Ruby master - Feature #16175

Object#clone(freeze: true) is inconsistent with Object#clone(freeze: false)

09/21/2019 05:33 PM - zverok (Victor Shepelev)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

In #12300, the new keyword freeze: was introduced, allowing this:

```
h = {}.freeze
h.clone.frozen?
# => true
h.clone(freeze: false).frozen?
# => false
```

Though, it turns to me that behavior is not symmetric:

```
h = {}
h.frozen?
# => false
h.clone.frozen?
# => false
h.clone(freeze: true).frozen?
# => false -- I expected true here!
```

I wonder, if it is "by design" and should be addressed in docs, or just an implementation inconsistency that can be fixed?

Associated revisions

Revision 4f7b435c - 03/22/2020 04:30 PM - jeremyevans (Jeremy Evans)

Support obj.clone(freeze: true) for freezing clone

This freezes the clone even if the receiver is not frozen. It
is only for consistency with freeze: false not freezing the clone
even if the receiver is frozen.

Because Object#clone is now partially implemented in Ruby and
not fully implemented in C, freeze: nil must be supported to
provide the default behavior of only freezing the clone if the
receiver is frozen.

This requires modifying delegate and set, to set freeze: nil
instead of freeze: true as the keyword parameter for
initialize_clone. Those are the two libraries in stdlib that
override initialize_clone.

Implements [Feature #16175]

History

#1 - 09/21/2019 08:26 PM - jeremyevans0 (Jeremy Evans)

- Backport deleted (2.5: UNKNOWN, 2.6: UNKNOWN)

- Subject changed from Object#clone(freeze: true) to Make Object#clone(freeze: true) return frozen clone even if receiver is not frozen

- Tracker changed from Bug to Feature

The freeze: false option was intended to be: "do not freeze clone if receiver is already frozen". The Object#clone documentation states: "#clone copies the frozen (unless :freeze keyword argument is given with a false value) and tainted state of obj". The behavior when freeze: true is provided is therefore unspecified, and therefore, I do not think this should be considered a bug.

The reason freeze: false was introduced is because you previously could not use clone with frozen objects with singleton classes/extended modules and keep the ability to modify copies of the frozen objects. Adding freeze: false made something previously impossible in Ruby possible. Adding freeze: true for consistency doesn't enable new behavior, as you can use clone.freeze instead of clone(freeze: true). That being said, I'm not

opposed to freeze: true being supported.

#2 - 09/22/2019 09:30 AM - shevegen (Robert A. Heiler)

I sort of agree with zverok at the least for the expectation of freeze: true working. IMO for boolean toggle-values I would think it is simpler to have both variants work; then again I don't think I need either of the two variants myself. It's interesting to point out that freeze adds a bit of complexity though.

#3 - 09/22/2019 09:59 AM - zverok (Victor Shepelev)

- Backport set to 2.5: UNKNOWN, 2.6: UNKNOWN

- Subject changed from Make Object#clone(freeze: true) return frozen clone even if receiver is not frozen to Object#clone(freeze: true) is inconsistent with Object#clone(freeze: false)

- Tracker changed from Feature to Bug

- File freeze-true.patch added

jeremyevans0 (Jeremy Evans) thanks for your answer.

Let me explain my point a bit.

I come upon this inconsistency (or what **I** see as an inconsistency) when working on 2.4 version of my RubyChanges project. Typically, when going through the language version's changelog and trying to rationalize it, I use to find some less-documented changes which makes me providing documentation patches and clarification tickets.

So, from where I am standing, the situation looks as follows:

1. The behavior is definitely an *inconsistency* (either behavioral or at least in documentation), I do believe PoLS should lead to having the symmetry or at least documenting of the lack of it.
2. As for three years since Ruby 2.4 I am the first to notice it (and even this is going through the changelog, not writing code), the problem is of pretty low priority.
3. Though, I don't really believe that having such an inconsistency is really justifiable, especially considering recent uprising of interest to immutability (and therefore freezing).
4. I don't think that "freeze cloned can be achieved with clone.freeze, therefore clone(freeze: true) is redundant" is a strong argument. For me, it looks a bit like saying "we can achieve subtraction with x.+(-y), therefore x.-(y) is redundant": clone(freeze: true) looks atomic, logical and readable, I don't see why it shouldn't work.
5. OK, "We don't need it because there are other ways to achieve it" could be justifiable **if** fixing the inconsistency requires significant effort (in planning and/or implementation). For what I can see, the change is pretty trivial. Attached is the patch implementing it. It is not the prettiest code possible, but at least it allows to estimate an amount of effort.

