Ruby - Feature #14392

Pipe operator

01/24/2018 02:35 PM - dsferreira (Daniel Ferreira)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

I would like to see implemented in ruby a pipe operator as we have in elixir.

An example of application I have in mind is this:

```
class Foo

  def bar(var)
    puts “Hello #{var}!”
  end |> private

  def baz(arg)
    bar(arg)
  end

end

foo = Foo.new
foo.baz("Fred") # => "Hello Fred!"
```

It can also help simplify method chains:

```
class Secret

  def initialise(user, password)
    @user = user
    @password = password
  end

  def second_pass(encrypted_string)
    encrypted_string.chain_4.chain_5.chain_6
  end |> private

  ##
  #  Super encryption

  def super_encryption
    @password.chain_1.chain_2.chain_3
      |> second_pass
  end |> public

end
```

And also simplify codes like this:

```
class Foo

  def bar(*args)
    baz = args.select { |arg| arg =~ /regex/ }.first
    good?(baz)
  end

  public :bar

  def good(arg)
```

```
    arg.to_s.size > 10
  end

  private :good

end
```

to become:

```
class Foo

  ##
  # Bar public method.

  def bar(*args)
    args.select { |arg| arg =~ /regex/ }.first
      |> good?
  end |> public

  def good(arg)
    arg.to_s.size > 10
  end |> private

end
```

Lots of local variables would be saved and I would expect some performance improvements with that.

Related issues:

Related to Ruby - Feature #15799: pipeline operator	**Closed**

History

#1 - 01/24/2018 02:36 PM - dsferreira (Daniel Ferreira)

- Description updated

#2 - 01/24/2018 02:38 PM - dsferreira (Daniel Ferreira)

- Description updated

#3 - 01/24/2018 02:42 PM - dsferreira (Daniel Ferreira)

- Description updated

#4 - 01/24/2018 02:43 PM - zverok (Victor Shepelev)

We already have yield_self (2.5+) exactly for this.

```
class Foo

  ##
  # Bar public method.

  def bar(*args)
    args.select do { |arg| arg =~ /regex/ }.first
      yield_self(&method(:good))
  end.yield_self(&method(:public))

  def good(arg)
    arg.to_s.size > 10
  end.yield_self(&method(:private))
end

# Though this method of publicize and privatize seem... well, less-than-optimal for me, pipe or yield_self
```

While introducing it, different alternatives have been discussed, including Elixir-like operator, and rejected.

#5 - 01/24/2018 02:47 PM - Hanmac (Hans Mackowiak)

```
class Foo

  private def bar(var)
```

```
    puts “Hello #{var}!”
  end

  def baz(arg)
    bar(arg)
  end

end
```

method def already return their method name, so they can be used with private/protected

#6 - 01/24/2018 02:54 PM - dsferreira (Daniel Ferreira)

zverok (Victor Shepelev) wrote:

> We already have yield_self (2.5+) exactly for this.

I will not use it like that.
Not for the situations I presented.
To ugly and verbose.

#7 - 01/24/2018 02:59 PM - dsferreira (Daniel Ferreira)

Hanmac (Hans Mackowiak) wrote:

> private def bar(var)

I don't like the syntax it provides.

What I do currently is:

```
class Foo

  def bar(var)
    puts “Hello #{var}!”
  end

  private :bar

  def baz(arg)
    bar(arg)
  end

  public :baz

end
```

Looking forward for a solution to reduce the impact of that declaration.
(In my libraries I want the declaration of the access modifier for every method in every class and every module with proper documentation in place to understand what is the visibility of each method at all times.)

#8 - 01/24/2018 03:13 PM - zverok (Victor Shepelev)

Let me, then, introduce some philosphy here (I know this issue will be rejected anyways, so whatever).

When you propose new language feature, it is often useful to think a bit of it: how it plays with other features and general language intuition? Would it benefit from existing concepts, or will require the introduction of a bunch of new concepts? Will it look similar to other situations or strikingly dissimilar "special case just for the prettyness for one particular usage"?

```
value |> method_name
```

...is really awful on all those accounts.

What is method_name here? Because in ALL other contexts, naked method_name means immediate call of the method. Can we have proc there? If no, why? If yes, how it will be compatible with method_name syntax?

