Ruby master - Feature #10548

remove callcc (Callcc is now going obsoleted. Please use Fiber.)

11/26/2014 11:28 AM - tarui (Masaya Tarui)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

We are paying a lot of costs for callcc's consistency.
and currently, we can use Fiber in many situation.

In https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20140517Japan,
matz agreed to remove callcc.

If there is no refutation, remove callcc in the future version.

Associated revisions

Revision 262245b0 - 11/26/2014 11:50 AM - tarui (Masaya Tarui)

- ext/continuation/continuation.c (Init_continuation): obsolete callcc.   first step of [Feature #10548].

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@48588 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 48588 - 11/26/2014 11:50 AM - tarui (Masaya Tarui)

- ext/continuation/continuation.c (Init_continuation): obsolete callcc.   first step of [Feature #10548].

Revision 48588 - 11/26/2014 11:50 AM - tarui (Masaya Tarui)

- ext/continuation/continuation.c (Init_continuation): obsolete callcc.   first step of [Feature #10548].

Revision 48588 - 11/26/2014 11:50 AM - tarui (Masaya Tarui)

- ext/continuation/continuation.c (Init_continuation): obsolete callcc.   first step of [Feature #10548].

Revision 48588 - 11/26/2014 11:50 AM - tarui (Masaya Tarui)

- ext/continuation/continuation.c (Init_continuation): obsolete callcc.   first step of [Feature #10548].

Revision 48588 - 11/26/2014 11:50 AM - tarui (Masaya Tarui)

- ext/continuation/continuation.c (Init_continuation): obsolete callcc.   first step of [Feature #10548].

Revision 48588 - 11/26/2014 11:50 AM - tarui (Masaya Tarui)

- ext/continuation/continuation.c (Init_continuation): obsolete callcc.   first step of [Feature #10548].

History

#1 - 11/26/2014 11:51 AM - tarui (Masaya Tarui)

- Status changed from Open to Closed

- % Done changed from 0 to 100

Applied in changeset r48588.

- ext/continuation/continuation.c (Init_continuation): obsolete callcc.   first step of [Feature #10548].

#2 - 11/26/2014 11:52 AM - tarui (Masaya Tarui)

- Subject changed from Callcc is now going obsoleted. Please use Fiber. to remove callcc (Callcc is now going obsoleted. Please use Fiber.)

- Category set to ext

- Status changed from Closed to Open

- Target version set to 2.6

#3 - 11/26/2014 02:33 PM - mame (Yusuke Endoh)

That's a bummer!

Callcc is the most exciting toy for the functional programmers.
If Ruby had no callcc, I couldn't have been interested in Ruby.

No one think it production ready.
When callcc is used, the user is (or should be) not serious but mischievous.
You don't have to pay any cost for its consistency.
SEGV is welcome when used in combination with another feature.

Instead of making it obsoleted, how about printing a big warning banner when "contination" is required?

```
rb_warn("****************************************");
rb_warn("** WARNING!! callcc is a JOKE feature **");
rb_warn("** Are you ready to see nasal demons? **");
rb_warn("****************************************");
```

I wish you'd reconsider.

--
Yusuke Endoh mame@ruby-lang.org

#4 - 11/30/2014 11:52 AM - shevegen (Robert A. Heiler)

I am neutral on this, I have no pro or con opinion here, but I wanted to comment
on what Yusuke Endoh wrote, and in particular suggest perhaps something for
future feature-references that are not extremely important but also not totally
useless or "fun features".

I think we also have goto operator in Ruby which is also a joke feature.

I also remember evil.rb from ... Florian G. or someone, which added things
like .unfreeze and such back then (I think it was in ruby 1.8.x era).

Perhaps the ruby stdlib/corelib can add a submodule or perhaps a smaller
subprojects within ruby of "JOKE features", where matz/the ruby core team
does not promote this, but if people want to play with that, they can do
so. And such things can be bundled together into that project, like

require 'ruby/jokes'

or

require 'ruby/jokes'

or

require 'specialities'

Or something like that. :)

If things would become more popular, they could be moved out
of it again, and if things are not popular then they can
remain there (and people who like those features ideally could
maintain it there too!)

#5 - 03/01/2015 02:52 AM - stephenprater (Stephen Prater)

I agree with Yusuke.

I'd be sorry to see it go - it does some neat tricks that are a lot harder to pull off with Fibers.

Here for instance is a PoC reversible debugger: https://gist.github.com/stephenprater/ca312d24578455f36550

I don't think you can do that with Fibers, since the control flow with fibers is still always forward.

prater

#6 - 01/10/2016 05:23 AM - tank_bohr (Alexey Nikitin)

Please don't do it. Continuations are awesome

#7 - 04/27/2016 11:35 AM - Overbryd (Lukas Rieder)

Please keep callcc, they are inherently awesome to build constraint solvers.

