Exceptions for flow of control

Asked 16 years, 2 months ago Modified 15 years, 11 months ago

Viewed 2k times

4

There is an interesting post over here about this, in relation to cross-application flow of control.

Well, recently, I've come across an interesting problem. Generating the nth value in a potentially (practically) endless recursive sequence. This particular algorithm WILL be in atleast 10-15 stack references deep at the point that it succeeds. My first thought was to throw a SuccessException that looked something like this (C#):

```
class SuccessException : Exception
{
    public string Value
    { get; set; }

    public SuccessException(string value)
        : base()
    {
        Value = value;
    }
}
```

Then do something like this:

```
try
{
    Walk_r(tree);
}
```

```
catch (SuccessException ex)
{
    result = ex.Value;
}
```

Then my thoughts wandered back here, where I've heard over and over to never use Exceptions for flow control. Is there ever an excuse? And how would you structure something like this, if you were to implement it?

algorithm exception data-structures control-flow

Share

Improve this question

Follow

edited May 23, 2017 at 11:53

Community Bot

1 ● 1

asked Oct 6, 2008 at 14:19

Matthew Scharley

132k ● 55 ● 196 ● 225

7 Answers

Sorted by: Highest score (default)

5

In this case I would be looking at your Walk_r method, you should have something that returns a value, throwing an exception to indicate success, is NOT a common practice, and at minimum is going to be VERY confusing to anyone that sees the code. Not to mention the overhead associated with exceptions.

Share Improve this answer

Follow

answered Oct 6, 2008 at 14:22

Mitchel Sellers
63.1k ● 15 ● 114 ● 174

Compared to the overhead to detect a success value at each level and break out of 10+ stack references? As for the usage, it's a personal project, not that that's any excuse I suppose. – Matthew Scharley Oct 6, 2008 at 14:24

if you are 10+ stack references deep and hault processing the framework has to do a bunch of additional processing anyway. If you setup the recursive calls to return the successful value, it shouldn't be an issue – Mitchel Sellers Oct 6, 2008 at 14:27

True enough. The other half of the question was how I could go about structuring this differently. Perhaps as "bool Walk_r(object tree, out string result)"? – Matthew Scharley Oct 6, 2008 at 14:29

I think to answer this we need to know a bit more about waht you are doing, and how the current method is structured. Typically the return value of a recursive call is the value, that indicates success/failure. – Mitchel Sellers Oct 6, 2008 at 14:40

1 Whatever is being done there is no good reason for throwing an exception for a non-exceptional condition. The way to return a value is to use return statement. If you are suffering performance problems you can use an explicit stack. – domgblackwell Oct 8, 2008 at 11:18

walk_r should simply return the value when it is hit. It's is a pretty standard recursion example. The only potential problem I see is that you said it is potentially endless,

1

which will have to be compensated for in the walk_r code by keeping count of the recursion depth and stopping at some maximum value.

The exception actually makes the coding very strange since the method call now throws an exception to return the value, instead of simply returning 'normally'.

```
try
{
    Walk_r(tree);
}
catch (SuccessException ex)
{
    result = ex.Value;
}
```

becomes

```
result = Walk_r(tree);
```

Share Improve this answer Follow

answered Oct 6, 2008 at 14:30

Robin
24.3k ● 5 ● 52 ● 58

I said effectively. The recursion depth is only 15, but the total problem space covered is literally in the billions, and wouldn't be able to be covered in real time, hence endless.
– Matthew Scharley Oct 6, 2008 at 14:34

I'm going to play devil's advocate here and say stick with the exception to indicate success. It might be expensive

1

to throw/catch but that may be insignificant compared with the cost of the search itself and possibly less confusing than an early exit from the method.

Share Improve this answer Follow

answered Oct 6, 2008 at 15:00

finnw
48.6k ● 24 ● 148 ● 223

The search is a handful of seconds. Not trivial, but not long as far as real time is concerned. An exception on top of that is probably nothing, but I managed to restructure the call so it didn't need it, and the points about readability for someone else are useful too. – Matthew Scharley Oct 6, 2008 at 23:42

0

It's not a very good idea to throw exceptions as a part of an algorithm, especially in .net. In some languages/platforms, exceptions are pretty efficient when thrown, and they usually are, when an iterable gets exhausted for instance.

Share Improve this answer Follow

answered Oct 6, 2008 at 14:26

Vasil
38k ● 27 ● 93 ● 116

0

Why not just return the resulting value? If it returns anything at all, assume it is successful. If it fails to return a value, then it means the loop failed.

If you must bring back from a failure, then I'd recommend you throw an exception.

Share Improve this answer Follow

answered Oct 6, 2008 at 14:28

schonarth

534 ● 7 ● 13

0

The issue with using exceptions is that tey (in the grand scheme of things) are very inefficient and slow. It would surely be as easy to have a if condition within the recursive function to just return as and when needed. To be honest, with the amount of memory on modern PC's its unlikely (not impossible though) that you'll get a stack overflow with only a small number of recursive calls (<100).

If the stack is a real issue, then it might become necessary to be 'creative' and implement a 'depth limited search strategy', allow the function to return from the recursion and restart the search from the last (deepest) node.

To sum up: Exceptions should only be used in exceptional circumstances, the success of a function call i don't believe qualifies as such.

Share Improve this answer

Follow

answered Oct 6, 2008 at 14:28

TK.

47.8k ● 47 ● 121 ● 148

0

Using exceptions in normal program flow in my book is one of the worst practises ever. Consider the poor sap who is hunting for swallowed exceptions and is running a debugger set to stop whenever an exception happens. That dude is now getting mad.... and he has an axe. :P

Share Improve this answer

Follow

answered Oct 6, 2008 at 14:30

Quibblesome

25.4k ● 10 ● 62 ● 104

Hence the custom exception and specialised catch. Swallowing exceptions is bad! – Matthew Scharley Oct 6, 2008 at 14:36

That's still going to halt execution when i'm looking for swallowed bugs. Exceptions in normal application flow are just wrong. Use architecture instead. – Quibblesome Oct 7, 2008 at 15:17