PS: I hope you will not find completely inappropriate that I've changed back the type of the issue to "Bug" and rephrased the title to show WHY I consider it so. From my perspective, it is not me requesting some new weird feature, but trying to straighten up some obvious inconsistency (even if a low-priority one).

#4 - 09/22/2019 10:52 AM - mame (Yusuke Endoh)

- Backport deleted (2.5: UNKNOWN, 2.6: UNKNOWN)

- Tracker changed from Bug to Feature

I agree with Jeremy. It is not a bug.

I think that the wording of freeze: false is a bit confusing, but the document explains clearly. Do you have a real-world use case for clone(freeze: true)?

Anyway, please do not modify the tracker. Bug/Feature is not significant for you. It is mainly used as an advice for branch maintainers: the change should be backported or not.

#5 - 09/23/2019 11:23 PM - jeremyevans0 (Jeremy Evans)

- File clone-freeze-true-16175.patch added

Attached is an alternative approach for implementing this. It uses VALUE for the kwfreeze variable, so we can use Qundef instead of the magic value of -1 for the default behavior. It also updates the documentation to reflect you can use true or false.

Technically, this feature breaks backwards compatibility, because freeze: true did not freeze clones of unfrozen receivers previously. However, it seems unlikely that someone would rely on that behavior.

#6 - 03/16/2020 04:16 AM - matz (Yukihiro Matsumoto)

I accept the proposal. As jeremyevans0 (Jeremy Evans) stated, it's slightly backward-incompatible but trivial.

Matz.

#7 - 03/16/2020 04:32 PM - jeremyevans0 (Jeremy Evans)

I'll work on implementing this after https://github.com/ruby/ruby/pull/2954 is merged or closed, as I don't want to cause conflicts for that pull request.

#8 - 03/17/2020 05:11 PM - jeremyevans0 (Jeremy Evans)

Pull request 2954 was merged early today, so I was able to work on this. I submitted https://github.com/ruby/ruby/pull/2969 to implement it.

In the recent dev meeting notes, akr (Akira Tanaka) stated freeze: nil should be supported, but matz (Yukihiro Matsumoto) said he didn't see a need to introduce it. After the merging of pull request 2954, Object#clone is partially written in Ruby via kernel.rb, and it would require more complex code not to support it. Additionally, delegate and set would both require more complex code if it was not supported. Therefore, I implemented support for freeze: nil, and made nil the default value of the freeze keyword argument for initialize_clone.

#9 - 03/19/2020 12:57 AM - Eregon (Benoit Daloze)

I'm confused, I thought clone(freeze: true) would be the same as clone(), i.e., freeze the cloned object only if the original object was frozen.

I don't think it's good to make #clone also #freeze if the original object was not frozen, that's a separate concern and out of #clone's role.

I think obj.clone.freeze is good enough for that use case, much clearer, and more compatible (#initialize_clone doesn't need to deal with this)

#10 - 03/19/2020 01:04 AM - Eregon (Benoit Daloze)

To clarify I'm not against the PR, but it seems a very niche use-case to me with no real-world example, and IMHO the documentation was clear enough.
I think #clone is about copying state, not about changing other aspects like freezing when the original wasn't.

#11 - 03/22/2020 04:30 PM - jeremyevans (Jeremy Evans)

- Status changed from Open to Closed

Applied in changeset git|4f7b435c955792af780fe9e94f98d3dde839e056.

Support obj.clone(freeze: true) for freezing clone

This freezes the clone even if the receiver is not frozen. It
is only for consistency with freeze: false not freezing the clone
even if the receiver is frozen.

Because Object#clone is now partially implemented in Ruby and
not fully implemented in C, freeze: nil must be supported to
provide the default behavior of only freezing the clone if the
receiver is frozen.

This requires modifying delegate and set, to set freeze: nil
instead of freeze: true as the keyword parameter for
initialize_clone. Those are the two libraries in stdlib that
override initialize_clone.

Implements [Feature #16175]

Files

freeze-true.patch	1.47 KB	09/22/2019	zverok (Victor Shepelev)
clone-freeze-true-16175.patch	4.42 KB	09/23/2019	jeremyevans0 (Jeremy Evans)