Ruby - Feature #8257

Exception#cause to carry originating exception along with new one

04/12/2013 01:40 AM - headius (Charles Nutter)

Status:	Closed	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

Often when a lower-level API raises an exception, we would like to re-raise a different exception specific to our API or library. Currently in Ruby, only our new exception is ever seen by users; the original exception is lost forever, unless the user decides to dig around our library and log it. We need a way to have an exception carry a "cause" along with it.

Java has getCause/setCause and standard constructors that take a cause exception. Printing out an exception's backtrace then reports both that exception and any "cause" exception.

Rubinius has added a similar feature: https://gist.github.com/dbussink/b2e01e51d0c50b27004f

The changes required for this feature are pretty benign:

- Exception#cause and #cause= accessors.
- A new set of Kernel#raise overloads that accept (as a trailing argument, probably) the "cause" exception.
- Modifications to backtrace-printing logic to also display backtrace information from the "cause" exception (and in turn, from any nested "cause" exceptions).

There's some discussion here about alternatives to #cause, none of which are quite as elegant as having it built in:
http://www.skorks.com/2013/04/ruby-why-u-no-have-nested-exceptions/

Related issues:

Related to Ruby - Bug #9338: Build failure of trunk with MSVC 2013	**Closed**	**01/01/2014**

Associated revisions

Revision b050cc5ab9b0774bbd16f49f4787f50b07c566fe - 11/10/2013 01:16 PM - nobu (Nobuyoshi Nakada)

error.c: Exception#cause

- error.c (exc_cause): captured previous exception.
- eval.c (make_exception): capture previous exception automagically.
  [Feature #8257]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@43636 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision b050cc5a - 11/10/2013 01:16 PM - nobu (Nobuyoshi Nakada)

error.c: Exception#cause

- error.c (exc_cause): captured previous exception.
- eval.c (make_exception): capture previous exception automagically.
  [Feature #8257]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@43636 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 549b35c1dc771d233a72466a6cae8363908f3350 - 11/15/2013 02:08 PM - nobu (Nobuyoshi Nakada)

eval.c: refactor exception cause

- eval.c (setup_exception): set up cause after exception to be raised
  is fixed. [Feature #8257]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@43684 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 549b35c1 - 11/15/2013 02:08 PM - nobu (Nobuyoshi Nakada)

eval.c: refactor exception cause

- eval.c (setup_exception): set up cause after exception to be raised is fixed. [Feature #8257]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@43684 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 8a46d4027c5d797eb07afe24d43c37aaa5e33c5d - 12/31/2013 02:49 PM - nobu (Nobuyoshi Nakada)

eval.c: raise with cause

- eval.c (rb_f_raise): add cause: optional keyword argument.
  [ruby-core:58610] [Feature #8257] [EXPERIMENTAL]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@44473 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 8a46d402 - 12/31/2013 02:49 PM - nobu (Nobuyoshi Nakada)

eval.c: raise with cause

- eval.c (rb_f_raise): add cause: optional keyword argument.
  [ruby-core:58610] [Feature #8257] [EXPERIMENTAL]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@44473 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision d689dca633e979bb888a6a287a07e0084a0f5187 - 06/17/2014 03:37 AM - nobu (Nobuyoshi Nakada)

eval.c: pass unknown options

- eval.c (extract_raise_opts): pass unknown options to the
  exception, so that exception class can receive a hash argument.
  [ruby-core:63203] [Feature #8257]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@46456 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision d689dca6 - 06/17/2014 03:37 AM - nobu (Nobuyoshi Nakada)

eval.c: pass unknown options

- eval.c (extract_raise_opts): pass unknown options to the
  exception, so that exception class can receive a hash argument.
  [ruby-core:63203] [Feature #8257]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@46456 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 48af6fd544e0b3cec83af9a629059ff8d304720c - 10/27/2018 09:45 PM - naruse (Yui NARUSE)

Print exception's cause like Java

Print cause of the exception if the exception is not caught and printed
its backtraces and error message [Feature #8257]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@65393 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 48af6fd544e0b3cec83af9a629059ff8d304720c - 10/27/2018 09:45 PM - naruse (Yui NARUSE)

