Ruby master - Feature #1122

request for: Object#try

02/07/2009 01:52 AM - authorNari (Narihiro Nakamura)

Status:	Rejected
Priority:	Normal
Assignee:	authorNari (Narihiro Nakamura)
Target version:	2.0.0

Description

=begin
Hi.

Object#try is new feature of rails2.3.

http://guides.rubyonrails.org/2_3_release_notes.html
http://github.com/rails/rails/blob/master/activesupport/lib/active_support/core_ext/try.rb
http://ozmm.org/posts/try.html

Matz said: "OK, good name is the last problem."

Anyone has better name?

thanks.

Narihiro Nakamura
=end

Related issues:

Related to Ruby master - Feature #11034: Nil Conditional	**Closed**
Related to Ruby master - Feature #11537: Introduce "Safe navigation operator"	**Closed**
Has duplicate Ruby master - Feature #8191: Short-hand syntax for duck-typing	**Closed**

History

#1 - 02/07/2009 02:41 AM - murphy (Kornelius Kalnbach)

=begin
Narihiro Nakamura wrote:

> Object#try is new feature of rails2.3.
> I mostly use @person.name rescue nil -^ #try is nice.
>
> Matz said: "OK, good name is the last problem." Anyone has better
> name?
> since IO and Groovy use a syntax with ?, what about Object#send? ?

@person.send?(:name)

but I think #try is the better name. send? also looks like it returns a
boolean.

#attempt would be synonymic, but it is an ugly word.

#try might be confused with "try...except" in Java and other
languages...but Ruby already uses Java keywords in a different way (new,
case, extend(s), super), so I wouldn't worry about it.

[murphy]

=end

#2 - 02/07/2009 10:48 PM - mame (Yusuke Endoh)

=begin
Hi,

2009/2/7 Narihiro Nakamura redmine@ruby-lang.org:

> Feature #1122: request for: Object#try
> http://redmine.ruby-lang.org/issues/show/1122
>
> Author: Narihiro Nakamura
> Status: Open, Priority: Normal
>
> Hi.
>
> Object#try is new feature of rails2.3.
>
> http://guides.rubyonrails.org/2_3_release_notes.html
> http://github.com/rails/rails/blob/master/activesupport/lib/active_support/core_ext/try.rb
> http://ozmm.org/posts/try.html
>
> Matz said: "OK, good name is the last problem."
>
> Anyone has better name?

Interesting. I think `try' is good enough for me as non-native.

But why does it take Object#send-like interface?
I dislike Object#send because complex modifications are needed:

foo.bar(baz) -> foo.send(:bar, baz)

I have to type "send(:", move the cursor, delete "(", and type ", ".
What a pain!

Of course, it can't be helped in the case of Object#send.
But there is no reason for Object#try to follow it.
I prefer:

foo.bar(baz) -> foo.try.bar(baz)

because it is much simpler (only to insert ".try")

For example:

x = nil; p x.try + 1 #=> nil
x = 100; p x.try + 1 #=> 101

Implementation image:

```
class Object
def try
self
end
end

class NilClass
def try
obj = BasicObject.new
def obj.method_missing(*)
nil
end
obj
end
end
```

--
Yusuke ENDOH mame@tsg.ne.jp

=end

#3 - 02/07/2009 11:43 PM - dblack (David Black)

=begin
Hi --

On Sat, 7 Feb 2009, Yusuke ENDOH wrote:

> Hi,

> 2009/2/7 Narihiro Nakamura redmine@ruby-lang.org:
>
>> Feature #1122: request for: Object#try
>> http://redmine.ruby-lang.org/issues/show/1122
>>
>> Author: Narihiro Nakamura
>> Status: Open, Priority: Normal
>>
>> Hi.
>>
>> Object#try is new feature of rails2.3.
>>
>> http://guides.rubyonrails.org/2_3_release_notes.html
>> http://github.com/rails/rails/blob/master/activesupport/lib/active_support/core_ext/try.rb
>> http://ozmm.org/posts/try.html
>>
>> Matz said: "OK, good name is the last problem."
>>
>> Anyone has better name?
>
> Interesting. I think `try' is good enough for me as non-native.
>
> But why does it take Object#send-like interface?
> I dislike Object#send because complex modifications are needed:
>
> foo.bar(baz) -> foo.send(:bar, baz)
>
> I have to type "send(:", move the cursor, delete "(", and type ", ".
> What a pain!
>
> Of course, it can't be helped in the case of Object#send.
> But there is no reason for Object#try to follow it.
> I prefer:
>
> foo.bar(baz) -> foo.try.bar(baz)
>
> because it is much simpler (only to insert ".try")

I don't think that's a big issue, because you'd probably either use
try or not, and not change it back and forth.

