Ruby master - Feature #10042

Deprecate postfix rescue syntax for removal in 3.0

07/15/2014 11:58 PM - headius (Charles Nutter)

Status:	Open	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	Next Major	

Description

The postfix rescue notation is convenient...but almost always is a really bad antipattern.

An example of the notation:

```
Integer(f) rescue f # returns f if it is not parseable as an Integer
```

It silently ignores all StandardError raised by a piece of code...which often covers *many* more exceptions than the user *wants* to be ignoring.

It also hides the cost of constructing and throwing away all those ignored exceptions.

I believe Matz has even said in the past that he regrets adding the feature.

In any case, I propose that "rescue nil" should be deprecated with a warning (either always on or only when verbose) and we should plan to remove it in 3.0.

Who's with me?!

Related issues:

Related to Ruby master - Feature #6739: One-line rescue statement should supp...	**Feedback**

History

#1 - 07/23/2014 06:20 PM - sowieso (So Wieso)

Hi, I'm against it :-)
Nobody is forced to use it, and for short scripts it can be quite useful.
For example to fetch some websites, if they exist:

```
require 'open-uri'
urls = …
urls.each do |url|
  open(url) do
    #save the file
  end rescue nil
end
```

But it would be ok for me if we need to specify the Exception.

#2 - 07/25/2014 06:29 PM - naruse (Yui NARUSE)

I sometimes use postfix rescue, and often want a new syntax to specify exceptions to rescue.

#3 - 07/25/2014 10:13 PM - phluid61 (Matthew Kerwin)

On 26/07/2014, naruse@airemix.jpnaruse@airemix.jp wrote:

> Issue #10042 has been updated by Yui NARUSE.
>
> I sometimes use postfix rescue, and often want a new syntax to specify
> exceptions to rescue.

This was a feature of MOO Code, on which I cut my teeth in the 90s. It lead me to create the 'try' gem. I would not be disappointed if the gem was made obsolete by a new language feature.

- https://rubygems.org/gems/try
- https://github.com/phluid61/ruby-experiments/tree/master/try

--
Matthew Kerwin
http://matthew.kerwin.net.au/

#4 - 07/26/2014 05:49 AM - nobu (Nobuyoshi Nakada)

- Tracker changed from Bug to Feature

#5 - 09/27/2014 07:32 AM - Anonymous

Nooo! Don't remove tail rescue! I suggested to introduce tail "resc" keyword with exception to be rescued somewhere here.

#6 - 09/27/2014 07:35 AM - Anonymous

Got it, this is what I proposed:

```
do_messy_job resc TypeError: 42, NameError: 43
```

#7 - 11/11/2014 02:13 AM - headius (Charles Nutter)

The problem is that postfix rescue is almost always used incorrectly, ending up swallowing 99% of errors that Ruby programs raise. It also masks the very high cost of creating an exception and backtrace (high on MRI, higher on Rubinius, and very high on JRuby/JVM). Finding these bugs (or performance hits) can be very difficult.

I believe the potential for serious bugs outweighs the convenience of this syntax.

#8 - 11/11/2014 02:18 AM - headius (Charles Nutter)

At the very least, we should introduce a syntax for rescuing a specific exception type, and warn when users don't use that syntax. The syntax proposed by Boris isn't too bad.

I wonder if there's a way we could make postfix rescue turn off backtraces downstream. If downstream code passes through another rescue, the optimization would be turned off because that rescue might want the trace.

#9 - 11/11/2014 03:25 AM - matz (Yukihiro Matsumoto)

- Assignee set to matz (Yukihiro Matsumoto)

- Target version changed from 2.6 to Next Major

Hedius, separate your concern, performance and language design.

I haven't seen any "serious" problem caused by exceptions swallowed by postfix "rescue". So I don't worry too much about the issue.

Of course specifying exception class reduce chance for misbehavior, so if some one come up with nice idea, I'd love to merge and encourage it. But adding new reserved word is unacceptable. Any idea?

I don't care much about the performance here. Adding new syntax won't solve the issue.
We can just discourage use of exceptions in tight loops in the documentations.