Print exception's cause like Java

Print cause of the exception if the exception is not caught and printed
its backtraces and error message [Feature #8257]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@65393 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 48af6fd5 - 10/27/2018 09:45 PM - naruse (Yui NARUSE)

Print exception's cause like Java

Print cause of the exception if the exception is not caught and printed
its backtraces and error message [Feature #8257]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@65393 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 04/12/2013 04:48 AM - headius (Charles Nutter)

Links to implementation of "cause" rendering and Kernel#raise logic from Rubinius:

"cause" rendering: https://github.com/rubinius/rubinius/blob/master/kernel/common/exception.rb#L72

Kernel#raise: https://github.com/rubinius/rubinius/blob/master/kernel/delta/kernel.rb#L27

Note that the changes almost exclusively affect rendering of exceptions that bubble out; no other visible aspects of exceptions change with this enhancement. This means that libraries can start using the feature without consumers having to change any code.

This feature would also be useful for associate a low-level cause with higher-level exceptions in core classes and standard library, such as wrapping a low-level IO exception in an SSLError or net/* exception.

#2 - 04/13/2013 12:13 AM - rkh (Konstantin Haase)

This would indeed be useful. We have built our own solutions for this many times.

Should there also be a mechanism to rescue based on cause that's transparent of wrapping exceptions?

#3 - 04/13/2013 01:03 AM - headius (Charles Nutter)

rkh (Konstantin Haase) wrote:

> Should there also be a mechanism to rescue based on cause that's transparent of wrapping exceptions?

Exception::=== could be enhanced to search causes, but I'm dubious about the utility of such an enhancement. It could also make exception-handling slower; even though it's obviously *exception* handling and doesn't necessarily need to be lightning fast, people still occasionally use it for flow control inappropriately.

I'd probably punt on that one myself and see over time how common it is that people rescue based on causes. I'd suspect it's pretty rare...such a feature has never been considered for Java (as far as I know) even though Java's Throwable has had "cause" forever.

Also want to avoid feature creep. Exception#cause is a nice, simple, low-impact feature as described by me here.

#4 - 04/14/2013 10:33 PM - rkh (Konstantin Haase)

I think allowing rescue lines to take other objects besides modules would this would greatly ease building something in user code:

def caused_by(matcher)
block = proc do |exception|
matcher === exception or exception.cause && block[exception.cause]
end
end

begin
...
rescue caused_by(NetworkError)
...
end

But I guess that's a separate feature request.

#5 - 04/16/2013 06:03 AM - headius (Charles Nutter)

rkh (Konstantin Haase) wrote:

> I think allowing rescue lines to take other objects besides modules would this would greatly ease building something in user code:
> ...
> But I guess that's a separate feature request.

Yeah, it might be nice, but there might also be optimization concerns. You're right though, it should indeed be a separate feature request, and it's certainly applicable to more than just Exception#cause.

#6 - 04/28/2013 07:23 AM - ko1 (Koichi Sasada)

(2013/04/12 1:40), headius (Charles Nutter) wrote:

> - A new set of Kernel#raise overloads that accept (as a trailing argument, probably) the "cause" exception.

(1) introduce new keyword?

example: raise(..., cause: e)

(2) How about to use `$!' forcibly?

--
// SASADA Koichi at atdot dot net

#7 - 04/28/2013 07:23 AM - ko1 (Koichi Sasada)

(2013/04/28 7:19), SASADA Koichi wrote:

> (2) How about to use `$!' forcibly?

because I think Exception#cause= is not cool.

--
// SASADA Koichi at atdot dot net

#8 - 04/28/2013 08:23 AM - headius (Charles Nutter)

On Sat, Apr 27, 2013 at 5:19 PM, SASADA Koichi ko1@atdot.net wrote:

> (1) introduce new keyword?
>
> example: raise(..., cause: e)

Yeah, that's not bad. Keywords allow us to avoid changing the
signature in new/incompatible/non-forward-compatible ways.

> (2) How about to use `$!' forcibly?

Do you mean automatically stick $! into Exception#cause when creating
(or perhaps better, when raising) a new Exception? That's not a bad
idea.

Interestingly, java.lang.Throwable also does not allow setting the
cause. You can set it during init or after init *at most once*. Here's
JavaDoc for it:
http://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html#initCause(java.lang.Throwable)

I quote: "This method can be called at most once. It is generally
called from within the constructor, or immediately after creating the
throwable. If this throwable was created with Throwable(Throwable) or
Throwable(String,Throwable), this method cannot be called even once."

