CreateInstance()?

3. What are the reflection related tips and tricks?

4. Is there some crucial part of reflection that I'm just not getting?

c# .net reflection

Share
Improve this question
Follow

edited Aug 28, 2008 at 3:12

asked Aug 28, 2008 at 2:50

Ben Robbins
2,630 ● 2 ● 25 ● 26

7 Answers

Sorted by: Highest score (default)

4

I'm just guessing here because from your code it's not obvious where do you have definition of IPlugin interface but if you can't cast in your host application then you are probably having IPlugin interface in your host assembly and then at the same time in your plugin assembly. This won't work.

The easiest thing is to make this work is to have IPlugin interface marked as public in your host assembly and then have your Plugin assembly **reference host application assembly**, so both assemblies have access to **the very same interface**.

Share
Improve this answer
Follow

edited Aug 28, 2008 at 3:54

answered Aug 28, 2008 at 3:46

lubos hasko
25k ● 10 ● 57 ● 62

2 Perhaps a little overkill for a simple application, but an alternative is to create a separate library which just contains the interface and reference it from all that need to know about it :o) – Andrew Oct 7, 2008 at 12:57

2

hmmm... If you are using Assembly.LoadFrom to load your assembly try changing it Assembly.LoadFile instead.

Worked for me

From here: http://www.eggheadcafe.com/community/aspnet/2/10036776/solution-found.aspx

Share Improve this answer Follow

answered Oct 7, 2008 at 12:52

ash

I had the identical problem as described here. Changing to Assembly.LoadFile solved it for me. – Adam Raney Mar 23, 2011 at 16:33

1

@lubos hasko

You nailed it on the nose. My original design did have three different assemblies with both the host and plugin implementation referencing the plugin interface assembly.

I tried a separate solution with a host implementation and interface assembly and a plugin implementation assembly. Within that solution, the code in the first block worked as expected.

You've given me a bit more to think about, because I'm not quite understanding why two assemblies referencing a common assembly don't get the same type out of the common assembly.

Share

Improve this answer

Follow

edited May 23, 2017 at 10:27

Community Bot

1 • 1

answered Aug 28, 2008 at 23:47

Ben Robbins

2,630 • 2 • 25 • 26

1

I was just trying to work this out myself and managed to stumble on the answer!

I had 3 different C# projects

- A - Plugin Interface project
- B - Host exe project -> references A
- C - Plugin Implementation project -> references A

I was getting the casting error as well until I changed the assembly name for my Plugin Interface proj to match the namespace of what I was trying to cast to.

E.g.

```
IPluginModule pluginModule = (IPluginModule)Activator.CreateInstance(curType);
```

was failing because the assembly that the IPluginModule interface was defined in was called 'Common', The -type- I was casting to was 'Blah.Plugins.Common.IPluginModule' however.

I changed the Assembly name for the interface proj to be 'Blah.Plugins.Common' meant that the cast then succeeded.

Hopefully this explanation helps someone. Back to the code..

Share Improve this answer Follow

answered Oct 7, 2008 at 12:26

ash

0

Is your type not public, if so, call the overload that takes in a boolean:

```
Activator.CreateInstance(type, true);
```

Also, in your first example, see if o is null and if not, print out o.GetType().Name to see what it really is.

Share Improve this answer Follow

answered Aug 28, 2008 at 2:57

Haacked

59k ● 14 ● 91 ● 115

0

@Haacked

I tried to keep the pseudocode simple. foreach's take up a lot of space and braces. I clarified it.

o.GetType().FullName returns Plugins.Multiply which is the expected object. Plugins.Multiply implements IPlugin. I stepped through the process in the debugger quite a few times until I gave up for the evening. Couldn't figure out why I couldn't cast it because I watched the constructor fire until I got grumpy about the whole mess. Came back to it this evening and made it work, but I still don't understand why the cast failed in the first code block. The second code block works, but it feels off to me.

Share

Improve this answer

Follow

edited May 23, 2017 at 12:34

Community Bot

1 ● 1

answered Aug 28, 2008 at 3:35

Ben Robbins

2,630 ● 2 ● 25 ● 26

The link to egghead above is the main solution to the problem use Assembly.LoadFile() instead of .LoadFrom()

0

Share Improve this answer Follow

answered Jan 26, 2011 at 18:45

Robel Robel Lingstuyl

1,571 ● 2 ● 15 ● 29