Use of the Exception class in c#

Asked 16 years, 3 months ago Modified 16 years, 3 months ago

Viewed 2k times

5

Errors that occur deep down in a data access layer or even higher up, (say within ADO.net operations for example) rarely make much sense to an end user. Simply bubbling these errors up to a UI and displaying them will usually achieve nothing except frustration for an end user.

I have recently employed a basic technique for reporting errors such as this whereby I catch the error and at least add some user friendly text so that at least the end user understands what failed.

To do this I am catching an exception within each specific function (say for example a fetch function in a data access layer), then raising a new error with user friendly text about the function that has failed and probably cause, but then embedding the original exception in the new exception as the "inner exception" of that new exception.

This can then occur at each layer if necessary, each consumer of the lower level function adding it's own context to the error message, so that what reaches the UI is an increasingly user friendly error message.

Once the error reaches the UI - if necessary - it can then iterate through the nested exceptions in order to display an error message that firstly tells the user which operation failed, but also provides a bit of technical information about what actually went wrong.

e.g.

> "The list of customer names your requested could not be displayed."
>
> "Obtaining the list of customers you requested failed due to an error with the database."
>
> "There was an error connecting to the database when retrieving a list of customers"
>
> "Login failed for user xx"

My question is this: Is this horribly inefficient (all those nested exceptions)? I suspect it is not best practice so what should I be doing to achieve the same thing - or should I in fact be trying to achieve something better?

.net **exception**

Share

Improve this question

Follow

edited Sep 21, 2008 at 0:52

asked Sep 20, 2008 at 23:43

Stuart Helwig
9,444 ● 8 ● 55 ● 69

6 Answers

Sorted by: Highest score (default)

9

It is just slightly horrible.

If you are showing an error to the end user, the user is supposed to be able to act about it. In "The list of customer names your requested could not be displayed." case, your user will just think "so what?" On all of these cases, just display a "something bad happened" message. You do not even need to catch these exceptions, when something goes bad, let some global method (like application_error) handle it and display a generic message. When you or your user can do something about the error, catch it and do the thing or notify the user.

But you will want to log every error that you do not handle.

By the way, displaying information about the errors occuring may yield to security vulnerabilities. The less the attackers know about your system, the less likely they will find ways to hack it (remember those messages like "Syntax error in sql statement: Select * From Users Where username='a'; drp database;--'..." expected: 'drop' instead of 'drp'. They do not make sites like these anymore).

Share Improve this answer Follow

edited Sep 20, 2008 at 23:56

answered Sep 20, 2008 at 23:50

Serhat Ozgel
23.7k ● 29 ● 104 ● 141

oh I WISH they didnt make sites like this anymore! Would make my job (as a HPC) a little more challenging... – AviD
Sep 21, 2008 at 5:56

8

It is technically costly to throw new exceptions, however I won't make a big debate out of that since "costly" is relative - if you're throwing 100 such exceptions a minute, you will likely not see the cost; if you're throwing 1000 such exceptions a second, you very well may see a performance hit (hence, not really worth discussing here - performance is your call).

I guess I have to ask why this approach is being used. Is it really true that you can add meaningful exception information at *every* level where an exception might be thrown and, if so, is it also true that the information will be:

- Something you actually *want* to share with your user?
- Something your user will be able to interpret, understand and *use*?

- Written in such a way that it will not interfere with later reuse of low-level components, the utility of which might not be known when they were written?

I ask about sharing information with your user because, in your example, your artificial stack starts by informing the user there was a problem authenticating on the database. For a potential hacker, that's a good piece of information that exposes something about what the operation was doing.

As for handing back an entire custom exception stack, I don't think it's something that will be useful to most (honest) users. If I'm having trouble getting a list of customer names, for instance, is it going to help me (as a user) to know there was a problem authenticating with the database? Unless you're using integrated authentication, and each of your users has an account, and the ability to contact a system administrator to find out why their account lacks privileges, probably not.

