Ruby - Feature #4326

Fiber should respond to call() and []

01/26/2011 04:08 PM - tenderlovemaking (Aaron Patterson)

Status:	Closed
Priority:	Normal
Assignee:	ko1 (Koichi Sasada)
Target version:	

Description

=begin
Fibers are considered to be coroutines. Knuth says "Subroutines are special cases of ... coroutines". This makes sense to me.

Method, Proc, and lambda respond to call and []. If Fiber also responded to call and [], we could use coroutines in places where we use lambdas, procs, and methods.

I've attached a patch that implements the two methods.
=end

History

#1 - 01/27/2011 10:45 AM - kstephens (Kurt Stephens)

=begin
Shouldn't Fiber#[] behave like Thread#[]? What about Continuation#[]?

=end

#2 - 01/27/2011 01:07 PM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Wed, 26 Jan 2011 16:08:52 +0900,
Aaron Patterson wrote in [ruby-core:34861]:

> Fibers are considered to be coroutines. Knuth says
> "Subroutines are special cases of ... coroutines". This
> makes sense to me.

Do you mean
"A is kind of B, A has a feature X, therefore B also should
have X",
but not
"A is kind of B, B has a feature X, therefore A also should
have X"?

Sounds strange to me.

--
Nobu Nakada

=end

#3 - 01/28/2011 07:15 AM - tenderlovemaking (Aaron Patterson)

=begin
On Thu, Jan 27, 2011 at 01:07:33PM +0900, Nobuyoshi Nakada wrote:

> Hi,
>
> At Wed, 26 Jan 2011 16:08:52 +0900,
> Aaron Patterson wrote in [ruby-core:34861]:
>
> > Fibers are considered to be coroutines. Knuth says
> > "Subroutines are special cases of ... coroutines". This
> > makes sense to me.

> Do you mean
> "A is kind of B, A has a feature X, therefore B also should
> have X",
> but not
> "A is kind of B, B has a feature X, therefore A also should
> have X"?
>
> Sounds strange to me.

Maybe code will be more clear. I want to do this:

```
def square block
  block.call ** 2
end

callables = [
  lambda { 10 },
  Fiber.new { Fiber.yield 10 }
]

callables.map { |thing| square thing }
```

Instead I have to do this:

```
def square block
  value = block.is_a?(Fiber) ? block.resume : block.call
  value ** 2
end

callables = [
  lambda { 10 },
  Fiber.new { Fiber.yield 10 }
]

callables.map { |thing| square thing }
```

--
Aaron Patterson
http://tenderlovemaking.com/

Attachment: (unnamed)
=end

#4 - 01/28/2011 07:41 AM - tenderlovemaking (Aaron Patterson)

=begin
On Thu, Jan 27, 2011 at 02:14:16PM -0800, Aaron Patterson wrote:

> On Thu, Jan 27, 2011 at 01:07:33PM +0900, Nobuyoshi Nakada wrote:
>
> > Hi,
> >
> > At Wed, 26 Jan 2011 16:08:52 +0900,
> > Aaron Patterson wrote in [ruby-core:34861]:
> >
> > > Fibers are considered to be coroutines. Knuth says
> > > "Subroutines are special cases of ... coroutines". This
> > > makes sense to me.
> >
> > Do you mean
> > "A is kind of B, A has a feature X, therefore B also should
> > have X",
> > but not
> > "A is kind of B, B has a feature X, therefore A also should
> > have X"?
> >
> > Sounds strange to me.
>
> Maybe code will be more clear. I want to do this:
>
> ```
> def square block
> ```

```
  block.call ** 2
end

callables = [
  lambda { 10 },
  Fiber.new { Fiber.yield 10 }
]

callables.map { |thing| square thing }
```

Instead I have to do this:

```
def square block
  value = block.is_a?(Fiber) ? block.resume : block.call
  value ** 2
end

callables = [
  lambda { 10 },
  Fiber.new { Fiber.yield 10 }
]

callables.map { |thing| square thing }
```

The way I think about it is:

A subroutine is a coroutine, but in ruby I cannot use a coroutine in
the place of a subroutine.

Am I thinking about this wrong?

--
Aaron Patterson
http://tenderlovemaking.com/

Attachment: (unnamed)
=end

#5 - 01/28/2011 11:10 AM - runpaint (Run Paint Run Run)

=begin
I understood what you meant, and agree in principle. I wanted to think about the implications given that (a) #call is duck-typed on, and (b) Fibers cannot be called across Threads, and, possibly, (c) the root Fiber has special semantics but no predicate for identifying them.
=end

#6 - 01/28/2011 04:43 PM - usa (Usaku NAKAMURA)

- Status changed from Open to Assigned

=begin

=end

#7 - 01/29/2011 02:57 AM - headius (Charles Nutter)

=begin
RPRR and Nobu make good points.