Applying some "next processing" to ONE value is strikingly different from applying some to COLLECTION:

In Elixir:

```
collection |> Enumerable.map(...)
one_value |> String.split(...)
```

In Ruby 2.5

```
collection.map { ... }
one_value.yield_self { ... }
# with some method
collection.map(&method(:puts))
one_value.yield_self(&method(:puts))
```

Your proposal:

```
collection.map { ... }
one_value |> something
```

It already looks messed up :)

I agree that yield_self(&method(:puts)) is less-than-optimal. Two problems here are:

- method(), the price we pay for having ()-less method calls; there are ongoing discussions of shorthand syntax;
- REALLY unfortunate yield_self name, which I'll never get tired to blame.

But otherwise, it is perfectly idiomatic and readable Ruby, playing well with all usual metaphors and intuitions.

People tend to constantly mix "syntax akin to other language" or "syntax as I imagine it in my head" with "syntax which is clear and idiomatic". ¯_(□)_/¯

#9 - 01/24/2018 03:16 PM - zverok (Victor Shepelev)

> private def bar(var)

> I don't like the syntax it provides.

So, what you saying is "I don't like how the consistent set of features in language looks [it can't even be said to be "verbose", man!], so please introduce a new inconsistent feature for me to like it"?

That's the way, obviously.

#10 - 01/24/2018 04:10 PM - dsferreira (Daniel Ferreira)

zverok (Victor Shepelev) wrote:

> It already looks messed up :)

Didn't understand very well how messed up it is.
Is that syntax impossible to implement or messed up just because the way ruby people tend to look at code syntax?

Is there a problem with this if we must?:

```
{ ... } |> collection.map
one_value |> something
```

Am I asking here to break backwards compatibility?

#11 - 01/24/2018 04:24 PM - dsferreira (Daniel Ferreira)

zverok (Victor Shepelev) wrote:

> That's the way, obviously.

Fortunately not every one look at things the same way, and I believe ruby core allows us to come in and discuss matters.
This is an important matter to me.
I know not many people looks at it the way I do but still it is my view and my requirement.

In my code I like to organise the methods in alphabetical order and not by public, protected, private hierarchy.

The reason why I do that?

Because that simplifies a bunch of things. It may be the way my mind works.
It doesn't mean I'm not able to work with the code format that is out there in the wild to see.

Code like mine I never saw.
But I do it like that because I consider it the best way.

This type of code:

```
class Foo

  private def bar
  end

  ##
  # Baz public method.

  public def baz
  end

end
```

hurts my eyes. So the best I can achieve right now is:

```
class Foo

  def bar
  end

  private :bar

  ##
  # Baz public method.

  def baz
  end

  public :baz

end
```

Which is fine but not ideal and I aim for the ideal.

It will be reject anyways but I will still continue to seek for a better option.

#12 - 01/24/2018 08:54 PM - shevegen (Robert A. Heiler)

> It will be reject anyways but I will still continue to seek for a better option.

Who knows - but I think it may be too difficult to change it at this point.

When it comes to personal preferences, what may work for you may not work for others.

At the end of the day it's matz who decides about features (and syntax).

In streem, matz uses a different concept/syntax, to some extent:

https://github.com/matz/streem

```
seq(100) | map{x->
  if (x % 15 == 0)      "FizzBuzz"
  else if (x % 3 == 0) "Fizz"
  else if (x % 5 == 0) "Buzz"
  else                  x
} | stdout
```

No |> though.

Note that I have nothing at all against the suggestion here per se, but I believe that there is a cost associated for change in many situations, and there should be some kind of benefit for the change.

I agree with "private def bar" style being ugly by the way - I always denote when methods are private after the trailing "end" of the method. But I also very rarely use private in ruby in the first place - I mostly do so if the class or API may need some additional "information" to people using the class, like what they should not use (as far as I know, they can call it anyway via .send() so the private/public distinction in ruby does not make a lot of sense to me; and I <3 .send(). Different languages define and use OOP differently).

Style in ruby differs immensely from one ruby hacker to the next. It is also difficult to reason about. I am sure others may find my ruby style ugly. I find it pretty. :)

The biggest "rule" I have is that code that I write, should be really simple as much as possible. Reason being primary is because I don't want to burden the cognitive load I have while writing (and as importantly, maintaining) ruby code.

