This proposal is to introduce # shareable_constant_value: true pragma to make constant values shareable objects. With this pragma, you don't need to add freeze to access from non-main ractors.

```ruby
# shareable_constant_value: true
A = [1, [2, [3, 4]]]
H = {a: "a"}

Ractor.new do
  p A
  p H
end.take
```

**Background**

Now, we can not access constants which contains a unshareable object from the non-main Ractor.

```ruby
A = [1, [2, [3, 4]]]
H = {a: "a"}

Ractor.new do
  p A
  #=> can not access non-sharable objects in constant Object::A by non-main Ractor. (NameError)
  p H
end.take
```

If we know we don't modify A and H is frozen object, we can freeze them, and other ractors can access them.

```ruby
A = [1, [2, [3, 4]].freeze].freeze].freeze
H = {a: "a"}.freeze

Ractor.new do
  p A #=> [1, [2, [3, 4]]]
  p H #=> {a=>"a"}
end.take
```

Adding nesting data structure, we need many .freeze method. Recently, I added Ractor.make_shareable(obj) makes obj shareable with freezing objects deeply (see [Feature #17274]). We only need to introduce this method for each constant.

```ruby
A = Ractor.make_shareable( [1, [2, [3, 4]]] )
H = Ractor.make_shareable( {a: "a"} )

Ractor.new do
  p A #=> [1, [2, [3, 4]]]
  p H #=> {a=>"a"}
end.take
```

However, if we have 100 constants, it is troublesome.

**Proposal**

With # shareable_constant_value: true, you can specify all constants are shareable.
# shareable_constant_value: true

A = [1, [2, [3, 4]]]
# compiled with: A = Ractor.make_shareable([1, [2, [3, 4]]])
H = {a: "a"}
# compiled with: H = Ractor.make_shareable({a: "a"})

Ractor.new do
  p A
  p H
end.take

(Strictly speaking, don't call Ractor.make_shareable, but apply same effect. This means rewriting Ractor.make_shareable doesn't affect this behavior)

You can specify # shareable_constant_value: false in the middle of the place.

# shareable_constant_value: true

S1 = 'str'
# p S1.frozen? #=> true

# shareable_constant_value: false

S2 = 'str'
# p S2.frozen? #=> false

The effect of this pragma is closed to the scope.

class C
  # shareable_constant_value: true
  A = 'str'
  p A.frozen? #=> true

  1.times do
    # shareable_constant_value: false
    B = 'str'
    p B.frozen? #=> false
  end
end

X = 'str'
# p X.frozen? #=> false

Ractor.make_shareable(obj) doesn't affect anything to shareable objects.

# shareable_constant_value: true

class C; end

D = C
# p D.frozen? #=> false

Some objects can not become shareable objects, so it raises an exception:

# shareable_constant_value: true

T = Thread.new{}
#=> `make_shareable': can not make shareable object for #<Thread:0x000055952e40ffb0 /home/ko1/ruby/src/trunk/test.rb:3 run> (Ractor::Error)

Implementation

https://github.com/ruby/ruby/pull/3681/files

Related issues:

Related to Ruby master - Feature #17274: Ractor.make_shareable(obj) Closed
Related to Ruby master - Feature #17145: Ractor-aware `Object#deep_freeze`  
Rejected
Related to Ruby master - Feature #17278: On-demand sharing of constants for R...  
Feedback
Related to Ruby master - Feature #17397: `shareable_constant_value: literal` ...  
Closed

Associated revisions

Revision b1bd2230 - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
Support shareable_constant_value pragma

Revision f43c71ab - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
Implemented shareable_constant_value
It does shallow freeze only for now.

Revision 65450e8f - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
Call FrozenCore.make_shareable

Revision 60f0c376 - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
Implemented shareable_constant_value op_asgn

Revision dc1cc33d - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
Determine shareable-ness after assignment operator

Revision 07a990b - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
Save and pass lex_context wholly

Revision 11d9983b - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
Make the value shareable deeply

Revision d19601fb - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
Test by Ractor.shareable?