> For example:
>
> x = nil; p x.try + 1 #=> nil
> x = 100; p x.try + 1 #=> 101

I prefer the send-like way. "try" without an argument doesn't make
sense to me here; it leaves me asking, "Try *what*?" The idea that
"try" in the abstract means "self or a BasicObject that returns nil
for missing methods" isn't a good fit, I think.

David

--
David A. Black / Ruby Power and Light, LLC
Ruby/Rails consulting & training: http://www.rubypal.com
Coming in 2009: The Well-Grounded Rubyist (http://manning.com/black2)

http://www.wishsight.com => Independent, social wishlist management!

=end

#4 - 02/08/2009 12:32 PM - mame (Yusuke Endoh)

=begin
Hi,

2009/2/7 David A. Black dblack@rubypal.com:

> Hi --
>
> On Sat, 7 Feb 2009, Yusuke ENDOH wrote:

>> Hi,
>>
>> 2009/2/7 Narihiro Nakamura redmine@ruby-lang.org:
>>
>>> Feature #1122: request for: Object#try
>>> http://redmine.ruby-lang.org/issues/show/1122
>>>
>>> Author: Narihiro Nakamura
>>> Status: Open, Priority: Normal
>>>
>>> Hi.
>>>
>>> Object#try is new feature of rails2.3.
>>>
>>> http://guides.rubyonrails.org/2_3_release_notes.html
>>>
>>> http://github.com/rails/rails/blob/master/activesupport/lib/active_support/core_ext/try.rb
>>> http://ozmm.org/posts/try.html
>>>
>>> Matz said: "OK, good name is the last problem."
>>>
>>> Anyone has better name?
>>
>> Interesting. I think `try' is good enough for me as non-native.
>>
>> But why does it take Object#send-like interface?
>> I dislike Object#send because complex modifications are needed:
>>
>> foo.bar(baz) -> foo.send(:bar, baz)
>>
>> I have to type "send(:", move the cursor, delete "(", and type ", ".
>> What a pain!
>>
>> Of course, it can't be helped in the case of Object#send.
>> But there is no reason for Object#try to follow it.
>> I prefer:
>>
>> foo.bar(baz) -> foo.try.bar(baz)
>>
>> because it is much simpler (only to insert ".try")
>
> I don't think that's a big issue, because you'd probably either use
> try or not, and not change it back and forth.

I don't expect that we will do it *back and forth*, but I think
that *forth* will often be needed.

In addition, I feel that the two styles are oddly different in
spite of almost the same meanings.
It is allowable for Object#send because there is no other way
and such a reflection feature is not so often used. But, try
will be routinely used, and thus should not emulate such a odd
style unless it really has to.

>> For example:
>>
>> x = nil; p x.try + 1 #=> nil
>> x = 100; p x.try + 1 #=> 101
>
> I prefer the send-like way. "try" without an argument doesn't make
> sense to me here; it leaves me asking, "Try *what*?" The idea that
> "try" in the abstract means "self or a BasicObject that returns nil
> for missing methods" isn't a good fit, I think.

Hmm...

BTW, some time ago, if I'm correct, nobu suggested a new syntax:

foo.?bar(baz)

as a syntax sugar to:

foo.respond?(:bar) ? foo.bar(baz) : nil

.
I prefer it too. What do you think about it?

