Ruby master - Feature #13923

Idiom to release resources safely, with less indentations

09/20/2017 06:21 AM - tagomoris (Satoshi TAGOMORI)

Status:	Feedback	
Priority:	Normal	
Assignee:		
Target version:		

Description

In programs which grabs and releases resources very often, we need to write so much begin-ensure clauses.

```
begin
  storage = getStorage()
  begin
    buffer = storage.get(buffer_id)

    # ...
  ensure
    buffer.close if buffer
  end
rescue StorageError => e
  # ...
ensure
  storage.close if storage
end
```

Such code makes our code fat, and difficult to understand.
I want to write such code like below:

```
# Class of storage and buffer should include a module (like Closeable)
# or be checked with respond_to?(:close)

begin(storage = getStorage(); buffer = storage.get(buffer_id)
  # ...
rescue StorageError => e
  # ...
end
# (buffer.close if buffer) rescue nil
# (storage.close if storage) rescue nil
```

Other languages also have similar features:

- Java: try-with-resources
- Python: with

History

#1 - 09/20/2017 08:42 AM - nobu (Nobuyoshi Nakada)

- Description updated

Probably a way which is close to it right now would be:

```
begin
  storage = getStorage()
  buffer = storage.get(buffer_id)

  # ...
rescue StorageError => e
  # ...
ensure
  buffer&.close
  storage&.close
end
```

#2 - 09/20/2017 10:41 AM - tagomoris (Satoshi TAGOMORI)

nobu (Nobuyoshi Nakada) wrote:

> Probably a way which is close to it right now would be:
>
> ```
> begin
>   storage = getStorage()
>   buffer = storage.get(buffer_id)
>
>   # ...
> rescue StorageError => e
>   # ...
> ensure
>   buffer&.close
>   storage&.close
> end
> ```

With this code, storage will not be closed if buffer.close raises exceptions.

#3 - 09/20/2017 11:01 AM - Eregon (Benoit Daloze)

tagomoris (Satoshi TAGOMORI) wrote:

> With this code, storage will not be closed if buffer.close raises exceptions.

IMHO if buffer.close raises an exception then that should be fixed and not silently ignored.
(e.g. if close fails it could mean the contents is not properly flushed and might be very hard to debug).

A more common way to write code using resources in Ruby is to use blocks:

```
File.open("foo") do |file|
  file.read
end
```

#4 - 09/20/2017 04:01 PM - KonaBlend (Kona Blend)

maybe new keyword **defer** which simply builds an array of blocks to be called at end of scope;
specifically immediately before **end** of **ensure** section;
i.e. if there is ensure code then defer underlying support should come afterwards.

```
begin
  ## underlying support:
  # __deferred = []

  r0 = Expensive.make_resource # might return nil or become nil
  defer { r0.close if r0 }
  ## underlying support:
  # __deferred.unshift(block)

  ## sugar
  r1 = Expensive.make_resource
  defer(r1)
  ## equivalent:
  # defer { r1.close if r1 }

  ## sugar
  defer r2 = Expensive.make_resource
  ## equivalent:
  # defer { r2.close if r2 }
ensure
  ## underlying support:
  # __deferred.each { |block| block.call }
end
```

note: edit: __deferred.push to __deferred.unshift for FILO ordering of block calls

#5 - 09/20/2017 05:01 PM - matz (Yukihiro Matsumoto)

I like defer idea, although adding a new keyword is hard.

Matz.

