# Ruby master - Feature #12008

## Immutable object graphs (a.k.a. deep freeze)

01/19/2016 10:54 PM - bascule (Tony Arcieri)

| | |
|---|---|
| **Status:** | Open |
| **Priority:** | Normal |
| **Assignee:** | |
| **Target version:** | |

**Description**

Hi there. I know some sort of "#deep_freeze" construct has been proposed many times before. I proposed it in this blog post in 2012: https://tonyarcieri.com/2012-the-year-rubyists-learned-to-stop-worrying-and-love-the-threads

I have also read this issue: https://bugs.ruby-lang.org/issues/2509

This is an area I have thought a lot about as I deal with many people's multithreaded Ruby programs and shared state mutation bugs / data races. I would like Ruby to support an option to fix this problem, and have guaranteed immutable state for entire object graphs rather than individual objects.

The main problem, as I understand it, is existing proposals wish to recurse entire object graphs and deep freeze all their contents. This is particularly problematic with things like Proc.

I propose a simpler #deep_freeze where it is opt-in by individual classes.

It should also be shallow instead of recursive: individual objects should only traverse their local references, determine if they are *already* #deep_freeze(d), and if not, bail out. No need for recursion.

The basic idea being: to #deep_freeze an object, it can only refer to other #deep_freeze(d)/frozen objects. We do not attempt to walk the entire object graph/aggregate and freeze all the children recursively. Instead merely *check* that all of the children are *already* #deep_freeze(d)/frozen, and if they aren't, bail out.

This has many beneficial properties:

- We rely on explicit support from objects to support #deep_freeze. If they don't, bail out. No weird behaviors recursing references and running into crazy Proc object fractals
- We only need an object-local view of the world to determine if we're #deep_freeze-able. The check is little more than walking *local* object references (i.e. ivars in most cases) and if everything checks out, setting a flag
- We get guarantees that entire object graphs/aggregates are completely frozen all the way down

This requires programmers build up deep frozen object aggregates by starting at the leaves and building more and more complicated aggregates out of deeply frozen pieces.

I think the changes that have gone into Ruby 2.3 are making this easier and more realizable. I would like to continue this trajectory and provide real guarantees for Rubyists about which objects reference object graphs that are truly immutable.

I think immutable state has huge benefits for a lot of areas, most notably concurrency and security.

### History

**#1 - 01/20/2016 09:50 AM - normalperson (Eric Wong)**

bascule@gmail.com wrote:

> I think immutable state has huge benefits for a lot of areas, most notably concurrency and security.

Agree. Maybe something along the lines of the following?

```
module DeepFreezable
  def deep_freeze_check!(obj)
    obj.frozen? or raise TypeError, "#{obj.inspect} not frozen in #{inspect}"
  end

  def ivar_deep_frozen?
    instance_variables.each do |iv|
```

```
      deep_freeze_check!(instance_variable_get(iv))
    end
  end
end

class Array
  include DeepFreezable

  def deep_freeze
    ivar_deep_frozen?
    each { |obj| deep_freeze_check!(obj) }
    freeze
  end
end

class String
  include DeepFreezable

  def deep_freeze
    ivar_deep_frozen?
    freeze
  end
end

class Hash
  include DeepFreezable

  def deep_freeze
    ivar_deep_frozen?
    each do |key, val|
      deep_freeze_check!(key)
      deep_freeze_check!(val)
    end
    freeze
  end
end

require 'test/unit'
class TestDeepFreeze < Test::Unit::TestCase
  def test_deep_freeze
    assert_predicate({}.deep_freeze, :frozen?)
    assert_raise(TypeError) { {'foo' => 'bar'.dup }.deep_freeze }
    assert_predicate({'foo' => 'bar'.freeze }.deep_freeze, :frozen?)

    assert_raise(TypeError) { %w(a b c).map(&:dup).deep_freeze }
    assert_predicate(%w(a b c).map(&:freeze).deep_freeze, :frozen?)

    s = String.new
    s.instance_eval { @foo = 'bar'.dup }
    assert_raise(TypeError) { s.deep_freeze }
    s.freeze
    assert_raise(TypeError) { s.deep_freeze }
    s.instance_variable_get(:@foo).freeze
    assert_predicate s.deep_freeze, :frozen?
  end
end
```



#### #2 - 06/27/2019 10:29 PM - jeremyevans0 (Jeremy Evans)

*- Backport deleted (2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN)*

*- Tracker changed from Bug to Feature*