That is also why I do not completely agree with the rubocop style guide - but rubocop's biggest "wins" are that you can declare your own style, and then auto-correct/enforce it, IMHO.

> I believe ruby core allows us to come in and discuss matters.

Indeed and at the end of the day, you only have to convince matz. But I think that the trade-off here is not really worth it; what may be worth exploring may be to have ruby hackers re-define syntax part of ruby ... but I am also not sure if it is ultimately worth to be had (I think most of ruby is very elegant; with more proliferation, we may have more ugly code too, and I already think there is quite a bit of ugly ruby out there in the wild, but that is my personal opinion).

> [...] "I don't like how the consistent set of features in language looks [it can't even be said to be "verbose", man!], so please introduce a new inconsistent feature for me to like it [...]

To the above, designing an absolutely perfect symmetric language that also follows a "there is more than one way to do things", is difficult. There are even some "functional" features in ruby; currying/lambda/proc and so on.

Sometimes later changes may change older parts of ruby. I think if you look at people writing ruby, say, 10 years ago or more, that ruby will be quite different from ruby today (provided that the author had similar experience, say after 3 years of using ruby extensively every day).

Not every change is necessarily due to consistency alone. Take the lonely person operator - that one came primarily because someone had a use case (lots of nil queries in a rails codebase), matz agreed and so came the lonely person operator staring at the dot.

Anyway, to conclude - I think elixir is cool but ruby is not elixir and simple 1:1 mappings in syntax changes may impact lots of people, without necessarily providing substantial benefits IMO. One can dispute whether there is enough benefit, but to me, I don't really see the real benefit that we would get in ruby at this point in time.

By the way, you can always try to have it discussed at the next developer meeting.

#13 - 01/24/2018 09:41 PM - dsferreira (Daniel Ferreira)

shevegen (Robert A. Heiler) wrote:

> No |> though.

Hi Robert, many thanks for the extensive comment.
The idea of using "|>" as the proposed pipe operator came from my analysis of https://bugs.ruby-lang.org/issues/13581 issue coupled with long time syntax thoughts.
I proposed a syntax sugar for 13581 as "|>" or "<|". You can read the logic there.
It makes sense in my head.
Resembles the way bash instructions work in multiple directions.
Ruby also has a lot of sh/bash logic in its core.
Using "|" coupled with "<" and ">" seems to me an easy and intuitive way to overcome the syntax problem of having the three operators already with their own meanings in isolation.

> But I also very rarely use private in ruby in the first place

People don't really care about it but I do care a lot.
I like to work in top of a very well defined API.
The library API should be reduced to its minimum as much as possible.
And I want total freedom to do whatever inside the library.
So I use extensively public, protected and private.
I have some feature requests still open that I'm planning to implement in the near future and submit a patch.
One of them is https://bugs.ruby-lang.org/issues/9992 which is about the introduction of an internal access modifier which would define an internal interface just visible to the library namespace.

So you can see that this matters are very important to me and impact my daily work.
I love ruby but I'm not being able to express myself in the language like I would like to.
We don't live in an ideal world I know but I believe that my way of seeing the design of a library/gem or module/class brings a new insight into the ecosystem that may be good to consider by the community.

It is not easy to present the full picture of what I have in my mind so I'm starting to do it bit by bit.
With time people will understand better what is my vision and what am I proposing.

> That is also why I do not completely agree with the rubocop style guide

I'm with you 100% on this.
But you know, when you work in a big company and if you are the only one speaking in one direction you better end your talk.
Rubocop is now the default in the enterprise ruby.
Luckily I'm now in a position where my voice is heard.
But what I can't do is to tell about features that just exist in my head.
So here I am trying to make my dream a reality.

> I think elixir is cool but ruby is not elixir

This proposal it is not about Elixir.
I have never coded a single line in Elixir and I'm not planning to do it in the near future.

I mentioned Elixir just as a reference nothing else.
It seems people like language references when discussing this matters.
If that is not the case please say so and I will stop using those references.

#14 - 01/25/2018 02:09 AM - mame (Yusuke Endoh)

dsferreira (Daniel Ferreira) wrote:

```
def bar(var)
  puts “Hello #{var}!”
end |> private
```

Well... I was neutral to a pipe operator, but if people overuse (or even abuse) it like this, I'm now a bit against the feature.