I am using them in a real life application. I am calculating the available tables for a full calendar with many time slots and with respect to many configurable business rules for restaurants.
Using callcc this feature got blazingly fast and very nicely readable. Also we use it to optimise table arrangements with respect to complex restaurant business rules (even something like: Guest A doesn't like to sit near Guest B).

Please just have a look at these resources:

- https://github.com/chikamichi/amb/tree/master/examples
- http://web.archive.org/web/20151116124853/http://liufengyun.chaos-lab.com/prog/2013/10/23/continuation-in-ruby.html

Please help me to keep Guest A away from Guest B. Bad things might happen.

#8 - 05/13/2017 10:32 AM - jphelps (Jeremy Phelps)

I just learned that Ruby has continuations. Then I learned that they're considered obsolete, and "instead" we're supposed to use a feature (basically just Python's yield statement) that has zero use cases in common with continuations.

#9 - 05/13/2017 11:47 AM - Eregon (Benoit Daloze)

jphelps (Jeremy Phelps) wrote:

> I just learned that Ruby has continuations. Then I learned that they're considered obsolete, and "instead" we're supposed to use a feature (basically just Python's yield statement) that has zero use cases in common with continuations.

This is untrue. Ruby Fibers have little in common with Python generators.
The most important feature is they have a stack.
In other words, they are stackful coroutines, which can be used both as asymmetric coroutines (yield/resume) and symmetric coroutines (transfer).

But yes, they cannot go back in the control flow, in opposition to coroutines.
IMHO coroutines are extremely hard to understand and have drawbacks similar to GOTO.

#10 - 05/13/2017 07:01 PM - jwmittag (Jörg W Mittag)

jphelps (Jeremy Phelps) wrote:

> I just learned that Ruby has continuations. Then I learned that they're considered obsolete, and "instead" we're supposed to use a feature (basically just Python's yield statement) that has zero use cases in common with continuations.

Even back when callcc was still officially considered to be "part of Ruby", it was actually not implemented by the majority of Ruby implementations.

- JRuby doesn't implement it,
- Rubinius doesn't implement it,
- Opal doesn't implement it,
- Topaz doesn't implement it,
- TruffleRuby doesn't implement it,
- IronRuby doesn't implement it,
- even MRuby, the implementation written by matz himself doesn't implement it.

The only two implementations that currently implement callcc are YARV and MagLev.

So, yes, it is *officially* still part of Ruby, but … no-one cares.

#11 - 05/13/2017 07:40 PM - jphelps (Jeremy Phelps)

Eregon, I have no idea what you're talking about. All the examples of Fiber usage over on ruby-doc.org shows identical behavior to Python's yield statement. The only difference is that in Python, "resume" is spelled "next", and you can iterate over the sequence of return values (just a monkey patch away with Fibers). Oh, and the way Python handles a dead generator is different (you start getting void instead of an exception).

#12 - 05/14/2017 02:32 AM - ko1 (Koichi Sasada)

On 2017/05/14 4:40, jeremy.phelps@instacart.com wrote:

> Eregon, I have no idea what you're talking about. All the examples of Fiber usage over on ruby-doc.org shows identical behavior to Python's

> yield statement. The only difference is that in Python, "resume" is spelled "next", and you can iterate over the sequence of return values (just a monkey patch away with Fibers). Oh, and the way Python handles a dead generator is different (you start getting void instead of an exception).

COMPLETELY Off topic:

"Context" manipulation is one of big topic and there are many related terminologies (academic, language/implementation specific, promotion terminologies). In fact, there is confusing.

In few minutes I remember the following related words and it is good CS exam to describe each :p

- Thread (Ruby)
- Green thread (CS terminology)
- Native thread (CS terminology)
- Non-preemptive thread (CS terminology)
- Preemptive thread (CS terminology)
- Fiber (Ruby/using resume/yield)
- Fiber (Ruby/using transfer)
- Fiber (Win32API)
- Generator (Python/JavaScript)
- Generator (Ruby)
- Continuation (CS terminology/Ruby, Scheme, ...)
- Partial continuation (CS terminology/ functional lang.)
- Exception handling (many languages)
- Coroutine (CS terminology/ALGOL)
- Semi-coroutine (CS terminology)
- Process (Unix/Ruby)
- Process (Erlang/Elixir)
- setjmp/longjmp (C)
- makecontext/swapcontext (POSIX)
- Task (...) (maybe more and more words in the world)

(1) describe each words (10 point/each).
(2) describe how to make each words (100 point/each).
(joking. do not submit it to me :p)

Reviewing my carrier, I may love to consider how to control "context".
My bachelor/master thesis is about how to make a threading library and doctor thesis is about how to introduce new threading mechanism on Ruby.
I introduced Fiber into Ruby 1.9 and recently I'm thinking about Guild.