Java 7 also added an additional mechanism for attaching exceptions,
called "supressed exceptions".
http://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html#addSuppressed(java.lang.Throwable)

This mechanism was added for exceptions that are not really the
cause, but which are related. For example, an IO read operation
raises an error, and while handling that error and attempting to close
the stream a second error is raised. The read error did not cause the
close error, but you don't want to lose either. You raise the close
error with the read error attached as "suppressed".

I think for purposes of this feature, adding cause, which can be
initialized during construction only, would be sufficient. I have no
strong opinion about automatically using $! as the cause, but it might
be nice.

#9 - 04/28/2013 08:23 AM - ko1 (Koichi Sasada)

(2013/04/28 7:59), Charles Oliver Nutter wrote:

> > (2) How about to use `$!' forcibly?
> > Do you mean automatically stick $! into Exception#cause when creating
> > (or perhaps better, when raising) a new Exception? That's not a bad
> > idea.

Yes.

> I think for purposes of this feature, adding cause, which can be
> initialized during construction only, would be sufficient. I have no
> strong opinion about automatically using $! as the cause, but it might
> be nice.

Cool.

Summary:
(1) raise method captures $! as @cause.
(2) Adding new method Exception#cause returns @cause.

- I'm not sure #cause is good name.
  Too short?

--
// SASADA Koichi at atdot dot net

#10 - 04/29/2013 04:23 AM - headius (Charles Nutter)

On Sat, Apr 27, 2013 at 6:09 PM, SASADA Koichi ko1@atdot.net wrote:

> Summary:
> (1) raise method captures $! as @cause.
> (2) Adding new method Exception#cause returns @cause.
>
> - I'm not sure #cause is good name.
>   Too short?

It seems ok to me, since Java's Throwable uses the same name (but I am
perhaps a bit biased.

.NET's Exception calls it the BaseException, via GetBaseException. I
think "base" or "base_exception" not as good as "cause".

I guess the next step is a patch.

- Charlie

#11 - 04/29/2013 04:23 PM - rkh (Konstantin Haase)

I love the idea of having $! be the cause. It would also mean instant adoption.

#12 - 09/27/2013 08:04 PM - headius (Charles Nutter)

Any further comments here? I might be able to do part of the implementation, but I don't know how to automatically stick $! into cause. I'd like to see this in 2.1.

#13 - 10/01/2013 09:39 AM - headius (Charles Nutter)

- Target version set to Ruby 2.1.0

#14 - 10/01/2013 04:54 PM - ko1 (Koichi Sasada)

- Assignee set to matz (Yukihiro Matsumoto)

I'm positive about this feature.

Matz, what do you think about?

#15 - 10/09/2013 10:27 PM - matz (Yukihiro Matsumoto)

Hi,

- Fundamentally accepted.
- I am against #cause=
- It's OK that #raise to have cause: keyword argument to specify cause
- I am not sure automagical capturing of $! would not cause wrong capturing or not

Matz.

#16 - 10/10/2013 12:28 AM - headius (Charles Nutter)

matz (Yukihiro Matsumoto) wrote:

> - Fundamentally accepted.

Hooray!

> - I am against #cause=

java.lang.Throwable does not have #setCause, so I agree. It does have #initCause, which can only be called once (if cause has not already been provided via constructor)...but I have never used it.

> - It's OK that #raise to have cause: keyword argument to specify cause

Agreed. Unfortunately this would still break backward-compatibility, since it will count as an extra argument on older Ruby versions. This might be the biggest justification for $! capturing.

> - I am not sure automagical capturing of $! would not cause wrong capturing or not

There are a few small risks:

- Many layers of exceptions being raised as an error propagates out would all be chained. This could cause more information/data/memory than necessary to be retained.
- Lower layers may not want higher layers to have access to the causing error for security or encapsulation reasons.
- Too much magic?

But I think the benefits may outweigh them:

- I can't think of any concrete examples of the above risks.
- Automatic adoption; all exceptions caused by other exceptions will immediately show their lineage.
- Reduced backward incompatibility (other than adding #cause) since users won't have to use the constructor to capture cause exception (of course, I still think we should have the cause constructor too).

