

Re: Exception Handling – help!

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.clr/2004-11/0041.html>

From: Jonathan Keljo [MS] (jkeljo_at_online.microsoft.com)

Date: 11/08/04

Date: Mon, 08 Nov 2004 18:33:33 GMT

Some responses inline.

This posting is provided "AS IS" with no warranties, and confers no rights.

| Jonathon,
|

| Thanks for clarifying this.
|

| I agree, George did a fine job of demonstrating this aspect of exception
| handling, and the behavior is counter-intuitive to what I expected. I
| suppose this means I need to install VB (groan) to duplicate this or
| start

| writing code directly in IL.
|

| BTW: I found the documentation where you mentioned; I would have expected
| it

| to also be in Partition I.12.4.2.5, the overview of Exception Handling.

As a

| side note, there is a discrepancy between Partition I, which states
| "Execution cannot be resumed at the location of the exception, except
| with a

| user-filtered handler", and Partition III.3.34, which states that the
| only

| possible return values from the filter are `exception_continue_search` and
| `exception_execute_handler`. In other words, resumable exceptions are not
| (yet) supported – not a big deal as I don't believe there's much need for
| it.

Yes, I noticed that as well as I read it. I agree with your assessment of
the need for resumable exceptions. :-)

|
| Partition III states that it is not legal to embed a try block within a
| filter block – is this true only for inline code, or does that include
| methods called from within the filter block?

microsoft.public.dotnet.framework.clr: Re: Exception Handling – help!

True only for inline code. The functions you call from within the filter can be arbitrary.

|
| The behavior of .NET seems to be a bit different then windows SEH nested
| exception handling. Based on Matt Pietrek's article on SEH,
| (<http://www.microsoft.com/msj/0197/Exception/Exception.aspx>) if I read
| it
| correctly, he indicates that if a nested exception occurs
| (DISPOSITION_NESTED_EXCEPTION or DISPOSITION_COLLIDED_UNWIND) either
| within
| the filter or the unwind the OS shuts down the app – obviously the CLR
| should not do that, but it does indicate that .NET diverges significantly
| from Windows SEH.

I actually conducted an experiment with nested SEH exceptions in WinXP a year or so ago. My comment was based on the results of that experiment, which surprised people around here as much as it surprises you. I might have messed up the experiment...if I can find the source for it I'll post it.

|
| I don't think any sane person would deliberately allow nested exceptions
| to
| escape but I can foresee situations where this could happen inadvertently.

Agreed. :-)

| For example, I've had exceptions thrown in catch and finally blocks from
| calls to Trace.WriteLine and Debug.WriteLine, which is not a failure I
| would
| ever expect unless the application was in the process of shutting down
| (which it was not when the exception was thrown).

|
| IMO the CLR's strategy you described for handling nested exceptions is
| fine
| for finally and catch blocks but I don't care for how it handles filters.
| Since the spec states that it is not legal to embed try blocks within a
| filter, this means that the normal means of handling errors may not be
| available within a filter, depending on whether this applies to any code
| called from a filter, or only to IL directly inline in the filter block.

|
| The aspect of this that I am uncomfortable with is that when an exception
| is
| thrown in a filter (which I assume can execute arbitrary code, including
| web
| method calls, etc.) it is essentially swallowed/thrown away and converted
| into a exception_continue_search return value from the filter. This means
| that if the normal result of running the filter is to execute the handler
| but in one exceptional circumstance it does not (because of an exception
| it

Re: Exception Handling – help!

| continues the search), then this can easily become one of those "blue moon"
| bugs that is nearly impossible to reproduce, let alone fix. I would
| definitely not want it to be swallowed and thrown away – it indicates
| that
| something is seriously wrong and continuing to execute code could result
| in
| unexpected behavior or data corruption. This is also one of those
| conditions
| that is very difficult to test, let alone predict/analyze the code paths.

This is somewhat mitigated by the intended use of filters. While you CAN run arbitrary code in them, at least today you SHOULD only run the code you need to determine whether to catch an exception. I have a hard time imagining that code being particularly heavyweight.

|
| I can see the logic in continuing the search – even if it resulted in a
| nested exception the new exception would have to walk the stack anyway
| from
| the same starting point looking for a handler – but that may not be the
| proper action to take. Is there any chance of getting something added to
| the
| CLR to leave some breadcrumbs behind when this sort of unusual event
| occurs?
| This could even be a DEBUG mode only behavior. Or perhaps there could be
| an
| option for specifying the default behavior if a nested exception in a
| filter
| occurs (execute_handler vs. continue_search).

There might be some things we could do based on some new Debugging API functionality we've put in in Whidbey. Filters aren't in common enough use for us to cram that in at this point in Whidbey, but I have added it to our list of things to do in the future.

|
| (sigh) I suppose I need to devise some unit tests for filters that force
| this behavior to occur. Ah well, at least I'm aware of it now.
|
| The side-effect of "unhandled exceptions" you mention is interesting; it
| can
| only happen with an unwitting exception type transformation from an
| exception type that is not caught (the first exception) to one that is
| explicitly caught (the nested exception that escapes). I think this would
| definitely confuse people.

