If there is no difference between two options, choose the one that is easier to debug

devblogs.microsoft.com/oldnewthing/20170725-00

July 25, 2017

Raymond Chen

In C# you have two ways of casting one object to another. One is to use the `as` operator, which attempts to casting the object and either returns the successfully cast result or returns `null`.

Another way is to use the casting operator.

In the case where you are going to use the result of the cast immediately, Which should you use?

```
// option 1
var thing = GetCurrentItem();
var foo = thing as Foo;
foo.DoSomething();

// option 2
var thing = GetCurrentItem();
var foo = (Foo)thing;
foo.DoSomething();
```

Now, suppose the `thing` is not a `Foo` after all. Both options will crash, but they will crash differently.

In the first version, you will crash with a `NullReferenceException` at the `foo.DoSomething()`, and the crash dump will confirm that, yes, the `foo` variable is null. But the `thing` might not be in the crash dump. Maybe the crash dump captures only the parameters that participated in the expression that resulted in the exception. Or maybe `thing` was GC'd away. You can't tell whether the problem is that `GetCurrentItem` returned `null`, or that `GetCurrentItem` returned an object that wasn't a `Foo`. And if it wasn't a `Foo`, what was it?

In the second version, there are two ways the code could crash. If the `thing` is null, then you will get a `NullReferenceException` at the `foo.DoSomething()`. But if the `thing` is the wrong kind of object, then the crash will occur at the point of the cast with an

`InvalidCastException`. And if you're lucky, the debugger will show you the thing that could not be cast. Even if it doesn't, you can at least determine from the type of the exception which of the two cases you're in.

Exercise: The following two lines of code are functionally equivalent. Which is easier to debug?

```
// option 1
collection.FirstOrDefault().DoSomething();

// option 2
collection.First().DoSomething();
```

Raymond Chen

Follow