Matz.

#10 - 11/11/2014 02:43 PM - recursive-madman (Recursive Madman)

An alternative syntax that doesn't introduce a new reserved word would be:

```
do_something_messy rescue(TypeError, NameError) do_something_with($!)
```

The syntax within the parentheses could be the same as after a block rescue, e.g.:

```
do_a rescue(SomeError => e) do_b(e)
```

This doesn't introduce a new reserved word and causes a syntax error on older ruby versions (thus preventing existing code from suddenly working differently).

#11 - 11/11/2014 03:41 PM - nobu (Nobuyoshi Nakada)

Recursive Madman wrote:

> An alternative syntax that doesn't introduce a new reserved word would be:
>
> ```
> do_something_messy rescue(TypeError, NameError) do_something_with($!)
> ```

It conflicts with existing code for single exception class.

I thought a syntax like rescue < exception, but can't fix conflicts yet.

#12 - 11/11/2014 04:00 PM - recursive-madman (Recursive Madman)

Nobuyoshi Nakada wrote:

> Recursive Madman wrote:
>
> > An alternative syntax that doesn't introduce a new reserved word would be:
> >
> > ```
> > do_something_messy rescue(TypeError, NameError) do_something_with($!)
> > ```
>
> It conflicts with existing code for single exception class.

Could you provide an example?

#13 - 11/11/2014 05:58 PM - nobu (Nobuyoshi Nakada)

Currently,

```
do_something_messy rescue(TypeError)
```

is valid code, and TypeError is a expression to be returned when an exception raised.

With your proposal, it becomes an exception class to be rescued.
That means the meaning of that parenthesized part changes by if it has succeeding expression.
It would result just another confusion, IMHO.

And I'm afraid that it might not be able to parse correctly.

#14 - 11/12/2014 08:19 AM - recursive-madman (Recursive Madman)

I see how it would conflict when both forms (do_a rescue(TypeError) and do_a rescue(TypeError) do_b) are supported (the parser can't decide between recognizing parenthesized exceptions or stmt), but when the former form (i.e. not specifying an exception) is no longer allowed, do_a rescue(TypeError) becomes invalid code.

#15 - 11/12/2014 02:38 PM - avdi (Avdi Grimm)

On Mon, Nov 10, 2014 at 10:25 PM, matz@ruby-lang.org wrote:

> I haven't seen any "serious" problem caused by exceptions swallowed by
> postfix "rescue". So I don't worry too much about the issue.

I won't claim that writing a book on Ruby exceptions makes me an expert. But at least I've made a special study of them for the past decade or so of Ruby use, and over numerous large-scale client projects. Of bugs that could be traced to use of a language feature, postfix rescue is easily in the top three. In fact, offhand I'm not sure I can think of *any* language feature whose use has led directly to more problems than this one. Unless you count "the existence of nil" as a language feature ;-)

Postfix rescue bugs are especially insidious, because they usually hide *other* bugs.

Can you accidentally throw away exceptions without it? Sure. But other methods are more visually obvious. Postfix rescue hides out at the ends of long lines in legacy codebases, quietly breaking things.

The abuse of postfix rescue is so bad in Ruby projects that at this point I consider only one usage of it acceptable: rescue $! for converting exceptions to returns. I flag any other usage as an error, because if it isn't causing problems already, it almost certainly will eventually.

I would like to see some equivalent of rescue $! stay in the language, but other than that I'd be more than happy to see it leave the building entirely.

--
Avdi Grimm
http://avdi.org

#16 - 02/06/2018 08:53 PM - Quintus (Marvin Gülker)

Please add as related: https://bugs.ruby-lang.org/issues/6739

That's a proposal to eliminate the problem described here (antipattern usage because it swallows exceptions) by extending the one-line rescue statement with a possibility to catch exception classes. There wasn't a solution to a proper syntax yet, but I don't think it's required to go through all of that again if we have it in #6739.

Marvin

#17 - 02/07/2018 01:14 AM - duerst (Martin Dürst)

- Related to Feature #6739: One-line rescue statement should support specifying an exception class added