--
Yusuke ENDOH mame@tsg.ne.jp

=end

#5 - 02/08/2009 01:24 PM - hramrach (Michal Suchanek)

=begin
On 08/02/2009, Yusuke ENDOH mame@tsg.ne.jp wrote:

> Hi,
>
> 2009/2/7 David A. Black dblack@rubypal.com:
>
>> Hi --
>>
>> On Sat, 7 Feb 2009, Yusuke ENDOH wrote:
>>
>>> Hi,
>>>
>>> 2009/2/7 Narihiro Nakamura redmine@ruby-lang.org:
>>>
>>>> Feature #1122: request for: Object#try
>>>> http://redmine.ruby-lang.org/issues/show/1122
>>>>
>>>> Author: Narihiro Nakamura
>>>> Status: Open, Priority: Normal
>>>>
>>>> Hi.
>>>>
>>>> Object#try is new feature of rails2.3.
>>>>
>>>> http://guides.rubyonrails.org/2_3_release_notes.html
>>>>
>>>> http://github.com/rails/rails/blob/master/activesupport/lib/active_support/core_ext/try.rb
>>>> http://ozmm.org/posts/try.html
>>>>
>>>> Matz said: "OK, good name is the last problem."
>>>>
>>>> Anyone has better name?
>>>
>>> Interesting. I think `try' is good enough for me as non-native.
>>>
>>> But why does it take Object#send-like interface?
>>> I dislike Object#send because complex modifications are needed:
>>>
>>> foo.bar(baz) -> foo.send(:bar, baz)
>>>
>>> I have to type "send(:", move the cursor, delete "(", and type ", ".
>>> What a pain!
>>>
>>> Of course, it can't be helped in the case of Object#send.
>>> But there is no reason for Object#try to follow it.
>>> I prefer:
>>>
>>> foo.bar(baz) -> foo.try.bar(baz)
>>>
>>> because it is much simpler (only to insert ".try")
>>
>> I don't think that's a big issue, because you'd probably either use
>> try or not, and not change it back and forth.
>
> I don't expect that we will do it *back and forth*, but I think
> that *forth* will often be needed.

In addition, I feel that the two styles are oddly different in
spite of almost the same meanings.
It is allowable for Object#send because there is no other way
and such a reflection feature is not so often used. But, try
will be routinely used, and thus should not emulate such a odd
style unless it really has to.

For example:

```
x = nil; p x.try + 1 #=> nil
x = 100; p x.try + 1 #=> 101
```

I prefer the send-like way. "try" without an argument doesn't make
sense to me here; it leaves me asking, "Try *what*?" The idea that
"try" in the abstract means "self or a BasicObject that returns nil
for missing methods" isn't a good fit, I think.

Hmm...

BTW, some time ago, if I'm correct, nobu suggested a new syntax:

foo.?bar(baz)

as a syntax sugar to:

foo.respond?(:bar) ? foo.bar(baz) : nil

.
I prefer it too. What do you think about it?

That looks pretty good except it would need parser changes, and would
possibly make method names previously valid into invalid ones.

You could probably change the interface to make it possible to do
something like:

foo.try_method(:bar).call(baz) which allows calling with the same argument list.

If the method does not exist it should obviously return a method that
just junks any arguments given.

It's much longer than

foo.try(:bar,baz)

but it makes it clear *what* is tried. It also makes it clear what is
rescued in a more complex expression avoiding the "rescue nil"
catching exceptions that were not expected.

However, it does not help the rails case at all because the intended use is

foo.try(:name)

with no arguments which expands into

foo.try_method(:name).call()

.. way too long for the simple case with no arguments.

However, I often get expressions where this does not help - the
problem is I do some actual arithmetic operation which fails if one
argument is nil.

I should scan my code for these because I am not sure if something like

"a".quiet + foo

would help much - how many "quiet" I would need for this to work in a
typical case.