--
// SASADA Koichi at atdot dot net

#13 - 05/14/2017 10:25 AM - Eregon (Benoit Daloze)

jphelps (Jeremy Phelps) wrote:

> Eregon, I have no idea what you're talking about. All the examples of Fiber usage over on ruby-doc.org shows identical behavior to Python's yield statement. The only difference is that in Python, "resume" is spelled "next", and you can iterate over the sequence of return values (just a monkey patch away with Fibers). Oh, and the way Python handles a dead generator is different (you start getting void instead of an exception).

Then look for other examples than just the ones in the documentation.
The fact that Fibers have a stack means Fiber.yield can be called anywhere during the Fiber execution, including in deeply nested method calls, while Python and JavaScript generators are lexically bound and restricted to only call "yield" in the generator function and not deeper. Try to reproduce this in Python:

```
def powers_of(x, max)
  n = x
  while n < max
    Fiber.yield(n)
    n *= x
  end
end

def powers_of_range(range, max)
  range.each { |x| powers_of(x, max) }
end

f = Fiber.new do
  powers_of_range(2..4, 100)
  raise StopIteration
```

```
end

loop {
  p f.resume
}
```

It's possible, but you either have to manually inline the two methods, or use multiple generators instead of just one Fiber here.
For a more interesting usage of Fiber see https://www.igvita.com/2010/03/22/untangling-evented-code-with-ruby-fibers/ for example.

#14 - 11/21/2018 09:17 PM - shevegen (Robert A. Heiler)

Since this may be discussed in the next upcoming developer meeting, perhaps callcc could be put into a standalone gem, if it is removed? Just in the event that some people may want to keep it, could install it if they want to (a bit like the syck gem is still about, for those whose yaml files are not yet usable via psych due to errors in these yaml files, such as non-unicode yaml files).

This is just an idea though, please feel free to disregard if it is not applicable or too much effort.

Edit: In regards to mame stating years ago how callcc is fun, this also reminds me a bit of the old evil.rb and fun with "shapechanging" objects in ruby. :D I never really used callcc much at all, but it may be nice to keep old code around for some more years (there are gems on rubygems.org that are like ~15 years old, almost). Just like with the more recent theme of "keeping ruby weird", we could say we could also keep old ruby weirdness "alive" - and weird. ;)

#15 - 11/28/2018 05:22 AM - normalperson (Eric Wong)

tarui@prx.jp wrote:

> Feature #10548: Callcc is now going obsoleted. Please use Fiber.
> https://bugs.ruby-lang.org/issues/10548

How about adding --disable-callcc configure option?

https://80x24.org/spew/20181128051213.19518-1-e@80x24.org/raw

#16 - 11/29/2018 09:32 PM - normalperson (Eric Wong)

Eric Wong wrote:

> tarui@prx.jp wrote:
>
> > Feature #10548: Callcc is now going obsoleted. Please use Fiber.
> > https://bugs.ruby-lang.org/issues/10548
>
> How about adding --disable-callcc configure option?
>
> https://80x24.org/spew/20181128051213.19518-1-e@80x24.org/raw

Also removed ec->protect_tag when callcc is disabled:

https://80x24.org/spew/20181129212602.12362-1-e@80x24.org/raw

I am also considering adding an rb_ensure_nocallcc internal function to reduce stack use when b_proc and e_proc are guaranteed to not yield.

#17 - 12/31/2018 11:28 AM - decuplet (Nikita Shilnikov)

FWIW, I do have a practical example for callcc which cannot be implemented with fibers, or at least I don't see the way it could.
I'm the author of the dry-monads gem. Along with a bunch of monads it provides an emulation of Haskell's do notation:

```
class Operation
  include Dry::Monads::Result::Mixin
  include Dry::Monads::Do

  def call(monadic_value)
    extracted_value = yield monadic_value
    # ...
    Success(extracted_value + 1)
```

```
  end
end
```

Here if monadic_value is Success(value) then yield will extract value from it and continue the execution. And if monadic_value is Failure(...) it will halt the execution and return this Failure as a result of call. Everything works perfectly fine and allows writing robust production code but only for monads that wrap 1 value inside. That is, it doesn't work for the list monad:

```
class Operation
  include Dry::Monads::Do
  include Dry::Monads::List::Mixin

  def call
    v = yield List[1, 2, 3]
    List[v + 1]
  end
end
```

this will return List[2] rather than List[2, 3, 4]. In order to make it work as expected, I would need to capture the continuation within yield so it can continue call more than once. I tested it, and it indeed works, but I don't want to depend on a to-be-removed feature.