Fibers are coroutines...agreed.

If we go by Knuth, a subroutine "is a" coroutine, but you're thinking about the relationship wrong. A Hash "is a" Object. We don't expect Object (coroutine) to do what Hash (subroutine) can do and you cannot use an Object (coroutine) in the place of a Hash (subroutine). Simply put, Knuth's statement doesn't mean Fiber should be usable everywhere a subroutine (Proc) can be used.

RPRR's points about threading and root fibers are also important, since they explicitly mean you can't use a Fiber in place of a thread-agnostic subroutine.

In practice I don't really care about adding [] and .call to Fiber, but they do mask the fact that these are not simply "resumable procs".
=end

#8 - 01/29/2011 08:29 AM - tenderlovemaking (Aaron Patterson)

=begin
On Sat, Jan 29, 2011 at 02:58:46AM +0900, Charles Nutter wrote:

> Issue #4326 has been updated by Charles Nutter.
>
> RPRR and Nobu make good points.
>
> Fibers are coroutines...agreed.
>
> If we go by Knuth, a subroutine "is a" coroutine, but you're thinking about the relationship wrong. A Hash "is a" Object. We don't expect Object (coroutine) to do what Hash (subroutine) can do and you cannot use an Object (coroutine) in the place of a Hash (subroutine). Simply put, Knuth's statement doesn't mean Fiber should be usable everywhere a subroutine (Proc) can be used.

My point is that it seems that LSP is broken. Either Fiber should
respond to "call" and "[]", or Proc should respond to "resume". Really
I don't care which.

> RPRR's points about threading and root fibers are also important, since they explicitly mean you can't use a Fiber in place of a thread-agnostic subroutine.

I agree these are problems. But I think fixing those issues is
different than fixing LSP problems.

--
Aaron Patterson
http://tenderlovemaking.com/

Attachment: (unnamed)
=end

#9 - 01/29/2011 09:45 AM - headius (Charles Nutter)

=begin
On Fri, Jan 28, 2011 at 5:29 PM, Aaron Patterson
aaron@tenderlovemaking.com wrote:

> My point is that it seems that LSP is broken. Either Fiber should
> respond to "call" and "[]", or Proc should respond to "resume". Really
> I don't care which.

LSP says the opposite. Assuming (from your interpretation of Knuth and
LSP) that coroutines (Fibers) are a generalization of subroutines
(procs), and conversely subroutines (procs) are a specialization of
coroutines (Fibers), then T = coroutines (Fibers), S = subroutines
(procs) and...

"Let q(x) be a property provable about objects x of type T. Then q(y)
should be true for objects y of type S where S is a subtype of T."

So subroutines (procs) should do everything Fibers can do, but not the
inverse relationship. This would also imply that subroutines should be
resumable, etc.

However...I'm not sure what Knuth was saying is implying a
generalization/specialization relationship. He was saying subroutines
⊂ coroutines. So you should be able to expect that for all operations
q on coroutines there should be a matching operation q on subroutines
and vice versa. Doesn't this mean we should also have resuming,
yielding, etc on subroutines? (please no)

So...there are three possibilities:

1. subroutines are a specialization of coroutines. Then it's perfectly valid for subroutines to define things coroutines do not.
2. subroutines are equivalent to coroutines. Then coroutines should add [] and call and subroutines should have resuming/yielding capabilities.
3. coroutines are a specialization of subroutines. Then coroutines should add [] and call, and we're done. And Knuth is wrong.

Now that I've thoroughly confused myself, I'll leave it up to you and
ruby-core... which reality would you like?

- Charlie

=end

#10 - 01/29/2011 09:47 AM - headius (Charles Nutter)

=begin
On Fri, Jan 28, 2011 at 6:45 PM, Charles Oliver Nutter
headius@headius.com wrote:

> LSP says the opposite. Assuming (from your interpretation of Knuth and
> LSP) that coroutines (Fibers) are a generalization of subroutines
> (procs), and conversely subroutines (procs) are a specialization of
> coroutines (Fibers), then T = coroutines (Fibers), S = subroutines
> (procs) and...