Thanks

Michal

=end

#6 - 02/08/2009 07:59 PM - dblack (David Black)

=begin
Hi--

On Sun, 8 Feb 2009, Yusuke ENDOH wrote:

> Hi,
>
> 2009/2/7 David A. Black dblack@rubypal.com:
>
>> Hi --
>>
>> On Sat, 7 Feb 2009, Yusuke ENDOH wrote:
>>
>>> Hi,
>>>
>>> 2009/2/7 Narihiro Nakamura redmine@ruby-lang.org:
>>>
>>>> Feature #1122: request for: Object#try
>>>> http://redmine.ruby-lang.org/issues/show/1122
>>>>
>>>> Author: Narihiro Nakamura
>>>> Status: Open, Priority: Normal
>>>>
>>>> Hi.
>>>>
>>>> Object#try is new feature of rails2.3.
>>>>
>>>> http://guides.rubyonrails.org/2_3_release_notes.html
>>>>
>>>> http://github.com/rails/rails/blob/master/activesupport/lib/active_support/core_ext/try.rb
>>>> http://ozmm.org/posts/try.html
>>>>
>>>> Matz said: "OK, good name is the last problem."
>>>>
>>>> Anyone has better name?
>>>
>>> Interesting. I think `try' is good enough for me as non-native.
>>>
>>> But why does it take Object#send-like interface?
>>> I dislike Object#send because complex modifications are needed:
>>>
>>> foo.bar(baz) -> foo.send(:bar, baz)
>>>
>>> I have to type "send(:", move the cursor, delete "(", and type ", ".
>>> What a pain!
>>>
>>> Of course, it can't be helped in the case of Object#send.
>>> But there is no reason for Object#try to follow it.
>>> I prefer:
>>>
>>> foo.bar(baz) -> foo.try.bar(baz)
>>>
>>> because it is much simpler (only to insert ".try")
>>
>> I don't think that's a big issue, because you'd probably either use
>> try or not, and not change it back and forth.
>
> I don't expect that we will do it *back and forth*, but I think
> that *forth* will often be needed.

I think in most cases you'd know whether you're going to do this
before you start, so you would rarely have to change it.

> In addition, I feel that the two styles are oddly different in
> spite of almost the same meanings.

> It is allowable for Object#send because there is no other way
> and such a reflection feature is not so often used. But, try
> will be routinely used, and thus should not emulate such a odd
> style unless it really has to.
>
>> I prefer the send-like way. "try" without an argument doesn't make
>> sense to me here; it leaves me asking, "Try *what*?" The idea that
>> "try" in the abstract means "self or a BasicObject that returns nil
>> for missing methods" isn't a good fit, I think.
>
> Hmm...
>
> BTW, some time ago, if I'm correct, nobu suggested a new syntax:
>
> foo.?bar(baz)
>
> as a syntax sugar to:
>
> foo.respond?(:bar) ? foo.bar(baz) : nil
>
> .
> I prefer it too. What do you think about it?

It's a bit punctuation-heavy but I think it's preferable. Of course
the whole thing leaves the question of how to handle ambiguous nils
(getting nil and not knowing whether it's because there was no method
or because the method returned nil). Several years ago I tried to
explore the idea of a "NOACK" response, which was not nil and not an
exception but which meant the object didn't know how to handle the
message. But that didn't work, because how do you know whether or not
to raise an exception? (I'm still interested if anyone has figured out
a way to do it.)

David

--
David A. Black / Ruby Power and Light, LLC
Ruby/Rails consulting & training: http://www.rubypal.com
Coming in 2009: The Well-Grounded Rubyist (http://manning.com/black2)

http://www.wishsight.com => Independent, social wishlist management!

=end

#7 - 02/08/2009 08:01 PM - dblack (David Black)

=begin
Hi --

On Sun, 8 Feb 2009, Roger Pack wrote:

>>>> Anyone has better name?
>
> For me try's name is good but not perfect.
>
> There are a few more options described at [1]. if_not_nil is one.
> andand is one.
>
> I'd probably prefer .try.method(args) to .try(:method, args) since it
> reads and codes more like a normal method call.
>
> I am assuming that try should protect against calling method on nil,
> not protect against calling methods that don't exist as
> instance_methods? That would make sense as it would allow for
> method_missing to define new methods on classes, etc. I suppose you
> could do both as separate methods, or have .try.method(args) do one
> thing and .try(:method, args) do another.
>
> One thing I "wish" were that you could do
> nilorstring.try.a.b.c.d # that allowed sub methods.

They're not really sub, though; they're just further messages. I think
if you have a whole bunch of things you might want to do with an
object conditionally, it's best to set up an if statement. Otherwise
it all gets very magic-dot-ish and obscure.

