Purpose of Activator.CreateInstance with example?

107

28

Can someone explain `Activator.CreateInstance()` purpose in detail?

c# .net reflection

edited Sep 30 '11 at 10:29

sll
50.5k 15 89 136

asked Sep 29 '11 at 13:32

Tabriz Atayi
2,254 5 21 28

6 Probably that the greater documentation and example isn't available in the generic overloads. I would encourage that the documentation from `CreateInstance(Type type)` is matched with the `CreateInstance<T>()` overload. – Claus Jørgensen Sep 29 '11 at 21:35

4 The MSDN page just has a single explanatory line, "Contains methods to create types of objects locally or remotely, or obtain references to existing remote objects. This class cannot be inherited." msdn.microsoft.com/en-us/library/system.activator.aspx – gonzobrains Feb 22 '16 at 20:00

1 If you've come here from a Java background, this is the `c#.net` way of doing `Object xyz = Class.forName(className).newInstance();` . – SNag Apr 5 '18 at 17:02

There's better documentation for it here. – jpaugh Apr 22 at 17:11

9 Answers

134

Say you have a class called `MyFancyObject` like this one below:

```
class MyFancyObject
{
 public int A { get;set;}
}
```

Into

```
MyFancyObject obj;
```

Using

```
obj = (MyFancyObject)Activator.CreateInstance("MyAssembly", ClassName))
```

and can then do stuff like:

```
obj.A = 100;
```

That's its purpose. It also has many other overloads such as providing a `Type` instead of the class name in a string. Why you would have a problem like that is a different story. Here's some people who needed it:

- Createinstance() - Am I doing this right?
- C# Using Activator.CreateInstance
- Creating an object without knowing the class name at design time

edited May 23 '17 at 12:34
Community ♦
1 1

answered Sep 29 '11 at 13:58
deepee1
9,985 4 26 40

2 This proved useful to me. In my case, the class was in a different namespace, so I had to make sure that I included the namespace in my ClassName string (i.e. `String ClassName = "My.Namespace.MyFancyObject";`). – Scott Jun 17 '13 at 20:14

1 You forgot to add the Unwrap(). You can also put null, instead of "MyAssembly", and system will search current Assembly. – Tony Sep 18 '13 at 1:34

1 Can I do something like this `obj = (MyFancyObject)Activator.CreateInstance("MyAssembly", ClassName))` but instead of casting with the type. Cast with type made from the ClassName? Like this `Type type = Type.GetType(ClassName);obj = (type )Activator.CreateInstance("MyAssembly", ClassName))` ? – rluks Mar 26 '15 at 16:09

1 How is the above different from: MyFancyObject obj = new MyFancyObject? – Ed Landau Sep 1 '16 at 15:39

42

Well i can give you an example why to use something like that. Think of a game where you want to store your level and enemies in an XML file. When you parse this file, you might have an element like this.

```
<Enemy X="10" Y="100" Type="MyGame.OrcGuard"/>
```

what you can do now is, create dynamically the objects found in your level file.

```
foreach(XmlNode node in doc)
   var enemy = Activator.CreateInstance(null, node.Attributes["Type"]);
```

This is very useful, for building dynamic enviroments. Of course its also possible to use this for Plugin or addin scenarios and alot more.

answered Sep 29 '11 at 13:43

dowhilefor
9,895 2 22 38

5 I understood that it was to create a new instance of a type, but this is a nice simple example of why one would want to do so. – jamiebarrow Feb 8 '12 at 13:51

12

My good friend MSDN can explain it to you, with an example

Here is the code in case the link or content changes in the future:

```
using System;

class DynamicInstanceList
{
    private static string instanceSpec = "System.EventArgs;System.Random;" +
        "System.Exception;System.Object;System.Version";

    public static void Main()
```

```
                // create the object from the specification string
                Console.WriteLine("Creating instance of: {0}", instances[i]);
                item = Activator.CreateInstance(Type.GetType(instances[i]));
                instlist.SetValue(item, i);
            }
            Console.WriteLine("\nObjects and their default values:\n");
            foreach (object o in instlist)
            {
                Console.WriteLine("Type:     {0}\nValue:    {1}\nHashCode: {2}\n",
                    o.GetType().FullName, o.ToString(), o.GetHashCode());
            }
        }
    }

    // This program will display output similar to the following:
    //
    // Creating instance of: System.EventArgs
    // Creating instance of: System.Random
    // Creating instance of: System.Exception
    // Creating instance of: System.Object
    // Creating instance of: System.Version
    //
    // Objects and their default values:
    //
    // Type:     System.EventArgs
    // Value:    System.EventArgs
    // HashCode: 46104728
    //
    // Type:     System.Random
    // Value:    System.Random
    // HashCode: 12289376
    //
    // Type:     System.Exception
    // Value:    System.Exception: Exception of type 'System.Exception' was thrown.
    // HashCode: 55530882
    //
    // Type:     System.Object
    // Value:    System.Object
    // HashCode: 30015890
    //
    // Type:     System.Version
    // Value:    0.0
    // HashCode: 1048575
```

1 Unlikely to happen for MSDN, and copying the content here is *almost* a violation of copyright ;) – Claus Jørgensen Jun 22 '13 at 18:07

