Ruby trunk - Feature #11262

Make more objects behave like "Functions"

06/15/2015 11:55 AM - jwmittag (Jörg W Mittag)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

What is a Function?

In Ruby, we have the Proc class to represent objects which are "function-like". But, in true object-oriented / duck-typing fashion, an object doesn't actually have to be an instance of Proc in order to be treated as a function, it only needs to respond to call. For cases, where a Proc instance is absolutely required (mostly, the & unary prefix ampersand "make-me-a-block" operator), there is the to_proc conversion.

So, in short: if an object wants to be a function, it **MUST** respond to call, and **SHOULD** also respond to to_proc.

There are some objects in Ruby that *could* be seen as functions, but currently don't respond to call or to_proc:

Array as mapping

An array is a mapping from indices to elements. "Mapping" is just a different word for (partial) function, though! I propose, that Array should implement call and to_proc in the following manner:

```
class Array
  alias_method :call, :[]

  def to_proc
    method(:call).to_proc
  end
end
```

Hash as mapping

A hash is a mapping from keys to values. I propose, that Hash should implement call and to_proc in the following manner:

```
class Hash
  alias_method :call, :[]

  def to_proc
    method(:call).to_proc
  end
end
```

[Note: #11653 implements the to_proc part of this proposal.]

Set as predicate

A set is a mapping from values to booleans, i.e. a set is the same as its include? predicate. This would mean, for example, that I can pass a Set as a predicate to methods like Enumerable#select. I propose, that Set should implement call and to_proc in the following manner:

```
require 'set'

class Set
  alias_method :call, :include?

  def to_proc
    method(:call).to_proc
```

```
  end
end
```

I believe that these three additions are worthwhile and fairly uncontroversial. They match with the way arrays, maps and especially sets are treated in mathematics and in other programming languages. E.g. in both Clojure and Scala, arrays, sets and maps are functions and use function application syntax for accessing values. Scala doesn't even have indexing syntax.

Here are some potential use cases:

```
numbers_to_words = %w[zero one two three four five six seven eight nine ten eleven twelve]

[4, 7, 1, 0, 8].map(&numbers_to_words)
# => ['four', 'seven', 'one', 'zero', 'eight']


allowed_languages = Set[:ruby, :python, :scala, :scheme]

%i[ruby c cplusplus scala java perl].select(&allowed_languages)
# => [:ruby, :scala]
```

Here is a more "wild" proposal that is much more controversial. I don't actually propose adding this to Ruby, but I will mention it here as food for thought:

Class as factory

If you squint your eyes, tilt your head sideways and look at it juuuuuuust right, a class is a factory for objects. In other words, it is a function from constructor arguments to instances:

```
class Class
  alias_method :call, :new

  def to_proc
    method(:call).to_proc
  end
end
```

Example:

```
class Person
  def initialize(name)
    @name = name
  end
end

%w[matz ko1 charlie].map(&Person)
# => [#<Person:0xdeadbeef481523 @name="matz">, #<Person:0xdeadbeef815234 @name="ko1">, #<Person:0x
deadbeef152342 @name="charlie">]
```

Incompatibilities

This proposal conflicts with #10829, which proposes to use Array#to_proc for a completely different purpose.

I believe that having Arrays behave as functions from indices to elements is natural, unsurprising, and well in line with both mathematics and other languages.

Related

1. #11653 implements a small subset of my proposal.
2. The code duplication encountered here suggests refactoring to extract two new mixins in the Ruby core library:

   ```
   module Callable
     def to_proc
       method(:call).to_proc
   ```

```
  end
end

module Indexable
  alias_method :call, :[]
end
```

However, this is out of scope of this discussion and *not* part of this particular feature proposal.

[NOTE: I originally posted this in CommonRuby, which according to its wiki is "The official place to submit feature proposal for Ruby" but from my observation, almost all Ruby feature requests actually get filed at Ruby trunk.]

History

#1 - 02/22/2016 03:27 PM - jwmittag (Jörg W Mittag)

- Project changed from CommonRuby to Ruby trunk

- Description updated

#2 - 02/22/2016 03:46 PM - zverok (Victor Shepelev)

For me, this thing looks like some kind of over-simplification (leading to ambiguity).
Both cases are handled with sligtly longer statements with much more clear intent:

```
%i[ruby c cplusplus scala java perl].select(&allowed_languages.method(:include?))
%w[matz ko1 charlie].map(&Person.method(:new))
```

This code is DRY, clear and readable even for novice (though can cause some kind of surpise "wow, I could do this?!").
The only "too long" thing here is entire word "method" (so, map{|s| Person.new(s)} is a bit shorter, while not being that DRY).

Though, I'm kind of retrograde :)

#3 - 02/23/2016 12:55 AM - jwmittag (Jörg W Mittag)

Victor Shepelev wrote:

> For me, this thing looks like some kind of over-simplification (leading to ambiguity).
> Both cases are handled with sligtly longer statements with much more clear intent:
>
> ```
> %i[ruby c cplusplus scala java perl].select(&allowed_languages.method(:include?))
> %w[matz ko1 charlie].map(&Person.method(:new))
> ```

I don't want to focus too much on the second example, because like I said: the "Class as Factory Function" is *not* actually part of my proposal, I mention it only as food for thought. However, if you have ever seen or implemented a Factory Pattern in Ruby, if you have ever seen or used JavaScript, if you have ever seen or used Dart, if you have ever seen or used one of the many object systems in Scheme or Clojure, if you are familiar with some of the theoretic formalisms for OO, if you have ever read some of William R. Cook's writings on OO, then the idea that a class is a function that creates objects will look completely natural to you.

What does a class do in Ruby? It holds methods, but that is actually just inherited by classes being special-cases of modules. What distinguishes classes from modules in Ruby is above all Class#allocate and Class#new, i.e. the possibility to create values. And isn't "creating values" basically exactly what a function does?

In other languages, classes are often described as "templates for objects". But that isn't actually true in Ruby. Classes don't describe the shape of objects. Instance variables get added to objects over time, as methods are being called on them, thus objects constantly change their shape over time, the shape is not determined by the class. In fact, all objects start out with the exact same shape, only by calling methods (initialize being one such method) on them *after* they have already been constructed by the class do they change their shape.

> This code is DRY, clear and readable even for novice (though can cause some kind of surpise "wow, I could do this?!").
> The only "too long" thing here is entire word "method" (so, map{|s| Person.new(s)} is a bit shorter, while not being that DRY).

Well, it's not really about saving a few keystrokes. It's basically about semantics: Ruby has a very clear idea about what constitutes a "Function", namely, a "Function" is any object that responds to call and to_proc. And it's also very clear (at least to me, and I am writing this proposal partly to figure out whether this is true more broadly as well) that an array is a function from integers to elements, a hash is a function from keys to values, and a set is a function from elements to booleans (in fact, as William R. Cook showed in *On Understanding Data Abstraction, Revisited*, identifying sets with their characteristic functions is *the* OO way to implement sets). In other languages, those data structures *literally* inherit from a Function class (e.g. in Scala) or implement a Function interface (e.g. in Clojure).

#4 - 02/23/2016 12:58 AM - jwmittag (Jörg W Mittag)

- Description updated

Fix link to the Clojure docs after they changed their URI structure.

#5 - 06/24/2016 08:40 AM - jwmittag (Jörg W Mittag)

- Description updated

#6 - 06/24/2016 08:43 AM - jwmittag (Jörg W Mittag)

Note that Hash#to_proc has already been implemented as part of another proposal: #11653