BTW, I'd love to do an audit of Ruby behaviors some time and see how frequently LSP is broken. I'd also love to hear why Liskov would say about IO#reopen changing the object's type completely. :)

- Charlie

=end

#11 - 01/29/2011 10:16 AM - mame (Yusuke Endoh)

=begin
Hi,

2011/1/27 Kurt Stephens redmine@ruby-lang.org:

> Shouldn't Fiber#[] behave like Thread#[]?

I think more notice should be taken of Kurt's remark :-)

--
Yusuke Endoh mame@tsg.ne.jp

=end

#12 - 01/29/2011 10:42 AM - headius (Charles Nutter)

=begin
On Fri, Jan 28, 2011 at 7:15 PM, Yusuke ENDOH mame@tsg.ne.jp wrote:

> Hi,
>
> 2011/1/27 Kurt Stephens redmine@ruby-lang.org:
>
> > Shouldn't Fiber#[] behave like Thread#[]?
>
> I think more notice should be taken of Kurt's remark :-)

Indeed. When coroutines were built for OpenJDK (in the MLVM project) the author quickly realized the need for coroutine-locals as well as thread-locals.

I tend to think of fibers more like threads or actors than subroutines, but of course I'm biased because on JRuby they *are* threads.

- Charlie

=end

#13 - 01/30/2011 02:41 AM - tenderlovemaking (Aaron Patterson)

=begin
On Sat, Jan 29, 2011 at 09:45:30AM +0900, Charles Oliver Nutter wrote:

> On Fri, Jan 28, 2011 at 5:29 PM, Aaron Patterson
> aaron@tenderlovemaking.com wrote:

> My point is that it seems that LSP is broken. Either Fiber should respond to "call" and "[]", or Proc should respond to "resume". Really I don't care which.

LSP says the opposite. Assuming (from your interpretation of Knuth and LSP) that coroutines (Fibers) are a generalization of subroutines (procs), and conversely subroutines (procs) are a specialization of coroutines (Fibers), then T = coroutines (Fibers), S = subroutines (procs) and...

"Let q(x) be a property provable about objects x of type T. Then q(y) should be true for objects y of type S where S is a subtype of T."

So subroutines (procs) should do everything Fibers can do, but not the inverse relationship. This would also imply that subroutines should be resumable, etc.

However...I'm not sure what Knuth was saying is implying a generalization/specialization relationship. He was saying subroutines ⊂ coroutines. So you should be able to expect that for all operations q on coroutines there should be a matching operation q on subroutines and vice versa. Doesn't this mean we should also have resuming, yielding, etc on subroutines? (please no)

So...there are three possibilities:

1. subroutines are a specialization of coroutines. Then it's perfectly valid for subroutines to define things coroutines do not.

Yes, but it is *not* perfectly valid for subroutines to *not* define things that coroutines do. We can either resolve that by defining "resume" on Proc, or "call" and "[]" on Fiber.

1. subroutines are equivalent to coroutines. Then coroutines should add [] and call and subroutines should have resuming/yielding capabilities.
2. coroutines are a specialization of subroutines. Then coroutines should add [] and call, and we're done. And Knuth is wrong.

> Now that I've thoroughly confused myself, I'll leave it up to you and ruby-core... which reality would you like?

I have been trying to choose reality number 1, the entire time, but apparently I cannot explain myself clearly. :'(

--
Aaron Patterson
http://tenderlovemaking.com/

Attachment: (unnamed)
=end

#14 - 01/30/2011 01:47 PM - headius (Charles Nutter)

=begin
On Sat, Jan 29, 2011 at 11:41 AM, Aaron Patterson
aaron@tenderlovemaking.com wrote:

> 1. subroutines are a specialization of coroutines. Then it's perfectly valid for subroutines to define things coroutines do not.
>
> Yes, but it is *not* perfectly valid for subroutines to *not* define things that coroutines do. We can either resolve that by defining "resume" on Proc, or "call" and "[]" on Fiber.
> ...
> I have been trying to choose reality number 1, the entire time, but apparently I cannot explain myself clearly. :'(

But doesn't reality 1 mean that subroutines should get resume/etc and

coroutines should *not* get call/[]? It seems to me you either want my reality 3 or Jim's reality 4.

- Charlie

=end

#15 - 01/30/2011 01:50 PM - headius (Charles Nutter)

=begin
On Fri, Jan 28, 2011 at 9:28 PM, Jim Weirich jim.weirich@gmail.com wrote:

> I wouldn't say Knuth is wrong, but a natural language "is-a" is not always an indication of subtype (c.f. the square/rectangle paradox where a square is-a rectangle, but having square inherit separate set_width/set_height methods from rectangle would be wrong).