I think the output of exception backtraces should also be enhanced to show cause exception's trace, as in the JVM. This could cause a problem for tools that parse the backtrace, though (IMO nobody should ever parse backtraces and expect that to be stable).

#17 - 11/10/2013 11:55 PM - nobu (Nobuyoshi Nakada)

Automagically captured exceptions doesn't feel \`cause' to me, now.
It might be irrelevant to the previously rescued exception.

#18 - 11/13/2013 03:32 AM - headius (Charles Nutter)

nobu (Nobuyoshi Nakada) wrote:

> Automagically captured exceptions doesn't feel \`cause' to me, now.
> It might be irrelevant to the previously rescued exception.

That's true, but will that be the exception or the rule? It seems to me that most "hidden" or "suppressed" exceptions will be direct triggers for the actually-raised exception.

#19 - 11/13/2013 08:04 AM - henry.maddocks (Henry Maddocks)

How is this different to 'wrapping' an exception? Eg.

http://www.jayway.com/2011/05/25/ruby-an-exceptional-language/

#20 - 11/13/2013 08:46 AM - headius (Charles Nutter)

henry.maddocks (Henry Maddocks) wrote:

> How is this different to 'wrapping' an exception? Eg.
>
> http://www.jayway.com/2011/05/25/ruby-an-exceptional-language/

Not much different, but built-in and automatically available.

#21 - 11/27/2013 05:57 AM - headius (Charles Nutter)

I think we still need to add Exception.new(:cause => ex) to allow constructing an exception with a *specific* cause. The $! capturing is great but often there may be more than one exception in play.

This needs to happen before 2.1 final.

#22 - 12/07/2013 09:14 PM - nobu (Nobuyoshi Nakada)

What about the backward compatibility?

#23 - 12/09/2013 03:09 PM - headius (Charles Nutter)

nobu: Because :cause is opt-in, I am not concerned about the backward compat. You'd have to be committing to 2.1 in order to start using that feature, and I think it will be slow to adopt in any case.

It would also be possible to monkey-patch older versions to just ignore the options.

I think we should add the option for future use, because it *is* needed, and we have to add it some time.

#24 - 12/10/2013 11:24 PM - nobu (Nobuyoshi Nakada)

Carelessly, I implemented raise cause: ex but not Exception.new(cause: ex), and nearly committed it.
Do you prefer the latter?

#25 - 12/11/2013 01:23 PM - ko1 (Koichi Sasada)

(2013/12/10 23:24), nobu (Nobuyoshi Nakada) wrote:

> Carelessly, I implemented raise cause: ex but not Exception.new(cause: ex), and nearly committed it.
> Do you prefer the latter?

How about to simply add Exception#cause= only?

This is because:

- Setting "cause" is not common usecase (nobody care about it)
  Some long steps such as
  e = Exception.new(...)
  e.cause =...
  raise e"
  is acceptable for unusual cases.

- No compatibility issue with raise()

However, matz is against "cause=" by [ruby-core:57773].
Because he doesn't want to introduce any states.
I also agree this point.

--
// SASADA Koichi at atdot dot net

#26 - 12/11/2013 08:09 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Actually, the only reason I don't use it is because it's not possible, but I'd most probably always send along the cause if this was supported, so I prefer a concise way to do that.

#27 - 12/11/2013 08:14 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

And sometimes a Runtime exception would be fine to me, so raise "Some explanation", cause: ex would be great, but sometimes I need to wrap the original exception in some specific one. In that latter case, I'd prefer to be able to specify the cause with CustomException.new("message", cause: ex). I'm usually interested in keeping the full backtrace in a simple way when doing that...

#28 - 12/11/2013 09:08 PM - nobu (Nobuyoshi Nakada)

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

> sometimes I need to wrap the original exception in some specific one.

Just wrapping in a new exception but don't raise it?
How frequent is such case?

#29 - 12/12/2013 12:28 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Yes, raising it too.

#30 - 12/12/2013 12:33 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

I believe you think I should write this instead:

raise WrappedException.new("message"), cause: ex

I wouldn't mind doing that but if I ever had to store the exception without raising it then it wouldn't be possible to do so, right? But in that case I could also store the cause separately, so it isn't really a big deal.

