

# Ruby master - Feature #15504

## Freeze all Range objects

01/04/2019 03:12 AM - ko1 (Koichi Sasada)

<b>Status:</b>	Closed
<b>Priority:</b>	Normal
<b>Assignee:</b>	matz (Yukihiro Matsumoto)
<b>Target version:</b>	

### Description

## Abstract

Range is currently non-frozen. How about freezing all Range objects?

## Background

We froze some types of objects: Numerics (r47523) and Symbols [Feature [#8906](#)]. I believe that making objects immutable solves some kinds of programming difficulties.

Range is mutable at least when written as Range literal. So we can write the following weird program:

```
2.times{
  r = (1..3)
  p r.instance_variable_get(:@foo)
  #=> 1st time: nil
  #=> 2nd time: :bar
  r.instance_variable_set(:@foo, :bar)
}
```

In range.c, there is a comment (thanks znz-san):

```
static void
range_modify(VALUE range)
{
  rb_check_frozen(range);
  /* Ranges are immutable, so that they should be initialized only once. */
  if (RANGE_EXCL(range) != Qnil) {
    rb_name_err_raise("`initialize' called twice", range, ID2SYM(idInitialize));
  }
}
```

## Patch

Index: range.c

```
-----
--- range.c (000000 66699)
+++ range.c (000000)
@@ -45,6 +45,8 @@
     RANGE_SET_EXCL(range, exclude_end);
     RANGE_SET_BEG(range, beg);
     RANGE_SET_END(range, end);
+
+   rb_obj_freeze(range);
 }
```

VALUE

## Discussion

There are several usages of mutable Range in the tests.

- (1) Taint-flag
- (2) Add singleton methods.
- (3) Subclass with mutable states

Maybe (2) and (3) are crucial.

Thanks,  
Koichi

#### Related issues:

Related to Ruby master - Feature #17195: Freeze Enumerator::ArithmeticSequenc...

**Rejected**

#### Associated revisions

##### Revision 0096d2b8 - 09/25/2020 01:16 PM - ko1 (Koichi Sasada)

freeze all Range objects.

Matz want to try to freeze all Range objects.  
[Feature #15504]

#### History

##### #1 - 01/04/2019 07:29 AM - marcandre (Marc-Andre Lafortune)

I think that (2) and (3) are indeed capital points. Freezing range literals (only) might be a better idea? with an approach like frozen string literals?

Not that even frozen ranges aren't completely immutable:

```
r = ('a'..'z').freeze
r.end.upcase!
r # => "a".."z"
```

##### #2 - 06/22/2019 02:38 PM - Eregon (Benoit Daloze)

I think it would make sense to freeze Range literals.

Adding methods to Range might be reasonable, but singleton methods, I would think much less so.

##### #3 - 06/23/2019 04:47 AM - mame (Yusuke Endoh)

I guess [Eregon \(Benoit Daloze\)](#) came from [#15950](#). Will `ary[-3..]` be as efficient as `ary[-3, 3]` by freezing and deduping a literal `(-3..)`? Looks good if we can confirm it by an experiment.

Some thoughts:

- Even if a range is frozen, `"a".."z"` should not be deduped because of the reason [marcandre \(Marc-Andre Lafortune\)](#) said.
- I'm for freezing all Ranges, not only Range literals. I hate the idea of freezing only literals because casually mixing frozen and unfrozen objects leads to hard-to-debug bugs that depend upon data flow.
- It would be the most elegant if the combination of MJIT and escape analysis solves this kind of performance problems.

##### #4 - 08/07/2020 06:12 AM - ko1 (Koichi Sasada)

I got an issue on Ractor.

```
def test
  (1..2)
end
```

```
r1 = test
```

```
Ractor.new do
  r2 = test
end.take
```

- compiler cached Range `(1..2)` because they begin and end are frozen literals. The test method returns a cached same range object.
- it means test returns not-immutable but same object. It violate the Ractor's memory model.

To solve it, there are two options.

- (1) avoid cache at compile time.
- (2) freeze Range objects which will be cached by th compiler.

For performance reason, I want to choose (2).

After that, could you please discuss all Range objects should be frozen or not.

Thanks,  
Koichi

**#5 - 08/07/2020 10:02 AM - sawa (Tsuyoshi Sawada)**

- Description updated

- Subject changed from Freeze all Range object to Freeze all Range objects

**#6 - 08/07/2020 11:21 AM - Eregon (Benoit Daloze)**

Right, every object cached at parse time must be deeply immutable, I would think that was an oversight.

**#7 - 09/25/2020 08:49 AM - matz (Yukihiro Matsumoto)**

I agree with making ranges frozen. I don't see any particular case that could be broken by frozen ranges. Since there's possibility of breakage, I'd like to experiment it.

Matz.

**#8 - 09/25/2020 11:53 AM - ko1 (Koichi Sasada)**

Ok, I freeze all Ranges except sub-classes.

<https://github.com/ruby/ruby/pull/3583>

**#9 - 09/25/2020 01:17 PM - ko1 (Koichi Sasada)**

- Status changed from Open to Closed

Applied in changeset [git0096d2b895395df5ab8696d3b6d444dc1b7730b6](https://github.com/ruby/ruby/commit/0096d2b895395df5ab8696d3b6d444dc1b7730b6).

---

freeze all Range objects.

Matz want to try to freeze all Range objects.

[Feature [#15504](#)]

**#10 - 09/25/2020 01:58 PM - Eregon (Benoit Daloze)**

Great!

Related, should Enumerator::ArithmeticSequence be frozen too?

**#11 - 09/25/2020 03:09 PM - ko1 (Koichi Sasada)**

Eregon (Benoit Daloze) wrote in [#note-10](#):

Related, should Enumerator::ArithmeticSequence be frozen too?

new ticket?

**#12 - 09/26/2020 01:20 PM - Eregon (Benoit Daloze)**

- Related to Feature [#17195](#): Freeze Enumerator::ArithmeticSequence objects added

**#13 - 09/26/2020 01:21 PM - Eregon (Benoit Daloze)**

ko1 (Koichi Sasada) wrote in [#note-11](#):

Eregon (Benoit Daloze) wrote in [#note-10](#):

Related, should Enumerator::ArithmeticSequence be frozen too?

new ticket?

I filed [#17195](#)