> re:
>
>> foo.?bar(baz)
>> as a syntax sugar to:
>> foo.respond?(:bar) ? foo.bar(baz) : nil
>
> One suggestion to a ruby quiz was the method name "_?" [2]
> Maybe foo.?.bar(baz) could mean foo.try.bar(baz)?

That's awfully punctuation-heavy.

David

--
David A. Black / Ruby Power and Light, LLC
Ruby/Rails consulting & training: http://www.rubypal.com
Coming in 2009: The Well-Grounded Rubyist (http://manning.com/black2)

http://www.wishsight.com => Independent, social wishlist management!

=end

#8 - 02/11/2009 07:43 AM - dblack (David Black)

=begin
Hi --

On Wed, 11 Feb 2009, Michal Babej wrote:

> Hi,
>
> On Sunday 08 of February 2009 11:58:18 David A. Black wrote:
>
>> It's a bit punctuation-heavy but I think it's preferable. Of course
>> the whole thing leaves the question of how to handle ambiguous nils
>> You could slightly modify it to be
>
> foo.?bar(*args) : &block
>
> as a syntax sugar to:
>
> foo.respond?(:bar) ? foo.bar(args) : yield args
>
> Or something similar... though i'd probably look awful if you wanted to pass a
> block to foo.bar :)

Sorry but that's syntactic vinegar :-)

David

--
David A. Black / Ruby Power and Light, LLC
Ruby/Rails consulting & training: http://www.rubypal.com
Coming in 2009: The Well-Grounded Rubyist (http://manning.com/black2)

http://www.wishsight.com => Independent, social wishlist management!

=end

#9 - 02/11/2009 09:33 AM - radarek (Radosław Bułat)

=begin
Providing new syntax change for such a small thing is IMHO
unnecessary. New method is good thing (because it can be easily
monkey-patched when necessary).

--
Pozdrawiam

Radosław Bułat
http://radarek.jogger.pl - mój blog

=end

#10 - 02/16/2009 07:28 PM - radarek (Radosław Bułat)

=begin
2009/2/15 Yehuda Katz wycats@gmail.com:

> Count me in as a +1 on foo.?bar(baz).

Don't you think that changes like this makes ruby more perlish? I
thought that ruby is trying to unleash from perl-style and such a
things do opposite. '?' have currently 3 meanings:
1) a ? b : c
2) ?a literal
3) if o.foo?

Do you want 4th overloading for '?' ?.
People, ruby have currently one of the most complicated grammar (I'm
not complain). Putting more things like this don't make things easier.

--
Pozdrawiam

Radosław Bułat
http://radarek.jogger.pl - mój blog

=end

#11 - 02/19/2009 03:10 AM - radarek (Radosław Bułat)

=begin
On Wed, Feb 18, 2009 at 6:29 PM, Roger Pack rogerdpack@gmail.com wrote:

>> IMHO, foo.?bar should behave as "call-except-if-nil". Not only it
>> would be more consistent with other languages (like Io and Groovy),
>> but I think it would also be more useful/less dangerous.
>
> IMHO there should be two methods, to avoid confusion, and they should
> be explicitly named, again to avoid confusion.
> foo.if_not_nil.bar
> and
> foo.if_respond_to.bar
>
> Thoughts?

Then how it is different from
foo.bar if foo.respond_to?(:bar)

?
--
Pozdrawiam

Radosław Bułat
http://radarek.jogger.pl - mój blog

=end

#12 - 02/19/2009 10:55 PM - hramrach (Michal Suchanek)

=begin
2009/2/19 Joel VanderWerf vjoel@path.berkeley.edu:

> Roger Pack wrote:
>
>> On Wed, Feb 18, 2009 at 2:35 PM, Joel VanderWerf
>> vjoel@path.berkeley.edu wrote:

Roger Pack wrote:

Then how it is different from
foo.bar if foo.respond_to?(:bar)

?

You don't have to write foo and bar twice :)

I'm with you in thinking that adding another syntax
foo.?bar
seems a little much when a new method will do.
-=r

Agree on both points. But what does the new method return?