I wouldn't mind to add cause just to raise...

#31 - 12/12/2013 05:00 PM - nobu (Nobuyoshi Nakada)

rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

> I believe you think I should write this instead:
>
> raise WrappedException.new("message"), cause: ex

I think this is more common in ruby:
raise WrappedException, "message", cause: ex

#32 - 12/26/2013 05:23 AM - avdi (Avdi Grimm)

I've been digging into this feature now that 2.1 is released, and I'm a
little confused. In the final implementation, can #cause only be set from
$!? None of the other methods for setting the cause discussed in this
thread seem to work, and for that matter I don't see any changes to the
#raise or Exception#initialize methods to support explicitly setting a
cause.

#33 - 12/30/2013 12:19 AM - headius (Charles Nutter)

Unfortunately it doesn't look like anything other than $! logic for this made it into 2.1. I was hoping we'd get either the constructor or a one-time-only #cause= but there was still some debate about which way to go.

Since the base #cause and $! logic made it in, perhaps we should call this bug closed as of 2.1 and add a new bug for additional ways to initialize the exception cause.

#34 - 02/02/2014 03:10 PM - Eregon (Benoit Daloze)

raise ErrorClass, msg, cause: cause was implemented with the rest in r44473.

```
e = (
begin
  raise ArgumentError, "arg error"
rescue
  nie = NotImplementedError.new("nie")
  raise StandardError, "stderr", cause: nie
end) rescue $!
e.cause # => #<NotImplementedError: nie>
```

But the cause is not shown in the error output, which I think is now the most important step forward.

#35 - 02/05/2014 03:48 AM - usa (Usaku NAKAMURA)

- Related to Bug #9338: Build failure of trunk with MSVC 2013 added

#36 - 06/17/2014 02:34 AM - naruse (Yui NARUSE)

Vagrant hits the incompatibility:
https://github.com/mitchellh/vagrant/blob/master/plugins/providers/virtualbox/driver/base.rb#L347

```
 raise Vagrant::Errors::VBoxManageError,
                 :command => command.inspect,
                 :stderr  => r.stderr
```

it raies ArgumentError: unknown keyword: command, stderr.

#37 - 06/17/2014 03:37 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

- % Done changed from 0 to 100

Applied in changeset ruby-trunk:r46456.

---

eval.c: pass unknown options

- eval.c (extract_raise_opts): pass unknown options to the
  exception, so that exception class can receive a hash argument.
  [ruby-core:63203] [Feature #8257]

#38 - 07/20/2014 08:20 PM - headius (Charles Nutter)

I agree with Benoit that the cause exception should be reflected in the error output, as on the JVM. What are the objections?

I do *not* believe this should show up in #backtrace since I have no idea how it would be formatted. This also lines up with the JVM (printStackTrace and friends print cause, but getStackTrace only returns trace for target exception).

#39 - 07/21/2014 01:23 PM - ko1 (Koichi Sasada)

(2014/07/21 1:50), headius@headius.com wrote:

> I agree with Benoit that the cause exception should be reflected in the error output, as on the JVM. What are the objections?

What happen on exception from deep backtrace, occurred by other more
deeper exception? Show all long two backtraces?

--
// SASADA Koichi at atdot dot net

#40 - 12/18/2017 12:35 AM - peterfaiman (Peter Faiman)

Is there a reason not to do caused-by stack trace printing? Or has it just not been implemented by anyone yet?

#41 - 12/20/2017 05:35 PM - Eregon (Benoit Daloze)

ko1 (Koichi Sasada) wrote:

> What happen on exception from deep backtrace, occurred by other more
> deeper exception? Show all long two backtraces?

Yes, although we can omit the common part of the backtrace in the cause backtraces like in Java:
http://www.codejava.net/java-core/exception/understanding-exception-stack-trace-in-java-with-code-examples

peterfaiman (Peter Faiman) wrote:

> Is there a reason not to do caused-by stack trace printing? Or has it just not been implemented by anyone yet?

I believe it was just no implemented yet.
@ko1 (Koichi Sasada) Who should we assign this to?

#42 - 12/23/2021 11:41 PM - hsbt (Hiroshi SHIBATA)

- Project changed from 14 to Ruby

- Target version deleted (Ruby 2.1.0)