I would begin by first deciding if there is really a semantic difference between the Framework exception thrown and the exception message you'd like to provide to the user. If there is, then go ahead and use a custom exception at the lowest level ('login failed' in your example). The steps following that, up to the actual presentation of the exception, don't really require any custom exceptions. The exception you're interested in has already been generated (the login has failed) - continuing to wrap that message at every level of the call stack serves no real purpose other than exposing your call stack to your users. For those "middle" steps, assuming any try/catch blocks are in place, a simple 'log and throw' strategy would work fine.

Really, though, this strategy has another potential flaw: it forces upon the developer the responsibility for maintaining the custom exception standard that's been implemented. Since you can't possibly know every permutation of call hierarchy when writing low-level types (their "clients" might not even have been written yet), it seems unlikely that all developers - or even one developer - would remember to wrap and customize any error condition in every code block.

Instead of working from the bottom up, I typically worry about the display of thrown exceptions as late in the process as possible (i.e. as close to the "top" of the call stack as possible). Normally, I don't try to replace any messages in exceptions thrown at low levels of my applications - particularly since the usage of those low level members tend to get more and more abstract the deeper the call gets. I tend to catch and log exceptions in the business tier and lower, then deal with displaying them in a comprehensible manner in the presentation tier.

Here are a couple of decent articles on exception handling best practices:

http://www.codeproject.com/KB/architecture/exceptionbestpractices.aspx http://aspalliance.com/1119

Jeez this got wordy...apologies in advance.

Share Improve this answer Follow

answered Sep 21, 2008 at 0:36
Jared
2,015 ● 15 ● 14

Thanks for all that effort Jared and thanks for the links. One point I would raise is that I am not that concerned with multiple permutations within the call hierarchy, because each layer (moving upwards towards the UI) only adds it's own context to the error. – Stuart Helwig Sep 21, 2008 at 0:55

Your point about giving the user information that isn't that useful is taken though, and I think I'll review how much of the information gathered I actually display, versus what else is available, maybe in a "Show details" function - for tech support reasons. Thanks again. – Stuart Helwig Sep 21, 2008 at 0:56

3

Yes, exceptions are expensive so there is a cost involved in catching and rethrowing or throwing a more useful exception.

But wait a minute! If the framework code or the library you're using throws an exception then things are already coming unstuck. Do you have non-functional requirements for how quickly an error message is propagated following an exception? I doubt it. Is it really a big deal? Something unforeseen and 'exceptional' has happened. The main thing is to present sensible, helpful information to the user.

I think you're on the right track with what you're doing.

Share Improve this answer Follow

answered Sep 20, 2008 at 23:49

Hamish Smith
8,181 ● 1 ● 38 ● 49

2

Of course it's horribly inefficient. But at the point that an exception occurs that is important enough to show to the end user, you should not care about that.

Share Improve this answer Follow

answered Sep 20, 2008 at 23:45

Thomas
181k ● 55 ● 376 ● 501

2

Where I work we have only a few reasons to catch exceptions. We only do it when...

1. We can do something about it - e.g We know that this can happen sometimes and we can rectify it in code as it happens (very rare).
2. We want to know it happens and where (then we just rethrow the exception).
3. We want to add a friendly message in which case we wrap the original exception in new excpetion derived from application exception and add a friendly

message to that and then let it bubble up unchanged from that point.

In your example we'd probably just display "Logon error occurred." anbd leave it at that while logging the real error and providing a why for the user to drill into the exception if they wanted too. (Perhaps a button on the error form).

4. We want to suppress the exception completely and keep going. Needless to say we only do this for expected exception types and only when there is no other way to detect the condition that generates the exception.

answered Sep 21, 2008 at 0:24
Brody
2,074 ● 1 ● 18 ● 23

1

Generally when you're dealing with exceptions, performance and efficiency are the least of your worries. You should be more worried about doing something to help the user recover from the problem. If there was a problem writing a certain record to the database, either roll the changes back or at least dump the row information so the user doesn't lose it.

answered Sep 20, 2008 at 23:47
Andrew Burgess
5,298 ● 5 ● 33 ● 37