Right...so my reality 2 and your reality 4 are pretty close; mine says that coroutines and subroutines are the same thing, but subroutines are a specialized subset. Yours says that coroutines and subroutines are both of the same (super)type, and so they should share in common operations from that (super)type.

In any case, I should restate again that I really don't care if Ruby adds call and [] on Fiber :) I just want to make sure it's not being done for a logically inconsistent reason.

- Charlie

=end

#16 - 01/30/2011 10:18 PM - rkh (Konstantin Haase)

=begin
This might go in a slightly different direction, but I would really love Fiber to define #to_proc allowing things like [1, 2, 3].each(&fiber).

Also note that in ANSI smalltalk (which has a rather common object model to Ruby) a block closure is implementing the valuable protocol (objects that respond to #value and akin, smalltalk's #call), which would be something like the common ancestor for procs and fibers.

Konstantin

=end

#17 - 01/31/2011 08:26 AM - tenderlovemaking (Aaron Patterson)

=begin
On Sun, Jan 30, 2011 at 01:47:06PM +0900, Charles Oliver Nutter wrote:

> On Sat, Jan 29, 2011 at 11:41 AM, Aaron Patterson aaron@tenderlovemaking.com wrote:
>
> 1. subroutines are a specialization of coroutines. Then it's perfectly valid for subroutines to define things coroutines do not.
>
> Yes, but it is *not* perfectly valid for subroutines to *not* define things that coroutines do. We can either resolve that by defining "resume" on Proc, or "call" and "[]" on Fiber.
> ...
> I have been trying to choose reality number 1, the entire time, but apparently I cannot explain myself clearly. :'(
>
> But doesn't reality 1 mean that subroutines should get resume/etc and coroutines should *not* get call/[]? It seems to me you either want my reality 3 or Jim's reality 4.

I don't know anymore. I guess I don't care anymore either.

--
Aaron Patterson
http://tenderlovemaking.com/

Attachment: (unnamed)
=end

#18 - 01/31/2011 08:37 AM - tenderlovemaking (Aaron Patterson)

=begin
On Sat, Jan 29, 2011 at 10:15:31AM +0900, Yusuke ENDOH wrote:

> Hi,
>
> 2011/1/27 Kurt Stephens redmine@ruby-lang.org:
>
> > Shouldn't Fiber#[] behave like Thread#[]?
>
> I think more notice should be taken of Kurt's remark :-)

Even if Fiber only implemented call, it would make the code in
[ruby-core:34909] much better.

Maybe I will change my patch to not implement [], and we don't have to
worry about the semantics of Fiber#[] for this ticket.

--
Aaron Patterson
http://tenderlovemaking.com/

Attachment: (unnamed)
=end

#19 - 01/31/2011 08:42 AM - tenderlovemaking (Aaron Patterson)

- File fiber.patch added

=begin
New patch without Fiber#[]
=end

#20 - 06/10/2011 04:23 AM - ko1 (Koichi Sasada)

Hi,

Sorry for late response.

(2011/01/26 16:08), Aaron Patterson wrote:

> Feature #4326: Fiber should respond to call() and []
> http://redmine.ruby-lang.org/issues/show/4326
>
> Author: Aaron Patterson
> Status: Open, Priority: Normal
> Assigned to: Koichi Sasada
>
> Fibers are considered to be coroutines. Knuth says "Subroutines are special cases of ... coroutines". This makes sense to me.
>
> Method, Proc, and lambda respond to call and []. If Fiber also responded to call and [], we could use coroutines in places where we use lambdas, procs, and methods.
>
> I've attached a patch that implements the two methods.

I don't understand all of this thread. However, as I said to aaron at
IRC, I want to reject this proposal.

Reasons:
(1) Proc is restartable. It can be called infinite times. In general,
Fiber is not for such usage.

(2) If we permit Fiber#call, maybe other guys say "should support
Fiber#[]". However, as you mention, Fiber#[] is ambiguous with Fiber
local storage.

--
// SASADA Koichi at atdot dot net

#21 - 06/26/2012 05:22 AM - ko1 (Koichi Sasada)

- Description updated

- Status changed from Assigned to Closed

I close this ticket.

Please re-open it if anyone have any comment which we need to discuss again.

Files

fiber.patch	997 Bytes	01/26/2011	tenderlovemaking (Aaron Patterson)
fiber.patch	853 Bytes	01/31/2011	tenderlovemaking (Aaron Patterson)