11 You're right. Personally I feel the principle also works to give better answers. I often come to SO with a lot on my mind from my current project. I usually just want a simple and to-the-point answer, so I can continue where I left off. I hate having to open up articles, which sometimes even link to other articles. Many answerers don't realize that many people don't come here with time on their hands to read through several articles, with a ton of unnecessary introductions and talks. Briefly summing up the important parts of a good article is key to some of the greatest answers I've seen. – Aske B. Jun 26 '13 at 20:42

9

You can also do this -

```
var handle = Activator.CreateInstance("AssemblyName",
                 "Full name of the class including the namespace and class name");
var obj = handle.Unwrap();
```

edited Aug 13 '13 at 18:18
LarsTech
71.6k 13 118 170

answered Aug 13 '13 at 17:44
William
310 4 6

The Unwrap() was the missing piece. :) – Tony Sep 18 '13 at 1:38

I can't find the 'Unwrap()' method above to use (as above). Has something changed in the new .NET APIs? – Sam May 6 '15 at 14:36

It's still there in System namespace. – William May 8 '15 at 16:24

2 Could you provide some additional explanation on what `.Unwrap()` does precisely and how this relates to other solutions? – Jeroen Vannevel May 10 '15 at 23:36

@Sam it's on a different overload of `CreateInstance` where it returns `System.Runtime.Remoting.ObjectHandle` . – nawfal Jul 3 '16 at 12:35

7

A good example could be next: for instance you have a set of Loggers and you allows user to specify type to be used in runtime via configuration file.

Then:

OR another case is when you have a common entities factory, which creates entity, and is also responsible on initialization of an entity by data received from DB:

(pseudocode)

```
public TEntity CreateEntityFromDataRow<TEntity>(DataRow row)
 where TEntity : IDbEntity, class
{
   MethodInfo methodInfo = typeof(T).GetMethod("BuildFromDataRow");
   TEntity instance = Activator.CreateInstance(typeof(TEntity)) as TEntity;
   return methodInfo.Invoke(instance, new object[] { row } ) as TEntity;
}
```

edited Apr 12 '17 at 11:16

answered Sep 29 '11 at 13:40

sll
50.5k 15 89 136

This does not work, `typeof(loggerType)` results in `loggerType is a variable and used like a type` – Martin Barker Mar 5 '17 at 21:36

1 @MartinBarker - thanks updated – sll Apr 12 '17 at 11:16

2

The `Activator.CreateInstance` method creates an instance of a specified type using the constructor that best matches the specified parameters.

For example, let's say that you have the type name as a string, and you want to use the string to create an instance of that type. You could use `Activator.CreateInstance` for this:

```
string objTypeName = "Foo";
Foo foo = (Foo)Activator.CreateInstance(Type.GetType(objTypeName));
```

Here's an MSDN article that explains it's application in more detail:

http://msdn.microsoft.com/en-us/library/wccyzw83.aspx

5 Or you could just use `new Foo()` . I think the OP wanted a more realistic example. – Konrad Rudolph Sep 29 '11 at 13:38

I agree with @Konrad. The reason for using `CreateInstance` is if you don't know the type of object you are going to instantiate at design time. In this example, you clearly know it's of type `Foo` since you are casting it as type `Foo` . You would never do this because you can just do `Foo foo = new Foo()` . – theyetiman Dec 23 '13 at 16:15

1

Building off of deepee1 and this, here's how to accept a class name in a string, and then use it to read and write to a database with LINQ. I use "dynamic" instead of deepee1's casting because it allows me to assign properties, which allows us to dynamically select and operate on any table we want.

```
Type tableType = Assembly.GetExecutingAssembly().GetType("NameSpace.TableName");
ITable itable = dbcontext.GetTable(tableType);

//prints contents of the table
foreach (object y in itable) {
    string value = (string)y.GetType().GetProperty("ColumnName").GetValue(y, null);
    Console.WriteLine(value);
}

//inserting into a table
dynamic tableClass = Activator.CreateInstance(tableType);
//Alternative to using tableType, using Tony's tips
dynamic tableClass = Activator.CreateInstance(null, "NameSpace.TableName").Unwrap();
tableClass.Word = userParameter;
itable.InsertOnSubmit(tableClass);
dbcontext.SubmitChanges();

//sql equivalent
dbcontext.ExecuteCommand("INSERT INTO [TableNme]([ColumnName]) VALUES ({0})",
userParameter);
```

answered Nov 30 '13 at 21:56

DharmaTurtle

894 1 12 27

```
label1.txt = "Pizza"
Magic(label1.txt) p = new Magic(lablel1.txt)(arg1, arg2, arg3);
p.method1();
p.method2();
```

If I already know its a Pizza there's no advantage to:

```
p = (Pizza)somefancyjunk("Pizza"); over
Pizza p = new Pizza();
```

but I see a huge advantage to the Magic method if it exists.

answered Sep 23 '17 at 2:31

user8659016

1

0

Coupled with reflection, I found Activator.CreateInstance to be very helpful in mapping stored procedure result to a custom class as described in the following answer.

answered Dec 27 '17 at 22:02

usefulBee

5,415 5 39 62