#15 - 01/25/2018 05:49 AM - dsferreira (Daniel Ferreira)

mame (Yusuke Endoh) wrote:

> I'm now a bit against the feature

Yusuke what would you propose as an alternative?

Note that my main requirement behind this is to be able to use private after the method definition without the need to type its name a second time.

I'm open to suggestions...

#16 - 01/25/2018 06:32 AM - shyouhei (Shyouhei Urabe)

I don't remember any programming language which puts the visibility of a method at the trailer of a method body. Is it really worth adding a new syntax to replace private def... ?

#17 - 01/25/2018 07:07 AM - dsferreira (Daniel Ferreira)

shyouhei (Shyouhei Urabe) wrote:

> Is it really worth adding a new syntax to replace private def... ?

For me it is Shyouhei.
Like I said before, I care a lot about the visibility of my methods.
The clear definition of each interface and its preservation over time it is a top concern in my architectural work.
The interface integrity is this way guaranteed by a strict rule deeming that every method has its own access modifier set.
This applies to public, protected and private.
Naming is hard and very important and the API resembles that. It is sweat printed in. (Just to try to express my emotions about this)

We are talking about hundreds to not say thousands of method name duplications that I have in my code base currently.
I don't mind having the extra work but I'm not everybody.

Now, why do I require the visibility being set after the method definition?
Because I order my methods alphabetically and I don't want to order them by visibility.
By using: "private def ..." alphabetical order based on method name makes little sense to my mind.

I code ruby because of its elegance and fluidity so I always strive to write code that makes sense to my eyes.

The proposal of the pipe operator makes perfect sense to me because it resembles my bash memories and it comes as natural as a no brainer.

#18 - 01/25/2018 07:25 AM - dsferreira (Daniel Ferreira)

Shyouhei,

To add up to this:

When I look into a specific method my first concern is about the source code not its visibility.
Visibility information it is important but secondary when doing code analysis.
Why? Because I'm totally confident that there are no issues associated to the visibility that is in place.
It is something that is their set to be forgotten.

For me something like this would be even better:

```
class Foo

  def bar
  end.pri

  def foo
  end.pro

  def foobar
  end.pub

end
```

Which I could easily do by extending Symbol class but I will not make my code dependent on some "quirk" of my mindset that is not part of ruby core reason being I don't own the code. Other developers use it as well and I don't want to impose this things to anyone.

#19 - 01/25/2018 07:35 AM - Hanmac (Hans Mackowiak)

Symbol#pri and others will never work because you failed to understand how Object work in ruby

so NO "easily do by extending Symbol class"

---

there is also using them as modifier for the class scope

```
class Foo

private
  def bar
  end

protected
  def foo
```

```
  end

public
  def foobar
  end
end
```

#20 - 01/25/2018 07:36 AM - shyouhei (Shyouhei Urabe)

dsferreira (Daniel Ferreira) wrote:

> shyouhei (Shyouhei Urabe) wrote:
>
> > Is it really worth adding a new syntax to replace private def... ?
>
> For me it is Shyouhei.
> Like I said before, I care a lot about the visibility of my methods.
> The clear definition of each interface and its preservation over time it is a top concern in my architectural work.
> The interface integrity is this way guaranteed by a strict rule deeming that every method has its own access modifier set.
> This applies to public, protected and private.
> Naming is hard and very important and the API resembles that. It is sweat printed in. (Just to try to express my emotions about this)
>
> We are talking about hundreds to not say thousands of method name duplications that I have in my code base currently.
> I don't mind having the extra work but I'm not everybody.

To be clear I am not against specifying method visibility.

> Now, why do I require the visibility being set after the method definition?
> Because I order my methods alphabetically and I don't want to order them by visibility.
> By using: "private def ..." alphabetical order based on method name makes little sense to my mind.

Don't you have difficulties writing elixir code then? They, too, don't take visibilities at the last.

> I code ruby because of its elegance and fluidity so I always strive to write code that makes sense to my eyes.
>
> The proposal of the pipe operator makes perfect sense to me because it resembles my bash memories and it comes as natural as a no brainer.