Revision 78cb9b62 - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
Added false case

Revision 25c1aca - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
Added continued line case

Revision 7060aeed - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
shareable_constant_value: is effective only in comment-only line

Revision 89e489d5 - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
Make shareable_constant_value tri-state

Revision 33231747 - 12/14/2020 10:19 AM - nobu (Nobuyoshi Nakada)
Support shareable_constant_value: literal

Revision 19a98a87 - 12/18/2020 03:34 PM - nobu (Nobuyoshi Nakada)
Fixed not to make non-literal expression shareable [Feature #17273]
Non-literal expression which is not a part of a literal expression
is not a subject of shareable_literal_value: literal.

History

#1 - 10/21/2020 05:58 AM - ko1 (Koichi Sasada)
- Description updated

#2 - 10/21/2020 06:48 AM - ko1 (Koichi Sasada)
Re naming, how about # shareable_constants: true instead of # shareable_constant_value: true?

(it's # frozen_string_literal: true but maybe it should have been # frozen_string_literals: true ...)

However, if we have 100 constants, it is troublesome.

This seems very unlikely, isn't it?
If there are that many constants, I think there is a high chance the value is an immediate or a String, then there is no need.
Or that metaprogramming is used to define the constant dynamically, and then not many Ractor.make_shareable call sites are needed.

Having the pragma allowed in the middle of the file feels like the C preprocessor, which I find rather unpretty.
If one needs some constants to not be frozen, maybe it's enough to use Ractor.make_shareable explicitly?

I can appreciate that such a pragma is more forward-compatible than explicit Ractor.make_shareable, though.

A = expr.deep_freeze (17145) seems nicer to me than A = Ractor.make_shareable(expr) in user code.
For constants' values, it seems unlikely to have shareable-but-should-not-be-frozen objects.

If one needs some constants to not be frozen, maybe it's enough to use Ractor.make_shareable explicitly?

Yes. I agree if there is few constants, explicit method call with good name is more friendly.

If there are that many constants, I think there is a high chance the value is an immediate or a String, then there is no need.

We can eliminate calling by compile time, in future.

I deep-freeze all my constants (expect modules of course). RuboCop will enforce that literal constants are frozen by default.

I like the idea of a pragma for this.

I also prefer:

- # shareable_constants: true
- only allowed at the top

Maybe the pragma should be # frozen_constants: true?
"Freezing a constant" is intuitively "deeply-freeze the value", isn't it?

And since we already have # frozen_string_literal: true it would make nice connection.

Also, shareable seems very abstract, while I'd think almost every Rubyst knows what frozen (and deeply frozen) means.

Semantics-wise, I think we could still use the same semantics as Ractor.make_shareable.
I guess nobody wants a deep_freeze that also freezes an object's class.
And freezing a shareable object which is not always frozen (immutable) seems of little value:

- no much point to freeze a Ractor/Thread::TVar
- those are probably uncommon to be used as a value for a constant
Are there other shareable but not immutable objects besides Ractor/Thread::TVar/Module?

Along that idea, I think `#deep_freeze` (#17145) could by default skip shareable values (so `skip_shareable` would default to true).

### #10 - 10/25/2020 01:38 PM - Eregon (Benoit Daloze)
- Related to Feature #17274: `Ractor.make_shareable(obj)` added

### #11 - 10/25/2020 01:39 PM - Eregon (Benoit Daloze)
- Related to Feature #17145: Ractor-aware `Object#deep_freeze` added

### #12 - 10/26/2020 04:29 PM - ko1 (Koichi Sasada)

Today's (yesterday's) dev-meeting, there is a comment:

- It is possible to break other library easily:

```ruby
require 'other-lib'

# sharable_constants: true
A = OtherLib::MutableArray # freeze OtherLib::MutableArray accidentally
```

In this case, bug report will be sent to the other-lib's maintainer. It is hard to recognize where the frozen attribute are attached.

