Best practices for catching and re-throwing .NET exceptions

272

102

What are the best practices to consider when catching exceptions and re-throwing them? I want to make sure that the `Exception` object's `InnerException` and stack trace are preserved. Is there a difference between the following code blocks in the way they handle this?

```
try
{
    //some code
}
catch (Exception ex)
{
    throw ex;
}
```

Vs:

```
try
{
    //some code
}
catch
{
    throw;
}
```

c# .net exception-handling rethrow

edited Oct 12 '18 at 9:41
Venkat
1,369 2 12 38

asked Aug 22 '08 at 15:12
Seibar
36.8k 34 77 98

257

```
try {
  // something that bombs here
} catch (Exception ex)
{
    throw;
}
```

`throw ex;` is basically like throwing an exception from that point, so the stack trace would only go to where you are issuing the `throw ex;` statement.

Mike is also correct, assuming the exception allows you to pass an exception (which is recommended).

Karl Seguin has a great write up on exception handling in his foundations of programming e-book as well, which is a great read.

Edit: Working link to Foundations of Programming pdf. Just search the text for "exception".

edited Sep 23 '17 at 13:06
LW001
1,573 4 14 24

answered Aug 22 '08 at 15:13
Darren Kopp
61.8k 9 67 87

10 I'm not so sure if that write-up is wonderful, it suggests try { // ... } catch(Exception ex) { throw new Exception(ex.Message + "other stuff"); } is good. The problem is that you're completely unable to handle that exception any further up the stack, unless you catch all exceptions, a big no-no (you sure you want to handle that OutOfMemoryException?) – ljs Jun 22 '09 at 12:10

2 @ljs Has the article changed since your comment as I don't see any section where he recommends that. Quite the opposite in fact, he says not to do it and asks if you want to handle the OutOfMemoryException as well!? – RyanfaeScotland Mar 31 '15 at 14:02

6 Sometimes *throw;* is not enough to preserve stack trace. Here is an example https://dotnetfiddle.net/CkMFoX – Artavazd Balayan Oct 7 '16 at 13:23

How do you properly rethrow an inner exception, when the exception comes from within a reflection call for example? – jocull Feb 28 '17 at 22:02

3 Or `ExceptionDispatchInfo.Capture(ex).Throw(); throw;` in .NET +4.5 stackoverflow.com/questions/57383/… – Alfred Wallace Apr 12 at 20:42

If you throw a new exception with the initial exception you will preserve the initial stack trace too..

answered Aug 22 '08 at 15:18

Mike
1,638 13 19

No matter what I try throw new Exception("message", ex) always throws ex and ignores the custom message. throw new Exception("message", ex.InnerException) works though. – Tod Apr 2 '15 at 9:23

If no custom exception is needed one can use AggregateException (.NET 4+) msdn.microsoft.com/en-us/library/… – Nikos Tsokos Nov 11 '15 at 11:24

`AggregateException` should only be used for exceptions over aggregated operations. For example, it is thrown by the `ParallelEnumerable` and `Task` classes of the CLR. Usage should probably follow this example. – Aluan Haddad Feb 27 '17 at 8:41

27

Actually, there are some situations which the `throw` statment will not preserve the StackTrace information. For example, in the code below:

```
try
{
  int i = 0;
  int j = 12 / i; // Line 47
  int k = j + 1;
}
catch
{
  // do something
  // ...
  throw; // Line 54
}
```

The StackTrace will indicate that line 54 raised the exception, although it was raised at line 47.

```
Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.
   at Program.WithThrowIncomplete() in Program.cs:line 54
   at Program.Main(String[] args) in Program.cs:line 106
```

```
private static void PreserveStackTrace(Exception exception)
{
  MethodInfo preserveStackTrace =
typeof(Exception).GetMethod("InternalPreserveStackTrace",
    BindingFlags.Instance | BindingFlags.NonPublic);
  preserveStackTrace.Invoke(exception, null);
}
```

I has a disadvantage of relying on a private method to preserve the StackTrace information. It can be changed in future versions of .NET Framework. The code example above and proposed solution below was extracted from Fabrice MARGUERIE weblog.

Calling Exception.SetObjectData

The technique below was suggested by Anton Tykhyy as answer to In C#, how can I rethrow InnerException without losing stack trace question.

```
static void PreserveStackTrace (Exception e)
{
  var ctx = new StreamingContext  (StreamingContextStates.CrossAppDomain) ;
  var mgr = new ObjectManager     (null, ctx) ;
  var si  = new SerializationInfo (e.GetType (), new FormatterConverter ()) ;

  e.GetObjectData    (si, ctx)  ;
  mgr.RegisterObject (e, 1, si) ; // prepare for SetObjectData
  mgr.DoFixups       ()         ; // ObjectManager calls SetObjectData

  // voila, e is unmodified save for _remoteStackTraceString
}
```

Although, it has the advantage of relying in public methods only it also depends on the following exception constructor (which some exceptions developed by 3rd parties do not implement):

```
protected Exception(
    SerializationInfo info,
    StreamingContext context
)
```


johnnyRose
4,339 11 36 54


CARLOS LOTH
3,292 2 31 40

1 You can catch the exception and publish this exception anywhere you want to. Then throw a new one explaining what happened to the user. This way you can see what happened at the current time the exception was caught, the user can careless what the actual exception was. – Çöđěxěř Nov 17 '12 at 15:16

1 With .NET 4.5 there is a third and - in my opinion - cleaner option: use ExceptionDispatchInfo. See Tragedians answer to a related question here: stackoverflow.com/a/17091351/567000 for more info. – Søren Boisen Dec 30 '15 at 14:38

19

When you `throw ex` , you're essentially throwing a new exception, and will miss out on the original stack trace information. `throw` is the preferred method.

answered Aug 22 '08 at 15:15


Forgotten Semicolon
12.1k 1 43 60

13

The rule of thumb is to avoid Catching and Throwing the basic `Exception` object. This forces you to be a little smarter about exceptions; in other words you should have an explicit catch for a `SqlException` so that your handling code doesn't do something wrong with a `NullReferenceException` .