If you prefer sound visibility specifiers rather than implicit ones, I think that a visibility of a method shall be placed somewhere near the method name. By placing private after the end of a method body, that specifier should become too hard to find out the corresponding method name. To me, def ... end |> private seems far less "fluent" than private def ...

#21 - 01/25/2018 08:06 AM - dsferreira (Daniel Ferreira)

Hanmac (Hans Mackowiak) wrote:

> Symbol#pri and others will never work

You're right. Not to much thought put on that one. It is not so easy as it seems from my comment but I wouldn't say never. When working with ruby "never" is a very big statement.

#22 - 01/25/2018 08:11 AM - dsferreira (Daniel Ferreira)

shyouhei (Shyouhei Urabe) wrote:

> Don't you have difficulties writing elixir code then?

I mentioned already that Elixir it is there only as a reference.
I do not code Elixir.

> I think that a visibility of a method shall be placed somewhere near the method name.

The reason I think otherwise is this:

When I look into a specific method my first concern is about the source code not its visibility.
Visibility information it is important but secondary when doing code analysis.
Why? Because I'm totally confident that there are no issues associated to the visibility that is in place.
It is something that is their set to be forgotten.

#23 - 01/25/2018 08:13 AM - dsferreira (Daniel Ferreira)

Also when I want to look specifically at the API of a certain class I look into its documentation not the code base.

#24 - 02/20/2018 07:08 PM - joelvh (Joel Van Horn)

Hanmac (Hans Mackowiak) wrote:

> Symbol#pri and others will never work because you failed to understand how Object work in ruby
>
> so NO "easily do by extending Symbol class"
>
> ---
>
> there is also using them as modifier for the class scope
>
> ```
> class Foo
>
> private
>   def bar
>   end
>
> protected
>   def foo
>   end
>
> public
>   def foobar
>   end
> end
> ```

Daniel,

I'm just curious... does the modifier in the class scope work for you as Hans mentioned, or is it really overall piping syntax that you want for more than just scoping methods? You can use that modifier before each method you define, and it doesn't have to be out-dented if that makes it appear like you're sectioning off the methods in your class.

#25 - 03/28/2018 04:41 PM - bughit (bug hit)

this important feature discussion has been derailed by a bad description/justification

the purpose of the pipe operator is to enable sane, readable, functional chaining. Controlling method visibility through it is not worth even mentioning.

it's fairly self evident that

fn1 |> fn2 |> fn3 |> fn4 |> fn5

is superior to

fn5(fn4(fn3(fn2(fn1()))))

the longer the chain the more so

This would not be a casual drop in for ruby, as it needs some sort of partial application syntax (using <> here), something like this

value |> method_or_proc<1, 2>

method_or_proc<1, 2> would produce (pseudocode since proc invocation uses different syntax) ->arg{method_or_proc(arg, 1, 2)}

method_or_proc<1, 2, _> would produce ->arg{method_or_proc(1, 2, arg)}

you could get fancier

method_or_proc<1, *, 2> => ->*args{method_or_proc(1, *args, 2)}

there's little doubt that this feature would make functional style a lot more pleasant in ruby

#26 - 12/13/2018 09:51 AM - shuber (Sean Huber)

@dsferreira @zverok (Victor Shepelev) @shevegen @bughit

How do you feel about the syntax from this working proof of concept? https://github.com/LendingHome/pipe_operator

```
"https://api.github.com/repos/ruby/ruby".pipe do
  URI.parse
  Net::HTTP.get
```

```
  JSON.parse.fetch("stargazers_count")
  yield_self { |n| "Ruby has #{n} stars" }
  Kernel.puts
end
#=> Ruby has 15120 stars

-9.pipe { abs; Math.sqrt; to_i } #=> 3

# Method chaining is supported:
-9.pipe { abs; Math.sqrt.to_i } #=> 3

# Pipe | for syntactic sugar:
-9.pipe { abs | Math.sqrt.to_i } #=> 3

[9, 64].map(&Math.pipe.sqrt)           #=> [3.0, 8.0]
[9, 64].map(&Math.pipe.sqrt.to_i.to_s) #=> ["3", "8"]
```

#27 - 06/30/2019 10:28 AM - Eregon (Benoit Daloze)

- Related to Feature #15799: pipeline operator added