Ruby master - Feature #5482

Rubinius as basis for Ruby 2.0

10/25/2011 09:46 AM - trans (Thomas Sawyer)

Status:	Rejected
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	3.0

Description

I'll give you three great reasons why:

1. It's Ruby in Ruby (mostly).
2. Ruby is better.
3. Read #1 and #2, again.

As Rubyists, if we truly believe that Ruby makes software development more enjoyable and more productive, then it only stands to reason that Rubinius be the future of Ruby.

History

#1 - 10/25/2011 11:22 AM - kosaki (Motohiro KOSAKI)

- Category set to Joke

Cute.
+1.

#2 - 10/25/2011 11:36 AM - headius (Charles Nutter)

I think we should roll back to Evan's original pure-Ruby implementation and get rid of all that nasty C++. After all, if we truly believe that Ruby makes software development more enjoyable and more productive, then it only stands to reason that we should eliminate all non-Ruby languages from Ruby implementations.

Also...JRuby instead!

#3 - 10/25/2011 01:59 PM - ko1 (Koichi Sasada)

(2011/10/25 12:46), Yusuke Endoh wrote:

> Come back when all 1.9 features and callcc are implemented :-)

Off topic:
I plan to propose that removing callcc from 2.0.

--
// SASADA Koichi at atdot dot net

#4 - 10/25/2011 02:25 PM - headius (Charles Nutter)

Remove fork, the C extension API, and ObjectSpace, and JRuby can be the new Ruby 2.0!

#5 - 10/25/2011 03:23 PM - spatulasnout (B Kelly)

Charles Nutter wrote:

> Remove fork, the C extension API, and ObjectSpace, and JRuby can be the new Ruby 2.0!

Add the JVM to the list and I'm sold!

;)

(Sorry Could Not Resist)

Bill

#6 - 10/25/2011 09:59 PM - headius (Charles Nutter)

How about we run on the Dalvik "I swear I'm not a JVM" VM and call it even? I think it can even fork!

#7 - 10/26/2011 07:23 AM - normalperson (Eric Wong)

Charles Nutter headius@headius.com wrote:

> I think we should roll back to Evan's original pure-Ruby
> implementation and get rid of all that nasty C++. After all, if we
> truly believe that Ruby makes software development more enjoyable and
> more productive, then it only stands to reason that we should
> eliminate all non-Ruby languages from Ruby implementations.

More seriously, I agree that Ruby should be used whenever it makes sense
and I feel MRI uses too much C.

However, I still spend more time with MRI (than Rubinius) for the
following reasons:

1) From my experience, Rubinius aims to provide a Ruby environment.
I don't want a "Ruby environment", I want Ruby to be a /part/ of my
existing Unix environment (which includes sh/awk/sed/perl among
others). I like MRI startup is still reasonably fast. I can
easily swap out existing pieces of awk/sed for one-off Ruby scripts
instead of one-off C programs when I need something I can't do as
easily in awk/sed.

```
I could use Perl instead of Ruby, but I prefer Ruby to Perl as a
language.
```

2) I have no interest/skill in debugging/fixing C++ bugs in Rubinius.
I usually fix C bugs in MRI without much effort. I suspect
more potential contributors are capable of debugging/fixing C code
written_by_other_people than C++.
Of course, more Ruby code would open the door to more maintainers, still.

With better Ruby APIs (like String#byteslice in 1.9.3) and proposed
non-blocking IO methods[1], I think more parts of MRI could be
implemented in Ruby without significant performance loss.

[1] http://redmine.ruby-lang.org/issues/5138

#8 - 10/26/2011 07:23 AM - headius (Charles Nutter)

On Tue, Oct 25, 2011 at 5:15 PM, Eric Wong normalperson@yhbt.net wrote:

> More seriously, I agree that Ruby should be used whenever it makes sense
> and I feel MRI uses too much C.

I agree wholeheartedly. This is perhaps the biggest reason I want MRI
to phase out the current C extension API in favor of one that's less
invasive, and why I'm excited for any promising JIT work for MRI. If
MRI can run Ruby fast enough, there will be less motivation to keep
moving libraries into C. That will make my life (and Evan's life) a
whole lot easier.

> However, I still spend more time with MRI (than Rubinius) for the
> following reasons:

And those are also reasons I don't think MRI will go away any time
soon. Suggestions to the contrary are just plain silly.

MRI is the most compatible Ruby implementation.

MRI is for many things still the fastest Ruby implementation (startup
time, some POSIX situations, C extensions).

MRI has the largest pool of regular contributors.

MRI is a solid, known quantity.

I will often recommend MRI when people need things that MRI does

better than JRuby, like fast startup, low-level POSIX stuff, or features we can't support like callcc and fork. MRI is a vitally important part of the Ruby implementation world.