Oh I gotcha.
It would return something like Yusuke Endoh's suggestion...
Implementation image:

```
class Object
def if_not_nil
self
end
end

class NilClass
def if_not_nil
obj = BasicObject.new
def obj.method_missing(*)
nil
end
obj
end
end
```

not sure how the if_responds_to method would look like, exactly.
-=r

Chaining would go like this then?

x.if_not_nil.do_foo.if_not_nil.do_bar

We'd have to repeat #if_not_nil at each link in the chain, and it would
become ambiguous whether the second #if_not_nil was intended to handle
x==nil or x.do_foo==nil (or both).

Or maybe the method_missing implementation should return self? Then we could
write:

x.if_not_nil.do_foo.do_bar

Then if you explicitly want to handle a nil result from do_foo, you could do
so unambiguously:

x.if_not_nil.do_foo.if_not_nil.do_bar

I think you are off the track here, at least WRT the original use of try.

The #try(:name,...) would try to call :name on the object, and return
nil (rather than an exception) when the object does not respond to the
method (or in the case of Rails it does not have the requested
attribute).

The other thing I occasionally do and which could be more streamlined
looks like this:

here fd of a pipe might be already closed but it may not fd.close rescue nil # hopefully prevents zombie hordes

assume Object#quiet returns an object that does not raise an exception:

fd.quiet.close # hopefully prevent zombie hordes

Here l is non-nil but it might not contain the field:

@src = l.scan(/src="(.*)"/)[0][0] rescue nil

Not helping here - two [] calls have to be escaped, hard to read

@src = l.scan(/src="(.*)"/).quiet[0].quiet[0]

However, the second quiet might not be necessary since the MatchData
can be always dereferenced twice. Still not nice to read, though.

@src = l.scan(/src="(.*)"/).quiet[0][0]

Here input might be a file or directory:

glob = Dir.glob input + File::Separator + "*" + $IN_EXT if
(File.directory? input rescue nil)

Much simpler condition here

glob = Dir.glob input + File::Separator + "*" + $IN_EXT if
File.quiet.directory? input

Thanks

Michal

=end

#13 - 02/20/2009 01:08 AM - radarek (Radosław Bułat)

=begin
On Thu, Feb 19, 2009 at 2:55 PM, Michal Suchanek hramrach@centrum.cz wrote:

> I think you are off the track here, at least WRT the original use of try.
>
> The #try(:name,...) would try to call :name on the object, and return
> nil (rather than an exception) when the object does not respond to the
> method (or in the case of Rails it does not have the requested
> attribute).
>
> The other thing I occasionally do and which could be more streamlined
> looks like this:
>
> here fd of a pipe might be already closed but it may not
>
> fd.close rescue nil # hopefully prevents zombie hordes
>
> assume Object#quiet returns an object that does not raise an exception:
>
> fd.quiet.close # hopefully prevent zombie hordes
>
> Here l is non-nil but it might not contain the field:
>
> @src = l.scan(/src="(.*)"/)[0][0] rescue nil
>
> Not helping here - two [] calls have to be escaped, hard to read
>
> @src = l.scan(/src="(.*)"/).quiet[0].quiet[0]
>
> However, the second quiet might not be necessary since the MatchData
> can be always dereferenced twice. Still not nice to read, though.
>
> @src = l.scan(/src="(.*)"/).quiet[0][0]
>
> Here input might be a file or directory:
>
> glob = Dir.glob input + File::Separator + "*" + $IN_EXT if
> (File.directory? input rescue nil)

> Much simpler condition here
>
> glob = Dir.glob input + File::Separator + "*" + $IN_EXT if
> File.quiet.directory? input

It doesn't convince me. It can lead to bad programming style and could
hide important error from programmer (the same goes to "blah() rescue
nil" thing which should be used very very rarely). If programmer
decide to raise some error it does mean that it's imporatnt and
shouldn't be omitted. It also allow to continue when it shouldn't and
give possibility to propagate error

For example:
@src = l.scan(/src="(.*)"/).quiet[0][0]

How do you now if @src have "error" or good value? Ok, probably you
check in next line if it's nil or something but if you won't then
error can be propagated later.

