Ruby - Feature #21359

Introduce `Exception#cause=` for Post-Initialization Assignment

05/22/2025 08:39 AM - ioquatix (Samuel Williams)

Status:	Assigned
Priority:	Normal
Assignee:	ioquatix (Samuel Williams)
Target version:	

Description

Ruby currently allows an exception's cause to be explicitly set **only at the time of raising** using raise ..., cause: .... However, there are valid use cases where it would be convenient to set the cause when creating an exception.

The Problem

Ruby supports exception chaining via the cause: keyword during a raise, like so:

```
raise StandardError.new("failure"), cause: original_exception
```

Proposed Solution

Introduce Exception#cause= as a public setter method on Exception, allowing post-initialization assignment of the cause:

```
cause = RuntimeError.new("low-level error")
error = StandardError.new("higher-level error")
error.cause = cause
```

It would be semantically equivalent to the following implementation:

```
error = StandardError.new("error")
cause = RuntimeError.new("cause")

class Exception
  def cause= value
    backtrace = self.backtrace_locations

    raise self, cause: value
  rescue Exception
    self.set_backtrace(backtrace)
  end
end

p error.cause # nil
error.cause = cause
p error.cause # => #<RuntimeError: cause>
```

Benefits

- Simplifies creation of rich exception objects in frameworks, tools, and tests.
- Enables structured error chaining in asynchronous and deferred execution environments.
- Avoids misuse of raise/rescue for control flow and metadata setting.

Related issues:

Related to Ruby - Bug #21360: Inconsistent Support for `Exception#cause` in `...	**Open**

History

#1 - 05/22/2025 10:15 AM - Eregon (Benoit Daloze)

Do you have real-world examples where you would use this, e.g. in gems?

#2 - 05/22/2025 12:12 PM - ioquatix (Samuel Williams)

Yes, in Async, I want to set the cause of an exception before raising it later on a fiber.

Additionally, serialisation and deserialisation of exceptions is almost impossible without being able to set the cause, e.g. any kind of Ruby RPC.

#3 - 05/22/2025 07:55 PM - Eregon (Benoit Daloze)

- Related to Bug #21360: Inconsistent Support for `Exception#cause` in `Fiber#raise` and `Thread#raise` added

#4 - 05/22/2025 08:00 PM - Eregon (Benoit Daloze)

ioquatix (Samuel Williams) wrote in #note-2:

> Yes, in Async, I want to set the cause of an exception before raising it later on a fiber.

#21360 would be enough for that, although you'd need to wrap the exception + cause-to-be in some extra object.

> Additionally, serialisation and deserialisation of exceptions is almost impossible without being able to set the cause, e.g. any kind of Ruby RPC.

WDYM by almost impossible?
Causes can't be cyclic so that's not an issue (which BTW means we would need to check that in this assignment method).
Marshal can do it I guess.
And if not raise self, cause: value + rescue but I agree that's hacky and inefficient.

Serializing exceptions properly without Marshal is probably quite hard yes, not only about the cause but also the internal backtrace representation, the backtrace_locations objects, other internal state for core exceptions that can't always be set in constructor, etc.
But is there a need for that?

#5 - 05/22/2025 11:44 PM - ioquatix (Samuel Williams)

> Serializing exceptions properly without Marshal is probably quite hard yes, not only about the cause but also the internal backtrace representation, the backtrace_locations objects, other internal state for core exceptions that can't always be set in constructor, etc. But is there a need for that?

Yes, serialisation with msgpack is painful and impossible to do correctly in some situations, if you have custom serialisation rules for certain object types (using a msgpack factory), you can't use Marshal.dump as part of msgpack's serialisation because it won't correctly serialise the internal objects of the exception.

#6 - 05/23/2025 02:08 PM - Eregon (Benoit Daloze)

msgpack doesn't seem to handle recursive data structures:

```
irb(main):003> a=[]
=> []
irb(main):004> a<<a
=> [[...]]
irb(main):005> a
=> [[...]]
irb(main):006> a.to_msgpack
msgpack-1.8.0/lib/msgpack.rb:41:in `write': stack level too deep (SystemStackError)
```

So I think it's unrealistic to try to serialize/deserialize exceptions or complex objects with msgpack, it seems really meant for non-recursive simple data.

#7 - 05/27/2025 01:39 PM - zzak (zzak _)

Another thing to consider is we don't have Exception#backtrace= but instead Exception#set_backtrace.

So an alternative is Exception#set_cause or Exception#set_backtrace(cause:). I guess.

#8 - 05/28/2025 05:07 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned

#9 - 06/05/2025 09:07 AM - matz (Yukihiro Matsumoto)

I am basically against cause= (or set_cause that is). It makes exceptions more complex. Is it impossible to defer exception creation til it is absolutely needed?

Matz.

#10 - 06/05/2025 09:25 AM - Eregon (Benoit Daloze)

Maybe Exception.new("message", cause: cause) should be supported, but I suppose that might be quite hard to support for all subclasses of Exception.
That's probably the reason it's on raise and not in exception constructor.
BTW it's also raise that sets the (initial) backtrace of an exception, not the exception constructor (unlike in Java).

Maybe Exception#exception could accept a cause: kwarg?