Ractor.make_shareable has same problem, but this pragma can introduce this issue easily (if some guide tell to newbe that "put shareable_constants: true at the beginning of file...")

We have no conclusion about this issue.

### #13 - 10/29/2020 04:14 PM - ko1 (Koichi Sasada)

By discussing with Matz and several MRI committers, we decided to introduce conservative option and radical option as experimental.

- the name of pragma is `shareable_constant_value` because it affects values referred from constants (Matz's preference)
- not true/false, but the following options.

```ruby
# shareable_constant_value:
#   none   # same as 2.x
#   literal # literal only
#   C = lits
#   =>
#   C = Ractor.make_shareable(lits)
#   #
#   lits contains any combination of Array, Hash, String, ...
#   # String interpolation is also accepted because it copies all strings.
#   # if lits contains Ruby expression, SyntaxError
#   experimental_everything
#     C = expr
#     =>
#     C = Ractor.make_shareable(expr)
```

- experimental_everything is proposed option, but it seems danger, so make it experimental and rename/delete it from later version.
- literal option is a conservative option.

### #14 - 10/29/2020 07:26 PM - Eregon (Benoit Daloze)

ko1 (Koichi Sasada) wrote in [#note-12](#):

- It is possible to break other library easily:

That sounds very bad code breaking the encapsulation of that other library. Of course, no gems should directly mutate constants of another gem. That sounds even worse than calling a private method of another gem.

### #15 - 10/29/2020 07:29 PM - Eregon (Benoit Daloze)

ko1 (Koichi Sasada) wrote in [#note-13](#):

if lits contains Ruby expression, SyntaxError

Could you give an example?
SyntaxError doesn't seem OK to me. It should simply not freeze if not a literal.

#16 - 10/29/2020 07:33 PM - Eregon (Benoit Daloze)
Eregon (Benoit Daloze) wrote in #note-14:

Of course, no gems should directly mutate constants of another gem.

I missed that the gem doesn't need to mutate A to break OtherLib.
I guess it's relatively rare that a gem would (intentionally) expose a non-frozen constant as part of its API, and that the gem relies on being able to mutate it.

#17 - 10/29/2020 07:47 PM - ko1 (Koichi Sasada)
Eregon (Benoit Daloze) wrote in #note-16:

I guess it's relatively rare that a gem would (intentionally) expose a non-frozen constant as part of its API, and that the gem relies on being able to mutate it.

To confirm it, we will introduce experimental radical API. Other verification methods are welcome.

#18 - 10/29/2020 07:53 PM - jeremyevans0 (Jeremy Evans)
Eregon (Benoit Daloze) wrote in #note-16:

Eregon (Benoit Daloze) wrote in #note-14:

Of course, no gems should directly mutate constants of another gem.

I missed that the gem doesn't need to mutate A to break OtherLib.
I guess it's relatively rare that a gem would (intentionally) expose a non-frozen constant as part of its API, and that the gem relies on being able to mutate it.

I agree that it's relatively rare and not a good idea. However, Sequel does this (Sequel::DATABASES). It's been around since 2008 and many external users rely on reading from it, and I haven't wanted to break backwards compatibility to remove it. Internal access is always protected by a mutex for thread-safety, and it only gets mutated for new or removed database connections, so it isn't a problem in practice.

#19 - 12/16/2020 06:36 PM - marcandre (Marc-Andre Lafortune)
- Status changed from Open to Closed

Closing, this has been implemented by Nobu

#20 - 12/16/2020 06:38 PM - marcandre (Marc-Andre Lafortune)
- Related to Feature #17278: On-demand sharing of constants for Ractor added

#21 - 12/16/2020 08:13 PM - Eregon (Benoit Daloze)
- Related to Feature #17397: “shareable_constant_value: literal” should check at runtime, not at parse time added