Now, it would be nice if MRI didn't make changes to "Ruby the language" without some direct participation and buy-in from the other implementers, but that's a completely separate issue.

> With better Ruby APIs (like String#byteslice in 1.9.3) and proposed non-blocking IO methods[1], I think more parts of MRI could be implemented in Ruby without significant performance loss.

Agreed. And those APIs could easily translate to other implementations, giving us a much smaller implementation target to keep up with. Let's make it happen!

- Charlie

#9 - 10/26/2011 01:53 PM - Anonymous

On Mon, Oct 24, 2011 at 9:58 PM, SASADA Koichi ko1@atdot.net wrote:

> (2011/10/25 12:46), Yusuke Endoh wrote:
>
> > Come back when all 1.9 features and callcc are implemented :-)
>
> Off topic:
> I plan to propose that removing callcc from 2.0.

Can we have partial continuations instead? They allow for much better optimization and have been shown to be more powerful. shift/reset style continuations, maybe?

#10 - 10/26/2011 05:53 PM - headius (Charles Nutter)

On Tue, Oct 25, 2011 at 11:45 PM, Tim Felgentreff tim@nada1.de wrote:

> Can we have partial continuations instead? They allow for much better optimization and have been shown to be more powerful. shift/reset style continuations, maybe?

Fun hack of the night: https://gist.github.com/1315794

I'm sure it's not perfect, but it runs simple things correctly.

- Charlie

#11 - 10/27/2011 12:23 PM - mame (Yusuke Endoh)

Hello, Charles

2011/10/26 Charles Oliver Nutter headius@headius.com:

> On Tue, Oct 25, 2011 at 11:45 PM, Tim Felgentreff tim@nada1.de wrote:
>
> > Can we have partial continuations instead? They allow for much better optimization and have been shown to be more powerful. shift/reset style continuations, maybe?
>
> Fun hack of the night: https://gist.github.com/1315794
>
> I'm sure it's not perfect, but it runs simple things correctly.

That is not a partial continuation.
A shift should first evaluate the given block, and then
jump to the corresponding reset with the result.
The execution must not return to reset's block.

```
reset { 2 * shift {|k| 2 } } #=> must be 2, not 4
```

The feature can be implemented by using callcc.

```
require "continuation"

def shift
  callcc {|c1| $ctn[yield(proc {|v| callcc {|c2| $ctn = c2; c1[v] } })] }
end

def reset
  callcc {|c| $ctn = c; v = yield; $ctn[v] }
end
```

You may think you can implement the feature in pure Ruby
by an exception or catch/throw. But I guess it is
difficult because it requires reentrant to a shift's
block. For example,

```
p reset { [1] + shift {|j| [2] + j[[3]] } + shift {|k| [4] + k[[5]] } }
```

prints [2, 4, 1, 3, 5].

It is difficult for me to explain how it does in English :-)
I just show a pseudo reduction diagram:

```
p reset { [1] + shift {|j| [2] + j[[3]] } + shift {|k| [4] + k[[5]] } }
#=> reset { [1] + [3] + shift {|k| [4] + k[[5]] } }
#=> reset { [1] + [3] + [5] }
#=> reset { [1] + shift {|j| [2] + j[[3]] } + shift {|k| [4] + [1,3,5] } }
#=> reset { [4,1,3,5] }
#=> reset { [1] + shift {|j| [2] + [4,1,3,5] } + shift {|k| [4] + k[5] } }
#=> reset { [2,4,1,3,5] }
#=> [2, 4, 1, 3, 5]
```

Check out Olivier's original paper if you are interested.

  • Olivier Danvy and Andre Filinski (1990). "Abstracting Control".

If callcc would be removed in 2.0, we should lose a right
to play with such an interesting puzzle. Bummer.

--
Yusuke Endoh mame@tsg.ne.jp

#12 - 10/27/2011 01:23 PM - headius (Charles Nutter)

I think it is more important for Ruby to be fast, expressive and easy to use
rather than filled with "interesting puzzles" but perhaps I am in a minority
there. In any case, I doubt that 99% of developers would care if callcc went
away, and it is not supported in several implementations already.

  • Charlie (mobile)

#13 - 07/14/2012 06:38 PM - ayumin (Ayumu AIZAWA)

+1

#14 - 07/19/2012 01:43 AM - naruse (Yui NARUSE)

- Priority changed from 5 to Normal

mruby should be the basis of Ruby 3.0.

#15 - 02/26/2013 11:18 AM - ko1 (Koichi Sasada)

- Assignee set to matz (Yukihiro Matsumoto)

Some spies may be here.

#16 - 05/17/2018 08:53 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Rejected

We are working Ruby 3.0 now.