THE VALUE OF BEING LAZY
or How I Made OpenStruct 10X Faster

Erik Michaels-Ober
@sferik
In Ruby, everything is an object.

∀ thing
thing.is_a?(Object) #=> true
In Ruby, every object has a class.

∀ object
object.respond_to?(:class) #=> true
In Ruby, every class has a class.

∴

Object.respond_to?(:class) #=> true
Object.class #=> Class
You can use classes to create new objects:

```ruby
object = Object.new
object.class #=> Object
```
You can use classes to create new classes:

```ruby
klass = Class.new
klass.class #=> Class
```
Usually, we create classes like this:

```ruby
class Point
  attr_accessor :x, :y
  def initialize(x, y)
    @x, @y = x, y
  end
end
```
You can replace such simple classes with structs:

Point = Struct.new(:x, :y)
OpenStruct requires even less definition:

```ruby
point = OpenStruct.new
point.x = 1
point.y = 2
```
In this way, OpenStruct is similar to Hash:

```
point = Hash.new
point[:x] = 1
point[:y] = 2
```
You can even initialize OpenStruct with a Hash:

```ruby
point = OpenStruct.new(x: 1, y: 2)
point.x #=> 1
point.y #=> 2
```
So why use OpenStruct instead of Hash?
Test double

validator = OpenStruct.new
expect(validator).to receive(:validate)

code = PostalCode.new("94102", validator)
code.valid?
API response

```ruby
user = OpenStruct.new(JSON.parse(response))
user.name #=> Erik
```
Configuration object

def options
    opts = OpenStruct.new
    yield opts
    opts
end
So OpenStruct is useful…but slow.
It’s a shame OpenStruct is so dang slow.
Steps to optimize code

1. Complain that code is slow on Twitter
2. ???
3. Profit
Actual steps to optimize code

1. Benchmark
2. Read code
3. Profit
Actual steps to optimize code

1. Benchmark
2. Read code
3. Profit
require "benchmark/ips"

Point = Struct.new(:x, :y)
def struct
  Point.new(0, 1)
end

def ostruct
  OpenStruct.new(x: 0, y: 1)
end

Benchmark.ips do |x|
  x.report("ostruct") { ostruct }
  x.report("struct") { struct }
end
Comparison:

struct: 2927800.2 i/s

ostruct: 84741.1 i/s - 34.55x slower
Actual steps to optimize code

1. Benchmark
2. Read code
3. Profit
def initialize(hash = nil)
    @table = {}
    if hash
        hash.each_pair do |k, v|
            k = k.to_sym
            @table[k] = v
            new_ostruct_member(k)
        end
    end
end
def new_ostruct_member(name)
    name = name.to_sym
    unless respond_to?(name)
        define_singleton_method(name) { @table[name] }
        define_singleton_method("#{name}=") { |x| @table[name] = x }
    end
    name
end
def method_missing(mid, *args)
    len = args.length
    if mname = mid[/.*(?==\z)/m]
        @table[new_ostruct_member(mname)] = args[0]
    elsif len == 0
        if @table.key?(mid)
            new_ostruct_member(mid)
            @table[mid]
        end
    end
end
def initialize(hash = nil)
    @table = {}
    if hash
        hash.each_pair do |k, v|
            k = k.to_sym
            @table[k] = v
            new_ostruct_member(k)
        end
    end
end
Before:

struct:  2927800.2 i/s

ostruct:  84741.1 i/s - 34.55x slower
After:

struct:  2927800.2 i/s

ostruct:  940170.4 i/s - 3.11x slower
Define OpenStruct attributes lazily #1033

sferik wants to merge 1 commit into ruby:trunk from sferik:lazy-struct-method-definition

sferik commented on Sep 26

Instead of defining two methods—a reader and writer—for each OpenStruct attribute when it is initialized, define them lazily, the first time either one is called. This adheres to the principle of “pay for use”: methods that are never accessed are never defined. This optimization makes initialization an order of magnitude faster for objects with 100 attributes. In the worst-case scenario, where every attribute is accessed, performance is no worse than it is today.

Benchmark

```ruby
require 'benchmark/ips'
require 'ostruct'

N = 100
ATTRS = {a:...zz}.take(N)
HASH = ATTRS.map { |x| [x, x] }{}

def ostruct
  OpenStruct.new(HASH)
end
```
It’s a shame OpenStruct is so dang slow.

Erik Michaels-Ober @sferik · Sep 26
@benlovell Fixed that for you: github.com/ruby/ruby/pull....

Andy Pike @andypike · Sep 26
@sferik @benlovell I just looked at your PR, saw the diff and was like o.O
commit 62e41d3f2e48422bbdf1bb2db83ae60b255b1a1a
Author: matz <matz@b2dd03c8-39d4-4d8f-98ff-823fe69b080e>
Date:   Fri Jan 16 12:19:09 1998 +0000

Initial revision

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@8 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

commit 29cef5f795043a048a58874d90de0dfe01aa12ea
Author: nobu <nobu@b2dd03c8-39d4-4d8f-98ff-823fe69b080e>
Date:   Wed Oct 2 16:45:35 2002 +0000

use Object#class instead of deprecated Object#type.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@8 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

commit aae36756d6cc4d5debcaabbd03379796bc41360bc4
Author: matz <matz@b2dd03c8-39d4-4d8f-98ff-823fe69b080e>
Date:   Sun Nov 3 11:04:35 2002 +0000

* object.c (Init_Object): added Object#object_id, new name for
  Object#id. [new]

* object.c (rb_obj_id_obsolete): give warning for Object#id.

* numeric.c (fix_intern): added Fixnum#to_sym. [new]
In case you missed it, OpenStruct is 10X faster in Ruby 2.3 than it was in earlier versions of Ruby.
Lazy evaluation
Enumerator::Lazy
lazy_integers = (1..Float::INFINITY).lazy
lazy_integers.collect { |x| x ** 2 }.select { |x| x.even? }.reject { |x| x < 1000 }.first(5)
#=> [1024, 1156, 1296, 1444, 1600]
require "prime"

lazy_primes = Prime.lazy

lazy_primes.select { |x| (x - 2).prime? }.collect { |x| [x - 2, x] }.first(5)

# => [[3, 5], [5, 7], [11, 13], [17, 19], [29, 31]]
module Enumerable
  def repeat_after_first
    unless block_given?
      return to_enum(__method__) { size * 2 - 1 if size }
    end

    each.with_index do |*val, index|
      index == 0 ? yield *val : 2.times { yield *val }
    end
  end
end
require "prime"

lazy_primes = Prime.lazy

lazy_primes.repeat_after_first.
  each_slice(2).
  select { |x, y| x + 2 == y }.
  first(5)

#=> [[3, 5], [5, 7], [11, 13], [17, 19], [29, 31]]
require "date"

lazy_dates = (Date.today..Date.new(9999)).lazy

lazy_dates.select { |d| d.day == 13 }.

select { |d| d.friday? }.

first(10)
lazy_file = File.readlines("/path/to/file").lazy
lazy_file.detect { |x| x =~ /regexp/ }
Being lazy is efficient.
Being lazy is elegant.
Thanks to:

Zachary Scott
ROSS Conf
Rails Israel
Thank you