Best practices for catching and re-throwing .NET exceptions

Asked 16 years, 4 months ago Modified 6 years, 2 months ago Viewed 196k times

311

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 rethrow

Share
Improve this question
Follow

edited Oct 12, 2018 at 9:41
Venkat
2,579 ● 3 ● 31 ● 64

asked Aug 22, 2008 at 15:12
Seibar
70.2k ● 38 ● 93 ● 100

11 Answers

Sorted by: Highest score (default)

281

The way to preserve the stack trace is through the use of the `throw;` This is valid as well

```
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".

Share

Improve this answer

Follow

edited Sep 23, 2017 at 13:06
LW001
2,855 ● 7 ● 32 ● 42

answered Aug 22, 2008 at 15:13
Darren Kopp
77.6k ● 9 ● 80 ● 93

11 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, 2009 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, 2015 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, 2016 at 13:23

4 Or `ExceptionDispatchInfo.Capture(ex).Throw(); throw;` in .NET +4.5 stackoverflow.com/questions/57383/... – Alfred Wallace Apr 12, 2019 at 20:42

This answer is outdated. here's why it is bad. Here is the correct answer in .NET 4.5 and later – NH. Jun 28, 2019 at 16:33

109

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

```
try{
}
catch(Exception ex){
        throw new MoreDescriptiveException("here is what was happening", ex);
}
```

Share Improve this answer Follow

answered Aug 22, 2008 at 15:18

Mike

1,993 ● 1 ● 14 ● 19

1 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, 2015 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, 2015 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, 2017 at 8:41

30

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
```

In situations like the one described above, there are two options to preseve the original StackTrace:

Calling the Exception.InternalPreserveStackTrace

As it is a private method, it has to be invoked by using reflection:

```
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
)
```

In my situation, I had to choose the first approach, because the exceptions raised by a 3rd-party library I was using didn't implement this constructor.

Share

Improve this answer

Follow

edited Jun 1, 2017 at 21:16
johnnyRose
7,480 ● 17 ● 42 ● 61

answered Jul 1, 2012 at 19:07
CARLOS LOTH
4,745 ● 3 ● 39 ● 44

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. – Trevor Nov 17, 2012 at 15:16

3 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, 2015 at 14:38

That a plain `throw;` here showed line 54 instead of line 47 should probably be considered a long-standing bug in .NET Framework, and was fixed in .NET Core 2.1 (github.com/dotnet/runtime/issues/9518). You can use `ExceptionDispatchInfo`, but this is not one of its primary use-cases (one of which is shown in stackoverflow.com/a/17091351), and suggesting it muddies the water and leads to less readable code. That said, having the line from the catch, and any further line numbers from the call stack has always been good enough for me. – Johann Sep 9, 2020 at 1:36

22

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.

Share Improve this answer Follow

answered Aug 22, 2008 at 15:15

Forgotten Semicolon
14.1k ● 2 ● 52 ● 61

15

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.

```
void CallingMethod()
{
    try
    {
        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.

Share
Improve this answer
Follow

edited Nov 14, 2016 at 11:12
loki
10.3k ● 7 ● 62 ● 84

answered Nov 14, 2016 at 10:24
Mark
617 ● 6 ● 5

3 Add a simple blurb to never use `throw ex;` and this is the best answer of them all. – NH. Apr 5, 2018 at 16:03

14

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.

Share Improve this answer Follow

answered Aug 22, 2008 at 15:18
swilliams
48.8k ● 27 ● 102 ● 130

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, 2009 at 12:23

Plus using those events means you *do* log all unhandled exceptions, whereas if you use big ol' try { ... } catch(Exception ex) { ... } blocks you might miss some. – ljs Jun 22, 2009 at 12:25

10

You should always use "throw;" to rethrow the exceptions in .NET,

Refer this, http://weblogs.asp.net/bhouse/archive/2004/11/30/272297.aspx

Basically MSIL (CIL) has two instructions - "throw" and "rethrow":

- 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.

Share

Improve this answer

Follow

edited Aug 8, 2013 at 18:26

Community Bot

1 ● 1

answered Jul 5, 2010 at 13:23

Vinod T. Patil

2,961 ● 3 ● 27 ● 22

The link -- well, actually the source that link cites -- is full of good information, and also notes a possible culprit for why many think `throw ex;` will rethrow -- in Java, it does! But *you oughta include that information here* to have a Grade A answer. (Though I'm still catching up with the `ExceptionDispatchInfo.Capture` answer from jeuoekdcwzfwccu.) – ruffin Jan 13, 2020 at 14:54

8

A few people actually missed a very important point - 'throw' and 'throw ex' may do the same thing but they don't give you a crucial piece 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.

Share Improve this answer Follow

answered Feb 8, 2012 at 1:09

notlkk
1,231 ● 2 ● 23 ● 40

That a plain `throw;` here also showed line 22 instead of line 17 should probably be considered a long-standing bug in .NET Framework, and was fixed in .NET Core 2.1 (github.com/dotnet/runtime/issues/9518). That said, I have never seen an instance where I cared to see the line from the `try` in the stacktrace. Having the line from the catch, and any further line numbers from the call stack has always been good enough. – Johann Sep 9, 2020 at 1:32

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.

Share Improve this answer Follow

answered Aug 22, 2008 at 15:32
Erick B
1,272 ● 10 ● 21

3

I would definitely use:

```
try
{
    //some code
}
catch
{
    //you should totally do something here, but feel free to rethrow
    //if you need to send the exception up the stack.
    throw;
}
```

That will preserve your stack.

Share
Improve this answer
Follow

edited Apr 30, 2015 at 4:25

answered Aug 22, 2008 at 15:19
Kevin Griffin
2,257 ● 4 ● 25 ● 27

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. – Kevin Griffin Apr 30, 2015 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, 2016 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, 2016 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)
    {
        int a = 1;
        int c = a/b;  // Generate divide by zero exception.
    }
```

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.

Share Improve this answer Follow

answered Apr 13, 2011 at 17:37

redcalx
8,617 ● 8 ● 61 ● 108

Which is why it's best not to try to catch exceptions unless you plan to do something with them – Nate Zaugg Oct 11, 2011 at 17:45