In the real world though, catching *and logging* the base exception is also a good practice, but don't forget to walk the whole thing to get any `InnerExceptions` it might have.

answered Aug 22 '08 at 15:18


swilliams
35k 22 88 129

2 I think it's best to deal with unhandled exceptions for logging purposes by using the AppDomain.CurrentDomain.UnhandledException and Application.ThreadException exceptions. Using big try { ... } catch(Exception ex) { ... } blocks everywhere means a lot of duplication. Depends whether you want to log handled exceptions, in which case (at least minimal) duplication might be inevitable. – ljs Jun 22 '09 at 12:23

9

Nobody has explained the difference between `ExceptionDispatchInfo.Capture( ex ).Throw()` and a plain `throw` , so here it is. However, some people have noticed the problem with `throw` .

The complete way to rethrow a caught exception is to use `ExceptionDispatchInfo.Capture( ex ).Throw()` (only available from .Net 4.5).

Below there are the cases necessary to test this:

1.

```
void CallingMethod()
{
    //try
    {
        throw new Exception( "TEST" );
    }
    //catch
    {
    //    throw;
    }
}
```

2.

```
void CallingMethod()
{
    try
    {
        throw new Exception( "TEST" );
    }
    catch( Exception ex )
    {
        ExceptionDispatchInfo.Capture( ex ).Throw();
        throw; // So the compiler doesn't complain about methods which don't either
return or throw.
    }
}
```

3.

```
        throw new Exception( "TEST" );
    }
    catch
    {
        throw;
    }
}
```

4.

```
void CallingMethod()
{
    try
    {
        throw new Exception( "TEST" );
    }
    catch( Exception ex )
    {
        throw new Exception( "RETHROW", ex );
    }
}
```

Case 1 and case 2 will give you a stack trace where the source code line number for the `CallingMethod` method is the line number of the `throw new Exception( "TEST" )` line.

However, case 3 will give you a stack trace where the source code line number for the `CallingMethod` method is the line number of the `throw` call. This means that if the `throw new Exception( "TEST" )` line is surrounded by other operations, you have no idea at which line number the exception was actually thrown.

Case 4 is similar with case 2 because the line number of the original exception is preserved, but is not a real rethrow because it changes the type of the original exception.

edited Nov 14 '16 at 11:12

loki

5,523 5 28 58

answered Nov 14 '16 at 10:24

jeuoekdcwzfwccu

201 3 3

Add a simple blurb to never use `throw ex;` and this is the best answer of them all – NH. Apr 5 '18 at 16:03

8

of imformation which is the line where the exception happened.

Consider the following code:

```
static void Main(string[] args)
{
    try
    {
        TestMe();
    }
    catch (Exception ex)
    {
        string ss = ex.ToString();
    }
}

static void TestMe()
{
    try
    {
        //here's some code that will generate an exception - line #17
    }
    catch (Exception ex)
    {
        //throw new ApplicationException(ex.ToString());
        throw ex; // line# 22
    }
}
```

When you do either a 'throw' or 'throw ex' you get the stack trace but the line# is going to be #22 so you can't figure out which line exactly was throwing the exception (unless you have only 1 or few lines of code in the try block). To get the expected line #17 in your exception you'll have to throw a new exception with the original exception stack trace.

answered Feb 8 '12 at 1:09

notlkk

676 1 11 34

- C#'s "throw ex;" gets compiled into MSIL's "throw"
- C#'s "throw;" - into MSIL "rethrow"!

Basically I can see the reason why "throw ex" overrides the stack trace.

edited Aug 8 '13 at 18:26
Community ♦
1 1

answered Jul 5 '10 at 13:23
Vinod T. Patil
2,131 3 21 20

3

You may also use:

```
try
{
// Dangerous code
}
finally
{
// clean up, or do nothing
}
```

And any exceptions thrown will bubble up to the next level that handles them.

answered Aug 22 '08 at 15:32
Erick B
1,132 9 21

3

I would definitely use:

```
try
{
    //some code
```

That will preserve your stack.

edited Apr 30 '15 at 4:25

answered Aug 22 '08 at 15:19

1kevgriff

1,287 4 19 26

1 To be fair to past me in 2008, the OP was asking how to preserve the stack -- and 2008 me gave a correct answer. What's missing from my answer is the part of actually doing something in the catch. – 1kevgriff Apr 30 '15 at 4:23

@JohnSaunders That is true if and only if you *don't* do anything before `throw` ; for example you could clean up a disposable (where you ONLY call it on an error) and then throw the exception. – Meirion Hughes Apr 21 '16 at 10:40

@meirion when I wrote the comment there was nothing before the throw. When that was added, I upvoted, but didn't delete the comment. – John Saunders Apr 21 '16 at 11:34

0

FYI I just tested this and the stack trace reported by 'throw;' is not an entirely correct stack trace. Example:

```
private void foo()
{
    try
    {
        bar(3);
        bar(2);
        bar(1);
        bar(0);
    }
    catch(DivideByZeroException)
    {
        //log message and rethrow...
        throw;
    }
}

private void bar(int b)
{
```

The stack trace points to the origin of the exception correctly (reported line number) but the line number reported for foo() is the line of the throw; statement, hence you cannot tell which of the calls to bar() caused the exception.

answered Apr 13 '11 at 17:37

redcalx
4,582 3 38 83

Which is why it's best not to try to catch exceptions unless you plan to do something with them – Nate Zaugg Oct 11 '11 at 17:45