Exceptions have concrete reason to be in language like Ruby and giving
more possibilities to omit them to programmer is bad idea.

PHP has something like '@'. It can be put before function name like:
file = @fopen(...)

It cause that php won't give any message about errors during execution
this function.
It sucks!

--
Pozdrawiam

Radosław Bułat
http://radarek.jogger.pl - mój blog

=end

#14 - 02/20/2009 04:49 AM - hramrach (Michal Suchanek)

=begin
2009/2/19 Radosław Bułat radek.bulat@gmail.com:

> It doesn't convince me. It can lead to bad programming style and could
> hide important error from programmer (the same goes to "blah() rescue
> nil" thing which should be used very very rarely). If programmer
> decide to raise some error it does mean that it's imporatnt and
> shouldn't be omitted. It also allow to continue when it shouldn't and
> give possibility to propagate error

yes, I am aware of "rescue nil" possibly hiding different error than
originally intended. That's another reason why a finer grained
approach that allows pointing to the exact place where the error is
expected would be useful.

> For example:
> @src = l.scan(/src="(.*)"/).quiet[0][0]
>
> How do you now if @src have "error" or good value? Ok, probably you
> check in next line if it's nil or something but if you won't then
> error can be propagated later.

The nil value of @src is a valid value in the case it was not
specified in the text.

> Exceptions have concrete reason to be in language like Ruby and giving
> more possibilities to omit them to programmer is bad idea.

Yes, exceptions have good reason, and so has catching them. If you
expect an exception can occur in a particular place in code you catch
it. You do not want your program to abort because of expected and
completely normal condition.

> PHP has something like '@'. It can be put before function name like:
> file = @fopen(...)
>
> It cause that php won't give any message about errors during execution
> this function.
> It sucks!

yes, it does. That's because these aren't exceptions, they are only
debug messages you can turn on or off.

Thanks

Michal

=end

#15 - 02/22/2009 12:39 AM - hramrach (Michal Suchanek)

=begin
On 21/02/2009, Roger Pack rogerdpack@gmail.com wrote:
...

>> The other thing I occasionally do and which could be more streamlined
>> looks like this:
>>
>> here fd of a pipe might be already closed but it may not
>>
>> fd.close rescue nil # hopefully prevents zombie hordes
>>
>> assume Object#quiet returns an object that does not raise an exception:
>>
>> fd.quiet.close # hopefully prevent zombie hordes
>
> That is interesting [but how does it know which exception to
> swallow--assume StandardError? what if the exception raised isn't
> quite the expected one?] :)

In this case I don't really care. I want to close the pipe if
possible, and ignore the case when it cannot be closed because it is
already invalid.

However, there are cases when the semantics would not be clear.

Does it catch any exception or only some subclass(es)?

Does it catch exceptions originating in that object or also ones
originating on other objects that might be invoked by the "quiet"
object?

I am not sure what would be the best option. Perhaps this tool is not
good fit for the ambiguous cases although the behaviour should still
be defined somehow.

Thanks

Michal

=end

#16 - 09/18/2009 04:22 AM - marcandre (Marc-Andre Lafortune)

- Category set to core

- Assignee set to matz (Yukihiro Matsumoto)

=begin

=end

#17 - 04/02/2010 08:13 AM - znz (Kazuhiro NISHIYAMA)

- Status changed from Open to Assigned

- Target version set to 1.9.2

=begin

=end

#18 - 04/04/2010 01:09 AM - znz (Kazuhiro NISHIYAMA)

- Target version changed from 1.9.2 to 2.0.0

=begin

=end

#19 - 08/25/2010 07:43 AM - runpaint (Run Paint Run Run)

=begin
Now we have rb_check_funcall(), this idiom is easier to realise in C than in Ruby. What's the next step here? More name suggestions? A patch? A call for further comments?
=end

#20 - 02/13/2012 09:56 PM - mame (Yusuke Endoh)

- Assignee changed from matz (Yukihiro Matsumoto) to authorNari (Narihiro Nakamura)

@nari: There is no strong objection about the name "try".
Did matz hate "try"? If not, it is okay to go ahead.

runpaint (Run Paint Run Run): We need matz's opinion about the name, as OP
said.