Yep, very confusing. We have considered eagerly ripping the process in these cases. We haven't done it yet, though, out of concern for breaking things.

|
| Thanks for responding and all the info. Regards,
| Dave

|
|
|
| ""Jonathan Keljo [MS]"" <jkeljo@online.microsoft.com> wrote in message
| news:wJHHlm5wEHA.1288@cpsmftngxa10.phx.gbl...

| > Hi Dave, George--

| > Great investigative work on this thread. George's experiments have
| > indeed arrived at the intended behavior.

| >

| > If an exception is thrown while executing a filter, and it escapes
the

| > filter, the filter is indeed considered to have returned

| > exception_continue_search. (I had thought this was well documented, but
| > the

| > only place I could find it is in Partition III.3.34 in the ECMA spec,

| > which

| > describes the endfilter instruction.)

| >

| > Nesting of exceptions in general is cute. As far as I can tell this

| > isn't documented anywhere, and it's not clear to me whether this should
be

| > specified in the ECMA spec, or chalked up to the vagaries of Windows
SEH.

| > The rule(s) of thumb for exceptions thrown from finally/fault blocks (or
| > from functions called by finally/fault blocks) are:

| >

| > 1. If the nested exception does not escape the finally/fault block, the

| > finally/fault will continue executing after the nested exception is

| > caught.

| > Naturally the second pass of the outer exception will continue normally.

| > 2. If the nested exception DOES escape the finally/fault block, the

second

| > pass of the outer exception is aborted and the nested exception becomes

| > the

| > outer exception.

| >

| > This can be generalized to any level of nesting, though why in the world

| > you'd want to escapes me. These rules are also true for catch blocks,

| > though conceptually you can think of a catch block as being run _after_

| > the

| > end of the second pass.

| >

| > One interesting behavior that can result from this is that you can have

an

| > exception go "unhandled" without taking down the process. If the
outermost

| > exception goes unhandled, but the nested exception is handled after

| > escaping from the finally in which it was thrown, the second pass of the

microsoft.public.dotnet.framework.clr: Re: Exception Handling – help!

|> outermost exception will be aborted and the process will keep going.
|>
|> Jonathan
|> CLR Exception System Program Manager
|>
|> This posting is provided "AS IS" with no warranties, and confers no
|> rights.
|>
|> -----
|> | Thread-Topic: Exception Handling – help!
|> | thread-index: AcTCYpnvYALC1JuhQluefyvFGqqP4w==
|> | X-WBNR-Posting-Host: 129.132.1.4
|> | From: =?Utf-8?B?R2Vvcmdl?= <George@discussions.microsoft.com>
|> | References: <DBE66BDF-B746-4077-89EE-8996635E8B78@microsoft.com>
|> | <Oi8ipzTweHA.4048@TK2MSFTNGP15.phx.gbl>
|> | <8E53C9F9-54C8-4FF9-925A-5FEF78A07018@microsoft.com>
|> | <eYIM7eZwEHA.1408@TK2MSFTNGP10.phx.gbl>
|> | <B81BC8C3-90E4-47BB-BAD1-14AB098DEE44@microsoft.com>
|> | <uqztQ\$lwEHA.3624@TK2MSFTNGP09.phx.gbl>
|> | Subject: Re: Exception Handling – help!
|> | Date: Thu, 4 Nov 2004 03:37:01 -0800
|> | Lines: 33
|> | Message-ID: <1BB07850-5E71-4DD2-8469-061E8FA0B2B7@microsoft.com>
|> | MIME-Version: 1.0
|> | Content-Type: text/plain;
|> | charset="Utf-8"
|> | Content-Transfer-Encoding: 7bit
|> | X-Newsreader: Microsoft CDO for Windows 2000
|> | Content-Class: urn:content-classes:message
|> | Importance: normal
|> | Priority: normal
|> | X-MimeOLE: Produced By Microsoft MimeOLE V6.00.3790.0
|> | Newsgroups: microsoft.public.dotnet.framework.clr
|> | NNTP-Posting-Host: TK2MSFTNGXA03.phx.gbl 10.40.1.29
|> | Path: cpmsftngxa10.phx.gbl!TK2MSFTNGXA03.phx.gbl
|> | Xref: cpmsftngxa10.phx.gbl microsoft.public.dotnet.framework.clr:12199
|> | X-Tomcat-NG: microsoft.public.dotnet.framework.clr
|> |
|> |>> not "execute handler" but "continue search" = look for the next
|> filter or
|> |>> catch that might want to handle the initial exception.
|> |>>
|> |>
|> |> That doesn't sound quite right. It doesn't make sense that a nested
|> |> exception's search for a handler would be terminated just because it
|> reaches
|> |> the site of the the original thrown exception.
|> |>
|> | when the 2nd exception reaches in the filter of the original
exception,
|> the

|> | first pass (stackwalk) is complete – for the 2nd exception. And then
the
|> | original exception continues its stackwalk (exactly as if the filter
|> would
|> | have returned 0 = false).
|> |
|> |> If anything I would expect
|> |> the following – the 2nd exception starts a new exception sequence
|> (stackwalk
|> |> looking for a catch handler, executing filter expressions), and
when
|> a
|> |> catch handler for the 2nd is found the original exception is thrown
|> away.
|> |
|> | No. You are still in the filter of the original exception – you cannot
|> | propagate further the 2nd exception. You go ahead evaluating the
filter
|> of
|> | the original exception.
|> |
|> | I will post an example.
|> |
|> |> Also, per your other reply, I'm do not believe that there's a linked
|> list of
|> |> exception events
|> |
|> | I'm thinking so because I see the exception are not lost. You start
the
|> | mechanism for one exception, you encounter another exception but the
|> first is
|> | still saved somewhere – related to what I state in the first part of
|> this
|> | message.
|> |
|> | Regards, George
|> |
|>
|
|
|