#6 - 09/24/2017 02:30 AM - nobu (Nobuyoshi Nakada)

While ensure in ruby is a syntax and share the scope, defer in go dynamically registers the clean-up code with capturing local variables implicitly. So a defer in a loop may registers multiple times with different objects, or may not register by a condition.

```
package main

import (
    "fmt"
)

type foo struct {
    n int
}

func Create(n int) *foo {
    fmt.Printf("Creating %v\n", n)
    return &foo{n}
}

func Delete(f *foo) {
    fmt.Printf("Deleting %v\n", f.n)
}

func main() {
    fmt.Println("Start")
    for i := 1; i <= 3; i++ {
        f := Create(i)
        if i % 2 != 0 {
            defer Delete(f)
        }
    }
    fmt.Println("Done")
}
```

shows

```
$ go run test.go
Start
Creating 1
Creating 2
Creating 3
Done
Deleting 3
Deleting 1
```

I think that Kernel#defer in Kona's proposal would be possible, but it would be hard to capture local variables.

#7 - 09/24/2017 12:59 PM - nobu (Nobuyoshi Nakada)

How about:

```
class Deferred
  def initialize
    @list = []
    yield self
  ensure
    @list.reverse_each do |res, clean|
      clean.(res)
    end
  end
  def defer(res = true, &clean)
    @list << [res, clean||CLOSE] if res
    res
  end
  CLOSE = :close.to_proc
end

if $0 == __FILE__
  class Resource
    @@number = 0
    def initialize
      @num = @@number += 1
    end
    def close
```

```
      puts "#@num closed"
    end
    def delete
      puts "#@num deleted"
    end
  end

  Deferred.new do |d|
    r0 = Resource.new
    d.defer {r0&.close}

    r1 = Resource.new
    d.defer r1

    r2 = d.defer Resource.new
    abort unless Resource === r2

    r3 = d.defer(Resource.new) {|r|r.delete}
  end
end
```

#8 - 09/24/2017 01:26 PM - Eregon (Benoit Daloze)

IMHO this defer idea is not appropriate to manage resources at it just moves the release of the resources at the end of the method.
But this is often not appropriate or later than needed.
A block captures more explicitly the scope in which the resource is needed.
Moreover, if moving the release of the resource is needed, moving the closing }/end of the block is enough and much simpler than creating a new method to use defer for a shorter scope.
It is generally unwise to hold resources longer than necessary, particularly locks.

Taking the original example, and using blocks:

```
begin
  getStorage() do |storage|
    storage.get(buffer_id) do |buffer|
      # ...
    end
  end
rescue StorageError => e
  # ...
end
```

This naturally prevents using the resource after #close and clearly delimits when the resource is available.

#9 - 09/25/2017 12:49 PM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Feedback

We looked at this issue in a developer meeting today and nobu's library solution written in comment #7 was popular there.

Is that OK for you, or you still want a new syntax than a library (and if that is the case; can you tell us why)?

#10 - 09/25/2017 06:47 PM - KonaBlend (Kona Blend)

shyouhei (Shyouhei Urabe) wrote:

> We looked at this issue in a developer meeting today and nobu's library solution written in comment #7 was popular there.
>
> Is that OK for you, or you still want a new syntax than a library (and if that is the case; can you tell us why)?

fwiw, combined with new #12906, comment #7 makes me happy.

#11 - 09/26/2017 04:52 AM - tagomoris (Satoshi TAGOMORI)

shyouhei (Shyouhei Urabe) wrote:

> We looked at this issue in a developer meeting today and nobu's library solution written in comment #7 was popular there.
>
> Is that OK for you, or you still want a new syntax than a library (and if that is the case; can you tell us why)?

I prefer to get resources at once, but #7 looks to work well for my use case.
Deferred is ok for me too.
Can we expect that it's bundled as standard library in ruby?

#12 - 09/26/2017 08:31 AM - shyouhei (Shyouhei Urabe)

tagomoris (Satoshi TAGOMORI) wrote:

> I prefer to get resources at once, but #7 looks to work well for my use case.
> Deferred is ok for me too.
> Can we expect that it's bundled as standard library in ruby?

No, no plan to bundle it yet. These days it's getting hard to include a new standard library unless there are very good reason for it to avoid being a normal gem. If you think it's worth, please add a separate feature request describing that reason. Thank you in advance.