I, personally, objected the API design of this proposal in
ruby-core:21907, but I,
as a release manager, must say that my personal objection
is negligible, if matz approved send-like API and does not
agree with me.

--
Yusuke Endoh mame@tsg.ne.jp

#21 - 02/15/2012 12:59 AM - Anonymous

> I, personally, objected the API design of this proposal in
> ruby-core:21907, but I,
> as a release manager, must say that my personal objection
> is negligible, if matz approved send-like API and does not
> agree with me.

I also disagree with send-like API. Much prefer the other suggested,
or an "andand" [1] like interface.

[1] http://andand.rubyforge.org/

#22 - 03/15/2012 06:25 PM - authorNari (Narihiro Nakamura)

- Status changed from Assigned to Rejected

Matz still hate "try", so I'll close this ticket.
Please reopen this ticket if you have good name which convince matz.

Thanks.

#23 - 03/16/2012 12:26 AM - trans (Thomas Sawyer)

How about just respond ?

```
foo.respond(:bar)
```

And really, there's no reason it can't support delegation style *too*.

```
foo.respond.bar
```

This later form will be slower of course because it has to create an intermediate delegator --at least until Ruby has some "high-order function" capability built-in (*hinthint*). But it's certainly the nicer notation in most cases.

#24 - 03/16/2012 02:24 AM - andhapp (Anuj Dutta)

How about 'can'?

user.can(:talk)

Just my two cents.

#25 - 03/16/2012 02:29 AM - Anonymous

> How about just respond ?
>
>   foo.respond(:bar)

I like it.
-r

#26 - 03/19/2012 07:29 PM - regularfry (Alex Young)

On 15/03/12 15:26, Thomas Sawyer wrote:

> Issue #1122 has been updated by Thomas Sawyer.
>
> How about just respond ?
>
> ```
> foo.respond(:bar)
> ```
>
> And really, there's no reason it can't support delegation style *too*.
>
> ```
> foo.respond.bar
> ```

or maybe ?

--
Alex

#27 - 05/17/2012 07:51 AM - niquola (nicola ryzhikov)

If this is not very cryptic chain:

foo?bar(args)?buz

#28 - 05/17/2012 12:20 PM - nobu (Nobuyoshi Nakada)

=begin
: niquola (nicola ryzhikov) wrote:
If this is not very cryptic chain:

```
foo?bar(args)?buz
```

It's the ternary operator already in use.
=end

#29 - 05/23/2012 03:23 AM - Anonymous

> =begin
> : niquola (nicola ryzhikov) wrote:
> If this is not very cryptic chain:
>
>   foo?bar(args)?buz
>
> It's the ternary operator already in use.

A bit confusing.
-r

#30 - 07/14/2012 03:04 PM - rogerdpack (Roger Pack)

- File try.txt added

Attaching presentation, contents:

Feature #1122 [request for Object#try (guarded method invocation)]

Today:

```
        if entry.at('description') && entry.at('description').inner_text
```

or:

```
        if (description = entry.at('description') && description.inner_text
```

Proposal: Object#&& method:

if entry.at('description').&&.inner_text

or alternate Proposal Object#andand or Object#present? or some other name:

if entry.at('description').present?.inner_text

Pros: easier to read, no variable allocation. Unknown cons.

#31 - 07/24/2012 11:02 PM - mame (Yusuke Endoh)

Narihiro Nakamura and Roger Pack,

Sorry but this proposal was rejected at the developer meeting (7/21).

Matz first said he hesitated to extend the syntax for this feature.
He then said there is no good reason to make this feature built-in;
people can use it as a gem (e.g., ActiveSupport).

--
Yusuke Endoh mame@tsg.ne.jp

#32 - 04/10/2015 04:34 AM - akr (Akira Tanaka)

- Related to Feature #11034: Nil Conditional added

#33 - 09/19/2015 05:10 AM - akr (Akira Tanaka)

- Related to Feature #11537: Introduce "Safe navigation operator" added

Files

try.txt	504 Bytes	07/14/2012	rogerdpack (Roger Pack)